ICL

KDF 9 Programming

ICL

KDF 9 Programming

The policy of International Computers Limited is one of continuous development and improvement of its products and services, and the right is therefore reserved to alter the information contained in this document without notice. ICL makes every endeavour to ensure the accuracy of the contents of this document but does not accept liability for any error or omission. Any equipment or software performance figures and times stated herein are those which ICL expects to be achieved in normal circumstances. Where-ever practicable, ICL is willing to verify upon request the accuracy of any specific matter contained in this document.

With effect from 12th August 1968 the name of
English Electric Computers
has been changed to
International Computers Limitud

Technical Publication 1002 mm (R)

First Edition January 1968

Second Edition October 1969

Issued by Technical Publications Service
International Computers Limited
Head Office: ICL House, Putney, London SW15
Produced by ICL Printing Services
at Kidsgrove, Staffordshire

CONTENTS

SECTION Page

1· INTRODUCTION

1·1 The Basic System

All electronic digital computers consist of two main categories of equipment:

(a) The 'black boxes' which perform all the calculations, processing of data, etc. and

(b) The 'peripheral devices' via which the black boxes communicate with the outside world.

KDF 9 is such a machine.

The black boxes are the Basic Computer, and unless peripheral devices exist to feed in (input) commands and give out (output) results of calculations, etc. it is virtually useless to man.

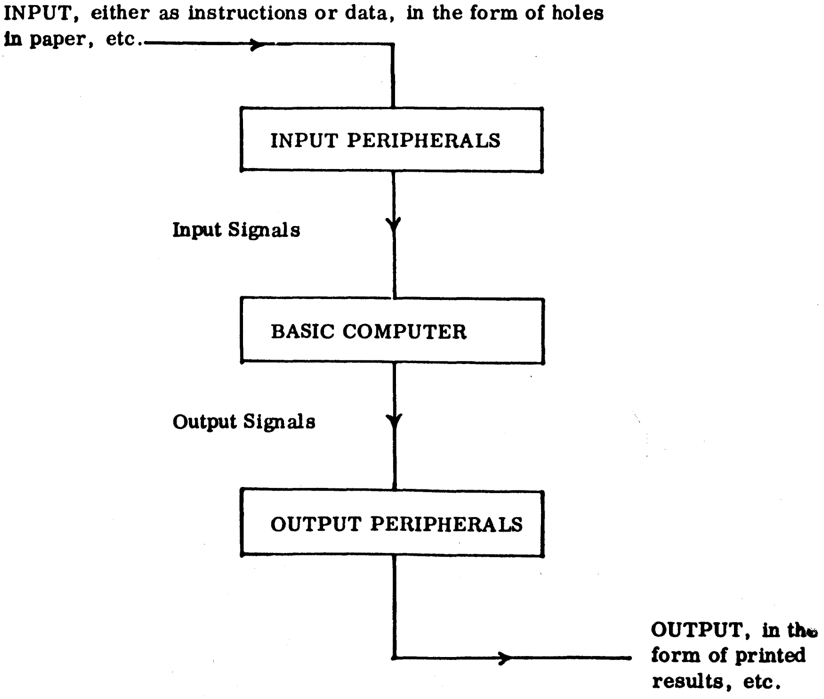

INPUT, either as instructions or data, in the form of holes in paper, etc.
INPUT PERIPHERALS
Input Signals
BASIC COMPUTER
Output Signals
OUTPUT PERIPHERALS
OUTPUT, in the form of printed results, etc.

Figure 1

1·1·1 Peripheral Devices.

Peripheral devices for input may be such items as paper tape readers, electric typewriter (flexowriter), punched card readers, magnetic tape units, etc., which are the media via which we communicate with the machine.

The output peripherals are such as high-speed-line-printers, and paper tape punches.

1·2 Program

A sequence of instructions which the computer is to obey is known as a Program.

After a programmer has written a program it is prepared in a manner acceptable to one of the input peripherals. This device converts the program into input signals which are fed to the computer and stored away in the 'memory'. When all the instructions of the program have been stored away, and when certain requisites have been carried out the computer fetches the instructions, one at a time, from its memory and obeys them. Should the instructions, whilst being obeyed, require some data from outside the computer, this will be asked for and be expected to be received from one of the input peripherals. Eventually the results are passed to an appropriate output device for printing, etc.

1·3 The Hardware and Software

The physical ironwork and electronics of the basic computer and its peripheral devices are known as the 'hardware'.

Those programs which enhance the ability of the hardware are called the 'software'.

The hardware by itself is like a car without a driver. It has ability to do certain things but must be instructed (by software) what to do.

The main 'driver' of the hardware is that part of software known as the "DIRECTOR", which is a program that is fed into the machine at the beginning of the day, and stays there governing the functioning of the computer. This Director program will arrange for other programs to be received one at a time into the computer and for them to be run. On some versions of KDF 9 the Director program may be controlling four programs running concurrently.

1·4 Programming Languages

The computer is designed to only understand instructions which are in a language called 'Machine Code' (sometimes referred to as Binary Code). A program of instructions in this code is not easily written, so another language has been devised called USERCODE which is a mnemonic code, easily understood and which has a one-to-one correspondence with Machine Code.

Software includes a program known as 'Compiler' which can take a Usercode program and translate it into its equivalent Machine Code version. Hence the programmer writes his program in Usercode, it is translated inside the computer by Compiler, (it is then said to be 'compiled') and the computer is then able to obey the instructions of the translated version.

1·5 Ability of the Computer

A widely held misconception about computers is that they can 'think' like a brain. This is not true. They obey instructions - admittedly much faster than man can do - but that is all. These instructions, the program, must be precise, and, if the results are to be of use, correct. Any logical error made by the programmer, generally, will not be detected by the computer. Hence a computer is not an electronic brain, it is an extremely fast obeyer of instructions.

1·6 Example of a USERCODE Program Body

This is an example of a simple Usercode program, which causes two numbers to be read in, added, and the result fed out. The reader is not expected to understand it until he has learnt, among other things, the computer's logical structure and information representation.

```
                    V0  =  Q 0/AY27/AY27;
                    V1  =  Q 0/AY92/AY92;
                    V2  =  Q 0/AY43/AY43;

V0; =Q1; SET2; SET5; OUT; (OBTAINED USE OF READER);
V1; =Q2; DUP: =C1; =C2;
V2; =Q3; SET1; SET5; OUT; (OBTAINED USE OF PUNCH); =C3;
PIAQ1; PARQ1; (1ST NO. READ INTO Y27); J1TR;
PIAQ2; PARQ2; (2ND NO. IN Y92); J1TR;
Y27; Y92; +; (NOS. ADDED); =Y43; (ANS IN Y43);
POAQ3; (Y43 PUNCHED);
1; C1; SET6; OUT; C3; SET6; OUT; ZERO; OUT;
FINISH;
→
```

2• NUMBER SYSTEMS

2•1 Information inside KDF 9

Information to be understood by KDF 9 has to be in binary patterns and on occasions communication between the computer and programmer requires the octal representation of these binary patterns. The following will explain binary and octal number systems with reference to the more familiar decimal system.

2•2 Rules of Number System (Integers)

When counting in the decimal system, only one of the digits 0 to 9 is needed, until it becomes necessary to specify the number "ten". We then put a 1 in the next column to the left and reset the original digit from 9 to 0. So to represent "ten" we put 10, where 1 is a digit and 0 is a digit; notice that 10 is not one digit, it is a combination of two digits representing a number.

When "nineteen" is reached and the next number is required to be written down there is no digit left to be put into the right hand column so 0 is placed there and "one" added to the tens column. This process continues until "one-hundred" is required when the carry goes even further to the left.

There are two essential points to notice about this familiar process:

(a) There could be as many zeros as we liked to the left of the number (assuming it to be an integer) so that 0000027 is just the same as 27,

and

(b) A carry into the next most significant digit position occurs whenever the given digit reaches the value "ten".

More formally it is said that any integer number n can be represented in the decimal system by the equation:

$$n = \ldots\ldots+(d_r \times 10^r)+\ldots\ldots\ldots+(d_o \times 10^o)$$

where d_o, d_1, etc. may be any of the digits 0 to 9.

Bear in mind that, for example, "sixteen" is the WORD used to represent the number, whereas 16 is the representation of that number in the decimal system. In fact, 16 should be written as $(1 \times 10^1) + (6 \times 10^0)$, but by convention we drop the 10^1 etc.

2•2•1

Assume man had used a number system based on eight digits only (0 to 7). Counting would be up to 7 and then the 'carry one' performed. This would mean that to represent "eight" 10 would have to be written. Such a system based on the digits 0 to 7 is called the Octal System. The octal 1Ø is thus the same value as the decimal 8.

Below a few decimal and octal numbers are listed for comparison.

Decimal	Octal
6	6
7	7
8	10
14	16
15	17
16	20
19	23
20	24

Formally it is said that the number n can be represented in octal by:

$$n = \ldots\ldots\ldots\ldots+(d_r \times 8^r)+\ldots\ldots\ldots\ldots+(d_o \times 8^o)$$

where the d's may be any of the digits 0 to 7.

2·2·2 The base in the decimal system is "ten" and in octal "eight". This base is called the RADIX of the system. In order to avoid confusion between various systems the radix is written as a suffix in parenthesis after the number unless it is abundantly clear which system is being used. Hence "fourteen" would be $14_{(10)}$ or $16_{(8)}$.

It is possible to have a number system with a radix of any number greater than 1 provided symbols are devised to represent digits greater than 9.

2·2·3 The Binary System is a number system with the Radix 2, the permissible digits being 0 and 1.

The corresponding Binary equivalents of the decimal numbers 0 to 10 are:

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Formally it can be said that a binary number such as 1101001 represents:

$$(1 \times 2^6)+(1 \times 2^5)+(0 \times 2^4)+(1 \times 2^3)+(0 \times 2^2)+(0 \times 2^1)+(1 \times 2^0)$$

2•3 Rules for Number Systems (Fractions)

Extension of the above section to cover fractions only requires the indices of the Radix to be continued to negative values.

The decimal number 26•905 in full should be written as

$$(2 \times 10^{1})+(6 \times 10^{0})+(9 \times 10^{-1})+(0 \times 10^{-2})+(5 \times 10^{-3}).$$

Conventionally the 10's are dropped and to indicate where the indices of the radix change from positive to negative a point (in this system called the decimal point) is inserted. There could be as many zeros as desired to the right of the fraction so that the above number is also 0026•905000; the same rule applies in other systems with radices other than "ten".

2•3•1 In the octal system $15{\cdot}4_{(8)}$ in full is

$(1 \times 8^{1})+(5 \times 8^{0})+(4 \times 8^{-1})$ which is $(1 \times 8)+(5 \times 1)+(4 \times \frac{1}{8})$

or $8 + 5 + \frac{1}{2}$

or $13{\cdot}5_{(10)}$

N.B. The point in $15{\cdot}4_{(8)}$ is called the octal point.

2•3•2 Similarly in the binary system $1011{\cdot}01_{(2)}$ in full is

$(1 \times 2^{3})+(0 \times 2^{2})+(1 \times 2^{1})+(1 \times 2^{0})+(0 \times 2^{-1})+(1 \times 2^{-2})$

which is

$(1 \times 8)+(0 \times 4)+(1 \times 2)+(1 \times 1)+(0 \times \frac{1}{2})+(1 \times \frac{1}{4})$

or $8 + 0 + 2 + 1 + 0 + \frac{1}{4}$

or $11{\cdot}25_{(10)}$

N.B. The point in $1011{\cdot}01_{(2)}$ is called the binary point.

2•3•3 In decimal, 15•63472 is regarded as representing a number to 5 decimal places, which are understood to be 5 decimal **fractional** places. In octal and binary the expressions are octal fractional places and binary fractional places respectively.

2•3•4 It should be exphasised that when we call 25, "twenty five", we are justified in doing so because in everyday language it is understood that the decimal system is being used. However, should be wish to talk of $31_{(8)}$ we must say, "three, one, octal", or "the octal representation of twenty five".

In the same way in binary, $101_{(2)}$ should be referred to as "one, oh, one, binary", or "the binary representation of five".

The reader should acquire the good habit of referring to numbers correctly; if he doesn't he will find himself making such mistakes as writing 1101 for "eleven hundred and one" and then think of it as "thirteen" because he can see only 1's and 0's and immediately treat it as a binary number.

2•4 Conversion between Systems

Conversion of a number from its representation in one system to another has to be performed in two stages: first the integer part is converted by one process and then the fraction part by another. The two results are then written down with the point inserted. The reason for this dual process is that integers have positive radix indices whereas those for fractions are negative.

N.B. The reader is asked to bear in mind that this process is used by man when converting on paper; in the computer the method is somewhat different. Confusion between the two methods and their particular application could unnecessarily increase the length of the program.

2•4•1 The Rules for Conversion (on paper)

1. Integers

(a) From Decimal to another system. Divide the decimal number by the radix of the other system and keep a record of the remainder; divide the quotient by the same radix recording the remainder and repeat this step until the quotient is 0.

The resulting remainders in reverse order are the digits of the number in the new system from left to right.

Example: to convert $8632_{(10)}$ to octal.

```
8 ) 8632
8 ) 1079  r. 0
8 )  134  r. 7
8 )   16  r  6
8 )    2  r. 0
       0  r. 2
```

Hence $8632_{(10)}$ is equivalent to $20670_{(8)}$

(b) From another system to Decimal. Multiply the most significant digit by the radix from which the conversion is taking place; add the result to the next digit of the original number (0 is a digit) and again multiply by the same radix; continued only to the stage when the current result is added to the digit in the units column of the original.

Example: to convert $3645_{(8)}$ to decimal.

$$\begin{array}{rrrr} 3 & 6 & 4 & 5 \\ \underline{\times\ 8} & \underline{+\ 24} & \underline{+\ 240} & \underline{+1952} \\ 24 & 30 & 244 & 1957 \\ & \underline{\times\ \ 8} & \underline{\times\ \ \ 8} & \\ & 240 & 1952 & \end{array}$$

Hence $3645_{(8)}$ is equivalent to $1957_{(10)}$

2. **Fractions**

(a) From Decimal to another system. Multiply the fraction by the radix of the other system and keep a record of the resulting integral part; multiply the resulting fractional part by the radix and repeat until the fraction becomes zero or until sufficient precision is obtained. The resulting integral parts are the digits of the number in the new system from left to right; the point must then be inserted to the left of the number.

Example: to convert $\cdot068359375_{(10)}$ to octal.

$$\begin{array}{rl} & \cdot068359375 \\ & \underline{\qquad\qquad\quad 8} \\ (0) & \cdot546875000 \\ & \underline{\qquad\quad\ 8} \\ (4) & \cdot375000 \\ & \underline{\quad\ 8} \\ (3) & \cdot000 \end{array}$$

Hence $\cdot068359375_{(10)}$ is equivalent to $\cdot043_{(8)}$

(b) From another system to Decimal. Divide the least significant digit by the radix, obtaining a fraction, insert the next digit of the original in front of this resulting fraction and again divide; continue until **all** digits of the original (including **all** zeros) have been brought down and divided.

Example: to convert $\cdot0273_{(8)}$ to Decimal.

$$\begin{array}{ll} 8 & \underline{/\quad 3\cdot0000\quad} \\ & \quad\ \ \cdot375 \\ 8 & \underline{/\quad 7\cdot375\quad} \\ & \quad\ \ \cdot921875 \\ 8 & \underline{/\quad 2\cdot921875} \\ & \quad\ \ \cdot365234375 \\ 8 & \underline{/\quad 0\cdot365234375} \\ & \quad\ \ \cdot045654296875 \end{array}$$

Hence $\cdot0273_{(8)}$ is equivalent to $\cdot045654296875_{(10)}$

N.B. The reader is not expected to learn these rules by heart.

2·4·2 Octal to Binary and vice versa. The group of three binary digits 111 represents "seven"; if a 1 is now added the group becomes 000 and a 1 appears to the left.

Since the largest possible value for the group of three digits is seven, the **group** can be looked upon as having a Radix of eight. There then must be some relationship between these groups of three binary digits and the octal number system.

By inspecting the following examples it will be seen that every octal DIGIT can be written down directly as its equivalent group of three binary digits. It must be emphasised that the full group of three must appear so that $\cdot 1_{(8)}$ is not $\cdot 001_{(2)}$ and NOT $\cdot 1_{(2)}$.

However, more significant integer zeros and least significant fraction zeros may be omitted, e.g., $\cdot 2_{(8)} = \cdot 010_{(2)} = \cdot 01_{(2)}$

Octal		Binary		
$5_{(8)}$	=	$101_{(2)}$		
27	=	010 111	=	10111
·23	=	·010 011	=	·010011
12·36	=	001 010 ·011 110	=	1010·01111

In the last example, had it been required to convert $1010\cdot 01111_{(2)}$ to octal, the necessary 0's would have had to be inserted in order to complete the groups of three before writing down the octal equivalent:

viz., 1010·01111 = 1 010 · 011 11
= 001 010 · 011 110 = $12\cdot 36_{(8)}$

Another point to remember is that when converting from binary to octal, the grouping of threes **must proceed from the binary point in either direction.**

2·5 Arithmetic Operations

The arithmetic operations used in decimal notation apply equally in the other number systems provided it is remembered that the digits available for use are restricted to one less than the radix being used, and that the multiplication tables are different.

Examples of the tables in octal are

$$\begin{aligned}
4_{(8)} &\times 3_{(8)} = 14_{(8)}\\
6_{(8)} &\times 6_{(8)} = 44_{(8)}\\
7_{(8)} &\times 5_{(8)} = 43_{(8)}\\
10_{(8)} &\times 10_{(8)} = 100_{(8)}
\end{aligned}$$

There is no need to learn the tables for the different systems; merely use the normal tables and convert the partial answer of each operation performed. So that, to multiply $35_{(8)}$ by $4_{(8)}$ the procedure is:

3 5	
4	$4 \times 5 = 20$
	$20_{(10)} = 24_{(8)}$
4	put down 4 and carry 2
	$4 \times 3 = 12$
	$12_{(10)} = 14_{(8)}$
	$14_{(8)}$ + the $2_{(8)}$ carried $= 16_{(8)}$
6	put down 6 and carry 1
1	no more working so put down the $1_{(8)}$ carried
Answer $164_{(8)}$	

This is rather a tedious process, probably the reader would rather convert both the multiplicands to decimal, obtain the product and then convert the answer back. In practice programmers seldom find the need to perform this operation, or in fact any of those in the next section but nevertheless he should know what to do (-or better-where to look) should the need arise.

2·5·1 Examples of the four arithmetic rules are now given, using the same equivalent number in each group of three.

	Decimal	Octal	Binary
Addition			
(a)	22	26	10110
	+21	+25	+10101
	43	53	101011
(b)	23·5	27·4	10111·1
	22·25	26·2	10110·01
	+21·5	+25·4	+10101·1
	67·25	103·2	1000011·01

Subtraction

```
                    24            30           11000
(a)                -20           -24          -10100
                     4             4             100

(b)                34·0          42·0        100001·0
                  -15·5         -17·4       -  1110·1
                   18·5          22·4         10010·1

                 Decimal        Octal          Binary
```

Multiplication

```
(a)                  3             3              11
                    ×4            ×4           × 100
                    12            14            1100

(b)              25·50          31·4         11001·1
               × 12·25        × 14·2       ×  1100·01
              25500  00       3140 0      110011000 00
               5100  00       1460 0       11001100 00
                510  00         63 0           1100 11
                127  50       4703       1001110000 11
              31237  50
            = 312·3750        =470·3       = 100111000·011
```

Division (This is nasty at the best of times.)

```
                 Decimal                        Octal
                0·5454......etc.               0·4272......etc.
(a)         11/ 6·0                        13/ 6·0
                5 5                            5 4
                  50                             40
                  44                             26
                   60                            120
                   55                            115
                    50                             30
                    44                             26
                     6                              4
```

```
                 Binary
                    0·100010.......etc.
            1011/ 110·0
                  101 1
                      10
                      00
                      100
                      000
                      1000
                      0000
                      10000
                       1011       etc.
```

2·6 Exercises

1. What is the formal representation of:

(i) $157_{(10)}$

(ii) $29{\cdot}5_{(10)}$

(iii) $157_{(8)}$

(iv) $35{\cdot}4_{(8)}$

(v) $101_{(2)}$

(vi) $1101{\cdot}01_{(2)}$

2. How would you represent "forty-five" in:

(i) decimal; (ii) octal; (iii) binary;

3. How would you represent "sixteen and a half" in:

(i) decimal; (ii) octal; (iii) binary;

4. Write down the binary equivalent of:

(i) $634{\cdot}23_{(8)}$

(ii) $2{\cdot}1_{(8)}$

(iii) $100{\cdot}001_{(8)}$

5. Write down the octal equivalent of:

(i) $101101{\cdot}101_{(2)}$

(ii) $1101{\cdot}1101_{(2)}$

(iii) $1{\cdot}1_{(2)}$

6. Perform the following:

(i) $276_{(8)} + 167_{(8)} + 32_{(8)}$

(ii) $12{\cdot}43_{(8)} + 762{\cdot}4_{(8)} + 0{\cdot}13_{(8)}$

(iii) $36{\cdot}27_{(8)} - 15{\cdot}37_{(8)}$

(iv) $521{\cdot}63_{(8)} - 43{\cdot}67_{(8)}$

7. Perform the following operations on the binary numbers given. (Hint: in some oases, when the binary number has many digits it may save time to convert to octal - calculate in octal - then convert back to binary.)

(i) 1011•1101 + 11•101 + 1•0001

(ii) 1010111000011•00101 - 1001001•00010

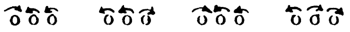
(iii) 1011•1101 × 11•101

(iv) 10101•1 ÷ 11•01 correct to 2 binary places.

3•1 Information Storage

To hold information, each storage location in the main "memory" is equipped with 48 miniature magnetic cores and depending on the direction of magnetisation, the computer will understand each core to represent either 0 or 1. Thus it is possible to hold a pattern of 48 binary digits, called 'bits', at each address. This pattern is called a 'word'.

For simplicity the contents of a 'word' can be represented on paper by sixteen octal digits, one for each group of three binary digits. If for a moment we assume that magnetisation of the cores anticlockwise means 0 and clockwise means 1, and assuming that each word has only 12 instead of 48 bits, then the configuration:



is understood to be:

$$\underbrace{1\;0\;0}_{4}\qquad\underbrace{0\;0\;1}_{1}\qquad\underbrace{1\;0\;0}_{4}\qquad\underbrace{0\;1\;0}_{2}\;\begin{matrix}(2)\\(8)\end{matrix}$$

which may be written as:

3•2 KDF 9 Character Code

It is now appropriate to discuss the KDF 9 Character Code, which is employed for the typewriter and 8 hole paper tape. Most readers will know that the Morse Code and the teleprinter code represent characters such as digits, letters of the alphabet and punctuation marks by dots and dashes from a buzzer, or holes and the absence of holes in paper tape. The KDF 9 Code is very similar.

A character is formed from 6 binary digits (bits), but it is more convenient to look upon it as being formed from two octal digits. The number of different patterns that may be constructed in this way from six bits is 64, viz., 0 to 63 inclusive. Since provision is made in the KDF 9 Character Code for a generous selection of punctuation marks and other symbols, and further since both capital and small letters are to be included, there are more than 64 items to be represented. This means that many patterns in the code must be used twice over. To distinguish between the two possible meanings of such a pattern, 'Case Shift' and 'Case Normal' characters are employed. A pattern is interpreted as being a character in case shift if the last 'case' character sensed was Case Shift, and in case normal if the last 'case' character was Case Normal. Note that those patterns with only one meaning have that meaning in case shift or case normal.

Octal Value	Function		Typewriter and 8-hole Paper Tape Version Octal Value	Symbol	
				Normal	Shifted
00	Space		40		
01			41	A	a
02	CR-LF		42	B	b
03			43	C	c
04	Tab		44	D	d
05			45	E	e
06	Case Shift		46	F	f
07	Case Normal		47	G	g
10			50	H	h
11			51	I	i
12			52	J	j
13			53	K	k
14			54	L	l
15			55	M	m
16			56	N	n
			57	O	o
	Symbol		60	P	p
	Normal	Shifted	61	Q	q
17	/	:	62	R	r
20	0	↑	63	S	s
21	1	[	64	T	t
22	2	]	65	U	u
23	3	<	66	V	v
24	4	>	67	W	w
25	5	=	70	X	x
26	6	×	71	Y	y
27	7	÷	72	Z	z
30	8	(	73		
31	9	)	74		
32	–	–	75	→	→
33	$_{10}$	£	76		
34	;	;	77		
35	+	=			
36	–	*			
37	–	,			

3·2·1 Explanation of Code. The above code is used in connection with paper tape. A slightly modified version applies for the high speed line printer. Items left blank such as $01_{(8)}$, $05_{(8)}$ are to be ignored.

$02_{(8)}$ CR-LF is the conventional short form of 'carriage return, line feed' i.e., causing the following character to be printed at the beginning of the next line.

$04_{(8)}$ Tab is the same as 'tabulate' on a typewriter.

$20_{(8)}$ Shifted Is used mainly in connection with 'Algol' and refers to exponentiation.

$32_{(8)}$ To cause a character to be underlined it is only necessary to precede the character with $32_{(8)}$.

The carriage of the flexowriter does not move after underline is typed.

$33_{(8)}$ Normal Is a suffix 10 below the line, and is on one character key.

$75_{(8)}$ Is the symbol called 'End Message'.

It must be emphasised that $20_{(8)}$ to $31_{(8)}$ inclusive (in case normal) only refer to the character form of the digits. If an address is to contain "thirty-eight", then feeding in $23_{(8)}$ followed by $30_{(8)}$ would not be sufficient. This would only have stored the character code of the digits 3 and 8. The method of converting $23_{(8)}$, $30_{(8)}$ into the NUMBER "thirty-eight" will be dealt with under 'Radix Convertion'.

3·3 Reference to a particular digit in a KDF 9 word

The 48 binary digits in a KDF 9 word are numbered for reference purposes D0-D47, with D0 the most significant (i.e., the extreme left) digit.

The abbreviation Dp is often used and interpreted to mean, according to context, either:

(i) The p^{th} digit of a word,

or

(ii) A word containing a 1 in the p^{th} position and 0's elsewhere.

Even though only the digits D0-D47 exist, there is no reason why we should not give p a value outside the range 0-47 and talk of, say, D49 or even D (-4), but of course these digits will not exist in the word.

3·4 Data

In Chapter 1 we saw that the words of the computer may contain either instructions to be performed or data relevant to the problem.

Data may be of two kinds:

(i) Numbers which the computer works upon, and

(ii) Characters of the Character Code.

It has already been shown how characters are stored; we shall now consider the storage of NUMBERS.

3·5 Number Representation

Before we consider the representation of numbers in the computer we need to be equipped with two new concepts of numbers.

(a) Any positive number may be represented by a fraction multiplied by an integral power of an integer.

e.g.,

$$+26{\cdot}3_{(10)} = \frac{26{\cdot}3}{100} \times 10^{2}$$

$$+26{\cdot}3_{(10)} = \frac{26{\cdot}3}{32} \times 2^{5}$$

$$+0{\cdot}014_{(10)} = \frac{{\cdot}014}{{\cdot}015625} \times 2^{-6}$$ (see Appendix 3 for powers of 2)

(b) Any fraction may be represented by the integer 0 or -1, plus a positive fraction.

e.g.,

$$+{\cdot}47_{(10)} = 0 + {\cdot}47$$

$$-{\cdot}47_{(10)} = -1 + {\cdot}53$$

By combining these two concepts we can represent any positive or negative number 'x' by the equation:

$$x = (-s+f) \times R^{p}$$

where s is 0 for positive numbers, and
1 for negative numbers;
f is any positive fraction;
p is a signed integer;
R is a positive integer.

e.g.,

$$+26{\cdot}3_{(10)} = (-0+\frac{26{\cdot}3}{64}) \times 2^{6}$$

$$+26{\cdot}3_{(10)} = (-0+\frac{26{\cdot}3}{32}) \times 2^{5}$$

) Note that different values of
) f and p may be used to
) represent the same number.

$+0{\cdot}47_{(10)}$ $\quad (-0+\frac{\cdot 47}{1}) \times 2^{0}$) Note that the representation
) of these two numbers does
) not only vary in 's' but
$-0{\cdot}47_{(10)}$ $\quad (-1+\frac{\cdot 53}{1}) \times 2^{0}$) necessarily also in 'f' because:
) $-1 + \frac{\cdot 47}{1} \qquad -0{\cdot}53$

3·5·1 Numbers in KDF 9. KDF 9 uses this equation to represent numbers, and since the computer works in binary the value of R is always 2, so that the equation becomes:

$$x = (-s+f) \times 2^{p}$$

Provided the values of s, f and p, are recorded, the value of x is uniquely specified.

There are two ways of recording s, f and p:

(a) In a system called 'Fixed Point' - where:
s is held in D0, and
f in D1-D47 in binary (the computer understands that the point is between
D0 and D1).

The programmer has to remember the value of each of the p's for every word containing a fixed point number.

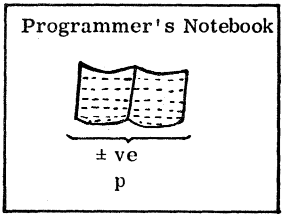
(b) In a system called 'Floating Point' - where:
s is held in D0,
f in D9-D47 in binary (the computer understand that the point is between
D8 and D9), and
p is held in D1-D8 in a kind of binary.

Since, in both systems, 's' represents either a positive or negative number by 0 or 1 respectively, and is held in D0, we call D0 the Sign Digit.

To differentiate between fixed and Floating-Point Number instructions the latter are followed by the letter F.

3·6 Fixed Point Numbers
The computer understands the layout of a fixed point number to be:

$$\underbrace{\text{D0}}_{\substack{\text{-ve}\\ \text{s}}}\ \ \underbrace{\text{D1 D2 D3 D4 D5..................D46 D47}}_{\substack{\text{+ve}\\ \text{f}}}$$



If s, f and p are recorded as shown above we say that 'the number x is held in the word to p **integral places**'. This means that the true value of x is obtained when the point is taken from between D0 and D1 and placed just after Dp; this implies that the digits D1 to Dp are binary **integral** places, (the Sign Digit, D0 is not included).

In fact the programmer always (in fixed point numbers) considers the point to be just after Dp.

Examples: If the word contains:

D0	D1	D2	D3	D4	D5	D6	D7	D8............	D47
0	0	0	0	0	1	0	0	0	0

and the programmer considers it is holding x to 5 integral places, (i.e., p=5 and the binary point is just after D5) then the true value of x is 00001.0000........$=1_{(2)}$.

If he considers that the number of integral places is 3, i.e., the binary point is just after D3, then x is 000.01000...$=\cdot 01_{(2)}$.

Again, if he considers it is -4 integral places, (i.e., the point is just after the imaginary D (-4)) then x is $\cdot 0000\ 00001 = 2^{-9}$.

Making p very large (and hence f very small) for a particular value of x may result in some of the least significant digits of f being too far to the right as to be held in D1-D47.

Similarly making p too small will result in making f so large that it ceases to be a pure fraction, this is not permissible. It is therefore necessary to ensure that the programmer knows the limits that p can taken for any x.

The equations: $x < 2^p$ for positive x

$|x| \quad 2^p$ for negative x (where $|x|$ means the value of x without the - sign)

will give the best value of p if the algebraically minimum is selected.

e.g., (i) the be3t (i.e., minimum) value of p for $28{\cdot}9_{(10)}$ is 5 because

$28{\cdot}9_{(10)}$ is less than 2^5 (N.B. $2^4 = 16$; $2^5 = 32$; $2^6 = 64$)

(ii) the best value of p for -64 is 6 because

64 is less than or equal to 2^6

(iii) and for $+0{\cdot}26$ it is -1 because

$0{\cdot}26$ is less than 2^{-1} (N.B. $2^{-2} = {\cdot}25$; $2^{-1} = {\cdot}5$)

If the value of p obtained this way is used, maximum precision will be obtained (i.e., the smallest possible number of least significant bits are lost off the right hand end).

A larger value of p could be used (but not smaller) if it were certain that no required digits would be lost off the right hand end.

Regarding integers, where there are never any bits to the right of the point, there is no reason at all why p should not be 47; in fact by making it 47, extremely large integers can be stored. It is left to the reader to satisfy himself that the largest positive integer possible under these circumstances is $2^{47}-1$, and the largest negative number is -2^{47}.

When considering negative numbers an interesting point emerges. We know that the computer requires (+x)+(-x) to equal zero. That this is the case (using the s, f and p equation) is shown as follows where $x = {\cdot}05_{(10)}$

$$\begin{array}{lcl}
 & +0{\cdot}05_{(10)} & = & \left(-0 + \frac{{\cdot}05}{{\cdot}5}\right) 2^{-1} \\
+ & -0{\cdot}05_{(10)} & = & \left(-1 + \frac{{\cdot}45}{{\cdot}5}\right) 2^{-1} \\
\hline
 & 0 & = & \left(\frac{{\cdot}05}{{\cdot}5} + \frac{{\cdot}45}{{\cdot}5} - 1\right) \; 2^{-1} \\
\hline
\end{array}$$

(It is essential, of course, that the p for both the numbers to be added be the same).

How can we quickly say what the contents of a word are, having been give the contents of a word holding a number of opposite sign?

Consider: $+12_{(10)}$ held to 4 integral places in a 6 bit word, added to

$-12_{(10)}$ held to 4 integral places in a 6 bit word.

```
      D0  D1  D2  D3  D4  D5

       0   1   1   0   0   0    =   +12
  +    1   0   1   0   0   0    =   -12
      ________________________
      ,0   0   0   0   0   0
     / ________________________
    1
```

There is no room in the 6 bit word to hold the 'extra' bit, so it is lost, which is exactly what we want, making the sum in the 6 bit word equal to zero. The reader is asked to verify for himself the following rule: to change the sign of the contents of a word, merely change all the 1's into 0's and vice versa starting at the most significant (left hand) end UP TO BUT NOT INCLUDING the least significant 1 bit.

3·6·1 Summary of Fixed Point.

(a) The computer considers a binary point to exist between D0 and D1, with D0 = -1 or 0, and D1 to D47 as a positive binary fraction.

(b) The programmer may consider the point to be anywhere, say after Dp, when he says the number is held to p integral places.

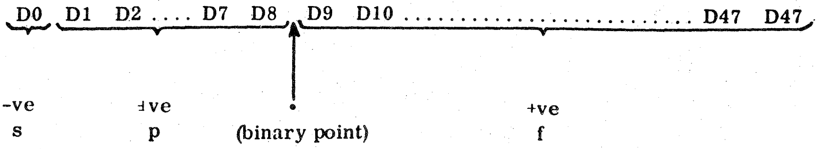
(c) For maximum precision p should have the algebraically smallest possible value.

(d) When working with pure integers p should be 47.

(e) p may be any value, even outside the range 1 to 47, in which case the extra D positions are assumed to exist.

3·7 Floating Point Numbers

The computer understands the layout of a floating point number to be:



In floating point we can not talk about "Integral places", the point is ALWAYS between D8 and D9. The similarity between D0 and D1-D47 in fixed point, and D0 and D9-D47 in floating point will be immediately obvious.

The difficulty arises when tring, in D1-D8, to represent p which may be positive or negative. From our previous discussion on the representation of numbers we saw that a sign digit is required. Unfortunately D0 has already been used as the sign digit for x. The problem is overcome by:

(i) D1 to D8 being considered as a positive binary integer with the point after D8, and

(ii) zero being represented by a 1 bit in D1 with 0's in D2 to D8.

This means that D1 to D8 contains $128_{(10)}$ for p 0. Any variation of p is added to or subtracted from this value. So that, for example, p 1 gives 10000001, and p -1 gives 01111111.

Examples of floating point:

$+16{\cdot}5_{(10)}$ where the computer has been told that p 7 is represented by:

D0 = 0
D1-D8 $128+7$ $135_{(10)}$ $10000111_{(2)}$
D9-D47 $00100001000\ldots_{(2)}$

$-1_{(10)}$ where the computer understands p 0 is represented by:

D0 - 1
D1-D8 $128+0$ $128_{(10)}$ $10000000_{(2)}$
D9-D47 = all 0's

Because p is indeterminate for x 0, zero is represented in the computer by D0-D47 being all 0's, i.e., exactly as with fixed point zero.

3·8 Standard Floating Point

To simplify floating point operations, the machine expects to find all floating numbers in a standard form. (There is one exception, which is the instruction which converts non-standard floating numbers to standard form.)

The standard form is arranged so that every number is expressed as precisely as possible, which implies that p and f are chosen so that p takes the algebraically minimum possible value.

3·9 Numbers Generally

When the point in any binary number pattern is moved m places to the left the effect is to divide the number by 2^m, similarly moving it to the right multiplies it by 2^m.

In fixed point we have f represented by 47 bits so the precision to which we can hold any fixed point number is 1 in 2^{47} (i.e., 1 in about $1{\cdot}4\times10^{14}$).

In floating point f has only 39 bits and the precision is 1 in 2^{39} (i.e., 1 in about $5{\cdot}5\times10^{11}$).

It may be questioned, "What is the advantage of floating over fixed point?" There are two answers, firstly the tedium of remembering p is removed from the programmer and secondly whereas the largest integer ($p=47$), that can be held fixed point is 2^{47} (i.e., $1\cdot4\times10^{14}$) in floating point p can be as large as 127 giving the largest integer as 2^{127} (i.e., $1\cdot7\times10^{38}$).

3·10 Examples

Character Code

1. A word containing $07550662073741_{(8)}$, when transferred to the typewriter causes

Mr. A.

to be typed.

2. If 4⊔JAN.⊔⊔is typed into a word it would eventually contain

$$2400524156370000_{(8)}.$$

(The manuscript way to indicate a space is by ⊔).

Fixed Point Numbers

1. The contents of a 12 bit word in octal when x is $111\cdot01001_{(2)}$ held to 5 integral places is

$$0722_{(8)}.$$

As the point can be imagined to be after D5 then the contents of the word are 000111 010010, which when grouped into threes and expressed in octal is 0722. We do not show the octal point because the point is not contained in the word.

2. The least number of integral places to which $-0\cdot124_{(10)}$ can be held in a word is -3.

$$(-0\cdot124) = 0\cdot124 \leq 2^{-3} \quad (\text{N.B. } 2^{-3} = 0\cdot125)$$

Floating Point Numbers

1. $7\cdot25_{(10)}$ in floating point with $p = 6$ is held as

0 10000110 00011101000.........

2. $7\cdot25_{(10)}$ in STANDARD floating point is held as

0 10000011 111010000...........

3. $-7\cdot25_{(10)}$ in STANDARD floating point is held as

1 10000011 000110000...........

3·11 Exercises

1. If the following sets of characters are read into a KDF 9 word, show the contents of the word in octal. Case normal is assumed.

(a) A B C D 3 4 ; →

(b) END⊔RUN⊔

(c) S̲T̲O̲P̲

2. If a KDF 9 word holding the following octal patterns is typed out, show the characters typed in the correct format.

(a) 06 33 07 21 00 31 06 63

(b) 07 02 41 42 06 02 41 42

(c) 07 32 56 32 57 00 75 00

(Spaces are to be indicated by⊔)

3. What is the largest positive integer which can be held fixed point in a word of 95 bits? Answer need not be expressed in decimal.

4. Convert the following decimal numbers to the octal pattern which represents the contents of a 12 bit word, holding them fixed point to the specified number of integral places.

	$\text{Number}_{(10)}$	$\text{Integral Places}_{(10)}$
(a)	16	11
(b)	-16	4
(c)	-0·575	0
(d)	31·5	5
(e)	-0·09375	-3
(f)	1865	as an integer
(g)	0·0001	with maximum precision

5. Write out in full the contents of a KDF 9 word, ending on the right with a string of dots to represent continuation of zeros if necessary, when the word contains the following decimal numbers in STANDARD FLOATING POINT. (Hint : compare with question 4).

(a) 16

(b) -16

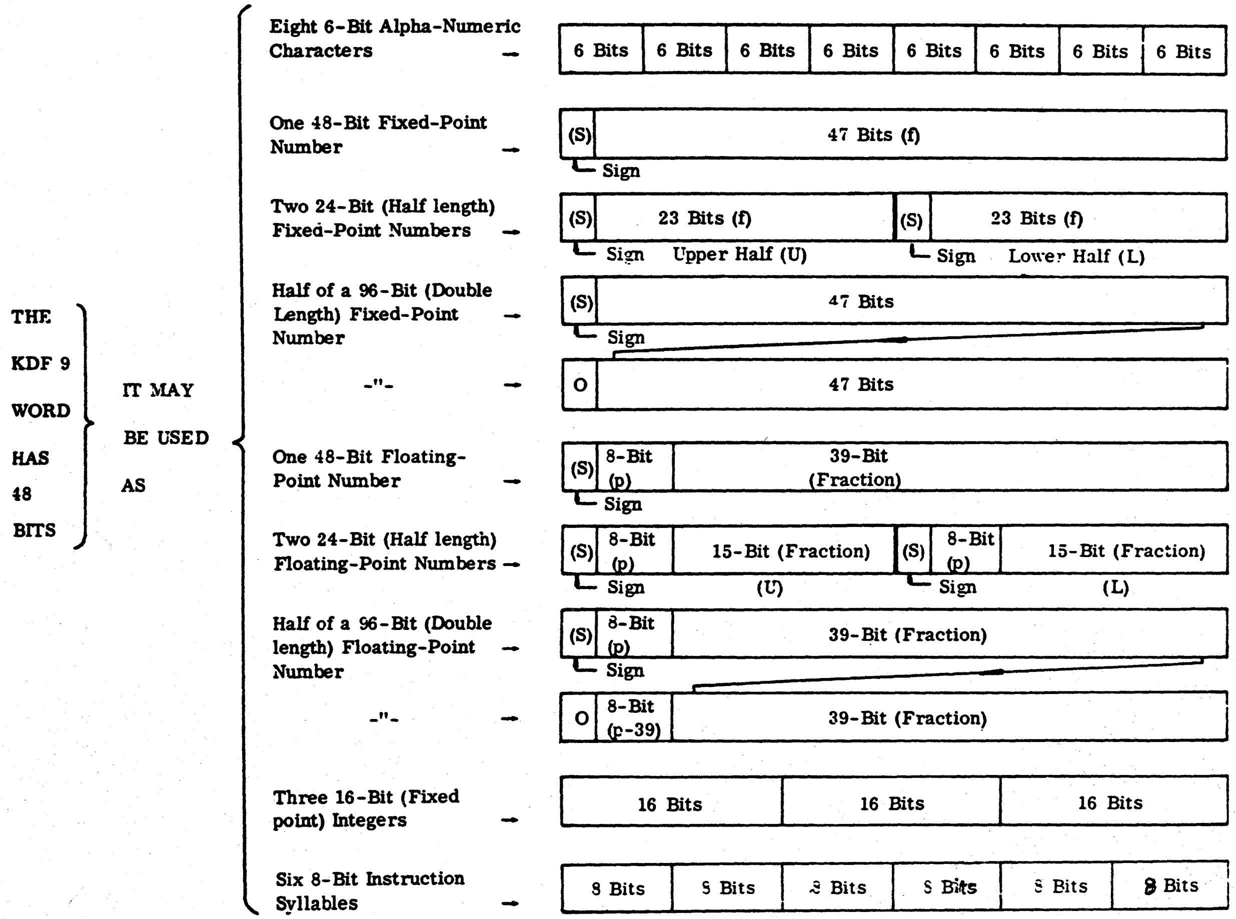
(c) -0·575

(d) 31·5

(e) 0

(f) -0·09375

APPLICATIONS OF THE KDF 9 WORD



THE KDF 9 WORD HAS 48 BITS

IT MAY BE USED AS

Eight 6-Bit Alpha-Numeric Characters → 6 Bits | 6 Bits | 6 Bits | 6 Bits | 6 Bits | 6 Bits | 6 Bits | 6 Bits

One 48-Bit Fixed-Point Number → (S) | 47 Bits (f)
Sign

Two 24-Bit (Half length) Fixed-Point Numbers → (S) | 23 Bits (f) | (S) | 23 Bits (f)
Sign Upper Half (U) Sign Lower Half (L)

Half of a 96-Bit (Double Length) Fixed-Point Number → (S) | 47 Bits
Sign

-"- → O | 47 Bits

One 48-Bit Floating-Point Number → (S) | 8-Bit (p) | 39-Bit (Fraction)
Sign

Two 24-Bit (Half length) Floating-Point Numbers → (S) | 8-Bit (p) | 15-Bit (Fraction) | (S) | 8-Bit (p) | 15-Bit (Fraction)
Sign (U) Sign (L)

Half of a 96-Bit (Double length) Floating-Point Number → (S) | 8-Bit (p) | 39-Bit (Fraction)
Sign

-"- → O | 8-Bit (p-39) | 39-Bit (Fraction)

Three 16-Bit (Fixed point) Integers → 16 Bits | 16 Bits | 16 Bits

Six 8-Bit Instruction Syllables → 8 Bits | 8 Bits | 8 Bits | 8 Bits | 8 Bits | 8 Bits

4· THE BREAKDOWN OF THE KDF 9 WORD

4·1
We are now in a position to understand the various ways that the KDF 9 word may be used.

(a) **Characters** Each word has 48 bits and since each character requires 2 octal digits (i.e., 6 bits) for its representation each word may be used to hold 8 characters of the character code. The reader will often hear the expressions 'left or right justified', which means that, if only, say, 3 characters are in a word then they are placed to either the extreme left or right of that word respectively, with the other 5 characters as spaces (Octal 00).

(b) **Single-Length Fixed Point Number** A fixed point number of 48 bits as explained in Section 3, may be held in one word.

(c) **2-Half-Length Fixed Point Numbers** Sometimes the problem in hand only requires fixed point numbers where 'f' can easily be held in 23 bits or less. Thus 'f' plus D0, making 24 bits, may be held in D0-D23; a similar half length number may be held in D24-D47. Note that integers are stored to 23 integral places for both the half length numbers.

(d) **Double-Length Fixed Point Number** When 'f' requires more than 47 bits, two 48 bit words are used to hold one fixed point number. The sign Digit is in D0 of the more significant word. D0 of the 2nd word is ALWAYS 0 and is ignored. 'f' is considered to be stored in D1-D47 of word 1 and continued in D1-D47 of word 2, so that a pure integer is held to 94 integral places.

(e) **Single-Length Floating Point Number** A floating point number as explained in Section 3 may be held in one word.

(f) **2-Half-Length Floating Point Numbers** As in item (c), above, two floating point numbers may be held in one word. The most significant half word contains 's' in D0 and 'p' in D1-D8 as for a full length number; 'f' is contained in D9-D23. The least significant half word is laid out in exactly the same manner in D24-D47. The values which 'p' can assume are the same as in single length.

(g) **Double-Length Floating Point Number** As in item (d), above, two words may be used to hold one floating point number. The sign digit is D0 of word 1. D0 of the 2nd word is ALWAYS 0 and is ignored. 'p' is held in D1-D8 of word 1. For engineering purposes p minus 39 is in D1-D8 of word 2. 'f' is considered to be started in D9-D47 of word 1 and continued in D9-D47 of word 2. If p is less than 39, word 2 is made into all 0's.

(h) 3-Third Length Integers For special purposes, which will be explained when we deal with Q-stores, it is necessry for a 48 bit word to be broken down into three 16-bit unsigned integers.

(i) Instructions Every instruction in KDF 9 is contained in a group of 8, 16 and 24 bits. In this context 8 bits are called a syllable, so that an instruction takes the length of 1, 2 or 3 syllables.

In general, instructions take the following number of syllables:-

Arithmetic operations 1 syllable.
Shifts, indirect fetches and stores 2 syllables.
Jumps, direct fetches and stores 3 syllables.

The number of syllables for any particular instruction will be found in Appendix 4.

The bits of a syllable are grouped like this:

0 0	0 0 0	0 0 0
B1B2	B3B4B5	B6B7B8

When the computer has executed an instruction, it looks at the next two bits in the store and finds B1 and B2 of the first syllable of the next instruction.

If they are 00 an instruction of	1 syllable	is understood,	
01	2 syllables	"	,
10 or 11 "	3 syllables		,

For example, the instruction

" +; "

is one syllable and stored as

00 101 110.

" -; "

is one syllable and stored as

00 011 110.

The computer senses the 00 and knows it need not look any further than this syllable for the complete instruction. The instruction " $M_K M_Q$ " is two syllables and stored as

01 000 000 kkkk qqqq

where the k's and q's are the binary of K and Q.

Any instruction of 2 or 3 syllables may overlap from one word into the next, as for example, if five syllables of a word are already filled and the next instruction has 2 syllables then one of these will go into the already partly filledword and the second into the first syllable of the next word.

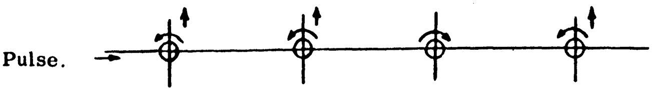
This code, which the computer uses for instructions is called KDF 9 MACHINE (or Binary) CODE and is obtained from the programmer's USERCODE instructions via the COMPILER program. Since this book only aims at teaching the USERCODE the binary form of instructions will not be explained further. What has been said is sufficient for the programmer's needs.

5·1 Main Store

The KDF 9 computer is centred around the main store, which is arranged in modules or blocks of 4096 words, each word containing 48 bits. Up to eight modules may be fitted to the machine, so that the maximum capacity of main store is 32768 words. As mentioned earlier, each bit is in fact a miniature magnetic core. Thin wires pass through these cores for purposes to be explained later.

As far as the programmer is concerned, only the sending of NEW information to a main store word will change its contents. Mere 'reading' of the word does not destroy it. However, from an engineering point of view this is not strictly true; reading a word clears it and immediately replaces it.

Imagine four cores, magnetised, that is, holding information, and with wires passing through them as shown.



If a pulse is sent through the main wire, this will cause all the cores to become magnetised (in effect) in the direction of the pulse thus clearing the information stored. As magnetic changes only take place in the cores holding 'ones' current is induced to flow in the sense wires as shown. It is by detecting these currents that the computer knows what each core held originally, i.e., it has 'read' the word.

The principle of reading data from and writing data into main store as follows.

The computer fetches a copy of the word into a 'transit camp' called the Fetch Buffer, by perforating six operations:

(a) The pulse is sent through the main wire.

(b) a current is induced in the sense wires.

(c) these are sent to the main store buffer, via a gate G which is closed.

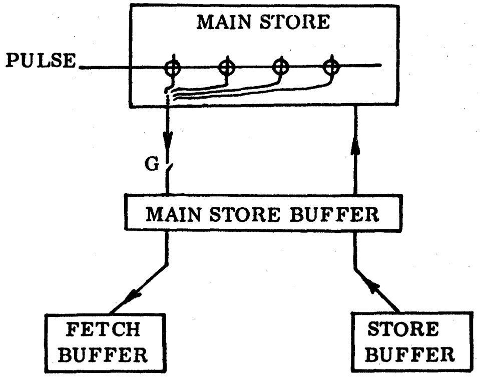
(d) a copy of the word is sent to the fetch buffer from the main store buffer. and the word is pulsed back into the main store from the main store buffer.

We are then left with the original word still in the main store and a copy in the fetch buffer.

To store a word from the store buffer into the main store, the five steps above are repeated but with the following variations.

(a) Gate G is not closed so that the buffer is not affected by the steps (a), (b) and (c).

(b) In step (d) instead of a transfer from buffer to fetch buffer there is a transfer from the store buffer into the main store buffer.



For those interested, it takes 6 microsecs, to perform the Mainstore-Mainstore Buffer cycle, and 10 microsecs. to complete a fetch instruction.

5·1·1 The words in the main store are numbered from 0 onwards. For an installation of maximum size, the words would be numbered 0 to 32767. In the sequel the reader will see that other address notations are also used.

5·2 Peripheral Devices

Information may be transferred into the KDF 9 system and results obtained from the system by means of a variety of input/output devices connected to the main store via sixteen peripheral buffers. Although up to four devices may be connected to each of nine of these buffers and one device to each of the other seven, a buffer can only control one of these devices at a particular instant.

Once a transfer of information to or from main store is initiated, the buffer can see it through to completion, while the main computer can continue with different work. The peripheral devices may include one or more of each of the following.

(a) Paper tape readers, reading characters punched in paper tape in 5-, 7-, or 8-hole code, at a speed of 1,000 characters per second.

(b) Paper tape punches, perforating 8-hole paper tape at a speed of 110 characters per second. These punches can be modified to punch 5-hole tape.

(c) Magnetic tape units capable of transferring information either to or from the computer at a rate of 40,000 characters per second. (These are the 1081 units)

(d) Similar units transferring 77,900 characters per second. (These are the 1085 units)

(e) Punched card readers capable of reading 80 column cards at 600 cards per minute.

(f) Card punches, punching cards at 300 cards per minute.

(g) A Magnetic drum capable of holding 40,000 KDF 9 words and transferring information to or from the computer at a rate of 5×10^5 characters per second.

(h) Four Disc File units, each capable of holding 16 discs, which contain up to 4×10^6 words/unit each. The two exchange rates are: $4{\cdot}8 \times 10^4$ and $9{\cdot}6 \times 10^4$ characters per second.

(i) A graph plotter allowing maximum plot size of 29½" wide on a continuous roll 120 feet long.

(j) High Speed Line Printers, capable of printing lines of up to 160 characters at 1000 lines per minute.

(k) An electric typewriter operating at 10 characters per second. Only one of these may be fitted, which, among other things provides a complete operating log.

Up to sixteen of the peripheral devices may operate at any one time Protective interlocks inside the machine ensure that no two input/output operations can proceed together if they refer to a common area of main store or to a common device. This precaution prevents the occurrence of effects detrimental to the program.

In a similar manner computation may proceed while an input/output operation is in progress, the system of protective interlocks again preventing any possibility of interference between the two processes. Thus no information may be processed inside the machine until the transfer bringing that information into the machine from some input device has been completed.

To assist in the control of peripheral devices a one-bit register, called the 'Test Register', is used enabling the program to interrogate the various devices as to their current state. The necessary information is transferred to the test register from the buffer unit of the device concerned.

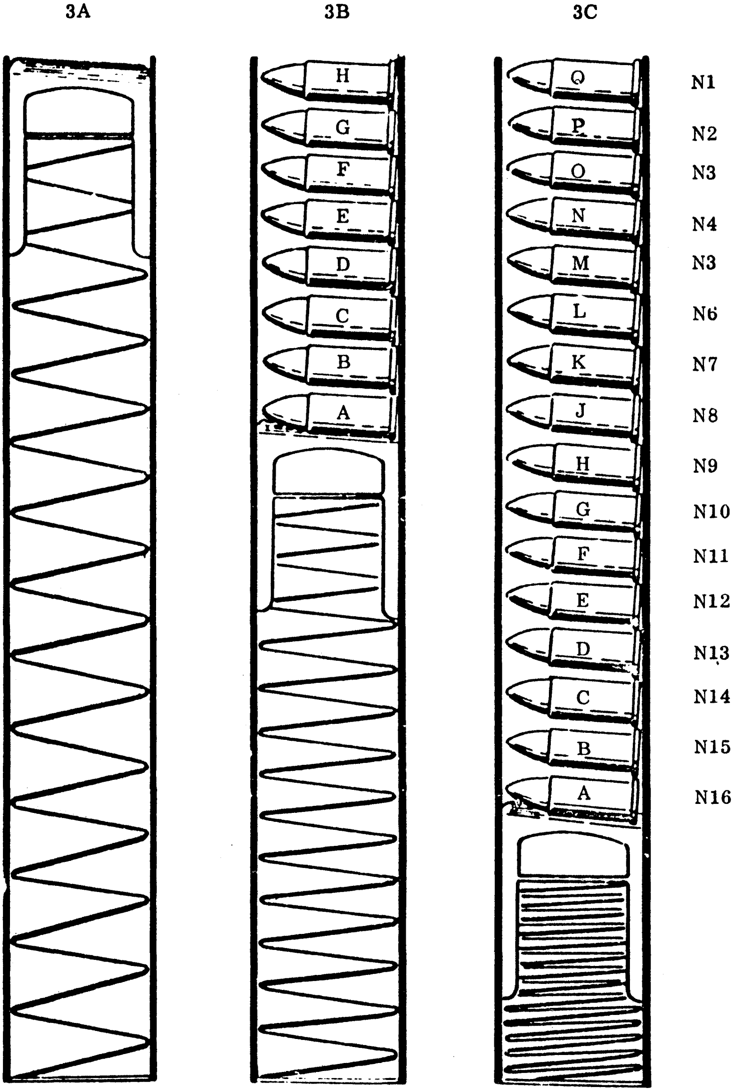
5·3 The Nesting Store

The nesting store of the KDF 9 can hold up to sixteen 48-bit words. The mode of operation of the nesting store is completely different from that of the main store, since the storage of words is organised in a way analogous to that used for bullets in the magazine of a sten gun. (See diagram). At the beginning of a new program the nesting store is empty. If a word, (labelled 'A' in the diagram) is fetched from the main store it is placed in the top of the nesting store, pushing the 'spring bottom' down one unit to make room. Further words fetched from the main store follow the same pattern, each new arrival pushing the rest down one place to make room for itself.

3B of Fig. 4 shows the state of the nesting store after eight words have been fetched, and 3C after sixteen have been fetched. Note the numbering of the cells of the nesting store, N1 to N16 on the diagram. N1 always contains the last word fetched. As there is only one way out of the nesting store, (the top), as with the sten gun, the words must emerge in exactly the reverse order to that in which they were inserted. The ~~word~~ labelled 'A' will be the last out.

The rule for the nesting store is, therefore, "first in, last out", except that there are a few instructions deliberately designed to rearrange items in the nesting store.

Automatic tests inside the machine check that no more than sixteen words have been fetched into the nesting store, and also that a program does not attempt to remove more words than have previously been put in. A contravention of either of these restriction leads to the immediate failure of a program.



Analogy of a Nesting Store

Figure 4.

5·4 Arithmetic Facilities

A comprehensive range of arithmetic operations, shifting operations, logical operations and conditional jump instructions is included in the order code. All of these operate on the top cell or cells of the nesting store. Since the location of the data for these operations is fixed, an operative instruction such as " +; ", to quote a simple example, is quite sufficient to take the numbers stored in cells N1 and N2, add them, and leave the result in N1.

The general rule for the nesting store during any of these operations is that the operands are removed from the nesting store into the Arithmetic Unit, processed, and the result put back into the most accessible cell or cells. If the number of words required for the result is less than the number of words occupied by the original operands, all the words in the less accessible cells that were not involved in the operation will have moved up one or more places to fill the now unused cells, a process known as 'nesting up'. Thus if N1 contains the number 2, N2 contains the number 5, and N3 contains the number 9, all other cells being unoccupied, the state of the nesting store after the instruction " +; " would be N1 contains the number 7 (=5+2) N2 contains the number 9 (previously in N3) N3 to N16 inclusive being now unoccupied.

It must be remembered at all times that any operand used in an arithmetic instruction is removed from the nesting store. Should it be required for a later operation a second copy of it must be made before the first is used. A special instruction exists for this purpose.

Arithmetic instructions may be performed on either single or double-length numbers and these numbers may be either fixed or floating-point. The floating-point arithmetic operations are performed by hardware functions and are programmed in just the same way as their fixed-point counterparts, except that a floating point lable must be added to each arithmetic instruction.

Floating-point instructions in general take longer to execute than the corresponding fixed-point instructions. However, their greater simplicity from the programmer's point of view can lead to considerable economies in time taken to write a working program. For this reason the floating-point facilities are very convenient for scientific calculations.

In all arithmetic operations there is the risk that a result may become too large for the word to hold. Because of this possibility there is a one-bit 'Overflow Register' connected with the arithmetic unit which is automatically set if such an overflow occurs. When the overflow register is set the machine does not automatically stop. The programmer should include overflow register test instructions at suitable points in his program together with appropriate remedial routines.

5·5 The Q-Stores

Information can be transferred between the top call of the nesting store and one of a set of fifteen Q-stores numbered Q1 to Q15. An extra store, Q0, may be used by the programmer, but for certain special reasons it always has the value zero. A fetch from Q0 puts the value zero in N1, while any quantity sent from N1 to Q0 is lost, the contents of Q0 remaining identically zero. Each of the remaining fifteen Q-stores consists of a 48-bit fast access register. These stores may be used for a variety of purposes during the running of a program. These uses include temporary storage of data or results when their presence in the nesting store would be inconvenient, and the storage of information which is obtained by calculation within a program but which is required for the execution of certain instructions. For this latter purpose the Q-store is often required to hold three independent 16-bit binary integers. When it is dividied into three parts in this way, the sections are known respectively as:-

(a) The COUNTER (Digits D0 to D15)

(b) The INCREMENT (Digits D16 to D31)

(c) The MODIFIER (Digits D32 to D47)

Instructions are available for operating on each of the three parts individually. No operation on one part can affect either of the other two, i.e., no "spill" from one part into another is allowed.

5·6 The Control Unit

The control unit exercises control over all parts of the machine. It extracts instructions from the main store as they are required, examines each in turn, and initiates the appropriate actions. The instructions are obeyed sequentially in the same order as they are stored until a transfer-of-control instruction (i.e., a jump) is encountered, in which case the sequence is broken and resumed at another point usually specified in the control transfer instruction itself.

5·7 The Subroutine Jump Nesting Store

The subroutine jump nesting store, usually abbreviated to SJNS, is used automatically by the machine to store the address to which control must eventually return when a subroutine is about to be entered. Since second or higher order subroutines are quite often needed and since the return address for the last one entered is required first, a nesting store is ideal for this purpose because the return addresses always emerge in the correct order.

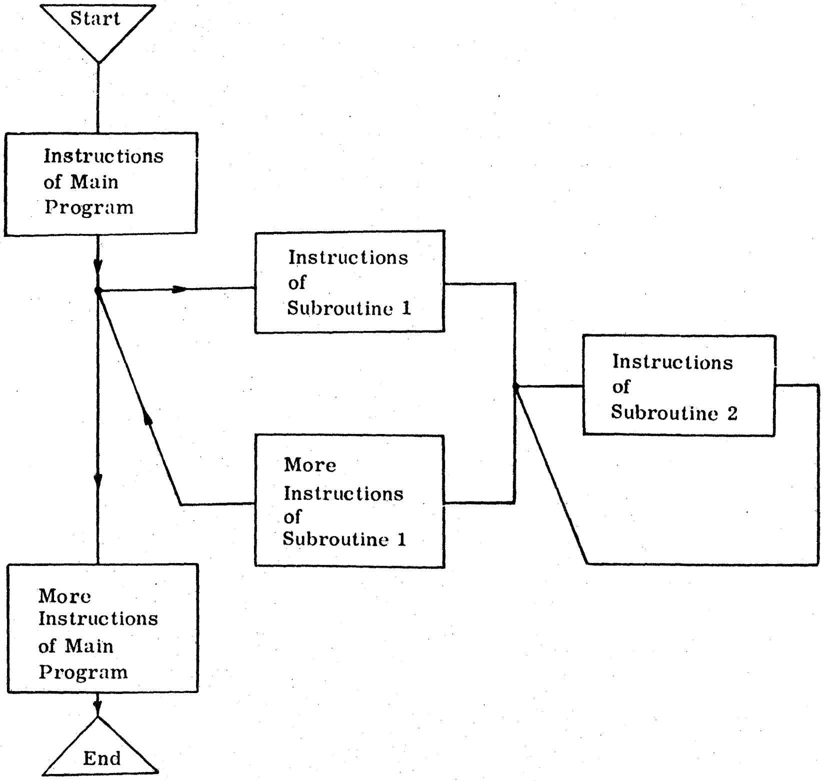

Start
Instructions of Main Program
Instructions of Subroutine 1
Instructions of Subroutine 2
More Instructions of Subroutine 1
More Instructions of Main Program
End

DIAGRAM OF PROGRAM USING 2ND ORDER SUBROUTINES

Sixteen cells are provided in the SJNS but programmers are recommended to restrict their use to fourteen cells, leaving the remaining two for use by certain control programs normally in use on the machine. This arrangement allows a programmer to use subroutines up to the fourteenth order and should present no practical restrictions. Communication is provided between the top of the SJNS and the ordinary nesting store so that extra addresses may be inserted or surplus ones removed.

Each cell of the SJNS has 16 binary digits, of which the first three represent the syllable number in the range 0 - 5. The remaining 13 hold a word address in the range 0 - 8191.

All instruction addresses in KDF 9 are of similar layout, leading to a rule that all instructions in a program must be within the first 8192 words allocated to that program - the rest of the store can, of course, be used for data. The size of the Director program does not reduce the limit of 8192 for other programs - when the instruction word address (13 bits) is extracted by the control unit, it adds the necessary correction factor (depending on where the first word of the program has been placed) into a 15 *bit register, thus allowing any possible address on the final result.

*2^{15} = 32768, thus 15 bits can accommodate the integers 0 to 32727 which are the main store word addresses of the maximum sized installation.

6· PROGRAMS

6·1 Mnemonic Significance Of USERCODE

Throughout the KDF 9 Usercode the individual instructions have been kept as short as possible, whilst at the same time they have been given some mnemonic connection with the operation required. Where a conventional mathematical symbol is available, it has been used to express the corresponding instruction in one symbol which is recognisable to all. This is possible for such instructions as 'multiply', 'divide', etc. For the other instructions the name of the operation, or an abbreviation of the name, has been used. Wherever possible the letters I and O have been excluded from these mnemonic forms, because of possible confusion with the figures 1 and 0. Spaces occurring between symbols are ignored by the Compiler program.

6·2 Example of a KDF 9 USERCODE Program

The example program which follows will be referred to on occasions throughout this manual. The program, although correct, is not intended to be the most efficient method to adopt; it is purely for the purpose of illustration, and should be referred to whenever a new instruction in the sequel is encountered.

Specification:

(a) To read from paper tape up to 99 integers each less than a million; the data having been punched in the format:

(1) (07)$_8$ (02)$_8$ N N N N N N
where there must be 6 N's e.g., 27 would be
punched as (CN) (CRLF) 000027

(2) After the last integer is to be (CN) (CRLF) →

(b) To sum these integers.

(c) To print the sum on the line printer.

```
P
KEABOD100UP1
SUM INTEGERS →
ST300; TL3;
V23; YD 124; YP2;
PROGRAM;

            V0   =  Q  O/AYDO/AYD99;
            V1   =  B  30;
            V2   =  P  ⌊7DC⌋
            V3   =  Q  O/AYPO/AYP2;
            V4   =  Q  100/1/0;
            V5   =  B  171717171717;
            V6   =  B  1212121212121212;
            V7   =  B  2020202020202020;
            V8   =  B  070275/17;
            V9   =  B  0702/11;
            V10  =  P  PARITY ⌊2D⌋;
            V11  =  P  DATAFAIL;
            V12  =  P  DATAEXSS;
            V13  =  B  7777/11;
            V14  =  B  404040404040;
            V15  =  B  202020202020;

            V0; =Q1; SET2; SET5; OUT; =C1; PREQ1; V4; =Q2;
            ZERO; V8; V2; =YP2; V1; =YPO; PARQ1; J101TR;
            C1; SET6; OUT;

*1;         YDOM2Q; DUP; PERM; J2=; (CHECKS FOR LAST);PERM; DUP; V13;
            AND; V9; -; J102≠Z; (CHECKS ON CRLF); DUP: V14; AND;
            J102≠Z; (CHECKS NO EXCESS 32); DUP; V15; AND; BITS;
            SET6; -; J102≠Z; (CHECKS FOR EXCESS 16); J103C2Z;
            (LEAVES IF DATA EXCESS); V5; AND; V6; REV; TOB; +; REV; J1;
101;        ERASE; ERASE; V10; C1; SET6; OUT;
8;4;        =YP1; ZERO; DUP; =YP2; =YP0; J3;
```

```
2;          ERASE; ERASE; V6; REV; FRB; V7; OR; =YP1;
3;          V3, SET8; OUT; ZERO; OUT;
102;        ERASE; ERASE; ERASE; V11; J4;
103;        ERASE; ERASE; ERASE; V12; J8;
            FINISH; →
```

Explanation
The section between the initial P and PROGRAM; is called the Program Heading, which is followed (up to FINISH;) by the Program Body. The items V0= etc., up to and including V15=. . . ; are called the Constant declarations.

The instructions of the program commence at V0; =Q1 etc., and continue to the end. Every instruction MUST be terminated with a semi-colon. Notice that the "spaces" are ignored so that the two following instructions are the same

```
V3   =   Q 0/AYP0/AYP3
V3   =   Q0/AYP0/A YP3
```

6·2·1 **Jumps** Instructions are obeyed in strict sequence, until a "jump" instruction is encountered, when the sequence will be broken. Consider the instruction J101TR; this means jump to reference label 101 if the test register is set, otherwise continue in sequence.

6·2·2 **Reference Labels** It is necessary to indicate to the machine the point to which jumps are to be made. The technique of counting so many syllables backward or forward has not been considered because of its extreme fallibility and because the count would have to be corrected whenever the program is adjusted. Instead, provision is made for any instruction to carry one, or, if so desired, more than one reference label. All control transfers indicate their point of resumption by naming the appropriate reference label. These reference labels are always numeric and may take values from 1 to 1,011 inclusive. The actual order in which the labels appear in the final program is immaterial. A given reference label may be used only once although any number of control transfer instructions may indicate a given label as their point of resumption. Any duplication of reference label will be detected by Compiler and a failure indicated.

The reference label is written in front of the instruction to which it refers, and is separated from it by a semi-colon. The semi-colon is the separator normally used between all items in the User Code. One label may be preceded by another, if desired, separated by a semi-colon. The first instruction of a program is automatically given the label "0", but if the program requires a jump to that point it must have a reference in the permissible range and the jump instruction must refer to this latter label.

6·2·3 **The Asterisk** In certain circumstances it is necessary for the programmer to ensure that a particular instruction starts at the first syllable of a main store word. To avoid the necessity of counting the number of syllables used in a program, with all the attendant risk of error esspecially if the program is later modified, the asterisk facility is provided. Compiler ensures that any instruction preceded by an asterisk will be compiled as the first instruction in a new word, any redundant spaces in the preceding word being filled with dummy instructions. If such an instruction also requires a label the asterisk should be written before the label, since Compiler will then compile a more efficient program.

6·2·4 **Manuscript and Typescript Conventions** When writing User Code programs it is recommended that a column be reserved on the left-hand side of the sheet for the labels, for easy reference when it is required to trace a control transfer instruction. Apart from this convention, User Code instructions, separated one from the next by a semi-colon, are written one after the other along a line. A new line may be started at any time, but it is recommended that this be done to separate the various stages in the logical structure of the program whenever possible. With this kind of layout the program may be more easily followed after it has been written. Punch operators should be instructed to follow exactly the layout of the program in manuscript, starting a new line as and when the manuscript version does. In this way the editing characteristic of the manuscript are preserved as carriage-returns etc., in the papaer tape version, and if the program is later reprinted from the tape the original format is precisely reporduced.

To prevent confusion the programmer should write:

letter O	as	O	number 0	as	Ø
I	as	I	1	as	1
Z	as	Ƶ	2	as	2
S	as	S	5	as	5
			7	as	7

Semi-colongs should clearly show the dot above the comma.

The programmer is advised to write his program in pencil on alternate lines - erros should be neatly crossed through or completely erased. Insertions can be made on the spare lines.

6·2·5 **The Comment Facility** Comments may be inserted at any stage of a User Code program provided each occurs between the semi-colon terminating the previous instruction and the next instruction. These comments must adhere to the following simple rules:-

(a) Each comment must be enclosed in round brackets.

(b) No comment may include a semi-colon or an End Message symbol.

(c) Any round bracket opened during the course of a comment must have the corresponding closing bracket, this implies that comments inside comments are allowed.

(d) Each comment must terminate with the closing bracket followed by a semi-colon.

(e) Although there is no limit to the length of comments, the programmer is advised to keep them short, if possible using only one letter referring to full comments kept in his notebook.

When Compiler detects an opening round bracket immediately following a semi-colon, it recognises that it has found a comment. It then ceases compiling while it scans the subsequent characters for opening and closing brackets, keeping a tally of them until the final closing bracket is identified. Then it checks again for a semi-colon, the detection of which signifies the end of the comment. Compilation is then resumed at the next instruction. This facility enables the course of a program to be described at the same time as it is written, the comments appearing with the instructions on the same tape. Note that these comments do not appear on the compiled Machine Code program tape.

6·2·6 **Finish and Zero Out** The Compiler program translates each Usercode instruction into Machine code, one at a time, in strict sequence, only ceasing when it senses FINISH;. The instructions of the machine code version are NOT obeyed at this stage. Because the last instruction to be obeyed may not be the last one written, the instructions ZERO; OUT; must be written to indicate the program's dynamic end.

After ZERO; OUT; in the Example program are two lines of instructions which are entered by "jumps" from somewhere above and which are left by jumping to before ZERO; OUT; so that in all cases ZERO; OUT; will be the last instructions to be obeyed.

6·2·7 **Fetch and Store** To fetch a copy of the word in address V4 (say) into the top cell of the nesting store (i.e., into N1) the instruction is V4;

To store the contents of N1 into YP2 (say) the instruction is =YP2;

The reader is not expected to understand anything further of the coding of this program at this stage.

6·3 Library Extraction

KDF 9 has a wealth of Software wherein are many self contained sets of instructions. These are housed on a special magnetic tape called NINEMASTER, and called Library Subroutines.

The programmer can save himself much time if he uses these routines instead of working out his own. He will find their specifications in the Service Routine Library Manual.

Let us suppose that between the two instructions:

REV; and =YP1; a routine is required which is the library subroutine numbered L59, all the programmer need to do is insert JSL59; between the above instructions, and just before FINISH; write

library L59;
(See later re EXIT from the subroutine back to the main program.)

This process, called Library Extraction, may, however, only be employed if the program is to be compiled by the Compiler of the system called POST.

6·4 Use Of The Main Store

When the machine is switched on, the first step is to store the Director program in the Main Store, starting at main store word address 0. The Director program is written (filled out with dummy instructions if necessary) so that it is an exact multiple of 32 words. Say it is 3200 words long and stored in main store word addresses 0 to 3199; then the rest of the main store, from 3200 onwards, is available for other programs' instructions and data.

There is more than one kind of Director, each of different sizes, used for special features of KDF 9, and the programmer is never certain which Director is in use when his program is run. Hence he cannot refer to words by their main store addresses when writing his program. To overcome this he writes the program as a self contained unit and uses what is called "Program's Absolute addressing". This system is a number, preceded by a letter E, so that the first word of his program is E0, the second E1 and so on to the end of his program and data areas.

After Director is fed into the computer and it has called for other programs, it will place these programs in the main store and keep a note of where the E0 of each has been placed. Note that E0 is always placed in a Main Store Word whose address is an exact multiple of 32. The form of Director which only provides for one program to be run at a time places the program fed in, such that its E0 address is immediately after the last word of Director.

When the programs are run Director modifies the E addresses of the program by adding the main store address of E0. So that if E0 is in Main Store Word 3200 and the program refers to E8, the machine understands this to mean store word address 3208.

6·4·1 Main Store - Instructions and Data The first eight words (E0 - E7) of a program are used for organisational procedures and will be dealt with later. From E8 onwards are the constant declarations followed by the instructions, the remaining words of the store are reserved for data.

When beginning to write a program, the programmer will not know the exact number of words his program will need, so he refers to the data storage areas symbolically. It is then left to Compiler to determine the space necessary for the instructions, and to add a correction factor determined by the number of instructions used, so converting the symbolic addresses to absolute addresses.

Normally the first word of the data storage area is referred to in User Code by the symbolic form YO, the subsequent words being Y1,Y2,Y3, etc. For some applications one set of data storage locations is not sufficient. User Code, therefore, allows the additional forms YA0, YA1...., YB0...., YC0...., and so on up to YZ0....,. The forms Y00...., YI0... are not allowed because of the risk of confusing the letters 0 and I with the numbers 0 and 1. It is also recommended that the forms YU and YV should not be used, since these are reserved for possible use in certain control and diagnostic routines. There are, therefore, 22 of these alternative sets in addition to the main Y set.

It is possible that an area of main store will be required as working space by a large subroutine. For this purpose stores known as W-stores are provided, numbered W0, W1, W2 etc. It should be remembered that these W-stores are common to all subroutines and should not be used for the permanent storage of information, since one subroutine may destroy the information left in the W-stores by a previous subroutine.

It is often necessary for a program to require certain contants during the execution of the program. User Code provides facilities for these, and a set of V-stores, numbered V∅, V1, V2, etc., are available for this purpose. Chapter 7 explains their use in greater detail.

Finally, it is repeated, that addresses of the program may also be referred to "absolutely" by using E0, E1, etc.

For the purpose of this manual the form Y_y will be used to represent any one of these possible forms, where y represents any integer. Wherever Y_y is used any one of the alternative V_v, E_e, W_w, YA_y, YB_y...YZ_y is permissible. The sizes of the integers e, w, y are limited only by the total capacity of the main store.

A point to bear in mind is that an address such as Y-49,. meaning the 49th word before YO is valid provided that its equivalent E address is not negative. However this method of negative addressing has no practical programming use unless it is known what are the contents of the words involved.

6·5 Operation

The KDF 9 User Code Compilers which exist will accept User Code programs either from paper tape or from magnetic tape, and will then process them character by character to generate the equivalent program in machine code instructions. During the compilation process, the instructions are checked in turn for agreement with the permissible User Code forms, translated into machine code, and stored in consecutive locations of the main store.

If an error is found it is reported and the Compiler continues working through the program, to check for further errors. Therefore at the end of one compilation run, either the correct machine code program is produced or a complete list of all invalid instructions is given. A second compilation run with a corrected input tape should result in a valid machine code program.

At the end of compilation the program in main store is either run or transferred to punched paper tape or magnetic tape as required, when it will be in the correct form for subsequent input by the Director program loading routines. The Compiler will require the main program to appear at the beginning of the input tape, preceded only by such declarations as are required. The main program is followed by any subroutines it needs; but where the library of standard subroutines is available on magnetic tape, these may be called for automatically and will not need separate presentation on the input tape if the POST system is being used.

The time taken to compile a User Code program depends on the number of instructions involved, but a rough estimate would be about twice the time taken to read the input tape, plus an allowance for the output. This output time is negligible for magnetic tape, but will be very much longer should paper tape output be required.

6·6 Layout Of Information In Main Store

Imagine the main store words to be arranged across the page in lines of 32 words, each line starting with a main store word address which is exactly divisible by 32. The main store will hold programs following the pattern below.

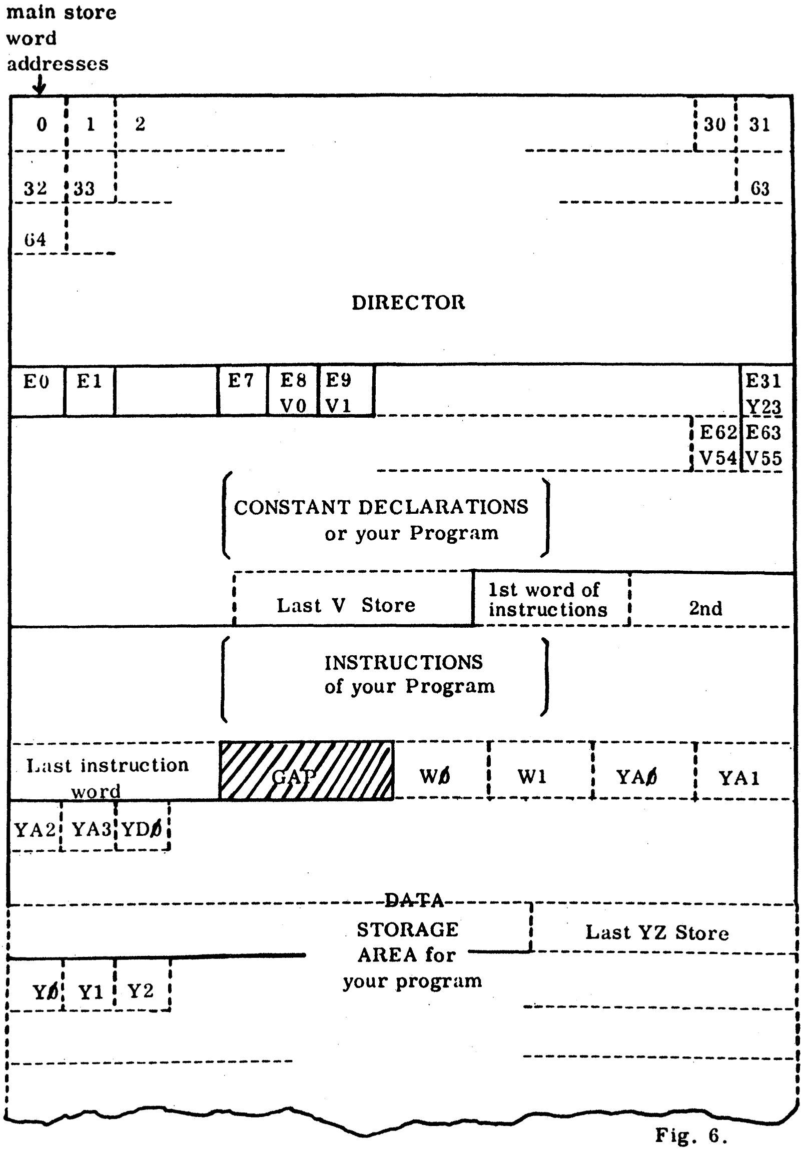


Fig. 6.

E0 - E7 is placed immediately after Director.

V0 is placed in E8.

The first instruction is placed immediately after the last V-store.

There is then a gap of possibly up to 31 words, followed by the W, YA, ... , YZ store, as required, each immediately following the other. The last of these required stores is followed by Y0, which will have an E address which is an exact multiple of 32.

6·7 Program Heading

The format of the program heading depends on the method used to compile the program. That given in the Example, Section 6·2 is for the Paper Tape Compiler.

P — indicates that the program to be compiled will be found on paper tape.

KEABCD100UP1 — is the identifier of the program. The first 2 characters are assigned to establishments and the following 5 are at the programmers' disposal. Characters 8 & 9 are known as the amendment number This is automatically incremented by 1 everytime a modification is made to the program. U stands for Usercode, P for Program, and the last character indicates how many modules of main store are required.

SUM INTEGERS — This is the title given by the programmer.

→ — indicates the end of the 'A' blocks (to be defined later).

ST300; — indicates that the program's instructions and data do not need more than 300 words, however the number is rounded up to the next multiple of 32. In this example E0 - E319 would be made available.

TL3; — indicates that the program is to be terminated if more than 3 seconds of running time have elapsed.

V23;YD124;YP2; — is an indication of the address of the last of all classes' symbolic addresses used in the program. The address of the last Y store must not be quoted. Only those required in the program need be specified. The first request must be the V stores, the next (if required) the W store, then follow the special Y stores in any order.

So that the "Instructions" may start at the "Beginning of a linc", (the reason for which will be explained when lock-outs are dealt with) it is wise to request the number of V stores which will terminate at the end of a line e.g., V23, V55, V87 etc. In our example the last V store used is V15, so we request V23; this will cause V16 to V23 to be set at zero. Similarly to end the gap at "the end of a line" the W, YA-YZ stores requested should total a multiple of 32. Note that by requesting, say YR31, we have in fact requested YR0-YR31, i.e., 32 YR-store.

6·8 Exercises - Set 1

1. If the main store word address of E0 is 1024 and E512 is called Y0

(a) What is the absolute address of Y19?

(b) What is the main store word address of Y0?

(c) What do we call E517?

(d) What is another name for the word called Y32?

(e) What is the main store word address of Y32?

(f) If the word whose main store word address is 1236 is called YA5, what is another name for YA0?

2. What is the absolute address of V0 in any program?

3. If a program contained the two consecutive instructions: V4; =E12; what would be the effect?

4. What is the maximum number of words which the "gap" after the instructions may be, and why?

5. Is it ever possible for E0 to be placed in the main store word address 1183? Explain your answer.

Exercises - Set 2

1. If the main store word address of E0 is 2048 and E544 is called Y0, and if Y35 contains the number x = $128_{(10)}$ as a fixed point number held to 47 integral places; what are the contents of the main store word address 2627 expressed in octal?

2. What are the contents of the same main store word address specified in the previous question if the contents were the same number held in "Standard Floating Point"?
Express your answer in Octal.

7· CONSTANT DECLARATIONS

7·1 Definition of Constants

Most of the words used in a program will contain different information every time the program is run. For example, such items as, the balances of accounts, the date, the value of a parameter in an equation, all will vary depending on the data fed in. Those numbers, words, etc., which stay constant independent of input data, such as the word "Debit" and "Credit", the value "2" in the equation Y = 2x, etc., are termed "CONSTANTS".

Since time would be wasted if the binary patterns of these constants had to be completely regenerated whenever the program is to be run, the concept of CONSTANT DECLARATIONS is used. These are special instructions in USERCODE which cause the Constants to be converted into binary by COMPILER when the Usercode version of the program is translated into Machine Code.

The actual instructions which introduce these constants into a program are of two distinct kinds. The first kind puts each constant into a V-store, from which it may be recalled any number of times during the operation of the program. The declaration of a constant for the V-stores takes the form Vv= (the appropriate quantity); The letter v represents the number of the particular V-store involved.

The second method of introducing constants is by use of the instruction "SET (followed by the constant);". This causes the constant to be placed into the first cell of the nesting store ready for immediate use. The constant used with SET must not require more than 16 bits for its binary representation.

7·2 Compiler Actions

A statement in the program heading informs Compiler how many V-store words are to be reserved for that program. When each constant declaration is encountered during the initial translation run, the corresponding binary pattern of a V-store declaration is generated and stored away in the nominated V-store. In the same way, all the other USERCODE instructions are converted into their binary code form and stored in the Main store. When one of these instructions is a SET instruction the constant required is converted into its 16-binary digit form, which then forms part of the 24 bits (3 syllables) necessary to hold the machine code version of the SET instruction. This is a pure instruction, and is placed in its correct sequential position in the instructions.

When all the instructions and constant declarations of the program have been translated and placed in their appropriate positions in main store by compiler, that entire area of store from E0 to the last instruction may be output, in its binary form, on paper or magnetic tape etc. Thus when this record of the machine code (binary) program is fed into the machine at a later date, the storage area allocated to the program is filled with the constants and instructions in their correct positions.

It is to be noted that the constant declarations themselves, in the form of Vv= etc., do not appear on the machine code tape. It is recommended that the constant declarations should all be written in order at the beginning of the Usercode program, immediately following the program heading. This serves to emphasize that they are dealt with on compilation and not at run time, and also makes it easier for the programmer to keep track of the values he assigns to the individual V-stores while the program is in preparation. As the programmer may not know what V-store constants the program will require until he encounters their need whilst writing the program, he is advised to write them on a separate sheet of paper, completing it as necessary.

Although V-stores are intended to hold CONSTANTS they may be used in just the same way as the Y-stores, etc. However, once the original information in a V-store has been overwritten with other data, it can never be retrieved. Consequently it is strongly recommended that a program should never cause a V-store to be overwritten except in very special circumstances.

The various forms that a V-store constant declaration may take now follow:

7·3 Numeric Constants

A numeric constant for a V-store will normally be declared as a decimal number. In the KDF 9 system a decimal number z is defined in the following manner:-

$z = (\text{sign})\ I\ .F_{10}\ (\text{sign})\ E.$

The individual parts of this expression have the following meanings:-

(sign) :- either of the symbols + or -. If the sign is omitted in either of the two positions where it may appear, a + sign is assumed.

I :- a decimal integer.

F :- A decimal point followed by a decimal fraction.

E :- An integer exponent giving the power of 10 by which the number must be multiplied to obtain its true value.

The parts I and .F may be omitted individually or both together, when $(\text{sign})_{10}$ (sign) E by itself is valid The exponent part $_{10}$ (sign) E if used, must be written in full; otherwise it must be the parts that remain must be in the order specified by the definition.

For example, the number 746 may be written in any of the following forms:-

746 $\quad 74.6_{10}1 \quad +7.46_{10}+2 \quad +7460_{10}-1$

In fixed-point working it is necessary to be able to specify the number of integral places required for each number. This may be done in the declaration by following the number z by the symbols /p, where p is an integer indicating the position of the binary point. It should be noted that for a fractional number the position of the unit digit may be beyond the more significant end of the 48 bit word, in which case p is given as a negative number. Again if the number is very large, p may be given a value greater than 47, so that the most significant 47 binary digits are stored, the remaining digits being lost.

As it is expected that integers will form a large proportion of the constants used in many programs, their declaration has been specially simplified. If the symbols /p are omitted the number z will be treated as an integer and automatically assigned the appropriate number of integral places in the binary scale, so that it will be stored at the less significant end of the 48-bit word i.e., p = 47. In double length fixed point the missing p is interpreted as /94. In half length /23 is understood.

The four possible forms for the declaration of a numeric constant will now be listed. The abbreviation Vv means the V-store constant numbered v, and z is a decimal number as defined above.

Vv = z/p; :- a single-length numeric constant z given to p integral places.

VvD = z/p; :- a double-length numeric constant z given to p integral places. (When working double-length it is convenient to consider D1 of word 2 to be D48 of word 1 - not D0 of word 2 which is ignored).

Vv = Fz; :- A Floating-point single-length numeric constant. In this case the symbols /p are not necessary since the number is automatically put into standard floating form.

VvD = Fz; :- a double-length Floating-point numeric constant. The symbols /p are omitted as in the single-length case.

Examples

V0 = 28; gives 28 to 47 integral places.

V1 = 49/6; gives 49 fixed point to 6 integral places.

V2 = 74·6; gives rounded integer result i.e., 75 fixed point to 47 integral places.

V3 = 1/0; gives failure indication from Compiler because 1 cannot be held to zero integral places.

V4 = $_{10}$-13/-3; gives +10^{-13} to -3 integral places.

V6D = F-6·3$_{10}$-4; gives -0·00063 in Standard Floating Point, with the most significant word in V6 and the least significant in V7.

V8D = 32·2/90; gives 32·2 fixed point with the point after D43 of V9. Note the V7 has not been used as a separate declaration; it was used in the declaration V6D = etc.

V10D = -240; gives -240 fixed point with the point after D47 of V11; i.e., it is held to 94 integral places.

7·4 **Binary Constants**

Any binary pattern may be expressed in constant form for use in a program, but for economy of space in writing it out, the octal system is used in its actual expression. Thus a maximum of 16 octal digits will express a 48 bit binary pattern of any configuration. A binary constant is always expressed in integer form. If fewer than 16 octal digits are required a space will automatically be left at the more significant end of the word in which it is stored. However, should the constant be required at the more significant end of the register, with zeros at the less significant end, the declared octal number may be followed by the symbols /p. The least significant digit expressed in the specification will then be put into position p, all register positions below this being left as zeros. In general the integer p may be chosen to position the binary pattern anywhere along the register.

The declaration of a binary constant takes the form:-

Vv = Bt/p where B is the label for a binary constant, t is the binary integer expressed in the octal form, p is the digit position of the least significant bit expressed

As before, if the symbols /p are omitted a value p = 47 is assumed. A failure will be reported if any non-zero bit is lost off either end. Double length binary constants are NOT allowed.

Examples

V0 = B4142434445464750; gives the character code of ABCDEFGH in V0.

V1 = B75/5; gives→ as the first character in V1, the remaining seven characters will be spaces (Octal 00).

V2 = B30; gives seven spaces followed by the character code representation of the digit 8.

V3 = B21/41; give six spaces followed by the representation of the digit 1 followed by one space.

V4 = B4/4; puts a 1 bit in D2 and O's elsewhere.

7·5 Address Constants

It is often necessary to know the actual address of the main store word at which a particular quantity is stored. Since such addresses are not known until the program is compiled, it is reasonable to expect Compiler to provide this information where required. So in User Code programs the addresses of main store words are written symbolically, the absolute addresses being substituted by Computer on compilation. Each address obtained in this way defines both the word and the syllable number of the location, so both data and instructions may be located precisely in the main store. If a data address is called for, the syllable number given will always be zero since an item of data is always stored starting at the beginning of a new word. An instruction on the other hand may begin at any syllable of a word.

The form of the declaration for an address constant is

Vv = AYy; where A is the label for an address constant.
Yy is the symbolic address of the word required; y being an integer

This will cause Vv to contain an integer identical to the absolute address.

The address Yy may be replaced in this declaration by any of the following valid forms of address:-

(a) YAy, YBy....YZy excluding YIy and YOy;

(b) Ww, Ee.

(c) Vv, VvPp, VvLl;

(d) Rr, Pp, Ll, RrPp, RrLl. (Explained in a later section)

The action of Compiler when dealing with an address constant is as follows. The E address equivalent of the symbolic address is calculated and the integer obtained is placed, (held to 47 integral places), in the appropriate V store.

When it is required to enter the address of either the upper or lower HALF of a word the address is to be followed by U or L. Upon sensing the U or L, Compiler doubles the E address-integer and then adds 1 if the lower (L) half of the word is being referred to.

Since the maximum store has 32768 words (0-32767) 16 bits are necessary to hold the largest address-integer possible, viz., E32767L.

To illustrate by examples; the declarations

(a) V1 = AY14;

(b) V2 = AY14L;

(c) V3 = AY14U;

(d) V4 = AE32700L;

would give the following address integers, if Y0 is assumed to be coincident with E640.

(a) 654 - calculated as 640 plus 14

(b) 1309 - calculated as 640 plus 14, then doubled, and then 1 added for the L.

(c) 1308 - calculated as 640 plus 14, then doubled.

(d) 65401 - calculated as 42700, then doubled and 1 added for the L.

Suitable instructions exist to enable the machine to comprehend the address integer 1309 as either AE1309 or AE654L as is appropriate.

Any of the address constant forms in (a), (b) or (c) above may be modified with U or L.

Addresses of the form in (d) overleaf, are all INSTRUCTION addresses and will always appear in syllable/word number form, as required for the jump nesting store.

7·6 Q-store Constants

It has been mentioned in Para. 5·5 that a Q-store will often hold three independent 16 bit signed integers, and that for this reason it may be referenced as three integers c, i and m. c is the counter, stored in bits 0 - 15; i is the increment, stored in bits 16 - 31; and m is the modifier, stored in bits 32 - 47.

The declaration of a Q-store constant takes the form:-

Vv = Q c/i/m where Q is the label for a Q-store constant.

c, i, and m, may represent signed integers limited to the range -32768 to +32767. This range is the greatest that can be accommodated in 16 bits. The integers stored in c, i or m may be used to represent signed integers as such, or an absolute word address in which case the 16 bits are taken as containing an unsigned 16 bit address-integer.

A typical declaration where i and m are to contain address integers is of the form Vv = Qc/AYy/AYy_2; where y and y_2 are **Y**-store addresses. Notice the A preceding the address: this indicates that the position holds the integer of the E address equivalent to the address quoted. i.e., V7 = Q 2/5/AY9L; would in effect be V7 = Q2/5/83; if E41 were identical to Y9.

The valid forms of address given in paragraph 7·5 may be used in place of Yy.

Example
The declaration to set a Q type constant, with c=100, i=AYP0, m=AYP99, into V9 would be:

V9 = Q100/AYPØ/AYP99;

Note that the Q-store constant does NOT put the constant into a Q-store, it only arranges at this stage to put the word into V9.

7·7 Half-Length Constants

Facilities exist on KDF 9 for half-length fetching and storing, and so provision has been made for the setting of half-length constants. With one exception, any kind of constant may be stored as a half-length constant. The exception is the Q-store constant which does not lend itself to half-length manipulation. The procedure for setting a half-length constant in some V-store word is, first, to specify the constant itself, remembering that it may not exceed a length of 24 bits, and then to state whether it is to be stored in the upper (more significant) or lower (less significant) half of the destination word.

The two forms for a half-length constant declaration are:-

VvU = (specification). This will be stored in the upper half (D0 - D23) of the constant store v.

VvL = (specification). This will be stored in the lower half (D24 - D47) of the constant store v.

In these two declarations, (specification) may take any of the forms given in paragraphs 7·3, 7·4 or 7·5 above.

Notice that for an integer, p requires to be 23 for either an upper or lower half length declaration.

Examples

V19U = 15/23;

V19L = F16.1;

V20U = B12121212;

```
V20L  =  B75/5;
V21U  =  AY28;
V21L  =  AV19U;
```

7·8 P-Constants

A single binary constant declaration of paragraph 7·4 only permits the setting up of one word of store. When a longer string of characters is required it is possible to set them into the store words by one declaration known as a P-Constant declaration, when it also is not necessary to convert the characters to their octal form. There are however restrictions as indicated below:

The general format of a P-constant declaration is:

Vv/w = P (string of items);

Vv/w specifies the area of main store, from Vv to Vw both inclusive, which is to contain the items in the string, counting 8 items per main store word.

P tells Compiler that what follows is to be compiled in its octal representation.

(string of items) may be a group of any combination of the items:

(a) the printable characters A to Z
0 to 9
/ + - . $_{10}$

(b) the space character (Octal 00) declared as *

(c) the format symbols P for page change
C for carriage return line feed
S for space
N for case normal
D for dummy (Octal 77)
T for tabulate
EM for end message (→)
Q for semi colon (when used on flexowriter, converts to "read")

The rules for the use of this constant declaration are:-

(a) v must not be greater than w.

(b) If only one word of store is required /w may be omitted.

(c) If Vv/w=P; and Vx=etc., are two declarations in a program and $v \leq x \leq w$ there will, in general, be a compilation failure report for the later one declared.

(d) The initial P, before (string of items), which may also be p, is not printed or counted.

(e) There is no limit to the number of items in the string provided the area specified by Vv/w can accommodate them.

(f) If there are less items in the string than could be accommodated in the area, the items quoted are placed at the most significant end of that area and the remaining area filled out with spaces (Octal 00).

(g) Any printable LETTER may be declared in case normal or case shift but will be printed in case normal. Formal symbols may be declared in case normal or case shift.

(h) The format symbols, either single or in groups, must be enclosed in underline square brackets ⌊ ⌋.

(j) If x items of a particular format symbol S, N, D or T are required consecutively, the symbol may be preceded by x (decimal) instead of specifying the symbol x times.

(k) For eventual printing purposes it may be necessary to arrange to have the format symbol P or C (if used) as the last items in a V store word. This can be achieved by preceding them with the appropriate number of D's.

(l) ⌊ and ⌋ are NOT counted as items for the purpose of filling main store.

(m) Groups of printable characters, space characters and format symbols may be repeated and be in any order.

(n) CRLF, non-printable characters, space and tab used on the flexowriter to punch the declaration are ignored.

(o) When the space character (*) is declared it should NOT be enclosed in the underlined square brackets. Dummy, octal 77, is ignored when sent for printing on the line printer or typewriter.

Examples:

```
V17/19  =  P KDF 9 ⌊S⌋  prog. ⌊s⌋   COURSE    ⌊4 dc⌋;
V20/24  =  P 18TH*JAN  *1965. ⌊c5s⌋
           to ⌊c⌋ 26TH/FEB./1965. ⌊P⌋;
V25     =  p parity ⌊dc⌋;
V0/1    =  P ⌊NC⌋M⌊c⌋ PROGRAM IDENT;
V26/28  =  P*M*TAPE*WANTED*⌊Q⌋;
V30     =  P ⌊pEM⌋;
```

```
V31/37  =  P ⌊N3C35S⌋ HEADING ⌊2CEM⌋;
V38/54  =  P ⌊16S⌋ HERE*IS*THE*TITLE. ⌊CEM⌋;
```

If the contents of V20 to V24, as set up in the second example, were sent to be printed on the high speed line printer, the output would be:-

```
18TH JAN. 1965
     TO
26TH/FEB./1965.
```

7·9 The Instruction 'SET'

There exists a completely different method of introducing integer constants into a program, by use of the instruction SET. SET is a three-syllable instruction which is obeyed by the machine every time it is encountered during the operation of a program. It allows a signed fixed point integer **of not more than 16 bits** to be stored actually amongst the instruction syllables. When SET is obeyed the specified integer constant is transferred to the top cell of the nesting store ready for immediate use. The 16 bits are stored in digit positions 32 - 47 of N1, i.e., in the least significant 16 bits. The bit in digit position 32, the sign digit, is copied into the remaining 32 bits 0 - 31 to give a true single-length signed constant in N1.

The valid forms for the instruction SET are:-

(a) SET n, where n is a signed decimal integer in the range -32768 to +32767.

(b) SET Bt, where t is an octal integer not greater than $(177777)_8$.

(c) SET AYy, to give the true address of Yy.

In form (c) the symbolic address Yy may be replaced by any of the valid forms listed in Paragraph 6 above

It should be noted that at run time the 16 bits generated by the SET instruction are placed right justified in N1 and that digit positions 0 - 31 will all contain 1's if the most significant of the 16 bits is a 1. It is the programmer's responsibility to take account of this.

The programmer may not specify the digit position of the least bit by the use of /p, when using the set instruction.

The use of the instruction SET for small constants is more economical in space than the corresponding procedure using V-store declarations, since each constant declaration requires a main store word in which to store the constant required. Further, each time a declared constant is required, for use a 'fetch' instruction has to be written in the program. In contrast, SET is a single instruction which requires no main store space in which to store the constant, apart from the 3 syllable SET instruction itself.

Examples:

SET 2: causes the integer +2 to be put into N1, fixed point, held to 47 integral places.

SET-32700;

SET10000;

SET B30;

SET AYD9U;

7·10 Exercises - SET 1

1. Compare and Contrast the action of Compiler when compiling the declaration V7=4; and the instruction SET 4;

2. Declare the following constants starting with V0 and using as many succeeding V-stores as necessary.

(a) Declare $2\cdot 3 \times 10^5$ as a single length fixed point number held to 18 integral places.

(b) Declare 17×10^{20} as a double-length fixed point integer.

(c) Declare $32\cdot 2$ as a double-length floating point number.

3. Declare a binary constant so that the word V3 will hold the characters:

(case normal) (CR-LF) ␣ ␣ ␣ ␣ ␣ ␣

4. Declare constants starting at V7 so that they will contain the addresses of:

(a) W3,

(b) V16,

(c) reference 12,

(d) the less significant half of YA0.

(e) If YA0 is E64 what will the V-store contain in case (iv) above.

5. Which of the following Q-store constant declarations are valid? Why are the others not valid? (By "VALID" is meant, will compiler accept it?)

(a) V0 = Q 16/- 1/0;

(b) V1 = Q 0/0/32768;

(c) V2 = Q 1/1/1,024;

(d) V3 = Q AY0U/AYA0L/AY-92;

(e) V4 = Q 70/AY0/AY0;

(f) V5 = AR4/AR4U/AR4L;

6. Which of the following declarations are valid? Why are the others not valid?

(a) V0U = B18/7;

(b) V0L = AY0U;

(c) V1U = 17·5/4;

(d) V1L = AR7;

(e) V2U = B1717171717;

(f) V2L = AV2L;

7. Which of the following declarations are valid? Why are the others not valid?

(a) V0 = P⌊7DC⌋;

(b) V1/2 = P⌊7DC⌋;

(c) V3 = P DATE⌊*⌋ 3Rd;

(d) V15/6 = P ABC⌊12SC⌋;

(e) V17 = P 32768*⌊2S⌋;

(f) V18/19 = P KD F9*Prog *Ex.⌊DC⌋;

(g) If V18 and V19 are sent to the line printer, what would be printed?

8. Which of the following instructions are valid? Why are the others not valid?

(a) SET+1754:

(b) SET - 16 3;

(c) SET A Y64U;

(d) SET AR9;

(e) SET B16/5;

(f) SET 17_{10}+2;

(g) SET Y40;

(h) SET0772;

9. What is a more straight forward way of setting V4 up with a 1 bit in D2 and 0's elsewhere, than that given in the examples of paragraph 7·4?

7·10 **Exercises - SET 2**

1. Declare a constant for V3/4 to hold $\frac{1}{10}$th. as a double-length fixed point number with maximum precision.

2. Declare a **binary constant** so that the word V5 will hold the binary number

+ 1101011 0011011 to 7 integral places.

3. Write a SET instruction to cause 3·5 to be put into N1 as a fixed point number held to 43 integral places.

8· OPERATIONS IN Q-STORES

8·1 General Manipulative Instructions for one Q-store

It has been stated that a Q-store may be regarded either as a single 48-bit word or alternatively as a group of three independent 16-bit signed integers. Separate instructions exist in the KDF 9 Usercode for dealing with Q-stores as a whole, or for dealing with one or more of the individual parts of a Q-store. These are summarised here in tabular form in such a way as to indicate the relations between the different forms available.

Qq;	Cq;	Iq;	Mq;	Fetch from Q-store to N1 of the nesting store.
=Qq;	=Cq;	=Iq;	=Mq;	Store contents of N1 in Q-store
=+Qq;	=+Cq;	=+Iq;	=+Mq;	Add contents of N1 to Q-store (no check on 16 bit capacity).
	=RCq;	=RIq;	=RMq;	Reset Q-store to 0/1/0 and then put contents of N1 into appropriate part.

Note that the four instructions in the third row of the table (=+Qq; etc.) operate in an identical manner to the corresponding sequence

Qq; + ; =Qq;

This implies that the nesting store is pushed down one place and subsequently pulled up two places, thus reducing the effective capacity of the nesting store by one cell during the execution of this instruction.

In general these instructions transfer information between the top cell N1 of the nesting store and the designated Q-store, numbered q. The direction of the transfer is indicated by the following convention. If the first character of the instruction is an '=', the information is sent from the nesting store to Q-store. If the '=' is not present, the information is fetched from the Q-store to the nesting store. It should be remembered that in any transfer from the nesting store to another location, the contents of the first cell N1 of the nesting store will not be preserved.

Any transfer from a location such as a Q-store to the nesting store will cause all the previous information contained in the nesting store to nest down one cell in order to make room for the new item in N1.

In the table it will be noticed that all entries in the first column contain the letter Q. This indicates that these instructions treat the Q-store as one complete

48-bit word. All entries in the second column contain the letter C, indicating that an operation is performed on the counter in digit positions 0-15. The I in the third column indicates that an operation is performed on the increment in digit positions 16-31. In the last column the M indicates that an operation is performed on the modifier in digit positions 32-47.

For the transfers listed in the first column, all 48 bits of the Q-store are involved, so that all 48 bits of N1 will also be used. For the transfers in the remaining columns, only 16 bits of the Q-store are involved. In these cases, whether the counter, the increment, or the modifier of the Q-store is involved, only 16 bits will appear in N1. Since these 16 bits represent an integer, they are stored in the least significant digit positions in N1, i.e., in digit positions 32-47. When a negative 16-bit number is transferred from a Q-store to the nesting store, the sign digit is copied into the remaining 32 bits of N1 to give a true single-length signed integer. When a 16 bit number is sent from the nesting store to a Q-store the most significant 32 bits of the word in N1 are ignored, and no check is made that they are all copies of the sign digit. The whole of N1 is cleared in the usual way when the store operation is completed. Any quantity stored in a Q-store will remain there until it is replaced by different information, and as many copies may be taken as are required. Q-stores behave like main stores in this respect. Thus a 'fetch' from a Q-store puts a copy into the nesting store, and the original information is retained in the Q-store with no change.

The 'reset' instructions in the bottom row of the table each involve changes to all three parts of the designated Q-store. These changes take place in two stages:

(a) the counter, increment, and modifier parts, stored in c/i/m, are reset to 0/1/0;

then

(b) the 16 bit pattern from N1 is stored in the appropriate one of the three parts.

In all of the instructions in the table the integer q is the number of the Q-store involved. q will normally take one of the values 1 to 15. A value of zero for q may be used, provided it is remembered that Q0 is by definition always identically zero.

8·2 Special Instructions involving one part of a Q-store

For each of the three parts of a Q-store there are a few special instructions which may refer only to that part and no others.

Firstly, for the counter there are two such instructions:

NCq;

DCq;

The first of these, NCq, changes the sign of the integer in the counter position of the Q-store. The machine does this by subtracting the original counter from zero and replacing the result in the counter position. The second, DCq, subtracts 1 from the integer in the counter position and replaces the result in the counter position.

Examples:
If Q8 contains -3/+7/+2
then the instruction NC8;
will cause Q8 to become +3/+7/+2

If Q15 contains -4/-6/3
then the instruction DC15;
will cause Q15 to become -5/-6/3

For the increment there are four special instructions each of which resets the integer in the increment position to one of the values +1, -1, +2, -2. These particular values are chosen because they are the values most commonly required in this position. The instructions to do this are:-

```
        Iq  =  +1;                    Iq  =  +2;
        Iq  =  -1;                    Iq  =  -2;
              (Mq and Cq remain unchanged).
```

Examples:
If Q2 contains -3/+7/+2
then the instruction I2 = -1;
will cause Q2 to become -3/-1/2;

If Q14 contains 16/1/AYØ
then the instruction I14 = +2;
will cause Q14 to become 16/2/AYØ

For the modifier there are two special instructions. The integer stored in the increment position may be either added to or subtracted from the integer stored in the modifier position, the result being used to replace the original contents of the modifier.

The two instructions are:-

```
                    M  +  Iq;
                    M  -  Iq;

              (Cq and Iq remain unchanged)
```

N.B. A common mistake is to write these instructions as I+Mq; I-Mq;

Examples
If Q9 contains 2/-18/5
then the instruction M - I9;
will cause Q9 to become 2/-18/+23

If Q10 contains 3/4/AY2
then the instruction M + I10
will cause Q10 to become 3/4/AY6

It will be noticed that none of these special instructions requires information from the nesting store. Therefore, they all leave the nesting store unchanged.

8·3 Operations involving two Q-stores

It is necessary on occasions to transfer information from one Q-store to another. It should be remembered that in any such transfer the Q-store from which the information is copied will remain unchanged. Only the sections of the Q-store into which the transfer is directed will be changed, those sections not involved in the transfer remaining unaltered. In the instructions listed below, information is transferred from a Q-store numbered k (which may take any value from 1 to 15, or 0 if required) into the Q-store numbered q (which may take any value from 1 to 15). k is the number of the Q-store which remains unchanged, and q is the number of the Q-store which changes either wholly or in part. The transfer instructions are:-

Ck	TO	Qq;	transfer counter only.
Ik	TO	Qq;	transfer increment only
Mk	TO	Qq;	transfer modifier only
IMk	TO	Qq;	transfer increment and modifier
CMk	TO	Qq;	transfer counter and modifier
CIk	TO	Qq;	transfer counter and increment
Qk	TO	Qq;	transfer whole of Qk.

None of these instructions disturbs the nesting store in any way.

Note that an instruction such as Ck TO Cq; is invalid, these instructions must end with TO Qq;

Example:
If Q5 holds 3/-1/AY2
and Q8 holds 2/AY5/AY6

The instruction CM8 TO Q5 will leave

Q5 holding 2/-1/AY6
and Q8 holding 2/AY5/AY6

8·4 Effect of Setting q = 0 or k = 0

If Q0 is used in any instruction involving operations on Q-stores, its effect may be understood from the following observations:-

Since Q0 has no physical existence inside the machine except by convention, and since by convention it is always required to yield the value 0, any fetch from Q0 or any of its parts will always produce the value 0. If the fetch is to the nesting store, the 0 will go into N1 and the rest of the store will nest down one cell.

A store into Q0 from part or all of another Q-store will produce no effect whatever. A store into Q0 from the nesting store has the effect of erasing the contents of N1, the rest of the store nesting up one cell in the usual way.

8·5 Example: Setting Q-stores

Suppose it is desired to set two Q-stores, Q1 and Q2. Q1 is to contain the value 6 in the counter, 1 in the increment, and the main store word address of Y47 in the modifier. Q2 is to contain 4 in the counter, 1 in the increment and 0 in the modifier. During the course of this process it will be necessary to use the instruction for fetching a constant from the main store to the nesting store. In Usercode this is done simply by naming the constant. Thus the instruction Vv; will fetch the constant numbered v from the main store into the top cell N1 of the nesting store, leaving the copy in the main store location undisturbed. The essence of the process, therefore, is that the constant declaration Vv = (specification); sets the binary pattern for that constant into the appropriate main store word through the action of Compiler (as described in paragraph 5·2). The instruction Vv; fetches it when required from the main store to the nesting store where it may be used in a calculation or, as in this case, transferred to another location. The Usercode instructions to set Q1 and Q2 are:-

```
V0  =  Q 6/1/AY47;
V0; =Q1; SET+4; =RC2;
```

The first line is the declaration to Compiler that a Q-store constant is required whose three parts are as designated. This will normally appear at the head of the program with the rest of the constant declarations.

The second line contains the instructions which are actually obeyed when the program is run. The first of these fetches the declared constant into the top cell N1 of the nesting store. The second instruction transfers it from N1 to Q1, leaving the nesting store empty. Q1 is now set as required. The instruction SET+4; puts the binary integer whose decimal value is +4 into N1. The last instruction sets Q2 to 0/1/0 and then transfers the number +4 from N1 into the counter position of Q2, thus giving the desired form 4/1/0.

This example has illustrated two possible ways of setting Q-stores, both of which are used very frequently in normal Usercode programs.

Note that in the specimen of Usercode given in the example, every instruction is terminated with a semi-colon ';'. It is an invariable rule in Usercode that a semi-colon must be written after every instruction.

Further Examples:

(i) If Q7 contains 4/2/0, the instruction M+I7; will result in Q7 becoming 4/2/2.

(ii) If Q15 contains -14/1/0, the instructions DC15; NC15; will result in Q15 becoming 15/1/0.

(iii) If Q14 contains 18/9/7, the instruction I14=-2; will result in Q14 becoming 18/-2/7.

(iv) To set Q15 to contain 37/AY5/18, use a V-store constant declaration V2(say) = Q37/AY5/18; and when it is required in the program to set up Q15, use the two instructions: V2; =Q15;

An alternative method is the string of instructions:
SET37; =C15; SETAY5; =I15; SET18; =M15;

(v) To set Q13 to contain 0/1/0, the following instructions could be used SET1; =RI13:

(vi) To set Q12 to contain 8/1/0, use SET8; =RC12;

8·6 Exercises - Set1

1. Write instructions to perform the following operations, without using V-stores. (The instruction +; may not be used, its use having not yet been fully explained.)

(i) The top 3 cells of the nesting store contain three integers. Form their sum in Q1.

(ii) The counter of Q5 contains the integer 7; change the contents of C5 to 5 in 3 different ways.

(iii) Set up Q1 so that it contains:
(a) 64/1/Ø
(b) 3/AYØ/AY64
(c) Ø/-2/Ø
(d) Ø/1/Ø
(e) 74/1/1

2. Q1 contains 1/AYØ/AY31
Q2 contains 512/1/Ø

Without using the nesting store set up Q3 so that it contains the following. The contents of Q1 and Q2 are not to be mutilated.
(a) Ø/AYØ/AY31
(b) 512/-2/Ø
(c) -511/1/Ø
(d) +513/1/Ø
(e) 512/AYØ/1

8·6 Exercises - Set2

1. If YØ is at E320:

(i) Declare a constant for V7 so that after the instructions V7; =Q5; are obeyed, the contents of Q5 will be AY3/-1/AE4U. What would this declaration

have been if it had been declared by a BINARY constant.

(ii) What would the binary constant declaration have had to be if Q5 were to contain AY3/-1/AE4L.

(iii) Declare a Q-store type constant for V8 so that after the instructions V8; =Y2; are obeyed, the contents of E322 will be the fixed point number $100_{(10)}$ held to 32 integral places.

2. If Q8 contains 63/9/3 what will it contain after the following sets of instructions have been obeyed. Q8 is assumed to be reset to 63/9/3 before each set of instructions.

```
(i)      SET10;     =+Q8;
(ii)     SETB10;    =+Q8;
(iii)    SETB101 0;=+Q8;
(iv)     SET2;      =+Q8;
(v)      SETB10;    =+M8;
(vi)     SET10;     =+M8;
(vii)    SET10;     =+I8;
(viii)   SET10;     =+C8;
```

3. What will Q5 contain after the following instructions?

```
SETB177777; =RI5; SET1; =+I5;
```

4. What will Q5 contain after the following instructions?

```
SETB177777; =RM5; SET1; =+Q5;
```

9· NESTING STORE MANIPULATIONS

9·1

The instructions introduced in this Section are concerned solely with the manipulation of information contained in the nesting store.

ERASE; Removes the word in N1 from the nesting store, the remainder of the store nesting up one place. Used whenever a word in the nesting store is redundant, to prevent overfilling.

ZERO; Puts a word of all zeros into N1 (this is zero in either fixed or floating point).

DUP; Takes the word in N1 and makes two copies of it, one in N1 and the other in N2, pushing all other words in the nesting store down one cell. Since a quantity is lost from the nesting store when it is involved in an operation, the DUP instruction is extensively used to make a second copy.

DUPD; Takes the double-length pattern of 96 bits in N1 and N2, and puts copies of into N1, N2 and N3, N4.

The rest of the store nests down two places.

REV; Interchanges the words in N1 and N2, any other words in the nesting store remaining unaffected. REV; is used for example, to set operands in the correct positions for such arithmetic instructions as subtract or divide, in which the order of the operands is significant.

REV D; Interchanges the double-length patterns in N1, N2 and N3, N4, any other words in the nesting store remaining unaffected.

PERM; Performs a cyclic shift of the three words in N1, N2 and N3 by bringing the word from N2 to the top, the word from N3 into N2, and putting the word from N1 down into the third cell. This instruction is frequently used to put a single-length result out of the way further down the nesting store until it is required. If N1, N2, N3 originally contain the word a, b, c respectively, the final order of these words after the instruction PERM; is b, c, a.

CAB; Performs a cyclic shift like PERM; but in the reverse direction. The word from N3 is brought to the top, and the words from N1 and N2 are moved down one cell. This instruction is used to bring an operand from N3 to N1 ready for immediate use. If N1, N2, N3 originally contain the words a, b, c, respectively, the final order of these words after the instruction CAB; is c, a, b, from which circumstance this instruction derives its name.

DUMMY; Has no effect at all. It serves only to occupy one syllable of instruction space that would otherwise be unused.

9·2 Example:

The nesting store will vary by the instructions given, as indicated.

N1	+2	AY5	+2	+32	AY5	AE4U	AE4U	0	0
N2	AY5	+2	+32	AY5	B 30	-37	AE4U	AE4U	AE4U
N3	+32	+32	AY5	B 30	AE4U	AY5	-37	AE4U	0
N4	B 30	B30	B 30	AE4U	-37	B30	AY5	-37	AE4U
N5	AE4U	AE4U	AE4U	-37	F+2·7	F+2·7	B 30	AY5	AE4U
N6	-37	-37	-37	F+2·7	18	18	F.2·7	B 30	-37
N7	F+2·7	F+2·7	F+2·7	18			18	F+2·7	AY5
N8	18	18	18					18	B 30
	START	REV	PERM	ERASE	ERASE	REVD	DUP	ZERO	DUP D

All others Empty

9·3 Exercises - Set 1

1. What single nesting store instruction is equivalent to

CAB; CAB; ?

2. What single nesting store instruction is equivalent to

PERM; PERM; ?

3. If the top three cells of the nesting store contain (N1) 1; (N2)2; (N3) 3; and the rest are empty what 3 nesting store instructions are necessary to obtain (N1) 1; (N2) 2; (N3) 2; (N4) 1; (N5) 3 ?

9·4 Exercises - Set 2

1. Write a string of instructions and declarations to cause:

Q5 to contain the standard floating point number 19.625

Q7 to contain the fixed point number 19.625 held to 44 integral places.

Q8 to contain 15/6/AYØ

Q9 to contain 15/6/AEØ

Q10 to contain 15/6/AY45

Q11 to contain AY3Ø/16/AY15

Only two V store declarations and 3 SET instructions at most may be used for constants. No Q-stores other than those mentioned may be used. It may be assumed that the nesting store is originally empty, and that YØ = E128.

The nesting store is to be left empty.

N.B. This question is only to test the reader's understanding of certain points, in practice six V store delcarations together with six pairs of fetch and store instructions would be used.

10• MAIN STORE OPERATIONS

10•1 General Principles

It has been seen that main store words may be referenced in Usercode in various ways: Yy, YAy, YBy, etc., Ww, Ee, Vv. But it must always be remembered that inside the machine no such distinction is made, and that it is the program's absolute address which is stored with the instruction, ready for swift reference when the instructions are obeyed at run time.

To avoid needless repetition, it is to be understood that wherever the symbolic address Yy is written in this section it may be replaced by any of the forms given above.

The only basic machine instructions concerned in main store operations are these:-

(a) The 'fetch' instruction, which transfers a copy of one word of information from the main store into the top cell of the nesting store, leaving the original in the main store location.

(b) The 'store' instruction, which stores the word from the top cell N1 of the nesting store into a specified word location in the main store, irrespective of the previous contents of that main store location. The word is erased from N1.

These two instructions are distinguished in Usercode by use of the '=' sign. Every store instruction written in Usercode is preceded by an = sign. To omit the = sign converts the instruction to a fetch instruction. It will be remembered that this notation has already been used in the discussion on Q-stores in Section 8.

10•2 Direct Addressing

10•2•1 Unmodified Addresses. The simplest main store operation is the transfer of a single word between the main store and the nesting store. The two corresponding instructions are:-

Yy; fetches a copy of the word whose symbolic address is Yy to the top cell N1 of the nesting store, leaving the original in the main store.

=Yy; stores the contents of N1 in the main store at the location whose symbolic address is Yy. The word is erased from N1, and any previous word in Yy is replaced by the new word.

Example:
To place in YP1 a copy of the word existing in YT45. The instructions are:-
YT45; =YP1;

10•2•2 Modified Addresses. It is often necessary to write a sequence of instructions in the form of a loop which performs the same operations every time it is obeyed, except that the main store addresses to which the instruction refer are required to change each time round the loop. Such changed addresses are referred to as modified addresses.

A modified address instruction is of two parts:

(i) the base address, and

(ii) the modifier, which is an integer added to the base address to determine the actual address required.

The simplest form of address modification employs only the modifier part of a Q-store, digit positions 32–47. The modifier is set to contain a signed integer m, the remainder of the Q-store being ignored. Then the instructions:

(a) YyMq;

(b) =YyMq;

will effect the transfer to or from main store of the word whose address is m words after Yy.

Example:
If the modifier position of Q5 contains the integer 93, then the instruction:

```
Y14M5;
```

is equivalent to Y107;

Consider now the sequence of instructions:

```
                SET 18;    = RC9;
      1;        Y1M9;     =YP1M9;  DC9;  M+I9;  J1C9NZ;
```

The first two instructions cause Q9 to become 18/1/Ø. The first time Y1M9 is encountered a copy of Y1 is brought into N1 and then put from N1 into YP1 because M9 contains the integer 0 at this stage. DC9; M+I9; will now cause Q9 to become 17/1/1, J1C9NZ has not been dealt with yet but the instruction means "jump to reference label 1 if the counter position of Q9 is not zero, otherwise continue in sequence" (Section 10·4). Since C9 is not yet zero, the computer goes back to reference 1 and now Y1M9 causes a copy of the contents of Y(1+1) [=Y2] to be brought into N1. =YP1M9 will cause N1 to be stored into YP(1+1) [=YP2]. The reader will now see that the 'loop' from 1; to J1C9NZ; causes copies of the words in Y1, Y2,....Y18, to be stored in YP1, YP2,....., YP18, respectively.

Had the instructions before the loop been:

SET18; =RC9; SET3; =I9; then the loop would have causes copies of the words Y1, Y4, Y7....., Y52 to be stored in YP1, YP4, YP7, ..., YP52 respectively.

The reader will now see the meaning of the terms counter, increment, and modifier, in connection with Q-stores.

10·3 Indirect Addressing

10·3·1 General Principles. The direct address instructions dealt with above are entirely satisfactory for all cases in which a routine is required to deal with a set of data starting at the one base address. However, this is not always the case. The same loop may be needed to fetch words (say)

YA1, YA2, ..., YA52 the first time the program is run

YA102, YA103, ..., YA153 the second time

YP28, YP29, ..., YP79 the third time, etc.

Since the base address is a variable it is not possible to specify it directly in the instructions.

What is needed for this purpose is a simple and convenient way of specifying a base address for each set of data, and also a modifier for accessing the contents of each set starting from its base address. This is done in KDF 9 Usercode by the use of indirect addressing instructions.

10·3·2 In this form of fetch or store instruction the base address is not specified directly in the instruction, but is given indirectly by referring the machine to a Q-store in which it will find the base address. Once again the modifier part of the Q-store is selected to hold this information. It is only necessary to specify in the instruction which Q-store to interrogate for the base address, and which Q-store is to be used as the modifying register. The specification of a Q-store for the base address takes only 4 bits of instruction space, whereas to specify the base address in a direct fetch or store instruction requires 15 bits. The indirect fetch or store instruction as a whole is a two syllable instruction as opposed to the three syllables required for a direct address instruction. These figures show that there are very real advantages in the use of the indirect fetch or store instructions even apart from the matter of their convenience.

It has been stated that in all indirect address instructions two Q-stores must be specified: the first contains the base address; the second contains the modifier which will be updated if requested. It should be emphasized that the first Q-store must contain the actual word address of the base, which will be written in Usercode programs in its symbolic form but exists inside the machine as an absolute address in binary form. Thus if it were required to set the base address of Y5 in the Q-store Q1, this could be done with the instructions:

```
SET AY5; =M1;
```

These instructions set the binary value of the address of Y5 into the modifier of Q1, resulting in a valid base address stored in Q1 ready to be used in an indirect fetch or store instruction.

The basic forms of the indirect fetch and store instruction are:

```
        MkMq;        Indirect fetch

       =MkMq;        Indirect fetch
```

In these instructions Mk contains the base address in the modifier position of Qk, and Mq contains the modifying integer in the modifier position of Qq.

Example:
If Q1 contains 18/5/AYX42
and Q13 contains 12/4/8
then the instruction M1M13; would be equivalent to YX50;

It should be noticed, in this example, that M1 actually contains an integer equivalent to the E address of YX42, so that if YX42 = E385, M1 will contain $385_{(10)}$ in binary form.

The instruction M1M13; then is equivalent to E(385+8) i.e., E393(=YX50). In consequence of this M1M13; is just the same as M13M1;

Caution: Although the contents of a particular address may be copied into N1 by quoting just the base address with DIRECT addressing (e.g., Yy;) such is not possible with INDIRECT addressing. The instruction M5; (say) means fetch into N1 a copy of the contents of M5. However, M0M5; means fetch into N1 a copy of the word whose address is E(Ø+integer in M5).

10·4 Jumps on Counters

In paragraph 10·2·2 we saw the need to introduce an instruction which compares the contents of the counter position of a Q-store with zero and then either causes a jump to a reference label or continues with the instruction in sequence.

There are two conditional jumps of this form:

```
JrCqZ;    jump to reference label r if the counter of Qq is zero.

JrCqNZ;   jump to reference label r if the counter of Qq is non-zero.
```

To illustrate one way in which these instructions may be used, suppose there are n numbers stored in consecutive words from Y1 to Yn inclusive and that it is required to move copies of them to locations YA1 to YAn inclusive, preserving the same order. The integer n is supposed given in the top cell N1 of the nesting store. The appropriate instructions are:

```
=RC1;     J1C1Z;

2;          Y1M1;  =YA1M1;  DC1;  M+I1;  J2C1NZ; •••••

1;
```

These instructions perform the following operations:

(a) =RC1; transfers n from N1 to the counter position of Q1, simultaneously setting the increment to 1 and the modifier to 0.

(b) J1C1Z; is the instruction to jump to reference 1 if the counter of Q1 is zero; i.e., if n is zero. This by-passes the loop if it is required to use it 0 times.

(c) The loop itself starts with the reference label 2, fetches Y(1 modified by contents of M1), stores it in YA(1 modified by contents of M1), and updates the Q-store ready for the next entry to the loop. The instruction J2C1NZ; causes a jump back to label 2 to repeat the loop if the counter of Q1 has not yet been reduced to zero. On the second entry to the loop, the modifier of Q1 contains the value 1, so that Y2 is transferred to YA2, and so on, until the n words have been transferred. C1 was originally set at n and 1 is subtracted from it at each pass through the loop. Hence the machine will only continue to reference 1 when C1 has been reduced to zero, i.e., when the machine has passed through the loop n times.

(d) Reference label 1; prefaces the instruction sequence for the case when no items are required to be transferred; i.e., for the case n = 0.

It is essential the reader notices that if the instruction J1C1Z; had not been included and 'n' had originally been zero then the instruction J2C1NZ; would only have been obeyed when C1 was negative - the loop could have been repeated infinitely. "Infinite loops", should always be guarded against by the programmer.

10•5 Modified Address with Incremented Q-stores

In the example on the previous page it was necessary to decrease the counter of Q1 by 1 and to increment the modifier by the contents of I1. This was done by the instructions DC1; M+I1; In USERCODE the facility to perform these two operations is given by writing a Q after the modifier part of either a direct or indirect modified address instruction. So that the example would be written as:

```
            =RC1;  J1C1Z;
 2;         Y1M1;  =YA1M1Q;  J2C1NZ;
 1;
```

The programmer should remember the sequence of events when using this Q-facility.

(i) Access the main store word whose address is the sum of the base address and of the integer m in the modifier part of Qq, then

(ii) Update the Q-store relating to the modifier part of the address, by adding the integer i in the increment position to the modifier, so that the modifier item has the value m+i, and

(iii) Subtract 1 from the signed integer c in the counter position.

Note: In the Indirectly specified base address form of main store addressing (MkMq;) if the Q facility is used (MkMqQ) it only updates the q Q-store; the k Q-store remains unchanged.

It will be noticed that the increment, which is unchanged by this process, specifies the interval in the main store between successive fetches or stores. For example, if it were required to fetch every third word the increment would be assigned value 3. Also, the counter may be used to specify the number of times the instruction is to be obeyed. For instance, if the instruction occurred in a loop to be performed 30 times, the counter would be set initially to hold the integer 30. Then, since 1 is subtracted from the counter each time the instruction is performed, the cycle will be completed when the counter reaches the value Ø. As an illustration, the set of instructions:

```
              V0     =    Q 4/7/3;

              V0;    =    Q1;  Y6M1Q;
```

will transfer the contents of V0 into Q1 (thus setting up the initial form of the Q-store), and will then fetch the contents of Y(6+3) into the nesting store, after which the Q-store will be updated by adding the increment to the modifier and subtracting 1 from the counter. The final state of the Q-store will, therefore, be Q1 = 3/7/10. If the instruction Y6M1Q; is now obeyed again, the word transferred to the nesting store will be the word Y(6+10) or Y16, and the Q-store will be further updated to Q1 = 2/7/17 ready for the next time it is required.

In any Q-store used for this purpose, the increment may be set to any desired value either positive or negative. It will be remembered from paragraph 8•2 that special facilities exist for setting an increment to +1, -1 +2, or -2, which are the most commonly required values.

10•6 Examples

(i) To place copies of the words in Y43 to Y98 into YP1 to YP56 respectively:-

```
                     SET56;  =RC1;

              1;     Y43M1;  =YP1M1Q;  J1C1NZ;
```

(ii) To interchange the words in Y43 to Y98 with those in YP1 to YP56 respectively:-

```
                     SET56;  =RC1;

              1;     Y43M1;  YP1M1;  =Y43M1;  =YP1M1Q;  J1C1NZ;
```

(iii) To interchange the words in Y1 and Y2; Y3 and Y4, etc., up to Y99 and Y100:-

```
                    SET50;  =RC1;  I1=+2;
          1;        Y1M1;  Y2M1;  =Y1M1;  =Y2M1Q;  J1C1NZ;
```

(iv) To place copies of the words in Y0, Y1, Y10, Y11, ---, Y90, Y91 into YS1, YS2, YS3 ---, YS20:-

```
                    SET10;  =RI1;  SET20;  =RC2;
          1;        Y0M1;  =YS1M2Q;  Y1M1Q;  =YS1M2Q;  J1C2NZ;
```

(v) If M12 contains the address of word (A) and M11 the address of a word (B); to interchange these two words:-

```
          EØM12;  EØM11;  =EØM12;  =EØM11;
```

N.B. M12; M11; =M12; =M11; will **not** interchange the words in the addresses given, it will only interchange the contents of M11 and M12.

(vi) To place copies of the words in Y0, Y1, Y10, Y11, ..., Y90, Y91 into consecutive addresses beginning at the address given in M4:-

```
                    SET10;  =RI1;  SET20;  =RC2;
          1;        Y0M1;  =M4M2Q;  Y1M1Q;  =M4M2Q;  J1C2NZ;
```

10·7 Exercises - Set1

1. Store the contents of Q1 in YØ.

2. Fetch the contents of the word whose address is in M1, and store them in the word whose address is in M2.

3. Transfer the contents of Y0, Y1, ..., Y31 in YA0, YA1, ..., YA31 respectively.
(a) Without using the 'Q' facility.

(b) Using the 'Q' facility.

4. A Q-type parameter in N1 consists of two addresses, in D16-D31 and D32-D47, and an integer n≥0 in D0-D15. Fetch n consecutive words beginning at the address in the modifier position of the parameter and store them in consecutive words from the address in the increment position onwards.

5. Fetch the contents of Y0 and Y1, Y3 and Y4, Y6 and Y7 ...., Y90 and Y91, and store them in consecutive words from the address given in M15 onwards.

10·7 Exercises - Set 2

1. Put the sum of the contents of Q1 and Q2 in E256.
The instruction +; must not be used.

2. If YØ is in E672, and Q5 contains 27/AY8/AY35, what will be the effect of obeying the following sets of instructions. Q5 is assumed to be reset to 27/AY8/AY35 before each set of instructions.

```
(i)      M5;  =Y35;

(ii)     MØM5;  =Y35;

(iii)    I5;  =M5;  Y27M5;  =Y35;

(iv)     I5;  =M5;  C5;  =M4;  M4M5;  =Y35;

(v)      I5;  =M5;  C5;  =M4;  M5M4;  =Y35;

(vi)     I5;  =M5;  C5;  =M4;  M5;  =M4;
```

3. If N1 contains the integer 7 fixed point (47 integral places).

N2 contains AY7.
N3 contains AE7.
N4 contains the integer 320 fixed point (47 integral places), and if YØ=E32Ø.

What will be the effect of the instructions:-

```
(i)      =M6;  REV;  =M5;  MØM5;  =MØM6;

(ii)     REVD;  =M6;  =+M6;  EØM6;  REV;

(iii)    CAB;  =M6;  =+M6;  MØM6;  =Y7;

(iv)     PERM;  =M5;  =M6;  M5M6;  =Y7;
```

4. What is the final overall effect of the following instructions if YØ=E64Ø and if N1 originally contained the integer 93, and N2 the integer -93, and Q1 = 0/1/0

```
SETAY93;  =RC2;  =+I2;  SETAE641;

Q2TOQ5;  =C5;  =+C1;  IØTOQ1;  C5;

=M5;  M5M1Q;  NC1;  DC1;  I1TOQ5;  M5;

NC5;  =+C5;  =Y2;  C5;  =M5;  Y43M5;  =Y3;
```

KDF 9 CHARACTER CODES

	2nd OCTAL DIGIT	0	1	2	3	4	5	6	7
1st OCTAL DIGIT									
0	C.N.	SPACE		CRLF		TAB		C.S.	C.N.
	C.S.	SPACE		CRLF		TAB		C.S.	C.N.
	P.	SPACE		LS	PC	ISS		%	'
	C.R.	BLANK	Y, 6, 8	Y, 2, 8	0, 2, 8	0, 5, 8	Y, 5, 8	0, 4, 8	Y, 7, 8
1	C.N								/
	C.S.								:
	P.	:	=	(	)	£	*	'	/
	C.R.	4, 8	3. 8	X, 5, 8	X, 2, 8	X, 3. 8	X, 4, 8	0, 3, 8	0, 1
2	C.N.	0	1	2	3	4	5	6	7
	C.S.	↑	[	]	<	>	=	×	÷
	P	0	1	2	3	4	5	6	7
	C.R.	0	1	2	3	4	5	6	7
3	C.N.	8	9	-	10	;	+	-	.
	C.S.	(	)	-	£	;	=	*	'
	P.	8	9		10	;	+	-	.
	C.R.	8	9	6, 8	X, 6, 8	Y, 4, 8	Y	X	Y, 3, 8
4	C.N.		A	B	C	D	E	F	G
	C.S.		a	b	c	d	e	f	g
	P.		A	B	C	D	E	F	G
	C.R.	0, 7, 8	Y, 1	Y, 2	Y, 3	Y, 4	Y, 5	Y, 6	Y, 7
5	C.N.	H	I	J	K	L	M	N	O
	C.S.	h	i	j	k	l	m	n	o
	P.	H	I	J	K	L	M	N	O
	C.R.	Y, 8	Y, 9	X, 1	X, 2	X, 3	X, 4	X, 5	X, 6
6	C.N.	P	Q	R	S	T	U	V	W
	C.S.	p	q	r	s	t	u	v	w
	P.	P	Q	R	S	T	U	V	W
	C.R.	Y, 7	X, 8	X, 9	0, 2	0, 3	0, 4	0, 5	0, 6
7	C.N.	X	Y	Z			→		
	C.S.	x	y	z			→		
	P.	X	Y	Z	E.F.	E.D.	E.M.		
	C.R.	0, 7	0, 8	0, 9	5, 8	6, 8	7, 8	X, 7, 8	2, 8

C.N.	Paper Tape Code. Case Normal.
C.S.	Paper Tape Code. Case Shift.
P.	Printer Code.
C.R.	Card Reader Code.

11•1 Addition and Subtraction

11•1•1 General Principles. An important property of the nesting store must be mentioned before these instructions are given in detail. In any arithmetic operation such as a+b, a-b, etc., 'a' will be referred to as the first operand, 'b' as the second operand and the +, -, etc., as the operation or the function. The logical way of proceeding with such an instruction is to fetch the first operand 'a', then to fetch the second operand 'b', and then to perform the operation of +, -, etc. This is in fact the way in which the arithmetic functions have been organised to operate inside the machine. It is particularly important to remember this when performing an operation such as -, in which the order of the operands is significant. With b in N1 and a in N2, the instructions +; or -; will produce respectively a+b or a-b in N1, a and b themselves having been erased from the nesting store in the usual way. This rule may be remembered from the phrase:

"N2 function N1"

which describes the order in which the machine deals with the operands in N1 and N2 in an arithmetic instruction. It means that the operands may be fetched in the order given in the problem and the arithmetic operation performed without the need for rearrangements in the nesting store.

It is essential, when adding and subtracting in fixed point, that both operands are held to the same number of integral places; the result will also be given to that number of integral places.

11•1•2 Addition and Subtraction Instructions. The add and subtract instructions for fixed-point numbers are:

+; Adds the number in N1 to that in N2, leaving the result in N1. Overflow set if both numbers have the same sign and the result exceeds single-length capacity (see Section 13•1•3 for testing state of overflow register).

+D; Adds the double-length number in N1 and N2 to the double-length number in N3 and N4, leaving the double-length result in N1 and N2. Overflow set if both operands have the same sign and the result exceeds double-length capacity.

-; Subtracts the number in N1 from that in N2, leaving the result in N1. Overflow set if the operands have opposite signs and the result exceeds single-length capacity.

-D; Subtracts the double-length number in N1 and N2 from the double-length number in N3 and N4, leaving the double-length result in N1 and N2. Overflow set if the operands are of opposite sign and the result exceeds double-length capacity.

NEG: Changes the sign of the number in N1 ('Negate') by subtracting the original contents of N1 from zero and leaving the result in N1. Overflow set only if the original number in N1 is negative and of maximum size.

NEGD; Changes of the sign of the double-length number in N1 and N2 by subtracting it from zero and leaving the result in N1 and N2. Overflow set if the original number is negative and of maximum size.

N.B. An incorrect result may be obtained if the D0 digit of the less significant half of a double-length number used in any arithmetic operation is not zero.

11·1·3 Double-length to Single-length and vice versa. It is sometimes necessary to add together a set of single-length fixed point numbers to form a double-length sum. To do this each single-length number must first be extended to double-length form, since it is not possible to add a single-length number to a double-length number. When this extension to double-length form is made the sign of the single-length number must be preserved. The instruction for this purpose is:

STR; ('Stretch'). Takes a single-length fixed-point number in N1, moves it down into N2, and fills N1 with 48 copies of the most significant (sign) bit of the original number. This produces a double-length number arithmetically equivalent to the original single-length number, except that the number of integral places is increased by 47. The D0 bit of N2 will be set to zero.

The inverse process to convert a double-length fixed point number to single-length, works in exactly the reverse way. The instruction is:

CONT; ('Contract'). Takes a double-length fixed point number in N1 and N2 and replaces it by a single-length number obtained by removing the more significant half. Dø of N1 is preserved. The result has 47 less integral places than the original double-length number. Overflow is set if the more significant half was not all zeros or all ones - this indicates that the number is too large to be held in a single-length register. Overflow is also set if Dø of N2 was 'one'.

11·1·4 Comparison of Single-length Numbers

When it is required to compare two numbers, this could be done using ordinary subtract operations. However, in certain awkward cases overflow could be set by this process. To avoid this possibility a special instruction has been provided in Usercode which compares the numbers in N1 and N2 and supplies an indication in N1 as to their relative magnitudes. The instruction is:

SIGN; Takes two single-length fixed point numbers in N1 and N2 and sets N1 equal to:-

(a) +1 if the word in N2 is greater than the word in N1.

(b) 0 if the two words are equal.

(c) -1 if the word in N1 is greater than the word in N2.

The numbers to be compared are treated as signed numbers in this test, so that any negative number is smaller than any given positive number. Note that the sign of the indicator left in N1 is the same as the sign of the result which would have been obtained had a -; instruction been performed with the original two numbers.

The original contents of N1 and N2 are, of course, erased.

The overflow register cannot be set by this instruction.

11·1·5 Miscellaneous Instructions

ROUND; Rounds off a double-length fixed point number in N1, N2 to single-length, giving result in N1. The rounding is achieved by adding the D1 digit from N2 into the D47 position of N1.

ROUND H; Intended for rounding-off a single-length fixed point number in N1 to half-length before storing using half-length store. The effect of this instruction is to add a 'one' in the D23 position if the D24 digit is a 'one'. The result appears in the D0-23 digits of N1; the state of the remaining 24 digits is undefined.

ABS; Produces in N1 the absolute (modulus) value irrespective of sign of the fixed point number previously in N1. The instruction subtracts the data word from zero if it is negative, otherwise leaves it unchanged.

MAX; Takes two **signed** fixed point numbers in N1, N2 and rearranges them so that the larger is in N1, the smaller in N2. The instruction effectively subtracts N1 from N2 and inspects the sign of the result. If it is negative, they are already the right way round. If it is positive, N1 and N2 are reversed and OVERFLOW is SET to indicate the reversal. This is the only time overflow is used other than to indicate that capacity is exceeded: it must be allowed for whenever MAX is used. Since in fixed point the number of integral places for each number is in the programmers notebook and not in the machine, each pattern is dealt with by the machine as though it were held to zero integral places. This infers that the two numbers must be held to the same number of integral places for this instruction.

11·2 Examples

1. If Y5 contains $19{\cdot}7_{(10)}$ fixed point held to 12 integral places (say) and Y6 contains $24{\cdot}5_{(10)}$ fixed point held to 12 integral places (say) then

(i) Y5; Y6; +; =Y7; will cause $44{\cdot}2_{(10)}$ held to 12 integral places to be stored in Y7.

(ii) Y5; Y6; -; =Y7; will cause $-4{\cdot}8_{(10)}$ held to 12 integral places to be stored in Y7.

(iii) Y6; Y5; -; =Y7; will cause $+4{\cdot}8_{(10)}$ held to 12 integral places to be stored in Y7.

2. If Y1 contains the more significant half of the double length fixed point number $19{\cdot}7_{(10)}$ held to 70 integral places (say) and Y2 the less significant half, and if Y3, Y4 similarly contain $24{\cdot}5_{(10)}$ held to 70 integral places (say) then the instructions Y2; Y1; Y4; Y3; +D; =Y5; =Y6;will cause $44{\cdot}2_{(10)}$ held to 70 integral places to be stored in Y5, Y6. Note the order of fetching and storing the individual halves of a double length number.

3. If N1 contains - 18·5 fixed point held to 12 integral places (say) then the instruction STR; will cause
N1 to hold the more significant half, and N2 the less significant half of the double length fixed point number -18·5 held to 59 integral places. (Since the number is negative N1 will be all ones, but as always with double length numbers, D0 of N2 will be zero.)

If the instruction CONT; is now obeyed N1 would contain -18·5 to 12 integral places.

4. If N1 contains -15·6 fixed point held to 9 integral places (say) and N2 contains +7·3 fixed point held to 9 integral places then the instruction:

(i) SIGN would erase N2 and set N1 equal to +1 held to 47 integral places.

(ii) MAX would cause N2 and N1 to be reversed and this reversed would be indicated by the overflow register being set.

11·3 Logical Operations

In KDF 9 the term 'logical operations' refers to procedures which treat a binary quantity as a pattern of individual bits, changing each bit if necessary from 0 to 1, or 1 to 0 according to some criterion, but never causing a carry from one digit to the next. Operations on one bit can in no way effect any of the other.bits. Some logical operations act on single binary patterns, and some compare two patterns to produce a third according to an appropriate set of rules.

The two logical instructions which act on a single binary pattern are:-

11·3·1 Logical Operations - Single Word of Data

NOT; Takes a 48-bit pattern in the top cell N1 of the nesting store and replaces it with a pattern generated by changing each 1 to a 0 and each 0 to 1. The form of the written instruction indicates that each digit in the result is "not" what it was before.

BITS; Takes a pattern of 48 bits in N1, counts the number of non-zero bits in this pattern and leaves the count as an integer in N1, held to 47 integral places. The original pattern is erased.

11·3·2 Logical Operations - Two Words of Data

The logical instructions which compare two patterns require these patterns to be in N1 and N2. A bit from N1 is compared with the corresponding bit from N2 under a given set of rules, to generate one bit in the result pattern. When all the bits have been compared in this way the original contents of N1 and N2 are erased and the resulting pattern left in N1. The possible combinations of the binary digits to be compared are:

(a) Both digits zero.

(b) Digits 0 and 1.

(c) Digits 1 and 0.

(d) Both digits 1.

(b) and (c) are effectively the same since no preference is given to either of the two patterns.

The three instructions of this kind are:-

OR; Gives a 1 in the result if one **or** other **or** both of the compared bits is a 1. Thus combinations (b), (c) and (d) produce a 1, while combination (a) produces a 0.

AND; Gives a 1 only if both one **and** the other of the compared bits are 1's. Combinations (a), (b) and (c) produce a 0.

NEV; ('Not Equivalent'). Gives a 1 when the two bits under comparison are different. Combinations (a) and (d) produce a 0.

The following examples using two 4-bit patterns will illustrate the effects of these instructions:-

```
N1  0011        0011        0011
N2  0101        0101        0101
    ----        ----        ----
OR  0111  AND   0001  NEV   0110
```

11·4 Radix Conversions

11·4·1 **General.** Ideally the six information bits in a numeric character punched on paper tape would form the binary equivalent of the decimal digit represented by the given character. However, in the system adopted for paper tape will show that the representation of decimal zero is octal 20 (decimal 16); decimal 1 is represented by octal 21 (decimal 17), and so on. It will be realised that each of these binary code representations differs from its true binary equivalent by the presence of an extra bit in the fifth position from the least significant end. This extra bit carries the octal value 20 or decimal 16, and for this reason it is referred to as the "excess-16" bit.

Therefore, when numeric information is read from paper tape into the main store as described, it is not in the binary form required by the machine. The six information bits for each character on tape are transferred directly into the main store without change. However, many characters on tape are required to specify any one decimal quantity, one character for each decimal digit, that same number of six-bit groups is read into the designated main store word. There may be a maximum of eight such characters to a number, or eight six-bit groups in one word. Each of these six-bit groups will still be in the excess-16 form, and the whole collection of six-bit groups will have to be converted to the true binary form before any calculations can be performed. A special instruction is provided for this purpose in KDF 9 Usercode, together with a corresponding instruction to convert binary information to character form in preparation for output.

The most common use of these instructions will be in the conversion from the decimal scale to the binary scale or vice versa, but there is no reason why some other scale should not be used instead of decimal, provided the end-product inside the machine is in binary. For instance, suppose that the input data are in hours, minutes and seconds.

To enable the machine to operate on such data the simplest procedure would be to convert each datum to seconds, expressed in the binary scale. For example, suppose that one input datum is 1 hour 23 minutes 45 seconds. The successive digits in this quantity represent 1 hour, 2 tens of minutes, 3 minutes, 4 tens of seconds and 5 seconds, and each digit will be in the form of a six-bit binary code representing its true binary value. The conversion to binary would have to proceed in two stages:

(a) To change from the coded excess-16 form of each character to the true binary form, and

(b) To convert from hours, etc., into seconds using the binary values from (a) and various conversion constants, the result being the number of seconds in binary.

11•4•2 Removal of excess-16 bit. Suppose that N1 contains eight numeric characters as read from paper tape, and that the following sequence of instructions is performed:

```
V1    =    B1717171717171717;

V1;   AND:
```

Restricting attention for the moment to one character, suppose that its binary form is 010101, which is the character form of the decimal digit 5. The effect of the instruction AND; on this character and V1 is as follows:

```
V1        001111

 5        010101
          ------
AND       000101
```

Evidently this result is the true binary representation of decimal 5, the excess-16 bit having been removed. Since this is also true for any of the other characters, after this sequence of instructions N1 will contain the eight characters in the form required for the second stage of the conversion to binary.

Before the removal of the excess links, the contents of N1 are said to be in "character form", after removal of the excess 16 bits they are in "binary coded form". When the second stage, now to be explained, has been performed, they are in "true binary form".

11•4•3 Principles of Radix Conversion

The conversion constants mentioned under (b) will now be further discussed. In the example quoted, in paragraph 11•4•1, one digit was required for the hour (although more than one could have been used), two digits for the seconds. The least significant of these digits; e.g., the 5 in seconds, may in general take any of the values 0-9, a carry of one into the next highest digit position occurring if a value of 10 or more is required, the remainder being left in the seconds position, as is normal in any operation in the decimal scale. The next digit; e.g., the 4 in tens of seconds, may take any of the values 0-5, a one being carried into the next highest digit position if a value of 6 or more is required, the remainder being left in the tens of seconds position. The value at which a digit in a given position requires a one to be carried into the next highest position is called the radix for that digit. Thus in the present example the radix for the seconds digit is ten and the radix for the tens of seconds digit is six. Similarly, the radix is ten for the minutes digit, six for the tens of minutes digit, while the radix for the hours digit is not specified if it is the largest unit used. These radices are the conversion constants necessary for the operations in stage (b) paragraph 11•4•1.

The radix conversion instructions in Usercode permit any set of radices to be used subject to the following restrictions: every radix must be an integer, and every such integer must be non-zero, and smaller than 32.

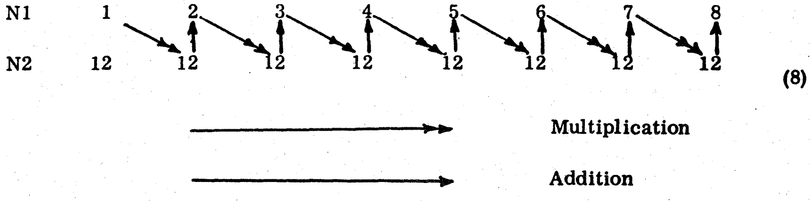
To illustrate how an infringment of this rule can arise, suppose data in shillings and pence are to be reduced to binary form. To represent the shillings and pence will in general require four digits, the first for the tens of shillings, the second for the shillings, the third for the tens of pence, and the fourth for the pence. The radix for the pence digit is ten but since there are 12 pence in a shilling the radix for the tens of pence digit is one point two. The radix for the shillings digit is ten and that for the tens of shillings digit is two if pounds are to be used. This system of radices is not permissible because of the occurrence of the non-integral radix one point two. Some other radix system has to be used for this particular problem.

11·4·4 **Radix Word.** To enter the radix conversion routine the top word N1 of the nesting store must contain the eight six-bit groups forming the number whose conversion is required, (i.e., the binary coded form of the number) and N2 must contain the corresponding eight six-bit radices. Notice that the radices in N2 must be in binary. If the conversion is from decimal to binary, all eight radices in N2 will be the binary equivalent of decimal 10. Decimal 10 will be written in the program as octal 12, which is the usual shorthand way of writing a binary number, so care must be taken not to confuse numbers written in octal and decimal when preparing the program.

The pattern of the eight radices held in N2 is called the Radix word. The radix word for a pure decimal number is $1212121212121212_{(8)}$, each group of six bits is equivalent to "ten".

11·4·5 **Operation of binary coded form to true Binary Conversion.** The executive instruction TOB; ("to binary") is sufficient to convert the eight binary coded characters in N1 to a binary single length fixed point integer (held to 47 integral places) in units of the least significant character. A simplified picture of the way the machine performs this operation will now be given.

The first (most significant) character is multiplied by the radix of the second character and the result added to the second character itself. This sum is then multiplied by the radix of the third character and the result added to the third character itself, and so on, the results accumulating with every operation, until after the seventh addition only a binary integer remains which gives the result in units of the least significant character. The diagram shows in schematic form how this is done for the conversion to binary from decimal.



The nesting store now contains the result as a binary integer in N1, the words previously in N1 and N2 having been erased. During the execution of this instruction no checks are made that a character does not exceed its radix or that the radix does not exceed 31. The overflow register cannot be set by this instruction.

To summarise the steps in the process for converting characters to binary integers are:

(i) Fetch radix word into nesting store.

(ii) Fetch character word into nesting store.

(iii) Fetch $1717171717171717_{(8)}$ into nesting store.

(iv) Perform the instruction AND; which with N1 as in step (iii) removes the excess 16 bits.

(v) Perform the instruction TOB; which converts characters to binary using scale as given in the radix word. The result is a single length fixed point integer (held to 47 integral places).

11·4·6 Example

If Y1 contains the character form of 19Ø23Ø45 which represents 19hrs. 23mins. 45secs. and it is required to convert this to the true binary in units of seconds, the instructions required are

```
VØ = B 12 12 01 06 12 01 06 12;
V1 = B 17 17 17 17 17 17 17 17;
VØ; Y1; V1; AND; TOB; =Y1;
```

VØ contains the radix word: since the radix of the tens of hours (1) is ten we write $12_{(8)}$ for that digit; similarly for the units of hours (9). The two Ø's in the character form are not digits as such but merely indicators of the separation between hours and minutes, and between minutes and seconds. These two positions require a radix of 1, because when the cumulative multiplication with this radix is performed we require no change - reference to the diagram in 11·4·5 will make this clear. So, for these two "spaces" we write 01. The radices for the tens of minutes (2) and tens of seconds (4) are both 6, so we write these as 06.

To convert binary results to character form ready for output on to paper tape, the two stages (a) and (b) of 11·4·1 have to be obeyed in the reverse order. For stage (b) N1 must contain the binary number and N2 the radix word which is obtained in exactly the same manner as before. Then the instruction FRB; ("from binary") does just the reverse of TOB; Its action is to divide the integer in N1 by the least significant radix and record the remainder as the least significant binary coded character of the result. The quotient from this division is then divided by the next radix, the remainder is recorded as the next binary coded character in the result and so on until all eight character spaces in the result word are filled.

11·4·8 **Insertion of excess-16 bit.** For stage (a), to insert the excess 16-bit after the instruction FRB; before output to a paper tape punch presuming the characters to be in N1, the following set of instructions is used:

```
V2  =  B2020202020202020;
V2;  OR;
```

If one of the characters was originally the true binary representation of 5, the effect of the instruction OR; on this character and on V2 is:

```
V2      010000
 5      000101
        ------
OR      010101
```

The excess-16 bit has now been inserted into this and into all the other characters originally in N1, N1 now containing these eight characters in the form required for output to paper tape.

11·4·9 **Example.** If Y1 contains the single length fixed point integer $69825_{(10)}$ held to 47 integral places, which represents 69825 seconds, and it is required to convert this to the character form of hours minutes and seconds in the format h h Ø m m Ø s s, the instructions required are

```
VØ  =  B 1212 010612010612;
V1  =  B2020202020202020;

VØ;  Y1;  FRB;  V1;  OR;  =Y1;
```

11·5 Exercises - Set1

1. Form the double-length sum in Y6, Y7 of 32a+16b+8c+4d+2e+f = 2(2(2(2(2(a)+b)+c)+d)+e)+f. Where a, b, c, d, e, f are single-length fixed point integers stored in Y0, Y1, Y2, Y3, Y4, Y5 respectively.

2. The single length fixed point integers $x_0, x_1, x_2, \ldots\ldots, x_{31}$ are stored in YA0-YA31 and the single length integers $y_0, y_1, y_2, \ldots\ldots, y_{31}$ are stored in YB0-YB1.

Find $(y_0 - x_0)+(y_1 - x_0)+\ldots.+(y_{31} - x_{31})$ and leave your answer as a single length integer in N1. Hint x_0 may be negative and so $(y_0 - x_0)$ may exceed single length capacity.

3. N1 and N2 contain two single length fixed point integers. If they are equal set M5 to contain zero; if (N1) > (N2) set M5 to contain +1; otherwise set M5 to contain -1. The original N1 and N2 are not to be lost.

4. N1 and N2 each contain a binary pattern. Write the instructions (which are to include 'AND' and 'NEV') to form the "OR" of the two patterns.

5. N1 contains a binary pattern. If there are an even number of non-zero bits in this pattern make the counter of Q1 contain 1; otherwise make it contain 0. (Do not use any jump instructions).

6. Write down the contents of N1 in octal if it contains the decimal integer ØØØ32768

(a) in character form

(b) in binary coded form

(c) in true binary.

7. N1 contains the eight decimal digits 12Ø13Ø14 in **binary coded form**, representing £12,013/14/0d. Convert this pattern to binary shillings, using an appropriate radix.

8. N1 contains the eight decimal digits 24030015 in **character form**, representing 24hrs. 30mins. 15secs. Convert this to binary seconds, using an appropriate radix.

9. N1 contains a binary integer. Convert it to **binary coded form**.
What is the largest permissible answer expressed as the octal representation of contents of N1?

10. N1 contains a binary integer representing a number of pints.
Convert it to gallons, quarts, and pints **in character form** in the format
GGGGØQØP

Where GGGG is the number of gallons
Q is the number of quarts
P is the number of pints
0 is the digit 0 used for spacing.

Exercises - Set2

1. Repeat question 10 of set 1 using the format GGGG␣Q␣P where ␣ represents a space.

2. If N1 contains a single length fixed point integer, what will it contain after the instructions:

(i) NOT; NEG;
(ii) NEG; NOT:

3. What is the difference between VØ = P 12345678;
VØ = BØ1Ø2Ø3Ø4Ø5Ø6Ø71Ø;

12· OUTPUT VIA OUT 8

12·1 General Principles

Attention is now turned to the process whereby the contents of an area of main store may be output to a peripheral device.

There are two main methods of outputting results:

The first is to cause the data to be transferred directly to the buffer of the device concerned when it will be instantly printed, punched, etc. This method called ON-LINE, can result in hold-ups if, for instance, the computer is dealing with more than one program simultaneously; for if there is only one line printer it can only be allocated to one program at a time so that another program requiring it will have to wait. (This ON-LINE method of outputting is dealt with in Section 17).

The second process, known as OFF-LINE, obviate such hold-ups. This method causes data which is to be output, to be copiedonto magnetic tape. This tape may then be removed from the computer for subsequent processing elsewhere.

A system called OUTS combines the advantages of ON-LINE and OFF-LINE outputting. The principle is, that at the discretion of the operator, data to be output may be either

(i) dealt with immediately ON-LINE or
(ii) copied onto magnetic tape for OFF-LINE processing, or
(iii) copied onto a magnetic tape, which need not be removed from the computer, and dealt with pseudo ON-LINE later.

12·2 OUTS

A program requests Director to output a block of data by the instruction OUT; with the integer 8 in N1, and by having previously specified the two folloing pieces of information.

12·2·1 Stream No. The first word of the block is to contain a "stream number". So that if the actual data to be printed say is in Y1-Y20, then the word immediately preceding this area, in this case Y0, must contain a "stream number". The stream number indicates to Director the type of unit for which the block is eventually destined. The programmer may choose any one of the eight stream in each group of streams he wishes.

Stream No. (Octal)	Destination Unit
00 (only one stream in this group)	On-line flexowriter
10 - 17	8-hole punch
30 - 37	high speed printer
50 - 57	5-hole punch
70 - 77	high speed printer or 8-hole punch

12·2·2 Output Block. Director must also be told the location of the block in the store. This is done by placing in N2, before OUT8 is performed, a

Q-store type word with the increment containing the low core address (i.e., the address of the stream number), and the modifier containing the high core address (i.e., the address of the last word of data). The counter should be set to zero, unless stream $00_{(8)}$ is required to type a query, when D0 of the counter must be 1. More will be said about typewriter queries in Section 17.

12·3 Director's Actions

When Director obeys OUT8 it writes the block on to the appropriate device, which it will previously have allocated to itself for the purpose. However, it does not directly copy the first word of the block, which contains the stream number:
BEFORE OUTPUTTING THE BLOCK IT OVERWRITES THIS WORD IN MAIN STORE WITH ANOTHER WORD FOR USE BY THE SYSTEM.
The programmer must always remember this when printing out the same area of store if necessary on the second and subsequent occasions - the stream number must be put into the first word every time OUT8 is to be obeyed.

12·4 Rules for various stream numbers

12·4·1 Stream $00_{(8)}$.

(i) Output stream $00_{(8)}$ will cause the data to be sent **direct** to the flexowriter.

(ii) The output on the flexowriter must be a single line and may NOT include CRLF or Tab characters.

(iii) If the output is a message, as opposed to a query, it may not include a semicolon.

(iv) If the output is a query, a semicolon must occur as the least significant character of one of the words, other than the first word (stream number word) or the last word. The semicolon may occur nowhere else. The reason for this will be explained later.

(v) Only data up to the first end message, $75_{(8)}$, if any, in the block will be output.

12·4·2 Streams other than $00_{(8)}$.

(i) Operator action will decide whether output will be dealt with direct or not.

(ii) No data after the first end message, $75_{(8)}$, if any, in the block will be output.

12·4·3 Additional rules for using streams $30\text{-}37_{(8)}$

(i) Streams $30\text{-}37_{(8)}$ are to be used for data eventually to be printed on the high speed printer. Since this printer does not possess the ability to print case shift characters a different code is used for which see (17·11·2).

(ii) The last character of the block **must** be either $02_{(8)}$ (line shift) $03_{(8)}$ (page change). No character other than the last of the block may be 02 or 03.

12·4·4 Additional rules for using streams $70\text{-}77_{(8)}$

(i) Data for these streams will cause output to be sent either to the high speed printer, or 8-hole punch if the former is not available.

(ii) Since output may be to printer or punch it is essential that only characters common to both Paper Tape and Printer Codes are used: these are:

Digits 0 - 9
Alphabet (capitals)
Symbols / ; + - ° $_{10}$ space

(iii) The last character of the block, and **only** the last, **must** be $02_{(8)}$
N.B. $03_{(8)}$ is not possible as there is no page change equivalent on paper tape equipment.

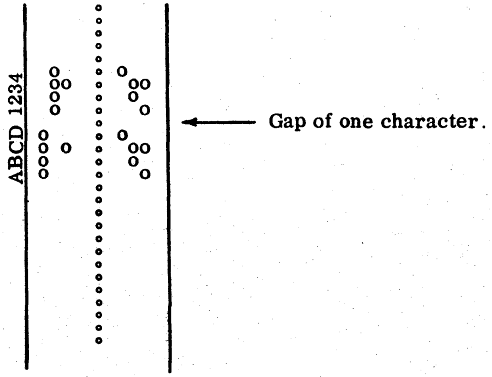
12·5 Standard Plug Board

What has been said above is for the Line Printer with standard wiring. This can be varied by changing the plugging of a Plugboard. Full details are given in the "Line Printer 1040 Users Manual".

12·6 Punched Tape

Paper tape is explained fully in Section 17; it is sufficient here, merely to state that along a punched paper tape is a series of very small holes (called sprocket holes) and whenever one of these occurs (i.e., every 1/10th inch), a pattern of larger holes, based on the Character code is punched across the width of the paper. This is then called a Character.

If no larger holes are punched where a sprocket hole exists then this is called a "gap" of one character length.



12·6·1 Gaps output in Punch Streams. In order to allow gaps to be output on punch streams a special 2 word block is used.

1st word is stream number
2nd word is a Q-type word 4095/-1/n where n is an integer. (0≤n< 512, if n<0 or n>512, n = 120 is substituted).

With these two words as the block a gap of n characters will be punched. (The punched version of a character occupies approximately 1/10th inch). If this block is used for printer streams, a page change will be output and thus it may be used for the printer/8-hole punch common streams, 70-77. The 2nd word will not be changed when this block is output, and is not output at all if the stream is being output direct.

12·7 Examples

1. If the declaration V1/4 = P[5S] SORT*ROUTINE*PRINCIPLE*18B[C]; which is intended to be used as a heading for subsequent output to the high speed printer is made, the following instructions would cause this heading eventually to be printed - the stream number chosen for the output of all data to this printer being, say, $35_{(8)}$. Q5 is assumed to contain Ø/AYPØ/AYP4;

```
SET B35;  =  YPØ

V1;  =YP1;  V2;  =YP2;  V3;  =YP3;  V4;  =YP4;

Q5;  SET8;  OUT;
```

Note that **after** the OUT8 has been performed N2(=Q5) and N1(=8) have been erased from the nesting store and that YPØ does **not** contain $35_{(8)}$. Also notice that the last character of the block, i.e., the C of [C] is entered as $02_{(8)}$ in D41-D47 of YP4.

(Although possible, it is generally not wise to print from the V-store area).

2. Assume (i) YR43 - YR48 contain six words, each in the character form (line printer) of ddddd⊔⊔⊔ where the d's are decimal digits, and that,

(ii) it is required to print these six numbers on one line of the high speed printer, and that,

(iii) the programmer is using stream $33_{(8)}$ for this printer.

A. If it is certain that YR42 and YR49 do not contain useful information required later in the program the process would be:

```
(i)    Declare, say,    V8  =  Q 0/AYR42/AYR49;
                        V9  =  B33;
                        V10 =  P [7DC];
```

(ii) and use the instructions

```
V9; =YR42; V10; =YR49;
V8; SET8; OUT;
```

B. If the contents of YR42 and YR49 are not to be lost the process would be:

(i) Declare, say,

```
V8 = Q 0/AYP0/AYP7;
V9 = B33;
V10= P [7DC];
```

(ii) and use the instructions

```
V9; =YP0; YR43;
=YP1; YR44; =YP2;
YR45; =YP3; YR46; =YP4;
YR47; =YP5; YR48; =YP6;
V10;  =YP7; V8; SET 8; OUT;
```

3. Assume the same word in YR43-YR48 as in Example 2, and that it is required to print the six numbers, one per line, on the high speed printer, and that stream $37_{(8)}$ is being used; the process would be:-

(i) Declare

```
V9 = QØ/AYPØ/AYP2;
V1Ø= B37;
V11= P[7DC];
V12= Q6/1/Ø;
```

(ii) and use the instructions

```
V12; =Q9; V11; =YP2;
1; V1Ø; =YPØ; YR43M9Q; =YP1;
V9; SET8; OUT; J1C9NZ;
```

4. Assume the same as in Example 3 except that the output is to be to the 8-hole paper tape punch using stream $16_{(8)}$ and that instead of each word being printed as in Example 3 they are each to be followed by ABCDEFG and $02_{(8)}$

The last item punched is to be followed by a gap of 9 inches.

The declarations would be

```
V18 = Q0/AYP0/AYP2;
V19 = Q0/AYP0/AYP1;
V20 = B16;
V21 = Q6/1/0;
V22 = PABCDEFG[C];
V23 = Q 4095/-1/90;
```

and the instructions:

```
V21; =Q9; V22; =YP2;
1; V20; =YP0; YR43M9Q; =YP1; V18; SET8; OUT; J1C9NZ;
V20; =YP0; V23; =YP1; V19; SET8; OUT;
```

12·8 Exercises - Set1

1. Declare the V-stores and write the instructions necessary to output to the high speed printer via OUT8, the words of data in YA47 and YR31. Both words are to be on the same line of the printer page with sixteen spaces between them. The contents of the other YA and YR stores are not to be mutilated. YP stores are at the programmer's disposal for printing.

2. Fifty items each containing 16 alphanumeric characters are stored in YA1-YA100. Declare the V-stores and write the instructions necessary to output these items to the 8-hole punch via OUT8. Each item is to be preceded by Case Normal, CRLF, and 6 "dummies". The paper tape is to start with a 12 inch gap before the first case Normal character, and end with a 12 inch gap.

Exercises - Set2

1. What are the effects of the following instructions and declarations?

```
          VØ  =  9;
          V1  =  P[P7D];
          V2  =  QU̅/AVØ̅/AV1;
      V2;SET8;OUT;
```

2. What are the effects of the following instructions and declarations?

```
          VØ  =  P[7S]/;
          V1  =  B0̅753̅444677003137;
          V2  =  B2ØØØØ11;
      V2;SETB1Ø;OUT;
```

13• JUMPS

13•1 Jump Instructions

We have dealt with jumps depending on the contents in Section 10•4, other jump instructions are explained below.

13•1•1 **Arithmetic Jumps.** It is often necessary to take one of two possible courses of action depending on the result of an arithmetic operation. KDF 9 has a set of suitable jump instructions for this purpose, all of which look at the contents of N1 and act according to the value found there. Since N1 is looked at, the computer follows normal practice and erases the contents of N1 after inspecting it, whether or note the jump actually takes place. Should the contents of N1 be required for subsequent use, a copy should be made before the jump instruction is obeyed.

The six alternative instructions are:-

Jr = Z; Jump to the instruction labelled r if the content of N1 is identically zero, otherwise proceed to the next instruction in sequence.

Jr ≠ Z; Jump to the instruction labelled r if the content of N1 is not identically zero, otherwise proceed to the next instruction in sequence.

Jr > Z; Jump to the instruction labelled r if the content of N1 is definitely greater than zero (i.e., if D0 is zero and at least one other digit is non-zero), otherwise proceed to the next instruction in sequence.

Jr ≥ Z; Jump to the instruction labelled r if the content of N1 is greater than or equal to zero (i.e., if D0 = zero), otherwise proceed to the next instruction in sequence.

Jr < Z; Jump to the instruction labelled r if the content of N1 is definitely less than zero (i.e., if D0 is a "ONE"), otherwise proceed to the next instruction in sequence.

Jr ≤ Z; Jump to the instruction labelled r if the content of N1 is less than or equal to zero (i.e., if D0 is a "ONE" or all digits are zero), otherwise proceed to the next instruction in sequence.

Note: The composite symbols ≤ and ≥ are obtained on a flexowriter by underline followed by the required symbol.

13•1•2 **Comparison Jumps.** These are two KDF 9 instructions which compare the contents of N1 and N2 and jump according to whether they are equal or not. These are non-standard in that, whilst both N1 and N2 are inspected during the instruction, only N1 is removed during the execution of the instruction (whether or not the jump takes places) leaving in N1 the word which was originally in N2. These are the only two instructions that look at a word in the Nesting Store, (N2) and do not erase it.

13·1·2 **Comparison Jumps**. These are two KDF 9 instructions which compare the contents of N1 and N2 and jump according to whether they are equal or not. These are non-standard in that, whilst both N1 and N2 are inspected during the instruction, **only N1** is removed during the execution of the instruction (whether or not the jump takes place) leaving in N1 the word which was originally in N2. These are the only two instructions that look at a word in the Nesting Store, (N2) and do not erase it.

The instructions are:-

Jr =; Jump to the instruction labelled r if the words in N1 and N2 are identical, otherwise proceed to the next instruction in sequence. Only N1 is erased.

J4 ≠; Jump to the instruction labelled r if the words in N1 and N2 are not identical, otherwise proceed to the next instruction in sequence. Only N1 is erased.

13·1·3 **Overflow Jumps**. It has been seen that if numbers get too large the overflow register is set but the computer will not stop. Instructions to 'jump' depending on the state of the overflow register are provided to enable the programmer to discover if overflow has occurred.

The instructions are:-

JrV; Jump to instruction labelled r if the overflow register is set, otherwise proceed to the next instruction in sequence. Overflow register automatically cleared.

JrNV; Jump to instruction labelled r if the overflow register is not set. Overflow register is automatically cleared.

It may be necessary to clear the overflow register other than in connection with the above jump instructions, in which case the instruction is:-

VR; Clear overflow register. No other part of the machine is affected.

13·1·4 **Test Register Jumps**. The test register is a single digit register used to interrogate input/output devices. The various questions that can be asked of such a device set the test register if the answer to the question is yes (if it is already set, there is no change), but leave it alone if the answer is no. Several questions can therefore be asked, the test register being set if any one or more give an answer yes. (The instructions necessary to ask the questions will be dealt with under INPUT/OUTPUT).

The test register can be interrogated by one of the following instructions:-

JrTR; Jump to the instruction labelled r if the test register is set, otherwise continue to the next instruction in sequence. This instruction clears the test register.

JrNTR; Jump to the instruction labelled r if the test register is not set, otherwise continue to the next instruction in sequence. This instruction clears the test register.

If necessary the test register can be preset by use of the instruction:

=TR; Set test register if the word in N1 has a "one" in the D0 position (i.e., if it is negative), otherwise clear the test register. The word originally in N1 is erased.

Jr; Jump to the instruction labelled r. As this instruction ALWAYS causes a jump, the next instruction must carry a label if it is to be obeyed. The label r is usually an integer in the range 1 to 1011, but the instruction may if required be replaced by one of the following forms:-

JPp; Jump to first instruction of subroutine Pp.

JL1; Jump to first instruction of subroutine L1.

JrPp; Jump to instruction labelled r in subroutine Pp.

JrL1; Jump to instruction labelled r in subroutine L1.

JrP0; Jump to instruction labelled r in main program. This will appear only inside private subroutines.
(PØ is the label for the main program.)

13·1·3 Unconditional Jumps (with return address). These jumps are intended for use with subroutines. When the jump is obeyed, the word and syllable address of the actual jump instruction (the return address) is stored automatically in the top cell of the Subroutine Jump Nesting Store, pushing down any addresses previously stored there. The subroutine is then entered and obeyed. At the conclusion of the subroutine the address stored is used to return to the main program.

It should be noted that each instruction in this group starts JS. The S indicates that the return address is to be stored - if this is omitted the jump into the subroutine will still take place but the return address will not be available, leading to eventual failure when the Jump Nesting Store is empty, and an address is required to exit from a subroutine.

The instructions are:-

JSPp; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the first instruction of subroutine Pp.

JSLl; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the first instruction of subroutine Ll.

JSrPp; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the instruction labelled r in private subroutine Pp.

JSrPO; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the instruction labelled r in the main program. This should appear only in PRIVATE subroutines.

JSrLl; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the instruction labelled r in the library subroutine Ll. This instruction should be used only if the operating instructions for the subroutine indicate that label r is a recognised entry point.

JSr; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the instruction labelled r in the current level of program.

13·1·4 Less used Jump Instructions. The following four instructions are intended for use by Director or certain Monitoring programs, which must empty the nesting stores, but have no other means of knowing if such stores are empty or not. They have no place in other types of program, as it is always possible to predict whether a nesting store will be empty or not at any point in a program - if used in a subroutine the result will probably be disastrous.

JrEN; Jump to the instruction labelled r if the nesting store is empty (i.e., all 16 cells unoccupied).

JrNEN; Jump to the instruction labelled r if the nesting store is not empty (i.e., at least one cell is occupied).

JrEJ; Jump to the instruction labelled r if the subroutine jump nesting store is empty (i.e., all 16 cells unoccupied).

JrNEJ; Jump to the instruction labelled r if the subroutine jump nesting store is not empty (i.e., at least one cell is occupied).

There are two other jump instructions intended for use in passing from one section of a program to another, where the sections are too large to be compiled in one sequence. In these circumstances, reference labels cannot be used as they are not available to Compiler at the requisite time, so the absolute word location is used instead.

Further, in order to make such a technique possible, Compiler must be directed to put a particular instruction in a predetermined store location. A Compiler specification therefore exists for this purpose and is included here.

JEe; Jump to the first syllable of word Ee.

JSEe; Store the address of this instruction in the top cell of the subroutine jump nesting store, then jump to the first syllable of word Ee.

REe; A specification to Compiler that the next instruction is to be compiled and store in word Ee. Subsequent instructions are stored in the next available space beyond Ee in the normal way.

13·2 Examples

(i) N1 contains a binary number obtained as the result of calculation. If it is required to put this into YPØ if the number is positive or zero, or into YNØ if it is negative, the method would be:

```
       DUP;    J6>Z;    =YNØ;    J2;
    6; =YPØ;
    2; ..............
```

(ii) YBØ contains seven spaces and an alphabetic character obtained as the result of an input instruction. V19 contains seven spaces and the letter R obtained by the V-store declaration V19 = P[7S]R; If it is required to cause the program (i) to jump to reference label 92 should the letter in YBØ be R, otherwise (ii) allowing the program to proceed in sequence, in either case leaving N1 clear, a method would be

```
       V19;  YBØ;  J92=;  ERASE;.......
       92;  ERASE;.......
```

(iii) Many calculations have previously been performed and overflow was not checked on the last occasion when it could have been set, and it is now required to add the contents of YN6 into YM15 and to set YM15 equal to zero if the overflow is set as a result.

```
       VR;  YN6;  YM15;  +;  J1NV;  ERASE;  ZERO;
     1; =YM15;
```

(iv) To cause a jump to reference 8 if the Test register is set the instruction would be J8TR;

13·3 Exercises - Set1

1. There are 1,024 binary integers stored from YØ onwards. Store in C1 the number of positive integers in this set, in C2 the number of zero integers in this set, and in C3 the number of negative integers in this set.

2. N1 contains the integer x. YØ-Y31 contain a set of 32 integers. Store those integers greater than x from YAØ onwards, those less than x from YBØ onwards. Count the number equal to x and leave this count in N1. Leave the count of the number of integers greater than x in N2, and the number of integers less than x in N3.

13·3 Exercises - Set2

1. What is the "set" instruction to put $19 \cdot 125_{(10)}$ into the top cell of the nesting store as a fixed point number held to 40 integral places.

2. Is it possible via a "set" instruction to cause N1 to contain the Q-format 0/0/-1; ?

14·1
Identical sequences of instructions are sometimes required in many different programs and so that the programmer may be relieved of the tedium of writing out these sequences and testing them every time they are needed a magnetic tape is kept containing all those sequences likely to be used. When a program is written requiring one of these sequences called "Subroutines" - it can be extracted from the tape and included in the program by using the appropriate instructions, **provided the program is compiled using the POST system** - to be explained later.

14·1·1 At any point in a Usercode program text the declaration library may occur, followed by a list of library subroutine identifiers in numerical order, each subroutine identifier being separated from the next, by a comma, and the list terminated by a semicolon. The library subroutine identifier must be of the form: one alphabetic character and up to 4 digits, e.g.,
library L3, L13, L14, L1000; (see Section 14·3).

This has the effect of inserting the appropriate subroutines at the point in the program at which the instruction occurs, by copying them from the POST SYSTEM TAPE.

Although this declaration may occur anywhere in the program, it is recommended, that it should be placed at the end of the program immediately prior to FINISH;→

14·1·2 The subroutine having been extracted from the library it is necessary to ensure that the subroutine be entered when required. This is achieved by the instruction JSL1; which causes a jump to the first instruction of subroutine L1 at the same time placing the address of the instruction JSL1; in the top cell of the subroutine jump nesting store.

14·1·3 To leave the subroutine and return to the next instruction after JSL1; the subroutine ends with the instruction EXIT n; where n is usually 1 or 2. This instruction obtains the address in the top cell of the subroutine jump nesting store adds n half words and jumps to that address. Since the instruction JSL1; is 3 syllables ($\frac{1}{2}$ word) long then EXIT 1; will cause the instruction following JSL1; to be obeyed next. The uses of EXIT n for n other than 1 and further reference to library subroutines will be dealt with later in Section 23.

14·2 Examples
1. A program is in the process of being written and is at the stage when N1 contains an integer. It is required to process this integer by a method identical to subroutine P98 which is on the POST SYSTEM TAPE and then to store the result in Y5. The subroutine P98 requires an integer in N1 on entry, and exits by EXIT 1. The instructions necessary are:

```
.......;JSP98;=Y5;...............;
library P98; FINISH;→
```

2. The next instructions to be written in a program are to cause the present contents of N1 (which holds an integer number of binary pence) to be converted into £. s. d. so that N1 holds the integer number of binary pounds
N2 holds the integer number of binary shillings
and N3 holds the integer number of binary pence.

The programmer knows that (say) subroutine P43 does this conversion but that P43 itself requires to enter a subroutine number R18 during the process. P43 exits with EXIT 1; and that Y1, Y2, Y3 are then to hold £ s. d. respectively.

The instructions necessary are:

```
............;JSP43;= Y1;=Y2;=Y3;.......
....................:
....................;
library P43, R18;
FINISH;→
```

14·3 Library calls format recommendation

Although library L1,L15,L94: is valid it is recommended that these calls be written as

```
library L1;
(L1);
library L15;
(L15);
library L94;
(L94);
```

The reason for this is that when the Post System causes the Usercode program test to be output on the line printer only the word LIBRARY is shown for library L1, L15, L94; whereas by using the above recommended calls the following would be shown:-

```
LIBRARY
(L1);
LIBRARY
(L15);
LIBRARY
(L94);
```

14·4 Exercises - Set1

1. Y4 contains a positive (non-zero) binary integer and it is required to convert it to character form. L1000, a subroutine to do this, requires a positive binary integer in N1 on entry and leaves the character form of the integer in N1 and N2, the result being right justified with more significant zeros, (including zero itself) replaced by spaces. Write the instructions (and the library call message assuming the POST system will be used for the program) necessary to use the subroutine ending by placing the more significant half of the result in YP1 and the less significant half in YP2. L1000 exits with EXIT 1; and uses Q15 and 3 cells of

the nesting store.

14·5 Exercises - Set2

1. Y4 contains a binary integer (+ve or -ve). Repeat the previous question but in addition to using the subroutine place the sign of the integer (as $35_{(8)}$ or $36_{(8)}$ in D42-D47 of YPØ with the first seven characters as dummies.

[Zero is to be considered as +ve and only one Ø (the least significant) is to be printed if the integer is zero.]

15· THE CONCEPT OF FLOWCHARTING

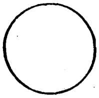
15·1

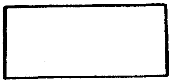
When programming extremely short and simple routines, it is sometimes possible to write down the instructions immediately, but generally it is necessary to have a schematic picture of the routine prepared to assist the programmer. This picture known as a "flowchart" depicts the routine in detail and from it one can follow the various stages easier than by reading a script of the same thing.

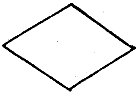
A flowchart comprises many differently shaped boxes in each of which is written the action to be taken at that stage. The boxes are linked together by arrowed lines so that the sequence in which the boxes are to be dealt with is determined.

15·2 Conventions for Flowcharts

There are three types of box:

(i)  Start, End and Connector, each of which has only one exit or entry line.

(ii)  Operational. Only one entry and one exit line.

(iii)  Decision. Only one entry and two exit lines.

The first has written in it "START" or "FINISH" or "the point of connection". The operational box contains a statement of what action is to be performed. When there is the possibility of more than one path to be followed depending on the value of a particular parameter a decision box is used wherein is written the criterion determining the next box to be obeyed. A connector is used much in the same way as "continued on . . ." and "continued from . . .".

The statements etc. written in the boxes could be written in normal English but the flowchart will be more concise if the following abbreviations are used:

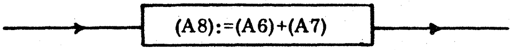
The **address of a word** is shown as e.g., A5, or Y9 when the actual address is known. When the address is known to be x words after say AØ but the value of x is variable then the convention used is e.g., A[x].

The **contents of an address** are shown by the address enclosed in round brackets, e.g., (A[x]).

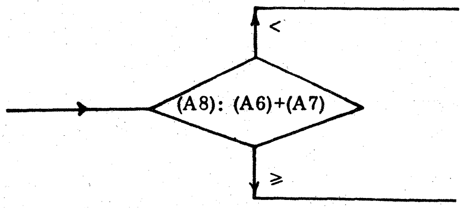
Symbol	**Meaning**
+	as in normal arithmetic.
-	as in normal arithmetic.
×	as in normal arithmetic.
/	as in normal arithmetic.
÷	integer division, giving an integer quotient and remainder.
=	"is equal to" (never used in an 'operational' box).
≠	"is not equal to".
>	"is greater than".
<	"is less than".
⩾	"is greater than or equal to".
⩽	"is less than or equal to".
:=	"becomes equal to" (only used in an 'operational' box).
∽	"is interchanged with"
:	"is compared with".

Particular attention must be paid to = and :=

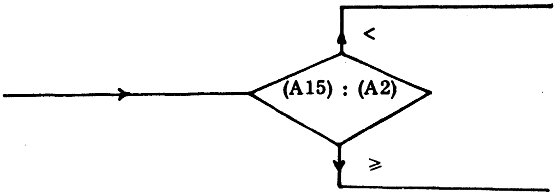
To emphasize the difference between them consider the situation when the addresses A6 and A7 contain two numbers and it is required to place the sum in A8. Then the order would be



However should it be required to take one path if the contents of A8 equals the sum of the contents of A6 and A7 and a different path otherwise, the decision would be

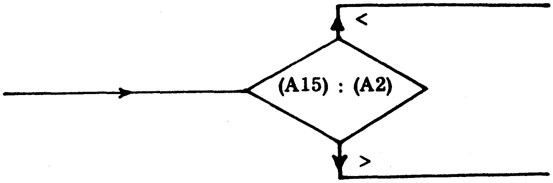


Notice here that the symbol : has been used to show that a comparison has to be made. In such a comparison it is the left hand term which is referred to as in the next sample.



Here, if (A15)< (A2) the upper path is taken otherwise the lower path is to be followed.

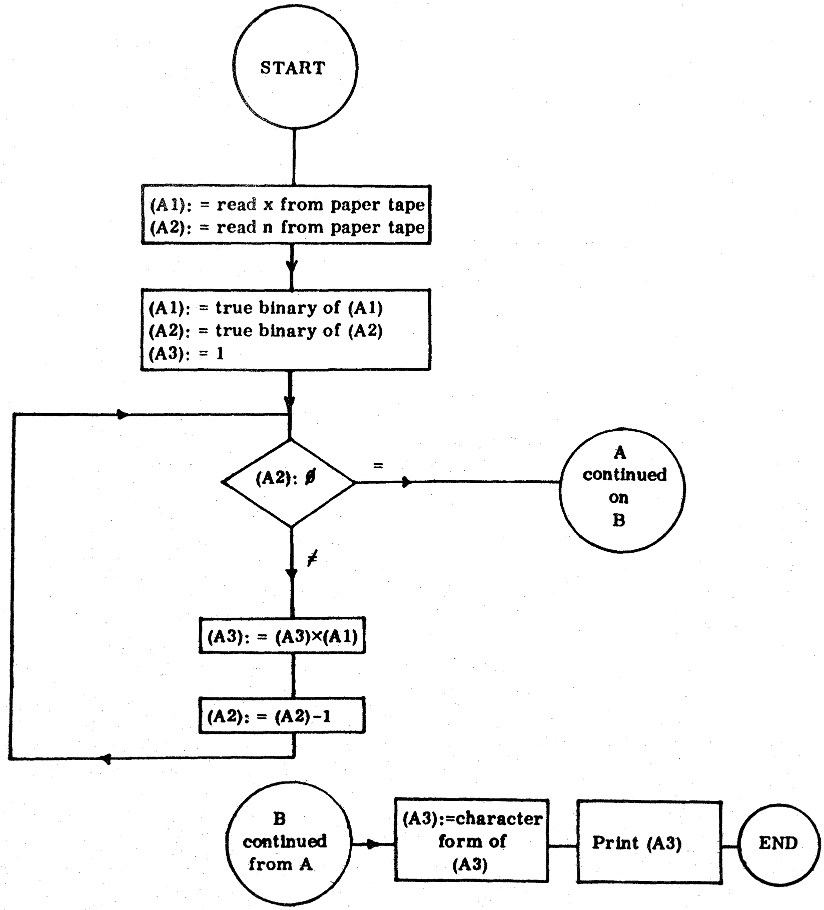
It is essential that the exits from a decision box should cover all possible results. Had the last example been written as



then it would be impossible to know which path to follow when (A15)=(A2).

15·3 A Simple Flowchart

Below is given the flowchart of the process to print the nth. power of x, when x and n are on paper tape in character form both being integers greater than zero.


START
(A1): = read x from paper tape
(A2): = read n from paper tape
(A1): = true binary of (A1)
(A2): = true binary of (A2)
(A3): = 1
(A2): ∅
=
A continued on B
≠
(A3): = (A3)×(A1)
(A2): = (A2)-1
B continued from A
(A3):=character form of (A3)
Print (A3)
END

The first box after START states that the addresses A1 and A2 are to contain x and n respectively by reading these values from paper tape.

This is followed by changing the contents of A1 and A2 to contain the binary representation of x and n which are in character form. A3 is to contain the binary integer 1.

Exit from the next (decision) box every time it is entered is along the ≠ path provided the contents of A2 are not zero. A3 is then made to hold its previous value multiplied by the contents of A1. The contents of A2 are then reduced by 1, and the exit line goes back to just prior to the decision box which is again obeyed. This process causes A3 to hold 1, then x, then x X x, then xXxXx, etc., until A2 holds zero when A3 holds xn,. By this time A2 will contain Ø and the "=" path will be taken from the decision box.

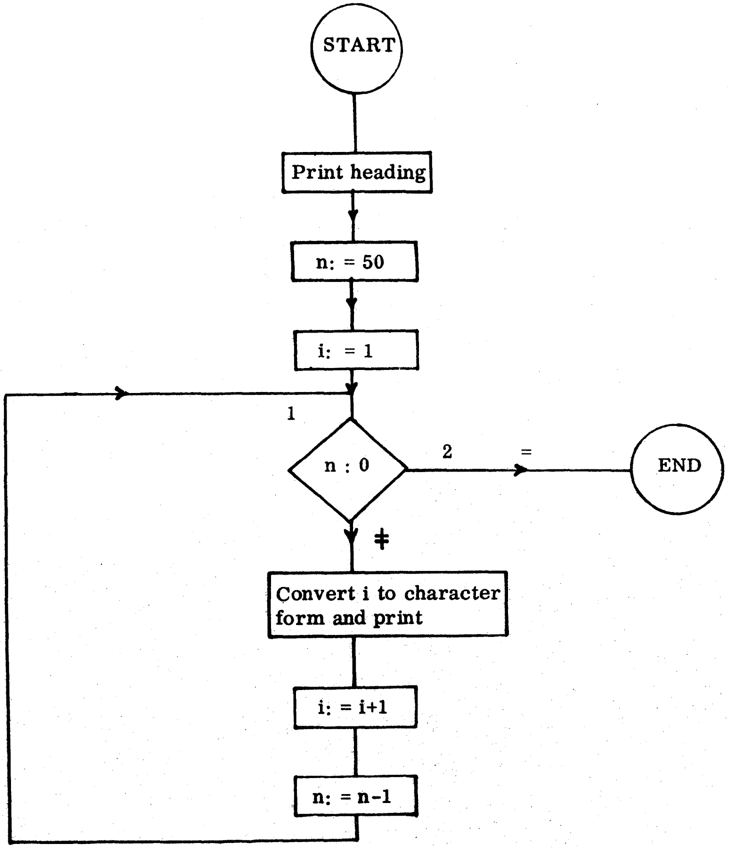
A connector box directs us to the bottom of the chart and to the orders to change the contents of A3 into character form and print.

16•1

A Usercode program will now be written to generate the integers 1, 2, 3 .... up to 50, and print them one per line on the line printer using OUT 8, stream no. $3\emptyset_{(8)}$.

A library subroutine is available, entry to which is by JSL1000. It requires a positive (non zero) binary integer in N1 on entry, and exits by EXIT 1 with N1 and N2 containing the decimal representation in character form of this integer. N1 is more significant than N2 and the characters are right justified with the more significant zeros replaced by spaces, i.e., octal 00. It is to be understood that the program will be compiled on the POST system. The output should be headed with a title.

The flowchart for this routine is as follows:



N.B. Although library subroutine L1000 is available for programs run by Training Department, Kidsgrove, it is not as yet (February 1966) in the normal KDF 9 library.

16·2 Solution

It is recommended that

(i) programs be written in pencil on alternate lines

(ii) the finished version is not copied out "for neatness sake" - invariably this results in an error, (of course this recommendation has to be tempered with common sense. If the program is rewritten great care must be taken not to miss the odd ; or =)

and that

(iii) Constant declarations be written on a separate sheet from the instructions.

(The following should be read in conjunction with the final version at the end of this section).

The first box requires a heading to be printed, which is to be output via OUT 8 Stream $30_{(8)}$ so write V0 = B30;. Let the heading be INTEGER GENERATION. This will require a P-constant; so write V 1/3 = P INTEGER*GENERATION [5DC]; declaration. To cause this to be printed will require N2 to contain the Q type word 0/AV0/AV3; so set this as V4. The instructions V4; SET8; OUT; can now be written - **note that this OUT8 instruction will cause V0 to be overwritten.** The solution to this question has been written using V-stores for output purposes merely to indicate to the reader that such is possible. However, in general, output should always be from YA-YZ or Y stores, - this may mean that a few V-stores will have to be transferred to another area for output.

The next requires 50 to be stored somewhere and it is noticed that this reduces by 1 every time round the loop - this suggests it should be kept in the counter position of a Q-store. Similarly the integer 1 needs to be stored and this is incremented by 1 every time round the loop - indicating that it could be stored in a modifier position with the increment position of the same Q-store as 1. So put the next constant declaration as V5 = Q 50/1/1; with the instructions V5; =Q1;.

We now come to the loop. Since re-entry to the loop just before the box n:0 will be needed it is necessary to give the junction a reference label, say reference 1. Exit from the decision n:0 is to one of two places this means that at least one of the exits also needs a reference.

Let us call the exit, which goes to FINISH, reference 2. So before we write the instructions to deal with n:0 write the label 1, at the beginning of the next line. n:0 is now dealt with by the instruction J2C1Z; If n ≠ 0 when the box is obeyed no jump to reference 2 will occur so the next instruction in sequence will be M1; to cause N1 to hold the current value of i which is in M1.

Library subroutine L1000 extracts N1 and leaves in N1 and N2 the character form. So we need JSL1000; Since the largest integer to be printed is 50 we see that N1 will always be left as eight spaces by the subroutine, we can therefore quite conveniently erase it from the nest - using ERASE; Nesting up will have taken place and we now need to cause the new N1 to be printed. Let us use YP stores for this printing area - YP0 holding stream number, YP1 the integer in character form, YP2 seven dummies and a CRLF. So the next instruction will be =YP1; To get the stream number into YP0 we need another V store =B30 (remember that V0 has been overwritten). So write V6 =B30; and the instructions V6; =YP0; For YP2 to hold [7DC] we need V7 = [7DC] ; and the instructions V7; =YP2; BUT it is not necessary to have this pair of instructions inside the loop. YP2 does not become overwritten by the OUT8 instruction (only the first word of the area, the stream-number, is overwritten) so why reset YP2 every time round the loop? - place it in just before the loop.

The area YP0-YP2 can now be printed, so a V store declaration V8 = Q 0/AYP0/AYP2 is needed with the instructions V8; SET8; OUT;

All that is required now is to update i and n which is done by M+I1; DC1; To get back to the box n:0 we use J1; . We can now enter reference 2 and end the run by ZERO; OUT; .

The program is now finished apart from entering the library call message <u>library</u> L1000; and final instruction FINISH;→.

Before entering the program for typing, the programmer should check that the loop does not "generate" or lose a cell in the nesting store every time round. Do this by noting how many cells are filled when 1; is encountered first time, then pass through the loop adding and subtracting to the count as appropriate for the instructions met until you are led back to reference 1. At this point the count MUST be the same as it was when 1; was first encountered.

It is programming etiquette to ensure that the nest is empty when ZERO; OUT; is obeyed. We notice that there are no cells filled when J2C1Z; is obeyed which means that no cells are filled at reterence 2.

Now have a look through what you have written to ensure that all ;'s are entered and that all instructions are valid.

```
VØ    = B3Ø;
V1/3 = P INTEGER*GENERATION [5DC];
V4    = Q Ø/AVØ/AV3;
V5    = Q 50/1/1;
V6    = B3Ø;
V7    = P[7DC];
V8    = Q Ø/AYPØ/AYP2;

        V4; SET8; OUT; V5; =Q1; V7; =YP2;
1;      J2C1Z; M1; JSL1000; ERASE; =YP1; V6; =YPØ;
        V8; SET8; OUT; M+I1; DC1; J1;

2;      ZERO; OUT;
        library L1000;
        FINISH;
        →
```

N.B. The reader may now like to improve this programme by using only one jump instruction (J1C1NZ) instead of two (J2C1Z and J1). He should consider the implications if "n" initially is zero.

16•3 Exercise

The reader is now asked to write a program using JSL1000 and OUT 8 (Stream $30_{(8)}$) to cause a heading to be printed followed by the first 40 integral powers of 2, one per line. The flowchart to be used for this is:

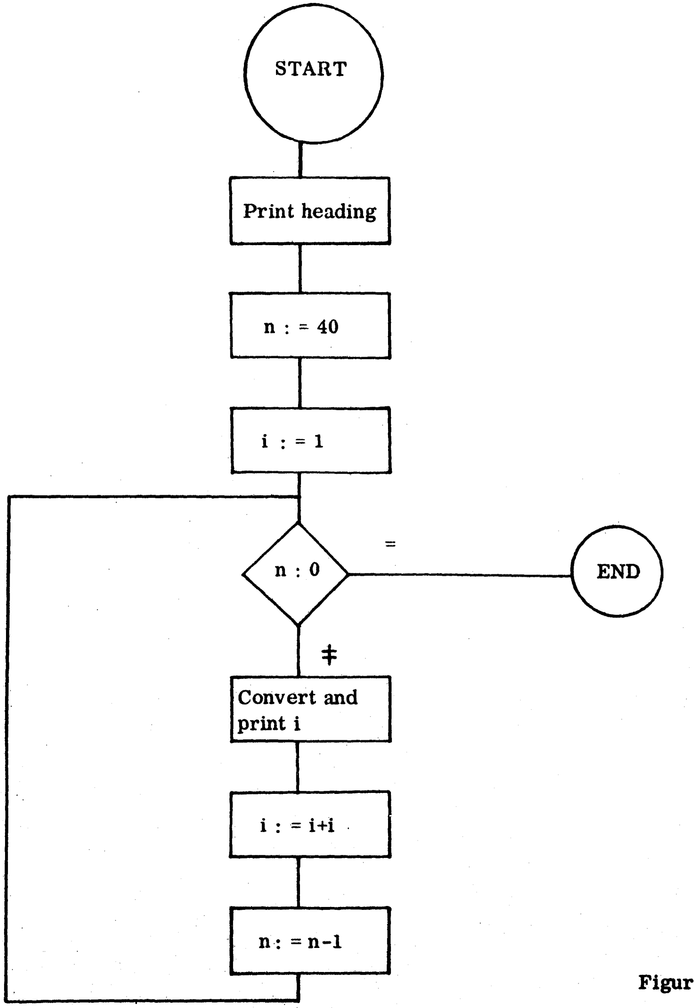


Figure 9

The heading is to be of the format:
NAME
blank line
TITLE
blank line

Do not, use V stores for output.

17·1 Basic Requirements

Most computer programs will require at some stage to be supplied with additional data, and certainly the vast majority will be expected to produce the results of their calculations in some tangible form. For these purposes a set of input/output instructions is required. Since no two programs require data in the same layout, this set of instructions must of necessity possess a large degree of flexibility. For a computer like KDF 9, which can be fitted with up to 16 input/output devices, the instructions must further include some provision for allocating the appropriate device for a given purpose.

In normal usage a library of subroutines will be available to perform all the necessary input/output operations. This library will contain a generous selection of routines, any one of which may be further tailored in certain specified ways to meet a given requirement. It may be the case, however, that such a modified subroutine will be too clumsy and inefficient for the use to which it is to be put. Again, it is always possible that a new special purpose input/output routine will be required. For these reasons all KDF 9 programmers should know how to write input/output routines for themselves. OUT 8 which was dealt with earlier, is an example of an output routine which is incorporated in the Director program.

The following presents the basic Usercode instructions concerned with input and output, with the rules governing their use, and will enable programmers to write this type of routine whenever the need arises.

To perform any basic input/output operation on KDF 9, three pieces of information are needed:-

(a) The nature of the operation, (e.g., reading from paper tape, or output on line printer).

(b) The particular device to be used, (e.g.,there may be two paper tape readers, both will be of type 2 but will have different device numbers).

(c) A specification of the quantity of data concerned, either as the area of main store involved in a transfer or as a simple count.

Item (a) is known at the time the program is written, and no further comment need be made here. Items (b) and (c) are in a different category because they may be unknown until the program is actually run. The input/output device that will be used is never known until run time: the programmer, when writing his program, will not know which of possibly two devices of the same type will be allocated to the program by the Director at run time.

In certain conditions item (c) is also unknown until the program is run, for instance if variable-length items are involved.

Therefore items (b) and (c) must be written into the program in such a way that they can be adjusted as the program is running by the incorporation of extra information. The Q-stores are used for this purpose, each input/output instruction nominating one of the fifteen Q-stores as the location of this extra information. It is the responsibility of the programmer by the use of the appropriate instructions within the program, to ensure that the correct information, once it has been discovered, is put into the appropriate Q-store before the input/output instruction is obeyed. If this is not done the machine will be unable to proceed with the operation, since it is into the Q-store that the machine looks for its directions.

17·2 Device and Type Numbers

Each individual input/output device has a permanent Device No. and further, each type of device has a type No.

The programmer then requests use of a device by quoting the Type No., Director allocates to the program an appropriate device and then informs the program which particular device is involved by leaving its Device No. in the top cell of the nesting store.

This technique facilitates the interchange of programs between different machines with different numbers of each type of device.

17·3 Outs

Control is transferred to Director by use of the special instruction:

OUT;

When this instruction is obeyed, Director extracts the contents of N1 of the nesting store and then performs a specific operation depending on the integer found. Sometimes, depending on the value of the integer, Director again extracts the contents of N1 (the N2 before OUT; was obeyed) for auxiliary information. The integers chosen for use in N1 for direct input/output are 4, 5, 6, and 10. These entries into Director are usually referred to as OUT 4, OUT 5, etc., but it should be remembered that the written instruction is OUT; and that the integer appears in N1. OUT 8, in Section 12, is an example of this, where extra information was required in N2.

Integers in N1 other than 4,5,6, and 10 used in connection with the instruction OUT; have uses not concerned with input/output, and will not be considered here.

17·3·1 **Out 5**. This is the request to Director for any of the input/output devices, other than the magnetic tape, drum and disc file units, to be allocated to the program. Out 5 only causes the device to be reserved to the program, it does not cause any transfer of data to occur. One of the reasons for the need for a program to claim a device is to prevent mixing of a time sharing machine where probably four programs are running concurrently. The transfer of data instructions is dealt with later.

For the OUT 5 entry to Director, N1 contains the integer 5 (47 integral places) and N2 contains an integer defining the type of input/output device required. The integers in N2 (47 integral places) may be any of the following:-

INTEGER (Type No.)	Device required
1 -	Paper tape punch
2 -	Paper tape reader
3 -	High speed on-line printer
4 -	Card reader.

(This list will be extended as required).

As an example of this instruction, consider a program requiring access to a paper tape reader. The set of instructions to obtain the appropriate device number would be:-

SET 2; SET 5; OUT;

When control is returned to the program, Director will have placed the appropriate device number in N1 as an integer (47 integral places), while the original integers 2 and 5 have been removed from the nesting store.

When a device is allocated to a program, the Director will type a message to the Operator stating which device has been allocated (as a record of what is happening and also state which device of that type will be allocated NEXT when the next OUT 5 to that type is obeyed. Director keeps a tally of devices of each type and allocates them in cyclic order.

It can happen that a particular program is one of a series of programs always obeyed sequentially. If all require just one of a particular type of device, but the computer has more than one such device available, it is often advantageous if all use the same one, to minimise operator actions in, for example, setting up printers or loading paper tapes.

This can be achieved by informing Director, whenever a device is claimed, that it will eventually be required again. This information is given by adding the integer 8 to the integer in N2 specifying the type of device (i.e., putting 11 in N2 for a line printer). Director will now NOT designate another device as next to be allocated.

It will be noticed that no code number for the on-line typewriter has been given. In fact the typewriter always has device number 0, and there is no need for it to be claimed by OUT 5 (or deallocated by OUT 6). All programs have ready access to the typewriter, but programmers always use it in extreme moderation since it operates at only ten characters per second and is shared by all programs.

Since the device claimed by OUT 5 will subsequently be required to read or write and then eventually needs to be deallocated before the program ends, and further since these transfer and de-allocation instructions will require the Device No. to be in the Counter position of a Q-store it is wise to place it there immediately after OUT 5.

Assume the program uses Q9 for the purpose of data transfer instructions of the paper tape reader, then the instructions to claim the device would be

```
                    SET2;   SET5;   OUT;   =C9;
```

With large programs it is advisable to keep a copy of the device number in a main store word in order to avoid extra programming in case of the counter being overwritten in programming error. The instructions may take the form SET2; SET5; OUT; DUP; =C9; =YD1;

17·3·2 **Out 6.** This is the entry to Director for de-allocating an input/output device. Before a program run is concluded it is always necessary to de-allocate all the devices used, otherwise any unfinished input/output transfer is liable to be truncated when the concluding entry to Director is made. This catastrophic truncation will not occur if the OUT 6 directive is made for each device: the program will not conclude until all the de-allocations have been completed, which in turn will not occur until all transfers to and from each device are concluded.

OUT 6 requires the nesting store to contain the integer 6 in N1, and in N2 the number of the input/output device to be de-allocated. When Director has taken the necessary steps, control is returned to the program, the contents of N1 and N2 having been erased.

If the device de-allocated is a tape station, Director will type out instructions to the operator to unload it. If the device is one of the others, Director will type the instructions to unload it and further will inform the operator if it is the next one to be used for input (this will occur if no designation was made when the previous allocation occurred).

When all the data transfer instructions for a device have been completed and the device number is in (say) YD1 then the instructions to de-allocate the device would be

```
                    YD1;   SET6;   OUT;
```

17·3·3 **Out 4.** OUT 4 instructs Director to locate a particular magnetic tape, which must be specified before the OUT instruction is obeyed by putting in N2 a quantity known as the tape identifier. This tape identifier is a pattern of eight characters which also must appear in the first block of every magnetic tape, a different identifier being assigned to every magnetic tape. When Director is entered, it locates the device on which is mounted the tape with the stated identifier, and supplies this device number back to the program. The device concerned is then allocated to the program and may be used for reading from or writing on to magnetic tape.

This facility makes it possible to load magnetic tapes in advance of requirements on whatever devices happen to be available, resulting in more efficient use of the machine. If the required tape is not discovered, the operator will be informed and asked to find the tape and to mount it on any one of the unused magnetic tape stations. Tapes which are to be used for temporary working should have a completely zero identifier and are referred to as "zero work tapes". If an output tape is required for later use it should have an identifier written on it by the program at output time. Since each magnetic tape carries its own identifier it should never be possible for tapes intended for one program to be claimed by another program. The identifier system will avoid costly confusions of this sort.

When control is returned to the main program after entering Director, the words originally left in N1 and N2 will have been removed and the device number, as an integer, will be in N1.

It will not be possible (in the general case) for the program to allocate identifiers to output tapes without operator assistance.

To claim the magnetic tape unit holding the magnetic tape with identifier KEP7D001, the form of instructions necessary is

```
        V2   =   P KEP7D001
      V2;SET4;  OUT;  DUP;  =C5;=YD7;
```

This causes the device number to be placed in C5 and YD7.

17•3•4 **Out 10.** Some magnetic tapes have identifiers of + followed by 15 characters. To claim one of these N2 will contains the more significant half of the identifier and N3 the less significant half. N1 will contain the integer 10. Otherwise the principle is as for OUT 4, except that after OUT1Ø, the tape serial number is left in N2.

To claim the device with the magnetic tape whose identifier is +KEDATAPRESET005 the form of instructions would be:

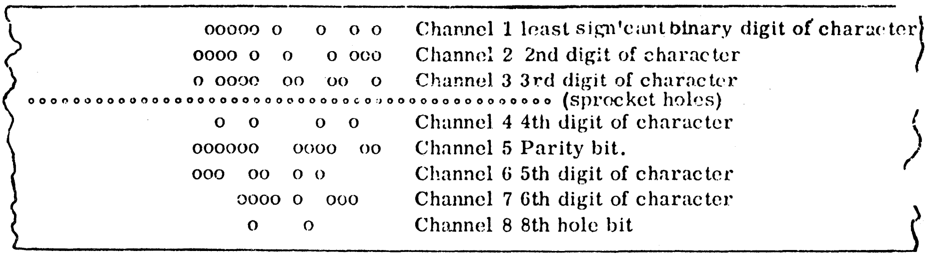
```
        V 7/8   =   P+KEDATAPRESET005;

  V8;  V7;  SET10;  OUT;  DUP;  =C3;  =YD2;  (TSN now in N1)
```

Deallocation of magnetic tape units is via OUT6. Instructions for transferring data to and from magnetic tape are dealt with in a later chapter.

17•4 Paper Tape

The majority of KDF 9 installations use 1" wide 8 channel tape. Before use, it is a roll of unperforated paper with arrow-heads printed every 7" indicating direction of the tape through punch and reader. When punched, a pattern of holes appears across the width of the paper together with a sprocket hole every 1/10" with 1 bits punched holes and 0 bits unpunched. The layout of the channels of 8 hole paper tape is indicated in the example of 8 hole paper tape below.


Channel 1 least sign'cant binary digit of character
Channel 2 2nd digit of character
Channel 3 3rd digit of character
(sprocket holes)
Channel 4 4th digit of character
Channel 5 Parity bit.
Channel 6 5th digit of character
Channel 7 6th digit of character
Channel 8 8th hole bit

By closely inspecting the above diagram the reader will notice that for each character (6 bits in main store)

(i) the most significant octal digit is held below the sprocket hole line in channels 7, 6 and 4

(ii) the least significant octal digit is held above the sprocket hole line in channels 3, 2, 1

(iii) there are two extra channels, 5 and 8, which do not form part of the Character Code.

E.g., Consider the character K, octal 53; this is held on tape as
(i) 101 below the sprocket hole line by holes in channels 7 and 4, and
(ii) 011 above the line by holes in channels 2 and 1.

The paper tape reader operates at 1,000 characters per second and the paper tape punch at 110 characters per second. Paper tape is therefore a slow medium for input/output compared with magnetic tape. However, this has certain advantages in use since it is possible at any time to stop either of these paper tape devices between two adjacent characters. This means that paper tapes do not require gaps between successive groups of characters, although it may often be desirable to leave such gaps if only to make a tape easier to inspect visually away from the machine.

The reader is arranged to read 8 channel tapes, but it can be converted to read 7 or 5 hole tape (each of which had a modified character code) by operating a manual switch and chaning a plug on the reader which is wired to re-arrange the input channels to appropriate channels.

17·5 Parity and 8th. Hole

Although only 6 information bits are necessary for the character code (8 hole) it is necessary as precautionary measure to have 2 extra bits on the data media. The two bits are the "Parity" and "8th. hole" bits. When data is prepared on the Flexowriter, Paper tape punch, magnetic tape, etc., these 2 bits are automatically inserted as necessary. When the data is read from the media into the main store these 2 bits are not transferred.

(i) Parity. As a protection to ensure that the data medium has not been mutilated since being prepared, the number of bits in a 6-information-bits-group are automatically counted and depending on the number of bits an extra bits is placed in a special position. There are two methods of approach.
(a) Even Parity. If the number of bits in the 6 information bits is odd an extra bit is inserted thus making the number of bits now "even".

(b) Odd Parity. If the number of bits in the 6 information bits is even an extra bit is inserted thus making the number of bits now "odd".

The device which reads the data checks that each group of bits (6 information bits plus the 2 extra bits) has correct parity, if the group passes the test then the 6 information bits only are transferred, otherwise a failure is indicated. Paper tape works on "even" parity and magnetic tape on "odd" parity.

If during reading, a character does not pass the Parity test, the Parity Indicator (a single bit register) on the device is set and reading automatically and instantly ceases and the paper tape does not move on to the next character. With the paper tape reader, parity checking is performed for 8-hole tape only (but may be suppressed if desired by a manual switch). If the reader is switched to 5 or 7 hole tape, the parity checking is automatically removed. When parity checking is off, the automatic recognition of spaces (gaps on tape) and delete (all channels) is also stopped therefore all characters will be transferred to the main store and the program must edit them accordingly.

(ii) 8th Hole. Blank tape and tape where all bit places are punched are ignored. This means that if the space character (octal 00) is to be transferred the tape would be blank and so ignored. To overcome this, provision is made for another bit, which is used, together with the parity bit, to specify the space character. This extra hole is also used for "delete".

17·6 Fixed and Variable-length Data

If the program is such that the amount of data to be transferred is fixed in length for each run, then it is sufficient merely to specify the area of main store involved by quoting the addresses of the first word (called the low core address) and of the last word (called the high core address).

When the amount of data varies from run to run, the low and high core addresses quoted are those which will allow the maximum amount of data likely to ever occur. The data is prepared finishing with → (Octal 75) called the "end message symbol". A special data transfer instruction is then used so that only data up to the →is sensed, thereby only filling, generally, part of the main store area specified.

17·7 Paper Tape Instructions

PRQq; Causes a block of information to be read (from paper tape mounted on the device whose number is in the counter position of Q store q) and transferred to the region of main store specified by the low core address in Iq and the high core address in Mq. Only the six information channels are transferred to main store.

It is essential than an exact multiple of eight characters to fill the main store area is prepared for this instruction. If too few are on the tape the paper tape reader will wait for more characters so that the transfer may be completed. If too many characters are on the tape, only sufficient will be read to fill the area involved. Words are filled in from D0 to D47 (left to right). As with all input/output instructions the contents of the Q-store remain undisturbed.

Example

If a declaration in a program is V9 = Q0/AY6/AY7; and the first instructions are
V9; =Q3; SET2; SET5; OUT; =C3; PRQ3;
The effect would be:

(i) to claim a paper tape reader and place its device number in C3;

(ii) to read 16 characters from paper tape and transfer the 6 information channels of each to fill Y6 and Y7.

If during the read a character is read which does not pass the Parity test an indication is set in the reading device, and the read ceases at that character until the error is dealt with.

PWQq; Causes a block of information to be punched on paper tape by the device whose number is in the counter positions of Q-store q. The information punched is that region of main store specified by the low core address in Iq and the high core address in Mq. In addition to the six information channels the punch will insert the parity and 8th. hole bits as necessary. Because the parity bit is automatically inserted no parity indicator exists in the device. No gap is left when punching ceases. Words are punched from D0 to D47 (left to right).

Example
The instructions to punch the contents of Y5 where the device number is in C12 would be SETAY5;DUP;=I12;=M12;PWQ12; The device must have been claimed earlier in the program for C12 to hold the device number.

PREQq; This is the same as PRQq; except that if an end message symbol is transferred before the specified area of main store is filled the transfer ceases. The → is transferred and if it is not the last character of a word, the characters in the word will be placed "left justified" and padded out with 0's on the right. Other words in the specified area will be left as they were before the transfer.

Since the reading will cease when the specified area is filled (if not previously completed by the transfer of an end message) the programmer should ensure that the specified area and the data should be of such a size that the end message is always read.

Note: Because $75_{(8)}$ is read as the end message symbol it is essential that only data in character form be transferred when using this instruction.

PWEQq; This is the same as PWQq; except that the transfer ends when either the last word of the area or an end message character is punched. See the note at the end of PREQq;

PRCQq; This is a special instruction for reading 8 hole tape in which all 8 channels contain information for the main store. All 8 bits are transferred to the least significant end of the appropriate main store word, the remainder of the word being filled out with zeros. One character from the paper tape therefore occupies one complete word of main store. The transfer ends when the number of main store words specified by the Q-store are filled.

The layout of digits in the main store word is:-

D0-39	Zeros
D40	Channel 8 .... 8th. hole
D41	Channel 5 .... Parity
D42	Channel 7
D43	Channel 6
D44	Channel 4
D45	Channel 3
D46	Channel 2
D47	Channel 1

There is no parity checking with this instruction:
all characters including 'delete' and 'gap' are transferred to the main store.

PRCEQq; As for PRCQq; but stopping if an End Message Character is transferred. For this purpose End Message is taken as any character having the configuration $75_{(8)}$ in the normal information channels 5 and 8 are disregarded.

PWCQq; This is the exactly inverse instruction to PRCQq; the punching of 8th. hole and parity depend on the contents of D40 and D41. D0-D39 are ignored.

PWCEQq; As for PWCQq; but stopping on octal 75.

PGAPQq; This instruction causes a gap of n blank characters to be punched on the tape by the device quoted in Cq. Only the sprocket hole will be punched in each character position. The integer n defines the number of sprocket holes punched and is quoted in Mq, (10 sprocket holes to the inch). The increment position of Qq is not inspected by this instruction and may contain any data.

It is wise to begin and end all tapes from the paper tape punch with a gap of about 12 inches. This ensures that the characters punched are well clear of the punching mechanism when the operator tears the tape off. The form of instruction for this would be (assuming C8 contains the device number)

```
SET120;=M8;PGAPQ8;
```

The parity indicator cannot be set by this instruction.

17·8 Checking facilities on Paper Tape

After either of the instructions PRQq;PREQq; the Parity Indicator on the device used should be inspected to ensure that the transfer was successful. This inspection takes the form of the instruction PARQq; where Cq contains the device number; Iq and Mq are not needed and may contain any data.

The instruction PARQq; causes the Test Register to be set if the Parity Indicator is set. The Parity Indicator is then automatically cleared. The state of the Test Register will not change if it was set before the PARQq; instruction, irrespective of the original state of the Parity Indicator.

This means that by correct use of PARQq; and JrTR; (normally in a pair, as PARQq;JrTR;) the program can, depending on the original state of the Parity Indicator, either continue in sequence or jump to "failure routine".

It is to be emphasised that each device has its own Parity Indicator but that there is only one Test Register in the machine. Also PARQq; clears the former and JrTR; clears the latter.

The full test for parity, then, involves transferring the state of the indicator to the Test Register and then jumping to a reference label if the Test Register is set.

Since the reader cannot be used with the Parity Indicator set it is essential that PARQq; be applied before the next read instruction is reached, but to save computer time, as will be appreciated later, the test should be left until as late as possible. However once PARQq; has been entered, the instruction JrTR; should follow immediately.

We shall now consider the paper tape instructions of the example program in Section 6·2.

The instructions V0;=Q1; cause I1 and M1 to hold addresses of YD0 and YP99 respectively, and SET2;SET5;OUT;=C1; claims a paper tape reader for the program and enters its device number in C1. The instruction PREQ1; causes the data to be read (up to end message) into the area YD0 to YD99. It will be noted that the specification of the program implies that there will never be more than 99 words of data followed by (CN) (CRLF)→; so that no more than 100 words need be reserved for this transfer, the end message will always exist in the area after the read. Since it takes time to actually read the tape there is no reason why processing of data outside the scope of the data read should not continue hence the use of the following instructions up to and including =YP0;

No more work independent of the input data can now be performed so we need to do the PARQ1; to ensure the read was successful. Should the read still be in progress when this instruction is encountered it will wait until the transfer is complete and then perform the parity test. Reference 101 is a failure routine to which the program jumps if PARQ1; has caused the test register to be set.

The instructions C1;SET6;OUT; now deallocate the reader from the program as it will not be required again for reading purposes. (Had a further read been required the deallocating would have been left until after it.) The program then proceeds to process that data which is now in the main store.

17·9 Protective Interlocks and Lockouts

It can often happen that an input/output device can be called upon to perform an operation before it has completed a preceding one. If this happens, an automatic interrupt into Director will occur before the second operation is initiated. Director will return control to the program to perform the second operation as soon as the device ceases to be busy. Thus no harm will come to the program.

It is also very easy for a program to try to transfer information out of or into a main store word whilst an input/output device is referring to an area including the same word. This will similarly cause an automatic interrupt into Director only returning when the main store word is once again available for use. This is achieved by KDF 9 keeping a "lockout store" which notes all words currently involved in input/output transfers, and this is referred to before any transfer from or to the main store is allowed.

Since it would require a large amount of storage to check each word independently, the lockouts only go in steps of 32 words, the bottom five bits of any address not being inspected for lockout purposes. It follows for this that, to avoid unnecessary lockouts, the program should keep the areas involved with input/output operations in separate groups of 32 words. Compiler will always guarantee that the address of Y0 is divisible by 32 exactly, so an address can be checked to see what lockouts are produced. For example, if we ask to read a word to Y93 and also try to transfer Y64 whilst the read is being performed, we will be temporarily locked out, because 64 and 93 both have the same binary configuration, ignoring the bottom 5 digits.

It is now possible to give a brief idea why V23 and V55 etc. are quoted in the heading sheet, and the "gap" arranged to terminate at the end of a line as requested in Section 6•7. If an instruction requires a peripheral transfer of data in the same block of 32 words as the instruction, the block will become locked out and so it will not be possible to perform the next instruction until the restriction is removed. This is known as being "locked out of your own program".

17•10 The On-Line Typewriter

For the purpose of completeness the actual typewriter instructions are given. However, it is strongly recommended that TWQq; and TWEQq; are not used - the corresponding OUT8, using stream number $00_{(8)}$ should be used instead - this is especially so when the program is to be run on a time sharing machine.

17•10•1 Principles of Operation. The on-line typewriter is the only device on KDF 9 which is shared by all programs, including Director, and this fact should be remembered at all times. It is possible for the program to use the typewriter in two successive instructions and yet for Director to use it in between, so that information intended for presentation on the typewriter in two consecutive lines is split up by extraneous material inserted by Director.

As the typewriter operates at only 10 characters per second programmers are advised to restrict their use of it to the absolute minimum. It is suitable only for short messages to the operator.

The on-line typewriter is equipped with a station for reading edge punched cards or paper tape, and also with a station for perforating edge-punched cards or paper tape. Information transferred from the computer to the typewriter will always appear on the typed copy and will also appear in punched form if the punch is switched on. Information may be transferred from the typewriter to the machine either from the manual keys or from paper tape or edge-punched cards via the reading station. It should be remembered however, that a typing error at the keys cannot be corrected. For this reason cards or paper tape are preferable when using the typewriter as an input device, as they can be checked for accuracy beforehand. Whichever of these means is employed, the typed copy is always a complete record of everything that has gone through the typewriter in either direction.

17•10•2 Typewriter Control Instructions. The typewriter input and output instructions are very similar to those for paper tape, so that only a brief explanation need be given here. The instructions are:-

TWQq; Write a fixed-length block from the main store to the typewriter.

TWEQq; Write a variable-length block from the main store to the typewriter.

TRQq; Read a fixed-length block from the typewriter to the main store.

TREQq; Read a variable-length block from the typewriter to the main store.

The Cq in each case must contain zero which is the device number of the typewriter. There is one special facility available when writing to the typewriter. When using either of the instructions TWQq; or TWEQq;, if one of the characters transferred is a semi-colon then writing will immediately stop, and the remainder of the instruction will be treated as a read instruction. This will now be amplified for each instruction in turn.

To use TWQq; an area of main store has to be specified in the Q-store. If a semicolon appears within this area, then as soon as it has been written to the typewriter the transfer will be truncated, and the remainder of the specified block in the main store may be filled with information supplied from the typewriter. This means that the part of the specified area following the semicolon must either be empty or contain redundant information. In particular, any character spaces following the semicolon in the same word MUST be empty, but any succeeding word will be cleared when the first character is transferred into it from the typewriter. In general, therefore, to reduce the organisational problems, the semicolon will be the last character in a word. Care must be taken to see that the block has been precisely filled when all the required information has been read in. If it is attempted to read in too much, the excess information will be lost. If too little, then the machine will wait until the block is properly filled from the typewriter instead of continuing with the program.

The situation is simpler with the instruction TWEQq; . If a semicolon is typed before the end message character is reached, then information may be read into the remainder of the block until an end message character is transferred. In this case it does not matter if the main store region specified in the Q-store is not entirely filled, although, of course, information will still be lost if the attempt is made to read in too much.

It is recommended that this semicolon technique be used only with the instruction TWEQq; . This facility allows the program to ask a question and for the operator to supply the answer via the typewriter all in one instruction, so that no interference from any other program can occur. The question and its answer will appear on the same line of the typed copy. This facility is used extensively by Director and may also be used by programmers with profit, but use of the Typewriter should be kept to a minimum due to its slow operating speed.

Programmers are asked to begin any transfer to the typewriter (except those via OUT8) with carriage return-line feed and case normal characters.

Use of the TAB character is reserved for Director so that on the log sheet the comments from Director and the control program will appear in their appropriate columns.

There is no parity checking available on the typewriter.

Example
A program inspects the contents of Y7 and performs a certain routine depending on whether the contents are YES.→ or NO.→ which is the operator's typewriter reply to the question, "Is there a spare work tape available?" The instructions necessary to arrange for Y7 to be set up are:-

```
V1      =  Q 0/AY5/AY7;
V2/3    =  P SPARE*WORK*TAPE [Q];
V1;=Q9; V2;=Y5; V3;=Y6; TWEQ9;
```

17·11 The High Speed Printer

17·11·1 Mode of Operation on-line. The high speed printer for KDF 9 is essentially a device for printing LINES of information at a speed of about 1000 lines per minute, where the maximum size of a line is 160 characters.

When a printer instruction is initiated, binary digits from the main store area involved are transferred to the printer's control electronics box in groups of 6 (i.e., an octal pair) at a time. These are inspected and if they agree with the "Printer Code" in Section 17·11·2 a marker is set in a matrix. This matrix has 160 columns and a row for each character. For the jth octal pair inspected, the marker is placed in the position defined by the jth column and the row determined by the octal pair.

Should an octal pair not fit into the code it will be completely ignored. The matrix is filled up (from the first column) until an octal pair, being one of the CONTROL SYMBOLS is encountered. Two such control symbols are generally used for the printer:-

(a) Line Shift 02$_{(8)}$, to advance the paper by one line only.

(b) Page Change, 03$_{(8)}$, to advance the paper to the first line of the next page.

The control symbol causes

(i) the current contents of the matrix to be printed,

(ii) the matrix to be cleared, and

(iii) the paper to be moved into position ready for the next print instruction.

It is to be noticed that no print takes place until a control symbol is reached; which implies that the last octal pair of any transfer must be a control symbol. This explains why in OUT8 when using stream 30-37 the last character of the block must be either $02_{(8)}$ or $03_{(8)}$.

The state of the matrix after a word holding 50 45 41 44 50 51 47 $50_{(8)}$ has been transferred to it is as follows.

Octal			1	2	3	4	5	6	7	8	9	10	11	12	13	-	-	160
41	=	A			1													
42		B																
43		C																
44		D				1												
45		E		1														
46		F																
47		G							1									
50		H	1				1			1								
51		I						1										
52		J																
'		'																
'		'																

If the first character of the next word which is also transferred, is $02_{(8)}$ or $03_{(8)}$ then the method of printing from the matrix is

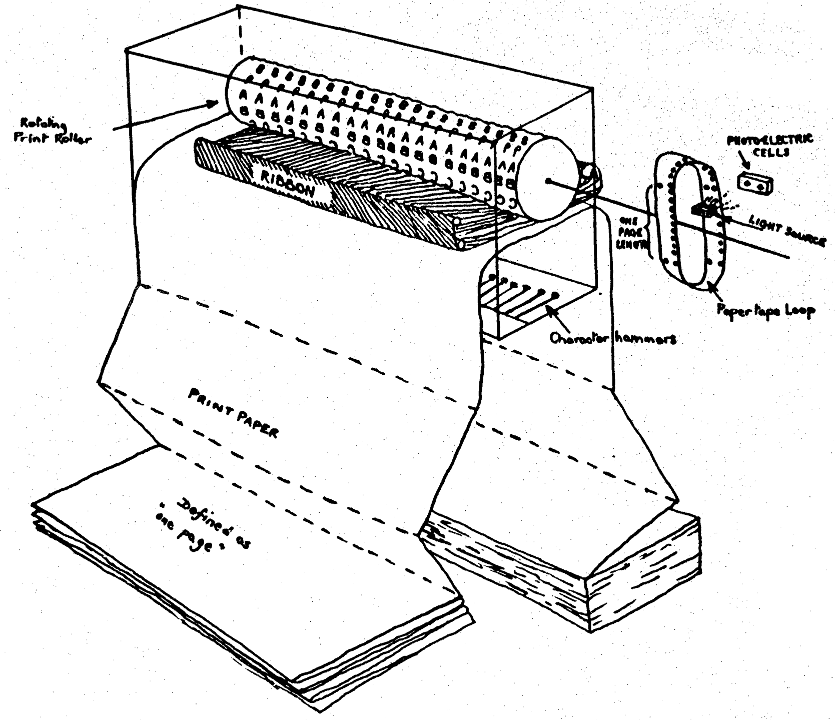
(i) all the A's are printed, in this case only one A in the 3rd character position from the left of the paper, and the marker cleared.

(ii) then the B's, C's, etc., as for the A's

(iii) H's would be printed in the 1st, 5th, and 8th positions

(iv) When all the markers are cleared from the matrix, then the paper moves either to the next line or the beginning of the next page.

The mechanism of printing is that the character hammers agreeing with the matrix column number strike the paper when the rotating print roller has the appropriate character in position.



The mechanism of paper motion is such that after printing the control symbol causes the paper to start moving, and halting control is transferred to a photo electric cell assembly. On the side of the printer is a set of rollers holding a loop of paper tape in which is punched a hole for every line position of the print paper and an extra hole for the position of the beginning of every page. The paper tape loop moves synchronously with the print paper.

If the control symbol initiating paper motion had been $02_{(8)}$ (line shift) the motion would stop when light reaches the appropriate photo electric cell through a hole in the paper tape loop. Had the control symbol been $03_{(8)}$ (page change) print paper motion would stop when light reaches the two appropriate cells. To ensure that printing will not continue right up to the perforation between two pages there is no line shift hole punched in the loop for six line positions either side of the position where a double (page change) punching is made. This causes an automatic page change if the printing is within six lines of the bottom of the page, even though the control symbol had only been for a line shift.

17·11·2 The KDF 9 Printer Code. For the purposes of this chapter the reader should only refer to the column "on-line".

Octal pair	Printer		Octal pair	Printer	
	On-line	Off-line		On-line	Off-line
00	Space	Space	40	Not Used	Selection
01	Not Used	Selection	41	A	A
02	LS	LS	42	B	B
03	PC	PC	43	C	C
04	Not Used	Horizontal Tab	44	D	D
05	Not Used	Selection	45	E	E
06	%	%	46	F	F
07	'	'	47	G	G
10	:	:	50	H	H
11	=	=	51	I	I
12	(	(	52	J	J
13	)	)	53	K	K
14	£	£	54	L	L
15	*	*	55	M	M
16	,	,	56	N	N
17	/	/	57	O	O
20	0	0	60	P	P
21	1	1	61	Q	Q
22	2	2	62	R	R
23	3	3	63	S	S
24	4	4	64	T	T
25	5	5	65	U	U
26	6	6	66	V	V
27	7	7	67	W	W
30	8	8	70	X	X
31	9	9	71	Y	Y
32	Not Used	Selection	72	Z	Z
33	$_{10}$	$_{10}$	73	Not Used	End File
34	;	;	74	Not Used	End Data
35	+	+	75	End Message	End Message
36	-(minus)	-(minus)	76	Start Message	Start Message
37	°	°		Ignored	Ignored

17·12 High Speed Line Printer Instructions

The printer is a type 3 device and must be claimed by the program before it can be used for printing. The claiming would take the form

```
SET3;SET5;OUT;=C1
```

The instruction to initiate a printer operation is LPQq; with the printer device number in the counter position, and the low core and high core addresses of the area of main store to be transferred in the increment and modifier positions respectively. As with most input/output instructions the Q-store and main store are not changed after the transfer.

A test of Parity is necessary before the next LPQq; is attempted.

After the last LPQq; and its PARQq; JrTR; the device must be deallocated by instructions of the form

C1;SET6;OUT;

Example
If Y20 contains

1421240021316300$_{(8)}$

and Y21, 77 77 77 77 77 77 77 02$_{(8)}$

and Q5 Device No. of LP/AY20/AY21:
then the instruction LPQ5; would cause

£14 19s

to be printed.

Should it be required to print other than from the left margin of the paper it will be necessary to ensure that sufficient "spaces" 00$_{(8)}$, precede the printable characters.

In each print operation a whole number of 8 character words are transferred, where by 'character' is meant ANY octal pair. Should the area to be printed contain redundant character positions, they should be padded out with dummies, 77$_{(8)}$. This together with the requirement that 02(or 03) should be the last character of the block explains why Vv = P [7DC]; is found as a V-store declaration in almost every program.

Generally, line printers are set and loaded with paper allowing only lines of 120 characters width, and if possible programs should be written accordingly. If, however, it is essential that the program be so written that 160 character wide paper be used, a message to this effect should be passed to the operator when the program is run. Similarly paper of only 80 characters wide is available but not generally used at all installations.

The instruction LPQq; basically only allows transfers of fixed length but this can be varied to allow a transfer to end message only by variation to the standard plug board - (see Section 12·5). For a more detailed explanation of the Line Printer the reader is referred to "1040 High Speed Line Printer Manual".

17·13 Exercise - SET 1 (Do not use OUT8 for these questions)

1. Write the necessary constant declarations and instructions to claim the paper tape reader; read a block of data to end message into YD0 - YD1000; read a second block fixed length (no end message) into YP53-YP57; and then deallocate the reader. Parity tests must be performed and in the event of Parity failure the message PARITY P.T.R. is to be typed on the typewriter, and the routine abandoned.

2. Claim a line printer and call for three page changes (the pages to be left blank). Eventually deallocate the device. In the event of Parity punch, on paper tape, characters such that when later fed through a typewriter it would type XxYy. Precede and end with 6" of blank tape.

3. Claim a line printer; call for a page change; print out the contents of YR7 (which are in printer character code) on the first line of that page; leave a blank line and then print out the same data again. Eventually deallocate. Arrange for a suitable message to be typed on the typewriter in the event of Parity. The contents of YR8 must not be destroyed.

4. A block of data is in YD5-YD9 in paper tape character code; the last character in each word is $75_{(8)}$. Do all that is necessary to cause the contents of the area to be punched on paper tape with a gap of 4 inches before each group of 8 characters punched and 6 inches at the end.

5. Arrange for the question "Is it Monday" to be typed on the on-line typewriter allowing for an operator reply Y. → or N. →

6. Claim the magnetic tape with identifier +KESUBLB.50-1470 and puts its device number in Q1.

Exercise - SET 2

1. A paper tape contains the characters
07 77 77 77 00 32 53 77 32 44 32 46 00 32 31 77 $02_{(8)}$
which are then read into Y1 and Y2. The instruction LPQ5; is then obeyed with Q5 holding Device No. of L.P/AY1/AY2. What is printed on the line printer and what would be typed if the paper tape were fed through an off-line typewriter?

2. A program contains the following declaration and instructions; what is printed on the line printer?

```
                  V0 = Q 0/AYP3/AYP3;
V0;=Q1;SET3;DUP;SET2;+;OUT;DUP;=C1;=YD3;
SET-1;SETB3675;-;=YP3;LPQ3;PARQ3;J1TR;........
```

3. Write a program body to:-
Read 8 words from paper tape (these contain only decimal digits and alphabetic characters).

Via OUT8 Stream $36_{(8)}$ print heading (Name) at the top of a new page, leave 2 lines and then print the 8 words read in, at 2 words per alternate line with reasonable spacing between each word.

In the event of any parity failure, type the word "Parity" on the on-line typewriter via OUT8 and terminate the program.

18·1 The NEXT Facility

Section 4·1 indicates that two words of KDF 9 may be assumed to be the two halves of a double-length word of 96 bits. It is a wise precaution that these two words should be in juxtaposition when in the main store with the lower addressed word containing the more significant half. Any operation in the Arithmetic Unit involving double length working requires the more significant half to be in N1 and the lower half in N2 (or in N3 and N4 respectively).

When fetching and storing double-length numbers to and from the nesting store by Direct (modified or unmodified) addressing no complication arises provided it is remembered that in fetching to the nest the lower half is brought BEFORE the upper half and that in storing to main store the upper half goes before the lower half.

Difficulty could arise in dealing with such numbers if access to them is by Indirect (modified) addressing. To overcome the problem a special facility has been provided in User code in which the more significant half of a double-length number is indirectly addressed in the ordinary way and the same instruction written followed by an N (before the ;) to provide easy access to the NEXT word being the lower half. The N has the effect of increasing the calculated address by 1.

To illustrate with an example, suppose that

Q4 contains ?/?/base address

Q5 contains n/2/0.

Then to fetch into the nesting store the double-length number whose more significant word is stored at the address given in M4 and whose lower half is stored at the next main store address, the two instructions, assuming the Q updating facility is required, would be:-

M4M5N; M4M5Q;

The two corresponding instructions for storing from the nesting store would be:

=M4M5; = M4M5QN;

Note the difference in the positioning of Q and N in these two sets of instructions, and that I5 requires to be set at 2 if the Q updating facility is used. Because the Q was included in both sets of instructions, Q5 has now been updated ready to fetch or store the next double-length number in sequence.

It must also be emphasised that the NEXT facility cannot be used with direct addressing: Y0M6N; is not valid, Y1M6 would have to be used.

18·2 Half-Length Fetch and Store

Half-length numbers in KDF 9 cannot be transferred between the main store and the nesting store using the direct forms of the fetch and store instructions. This is only possible using the indirect forms together with the facilities about to be described.

Half-length numbers are used to economise in main store space if half or less of the full 14 decimal digit precision is sufficient for the problem in hand, so that two such numbers may be stored in one main store word. Since it is not worthwhile to build half-length arithmetic facilities into the machine, and since the nesting store in consequence is capable of holding only full-length words, a half-length number must be expanded to full-length form if it is fetched to the nesting store, and a full-length number must be contracted to half-length form if it is transferred from the nesting store into half of a main store word.

The way in which this is done is very simple. The half-length fetch and store instructions use the label H to distinguish them fron the standard forms of fetch and store.

The instructions are:

MkMqH;	Half-length fetch.
=MkMqH;	Half-length store.

A half-length fetch instruction selects the required 24 bits from the main store word and puts them into the **more significant half** of the top cell N1 of the nesting store, the remainder of N1 (D24 - 47) being filled out with zeros. A half-length store instruction selects **the top 24 bits of N1** (D0 - D23) without rounding off, and stores them in the specified half-word in the main store, finally erasing the whole of N1 in the usual way.

When a half-length fetch or store instruction is obeyed the transfer address is calculated as follows:

A copy of the integer in Mq is divided by 2 and the integer quotient added to a copy of the integer in Mk. The result determines the E address of the word to be accessed. When Mq was divided by 2, if the remainder were 0 then the upper half of the E address would be accessed, if the remainder were 1 then the lower half would be accessed.

N.B. The contents of Mq must always be positive in this context.

If now the reader refers to the latter half of section 7·5 the reason for calculating half-length addresses in the manner shown will be clear.

In neither case of a half-length store is the other half of the main store word disturbed but the whole of the top cell of the nesting store is erased.

18·3 Q, H and N Facilities

The three facilities Q for Q store updating
H for Halflength addressing
N for Next word addressing

may be used singly, in pairs or all together as the need arises, but when so used they must be in the order Q H N.

Points to note are:

(a) even if Q is used with H and for N it still performs the operation explained previously - viz. a copy of the increment is added to the modifier and the counter decreases by 1 AFTER the main store has been accessed.

(b) the Q refers to the Q store q; the Q store k does not alter.

(c) Half-length fetches and stores are to and from the top 24 bits of N1 in the nesting store - during fetches the bottom 24 bits of N1 are made zero and during stores they are erased with the top 24 bits.

(d) when H and N are used together the N still performs the previously explained operation - viz. ONE WHOLE WORD is added to the address calculated without the N.

(e) If a consecutive sequence of half-lengthwords are accessed using QH the increment of Qq must be 1.

(f) If a consecutive sequence of upper (or lower) halflength words are accessed using QH the increment of Qq must be 2.

(g) When working double-length, integers should normally be held to 94 integral places.

(h) When working half length, integers should normally be held to 23 integral places.

18·4 Examples

1. Y0, Y1; Y2, Y3; .... ; Y90, Y91; contain 46 double length words in pairs. Write a loop to place copies of the more significant words of the pairs into YN0 - YN45 and the lower words in YL0 - YL46.

```
V0  =  Q46/2/0;
V1  =  QØ/1/Ø;
V0;=Q1;V1;=Q2;
1;   Y1M1;Y0M1Q;=YN0M2;=YL0M2Q;J1C1NZ;
```

(N.B. The NEXT facility is **not** used with DIRECT addressing).

2. Repeat the above example with the variation that the 46 double length words are in the 92 storage locations commencing at that address which is in M4.

19·1 Multiplication

19·1·1 Theory of Multiplication Since KDF 9 has a fixed word-length of 48 bits, the system of multiplication used is also fixed length.

The rules are precisely the same as for decimal multiplication (except of course, that binary is used in place of decimal) but it should be remembered that decimal multiplication, as it is commonly understood, is generally not performed fixed-length.

Consider two examples:

99	15
97	13
8910	150
693	45
9603	195

These two multiplications have been carried out in the commonly accepted manner, but it should be noted that in the case of the first a four digit answer has resulted, whereas the second has provided only a three-digit answer. This is because the workings were not performed in fixed-length. To calculate the examples in fixed-length the procedure is as follows:

99		15	
97		13	
8910	record 0; carry 0	0150	record 0; carry 0
	9×9=81;+0;=81;record 1; carry 8		1×5=5;+0;=5
	9×9=81;=89;record 9; carry 8		record 5; carry 0
	No more digits so record the carry		
693	7×9=63;record 3;carry 6		1×1=1;+0;=1
9603	7×9=63;+6=69; record 9; carry 6		record 1; carry 0
	no more digits so record the carry		no more digits so record this carry
		045	3×5=15; record 5; carry 1
			3×1=3;+1=4; record 4 carry 0
			No more digits so record carry
		0195	

A closer look at these examples reveals several rules of mult'plications, which apply irrespective of the scale used (i.e., decimal, binary, octal, etc.)

Rule 1. If two similarly fixed length numbers are multiplied together, the result has twice the number of digits of the original length (i.e., it becomes double length). In our example, 2 numbers each of 2 digits generate a 4 digit result.

Rule 2. The number of **integral** places in the result is always equal to the sum of the number of **integral** places of the two operands. This can be verified by inserting decimal points in the examples.

99·×97 = 9603 2+2 gives 4 integral places
9·9×97 = 9·603 1+0 gives 1 integral place

15·×13· = 0195· 2+2 gives 4 integral places
·15×·13 = ·0195 0+0 gives 0 integral places

Rule 3. If a single-length result is required half the digits in the product will be lost in changing from double to single-length.

In general all digits are significant (because a good programmer sees to this to reduce errors as far as possible) and therefore if contraction of the answer from double to single is required, the least significant digits are the ones to be removed, those retained being rounded off as necessary. The rounding off rule is simple - if the digits removed are less than half of one unit in the least significant digit position of the most significant half, no rounding occurs.

Otherwise, one unit is added to the part kept.

9·9×·97 = 9·603 ·003 is less than ·05 therefore no rounding. Result = 9·6
·15×·13 = ·0195 ·0095 is not less than ·005 therefore round off. Result = ·02

KDF 9 will give results like this if required.

Note that the number of integral places is not changed by this rounding and truncation.

In calculations involving integers only, however the result will be single-length (in general) and will also be an integer. In this case, the more significant half is the half to be removed. Note that this procedure reduces the number of integral places.

e.g., 15×03 gives 0045 (4 integral places) removing the more significant half gives 45 (now with 2 integral places).

19·1·2 Multiplication on KDF 9 The basic principle in multiplication on KDF 9 is that if a number A held to p integral places is multiplied by B held to q integral places the result is A×B held to (p+q) integral places.

The instructions needed in multiplication are:-

XD; Multiply, giving double-length result in N1, N2.

Multiplies A (to p integral places) in N2 by B (to q integral places) in N1 to give the result A×B held to (p+q) integral places in N1+N2 (N1 more significant than N2). The original single length numbers in N1 and N2 are erased.

The Overflow register can be set only if the original numbers are both negative and of maximum size.

×; Multiply giving single-length rounded-off result in N1.

As for XD but then removes the contents of the **less significant** half of the answer in N2 and rounds N1 if necessary. The result in N1 is still held to (p+q) integral places.

The original single-length numbers in N1 and N2 are erased.

Overflow is set as for XD;

CONT; An abbreviation for contract. Takes a double-length number in N1 and N2 and replaces it by a single-length number obtained by removing the **MORE** significant half.

The result has 47 integral places less than the original double-length number. Overflow is set if the more significant half was **NOT** all zeros for all ones - this indicates that the number is too large to be held in a single length register

This instruction is used in multiplying small integers in the sequence XD; CONT; and gives the product A×B held to (p+q - 47) integral places.

19·2 Division

19·2·1 Theory of Division As in multiplication, the operands and the result of division in the machine are fixed-length; it is therefore simple to explain the theory of division if we start without knowledge of fixed length multiplication.

In fixed-length working we know that A (held to p integral places) × B (to q integral places) is AB (to p+q integral places).

It is therefore clear that

AB (to p+q integral places) divided by B (to q integral places) is A (to p integral places), or expressed in another way:

A (to r integral places) divided by B (to s integral places) is A/B (to r-s integral places).

The reader will now see why in Section 3·5·1 it was stated that the computer considers the point to be between D0 & D1. In multiplication 0+0 integral places still leaves the point between D0 & D1, in division 0-0 integral places still leaves it in the same place.

Caution! this last statement about division requires a little more thought.

Consider $\cdot 01_{(2)} \div \cdot 001_{(2)} = (\cdot 25_{(10)} \div \cdot 125_{(10)})$

the answer of which is $10_{(2)} = (2_{(10)})$

But it is NOT possible to hold $10_{(2)}$ to 0 integral places - what has gone wrong? The numerator was larger than the denominator. Refer back to the multiplication A (to p) × B(to q) =AB (to p-q).

Because the **computer considers** both multiplicands to be held to 0 integral places A and B were considered factional so that A×B must be less than either A or B. This implies in division, so far as the computer is concerned (with the point being between D0&D1), that the numerator must be numerically less than the denominator.

This restriction is no. so serious as it may first appear.

The programmer does not consider the numerator and denominator to be held to zero integral places; all he needs to worry about is "can the result of the division be held to the number of places given":- if the answer can be so held then the numerator (as considered by the computer) must have been numerically less than the denominator.

Consider 10^5 (to 17 integral places) $\div 10^2$ (to 7 places)

the result is $\frac{10^5}{10^2}$ (to 17-7 integral places)

$= 10^3$ (to 10 places)

Let us see what these numbers look like in the machine.

```
       DØ  D1                                     D17 •
10^5   [0  1 1 000   011    010    100    000   ↓   000.........
                             D7•
10^2   [0  1 1 001   00↓0   000    000.........
                                  D10•
10^3   [0  1 1 111   010    00↓0   000.........
```

Although in fact 10^5 is greater than 10^2, the **pattern** in the word looks smaller to the computer so that a valid result is possible.

To summarise: Provided the result can be held to the calculated number of integral places, division will proceed correctly.

The question may now be asked: "Suppose the quotient can not be held to the calculated number of integral places?"

When a program is being written the maximum absolute magnitude of the numerator is known, as is the minimum absolute magnitude of the denominator. Dividing these will give the largest possible absolute quotient and hence the maximum number of integral places necessary for it. The programmer then only needs to ensure that the numerator and denominator are held so that

$\frac{A}{B}$ (p) leads to $\frac{A}{B}$ (p-q) where (p-q) is the maximum value necessary to hold all quotients.

19·2·2 **Division on KDF 9** The actual division instructions on KDF 9 are:-

÷; Divides the single length numerator in N2 (held to p integral places) by the single length denominator in N1 (held to q integral places) and gives the rounded single length result in N1 (held to p-q integral places). The original contents of N1 and N2 are erased. Overflow is set if the denominator is 0, or if the result requires more than p-q integral places (i.e. if the pattern in N2 "looked" larger to the machine than that in N1).

÷D; Divides the **double** length numerator in N2 and N3 (held to p integral places) by the **single** length denominator in N1 (held to q integral places) and gives a single-length rounded result in N1 (held to p-q integral places). The original contents of N1, 2, 3 are erased. Overflow is set if N1=0 or if result requires more than p-q integral places.

When working with integers (single-length) the values of p and q will be 47 from which it may be thought that such a problem as 32·5 would not be possible on the machine because 6·4 requires 3 integral places at least. This difficulty is easily overcome but requires a knowledge of "shifting" which is dealt with later.

÷I; Divides a **single**-length INTEGER A (47 integral places) in N2 by a **single**-length INTEGER B in N1, giving an INTEGER quotient in N2 and a remainder in N1. **The remainder will be of the same sign as the denominator and of smaller magnitude.**
e.g., (+16)÷I(-3) gives -6 as quotient
-2 as remainder.

Overflow is set if B=0.

N.B. No shifting of operands is necessary with ÷I; the method used ensures that the result is always valid unless B=0. This instruction uses an extra cell of the nesting store for the calculation which the programmer must allow for.

19·3 Examples

1. Find the product of the two fixed point integers in Y6 and Y7 and place the result, held to 47 integral places, into Y8.

```
Y6;Y7;×D;CONT;J16V;=Y8;
```

Y6 and Y7 have p=47, therefore the product will have p = 47+47 = 94. Assume Y6 holds 2 and Y7 holds 3, the product is 6, which held to 94 integral places is 91 zeros followed by 110. This means that double length multiplication is required. Had the instruction ×; been used, only the 47 most significant bits (ignoring sign digit) would have been retained, thus giving a zero result.

XD; has given us a double length result, but it may be that the integer held to 94 places is so small that the more significant half is all zeros (for positive numbers) or all ones (for negative numbers). Let us then follow the multiplication instruction with CONT; which will reduce the result by 47 integral places. The purpose of the instruction J16V; is to cause the routine at referance 16 to be followed if the contraction caused the overflow register to be set.

2. Y43 contains a fixed point number held to 4 integral places and Y44 another one held to 13 integral places. Place their product in Y45, stating the value of p for Y45. The precision essential at this stage of the imaginary program is at least 20 binary fractional places.

```
Y43;Y44;×;=Y45;(Y45 now has p of 17);
```

The reason for the statement about precision is made clear by the following example.

3. Y100 and Y101 contain fixed point numbers held to 23 integral places. Obtain their product. Before we know whether to use ×D; only (and so hold the answer double length to 46 integral places), or ×D;CONT; or
×;
We need to know what is required from the result.

If only the integral part of the answer is significant to the problem then ×; would be sufficient. The same applies if the result is necessary only to the nearest $\frac{1}{2}$. The answer would of course be held to 46 integral places.

If the whole of the answer must be retained, if it is known from the nature of the problem being programmed that the product will always be of less magnitude than $\frac{1}{2}$ the instruction ×D;CONT; would suffice. In this case the result in N1 would be held to -1 integral places. The contraction is possible because the more significant 47 bits (ignoring sign digits) will never contain any significant information.

However, if the whole answer must be retained and if the result could have greater magnitude than ½ the only possible instruction is ×D;

It may be possible from the nature of the problem to "shift" (and so alter the number of integral places of) the result after the appropriate multiplication instruction, but this is beyond the scope of instructions dealt with so far,

4. YP8, YP9 contain two integers (p=47).
Place the integral quotient of (YP8)+(YP9) in YR1 and the integral remainder in YR2.

```
YP8;YP9;+I;=YR2;YR1;
```

5. YR3 contains a fixed point number to 18 integral places and YR4 another to 15 integral places. The numbers are such that the quotient (YR3)+(YR4) will always be in the range 1⩾Q⩾-1.
Place this quotient in YS2, stating the number of integral places it is being held to.

```
YR3;YR4;+;=YS2;(YS2 has p of 3);
```

6. YN9&10 contain a double length fixed point number to 49 integral places and YD11 a single length fixed point number to 2 integral places.
The integral part (rounded) only is required, and this is to be placed in Y8.

```
YN10;YN9;YD11;+D;=Y8;
```

Had it also been required to keep as much of the fractional part as possible it would have been necessary to determine the maximum magnitude of the quotient possible and, before the instruction YD11; to "shift" the double length numerator an appropriate number of places.

19·4 Exercises - SET 1

1. Find the double length sum of the squares of the single-length integers in Y0-Y63.
Leave your answer in N1, N2.

2. Two half length integers are stored in Y0.
Find their product as a single length number in N1, and state the number of integral places in your answer.

3. Y0 contains a decimal integer as 8 numeric characters representing a number of pounds. Convert the contents of Y0 to single length binary pence.

4. There are 64 single length binary integers stored from Y0 onwards. Find their average to the nearest integer, leaving your answer in N1.
HINT: The sum may exceed single length.

5. N1 contains a number of binary pence.
Convert this to £.s.d. using +I; and store the pounds in Y0, the shillings in Y1, and the pence in Y2.

6. If N1 and N2 contain fixed point numbers specified below and the instruction ÷; is obeyed. Which divisions will give a valid answer?

a)	N1	contains	19	to	5	integral	places.
	N2	"	128	"	8	"	"
b)	N1	"	17·785	"	5	"	"
	N2	"	19	"	5	"	"
c)	N1	"	2^{12}	"	13	"	"
	N2	"	2^{10}		11	"	"
d)	N1	"	-1	"	47	"	"
	N2	"	1	"	47	"	"

Exercise - SET 2

1. What will be the output from the following program body assuming it is compiled on the POST system?

2. How would you have written the constant declarations V0 to V7?

3. What is the effect of the asterisks before 1; and J1C1NZ;?

```
V0      =       -13
V1      =       B7777777777777775;
V2      =       P ⌊7D⌋S;
V3      =       B3;
V4      =       B15;
V5      =       P ⌊7DEM⌋;
V6      =       6•5/46;
V7      =       P ⌊7SP⌋;
V8      =       Q 4/2/AV0;
V9      =       P ⌊7DC⌋;
V10     =       Q 2/1/AYP4;
V11     =       Q 4/1/AYP1;
V12     =       Q 0/AYP0/AYP6;

V8;=Q1;V9;=YP6;ZERO;=YP3;

*1;     M0M1;M0M1QN;÷I;*J1C1NZ;
        V10;=Q1;V11;=Q2;
 2;     DUP;J3<Z;SETB35;
 4;     =M0M1; JSL1000; ERASE;=M0M1N;M1T0Q3;M2TOQ1;M3TOQ2;
        DC1;J2C1NZ;
        SETB30;=YP0;V12;SET8;OUT;DC2;SET2;=C1;J2C2NZ;
        ZERO;OUT;
 3;     NEG;SETB36;J4;
        library L1000;
        FINISH;
        →
```

20· FURTHER ARITHMETIC INSTRUCTIONS

20·1 Shift Instructions

20·1·1 General Rules for Shift Instructions KDF 9 has a variety of shift instructions designed for use in various circumstances. All operate by taking a pattern of digits (either single-or double-length) and moving them either to the left or to the right.

If the pattern represents a fixed point number, shifting it one place to the left in a binary register can be interpreted to have one of two effects:-

(a) to multiply the value by two, without changing the number of integral places,

(b) to reduce the number of integral places by one, but leave the value unchanged.

Consider this example in an 8-bit register.

00110100 represents $3\frac{1}{4}$ to 3 integral places (not counting the sign at the top end).

01101000 represents either $6\frac{1}{2}$ to 3 integral places or $3\frac{1}{4}$ to 2 integral places.

Since a shift to the left can be interpreted as increasing the value of the number by a factor of 2 for each place shifted, a shift of n places to the left increases the value by 2^{+n}. Similarly a shift to the right "increases" the value by 2^{-n}, which being a fraction, actually decreases the value. KDF 9 will interpret a shift in accordance with its sign; a positive value shifts to the left and a negative value shifts to the right, so to speak of a shift of "minus five" implies shifting the word 5 places to the right, sometimes referred to as "shift down 5".

It is further necessary to arrange a shift instruction in which the amount and/or direction of the shift can be varied as the program is operating, depending on data or conditions.. To allow for this possibility, KDF 9 offers two methods of specifying the amount of shift:-

(a) by inserting the required fixed amount as a signed number included within the instruction - for this case the amount of the shift must be between -64 and +63,

(b) by directing the shift instruction to look at a designated Q store, and shift the amount given in the Counter location of that Q store. The amount of shift in this case is limited to the range -128 to +127, any attempt to go outside this range producing incorrect results with no warning.

N.B. In either case a shift of zero places is allowed and will leave the operand completely unchanged.

20·1·2 Arithmetic Shifts Arithmetic Shifts are designed to deal with NUMBERS only, and therefore need to recognise the presence of a sign digit, preserving the sign during shift down and setting overflow if the register capacity is exceeded during shift up Rounding off is also performed during shift down of single-length numbers but not for double-length numbers Any vacant digit positions created during shift up are filled with zero digits. The available instructions are:-

SHA±n; Shift the NUMBER in N1 an amount ± n. Set overflow if register capacity is exceeded.

SHACq; Shift the NUMBER in N1 an amount given by the counter of Qq. Set overflow if the register capacity is exceeded.

SHAD±n; Shift the NUMBER in N1, N2 an amount ± n. Set overflow if register capacity is exceeded. Remember that the D0 digit of N2 is **not** part of a double-length number - digit D1 of N2 comes immediately below D47 of N1 in order of significance and digits are shifted accordingly, by-passing the D0 digit of N2. For example, SHAD-47; will shift the word in N1 completely into N2, leaving N1 as 48 copies of the sign digit of the original number, with a zero digit in D0 of N2.

SHADCq; Shift the NUMBER in N1, N2 an amount given by the counter position of Qq. Other rules as for SHAD±n;

20·1·3 Logical Shifts A logical shift is designed to operate on PATTERNS of digits. There is no provision for rounding off, overflow, preservation of signs, or, indeed, recognition of the existence of sign digits. Any word is presumed to contain 48 digits of all equal importance. Any digits shifted off either end of the register are lost without trace; any vacant space produced by the shift is filled out with zero digits. For double-length logical shifts, the register is presumed to have 96 bits all of equal significance, with D0 of N2 coming immediately below D47 of N1 in the order of significance.

The logical shift instructions are:-

SHL±n; Shift the PATTERN in N1 an amount ± n.

SHLCq; Shift the PATTERN in N1 an amount given by the counter of Qq.

SHLD±n; Shift the PATTERN in N1, N2 an amount ± n.

N.B. SHLD-48; will shift the pattern from N1 into N2 leaving N1 set as all zeros. This should be compared with the example for SHAD-47; above.

SHLDCq; Shift the PATTERN in N1, N2 an amount given by the counter of Qq.

20·1·4 Cyclic Shifts

20·1·4 Cyclic Shifts A cyclic shift (which is allowed only single-length) will move digits in N1 in a cyclic manner any digit spilling off one end of the register will reappear to fill the space generated at the other end. The amount of shift is limited to the range -48 to +48; a shift outside this range will give incorrect results, but in any case is illogical for a cyclic shift.

The two instructions involved are:-

SHC±n; Shift the PATTERN in N1 an amount ± n in a cyclic manner.

SHCCq; Shift the PATTERN in N1 an amoutn given by the counter of Qq, in a cyclic manner.

20·2 Fixed-Point Accumulative Multiplication

It is often required to form a sum of products (i.e., to evaluate a formula of the kind a.b + c.d + e.f + ...). If this is done and the data are kept to a minimum number of integral places, the sum will (in the worst case) exceed capacity on the first addition and, therefore, will require a shift down to remain within capacity. (A shift of n is suitable for m additions if $m < 2^n$ - this can easily be verified by taking an example).

The set of instructions xD; SHAD-n; +D; would form the basis of a loop to perform this operation. This takes 4 syllables of instructions - KDF 9 provides a single two-syllable instruction to perform the same operations (performed effectively by obeying the three instructions above in a single sequence). Such a reduction in space can often prove valuable, as will be seen in a later section.

The instructions involved are written:-

X+±n; Take two single-length numbers in N1 and N2, multiply them together to form a double-length product, shift this product ± n places (n = 0 is allowed, in which case the instruction would be written x+;) and then add the shifted product to the double-length sum previously stored in N3, N4. Set overflow if final result or any intermediate result exceeds capacity.

X+Cq; As above, but the amount of shift is given in the counter position of Qq.

20·3 Multiple-length Division

A lesser used arithmetic instruction is:

+R This is designed for the case where an n-length number (i.e., a number stretching over n words) is to be divided by a single-length number to give an (n-1) length quotient. The process is to divide the top two word of the denominator, to give the top word of the quotient, and leave a remainder which can be combined with the third word of the numerator ready for the next stage of division. This instruction leaves the remainder in exactly the required form for this operation.

The rules are:-

Divide the double-length number in N2, 3 by the single-length number in N1, leaving the single-length quotient in N1 (just as for + D) and the remainder in N2. The value of the remainder will satisfy the following conditions, where a is the denominator:

(a) if $a > 0$ then $0 \leq r.2^{-(p-47)} < a.2^{-q}$

(b) if $a < 0$ then $a.2^{-q} \leq r.2^{-(p-47)} < 0.$

20·4·1 Floating Point Section 3·5·1 showed how any number can be expressed as

$$n = (-s+f)\times 2^{p}$$

and that when working in fixed point the main store word only contains the s and f, with the p being recorded in the programmer's workbook.

The reader, by now, will have realised the tedium of having to perform certain arithmetic operations entailing the manipulation of the integral places of the operands. This is overcome by the use of "floating point".

Section 3·7 showed that in "floating point" the value of p is recorded in the digit positions D1 to D8. This means that f is contained to a precision of only 39 binary digits, as opposed to 47 in fixed point. This restriction in general causes no hardship. In fact the use of floating point has the advantage that the value of p can range from -128 to +127 thus allowing numbers from ($\pm$) 2^{-128} to $\pm 2^{+127}$ without worry of overflow.

Because "p" is automatically dealt with in floating point the method is easier to program, but in general, arithmetic operations in in floating point take about 2½ times as long to perform as their fixed point counterpart. This means that programs which are only to be run a few times and are of a scientific nature are written in floating point whereas those to be run often, usually of a commercial nature, work in fixed point.

When the precision of 39 bits for f is not sufficient double-length floating point is used. In this case

(a) the sign digit of the number is D0 of the more significant word.

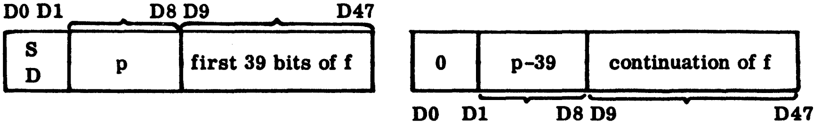
(b) the fraction f has 78 bits, the more significant 39 bits being in D9-D47 of the first word as before and the less significant 39 bits following on in D9-D47 of the second word.

(c) a form of the value of p is, as in single length, placed in D1-D8 of the first word.

(d) D0 of the second word is ALWAYS zero.

(e) D1-D8 of the second word is the same as D1-D8 of the first word reduced by 39.

If the value of p of the double-length number is less than 40, it is impossible to assign a characteristic to the less significant half; the complete less significant half is set to zero in this case.



In double length STANDARD floating point the same principle applies as in single length, viz., the value of 1 is made as small as possible thereby D0 & D9 of the more significant word are opposite digits.

20·4·2 Overflow with Floating Point Numbers Overflow can still occur with floating-point numbers, but only when the CHARACTERISTIC exceeds 8 bit capacity, thus allowing a range of about 10^{-38} to 10^{+38}. Note that in certain cases overflow can be set during the execution of an instruction when theoretically the result is within the range, but this can happen only if the correct value of p should be +127.

The concept of underflow also arises in floating numbers. If the p becomes less than -128, either in the result or during execution of the instruction, the result is set to zero, as the true result is too close to zero to be expressed in standard form. No indication of this occurrence is given to the programmer.

20·5 Single-Length Floating-Point Operations

In these operations all numbers **must** be in standard Floating form: all numeric results will be in standard floating form.

20·5·1 Floating-Point Add/Subtract

+F;	Add N1 and N2, giving rounded result in N1.
-F;	Subtract N1 from N2, giving rounded result in N1.
NEGF;	Change sign of N1 (performed by subtracting N1 from zero).
ABSF;	Find absolute value of N1 irrespective of sign. Performs NEGF; if N1 is negative, otherwise no action.

MAXF; Rearrange N1 and N2 such that the algebraically larger is in N1, the other in N2. If N2 - N1 would yield a negative answer, they are already arranged; if the result would be positive or zero, they are reversed and OVERFLOW set to indicate reversal.

SIGNF; Compares the two numbers in N1 and N2 and sets an indicator word in N1 to indicate which is larger. N1 will contain:-

(a) All zero if N2 = N1.

(b) D0 - 46 zero and D47 'one' if N2 larger than N1.

(c) All 'ones' if N2 less than N1.

[This indicator is NOT a floating number].
Overflow can never be set by this instruction.

ROUNDHF; Rounds a single-length floating number in N1 to halflength (ready for half-length store). The instruction effectively adds one to the D23 digit if the D24 digit is a 'one'. Since the complete word may now be shifted (to put the result in standard form), the state of D24 - 47 is undefined at the end of this instruction.

20·5·2 Single-Length Floating Multiply/Divide

×F; Multiply N1 and N2 together, giving a rounded single-length floatin result in N1.

÷F; Divide N2 by N1, giving a rounded single-length quotient in N1.

20·5·3 Non-Standard Floating Numbers If p is not the minimum possible for the number then the number is in non-standard form.

STAND; is the KDF 9 instruction designed to take a number in non-standard form and put it into standard form.

No other KDF 9 floating-point instruction is guaranteed to work correctly on non-standard data.

20·5·4 Double-Length Floating Point Operations

The double-length floating instructions are:-

+DF; Add N1, N2 to N3, N4 giving unrounded double-length result in N1,N2.

-DF; Subtract N1, N2 from N3, N4 giving unrounded double-length result in N1,N2.

NEGDF; Change sign of double-length number in N1,N2 by subtracting it from zero.

×DF; Multiply the SINGLE-length numbers in N1 and N2 together to give unrounded DOUBLE-length result in N1, N2.

×+F; Multiply the SINGLE-length numbers in N1 and N2 together to give DOUBLE-length product; then add this product to the DOUBLE-length number previously placed in N3, N4, leaving unrounded result in N1, N2.

+DF; Divide DOUBLE-length number in N2, N3 by SINGLE-length number in N1, giving rounded SINGLE-length quotient in N1.

ROUNDF; Round off a DOUBLE-length number in N1, N2 to SINGLE-length in N1. If the D9 digit of N2 is a 'one', a 'one' is added to the D47 digit of N1 and the result standardised if necessary.

20·5·5 Conversions Between Fixed-and Floating-Point

FLOAT; Take a single-length fixed-point number (expressed to p integral places) in N2 together with the integer p in N1: from these the corresponding floating-point number is generated in N1.

N.B. the value of p in N1 may not be the algebraic minimum. In this case a true STANDARD form result will be placed in N1.

FLOAT D; Takes a double-length fixed-point number (expressed to p integral places) in N2, 3 together with the integer p in N1, and produces the corresponding double-length floating number in N1, 2.

FIX; Takes a single-length floating number in N1 and from it produces in N2 the fixed-point version of the same number given to p integral places. The integer p is left in N1.

The word in N2 will have:-

(a) The same sign in D0 as the input word.

(b) The D9-47 digits of the input word, but moved up into the D1 - 39 positions.

(c) Zeros in the D40 - 47 positions.

Example 1

SET+5; SET+47; FLOAT; (gives 5 in floating binary);
The result in N1 will be:

0100000111010000... 0

i.e., an "f" of 5/8 with "p" of 3.

Example 2

If N1 is as result above, obey the instruction FIX; result will be:

```
N1    00000 ... 0000011

N2    01010 ... 0000000
```

i.e., N2 contains 5 to 3 integral places:
N1 contains the integer 3.

20·6 Conversion of a number other than a pure integer to character form

It must be emphasised that the instructions FRB; and TOB; explained in Section 11 only work with pure integers (held to 47 integral places). When studying this the reader may have asked: How are mixed numbers converted to character from? The complete solution is not going to be given as this question forms part of a programming exercise to follow; however a few hints are now given.

Assume a word contains a number n (with integral and fractional parts) in binary fixed point held to p integral places and it is required to print this number on the line printer. The programmer must know how many **decimal fractional** places (d) he requires to express the result to. He then will multiply his number by 10^d. (Held to 47 - p integral places.) This will give him $n \times 10^d$ held to 47 integral places.

He then converts the new number (which is an integer) to character form by use of FRB; and insertion of excess 16 bits. This pseudo result is 10^d times too big. By appropriate shifting, the pattern $37_{(8)}$ can be inserted in the proper position to represent the decimal point.

20·7 Examples

1. Y0 contains a binary fixed point number to 8 integral places. For purposes of a later division it is now necessary to hold the number to 10 integral places (in the same address). The instructions necessary are:

```
Y0;SHA-2=Y0;
```

Note that a negative shift (viz. moving the PATTERN to the right) **INCREACES** the number of integral places.

2. YP1 contains a binary fixed point number to 27 integral places. Write the instructions to cause the number to be held to the minimum number of integral places possible.

```
YP1;DUP;SET27;DUP;=C1;FLOAT;FIX;NEG;=+C1;ERASE;SHAC1;=YP1;
```

N.B. **(a)** The two instructions FLOAT;FIX; are useful when requiring to find the minimum value of p because FLOAT always gives the STANDARD floating point.

(b) The reason for not shifting the FIXED version and placing this back into YP1 is that the FLOAT only has 30 bits for f and therefore so has the FIXED version.

20·8 Exercises - Sets 1 & 2

1. Y0 to Y59 contain a set of binary fractions (fixed point) each to zero integral places. Write the instructions to leave the double-length sum of the double-length-squares (shifted to avoid overflow) in N1 and N2. How many integral places are there in your answer.

2. The number (A) in N2 lies in the range 9 - 30, and has 5 integral places. The number (B) in N1 lies in the range 18 - 34 and has 6 integral places. Write the instructions to divide A by B, giving as precise a result as possible.

3. N1 contains an octal number as eight octal digits in character form. Convert it to true binary by shifts.

4. N1 contains a Q-type parameter in the form n/1/A where there are n numbers stored from address A onwards (n<0). Write instructions to leave the largest number in the set defined by the parameter in N1, the smallest in N2. If n=0, set N1=N2=0.

5. The table below shows the conversion from an ordinary decimal number to the two component parts for standard floating form inside KDF 9. Fill in the missing items (as ordinary decimal numbers).

Number	f	p
2·5	·625	2
1·0		
-1·0		
5·25		
·25		
	·5875	3
	·8	-3

6. Given a string of n single length floating point numbers (not necessarily in standard form) in Y1 - Yn (n is given as an integer in Y0) calculate the mean value and store it in Y(n+1).
(The mean is the sum of all n numbers divided by n.)

7. Given the integer n in Y0, and a set of n single-length standard floating numbers (which may be positive or negative) in Y1 - Yn, write the instructions necessary to convert these numbers to fixed point expressed to p integral places and store the results in Y(n+1) - Y(2n). Store the value of p (which should be the minimum possible for the particular set of numbers involved) in Y(2n+1).

8. 64 binary fixed point integers are stored in Y0-Y63. Find their sum (single-length). Convert them to floating point numbers and find their floating point sum. Fix this sum and compare it with the original fixed point sum, setting N1=0 if the sums are equal, N1 =+1 if the second sums greater, otherwise setting N1=-1.

9. What instruction(s) would you use to contract a double length standard floating point number held in N1 and N2 into a single length standard floating point number held in N1.

21· PROGRAMMING EXERCISE B

It is proposed to read a list of N names from paper tape, sort them into alphabetical order, and then print the results on the high speed line printer, via OUT8 using stream number 30.

The data on paper tape which will be supplied is arranged as follows:-

1st Word.	Number N of names in character form in least significant end of word.
2nd-(N+1)st Words.	One name is contained in each word, the eight character positions being filled as follows:-
	(1 full stop, 7 alphabetic characters)
or	(1 full stop, 1-6 alphabetic characters, full stops)
(N+2(nd Word.	End Message character.

The number N of names will not exceed 31, and the data should be read to End Message.

Each of the sorted names (including the full stops) should be printed on a new line.

Only the V-store declarations and instructions need be written, i.e. you are not expected to prepare a heading sheet for the program.

The flowchart to be used is as follows.

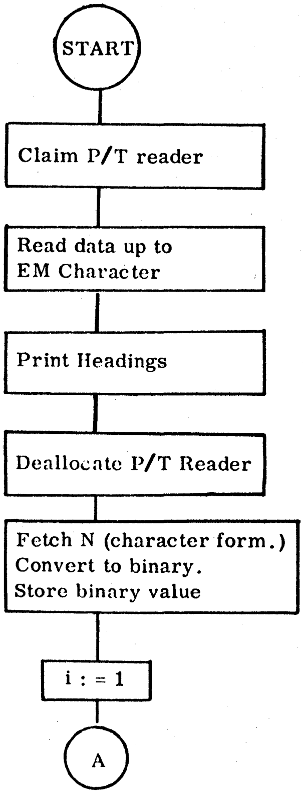
Hints:

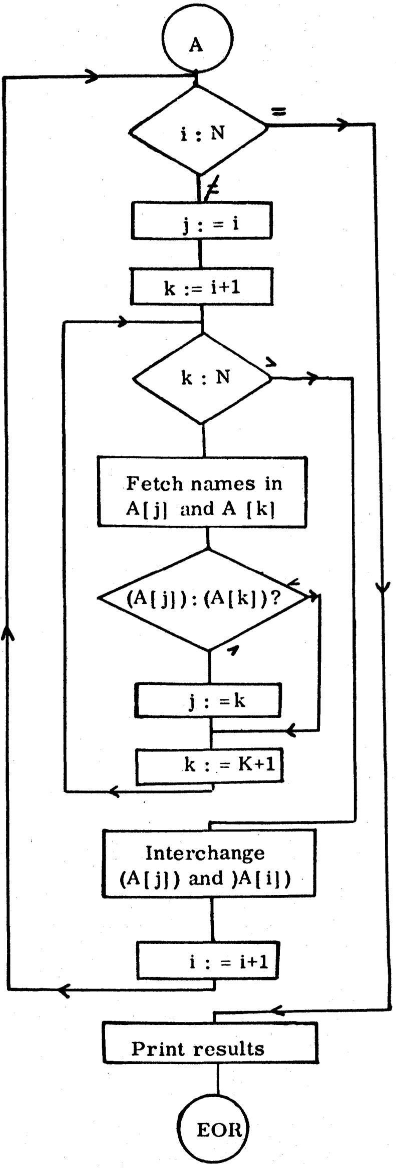
(a) Instructions such as -; look upon N1 and N2 as holding fixed-point numbers

(b) none of the instructions of the two preceding chapters are necessary.

(c) Before attempting the program the reader should work through the flowchart assuming a value of N = 5 (say) and with the five (say) words of data being the numbers 3,2,8,6,4 (say). This will give him an understanding of what the program is in fact doing. Start as follows:-

N	A[1]	A[2]	A[3]	A[4]	A[5]	i	j	k
5	3	2	8	6	4	1		
							~~1~~	
								~~2~~
							2	
								3





NOTE

A[1], A[2], A[3],...A[N]
reference the address in which the
N names to be stored are stored.

i references starting item in each pass.
j references smallest item in each pass.
k references successive items in each pass.

Each of the variables i, j, k, should
be stored in the modifier position of
a Q-store.

The reader after writing this program and referring to the model solution should read Appendix 5 which lists the output from the computer relating to failure reports etc.

22• MAGNETIC TAPE

22•1 Magnetic Tape Units

22•1•1 Principles of Magnetic Tape Recording. Use of magnetic tape as a recording medium has the one drawback that the tape cannot be visually checked after information has been written on to it. Consequently it is very necessary that programmers should have clear ideas on the usage of magnetic tape, and that they should appreciate the capabilities and the limitations of this method of information storage.

The process of recording information on magnetic tape involves the motion of the tape at high speed past a fixed recording head. If the tape is not moving no recording is possible. Similarly, the reading of recorded information is not possible unless the tape is in motion. An immediate consequence of this tape motion is that information on magnetic tape **must be recorded in blocks**, each block separated from the next by a gap. This is the space needed for the tape to slow down to rest at the conclusion of one recording and to accelerate to the recording speed at the commencement of the next. These gaps will always appear on magnetic tape whatever recording method is used. This is in contrast to the situation with paper tape, which may be stopped dead after any character and then restarted without the need for a space in which to build up speed.

When writing to a magnetic tape the length of the block required is defined simply by the amount of information being transferred, the tape motion automatically ceasing when the transfer is complete and thus leaving a gap on the tape as it comes to rest.

However, when a block of information is being read from magnetic tape into an area of main store reserved for it, three possibilities arise:-

(a) The reserved area of main store contains the same number of words as does the block of information on the magnetic tape. In this case the information transfer and the motion of the tape come to an end simultaneously.

(b) The main store area is larger than is required for the amount of information on the tape, so that the gap on the tape is reached before the reserved main store area is filled. In this case the tape stops on reaching the gap, and the remaining words in the reserved area of main store are left untouched.

(c) The information on the tape contains more words than does the area of main store reserved for it. In this case it would be disastrous to continue transferring to the main store until the gap on the tape is reached, since information required for other purposes could be over-written. For this reason the transfer will stop the moment the reserved area of main store has been filled, but the tape will continue to run until the gap has been reached, and only then will it stop.

These rules may be briefly summarised as follows:-

Information is transferred from magnetic tape to the main store until either the reserved main store area is filled or a gap on the tape is reached. In either case the transfer of information from tape to the main store ceases, but the tape itself will not stop until a gap is reached.

Thus the format of any magnetic tape will be: block of information; gap; block of information; etc. These blocks of information may be of any size, although an upper limit will be recommended later.

22·1·2 **Layout of Information on Magnetic Tape.** Information is recorded on magnetic tape at the rate of 40,000 characters per second, and the tape itself moves at a nominal speed of 100 inches per second. It should be noted that this tape speed is likely to vary for various reasons, so that the packing density of 400 characters per inch of tape is nominal only. To enable the machine to cope with such variations, when information is recorded on a tape a special timing bit is included with each character. When the tape is subsequently read the machine uses these timing bits to adjust itself to the rate at which the information arrives. This process is entirely automatic and requires no action by the programmer.

The length of the gap between one information block and the next, which depends essentially on the inertia of the tape, is of the order of one third of an inch, or the equivalent of about 140 characters of information. It will be realised from this figure that if only a few characters at a time are written on to the tape, then most of the tape will consist of inter-record gaps. Therefore, it is recommended that for optimum efficiency the information blocks should be large enough to give a reasonable packing density on the tape.

Information is recorded on magnetic tape as a sequence of six bit characters. Each 48 bit word from the main store is divided into eight groups each containing six bits, and the word is recorded on to the tape group by group, starting from the more significant end of the word (DO - D5) and finishing at the less significant end (D42 - D47). When the tape is subsequently read, the main store word is reassembled in its original form from the characters on the tape, so that this recording procedure places no restrictions on the nature of the information to be recorded.

With each six bit character there is automatically recorded a parity bit. This is an extra bit used for checking purposes when the tape is subsequently read. The convention governing the value of this parity bit is as follows:
if the binary pattern for the character contains an even number of 1's then the parity bit also takes the value 1. If the binary pattern for the character contains an odd number of 1's the parity bit takes the value 0 (zero). In other words, the convention is that the six bit character and the parity bit taken together must form a binary pattern containing an odd number of 1's. A parity failure on input means that the character has been found whose associated binary pattern in fact contains an even number of 1's, which implies a fault in the reading device, a fault in the device which made the recording, or a fault on the tape itself.

This is one failure which is never the fault of the programmer, unless he attempts to read beyond the end of the data on the tape. In this event a block which is not a multiple of eight characters may be read (some characters having been erased by the previous writing operations) which will result in a parity fail indication being set.

If a character with incorrect parity is found during reading, the computer will complete the read, skip the tape in the opposite direction and attempt to re-read the block. If the failure has now disappeared, the transfer will be completed and no indication given to the programmer: if it still persists, the parity fail indication will be set at completion of the transfer for the programmer to find subsequently.

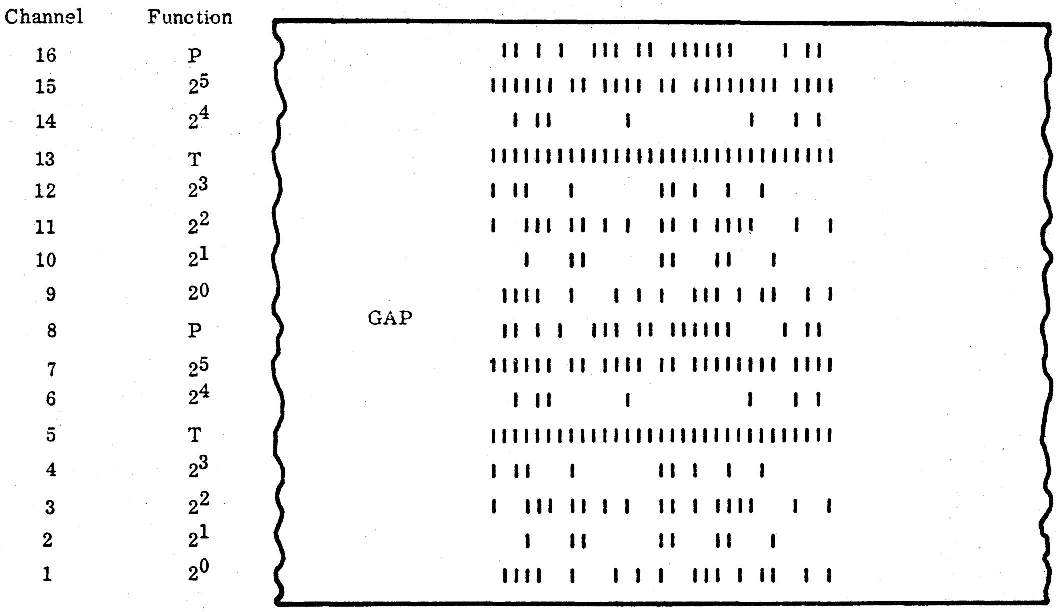
As an example, suppose that the six bit character to be recorded has the octal configuration 12(binary 001 010). The binary pattern for this character contains two 1's, so that the channel containing the parity bit must contain a 1 to preserve the odd parity required.

Together with these seven bits (six for the character itself and one for the parity bit) there is also recorded the timing bit mentioned earlier in this section. Therefore eight channels are required on the tape to record all the information needed for each item.

As an additional safeguard, when information is recorded on magnetic tape, these eight channels are duplicated side by side, i.e., once on the left-hand side and once on the right-hand side of the tape. Therefore, in its final form the tape has sixteen channels recorded along its surface. This dual recording technique is a means of safeguarding the information to be recorded against read failures due to faulty tape. Both copies of the contents of each digit position are scanned simultaneously when the tape is read, and if either or both of these copies give a valid signal for each of the eight channels, a correct character is transferred into the main store.

"A parity failure can occur during write, but only as a result of a hardware error. All programs should check for this, and if detected, arrange to rewrite the tape using a different tape station".

The diagram represents a length of magnetic tape. Underneath the tape are set out the alphabetical values of the characters recorded.



LAYOUT OF DATA ON MAGNETIC TAPE

Diagram of Magnetic Tape Recording

22·1·3 Control of Magnetic Tape. When it is required to begin writing information on to magnetic tape, it is necessary to move the tape to a standard position at its beginning. Any previous information on the tape is erased when the new recording is made, so by starting at the beginning of the tape rather than at any other point it can be ensured that no superfluous material will be left on the tape near the beginning. This positioning of the tape is accomplished by the use of a transparent section in the tape called the "Beginning of Tape Window". A light on the tape unit shines on to the tape, and when the window is in the right position the light passes through it and falls on to a photocell, which then signals that the tape is positioned at its beginning. It is from this position that the recording must always start as it is the only available reference point.

However, the beginning of tape window has a finite width and cannot be used to position the tape to an accuracy of better than about an inch. For this reason any recording made from the beginning of the tape is automatically preceded by running a few inches of tape past the recording head, all previously recorded information on this stretch of tape being erased. Then the actual recording begins. In this way it is arranged that the 'zero error' in the initial positioning of the tape shall be no trouble to the programmer.

Similar protection is necessary to warn the programmer when the end of the tape is near, to avert the danger of running off the end of the tape while information is still being recorded. The protection provided is twofold:

(a) a warning to the programmer that the end of the tape is approaching, and

(b) a command to the tape unit causing an immediate shut-down when the end of the tape is actually reached.

The two signals associated with cases (a) and (b) are called respectively the "End of Tape Warning" (ETW) and the "Physical End of Tape" (PET). The programmer should check for the ETW signal while information is being written on to a tape. Once this signal has been detected no attempt should be made to write anything further except for a short termination block indicating that the tape holds no further information.

Evidently no such check is necessary when a tape is read, because the information on the tape will have been terminated short of the end of the tape when it was recorded. In fact to test for ETW while reading can be dangerous, because in certain marginal cases the signal might not appear while the tape was being written but might appear while it is being read, an effect which could mean the loss of a block of information at the tail end of the tape. This possibility arises because of the configuration of the tape unit. The tape which passes under the read/write head is first unwound from a spool on one side, and afterwards wound on to a spool on the other side. Between each spool and the read/write head there is a bin into which the tape is allowed to spill in controlled quantities The test for ETW is made on the tape before it has entered the ingoing bin. It is because this bin between the ETW test device and the read/write head cannot be guaranteed to hold the same length of tape at all times that this discrepancy can occur. The test for the beginning of tape window is made at the read/write head itself, and so is not subject to this effect.

The End of Tape Warning is set by a marker on the tape as it enters the bin. ETW is cleared by the run reverse level in the tape station or by Beginning of Tape.

Since the movement of tape from the spool into the bin is liable to continue for some time after a forward operation has been terminated, ETW may be set at any time during this period. A reverse operation obeyed during this period would result in ETW being set, but immediately cleared again.

A programmer should, therefore, either:

(a) having commenced writing to a tape, refrain from any reverse operation on that tape until either ETW is detected or no further writing is required,

or

(b) ensure that any reverse operations performed either return the tape to Beginning of Tape, or cause the tape to move at least 50 feet in the reverse direction. This will ensure that the marker is back on the spool and ETW will be set again when it next enters the bin during a subsequent forward operation.

The physical distance between the markers along the tape is fixed, but the length of tape available for recording between the sensing of ETW and PET varies for the reason just indicated. The minimum length of useful tape after ETW, in the worst case, is five feet. With a recording density of 400 characters to the inch, this means that not more than 3,000 words of main store can be written to the tape after ETW has been sensed. For this reason programmers are advised in their own interests to limit the size of all their information blocks on magnetic tape to 3,000 words or less. This will ensure that the programmer can always finish writing his last block before PET, and will enable a short termination block to be added to indicate the end of the tape.

The following User Code instructions refer to the positions of a magnetic tape:-

MBTQq;
METQq;
MLBQq;

MBTQq; sets the test register if the read/write head is over the beginning of tape window. This instruction requires a Q-store in its simplest form with the device number in the counter position, the increment and modifier being ignored.

METQq; sets the test register if the end of tape warning signal is present. The Q-store contains the device number, as before.

MLBQq; sets the test register if the last block read was terminated by a last block marker. Once again, the Q-store contains only the device number. There is a special instruction for writing a block terminated by a last block marker which is a special mark written on the tape after all the information in the block - all forward read and skip instructions look for this marker and the device remembers whether it was present or not in case the program inspects for it.

22·1·4 Writing Fixed-Length Blocks. The simplest way of writing information on to magnetic tape is as a series of information blocks each of which as a length specified once and for all at the time the program is written. There fixed-length blocks may contain information in any form whatever. In fact, binary information, as will be seen in the next section, can be recorded only in fixed-length blocks. Note that by fixed-length we mean that this particular block always contains a given number of characters irrespective of the data used in the problem. The size of any other block on the tape does not apply as we are considering only the effect of one instruction that writes one block on the tape.

To write a fixed-length block a·Q-store is needed in the form:-

Qq = device number/lowest main store address/highest main store address.

The executive instruction MWQq; (Magnetic Tape Write) is then sufficient to record the information, starting from the DØ end of the word at the address given in the increment position of the Q-store and finishing at the D47 end of the word whose address is specified in the modifier, as one block on the tape designated by the device number in the counter position. Any previous information in this area on the tape is erased and a gap is left at the end of the block as the tape slows down to rest.

As an example, suppose it is desired to write the contents of the main store between the words YO and Y8 inclusive on to a magnetic tape, and further suppose that the device number to be used is at present in the top cell N1 of the nesting store. It will be necessary to declare a Q-store constant to contain the addresses of YO and Y8, and then to put it in a Q-store. For the purposes of this example the store Q1 will be used. Q1 will also be required to hold the device number. Since the information is to be written on to magnetic tape, the write instructions will be preceded by a check for the end of tape warning. The necessary instructions are:-

```
V1 = QØ/AYØ/AY8;
V1; = Q1;=C1; (Q-store now set up);
METQ1;J1TR;MWQ1;PARQ1; J2TR;
```

The instruction J1TR; transfers control to the end of tape routine, presumed to carry the label 1, if the test for ETW sets the test register. Similarly, J2TR; transfers control to a routine, presumed to carry the label 2, for dealing with parity failures if they arise whilst the tape is being written.

It was mentioned in Para.22·1·3 that it is often necessary to record a block followed by a last block marker. The instruction for this is MLWQq; (Magnetic tape Last block Write), which has the same effect as the instruction MWQq; with the addition of the last block marker immediately following the information block.

22·1·5 Reading Fixed-Length Blocks from Magnetic Tape. There is a similar set of instructions for reading magnetic tape. As indicated in Para. 1·1 however, the read operations are rather more complex. When reading an information block of a given size, the destination area of the main store can be:-

(a) exactly the right size;

(b) too large;

(c) too small;

"Note that if the block does not contain a multiple of eight characters the final (incomplete) word will NOT be transferred to the store under any circumstances."

All three possibilities have to be considered. The rules for reading blocks from magnetic tape are quite simple. Reading continues either until the allocated area of main store has been filled, or until the end of the block on tape has been reached, whichever is the earlier. In case (b) above the surplus main store words are left unchanged. Any words remaining on tape after the main store area has been filled (case (c)) will not be transferred to the main store, but the tape will continue to run past the reading head until the gap at the end of the block is reached.

Since it is possible to read a magnetic tape in either the forward or the backward direction, the read instruction to be given here will contain the extra letter F to indicate a 'forward' read. Reverse reading will be considered in a subsequent paragraph. The forward read instruction is MFRQq;. It requires a Q-store in precisely the same format as that used for the write instruction, i.e., containing the device number and the two limits of the main store to be filled from the tape. The normal sequence of instructions for reading a block of information from magnetic tape is:-

```
MFRQq;     PARQq;     JrTR;
```

MFRQq; is the magnetic forward read instruction, PARQq; is the parity check, and JrTR; transfers control to reference label r if the parity check set the test register. It should be realised that computations may be performed while this read instruction is being executed, provided they do not concern any part of the main store area involved in the transfer. This is done simply by inserting the instructions to be executed between the read instruction and the parity check. In fact this is true not only for the read instruction quoted here, but for any transfer instruction. Once the parity instruction is reached, the machine will wait if necessary until the transfer is complete so that the parity check can be performed on the complete block.

If a block is read which is followed by a last block marker, an indicator is set in the tape unit which may be interrogated by use of the following instructions:-

MLBQq;   JrTR;

MLBQq; (Magnetic Last Block) transfers the last block indicator from the tape to the test register. JrTR; is the jump instruction which transfers control to reference label r if the test register is set, i.e. if the block was in fact followed by a last block marker.
N.B. The last block marker indicator is automatically cleared as soon as the tape is moved again so that MLBQq; must be used before any subsequent write, read etc. instruction.

It is important to remember that the instruction JrTR- always clears the test register.

The instructions introduced for reading or writing fixed-length blocks may be briefly summarised thus:-

MWQq;   Write block of information on to tape.

MLWQq;   Write block of information on to tape followed by a last block marker.

MFRQq;   Read block of information from tape in the forward direction. Set tape unit indicator if block is followed by a last block marker.

MLBQq;   Transfers contents of tape unit indicator to test register, clearing the indicator.

JrTR;   Jumps to reference label r if the test register is set, clearing the test register.

22·1·6   Writing Variable-Length Blocks on Magnetic Tape. It is sometimes inconvenient to write information to magnetic tape in fixed-length blocks. Facilities are, therefore, provided for writing variable-length blocks by use of a specified control symbol. The symbol used for this purpose is the 'end message character'→ (octal 75). This character may be used with complete safety if the rest of the block in which it appears contains information entirely in character form, since no oonfusion can possibly arise between any of the preceding characters and the end message character itself. However, should the preceding portion of the block contain binary information (as opposed to the binary equivalent of character information), then there can be no guarantee that all the binary items shall have patterns distinct from the binary pattern for the end message character. If duplications of this sort are present then each one will be intepreted as an end message character. Since it is vital to avoid confusions of this sort it must be made a strict rule that the end message character should be used only for blocks containing information in character form. Binary information must be recorded in fixed-length blocks as described in the last paragraph.

The end message character on magnetic tape is designed for use when several main store areas of differing sizes, all containing information in character form, are to be written on to magnetic tape. For this purpose each block in the store should be terminated with a word containing an end message character. It has to be assumed that there is a maximum block size, and further that this maximum size is known to the programmer. The blocks of information must be stored in the main store as though each of them has this maximum size, i.e. assigning storage space for each block equal to the storage space needed for the block of maximum size. In general each such storage space will be only partially filled with information. To write each of these blocks on to the tape a Q-store must be set up to contain the device number and the addresses defining the size of the maximum space that the block can occupy in the main store.

Then the executive instruction MWEQq; will cause the contents of the space assigned to be written on to magnetic tape as a variable-length block. Writing will cease either:-

(a) When a word containing an end message character is written on to the tape, or

(b) when the highest address of the specified area of core has been written. This occurs when an end message character has not been discovered.

In case (a) this instruction writes a variable number of words as determined by the position of the end message character. In case (b) it writes a fixed-length block. In either case the block on tape will contain an integral number of words, as writing will cease at the **end of the word** containing the end message character.

The instruction MLWEQq; writes a variable-length block on to tape in just the same way as the previous instruction, but followed by a last block marker.

22·1·7 Reading Variable-Length Blocks from Magnetic Tape. A facility exists for reading variable-length blocks from magnetic tape. However, it should be pointed out that if the blocks were originally written in the way outlined above, then each will be followed, after the end message character, by a gap on the tape. Since reading will always cease when a gap is reached, it will not be necessary to use the end message character at all for this prupose. Nevertheless, since it may be of occasional use, the instruction MFREQq; (Magnetic Forward Read to End of message) has been provided which reads a block of information into the area of main store specified by the contents of the Q-store, ceasing:-

(a) when the end of the block on tape is reached, or

(b) when the designated area of main store is filled, or

(c) when a word containing an end message character has been transferred.

The last block indicator will be set if a last block marker is discovered. This instruction can be used to read the first few words of a block, ignoring the remainder of the block, if an end message character has been included in the right place. But in this case it should be remembered that although nothing after the word containing the end message character will be read into the main store, the tape will continue to move until the next gap is reached.

The instructions introduced for reading or writing variable-length will now be briefly summarised:-

MWEQq; Magnetic write to end message character.

MLWEQq; Magnetic write to end message character and terminate with last block marker.

MFREQq; Magnetic forward read to end message character.

22·1·8 **Reverse Reading from Magnetic Tape.** It is possible to read information from magnetic tape with the tape moving in the reverse direction (but it is **not** possible to **write** in the reverse direction). The instructions are similar to the forward read instructions given above, but contain the letter B (for 'backwards') in place of the letter F.

"Note that a backwards read instruction attempted on a tape positioned at the Beginning of Tape window will cause a failure (L.I.V.); the program should check for BT before a reverse read if such a condition is likely to arise".

When information is written on to tape, the first word on the tape comes from the lowest main store address specified, and the last word written on to the tape in the given block comes from the highest specified main store address. When the same block is read from the tape in the reverse direction, the first word encountered (which was the last one written) will go into the lowest designated main store address, and the last word encountered (which was the first one written) will go into the highest designated main store address, the intervening store area being filled from the bottom end. It is clear that the order of words in the main store has been reversed. This is a vital point to remember. Note, however, that the contents of each word are not changed in any way.

As an example, suppose that Y0 contains the characters A, B, C, D, E, F, G, and H, and that Y1 contains the characters 1, 2, 3, 4, 5, 6, 7 and 8, and that the following two instructions are obeyed:

MWQq; MBRQq;

where Qq contains device number/AY0/AY1. The first instruction, MWQq;, writes the two words on to the tape and MBRQq; reads them back with the tape moving in the reverse direction. The end result is that YO contains 1, 2, 3, 4, 5, 6, 7, 8, which was the last word written on the tape, and Y1 contains A, B, C, D, E, F, G, H, so that the order of the words in the main store has been reversed.

The two possible reverse read instructions are:-

MBRQq;	Magnetic backward read.
MBREQq;	Magnetic backward read to end message symbol.

22·1·9 **Positioning of Magnetic Tape.** It is sometimes necessary to re-position a magnetic tape without transferring information to the main store. For this purpose two skip instructions are provided in User Code, written MFSKQq; or MBSKQq; . For either of these instructions the Q-store is required with the device number in the counter position and a positive integer count not equal to zero in the modifier position. It must be emphasized that an attempt to use a zero or negative count with these skip instructions will not work. A zero count is interpreted as a count of 32,768.

The actual purpose of this count is to specify the number of blocks to be skipped. When a skip is performed in either direction information is read from the tape into the buffer but is not transferred to the main store. A copy of the integer · in the modifier position is put into a special count indicator. Every time a gap is encountered 1 is subtracted from the count, and when the count is reduced to zero the operation will cease. Note that the integer in the modifier does not change. As for all magnetic read instructions, a parity check is performed on all characters read, and on completion of the skip an indication is given if a parity failure has been discovered.

A parity fail indication will be given if any block skipped contains other than a multiple of eight characters.

A failure entry to Director (LIV) will occur if a backward skip is initiated on a tape positioned at the Beginning of Tape window.

If during the execution of a forward skip instruction a last block marker is encountered, or if during a reverse skip the beginning of tape window is reached, then the skip operation is immediately terminated since there can be no point in skipping beyond either of these marks. Note that there is always an extra long gap on the tape between the beginning of tape window and the first block. Therefore, when skipping backwards the tape may be stopped just in front of the first block and yet well short of the beginning of tape window.

As an example, suppose that n blocks have just been written on to a fresh tape and that it is required to check that the tape is correct before proceeding. It is assumed that Q1 has already been set up with the device number in the counter position and the count n in the modifier position. The appropriate instructions are:-

MBSKQ1; PARQ1; J1TR; MBTQ1; J2TR; SET+1; =M1; MBSKQ1; PARQ1; J1TR- MBTQ1; J3NTR;

These instructions perform the following operations:-

(a) MBSKQ1; skips back n blocks. If there really are n blocks on the tape then the read head on the input device will be positioned **just in front of the first block**, not yet having reached the beginning of tape window. This is because the tap between the BT and the first block is about seven inches long, but the tape stops as soon as the beginning of the gap is sensed.

(b) PARQ1; sets the test register if a parity failure has occured.

(c) MBTQ1; sets the test register if the beginning of tape has been sensed; this would occur if fewer than n blocks were present on the tape. J2TR; jumps to reference 2 if the tape is at BT. This prevents a possible failure on backward skips if the number of blocks found is too few.

(d) The instructions SET+1; =M1; reset Q1 ready for a further backward skip of one block, and MBSKQ1; performs this skip. If the tape is correct this operation will position the tape at the beginning of Tape window.

(e) PARQ1; sets the test register if a parity failure is detected during this second skip.

(f) J1TR; jumps to reference 1 if test register has been set by any of the faults in (b) or (a) and also clears the test register.

(g) MBTQ1; sets the test register if the tape is positioned at the beginning of tape window, as it will be if the tape has the correct number of blocks n. J3NTR; jumps to reference 3 if the test register has not been set, otherwise it clears the test register. This jump will be made if there are too many blocks on the tape.

If no parity failures have been found and if the tape contains the correct number of blocks n, the program will proceed to the instructions immediately following.

If it is required to ensure that a tape is positioned at the beginning of tape, the instruction MRWDQq; will do this (requiring only the device number in the counter at Qq). The tape will start to rewind and will continue to do so until the BT window is sensed and will then stop. There is no need to check for beginning of tape - it will not stop until this point is reached. As the tape is not inspected during rewind, the instruction cannot give rise to a parity fail indication.

When a tape station is claimed for use by a program there is no need to use MRWDQq; as Director always leaves the tape positioned at BT after allocating it to the program.

Note that the modifier of Qq must be positive or zero when a rewind instruction is obeyed. If a rewind instruction is given with the tape at BT, the instruction will immediately terminate as the desired condition exists (this is the only reverse tape instruction that can be obeyed from BT): if the tape is not at BT, it will move in the reverse direction until BT is reached.

22·1·10 **Tape Labels.** The first block on any magnetic tape must contain a statement known as the tape label. The tape label contains a minimum of two words and a maximum of 16 words. The first word contains the physical number of the spool and must be retained on the tape at all times. The second (or second and third) word contains what is known as the tape identifier which is of either 8 or 16 characters (known as 1 or 2 word identifiers). For 2 word identifiers the first character of the 16 **must** be "+". The remaining 13 or 14 words of the label are entirely at the disposal of the programmer. For instance, they may describe in words the present contents of the tape.

22·1·11 **Claiming Tape Units.** The reader is asked to revise sections 17·3·3 and 4.

22·2 **Examples**

1. Write the contents of YPO to YP63 immediately after the tape label block of the magnetic tape with the identifier KAXZ 1274. Claim the device but do not deallocate

```
V0  =  Q O/AYPO/AYP63;
V1  =  P KAXZ1274;
V2  =  Q Ø/Ø/1;

VØ;=Q1;V2;=Q5;V1;SET4;OUT;=C1;C1TOQ5;MFSKQ5;PARQ1;J5TR;MWQ1;
PARQ1;J1TR;
```

It will be seen here that no test MBTQq;METQq; has been made: this is because the tape must be at BT since it has only just been claimed and Director always leaves the tape at BT when allocating. If METQq; should cause the indicator to be set it would mean that the tape would only be about a foot or so long which is unreasonable to check for at this point. Reference 1 is a routine if parity is sensed in 1st block. Reference 5 is a routine to be obeyed if parity is sensed in label block.

2. Follow on example 1 writing a further 10 blocks each of 2000 words from Y2001-Y4000,...,Y200,001-Y22,000. Finish up with a last block marker and deallocate.

```
V2  =  Q 0/AY1/AY2000;

V2;=Q2;C1TOQ2;SET9;=C3;
```

```
4;  METQ2;J2TR;SET2000;DUP;=+I2;=+M2;MWQ2;PARQ2;J3TR;DC3;
    J4C3NZ;METQ2;J2TR;MLWQ2;PARQ1;J3TR;
    C1;SET6;OUT;

2;  (Routine if end of tape is sensed);

3;  (Routine if parity is sensed in last 10 blocks)
```

The reader is asked to note that the parity test should be left until just before the next write is performed, thus saving time. It is normal to check that the tape has been correctly written; this is shown in the next example.

3. Claim the magnetic tape unit with the tape which has identifier +KARSTUXZ31ABC36, write onto the tape immediately after the label block the contents of Y1 - Y1000 as one block, with a last block marker. Check that the tape has been written correctly and deallocate the device.

```
V0/1  = P+KARSTUXZ31ABC36;
V2    = Q 0/AY1/AY1000;
V3    = Q 0/0/1;

V2;=Q1;V1;V0;SET10;OUT;=C1;ERASE;V3;=Q2;C1TOQ2;MFSKQ2;PARQ2;J1TR;MLV
SET1;=+M2;PARQ1;J1TR;
MBSKQ2;PARQ1;J1TR;MBTQ2;J2TR;SET1;=M2;
MBSKQ2;PARQ2;J1TR;MBTQ2;
J3NTR;C2;SET6;OUT;
1;  (Parity failure routine);
2;  (Routine for too few blocks written);
3;  (Routine for too many blocks written);
```

N.B. Above are the two instructions MBSKQ2; and PARQ1;, provided the device number in Q1 is the same as that in Q2 this is quite valid.

4. Claim the magnetic tape with identifier ABCD1234, read the first block after the label block into YA0-YA63. (The block is known to contain only 64 words.) Deallocate after use.

```
V0  =  PABCD1234;
V1  =  Q 0/AYA0/AYA63;
V2  =  Q 0/0/1;

V1;=Q1;V0;SET4;OUT;=C1;V2;=Q2;C4TOQ2;MFSKQ2;PARQ2;J1TR;MFRQ1;
PARQ1;J1TR;C1;SET6;OUT;
```

22·3 Exercises - Set 1

In these examples all units should be claimed and de-allocated where appropriate, all relevant checks should be made. If a parity failure is detected the tape should be de-allocated. If there is not enough tape available for writing the full block required write a 1 word block of zero terminated with a last block marker and then terminate.

1. The three words Y0-Y2 contain the following in character form:-

Y0 THE␣CAT␣
Y1 SITS␣ON␣
Y2 THE␣MAT␣

Give the instructions to write these three words to Magnetic tape (device number in C1) - and then read them reverse into YB0-YB2. Show the contents of YB0-YB2 at the end of the operation.

Assume the tape is positioned just after the label block and ready for writing.

2. Position the tape with identifier ABCD1234 in the gap between the 3rd and 4th block of information. Do not count the label block as a block of information.

3. Write the contents of Y0 - Y1000 to a magnetic tape labeled ABCD1235 as a simple block immediately following the label block, then check that the block contains no parity and that one and only one block was written.

Exercises - Set 2

1. Copy the magnetic tape with identifier KEEM0000 outs tape with identifier KEDJ9999 reading and writing blocks to end message. The maximum size of a block is 1024 words. Stops when a block with a last block marker has been copied (with a last block marker). Check that the new tape has been correctly written before de-allocating. A system of double buffering is expected to be used. I.e., this program should be so wirtten that whilst one tape is being written onto, a simultaneous read from the other should be taking place.

23•1 Functions of a Subroutine

A subroutine is a self-contained set of instructions which, when presented with data in pre-defined storage locations, performs a particular operation using that data, and leaves results again in pre-defined locations. Note that particular subroutines can exist that either require no data, or give no results, or both.

The question arises as to why subroutines are used at all. The reasons for the use of subroutines are:-

(a) Where the program involves the use of certain sequences of instructions more than once - often many times - the use of subroutines covering such sequences relieves the programmer of the tedium of writing them all out in full each time they are needed. A private subroutine is the ideal way of achieving this.

(b) Certain sets of instructions, particularly those covering established mathematical procedures, have already been previously established and registered in a subroutine library for future use. It is obviously preferable to accept the rules for existing routines of this nature rather than to formulate, write, and test, a different set of instructions to achieve the same end.

The growing library of KDF9 subroutines is available to all users of the machine who are invited to add to it any new routines of general interest they have developed, or any useful alternatives to existing routines. In this way the library will continue to grow in scope and capacity to the benefit of all.

The method of extracting library subroutines from magnetic tape when compiling with the POST system has already been explained. When this method is not used the subroutine must be written in full at the end of the main program, and the rules for this now follow.

23•2 Rules for Writing Subroutines

23•2•1 Beginning of a Subroutine. The start of a subroutine on a User Code tape is detected by Compiler from the label which MUST be the first thing the appears. Once the label has been found the list of reference labels held by Compiler is restarted, thus allowing any subroutine to commence using labels from 1 onwards up to 255 for jumps etc., without confusion. Similarly the list of constants is restarted so that the first constant used by the subroutine will be called V0. The subroutine label will be of the form:-

(a) A letter (L for library subroutines, P for private subroutines.

(b) The subroutine number (numbers above 1000 will be used only for special purposes.

(c) The letter V followed by a number v where the subroutine requires space for constants from V0 to Vv inclusive (if no constants are required this item will be omitted).

(d) The semi-colon ending the label.

Once the label has been detected all that follows is interpreted as part of the subroutine until either another subroutine label is encountered, or until the FINISH; label appears on the input tape (it follows from this that subroutines must appear after the main program).

23•2•2 Use of Stores by Subroutines. Any data required by subroutines should be obtained from the nesting store (and removed during the operation of the subroutine) and any result put into the nesting store - this makes the routine look as much like the built-in computer operations as possible. If larger amounts of data are required, the data in the nesting store should be addresses telling the subroutine where the rest of the data are stored or where the results are to be placed.

Q stores should be used from Q15 downwards to avoid conflict with the main program using them from Q1 upwards.

V constants may be used as required, numbered from V0 upwards, without risk of conflict with similarly numbered constants in other places.

W stores may be used for storage space if required, but the subroutine should not expect to find any particular patterns in the W stores on entry. This rule removes any obligations for a subroutine to clear any W stores used on exit.

Reference by a subroutine to the main store using DIRECT addressing should be avoided at all times (except for V constants and W stores), as this would seriously impede the usefulness of a subroutine. INDIRECT addressing by a basic address provided in the nesting store at entry is a far more useful and flexible technique if main store areas are involved.

23•2•3 Exit from a Subroutine. At the conclusion of a subroutine it is necessary to return to the main program at a point immediately following the point from which is was left to enter the subroutine. This makes the jump to subroutine instruction look just like a rather powerful machine instruction.

To perform the necessary jump back to the main program we use the address put into the jump nesting store on entry to the subroutine (this tells us the point at which we left the main program) and a numerical value indicating the displacement beyond this point (measured in HALF-WORDS, this being the most general unit for this application, as a jump instruction takes one half-word). A single instruction, which may be written EXIT n; where n represents a numerical value in units of half-words, will perform this function. The usual form is EXIT 1; to return to the main program at the instruction following the jump to subroutine instruction. For example, the sequence of instructions:-

```
Y6;      JSP4;      =Y7;
```

will perform the following actions:-

Y6; fetch a word from Y6 to N1.

JSP4; store the address of THIS INSTRUCTION in the top cell of the jump nesting store; then enters subroutine P4 at its first instruction and obey its instructions. terminating with

EXIT 1; fetch the address from the top cell of the jump nesting store, add to it 1 half-word (i.e., 3 syllables) and then jump to the resulting address. In this example, the address stored in the jump nesting store is that of the instruction JSP4; which occupies 3 syllables. Adding one half-word (i.e., 3 syllables) to this address gives the address of the next instruction (=Y7;) so that is the instruction to be obeyed after the exit jump.

=Y7; store the result from the subroutine.

This illustrates how a subroutine can be obeyed at any point in a program by writing just one instruction.

23•2•4 **Subroutines with two exits.** On occasions, subroutines may require alternative exits, one for the normal case and a second to indicate a failure or some other unusual occurrence. The exit instruction can deal with this; EXIT 1 is used for the UNUSUAL case and EXIT 2 for the normal case. It is not advisable to provide more than two exits as the use of such a subroutine becomes involved. The subroutine is then obeyed by a sequence such as:-

```
Y8;      JSP9;      J3;      =Y9;
```

Broken down into steps this becomes:

JSP9; Enter subroutine storing address of THIS INSTRUCTION

For NORMAL subroutine exit.

EXIT 2; Jump to point 2 half words beyond address stored; this takes us to =Y9; and the following sequence of instructions.

For FAILURE subroutine exit.,

EXIT 1; takes us to J3; which jumps out of the sequence to a routine for dealing with the failure.

23•2•5 Use of Overflow and Test Register in Subroutines. Any subroutine must ensure that, on exit, the states of the overflow and test register are precisely the same as they were on entry.

The sequence ZERO; DUP; MAX; ERASE; ERASE; will set the overflow register.
SET-1;=TR; will set the test register.
VR; clears the overflow register and ZERO;=TR; clears the test register.

23•3 Control of Subroutine Jump Nesting Store

23•3•1 General Uses of SJNS. When subroutines are used as described above, the Jump Nesting Store will look after itself, removing each return address as it is used. The only point to remember is that not more than foruteen return addresses should be in the store at any time - this allows one space for a program-testing subroutine to use and one for use during interrupts into Director. Since this allows 14th order subroutines, it is not a serious limitation.

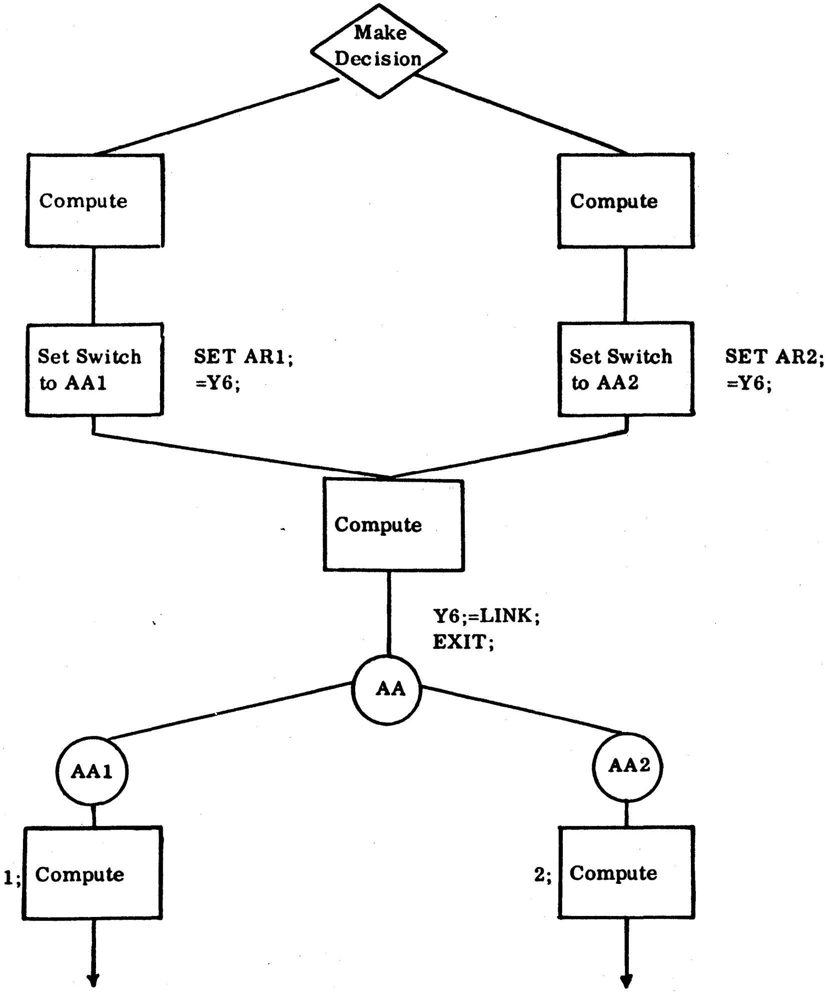
It can sometimes happen that a return address may exist, but not be required, or an extra address be desired. Instructions therefore exist to remove or insert addresses by transferring to or from the top cell N1 of the nesting store. Such an address when in N1 is of the form:

D0 - 31: Zeros (ignored when transferring to SJNS).
D32 - 34: Syllable number (in range 0 - 5).
D35 - 47: Word address (in range 0 - 8191).

The two instructions involved are:-

LINK; Fetch address from top cell of the subroutine jump nesting store into N1.

=LINK; Transfer address from N1 into top cell of the subroutine jump nesting store (ignoring digits DO - 31 of N1).



Use of the Subroutine Jump Nesting Store for Switches.

23·3·2 Use of SJNS for Switches. It often happens that a decision made at one point in a program controls the route that the program will follow at a later stage. To avoid having to make the decision twice, a switch can be set after the first decision has been made, and used later to direct the program along the desired path.

The diagram illustrates the logic to be followed. Two steps are considered here:

(a) setting the switch.

(b) branching according to switch setting. Each branch needs a reference label; in the example 1 and 2 are chosen.

The instructions for setting the switch on the left-hand branch are:

SET AR1; Puts word and syllable address of the instruction labelled 1 into N1.

=Y6; Stores the address in a known position. Any location will do - Y6 is an arbitrary choice.

For the right-hand branch we have:-

SET AR2; Different address this time ....

=Y6; ....but stored in SAME location.

To branch on the setting of the switch we use:-

Y6; Fetch address back to N1.

=LINK; Send it to the jump nesting store,

EXIT; and jump to it (Note: this is EXIT n with n = 0).

23·3·3 Use of SJNS for Trees. It is sometimes necessary to jump to one of a number of points, depending on the value of an integer (either computed or provided as data). An example might be an electricity billing program, where the cost is computed in differing ways according to the particular tariff employed. This would be done by a series of tests, but above a small number this becomes time consuming. An alternative means is provided by a variation on the EXIT instruction which, being similar in form to an ordinary jump instruction, has space to keep an address with the instruction.

Consider these instructions:-

```
YO;     =LINK;     EXIT AR10; ..........
*10;     J100;
        *J101;
        *J102;  .....
```

Reference 100 is assumed to be the first instruction of the routine for tariff zero, 101 for tariff one, and 102 for tariff two. The effect of the instructions is thus:-

YO;   Fetch tariff number to N1.

=LINK; Send tariff number to SJNS (as word address 0, 1 or 2).

EXIT AR10; The address in this jump is that of the asterisked instruction labelled 10. The address in the jump nesting store is either 0, 1 or 2 with syllable equal to zero. The jump is, therefore, to reference 10 for tariff zero, one word beyond if tariff one, or two words beyond if tariff two. In these three locations (all asterisked to ensure they are in seperate words) are jumps to the start of the appropriate routines - these may be anywhere.

23•4 Examples

1. Write a subroutine to convert a set of n single-length positive integers from true binary to decimal character form, storing the results in the main store words from which the data was obtained. Assume that on entry the top cell of the nesting store contains a parameter in Q store format n/1/address, where n is the number of words to be converted and the address is the starting address of the data. Exit 1 if one of the single-length fixed point numbers is too large to be expressed in 8 characters, (there is no need to reset the original data in this case), Exit 2 is to be normal exit.

```
2.  P3V1;
    V0 =    B 1212121212121212;
    V1 =    B 2020202020202020;
    =Q15;J1C15Z;V1;V0;
2;  DUPD;MØM15;
    FRB; OR;J3V;=MØM15Q;J2C15NZ;
    ERASE;ERASE;
1;  EXIT 2;
3;  ERASE;ERASE;
    ERASE;EXIT1;
```

23•5 Exercises - Set 1

1. Write a subroutine to claim a magnetic tape from Director and leave

(a) the device number in N1

(b) the tape positioned in the gap following the label block.

Use the contents of N1 as the identifier for the tape.

Use a failure exit if a parity failure occurs.

2. A set of true binary integers each less than 10^9 are stored in consecutive words of the main store. Write a subroutine to set the top 3 cells of the nesting store to contain:

N1: the total of all the members.
N2: the largest number in the set.
N3: the smallest number in the set.

At entry N1 contains n/1/A where n is the number of integers in the set, A is the first. The n words of data are known to be correct and n is known to be greater than zero.

3. Transfer control to Reference 3 in a program if N1 is zero; transfer control to 1 word beyond reference 3 if N1 is non-zero.
Preserve the contents of N1 in either case. Do not assume an asterisk on Reference 3.

Exercises - Set 2

1. Write a subroutine to interchange the Dr bit and Ds bit in a binary pattern in N1 where r is stored as an integer in N2 and s in N3.

24•1•1 Operation of the Control Unit. There are two classes into which computational instructions may fit:-

(a) fetching or storing words in the main store, and

(b) performing arithmetic operations.

If each instruction were obeyed to completion before control moved onto the next instruction, we would find that at any one instant either the main store or the arithmetic unit would be in use, but not both. This would be inefficient as it would imply that one or the other must be idle.

The advance control feature of KDF 9 is designed to reduce this inefficiency by allowing operations to proceed in both parts at once, but in such a way as to completely safeguard the programmer from error due to this dual working and without imposing any extra restriction on him at all.

The system used requires a Control Unit in two parts:-

(a) one to look after the main store and its associated parts,

(b) the other to look after the arithmetic unit and the nesting store.

There is a two-word (12 syllable) instruction buffer in the control unit which is a temporary transit camp for instructions obtained in sequence from the program area of the main store. From the programmer's point of view the instructions are brought into this buffer at the right-hand end and are gradually shifted along to the left as and when control has finished with the instruction at the extreme left. Since instructions may be one, two or three syllables (8, 16, 24 bits) long it will be seen that the number of syllables brought onto the right hand end may be less than is sufficient for a complete instruction. This may be clearer if the sequence

+;=Y6;-;ZERO;DUP;AND;NOT;=Y5;Y7;ZERO;

is considered.

At one instant the 12 syllables + to =Y5 will be in the buffer, but when + is completed it (1 syllable) will be pushed off the left-hand end and the first syllable of Y7 will be brought into the vacancy created by the shift. This will not affect the programmer because when =Y6 is dealt with, the final 2 syllables of Y7 will be brought in and so the complete instruction (Y7) will be re-united.

This picture of instructions at the left hand end of the buffer being performed one at a time is not exactly what happens. In fact, each instruction passes to the main store part of control and then any which are purely arithmetic are passed onto the arithmetic part of control, each part taking the necessary actions. If the main store part of control senses that the instruction is purely arithmetic it starts to look at the next instruction leaving arithmetics to deal with the previous instruction when it is free to do so. Hence it is possible for arithmetic control to be obeying the first syllable of the buffer with main store control having moved right up to the last syllable of the buffer.

24•1•2 **Main Store Buffers.** There is one point at which the activities of the two parts meet - when information passes to or from the nesting store. In cases of this kind main store control will obtain the word from its storage location, but arithmetic control will put it into the nesting store.

The hand-over is effected by interposing a set of buffers between the two parts. Any word from the main store is sent to one of two "fetch buffers" by main store control and waits there until arithmetic control is ready for it (since the buffers are used alternately, both sides know which to use next).

Any word sent out of the nesting store goes to a single "store buffer", where it waits for main store control to deal with it. Another problem arises here: main store control has finished with such an instruction before arithmetic control starts, but the result is not available until both have dealt with it. This is overcome by main store control placing the address into which the result will go into a fourth private register before allowing arithmetic control to deal with this instruction: when arithmetic control has placed the result in the store buffer it signals main store control that the result is available and leaves main store control to store it (at some later date) into the location specified by the fourth register.

By this means both parts can be kept busy for a greater portion of the time, resulting in a reduction of the total elapsed time to perform a given job.

24•1•3 Programming for Advance Control. The Advance Control feature does NOT put any restriction on the programmer whatsoever - he can obey instructions in any order he likes and KDF9 will act accordingly. However, it does offer the chance for the programmer to save time by giving Advance Control as much scope as possible. In general this is done simply by keeping references to locations outside the nesting store as far apart as possible by fitting the arithmetic instructions between them.

For example, to add together the 6 floating numbers in Y0 to Y5 we have two alternatives:-

(a) Y0; Y1; Y2; Y3; Y4; Y5; +F; +F; +F; +F; +F;

(b) Y0; Y1; +F; Y2; +F; Y3; +F; Y4; +F; Y5; +F;

Both arrive at the same result but take differing times. The first works on the main store only until all six operands are in the nesting store then leaves all except the arithmetic side idle whilst the numbers are added. The total time is thus the sum of the individual times. The second solution spreads the load as best it can: as soon as the arithmetic unit can start, it does so, and whilst it computes the first sum, main store control is fetching the next numbèr. As the times for floating add and main store fetch are roughly equal, the second solution could be up to 35% faster than the first.

Because of the dual workings inherent in the Advance Control feature, it is difficult to specify how long an instruction will take to be obeyed without studying its context - for this reason no attempt has been made to quote times for instructions when they have been introduced in earlier sections.

24•2 Short Loops

24•2•1 Theory of Short Loops. We have seen earlier that several instructions (12 syllables) can be stored in the instruction buffer, and that as the leading instructions are completed they are shifted out of the buffer. When a jump is encountered the whole of the buffer has to be cleared and re-filled from the appropriate part of main store. In the case of a loop of instructions which is so short that they all fit into the instruction buffer at one time it is time-wasting to force the control unit continually to fetch the same two words of instructions every time round the loop. To prevent this continual fetching, a special jump instruction is used:

JrCqNZS;

The actual operation of this instruction is to jump **to the first syllable of the word preceding the word in which the jump instruction is stored** - hence an address is not required in the machine code instruction and therefore this jump instruction occupies only TWO syllables as opposed to 3 for all the other jump instructions.

Both syllables of JrCqNZS must be stored in the same word. Note that JrCqNZS; has the same operation as JrCqNZ; except that the jump is to the first syllable of the preceeding word. The effect in the instruction buffer of JrCqNZS; is that the first time it is encountered the previous word is placed into the first word and the following word (that containing JrCqNSZ;) into the second word of the two-word instruction buffer. The instruction buffer is not now cleared until Cq is zero.

The reader should now re-read para 6•2•1•3 concerning the "asterisk".

Since the machine code version of JrCqNZS; does NOT contain the address of reference r as a jump address it must be remembered that any jump instruction (other than JrCqNZS itself) that is obeyed during a short loop and that causes a jump to take place will remove the indication that a short loop is in progress. Such a jump MUST upset the contents of the instruction buffer and, consequently the short x loop.

24•2•2 Writing Short Loops

(a) Remember that the instruction buffer holds 12 syllables, and that JrCqNZS; is only 2 syllables.

(b) The whole of JrCqNZS; must be in the last 6 syllables of the buffer.

(c) To ensure that reference r is at the beginning of a word, precede it by an asterisk.

(d) Any unfilled syllable(s) occuring when an asterisk is employed is filled with the 1 syllable instruction DUMMY;

(e) If an ordinary loop using JrCqNZ; has been written which has (or could be reduced to have) 13 syllables or less then by using JrCqNZS; it can be made into 12 syllables and eventually by inserting an appropriate asterisk a short loop may be formed.

(f) Appendix 4 lists the syllable count of instructions.

Some examples are:-

Example 1

1; Y0M1; Y64M1; xD; CONT; =Y128M1Q; J 1C 1NZ; (14 syllables).
Either leave alone or replace by a more economical form such as:-

1; M2M1; Y64M1; xD; CONT; =Y128M1Q; J1C 1NZ; Now 13 syllables, so use J1C1NZS; resulting in

*1; M2M1; Y64M1; xD; CONT; =Y128M1Q; J1C 1NZS;

Example 2

2; M0M1Q; M0M2Q; x+F; J2C 1NZ; (8 syllables) becomes

*2; M0M1Q; M0M2Q; x+F; *J2C 1NZS; (this will be discussed further in next paragraph, as it can be rewritten to give faster computation).

24•2•3 **Effect of Advance Control in Short Loops.** Let us consider just what happens when the short loop of the last example is obeyed.

The two instruction words are laid out:-

	Word 1						Word 2					
Syllable No.	Ø	1	2	3	4	5	Ø	1	2	3	4	5
	M0M1Q		M0M2Q		x+F	DUMMY	J2C1NZS					

Main store control obeys the two fetch instructions, with arithmetic control waiting for main store control.

Syllable 4 & 5 are inspected and immediately ignored by main store control as they are purely arithmetic instructions.

Arithmetic control starts obeying syllable 4 whilst main store control continues into syllable 0 of word 2. The jump instruction is obeyed quicker than x+F so main store control tries to obey syllable 0 and 1 of word 1 but cannot do so yet because arithmetic control is still in that word on the previous loop (this precaution prevents main store control from "lapping" arithmetic control). Therefore main store control waits until arithmetic control has finished with

(a) x+F and

(b) inspected DUMMY, and finally left WORD 1.

If the layout of the instructions had been

```
 M0M1Q   M0M2Q   DUMMY   x+F     J2C1NZS
```

then at the point in the loop when main store control enters word 2 the arithmetic control would have just finished checking that DUMMY does not have to be obeyed and starts on x+F. This means that main store control can start obeying M0M1Q as soon as arithmetic control has taken what information it needs from the x+F instruction.

In the previous layout with x+F followed by DUMMY, arithmetic control would not allow main store control to reenter Word 1 until it (A.C.) had finished x+F and inspected the DUMMY, and hence the main store control would be needlessly held up until A;C. had checked DUMMY. However in the second layout the DUMMY is inspected by A.C. whilst M.S.C. is busy.

Basically the rules for optimisation of a short loop are as follows.

(a) If it is possible, arrange the short loop so that one and only one arithmetic instruction occurs in the first word together with main control instructions.

(b) The arithmetic instruction of (a) must occupy the last syllable of the first word (or last two in the case of a two syllable arithmetic instructions). To achieve this, pad out with dummies if necessary.

(c) Employ the Q facility only once in the loop, if this is possible.

24•3 Exercises - Set 1

Use a short loop in your solutions to the following excercises.

1. 64 single-length floating point numbers are stored from W0 onwards. Find the sum of their squares as a double-length floating point number leaving the answer in N1, N2.

Excercises - Set 2

1. 128 integers are stored in Y0 - Y127. Find the sum of the numbers leaving it in N1; the largest number in the set, leaving it in N2; the smallest number in the set, leaving it in N3.

25· PROGRAMMING EXERCISE C

It is proposed to generate a sequence of 48 numbers according to the following rule:

"The nth number in the sequence is obtained by adding together the two previous terms of the sequence, i.e., by adding the (n-2)nd and (n-1)st terms. (Using zero to initialise the procedure the first terms of the sequence are

1, 1, 2, 3, 5, 8, 13, - - - -)"

When all the numbers have been generated they should be written as a single block to magnetic tape. This block should then be read backwards from the tape, and the ratio for each pair of successive terms calculated from these numbers. Each term of the given series, together with the corresponding ratio, should be arranged to be printed via OUT8 using stream number 30.

Notes

1. In calculating the ratio the larger number should be made the numerator, and the smaller the denominator. The ratio should be calculated to 13 decimal places.

2. A library subroutine will be provided to convert binary numbers to the characters required for output on the printer, it is L1000 and has the following specification:-

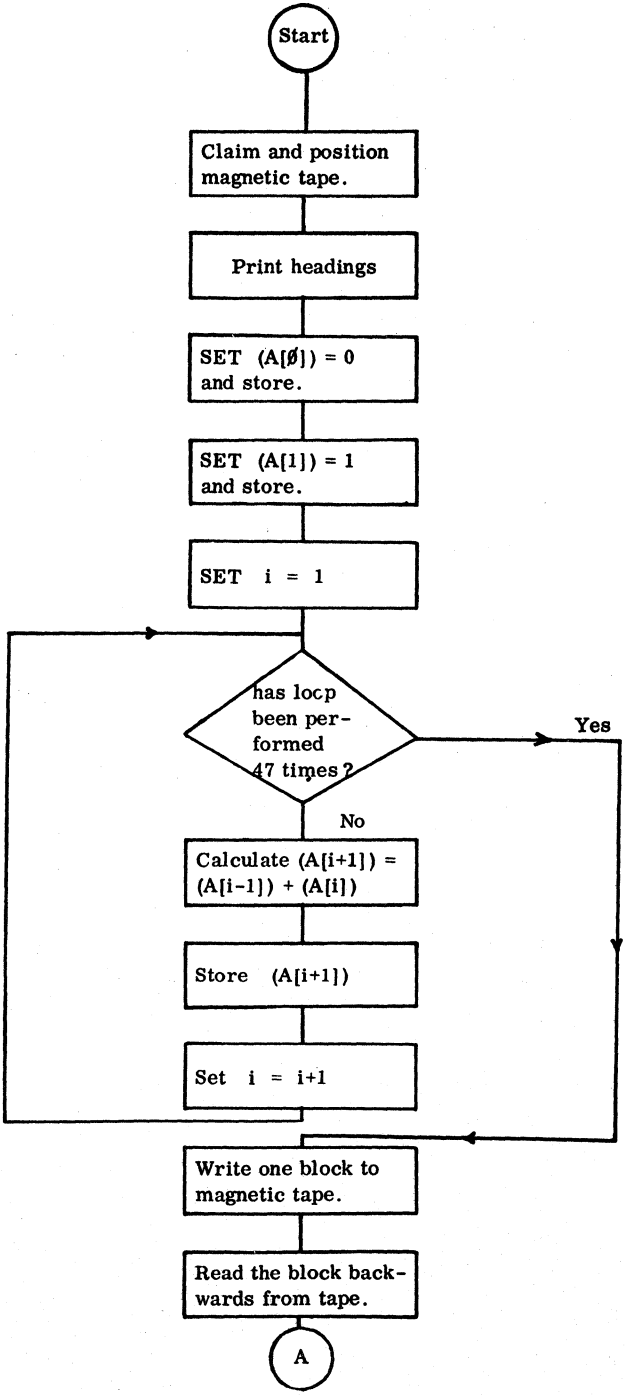
(i) At entry into L1000 there must be a Positive binary integer in N1.

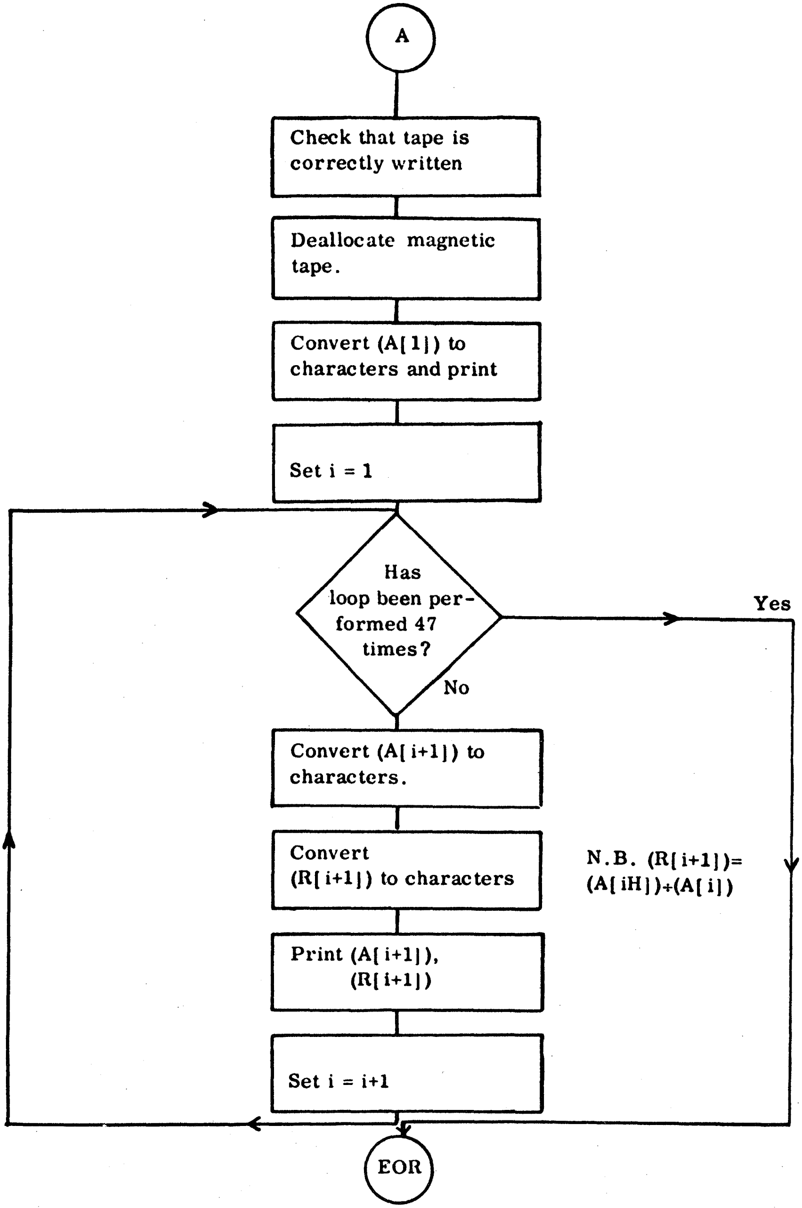
(ii) At exit from L1000 N1 and N2 will contain a double length word, N1 being the more significant half and N2 the less significant half. The double length word is the decimal character form of the binary integer. It is at the least significant end of the double length word with any more significant zeros replaced by spaces (Octal 00).

3. The tape to be used is a WORK tape (i.e., with identifier 00000000).

4. The format of the output should be:-

```
N                R
1
1                1·0000000000000
2                2·0000000000000
3                1·5000000000000
5                1·6666666666667
```


Start
Claim and position magnetic tape.
Print headings
SET (A[Ø]) = 0 and store.
SET (A[1]) = 1 and store.
SET i = 1
has locp been performed 47 times?
Yes
No
Calculate (A[i+1]) = (A[i-1]) + (A[i])
Store (A[i+1])
Set i = i+1
Write one block to magnetic tape.
Read the block backwards from tape.
A


A
Check that tape is correctly written
Deallocate magnetic tape.
Convert (A[1]) to characters and print
Set i = 1
Has loop been performed 47 times?
Yes
No
Convert (A[i+1]) to characters.
Convert (R[i+1]) to characters
N.B. (R[i+1])= (A[iH])+(A[i])
Print (A[i+1]), (R[i+1])
Set i = i+1
EOR

A[1], A[2], ... A[48] reference the 48 numbers to be calculated.
R[2], R[3], ... R[48] reference the 47 ratios to be calculated.

26· THE DIRECTOR

26·1 Basic Functions of Director

The KDF 9 System has been designed to provide

(i) a wide variety of instructions for a programmer to use as he wishes,

(ii) a selection of protective interlocks to interrupt an object program automatically if there is any reason why it should be held up, and

(iii) a special instruction to allow assistance to be given to the object program as and when required,

The first requirement is met by the USERCODE; the other two are provided by a control program, known as Director.

The computer itself organises entry to the Director program from the object program as required - this is referred to as "an INTERRUPT into Director". On such an interrupt sufficient information is available to enable Director to ascertain why it has been entered and also to allow the object program to be re-entered at the appropriate place when the reason for interrupt has been dealt with by Director.

There are two kinds of KDF 9 system: one has the facility for the Director to call in one object program to be run: the other has a special Director program which can call in up to four object programs which are

(i) placed in areas of the main store determined by operator action

and

(ii) run on a time sharing basis -

(this system requires the computer to have three extra sets of Nesting Stores, Q-stores, etc.) - part of one program is run until it is held up temporarily for some reason, when the next object program in priority is run until that is temporarily held up: this process of priority running applies to the four programs.

To allow enhancements to the facilities of Director the sizes of different versions of Director vary. Therefore it is not possible for a programmer to say exactly at which word in the main store his program starts. Any difficulties of this type are, however, avoided by combined action between Director and the electronics of the computer: the Director places in a special register the actual address of the first word of the main store allocated to the programmer and any reference to the main store by the program is automatically increased by this amount, thus enabling the programmer to assume his program starts at word zero (E0) irrespective of this size of the Director currently in use.

Only Director need therefore know how big it itself is. The contents of this special register are automatically set to zero whenever the Director is entered (its word zero is always word zero of the main store) to ensure it gets its addresses right: at the same time, restrictions on the use of certain instructions are lifted enabling the control routine to obtain access to registers inaccessible to an object program. These restrictions are reset as control is transferred back to the object program. There is no mention of these restricted instructions in this manual as they are only of interest to authors of control routines.

Director, is a program that can be read into the main store from paper tape very easily and, therefore, can be changed at very short notice, simply by reading a different paper tape. The facilities to be described in this manual represent those catered for in the current version of the Non-Time Sharing Director (October 1965) program.

26·2 Entries to Director

An entry to Director can be made for various reasons, some at the request of the program, some caused by program hold-up or failure, and the rest by Director itself in conjunction with the hardware of the computer. Let us consider the classes separately: this manual describes only those aspects of interest to the programmer and makes no attempt to describe the precise mechanism involved - the causes of each entry to Director and its resulting effect on the program are all that is of interest to a programmer.

26·2·1 Programmed Entries to Director. There are two instructions available to the programmer to call for entry to Director:-

INTQq; Interrupt if the device whose number is given in the counter position of Qq is busy. This is intended only for time-sharing machines: Director will return control to this program when ANY device used by this program becomes available, resuming at the instruction FOLLOWING INTQq;.

OUT; An unconditional entry to Director, to enable the program to utilise any special facilities built into Director.

The facility required is selected by a code number placed in the top cell of the nesting store before obeying OUT; Director will then perform the selected function, using if necessary, auxiliary parameters placed in the second c ell of the nesting store.

On completion of the function, Director will return control to the program at the instruction following OUT; unless the particular function decrees otherwise.

The various functions are known (using the code number as reference) as OUT 0, OUT 1, etc. [Note the INSTRUCTION is still OUT;]. The actions are:-

OUT 0 called by obeying OUT; with zero in the top cell of the nesting store, or with the nesting store empty. Ends program at this point:

deallocates any peripheral devices (any transfer actually in progress is stopped immediately without warning), then calls for next program. Both nesting stores are cleared, but other storage locations are untouched. Overflow and Test Register will be cleared.

OUT 1 obey OUT; with N1 = 1. Requires program number in N2 and N3. Terminates this program (but without deallocating peripherals or clearing nesting stores) then enters program whose number was given in N2 and N3, entering at the first instruction. Intended for calling subsequent sections of a multi-sectioned program. Time limit for new program set to time taken by preceding sections plus time allowed for this section. Overflow and Test register will be cleared.

OUT 2 obey OUT; with N1 = 2. Requires the Time Limit for next program i N2: expects the next program to be already in the main store. Direct ends previous program (as for OUT 0): then starts program in store a E0. Used in Compilers where the compilation is followed by obeying the compiled program. Overflow and Test Register will be cleared.

OUT 3 obey OUT; with N1 = 3. Returns to the next instruction of the progra having put the time taken so far (seconds given to 23 integral places) in N1.

OUT 4 to OUT 7 see Section 17. The other OUT's will be covered in later Sections.

26·2·2 **Unscheduled Entries to Director.** These can occur due to either:-

(a) A program attempting to use a busy input/output device or attempting to refer to a locked-out portion of main store. In either case, it serves to prevent a program from performing operations until it is safe to do so; control is returned to the program when the reason for the hold-up disappears.

(b) A program puts too many words into the nesting store or the jump nesting store (or tries to remove one more than it has put in). This is catastrophic, and Director will tell the operator so, but the opportunity for restarting the program will be offered

(c) A program attempts to obey an unrecognised instruction (as, for example, if data are obeyed as instructions) or attempts a transfer on a device indicating a parity fail Again, these are catastrophic and are treated as in (b) above.

26·2·3 Control Entries to Director. These entries are cause solely by Director and the computer between them - the program can have no influence over the but they can influence the course of a program. The entries are:-

(a) Typewriter interrupt: the only way the operator can influence the course of the machine, by pressing the INTERRUPT button on the console typewriter, Director will then call for instructions from the operator via the typewriter.

(b) Clock interrupt. This is caused by a timing device attached to the machine, causing this interruption every 1·048576 seconds. Director uses this to count how long the program has been in progress on the machine (discounting any time used by Director itself) and thus to inform the operator of the total time used by a program, or to indicate that a program is lasting longer than it was anticipated.

(c) End of Director Transfer. A purely internal matter for Director.

Other reasons for entry to Director will occur on a Time-Sharing KDF 9, but these will not influence the programmer.

It should be remembered that, apart from using one cell of the jump nesting store for a return address (this explains why programmers should never use the full set), any stores required for use by Director will be replaced before return to the main program, and, therefore, these periodic excursions into Director will have no effect on the course of a program unless a definite reason for interference is discovered by Director, in which case the operator will be informed.

26·3 Program Format after Compilation

Since Director is responsible for the initial loading of the program into KDF 9, the layout of programs after compilation is governed by Director requirements. A program is generally broken into three distinct blocks called A, B, and C, although a program on magnetic tape is likely to have additional parts dictated by the needs of magnetic tape storage and operation. The C block is further subdivided.

26·3·1 **The Program A block.** Each program starts with an A block in a standard layout. This block serves to identify the program both inside and outside the machine, using a 12-character alphanumeric reference, and also contains (if required) a title of up to 46 characters. This A block is generated by Compiler from information on the front sheet of the program (called the Program heading).

The precise layout of the A block is:-

1st Word	1st character:	Case Normal (octal 07).
	2nd character:	Carriage return line feed (octal 02)
	3rd character:	One or other of the letters P M or D (octal 60 55 or 44)
	4th character:	Carriage return line feed (octal 02)
	5th to 8th characters:	First 4 characters of the 12 character program reference number.
2nd Word	1st to 8th characters:	Last 8 characters of the 12 character program reference number.

3 to 8th Word — The program title, consisting of a carriage return line feed character (octal 02), 46 alphanumeric characters and an End Message symbol (octal 75) at the end.

Note that all characters appearing in the A block should be NORMAL CASE characters. The A block is used only to find the program for insertion into the machine at run time - it is not available to the program during running.

26·3·2 **The Program B Block**. The B block is generated by compiler from information contained in the program heading sheet, and is always 8 words long. The B block is read into words E0 to E7 of the program space, but parts of it (which are of use only to Director) are subsequently overwritten by Director with information available only at run time. We therefore have the two states of the B block - the appearance on tape and the appearance in the main store.

(a) B block on tape

word 1	Jump to first instruction of program	Blank	
word 2	Time limit in seconds	Total storage required	
word 3	Copy of first word of A block		
word 4	Copy of second word of A block		
word 5	Restart Jump 1 to failure routine	Restart Jump 2	
word 6	Spare word - left blank		
word 7	Spare word - left blank		
word 8	Marker to indicate there is more than one C block	First address of first C block	

(b) B block in Main Store at Run Time.

E0	Initial Jump as before	Blank
E1	Time limit in seconds	Total storage required
E2	Copy of first word of A block	
E3	Copy of second word of A block	
E4	Restart Jump 1 as before	Restart Jump 2
E5	Device number of tape unit from which program was read if appropriate	
E6	Spare word - left blank	
E7	Today's date DD/MM/YY	

The total storage in the less significant half of word 2 on tape may be left blank - in this case Director will insert the maximum value for the particular machine on which the program is running (this MUST be followed in if a Time-Sharing machine is used).

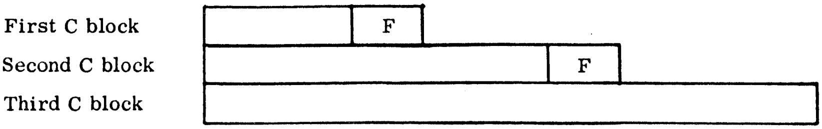
The word in word 8 is in Q-store format and is used by Director to load the C blocks. The counter position is zero if there is only one C block following and non-zero otherwise. The increment position gives the address for the first block, so Director knows where to put the block. These addresses are relative to E0, so Director will add the appropriate correction for the absolute location of E0.

If the program is read from magnetic tape, the identifier of that tape will be inserted in E5 - this enables a program to claim that tape if it intends to read from it without Director assistance.

The word in E7 is replaced by today's date by Director, for use by any program requiring it. The format is:-

2 decimal characters for the day of the month.
1 separator character.
2 decimal characters for the month of the year.
1 separator character.
2 decimal characters for the year.

26·3·3 The Program C Blocks. The program proper is contained in one or more C blocks, depending on the length of the program. Each C block contains an integral number of words of instructions: each C block except the last is followed by a filler word of similar format to WORD 8 of the B block. The B block tells Director where to put the first C block and each C block defines the locations for its successor. The diagram below shows how the filler word is removed by the following block.



When Director finds a filler word with zero counter, it knows there is only one more block to load, so no filler is needed after the last block.

26·3·4 Loading of Program Ready for Running. When Director is ready for a new program (i.e., when first read in, or when any program finishes), a read is called from paper tape. Director will read (to End Message) at least 2 words (with maximum of 8 words) and expect the first two words read in to be in the format of the first two words of the A block of a program. This defines the program to be obeyed next - the third character of the first word (PM or D) defines whether it is on paper tape, magnetic tape or the drum. If it is on magnetic tape, Director will search the program tape (asking which it is if it is not already defined) to find a program having precisely the same two words in the A block as those read from paper tape.

When the program is located, reading can commence, as the next block - be it on paper, magnetic tape or drum - is the B block for the program, followed by the C blocks. After loading the last C block, Director jumps to the first syllable of E0. The jump instruction placed here by Compiler will then go to the first instruction of the program, which is after the constants for the program. This double jump is necessary since Director has no means of telling how many constants there are, and therefore cannot find the first instruction without assistance from Compiler.

26·4 Typewriter Interruptions

These are provided solely for use by the operator to control the machine. A detailed account for each appear in KDF 9 Instructions to Operators, but the brief details are given below. When the operator presses the button labelled INTERRUPT on the console typewriter, Director will type out a message (using TWEQq;) which begins:

Case Normal
Carriage Return Line feed
Tab.
INT;

This tells the operator that the interrupt has started, but the semi-colon will change the instruction to a READ from this point onwards. The operator then types an alphabetical letter to define the particular request, followed by any additional information required, finishing up with full stop, and message (octal 37, 75). A few of the various actions available, listed under the appropriate alphabetical letters, are:-

A. End Program

B. Read 8 octal digits to least significant half of E0. These will be punched in the character code as 8 six-bit characters, compressed by Director into 24 binary digits. This word can be used to control the action of the program subsequently.

E. Define Program Tape. Used to tell Director which tape to use when loading programs.

F. Dummy - in case INTERRUPT pressed in error

G. Check states of input/output devices. Gives the operator a list of the current states of all input/output devices including tape identifiers.

H. Used by the operator to obtain list of all magnetic tapes required by Director, either for an outstanding OUT 4 request or for Director use.

I. Restart. Enables the program to be restarted by obeying one or other of the jump instructions in E4. TINT E0 causes a jump to E4 upper half; TINT I1 causes a jump to E4 lower half.

J. For control of Director pseudo-off line activities, used mainly in connection with OUT 8.

L. Used to change the type number of particular device.

M. Used to obtain a "Post Mortem" punch-out of certain main store words.

26·5
For more detailed information concerning Director the reader is referred to Section 10 of the Service Routine Library Manual.

26·6 Typewriter Log

26·6·1 Non Time Sharing. The on-line console typewriter is used by the Director program

(i) to inform the operator to perform certain actions, e.g., load a particular magnetic tape,

(ii) to request the operator to perform certain actions, e.g., load a particular magnetic tape,

(iii) to ask the operator certain questions e.g., what to do if an object program fails.

The complete list of messages which the Director can cause to be typed is listed in Section 10 of the Service Routine Library Manual: a few typical examples are listed here in the form of part of a normal log sheet.

N_{10} 22/10 1604

N_{10} M KES000602UP1

02A01→

FAILS 72

LINK→100106

SJNS→ CLEARED

N1→ 000000000000037

REACT;A →

KES000602UP1 ENDS7

N_{10} RAN/EL/000M20S/001M24S

N_{10} 22/10 1606

These are:

(i) The date followed by time.

(ii) Identifier of program being run.

(iii) Device number 01 (type number 02) has been allocated to program.

(iv) The program fails, code of failure being 72.

(v) Program failed at word $106_{(8)}$ syllable 1.

(vi) SJNS was empty.

(vii) The contents of N1 were as shown.

(viii) Because of failure, Director program types REACT; to which operator replies A. → indicating that program is to be terminated.

(ix) Program identifier followed by reason for ending.

(x) Run and elapse times for the program signifying time program in control and normal clock time respectively.

26·6·2 **Time Sharing.** On a time sharing machine the log is similar but with main difference that the log paper is wider and has five columns.

Column 1 is for messages, etc. from Director and the other four columns are used one for each level of program being run.

For further details of Director the reader is referred to Section 10 of the Service Routine Library Manual.

27• THE USER CODE COMPILER

27•1 The User Code Heading Sheet

All User Code programs except those to be processed by POST, will start with a heading sheet which contains information necessary for Compiler. Some of this information is mandatory, but some is included only if required by the particular program involved. The two classes are therefore listed separately below.

For the purpose of these lists several abbreviations are used:-

(a) CR-LF to mean Carriage Return and Line Feed (the octal character 02).

(b) <unsigned integer> to mean any positive whole number expressed in decimal (spaces before or between digits to be ignored).

(c) <letter> to mean any alphabetic letter excluding I or 0.

27•1•1 Mandatory Items on Heading Sheet

Case Normal and CR-LF	desirable for playback: ignored by Compiler.
P or M	the first printable character on the tape.
CR-LF	all characters up to this are ignored.
Twelve character identifier	any non-printables before the first are ignored. After the first, the succeeding 11 are taken.
CR-LF	any redundant characters before this are ignored.
Up to 46 printable or non-printable characters for title	these are reproduced unchanged but end message replaced by space if found.
End Message to terminate headings	last character of block made into End Message, whatever the length of the title.
ST <unsigned integer>	number of word of storage required. (Includes instruction and data space).
TL <unsigned integer>	time limit for machine code program, given in seconds.

Insert here any options required in the order listed below.

PROG;	PROG; or PROGRAM; or PROGRAMME; are alternatives in this position.

27·1·2 Optional Items on Heading Sheets

START <unsigned integer> ; program origin relocated at this address in place of zero.

Y0=E <unsigned integer> ; the address of Y0 is to be that stated (rounded up to multiple of 32).

V <unsigned integer> ; space to be left for constants up to this limit.
(N.B.) V3; implies space for FOUR constants numbered 0,1,2,3).

W <unsigned integer> ; space to be left for W stores up to this limit (see note on V stores).

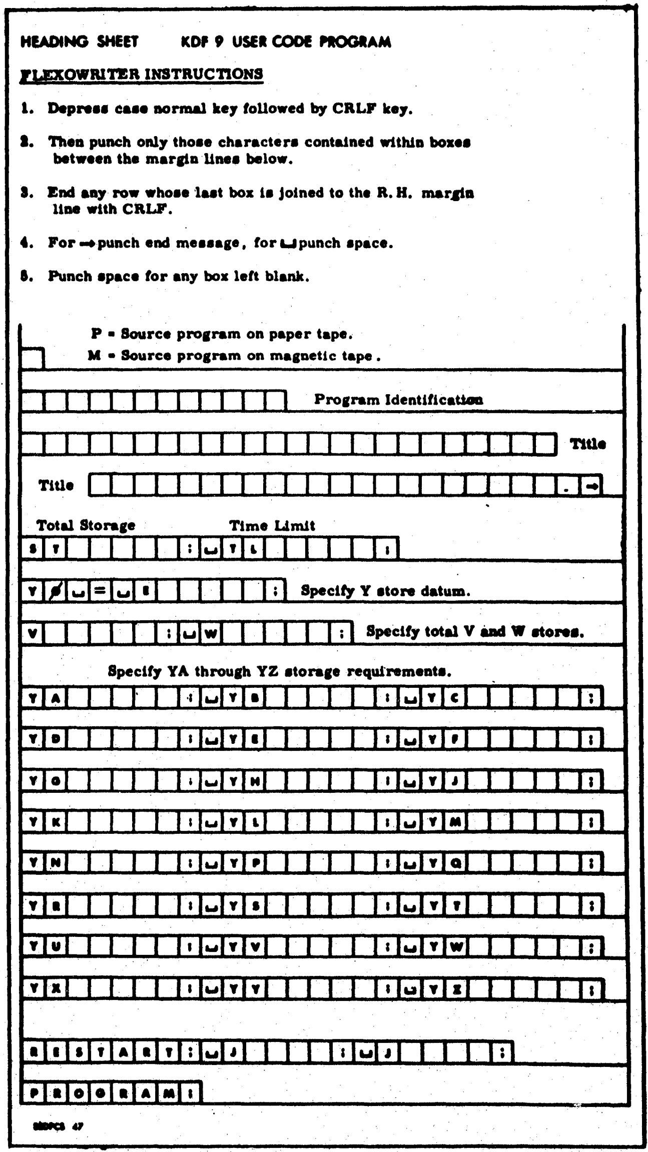
Y <letter> <unsigned integer> ; the letter defines the sub-class of data storage: the unsigned integer defines the upper limit of the sub-class. This item is repeated for all sub-classes required, and the order for such repetitions is immaterial.

RESTART; <restart sequences> ; the restart sequences can be any valid User Code instructions up to maximum of 6 syllables. These will be stored in E4 of B-block

Example of Heading Sheet Print-out (Note: CR-LF implied by new line starting).

```
P
KE123456/123
SAMPLE HEADING SHEET
ST3456;
TL180;
START256;
Y0=E1536
V15;
W2;
YZ14;
YA29;
RESTART; J5; J23;
PROGRAM;
```

27·1·3   Below is shown a reduced-size reproduction of a Heading Sheet. Such a sheet should be completed and submitted for punching with the usercode body when the normal paper tape compiler is to be used.



HEADING SHEET   KDF 9 USER CODE PROGRAM

FLEXOWRITER INSTRUCTIONS

1. Depress case normal key followed by CRLF key.
2. Then punch only those characters contained within boxes between the margin lines below.
3. End any row whose last box is joined to the R. H. margin line with CRLF.
4. For → punch end message, for ␣ punch space.
5. Punch space for any box left blank.

□ P = Source program on paper tape.
M = Source program on magnetic tape.

□□□□□□□□□□□□ Program Identification

□□□□□□□□□□□□□□□□□□□□□□□□ Title

Title □□□□□□□□□□□□□□□□□□□□□ . →

Total Storage   Time Limit

S T □□□□□ ; ␣ T L □□□□□ ;

Y Ø ␣ = ␣ S □□□□□ ; Specify Y store datum.

V □□□□□ ; ␣ W □□□□□ ; Specify total V and W stores.

Specify YA through YZ storage requirements.

Y A □□□□□ ; ␣ Y B □□□□□ ; ␣ Y C □□□□□ ;

Y D □□□□□ ; ␣ Y E □□□□□ ; ␣ Y F □□□□□ ;

Y G □□□□□ ; ␣ Y H □□□□□ ; ␣ Y J □□□□□ ;

Y K □□□□□ ; ␣ Y L □□□□□ ; ␣ Y M □□□□□ ;

Y N □□□□□ ; ␣ Y P □□□□□ ; ␣ Y Q □□□□□ ;

Y R □□□□□ ; ␣ Y S □□□□□ ; ␣ Y T □□□□□ ;

Y U □□□□□ ; ␣ Y V □□□□□ ; ␣ Y W □□□□□ ;

Y X □□□□□ ; ␣ Y Y □□□□□ ; ␣ Y Z □□□□□ ;

R E S T A R T ; ␣ J □□□□ ; ␣ J □□□□ ;

P R O G R A M ;

EDPCS 47

27·2 Layout of Store by Compiler

The various parts of a KDF 9 program will be stored in the following order. Any item preceded by an asterisk * is optional - if none is asked for, none will be allocated. Word locations cannot in general be given - they depend on the size of each section.

Words E0 - E7.	B block.
	*Main Program Constants.
	Main Program Instructions.
	*First Subroutine Constants.
	*First Subroutine Instructions.
	. . .
	. . .
	. . .
	*Last Subroutine Constants.
	*Last Subroutine Instructions.
	Unused space of from 0 to 31 words (increased if Y0 specified).
	*W stores
	*YA stores.
	*YB stores.
	.
	.
	.
	*YZ stores.
	Y0 ALWAYS stored in a word which is a multiple of 32
	Y stores - all remaining space available as Y stores.

28·1 Presentation

When a programmer has written his program body and heading sheet and had it punched on paper tape his next step is to prepare a list of instructions to inform the operator on certain points.

Each KDF 9 establishment will have its own documentation system for the orderly presentation of Operating Instructions, and programmers should familiarize themselves with whatever system is being followed at their particular installation.

A typical example of the way data should be submitted for processing on KDF 9 is described in the paragraphs which follow. The format described is that in use at the Kidsgrove Bureau of English Electric Computers Ltd.

28·2 Format

Operating instructions should be presented in the format shown overleaf. Sequentially numbered Continuation Sheets quoting the Job Number and Identifier should be used as necessary.

KDF 9 PROGRAM TESTING INSTRUCTIONS

Page	1	of	

Job No. .... Date ....

Identifier .... Programmer ....

Est. El. time .... Phone extension ....

Accurate storage, no. of words	

OUT 8 ?	Yes	No
Stream number		
Device type		
Tape edit only?	Yes	No
Printer output to magnetic tape?	Yes	No

Magnetic tape units, model 1081	
Magnetic tape units, ampex	
Maximum no. of tape units required at one time	

Printer, on-line, 120 chars/line	
Printer, on-line, 160 chars/line	
Readers, card	

Readers, paper tape	1	2
Punches, paper tape (5-hole)	1	2
Punches, paper tape (8-hole)	1	2

Usercode	W K Algol	Alphacode	Matrix	Fortran	Post no. =

Step No.	Expected actions and monitor indication	Operator actions

Serial no. field cols. =
(punch cards only)

Continued on next sheet Yes/No

Failure action

REACT STORES PRINT WORDS LIV. PM.

Operator comments

*Signature.................................

KDF 9 PROGRAM TESTING INSTRUCTIONS
(CONTINUED)

Page		of	

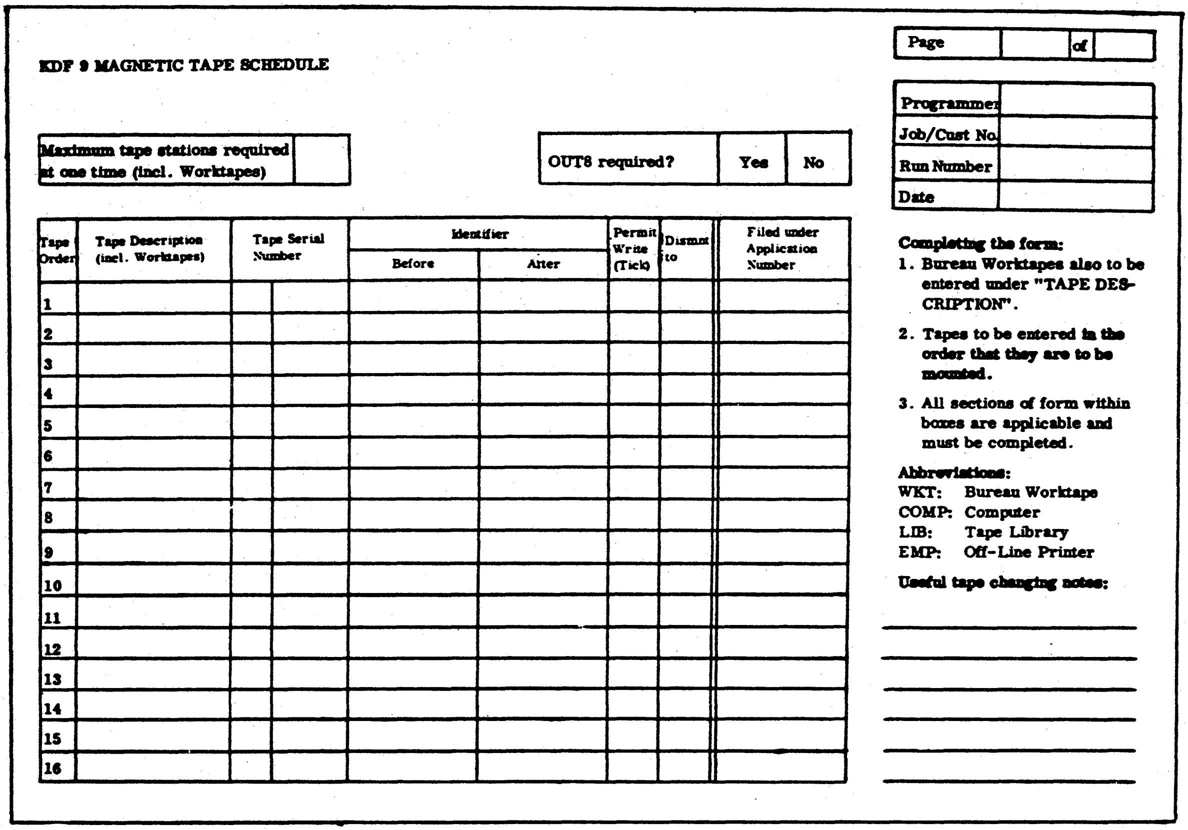
Job no.	
Identifier	

Step No.	Expected actions and monitor indication	Operator actions

Continued over Yes/No

28·3 Magnetic Tape

If magnetic tape is required the following schedule must also be completed:



KDF 9 MAGNETIC TAPE SCHEDULE

Maximum tape stations required at one time (incl. Worktapes)	

OUT8 required?	**Yes**	**No**

Page		**of**	

Programmer	
Job/Cust No	
Run Number	
Date	

Tape Order	Tape Description (incl. Worktapes)	Tape Serial Number		Identifier		Permit Write (Tick)	Dismnt to	Filed under Application Number
				Before	After			
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								

Completing the form:

1. **Bureau Worktapes also to be entered under "TAPE DESCRIPTION".**
2. **Tapes to be entered in the order that they are to be mounted.**
3. **All sections of form within boxes are applicable and must be completed.**

Abbreviations:

WKT:	Bureau Worktape
COMP:	Computer
LIB:	Tape Library
EMP:	Off-Line Printer

Useful tape changing notes:

28•4 Operating Instructions

1. Operator actions should be numbered and written on separate lines.

2. Where several operating sheets are used, they should be stapled together as well as being numbered. This avoids the possibility of instructions being obeyed out of sequence.

3. Tape schedules are best left separate from operating sheets, (i.e., not stapled) for Tape Library use.

4. Where a previous "WRITE IDENTIFIERS' (See section 29) is required, this should be filled in as the first sheet of the operating instructions.

5. Subsidiary programs such as 'TAPE EDIT' (See section 29) should also have their instructions on a separate sheet.

6. The Application number on Tape Schedules should be filled in correctly, as it is this number and not always the Job Number of the run being processed that the tapes are filed under.

28•5 Restarts (TINT I0, TINT I1, and REACT B)

1. An odd or even restart, (TINT I1 or TINT I0 respectively - see Section 26•4) is useful for failure routines such as 'STORES PRINT', or for private failure routines. OUT 1 can be used to advantage here, because with the time-sharing director, programs such as 'STORES PRINT' cannot always be performed successfully.

2. L1, L83, and L64 are useful diagnostic routines, and are available on the Ninemaster tape. (See section 29).

3. If the program fails, provision should be made to relabel back to zero any zero tapes used by the program, by means of a restart.

28•6 Use of Work Tapes

1. Zero worktapes should be relabelled with a **unique** identifier when used by a program. This avoids the possibility of the wrong tape being used for the second section of a program.

2. Tapes which have been relabelled in this manner should be zero on deallocation.

3. Time is charged for manually rewriting zero tapes which have been overwritten, and not relabelled to zero or deallocation.

4. If a tape for OUT 8 output is required, this should be entered and started as such in the Worktape requirements section.

28·7 Flexowriter

1. The on-line flexowriter should be used at a minimum, and then in OUT8 Stream 0 (So that the printout will appear in the correct column on the monitor log during time-sharing).

2. The flexowriter should not be used to any large extent as an edge-punched card reading device. The paper tape reader is designed much better for paper tape input, and is 100 times as fast.

28·8 Use of OUT5 and OUT6

1. Time-sharing operates much more efficiently with programs which deallocate their peripherals as soon as they are finished with. These devices become free for use by other programs.

2. Peripherals should be claimed when and where needed. This helps especially with programs which use several magnetic tapes, and which take a certain amount of time to load.

3. The generally-used practice of claiming all devices at the beginning of the program and deallocating them all at the end limits time-sharing considerably.

4. If programs did follow the recommended procedure, the possibility of using more levels of time-sharing shows itself, and this makes for a better turn-over of work.

28·9 Paper Tape

1. Paper tape reels should be numbered or lettered in sequenc.

2. Where there are two or more runs in the same box with their own sets of paper tapes, these paper tapes should be clearly differentiated. It does sometimes happen that there are two tapes labelled the same, or so similar that it can cause errors.

3. Particular attention should be given to splices. Splices which cause a jam in the reader are a regular occurrence and account for considerable wasted time.

4. Call tapes (i.e., the first 16 characters of an A block followed by→) when provided should have a definite amendment number and store module size. Time can be wasted searching for non-existent programs on magnetic tape.

28·10 OUT8

1. The 'END DATA' character should not be output to the OUT8 tape. This character is written by a TINT J.→ when the OUT8 tape is deallocated and it is the terminating character on the tape. Off-line operators only print as far as an ED.

2. Direct Printer Output (i.e., not via OUT8) should commence with a Page Change and Title.

3. A Heading (followed by Page change if printout is required separate) is very useful to Off-Line Services.

28·11 Time Sharing and Store Limits

1. Storage requirements should be kept to a minimum whilst program testing so that programs can be time-shared.

2. Any prog:am, which uses more than 16000 words of store cannot be efficiently time sh ıred, and because time-sharing is used continually in the Bureau, this may mear that these programs have to wait until they can be fitted in.

3. A Time-s.aring m/c reverts to a NON-Time-Sharing m/c when programs using 24,000 words of store are run.

4. In general, the storage requirements of programs using up to 14,000 words are most suitably fulfilled.

5. The more extensive use of magnetic tape as a data working medium should be explored as an alternative to using the bulk of the core store.

6. The limiting factor on a m/c with 8 tape stations and 32K of store is not by any means the number of tap Stations.

7. Generally the programs which use 24,000 words of store or more are those which do not make full use of magnetic tapes.

8. A Precise statement of storage (i.e., not to nearest 100 words etc.) could mean difference between the job being run early or not.

28·12 Time Lımits on Programs

1. An Estimated Elapsed time statement is especially important to an operator, in order that jobs can be arranged for time-sharing. Any prevıous run of the program should give an idea for an estimate.

2. For the case where the program goes into a loop etc., and the time limit within the program does not cause the run to terminate itself, a statement of the number of minutes after which the program **must** be terminated (manually) is of particular use to an operator.

3. During program testing, realistic time limits are advisable. An undetectable loop or reams of incorrect output could be avoided by a good time limit.

29· SOFTWARE

29·1 Ninemaster

The magnetic tapes containing standard programs and subroutines which are of general use are called the NINEMASTER Library Tapes, and are made available to all KDF 9 users.

At present there are two tapes to the series, one containing the routines in machine code and the other in normal programming text (viz., Usercode, Algol or Fortran).

The 4 volumes of the Service Routine Library Manual cover the details of Ninemaster, listing its contents in Section 4. Section 3 is a cross reference indicating where the specification of each routine can be found.

29·2 Normal Usage

It is normal practise for the Ninemaster tapes to be kept as Master tapes and for each department to have its own Library ape which will contain copies of routines from Ninemaster which are likely to be of use.

The "library extraction" system used in "POST" (section 30) is ideal for incorporating routines from a Library tape into a program.

29·3 Service Routine Library Manual

The reader is recommended to read the manual in the following order:

(a) The titles of all routines listed in Section 3.

(b) Library subroutine L89 (to be found in Section 16) - merely read through this write up, to gain an idea of the content of a subroutine specification.

(c) Section 1·2·2 which shows the format of each specification in the manual. (Attention must be paid to the storage requirements e.g., Q stores and W stores used by each routine, and the main program must be written to take account of these.)

(d) A detailed reading of the routines likely to be of use to the student. (i.e., commercial or scientific, etc., routines.)

29·4 Exercise

The student is now asked to prepare a flowchart and write the program to the following specification.

It is proposed to compute the solutions to the equations

$$R = \frac{1}{1+e\cos\theta}$$

$$X = R\cos\theta$$

$$Y = R\sin\theta$$

for all values of

($-1{\cdot}0<e<1{\cdot}0$ in 0·1 steps
(
($0^\circ\leqslant\theta\leqslant90^\circ$ in 10 steps

The output is to be on the line-printer allowing a 120 character width of paper and of the following format:

```
NAME HEADING                              THETA (DEGREES)

E                          0    10 . . . . . . . . . .  90

-4·0                  R   ...   ...                     ....
                      X   ...   ...                     ....
                      Y   ...   ...                     ....

-3·9                  R   ...   ...                     ....
                      X   ...   ...                     ....
                      Y   ...   ...                     ....

+4·0                  R   ...   ...                     ....
                      X   ...   ...                     ....
                      Y   ...   ...                     ....
```

The output need only specify actual valves which lie between ±10. Routines in the following list should be used L10, L59, L11.

N.B. L1000 is NOT on Ninemaster. A solution to this exercise is not given at the end of this manual.

30·1 Principle

The purpose of the POST system is to provide a means for assembling compiling and amending batches of programs on magnetic tape. The system requires, at minimum, three magnetic tape units to hold

(a) the old text tape - that currently containing source programs.

(b) the new text tape - which received updated source programs from the old text tape.

(c) the binary tape - which receives the compiled binary programs of the new text tape.

The procedure is to supply, on paper tape, a number of programs and/or corrections and certain Directives specifying, among other things, the action to be taken by the machine. The machine then enters an Assembly Phase when new programs (from paper tape), old programs (from old text tape) and amended old programs (from old text tape - corrected as per paper tape) are written onto the new text tape. Next comes the Translation Phase when specified programs on the new text tape are compiled and written onto the binary tape. This tape is now ready for the programs to be called down for running.

30·2 Postmaster Directives

Control of POST is by means of Directives contained on the input paper tape.

The format of these Directives is as follows:-

(a) ASSEMBLE;FROM <ident> ;
    ONTO <ident> EX <ident>;
    P;L;8;5; →
    followed by "Assembly Phase Messages", (see section 30·3)

(b) TRANSLATE;FROM <ident>;
    ONTO <ident>EX<ident>;
    Md;
    P;L;8;5; →
    followed by "Translate Phase Messages", (see section 30·5)

(c) END POST→

Notes:

Block (a) is mandatory whenever correction or establishment is required.

Block (b) is mandatory whenever compilation is required.

Blocks (a) and (b) when used, must be followed by appropriate Phase Messages.

Blocks (a) and (b) may be repeated as often as required and appear in any order.

Block (c) the directive to terminate the run and deallocate all peripherals is mandatory.

P;L;8;5; → indicates that an off-line Printer
an on-line Printer
an 8-hole Punch
a 5-hole Punch

will be required (not all four are mandatory, but at least one must be quoted, those required are to be placed in order preference.

FROM<ident>; specifies the label of the magnetic tape to be input for the run (old text tape).

ONTO<ident> specifies the label to be written to the output tape. (new text tape for assembly phase, binary tape for translate phase).

EX< ident>; an optional - specifies the present label of the tape to be used for output. If this parameter is omitted a work tape is used.

<ident> is any 8 or 16 character tape label as in Section 22•2 of the Service Routine Library Manual.

Md; indicates number of magnetic tape units available to the system, "d" must be 3, 4 or 5. This count does not include the off-line printer tape. If off-line printing is requested, the label +POSTOFF is written to a zero work tape which is used for this purpose.

30•2 Assembly Phase Messages

(a) $\underline{\rightarrow}$ ESTABLISH program identifier;
program title;

0/P 5, 8, L; →
followed by a source program
$\underline{\rightarrow}$

Notes:

$\underline{\rightarrow}$ ESTABLISH indicates that the following source program is to be established on magnetic tape from paper tape.

0/P 5, 8, L → indicates 5-hole, 8-hole punch and line printer. These should appear in order of preference. At least one device must be specified.

$\underline{\rightarrow}$ indicates end of source program. (→ and underline must be in case normal)

(b) → CORRECT program identifier;
RENAME new identifier;
new title;
0/P 5, 8, L; →
followed by "Correction texts", (see section 30·4).
→

Notes:
→ CORRECT indicates that the source program on the existing tape is to be corrected,
RENAME is optional, indicating the new identifier and title to be given to the amended program.

Whenever a program is amended by a "CORRECTION" the POST system automatically updates the amendment number of the program's identifier. The 8th and 9th characters of the 12 character program identifier are the amendment number which should be 00 when the program is originally established.

(c) → DELETE program identifier →
An optional - indicates that the program is NOT to be copied to the new text tape.

(d) → TRANSFER program identifier →
An optional - indicates that the program is to be copied.

(e) → COPY REST →
Causes all unprocessed programs, other than those transferred, to be copied to the new text tape.

(f) → COPY NONE →
causes no unprocessed programs, other than those transferred, to be copied to the new text tape.

Note Blocks (a) to (d) are optional and may appear in any order. Either block (e) or (f) must be the last of the Assembly Phase Messages.

30·4 Correction Texts

Corrections of source programs, to be applied under the Control of Assembly Phase Messages (section 30·3) must conform to the following rules. They must be given in the order in whch they will be processed assuming a single scan through the source text, starting at the first line and must be in one of the forms given below.

30·4·1 Copying Facilities

AFTER n LINE(S) →	copies the next n (decimal digits) lines of text.
AFTER LINE →	copies up to, and including, the next line which starts with the indicated characters.

30·4·2 Deletion Facilities

DELETE N LINE(S) → deletes the next n lines of text.

DELETE TO AFTER LINE character →

DELETE TO BEFORE LINE character →

30·4·3 Insertion Facilities

INSERT text → copies the given text, which may be of any length, layout being preserved. The final → is not preserved.

30·4·4 Other Facilities

CHECK character → causes the next line to be examined to see whether it begins as indicated. If the check is not satisfied, the item fails.

CONNECT → cause the following lines of text to be printed on the device selected from the Correct 0/P list (section 30·3 (b),), UNTIL:-

DISCONNECT → causes the previous CONNECT → instruction to be cancelled. If this option is not included then, at the end of corrections for this particular program, the device will be automatically deallocated.

3·4·5 Library Extraction The student should now read Appendix 6 of Section 7·3 of the Service Routine Library Manual.

It will ne noticed that **library** .....; can be included anywhere in the program. If it occurs other than just before FINISH;→the format should be

;A;B;Jr; **library** ....;P0;r;C;

The reason for this is that the actual text of the library subroutines will be substituted for **library**....; in which case, were the jump instruction not included the instruction to be obeyed after instruction B; would be the first in the library subroutine. But entry to the subroutine must be by JSL....; Using the format above ensures that after B; control jumps to r; of the main program (P0) and so obeys instruction C; next. This complication does not occur if **library** is placed just before FINISH;→and it is advised that it should always in general be in that position.

Also pay particular attention that DELETE **library**..;→is NOT a valid correction message. The method to use when requiring to alter the library call is clearly explained in the specification.

30·5 Translation Phase Messages

(a) PROGRAM program identifier;
or
SEGMENT program identifier;
Indicates the program or segment to be compiled.

(ii) Compiler; This must be either USERCODE;
or KIDSGROVE;
or WHETSTONE;
or FORTRAN;

(iii) With Tables; Indicates that reference tables are to be output to the first available device specified in (vi) below. There are other items which may be used, a full list of which is given in the SRLM.

(iv) STdddd; This option indicates the store limit dddd to be assigned to the compiled program. If omitted the store limit of the translator is assigned.

(v) TLeeee; This option indicates the time limit eeee (seconds) to be assigned: if omitted a limit of 120 seconds is set.

(vi) O/P5, 8, L;→ (details as in Section 30·3(i)).

(vii) END TRANSLATION→

blocks (i), (ii), (vi), (vii) are mandatory.

30·6 Example of a POST run

```
M
KAB000501UP3
ASSEMBLE;FROM <POSTOU12>;
ONTO <POSTOU13> EX<POSTOU13>;
L;8;→
→CORRECT KEZGL0403UP1;
0/P L, 8;
→CONNECT
→INCLUDE LIBRARY
→AFTER LINE V11=
→DELETE 1 LINE
→INSERT V12 = P⌊7DC⌋;

⇉
→ESTABLISH KESA00100UP1;
SAMPLE;
0/P L, 8;
V23;
PROGRAM;
               V0  =  B30;
               V1  =  etc.,
                   ........
                   .........;J9;
library L59
FINISH;
→

⇉
→DELETE KEFAB0019UP1
→COPY REST
TRANSLATE;FROM<POSTOU13>;
ONTO <POSTOU11> EX<POSTOU11>;
M4;
L;8; →
PROGRAM KEZGL0403UP1;
USERCODE;
WITH TABLES;
ST 1024;TL3;
O/P L, 8 →
PROGRAM KEDATAA04AP2;
KIDSGROVE;
0/P L→
END TRANSLATION →
END POST →
```

Chapter 2

1· (a) $157_{(10)} = (1\times10^2)+(5\times10^1)+(7\times10^0)$

(b) $29\cdot5_{(10)} = (2\times10^1)+(9\times10^0)+(5\times10^{-1})$

(c) $157_{(8)} = (1\times8^2)+(5\times8^1)+(7\times8^0)$

(d) $35\cdot4_{(8)} = (3\times8^1)+(5\times8^0)+(4\times8^{-1})$

(e) $101_{(2)} = (1\times2^2)+(0\times2^1)+(1\times2^0)$

(f) $1101\cdot01_{(2)} = (1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)+(0\times2^{-1})+(1\times2^{-2})$

2· (a) $45_{(10)}$

(b)
```
8) 45
8)  5r5
    0r5
```

Ans. $55_{(8)}$

(b)
```
2) 45
2) 22 r  1
2) 11 r  0
2)  5 r  1
2)  2 r  1
2)  1 r  0
    0 r  1
```

Ans. $101101_{(2)}$

N.B. The answer in binary could have been written down directly from the octal representation:

$55_{(8)} = 101\ 101_{(2)} = 101\ 101_{(2)}$

3· (a) $16\cdot5_{(10)}$

(b)
```
8) 16          ·5
8)  2 r 0       8
    0 r 2     4·0
```

Ans. $20\cdot4_{(8)}$

(b) Taking answer directly from $20\cdot4_{(8)}$

we have $010\ 000\ .\ 100 = 10000\cdot1_{(2)}$

4. (a) $634{\cdot}23_{(8)} = 110\ 011\ 100\ .\ 010\ 011_{(2)} = 110011100{\cdot}010011_{(2)}$

(b) $2{\cdot}1_{(8)} = 010\ .\ 001 = 10{\cdot}001_{(2)}$

(c) $100{\cdot}001_{(8)} = 001\ 000\ 000\ .\ 000\ 000\ 001_{(2)} = 1000000{\cdot}000000001_{(2)}$

5. (a) $101\ 101{\cdot}101_{(2)} = 101\ 101\ {\cdot}\ 101_{(2)} = 55{\cdot}5_{(8)}$

(b) $1101{\cdot}1101_{(2)} = 001\ 101\ {\cdot}\ 110\ 100_{(2)} = 15{\cdot}64_{(8)}$

(c) $1{\cdot}1_{(2)} = 001\ {\cdot}\ 100_{(2)} = 1{\cdot}4_{(8)}$

6. (a) $276_{(8)}$

$167_{(8)}$ $\quad 6+7+2_{(10)} = 15_{(10)} = 17_{(8)}$: put down 7 carry 1.

$32_{(8)}$ $\quad 7+6+3+1_{(10)} = 17_{(10)} = 21_{(8)}$: put down 1 carry 2.

517 $\quad 2+1+2_{(10)} = 5_{(10)} = 5_{(8)}$: put down 5 carry 0.

21

Answer $517_{(8)}$

(b)
$$\begin{array}{r} 12{\cdot}43 \\ 762{\cdot}4 \\ 0{\cdot}13 \\ \hline 775{\cdot}16_{(8)} \\ \hline \end{array}$$

(c) $36{\cdot}27_{(8)}$ $\quad 7-7 = 0$

$15{\cdot}37_{(8)}$ $\quad$ 2-3 will not go; borrow 1 (which when brought over = 8)

$20{\cdot}70_{(8)}$ $\quad 8+2-3_{(10)} = 7$

pay back the 1

$6-(5+1) = 0$

(d)
$$\begin{array}{r} 521{\cdot}63_{(8)} \\ 43{\cdot}67\phantom{_{(8)}} \\ \hline 455{\cdot}74_{(8)} \\ \hline \end{array}$$

7. (a) $10000{\cdot}1000_{(2)}$

(b) $1010101111010{\cdot}00011_{(2)}$

(c) $101010{\cdot}1101001_{(2)}$

(d) $110{\cdot}1_{(2)}$

Chapter 3

1. (a) 41 42 43 44 23 24 34 75

(c) 32 63 32 64 32 57 32 60

2• (a) £1 ␣9s

(b) AB
ab

(c) <u>NO</u> ␣→␣

3• Of the 95 bits, D0-D94, D0 is the sign digit. To hold an integer the binary point is placed after D94. We can use D1-D94 to hold the integer, i.e., 94 bits.

With 94 bits all holding 1's we have the largest positive integer possible.
1 added to this makes 1 followed by 94 0's = 2^{94}.
Answer $2^{94}-1$.

4• (a) 11 integral places implies that the programmer imagines the binary point to be after D11. The binary pattern for +16 is 10000.

Hence the the 12 bit word contains $\underbrace{0\ 00}_{0}\ \underbrace{000}_{0}\ \underbrace{010}_{2}\ \underbrace{000}_{0}$
Written in octal this is

(b) Using the equation $x=(-s+f)\times 2^{P}$ and substituting the known quantities we have

$-16 = (-1+f)\times 2^{4}$.

But $2^{4} = 16$, hence f = 0.

So that the 12 bit word contains 100000000000

i.e., 4000 in octal.

(c) $-0\cdot 575_{(10)} = (-1+f)\times 2^{0} = -1+f$

Therefore $f = +0\cdot 425_{(10)}$

Convert this octal

$$\begin{array}{r} \cdot 425 \\ \underline{8} \\ \underline{3}\cdot 400 \\ \underline{8} \\ \underline{3}\cdot 2 \\ \underline{8} \\ \underline{1}\cdot 6 \\ \underline{8} \\ \underline{4}\cdot 8 \\ \underline{8} \\ \underline{6}\cdot 4 \end{array}$$

(Since it is only a 12 bit word, and each octal digit uses up 3 of these bits, we really only need to calculate to 4 octal digits. However it is better to go to 5 for purposes of rounding up if necessary).

$$\text{Therefore } \cdot425_{(10)} = \cdot33146_{(8)}$$
$$= \cdot011\ 011\ 001\ 100\ 110_{(2)}$$

s=1; so the contents are 1 01 101 100 110 (no rounding necessary).

Answer: 5546

(d) $31\cdot5 = (-0+f)\times 2^5$
$= f\times 32$

therefore $f = 31\cdot5/32 = 0\cdot984375_{(10)}$

Convert to octal:-
$$\begin{array}{r} 0\cdot984375 \\ \underline{8} \\ \underline{7}\cdot875000 \\ \underline{8} \\ \underline{7}\ 000 \end{array}$$

$$31\cdot5/32_{(10)} = \cdot77_{(8)} = \cdot111111_{(2)}$$

s = 0, so the contents are 0 11 111 100 000

Answer: 3740

(e) $-0\cdot09375 = (-1+f)\times 2^{-3} = (-1+f)/8$

Therefore $f = +0\cdot25_{(10)} = \cdot01_{(2)}$

s = 1, so the contents are 101 000 000 000

Answer: 5000

(f) As this is required as an integer we require the point to be after D11 (in a 12 bit word).

$1865_{(10)}$ converted:-
$$\begin{array}{rl} 8/\ \ 1865 & \\ 8/\ \ 233 & r\ 1 \\ 8/\ \ 29 & r\ 1 \\ 8/\ \ 3 & r\ 5 \\ 0 & r\ 3 \end{array}$$

$= 3511_{(8)}$

$= 011101001001\cdot_{(2)}$

Hence contents are 3511

(g) For maximum precision $-0\cdot0001_{(10)} \leqslant 2^p$

therefore p has to be -13

N.B. $2^{-13} = \cdot 000122\ldots\ldots_{(10)}$

$2^{-14} = \cdot 000061\ldots\ldots_{(10)}$

$-0\cdot 0001_{(10)} = (-1+f) \times 2^{-13}$

$f = 1-(\cdot 0001\times 2^{13})$ $[2^{13} = 8192.]$

$= 0\cdot 1808_{(10)}$

Convert to octal $= \cdot 13444_{(8)}$

$= \cdot 001\ 011\ 100\ 100\ 100_{(2)}$

$= f$

s = 1, so word contains 1 00 101 110 010
Which in octal is 4562

5· **(a)** $16<2^5$ Therefore p=5 (N.B. 16 is not **less** than 2^4)

$+16 = (-0+\frac{1}{2})\times 2^5$

Pattern is 01000010110000..........

(b) $|-16| \leqslant 2^4$ Therefore p=4

$-16 = (-1+0)\times 2^4$

Pattern is 1100001000000........

(c) $-0\cdot 575 \leqslant 2^0$ Therefore p = 0

From question 4c we see that f = ·011011001100110........

So that the standard floating point pattern is
1 10000000 011011001100110........

(d) $31\cdot 5 < 2^5$ Therefore p=5
From question 4d, f=·111111

Answer is 0 10000101 1111110000........

(e) It is explained in the text that floating point zero is always set the same as fixed point zero.

Answer is 48 0's

(f) $-0\cdot 09375 < 2^{-3}$ Therefore p=-3

From question 4e, f=·01

Answer is 1 01111101 0100000........

Chapter 6 ·

1· **Set 1**

(a) Y0 = E512
Y19 = E531

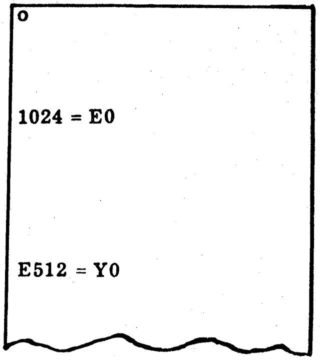
(b) E0 = 1024
Y0 = E512 = 1536

(c) E512 = Y0
E517 = Y5

(d) Y0 = E512
Y32 = E544

(e) Y0 = 1536
Y32 = 1568

(f) YA5 = 1236
YA0 = 1231
1024 = E0
1231 = E207
YA0 is· E207 (or 1231).



2· This first V store declaration of any program, V0, is ALWAYS placed in E8.

3· E12 is the same as V4
V4; =E12; is the same as V4;=V4;
i.e., the effect is to fetch a copy of the contents of V4 into the top cell of the nesting store and then overwrite what is in V4 with what is in the top cell of the nesting store - resultant effect is Nil.

4· 31, because if the gap were greater than or equal to 32 the data area would be pushed up as far as possible a multiple of 32 words.

5· No, because 1183 is not a multiple of 32 and E0 is always placed at the beginning of a line.

1· **Set 2**

E0 = 2048
E544 = 2592 = Y0
Y35 = 2627

main store word address 2627 is also Y35.

$128_{(10)} = 200_{(8)}$

```
8) 128
8)  16 r 0
8    2 r 0
     0 r 2
```

Since the number is held to 47 integral places, the point comes after D47. Hence the contents of the word, expressed in octal is 0000 00 00 00 00 02 00.

2· For Standard floating point the value of p has to be the algebraic minimum To find this, for positive numbers, we use $128 <_2 p$.

Try $p = 7 \quad 2^7 = 128$.

Is $128 < 2^7$, answer No.

So increase p by 1, $2^8 = 256$

Is $128 < 2^8$, answer Yes.

Hence p must be 8.

We now need to find s and f from the equation:-

$128 = (-s+f)\times 2^8$

s is 0 because the original number, 128, is positive.

$128 = f\times 256$

$\frac{128}{256} = f \qquad \therefore \quad f = 0{\cdot}5_{(10)}$

$\qquad\qquad\qquad\qquad = 0{\cdot}1_{(2)}$

Hence $s = 0; f = {\cdot}1000_{(2)}; p = 8_{(10)} = 1000_{(2)}$

So, in standard floating point, 128 is

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10......			
0	1	0	0	0	1	0	0	0	1	0	0	0......	
	2			1			0			4			0......

Answer: 2104000000000000

(N.B. The full 48 bits (16 octal digits) must be shown).

Chapter 7

1· Set 1

During the compilation of the Usercode program into machine code the declaration V7 = 4; is converted into the integer 4, as a binary number held to 47 integral places which is placed into the position reserved for V7 (i.e., E15). This is not an "instruction" and so will not be obeyed at run time. The way to obtain the constant during run time is by the instruction V7; which fetches the contents of V7 into the nesting store.

When the instruction SET4; is encountered during compilation a 3 syllable (24 bit) machine codt equivalent is generated - the first 8 bits are the machine code equivalent of SET and the latter 16 bits will hold the integer 4. This is an instruction which, when obeyed during run time, causes a copy of the last

16 bits of itself to be entered into the least significant 16 bits of the top cell of the nesting store.

2· (a) V0 = 2·3$_{10}$+5/18;

(b) V1D = 17$_{10}$20; or 27$_{10}$20/94;

(c) V3D = F32·2;

3· V3 = B0702/11;

4· (a) V7 = AW3;

(b) V8 = AV16;

(c) V9 = AR12;

(d) V10 = AYA0L;

(e) 129 as a binary integer held to 47 integral places.

5· (a) Valid

(b) Not Valid; 32768>32767.

(c) Not Valid 1,024 should be 1024

(d) Valid provided the equivalent E address of Y(-92) is not negative - this means that all absolute addresses must be positive.

(e) Valid

(f) Not Valid; a reference specifies a "point" and so cannot have an Upper or Lower half.

6· (a) Not Valid; 8 is not an octal digit.

(b) Valid

(c) Not Valid; 17·5 cannot be held in 4 binary integral places.

(d) Valid

(e) Not Valid. A half length word only has 24 bits so that, at most, only 8 octal digits can follow the B.

(f) Valid; there is no reason why the lower half of V2 (i.e., the lower half of E10) should not contain the integer 21 (21 is the half length address of E10L).

7· (a) Valid

(b) Valid

(c) Not Valid; the *, representing a space, does not require [ ] around it.

(d) Not Valid; should be V15/16=......

(e) Valid.

(f) Valid

(g) KDF 9 PROG. EX.

8· (a) Valid

(b) Valid; equivalent to SET - 163;

(c) Valid

(d) Valid

(e) SET Bt/p; is not valid.

(f) SETz; is not valid. The correct version of this is SET 1700;

(g) Not Valid; should be SET AY40;

(h) Valid

9· V4 = B1/2;

Set 2

1· For maximum precision p must be algebraic minimum:-

$\frac{1}{10} < 2^{p}$ $\qquad$ $\frac{1}{10} < 2^{-3}$ $\qquad$ (i.e., $\frac{1}{10} < \frac{1}{8}$) $\qquad$ p = -3.

V3D = 0·1/-3;

2· The binary point is to be considered as being just to the right of D7 making the contents of the word.

011010110011011000......

V5 = B32633/14;

3· A SET instruction cannot be of the form SETz/p;

$3{\cdot}5_{(10)} = 11{\cdot}1_{(2)}$; this, held to 43 integral places gives 00.....000111000 ← D47

Answer: SETB70; or SET56;

Chapter 8

1· **Set 1**

(a) =Q1;=+Q1;=+Q1;

(b) 1. DC5;DC5;

2. SET-2; =+C5;

3. SET5; =C5;

(c) 1. SET64;=RC1;

2. SET3;=C1;SETAY0;=I1;SETAY64;=M1;

3. SET-2;=RI1;

4. Q0;=RC1;

5. SET74;=RC1;M+I1;

2· 1. IM1T0Q3; C0T0Q3;

2. Q2T0Q3;I3=-2;

3. Q2T0Q3;DC3;NC3;

4. Q2T0Q3; NC3;DC3;NC3;

5. Q0T0Q3;I2T0Q3;M+I3;

6. I1T0Q3;C2T0Q3;

Set 2

1· Y0 = E320, ∴ Y3 = E323.

AY3 = 323

AE4U = 8

The contents of Q5 after the two given instructions will therefore be:

323/-1/8

$503_{(8)}/-1/10_{(8)}$

```
8 ) 323
8 )  40 r 3
8 )   5 r 0
      0 r 5
```

$0000000101000011_{(2)}/1111111111111111/0000000000001000$

Expressed in octal this is 0024177777600010

Answer: V7 = B0024177777600010;

(N.B. It is to be emphasized that in practice a Q Type constant would be used as V7 = Q AY3/-1/AE4U;)

(b) The only difference between this and the previous question is that the modifier is to contain AE4 (i.e., 9) instead of AE4U (i.e., 8).

Answer V7 = B0024177777600011;

(c) Y0 = E320, $\therefore$ E322 = Y2

$100_{(10)} = 144_{(8)}$

```
8 ) 100
8 )  12 r 4
8 )   1 r 4
      0 r 1
```

E322 is therefore to contain

```
                        ↓
.....00001100100         000000000000000
               ↑
              D32
```

Breaking this down to its 3 groups of 16 bits this becomes

c = 0
i = $50_{(10)}$
m = 0

Answer: V8 = Q0/50/0;

2. (a) 63/9/13

(b) 63/9/11

(c) 63/9/523 (Note that the digits after the B are 'octal', and that any spaces are ignored)

(d) 63/9/5

(e) 63/9/11

(f) 63/9/13

(g) 63/19/3

(h) 73/9/3

3. 0/0/0

4. 0/2/0 (Note that after the instruction =RM5; Q5 will be 0/1/-1, and when = +Q5; is obeyed the process is to bring all of Q5 into N1, N1 and N2 are added (assuming the bits form a fixed point integer), and then the result is placed back into Q5).

Chapter 9

Set 1

```
1·  PERM;

2·  CAB;

3·  REV;DUPD;REV;
```

Set 2

```
1·  V0  =  F19·625;
    V1  =  Q AY30/16/AY15;
    V0; =Q5; SETB235;=Q7;
    V1;=Q11;I11;=RC9;DC9;
    SET6;=I9;SET128;=M8;
    CI9TOQ8;CI9TOQ10;C11;=M10;C10;=+M10;
```

Chapter 10

Set 1

```
1·  Q1; =Y0;

2·  M0M1; =M0M2;     or  E0M1; =E0M2;

3·  (a)  SET32; =RC1;
         1; Y0M1; =YA0M1;M+I1;DC1;J1C1NZ;.....

    (b)  SET32; =RC1;
         1; Y0M1;=YA0M1Q;J1C1NZ;.....

4·  =Q1;J1C1NZ;I1;=RM2;I1=+1;
    2; M0M1Q;=M0M2Q;J2C1NZ;.....
    1;......

5·  SET31;=RC1;SET3;=I1;I15=+1;
    1; Y0M1;=M0M15Q;Y1M1Q;=M0M15Q;J1C1NZ;.....
```

Set 2

```
1·  Q1;=+Q2;Q2;=E256;
```

2· (a) Y35 will contain the integer $707_{(10)}$ held to 47 integral places.

(b) No overall effect on Y35.

(c) Q5 becomes 27/680/680 and a copy of the contents of Y707 will be placed in Y35.

(d) No overall effect on Y35, Q5 becomes 27/680/680 and M4 becomes 27.

(e) no overall effect on Y35, Q5 becomes 27/680/680 and M4 becomes 27.

(f) Q5 becomes 27/680/680 and M4 becomes 680.

3· (a) No overall effect on E7, but M6=7 and M5=7.

(b) E7 will contain a copy of the contents of Y7,
N1 will contain integer 327 and M5=7 and M6=327.

(c) Y7 will contain a copy of the contents of E14,
N1=327, N2=320, M6=14,

(d) Y7 will contain a copy of contents of Y14,
N1=7, N2=320, M5=327, M6=7.

4· Nesting store empty.
Y2 contains a copy of contents of Y1, and Y3 a copy of Y42.
Q1=93/0/0, Q2=AY93/94/0, Q5 -1/0/-1.

Chapter 11

Set 1

```
1·  SET6;=RC1;
    Y0M1Q;STR;
    1; DUPD;+D;Y0M1Q;STR;+D;J1C1NZ;=Y6;=Y7;

2·  SET32;=RC1;ZERO;STR;
    1;YB0M1;STR;YA0M1Q;STR;-D;+D;J1C1NZ;
    CONT;.............
    2;

3·  DUPD;REV;SIGN;=M5;

4·  DUPD;AND;NEV;NEV;

5·  BITS;NOT;SET1;AND;=C1;

6·  (a)  20 20 20 23 22 27 26 30
    (b)  00 00 00 03 02 07 06 10
    (c)  00 00 00 00 00 10 00 00

7·  V0=B1212121212010212;
    V0;REV;TOB;

8·  V0=B1717171717171717;
    V1=B 1212010612010612;
    V0;AND;V1;REV;TOB;
```

```
9·   V0=B1212121212121212;
     V0;REV;FRB;
     11 11 11 11 11 11 11 11

10·  V0=B 1212121201040102
     V1=B 2020202020202020;
     V0;REV;FRB;V1;OR;
```

Set 2

1· As for answer to question 10 Set 1 except that V1=B2020202000200020;

2· (a) The original integer +1
   (b) The original integer -1

3· V0=P12345678; sets the Contents of V0 (expressed in octal) as
2122232425262730
V0=B0102030405060710 sets the Contents of V0 (expressed in octal) as
0102030405060710
So that the first is the character form of the digits and the second is the binary coded form of the digits.

Chapter 12

Set 1

```
1·   V0  =  Q0/AYP0/AYP19;
     V1  =  B30;
     V2  =  P ⌊7DC⌋ ;
     V1; =YP0;YA47;=YP1;
     SET 16;=RC1;ZERO;

     1;DUP;=YP2M1Q;J1C1NZ;
     YR31,=YP18;
     V2;=YP19;
     V0;SET8;OUT;

2·   V0  =  B10;
     V1  =  Q4095/-1/120;
     V2  =  P ⌊NC6D⌋;
     V3  =  Q 0/AYP0/AYP1;
     V4  =  Q50/2/0;
     V5  =  Q0/AYP0/AYP3;

     V0;=YP0;V1;=YP1;
     V3;SET8;OUT;V4;=Q1;

     1;V0;=YP0;V2;=YP1;YA1M1;=YP2;
     YA2M1Q;=YP3;SET8;OUT;J1C1NZ;
     V0;=YP0;V1;=YP1;V3;SET8;OUT;
```

Set 2

1· V0=9; is just the same as V0 = B11;

V1=P⌊p7D⌋; is just the same as V1 = Q4095/-1/-1
Hence the effect of the instructions is to cause a punch gap of 12 inches using stream number B11.

(N.B. After the OUT8 V0 will have been overwritten).

2· V0 = P⌊7S⌋/; is just the same as V0 = B17;
V1 = B0753444677003137; the same as
V1 = P⌊N⌋ KDF ⌊D⌋ *9. ;
V2 = B2000011; is just the same as V2 = Q0/AV0/AV1; because V2 = really contains Q 0/AE8/AE9 or in other words 0/8/9 which is (in binary)
16 0's/12 0's 1000/12 0's 1001, which is in octal 000000002000011.

Hence the effect is to cause the contents of V1 to be punched on the paper tape punch using stream B17 which is in V0. (N.B. SET10; is the same as SET 8;).

Chapter 13

Set 1

1·
```
SET1024;=RC4;
Q0TOQ1;=1;Q1TOQ2;
Q1TOQ3;

5; Y0M4Q;DUP;J1>Z;J2=Z;DC3;J4;
1; ERASE;DC1;J4
2; DC2;J4;
4; J5C4NZ;NC1;NC2;NC3;
```

Note the DUP; after Y0M4Q; this is necessary because even if a jump does not occur on J1>Z; the original contents of N1 will be erased by they are still needed for the next instruction J2=Z; Should the jump occur on J1>Z then it is (in this case) necessary to have ERASE; after 1;

2·
```
SET32;=RC1;
Q0TOQ2;I2=1;Q2TOQ3;Q2TOQ4;

5; Y0M1Q;DUPD;-;DUP;J1=Z;J2<Z;=YB0M4*;J4;
1; ERASE;ERASE;DC2;J4;
2; =YA0M2Q;
4; J5C1NZ;ERASE;C3;NEG;C4;NEG;C2;NEG;
```

Set 2

1· SET2448;

2· Had we used SET-1; all the 48 bits of N1 would have been 1's.

Note that the SET instruction first sets up the least significant 16 bits of N1 **and then** puts a copy of D32 into D0-D31. Hence it is **not** possible to set D0-D31 as 0's and D32-D47 as 1's by a set instruction.

Chapter 14

Set 1

1· Y4;JSL1000;=YP1;=YP2;
The call message **just before** FINISH; →
will be

```
library L1000;
(L1000);
```

Set 2

1·
```
V0=P⌊7D⌋ +;
V1=P⌊7D⌋ -;
V2/3 = P*140737488355328;
V4    = P⌊7S⌋ 0;
    Y4;DUP;J1  Z;V0;REV;DUP;J2=Z;

3; JSL1000;J4;
1; V1;REV;VR;NEG;J3NV;
   ERASE;V3;V2;J4;
2; V4;REV;
4; =YP1;=YP2;=YP0;......
```

Chapter 16

```
V0  =  B30;
V1  =  P NAME ⌊3DC⌋;
V2/3  =  P POWERS*OF*TWO⌊2DC⌋ ;
V4  =  P ⌊7DC⌋;
V5  =  Q0/AYP0/AYP1;
V6  =  Q0/AYQ0/AYQ1;
V7  =  Q0/AYP0/AYP3;

V0;=YP0;V1;=YP1;V5;SET8;OUT;
V0;=YQ0;V4;=YQ1;V6;SET8;OUT;
V0;=YP0;V2;=YP1;V3;=YP2;V5;NOT;NEG;SET8;OUT;
V0;=YQ0;V6;SET8;OUT;
V4;=YP3;SET40;=RC1;SET1;
1; J2C1Z;DUP;JSL1000;=YP1;=YP2;V0;=YP0;
   V7;SET8;OUT;DUP;+;DC1;J1;
2; ERASE;ZERO;OUT;
   library L1000;
   (L100);
   FINISH;
   →
```

Chapter 17

Set 1

1·
```
V0  =  Q0/AYD0/AYD1000;
V1  =  Q0/AYP53/AYP57;
V2/3  =  P [NC]PARITY*P.T.R.[C]; (N.B. Carriage return line feed is
not allowed on the flexowriter when using OUT8);
```

```
V4 = Q0/AV2/AV3;

        V0;=Q1;SET2;SET5;OUT;=C1;
        PREQ1;
        V1;=Q2;C1T0Q2;PARQ1;J3TR;
        PRQ2;PARQ2;J2NTR; (Note the use of JrNTR);
     3; V4;=Q8;TWQ8;
     2; C1;SET6;OUT;
```

(N·B. TWQq; has been used to show its use but please bear in mind the comment at section 17·10 and particularly that when output goes to the typewriter none of the characters may be "Carriage return line feed at "Tab").

```
2·  V0  = Q0/AV1/AV1;
    V1  = P⌊7DP⌋;
    V2  = B 07 70 06 70 07 71 06 71;
    V3  = Q0/AV2/AV2;
    V4  = Q0/0/60;

        V0;=Q1;SET3;SET5; OUT;=C1;
        LPQ1;PARQ1;
        LPQ1;PARQ1;
        LPQ1;PARQ1;J3NTR;
```

(N.B. there are no Jumps after previous PARQq's. We are only interested if a parity occurred anywhere);

```
        V4;=Q3;SET2;SET5;OUT;=C3;
        PGAPQ3;
        V3;=Q2;C3T0Q3;
        PWQ2;PGAPQ3;SET6-OUT;
    3;  C1;SET6;OUT;
```

```
3·  V0  =  Q0/AV1/AV1;
    V1  =  P ⌊7DP⌋;
    V2  =  P ⌊7DC⌋;
    V3  =  Q 0/AYP0/AYP1;
    V4  =  Q 0/AYP1/AYP1;
    V5  =  P ⌊NC⌋P*L.P⌊C⌋;

        V0;=Q1;SET3;SET5;OUT;=C1;
        LPQ1;YR7;=YP0;V2;=YP1;V3;=Q2;C1TOQ2;
        PARQ1;J1TR;LPQ2;V4;=Q1;C2TOQ1;PARQ2;J2TR;
        LPQ1;PARQ1;J3TR;LPQ2;PARQ2;J4TR;

    5;  C2;SET6;OUT;..............
        ..........................
    1;2;3;4; V5;=YP1;V4;=Q1;TWQ1;J5;
```

(Notice how the four parity failure jumps have been linked together by giving one point more than one reference. Also refer to the note after the solution to question 1).

4·

```
V0  =  Q0/0/40;
V1  =  Q0/AYD5/AYD5;
V2  =  Q0/1/1;

     V0;=Q1;SET1;SET5;OUT;DUP;=C1;
     V1;=Q2;=C2;
     SET5;=RC3;

1;   PGAPQ1;PWEQ2;
     V2;=+Q2;DC3;J1C3NZ;
     SET60;=M1;
     C1;SET6;OUT;
```

5·

```
V0/2  =    P ⌊NC⌋ IS*IT*MONDAY ⌊DQ⌋;
V3    =    Q0/AY0/AY2;
V0;=Y0;V1;V2;=Y2;
V3;=Q1;TWQ1;
```

(Refer to the note in solution to question 1.)

(N.B. The operator's reply Y.→ or N.→ will go into the first 3 characters of Y2 - the remaining 5 characters being $00_{(8)}$. It would have been possible for V3 = Q0/AV0/AV2; but later in the course, when RESTARTS are dealt with, the reader will see that, in general, it is not wise to print out from V-stores).

6·

```
V0/1 = P+KESUBLE.50-1470;
V1;V0;SET10;OUT;=C1;
```

(Note that V1 must be brought into the next than the least significant half).

Set 2

1· (a) ⊔KDF⊔9

(⊔indicates a space. The first character of the output to the lineprinter by LPQq; will be printed at the left of the next line).

(b) ⊔$\underline{\text{KDF}}$⊔$\underline{9}$

2· **Answer:** A

N1 originally contained $3675_{(8)}$

N2 originally contained $-1_{(8)}$ $[=-1_{(8)}]$

∴ after -; N1 contains $-3676_{(8)}$

which is $7777777777774102_{(8)}$

i.e., in P-constant format this is ⌊6D⌋A⌊C⌋

3· (a)

```
V0   =  Q0/AYP1/AYP8;
V1/2 =  P JOE*SOAP. ⌊6DC⌋
V3   =  P⌊7DP⌋;
V4   =  Q0/AYQ4/AYQ5;
```

```
        V4 =   Q0/AYQ4/AYQ5;
        V5 =   P ⌊7DC⌋;
        V7 =   Q0/AYQ0/AYQ5;
        V8 =   P⌊N⌋ PARITY;
        V9 =   Q0/AYQ0/AYQ2;

        V0;=Q1;SET2;SET5;OUT;=C1;PRQ1;
        SETb36;YQ4;V3;=YQ5;
        V4;SET8;OUT;
        SETB36;=YQ1;V2;=YQ2;
        V9;SET8;OUT;
        V5;=YQ5;PARQ1;
        C1;SET6;OUT;J1TR;
```

(Notice reason for deallocating before jumptng on test register after PARQq;)

```
        SETB36;DUP;=YQ4;V4;DUP;SET8;OUT;REV;
        =YQ4;SET8;OUT;ZERO;DUP;=YQ2;=YQ3;
        SET8;=RC1;SETB36;
    2;  DUP;=YQ0;YP1M1Q;=YQ1;YP1M1Q;=YQ4;
        V7;SET8;OUT;DUP;=YQ4;V4;SET8;OUT;J2C1NZ;
    3;  ZERO;OUT;
    1;  V8;=YQ5;ZERO;=YQ4;V4;SET8;OUT;J3;
        FINISH;
        →
```

Chapter 18

Set 1

1.
```
    V1  = Q10/10/AY0;
    V1;=Q1;I15=+1;
 1; MØM1;=MØM15Q;
    MØM1QN;=MØM15Q-J1C1NZ;
```

2.
```
    V1  = Q32/2/AYA0L;
    V2  = Q32/3/AYBØU;
    V1;=Q1;V2;=Q2;
 1; MØM1QH;=MØM2QH;J1C1NZ;
```

3.
```
    V1  = Q9/16/AYØU;
    V2  = Q9/16/AYØL;
    V1;=Q1;V2;=Q2;
 1; MØM1H;MØM2H;=MØM1H;=MØM2H;
    MØM1HN;MØM2HN;=MØM1QHN;=MØM2QHN;J1C1NZ;
```

Set 2

1. $0000000000201235_{(8)}$

2. Answer: NO.

MØMØHN; fetches a copy of the half length word E1U into D0 - D23 of N1.

SET5; Sets the integer 5, held to 47 integral places, into the nest.

Hence when N1 is added to N2 a mixed result is obtained.

=M0M0HN; overwrites the half word E1U with the contents of D0 -D23 of N1, which will not have changed.

Chapter 19

Set 1

```
1·  SET64;=RC1;ZERO;STR;
  1; Y0M1Q;DUP;×D;+D;J2V;
    J1C1NZ;
```

```
2·  (a) SETAY0U;=RM1;
        M0M1QH;M0M1H;×;
```

```
    (b) 46 integral places
```

```
3·  V0 = B 1717171717171717;
    V1 = B 1212121212121212;

    V0;Y0;AND;V1;REV;TOB;
    SET240;×D;CONT;=Y0;
```

```
4·  V0 = Q64/1/0;
    V0;=Q1;ZERO;DUP;
  1; Y0M1Q;STR;+D;J1C1NZ;
    M1;÷D;
    (Note that M1 now contains 64 and C1 contains 0).
```

```
5·  SET12;÷I;=Y2;
    SET20;÷I;=Y1;
             =Y0;
```

6· (a) Valid $\frac{128}{18} = 6 \cdot 7$

and 6·7 can be held to (8-5) integral places

(b) Not Valid $\frac{19}{17 \cdot 785} > 1$

and this cannot be held to (5-5) integral places.

(c) Not Valid $\frac{2^{10}}{2^{12}} = 2^{-2}$

Since this is +ve we use $2^{-2} < 2^{p}$

But p we are allowed is (11-13), hence 2^{-2} (= $\frac{1}{4}$) cannot be held in -2 integral places.

(d) $\frac{1}{-1} = -1$

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

Set 2

```
1.  uuuuuuu+uuuuuuu4uuuuuuuuuuuuuuu+uuuuuuu1
           -       5               -       2
           -       5               +       2
           +       4               -       1
```

```
2.  V0  =  -13;
    V1  =  -3;
    V2  =  =13;
    V3  =  +3;
    V4  =  +13;
    V5  =  -3;
    V6  =  +13;
    V7  =  +3;
```

3. The first syllable of the two syllable instruction M0M1; will be placed in the main store so that it is in the 1st syllable of the next available word.

Similarly for the three syllable instruction J1C1NZ; Any unused syllables of the preceding word will automatically be filled with sufficient one syllable DUMMY; instructions.

Chapter 20

Sets 1 and 2

```
1.  (a) V0  =  Q60/1/AY9;
        V0;=Q1;ZERO;DUP-
      1; M0M1Q;DUP;×+-6;J1C1NZ;
```

(The instruction X+-6; could have been written as ×D;SHAD-6;+D)

(b) 6 integral places.

2. The largest possible answer is $\frac{30}{18} = 1\ 2/3$

To find the p to hold the answer to maximum precision:-

$1\frac{2}{3} < 2^p$

Therefore p needs to be 1.

But N2 has p=5
N1 has p=6

So that result may have p=1 the contents of N2 (the numberator needs to be shifted -2 binary places so as to make its p=7.

Then 7-6 = 1.

Solution:-

```
ZERO ;CAB;SHAD-2;CAB;+D;
```

```
3·    ZERO;REV;SET8;=RC1;
   1; SHLD-3;SHL-3;DC1;J1C1NZ;
      ERASE;SHL-24;
```

```
4·    =Q1;J3C1Z;M0M1Q;J2C1Z;
      M0M1Q;MAX;J4C1Z;
   1; M0M1Q;MAX;PERM;MAX;ERASE;
      REV;J1C1NZ;J4;

   2; DUP;J4;

   3; ZERO;DUP;

   4; VR;........
```

5·

Number	f	p	
2·5	·625	2	
1·0	·5	1	
-1·0	0	0	(N.B. s=1)
5·25	·65625	3	
·25	·5	-1	
4·7	·5875	3	
0·1	·8	-3	

```
6·    Y0;DUP;J2=Z;DUP;
      =RC1;SETAY1;=M1;ZERO;
   1; M0M1Q;STAND;+F;J1C1NZ;
      REV;SET47;FLOAT;÷F;
      =M0M1;J3;

   2; ERASE;

   3; .......
```

```
7·    Y0;DUP;J3≤Z;=RC1;Q1TOQ2;SET-128;
   1; Y1M1Q;FIX;REV;ERASE;MAX;REV;ERASE;J1C1NZ;

   2; DUP;Y1M1Q;J2C2NZ;=;=C15;
      SHAC15;=Y1M1Q;J2C2NZ;=Y1M1;

   3; (n≤0);
```

```
8·    SET64;=RC1;ZERO;DUP;
   1; Y0M1Q;DUP;CAB;+;PERM;SET47;
      FLOAT;+F;REV;J1C1NZ;
      REV;FIX;SET47;-; =C1;
      SHAC1;REV;SIGN;
```

9· ROUNDF;

Model Solution to Programming Exercise B

(Although the following is neatly finalised with the V store declarations followed by the instructions, the reader must not be led to think that the course of writing the program was to write all the declarations then the instructions. The process was to write the instructions and on a separate sheet to write the declarations as the need for them arose.)

An elementary solution would be:

```
     V0  =  Q0/AY0/AY32;
     V1  =  B30;
     V2  =  P SORT ⌊3DC⌋;
     V3  =  B 1212121212121212;
     V4  =  B 1717171717171717;
     V5  =  Q 0/AV1/AV2;
     V6  =  P PAR.P.T.;
     V7  =  P ⌊7DC⌋;
     V8  =  *0/AYA0/AYA2;
     V9  =  *0/AYA0/AYA1;

     V0;=Q1;SET2;SET5;OUT;=C1;
     PREQ1;V5;SET8;OUT;(V1 is now B30;)PARQ1;J101TR;
     C1;SET6;OUT;
     Y0;V4;AND;V3;REV;TOB;=RC2;M+I2;
     SET1;DUP;=RM3;=RM5;
  2; M3;C2;-;J3=Z;
     M3T0Q4;M3T0Q5;M+I5;
  4; M5;C2;-;J5>Z;
     Y0M4;Y0M5;-;J6<Z;
     M5T0Q4;
  6; M+I5;J4;
  5; Y0M4;Y0M3;=Y0M4;=Y0M3;M+13;J2;
  3; V7;=YA2;
  4; SETB30;=YA0;Y0M2Q;=YA1;V8;SET8;OUT;J4C2NZ;
  7; ZERO;OUT;
101; SETBØØ;=YA0;V6;=YA1;V9;SET8;OUT;C1;SET6;OUT;J7;
     FINISH; →
```

Probably a better solution, making use of more types of Usercode Insturctions, would be as below:-

```
     V0 = Q 0/AY0/AY32;
     V1 = B 30;
     V2/5 = P MODEL*SOLUTION*TO
            *SORT*EXERCISE;
     V6 = P ⌊7DC⌋;
     V7 = Q 6/1/0;
     V8 = Q 0/AYP0/AYP5;
```

```
V9  =  B 1212121212121212;
V10 =  B 1717171717171717;
V11 =  Q 0/AYP0/AYP2;
V12 =  P PARITY;
V13 =  P N*IS*0;
V14 =  Q0/AYP0/AYP1;
       V0; =Q1; SET2; SET5; OUT; =C1;
       PREQ1; V7; =Q2;
1;     V1M2; =YP0M2Q; J1C2NZ;
       V8; SET8; OUT; V1; =YP0;V6=YP1;V14;SET8;OUT;
       PARQ1; J101TR; C1; SET6; OUT;
       V9; Y0; V10; AND; TOB; DUP; =RC2; =RC3;
       J102C3Z;
       SET1; =M3; DC3;
2;     J3C3Z;
       M3; =RM4; M4T0Q5; M+I5;
4;     M5; C2; SIGN; J5>Z;
       Y0M4; Y0M5; SIGN; J6<Z;
       M5; =M4;
6;     M+I5; J4;
5;     Y0M4; Y0M3; =Y0M4; =Y0M3Q; J2;
3;     V6; =YP2;
8;     V1; =YP0; Y1M2Q; =YP1;
       V11; SET8; OUT; J8C2NZ;
7;     ZERO; OUT;
101;   V12; J103;
102;   V13; J103;
103;   =YP1; V1; =YP0; V6; =YP2;
       V11; SET8; OUT; J7;
       FINISH;
       →
```

Chapter 22

Set 1

```
1·  V0  =  Q0/AY0/AY2;
    V1  =  Q0/AYB0/AYB2;
    V2  =  Q0/AYZ0/AYZ0;

    V0;=Q2;C1T0Q2;METQ2;J1TR;MWQ2;
    V1;=Q3;C1T0Q3;PARQ2;J2TR-MBRQ3;
    PARQ3;J2TR;.............

  2; ZERO;=YZ0;V2;=Q4;C1T0Q4;MWQ4;............
  2; C1;SET6;OUT;......

2·  V0  =  PABCD1234;
    V1  =  Q0/1/4;
    V1;=Q1;V0;SET4;OUT;=C1;MFSKQ1;
    PARQ1;J1TR;.........
  1; C1;SET6; OUT;
```

```
3·     V0  =  PABCD1235;
       V1  =  Q0/AY0/AY000;
       V2  =  Q0/AYZ0/AYZ0;

       V0;SET4;OUT;=RC1;M+I1;MFSKQ1;
       V1;=Q2;C1T0Q2;PARQ1;J1TR;METQ2;J2TR;
       MWQ2;PARQ2;J1TR;M+I1;MBSKQ1;MBTQ1;J3TR;
       PARQ1;J1TR;M-I1;MBSKQ1;MBTQ1;J4NTR;
       PARQ1;J1TR;..............

    1; C1;SET6;OUT;.........
    2; ZERO;=YZ0;V2;=Q7;C1T0Q7;MWQ&;........
    3; (Routine required to show that "Too few blocks  written");....
    4; (Routine required to show that "Too many blocks written");....
```

Set 2

```
       V0  =  P KEEM0000;
       V1  =  P KEDJ9999;
       V2  =  Q0/1/1;
       V3  =  Q0/AYA0/AYA1023;
       V4  =  Q0/AYB0/AYB1023;

       V0;SET4;OUT;DUP;=YD0;=RC3;M+I3;MFSKQ3;
       V1;SET4;OUT;DUP;=YD1;=RC4;M+I4;MFSKQ4;
       PARQ3;J1TR;
       (Claimed and positioned both tapes);
       V2;=Q1;(Block counter);V3;=Q5;V4;=Q6;
       C3T0Q5;C4T0Q6; (Sett up areas for read and write, not locked out);
       PARQ4;J1TR;
    3; MFREQ5;IM5T0Q7;IM5T0Q5;IM7T0Q6;
       (Interchange areas for double buffering);PARQ5;J1TR;
       PARQ6;J1TR;MLBQ5;J4TR;
       METQ6;J5TR;
       METW6;J5TR;
       MWEQ6;M+I3; (Add 1 to counter); J3;
    1; (Print mag parity via OUT8);J10;
    4; MLWEQ6;M+I1;APRQ6;J1TR;C6T0Q1;MBTQ1;J1TR;
       MBSKQ1;
       MBTQ1;J6TR;M4T0Q1; (Sets 1 in Mode); PARQ1;J1TR;
       MBSKQ1;
       MBTQ1;J7NTR; (Check tape correctly written); PARQ1;J1TR;
   10; YD0;SET6;OUT;YD1;SET6;OUT;ZERO;OUT;
    5; (Print end of tape); J10;
    6; (Pr int too few); J10;
    7; (Print too many); J10;
       FINISH;
```

Chapter 23

Set 1

```
1·    P2V0;
      J1TR;ZERO;=W0;
   2; SET4;OUT;DUP;=RC15;M+I15;
      MFSKQ15;PARQ15;J3TR;W0;=TR;EXIT2;
   2; SET-1;=Wo;J2;
   3; EXIT1;
```

```
2·    P4V0;
   3; =Q15;Q0T0Q14;J2V;SET1;=W0;
   4; M0M15Q;DUP;DUP;=Q14;J5C15Z;
   5; Q14;VR;
      ZERO;W0;MAX;ERASE;ERASE;
   6; EXIT;
   2; ZERO;=W0;J3;
```

The principle here is to keep the current largest and smallest in the nesting store and to sum the rest in Q14.

The largest possible value of n is 32767, hence the largest possible total of all the numbers is approximately

$3\cdot 2\times 10^{4} \times 10^{9} = 3\cdot 2\times 10^{13}$, this is less than $1\cdot 4\times 10^{14}$ and so the result can be held single length. No failure is possible so only the one exit is needed.

Notice the method of ensuring that the overflow register is the same on exit as it was on entry. (N.B. ZERO;ZERO;MAX; sets the overflow register).

```
3·    DUP;J3=Z;SETAR3;=LINK;EXIT2;
      3;..........
```

Set 2

```
1·    P6V0;
      PERM;=C14;=C15;
      SHCC14;NC14;C14;=+C15;:J1C15Z;STR;
      REV;SHXX15;STR;PERM;REV;CONT;NC15;
      SHCC15;REV;CONT;
   1; SHCC14;EXIT1;
```

Chapter 24

Set 1

```
1·    SET64;=RC1;
      ZERO;STR;

      *1;W0M1Q;DUMMY;DUMMY;DUP;
      *X+F;J1C1NZS;
```

Chapter 24

Set 1

```
1·    SET64;=RC1;
      ZERO;STR;

      *1;W0M1Q;DUMMY;DUMMY;DUP;
      *×+F;J1C1NZS;
```

Set

```
1·    V0  =  Q128/1/AY0;

      V0;=Q1;Q0TOQ2;M0M1Q;M0M1Q;MAX;
      *1;M0M1Q;MAX;PERM;*MAX;=+Q2;REV;J1C1NZS;
      DUPD;+;Q2;+;
```

Although a short loop is the best method for this routine it is unfortunate that the first two rules for optimisation of short loops given in paragraph 24·2·3 cannot be obeyed.

Chapter 25

Model Solution to Programming Exercise C.

```
      V0  =  Q0/1/1;
      V1  =  P NAME ⌊3DC⌋;
      V2  =  Q0/AYP0/AYP1;
      V3  =  P ⌊7DC⌋;
      V4  =  Q0/AYQ0/AYQ1;
      V5  =  Q47/1/AY0;
      V6  =  Q0/AY1/AY48;
      V7  =  B30;
      V8  =  Q0/AYP0/AYP6;
      V9  =  P ⌊7DC⌋;
      V10 =  Q47/-1/AY47;
      V11 =  10^13;
      V12 =  P L.B.PAR ⌊C⌋;
      V13 =  P END.TP. ⌊C⌋;
      V14 =  P WRT.PAR ⌊C⌋;
      V15 =  P B.R. D.PR ⌊C⌋;
      V16 =  P TOO_ FEW ⌊C⌋;
      V17 =  P LB-BTPR ⌊C⌋;
      V18 =  P TOO MANY ⌊C⌋;
      V19 =  P LBBKSKP ⌊C⌋;

      V0;=Q1;V2;=Q2;
      ZERO;SET4;OUT;=C1;MFSKQ1;SETB30;=YP0;V1;=YP1;
      V2;SET8;OUT;
      SETB30;=Y0-V3;=YQ1;V4;SET8;OUT;PARQ1;J101TR;
      ZERO;=Y0;SET1;=Y1;V5;=Q2;
  *1; M0M2;M0M2QN;+;*=M0M2N;
      J1C2NZS;
```

```
2; V6;=Q2;C1T0Q2;METQ2;J102TR;MWQ2;PARQ2;J104TR;
   MBRQ2;PARQ2;J104TR;MBSKQ1;PARQ1;J105TR;MBTQ1;J106TR;
   MBSKQ1;PARQ1;J107TR;MBTQ1;J108NTR;C1;SET6;OUT;
   Y48;JSL1000;=YP1;=YP2;V7;=YP0;ZERO;DUP;=YP3;DUP;=YP4;=YP5;V9;=YP6;
   V9;SET8;OUT;
   V10;=Q1;
3; M0M1;DUP;JSL1000;=YP1;YP2;V7;=YP0;M0M1QN;V11;×D;CAB;+D;
   JSL1000;ZERO;SHLD+18;SHL+6;SETB37;OR;SHLD-18;ERASE;
   =YP4;=YP5;V8;SET8;OUT;J3C1NZ;4;ZERO;OUT;

   101;    (MT LB1 Parity); V12;J5;
   102;    (End Tape); V13;J5;
   103;    (MT Write Parity); V14;J5;
   104;    (MT B.read Parity); V15;J5;
   105;    (MT LB BK Sk); V19;J5;
   106;    (Two few); V16;J5;
   105,    (MT parity LB-BTW); V17;J5;
   108;    (MT Too many); V18;J5;
     5;    =YP1;V7;=YP0;V2;SET8;OUT;C1;SET6;OUT;J4;
           library L1000;
           (L1000);
           FINISH;
           →
```

(N.B. This program causes the program to wind-up in the case of a failure jump, but there is no reason why an appropriate corrective routine should not have been used for the program to continue.)

APPENDIX 2 REVIEW OF ELEMENTARY MATHEMATICS

This appendix is intended for those who require a quick refresher or a review of the terminology of algebra.

1• Algebraic Equations

The "equality" symbol is used to indicate that what is on the left hand side (LHS) of it is exactly equal to what is on the right hand side (RHS), (e.g., 2+3=4+1). This is known as an "equation". When one of the terms is not known a letter of the alphabet may be used to replace the unknown quantity.

E.g., if in the above equation the second term on the LHS were not known, the equation could be written as

$$2 + y = 4 + 1 \ldots\ldots\ldots \text{(i)}$$

To find the value of y from the equation $2 + y = 4 + 1$ all that is needed is to subtract the 2 from the LHS and RHS.

So:- $2 + y - 2 = 4 + 1 - 2$

Therefore $y = 3$

The rule is: to find the value of an unknown in equation add to, subtract from, multiply by, divide by, etc., both sides of the equation until the unknown is isolated on one side.

E.g., to find the value of s given that

$$\begin{aligned}
2s + 4 &= 3 - 3s \\
2s + 4 - 4 &= 3 - 3s - 4 \\
2s &= -3s - 1 \\
2s + 3s &= -3s - 1 + 3s \\
5s &= -1 \\
\frac{5s}{5} &= \frac{-1}{5} \\
s &= \frac{-1}{5}
\end{aligned}$$

Sometimes an equation has more than one unknown term, e.g.,

$$x = 4-s.$$

If we can, by some other means, find the value of s, we can then find the value of x.

Let us say that $s = 3$
we then know that $x = 4-3$
$\therefore\ x = 1$

This process of putting in a known value for an unknown value is called "substitution".

Let us now consider the equation

$$y = 2 + 3 \times 4$$

What is the value of y?

Is it 2 plus 3 giving 5 then multiplied by 4 giving y = 20, or is it 3 multiplied by 4 giving 12 then plus 2 giving y = 14? The answer lies in the rule that multiplicati (and divisions) must be done before additions (and subtractions). Hence in the equation above 3 × 4 = 12 then +2 gives y = 14.

Some more examples are

$$\begin{aligned} a &= 4 \times 2 + 3 \times 1 \\ &= (4\times2) + (3\times1) \\ &= \quad 8 + 4 \\ &= \underline{12} \end{aligned}$$

It will be noticed that the use of brackets has been made here: they are used to indicate that all operations inside the brackets are to be performed first.

$$\begin{aligned} b &= 3 - 8 \div 2 \times 10 + 140 \\ &= 3 -(8\div2) \times 10 + 140 \\ &= 3 - 4 \times 10 + 140 \\ &= 3 -(4\times10) + 140 \\ &= 3 - 40 + 140 \\ &= \underline{103} \end{aligned}$$

Notice the set of brackets which ensures the correct result is obtained. Brackets can be used at any time but notice how they work from the following examples.

$$\begin{aligned} c &= 3 + (4\times2) \\ &= 3 + 8 \\ &= \underline{11} \end{aligned}$$

$$\begin{aligned} d &= (3+4) \times 2 \\ &= 7 \times 2 \\ &= \underline{14} \end{aligned}$$

It is convention to leave out the multiplication sign just in front of or after a bracket when it is clear what is intended;

Consider

$$\begin{aligned} d &= 2 \times (3+4) \\ &= 2 \times 7 \\ &= 14 \end{aligned}$$

It is clear that d = twice (3+4)
Therefore we could write d = 2(3+4)

An example linking the items so far reviewed is:

$$\begin{aligned} f &= 14 - (3-2)\ 8 - 2 \times 4 \\ &= 14 - (1)\ 8 - (2\times4) \\ &= 14 - 8 - (2\times4) \\ &= 14 - 8 - 8 \\ &= \underline{-2} \end{aligned}$$

2• Indices

There is the type of equation where the LHS is equal to something on the RHS multiplied by itself many times, e.g., $y = 5\times5\times5\times5$

It is conventional in this case to only write the 5 once and to put a little number above it to show how many times it is to be multiplied by itself.

In this example we would write $y = 5^4$

Let us now see what x is in the following case

$$\begin{aligned} x &= (-6+3) \times 2^3 \\ &= (-3) \times2\times2\times2 \\ &= (-3) \times 8 \\ &= \underline{-24} \end{aligned}$$

The "raised number" is called the index.

To consider the index a little further we can see that

$$\begin{aligned} 4^5 &= 1024 \\ 4^4 &= 256 \\ 4^3 &= 64 \\ 4^2 &= 16 \end{aligned}$$

A close inspection reveals that reducing the index by 1 has the effect of dividing the number by the "base" number, in this case 4 (1024+4 = 256).

From this we would assume that

$$\begin{aligned} 4^1 &= 16+4 \\ &= 4 \text{ (i.e., 4 multiplied together once)} \end{aligned}$$

What then is 4^0.

Surely if reducing the index by 1 has the effect of dividing the answer by the base, then knowing that

$$4^1 = 4$$

we must conclude that $4^{0} = 4 \div 4 = 1$

continuing further $4^{-1} = 1 \div 4 = \frac{1}{4}$

$$4^{-2} = \tfrac{1}{4} \div 4 = \frac{1}{16}$$

Two points emerge:

(a) Any number with an index 0 has the value 1

(b) Any number with a negative index has the value of 1 divided by the same number with the corresponding positive index.

E.g.,

$$4^{-2} = \frac{1}{16} = \frac{1}{4^2}$$

Therefore $\underline{4^{-2} = 1 \div 4^{2}}$

If then the equation $x = (-s+f)\times 2^{p}$ is to be solved where s, f and p are given values, the value of x can be found

e.g.,

$$\begin{aligned} x &= (-1+\tfrac{1}{2})\times 2^{3} \\ &= (-\tfrac{1}{2})\times 8 \\ &= \underline{-4} \end{aligned}$$

e.g.,

$$\begin{aligned} n &= (-1+\tfrac{1}{3})\times 2^{-2} \\ &= (-\tfrac{2}{3} \times \tfrac{1}{4}) \\ &= \frac{-2}{12} \\ &= \underline{-\frac{1}{6}} \end{aligned}$$

The use of the index is useful when large numbers are being referred to

e.g., 17000000

can be written as 17×10^{6}

or as $1{\cdot}7\times 10^{7}$

etc.

3·

Two terms often used are "algebraic" and "numeric" minimum.

(a) Algebraic minimum refers to that quantity which is less than any other being considered, where a negative quantity is less than any positive quantity.

(b) Numeric minimum refers to that quantity which is nearer to zero - or in other words the sign in front of the numbers are ignored and the smallest found.

Hence $+\frac{1}{2}$ is numerically less than -1.

4. Suffixing

Although any letters can be used in an equation to represent unknown quantities sometimes the same letter is used with a "suffix" to differentiate between the various terms.

e.g., $x = 2 \times 4 + 7 - 3$

could be written as

$$n = a_1 \times a_2 + a_3 - a_4$$

where

$$a_1 = 2$$
$$a_2 = 4$$
$$a_3 = 7$$
$$a_4 = 3$$

There is no reason why the suffix should not be negative but in this case, to prevent confusion, the suffix is placed in parenthesis, e.g., $a_{(-4)}$

5. Inequalities

There is a class of equation where the LHS need not necessarily be equal to the RHS. This is expressed by the symbols

$<$ meaning algebraically less than
$>$ meaning algebraically greater than
$\leqslant$ meaning algebraically greater than or equal to
$\geqslant$ meaning algebraically greater than or equal to.

Examples: $x \leqslant 4^2$ means that x can have any algebraic value which is less than or equal to 16.

$x < 2^3$ means that x can have any algebraic value less than 8.

6. Modulus

When it is required to refer to the magnitude of a quantity without quoting whether it is positive or negative, then the quantity is placed between two vertical bars and called the "modulus of the quantity".

e.g., $|5|$ is the same as 5
$|+\frac{1}{2}|$ is the same as $\frac{1}{2}$
$|-4|$ is the same as 4

$|x|$ (where $x = -2$) is the same as 2
$|y|$ (where $y = 4$) is the same as 4

7. Significance

In the number 34·278, the digit 3 is the most significant
and the digit 8 the least significant.

The "value" to the left of the decimal point is called the "integral part" and that to the right the "fractional part".

APPENDIX 3 POWERS OF TWO

N	2^N	2^{-N}
0	1	1•00000000000000000000000000000000000000000000000000
1	2	•50000000000000000000000000000000000000000000000000
2	4	•25000000000000000000000000000000000000000000000000
3	8	•12500000000000000000000000000000000000000000000000
4	16	•06250000000000000000000000000000000000000000000000
5	32	•03125000000000000000000000000000000000000000000000
6	64	•01562500000000000000000000000000000000000000000000
7	128	•00781250000000000000000000000000000000000000000000
8	256	•00390625000000000000000000000000000000000000000000
9	512	•00195312500000000000000000000000000000000000000000
10	1024	•00097656250000000000000000000000000000000000000000
11	2048	•00048828125000000000000000000000000000000000000000
12	4096	•00024414062500000000000000000000000000000000000000
13	8192	•00012207031250000000000000000000000000000000000000
14	16384	•00006103515625000000000000000000000000000000000000
15	32768	•00003051757812500000000000000000000000000000000000
16	65536	•00001525878906250000000000000000000000000000000000
17	131072	•00000762939453125000000000000000000000000000000000
18	262144	•00000381469726562500000000000000000000000000000000
19	524288	•00000190734863281250000000000000000000000000000000
20	1048576	•00000095367431640625000000000000000000000000000000
21	2097152	•00000047683715820312500000000000000000000000000000
22	4194304	•00000023841857910156250000000000000000000000000000
23	8388608	•00000011920928955078125000000000000000000000000000
24	16777216	•00000005960464477539062500000000000000000000000000

Powers of two (continued)

N	2^N	2^{-N}
25	33554432	·000000029802322387695312500000000000000000000000
26	67108864	·000000014901161193847656250000000000000000000000
27	134217728	·000000007450580596923828125000000000000000000000
28	268435456	·000000003725290298461914062500000000000000000000
29	536870912	·000000001862645149230957031250000000000000000000
30	1073741824	·000000000931322574615478515625000000000000000000
31	2147483648	·000000000465661287307739257812500000000000000000
32	4294967296	·000000000232830643653869628906250000000000000000
33	8589934592	·000000000116415321826934814453125000000000000000
34	17179869184	·000000000058207660913467407226562500000000000000
35	34359738368	·000000000029103830456733703613281250000000000000
36	68719476736	·000000000014551915228366851806640625000000000000
37	137438953472	·000000000007275957614183425903320312500000000000
38	274877906944	·000000000003637978807091712951660156250000000000
39	549755813888	·000000000001818989403545856475830078125000000000
40	1099511627776	·000000000000909494701772928237915039062500000000
41	2199023255552	·000000000000454747350886464118957519531250000000
42	4398046511104	·000000000000227373675443232059478759765625000000
43	8796093022208	·000000000000113686837721616029739379882812500000
44	17592186044416	·000000000000056843418860808014869689941406250000
45	35184372088832	·000000000000028421709430404007434844970703125000
46	70368744177664	·000000000000014210854715202003717422485351562500
47	140737488355328	·000000000000007105427357601001858711242675781250
48	281474976710656	·000000000000003552713678800500929355621337890625

APPENDIX 4: INSTRUCTION CROSS-REFERENCE (With Syllable Counts)

This list is intended to provide easy reference to any User Code instruction described in this manual, and also quotes the number of syllables occupied by each instruction.

Any item in brackets is optional.

The form Yy has been used to indcate where any of the usual alternatives may exist.

Form	Syllables	Section	Form	Syllables	Section
ABS	1	11•1	Jr	3	13•1•5
ABSF	1	20•5•1	JS	3	13•1•6
AND	1	11•4•2	JE	3	13•1•7
			JSE	3	13•1•7
BITS	1	11•4•1	J(N)EJ	3	13•1•7
BUSYQq	2	App. 6	J(N)EN	3	13•1•7
			Jr(N)V	3	13•1•3
CAB	1	9•1	Jr(N)TR	3	13•1•4
CIkTOQq	2	8•3	Jr=, Jr≠;	3	13•1•2
CMkTOQq	2	8•3	Jr=Z, Jr>Z, etc	3	13•1•1
CONT.	1	(11•1•3	JrCq(N)Z	3	10•4
		(19•1•2	JrCqNZS	2	24•2•1
CkTOQq	2	8•3	LINK	2	23•3•1
Cq	2	8•1	LPQq	2	17•12
DCq	2	8•2	MANUALQq	2	App. 6
DUMMY	1	9•1	MAX	1	11•2
DUP	1	9•1	MAXF	1	20•5•1
DUPD	1	9•1	MBR (E) Qq	2	22•1•8
			MBSKQq	2	22•1•9
ERASE	1	9•1	MBTQq	2	22•1•3
EXIT(n)(ARr)	3	(23•2•3	METQq	2	22•1•3
		(23•3•3	MFR	2	22•1•5/7
			MFSKQq	2	22•1•9
FINISH	0	6•2•6	MGAPQq	2	App. 6
FIX	1	20•5•5	MLBQq	2	22•1•3
FLOAT	1	20•5•5	MLW(E)Qq	2	22•1•5/7
FLOATD	1	20•5•5	MW(E)Qq	2	22•1•5/7
FRB	1	11•5•7	M±Iq	2	8•2
			Mq	2	8•1
IkTOQq	2	8•3	MkMq(Q)(H))N)	2	(10•3
IMkTOQq	2	8•3			(10•5
INTQq	2	App. 6			(18
Iq	2	8•1	MkTOQq	2	8•3
Iq=±1	2	8•2	MRWDQq	2	22•1•9
lq=±2	2	8•2	MWIPEQq	2	App. 6

Form	Syllables	Section
NCq	2	8•2
NEG	1	11•1•2
NEGD	1	11•1•2
NEGF	1	20•5•1
NEGDF	1	20•5•4
NEV	1	11•4•2
NOT	1	11•4•1
OR	1	11•4•2
OUT	3	17•3
PARQq	2	17•8
PERM	1	9•1
PGAPQq	2	17•7
PRC(E)Qq	2	17•7
PRQq	2	17•7
PWQq	2	17•7
PWC(E)Qq	2	17•7
PREQq	2	17•7
PWEQq	2	17•7
Qq	2	8•1
QkTOQq	2	8•3
r	0	6•2•2
REV	1	9•1
REVD	1	9•1
ROUND	1	11•2
ROUNDF	1	20•5•4
ROUNDH	1	11•2
FOUNDHF	1	20•5•1
REe	0	13•1•7
SET	3	7•9
SHA	2	20•1•2
SHL	2	20•1•3
SHC	2	20•1•4
SIGN	1	11•1•4
SIGNF	1	20•5•1
STAND	1	20•5•3
STR	1	11•1•3
TLOQq	2	App. 6
TOB	1	11•5•5
TR(E)Qq	2	17•10•2
TW(E)Qq	2	17•10•2
VR	1	13•1•3
Vv(D)=z(/s)	0	7•3
Vv(D)=Fz	0	7•3
Vv=A	0	7•5
Vv=B	0	7•4
Vv=P	0	7•8
Vv=Q	0	7•6
Vv(U)=	0	7•7
Vv(L)=	0	7•7
Yy(Mq)(Q)	3	(10•2
		(10•5
ZERO	1	9•1
+	1	11•1•2
+F	1	20•5•1
+D	1	11•1•2
+DF	1	20•5•4
-	1	11•1•2
-F	1	20•5•1
-D	1	11•1•2
-DF	1	20•5•4
x	1	19•1•2
xF	1	20•5•2
xD	1	19•1•2
xDF	1	20•5•4
x+±n	2	20•2
x+Cq	2	20•2
x+F	1	20•5•4
÷	1	19•2•2
÷F	1	20•5•2
÷D	1	19•2•2
÷DF	1	20•5•4
÷R	1	20•3
÷I.	1	19•2•2
=LINK	2	23•3•1
=TR	1	13•1•4
=Qq	2	8•1
=+Qq	2	8•1
=(R)Cq	2	8•1
=(R)Iq	2	8•1
=(R)Mq	2	8•1
=+Cq	2	8•1
=+Iq	2	8•1

Form	Syllables	Section
=+Mq	2	8•1
=MkMq(Q)(H)(N)	2	10•3
		10•5
		18
=Yy(Mq)(Q)	3	(10•2
		(10•5

APPENDIX 5 REFERENCE TABLES, FAILURE REPORTS, LIBRARY SUBROUTINES

Reference Tables:
During the compilation stage of a program the computer keeps a record of the E addresses of all the reference labels in the program. The programmer may, obtain a print out of this record when the compilation stage is completed.

The record is called the "Reference Tables" and appears in the following format.

```
KEABCD001UP1              TITLE           DATE:16/12/65   TIME: 21•50.
Y SPEC E512
                    REFERENCE TABLES
P0    40   0        1   51   3   90   65   5
P1    73   0
P2    75   0        1   77   0
L54  106   0        1  106   0    2  122   5   3   125   1   4   111   5
   5 121   0
```

Explanation

(a) Y SPEC E512 indicates that YØ is placed in $E512_{(10)}$

(b) The main program is always PØ, in this case the first instruction begins at $E40_{(8)} = E32_{(10)}$ syllable Ø. Reference 1 of PØ is at the Syllable 3 of $E51_{(8)} = E41_{(10)}$.

(c) The program was written with two private subroutines P1, and P2 and one library subroutine L54.

(d) Note in L54, that the first instruction is at syllable Ø of $E106_{(8)} = E70_{(10)}$ and that reference 1 is also at that point. Notice also that reference 5 is at syllable 0 of $E121_{(8)} = E81$ and yet reference 2 is at 122/5, i.e., the reference tables are laid out in the order of the reference labels as they are encountered in the program.

(e) Note particularly that in the Y Spec. the E address is given in decimal, but that in the reference tables the E address is in octal.

Failure Reports
If during the compilation stage, Usercode errors are detected, each error will be reported on the lines between the program identifier and the Y spec.

There will be a failure report for each error found and these will be of the form.

```
FAILURE REPORT
P0   R90   +4
V13 = YP5;
AT 64/4 L54 R6 NOT IN
```

Explanation

(a) P0 R90 +4	indicates that an error was found at the 4th instruction after Reference 90 in Program P∅.
(b) V13 = YP5;	shows what the erroneous instruction is: in this case it is obvious the instruction should be V13; = YP5;
(c) AT 64/4 L54 R6	NOT IN means that at syllable 4 of E64 is an instruction which refers to Reference 6 of L54 but that such a reference label is not in the subroutine.

Any compilation failure can be found by use of the reference tables and failure reports.

In addition to obtaining reference tables and failure reports the programmer may obtain a printout of the Usercode Version of the program as held in the computer just before compilation is started. This is useful to ensure that the punching of the program on paper tape was performed correctly - and it is this computers Usercode Version of a program which should be referred to when corrections or modifications are subsequently made to the program.

Library Subroutines

When a program is to include a subroutine which is in the library the programmer has three possible paths open to him.

(a) Copying out the instructions of the subroutine just before FINISH;→in the same way as he would if he had written the subroutine himself. This approach of course will mean that he will have to obtain the actual instructions of the subroutine.

(b) Obtaining a paper tape copy (in Usercode) of the subroutine and, when the full program is to be compiled to send three tapes to the operator - tape 1 being his program excluding FINISH; →
Tape 2 being the subroutine
Tape 3 being merely the message FINISH;→

(c) Using the POST SYSTEM and so allow the possibility of a library call message in this program (see Section 14).

When using a library call message the general format is

<u>library</u> L1, L2, L16;

In the printout of the Computer's Usercode Version of the program this will merely appear as

LIBRARY

If the call message be written as :

```
library L1;
(L1);
library L2;
(L2);
library L16;
(L16);
```

the printout will appear as

```
LIBRARY
(L1);
LIBRARY
(L2);
LIBRARY
(L16);
```

This will be of more use to the programmer when he refers to the program some months later.

It must be borne in mind however that in fact the actual instructions of the library subroutines will be in the computer - it is only the "print-out" of the program which is abbreviated in this way.

For more details concerning failure reports, etc., reference should be made to the Service Routine Library Manual Section 7•1.

APPENDIX 6: OTHER PERIPHERAL DEVICES AND THEIR INSTRUCTION CODE

1· Generalised Peripheral Instructions

As alternative instructions for peripheral devices the programmer may use any of the following general forms each of which is to be followed by Qq, e.g., instead of TWQq; the programmer may use POAQq;

Flexowriter

	Instructions
POA	WRITE (TWQq)
POB	WRITE TO EM (TWEQq)
POD	CHARACTER WRITE
POE	L.I.V
POF	L.I.V
POG	Undefined
POH	
POK	
POL	
PIA	READ (TRQq)
PIB	READ TO EM (TREQq)
PIC	CHARACTER READ
PID	CHARACTER READ TO EM
PIE	READ
PIF	READ TO EM
PIG	CHARACTER READ
PIH	CHARACTER READ TO EM
PMA	LIV
PMB	NO EFFECT
PMC	"
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG	Director only
PMH	
PMK	Undefined
PML	Undefined

Paper Tape Reader

	Instructions
POA	LIV
POB	"
POC	"
POD	"
POE	"
POF	"
POG	"
POH	"
POK	"
POL	"
PIA	READ (PRQq)
PIB	READ TO EM (PREQq)
PIC	CHARACTER READ (PRCQq)
PID	CHARACTER READ TO EM (PRCEQq)
PIE	READ
PIF	READ TO EM
PIG	CHARACTER READ
PIH	CHARACTER READ TO EM
PMA	LIV
PMB	SET TR IF 8 CHANNEL SET
PMC	NO EFFECT
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG }	Director only
PMH	
PMK }	Undefined
PML	

Paper Tape Punch

	Instructions
POA	WRITE (PWQq)
POB	WRITE TO EM (PWEQq)
POC	CHARACTER WRITE (PWCQq)
POD	CHARACTER WRITE TO EM (PWCEQq)
POE	CHARACTER GAP (PGAQq)
POF	WORD GAP
POG	Effect dependant on configuration and therefore not defined (see DISC, CARD PUNCH, Graph Plotter Drum, IBM Deck)
POH	
POK	
POL	
PIA	LIV
PIB	"
PIC	"
PID	"
PIE	"
PIF	"
PIG	"
PIH	"
PMA	LIV
PMB	NO EFFECT
PMC	NO EFFECT
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG	DIRECTOR ONLY
PMH	
PMK	Undefined
PML	

Card Reader

	Instruction
POA	LIV
POB	"
POC	"
POD	"
POE	"
POF	"
POG	"
POH	"
POK	"
POL	"
PIA	BINARY READ
PIB	BINARY READ TO EM
PIC	BINARY CHARACTER READ
PID	BINARY CHARACTER READ TO EM
PIE	ALPHA-NUMERIC READ
PIF	ALPHA-NUMERIC READ TO EM
PIG	ALPHA-NUMERIC CHARACTER READ
PIH	ALPHA-NUMERIC CHARACTER READ TO EM
PMA	LIV
PMB	SET TR IF RECHECK SWITCH OFF
PMC	NO EFFECT
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG	DIRECTOR ONLY
PMH	
PMK	Undefined
PML	

Card Punch

	Instructions
POA	PUNCH, DIRECT MODE
POB	PUNCH TO EM, DIRECT MODE
POC	CHARACTER PUNCH, DIRECT MODE
POD	CHARACTER PUNCH, DIRECT MODE TO EM
POE	CHARACTER PUNCH, DIRECT MODE
POF	PUNCH, DIRECT MODE
POG	PUNCH CONVERTED MODE
POH	PUNCH CONVERTED MODE TO EM
POK	CHARACTER PUNCH CONVERTED MODE TO EM
POL	CHARACTER PUNCH CONVERTED MODE
PIA	LIV
PIB	"
PIC	"
PID	"
PIE	"
PIF	"
PIG	"
PIH	"
PMA	LIV
PMB	NO EFFECT
PMC	NO EFFECT
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG	Director only
PMH	Director only
PMK	Undefined
PML	Undefined

Magnetic Tape (1081 and 1085)

	Instructions
POA	WRITE (MWQq)
POB	WRITE TO EM (MWEQq)
POC	LAST BLOCK WRITE (MLWQq)
POD	LAST BLOCK WRITE TO EM (MLWEQq)
POE*	GAP (MGAPQq)
POF*	WIPE (MWIPEQq)
POG	Undefined
POH	
POK	
POL	
PIA	FORWARD READ (MRFQq)
PIB	FORWARD READ TO EM (MFREQq)
PIC	FORWARD READ
PID	FORWARD READ TO EM
PIE	BACKWARD READ (MBRQq)
PIF	BACKWARD READ TO EM (MBREQq)
PIG	BACKWARD READ
PIH	BACKWARD READ TO EM
PMA	FORWARD SKIP (MFSKQq)
PMB	SET TR IF BTC (MBTQq)
PMC	SET TR IF LBC PRESENT (MLBQq)
PMD	REWIND (MRWDQq)
PME	BACKWARD SKIP (MBSKQq)
PMF	SET TR IF ETW (METQq)
PMG	DIRECTOR ONLY
PMH	
PMK	Undefined
PML	

* These two instructions have not been dealt with in this manual because they should only be used when tapes are used on the 1081 decks, and could cause catastrophic errors if used on other decks. Those readers interested in these instructions will find details of their effect at the end of this appendix.

IBM Compatible Tape

	Instructions
POA	WRITE ODD PARITY
POB	WRITE EVEN PARITY
POC	WRITE TAPE MARK OLD PARITY
POD	WRITE TAPE MARK EVEN PARITY
POE	GAP
POF	WIPE
POG	Undefined
POH	
POK	
POL	
PIA	READ ODD PARITY
PIB	READ EVEN PARITY
PIC	READ ODD PARITY
PID	READ EVEN PARITY
PIE	BACKWARD READ ODD PARITY
PIF	BACKWARD READ EVEN PARITY
PIG	BACKWARD READ ODD PARITY
PIH	BACKWARD READ EVEN PARITY
PMA	FORWARD SKIP ODD PARITY
PMB	SET TR IF TAPE ON BTC
PMC	SET TR IF LAST BLOCK LEVEL
PMD	REWIND
PME	BACKWARD SKIP ODD PARITY
PMF	SET TR IF ETW
PMG	Director only
PMH	Director only
PMK	FORWARD SKIP EVEN PARITY
PML	BACKWARD SKIP EVEN PARITY

Disc (Director only)

	Instructions
POA	WRITE
POB	WRITE TO EM
POC	WRITE (FIXED HEADS)
POD	WRITE TO EM (FIXED HEADS)
POE	WRITE (FIXED HEADS)
POF	WRITE
POG	WRITE NEXT SECTOR
POH	WRITE NEXT SECTOR TO EM
POK	WRITE NEXT SECTOR (FIXED HEADS) TO EM
POL	WRITE NEXT SECTOR (FIXED HEADS
PIA	READ
PIB	READ TO EM
PIC	READ (FIXED HEADS)
PID	READ (FIXED HEADS) TO EM
PIE	READ NEXT SECTOR
PIF	READ NEXT SECTOR TO EM
PIG	READ NEXT SECTOR (FIXED HEADS)
PIH	READ NEXT SECTOR (FIXED HEADS) TO EM
PMA	SEEK
PMB	NO EFFECT
PMC	NO EFFECT
PMD	CLEAR HEAD POSITIONS
PME	Undefined
PMF	SET TR IF END OF AREA
PMG	Director only
PMH	Director only
PMK	Undefined
PML	Undefined

Drum (Director only)

	Instructions
POA	WRITE
POB	WRITE TO EM
POC	WRITE
POD	WRITE TO EM
POE	WRITE ZEROES
POF	WRITE ZEROES
POG	Undefined
POH	
POK	
POL	
PIA	READ
PIB	READ TO EM
PIC	READ
PID	READ TO EM
PIE	READ
PIF	READ TO EM
PIG	READ
PIH	READ TO EM
PMA	LIV
PMB	NO EFFECT
PMC	NO EFFECT
PMD	LIV
PME	LIV
PMF	NO EFFECT
PMG	Director
PMH	Director
PMK	Undefined
PML	Undefined

Graph Plotter (CALCOMP)

	Instructions
POA	PLOT
POB	PLOT TO EM
POC	CHARACTER PLOT
POD	CHARACTER PLOT TO EM
POE	NO EFFECT
POF	NO EFFECT
POG	Undefined
POH	
POK	
POL	
PIA	LIV
PIB	LIV
PIC	LIV
PID	LIV
PIE	LIV
PIF	LIV
PIG	LIV
PIH	LIV
PMA	LIV
PMB	SET TR IF PLOTTER ATTACHED
PMC	NO EFFECT
PMD	LIV:
PME	LIV
PMF	NO EFFECT
PMG	Director only
PMH	Director only
PMK	Undefined
PML	Undefined

Standard Interface Buffer

	Instructions
POA	WRITE
POB	WRITE TO EM
POC	CHARACTER WRITE
POD	CHARACTER WRITE TO EM
POE	CHARACTER GAP
POF	WORD GAP
POG	Undefined
POH	
POK	
POL	
PIA	READ
PIB	READ TO EM
PIC	CHARACTER READ
PID	CHARACTER READ TO EM
PIE	READ PARITY OFF
PIF	READ TO EM PARITY OFF
PIG	CHARACTER READ
PIH	CHARACTER READ TO EM
PMA	LIV
PMB	SET TR IF 8 CHANNEL SET
PMC	SET TR IF 8 CHANNEL SET
PMD	LW
PME	LIV
PMF	NO EFFECT
PMG	Director only
PMH	Director only
PMK	Undefined
PML	Undefined

2· General

So far detailed use of the flexowriter, paper tape reader and punch, magnetic tape units and the line printer has been given. We shall now briefly refer to the other peripheral devices available for use on the KDF 9 system.

3· Card Reader

3·1 Principles of Operation The card reader is designed to read standard 80-column cards at an average rate of 600 cards per minute. The feed hopper in which the cards are loaded for input holds approximately 2000 cards, which are stacked face down with the most significant column (Column 1) towards the reading head. There is a reject pocket into which cards are directed if a failure occurs.

The reader interprets a column of a card as one six-bit character (when operated in the converted or alphanumeric mode), or two six-bit characters (when operated in the direct or binary mode). The resultant six-bit characters are stored away in the main store as for any other peripheral device.

Elaborate checks in the reader itself ensure that any mechanical mal-function is detected, and the reader stopped to enable the operator to take corrective action.

3·2 The 80-column Punched Card The standard 80-column card is laid out with the columns numbered from 1 to 80 across the face of the card from left to right. The card also contains 12 rows, numbered Y,X,0,1,2,...,9 from top to bottom. We can thus have 960 possible positions in which a hole can be punched, which can be referred to by co-ordinates, (e.g., 6/ 49 refers to a punching in column 6 of row 49.

There are many different codes in use for punched cards, but the code known as "IBM 4-zone" is one of the most popular and has been adopted as the basis for the code built into KDF 9. If this code is used, the alphanumeric mode of reading will automatically convert the card punchings into the 6-bit characters used for the on-line printer, before transferring them to the main store.

(N.B. these are the same as normal case characters on paper tape.)

If a non-standard code is used, the binary mode of reading should be used, and the bit patterns interpreted by the program.

For the standard code, we require to produce 64 patterns (i.e., all combinations of 6 bits). It is possible to obtain 4096 combinations from the 12 rows in any one card column, so a subset of 64 of these has been chosen for the standard set. Th 64 different card punching arrangements together with their KDF 9 equivalent are shown on the table at the end of this appendix. There is no check that the configuration found on a card is valid, and any invalid configuration encountered will be converted to a 6-bit character. If Y and 0 or X and 0 are punched together in one column the 0 punching is ignored. Similarly, if Y and X are punched together in a column the X punching is ignored.

3·3 The Reading Mode As previously stated, the reader interprets a column of a card as one or two 6-bit characters according to the instruction specified by the programmer.

In the binary Read Mode the reader inteprets a column as two characters and transfers the information without conversion. The 1st. character consists of the information read from rows Y to 3 inclusive and the 2nd consists of rows 4 to 9. Y and 4 are the most significant bits of the two characters.

In the Alphanumeric Read Mode, the reader converts the information in a column to one 6-bit character before performing the transfer. Each character is converted to the 6-bit KDF 9 code representation of the characters punched on the card.

A read to E.M. may be specified by the programmer for either the binary or the alphanumeric mode, so that a transfer may be terminated by an E.M. character in either mode.

A single character read may also be specified for either mode, so that successive characters can be placed in the least significant character positions of successive main store words (Digits D0 to D41 are set to zero).

If any transfer ends part way through a card, the transfer will cease as requested but since the reader cannot stop with a card only partially read, the remaining information on the card will be lost.

For a card transfer rate of 600 cards per minute the mean transfer rate for the Alphanumeric Read Mode is 800 characters per second, and 1600 characters per second for the Binary Read Mode. The actual transfer rate when the columns are passing over the sensing station is 1400 and 2800 characters per second respectively. This discrepancy is due to the time lag between reading successive cards.

After the conclusion of a transfer the drive motion of the card reader continues to run for approximately 7 seconds. The computer can therefore call for a further transfer in the 7 second period without having to wait for the drive motor to accelerate up to speed.

3·4 Card Reader Control Instructions The card reader device number is obtained from Director by the instructions SET4;SET5;OUT; Deallocation is effected in the normal way via OUT6; with the device number in N2.

In contrast to the Input/Output devices already considered, there is no specialised set of instructions for the card reader alone. The instructions for reading information from punched cards into the main store are

PIAQq; Binary Read: Puts 6 information bits Y-3 into 1st character position, Puts 6 information bits 4-9 into 2nd character position etc., PIBQq; Binary Read to End Message.

PIBQq; Binary Character Read: Puts each character into one word.

PIDQq; Binary Character Read to End Message.

PIEQq; Alphanumeric Read: Looks at 12 bits in each column and forms the corresponding 6 bits character in printer code and feeds them into the word character at a time.

PIFQq; Alphanumeric Read to End Message.

PIGQq; Alphanumeric Character Read. Puts each character into one word.

3·5 Checking Facilities on the Card Reader By using two reading stations placed a column apart, a column by column comparison is made before the information is transferred. Parity checking within the computer is ignored. The action taken in the event of a failure depends on the position of the RECHECK switch.

If the RECHECK switch is on, the reader will reject the failed card and stop.

None of the information in the column at which the failure occurred or in subsequent columns of the card will be transferred. The operator may then replace the card in the freed hopper, and it will be re-read, all columns preceding the failed column being ignored. At the failed column the reader "rechecks" and if the comparison is good the transfer continues as usual, otherwise the above failure sequence is repeated.

If the RECHECK switch is off and a failure occurs the failed card and all subsequent cards are rejected but the transfer is continued as normal. The failure is indicated by the illumination of the CHECK FAIL lamp on the reader, which is only reset by the computer after the instruction PARQq; The state of the RECHECK switch can be detected by the test instruction PMBQq; which sets the test register if the recheck switch is off.

4· Magnetic Drum

There are no new user code instructions to be learned in connection with the drum, all communication with the drum being made through the Director via "OUT" instructions. It should be also emphasised that there is no possibility of "optimum programming" of the drum.

The drum consists of 320 sections, each sector containing 128 KDF 9 48-bit words. Thus the total storage space is 320×128 = 40,960 words. The uses to which this storage space can be put full into 3 categories and accordingly the drum is considered to be divided into 3 parts:-

1. **Permanent Program Space (PPS)**
2. **Temporary Program Space (TPS)**
3. **Data Space (DS)**

Any or all of these 3 parts may be absent (i.e., occupy no space or the drum) at any time.

4·1 **Permanent Program Space** Programs to be stored in PPS will be called for by Director, along with the date, time and list of available peripheral devices. Director requires these immediately after it has been read into the machine, before the first program of the day is run. Programs in PPS are thus intended for frequent use during the day, or during a long run, and they are permanent in the sense that they cannot be removed except by reloading Director without them. Director uses a subroutine to load programs into PPS.

4·2 **Temporary Program Space** This will contain a series of programs to be called down in succession, or independently compiled sections of a large program which could not be held completely in the main store. Programs for storage in TPS are loaded by means of a special loading subroutine, which is available in the KDF 9 library. Programs in TPS can be removed by loading new programs when TPS is 'cleared'. Here 'clearing' means that the space is made available for subsequent overwriting, not that zeros automatically overwrite the old TPS. 'Clearing' will occur after obeying OUT0, OUT2, typewriter interrupt A, or after termination following a program failure.

4·3 **Data Space** Programmers may wish to use part of the drum for data storage. Before writing to or reserve some of this space for his own use and no transfer of data can occur until this reservation has been made. Further, such a reservation of data space can be made only once in a program. It should be noted that successor programs or sections called by OUT 1 cannot reserve additional data space, nor can they cancel or modify reservations made by their predecessor. Like TPS, DS can only be 'cleared' by obeying OUT0, OUT2 or as a result of termination by the operator of program failure.

4·4 **Calling Program from the Drum** Programs in PPS may be called to the core store and entered by the same means as any other program by standard 'new program' action after termination by operator or by program failure, after OUT0 or by obeying OUT 1.

Programs in TPS can be called and entered by obeying OUT 1. If a program is to be called from the drum, the 3rd character of the first word of the 'A Block' must be "d" to indicate the source of the program. The maximum number of programs which may be stored on the drum is 16.

All transfers of information to or from the drum are controlled by Director. All information about the drum, which may be required at run time is also obtained through Director. These functions are effected by the following 'OUT' faciliites.

4·5 **OUT 14** At any time in a program a request may be made to discover how many sectors are available for use on the drum. This is effected by the instructions SET14;OUT; No additional parameters are necessary. On returning to the instruction following 'OUT' N1 contains the number of sectors available. If a reservation of data space has already been made (or been made by a predecessor program if the durrent program was called by OUT 1). Then D0 of N1 will be 1 on return from OUT 14, although the number of sectors unused will still be in the least significant part of N1.

The Test Register is also set on return from OUT 14, if a parity failure occurred during the last read-from-drum operation.

4·6 **OUT 13** Once and once only in a program may a reservation of data space be made. This is accomplished by obeying OUT 13, with N2 containing the number of sectors required.

e.g. SET42; SET13; OUT;

would reserve 42 (decimal) sectors, which the programmer can then address as sectors 0 - 41.

It is possible to reserve a number of sectors computed at run time. The number of sectors available to a programmer at run-time may not be known to him, but he can check that his request is possible by using OUT 14.

4·7 **OUT 12** This is a read-from-drum operation and reads from the drum Data Space. Thus the instructions SET12;OUT; to read from the data space may not be used until a DS reservation, using OUT 13; has been made.

For an OUT 12 operation, N2 must contain a Q-type parameter of the form (1st sector number involved in the transfer)/L.C.A./H.C.A.

The sector is the programmer's sector number, referring to his program's reserved area of DS. The numbering is from 0 to (n-1) where n sectors have been reserved. The low and high core addresses specify that part of main store which is to hold the information read from the drum. No core area other than this is altered. The 1st sector specified in the counter position of the parameter in N2 will be written to the core area specified and further sectors will be transferred if necessary until this core area is completely filled.

Parity may be tested by OUT 14 after a read operation. If the read is still in progress when OUT 14 is obeyed, the program is not re-entered until the transfer is over; OUT 14 will set the test register if a parity failure occurred.

4·8 **OUT 11** The instructions SET 11;OUT; with a Q-type parameter in N2 as for OUT 12, allow the programmer to write information to the drum DS.

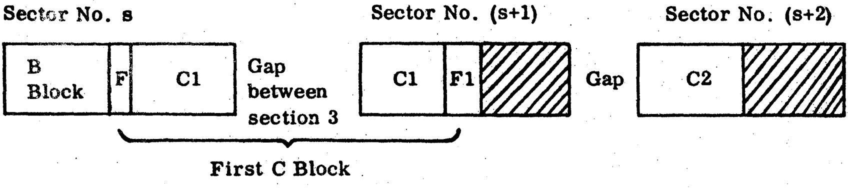
Thus OUT 11 may not be used until a reservation of DS, by OUT 13, has been made.

When OUT 11 is used, the core area specified will be written to the drum beginning at the 1st word of the specified sector and continuing into successive sectors if necessary until the whole of its core area specified has been transferred.

Any remaining part sector will then be filled with zeros, since writing cannot stop in the middle of a sector but only in the gaps between sectors.

4·9 OUT 15 This is used by the program-loading subroutine and is mentioned only for completeness. Programmers wishing to load programs to the drum will normally be expected to use the Library Subroutine provided for this purpose.

4·10 **Layout of Programs on the Drum** The diagram represents successive sectors in TPS or PPS.



The program begins in Sector s. The B-block with its filler word occupies the first eight words. The first C block will always be stored directly after the B-block on the drum, even though the filler word (see section 26·3·3 refiller-words) may indicate that a gap is to be left in the core store between E7 and the first word of the C-block.

The remaining C-blocks will be stored consecutively in succeeding sectors, each C-Block starting at the beginning of a sector. Any C-Block which occupies only a part of a sector, or which 'spills over' into part of a sector will have the remainder of that sector written up with zeros. The next C-block will begin at the start of the next sector and so on.

Although there is an inter-sector gap on the drum between sectors (e.g., between the two parts of C1, the first C-block) the parts are continuous when brought down into the core store. Any gap which may occur in the core store between successive C-blocks is determined only by the appropriate filler word. Thus the gap between C1 and C2 in the core store is determined by F1 and the shaded area bears no relationship to this gap in the core store. The shaded areas merely illustrate the filling up of the parially filled sectors with zeros.

Each program block i.e., the B-block and 1st C-Block together, followed by further individual C-Blocks, is written to the drum using OUT 15. This requires in N2 the core address (relative to E0) at which the program starts - it does not require the final address or the sector number. The final address is determined by the last filler word, whilst Director itself determines the sector number. It should be noted that when a program block is to be written to the drum by OUT 15, it does not have to be stored in the same part of the core store which it would occupy when brought back from the drum. (Where the term "program" is mentioned in connection with OUT 15 it is understood to be the compiled version of that program - similar to the "ontape" schedule in section 26·3·2).

When the first block of a program is stored on the drum, OUT 15 will check that its 3rd and 4th words, contain a proper program name, and will insert the letter 'D' as the 3rd character of the first of these two words. OUT 15 will not assume that a program has been completely written to the drum until it has written a C-block which follows a filler word with zeros in D0-D15. Until the program is completely written no magnetic drum OUT, except OUT 15 may be used. Once OUT 13 has been used, OUT 15 may not be used again. Thus the reservation of data space prevents the further expansion of TPS.

4·11 Failures Any logical failures will cause Director to terminate the program. Examples are:-

1. A negative sector number
2. L.C.A. greater than H.C.A.
3. Attempting to reserve more sectors than are available.

5· Graph Plotter

5·1 General There are four models of the graph plotter available for use on KDF 9 which from the programmer's standpoint have the following details.

Model Number	563	564	565	566
Plotting Length	120ft.	120ft.	120ft.	120ft.
Plotting Width	$29\frac{1}{2}$ins.	$29\frac{1}{2}$ins.	11ins.	11ins.
Step size	0·01ins.	0·005ins.	0·01ins.	0·005ins.
Pen Speed:Raise or Lower :Plotting	10/sec 200/sec	10/sec. 200/sec.	10/sec. 200/sec.	10/sec. 200/sec.

Plain or pre-printed (e.g., graph) paper is available.

The plotter holds up to 120ft. of paper so that many curves may be drawn between successive re-loading of the device. There are sprocket holes down both sides of the paper to facilitate accurate positioning. There are no perforations nor any other phusical divisions across the paper, so it would be possible to use the entire paper supply for one continuous picture.

Paper is fed from a supply reel, over a drum, and onto a take-up reel. The paper may also be allowed to hang over the front so that the graph or figure drawn may be seen. Above the drum, on guides parallel to the drum axis, runs a carriage upon which rides a pen. The pen may be lowered onto the paper or raised from it. The pen carriage may be moved across the paper, but not along it. Very small paper feed movements are possible. When the pen is lowered, drawing is achieved by combining pen carriage movement and paper feed.

Paper feed may take place in either direction as required, which is a most useful feature of this type of plotter and one which makes it fundamentally different from other output devices, e.g., a line printer.

5·2 Principles of Control The plotter is a digital incremental plotter, i.e., commands for pen carriage movcment and paper feed are of the type "move a single step from the present position".

The step size is fixed for any one model of the plotter, and is the same size for paper feed and for pen carriage movement.

A single command may be used to achieve either paper feed, or pen carriage movement, or simultaneous paper and pen carriage motion thus giving a true diagonal movement of the pen across the paper.

Hence when the pen is lowered, one plotter command may draw a (Short) line in any one of eight standard directions each separated by 45°. (N.B. a "diagonal" command produces a line of $\sqrt{2}$ times the step size quoted for the model. The thickness of the line drawn is of the same order as the step size so that virtually a pure curve may be drawn.

Thge plotter is connected to KDF 9 via an eight hole paper tape punch buffer. A switch is provided to connect the buffer to either the paper tape punch or the plotter. If the switch is set to the plotter the instruction PMBQq; (where Cq contains the device number) will set the test register.

Commands to the plotter consist of 6 bits. Since this is the same size as characters sent out to other KDF 9 peripherals, plotter commands can be sent out in the normal manner using the instruction POAQq; (Cq has device number, Iq and Mq have the low and high core addresses respectively). EM(75_8) will cause a peculiar pen movement and will not terminate a transfer.

5·3 Plotter Commands The eleven types of command which may be sent to the plotter and their effects are listed below. No other characters may be output to the plotter since the effect is unpredictable.

Command Code

000 000	Dummy-no effect.	
100 000	Pen raise.	
010 000	Pen lower.	
001 000	Move pen carriage left (+Y).	
000 100	Move pen carriage right (-Y).	
000 010	Feed paper forward (+X).	
000 001	Feed paper back (-X).	
001 010	+Y and +X together	true diagonal
001 001	-Y and -X together	

5·4 Use of Algol The plotting of graphs is slow, being 200 commands per second, consequently it is recommended that commands should be output to an "off-line" magnetic tape for subsequent plotting.

There are organisation difficulties to be overcome when using the plotter, e.g., ensuring that one graph does not extend so far as to overlap the previous one on the paper. To prevent such calamities it is recommended that the plotter is only used with "KDF 9 Algol". For detailed usage the reader is referred to "KDF 9 Graph Plotter Users Manual".

6· The KDF 9 Disc File - Users Description

6·1 Physical Properties The KDF 9 Disc file Unit contains a number of magnetically coated discs (31" diameter) mounted on a vertical common shaft, rotating at 1000 r.p.m. There are 16 discs in the unit for storage, plus a baffle disc at top and bottom used for control purposes. Information is stored in circular tracks on both sides of the storage discs.

Access to the recording surfaces is by means of rigid sets of read/write heads mounted on movable positioners. There is one positioner for each disc. Each positioner carries 8 read/write heads, 4 above and 4 below the disc, thus allowing 8 tracks on a disc to be accessed without moving the positioner. The positioner can be moved into any one of 64 positions thus giving 512 tracks on each disc.

In order to reduce the effects of the large differences in the speed of the disc surface between the innermost track and the outermost track, the disc surface is divided into tw zones, an inner and an outer. There are equal numbers of tracks in each zone.

Each track is divided into distinct sectors; there are 16 sectors in an outer zone track and 8 sectors in an inner zone track.

Each sector will contain 40 KDF 9 words, and is preceded by an address header to identify the sector.

The storage capacity is therefore organised as follows:

1 Sector = 40 words
1 Inner track = 8 sectors = 320 words
1 Outer track = 16 sectors = 640 words
1 Position = 4 outer tracks + 4 inner tracks = 3840 words
1 Disc = 64 positions = 245,760 words
1 Disc file = 16 discs = 3,9432,160 words

Transfer of information to the disc is in multiples of 40 words (1 sector), any unused part of a sector being written with zero words. Transfer from the disc is also in units of 1 sector, and will cease when the area of main store given in the usual way via a Q store is filled. It is not possible to move the positioner during a transfer: the largest block of information transferable is therefore 640 words.

The disc file is a restricted device (as it is shared by all levels in a time-sharing machine) and access can only be obtained via the Director program, using a set of OUT instructions.

The speed of 100 r.p.m. gives approximate transfer rates of:-

Outer zone 85,000 characters/sec (10,600 words)
Inner zone 42,000 characters/sec (5,300 words)

The time to move a positioner varies, dependent on actual movement, but average figures are:-

Access record in adjacent track	:	156 millisecs
Access any record	:	231 millisecs
Worst case access time	:	367·5 millisecs.

6·2 Users Aspects From the user's point of view, it is essential that he should have control over the movement of the positioners to allow some measure of optimum usage. Director therefore allocates space on the disc file to a user in units of complete discs, up to a maximum of 8 for one program. As the 16 positioners move independently one of the other, there is no inter-action between two separate programs.

To minimise the movement of the positioners, it is better to use what is known as "cylindrical addressing" over several discs rather than to use the discs in serial order. The cylindrical addressing technique involves working down from one disc to the next with all the positioners in similar positions until all discs allocated have been used: then all positioners are moved to the next cylinder.

To reduce the effort required on the part of the user, the concept of "logical blocks" is introduced. The program claims a set of discs, and thereafter refers to the logical block by number. Director translates the number into the actual disc address before initiating the transfer. For each logical block, a record is kept by Director of whether any part of it has been written to: if an attempt is made to read from a logical block that has never been written, Director will give a failure without having to read an unwritten area. It is not possible to keep this check at sector level, so the writing of **part** of a logical block will allow the subsequent reading of any or all parts of it, with a risk of a catastrophic parity error. It is advisable to use the disc only in units of logical blocks to reduce this possibility.

The numbering of logical blocks is in the order:-

0-3	First disc, outer zone
4-5	First disc, inner zone
6-9	Second disc, outer zone
10-11	Second disc, inner zone
12-15	Third disc, outer zone etc.

The positioners start from the outermost track.

The relationship between logical block number (b), position (p), disc used (d) and head used (h) for a set of D discs are:-

$\frac{b}{6D}$ gives p (integer quotient)
+r (integer remainder)

$\frac{r}{6}$ gives d (integer quotient)
+h (integer remainder)

If $0 \leqslant h \leqslant 3$, an outer zone head is used
If $4 \leqslant h \leqslant 5$, an inner zone head is used

Each logical block is divided into 16 sectors of 40 words each. Any transfer can specify any sector as the starting point, provided the transfer does not extend beyond sector 15. As the length of the transfer is specified in the usual way (low and high addresses in a Q store format word) there is a relationship between the starting sector and the addresses.

For a valid transfer,

Start sector no. $\leqslant 16$ - (high address - low address + 1)/40

i.e. There are 6 logical blocks for each position
384 logical blocks on each disc.

6·3 **Claiming of Discs - OUT 44** Any users may at any time have two sets of discs allocated to his program, providing the total number of discs allocated to a single program at any time does not exceed 8. The two sets are known as:-

(a) the FIRST set - that which was claimed earliest

(b) the SECOND set - that which was claimed latest.

Note that the subsequent deallocation of the first set will promote the other.

Discs are claimed using OUT 44; i.e., by obeying the instruction OUT when:-

N1 contains the integer + 44

N2 contains the integer n (to claim n discs)

The integers in N1 - N2 will be removed by Director whilst obeying the OUT; instruction.

The sequence of instructions:-

SET + 5; SET + 44; OUT; SET + 3; SET + 44; OUT;

will claim 2 sets of discs, the FIRST has 5 discs and the SECOND has 3 discs.

6·4 Deallocation of Discs - OUT 45 Discs can only be deallocated in SETS: the integer placed in N defines which set is to be deallocated.

Deallocation is performed by OUT 34, i.e., by obeying the instruction OUT: when:

N1 contains the integer 45
N2 contains wither zero to deallocate the FIRST set or the negative integer -1 to deallocate the SECOND set.

The integers in N1 and N2 will be removed by Director whilst obeying the OUT; instruction.

Note that deallocating the FIRST set promotes the SECOND set: since only one set is now allocated, it becomes the FIRST in all subsequent operations.

A failure will occur if an attempt is made to deallocate a set of discs not previously claimed by OUT 44.

6·5 Select Disc Set - OUT 43 Any transfer to or from a disc does not in itself define which of the two possible sets is the correct one to use. OUT 43 is d designed to define which set is currently used.

The instruction OUT: when

N1 contains the integer + 43
N2 contains either zero for the FIRST set or the negative integer -1 for the SECOND set.

defines which set of discs is to be used for all data transfers until a further selection is made.

The instructions:-

ZERO; SET + 43; OUT;
prime Director to use the FIRST set of discs until further notice. The integers in N1 and N2 will be removed by Director whilst obeying OUT;

Director is initially primed to use the FIRST disc set; if this is the only set claimed, there is no need to use the Select option, but an unnecessary use will not cause a failure.

6•6 **Write Data to Disc - OUT 41** The maximum size of transfer permitted is 640 words (1 logical block), and this implies starting at sector zero. If the start is not at sector zero, the size is reduced by 40 words for each missing sector. In the general case, for a start at sector S where 0≤S⩽15.

Block length ≤40 (16-S) words

The write is performed by obeying the instruction OUT; when:-

N1 contains the integer + 41
N2 contains the parameter word

The parameter word (basically in Q store format) contains

D0-11: logical block number
D12-15: Starting sector number within block
(these two share the COUNTER part in Q store format)
B16-31: lower address of main store (Increment)
D32-47: Higher address of main store (Modifier)

Note: Counter = 16 x (logical block number) + starting sector no.

Disc must be claimed using OUT 44, before this instructions is obeyed.

If the main store area defined is not a multiple of 40 words, the remaining part of the partly filled sector will be filled with zero words.

A failure will occur if the transfer will extend beyond sector 15.

Example V0 = Q 86/AY0/AY63;

SET + 1; SET + 44; OUT; SET + 1; SET + 44; OUT;
SET - 1; SET + 43; OUT;
V0; SET + 41; OUT;

This will:-

(a) Claim 2 sets of discs (1 disc only per set)

(b) Select SECOND set

(c) Write YO-63 to sectors 6 and 7 of logical block 5 on disc set 2, and pad out with 16 zero words (86 = 5 x 16 + 6).

6•7 **Read Data from Disc - OUT 42** The maximum size of transfer permitted is 640 (1 logical block) and this implies starting at sector zero. If the start is not at sector zero, the maximum length is reduced by 40 words for each missing sector. In the general case, for a start at sector S where 0 S 15

Block length ≤40 (16 - S) words

A read is performed by obeying the instruction OUT; when:

N1 contains the integer + 42
N2 contains the parameter word

The parameter word (basically in Q store format) contains:-

D0-11: logical block number
D12-15: Starting sector number within block
(these two share the COUNTER part in Q store format)
D16-31: lower address in main store (Increment)
D32-47: Higher address in main store (Modifier)

Note: Counter = 16 x (logical block number) + starting sector number.

Director will check that the number of words requested (high address - low address + 1) does not extend byond the end of sector 15 of the logical block: a failure will result before readin the disc if this check fails.

A catastrophic disc parity failure in Director is liable to occur if an attempt is made to read a sector that has not been written by the program concerned.

Out 44 must have been used to claim a disc before OUT 42 is obeyed.

Example

```
V0 = Q80/AY0/AY639;
V1 = Q86/AY128/AY169;

SET + 1; SET + 44; OUT;
V0; SET 41; OUT;
V1; SET + 42; OUT;
```

This will:

(a) claim 1 disc for the first set

(b) Director assumes FIRST set in the absence of specific directives

(c) Write 640 words to logical block 5.

N.B. The programmer never knows which actual disc(s) have been allocated to him. Therefore, he cannot assume anything concerning the information he will find there. This means that, once a logical block has been written, any sector in it is available for reading or writing only until the disc containing the logical block is deallocated.

(d) Read 42 words from sectors 6-7 of logical block 5: the remaining 38 words of sector 7 will not be transferred.

6·8 Write Programs to Temporary Program Space on Disc - OUT 46

This will write a single program block to a suitable place on the FIRST set of discs claimed by the program. One set of discs must be claimed before attempting to write programs, but the read, write or select options (OUT 41, 42 or 43) must NOT be used prior to any use of this facility. The space available for programs is closed immediately a read, write or select operation is initiated.

The program is assumed to be in standard binary format - a B block of 8 words with a standard filler word in word 8, and one or more C blocks with filler words in the last word of all except the last (just as on paper tape).

The first occurrence of OUT 46 is assumed to relate to a B block: this is written to the disc, and the filler word remembered, to allow Director to compute the length of the next C block.

The next occurrence of OUT 46 is a C block: Director already knows its length, and whether it is the last C block or not (from the previous filler word), so it is written, starting on a sector boundary.

Note: that for all C blocks, ONE SPARE WORD is required by Director preceding the block: this will be changed in the Program area by Director.

The parameter for OUT 46 is the address of this spare word: it should NOT however be included in the block defined by the filler words, as it will not appear when the program is subsequently loaded.

Note that the filler words relate to the location of the program when it is subsequently loaded: the parameter for OUT 46 relates to the current location which may be different.

The instruction OUT; obeyed when:-

N1 contains the integer + 46
N2 contains a store address

will:-

(a) on first occurrence, or on the occurrence following the writing of the last C block of the previous program (defined by marker in filler word), write a B block of 8 words starting from the location specified in N2, and use the 8th word as the next filler.

(b) on other occurrences, write a block of length (final address) - (initial address) + 2 words starting from the location specified in N2. The initial and final addresses are obtained from the filler word of the previous block: the last word written is assumed to be filler word for the next block, unless this is the last C block of the program.

The integers in N1, N2 will be removed by Director whilst obeying OUT 46.

Note that the maximum length of a C block is limited to:-

(a) For first C block - 3758 + 1 (filler) + 1 (spare)

(b) For last C block - 3839 + 1 (spare)

(c) For other C blocks - 3838 + 1 (filler) + 1 (spare).

For efficient use of disc space, the maximum size possible should be used. Each C block written will then occupy 6 logical sectors - the reduced size of the first C block allows space for the B block and some control information used by Director. Thus if N program C blocks are written to the disc, using the rules given above, 6N logical sectors will be occupied.

The program will then have 384D-6N logical blocks left for data, on a set of D discs.

However, to make data transfers easier, the block called logical block zero by the program will be stored following the program space: a correction factor will be added by Director to put logical block zero into the next available position, that is an exact multiple of 6, to make the timings still predictable. The only problem left for the user is to ensure that the number of logical blocks used does not exceed the limit given above.

It is possible to read the program back from this disc using data transfers - the adjustment to logical block numbers makes it impossible. However, an OUT 1; instruction to Director, with the program identifier having a U in the third character position (U = User program) will cause a program in temporary program space to be loaded and entered.

When the program that loaded other programs to the disc using OUT 46, or any of its logical successors called in by OUT 1, terminate via OUT 0 or by a failure termination, the temporary program space is deallocated, and will not be available to any other program.

Table for calculating addresses for OUT 46

	First Block	General	Last Block	Example (problem below)		
No. of words to write L.	L	L	L	8183	4425	587
First word used at E	E	E	E	8	3766	7604
First word now at Y	Y	Y	Y	1	3759	7597
Filler address (low)	E	E	E	8	3766	7604
Filler address (high)	E + 3758	E + 3838	E + L - 1	3766	7604	8190
Final word written/block	Y + 3758	Y + 3838	Y + L - 1	3759	7597	8183
Parameter address	Y - 1	Y - 1	Y - 1	0	3758	7596
Next L	L - 3758	L - 3838	0	4425	587	0
Next E	E + 3758	E + 3838	-	3766	7604	-
Next Y	Y + 3758	Y + 3838	-	3759	7597	-

Notes: L is the number of words of program still to write at each stage
E is the address to be occupied by the next word to be written when the program is **obeyed**
Y is the address currently occupied by the next word to be written
Filler addresses are those to go in increment and modifier positions of filler word.
Parameter address is the address required in N2 for OUT 46 - the spare word for Director use.
Final word written is the one that is to contain the filler word (it will require special treatment).

Example (refer to example in table above)
A program is in store, with its B-block in locations YB0 to YB7, and the body in locations Y1 to Y8183.

It is required to write this to the disc, with fillers to put the body into locations E8 to E8190 (the B block must as always go to E0 to E7).

```
V0  =  Q - 1/8/3766;                       first filler, 3759 words
V1  =  Q - 1/3766/7604;                    second filler, 3839 words
V2  =  Q 0/7604/8190;                      third filler, 587 words

SET + 1; SET + 44; OUT;                    claim disc set.
V0; = YB7;                                 store filler for first C block
SETAYB0; SET + 46; OUT;                    write B block
Y 3759;                                    preserve word of program
V1; = Y3759;                               replace by second filler
```

```
SET AY0; SET+46; OUT;               write first C block
= Y3759;                            replace program word
Y7597;                              preserve word of program
V2; = Y7597;                        replace by third filler
SET AY 3758; SET + 46; OUT;         write second C block
=Y7597;                             replace program word
SET AY 7596; SET + 46; OUT;         write last C block
SERO; SET + 43; OUT;                close program space.
```

7· Miscellaneous Instructions relating to Input/Output

TLOQq; Will cause the Test Register to be set if any part of the area of main store area defined, as usual, by the increment and modifier positions of Qq, is locked out. The counter of Qq is ignored.

BUSYQq; will cause the Test Register to be set if the device specified in the counter position of Qq is still busy. This can be useful to the programmer, enabling him to program to perform other operations until the device is once again available.

INTQq; (called "interrupt if busy" is intended for time-sharing machines only), causes Director to be entered (thus allowing a lower level program to be obeyed) if the device specified in the counter position of Qq is busy, but on return to the program does NOT try to obey the same instruction again. The program will continue when any one of its devices ceases to be busy. As it moves to the next instruction, all devices can again be inspected using BUSY to find which to use next.

An example of BUSY and INT where the program uses three devices is

```
1. BUSYQ1; J2TR;  (PRINT)Q1;
2. BUSYQ2; J3TR;  (PRINT)Q2;
3. INTQ3;   J1TR;  (PRINT)Q3;
```

In this arrangement if devices C1, C2, C3 are all in use then after INTQ3 Director is interrupted and another program until ANY one of C1, C2 C3 are free when control returns to this program and J1TR is obeyed.

MANUALQq; sets the device whose number is in the counter of Qq into an already state (i.e., under operator control), which prevents ANY further operation on that device from starting until manual reset. It does not, however, stop an operation already started.

8· POEQq and POFQq (MGAPQq & MWIPEQq) on 1081 Magnetic Tape Decks ONLY

These rules do NOT apply on the other taypes of tape desk.

It may sometimes be essential to overwirte one block on a magentic tape during an updating sequence, rather than rewrite the complete tape. This is possible provided the tape originally written in a form to allow overwriting, and certain rules are obeyed. The derivation of these rules involves many factors in the engineering of the tape system - they will only be stated here without any attempt to justify them.

The two instructions required if overwriting is to be possible are:

MWIPEQq; Write a gap on tape equivalent in length to a block of m words, when m is given in the modifier of Qq. Overwrites previous contents of tape.

MGAPQq; As for MWIPE but the erase will stop BEFORE the tape stops, thus complete removal of previous information is not possible. For this reason the tape MUST always stop in a gap previously made using MWIPE.

An unchanging rule for the use of these instructions is "Use MWIPE only when first writing a tape, use MGAP in all subsequent overwriting instructions".

The lengths of the gap required are functions of the mechanical features of the tape system are also of the length of the block being overwritten. The values to be used are:-

$g = 0\cdot 11B+8$

$G = 3g+60$

Where g is the gap for MGAP
G is the gap for MWIPE
B is the block length in words.

It should also be noted that overwriting of every block on a tape is not allowable: a sentinel block (of any length) must be written before each block that may be overwritten, to provide a reference point from which to start the overwriting operation. Such sentinel blocks can with profit contain a block number, to ensure that the correct block is about to be overwritten.

The normal sequence of events for overwriting a tape (ignoring all checks etc. for the purpose of illustration) is:-

(a) First writing:

MWQq	write fixed sentinel block
MWQq;	write block for subsequent overwriting
MWIPEQq;	write gap of (3g+60) words

(b) Subsequent overwriting:

MFRQq;	read sentinel block to check position
MWQq;	overwrite block
MGAPQq;	write gap of g words.

If the block to be overwritten is the first on the tape (i.e., the label block of up to 16 words), the values used may vary to allow for the increased delay on writing from BTC.

The sequences are:-

(a) Write label (for subsequent overwriting) before writing new tape

MRWDQq;	rewind tape
MWQq;	write label
MWIPEQq;	write initial gap of 526 words

(b) Read existing label, leave space for overwriting then write reset of tape

MRWDQq;	rewind tape
MFRQq;	read label (or skip over it)
MWIPEQq;	write gap of 542 words.

(c) Overwrite label without rewriting tape

MRWDQq;	rewind tape
MWQq;	overwrite label
MGAPQq;	write gap of 232 words.

The programmer is advised.

(a) not to use these two instructions if there is the possibility that his magnetic tape may not always be run on 1081 deck.

and

(b) to only use them if the effect cannot be obtained by other instructions.

9• Other Devices

Programming details for the other peripheral devices will be included in this appendix at a later date.

APPENDIX 7 PROGRAMMING FOR TM 4 ON KDF 9

1· I.B.M. Magnetic tapes are of two types, binary tapes which are written with ODD parity and alpha-numeric tapes which are written with EVEN parity.

1·1 Binary tapes are recorded and read using the following instructions:- (ODD parity)

1·1·1	Forward Read	MFRQq	PIAQq	124 2q, 0
1·1·2	Backward Read	MBRQq	PIEQq	126 2q, 0
1·1·3	Write	MW Qq	POAQq	130 2q, 0
1·1·4	Write Tape Mark	MLWQq	POCQq	130 (2q+1), 0
1·1·5	Forward Skip	MFSKQq	PMAQq	134 2q, 0
1·1·6	Backward Skip	MBSKQq	PMEQq	136 2q, 0

1·2 Alpha-numeric tapes are recorded and read using the following instructions:- (EVEN parity)

1·2·1	Forward Read	"to Group Mark"	MFREQq	PIBQq	125 2q, 0
1·2·2	Backward Read	"to Group Mark"	MBREQq	PIFQq	127 2q, 0
1·2·3	Write	GROUP MARK	MWEQq	POBQq	131 2q, 0
1·2·4	Write Tape Mark	TAPE MARK	MLWEQq	PODQq	131 (2q+1), 0
1·2·5	Forward Skip	Even Parity	-	PMKQq	134 2q, 4
1·2·6	Backward Skip	Even Parity	-	PMLQq	136 2q, 4

1·3 The Erase, Rewind and Test instructions are identical for both types of tape and are identical to - and have the same effect as - the KDF 9 1081 tape system.

2· The I.B.M. Tape Mark is a one character block, the character being $17_{(8)}$ with even or odd parity as appropriate.

2·1 This is produced by the user code Write Last Block instruction. The Q store specified must contain the device address and an initial and final address. The initial and final address have no significance but must be present; they may conveniently be those relating to a previous transfer.

2·2 The tape mark is read as a block and overwrites the first word of the main store as specified by the initial address.

2·3 The instruction MLBQq (PMCQq) test for the tape mark having been read in the forward direction only. Tape marks are not read in the reverse direction.

3• The I.B.M. Group Mark (character $77_{(8)}$) terminates alpha-numeric transfers (EVEN Parity) in a manner analogous to the KDF 9 End Message character.

3•1 On I.B.M. systems the group mark is noι physically recorded. It is in fact only a character marker within the main store defining the end of the transfer area. This is unlike the KDF 9 End Message termination of alpha-numeric transfers since in the latter case the end message character is actually recorded.

As a result, EVEN Parity tapes produced on the I.B.M. SYSTEM (1401 - 7090 - Stretch) if written to Group Mark will contain any number of characters, not necessarily modulo eight characters.

3•2 When writing even parity tapes via. KDF 9/TM 4 using the instruction MWEQq the character $77_{(8)}$ which is the I.B.M. Group mark (held in store to define the transfer area) will terminate the transfer when the last character of the word contaning the group mark character has been recorded on tape. (N.B. Unlike I.B.M. systems which do not record the group mark).

4• In I.B.M. 1401 Systems it is not possible to write other than immediately following a previous write operation. It is possible however to do this on KDF 9/TM 4 providing provision has been made after the previous write instruction. A length of tape must be erased using the User Code instruction MWIPEQq after the last block written. Back skipping (or reading) followed by a forward (re) read of the last recorded block will permit further recording.

A one-word length of erased tape is sufficient.

5• It should be noted that the instruction MLWQq and MLWEQq produce a one-character block which must be recorded with the correct lateral parity, MLWQq for binary (odd parity) tapes and MLWEQq for alpha-numeric (even parity) tapes.

Note: Any such data interchanges between magnetic tapes for different computers must conform to the standards as laid down in ECMA5.

NOTES

NOTES