DAHL, OLE-JOHAN / JAN V. GARWICK

PROGRAMMER'S HANDBOOK

for the

FERRANTI MERCURY COMPUTER

FREDERIC

PROGRAMMER'S HANDBOOK

for the

FERRANTI MERCURY COMPUTER

FREDERIC

at the

Norwegian Defence Research Establishment

(SECOND EDITION)

by

O-J DAHL and Dr JAN V GARWICK

1958
Merkantile Trykkeri - Oslo

PREFACE

This book is the second edition of the Programmer's handbook for the Ferranti Mercury computer. It describes the machine and the input system, the latter consisting of the input programme, the test programme and the facilities for correcting programmes.

The basic ideas of the input programme are proposed by Ferranti Ltd., but these ideas have been modified somewhat and considerably extended as a result of the programming experience accumulated at the Norwegian Defence Research Establishment.

Kjeller, August 1958

Ole-Johan Dahl *Jan V. Garwick*

This book and the tapes for the input system herein described are obtainable free of charge from the Norwegian Defence Research Establishment, Kjeller pr. Lillestrøm, Norway.

CONTENTS.

		Page
1.0	*Introduction*	1
1.1	Notation	1
2.0	*Computing registers and addresses*	3
2.1	The store	3
2.2	The computing registers	4
3.0	*The orders*	5
3.1	The form of an order	5
3.2	The B-orders	6
3.3	The B-modification of orders	7
3.4	The SAC-orders	7
3.5	The jump orders	9
3.6	The input and output orders	10
3.7	The accumulator orders	12
3.7.1	Jump orders associated with the accumulator	14
3.8	Exponent orders	15
3.9	Magnetic drum orders	15
3.10	Timing	16
4.0	*The input programme*	17
4.1	Orders	18
4.1.1	The address part	18
4.1.2	Floating addresses	20
4.1.3	Routines	22
4.1.4	More about flads	23
4.1.5	Mixed addresses	24
4.1.6	Addresses in n-type orders	27

		Page
4.1.7	H-type orders	28
4.1.8	Summary	30
4.1.9	Automatic checks on flads	31
4.2	Input of long numbers	32
4.3	Input of short numbers	32
4.4	The warning characters	34
4.5	Programme organization	35
4.5.1	W.ch. R (ROUTINE)	35
4.5.2	W.ch. C (CHAPTER)	38
4.5.3	W.ch. P (PAGE)	39
4.5.4	W.ch. F (FIRSTSECTOR)	40
4.5.5	W.ch. S (SECTOR)	40
4.5.6	W.ch. J (JUMP)	40
4.6	The chapter changes	41
4.6.1	W.ch. A (ACROSS)	42
4.6.2	W.ch. D (DOWN)	42
4.6.3	W.ch. U (UP)	43
4.6.4	W.ch. Z	44
4.6.5	The layout of P0, the CCS	44
4.7	Aids to programming	46
4.7.1	W.ch. Q (QUICKIE)	46
4.7.2	W.ch. G (GILLKUTTA)	52
4.7.3	W.ch. N	54
4.7.4	W.ch.'s K (KOMPLEX)	55
4.7.5	W.ch.'s L (LONG)	58
4.8	Miscellaneous	60
4.8.1	W.ch. B (BINARY)	60
4.8.2	W.ch. T (TITLE)	60
4.8.3	W.ch. £	61
4.8.4	W.ch. W (WAIT)	61
4.8.5	W.ch. E (ENTER)	61
4.9	Correcting programming errors, W.ch. X	62
5.0	*Use of storage space*	65
6.0	*The starting of the machine*	68
6.1	H.s. 9. Decimal input	68

		Page
6.2	H.s. 8. Binary input	69
6.3	H.s. 7. Binary output	69
6.4	H.s. 6. Output of storage allocation	70
6.5	H.s. 5. Output of title	70
6.6	H.s. 4. Test programme	71
6.7	H.s. 3. Preserve directories. (W,ch, X)	71
6.8	H.s. 2. Store — restore	71
6.9	H.s. 0. Stop	73
7.0	*The test programme*	73
7.1	Aims of the TP	73
7.2	Entry of the TP	73
7.3	Definition of a break point	74
7.4	Output directives	76
7.5	Form of output	78
7.6	Limitations of the TP	79
7.7	Fault indications	80
8.0	*Programming*	80
8.1	Special instructions	81
8.2	A fully worked example	83

0 INTRODUCTION

FREDERIC is a binary electronic computer and all numbers and orders inside the machine are therefore expressed in the binary system. As it is rather inconvenient to write data on paper and later punch it for input into the machine in this system, it is preferred to operate outside the machine in the decimal system. The input equipment will translate each decimal digit (and other characters we may want to use) into 5 bit numbers (one bit is one binary digit). A certain input programme in the machine will then operate upon these short binary numbers and assemble the complete numbers and orders.

An input programme being needed in every case, it may as well be an elaborate one, performing a great number of clerical tasks which will simplify programming for the programmers. Besides a description of the machine, this book will mainly contain a discussion of the various features of the NDRE input programme and the associated test programme.

It is supposed that the reader is acquainted with the principles of electronic computing as the difficulties of drawing up a programme will only be mentioned briefly.

1.1 *Notation*

For later use, we collect here the notations to be defined and used in the description of the machine. The contents of some part of the machine are denoted by a letter, mostly a capital

letter, often followed by a number. At other occasions, these letters denote the corresponding part of the machine, but no confusion should be possible.

A prime on a letter denotes that the contents are considered after an order has been executed.

A	= Floating accumulator
B 1	= B-register no 1
.....	
B 7	= B-register no 7
S	= Short accumulator (SAC)
B t	= B-test register
S t	= SAC-test register
L m	= Long word no m
M m	= Storage register no m
H m	= Half register (short word) no m

(The number m is written as one would write the address of an L-, M-, or H-type order)

C	= Control register
n	= a ten bit number
T	= Sector register or the selected sector on the drum
P	= Page in the high speed store
hs	= Hand switches
t	= Tape row under reader, or tape row on punch staticisor

An operation inside the machine is described by means of the prime notation, or by using an arrow. For instance

$$S' = n \quad \text{and} \quad n \rightarrow S$$

both mean that a ten bit number n is put into SAC.

0 COMPUTING REGISTERS AND ADDRESSES

1 *The store*

FREDERIC contains a computing store holding 40960 bits. The first 20480 bits are divided into 2048 words (short words, integers) each of 10 bits. They are numbered consecutively from 0 to 2047. We have thus defined one of the address systems in the machine. These words are denoted by H.

Two and two words, the first with an even, the second with an odd address can be combined to form «registers». Each register can contain one order (instruction). The registers are numbered from 0 to 1023. The address of a register is therefore half the word-address of the lowest numbered (first, left hand) word it contains.

This is the second address system of the machine. The registers are denoted by M.

Two and two registers, the first with an even, the second with an odd address can be combined to form a «long word». Each long word can contain a «long number» (floating number) to be defined later. The long words are numbered from 0 to 1023 and their addresses are found by dividing the address of the first word in the long words by 4 or the address of the first register by 2. This is the third address system used in the machine. These double registers are called L.

The long words can be placed anywhere in the computing store, and the L system is therefore usable everywhere. The H and M systems have no meaning in the second half of the computing store.

The store is divided into 32 pages (P), numbered from 0 to 31. Each page contains 128 words, 64 orders or 32 floating numbers.

This makes 4 address systems in all referring to the computing store of the machine, the H-, M-, L-, and P-systems. Which of them to use depends upon the type of

instruction, whether it has to do with short numbers, orders, long numbers, or pages.

Besides the high speed store, the machine has a magnetic store (4 magnetic drums). Each track on a drum is divided into two sectors (T). There are 512 sectors on the drums. Each sector contains the same amount of information as one page in the high speed store. Communication between the two types of store is possible only by transferring an amount of data contained in one page and one sector. The sectors are numbered consecutively from 0 to 511.

2.2 *The computing registers*

The machine contains two types of computing registers, 7 B-registers which operate upon short words and therefore can contain 10 bit numbers, and the accumulator which operates upon floating numbers. The B-registers are also used for address modification as described later.

The computing facilities associated with the B-registers are: copying of numbers into the register from the store, and vice-versa, addition, subtraction, the logical operations & and ≠ and finally halving of the contents of a B-register.

Further the contents of the B-registers may influence the course of a computation. Alternative ways may be followed depending on

a) the sign of the contents of a B-register, or
b) the contents being zero or non-zero, or
c) the sign of the difference between the contents of a B-register and another number, or
d) this difference being zero or non-zero.

The accumulator can receive long numbers from the store and put them back into the store. Numbers may be added, subtracted and multiplied. There are several variants of these operations and they are described later.

Besides the orders connected with the computing registers, there are orders connected with the control register (C). This

register contains the register address of the next order to be executed. Finally there are orders controlling the input, the output and the communication between the two types of store.

0 THE ORDERS

1 *The form of an order*

The two words of a register containing an order comprise the function part (left) and the address part (right) of the order. The 10 bits of the function are divided into two parts. The 7 most significant digits give the function itself, the 3 least significant give the B-number. The B-number may either indicate the B-register to be operated upon, or it may point out the B-register which modifies the address of the order. The meaning depends upon the 7 function digits. The address part usually contains the address of the number to be operated upon in the proper address system.

The 7 function digits are denoted outside the machine with a decimal two-digit number ranging from 00 to 99. As there are less than 100 different orders, superfluous combinations have all been given the meaning of stop orders. From the two-digit number indicating a function, the true function digits can be found from a table, but there is no simple relation between the two representations of a function.

There are 4 main types of orders. The type depends upon whether the address is in the word system, the register system or the long word system, or if the order is without an address but operates upon the number which is placed where the address should normally be. These types are denoted as H-type, M-type, L-type and n-type respectively.

3.2 *The B-orders*

We denote B register number m by Bm. B0 does not exist and may be considered to contain zero at all times, irrespective of what is put into it.

In the B orders the B-number tells which B-register shall be operated upon. Associated with the B-registers is one B-test register (Bt). The use of this register will be discussed later.

In the following description the B-number is usually left out. When using the orders, a B-number (even if it is zero) must always be put after the two function digits.

Function	Effect	
00	$B' = Bt' = H$	Put number into B-register
01	$H' = B$	Store contents of B-register
02	$B' = Bt' = B + H$	Add in B-register
03	$B' = Bt' = B - H$	Subtract in B-register
04	$B' = Bt' = \frac{1}{2}B - H$	Halve B-register
05	$B' = Bt' = B \,\&\, H$	Logical multiplication (extraction)
06	$B' = Bt' = B \not\equiv H$	Logical non-equivalence

H is the short word specified by the address of the order.

The order 04 considers B as an unsigned integer in the range $0 \leqslant B \leqslant 1023$. Fractions appearing in the halving are dropped. The result of the halving is less than 512 and therefore always positive (see 3.4 for a definition of the sign of a ten bit number). These orders are all of the H-type.

We have previously mentioned that the store contains 2048 short words, but that the address of an order only contains 10 bits. Only the first 1024 words (numbered from 0—1023) are therefore directly addressable. The next 1024 words are, however, addressable by having a second set of all the functions where the address is increased by 1024. One may consider this as moving the most significant address

digit into the function (in fact into the most significant bit of the function). There is, however, no need to have two different names for these functions. If the address of a function is greater than 1023, the input programme, to be described later, automatically changes the function to the variant with a one in the most significant digit.

There is a second set of B-orders where the operand itself is put in the address part of the order. If the address is n, these orders are

Function	Effect
10	$B' = Bt' = n$
11	Stop
12	$B' = Bt' = B + n$
13	$B' = Bt' = B - n$
14	$B' = Bt' = \frac{1}{2}B - n$
15	$B' = Bt' = B \,\&\, n$
16	$B' = Bt' = B \not\equiv n$

These orders are of the n-type.

3 *The B-modification of orders*

Nearly all the orders not mentioned in the previous section are B-modifiable. This means that the address part is increased by the contents of the B-register specified by the B-digit, before the order is executed. If the B-digit is zero, no B-modification takes place (B0 is always zero).

4 *The SAC-orders*

SAC or S (short accumulator) is physically identical to B7 and can be operated upon by the orders in 3.2, but there exists a set of orders operating only upon S.

The B-digit in the function is in this case free, there being no need of identifying the B-register to be operated upon. It can therefore take its usual meaning of indicating the B-

register whose contents are added to the address part of the order before the order is executed.

Associated to S is a test register St comparable to Bt.

The S-orders are

Function	Effect
20	S′ = St′ = H
21	H′ = S
22	S′ = St′ = S + H
23	S′ = St′ = S — H
24	S′ = St′ = ½S — H
25	S′ = St′ = S & H
26	S′ = St′ = S ≠ H

H is here the short word indicated by the address of the order (after the contents of the specified B-register have been added to the address).

These orders are of the H-type.

Just as for the B-orders, the SAC-orders are in two variants. Whereas this did not trouble the programmer any for the B-orders, it does have a peculiar effect when B-modifying a SAC-order. The modifying register has only 10 bits and may be considered to be in the range 0 to 1023 or —512 to 511. For the purpose of B-modification, the latter is used. If therefore the address of a SAC-order is e. g. 1000, the words from 1000 — 512 = 485 to 1000 + 511 = 1511 may be «reached» by B-modification. The first half of the store is cyclic with respect to H-addresses, so that if the address is, say 2000, one may «reach» the addresses from 2000 — 512 = 1488 to 2047 and further from 0 to 2000 + 511 — 2048 = 463.

There is a second set of S-orders where the address part (n) itself is operated upon. If a B-register different from B0 is specified by the B-digit, the contents of this B-register are added to n before the order is executed.

The orders are:

Function	Effect
30	S′ = St′ = n
31	Stop
32	S′ = St′ = S + n
33	S′ = St′ = S — n
34	S′ = St′ = ½S — n
35	S′ = St′ = S & n
36	S′ = St′ = S ≠ n

These orders are of the n-type.

The jump orders

Usually the contents of the control register (C) (the address of the next order to be executed) are increased by one after the execution of each order. There are, however, some orders which put their own address part (n) into the control register forcing the machine to «jump» in the order sequence.

These jump orders are both unconditional and conditional. The conditional orders may depend upon the test-registers or upon conditions associated with the accumulator. This last case will be described later. (Cf. 3.7.1).

In the orders we have discussed so far, the test-registers contain the same number as the B-register last operated upon. There are, however, six orders which only operate upon the test-register. The first four are:

Function	Effect	Type
07	Bt′ = B — H	H
17	Bt′ = B — n	n
27	St′ = S — H	H
37	St′ = S — n	n

These orders are called comparison orders.

The two last orders can be B-modified.

We must now define the sign of the ten bit numbers in the test-register. If the most significant digit is 0, the number is positive, if the most significant digit is 1, the number is

negative. This corresponds to the usual convention of signed numbers. If we consider the numbers as being in the range 0—1023, this means that 1024 has been added to the negative numbers. With this convention, the test-registers treat their contents (n) as if they were in the range $-512 \leqslant n \leqslant 511$.

We can now make a list of the jump orders (except two which are associated with the accumulator).

Function	Condition	Effect	Condition	Effect
59	No	C′=n		
08	Bt≠0	C′=n	Bt=0	C′=C+1
09	Bt≥0	C′=n	Bt<O	C′=C+1
18	Bt≠0	C′=n, B′=Bt′=B+1	Bt=0	C′=C+1, B′=Bt′=B+1
28	St≠0	C′=n	St=0	C′=C+1
29	St≥0	C′=n	St<0	C′=C+1
38	St≠0	C′=n, S′=St′=S+1	St=0	C′=C+1, S′=St′=S+1

All these orders are of the M-type. They are all *except 18* B-modifiable. Observe further that 18 changes the contents of Bt and 38 the contents of St.

3.6 *The input and output orders*

The input to the machine is usually from teleprinter tape. If the five bit character under the reading head of the reader is t, the input order is

Function	Effect
60	H′ = t (zero to the five most significant bits of H)

We will here mention that the machine is equipped with two tape readers, TR2 which can read 200 characters per second and can, from full speed, stop on the next character, and TR3 which can read 400 character per second but may miss one character when being stopped. The use of these two readers will be discussed later.

Numbers may also be input from a set of ten handswitches (hs), viz. those in the bottom row on the console.

Function	Effect
61	H′ = hs

Output is usually by a tape punch. If the 5 bits which are punched, are called t′, we have

Function	Effect
62	t′ = least significant 5 bits of n
63	t′ = least significant 5 bits of H

60, 61 and 63 are of the H-type, 62 of the n-type.

It is important to know that the 63-order does *not* exist in the H + 1024 variant. Only words with addresses less than 1024 may therefore be punched using this order, if no B-modification takes place. If the order is B-modified, words with address up to 1023 + 511 = 1534 or down from 2047 to 0—512 (+2048) = 1536 may be punched.

There are also two cathode ray tubes where a long number may be displayed. If the number shown on the tubes is cr, we have

Function	Effect
64	cr′ = L

This order is of the L-type.

This order will only give a visible picture if it is enclosed in a loop and repeated with short time intervals. The switches under the monitors must point to the word DISPLAY.

A different type of «output» is supplied by a loudspeaker which will give a sharp click when the order 58 is encountered. We will here also mention the dummy order 57 which has no effect at all, except increasing the contents of the control register by one, and the stop order 99.

The B-numbers and address parts of the last orders have no influence upon the effects of the orders. However, the input programme will treat the address part as n-type addresses.

3.7 *The accumulator orders*

The floating numbers treated by the accumulator are of the form $x \cdot 2^y$ where $-256 \leqslant y \leqslant 255$ and x is a signed 29 bit number (30 bits in all) in the range $-1 \leqslant x \leqslant 1 - 2^{-29}$. If x is a negative number it is represented in the machine as $2 + x$ i.e. the usual complement of two convention. The binary point is considered to be between the two most significant digits.

A floating number is called standardized if the two most significant digits (the sign digit and the digit following the sign) are different. x is then in the range $-1 \leqslant x < -\frac{1}{2}$ or in the range $\frac{1}{2} \leqslant x < 1$.

In the accumulator a floating point number is stored in two registers, a 10-bit register for the exponent and a 30-bit register for the fractional part. In the store, the floating point number is stored in 4 consecutive short words, the address of the first being divisible by 4 (a long storage location). If the contents of these storage locations are a, b, c and d, and if we have

$$-256 \leqslant a \leqslant 255,\ 0 \leqslant b \leqslant 1023,\ 0 \leqslant c \leqslant 1023,\ -512 \leqslant d \leqslant 511,$$

then the corresponding long number is given by

$$(d \cdot 2^{-9} + c \cdot 2^{-19} + b \cdot 2^{-29}) \cdot 2^a$$

A number is put into or taken out of the accumulator by the orders:

Function	Effect
40	$A' = L$
41	$L' = A$

These orders are of the L-type and may be B-modified.

Addition takes place according to the following rules: If we shall add $x \cdot 2^y$ and $x' \cdot 2^{y'}$ the machine forms $2^y (x + x' \cdot 2^{y'-y})$ if $y \geqslant y'$. Otherwise the roles of the two numbers are interchanged. The least significant $y - y'$ bits

of x′ are discarded. If the addition is rounded, the least significant bit in $x + x' \cdot 2^{y'-y} = z$ is forced into a «one». If $z \geqslant 1$ or $z < -1$, $\frac{z}{2}$ is computed and rounded once more and the result is given as $\frac{z}{2} \cdot 2^{y+1}$. If, however, $-1 \leqslant z < 1$, a non-negative integer r is found, such that $z \cdot 2^r$ is standardized. The result is then given as $(z \cdot 2^r) \cdot 2^{y-r}$, no further rounding taking place.

The orders governing addition or subtraction are

Function	Effect	
42	$A' = A + L$	rounded
43	$A' = A - L$	rounded
44	$A' = A + L$	unrounded
45	$A' = A - L$	unrounded
46	$A' = A + 0.2^{n+1}$	unrounded and unstardardized

The first 4 orders are of the L-type, the last one of the n-type. n is in the address part of the order.

If, after an addition or a subtraction, before rounding, we have $z = 0$, the rounded operations will give the result $\frac{1}{2} \cdot 2^{y-29}$ and the unrounded versions 0.2^{-256}

The description of the order 46 is valid only if n is greater than or equal to the contents of the exponent register, y. In this case the fractional part, x, of the accumulator is shifted down, the exponent being increased simultaneously until the latter is equal to $n + 1$. The result is thus $(2^{y-n-1} \cdot x) \cdot 2^{n+1}$. If all significant digits of x are lost because of the shift, the result will be an ordinary exact zero, 0.2^{-256}. If we have $n < y$, the fractional part is shifted one down (i.e. halved) and y is increased by one, that is the contents of the accumulator are replaced by $\frac{1}{2} x \cdot 2^{y+1}$.

The multiplication orders are all of the types $A' = AL$ or $A' = -AL$, but several variants exist.

Function	Effect	
50	$A' = AL$	rounded and standardized
51	$A' = -AL$	rounded and standardized
52	$A' = (AL)_m$	unrounded but standardized
53	$A' = -(AL)_m$	unrounded but standardized
54	$A' = (AL)_l$	least significant 29 bits
55	$A' = -(AL)_l$	preceeded by 0. Exponent reduced by 29. Unrounded and unstandardized

These orders, which are B-modifiable, are of the L-type. A more exact description of the orders is as follows:

The numbers may be considered as signed 29 bit numbers. Their product is then a signed 58 bit number. As to the first four multiplication orders, the last 29 bits of the product are dropped and the first 29 bits are kept. The resulting number will be rounded by the orders 50 and 51, but not by 52 or 53. A standardization takes place in all cases.

The multiplications 54 and 55 give the 29 bits which were dropped in the first case, preceeded by a sign digit 0. The result is neither rounded nor standardized and the exponent is 29 less than the sum of the exponents of the factors.

If the accumulator as the result of an arithmetic order (not the order 40) obtains an exponent in the range —512 to —257, the exponent is automatically replaced by —256. If the exponent is in the range 256 to 511 the machine stops. It will resume operation if the prepulse button is pressed.

3.7.1 *Jump orders associated with the accumulator*

There are two jump orders associated with the accumulator, they are:

	Effect	Condition	Effect	Condition	Effect
48	$\lvert y - y' \rvert \leqslant 31$	$C' = n$	$\lvert y - y' \rvert > 31$		$c' = c + 1$
49	$A \geqslant 0$	$C' = n$	$A < 0$		$c' = c + 1$

$y-y'$ is the difference between the exponents in the last addition or subtraction. These orders are of the M-type.

3.8 *Exponent orders*

Function	Effect
70	$B' = Bt' = E + n$
71	$E' = B$
72	$B' = Bt' = B + E + n$
73	$B' = Bt' = B - (E + n)$
74	$B' = Bt' = \frac{1}{2}B - (E + n)$
75	$B' = Bt' = B \,\&\, (E + n)$
76	$B' = Bt' = B \not\equiv (E + n)$
77	$Bt' = B - (E + n)$

The B-digit will indicate which B-register is operated upon. The orders are of the n-type.

The second set refers to SAC and is as follows:

Function	Effect
80	$S' = St' = (E \vee B) + n$
81	$E' = S$ (see note)
82	$S' = St' = S + (E \vee B) + n$
83	$S' = St' = S - ((E \vee B) + n)$
84	$S' = St' = \frac{1}{2}S - ((E \vee B) + n)$
85	$S' = St' = S \,\&\, ((E \vee B) + n)$
86	$S' = St' = S \not\equiv ((E \vee B) + n)$
87	$St' = S - ((E \vee B) + n)$

The order 81 is really non-existent; the input programme will replace it by the order 717. If the B-number is different from zero, a stop order is put in.

The sign v is the logical sign «or». We have $0 \vee 0 = 0$, $1 \vee 0 = 1$, $0 \vee 1 = 1$, $1 \vee 1 = 1$. The 80-orders will usually be used with $B = 0$ when $E \vee B = E$.

3.9 *Magnetic drum orders*

At any time one and only one sector of the drum is «activated» by setting a sector-register (T). When T is set, we may «read» the contents of the corresponding sector on the drum (D) on

a page (P) in the high speed store or «write» the contents of a page on the drum. The necessary orders are:

Function	Effect
67	$T' = n$
68	$P' = D$
69	$D' = P$

These three orders are of the P-type. (The first one does not refer to the computing store, but there are still occasions when it is convenient to consider it a P-type order.)

If the address in the 68 or 69 orders are greater than 31 a peculiar phenomenon appears. If the address is $a + b \cdot 2^5 + c \cdot 2^7$ where $a \leqslant 31$, $b \leqslant 3$, and $c \leqslant 7$ the transfer will contain 32 long words as usual, but beginning with long word no. c on page a. The transfer can therefore begin with any long word in the first quarter of the page. This feature will probably only rarely be of any use.

3.10 *Timing*

All instructions operating upon B-registers or SAC take 60μs. The same time is taken by the jump instructions and the comparison orders. The handswitch to store instruction is likewise concluded in 60 μs. The input and output instructions will also take 60 μs if the tape is in position, but two $H' = t$ instructions can not be less than 5 ms apart, otherwise the machine waits. The waiting time for the output will be 30 ms. The sector activating order takes 120 μs. The time for a reading (from drum to high speed store) or writing (from high speed store to drum) operation involves a possible waiting time followed by a transfer time. The transfer time is 7.68 ms. Each drum-track contains two sectors and two gaps of 0.96 ms (16×60 μs) between the sectors. As a transfer can only commence at the beginning of a sector, one needs on the average a waiting time of ½ drum revolution or 8.64 ms. If several transfers shall be made and each transfer

to (or from) an even sector is followed by a transfer to (or from) an odd sector and vice versa, and if further the next transfer is initiated (sector activating and transfer order) before the end of the gap, the waiting time will only be the gap time (0.96 ms).

The accumulator orders take 120 μs for transfer to and from the accumulator, 180 μs for all types of addition and subtraction and 300 μs for all types of multiplication.

Table 1 in the appendix contains all the orders.

0 THE INPUT PROGRAMME

The input to the computer consists of

1) orders
2) long numbers
3) short numbers
4) various warning signals.

The items are normally punched on the tape in a sequence corresponding to the order of the positions they are to occupy inside the machine. Exceptions from this rule may in certain cases be made by the use of warning signals. The input programme will distinguish between the different types of information by the first teleprint character on a line. Each line can therefore in general only contain one item and the line must be terminated by CR, LF (carriage return, line feed). With one exception (to be mentioned under warning character T) the teleprint characters SP (space), LF (line feed). Error and FS (figure shift or blank tape) are always ignored.

The error symbol consists of 5 holes in the tape and is used, by backspacing the tape, to erase a faulty punched

character. At the same time a corresponding symbol is printed on the teleprint sheet.

Most punching errors will be detected by the input programme. There are several loop stops for this purpose. (See table III.)

4.1 *Orders*

An order is distinguished by beginning with a digit. The three first digits give the function and the B-digit. We will here once more state that the function digits used by the programmer do not correspond in a simple way to the true function value inside the machine. Table 2 in the appendix contains the true equivalents. To the equivalents must be added the B-number if it is different from zero.

It is important to distinguish between the input and the execution of an order. No order read in from tape is executed until the complete programme has been assembled inside the machine.

4.1.1 *The address part*

We have in 2.1 described the address systems used by the machine. However, such addresses would be most inconvenient to use directly, and we shall therefore introduce some simplifications. The calculation of the actual numerical value of an address will in many cases be left to the machine.

We shall first describe an address system that can be used in all orders regardless of the type. It is a combination of the page and register systems.

The address consists of the page number followed by the register number inside the page. The two numbers are separated by a point. As there are 64 registers in one page a . b means 64a + b. The formula can be generalized by removing the restriction b < 64. Thus the address 3.02 can be written 3.2, 2.66, 1.130, 0.194 or .194.

We establish the following *rule:* A numerical address containing a point will be modified by the machine to the address system applying to the order in which it is used. The address must be written in the page/register system.

1) L-type orders: the address is halved before it is placed in the store.
2) M-type orders: the address remains unchanged.
3) H-type orders: the address is doubled.
4) P-type orders: the address is divided by 64.

The addresses of n-type orders are discussed in sect. 4.1.6.

A numerical address without a point is never modified by the input programme (except in one case, see below). The reason for this choice will become apparent in section 4.1.5.

Examples.

The order 400 1.2 means the same as 400 33. The long number in question consists of the registers 1.2 and 1.3. The register address of a long number should be even.

The order 590 1.2 means the same as 590 66.

The order 007 1.2 means 007 132.

The short word referred to in the last order is the left hand word of register 1.2. Because of the doubling mentioned in 3) above, a register address can so far only refer to an even numbered short word. In order to specify an odd numbered one, i.e. the right hand word of a register, we add a «+» to the register address. Thus the short word 1.2+ is number 133. Because of this use of the symbol a plus must never be used as a sign for a positive address.

The plus sign has *always* the effect of doubling and adding one, even if the address does not contain a point. Thus the right hand word of register 25 can be denoted 25+, using the register system, whereas the left hand one must be denoted .25.

In an order of the P-type, e.g.

680 a . b

the second part b of the address is discarded, provided b < 64.

4.1.2 *Floating addresses*

When using only numerical (absolute) addresses a programme has to be very specific concerning the exact location in the store of the orders and the numerical data. It is often impossible to make even minor changes in such a programme without renumbering the addresses to a great extent. In order to increase the flexibility there are available a number of so called floating (or symbolic) addresses, «flads». A flad can be given any value, and is then used instead of the explicit numerical address.

There are two sets of floating addresses, 20 x-addresses written x0, x1, ..., x19 (x = multiplication sign on the teleprinter), and 256 *v*-addresses *v*0, *v*1, ..., *v*255.

Associated with the latter group there are addresses *n*0, *n*1, ..., *n*255, whose values are the negatives of the corresponding *v*-addresses.

The most common way to define the value of a flad is to attach a label to an order, or to some other item of the programme. The label is a left hand bracket written to the right of the item, followed by a number. As a result of this the *v*-address with the same number is set equal to the address of the storage location containing the labelled item. This procedure applies only to flads of the *v*-type.

Example. The following is a programme for halving the contents of SAC 5 times.

```
  106   —4
┌→340    0    (7
└—186    v7
```

The *v*-number is unimportant. Note that the label attached to the order 340 0 does in no way alter this instruction. It serves only to define the value of the *v*-flad used in the last order. Note also that the programme will work correctly where ever it is placed in the store, since the machine itself records the value of the *v*-flad. In this case the *v*-address

is defined before it is used, however, forward references are also permitted.

Two or more labels may be attached to the same item. In that case the *v*-flads in question are all given the same value.

A *v*-flad defined by means of a label is modified to the address system appropriate for the order in which it is used. In case this system does not correspond to the type of item labelled, the flad is taken to be the address of the *first* storage location containing a part of the item, or of the one embracing it.

Examples

Let *v*1 be defined by labelling a long number (see 4.2). Then the order

200 *v*1

takes the exponent part to SAC.

Supposing *v*2 has been defined by marking an instruction, the order

690 *v*2

transfers to the drum the page containing the labelled one.

Another way to define a flad is to use an equation whose second member is an absolute address in the register system, (with or without a point or a plus). This method applies to flads of both types. A flad defined by an equation is modified to the appropriate system when used in an order of the H-, M-, L- or P-type. (For n-type orders, see 4.1.6.)

Examples

The equation

x1 = 12.0

sets x1 equal to the H-, M- or L-type address 12.0, or the page number 12, depending on the type of order in which it is used.

The equation

*v*1 = 25+

sets *v*1 equal to the short word address 25+, or to the

address of the register, long word or page containing it, i.e. M25, L.24 or P0.

By using an arrow instead of the equality sign in the defining equation we get a flad which is never modified. The significance of this is shown in sect. 4.1.5. This method is used when the flad in question is going to represent a constant rather than an address, e.g. the order of a matrix, the number of simultaneous differential equations to be solved, etc. The second member of such an «equation» can be a positive or negative integer (within the scope of a short number). It should not, of course, contain a point or a plus.

Although forward references are allowed by *v*-flads, any x-address must be defined *before* it is used. An x-flad may be given new values any number of times, but no *v*-flad can be redefined. There are checks in the input programme to see that the above rules are adhered to (see 4.1.9).

The second member of a defining equation may be a previously defined x-address, or a mixed address (see 4.1.5) containing such flads. *v*-addresses must not be used in this connection. An x-address to the right of an equality sign or an arrow will normally have been defined by means of the same symbol. (If not, an x-flad to the right of an equality sign defined by an arrow will count registers. Conversely a flad to the right of an arrow defined by an equality sign will take the numerical value corresponding to the register system.)

4.1.3 *Routines*

Before we can proceed with our description of the floating addresses, it is necessary to say a few words about programme organization. Large programmes are broken up into parts called routines. Each routine is designed to carry out some minor task, easy to overlook, such as printing a number, evaluating some function, etc. The construction of the complete programme is then reduced to the comparatively

simple task of linking together the various routines. It is often possible to find identical processes, or nearly identical ones to be carried out in different connections. One routine designed for this purpose can then be used in all cases.

There are two species of routines, x-routines and *v*-routines. The former are standard library routines of frequent use in different programmes, whereas the latter are routines drawn up to solve the particular problem at hand. Each routine is given a number. The x-routines have permanent numbers greater than 1000, the *v*-routines are given numbers less than 1000. (See also 4.5.1.) All the *v*-routines of a programme, even if some should be identical, must have different numbers, but the same x-routine, with the same number, may occur more than once.

4.1.4 *More about flads*

The x-addresses are unchanged inside large parts of a programme, usually inside a whole programme. It will keep its former value until a new definition is encountered. A *v*-address, however, has a fixed meaning only inside the *v*-routine in which it is defined. That is, the definition holds good only inside the *v*-routine. In another *v*-routine any flad may be defined anew to be used in an entirely different connection. A *v*-address belonging to another routine can be used by specifying the number of the routine after the *v*-number. The two numbers must be separated by a «/». For instance the address *v*5/2 will mean the flad *v*5 belonging to (*v*-) routine no. 2. Since all *v*-routines of a programme have different numbers, a flad specified as above is unique.

Cross references between routines belonging to different *chapters* (see 4.5.2) are usually meaningless, because the routines will not normally be present in the computing store simultaneously. Exceptions are the address parts of the warning characters A, D and Z (sect.s 4.6.1, 2, 4).

v-flads should neither be defined nor used in an x-routine, x-addresses, however, can be defined and used anywhere.

The flads *v*0 and x0, (the index is usually omitted), are both defined automatically. *v* is set equal to the address of the first storage location occupied by the present *v*-routine. The address behaves as if it had been defined by labelling an order. Cross reference to the *v* belonging to another routine is permitted. The flad x has a similar meaning inside an x-routine. If used in a *v*-routine the address will refer to the x-routine last read in. Routine specification is impossible in the address x, and in any other x-address.

4.1.5 *Mixed addresses*

When writing a numerical address in front of a flad, the two are added together before the final address is placed in the store. The former can be regarded as an ordinary address specifying a storage location relative to an artificial origin defined by the flad. The relative address can be positive or negative. For instance, the storage location specified by —1.0 *v*1 precedes *v*1 by one page.

We may, as a rule, consider the computing store as being divided in regions, each consisting entirely of storage items of one type, long or short numbers, or orders. (The layout of the store will of course vary from one programme to another, and often also during one calculation.) It is often convenient to let one flad represent a whole region. Let x1 refer to a location inside a region of, say, long numbers. We are then able to cover the entire region by using B-modified orders. E.g. the order

406 x1

with a suitable number in B6, will fetch any long number in the region. The same applies to any region, whatever be the storage items in question.

Another method to achieve the same result is to add a relative address to the flad, to play the role of the B-modification in the order above. The address —1x1 specifies the

item preceeding the one referred to by x1. The address 7*v*3 specifies the item 7 beyond *v*3, etc. Here the word «item» may stand for long or short words, registers, or pages, depending on the type of order in which the address is used. This is because the numerical address is invariant when written without a point, whereas the flad is still modified to the correct system.

There seems to be a certain advantage in using the «natural» systems in relative addresses as above, since the internal numbering of a region in the store is then independent of the nature of the storage items involved. We also note that the relative address and a B-modification can be directly added together mentally.

Example

Supposing B6 = 10, the orders

406 —3x1 and 206 —3*v*1

both refer to the item 7 beyond the one used as an origin.

If one for some reason should prefer to use the register system in the relative addresses, this can be done by including a point (or a plus) in those addresses for which the register system is not the «natural» one. That is probably the best thing to do by internal references in an x-routine, because all such references must be relative to the flad x. The relative addresses will then coincide with the printed numbering in the manuscript. This problem does not arise in *v*-routines, since new origins can be defined by means of labels.

In general any number of flads can be strung together in an address. The value of the address is always formed by adding together the various parts. In case flads of both types are present in an address, the x's must precede the *v*'s. A numerical part must always be the very first one. The invariant parts will count items of the type in question, and

the other ones will be modified to the correct address system given by the type of the order.

Just as numerical addresses without a point, invariant flads, i.e. flads defined by arrow, are often used as counting limits and «relative» addresses. This can be particularly useful in a general purpose programme where e.g. the size of a set of numbers can vary from time to time.

Let us as a simple example consider the problem of adding together N long numbers stored consecutively, the first one having the address $v1$. It is assumed that the flad x1 has been defined by an arrow

x1 → N

earlier on the tape, to represent the number of addends. Taking the numbers in reverse order the programme can be

```
400   —1x1v1
106   —2x1
┌→426   v1      (2
136   1
└—090   v2
```

An easy way to check a counting like this is to see what happens the first and last times the loop is executed. We observe first that the numbers to be added together have the following addresses:

$v1$, $1v1$, $2v1$, ..., —2x1$v1$, —1x1$v1$

The last number of the set is taken in by the order 400. The first time the order 426 is executed, the contents of B6 are —2x1. Then the number in the address —2x1$v1$ is added. The last time we have (B6) = 0, and the number added is the one in $v1$, i.e. the first one of the set.

A shorter programme than the one above can be made by taking the numbers in natural order, counting B6 through negative numbers up to zero. Since the negative of an x-address can not be stated we define an auxiliary v-flad


```
v3  →  x1
400    v1
106    2n3
┌→426    —1v3v1     (2
└—186    v2
```

Note that the definition $v3 \rightarrow$ x1 forms no part of the programme when stored in the machine. It is only a signal for the input programme concerning the meaning of the flad $v3$ (and $n3$). We leave it as an exercise for the reader to check this counting.

Note that the two countings shown above apply equally well to short numbers. The address parts of the orders will be exactly the same.

4.6 *Addresses in n-type orders*

As explained previously the numerical addresses with a point and most flads are modified to the correct address system in H-, M-, L-, and P-type orders. In n-type orders, however, the rules governing such addresses are more complicated, because the «correct» system is not uniquely defined. The examples below show that the wanted system can be any of the four, depending on the circumstances.

101	$v1$	101	$v1$	101	$v1$	101	$v1$
201	0	591	0	401	0	681	0

The rules are the following:

1) v-flads defined by marking give the address system corresponding to the type of item labelled:
   H if a short number
   L if a long number or the warning character Z (see 4.6.4).
   M if an order or any warning character different from Z.
2) The flads x and v are taken in the M-system.
3) Numerical addresses with a point and flads defined by an equation are taken in the L-system.
4) A numerical address containing a plus is interpreted as an H-type address.

5) Numerical addresses without a point (or a plus) and flads defined by an arrow are invariant as usual.
6) In an address containing more than one flad all modifiable flads are taken in the same system, defined by the first *flad* not invariant. Exception: a plus in the numerical part will modify the rest of the address to the H-system.

In most cases the rules above will give the wanted address system, with the possible exception of point 3. It might seem more natural to use the M-system here, however, nearly all M-type addresses will be *v*-flads defined by labels. Note that even an address defined as

$$v1 = 25+$$

will come out in the L-system (as .24) in an n-type order.

In those cases when the rules give an unwanted system the address can be modified to the H-, M-, L-, or P-system by writing the symbols $>$, $=$, $\neq$, or $\geqslant$, respectively, in front of the address.

Examples

The addresses of the instructions

101 x and 101 20x

are both of the M-type. In

101 2.0x

the numerical part is taken in the L-system, whereas the flad is still an M-address. To get the M-system in both parts we must write

101 = 2.0x

4.1.7 *H-type orders*

As described earlier the input programme treats an H-type order as if the address part contained 11 digits, the 11'th one being in fact one of the function digits. This applies of course only to the address which is punched on the tape, not to addresses placed in the address part by the programme itself. The function digit deciding whether the order will refer to the first bloc (P0—P7) or the second one (P8—P15),

is determined during input of the order. For instance, the order

007 8.0

will fetch a number from the second bloc, no matter what is placed in the address part at a later stage.

The fact that the B-modification of a SAC-instruction runs from —512 to 511 may occasionally cause an unexpected result. One should expect the orders

106 a
206 0

to have the same effect as

200 a

However, this is only the case if $-512 \leqslant a \leqslant 511$. If the address does not satisfy this condition, the former instructions must be replaced by

106 a
206 8.0

Example

The orders

106 > 7.0
206 0

will take the short number in 15.0 to SAC. The H-address 7.0 being greater than 511 is taken to be negative. Therefore the B-modification goes one page backwards instead of 7 pages forward.

A short number region to be covered by a B-modified instruction, which contains more than 512 numbers (4 pages) must have an origin in the middle. If not, the B-modification will not always satisfy the condition given above.

We will state once more that the address part of the H-type order 630 can only refer to words in the first 8 pages of the store. (If a greater address is written, the function is transformed to a 620-order.) However, words in the second quarter of the store can be reached by a suitable B-modification.

4.1.8 *Summary*

The main properties of the addresses can be summed up in the following points.

1) Numerical addresses (absolute and relative) *without* a point are invariant. They must be written in the «correct» system, which depends upon the function.
2) Numerical addresses *with* a point are modified to the correct system in H-, M-, L-, and P-type orders. In n-type orders they are treated as addresses in the L-system. These addresses must be written in the page/register system.
3) x- and *v*-flads defined by an arrow are treated as numerical addresses *without* a point.
4) x- and *v*-flads defined by an equality sign are treated as numerical addresses *with* a point.
5) *v*-flads defined by marking are treated as numerical addresses *with* a point, except in n-type orders. The system corresponds here to the type of item labelled.
6) The flads x0 and *v*0, whose values are determined automatically, behave as if they were defined by labelling an order.
7) In n-type orders we can alter the system of a modifiable address at will by writing one of the symbols $>$ (H-type), $=$ (M-type), $\neq$ (L-type), or $\geqslant$ (P-type) in front of the address. The invariant parts of the address will remain unaltered.

When using the system one should

a) write absolute addresses in the page/register system *with* a point,
b) write relative addresses and constants *without* a point,
c) define a flad by an equality sign when it refers to the computing store, and by an arrow when it represents a constant. (An address referring to the magnetic drum should be defined by an arrow.)

1.9 *Automatic checks on flads*

When preparing a programme tape one of the most common mistakes is to forget to define a floating address. An error of this type is always detected by the input programme. If an un-defined x-address is used the machine comes to a loop stop. (See table III.) The stop is encountered immediately after the address is read in, and by checking the tape it is easy to locate the missing definition.

An undefined *v*-address is not detected until the warning character E (or a warning character X) has been encountered. (See 4.8.5.)

The machine will emit a high pitched note of about one second's duration, and then come to a B7-«modified» stop order. The contents of B5, B6 and B7, to be seen on the monitors, will define the faulty address. The contents are

B5: the *v*-number,
B6: the number of the chapter where the address is *written,*
B7: the number of the routine *referred to.*

(If the address was part of a w.ch. E or X we have B6 = —1.)

If a *v*-address is not only undefined, but refers to a non-existent routine, the note emitted will be of a lower pitch. Otherwise the course of actions is the same as above.

When the operator has recorded the contents of B5, B6, and B7, he should give a prepulse. Then the machine resumes its operation and will stop again if another faulty *v*-flad is detected. (The entering of the programme should be prevented by means of the hand switch no. 0.) If the same undefined flad has been used more than once, each occurrence will cause a stop. This is of value if the faults shall be corrected by means of X-corrections (see 4.9). *NB!* In this case prepulses *must* be given until the stop order caused by the hand switch is reached. (A 993 order.)

As mentioned previously a loop stop is encountered if one tries to define the same *v*-flad more than once in the same

routine. (See table III.) The machine will stop immediately after input of the second definition. The v-number is found by checking the tape.

4.2 *Input of long numbers*

A long number to be read in together with the programme is preceded by a + or a —. The first action taken by the input programme is to find the next empty long word storage location. If a whole register is skipped in this way, a dummy order is inserted there. After the sign the number follows in decimal notation with or without a decimal point. It may or may not be followed by a decimal exponent separated from the number by a comma. If this exponent is positive it must not be preceded by a +. The following ways of writing a number are equivalent.

+123.45 +.12345,3 +12345,—2

In most cases rounding errors will occur during input of a long number. Only integers written without a decimal point and without an exponent will be stored in exact form. In order to avoid unnecessary rounding errors one should never give more than 9 significant digits. Superfluous zeros after the decimal point should be omitted. If extreme accuracy is required in a long number as stored in the machine, it should be read as 4 short numbers (see next section).

A long number can be labelled. The flad thus defined will come out in the L-system in an n-type order.

4.3 *Input of short numbers*

Short numbers are preceded by one of the symbols $>$, $=$, $\neq$ and $\geqslant$, and may include flads. The interpretation of a short number is exactly as for the address part of an instruction. The symbol preceding it serves as usual to define the wanted address system, H, M, L and P, respectively. If all parts of a short number are invariant the symbol used is immaterial,

however, it is customary to use the equality sign in such cases.

A short number may be labelled. The corresponding flad gives an H-address in an n-type order.

In order to ensure that four consecutive short numbers combine into one long number, i.e. are stored in a long word storage location, one can write a «+» in front of the first one. The warning character £ can also be used (see 4.8.3).

Example

The short numbers

+ = 2 (4
= 852
= 126
= 402

form together the standardized floating point number π with maximal accuracy. The three fractional parts follow in *reverse* order, the last number being the most significant part.

(If π is read in as a long number with 9 decimal digits, the least significant part of the number inside the machine will be equal to 850, due to the rounding errors. See the last section.)

We shall in this connection describe briefly a simple method to obtain the 4 short word parts of a floating point number.

For a positive number:

Multiply the number by such a power of 2 that the result lies in the range $0.5 \leqslant x < 1$.

The negative of this power (the exponent) is equal to the exponent part.

Multiply x by 512. The integral part of the product is equal to the most significant short number.

Multiply the remaining fractional part twice by 1024 to obtain the middle and least significant parts. The latter should be rounded.

For a negative number:

The power of 2 should be such that the product lies in the range $0.5 < -x \leqslant 1$. Add 2 to the negative number x and proceed as above.

In the example above we have labelled one of the short numbers. The order 400 *v*4 will of course take the complete long number to the accumulator. However, we may for some reason want to put the address of this number into B6, and then place the number in the accumulator using the order 406 0. We must then write 106 ≠ *v*4, if not *v*4 would be taken as an H-address.

4.4 *The warning characters*

A warning character is a symbol on letter shift. The term warning character is not a good one. The fact that is has been used for a long time at the NDRE is the only excuse for using it here.

Many warning characters form no part of the programme inside the machine. These are just «warnings» to the input programme to take actions of certain kinds. Others, viz. those described in the sections 4.6 and 4.7 represent orders to be inserted into the programme. These w.ch.'s may be labelled. A flad defined in this way will always be interpreted in the M-system when used in an n-type order, except when w.ch. Z is labelled (4.6.4). In the latter case it is taken as an L-type address. As usual the label refers to the first storage location occupied by the programme represented by the labelled warning character. All w.ch.'s of this type must begin in an *even numbered register,* with the exception of w.ch. U (4.6.3). The machine will insert a dummy order in front of the w.ch. if necessary. This order does not count as a part of the latter programme, and will not be referred to when labelling the w.ch.

Nearly all warning characters are the initial letters of single English words. If one wishes one can provide the whole

word on the tape. In any case the machine will only use the first letter, i.e. the first tape character after the LS, and skip the following ones until the next FS.

4.5 *Programme organization*

The following sections describe the ways of breaking a programme up into parts, and how to arrange such parts in the computing store and on the drum. The various topics are discussed in connection with the corresponding warning characters.

4.5.1 *W.ch. R (ROUTINE)*

Each routine of a programme is headed by a warning character R followed by the routine number. As explained previously the two species of routines are distinguished by having numbers less than 1000 (*v*-routines) and greater than or equal to 1000 (x-routines). A programme can not contain more than 51 *v*-routines, whereas an arbitrary number of x-routines can be used.

We will state once more that all the *v*-routine headings of a programme must be different, whereas an x-routine may occur more than once with the same heading. In fact, the heading of an x-routine is included on the master copy of the routine library, and will thus be copied into the programme tape together with the routine itself.

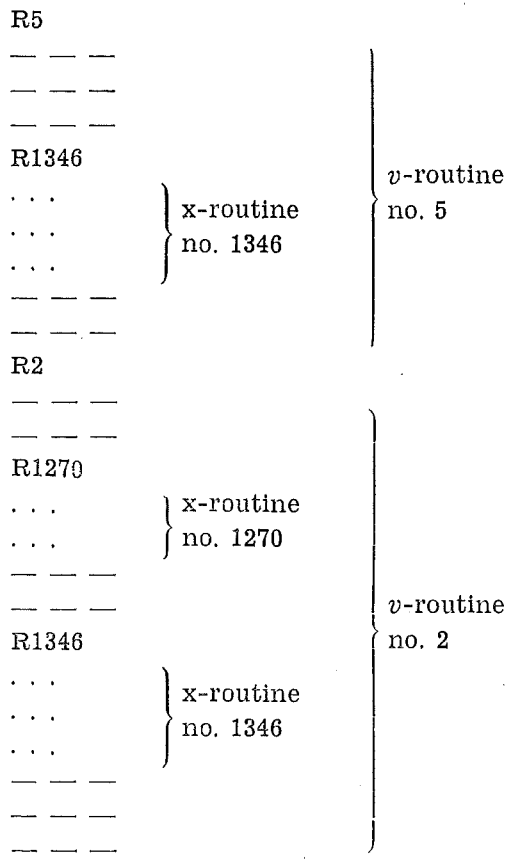
Any routine will begin in an even numbered register. If a dummy order is inserted by the input programme it will not count as a part of the routine.

There must be at least one *v*-routine heading in a programme. As a rule every part of it will belong to some *v*-routine. The *v*-routines will lie side by side in the store, that is, a new heading means also the end of the previous routine. An x-routine, however, will always be embedded in a *v*-routine. In fact, the w.ch. R followed by a number ⩾ 1000 serves no

other purpose than to find the first empty even numbered register and to see that the flad x is defined. Therefore the orders following the routine will belong to the same *v*-routine as those preceding it.

It is not necessary (nor possible) to specify explicitly the end of an x-routine.

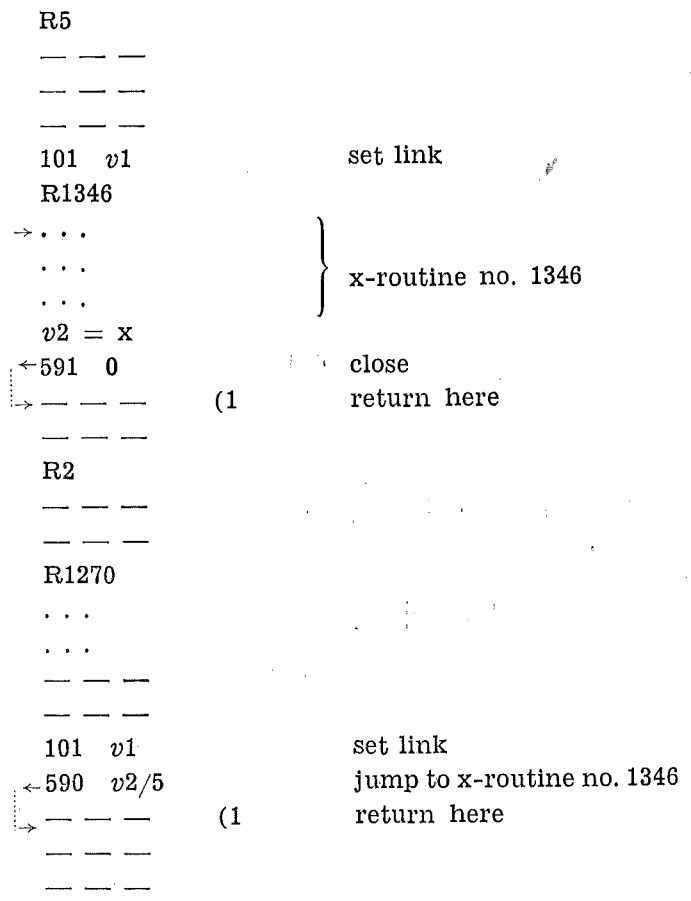
The example below shows a possible arrangement of x- and *v*-routines.



A routine which is used by another one is called a subroutine. It is said to be *open* if it only squeezed in between the orders

of the other routine. In the example R1346 is used twice as an open subroutine.

By terminating a subroutine by a B-modified jump order one can supply information in a B-register about where to return, before it is entered. Such a subroutine is called *closed*. The return address given is called a *link*. It is customary to use B1 for such purposes. By means of this technique the arrangement above can be replaced by the following.



Here R1346 is written only once, as a closed subroutine. The order 591 0 is called the closure order. The orders of an x-routine can not be labelled in the ordinary way. (At least there must be no label on the library tape.) Instead we have written immediately after the routine the line $v2 = x$, which sets $v2$ equal to the address of the first order of R1346.

Note that the second label (1 is no redefinition of the flad defined by the first one, because the two labels are written in different routines.

4.5.2 *W. ch. C (CHAPTER)*

When a programme is too large to be contained in the first half of the computing store, it must be broken up into parts called chapters. Each part is headed by a w.ch. C followed by the chapter number. The latter can be chosen at will between 0 and 24. It follows that no programme can contain more than 25 chapters. There must be at least one chapter heading in any programme.

The order in which the chapters of a programme are read in is immaterial.

A chapter can contain any number of routines, and must contain at least one v-routine. The routines of a programme can be assembled into chapters in an arbitrary way, provided that no routine is divided. However, one should see that the number of jumps from one chapter to another is as small as possible, because such «jumps» require magnetic transfers which are time consuming. (See sect. 4.6.)

The length of a chapter is measured in pages. It will always begin on the top of a page, usually P1, the last one being filled up with short word zeros (stop orders). The chapters are stored consecutively on the drum, in the order in which they follow on the tape, but not necessarily end to end. In fact, when nothing else is said, 15 sectors will be reserved for each chapter. The first one will normally be stored from sector no. 55 and onwards.

The rules stated in the last paragraph can be modified by the use of the w.ch.'s P, F, S and J, discussed in the following sections.

4.5.3 *W. ch. P (PAGE)*

This warning character is used for stating the area of the computing store available for a chapter. It must be written immediately after the corresponding w.ch. C, if one wishes, on the same line. A complete w.ch P reads

Pp — pm

where p and pm are page numbers in the range 1—15. As a result of this the chapter will begin on the top of page no. p, and will be allowed to extend to and include page no. pm. The latter number (and the dash) may be omitted in the warning character. Then pm is taken to be equal to 15. If no w.ch. P is written at all the chapter will begin on P1 and can extend to P15. In any case the input will come to a loop stop if a chapter exceeds the space allowed for it in the computing store. (See table III.)

The number of drum sectors reserved for a chapter is equal to the number of pages indicated by the w.ch. P. Usually the drum space is not very critical, and it will pay to take good measure when estimating the length of a chapter. The significance of this is shown in connection with w.ch. X. (See 4.9.) Only the sectors actually used are read down when a chapter shall be executed. (Cf. 4.6.)

When stated explicitly a chapter can extend into the second half of the computing store (pm $\geqslant$ 16). However, those parts of the chapter which are stored from P16 and onwards can only contain long numbers, or other data to be operated upon by means of the accumulator.

4.5.4 *W. ch. F (FIRSTSECTOR)*

As already said a programme is normally stored on the drum from sector 55 and upwards. This can be changed by writing the w.ch. F followed by another sector number at the beginning of the tape. By this means the programme is moved. One must take care not to use any of the isolated sectors, or the ones used for directories by the input programme. (See 5.0.)

There are cases when it is desirable to store a programme as far down as possible in the backing store. If desired a programme can be stored from sector 49 and onwards. However, the sectors 49—54 are normally reserved for the standard routines for complex and double length arithmetic.

4.5.5 *W.ch. S (SECTOR)*

W.ch S can be used to state explicitly the sectors to be used for storing a chapter. The sector number indicated must be relative to the first sector of the programme. The w.ch. must be written immediately after the corresponding w.ch. C (or P). It is not much used.

4.5.6 *W.ch J (JUMP)*

This warning character is used for introducing gaps in the programme, inside or in front of a chapter. The letter is followed by a register address, which can contain x-addresses, but no *v*-flad. The effect is that the following items on the tape will be stored from the indicated register and onwards. The gap will be filled up with short word zeros. It is always included in the chapter, even if it amounts to several pages.

In certain cases it is allowed to over-write orders previously stored. The conditions are that these orders lie on the same page as the first empty storage location, and that they do not contain v_- (or n-) addresses.

Note that a w.ch. J is only a signal to the input programme. It represents no jump order in the programme read in.

4.6 *The chapter changes*

All magnetic transfers of programme during a calculation are ordinarily organized automatically. The programmer need not know much about the underlying details, and the reader may, if he wishes, skip the detailed explanations of this and the following sections. However, it is necessary to know the general effects of the w.ch.'s A, D and U, and the layout of page no. 0.

Any change of chapter will destroy the contents of SAC, St, and the standard working position L.32. The latter is set equal to the contents of the accumulator.

During a calculation programme is only transferred from the drum to the high speed store. This means that every time a chapter is brought down and reentered it appears in its original shape, no matter what modifications were made last time it was executed.

The routine which actually performs the necessary magnetic transfers of programme, is stored permanently on page no. 0. (See sect. 4.6.5.) This is the so called chapter changing sequence (CCS). The purpose of the CCS is to bring down a new chapter from the drum to the high speed store, and to enter it in the appropriate instruction. To be able to do this the CCS must be supplied with the following information:

1) the location on the drum, given by the number of the first sector, T,
2) the location in the high speed store, give by the number of the first page, p,
3) the length of the chapter, given by the number of the last page, P,
4) and, finally, the address of the order to be executed first, the so called entry point, E.

These data are strung together in a long number called the *cue* to the chapter. Its four components read

T — p p
P E

All w.ch.'s concerning change of chapter contain such a cue, except w.ch. U (4.6.3).

Note that P can be less than, but not greater than the page number pm stated in the w.ch. PAGE. All cues are constructed automatically by the machine and inserted automatically in the programme where they are needed.

4.6.1 *W. ch. A (ACROSS)*

The warning character must be followed by a *v*-flad (or a mixed address) with a routine specification. When executed w.ch. A has the effect that the chapter containing the specified routine is brought down, and entered in the indicated address. One can say that the w.ch. A is logically equivalent to a 590 order with the same address part. The w.ch. occupies 4 registers (with a possible dummy order in front of them).

The following orders are placed in the programme:

2m	300	m + 1	cue address
2m + 1	590	13	jump to CCS
2m + 2 } cue			
2m + 3 }			

The CCS will take over, fetch the cue, read down the new chapter, and enter it. The entry point E of the cue is of course equal to the address part of the warning character.

4.6.2 *W. ch. D (DOWN)*

A chapter used as a «subroutine» is called a subchapter. Such a chapter is entered by means of a w.ch. D. The warning character must be followed by an address with a routine specification, and it will occupy 4 registers in the programme. When the closure of the subchapter is reached, (see next section), the machine will return to the chapter where the

w.ch. D was written, in the order immediately following it. Thus we can say that a w.ch. D*pvq/r* is logically equivalent to the orders

101	*v*1	set link
590	p*v*q/r	jump to subroutine
————	(1	return here

A w.ch. D represents the following instructions in the programme:

2m	300	2m + 4	return address
2m + 1	590	0	jump to CCS
2m + 2 } cue			
2m + 3 }			
2m + 4	———		return here

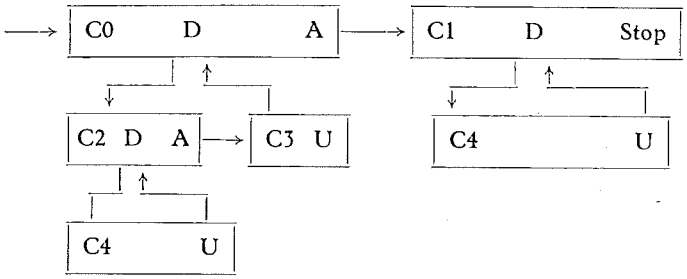
The CCS will construct and store a «link», which is in this case the cue to the present chapter with the old E replaced by the return address in SAC. Since the cue to the new chapter always lies in the long word preceding the return point, CCS can calculate the address of the cue, fetch it, and proceed as by w.ch. A.

4.6.3 *W. ch. U (UP)*

This warning character is used to close a subchapter. It has no address part, and is logically equivalent to the closure order 591 0 of a subroutine. As already explained the machine will return to the chapter left by a w.ch. D. Subsubchapters are permitted, i.e. a subchapter may itself use a new subchapter. In that case they are both terminated by a w.ch. U.

The drawing shows a possible logical arrangement of the chapters of a programme.

The chapter C4 is used both as a subchapter and as a subsubchapter. (Of course it is punched only once on the tape.) The chapters C2 and C3 form together a «subroutine» used by C0.



The warning character U represents the single instruction

590 10 (jump to CCS)

The cue defining the chapter change to come is one of the links stored by the CCS at a previous w.ch. D. Which link to use is given by the so called «level number», which tells whether the present chapter is used as a sub- or a sub-subchapter. (See 4.6.5.) There must not be more than three «levels» in a programme.

4.6.4 *W. ch. Z*

This warning character represents the cue to the chapter specified by the address part as usual. It is not used very often, but will in certain cases increase the flexibility. For instance, let *v*1 refer to a list of w.ch.'s Z. Then the orders

306 *v*1
590 13

cause an «across» to a chapter which depends upon the contents of B6. Note that a *v*-flad defined by labelling a w.ch. Z is taken in the L-system in the n-type order 306.

4.6.5 *The layout of P0, the CCS*

The first 32 registers of P0 are permanently occupied by the CCS and the link list.

The following three long words L.32, L.34 and L.36 are used as working space by most standard routines. In L.38

there is a long number 0.2^9, used for converting a signed short number to a long one. (This is done by placing it in H39+.) The locations L.40 and L.42 contain the numbers 0.2^{-256} and -1.2^0, respectively. The last three long words (except H39+) and the CCS should be regarded as «sacred cows» never to be disturbed by a programme. The remaining parts of P0 can be used as unsystematic storage space for data. Programme may not be placed here. The details of the CCS are shown below.

D				
→	0	210	27+	return addr. → present cue
	1	300	.28	level no., i
	2	410	.32	store acc.
	3	400	.26	present cue → link list
	4	417	0	
	5	320	1	i + 1 → i
	6	210	1+	
	7	200	27+	restore SAC
	8	340	1	form cue address
U	9	—590	14	
→	10	200	1+	i — 1 → i
	11	330	1	(here cue addr. = i)
A	12	210	1+	
→	13	410	.32	store acc.
	14	→407	0	fetch cue
	15	410	.26	new present cue
	16	200	.26	T — p
	17	210	19+	
	18	200	26+	p
	19	→677	0	read down to p, p + 1, ..., P
	20	687	0	
	21	270	.27	
	22	—380	19	
	23	200	27+	entry point
	24	400	.32	restore acc.
	25	597	0	enter new chapter

26	T—p p	present cue
27	P E	
28		link list
29		
30		
31		
32		standard working space
33		
34		
35		
36		
37		
38	9 0	0.2^{9}
39	0 0	
40	—256 0	0.2^{-256}
41	0 0	
42	0 0	-1.2^{0}
43	0 512	

4.7 *Aids to programming*

A number of short routines are of sufficiently common occurrence to warrant special treatment. These routines are stored permanently on the drum, and can be incorporated or referred to in the programme by means of various warning characters.

4.7.1 *W. ch. Q (QUICKIE)*

This w.ch. represents a routine, a «quickie», incorporated as an open subroutine. There are 15 quickies, numbered from 0 to 14. Most quickies make use of the constants 0.2^{9}, 0 and —1 on P0. With the exception of Q7 all function quickies ensure constant *relative* accuracy in the function value. The errors are negligible.

The quickies are described below. The term «uses» means really «changes the contents of».

Q0 $(A)^{-1} \rightarrow A$

Length: 17 registers
Time: 4.17 ms.
Uses: A, S, St, L.32, L.34, H39+.

The contents of A must be standardized. When A contains an exact zero, or another number whose exponent part is equal to —256, the machine will stop, due to exceed positive in the exponent register.

Q1 $(A)^{-\frac{1}{2}} \rightarrow A$

Length: 24 registers.
Time: 4.83 ms.
Uses: A, S, St, L.32, L.34, H39+.

The contents of A must be standardized. If it is negative the machine comes to a B1-modified stop order.

Q2 $\sin \frac{\pi}{2} (A) \rightarrow A$

Length: 31 registers.
Time: 4.17 ms.
Uses: A, S, St, L.32, L.34, L.36.

There is no restriction on the range of the angle. Note that it is given in units of right angles. The same applies to Q3, Q4 and Q5.

Q3 $\frac{2}{\pi} \arcsin (A) \rightarrow A$

Length: 52 registers.
Time: t_{min} = 5.22 ms. ($|A| < 0.5$),
t_{max} = ca. 25 ms. ($|A| = 1 - 2^{-29}$),
t_{av} = 7.34 ms.
Uses: A, S, St, L.32, L.34.

If $|A| \geqslant 1$, the function value is set equal to $\pm$ 1. In this case the time is 1.08 ms. The angle lies always in the range [—1, 1].

Q4 $tg \frac{\pi}{2}$ *(A)* → *A*

Length: 52 registers.
Time: 5.82 ms. if (A) = a + 2k,
9.81 ms. if (A) = a + 2k + 1,
where | a | < 0.5 and k is any integer.
Uses: A, S, St, L.32, L.34, L.36.

Q5 $\frac{2}{\pi}$ *arctg (A/L)* → *A*

(The address of L in S.)
Length: 71 registers.
Time: 12.24 ms. (average value).
Uses: A, S, St, L.32, L.34, L.36, H39+.
The long number L must be standardized. Otherwise there is no restriction on it. (Even an exact zero can be used.) The angle lies in the range [—2, 2]. Its sine has the same sign as A, its cosine the same sign as L.

Q6 *exp (A)* → *A*

Length: 31 registers.
Time: 4.68 ms.
Uses: A, S, St, L.32, L.34.
The range of A′ is not checked. (The final exponent is calculated in SAC and set by a 717-order, which is not checked internally against exeed positive or negative.)

Q7 *ln (A)* → *A*

Length: 37 registers.
Time: 6.18 ms.
Uses: A, S, St, L.32, L.34, H39+.
There is no check against negative numbers. The contents of A must be standardized.

Q8 *Read in integer to S*

Length: 28 registers.
Uses: S, St, B6, Bt.
The machine will search for the first decimal digit to occur on the numerical tape, ignoring all characters

except possible — or + signs. (The last one, if any, will count as the sign of the number.) Then the number is read in, ER's and FS's between the digits being ignored. The number can be terminated by any tape character different from digit, ER and FS. This character in converted form (see table V), will be in B6 when leaving the routine.

Q9 *Punch S as a positive number*

(The wanted number of digits in B6.)
Length: 26 registers.
Uses: S, St, B6, B5, Bt.
There is zero suppression, i.e. zeros in the most significant digits are replaced by spaces. 4 is the maximum no. of digits. (See also Q14.)

Q10 *Read in fixed point number to A*

Length: 41 registers.
Uses: A, S, St, B6, B5, Bt, H39+.
The machine ignores all tape characters until the first decimal digit, except the characters + — and . which are recorded. Inside the number ER's, FS's and *single* SP's are ignored. The number must be terminated by *double* SP, a LF, or any character greater than 16 in converted form (se table V). The final contents of B6 are equal to the converted terminating tape character. The numbers read in are subject to small round off errors, except integers punched without a decimal point.

Q11 *Punch A as a fixed point number*

(Layout in S.)
Length: 56 registers.
Uses: A, S, St, B6, Bt, L.32, L.34.

The binary digits of the layout constant have the following meaning.

1: digit,

0: digit followed by a space,

00: digit followed by a decimal point.

The layout begins in the leftmost (most significant) digit. Zeros to the right of the last one are ignored. The most significant digit must be changed to the «opposite» value.

Since these conventions are rather complicated one will, as a rule, let the input programme calculate the layout constant. The layout can be defined by a line in the programme exemplified by

* . * * * — * * *

The asterisks denote digits, and the minus sign is used instead of a space. (An ordinary space would be ignored by the input programme.) The number of characters, discounting minus signs, must not exceed 10. The decimal point, if any, and possible minus signs divide the asterisks into groups. The groups must all, except possibly the first and last ones, contain two or more asterisks. The layout must begin and end with an asterisk.

A line as the one above represents an order of the form

300 a

where a is the layout constant. (In this case we have a = 1011011100 = 732.) Such an order must always be executed before Q11 is entered. The layout can be labelled just as an ordinary instruction.

The number punched out by Q11 is rounded off in the last decimal digit. It is preceded by a space or a minus depending upon the sign.

There is zero suppression in the digits preceding the decimal point. A number punched by Q11 can be read in again by Q10 or Q12.

Q12 Read in floating point number to A

Length: 64 registers.

Uses: A, S, St, B6, B5, Bt, L.32, L.34, H39+.

The quickie behaves exactly as Q10, except that it also allows for a signed decimal exponent, separated from the fractional part by a comma. The fractional part must always be written, even if it is equal to 1.

When reading in data by means of one of the input quickies, the fast reader, TR3, can be used. However, when using Q10 or Q12 the reader will stop after the input of a complete number, except possibly after an integer. Therefore the tape must be prepared so that one character can be missed without any damage being done. The numbers should be terminated by *3* SP's or CR LF, etc., if TR3 shall be used.

Q13 Punch A as a floating point number

(Number of decimals, n, in S and St, $n \neq 0$.)

Length: 89 registers.

Uses: A, S, St, B6, Bt, L.32, L.34, L.36, H39+.

The contents of A must be standardized.

Form of output:

$$-d\,.\,\underbrace{dd\ldots d}_{n},-dd$$

If the fractional or exponent part is positive, the minus sign is replaced by a space. If the first digit of the exponent is equal to zero it is omitted, and a space is punched after the number. The contents of S may be negative. In that case only the exponent part is punched out. A number punched out by Q13, with $n > 0$, can be read in again by Q12.

Q14 Punch the sign of S

Length: 6 registers.

Uses: S, St.

When this quickie is written immediately in front of

Q9, the contents of S will be punched as a signed number in the range $-512 \leqslant S \leqslant 511$. (No. of digits in B6.) The output produced by Q14 is a space if $S \geqslant 0$ and a minus of $S < 0$. In the latter case the sign of S is changed.

4.7.2 *W. ch. G (GILLKUTTA)*

This warning character represents a routine for integrating numerically a set of ordinary differential equations of the 1'st order. The routine occupies 38 registers and uses A, S, St, B6, B5, Bt, L.32.

The time for advancing the solutions one step is equal to

$$(0.72 + 14.10n) \text{ ms} + 4t,$$

where n is the number of equations, and t is the time used for evaluating all the right hand sides.

The integration method is the one of Kutta, modified by S. Gill. A thorough description of the method can be found in the original paper of Gill.* We shall follow the notation used there.

There are 3 sets of numbers associated with the Gill-Kutta routine, the dependent variables y_i, the variables k_i which are equal to hy_i', (h = step length in the independent variable), and a set of auxiliary variables q_i. The number of variables in each set is equal to the number of simultaneous equations. The quantities q_i are used only inside the Gill-Kutta routine, and do not concern the programmer. However, space must be reserved for them in the computing store, and they must be equal to zero when the integration is started. The q_i's will in effect carry extra digits of the dependent variables, the theoretically best value of y at any stage being $y_i - \frac{1}{3}q_i$. The actual number of working

*) S. Gill: A process for the step by step integration of differential equations in an automatic digital computing machine. (Camb. ph. soc. Vol. 47, Pt. 1.)

binary digits is equal to $28 + Ey_i - Ek_i$, when the letter E refers to the binary exponent of the long numbers. Note that the scaling problems discussed in the treatise of Gill do not exist here because of the floating point number representation.

The independent variable must be vacant in the right hand sides of the equations. However, this restriction is easily circumvented by replacing it by an extra dependent variable, say y_o, satisfying the equation $y_o' = 1$. The complete set of equations then reads

$$y_o' = 1$$

$$y_i' = f_i(y_0, y_1, \ldots, y_{n-1}) \quad (i = 1, 2, \ldots, n - 1)$$

The w.ch. G must be followed by 5 addresses separated by asterisks. The first three are L-addresses referring to the first storage locations of the sets y_i, k_i and q_i, in that order. (The numbers of each set are stored consecutively.) The next one is a constant equal to the number of equations. These four addresses can include x-flads, but not *v*-flads. If the number of equations is represented by an x-flad, it must of course be defined by an arrow. The last address is of the M-type, usually some *v*-flad. The complete w.ch. G then reads

$$G\ [y_0]\ *\ [k_0]\ *\ [q_0]\ *\ n\ *\ v\alpha$$

The *v*-flad refers to the first order of a routine, which computes the right hand sides multiplied by h, i.e. performs the operations

$$hf_i(y_0, y_1, \ldots, y_{n-1}) \rightarrow k_i$$

This routine will as a rule precede immediately the w.ch. G. (No separate routine heading is needed.) The calculations above are performed four times during one step of integration. The counting to 4, taking place inside w.ch. G, is carried out in B5. Therefore the routine above must preserve the contents of this B-register. Furthermore B5 must be set equal to zero at the beginning of each step of integration.

Before the integration is started, the y_i's are given their initial values and the q_i's are set equal to zero. The layout of an integration programme can be as follows.

```
initial values → yi
0 → qi
┌──→0 → B5                                  ⎫
│  ┌→hfi → ki        (α                     ⎪
│  └─G [y0] * [k0] * [q0] * n * vα          ⎭
└─────count steps of integration
```

As already said the innermost loop is implicit in w.ch. G, the jump address being the last of the five addresses written after the warning character. The part of the programme indicated by the bracket advances the integration by one step, i.e. the variables y_i are replaced by $y_i + \triangle y_i$, with $\triangle y_0 = h$.

If desired the interval can be changed at any time. The truncation error of the process is of the order h^5 per step.

4.7.3 *W. ch. N*

This is an open subroutine for the generation of pseudo-random numbers. The length is 6 registers, and the time is 0.60 ms. Only the accumulator is used. The letter N is followed by an L-address, which must *not* contain *v*-flads. The routine will replace the indicated number by a new random one. The latter is present in the accumulator when leaving the routine. The random numbers generated by repeating this process will be positive numbers in general unstandardized. The exponent part will remain constant. The number should be given a *rounded* initial value (different from zero). If this value lies between 1 and ½, the random numbers will be distributed uniformly in the range [0, 1]. The middle word of the fractional part can be considered a random short number.

Different patterns of random numbers can be obtained by choosing different initial values. The generating process is cyclic, and the period is 2^{27}, provided that the last bit of the initial value is a one.

4.7.4 *W. ch.'s K (KOMPLEX)*

The letter K represents a series of warning characters for simplifying the programming of complex arithmetic. The real and imaginary parts of a complex number are stored in adjacent storage locations, the former preceding the latter. The operations upon such numbers are carried out by a series of B1-closed subroutines covering the pages no. 1, 2 and 3.

These routines are stored on the sectors 49—51, which are not isolated. If the routines are not present in the backing store, they can be read in from a standard binary tape (see 4.8.1), which also contains the subroutines for double length arithmetic (4.7.5) stored on the sectors 52—55.

The subroutines for complex arithmetic are read down from the drum by means of the w.ch. K, followed immediately by CR LF, or possibly a label. The warning character represents the orders

```
670 49
680 1
670 50
680 2
670 51
680 3
```

which are stored in the programme.

Note that a chapter making use of the subroutines for complex arithmetic must begin on P4.

The locations L1.0 and L1.2 constitute a «pseudo accumulator» which plays the role of an ordinary computing register. Corresponding to the ordinary arithmetical orders these are «orders» associated with the complex pseudo accumulator. The first «function digit» is the letter K, the second

some symbol on figure shift, and the third one is the B-number as usual. The address part is an ordinary L-address of any type, specifying the real part of the operand. The imaginary part is normally the next long number. However, it is possible to refer to a «real» operand by terminating the address by an asterisk, in that case the imaginary part is considered to be the floating point zero in L.40.

The complex orders are:

K = b a(*)	copy into the pseudo acc.
K → b a	copy from the pseudo acc.
K + b a(*)	add.
K — b a(*)	subtract.
K x b a(*)	multiply.
K n b a(*)	negative multiplication.
K ≠ b a(*)	convert from polar form.

The arithmetic operations are rounded. The B-number, b, must always be written, even if it is equal to zero. Note that the modifying B-register must count in steps of 2, except when the address refers to a real operand. *NB!* The address part of the order K → b must *not* be terminated by an asterisk. In all cases above the operand can be the pseudo accumulator itself.

The function K≠ requires the numbers r and φ to be in consecutive storage locations, the arc being in units of right angles. The result is left in the pseudo accumulator. There is no built in sequence for the conversion into polar form, however, this is easily programmed by means of Q5 (arctg).

The warning characters above occupy four registers. The following orders are placed in the programme.

2m	40b	a	real part of operand
2m + 1	30b	a + 1	address of imaginary part
2m + 2	101	2m + 4	set link
2m + 3	590	———	jump to proper subroutine
2m + 4	———	———	return here

If the address part contains an asterisk, the second order is replaced by 300 .40.

In addition to the complex orders mentioned so far there are four which only operate upon the pseudo accumulator or the ordinary one. These have no address part and no B-number.

K/	Take the reciprocal of the contents of the pseudo accumulator.
Kv	Take the square root of the contents of the pseudo acc. (The real part of the result always positive.)
K⩾	Take the reciprocal square root of the contents of the pseudo acc. (The real part of the result always positive.)
*K**	Form the square of the modulus of the contents of the pseudo acc., and leave the result in A. The pseudo acc. remains unchanged.

These warning characters occupy two registers in the programme, the first two orders of the ones shown above being omitted.

All complex operations change the contents of A, S, St, B1 and Bt. In addition K/ and K*v* use L.32, L.34 and H39+. K≠ uses L.32, L.34 and L.36. The final contents of A are equal to the imaginary part of the pseudo accumulator, except by the w.ch.'s K* and K*v*. In the latter case it can be either of the two parts.

We finally list the times for the various complex operations, together with the entry points of the corresponding subroutines.

operation	time	entry point
K=	0.72 ms	1.10
K→	0.84 ms	1.14
K+	1.14 ms	1.19
K—	1.44 ms	1.24
Kx	3.00 ms	1.31

Kn	3.72 ms	1.30
K≠	11.10 ms	3.20
K/	6.99 ms	1.49
Kv	14.61 ms	2.23
K⩾	15.87 ms	2.21
K*	1.32 ms	1.51

The quickies Q0, Q1 and Q2 are incorporated in the subroutines for complex arithmetic. They are all B1-closed, and the entry points are 1.52, 2.38 and 3.26, respectively. If none of the operations K≠, K/, Kv and K⩾ are required, only P1 (sector 49) need be read down from the drum.

4.7.5 *W. ch.'s L (LONG)*

This is a group of warning characters similar to the w.ch.'s K, used for double length arithmetic. The B1-closed subroutines performing the operations occupy three pages, P1, P2 and P3. They are stored on the sectors 52—54. If these sectors have been destroyed, the routines are read in again from a standard binary tape (cf. 4.7.4).

A double length number is represented by two adjacent floating point numbers. The last binary digit of the most significant part (the first one) is always equal to zero. The least significant part is positive. The difference of the binary exponents of the two numbers is ⩾ 28, except when both parts are equal to zero. A double length number is referred to through the address of the most significant part. If the address is terminated by an asterisk the least significant part is considered to be the floating point zero in L.40.

The double length numbers are operated upon via a pseudo accumulator in 1.0 and 1.2. The available operations are exactly analogous to those applying to complex numbers, (the letter K should be replaced by L in the written «orders»), with two exceptions. L≠ does not exist, and L* forms in A

the best rounded value of the most significant part of the pseudo accumulator, the latter remaining unchanged.

The contents of the pseudo accumulator must always have the standard form described above. To convert an ordinary number (with the address a) into a double length one, it is necessary to use the operation

$$L = 0 \quad a^*$$

unless one can be sure that the least significant bit of the number is equal to zero. In the latter case only, the necessary operations may be written out explicitly using ordinary orders.

The representation of the w. ch.'s L as stored in the programme is exactly as for the w. ch.'s K. The routines for double length arithmetic also use the same registers and storage locations as used by the complex operations. The contents of A will be the least significant part of the pseudo accumulator.

We list the times for the operations and the entry points of the corresponding routines.

operation	time	entry point
L=	0.96 ms	1.14
L→	0.84 ms	1.35
L+	4.35 ms	1.40
L—	5.31 ms	2.09
Lx	4.86 ms	2.12
Ln	5.88 ms	2.15
L/	20.64 ms	2.53
Lv	32.34 ms	3.17
L⩾	27.00 ms	3.21
L*	0.84 ms	2.41

If the address part of one of the first six warning characters (excluding L→) is terminated by an asterisk, the time will increase by 0.60 ms. The quickies Q0 and Q1 are incorporated in the subroutines for double length arithmetic. They are

B1-closed and have the entry points 2.40 and 3.16, respectively. If none of the operations L/, Lv, and L$\geqslant$ are used in a programme, the last page, P3, need not be read down from the drum.

4.8 *Miscellaneous*

4.8.1 *W. ch. B (BINARY)*

This warning character is used for the input of information in binary form. (See also sect. 6.2.) It is followed by 130 short words, the ten binary digits of each being divided on two consecutive tape rows. The first word is a sector number, then come the 128 words to be stored on this sector, and finally there is a control word chosen so that the sum modulo 1024 of all 130 words shall be equal to zero. If this check sum fails the machine comes to a loop stop, (see table III).

When one sector has been read in and stored, the input proceeds in the normal way, except when the sector read is number 48. In the latter case the machine will stop reading and enter the programme now stored on the drum. (See sect. 6.2.)

When a programme is partly in binary partly in decimal form, the binary parts must be read in first. The binary programme can not be given chapter headings and referred to by the w. ch.'s A and D in the usual way.

4.8.2 *W. ch. T (TITLE)*

Usually the very first information on a programme tape is the w.ch. T followed by a title defining the problem to be solved. The title which is read in and stored on sector 48 is a «chinese copy» of the tape, including errors, etc. It does not, however, include the w.ch. T itself. Any combination of tape characters is permitted, except consecutive figure shifts. The title can contain up to 248 characters (more than three full lines on the teleprinter sheet), and is terminated by

three FS's. It is customary to preface the title by CR LF, however, even if it begins with characters on letter shift, the w.ch. T must be terminated by a FS. The title can be punched out before the calculation is started. (See 6.5.)

4.8.3 *W. ch. £*

This warning character can be used to ensure that the next item is stored in an even numbered register. If the first empty one has an odd address, a dummy order is inserted here. A plus (or a minus) can also be used, if necessary written on a separate line.

4.8.4 *W. ch. W (WAIT)*

When a w.ch. W is read, the machine stops (in a 992-order). The input is restarted by pressing the *prepulse button*. The warning character can be inserted between any two lines of the programme. It is used for changing tapes.

4.8.5 *W. ch. E (ENTER)*

This warning character is the last item on the programme tape. The machine will stop reading, calculate all the v- and n-addresses, (which up to now have only been recorded), and finally enter the programme. Various output may be given, (see sect. 6). During the calculation of the v-addresses the machine will check that they have all been defined, (see sect. 4.1.9).

W.ch. E must be followed by an (M-type) address containing a routine specification and terminated by *CR LF*. It represents in fact an «across» from the input programme to the chapter containing the specified routine. The cue to this chapter, the «entry cue» is stored together with the title on sector 48, where it can be used to re-enter the programme on later occasions, (see sect. 6).

When entering the programme page 15 contains the title

sector with the entry cue, and page 0 contains the CCS, etc. The rest of the high speed store contains short zeros, except the pages covered by the first chapter. The contents of the standard working positions L.32, L.34 and L.36 are floating point zeros, whereas the locations L.44 to L.62 contain parts of the input programme. The initial contents of all computing registers are equal to zero, except B7, which contains the entry address.

4.9 *Correcting programming errors. W. ch. X*

When an error has been located in a programme an obvious method to correct it is to reproduce the tape, and insert the necessary alterations. However, this is a tedious and time consuming process, which may often have to be repeated several times until the programme eventually is correct in all details.

We shall in this section show how a programme can be corrected when stored in the machine, by means of small additional tapes specifying the corrections. Such a tape begins with a w.ch. X (the character on letter shift) followed by a floating (or mixed) M-type address containing a routine specification.

The orders etc. which follow will be stored from the specified register and onwards, thus overwriting the old contents of these storage locations. The correction can be terminated by a new w.ch. X, followed by another correction, etc. The last one must be terminated by a w.ch. E with the same address part as the one on the original programme tape, or another one if this entry address is no longer adequate. No chapter or v-routine heading may be used in an X-correction, unless the address part of the w.ch. X refers to the end of the last routine on the programme tape.

It is important that any w.ch. X contains a routine specification. The v-flads used in the correction will be those belonging to this routine, (unless another one is specified

explicitly). It is also possible to define new v-flads in the programme which follows after a w.ch. X. There are 4 v-numbers reserved for this purpose in each routine, viz. those immediately following the largest v-number in use in the original routine. Furthermore, if the v-flads used here are not numbered consecutively, the numbers lying between are free to be used in an X-correction. Any number of new v-flads can be defined belonging to the last routine on the original programme tape.

It is often necessary to extend a chapter by X-corrections in order to correct a failure. This is permitted, provided that the chapter will not exceed the space reserved for it in the computing store and on the drum, (see sect. 4.4.3, w.ch. P). If it does, the machine comes to the loop stop mentioned in section 4.4.3. The orders added must be placed at the end of the chapter, no norder can be added in front.

When a chapter, as a result of an X-correction, is increased by one or more sectors, it is necessary to modify those cues in the programme which refer to this chapter. This will be done, but only if the total number of cues in the programme and previous X-corrections (w.ch.'s A, D, Z) is $\leqslant 32$. If there are more than 32 cues in all, the last ones will not be treated correctly, when a w.ch. X is encountered.

A page added to a chapter because of an X-correction will not in general be filled up by short zeros. The «empty» part of it will contain the old rubbish originally found on the drum sector in question.

When w.ch. X is used to correct an undefined v-address, it is important that every order using the address is corrected. (They have all been indicated by the machine, see sect. 4.1.9.) If the missing definition is a label the easiest method is to refer to the point through another label, i.e. to replace the faulty address by one which is defined. However, there are cases when this is impossible. Then the missing label must be supplied by an X-correction too. (See the example below.)

x-flads can not be used in an X-correction without being defined anew. A w.ch. E destroys all x-addresses, and the one terminating the original tape has already been encountered.

X-corrections to a programme can only be used if the drum directories mentioned in sect. 5.0 are intact. In order to prevent these from being cleared when the initial transfer button is pressed, the hand switch no. 3 should be up. (See 6.7.)

When all necessary X-corrections have been provided the programme can be punched out in binary form, (see sect. 6.3). If it is wanted in decimal form the various correction tapes must be considered integral parts of the original one. However, it is recommended to reproduce the programme tape and include the necessary alterations in the final copy.

Example.

In the orders below, which are part of, say R5, in a large programme, the label (14 is missing on the tape.

```
————
101 v14                set link
————      (20
————
————
* * *      (23   } punch A as a three
Q11              } digit integer
—591   0          close
→620   30   (14   punch CR
620    13         punch LF
———
———
```

Since it may be difficult to judge whether or not there is a dummy order in front of Q11, we can not replace $v14$ by an address referring to $v23$. The flad $v14$ must therefore be defined. We go backwards in the manuscript until we find an item which can be referred to unambiguously, e.g. the

asterisks, which are labelled. We rewrite the programme starting from there until we reach the order 620 30, this time including the label. The next X-correction serves to correct the order 101 $v14$, which would otherwise continue being without an address.

```
Xv23/5
* * *
Q11
591  0
620  30  (14
X  —1v20/5
101  v14
E ————
```

It is necessary that the X-correction defining $v14$ precedes one using it, because forward references *across* a w.ch. X are impossible. Note that the label (23 must be left out this time. This label has already been recorded during input of the original tape, and the machine would otherwise think that we tried to redefine the flad $v23$, which is not allowed.

Further examples on the use of X-corrections are given in sect. 8.2.

5.0 USE OF STORAGE SPACE

The general layout of the high speed store has already been discussed. We will here only mention that the data are usually placed on the highest numbered pages. This is obvious when the data concerned are long numbers, since the second half of the store can not contain programme.

On the drum the 48 first sectors are permanently isolated.

That is, their contents can be read down to the computing store, but nothing can be written on them. The contents of these sectors are thus protected against being destroyed by accidents. We list the contents of the first 4 sectors.

Sector 0: starting programme.
Sector 1: binary input programme.
Sector 2: short zeros.
Sector 3: the CCS etc.

The sectors 5 to 47 contain the general input programme, the standard subroutines (quickies etc.), and the test programme (see sect. 7). Sector 48, which is not isolated, is used to store the title and the entry cue, and the sectors 49—54 are normally reserved for the subroutines for complex and double length arithmetic. The programme itself is usually stored from sector 55 and onwards.

The 32 highest numbered sectors (480—511) contain various engineer's tests, which do not concern the programmer. However, these sectors are normally isolated. The following sectors (going downwards) contain various directories used by the input programme. They are stored in an order corresponding to the relative importance of the data.

1) *Sector 479*
   Data concerning the chapters (max. 25).
2) *Sectors 477—478*
   Data concerning the v-routines (max. 51).
3) *Sector 476*
   Data concerning the cues (max. 32.)
4) *Sectors 475 and downwards*
   Data concerning the v-addresses, their values and how they are defined. There are 64 v-addresses on each sector, and to each v-routine is reserved space for m + 5 flads, where m is the highest v-number in use in the routine. The directory is allowed to extend down to sector 436 (40 sectors in all). If in some enormous programme there should be too many v-flads defined, this will be indicated by a loop stop. (See table III.)
5) *Sectors 435 and downwards*
   Data concerning the address parts of the programme using v- or n-flads. One sector can contain information about 32 address parts, (less if they contain more than one v- or n-flad each), and the directory corresponding to each chapter begins on the top of the next sector. (The directory really starts on sector 434; sector 435 will contain some rubbish.)

When binary output of the programme is demanded (see 6.2), the directory 1 must be intact. The test programme and w.ch. X both need the directories 1, 2 and 4, the latter also uses 3. The directory 5 is used only temporarily by the input programme itself. When a w.ch. E or X is encountered the v- and n-flads in use are calculated, and afterwards the directory is no longer needed. A new directory of the same type is formed (starting on sector 434), when the orders following a w.ch. X are read in.

One must see that a programme is stored on the drum so that it does not collide with the directories mentioned above. There is a loop stop in the input programme for checking this (see table III). The exact condition for the stop to occur is that the last sector of the programme lie within the directory 5.

As already said a programme is usually stored from sector 55 and onwards. The drum space from the end of the programme up to sector 421 (see sect. 7.6), can be used freely as storage space for data during the calculation. If higher numbered sectors are used the facilities for testing and correcting etc. will be limited. It is recommended to store the data as far down on the drum as possible. (Cf. 6.8.)

The entire drum can be brought into use by de-isolating all sectors and storing the programme on the highest num-

bered ones. The input programme is read in again from a binary tape by means of a simple «bootstrap», or by ordinary binary input (hand switch 8) if the sectors 0 and 1 are still intact.

6.0 THE STARTING OF THE MACHINE

Before starting the machine the tape flip-flops are reset by pressing the appropriate button. The tape, if any, is inserted in the tape reader, TR2 or TR3, and the hand switches (see below) are set according to the desired effect. Finally the initial transfer button is pressed. The effect is that of the three orders

```
670  0
680  0
590  0
```

i.e. the contents of sector 0 is read down to page 0 and entered in register 0. The further course of action depends upon the setting of the 10 hand switches (no. 0—9) in the bottom row on the console. The effects of these switches are described in the following sections, except the switch no. 1. This has to do with engineer's tests and is of no use to the programmer.

The other switches on the control desk will not be described in this handbook.

6.1 *H.s. 9. Decimal input*

Programme in ordinary decimal form is read in by having h.s. 9 up. The slower tape reader, TR2, must be used. H.s. 9 can be up together with any of the switches 7, 6, 5, 4, 3, 2 and 0. (It assumes an entirely different meaning when h.s. 2 is up.) If the hand switches 9 and 8 are both down no input will take place. Instead the machine will enter the programme already on the drum, according to the entry cue on sector 48. (This sector must not have been disturbed since the programme was read in.) Also in this case the switches mentioned above can be up, provided that the necessary directories are intact.

6.2 *H.s. 8. Binary input*

Binary tape can be read in by having either of the switches 9 and 8 up. In the latter case the faster tape reader, TR3, can be used. When h.s. 8 is up the entire programme must be on one single binary tape. No warning character (except B) will be obeyed.

The last sector read in is always the title sector (no. 48). Then the programme is entered according to the entry cue stored together with the title. If the check sum fails during binary input, the machine will come to a loop stop (see 4.8.1). Only the switches 5 and 0 can be up together with h.s. 8.

The standard binary tape containing the routines for complex and double length arithmetic is terminated by a w.ch. W (not by a title sector). Therefore this tape must be read in with h.s. *9* up.

6.3 *H.s. 7. Binary output*

When this hand switch is up the machine will punch out in binary form the programme stored on the drum (except parts read in from a binary tape). Each sector is punched as described in section 4.8.1, preceded by the tape characters FS, LS, B, FS. The programme is terminated by the title sector containing the title and the entry cue. When this has been punched the machine comes to a stop order (996). If restarted by a prepulse it will examine the hand switches further and finally enter the programme. If there is more output to come, the binary tape should first be removed from the punch. Binary output is possible only if the directory on sector 479 is intact.

If h.s. 3 is up together with h.s. 7, the directories 1—4 of section 5 are punched out in front of the programme. When such a tape is read in again the facilities for testing, correcting, etc. are still available.

6.4 *H.s. 6. Output of storage allocation*

This hand switch is used to acquire information concerning the storage of the programme on the drum and in the computing store. When the hand switch is up output exemplified by the following is given.

CHAPTER 00	PAGE 01 - 10	SECTOR 049
ROUTINE 000	FROM 01 . 00	
ROUTINE 010	FROM 05 . 42	
ROUTINE 011	FROM 08 . 06	
CHAPTER 01	PAGE 01 - 06	SECTOR 060
ROUTINE 011	FROM 01 . 00	
ROUTINE 012	FROM 04 . 26	

The output is separated from further output by LF's and FS's.

The page numbers given represent the actual lengths of the chapters, and may be different from those given in the w. ch.'s P on the programme tape. As will be seen 11 sectors have been reserved on the drum for chapter 0, therefore the corresponding w.ch. P must have read P1—11.

The directories 1, 2 and 4, mentioned in section 5.0 must be intact when h.s. 6 is used.

6.5 *H.s. 5. Output of title*

When h.s. 5 is up the title is punched out. This output is given immediately before the programme is entered, and the title will therefore come out as a heading to the numerical results of the calculation. Since the title is always present on a binary tape, it does not matter whether the programme was read in in decimal or binary form, however, sector 48 must not have been destroyed.

6.6 *H.s. 4. Test programme*

This hand switch is used in conjunction with the test programme. See section 7.2 for an exact description.

6.7 *H.s. 3. Preserve directories. (W.ch. X)*

When reading in a programme in the normal way by having h.s. 9 up, the first thing to happen after the initial transfer has been given, is that all drum directories are cleared. Since the old contents of these sectors are needed when reading in an X-correction, the clearing must be prevented. This is achieved by having h.s. 3 up too. The hand switch has also a significance in conjunction with h.s. 7.

6.8 *H.s. 2. Store — restore*

There are occasions when it is desirable to have more than one programme inside the machine at a time. H.s. 2 can be used to store a programme on some safe part of the drum, and later bring it back again into its original position, where it can be reentered and executed anew. All directories needed for X-corrections and the test programme (1—4, sect. 5) are transferred back and forth together with the programme. (However, only the directories 1 and 2 need be intact for the storing.) All transfers are checked, faulty ones being repeated. Parts of the programme read in from binary tape will not be stored. The procedure is as follows.

Store

Press the initial transfer button with h.s. 2 up. The machine stops in a 995-order. Set the hand switches to a chosen binary sector number and press the prepulse button. The programme and the directories are then stored from the indicated sector and *downwards*. When the machine comes to a halt (in the 996-order mentioned in sect. 6.3), the contents of B1 are equal to the number of the next free sector, (should be written down).

NB! If the TP has been used (see sect. 7), the programme *must be restored* before it is moved. (Initial transfer is given with hand switch 4 up.)

Restore

An initial transfer is given with the switches 9 and 2 up. The further course of actions is exactly as described above, the same sector number being used. As a check one should see that the final value of B1 is the same as before. The programme is entered by a prepulse (two if h.s. 0 is up), or, as usual, by an initial transfer with the hand switches reset.

The store—restore facility is particularly useful when several users are developing their programmes. In order to utilize the machine efficiently there should be a certain circulation in front of the machine. Each programmer should prepare all his tapes in advance, and leave the machine when the output has been given. On the basis of the information obtained he can prepare new tapes (X-corrections and test tapes), and wait for his next turn, etc. To prevent waste of machine time by reading in the programmes again and again they should be stored. The following procedure is recommended.

Some high sector number is selected in the morning. (It should be as large as possible, but sufficient space for directories must be allowed. As a rule sector 384, 0110000000, will be suitable.) The first programmer uses this sector number when storing his programme. He also records the final value of B1, which should be the sector number used by the next person, and so on. Of course the sectors used must be left unchanged by the programmes under development. However, data are normally stored on low numbered sectors, and, as a rule, only moderate amounts of data are used when developing programmes.

6.9 *H.s. 0. Stop*

When this hand switch is up the machine will stop in a 993-order immediately before the programme is entered. The machine is restarted by pressing the prepulse button.

7.0 THE TEST PROGRAMME

7.1 *Aims of the TP*

When a new programme is tried on the machine it will usually be found to contain faults. These faults will show up in several ways. Either the machine goes into a loop, or it stops, or it punches wrong answers. The reason for a stop is either that a stop order has been reached or the exponent register has exceeded positive. In all cases it may be difficult to find out where the fault has occured and what it was.

The TP is an aid to trace the actions of the programme. At certain chosen *break points* the TP will cause information to be punched out which should help the programmer to locate his errors. Outside the break points the programme is obeyed at the machine's normal speed, except the chapter changes which are slowed down.

7.2 *Entry of the test programme*

The TP is entered by pressing the initial transfer button with the hand switch no. 4 up. This is the case when the programme to be tested is already on the drum. If it is not, the programme can be read in by an initial transfer with the switches 9 and 4 up. In either case a stop is encountered (a 994 order). A steering tape defining the break points and stating the required output is inserted in the tape reader (TR2 must be used). The tape is read in when a prepulse is given. Now the machine changes the orders found in the

chosen break points to jump instructions and enters the programme.

When one of the inserted jump orders is encountered, the TP takes over and may give output. The programme calling in the TP occupies the same space as the CCS does normally. It follows that the first half of P0 must be left undisturbed by the tested programme, if the TP shall be able to work correctly.

If new tests shall be made, the test programme is entered as before. Before the machine comes to the stop it will restore the old break points by inserting the orders which originally were found here. *NB!* When an error found by the TP shall be corrected with an X-correction, the programme must be restored first. That is, the initial transfer button must be pressed with the hand switch no. 4 up before the correction tape is read in.

7.3 *Definition of a break point*

All break points must be defined by *v*-flads. The break points are numbered from 1 to 18. There may be up to 36 in one programme, but the number of break points in one chapter must not exceed 18. If necessary the same numbers can be used for break points in different chapters. A break point is defined on the steering tape by an equation exemplified by

$$C2 = 3v7/5$$

The routine no. is not always needed as the TP will always use the routine defined last. If no routine at all is defined routine no. zero is taken. The break point thus defined is before the order specified on the steering tape and after the previous order, or stated differently, the output from the TP takes place before the order in the break point is obeyed.

There should be a sufficient amount of labels in the programme to ensure that reference to any point can be made.

It is often difficult to know whether or not a dummy order has been inserted in front of quickies, long numbers, etc. Therefore it will often pay to write labels at points where they are not needed in the programme itself.

It is possible to define a conditional break point. The machine will then give output only if one or more B-registers have specified values at the break point. If e.g. output is wanted only when B2 = 3 *and* B3 = —7, the steering tape contains after the definition of the break point itself, but not necessarily before any output directives, the equations

$$R2 = 3$$
$$R3 = -7$$

It is also possible to obtain output if B2 = 3 *or* B3 = 7. The tape may then be

```
C1 = v3/8
R2 = 3
———                 }
———                 }
———                 }  output directives
———                 }
C2 = v3/8
R3 = 7
———                 }
———                 }
———                 }  output directives
———                 }
```

The actual course of actions is then as follows. When the first break point is read in, the order in *v*3/8 is replaced by 590 1 and the original order is stored. Then the condition and the output directives are stored. When the second break point is read in, the order 590 1 is replaced by 590 2 and the former is stored with the new condition and the output directives.

When the programme is obeyed the second break point is reached first and the condition R3 = 7 is tested. Afterwards the order 590 1 is obeyed which leads to the first break point where the condition R2 = 3 is tested. Finally the original order in *v*3/8 is obeyed and the programme is reentered.

When the machine reaches a break point with a not satisfied condition, it uses about 1/3 sec to find this out. Two break points at the same place will therefore take 2/3 sec etc.

7.4 *Output directives*

When output is given the break point itself is always punched out. Further the contents of a B-register may be punched by a directive on the steering tape exemplified by

```
B5
```

The directive B0 refers to the SAC-test register and B8 to the B-test register. These are punched out in the form 0, 1 or 1023.

The accumulator is punched out by the directive

```
A
```

A short word from the first half of the computing store is punched by directives exemplified by

```
H  .32
H  46+
H  3v7/8
H  4v5
```

In the last case *v*5 is taken to be in the routine last defined by a /. (x-addresses must not be used at all on the steering tape.)

A long word anywhere in the high speed store is punched out by directives exemplified by

```
L  .32
L  5v7/8
L  v6
```


In the last case the routine is again taken to be the last one defined.

If several consecutive long or short numbers are wanted, directives exemplified by

```
H  v3/7     (6)
L  13.0     (50)
```

are used. The address part refers in both cases to the first number of the series, and the required number of words is enclosed in brackets.

It is also possible to punch out short or long numbers from the drum. The directives

```
S  103
L  0        (5)
H  .17      (8)
```

will cause output of the first 5 long numbers and the eight short numbers from relative address .17 on sector 103.

The directive

```
S  103
L  .20      (40)
```

will punch out 22 long numbers from sector 103 and the first 18 numbers on sector 104.

The directive

```
S  103      (4)
L  6        (4)
```

punches the four long numbers with relative addresses .12, .14, .16, .18 on the sectors 103, 104, 105 and 106.

If, in one break point, output is required from both types of store, the directives referring to the high speed store must always preceed those referring to the drum. A complete set of output directives for one break point may e.g. look like this

```
C3 = v6/18
R2 = 6
A
B3
L.32
S  104
L.14      (6)
H0        (3)
S109      (3)
L0
```

Specifications for further break points may follow. The steering tape is terminated by the letter E (for ENTER).

7.5 *Form of output*

The first output begins with punching out the chapter no. in the form

CH3

followed by the break point in the form

C4 = 5.48

As no floating addresses will appear, the break point number is used (together with the chapter number) for identification of the break point. For further break points within the same chapter the chapter no. is not punched out again except if chapter changes have taken place. The fact that the chapter no. is punched out therefore tells that one or more chapter changes have taken place since the last output.

If output directives are included on the steering tape outputs exemplified by

```
B3 = 26
A = 3.59278356,  —3
H12.63+ = 1022
L6.48 = —4.59327286,  6
```

are given.

The output corresponding to a directive as

H3.18 (10)

takes the form

```
H3.18 = 60
13
7
1023
216
0
368
900
 135
23
```

There is a space in front of every eighth number.

Output from the drum is preceeded by a heading of the form S209.

The second and following times the same break point is encountered by the machine, the output is simplified. All names and addresses are left out, except the sector and break point headings. The address of the latter is also omitted.

The first number in a sequence corresponding to an L or H directive is preceeded by an equality sign. This is the case even if the sequence contains only one number.

7.6 *Limitations of the test programme*

It has been said previously that the first half of P0 must be left undisturbed by a programme under test. Also the drum directories mentioned in sect. 5.0 must be intact every time the TP is entered. It must in this connection be mentioned that the test programme uses the sectors 422—435, whose contents are destroyed when a break point is encountered.

The choice of break points is subject to two restrictions. One is that orders whose addresses or functions are set by

the programme must never be used as break points. This is because the jump order inserted by the TP is then destroyed. The second case is that a break point must never be placed when the contents of the sector register is of importance. This is because each time a break point is encountered several magnetic transfers take place, and as it is impossible to read out of the sector registers, it cannot be restored. This case happens most often when a certain sector is read down from the drum, the information it contains treated in some way and then written up again to the same sector. It is then not necessary to set the sector register by the order 67 more than once. To facilitate the use of TP this practice should be discouraged.

The other limitations to the TP are unimportant. We have already mentioned that no chapter may contain more than 18 break points and no programme more than 36. There must furthermore not be more than 72 output-directives and 20 R-conditions.

7.7 *Fault indications*

There are two loop stops in the TP. One occurs in 3.2, the other in 9.7 to 9.9. The first indicates than a non defined *v*-address has been used, the second that a non-existent routine has been referred to.

8.0 PROGRAMMING

In this short chapter we shall give a few hints on useful programming tricks, and conclude with an example showing the use of some features of the input programme and the associated test prgoramme.

8.1 *Special instructions*

There is no order for doubling a B-register, but SAC may be doubled by the instruction

327 n (S + (S + n) → S).

The order

357 —1 (S & (S — 1) → S)

can be seen to replace the least significant one of S by a zero.

This instruction is used for converting the tape characters on figure shift from their seeming disorder into a well ordered system. The most important teleprint characters are all originally given numbers less than 16, however, to those containing an even number of binary ones 16 has been added. The less important characters, originally given numbers ⩾ 16, have been modified by subtracting 16 from those containing an odd number of ones. Then the characters of the two groups are distinguished by having an odd or an even number of digits equal to one, respectively. (See table V.) This has the advantage that an error in the output punch, causing a «one» to be dropped or gained, will never convert a character in one of the groups into another belonging to the same group. In particular decimal digits can not be interchanged.

In order to convert teleprint characters from one system into the other the following instructions may be used. It is supposed that the character to be converted is in B6

```
    306   16
┌→  166   16   (1
│   357   —1
└—  280   v1
```

B6 will then contain the converted character.

The non-existent B-register B0 may sometimes be brought into «use». If we want to put a short number in the address H to test register Bt, we may write

000 H

If we want —H → Bt, we may use

```
070  H
```

The order 010 H has the effect of clearing the short word.

We have remarked that the order 14b for halving a B-register treats the contents of it as a positive number. The following orders will halve B6 considered as a signed number in the range $-512 \leqslant B6 \leqslant 511$.

```
126  512
146  256
```

The instructions

```
320   512
347  —256
```

will leave in SAC $[\frac{1}{2}(-S)] = [-\frac{1}{2}(S + 1)]$, where $-512 \leqslant S \leqslant 511$.

Often we want a programme to alternate between two possibilities characterized by two integers a and b. If then $a \not\equiv b = p$, we may perform the alternation by a non-equivalence with p, as

$$a \not\equiv p = b \quad \text{and} \quad b \not\equiv p = a.$$

The orders

```
136  (m—k)
090  v1
126  m
———  ————  (1
```

will count B6 cyclically modulo m in steps of k. If m is equal to some power of 2, the following orders can be used instead.

```
126  k
156  (m—1)
```

As previously explained a signed short number can be converted to a long one by placing it in H39+, the most cignificant part of 0.2^9. An unsigned short number in the range [0, 1023] can be converted by placing it in the middle

fractional part of a long number 0.2^{19}, or in the least significant part of 0.2^{29}.

A long number can be converted to a short integer by using the orders

```
460  28
410  v1
```

The least significant part, H1v1, of the long number is then equal to the integral part of the former contents of A.

The product of two integers can be found by placing them in the least significant parts of (arbitrary) long numbers. When these are multiplied together using the order 540, the least significant part of the result is equal to the product modulo 1024.

A fully worked example

As an example we will compute auto- and cross correlation coefficients of periodic functions. Let the two functions be f_k and g_k and let

$$f_{k+n} = f_k$$
$$g_{k+n} = g_k$$

We then want to compute

$$C_m = \frac{1}{n} \sum_{k=0}^{n-1} f_k \; g_{k+m}$$

We store the function values f_k on the drum and g_k in the high speed store.

```
C1  P  1—3
R1
990  0          wait for numerical tape
102  0          no. of variable
103  0          rel. sector no.
104  0          no. on page
101  v1
→ Q 10  (2      read in, ignore
  176  30       numbers not terminated
  080  v2       by CR
  591  0        closure
  414  6.0  (1  store f_k
  124  1        count modulo 32
  154  31
  080  v3
  673  100      write up, increase
  690  6        sector no. by one
  123  1
→ 172  —1x1  (3 test end
  182  v2
  673  100      write up last sector
  690  6
  990  0        wait for new tape
  102  0
  101  v4
  590  v2       read in all g_k's to
→ 412  4.0  (4  the computing store
  172  —1x1
  182  v2
  400  v5       x1 in acc.
  440  .40      standardize
  Q0            1/x1 in acc.
  Av/2
  =9  (5
  =0
  =0            x1 as a long number
  =x1
```



```
C2  P1—2
R2
410  v4         store 1/x1
101  0          0 → m
→ 301  0  (5
  102  0        sector no.
  103  0        no. on sector
  670  100      first sector down
  680  3
  400  .40      0 → cor. coeff.
  410  .32
→ 407  4.0  (3  g_{k+m} f_k
  503  3.0
  420  .32
  123  1        count modulo 32
  153  31
  080  v1
  122  1        increase sector no.
  672  100      read next sector down
  680  3
→ 320  —1x1  (1 count modulo x1
  290  v2
  320  x1
→ 371  0  (2    test end of coeff.
  280  v3
  500  v4       normalize
  620  30       CR
  620  13       LF
  106  3        punch index with 3 figures
  301  0
  Q9
  620  14  (10  2 spaces
  620  14
  * * * . * * *  punch coefficient
  Q11            with layout
  171  —1x1     test end, count on m
  181  v5
  990  0        stop
  +0  (4        1/n
  E  v/1
```

The tape containing the given data may also contain the numbering. During the reading in of the tape we will therefore only keep those numbers which are terminated by a CR. The orders

```
→ Q10      (2
  176  30
  080  v2
  591  0
```

form as we see a closed subroutine, but it has not been given any routine no. The input of the data and the formation of 1/n take place in chapter 1. There are two stop orders, one before the reading in of each tape. This part of the programme uses n, the number of function values. It is given by the x-address x1. The programme is therefore preceded by a short tape containing

```
x1 → n
W
```

The second chapter contains the computing proper and is described in the programme. If a different layout is wanted, one may use an X correction exemplified by

```
X2v10/2
*.***__***
Ev/1
```

When the programme shall be tested, we use a simple case. We should have more than 32 variables in order to test the magnetic transfers. We have chosen $n = 40$ and will compute the auto-correlation coefficients of a function consisting of ones and zeros. By simple counting we find $C_0 = 0.5$ and $C_1 = 0.35$. The introductory tape reads x1 → 40.

The programme is read in without accidents (no undefined v-addresses or other wrong punchings), but when the data tape is read in (twice in this case as we compute an auto-correlation function) the machine stops. The monitor for Ya shows that the highest bit of the exponent is zero, the next

highest equal to one, we have exceed positive. This is usually the sign that we treat an order as a number.

We will investigate this with the TP and put in break points at 1v3/2 and 2v3/2 and punch the accumulator. The steering tape is

```
C1 = 1v3/2
A
C2 = 2v3
A
E
```

The output begins as follows

```
CH2
C1 = 1.10
A  = 0.04318084,—78      (an exact zero, 0.2^-256)
C2 = 1.11
A  = 4.31808494,—78      (a rounded zero)
C1
0.04318084,—78
C2
4.31808494,—78
— — — — —
```

Soon we get for the first break point

```
C1
0.00000002,—1            (a strange zero, 0.2^0)
```

In the end the contents of A are not punched out after the break point heading C1 because of an exceed positive. Clearly the accumulator contains some rubbish after the order 407 4.0. We then investigate the B registers no. 7 and 3 (the last one is really unnecessary but added for good measure). The steering tape is

```
C1 = v3/2
B7
B3
E
```

and the output begins with

CH2
C1 = 1.9
B7 = 0
B3 = 0
C1
39
1
C1
78
2
etc.

Clearly B7 behaves wrongly. Instead of counting one at a time it counts in steps of 39. The error is then found in the cyclic counting on B7 where the order 320 —1x1 is put instead of 330 —1x1. This is easily corrected with an X-correction which takes the form

Xv1/2
330 —1x1
Ev/1

Before this is read in x1 must be defined again. We restore the break points of the programme, (initial transfer with hand switch 4 up). Then the tape defining x1 is read in (initial transfer with the h.s. 9 and 3 up), followed by the X-correction (prepulse). Finally the data are read in twice.

This time the result was better, in so far as 40 numbers were punched out, numbered from 0 to 39, but they are all zero. The reason for this is spotted easily enough. We punch the contents of the accumulator which has just been multiplied by 1/n. As this number (in v4) clearly is set (the first order in R2) we conclude that we have zero in A the whole time. The reason is obvious when we observe that after the order 420 . 32 we have forgotten to put 410 . 32.

There is a special technique for correcting this type of error. We replace one order (e.g. 420 . 32) by a jump order to the bottom of the chapter, and there we put the over-

written order, the missing orders and a jump back. The bottom of the chapter is found in relation to v4, but we must remember that the X-correction uses M-type addresses, so the first free register is 2v4/2. The correcting tape is then

X2v3/2
590 2v4
X2v4/2
420 . 32
410 . 32
590 3v3

Under this we copy in the previous X-correction (terminated by Ev/1) so we get all corrections on one tape. We again read in the tape defining x1 (we are repeating the X-correction containing x1) and then the new X-tape. To our surprise the machine comes to a high pitched note and stops indicating an undefined v-flad. We check the punching of the tape, but all is in order. We look at B registers 5, 6 and 7 and see that we have an undefined v32 in C2 referring to R2! There is no v32 on our tape so we print it out and find that the order 420 . 32 has been printed as 420 n32. The teleprinter punch has failed. We correct the tape and read it in again followed by the data and this time the correct output appears headed by C_0 = .500 and C_1 = .350 as it should be.

LIST OF FUNCTIONS.

Table I.

	0	1	2	3	4	5	6	7	8	9
0	B' = Bt' = H	H' = B	B' = Bt' = B + H	B' = Bt' = B — H	B' = Bt' = ½B—H	' = Bt' : B & H	B' = Bt' = B ≠ H	Bt' = B—H	Bt ≠ 0 C' = n	Bt ⩾ 0 C' = n
1	B' = Bt' = n	B' = Bt' Carriage	B' = Bt' = B + n	B' = Bt' = B — n	B' = Bt' = ½B—n	' = Bt' : B & n	B' = Bt' = B ≠ n	Bt' = B—n	Bt≠0, C'=n always Bt' = B' = B + 1	
2	S' = St' = H	H' = S H' = S	S' = St' = S + H	S' = St' = S — H	S' = St' = ½S—H	= St' : S & H	S' = St' = S ≠ H	St' = S—H	St ≠ 0 C' = n	St ⩾ 0 C' = n
3	S' = St' = n	R' = Rd' Space	S' = St' = S + n	S' = St' = S — n	S' = St' = ½S—n	= St' : S & n	S' = St' = S ≠ n	St' = S—n	St≠0, C'=n always St' = S' = S + 1	
4	A' = L	L' = A	A'=A+L round off standard.	A'=A+L round off standard.	A'=A+L standard.	= A—L andard.	A'=A+0.2^{n+1} no round off no standard.		Shift ⩽ 31 C' = n	A ⩾ 0 C' = n
5	A' = AL round off standard.	A' = —AL round off standard.	A' = (AL) standard.	A' = —AL standard.	A' = $(AL)_1$ no round off no standard.	=—$(AL)_1$ round off standard.		Dummy	Hoot	C' = n
6	H' = t	H' = hs	t' = n	t' = H	cr' = L			T' = n	P' = D	D' = P
7	B' = Bt' = E	E' = B	B' = Bt' = B + E	B' = Bt' = B — E	B' = Bt' = ½B—E	= Bt' B & E	B' = Bt' = B ≠ E	Bt' = B—E		
8	S' = St' = E	(E' = S)	S' = St' = S + E	S' = St' = S — E	S' = St' = ½S—E	= St' S & E	S' = St' = S ≠ E	St' = S—E		
9										Stop.

LIST OF TRUE FUNCTION VALUES.

	0	1	2	3	4
0	0101001 328	0100010 272	0101000 320	0101101 360	0101100 352
1	0001001 72		0001000 64	0001101 104	0001100 96
2	0111001 456	0110010 400	0111000 448	0111101 488	0111100 480
3	0011001 200		0011000 192	0011101 232	0011100 224
4	0100001 264	0100011 280	0100100 288	0100101 296	0100110 304
5	0110100 416	0110101 424	0110110 432	0110111 440	0010110 176
6	0000010 16	0010010 144	1010101 680	0010101 168	0110011 408
7	1001001 584	0010011 152	1001000 576	1001101 616	1001100 608
8	1011001 712		1011000 704	1011101 744	1011100 736
9					

The table gives the true function values in decimal and bits giving the B-numbers.

Table II

5	6	7	8	9
0101011 344	0101010 336	0101110 368	0000111 56	0010000 128
0001011 88	0001010 80	0001110 112	0001111 120	
0111011 4 72	0111010 464	0111110 496	0010111 184	0100000 256
0011011 216	0011010 208	0011110 240	0011111 248	
0100111 312	1100110 816		0110001 392	0010001 136
0000110 48		0000011 24	0110000 384	0000001 8
		0000100 32	0010100 160	0000101 40
1001011 600	1001010 592	1001110 624		
1011011 728	1011010 720	1011110 752		
				0000000 0

binary. The binary numbers should be followed by the three

Table III.

LIST OF LOOP STOPS.

PF = 0000111000 (80), CA = 0001011011 (1.27)
General tape test.
Contents of B6: Last read-in character in converted form.

PF = 0100000000 (290), CA = 0000010100 (0.20)
The second character of a function was not a digit.
Contents of B6: The faulty character in converted form.

PF = 0100000000 (290), CA = 0000100100 (0.36)
The B-number in the function was not a digit.
Contents of B6: The faulty character in converted form.

PF = 0100000111 (297), CA = 0010101111 (2.47)
Faulty character on Letter Shift.
Contents of B5: The faulty character.

PF = 0010111000 (280), CA = 0100101001 (4.41)
Definition of a v-flad already defined (just read in).

PF = 0100000000 (290), CA = 0011010110 (3.22)
Undefined x-address in use in a programme (just read in), or undefined v-address on a test tape.

PF = 0000001000 (590), CA = 0101111101 (5.61)
The chapter is too long.
Contents of B6: The number of chapter in question.

PF = 0100000000 (290), CA = 0110001101 (6.13)
Too many v-flads in the programme.

PF = 0100000000 (290), CA = 0100100000 (4.32)
Overlap between the programme and drum directories.

PF = 0010111000 (280), CA = 0000100111 (0.39)
Binary check sum failing.
Contents of B7: The check sum.

Table IV.

LIST OF STOP ORDERS.

PF = 0000000001 (991), CA = ?
Will occur if Q1 (or any of the functions Lv and L$\geqslant$) is *executed* with a negative number in A (or in the pseudo-accumulator).

PF = 0000000010 (992), CA = 0000000001 (0.1)
Caused by a Warning character W on the programme tape.

PF = 0000000011 (993), CA = 0000011000 (0.24)
Caused by Hand Switch 0.

PF = 0000000100 (994), CA = 1111010010 (15.18)
Waiting for a test tape to be inserted.

PF = 0000000101 (995), CA = 0010001111 (2.15)
Waiting for a binary sector number to be set on Hand Switches before moving a programme.

PF = 0000000110 (996), CA = 0010000110 (2.6)
Occurs when a programme has been moved or punched out in binary.

PF = 0000000111 (997), CA = 0110100101 (6.37)
Faulty v-address in use on the programme tape (Preceded by a hoot warning).
Contents of B5: the v-numbebr,
Contents of B6: the number of the chapter where the address is *written,*
Contents of B7: the number of the routine *referred to.*
B6 = —1: Faulty warning character E or X.
High pitched hoot: The v-flad is undefined.
Low pitched hoot: The routine referred to does not exist.

Table V.

TELEPRINTER CODE.

Character on tape	Decimal equivalent	Letter shift	Figure shift	Converted figure shift
000.00	0	Figure shift		0
000.01	1	A	1	1
000.10	2	B	2	2
000.11	3	C	*	3
001.00	4	D	4	4
001.01	5	E	(	5
001.10	6	F	)	6
001.11	7	G	7	7
010.00	8	H	8	8
010.01	9	I	≠	9
010.10	10	J	=	+
010.11	11	K	—	—
011.00	12	L	*v*	.
011.01	13	M	Line feed	LF
011.10	14	N	Space	SP
011.11	15	O	,	ER
100.00	16	P	0	FS
100.01	17	Q	>	>
100.10	18	R	⩾	⩾
100.11	19	S	3	*
101.00	20	T	→	→
101.01	21	U	5	(
101.10	22	V	6	)
101.11	23	W	/	/
110.00	24	X	×	×
110.01	25	Y	9	≠
110.10	26	Z	+	=
110.11	27	Letter shift		LS
111.00	28	.	.	*v*
111.01	29	?	*n*	*n*
111.10	30	£	Car ret	CR
111.11	31	Erase		,