A MANUAL OF GIER PROGRAMMING

VOLUME II

by

Chr. Gram

(translated by Alan George Lake)

A/S REGNECENTRALEN

Copenhagen, May 1964

PREFACE

When the English edition of volume I of the GIER manual was finished in October 1963 it was decided also to translate volume II, published in Danish in September 1963. A few chapters are omitted in the translation, as seen from the list of contents.

The present manual describes HELP, the package of auxiliary routines and the loading program SLIP, as seen from the point of view of a programmer.

Furthermore the drum memory, the peripheral units, and the control panel are described in detail. Besides the standard peripheral units you find descriptions of the RC 2000 paper tape reader, the punched card reader, and the line printer. However, the 4096-words buffer store, the tape units, the real-time input-output unit, and their connections to GIER are not mentioned here.

You will find many references to volume I, and the chapters are numbered in continuation of the chapters of volume I; this underlines that the two volumes constitute one book.

Acknowledgement is due to all who have assisted me during writing the book; but first of all I am indebted to Mr. H.Isaksson and Mr. P.Mondrup who have commented on various parts of the manuscript and suggested many improvements. Mr.Isaksson has contributed to the chapters on GIER's structure while Mr.Mondrup,

who together with Mr. L.Hansson has programmed all the auxiliary and the loading routines, has read and criticized the chapters on HELP and SLIP.

Finally, I wish to thank Mr. Alan George Lake who has carefully translated the whole of the manuscript and, at the same time, improved it considerably.

Christian Gram

CONTENTS

		page
PREFACE		
9. THE DRUM STORE AND PERIPHERAL UNITS		1
9.1 Introduction		1
9.2 The drum store		1
9.2.1 Addresses in the core store	2	
9.2.2 The track address; locked tracks	3	
9.2.3 Simultaneous drum transfers; the check on reading	7	
9.2.4 Drumfree	10	
9.3 Paper tape reader, paper tape punch and on-line typewriter		11
9.3.1 The paper tape reader	11	
9.3.2 The paper tape punch	16	
9.3.3 The on-line typewriter	17	
9.3.4 Operation times	20	
9.4 The anelex line printer		20
9.4.1 Margin	24	
9.4.2 Carriage Return, Page Change and Stationary formats	24	
9.4.3 The printer code	25	
9.5 The card reader		26
9.5.1 The mechanical construction	26	
9.5.2 Input to GIER	31	
9.5.3 Examples of punched card input	33	
10. THE CONSOLE AND REGISTERS		38
10.1 The main console (control panel)		38
10.1.1 Lamps and buttons on the main console	38	
10.1.2 Start and Stop button	42	
10.2 Operation of the main console		45
10.2.1 Computer stop	45	
10.2.2 Start (Re-start)	46	
10.2.3 Execution of single instructions	46	
10.3 The HP button etc.		50

		page
11. SLIP (SYMBOLIC LANGUAGE INPUT PROGRAM)		54
11.1 Introduction		54
11.2 An example of simple loader program (deleted)		55
11.3 The loader program SLIP		55
11.3.1 Introduction. Storage allocation	55	
11.3.2 Lines of information	56	
11.4 Loading of Instructions Using SLIP		59
11.4.1 The Basic Operation, Modification and Indicator Instruction	59	
11.4.2 The Address	61	
11.4.3 Increment	71	
11.5 Input of numbers and text using SLIP		74
11.5.1 Input of Numbers	74	
11.5.2 Input of Text	77	
11.6 Control Lines and Blocks in SLIP		78
11.6.1 The Serial Address and Serial Track No.	78	
11.6.2 Program Blocks	80	
11.6.3 Definition Lines and labelled program-lines	85	
11.6.4 Drum blocks	89	
11.6.5 Control Codes	94	
11.6.6 Programs	99	
11.7 Entry to and exit from SLIP		101
11.7.1 Manually-controlled entry	101	
11.7.2 Program-controlled entry	102	
11.7.3 Exit from SLIP	103	
11.8 Messages and Reports output by SLIP		104
11.8.1 Error Messages	104	
11.8.2 Reports	106	
11.9 Syntax for input to SLIP		108
11.9.1 Programs and blocks	108	
11.9.2 Lines	108	
11.9.3 Instruction lines	109	
11.9.4 Addresses and Increments	109	
11.9.5 Constant lines	110	
11.9.6 Control lines	111	
12. OUTPUT		112
12.1 Introduction		112
12.2 Editing of numbers using the sub-routine in HELP		113
12.2.1 Function	113	
12.2.2 Location of the routine; entry and exit	114	
12.2.3 Initialisation parameters	116	
12.2.4 Scale factors	123	
12.2.5 Examples of layouts; special facilities	125	

page

12.3 Printing of text using the sub-routine in HELP 127

12.3.1 Function 127
12.3.2 Location of the routine; entry and exit 128
12.3.3 Example of the use of HELP output-routines 129

13. UTILITY PROGRAMS; THE HELP SYSTEM 132

13.1 A system of utility programs 132

13.2 The HELP administrator 134

13.2.1 The HP button 135
13.2.2 Exit 140
13.2.3 Programmed entry 141
13.2.4 Floating-point overflow 143

13.3 Activation of HELP routines 144

13.3.1 HELP-routine call-line. Control parameters 144
13.3.2 Activation of routine from typewriter or tape 144
13.3.3 Programmed activation of a HELP-routine 146
13.3.4 Corrections 148

13.4 Standard HELP routines 150

13.4.1 Location of HELP routines 150
13.4.2 "kontrol" (check) and "start" 152
13.4.3 "tryk" (output) 156
13.4.4 "kompud" (condensed dump) 165
13.4.5 "gem" (preserve), "hent" (retrieve) and "sam" (compare) 173
13.4.6 "ret" (correction) 177
13.4.7 "hp ind" (patch) 180
13.4.8 "hp ud" (de-patch) 187
13.4.9 "slip" 189
13.4.10 Incorporation of other HELP routines 190
13.4.11 Rules for preparation of HELP routines 191

13.5 Error messages from HELP 193

13.5.1 Wrong control parameters 193
13.5.2 "tomt hp" (not in catalogue) and "sumfejl" (error in check total) 194
13.5.3 Errors in the locked tracks 195

14. LIBRARY ROUTINES (deleted) 196

15. EXERCISES (deleted) 196

16. SUMMARIES AND TABLES 197

16.1 Numerical Representation of the Typographical Symbols 198

16.2 Entries and Layouts when Editing numbers 199

page

16.3 Underlined letters in SLIP 200

16.4 Error messages from HELP and SLIP 201

16.5 The effect of HELP routines and types of parameters 202

PHOTOS of consoles

INDEX

9. THE DRUM STORE AND PERIPHERAL UNITS

9.1 Introduction

The paper tape reader, on-line typewriter and paper tape punch are mentioned in section 2.7 of the Volume 1 of this Manual. These three units will be described here in more detail together with the (Anelex) line printer and the punched card reader, both of which may be connected to the GIER computer.

We will, however, first discuss the drum store in more detail (cf. section 2.3).

9.2 The drum store

As mentioned in section 2.3, GIER is equipped with a magnetic drum store consisting of a number of tracks containing 40 (42-bit) words. Normally, there are 320 tracks but GIER may be equipped with 960 tracks (i.e. corresponding to 3 drums). The tracks are numbered from 0 to 319 (alternatively 959) and all data is transferred to and from the core store in blocks of 40 cells at a time. Thus 2 instructions are required, the one - a VK instruction - indicating the track number and the other -

either an LK instruction or an SK instruction - indicating, firstly, the corresponding part of the core store (40 consecutive cells) where transfer is to take place and, secondly, in which direction data is to be transferred (from drum or to drum).

These two instructions do not necessarily need to be placed next to each other in the program, since the instruction

VK <c>

for instance, serves only to select track no. <c>, i.e. the track is made ready to send or receive the next data transferred. The selection thus made remains effective, (the track no. is held in the drum track register tk), until a further VK instruction is executed. Every LK (read track) or SK (write track) instruction will thus refer to the track which was selected by the last previous VK instruction.

9.2.1 Addresses in the core store

The address in every read or write instruction is always the address of the first of 40 consecutive cells in the core store where the drum transfers are to take place. If this address is between 0 and 984, the 40 cells constitute a compact block in the core store whereas, if the address is $\geq$ 985, the 40 cells are made up of the last and some of the first cells in the core store, the addresses being calculated modulo 1024.

Example 9.1

The piece of coding:

```
VK 70
-
-
SK 1000
-
-
LK 860, VK 0
```

will cause the contents (including marker bits) of cells 1000 - 1023 followed by cells 0 - 15, to be transferred to track 70. After this, the track is read back into cells 860 - 899 of the core store. The contents of cells 1000 - 1023 and 0 - 15 remain, of course, unchanged while their contents have been copied to cells 860 - 899 (and track 70). The final instruction VK 0 has been set there to ensure that GIER does not go further along the program before the second drum transfer is complete (cf. section 9.2.3).

9.2.2 The track address; locked tracks.

The address in a VK instruction is the number of the track which is to be selected: As the VK instruction is executed, the address in this instruction is transferred to the tk register; when an SK or LK instruction is to be executed, the contents of the tk register determines the track to take part in the operation. Thus a VK instruction will be executed even though the address is ≧ 320 (alternatively ≧ 960); only in the following read or write instruction will GIER realize that the track selected does not exist. What happens after this depends on the number of actual drums connected to GIER since the drum circuits are slightly different when one or three drums are connected. Similarly the possibilities for locking drum tracks for 1-drum and 3-drum GIERs are different:

A) GIER with 1 drum, 320 tracks: If the contents of the tk register are $\geqq$ 320, the only consequence of an SK instruction is that GIER will wait until any drum transfer in progress is complete. Apart from this such a "write" instruction is dummy (taking, however, approx. 36 µs, if the address calculation is simple).

An LK instruction when the track no. is $\geqq$ 320 will not cause a stop, either, but does have the effect that access to the drum is prohibited and that the TR lamp will be lit permanently (the sign that the parity check, which is performed when a track is read, has failed and the "track" is being read over and over again). GIER will however come to a halt on the next drum instruction, because the drum circuits are still occupied.

Track 0 is normally locked and can only be opened by turning the power for the computer off and removing a contact pin from one of the printed circuits. Tracks 1 – 31 may be locked and unlocked using a switch on the operating panel at the top of the main GIER cabinet. The effect of "locking" a track is to prevent the contents of that track from being overwritten, in such a way that an SK instruction operating on the track will have no effect (in the same way that an SK instruction to a non-existent track is without effect). A read instruction may naturally operate on a locked track just as well as an unlocked track.

Example 9.2.

On a 1-drum GIER, in the following piece of coding:

```
  [m] VK -1, SK 120
[m+1] LK 400
```

[m+17] VK 20

GIER will not be able to execute the VK instruction in cell m+17, because the reading of "track -1" will never be completed. The TR lamp will be lit but GIER will not be halted, and thus pressing NORMAL START will have no effect; only by pressing RESET can the drum circuits be relieved and after this NORMAL START must be pressed before GIER can execute the instruction VK 20 and continue with the program.

It is also possible to restore the situation by canceling the parity check (using a switch on the operating panel at the top of the main GIER cabinet - a red lamp by the side of the switch lights up when the parity check is switched off), after which the computer will immediately continue with the program. (But the contents of cells 400 - 439 are changed.)

B) <u>GIER with 3 drums (960 tracks)</u>: If the contents of the tk register are $\geq$ 960, both an SK and an LK instruction will cause a stop such that

a) GIER halts on the verge of the next instruction without having performed any transfer.

b) The error lamps TO and TR are lit.

c) The "Klar" lamp is lit.

On pressing NORMAL START, GIER starts running again.

There are, at the foot of the operating panel in GIER's main cabinet, two push-button registers each with 10 buttons which can be set to represent, in binary form, any number between 0 and 1023. It is thereby possible to lock an arbitrary number of consecutive tracks by setting the lower register to represent the address of the first locked track, and the upper register to

represent the address of the last locked track. Track no. 0 is, however, locked in the same way as in GIER with 1 drum.

If both registers are set to the same address, only the track with this address will be locked (apart from track 0), and if the address in the lower register is greater than that in the upper register, then only track 0 will be locked.

A locked track may be read in the usual way, but an SK instruction referring to a locked track will cause GIER to stop just as if the track did not exist (see above).

In short, if the two push-button registers are set to the addresses, Kmin and Kmax respectively, the following instructions will cause GIER to stop (with the effects listed under a), b) and c) above):

LK instructions where $960 \leqq tk \leqq 1023$

SK instructions where $tk = 0$, $960 \leqq tk \leqq 1023$

or $Kmin \leqq tk \leqq Kmax$.

Example 9.3.

In a GIER computer with 3 drums, the effect of the instruction

VK 1, SK 200

depends on the settings of the push-button registers: if the lower register is set to 0 or 1, and the upper register is set to anything else but 0, then track no. 1 will be locked and GIER will halt after "execution" of the SK instruction without having written on any track; otherwise, the instruction will be executed quite normally, the contents of cells 200 – 239 being copied on to track 1 of the drum.

9.2.3 Simultaneous drum transfers; the check on reading.

Transfer of one track, either to or from the drum usually takes 20 milliseconds (the time required for 1 revolution of the drum) but, as soon as transfer has been initiated by an LK or SK instruction, GIER can continue with succeeding instructions in the program at the same time as the drum transfer is taking place. It is only when a further drum instruction - VK, LK or SK - is met that GIER must wait for the previous drum transfer to be completed. The actual SK or LK instruction is executed in 9 μs (+ the time necessary for address modification), but the execution of the succeeding instructions is delayed slightly because the control unit is occupied for short periods during the drum transfer, altogether approx. 1 millisecond spread over the 20 milliseconds which the transfer lasts.

Since the drum transfers commence as soon as GIER meets an LK or SK instruction, it depends on the instantaneous position of the constantly rotating drum which cell on the drum track is the first to send or receive information; GIER must thus calculate the corresponding address in the core store (within the block of 40 cells defined by the address of the LK/SK instruction). After this the transfer continues between the drum and consecutive cells in the block in the core store until the cycle is complete; it will thus occur quite often that the 40 cells are not dealt with from first to last but in a cyclically displaced order.

This matter is further complicated by the fact that an automatic parity check is performed when the drum is read: each cell on the drum has an extra bit which, when the cell in question is

written into, is set equal to 1 or 0 according to whether the number of ones in the cell is odd or even.*) When a cell is read, GIER determines whether the parity check is valid or not and, if it is not, GIER continues to read the track (cyclically) until it has read 40 cells in succession without error.

When the parity check fails, the TR lamp on the console is lit but it is turned off if the error disappears. If there is a persistent error on reading, one will see the lamp lit up.

A consequence of this is that, while the execution of an SK instruction always last 20 milliseconds, an LK instruction may occupy the drum circuits for a longer time if parity errors occur. Before one uses data which has been transferred from the drum to the core store, one should ensure that the transfer is completely finished. This can be achieved by writing in the program a (redundant) VK instruction with an arbitrary address, since the selection of a fresh track can not be performed before the drum circuits are vacant.

One should also do the same before storing fresh data into a block in the core store the contents of which have just been written on to the drum.

In the case of drum errors when the TR lamp remains lit, GIER can be stopped by pressing the RESET button or, alternatively, the track may be caused to be read uncritically by using the button "Annullering af paritetscheck" on the operating panel at the top of the main cabinet.

---

*) In a 3-drum GIER, each cell has 3 extra bits and the parity check is made modulo 8 instead of modulo 2.

Example 9.4.

In the piece of program:

```
VK 210, LK 400
----
VK 0, ARSF 403
----
```

the instruction VK 0 ensures that all reading from the drum is complete, before the contents of cell 403 are used in the following calculations. If, however, one were to write

```
VK 210, LK 400
----
ARSF 403
----
```

it would be impossible to predict whether it was the old or the new contents of cell 403 that were used in the following calculations (and this may change from run to run).

Example 9.5.

Let us consider the piece of program:

```
VK 310, SK 200
----
GM 200
----
```

If one can calculate that it takes GIER more than 20 milliseconds to run through the series of instructions between SK 200 and GM 200 then this last instruction will have no effect on the contents of track 310 on the drum. But if it takes less than 20 milliseconds, it is impossible to predict whether the quantity written on to the first cell of track 310, is the original contents of cell 200 or the contents of the M register (and this may too be different from run to run).

One should, to be quite certain, insert a VK instruction, e.g.,

```
VK 310, SK 200
----
----
VK 0, GM 200
----
```

whereby one can ensure that GM 200 is first executed after the complete drum transfer.

9.2.4 Drumfree jump.

As mentioned above the instructions, VK, SK and LK serve, among other things as "wait" instructions, which ensure that GIER does not continue with the program before a drum transfer, previously initiated, is complete. The purpose of doing this is usually that during a calculation which takes place simultaneously with a drum transfer, one needs to ensure that the drum transfer is complete before the program uses a part of the core store taking part in the transfer.

However there is another facility which can be used to let GIER itself optimise the exploitation of "slack time" during a drum transfer. This is the HK instruction ("drum free jump", described in Volume 1 of this manual page 87).

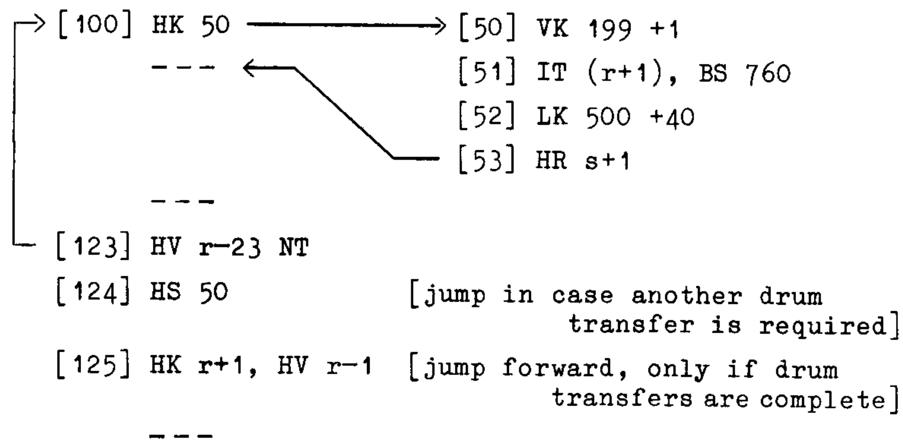
The effect of an HK instruction depends on whether a drum transfer is in progress or not: if this is so, the HK instruction is dummy (apart from a possible S-modification), and GIER continues immediately with the following instruction; if however the drum is free, the HK instruction works exactly like the subroutine jump i.e. the instruction HS.

This jump can be used, for instance, in the following situation: a program loop is to be run through many times and lasts a considerable time, i.e., 0.1 seconds or more, while one wishes to transfer the contents of another part of the core store to the drum at the same time. In such a case one may insert an HK instruction in the main loop of the program, so that when the

drum is free, a jump will be made to a few instructions which initiate the "next" drum transfer and thereafter jump back to the main loop of the program (see the example below).

Example 9.6.

A program loop in cells 101 – 123 are to be run through a large number of times, and the 7 drum tracks 200 – 206 are to be read to the core store, cells 540 – 819, at the same time. Assuming that cells 50 – 53 are free, the program could be as follows:



An HK 50 instruction has been inserted in the main program loop. This instruction causes a subroutine jump to cell 50, every time a drum transfer is complete. The instructions in cells 50 and 52 serve to initiate fresh drum transfers while the BS instruction in cell 51 makes sure that only 7 tracks are read in. However, one must also make certain that all the tracks are transferred by the time the computer has finished looping, and, therefore, in cells 124 – 125, 3 instructions have been put in to ensure that GIER does no continue with the next part of the program before the last track has been read into the core store: only when the drum has completely finished transferring will the jump effect of HK r+1 take place.

The HK instruction is rather treacherous in that the above program may not necessarily work properly when running

at micro tempo, as every drum transfer will undoubtedly be complete before the next instruction is executed and the HK instruction in cell 125 will thus cause a jump the first time it is executed, even though several tracks has not been read in at this time.

9.3 Paper tape reader, paper tape punch and on-line typewriter.

GIER is equipped with an 8-channel paper tape reader and paper tape punch and also an on-line typewriter for both input and output. This section will only deal with the way in which these units work whereas the construction of input/output programs is described in later chapters.

Both input and output take place one frame of holes (or one character) at a time, as an LY instruction causes input, an SY instruction output, of only one frame of holes (one character).

These peripheral units are connected to GIER via the buffer register bl consisting of 10 bits, of which only the last 8 (pos.2-9) are used here. In addition, the activation of the different peripheral units is controlled by the by register; selection of the units to be activated by LY/SY instructions is performed using a VY instruction which can change the contents of the by register. Positions 3-6 of this register cover the selection of output units and positions 7-9 cover the selection of input units.

9.3.1 The paper tape reader.

GIER is equipped with either the Facit reader or the RC 2000 reader, reading about 1000 and 2000 characters/second respective-

ly. In the sequel we first describe the Facit reader in detail, and then mention the points where the RC 2000 reader differs from the other one.

A) <u>The Facit tape reader</u>: After the tape has been placed in the reader, pressing a button on the reader will cause one frame of holes to be read into the bl register, but no transfer to GIER itself will take place.

(A frame with only the little sprocket-hole, Blank tape, is ignored by the reader which searches for the first frame with a hole in one of the 8 meaningful channels; the sprocket-hole in fact only serves to define the presence of a frame).

After this the effect of an LY instruction (assuming that the paper tape reader has been selected) is that the contents of pos.3-9 in the bl register are transferred to pos.3-9 in the R register (accumulator) and to pos.3-9 in the cell indicated by the final address of the LY instruction; a character is then read in from the tape to the bl register, so that the parity sign (the 4th channel from the top) is put into pos.2 of the bl register while the remaining 7 channels are stored in pos.3-9 of the bl register.

The parity bit is <u>not</u> transferred to the accumulator, neither is it transferred to the cell in the store but a parity check is performed together with the transfer from the bl register to the accumulator and the cell: if the number of ones in the bl register (i.e. the number of holes in a frame) is <u>even</u>, GIER will stop after the LY instruction - as with the standard halt instruction - with the <u>corrupt character</u> minus parity bit in the accumulator and cell and with the <u>next</u> character in the bl register. At the same time the L lamp on the console is lit

(also "str.læs. par.fejl" (tape reader parity error) on the auxiliary panel); the two lamps will first be switched off when the next LY instruction is executed or when the button on the reader is pressed (reading the next character to bl).

An exception to the above is the treatment of a frame with holes in all 8 channels (All Holes): in spite of the fact that the number of holes is even, GIER will not stop; pos.2-9 of the bl register will be filled with ones, and pos.3-9 of the accumulator (and the cell in the store) will also be filled with ones corresponding to the value 127.

From a comparison of the tables 8.3 and 8.4 in Volume 1 (pages 152-153) one can see that if one ignores the parity channel (the 4th. from the left), the holes in the tape correspond exactly to the ones in the binary representations of the values in GIER. This principle is extended in fact also to the punching codes which can not be punched directly on the Flexowriter; for instance, the code |oo oo. o| is accepted by GIER and makes the value 109 in Raddr and in the address positions of the appropriate cell of the store.

<u>Example 9.7.</u>

Since GIER first stops after parity error when the corrupt symbol has been placed in Raddr and in the address position of a cell, any tape input program to be fully satisfactory, ought to deal with the contents of the accumulator ignoring completely the contents of the cell of the store involved in the LY instruction. The reason for this is that when

GIER is halted after a parity error while reading a tape, it is easy enough to correct the contents of Raddr by means of push buttons on the console - on pressing NORMAL START the input program will deal with the corrected symbol and thereafter continue reading - whereas it is much more difficult to correct the contents of a cell as well (operation of the console is described in detail in chapter 10).

B) The RC 2000 tape reader: This photoelectric reader has a buffer store with a capacity of 256 characters. This store is used quite literally to "buffer" the incoming characters from the paper tape so that data can be read from the tape at 2000 characters/sec. without fear of "losing" any characters if a particular program has a processing cycle of less than 2000 characters/sec. The buffer is used as follows: As characters are read from the tape, they are stored cyclically in the buffer. Similarly, as LY instructions are executed, characters are taken cyclically from the buffer beginning with the "oldest" character. During run the tape speed is controlled so that the buffer is always about half full: When only 1/4 or less of the buffer is occupied the tape is read with full speed, but when the buffer becomes fuller the speed is slowed down and the driving motor is stopped when about ½ of the buffer is occupied.

When a tape is placed in the reader (and the reading head is pressed down), use of the button RESET causes the buffer to be cleared and 100-200 characters to be read into the buffer ready for transfer to GIER.

LY instructions now cause the transfer of the first characters from the buffer to R and to a cell (via the bl register), and when less than 128 characters are left in the buffer, the reader starts again. So, when the last input instruction of a program is performed, the reader has usually read 120 - 200 characters too many.

The effect of the other three buttons on the reader is as follows:

READ causes reading into the buffer without clearing it.

SKIP causes the tape to be moved forward without reading it. This is usually the quickest way to get a tape out of the reader.

UP lifts the reading head.

9.3.2 The paper tape punch.

The effect of an SY instruction with the final address c has the effect (assuming that the punch has been selected) that a frame of holes is punched, the pattern of holes being an image of the 7 last bits in the binary representation of the address c; in addition a parity hole may be punched to make the number of holes in the frame always odd. At the same time, the 7 bits are transferred to pos.3-9 of the bs register, the parity bit being placed in pos.2 of this register. Also, the value of the 7 bits - as an integer - is added to the contents of cell 1023 in pos.0-19; this may be used to form a check total of all punched symbols.

Even those values of c which do not correspond to Flexowriter symbols will cause a frame of holes to be punched.

In these cases also the last 7 bits in the binary representation of c will be punched with a hole for each one (plus a possible parity hole).

Example 9.8.

The instruction SY 127 causes the code |ooo o.ooo|; the "All Holes" code can not be produced by GIER.

The instruction SY 128 causes a parity hole alone (the SPACE symbol) to be punched; in the bs register, pos.2 is set equal to one while the remaining positions are set to zero.

9.3.3 The on-line typewriter.

The typewriter attached to GIER can be used for both input and output; its function will be described, firstly, as an input medium, and secondly, as an output medium; finally, some points which require careful attention when using the typewriter alternately to input and output, are given.

A) Input: When GIER arrives at an LY instruction while the typewriter is selected, a green lamp on the typewriter lights up; GIER then waits in the process of executing the LY instruction until a symbol is typed on the typewriter. The numerical value of this symbol (cf. Table 16.1 in this volume) is transferred to pos.3-9 of the accumulator and of the cell indicated by address part of the LY instruction; after this GIER continues with the succeeding instructions and the green lamp is turned off. Note that in this case there is no buffer register acting as a go-between, but that the symbol is read directly from the typewriter to the accumulator and cell.

The typewriter can not be locked in Upper Case (unlike ordinary typewriters and Flexowriters), and if several characters are to be read in in Upper Case (using a number of LY instructions), the Upper Case key must be held down while each character is typed; it is true that the value of the character is the same in Upper or Lower Case but when the Upper Case key is released, a Lower Case symbol (value 58) is read in and may have a quite undesirable effect on the interpretation of the other symbols by the program.

Example 9.9.

It can be seen from table 16.1 that when using the typewriter as input medium it is not possible to put the values 10, 11, 12, 15, 26, 28, 31, 42, 44, 45, 46, 47, 61 and 63 into Raddr as these correspond to symbols which are not on the typewriter; apart from the above values, however, it is possible to introduce all other values between 0 and 64 incl.

B) Output: When GIER arrives at an SY instruction, the last 7 bits of the address are transferred to pos.3-9 of the bs register while the parity bit is set in pos.2 of bs; if the typewriter has been selected, it types the character corresponding to the contents of bs i.e. the address in the SY instruction modulo 128. At the same time a check total of the characters output is formed in cell 1023; see section 9.3.2 above.

If the address (modulo 128) of the SY instruction does not correspond to any symbol on the typewriter, the effect of the instruction is as follows: the address is transferred to the bs register as described above, and GIER continues with the next instruction without any visible activation of the typewriter.

(The next SY instruction must wait however for approx. 0.1 second until the typewriter has sent a "ready" signal to GIER).

Lower Case is selected by means of the instruction SY 58, and all SY instructions that follow will cause typing in Lower Case until the instruction SY 60 causes all the SY instructions that follow this to type in Upper Case (until the next SY instruction with an address part = 58). Thus by using these 2 instructions it is possible to "lock" the typewriter in the relevant Case.

Example 9.10.

The instructions

```
VY 16 t7    [output to typewriter, input unchanged]
SY 58, SY 32
SY 11, SY 84
SY 129, SY 26
```

cause the typewriter to write (in Lower Case) a minus sign and a 1 (SY 129); the remainder of the SY instructions are dummy.

C) Points for careful attention: If the typewriter is used for output immediately after it has been used for input, the symbol (last) read should not be

a) Upper Case or a character in Upper Case

b) Car.Ret., Tab or Space.

The reason for this is that during input GIER does not wait for these typographical operations to be completed; it continues with the program as soon as the appropriate value has been introduced into Raddr (and a cell). An output instruction may therefore make a conflicting situation for the typewriter if it

has not managed to finish the typographical operation activitated by the last character read. It may well happen that the output instruction has no effect and that GIER will continue in the program.

9.3.4 Operation times.

A single LY instruction will take up GIER's time for approx. 50 μs (including simple address modification), this being the time necessary for the transfer from the input buffer to the accumulator and cell; after this GIER continues with the next instructions in the program, but a fresh LY instruction can only be executed, when the appropriate peripheral unit is ready again (in the case of the paper tape reader after 1-2 milliseconds and for the typewriter after 0.1 seconds).

Correspondingly an SY instruction takes up GIER's time for as little as 60 μs, whereas the punch is active for approx. 8 milliseconds and the typewriter for 0.1 seconds.

9.4 The Anelex line printer

GIER may be equipped with an Anelex line printer, which can print output at the rate of max. 1000 lines/min., the maximum no. of characters in each line being 119 (in some cases even 120).

The printer is connected to GIER via 2 "parallel" buffer registers with room for 120 characters, corresponding to one line of print, in each register; while information is being printed from one register, the other register collects the char-

racters coming from GIER. When the printer has finished printing from the one of the registers and the other register has received the symbol for CR, the instruction SY 64, the two registers exchange rolls: after the paper has been spaced forward to the next line, the latter register which has just been filled is printed out while the former register collects the characters output from GIER, and so on.

The line printer can be activated by SY instructions otherwise destined for the typewriter or punch, or simultaneously with these, or alone by means of the by register together with a special set of buttons on the auxiliary console containing the HP button. There are 16 buttons, which are arranged 4 by 4,

	by[3]	by[4]	by[5]	by[6]
Line printer	☐	☐	☐	☐
Typewriter	☐	☐	☐	☐
Punch	☐	☐	☐	☐
Reserve	☐	☐	☐	☐

each of the 4 columns indicating a bit in the by register and each of the 4 rows indicating an output unit.

If <u>no</u> buttons are depressed, all SY instructions are output to a non-existent unit i.e. GIER goes through all the motions of the SY instruction without activating any unit.

If any button is depressed, an SY instruction will cause output to the unit corresponding to the row in which the button lies, if the bit of the by register, corresponding to the column of the button in question, is one. Any number of buttons may be

depressed at the same time; more than one button in the same row will thus mean that several different bits in the by register will activate the same output unit, and if more than one button in one column is depressed one bit in the by register will activate a number of output units and thereby cause simultaneous output to several units.

The introduction of these push-buttons nullifies the fixed effect that by[4] and by[5] have on the standard GIER without printer; however, it is wise to keep a standard for ease of operating etc. e.g.

by[4] = 1 activates punch

by[5] = 1 activates typewriter

by[6] = 1 activates lineprinter

This standard should always be chosen when using the HELP routines.

Example 9.11.

A program commences with the instruction

VY 16 +7

which set by[5] equal to 1 without changing the input selection. While the program is being tested all output is to be written on the typewriter in which case the button marked

	by[3]	by[4]	by[5]	by[6]
Line printer	☐	☐	▨	☐
Typewriter	☐	☐	☒	☐
Punch	☐	☐	▨	☐
Reserve	☐	☐	☐	☐

with a cross is depressed; for routine runs of the program all output is to be made both on the line printer and the tape punch and in this case the two shaded buttons are depressed.

Example 9.12.

In a program 3 kinds of output are required: Check output (only during testing), punched output (to be used as data for later runs), and printed output (which is in fact the object of the run in question). The program is written with the instruction

VY 32 +7 [set by 4 = 1]

before any check output, and the instruction

VY 16 +7 [set by 5 = 1]

before punched output, and the instruction

VY 8 +7 [set by 6 = 1]

before printed output.

During a program test the 3 buttons marked with crosses are depressed whereby the check output is written by the typewriter and the remainder is printed; during routine runs the 2 shaded buttons only are depressed:

	by[3]	by[4]	by[5]	by[6]
Line printer	☐	☐	☒	☒▨
Typewriter	☐	☒	☐	☐
Punch	☐	☐	▨	☐
Reserve	☐	☐	☐	☐

The same effect could in fact be achieved using many other combinations of the VY instruction and the push-buttons.

9.4.1 Margin.

For output on the printer the margin must be set manually by means of a control panel on the printer.

The margin may set to any of the first 63 left-hand printing positions.

9.4.2 Carriage Return, Page Change and Stationery formats.

As mentioned above the instruction SY 64 causes one line to be printed and vertical spacing of the paper. The printer has also a built-in control of whether a line is filled up or not: if one of the printer's buffer register has received 119 characters without a CR among them, the whole line is printed when the 120th character is received, and the paper is spaced vertically (if the 120th character is the CR symbol, the paper will be vertically spaced 2 times).

The printer is normally adjusted to change to a new "page" of continuous stationery (page change), when one "page" is filled. In the case of the 3 standard stationery formats, a "page" is defined as follows:

A4 vertical: Automatic Page Change after 72 lines (81 printing positions/line)

A4 on side: - - - - 48 lines (111 printing positions/line)

Large Size: - - - - 102 lines (120 printing positions/line)

Besides this the printer will make a page change on the instruction SY 42 (42 is one of the unused values on the typewriter and flexowriter). Note particularly that if one attempts to use more than the allotted number of printing positions (81 for A4

vertical, 111 for A4 on side) the last characters in the line will not be printed on the paper and no indication will be given by the printer that the characters have been lost.

9.4.3 The printer code.

The printer has a wiring panel, by which it is possible to express an arbitrarily required correspondance between the values in the address in an SY instruction and the symbols on the print barrel. The symbols on the print barrel are as follows:

the digits 0 to 9, the capital letters A - Z and the Danish letters Æ, Ø, Å

+ − × / ↑ ( ) [ ]

= ≠ > < . , : ;

* % ' & α £ $ ₁₀

The wiring panel is normally wired to a standard code which resemples very closely the codes for the typewriter and Flexowriter; there are however the following small divergences:

Letters are always printed as capital letters.

SY with the addresses 10, 12, 28, 45, 46, 47 and 61 produces a space instead of nothing.

SY with the addresses 29, 62 and 63 has no effect on the printer.

SY with the addresses 1, 14, 15, 16, 26, 31, 42 and 44 causes symbols differing from those on the typewriter and Flexowriter to be printed.

For more detailed information, the reader should refer to table 8.4 in Volume I, page 153 or to the table 16.1 in the Appendix of this Volume.

9.5 The card reader.

GIER can be equipped with a card reader which can sense both punched and pencil-marked cards. The unit can also be used as a sorter, although only a primitive form of sorting (selection of a particular class of cards) can be controlled by GIER itself; more complicated forms of sorting can be performed using a plugboard and a criteria card which is read in prior to the sorting operation. The sorting of cards in this way may be accomplished independent of any connection to GIER and the unit can thus be used as an off-line sorter.

The maximum rate of reading is approx. 12 cards/sec. giving a cycle time of 80 milliseconds per card. If the reader is running at this speed, GIER must be able to process each card within 80 milliseconds and must furthermore be able to read the desired information fields in each card within 72 milliseconds (cf. section 9.4.2). If this is not possible, the information in the next card will not be available to GIER at all, and the card will be deposited in a special stacker indicating that it has not been processed.

The continuous feed-rate may however be decreased to as little as 1.5 cards/sec. giving GIER up to 665 milliseconds in which to process a card (the reading itself must not take longer than 635 milliseconds).

9.5.1 The mechanical construction.

Cards: The cards used are divided into 80 columns, numbered from 0 to 79, each with 12 rows, for punched information, and 27 fields, numbered from 80 to 106, also with 12 rows per column,

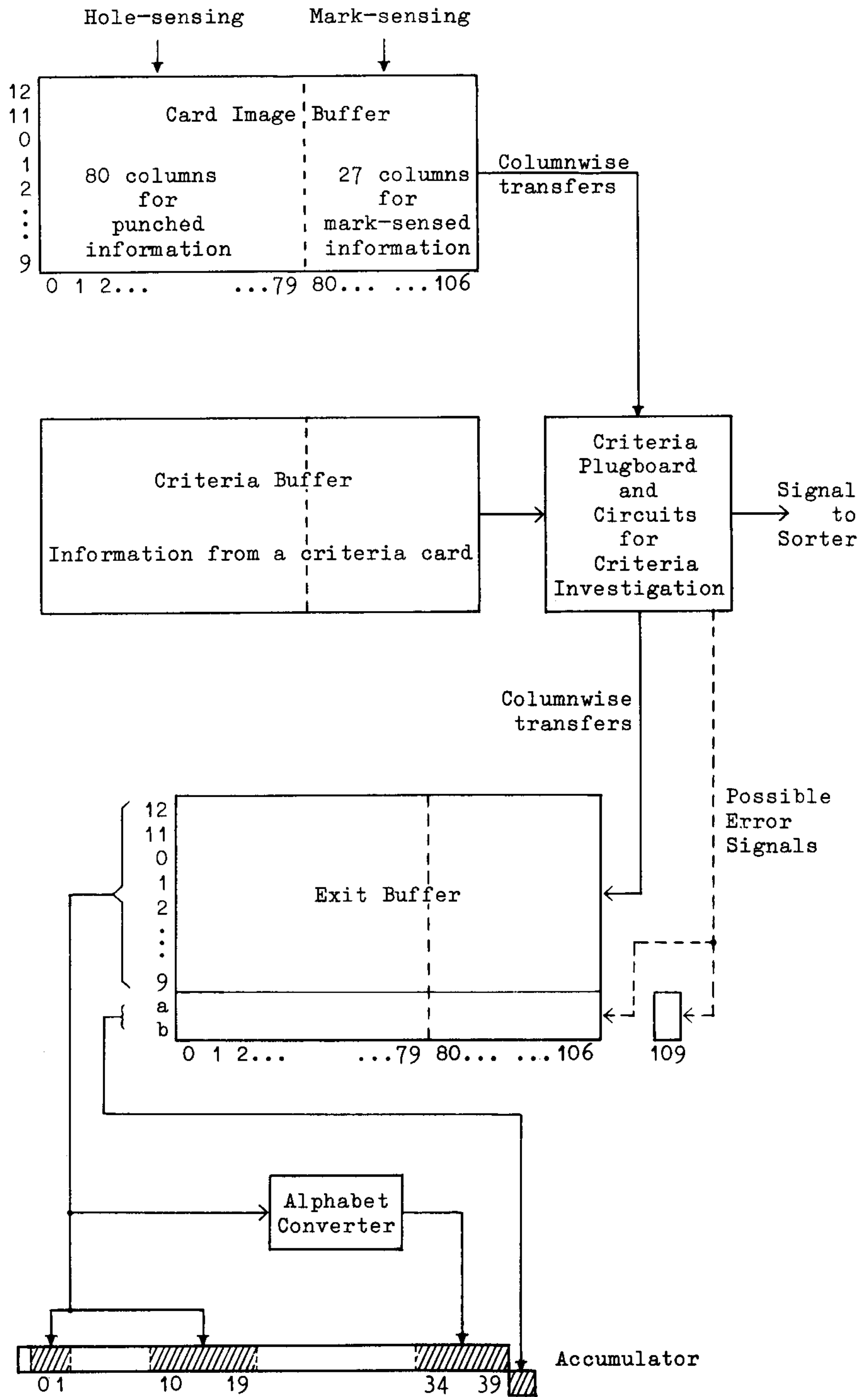
for pencil-marked information. The 12 rows are identified from top to bottom by the numbers 12 - 11 - 0 - 1 - 2 - - - 8 - 9, rows 12 and 11 being the so-called rows for zone punching; the 12 rows for pencil marking correspond to the rows for punching and are numbered in the same way. Each of the fields for pencil marking is superimposed upon 3 punching columns so that, for instance, field no.80 occupies the same part of the card as column 0 - 2. The last field, no.106 covers columns 78 and 79 and a dummy column to the right. The fields for marking can, however, quite conveniently be regarded as columns for programming purposes and will be referred to as such in the remainder of this section.

The reader: The reading unit includes 2 set of brushes, the first set being used to sense pencil-marks (3 brushes per column) while the second set is used to sense holes in the card. The card is read row by row to a buffer store, the Card Image Buffer (CIB), which has room for 80 + 27 columns; when a card has passed both sets of brushes, the CIB contains a complete picture of the card (where marks and holes are shown as ones, the rest being zero). This card image is transferred to another buffer store, the Exit Buffer and the next card is then read into the Card Image Buffer*). When the complete image has been transferred to the Exit Buffer, the information in this buffer is available while the next card is being read into the Card Image Buffer.

While the card image is being transferred, column by column from the one buffer to the other, it is subjected to a check,

---

*) In reality the Card Image Buffer has 27 extra columns into which reading of the pencil marks of the next card is commenced at the same time as the punched columns of the previous card are read.



Hole-sensing
Mark-sensing
12
11
0
1
2
9
Card Image Buffer
80 columns for punched information
27 columns for mark-sensed information
0 1 2... ...79 80... ...106
Columnwise transfers
Criteria Buffer
Information from a criteria card
Criteria Plugboard and Circuits for Criteria Investigation
Signal to Sorter
Columnwise transfers
Possible Error Signals
12
11
0
1
2
9
a
b
Exit Buffer
0 1 2... ...79 80... ...106
109
Alphabet Converter
Accumulator
0 1
10
19
34
39

which can lead to a possible error marking, as follows:

Certain criteria for the punching and marking of cards can be selected prior to a run. According to these criteria, a criteria plugboard must be wired and a special criteria card must be read into a store called the Criteria Buffer.

Each column of the image read in can now be checked using the selected criteria: if the column is not in error, it is sent on to the Exit Buffer; if there is an error in a column, one of two error bits are sent from the criteria plugboard, and at the same time as the column is transferred to the Exit Buffer, an error bit for that column may also be transferred to one of 2 extra rows available for this purpose in the Exit Buffer. It is also possible to "hold" the error bit at the plugboard for the duration of the transfers of columns so that the logical sum of all error bits can be sent to an extra column consisting only of the 2 error positions, column 109 in the Exit Buffer. It is thus possible to indicate the types of error for each column.

For further details about the above and also the possibilities of making error-marking dependent on as many as 6 columns of information, the reader should consult a more technical description of the reader, e.g. A Fast Reader for the GIER Computer by L.Prøhl Hansen and B.Scharøe Petersen, BIT Vol.3 no.1, obtainable as a reprint.

Stacking: When the cards have passed the two sets of brushes, they are lead past 14 stackers or compartments, called 9 - 8 - ... - 1 - 0 - 11 - 12 - S - R, where the reject stacker R is furthest away from the reading brushes.

Using a plugboard and a criteria card one may choose between different forms of stacker-selection.

a) <u>Consecutive stacking</u>: The stackers 9-0 are used for normal unselected cards which have been run through the reader; stacking commences in stacker no.9 and when this has been filled up, stacking is continued in stacker no.8 and so on. When all the stackers 9-0 are filled up, the reader will stop but if one of the stackers is emptied, stacking will take place in this stacker and will continue until all the stackers are full again.

Stacking in compartments 11, 12 and S is directed by the plugboard wired for logical criteria and these compartments will thus be used for stacking error cards or cards selected by any other form of criteria.

The reject compartment R is used for stacking cards which GIER is not able to process.

b) <u>Simple selection controlled by GIER</u>: GIER may control the stacking of selected cards in compartment no.8 as follows: The reader deposits cards in consecutive stackers as above, but only in compartment nos. 7-0; the special stackers 11, 12 and S and the reject stacker R are used as above. However a card will be deposited in compartment 8 if the <u>instruction LY 120</u> is executed by GIER before the next card arrives. Compartment no.9 is not used at all with this form of sorting. The instruction LY 120 must be executed before 9/10ths of the card cycle (at top speed 72 milliseconds) has elapsed, and before the instruction LY 127 has been given, meaning that GIER is ready for the next card.

c) <u>Off-line sorting</u>: Any other more complicated form of sorting can only be achieved by manual operation of the sorter, with selection controlled by the plugboard and a criteria card.

9.5.2 Input to GIER.

Information on a card is read into GIER column by column, from the Exit Buffer on the reader to the accumulator R, using the LY instruction.

This requires that the card reader is selected as peripheral unit by placing the bit pattern 010 in the by register pos.7-9 (cf. the list of operations in Volume I, page 86); this may be done with the instruction

VY 18 [input from card, output to typewriter]

or with

BY 2 +120 [input from card, output unchanged]

After this an LY instruction will cause a column to be read to the accumulator, as follows:

rows 12,11 are read to R pos.0,1

rows 0-9 are read to R pos.10-19

error bits are read to R's marker-bits, positions 40-41.

At the same time, the column's bit pattern (rows 12,11 and 0-9) is converted to a 6-bit number which is placed in R pos.34-39. Using a plugboard it is possible to convert an arbitrary punched card code to any 6-bit code. It is only possible to read a column once as reading corrupts the corresponding column in the Exit Buffer.

The final address c in an LY instruction indicates, mainly, which column is to be read from the Exit Buffer to R:

$0 \leqq c \leqq 79$ or $80 \leqq c \leqq 106$	Card column c is read to R pos.0,1 and 10-19; the same column is read via the alphabet converter to R pos.34-39; the error bits are read to the marking pos.40-41 in R. The remainder of R is set to zero.

c = 109 The two error bits styled as column 109, are read to the marking positions 40-41 in R whilst the remainder of R is set to zero.

c = 120 This value of c is only used when sorting is controlled by GIER: the card last read is deposited in stacker no.8 (while all other cards are deposited in stackers nos. 7-0, 11, 12, S or R).

c = 127 This value of c is a signal from GIER to the reader that reading is completed: the next time the Card Image Buffer is filled up, its contents are transferred to the Exit Buffer; only when this has happened, is the reader ready to transmit information to GIER which cannot otherwise execute the next LY instruction.

If the reader does not receive the "ready" signal (via LY 127) from GIER, before 9/10ths of the card cycle has elapsed, the image of the next card will not be transferred to the Exit Buffer; in addition the card itself will be deposited in the reject stacker indicating that GIER has not been able to process it.

c = 107, 108, $110 \leqq c \leqq 119$ or $121 \leqq c \leqq 126$. These values of c are invalid addresses; their use will cause the reader to stop (because a parity check, performed as each column is read by GIER, does not tally) and GIER will wait at the next LY instruction because the reader is not ready to transmit to GIER.

$c \geqq 128$ In this case the value of c will be taken modulo 128, thus having one of effects described above.

9.5.3 Examples of punched card input.

A program for input and processing of information on cards will, frequently, have the following structure:

1) An introductory part (initialisation of addresses etc.) including a VY instruction, selecting the card reader as peripheral unit, plus the instruction LY 127, meaning "ready to read the first card".

2) The head of a loop, which is run through once for each card; the speed of the reader must be selected so that the time taken to read a card is not less than the time required for one loop.

3) One or more loops within the main loop. Each loop commences with an instruction of the type LYS a t+1, which reads the "next" column to an empty accumulator; each loop will thus read in and process a field from the same card.

4) The tail of the main loop; this part consists of the instruction LY 127, a possible final and collective processing of the card just read and a return jump. The reason for arranging things in this way is that, as soon as the last column required has been read, it is desirable to send the "ready" signal LY 127: 1/10th of the card cycle must always elapse after this before a column from the next card can be read; one should thus attempt to make use of this time internal for the final processing of the card.

The actual time required to read one column is approx. 50

μsec. and if the address part is well-bestowed with brackets and increment it could take up to 75 μsec.

Example 9.13.

500 cards are punched with positive integers, with 3 digits having units in column 10 (tens in col.9 and hundreds in col.8). These integers are to be read in and totalled, the result being placed in cell 1022, as an integer.

The alphabet converter is wired so that the values of the digits 0-9 is formed in the usual binary manner in the last positions in the accumulator; the program could be as follows (using the SLIP notation with symbolic addresses etc.):

```
        VY 2 t 120          [select card reader]
        PM 10 D             [set factor 10 x 2^-9 in M]
        LY 127, GRS 1022    [ready for 1st card, clear total cell]
        PA a3 t 499
┌──→a5: PA a0 t 7           [the main loop begins here]
│       GRS a2              [clear work cell a2]
│ ┌→a0: LY 7 t1             [sub-loop: read next col.]
│ │     TL -6, TLS 6        [clear first 34 bits of R]
│ │     GR a1, MKS a2       [store digit from this col, mult. last
│ │                                    value of scaled no. by 10]
│ │     AR a1, TK 10        [add digit from this col., scale by
│ │                                           factor 10 x 2^-9]
│ │     GR a2, IT (a0)      [store scaled no.]
│ └──── BS 10, HV a0        [end of sub-loop]
│       LY 127              [ready for next card]
│       ARS a2, TK -10      [put no. in R pos.39]
│       AC 1022, IT -1      [total in cell 1022]
└────── BT 499, HV a5       [the main loop ends here]
        ZQ 0
    a1: QQ 0                [work cell: scaled no., units in
                                                       pos.29]
    a2: QQ 0                [work cell: last char. read in]
```

It takes less than 2.5 milliseconds to run through the main loop (from cell [a5] onwards), which means that each time GIER has executed the instruction LY 127, it will wait at the next LY instruction for some time (depending on the speed of the reader) before processing the next card.

Example 9.14.

In a primitive service routine a print-out of the image of the pencil-marks on cards is required: the image is to consist of ones where marks are made and noughts in all other positions; every row in the column is to be investigated and the image of each column is to be printed on one line beginning with column 106 working towards column 80:

```
b0: VY 34, LY 127    [select card reader and tape punch,
                                  ready for next card]
    PA b1 t 107      [initialising]
b2: SY 64, SY 64<-┐
b1: LY 107 t -1   │  [read next column]
    PA b6 t 8     │  [initialising]
┌->b4: GA b3, SR b3 │  [clear pos.0-9 in R]
│   TK 1, CA 0    │  [investigate next bit in column]
│   SY 16, HH b5  │  [write either 0]
│b5: SY 1, IT -1  │  [or 1]
└-b6: BT 8, HV b4 │  [jump back 9 times]
    BS (b1) t 80  │
    HH b2 ────────┘  [jump back to next column]
    HV b0            [jump to next card]
b3: 0
```

When this program runs, the contents of the first card will be punched as required, but since it takes approximately 2 seconds on the tape punch (280 characters must be punched, it is very possible that several cards will have run through the reader (the number being dependent on the speed of the reader), before GIER is ready to accept another card; the card which happens to be passing the reading brushes at that

time will then be printed out; the preceding cards will not be printed out but deposited in the reject stacker.

If, however, the line printer is used for the print-out, the input-output times will be compatible and all the cards will be printed out and deposited in the normal stacker.

Example 9.15.

A number of cards are to be sorted from a deck of cards depending on the punching in columns 35 and 36, the criteria for selection is of such a nature that it cannot be wired on the criteria plugboard but must be programmed in GIER. The selection instruction LY 120 can be used, for instance, as follows:

```
         VY 2 t 120      [select card reader]
   ->a0: LY 127, LY 35   [ready for next card, read col.35]
  |      GR 0 D          [store the column just read in pos.0-9
  |                                              of this cell]
  |      LY 36
  |      GR 0 D          [store column 36 in pos.0-9 of this
  |                                                     cell]
  |      ----            [the criteria for selection is calcu-
  |      ----             lated and reduced to be indicated by
  |      ----                                    the sign of R]
  |      LY 120 LT       [the card just read is deposited in
  |                                        stacker no.8 if R < 0]
   ----- HV a0           [jump to read next card]
```

Assuming that the speed of the reader can be adjusted so that the above loop can be run through in less than 9/10ths of the card cycle, the selected card will be deposited in stacker no.8, while the remaining cards will be deposited successively in stacker nos.7-0.

If, however, the loop lasts longer, the first card will be deposited in stacker no.7 irrespective of sorting criteria (because the selection instruction comes too late); the next card will be deposited in the reject stacker (because

it has not been processed); the third card will be read but will always be deposited in stacker no.7, the fourth will always be rejected etc.

10. THE CONSOLE AND REGISTERS

10.1 The main console (control panel).

Although the main console has, with the development of the HELP system, become more or less redundant for the general user, the meaning and effect of the various lamps and buttons on the console will be described here. The auxiliary console which contains the HP button will be described in section 10.3.

In the bottom left-hand corner of the console panel there is a key-hole into which a key is inserted to turn the power for GIER on or off. The volume control for a loudspeaker which is connected to R00 and makes noise whenever R00 changes is situated in the middle of the lower half of the panel.

10.1.1 Lamps and buttons on the main console.

The photograph at the end of this volume shows GIER's main console. The left-hand part of the upper row of lamps shows the contents of the indicator register etc.; the centre part of this row shows the current situation (mainly for the benefit of service engineers) and the right-hand end shows which register the operator last displayed (using push-buttons). The second row of

lamps shows, when the computer is stopped the current contents of the selected registers, the digits 1 and 0 being represented by, respectively, a lit and unlit lamp. If the operator selects a 10-bit register (the p register, for instance) it is shown in the left-hand end of the row of lamps, in the field marked ADRESSEDEL.

Below is given a detailed list of the particular functions of the lamps and buttons beginning from the top left-hand corner:

0	shows the contents of the overflow register 0 (1 meaning overflow and 0 no overflow).
T	shows the contents of pos.00 of the accumulator, R, which govern the sign of R; this can be changed using the buttons below T.
OA, OB, .., KB	show the contents of the indicator register. KA and KB can be set or reset using the corresponding buttons.
YE	is lit up when input or output is made to or from a peripheral unit and also when GIER waits for a unit, e.g. input from typewriter.
M1, M2, M3, M4	show which status level, GIER is executing: M1 is lit during the introductory address determination, M2 during the execution of the basic operation, and M3, M4 during the possible concluding modifications.
h	is lit when GIER executes a left-hand half-word instruction, and also when GIER is stopped after execution of a LH half-word instruction. Otherwise, the lamp is not lit.

KLAR is lit when GIER is stopped i.e. when the power is on but the computer is not running.

SF is lit when the power is not on (e.g. during starting up or when there is a breakdown).

TO is lit when the current instruction contains an undefined basic operation (or when there is a breakdown); GIER stops after the introductory address determination.

TR is lit when there is a parity error during reading from the drum.

L is lit when there is a parity error during reading from the paper tape reader; GIER stops ready to execute the next instruction (cf. section 9.3.1).

The rightmost lamps in the upper row indicate the register whose contents are being shown in the lower row of lamps, the operator may select a fresh register using the corresponding button. The possibilities are as follows:*)

R The Accumulator (excl. pos.00 which is shown at the left of the upper row and is called T).

M The Multiplier Register.

O The Operand Register (one of the registers in the arithmetic unit).

H The Hold Register where calculations (including address calculations) take place.

L The Storage Transfer Register which acts as a buffer between the core store and the arithmetic unit.

---

*) Some of these registers are described in more detail in chapter 2 of Volume I.

F The Function Register which contains the current instruction excluding the address constant: during execution of a whole-word instruction, pos.20-41 contain the current operation code (plus modifications etc., while pos.10-19 contain the increment, during the first micro step, or status level, only. During execution of a half-word instruction, pos.30-39 are set to zero. Pos.0-9 of the F register constitute the <u>index register p</u>.

r1 The Control Counter, which shows the address in the core store of the current (or next) instruction.

s1 The Subroutine Register, which is used during calculation of s-modified addresses.

r2 The Address Register, which contains the core store address.

s2 The Auxiliary Address Register which is used during address calculations (together with r2).

in The Indicator (whose contents are also available at the left-hand end of the upper row of lamps, but it is only possible to change the contents by selecting in and using the buttons below the lower row of lamps).

ta The Drum Address Register which contains addresses in core store during drum transfers.

tk The Drum Track Register.

bl The Tape Reader Buffer; bl always contains the next character to be read (from tape) by an LY instruction. Immediately after the LY instruction's execution, the next character on tape will be transferred to bl (cf. also section 9.3).

bs The Output Buffer; bs always contains the last character which has been output to a peripheral unit (cf. also section 9.3).

by The Peripheral Unit Register which contains information about which peripheral unit has been selected (cf. also chapter 9).

The text over the lower row of lamps indicates where the separate parts of an instruction are placed when the selected register contains a whole or half-word instruction. Note, especially, that:

a) Pos.0-9, which are designated ADRESSEDEL on the console panel, show only the address constant, while the remainder of the address - indirect modification, indexing (p or s) or relative addressing - is shown in pos.27-29.

b) Pos.33-34, designated IO show the indicator operation (I, M, N or L) and pos.35-39, designated IA, I_A and I_B show together the indicator address (cf. section 4.9 in Volume I).

When GIER is stopped, one may set the contents of each position separately as desired, using the buttons below the row of lamps; using the two buttons at the extreme left, it is possible to fill the whole of the register with noughts or ones (pos.00 of the accumulator must, however, always be set using the buttons below the lamp T).

10.1.2 Start and Stop buttons.

At the base of the console panel, there are two sets of start and stop buttons designated NORMAL and MIKROTEMPI. Their function is as follows:

NORMAL START may only be used when GIER is stopped (the KLAR lamp is lit) as a result of:

a) the execution of a Halt instruction, or

b) the use of the button NORMAL STOP, or

c) the use of the button MIKROTEMPI STOP.

When NORMAL START is pressed, GIER will begin executing the instruction in the cell whose address is held in r1. If NORMAL STOP is kept depressed, then each time NORMAL START is pressed, one instruction will be executed.

If the MIKROTEMPI STOP button has been used, a number of special conditions come into play (see below).

NORMAL STOP: When this button is pressed, GIER stops when the execution of the current instruction is completed (cf. section 10.2). Any drum transfers in progress will also be completed before the computer stops. (If the instruction in progress is a left hand half-word instruction, the lamp h will be lit after the halt).

If, on account of a programming error, the computer comes into a closed chain of brackets (a loop of indirect addresses, which "refer to itself") or, if, on account of a machine error, the computer enters a loop in a micro-program (for instance, when the "ready" signal from a peripheral unit does not arrive), GIER will not stop when NORMAL STOP is depressed *). In this case, it may be necessary to use MIKROTEMPI STOP (see below).

---

*) GIER "remembers" however that this button has been pressed and will under all circumstances stop before execution of the next instruction, if, for instance, a missing ready signal suddenly arrives.

MIKROTEMPI START: Assuming that GIER is stopped, pressing this button will cause one step in the micro program of the instruction in progress to be executed. The microprogram for each instruction consists of many (i.e. 20-30) micro-steps which are split up into 4 main levels of operation, and the lamps M1, M2, M3, M4 in the upper row of lamps show which status level the machine is in the progress of performing.

MIKROTEMPI STOP or RESET: When this button is pressed, GIER stops immediately, if necessary in the middle of an instruction; any drum transfer in progress is also interrupted (before completion). Since this button will, at the same time, bring into play a number of other functions (zeroising of certain registers etc.), it is rather dangerous to use the button as a stop button during a normal run, it being possible to corrupt the contents of a cell in the core store in this way. The functions brought into play are:

a) The control unit prepares to execute a fresh whole-word instruction or a left-hand half-word instruction: the lamp h is turned off, if it was lit, and the lamp M1 lights up. The next instruction will be taken from cell [r1] and depending on when the RESET button is pressed, this will be the current instruction or the next instruction. (If RESET is pressed during execution of a LH half-word instruction, this instruction will be repeated when GIER is re-started).

b) The typewriter is set in Lower Case.

c) All functions concerning the drum are interrupted (if there is a persistent error while reading from the drum or if a non-existent track has been selected, GIER can only be stopped by use of MIKROTEMPI STOP, see section 9.2.2).

d) All functions concerning peripheral units are interrupted (the "ready" circuits are reset).

e) by[0] is set to zero. (In connection with the HP button). Thus, if during a run, one wishes to stop GIER, NORMAL STOP must be used. If, during a run, one wishes to make changes in a program, the HP button on the auxiliary console can be used (see below).

10.2 Operation of the main console.

10.2.1 Computer stop.

When programs are run using the HELP-SLIP system, it is normally not necessary to stop GIER. (When GIER is "idle" it should always be waiting for typewriter input). However, the GIER can be stopped in one of two ways:

a) By pressing NORMAL STOP. GIER stops when the instruction being processed has been completed, on the verge of executing the next instruction (in technical terms, GIER stops on the first micro-step of status level M1 of the next instruction). The instruction just completed will be held in pos.10-41 of the F register; the register r1 will contain the address of the cell containing the next instruction.

b) By a programmed stop instruction, GIER stops as above on the verge of the next instruction, after the stop instruction including any modifications has been processed.

In both cases any drum transfers in progress or the activation of peripheral units will be completed correctly and when NORMAL START is pressed, GIER will continue with the program.

10.2.2 Start (Re-start).

If GIER has been stopped by a programmed halt (or NORMAL STOP) and one wishes to continue normally, it is only necessary to press NORMAL START.

If, however, GIER has been stopped in one way or another and one wishes to transfer control to another part of the program, it is first necessary to press MIKROTEMPI STOP; r1 should be selected and set equal to the address of the cell containing the first (LH half-word or whole-word) instruction to be executed; when NORMAL START is pressed GIER will begin "from the left" in the required cell.

It should be noted that, if GIER has been stopped by a LH half-word stop instruction and one changes the contents of r1 (without using MIKROTEMPI STOP), when NORMAL START is pressed GIER will begin to execute the right-hand half-word instruction of the cell indicated in r1.

10.2.3 Execution of single instructions.

An instruction can be executed manually in two ways, since it is possible to execute the instruction without placing it in the store or to place it in the store and at the same time let it be executed.

In both cases it is necessary that the computer is stopped:

a) An instruction is required to be executed without being placed in the store:

1. Press MIKROTEMPI STOP.
2. Press MIKROTEMPI START 3 times.

3. Select the L register and put in it the required instruction bit for bit, either as a whole-word instruction or as a left-hand half-word instruction.

4a. If NORMAL STOP is held down and NORMAL START is pressed once, the required instruction will be executed, GIER stopping on the fringe of the next instruction.

4b. If, however, one wishes to let GIER continue as if the inserted instruction had been placed in that cell whose address is in r1 at the start of the routine, it is only necessary to press once on NORMAL START.

b) <u>An instruction is required to be placed in the store and thereafter executed:</u>

1. Press MIKROTEMPI STOP.
2. Select the register r1 and set in it the desired storage address.
3. Press MIKROTEMPI START <u>2 times</u>.
4. Select the L register, which now contains the selected cell's previous contents. The desired contents of this cell must now be set in L.
5. Press MIKROTEMPI START <u>once</u>. The new contents of the L register will be placed in the selected cell in the store.
6. The instruction can now be executed using one of the two methods described under 4a or 4b above.

It must be emphasized that after MIKROTEMPI STOP, the NORMAL START button must not be used before MIKROTEMPI START has been pressed 3 times, since GIER may not otherwise function properly.

c) <u>Step-by-step running</u>: If one holds the NORMAL STOP button depressed and continually press NORMAL START, it is possible

to run through a program instruction by instruction. Each time the start button is pressed, one instruction will be executed (however, a substitution instruction or a chain of such instructions <u>and</u> the following instruction will be executed in one "step").

d) <u>Changes in the contents of the core store</u>: This is achieved by first performing steps 1, 2, 3, 4 and 5 as described in subsection b above. Step no.6 will instead consist of pressing MIKROTEMPI STOP (after which it is possible to select the register r1 and change its contents so that GIER will begin at the required part of the core store).

<u>Example 10.1.</u>

A program consisting of 40 cells and stored on track 0 on the drum is required to be placed in the core store in cells 10-49, after which the program is to be executed beginning with cell 10.

Assuming that the computer has been stopped, this can be performed as follows:

1. Press MIKROTEMPI STOP.
2. Select the register tk and clear it.
3. Press MIKROTEMPI START 3 times.
4. Select the L register; clear the register and set the address part equal to 10 (1 in positions 6 and 8) and the operation equal to LK, that is the value 52 (1 in positions 20, 21 and 23).
5. While NORMAL STOP is depressed, press NORMAL START once.
6. Press MIKROTEMPI STOP.
7. Select the register r1 and set in it the address 10.
8. When NORMAL START is pressed, GIER will begin to execute the instructions in cells 10, 11, ...

Example 10.2.

One wishes to read in a program tape using a loader (input program) stored from cell 512 onwards.

Assuming that GIER has been stopped, this can be done as follows:

1. Put the tape in the reader and press the button which reads the first character from the tape to the input buffer b1.
2. Press MIKROTEMPI STOP.
3. Select the register r1 and set it equal to 512.
4. Press NORMAL START after which loading will commence.

Example 10.3.

GIER has been stopped. The contents of cell 203 are to be changed so that the right-hand half-word instruction is changed to SR p+4. The program is to be re-started at cell 200. This is achieved as follows:

1. Press MIKROTEMPI STOP.
2. Select the register r1 and set it to the address 203.
3. Press MIKROTEMPI START twice.
4. Select the L register; put the address 4 in the increment (1 in pos.17), the operation SR in the right-hand operation part (i.e. the value 3 in positions 30-35, namely ones in pos.34 and 35), and also p indexing in the right-hand half-word, i.e. ones in pos.38 and 39. The left-hand half-word must not be disturbed. Check that the contents of this word contain a half-word flag (Lpos[40] = 1) but no F flag (Lpos[41] = 0).
5. Press MIKROTEMPI START once. The whole of the contents of the L register are thereby transferred to cell 203.
6. Press MIKROTEMPI STOP.
7. Select the register r1 and set it equal to 200. When NORMAL START is pressed GIER will begin to execute instructions beginning with cell 200.

Example 10.4.

GIER has been stopped and one wishes to re-start the program at the RH half-word instruction in cell 30. This is achieved as follows:

1. Press MIKROTEMPI STOP.
2. Select the register r1 and set it equal to the address 30.
3. Press MIKROTEMPI START 3 times.
4. Select the L register and set an innocuous instruction as the LH instruction; it is easiest to put in the instruction QQ 0 by clearing pos.0-9 and 20-29. (The RH half-word contained in the L register is the required re-start instruction).
5. When NORMAL START is pressed GIER will begin by executing the instruction QQ 0 which is dummy, and continue with the required program. The contents of cell 30 are not changed by this procedure.

10.3 The HP button etc.

The auxiliary console, situated beside the typewriter, consists of a number of lamps and buttons of which the HP button is the most important. By using this button it is possible to interrupt programs at will, in order to influence the program (for instance, to make corrections or to obtain storage dumps), and thereafter be able to continue running the program "as if nothing had happened".

The HP button in itself part of the machine's hardware is, however, designed to work in conjunction with the HELP-SLIP software. The effect of the HP button may thus be other than described here if used, for example, in conjunction with ALGOL

programs. This manual will only be concerned with programs which are written using the HELP-SLIP programming system. An advantage of the use of the HP button is that the "main" console becomes redundant and all operator communication is achieved using the auxiliary console and the typewriter. These things will be described in detail in the following chapters, so it will be sufficient here to mention only the lamps and buttons which are on the auxiliary console itself:

At the left there are two large buttons:

RESET, which is in fact identical with the MIKROTEMPI STOP button on the main console, described above.

HP, the interrupt button. When this button is pressed the following functions are performed: the instruction being processed is completed, after which an image of the core store is stored on the last tracks of the drum and a jump is made to the input program SLIP which requires typewriter input in order to continue. (A more detailed description of this function can be found in section 13.2).

At the right of the console there are a number of lamps, which, taken from the top, are as follows:

"K_A" and "K_B" with their respective push-buttons, which are identical with the corresponding lamps and buttons on the main console. It is thus possible to ascertain and determine contents of the registers KA and KB from both console panels.

"Klar" is a copy of the Klar lamp on the main console. It is lit when GIER has been stopped, which ought not happen

when running with the HELP system. GIER can however be started by pressing the HP button which causes a jump to the HELP administration (see chapter 13).

"YE" is a copy of the YE lamp on the main console. It is lit when a peripheral unit is activated.

"str.læs. par.fejl" is a copy of the L lamp on the main console. It is lit when GIER has stopped owing to a parity error on a paper tape; cf. also section 9.3.1.

"Tromlefejl" is a copy of the TR lamp on the main console. It is lit when there occurs parity errors during reading from the drum.

"TO fejl" is a copy of the TO lamp on the main console. It is lit when the current instruction contains an undefined basic operation.

"HP spærret", is lit when the interrupt button is locked i.e. when by[0] = 1; see also the description of HELP in chapter 13 *).

"i-løkke" is lit when GIER cycles a long time in a chain of indirect addresses (and the lamp M1 is permanently lit). The only way to stop this is to press the RESET button and GIER will then stop, about to execute the next in-

---

*) If the HP button is depressed while the computer is running and also while by[0] = 1, nothing happens before by[0] is cleared (by a VY instruction); the interrupt will first occur at this point; thus, GIER "remembers" when the HP button has been pressed during a run. This is, however, not the case when GIER is stopped and the HP button is locked (this should incidentally never occur!). When RESET (≡ MIKROTEMPI STOP) is pressed, by [0] is cleared.

struction even though the instruction with the chain of indirect addresses has not been executed; if this happens in a LH half-word instruction, GIER will however stop, ready to repeat the same LH instruction (because the Control Counter r1 has not been increased and because the half-word bit h is cleared.

11. SLIP (SYMBOLIC LANGUAGE INPUT PROGRAM)

11.1 Introduction.

This chapter deals with the use of the loader program SLIP, which allows for, among other things, symbolic addressing. The program itself is closely related to the system of service routines called HELP and may, in fact, be regarded as a part of this system. The whole of the HELP system is described in more detail in chapter 13, while the present chapter will, in general, be confined to describing the way in which instructions and numerical and literal constants should be written and how they can be knitted into a complete program. Only when one understands the general mechanism of the HELP system will it be possible to understand the way in which SLIP works especially as regards the entry to and exit from the loader program. Thus this chapter should be regarded as a survey of the rules for writing programs which are to be loaded by SLIP.

Note that, although SLIP means literally "Input Program" its major use should be to load program and program-constants (including text). SLIP ought not to be used for input of data during a run; each call of SLIP takes approx. 1 sec. for admin-

istration (storage and replacement of the core store image).

Thus, for input of data, one should use the library routines designed for this purpose (see the GIER System Library, edited by Regnecentralen).

11.2 An example of a simple loader program.

(This section has been deleted).

11.3 The loader program SLIP.

11.3.1 Introduction. Storage allocation.

SLIP may, as mentioned in section 11.1, be regarded as a subroutine in the HELP system designed to load program and program-constants.

An important facet of SLIP is that symbolic addressing may be employed for both address constants and increments; this is a great help during programming and debugging. It is also recommended that the use of relative addresses is combined with the use of symbolic names so that it is comparatively easy to insert or remove instructions in the course of testing a program.

The part of the core store available for loading via SLIP is cells 10–1022 since the permanent SLIP administrator is stored from cells 0–9 and cell 1023 is used to store check totals for output. On the drum, tracks 58–293 are available, since tracks 0–57 are used for storage of SLIP and HELP (of which tracks 0–31 containing SLIP and the main part of HELP are normally locked), while tracks 294–319 are reserved for an image of the core store, formed as the program is loaded: everything

which is programmed for the core store is, in fact, stored in a corresponding position in the image on the drum, and only when loading is complete (just before the exit from SLIP), are tracks 294-319 transferred to the cells 0-1023 in the core store (it is in fact possible to load program or data into cells 0-9 but it is not recommended since the SLIP administrator will be overwritten).

Similarly, prior to each run of SLIP the current contents of cells 0-1023 and all the registers are stored on tracks 294-319, after which a new program is loaded. Thus the net result of a complete run of SLIP is as if the program read in had been loaded directly into the core store.

The time taken to transfer the 26 tracks of core store image to or from the drum is 26 x 0.02 sec. ≈ 0.5 sec. so a program should be loaded using as few calls of SLIP as possible i.e. as large portions of program as possible, in one go, to avoid unnecessary drum transfers.

On GIER systems with 3 drums (tracks 0-959), track nos. 934-959 are used for the image of the core store; throughout the following, where tracks 294-319 are discussed, 640 should be added to these track numbers to make these remarks appropriate for GIER with 3 drums.

In the following sections the most important facets of SLIP are described in detail, while in section 11.9 a complete syntax for input to SLIP is given. (The syntax is presented in the same manner as for the ALGOL 60 Report).

11.3.2 Lines of information.

In order to load a program consisting of instructions, lit-

eral constants and numerical constants, SLIP must be supplied with extra information, control information. It is thus necessary to distinguish between the <u>program</u> information (instructions and constants), which are loaded into GIER as part of the complete program, and <u>control</u> information, which does not appear in the computer when loading is complete but serves to control the loading operation.

Each program to be loaded by SLIP consists of a number of <u>lines of information</u> which are separated from one another by the Carriage Return symbol (CR). Each line may be one of 6 different types, viz.

Program Information:

- Instruction lines, consisting of one or two instructions
- Number lines, – – one or more numbers
- Text lines, – – one or (paradoxically so) more lines of text
- Binary information lines (the output in condensed form from the HELP routine "kompud", see chapter 13)

Control Information:

- Control line, which may define addresses (definition line)
  - – – control the way in which numbers are stored (control code)
  - – – select peripheral units etc. (control code)
- Help-call line (which selects HELP routines and presents appropriate parameters for these – not in fact mentioned in this chapter but in chapter 13)

The first 3 types of program information are described in sections 11.4 and 11.5, while the control information is discussed in section 11.6.

<u>Dummy information</u>: 1) When lines of text are read in only Tape Feed and All Holes (i.e. characters with holes in all 8 channels) are ignored.

2) When lines other than text lines are read in, Space*), Tab, Stop Code, Punch On, Punch Off, Tape Feed and All Holes are ignored; furthermore, everything from [to] inclusive (including CR) is ignored; finally, everything from and including semi-colon to - but not including - CR is also ignored. This gives possibilities for writing comments and memos in each line, for instance, when writing instructions and numbers: if one wishes to write a comment at the beginning or in the middle of the line it must be embraced by the square brackets; if a comment is required at the end of a line the square brackets may also be used but it is also sufficient to begin the comment with a semi-colon.

The symbols _ and | are ignored, unless they come at the beginning of a line or precede a symbol in Lower Case (in which case only those combinations mentioned in section 11.6 will be accepted, the remainder being treated as <u>s</u>).

A number of CR symbols after each other have the same effect as one CR, so that all but the first CR symbol are dummy.

---

*) If a program is input from typewriter a space immediately following a CR will cause the instantaneous address i to be typed out in red (and perhaps also the instantaneous track no. k); cf. chapter 13.

11.4 Loading of Instructions Using SLIP.

In this section the make-up of an instruction line is described. This description will assume the character of an extension of the rules for writing instructions laid down in chapters 3, 4, 5 and 6 of Volume I. The way in which instructions are loaded to definite sections of the core store or drum is discussed later, in section 11.6.

An instruction line is always loaded to a complete cell; the line consists either of 2 half-word instructions or one whole-word instruction; half-word instructions are separated by a comma (as previously mentioned) or a stroke (slash). Note that a half-word instruction may be empty. Thus, for instance, the instruction line

,AR 17

will cause the LH half-word to be cleared, corresponding to insertion of the instruction QQ 0 while the instruction AR 17 will be loaded as a RH half-word instruction.

11.4.1 The Basic Operation, Modification and Indicator Instruction.

The operation code may be written with letters in either lower case or upper case or a mixture of the two and apart from the 57, 2-letter mnemonics mentioned in chapter 5 and section 8.1, mnemonics for 7 other dummy*) operations are acceptable,

---

*) 4 of these, namely IL, US, GC and PC are additional basic operations for the GIER system with a Buffer Store (4096 words of ferrite core) or Process Control Unit.

and will cause the following bit patterns to be loaded by SLIP into pos.20-25 (or pos.30-35) of the appropriate cell:

dummy operation	bit pattern	decimal value
IL	101100	44
US	101101	45
ZG	101110	46
GC	101111	47
PC	110000	48
ZJ	111001	57
ZL	111110	62

In this way it becomes possible to represent each of 2↑42 possible bit patterns in a cell, completely in the form of ("pseudo") instructions (with attendant modifications).

S modification can be written as S (as previously) or as n. (S is a mnemonic for Sletning, the Danish for clearing and n may be regarded as a mnemonic for "null and void").

The floating-point modification F may be written with a capital or a small letter.

The modifications X, V, D must be written using capital letters (as previously).

Indicator instructions are to be written with capital letters as laid down in Volume I.

The order in which the different parts of an instruction are written is quite arbitrary, with the exception: each instruction must begin with the (basic) operation code and the address constant must precede the increment; otherwise, one may shuffle modifications, indicator instruction, address constant and increment together as one will. The reason for this is that the bit pattern corresponding to a given instruction line is built up using a series of logical additions; each time a constituent

part of an instruction is read in by SLIP, the bits corresponding to that part, appropriately scaled, are added logically to the bit pattern already read in from the same instruction line. The use of this technique means also that any number of modifications or indicator instructions may be written in one instruction line; for instance, twentyfour X-s and three n-s in one instruction line will have the same effect as one X and one n.

11.4.2 The Address.

In chapters 3 and 4 the concepts of absolute addresses, indirect addresses, p- and s-indexed addresses and increments were introduced; everything which was said about these concepts regarding notation and meaning is still valid for input to SLIP and the rules mentioned below are simply extensions of the previous notation:

1) A simple address may consist of:

a) r, s, p or nothing

b) +, -, or nothing

c) an integer or nothing.

d) The above may be enclosed by brackets (for indirect addressing).

If the sign between a) and c) is omitted, SLIP will assume a plus. Thus, for instance, the addresses

s24 and s+24

are equivalent. The order of writing the elements a), b), c) is not irrelevant as the integer must always come last. Any integer is allowed, but the address constant is always formed modulo 1024. An empty address is the same as 0.

2) <u>Labels</u>: Labels comprising one of the letters a, b, c, d, e followed by an integer may be used to represent values in the address (or increment). Thus the following labels are available

a0, a1, a2, ...
b0, b1, b2, ...
⋮
e0, e1, e2, ...

(The labels a0, b0, c0, etc. may be written as a, b, c, etc.).

Labels may be <u>defined</u>, i.e. become associated with a numerical value, in one of two ways: either, by means of a definition line or by preceding a line with the name of the label (cf. section 11.6.3 Definition lines etc.). But, irrespective of the way in which the label is defined it is of significance for the loading of an instruction line whether any label occurring in this line is defined before the line in question or whether the definition comes after this line. In the first case the label is known as a <u>pre-defined label</u> and the value thus assigned to this label is directly set in the appropriate address when it is read in; in the other case the label is known as a <u>post-defined label</u> and SLIP must store information about the instructions where <u>post-defined labels</u> are used so that when the definition is made the appropriate values can be set in the addresses of these instructions. The difference between SLIP's treatment of pre-defined and post-defined labels causes a certain difference in the rules for their respective use.

3) <u>Addresses with post-defined labels</u>: The structure of addresses containing labels which are first defined after the instruction containing the address, is as follows:

a) r, s, p or nothing

b) + or nothing

c) a post-defined label.

d) The above may be enclosed by brackets (for indirect addressing).

Note that a minus is not allowed and that absence of a sign is regarded as plus.

Example 11.2.

Examples of address parts with post-defined labels:

```
c0
s+c1
(pd11) which is the equivalent of (p+d11).
```

Whereas the following are unacceptable if the label is post-defined

```
-c0
c1+3
3+c1
s-d11
```

because neither a minus nor an integer may appear together with a post-defined label. (In the case of the address 3+c1 the expression will be in fact regarded as the address constant 3 together with the increment c1; see below in section 11.4.3).

If the address is relative, the instantaneous value of the word counter is subtracted from the value of the label before the address constant is set. This is necessary to enable one to create relative addresses with symbolic address constants so that the address is independent of where the program is placed in the store.

Example 11.3.

In the section of program

```
    ---
    HVS r+3 IZA
    AR p+2, GR 124
    MK 124, GR 125
    AR p+4, MK 126
    ---
```

it is not possible to insert or remove an instruction without changing the relative address of the jump instruction HVS r+3. If one writes

```
    ---
    HVS r+c1 IZA
    AR p+2, GR 124
    MK 124, GR 125
c1: AR p+4, MK 126
    ---
```

the effect of the program will be the same as the original one since SLIP performs the following functions: The label c1 is not defined when the instruction line HVS r+c1 IZA is read in, and is therefore put on a "waiting-list". When SLIP reaches the instruction line c1: AR p+4, MK 126 the symbols c1: serve to define the value of the label c1 as the address of the cell into which the instruction is to be loaded (see further section 11.6.3). SLIP will now look up in the waiting list and find that c1 has been used earlier whereupon the defined value will be set in the address where c1 has occurred. But since the address in the HVS instruction is relative, the address of the cell where the HVS instruction has been loaded will be subtracted from the value associated with the label c1; the final form of the instruction will thus be HVS r+3 IZA.

In this program instructions can be inserted or removed without affecting the meaning of the instruction HVS r+c1 IZA since SLIP will always interpret the required address constant as being the difference between the addresses of the

cells containing the instructions AR p+4 and HVS r+c1 IZA. Furthermore the program can be loaded anywhere in the core store thanks to the use of relative addressing. If, on the other hand, one wrote

```
    ---
    HVS c1 IZA
    AR p+2, GR 124
    MK 124, GR 125
c1: AR p+4, MK 126
    ---
```

c1 in the HVS instruction would be replaced by the absolute address of the cell containing the AR instruction and the program would be forced to remain in exactly that part of the core store where it was first loaded.

4) <u>Addresses with pre-defined labels</u>: Labels which are defined before being used in the address parts of instructions may be used in the said address parts in several ways but the basic structure of such addresses is as follows:

a) r, s, p or nothing

b) +, - or nothing

c1) an arbitrary number of pre-defined labels separated by + or -. The last (and only the last) term may also be an integer.

d) The above may be enclosed by brackets (for indirect addressing).

As previously, absence of a plus or a minus will be regarded as meaning +. In contrast to addresses with post-defined labels where each address can include only one label, any number of labels may be used here.

Example 11.4.

If a3, a7 and b14 are defined before they are used in addresses, the following addresses will be accepted by SLIP:

(a3-a7)
s + b14 + a3 + 1
pa7 - a3 which is the same as p + a7 - a3

whereas the following are invalid

b14 - 3 - a3
pa7a3

because in the case of the former the number 3 may not be followed by the terms -a3, and in the case of the latter a plus or minus sign has been omitted between a7 and a3. (Incidentally, in the case of the first expression, b14 - 3 - a3, it will be regarded as consisting of the address b14 - 3 and the increment -a3 which is quite admissible, whereas the other expression will cause an error message).

The limitations regarding the positioning of integers in address expressions illustrated in Example 11.4 can be overcome since item c) includes a further possibility

c2) Any of the terms occurring in expressions named in c1) may comprise an integer followed by a pre-defined label without any sign between them.

The "missing" sign will be interpreted as a plus (and not - as may be expected - as a multiplication sign) and the whole term, integer plus label, is regarded as being enclosed in the usual arithmetical brackets.

Example 11.5.

In accordance with rule c2) the addresses

```
s + 5a7 - a3
b14 - 3a3
p1b14
```

are acceptable and SLIP will calculate the values of

```
5 + a7 - a3
b14 - 3 - a3
1 + b14
```

as the address constants for the respective instructions. (The first address will be s-indexed and the last p-indexed).

If an address is relative, it will be treated in much the same way as for post-defined labels, the instantaneous value of the word counter being subtracted from the value of the label before this value is used in calculating the address constant. If only one label occurs in the address and if the preceding sign is plus, the effect will be exactly as described in example 11.3; but it must be emphasized that if several (pre-defined) labels occur in a relative address, the word counter will be subtracted from the value of each label before the terms are added/subtracted as indicated. (The result will, presumably, be very rarely of any use).

Example 11.6.

In the following 2 lines of program

```
a1: PS s+1, AC 100
    ARS r+a1-2 X
```

the second line will be stored having an address constant -3 i.e. as the instruction ARS r-3 X.

As a program is read in, SLIP keeps a record of the cell and the drum track into which the program is to be loaded. The instantaneous values of the <u>serial address</u> and the <u>serial track number</u>, signified by i and k respectively, may appear in addresses in the same way as pre-defined labels:

c3) i or k may occur instead of pre-defined labels in any situation covered by c1) and c2) above.

If an address is relative, the values of i and k are not however reduced as labels are.

In section 11.6 below the serial address and track number are discussed in detail.

<u>Example 11.7.</u>

If the instruction line

LK i+25, VK 0

is to be read into cell 100, it will be loaded as the instructions LK 125, VK 0; the line LK r+25, VK 0 will on the other hand be loaded as the instructions LK r+25, VK 0.

5) <u>Scaling</u>: In the case of all the addresses mentioned above, each term is treated as an integer with units in position 9 or, when the address is for a RH half-word instruction, position 19. It is often expedient tc be able to read in numbers with the units justified to other positions in the word (see, for instance, the initialisation parameters for editing numbers, in chapter 12). This can be done by scaling the term in the address expression:

c4) Every term in an address containing pre-defined labels may be scaled by means of a suffix consisting of a

period (full stop) followed by an unsigned integer. (The term may be any of the types mentioned in c1, c2 and c3).

NB. A scaled integer does not need to be the last term in an address expression.

The effect of scaling is that the integral value of each term is scaled (using the TK shift instruction) so that the units coincide with the stated position in the word, before the term is used to evaluate the address expression*). It is quite possible to use scaling which falls outside the address part of a cell.

One should note here the way in which addresses in RH half-word instructions are treated: as these addresses are read in, they are formed in the address positions 0-9 with appropriate consideration for scaling; when an address has been read in completely, the whole of it is shifted 10 positions to the right. In this way the scale factor is automatically increased by 10 if applied to right-hand addresses. (This applies also to increments; cf. section 11.4.3 below).

The expressions introduced under c1), c2), c3) and c4) will in the remainder of this manual all be referred to as <pre-defined address>. A <pre-defined address> is thus a collection

---

*) In this respect there is a somewhat illogical difference in the treatment of integers $\geq$ 512 (or negative integers) and labels, whose value is $\geq$ 512: The integers are treated as negative numbers which, on being shifted to the right in a cell, will always be supplied with a string of ones in front; the value of a label is always treated as a positive number which, on being shifted to the right, will be supplied with a zero string in front.

of terms separated by + or −, where each term may be

a) a pre-defined label (including i or k) with or without a preceding integer and with or without scaling.

b) an integer with scaling.

The last term in a pre-defined address may be an integer without scaling.

Example 11.8.

The instruction line

    PI 1.7+1.8, VY 1.4

will be loaded as the instruction PI 6, VY 32.

Example 11.9.

The instruction line

    AR (p+1.0+1.1) X

will be loaded as the instruction AR (p−256) X.

Example 11.10.

SLIP does not make any check as to whether the extent of the cell has been exceeded by scaling but executes slavishly the required shift operations (without round-off):

Let a1 be (pre-)defined as 100. Then the instruction line

    MKF a1.2, AR a1.50−1

will be loaded as the instructions MKF −512, AR −1.

Example 11.11.

In a type c2) term, i.e. an integer followed by a label, for instance 17a1, with scaling, the whole term e.g. 17a1 is scaled. Thus the instruction line

ARn 17a1.8

is loaded as the instruction ARS 234, if a1 has the value 100.

Example 11.12.

Initialisation parameters for editing of numbers can easily be read in by means of scaled integers. Let us consider the initialisation which is required for a number-editing routine, discussed in the next chapter. The parameters selected are b = 6, h = 4, f1 = 1, d = 3, n = 1, bE = 3, f2 = 2, g1 = 4 and g2 = g3 = g4 = g5 = 0.

The meaning of these parameters will be evident in the next chapter where it is also possible to discover that the parameters must be introduced into a cell as follows:

b with units in pos.	3		h with units in pos.	7
f1 – – – –	9		d – – – –	13
n – – – –	14		bE – – – –	17
f2 – – – –	19		g1 – – – –	23

These parameters could thus be read in with the whole-word instruction

QQ 6.3 + 4.7 + 1.9 + 3.13 + 1.14 + 3.17 + 2.19 + 4.23

The scaling .9 is in fact superfluous but if it is omitted the integer 1 must be the last term in the address expression; one could therefore just as well write

QQ 6.3 + 4.7 + 3.13 + 1.14 + 3.17 + 2.19 + 4.23 + 1

(and in fact the remaining scaled integers do not need to be written in any particular order).

11.4.3 Increment.

An increment may be written in the same way as the address with the exclusion of relative, indirect and indexed addresses. The increment part may thus be either

A simple increment consisting of

a) +, - or nothing

b) an integer

or

An increment containing a post-defined label consisting of

a) + or nothing

b) a post-defined label

or

An increment containing pre-defined label(s) consisting of

a) +, - or nothing

b) a <pre-defined address> as defined above.

If scaling is used on an element of an increment, the position number indicated is increased by 10 as the increment is read in (see the previous section).

Separation of address part and increment. Since the address may consist of an arbitrary number of elements and since the increment can easily be mistaken for an address it may be necessary to separate them in an easy and unambiguous way. The following possibilities are available in SLIP:

a) The increment may be preceded by t (small letter). This possibility is always available and it is recommended that one makes a habit of writing t in front of all increments.

b) Any of the instruction modifications S, n, F, f, X, V, D may be written between the address and increment and will separate the two "addresses".

c) Any indicator instruction including the dummy indicator operation I placed between the address and increment will distinguish the one from the other.

d) If the address expression is concluded by an integer any term following this will be regarded as an increment (cf. the rotation used in Volume I e.g. AR p+4+1).

Example 11.13.

In the instruction line

VK r+a17 t1

the address is r+a17 and the increment is 1.

Example 11.14.

In the instruction line

LY a21 t−1

the address is a21 with increment −1, whereas the same instruction line without the separator t i.e.

LY a21 −1

will be read in as having the address a21 − 1 and increment 0 if a21 is pre-defined. (If a21 is temporarily undefined (i.e. post-defined) the line will be syntactically wrong and will cause error reports to be made).

Example 11.15.

The instruction line

BT s+5 −2

will be read in as the operation BT with address s+5 and increment −2 . One might just as well have written BT s+5 t −2 or BT s+5 I −2 , since I is a dummy indicator operation which is stored as zero in pos.33−34 (cf. section 4.9.2 of Volume I).

Example 11.16.

In the instruction line

ar a1 n a2

the operation-modification n (clearing) separates the address a1 and the increment a2 , whereas the instruction line ar n a1 a2 will cause an error report because a separator between a1 and a2 is missing; if one writes ar n a1+a2 the loaded instruction would acquire an address a1+a2 and increment 0 , assuming of course that a1 and a2 are pre-defined.

11.5 Input of numbers and text using SLIP.

11.5.1 Input of Numbers.

SLIP can read and load numbers in 4 different ways, as either floating-point numbers, fixed-point numbers, integers, or packed integers; a number-line is distinct from an instruction in that it always commences with a digit, a sign or a point; selection of the way in which a number is loaded depends partly on the structure of the number and partly on the administrators m and f. (m and f are examples of control-code lines).

Each number or group of numbers (packed integers) is loaded into one cell. Numbers are separated by at least one of the following terminators:

1) CR or comma; if either of these symbols follows immediately after a number the marker-bits of the cell in question are cleared.
2) The letters a, b or c; if any of these symbols follow immediately after a number the marker-bits of the cell in question are set appropriately.

Numbers are written in the same way as defined in the ALGOL report, section 2.5, i.e. with or without a decimal point and with or without an exponent to the base 10. Otherwise numbers are read in as:

floating-point numbers if the most recent administrator read in previously is f; any base-10 exponent may not be larger than 152;

fixed-point numbers if the most recent administrator read in is m and, at the same time, the number contains a decimal point or base-10 exponent; the number 1.0 will be stored as the largest possible number in GIER, being $1 - 2\uparrow(-39)$; if one attempts to read numbers outside the range $-1.0 \leq x \leq 1.0$, SLIP will give an error report (see section 11.8 below);

integers with units in position 39, if the most recent administrator read in is m and, at the same time, the number contains as decimal point or exponent.

Packed integers consist of one or more integers, separated or concluded by one or more oblique strokes (slashes). The presence of a stroke has the effect that, firstly, numbers are read in as integers irrespective of the current mode indicated by the administrator, and secondly, the units position of the numbers is shifted 10 positions to the left for each stroke to the right of the number. If an integer is too large to be included in the space so allocated, there are stored as many bits, taken from the (RH) end of the number, as there is room for (i.e. the last 10, 20 or 30 bits). In particular the ones prefixed before negative numbers will only be set within the allocated space (see Example 11.18.).

NB. Packed integers may not commence with a stroke.

Example 11.17.

The control and number lines

<u>m</u>
1, 1.0, $50_{10}-2$, −0.5, 300, 250/

will cause the following to be stored (in 6 consecutive cells): the integer 1 (units in pos.39), the fixed-point numbers $1 - 2\uparrow(-39)$, 0.5 and −0.5 , the integer 300 (units in pos.39) and, finally, the integer 250 with units in pos.29. However, the lines

<u>m</u>
$-0.1_{10}2$

will cause an error report because the limits for a fixed-point number in GIER have been exceeded.

Example 11.18.

The lines

<u>f</u>
1, 1.0, $50_{10}-2$, −0.5, 300, 250/, 0/−200

will cause the following to be stored (in 7 consecutive cells): the floating point numbers 1.0, 1.0, 0.5, −0.5 and 300.0, the integer 250 with units in pos.29 and finally the integer −200 in positions 30−39 with units in pos.39.

Example 11.19.

Irrespective of administrators, the line

1//−4a

will cause a cell to be loaded with the number 1 in the LH half-cell (pos.0−19) with units in pos.19 and the number −4 in the RH half-cell (pos.20−39) with units in pos.39. Finally, the cell will be a-marked.

1025///

will cause the number 1 to be loaded with units in pos.9 while the rest of the cell will be cleared.

11.5.2 Input of Text.

If a line commences with the symbol t, every symbol (including any CR symbol) up to and including the first semi-colon, will be regarded as a text string so that all symbols that are read in will be stored in successive cells, 7 symbols per cell. In each cell, symbols are, however, stored "backwards", the first symbol read being stored in pos.36-41, the next in pos. 30-35 and so on. The 6 positions which each symbol fills contain a bit-pattern which is identical excepting parity bit, with the punched tape code for the symbol in question; the only exception from the above being the symbol CR, (the only symbol using the 8th. channel on the tape), which is stored as the pattern 111111.

The only symbols which are ignored as text is read in are Tape Feed (7 holes) and All Holes (8 holes).

When SLIP reads the concluding semi-colon, this symbol is not stored but a special terminal symbol (the unused combination 001010) is stored instead. After this loading will take place in the next whole cell regardless of how much the previous cell has been filled up. Thus a text string will always fill a whole number of cells.

There exists a HELP routine which can be used to print (output) text strings read in by SLIP; see the next chapter.

Example 11.20.

The text "line"

```
t Yes, we have no bananas
Easter 1964;
```

will produce a text string consisting of 40 symbols (including Case Shift, Space, Terminal symbol etc.) which will be stored in 6 whole cells.

11.6 Control Lines and Blocks in SLIP.

11.6.1 The Serial Address and Serial Track No.

Loading via SLIP is always made to successive cells in the core store or to successive cells and tracks on the drum. SLIP uses two "registers" or counters to keep control of the addresses involved:

The serial address i always indicates in which cell in the store the next piece of program information is to be loaded; when the cell is filled up i is increased by 1.

i can be set to a desired value (e.g. the first address of a block of cells into which a program is to be loaded) in two ways: 1) by writing i=<pre-defined address> in a block head (see below), where <pre-defined address> means an expression of the same type as that introduced in section 11.4.2; 2) by writing, anywhere in the program, a line similar to the above viz:

i=<pre-defined address>

In each case the serial address is set equal to the value of the pre-defined address on the right-hand side.

When input to SLIP is started from scratch the serial address is i=10 and if there is no good reason for changing it one need not define it more explicitly; loading will thus take place into cell 10 and onwards.

The serial track number k always indicates the number of the track on which the next cells of program are to be loaded. Each time a track is filled up 1 is added to k.

It is only possible to assign a value to k via the drum block head (see below) where one writes k=<pre-defined address>. k will then assume the value of the defined address.

When input to SLIP is started from scratch the serial track number is set automatically to k=294. The reason for this is that all input - as mentioned in the introduction - is in reality loaded to the drum, and any parts of the program which are destined for the core store, will be loaded into the Core Store Image i.e. tracks 294-319. It is only necessary, in the latter case, for the programmer to specify the serial address i , as SLIP keeps an automatic record of the appropriate track numbers (and automatically transfers the Core Store Image to the appropriate parts of the core store when input is finished). Only in those cases where a part of a program is to be loaded elsewhere on the drum is it necessary to define a value of k; a further discussion of these matters is given in the section on drum blocks, 11.6.4. It should be mentioned here, however, that in the Core Store Image, the 1st cell on track 294 corresponds to cell 0 in the core store, the 2nd cell on the track to cell 1 etc. etc.

Example 11.21.

The definition line

    i = 401

causes the following program to be loaded to cell 401 and onwards.

The definition line

i = i+5 or i = 5i

causes 5 cells to be skipped (left undefined) as a program is loaded.

11.6.2 Program Blocks.

SLIP has a facility for restricting the scope of labels by means of blocks similar to those in ALGOL; the most important aspect of this is that labels which are "declared" in a block are local for that block as in ALGOL. For the purposes of this section the name "block" will only refer to parts of programs stored in the core store; the rules applicable to drum blocks, described in section 11.6.4, are slightly different.

A block consists of a block head, the program sequence and a block end.

Block head: A block begins with the symbol b followed by optional control information about the start address in the core store and the labels which are to be used internally within the block. More formally, the structure of the block head is as follows:

a) The symbol b

b) i = <pre-defined address>

c) One or more labels, separated by commas; the maximum number of labels is 5 since each of the initial letters permissible may only occur once.

The parts b) or c) may be omitted but the symbol b by itself is ignored completely.

The effect of a block head is as follows: i = <pre-defined address> causes the serial address to be set equal to the value

of the address on the right-hand side, i.e. the piece of program following is loaded to the cell (and following cells) having this address in the core store. If part b) is omitted, the program will be loaded in continuation of the program read in immediately before the block head.

If labels are used in the body of a block SLIP must have indication of this in the head of the same block or in the head of a block which embraces it, in the form of a "declaration": if a label occurs in a block head, one may use labels with the same initial letter as this label, freely within the body of the block; the digital part of such labels must be less than or equal to the digital part (or this number + 1 if it is even) of the declared label. Values are assigned to these labels either by means of a definition line or by using it to "label" a line. (The occurrence of a label in a block head in SLIP has thus a similar effect to that of a declaration in ALGOL).

Example 11.22.

A label in a block head corresponds to a declaration in ALGOL in the respect that values are not assigned but that the possibility of using certain labels is established.

Example 11.23.

The block head

b i=250, a3

causes loading to take place in cell 250 and after; the labels a0, a1, a2 and a3 may occur in the block. The block head

b i=250, a4

has the same effect on the loading as above but the labels a0, a1, a2, a3, a4 and a5 are now allowed.

In many programs blocks will occur inside each other, and the scope of the declared labels will correspond exactly to the situation in ALGOL: 1) If a label is declared in an "inner" block, it may only be used in the part of program within that block (including any other blocks embraced by that block). 2) If a label is declared twice, once in an outer block and once in an inner block, it will act as two different labels, of which one may only be used in that part of the outer block which surrounds the inner block. This means that the label must be defined twice, once at each "level", and that the value assigned to the label in the outer block is inaccessible from the inner block and vice-versa.

The Block Tail: A block is terminated as follows:

a) The symbol e

b) a <pre-defined address> or nothing

The effect of a block tail is firstly that labels declared in the corresponding block head are deleted, i.e. they may not be used anymore (unless they are also declared in a block embracing the terminated block); the values which have (possibly) been assigned to these labels are inaccessible. Regarding the reports made by SLIP and the internal "catalogue of labels", one is referred to sections 11.6.5 and 11.8 below.

Secondly, loading may be terminated completely: If a <pre-defined address> follows e , a jump is made to the cell indicated by the <pre-defined address> and the program will be executed.

If the symbol e alone is followed by CR, loading will only be terminated if the number of e symbols read in exceeds the

number of b symbols; in this case GIER will jump back to that part of the core store from which SLIP was entered (cf. section 11.7 below and chapter 13). As long as the number of e symbols (without following address) is ≤ the number of b symbols, loading will continue. For more details see section 11.6.4 on drum blocks and 11.6.6 on complete programs.

Example 11.24.

In the program

```
┌ b i=100 d3
│ d0: PM 700 IPA
│     MK (d0), GR (d0)
│   ┌ b a1
│   │     ARS (d0), NK r+a0
│   │ a0: PP    0    t+1
│   └ e
│     GR (d0), ZQ 0
└ e 100
```

the label d0 is replaced by the value 100 throughout, because the labels d0, d1, d2 and d3 are only declared in the outer block. The label a0 is assigned the value 103, but may only be used in the inner block.

Example 11.25.

As the following program:

```
┌ b i=100, d3
│ d0: PM 700 IPA
│     MK (d0), GR (d0)
│     ┌ b d0, a1
│     │     ARS (d0), NK (r+a0)
│     │ a0: PP    0    t-1
│     └ e
│       GR (d0), ZQ 0
└ e 100
```

is input, GIER will, on termination of the inner block, protest against the use of the label d0 in the inner block where it is only declared but not defined (see section 11.8.2 below). On the other hand, both the MK instruction and the GR instructions (before and after the inner block) will be loaded with an address constant of 100.

For the block heads shown above, d1 is in exactly the same situation as d0 , since the block head of the inner block

```
b d0, a1
```

makes the "original" values of d0 and d1 inaccessible within the inner block. Whereas, d2 and d3 are usable throughout the program (if, of course, they are defined at some point or other).

Example 11.26.

As the following program:

```
b i=100, d3
d0: PM 700 IPA
    MK (d0), GR (d0)
    b a1
        ARS (d0), NK (r+a0)
    a0: PP     0  t-1
    e
    QQ 512 t512
    HV i+3
    b i=i+2
    IT (a0), GT d0
    ZQ 0, HV d0
    e 100
```

is read in, GIER will protest against the use of the label a0 in the final small block. This can be resolved by moving the declaration of the label a0 (and thus automatically a1) up to the head of the outer block, so that the program will commence with

```
b i=100, d3, a1
```

thus making the value of a0 available throughout the program.

The object of this complicated mechanism to restrict the scope of labels is the same as in ALGOL: It should be possible to include sub-routines (in ALGOL: procedures) written by others in SLIP without having to worry about avoiding those labels, which are used in the sub-routine. This can be achieved by making all sub-routines into blocks which commence with declarations of the names used.

11.6.3 Definition lines and labelled program-lines.

As mentioned in section 11.4.2 above a label (which must have been declared in a previous block head) can get a value, an integer in the range $0 \leq t \leq 1023$, in two ways:

1) In a definition line which looks like this:

<name> = <pre-defined address>

where <name> is one of the labels introduced in section 11.4.2, or the letter i (indicating the serial address). One may define several names in the same definition line if each definition is of the above form and separated by commas.

The effect is that the value of the pre-defined address on the RH side is calculated, after which the value is assigned to the name on the LH side. (It becomes thereafter a pre-defined label, cf. section 11.4.2 sub-section 2). *)

---

*) There is also a rather special facility for reference to a RH half-word: <label> h = <pre-defined address> has besides the usual effect (assignment of value) the following effect: In those instructions which have been read in previously, where the label occurs as a post-defined label the basic operations HV and PA will be changed to HH and PT respectively. Note that only these two basic operations will be changed and this only in instructions read in prior to the definition and never in succeeding instructions where the label is pre-defined.

In a definition where i occurs on the LH side, the effect is that the value of the pre-defined address on the RH side is assigned to the serial address, and loading will proceed beginning with this address.

2) A label may also be defined by labelling a program line i.e. when a label followed by a colon precedes a program line; a program line may be an instruction line or a constant line; a label referring to a RH half-word is also written at the beginning of a line and will normally look like an ordinary label, although it may include the letter h between the label and the colon. One may write several labels in the same program line; each label must be followed by a colon.

The effect of labelling is that the label acquires the current value of i , which is thus the address of the cell in which the following program line is to be loaded. The letter h has, as with definition lines, only one effect, namely that, if any HV or PA instructions containing the label in the address part have been read in previously, these instructions are changed to HH or PT instructions, respectively.

Example 11.27.

In the piece of program

```
a1=7
ARS 202 IPA a1
```

the addition instruction acquires the increment 7 at the time it is read in, whereas in the piece of program

```
ARS 202 IPA a1
a1=7
```

this will first occur after the definition line has been read.

Example 11.28.

The piece of program

```
i=25, a0=1, a1=100

ARS 5a1 ta0
MK 100a1 ta0
HV i-2 NT
```

is read into cells 25, 26 and 27 as the instructions

```
[25] ARS 105 t1
[26] MK  200 t1
[27] HV   25 NT .
```

Example 11.29.

In the piece of program

```
b i=100, a2

a0: PPS 10, PP p-1
    PM p+700, MK p+720
    BS p t0
    HH r+a0

a1: ZQ a1 t1
```

a0 acquires the value 100, a1 the value 104, and when the program has been loaded the instructions look like this:

```
[100] PPS 10, PP p-1
[101] PM p+700, MK p+720
[102] BS p+0 t0
[103] HH r-3
[104] ZQ 104 t1
```

Since the jump instruction has a relative address, the serial value of i, in this case i=103, is subtracted from the address constant during input.

Example 11.30.

The piece of program

```
    i=200

    AR (r+b3), GR (r+b3)
    HV r+b3 NZ
    ---

b3h: ---
    ---
    HV r+b3 LO
```

will be loaded as

```
[200] AR (r+3), GR (r+3)
[201] HH r+2 NZ
      ---
[203] ---
      ---
[205] HV r-2 LO
```

since only the one jump instruction, where b3 occurs as a post-defined label, is changed to a HH instruction through the label b3h. The addresses in cell 200 refer as normally to the LH address-part of cell 203.

Re-definition. A label may be re-defined i.e. a new value may be assigned to it at the same block-level as the original definition as long as the re-definition is accomplished by a definition line, and not by labelling a program line. (There are however no restrictions regarding new definitions at other block-levels assuming of course that the label in question is re-declared).

Example 11.31.

Consider the program (where dashes represent instructions)

```
b d1
---
d1=14
---
i=100, d1=i-2
---
```

d1, when it occurs in the 1st and 2nd sections of the program, will be replaced by the value 14, while d1 in the 3rd section will be replaced by the value 98.

SLIP will, however, probably protest against the program

```
b d1
---
d1=14
---
d1: AR 400 t1
---
```

because the re-definition is made by labelling. The program will only be accepted by SLIP in the event that the AR instruction is in fact loaded to cell 14, since it is permitted to check the value of a pre-defined label by labelling the cell which the value of the cell indicates.

Labelling on an empty line: If a label and colon are the only items on a line, the labelling will refer to the following line of program; the CR between the label and the program line has only typographical significance - no cells will be skipped during input.

Example 11.32.

The part of the program

```
i=50
GR r+a4, MKS p+31
AR r+a4 D
a4:
SR (s-1) t1
```

will be loaded to cells 50-52 as the instructions

```
[50] GR r+2, MKS p+31
[51] AR r+1 D
[52] SR (s-1) t1 .
```

11.6.4 Drum blocks.

If a program is too large to be in the core store at one time, it can be split up into drum blocks, which are thus loaded

to the drum and can successively be read into the same section of the core store. From the syntactical point of view, the drum block resembles the program blocks mentioned previously, since each drum block consists of a block head, the program body and a block tail; of these, the program body and block tail have exactly the same structure as the core store block, and the block head differs only from the description in 11.6.2 in that there must be a definition of the block's location on the drum tracks:

A drum-block head consists of

a) the symbol b

b) k = <pre-defined address>

c) i = <pre-defined address>

d) One or more labels separated by commas; these may not be more than 5 labels since each initial letter may only occur once.

The parts b), c) and d) are separated by commas. Any one, two or all of these parts may be omitted. A drum-block head causes the succeeding program to be loaded to the drum until the corresponding block tail is read; the program is loaded to track k and onwards, k being, during input of part b), set to the value of the expression <pre-defined address>.

The serial address i does not directly influence the loading of a drum block, as it always begins in the 1st cell of the track indicated, but i should usually correspond to the block's later location in the core store: If i is used in addresses or increments or definition lines in the drum block and if reference to labels is made without the use of relative address-

ing, one must set i equal to the address of the cell in the core store to which the 1st cell of the drum block will be transferred during a run of the program.

Irrespective of whether i is defined in the drum-block head or not, the serial address is increased in the normal way with 1 for each cell that is loaded on input, and for every 40th cell k is automatically increased by 1. When a drum block is terminated i and k are reset to the values they had before the drum block (including the block head) was entered, i.e. the status of the core store is exactly as if the drum block has not been read in at all. (i and k are <u>not</u> reset on termination of a core store block).

<u>Example 11.33.</u>

The program block

```
┌ b k=60, i=100, a7
│ --- } program filling 100 cells
└ e
```

will be loaded to tracks 60, 61 and the first half of track 62. During input i will assume the values 100, 101, ..., 199, successively, but after the symbol <u>e</u> has been read in, i and k will be reset to the values they had just before the drum block was entered.

If i does not occur in the drum-block program, the block head

<u>b</u> k=60, a7

will have exactly the same effect.

Example 11.34.

A large program may, for example, be split up into a main routine which is always situated in the core store and two minor routines loaded as drum blocks and sharing the same portion of the core store; it might be written as follows:

```
b i=10, a4, b7, c13
---
---  } the main routine
┌ b k=60
│ ---
│ ---  } 1st drum block
└ e
┌ b k=75
│ ---
│ ---  } 2nd drum block
└ e
e
```

The 1st and 2nd drum blocks will be loaded to, respectively, tracks 60 and 75 (and onwards), but in both cases the serial address will be adjusted as if they had been placed in the core store in continuation of the main routine. The main routine must thus contain instructions which transfer the appropriate block to this common part of the core store when it is required.

Restrictions on the use of drum blocks.

1) If global labels (i.e. labels declared outside a block) are used within a drum block, they must be defined before they are used; it is irrelevant whether the definition occurs before the drum block is entered, or the definition occurs in the drum block itself before the label is used. Labels declared in a drum block are not restricted in this way but must conform to the rules of section 11.6.3.

Example 11.35.

In the section of program,

```
┌ b a2
│ ---
│ ┌ b k=50
│ │     HV r+a2
│ │     ARS p-100 LZ
│ │ a2: GR 496 t+1
```

SLIP will protest against the occurrence of the global, post-defined label a2 in a drum block. But if one moves the declaration and writes

```
┌ b
│ ---
│ ┌ b k=50, a2
│ │     HV r+a2
│ │     ARS p-100 LZ
│ │ a2: GR 496 t+1
```

it will be accepted, because a2 has now become a local label. (If the block were not a drum block then there would be no problem).

2) SLIP does not keep control of the number of tracks used nor does it take any notice of whether the serial address is a reasonable size or not; the programmer must thus make sure for himself that drum blocks do not overlap each other on the drum and that while drum blocks are being loaded, the serial address i does not change to a value less than the first address, or to a value which is beyond the extent of the core store.

3) Normally, it serves no useful purpose to load drum blocks to tracks 294-319 as this section of the drum is used by SLIP for an image of the core store.

11.6.5 Control codes.

As mentioned previously, some of SLIP's functions are governed by control codes, each one an underlined letter. Some of the control codes have been mentioned previously but in this section a review is made of all the control codes and other control lines with a detailed description of those not mentioned previously.

1) Control codes:

f : Input of floating-point numbers (see section 11.5.1).

m : Input of fixed-point numbers and integers (see section 11.5.1).

t : Input of text (concluded by semi-colon) (see section 11.5.2). *)

l : Paper tape reader is selected as input unit (with typewriter as output unit). Input from punched tape commences immediately - this is normally the situation required when a program is read in.

s : Typewriter is selected as input (and output) unit. If GIER has read - from tape - the control code s, a CR is written to the typewriter and GIER will await further input from the typewriter this being indicated by the lighting of the green lamp on this unit.

If a space is typed, a report will be typed (in red) indicating the state of the serial address i and the serial track number k. This report can also be obtained if a CR and space is typed immediately after a block head or a block end.

*) t is not really a control code in the same sense as the others but has been included in this survey.

A CR followed by a space will otherwise only result in a report about i.

r : In instruction-lines following this symbol all addresses which are not p- or s-indexed will be automatically made relative. This effect can however be suppressed by preceding an address (in place of r, s or p) by the letter m.

This facility has been introduced because in many programs, especially those written as sub-routines, there will be far more relative addresses than absolute addresses.

The automatic creation of relative addresses continues in a program until the control code n is read.

n : In instruction lines following this symbol all adresses are read in normally i.e. without the automatic creation of relative addresses described above. (The letter m becomes thereby superfluous, but harmless, in absolute addresses). This situation remains until r is read (cf. example 11.37 below).

2) Other control lines:

b i = <pre-defined address>, <label>, ..., CR: Block head with definition of serial address, and declarations (see section 11.6.2).

b h = <pre-defined address>, i = <pre-defined address>, <label>, ..., CR: Drum-block head with definition of serial track number and core-store address and with declarations (see section 11.6.4).

b CR: Dummy information.

e CR: Block end (see sections 11.6.2 and 11.6.4).

e <pre-defined address> CR: Block end together with termination of input by jumping to the cell indicated by the address (see section 11.6.2).

<label> = <pre-defined address>, <label> = <pre-defined address>, ..., CR: Definition or re-definition of one or more labels and possibly the serial address i (see section 11.6.3).

d <label> = <pre-def.addr.>, <label> = <pre-def.addr.>, ..., CR: This line has exactly the same effect as the same line without d (cf. above). The symbol d is thus quite superfluous but may be included to improve readability.

c <pre-def.addr.> CR: This line has the same effect as the line i = <pre-def.addr.> CR i.e. it is a definition of the serial address. Note that this line can be written completely in Lower Case (unless + occurs in the address expression) as opposed to the line with i = etc.

x <pre-defined address> CR: Label Table Dump: The tables containing all the declared labels and their values, if defined, together with certain administrative informations (block-levels etc.) can be stored on 7 consecutive drum tracks using the control code x. The number of the first track to be used is given in the address which should be in the range 58 to 287 inclusive. If no address is given, the tables are stored on tracks 287–293.

z <pre-defined address> CR: This control code may only be used when x has been used earlier with the same address, as z causes the tables etc. which are stored on the drum beginning at the track indicated, to be retrieved in order to re-establish the input situation as when the corresponding x function was called; this includes the selection of peripheral units and other settings made by the other control codes. (See example 11.38 below).

u <pre-defined address> CR: This control line sets the exit

address from SLIP equal to the address indicated. This means that, when sufficiently many e symbols have been read, input will be interrupted to execute the program whose first instruction is at the cell indicated by the address. (See section 11.7 below).

h <HELP routine name> CR: This control line causes a HELP routine to be executed. Most HELP routines require that a number of control parameters follow this line. See otherwise chapter 13 below.

All other small letters which are underlined work in exactly the same way as s, so that erroneous underlined letters will return one to the typewriter where corrections can be made. For special (historical) reasons, however, g is regarded as dummy information.

Example 11.36.

If a long program is required to be split up into a number of shorter tapes, one can conclude each tape with s. When one tape has been read in, the next is set in the tape reader after which one can type l whereupon this tape is read in, in continuation of the one read in previously, etc., etc.

In order to check whether the tape has been read in to the correct place in the store, one may type CR followed by space before typing l. In this ways a report will be typed giving the serial address i and the serial track no. k. i and k indicate where the next tape will be loaded.

All tapes should, in fact, terminate with s as SLIP will then be set in a natural "idle" situation.

Example 11.37.

The following part of a program

```
r
a1: ARS a4 t1
    SR (a1) t1
    DK m+100, IT p-1
    BT m+7, HV a1
    HR s+1
a4:
n
```

will be loaded as the instructions

```
ARS r+5 t1
SR (r-1) t1
DK 100, IT p-1
BT 7, HV r-3
HR s+1
```

and the terminating n causes the program which follows to read in in normally. Note that the control codes r and n have absolutely no effect on indexed addresses.

Example 11.38.

During de-bugging, when one wishes to make corrections to a program which has been read in it is very useful to be able to use the same labels as were used when the program was written. This is, however, only possible if the control codes x and z are used, as otherwise one loses all information about the labels as soon as the input is terminated. One should thus insert the following

x <track number> CR

just before the block tail of every large block in the program. (The track numbers selected in each case must differ by at least 7 from each other, because the tables fill 7 tracks).

Before one reads in corrections to a given block, one must thus restore the SLIP situation (the meaning of labels etc.) by writing the line

z <track number> CR

where the track number must be the same as with the corresponding x-line.

Consider that the following program has been read in:

```
┌ b a7, b9
│ a1: ---
│ a2: ---
│     ---
│
│   ┌ b a5
│   │ ---
│   │ a1: ---
│   │     ---
│   │ b0: ---
│   │ x            [store label tables on tracks 287-293]
│   └ e
│     ---
│ b1: ---
│ x 280            [store label tables on tracks 280-286]
└ e a1             [loading completed, jump to cell a1]
```

If one requires to correct cell no.a1 in the inner block, during a run, one must interrupt the run (using the HP button) and type

```
z
i=a1
```

followed by the correction, and finally 3 times e CR after which the run will be re-started just where it was interrupted.

If one wishes to correct cell no.a1 in the outer block, one should type

```
z 280 CR
```

thereby re-establishing the labels b0 and b1, and a1 and a2 with their values in the outer block.

11.6.6 Programs.

When the loading of programs via SLIP is commenced the effect is as if the following fictitious block head had been read in:

b k=294, i=0

i=10

n

m

All program destined for the core store are in fact, first read to drum tracks 294–319 and only after input is terminated, are these tracks read back to the core store. Cell 0 in the core store corresponds to the first cell on track 294 but input is directed to cell 10 (on track 294, in fact) as cells 0–9 contain the fixed SLIP administration. The control codes n means that SLIP is conditioned to read "normal" addresses, that is, without automatic creation of relative addresses, and m means that SLIP is conditioned to read fixed-point numbers.

If one wishes to read a program under these conditions without the use of labels, it is not necessary to write any block head and one can instead begin to feed instructions and constants directly. The program tape may be terminated by s, so that one can leave SLIP by writing e and possibly an address (and CR) on the on-line typewriter. (See also example 11.39 below).

SLIP regards, in fact, a program as being terminated either when one more e symbol than the number of b symbols has been read (the block structure being thereby balanced when one takes the fictitious block head into consideration), or as soon as an e followed by a <pre-defined address> has been read in, (irrespective of the number of b and e symbols).

11.7 Entry to and exit from SLIP.

11.7.1 Manually-controlled entry.

Irrespective of GIER's instantaneous situation (whether the computer is running or it has been stopped in one way or another), pressing the HP button on the auxiliary console will activate the following functions *):

1. The current contents of the core store are stored on tracks 294-319;
2. The words "hp-knap" (Danish for hp button) is typed in red on the on-line typewriter, followed by the exit address, i.e. the address to which a jump will be made on exit from SLIP (unless the input which follows gives orders to the contrary).
3. SLIP is transferred to the core store and the program is initialised i.e. made ready for the first input.
4. SLIP awaits input from the typewriter.

Thus if one types l on the typewriter, SLIP will start reading a program tape in. If this tape terminates with s, one can thereafter write the concluding e on the typewriter, (followed possibly by an address) and a CR, causing a direct jump from SLIP to the program; one may also call HELP routines from the typewriter at this point. A program tape may terminate with e, with or without address (and CR) but this eliminates the possibilities of manual control at this point and is not to be recommended.

---

*) These functions are described in more detail in chapter 13.

11.7.2 Program-controlled entry.

During execution of a program, jumps can be made directly to SLIP in one of 2 ways:

1) The instruction HSF 2, activates the same 4 functions as with the HP button, except that the word "hsf 2" is written instead of "hp-knap"; the exit address always indicates the cell following that containing the HS instruction (i.e. SLIP acts exactly like a normal sub-routine which terminates with the instruction HR s+1).

2) The instruction HS 2 activates the HP-button functions 1) and 3) given above; no messages are typed, and the actual situation with regard to selected peripheral units is not changed; as above, SLIP prepares to return, on exit, to the cell following the HS instruction.

Execution of each of these instructions (et seq.) takes about 0.5 sec. because the whole of the core store must be copied to the drum.

Finally, one may (involuntarily) enter SLIP, if overflow occurs after floating-point operation; GIER jumps, as mentioned in the list of operations, to cell 0, and this activates the same 4 functions as with the HP button, the typed message being replaced by the word "fl.overløb" (Danish for floating (pt.) - overflow), followed by the address of the cell containing the instruction which caused the overflow. (This means that exit from SLIP will cause a jump to the same instruction unless one indicates another exit address).

11.7.3 Exit from SLIP.

SLIP is, like other sub-routines, disposed to "return to the place from which it was called", and this means that when SLIP is entered it will normally be prepared to jump to the cell following that from which the entry took place. When using the HP button, the re-entry point will be at the cell following the instruction which GIER was executing when the button was pressed.

If the program terminates with e alone, exit from SLIP will take place as described above, but the exit address can be changed in two ways:

1) By writing u followed by a pre-defined address somewhere in the program; in this way the exit address is re-set to the value indicated, and input will continue until terminated by an e.

2) By terminating the program with e followed by a pre-defined address; exit from SLIP to the address indicated will, in this case, occur immediately.

At all events, the Core Store Image will be transferred to the core store from drum tracks 294-319, immediately before exit from SLIP takes place; in this way, that part of the program which was required to be loaded to the core store will be put in its correct place.

A result of the use of this Core Store Image, is that the contents of the core store is identical with the contents of tracks 294-319, immediately after entry to and immediately before exit from SLIP.

Example 11.39.

Execution of the program shown in example 11.38 is to start in cell no.a1-1 in the inner block. Moreover, there is to be possibility for manual control after the program is loaded. Thus, the program should be written as follows:

```
┌ b a7, b9
│ a1: ---
│ a2: ---
│     ---
│
│   ┌ b a5
│   │   ---
│   │ a1: ---
│   │     ---
│   │ b0: ---
│   │ u a1-1        [set exit address = a1-1]
│   │ x
│   └ e
│     ---
│ b1: ---
│ x 280
└ e
  s                 [continue input from typewriter]
```

This tape would be read by first pressing the HP button and then typing 1. When the tape is read in, and the green lamp on the typewriter lights up, one may correct the loaded program or call HELP routines using the typewriter; and when one types e CR, the program will be executed, commencing at the required cell.

11.8 Messages and Reports output by SLIP.

11.8.1 Error Messages.

During input, SLIP makes a syntactic check of the program read in, and when an error is encountered a message is written on the typewriter, giving information on the type and location of the error; after this, in almost all cases, input will continue until the end of the program. In the case of most errors,

this means that only a single cell will be loaded with incorrect information (which can sometimes be corrected from the typewriter on completion of input), and that there is a chance of finding all the syntactical errors in one test-run.

SLIP distinguishes between 9 different types of error, although they all have a common notation: The typed message consists of (CR), an error-type number and the serial address; when program is read in from tape, the last symbol read in is also typed followed by the next 3 lines; after this, input usually continues normally.

The meaning of each error-type number and the action taken by SLIP are briefly described in the table below.

Error Type	Meaning	Action taken by SLIP
1	Syntactical error in instruction line, text line or control line.	Symbols following until the first colon, comma, stroke or CR, are skipped. After this input continues normally (ɔ: As a rule only one call is loaded incorrectly).
2	Illegal use of post-defined label in address or increment.	As with error no. 1.
3	Use of undeclared label.	As with error no. 1.
4	Error in declaration in block head.	As with error no. 1. (ɔ: Normally, only the erroneous declaration is skipped).
5	"Unused" code punched on tape.	The symbol is skipped and input continues normally.
6	Redefinition by labelling where the value does not tally with the earlier definition.	The redefinition has no effect and input continues normally.

Error Type	Meaning	Action taken by SLIP
7	Syntactical error in number line.	The cell in question acquires undefined contents and input continues.
8	Number outside the range allowed (fixed pt.: $-1 \leq x \leq 1$ float.pt.: fl.pt.range and exponent ≤ 152).	As with error no.7.
9	Too many labels/blocks have been used at one time. (Label table has slightly more than 250 cells of which is used ½ cell for each label 1 – – each declaration 1 – – each core-store block-head 2 – s – each drum-block head	Input taken from typewriter as after s. The program can usually not be further read in satisfactorily.

11.8.2 Reports.

The serial address and the serial track number: When a program is input from the typewriter it is possible to cause the serial address i to be typed out after each new line by typing a space after CR. If CR-space is typed after a block head (b followed by declarations etc.), a block tail (e without address), a definition line or s, has been read in, both the serial track no. and the serial address (in that order) are typed out. These messages are typed in red.

The values of labels used: If KA and/or KB is set, a list of all labels used within a block, including values of i and k, is output at the end of a block. If KA is set (= 1) the output is punched on tape and if KB = 1 the output is typed. The format of this output is as follows:

a) The address of the first track of the drum block last entered.

b) The address in the core store of the first cell of the block just terminated.

c) The serial address, i.e. the address of the next unoccupied cell in the core store, following the block just terminated.

d) The address of the track, to which the next program line will be loaded in the block just terminated.

e) All labels which are declared and used within the block just terminated, followed by their defined values.

Example 11.40.

Consider a drum block which has been read in with the block head

<u>b</u> k=48, i=218, a5

and let us assume that the block occupies just over 8 tracks. If KA = 1, for instance, the following output will be punched on tape, when the block tail <u>e</u> is read:

```
48     218      320   50
a    225
a1   228
a4   318 219
a5   246
```

and at the same time there will be typed (in black):

```
a4  318  219
```

This report shows that the drum block is loaded to tracks 48-50 corresponding to location in the core store in cells 218 to 319 inclusive. Moreover one can also see that a0, a1 and a5 have acquired the values of 225, 228 and 246 whereas a2 and a3 have not been used; finally, one can see that a4 has been used (last, in the address of cell 318 and the

increment of cell 219) but that it has not been defined within the block.

If KA were set = 0 there would only be typed the error message

a4 318 219

(cf. example 11.25).

11.9 Syntax for input to SLIP.

11.9.1 Programs and blocks.

<program> ::= <group>

<block> ::= b<block head>CR<group><block tail>|<drum block>

<drum block> ::= b k=<pre-defined address>,<block head>CR
<group><block tail>|
b k=<pre-defined address>CR<group><block tail>

<group> ::= <line>|<block>|<group><group>

<block head> ::= i=<pre-defined address>|<label>|
<block head>,<label>

<block tail> ::= eCR|e<pre-defined address>CR

11.9.2 Lines.

<line> ::= <program line><line end>|<controlline>|
<labelling><line end>|<labelling><line>

<program line> ::= <instruction line>|<constant line>|
<condensed line>

<control line> ::= <definition line><line end>|
<control code>|<auxiliary line><line end>

<line end> ::= CR|;<arbitrary string not including CR>CR|
<line end><line end>

<labelling> ::= <label>:|<label>h:

<label> ::= <initial letter><index>

```
<initial letter> ::= a|b|c|d|e

<index> ::= <digit|<index><digit>|<empty>

<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

11.9.3 Instruction lines.

```
<instruction line> ::= <half-word instrn.>,<half-word instrn.>|
                       <half-word instrn.>/<half-word instrn.>|
                       <whole-word instrn.>

<half-word instrn.> ::= <basic operation><half-word mod.>
                        <address><half-word mod.>|<empty>

<whole-word instrn.> ::= <basic operation><bi-operation>
                         <address><bi-operation><t>
                         <increment><bi-operation>

<basic operation> ::= <letter><letter>
(only the combinations listed in section 11.4.1)

<half-word mod.> ::= <clearing flag>|<floating flag>|<empty>
                     <half-word mod.><half-word mod.>

<clearing flag> ::= n|S

<floating flag> ::= f|F

<bi-operation> ::= <half-word mod.>|<whole-word mod.>|
                   <indicator op><indicator addr.1>
                   <indicator addr.2>|<bi-operation>
                   <bi-operation>

<whole-word mod.> ::= X|V|D

<indicator op> ::= I|M|N|L|<empty>

<indicator addr.1> ::= K|Z|O|T|P|Q|R|<empty>

<indicator addr.2> ::= A|B|C|<empty>

<t> ::= t|<empty>
```

11.9.4 Addresses and Increments.

```
<address> ::= <relative flag><address constant>|
              (<relative flag><address constant>)
```

<relative flag> ::= m|r|s|p|<empty>

<increment> ::= <address constant>

<address constant> ::= <post-def.label>|+<post-def.label>|
<sign><pre-def.address>|<empty>

<pre-def.address> ::= <integer>|<term>|<term+<pre-def.address>|
<term>-<pre-def.adress>

<term> ::= <symbolic addr.>|<symbolic addr.><scaling>|
<integer><scaling>

<symbolic addr.> ::= <pre-def.label>|<integer><pre-def.label>|
i|k|<integer>i|<integer>k|

<scaling> ::= .<integer>

<sign> ::= +|-|<empty>

<integer> ::= <digit>|<digit><integer>

<pre-def.label> is a <label> whose value is defined before the label is used in an instruction.

<post-def.label> is a <label> whose value is not defined until after the label is used in an instruction.

11.9.5 Constant lines.

<constant line> ::= <number line>|<text line>

<number line> ::= <number>|<number><terminator>|
<number><terminator><number line>

<number> ::= <number as defined in the ALGOL report section 2.5>|
<packed integers>

<terminator> ::= a|b|c|A|B|C|,|<terminator><terminator>
(only the first terminator, following a number, influences the marker-bits)

<packed integers> ::= <sign><integer>|
<sign><integer><stroke>|
<sign><integer><stroke><packed integers>

<sign> ::= +|-|<empty>

<stroke> ::= /|/<stroke>

<text line> ::= t<arbitrary string not including;>;

(<condensed line> is not described further here but has the format of output from the HELP routine "kompud", cf.chapter 13).

11.9.6 Control lines.

<definition line> ::= <definitions>|d<definitions>|
c<pre-def.address>

<definitions> ::= <definition>|<definition>,<definitions>

<definition> ::= i=<pre-def.address>|
<label>=<pre-def.address>|
<label>h=<pre-def.address>

<control code> ::= f|m|l|s|r|n

<auxiliary line> ::= x|x<pre-def.address>|
z|z<pre-def.address>|
u<pre-def.address>|
h<help-routine name>|
h<help-routine name>|<integer>

(See also chapter 13).

12. OUTPUT.

12.1 Introduction.

The routine(s) which provide(s) for the output of results in a readable form constitute a very important part of a program; one must pay a great deal of attention to the planning of output, so that the required information is printed*) in a rational and unambiguous way, and also so that superfluous information is omitted.

When one has decided which results should be printed, one must plan how they are to be printed, and it is useful here to distinguish between the layout of a single number - the local typography, which determines the number of printed digits, the rules for printing signs etc. - and the layout of numbers in tabular form - the global typography which determines the number og values per line, the number of lines per page, the presence of text etc., etc.

---

*) Since output of results via punched tape is generally analogous with printing on on- or off-line equipment, the verbs to print and to punch and related expressions may be used interchangeably in this and succeeding sections.

Routines for editing numbers and printing text are included in the HELP system; these routines are described below but they are only concerned with local typography, and the user must himself program the administration of the global typography around the individual entries into the routines described. An example is therefore given in section 12.3.3, showing a program which prints results in a tabular form using the output routines mentioned above as sub-routines.

12.2 Editing of number using the sub-routine in HELP.

12.2.1 Function.

For each entry into the routine, one number is printed according to a layout which is determined by a number of parameters. There are 5 possible entry points each of which causes the contents of the accumulator R (or part thereof, or the floating point ditto, RF) to be printed in a different way; if the routine is placed from cell [m] onwards the table below illustrates the 5 possibilities:

Entry at	Contents of	Printed as
cell [m+0]	Rpos. 0 – 9	integer in range $0 \leq h \leq 1023$
cell [m+1]	Rpos. 0 – 9	integer in range $-512 \leq h \leq 511$
cell [m+2]	Rpos. 0–39	integer in range $-2^{39} \leq h < 2^{39}$
cell [m+3]	Rpos. 0–39	fixed point no. in range $-1 \leq x < 1$
cell [m+4]	RF	floating point number

The appropriate number is converted to decimal form and rounded-off to the required number of digits; the number can optionally

be scaled to a multiple of a power of 10 before printing (cf. parameter initialisation and scale factor, below).

Before the routine is entered, the required output unit must be selected by means of a VY instruction; the routine assumes that the selected unit has been set in Lower Case on entry, but the unit is always set in Lower Case on exit.

On exit from the routine, both the accumulator and the M register will have been changed; if the number to be edited is to be used later in the program, it must be stored before entering the editing routine.

12.2.2 Location of the routine; entry and exit.

The routine occupies 120 cells and is stored on tracks 33-35 within the fixed part of the HELP system. The routine may be located anywhere in the core store where it occupies 123 cells, 3 extra cells, in continuation of the routine itself, being used as working locations. The user himself must program the transfer from drum to core store of tracks 33-35; it follows that the start address in the core store may not be greater than 900, since cell 1023 is reserved. A check total of the values of all symbols output is accumulated in cell 1023 (cf. the Operation List in Vol.I).

Entry into the routine from the main program should be made by means of an HS instruction accompanied by a program parameter indicating the address of the cell in which the initialisation parameters (determining the layout as required) are stored. The program parameter and the sub-routine jump may be written, either, as two half-word instructions

QQ<address of parameter-word>,HS<entry address>

or as a whole-word instruction followed by a half-word or whole-word instruction

HS<entry address>

QQ<address of parameter-word>, ---

The entry address should be one of the 5 mentioned in the previous section, and the initialisation parameters will, in each case, be taken from the cell whose address is indicated in the QQ instruction. Since the parameter-word is accessed by means of ARS (s+0) or ARS (s+1) , as the case may be, the address given in the program parameter may be indirect or such-like.

Exit from the editing routine is always made by means of the instruction HR s+1 , on return to the main program the contents of the registers, R and M, will have been changed. If the sub-routine jump is a half-word instruction, the sub-routine index register will not be reset correctly; thus, if the routine is used as "a sub-routine for a sub-routine", the entry into the routine must be made using a whole-word HS instruction viz.

HS<entry address>

QQ<address of parameter-word>, ---

if the sub-routine return mechanism is to function properly.

Example 12.1.

In a program which does not otherwise use 900-1022, one wishes to edit the fixed-point number contained in cell [a2] using a layout which is stored in cell [a5]. This may be programmed as follows:

```
---
VK 33, LK 900
VK 34, LK 940
VK 35, LK 980     [get editing routine from drum]
VK  0, ARS a2     [R:= the number to be edited]
HS 903            [jump to editing routine]
QQ a5, ---
```

12.2.3 Initialisation parameters.

The layout of the printed number is determined by 12 parameters which are selected by the user and packed in a certain way in the cell indicated by <address of parameter-word> . These parameters are used to specify the number of significant digits to be printed, the maximum number of digits before the point and after the point, the maximum number of digits in the base 10 exponent if required, the rules for printing signs in both the mantissa and exponent, an optional grouping of digits with spaces between, and a subtlety regarding the printing of the number 0.

Finally, it is possible to incorporate the automatic multiplication of the contents of R (or RF) register by a given power of ten, into the routine but this is accomplished more or less directly and not via the parameter-word (see section 12.2.4, below).

A layout may be loaded by expressing it as an instruction with a series of scaled terms as the address,

```
QQ <b>.3 + <h>.7 + <f1>.9 + <d>.13 + <n>.14 + <bE>.17 + <f2>.19
    + <g1>.23 + <g2>.27 + <g3>.31 + <g4>.35 + <g5>.39
```

where the acceptable value and meaning of each term is as follows:

$0 \leq b \leq 15$ indicates the number of significant digits to be printed; that is from and including the first digit $\neq 0$, b digits are printed with correct rounding-off of the last digit. However, no more than d decimals will be printed.

$0 \leq h \leq 15$ indicates the maximum number of digits before the decimal point. If the printed number does not have so many such digits, an appropriate number of spaces will be printed first, instead.

$0 \leq d \leq 15$ indicates the maximum number of decimals to be printed. If b significant digits are printed before d decimals have been used, they will be followed by an appropriate number of spaces.
If $d = 0$, the decimal point is not printed; if $d > 0$ but all the significant digits come before the point the decimal point is replaced by a space.

$0 \leq f1 \leq 3$ governs the way in which the sign of positive mantissae is printed, and where the sign is to be placed: Since a minus sign is always printed before negative mantissae there are the following 4 possibilities:

1) The sign is to be printed immediately in front of the first digit (or the decimal point) with

No sign (empty) in front of mantissa	≥ 0:	f1 = 0
Space – – – –	≥ 0:	f1 = 1
Plus sign – – – –	> 0	f1 = 2
Space – – – –	$= 0$	f1 = 2

2) The sign is to be printed as the first symbol of any number before any space or digit with

Plus sign in front of mantissa > 0 }
Space – – – – $= 0$ } : f1 = 3

$0 \leq n \leq 1$ has only significance regarding the printing of the nought before the point in mantissae between −1 and +1. In this case, n = 0 will cause printing of h spaces before the point, whereas n = 1 will cause the printing of h − 1 spaces followed by a nought (before the point).

$0 \leq bE \leq 7$ indicates the maximum no. of digits in the exponent which is printed as the symbol $_{10}$ followed by a sign and a maximum of bE digits. If the exponent has less than bE significant digits an appropriate number of spaces are inserted between the mantissa and the symbol $_{10}$, so that the last digit is always printed in the (bE+2)th position after the mantissa.

If the exponent = 0 but bE has been selected > 0, the exponent is printed as an equivalent no. of spaces.

If bE is set = 0 (or omitted completely), the number is printed without a separate exponent (nor with "equivalent" spaces).

$0 \leq f2 \leq 3$ governs the way in which the sign is printed before the exponent, in exactly the same way as f1 does for the mantissa (see above).

$g1$, $g2$, $g3$, $g4$, $g5$ should fall within the range $0 \leq g \leq 15$ and controls a possible grouping of digits with intervening spaces or, where appropriate, the decimal point: $g1$ indicates the number of digits (or positions) in the first group, $g2$, the number of digits (or positions) in the second group, and so on. The mantissa can thus be dealt up into 6 groups, the decimal point being always regarded as a separator between groups. If the mantissa is preceded by a number of spaces due to "insufficient" integral digits, these spaces are counted as part of a group of digits; the grouping is thus performed on the h+d positions which the mantissa occupies (excluding the sign and decimal point).

Example 12.2.

Some numbers the absolute values of which are between 10^3 and 10^{-1}, are required to be printed with 4 significant digits and without exponent; the sign of positive values is to be represented by a space, and no special grouping is required other than that given by the decimal point. Thus the following values of the parameters should be used: $b = 4$, $h = 3$, $d = 4$ (to make room for 4 significant digits also for the smaller values), $f1 = 1$, $bE = f2 = 0$ (no exponent), $g1 = h = 3$, $g2 = d = 4$ and $g3 = g4 = g5 = 0$. The only parameter not yet assigned a value is n, and if the true decimal fractions are to be printed with a nought before the decimal point, n is set = 0.

This layout might be stored in cell [a5], for instance, by means of the following instruction line:

```
a5: QQ 4.3 + 3.7 + 1.9 + 4.13 + 1.14 + 3.23 + 4.27
```

For positive numbers within the given range editing will have the following effect:

```
|***.*   |
| **.**  |
|  *.*** |
|  0.****|
```

where each asterisk indicates a digit and where the vertical lines indicate where printing begins and ends i.e. each number occupies $h+d+2=9$ positions.

The layout selected here corresponds completely to the layout expression {-ndd.d000} for output procedures in GIER ALGOL.

There are a number of rules regarding the interplay between the different parameters:

1) The mantissa occupies normally $h+1$ printing positions when $d = 0$ and otherwise $h+d+2$ positions, in which case some of the first and some of the last positions may be filled up with spaces; if the option $f1=0$ for printing of signs is used and the mantissa is ≥ 0, it will occupy one position less (this option can be used if one wishes to save space when printing positive numbers).

2) The exponent (including sign) uses no space if $bE=0$, but otherwise normally occupies $bE+2$ positions, of which some or all may be spaces (in the same way as for mantissa, the selection of the option $f2=0$ will cause the exponents ≥ 0 to occupy one position less).

3) One should always select h, d and b so that $h+d \geq b$ **since the maximum number of digits which can be printed in the** mantissa is $h+d$, If editing takes place <u>without</u> an exponent

(i.e. when $bE = 0$) numbers whose absolute value ie $\geq 10^h$ - i.e. numbers with more than h digits before the point - will be printed with the necessary number of integral digits and the correct value, but the number will occupy more positions than planned (if the absolute value is $< 10^{15}$; otherwise the routine has no other alternative than to supplement the number with an exponent).

Numbers whose absolute value is $< 10^{d-b-1}$ - in other words, numbers whose b significant digits extend "to the right" of the d decimals - will, when printed without exponent, be always printed with exactly d decimals, that is less than b significant digits. Very small numbers will thus be printed as 0.000...0 with d noughts after the decimal point (they are always regarded as positive).

4) When a number is printed with exponent ($bE > 0$) and if $b = h + d$, the mantissa will always have exactly h integral digits and exactly d decimals, all being significant, and the exponent printed will be accomodated thereafter. But if $b < h + d$ there will be several possible powers of ten, all giving the required number of significant digits within the limits indicated by $h + d$. The routine will then select the (uniquely determined) exponent which is a multiple of $h + d + 1 - b$ and which gives b significant digits in the mantissa. *)

*) If the value of the number is so small that this cannot be achieved with an exponent having bE digits it will be at the cost of the number of significant digits in the mantissa: The routine selects the smallest exponent that has bE digits and which is a multiple of $h + d + 1 - b$, and the corresponding mantissa will then be printed with less than b significant digits.

5) Frequently one does not wish to have grouping of the digits of the edited mantissa, and in this case when printing integers (i.e. with d = 0), g1 is set = h, while the remaining 4 g-parameters can be left out. When printing decimal fractions without grouping of digits the setting g1 = h, g2 = d is used the remaining 3 being left out (i.e. set to zero).

Grouping is usually only used for printing mantissae with many digits, where separation makes the output more presentable.

Example 12.3.

In order to illustrate point 3 above, let us consider the following layout

QQ 2.3 + 4.7 + 1.9 + 2.13 + 1.14 + 4.23 + 2.27

i.e. one has selected b = 2, h = 4, d = 2, "usual" way of printing signs, no exponent, no grouping besides the decimal point.

The table below shows the results of editing different numbers using this layout. The vertical lines indicate how many positions are used for each number:

Number in GIER	Printed as
-0.0123...	-0.01
1.234...	1.2
1234.5...	1200
12345.6...	12000
-12500.0...	-13000

the two last numbers exceed the limits of the layout because their absolute values are larger than 10^4.

Example 12.4.

In order to illustrate point 4 above, let us consider the following layout

QQ 2.3 + 2.7 + 1.9 + 3.13 + 1.14 + 2.17 + 2.19 + 2.23 + 3.27

which corresponds to the layout {-nd.000 +dd} in GIER ALGOL. This layout is to be used to edit the numbers 1.2468 10^p where p varies from -4 to 6. The powers of 10 which are used as exponents of these numbers will consistently be multiples of $h + d + 1 - b = 2 + 3 + 1 - 2 = 4$; in addition, the numbers will be rounded-off to 2 significant digits and the results will be as follows:

```
 1.2   ₁₀-4
12     ₁₀-4
 0.012
 0.12
 1.2
12
 0.012 ₁₀+4
 0.12  ₁₀+4
 1.2   ₁₀+4
12     ₁₀+4
 0.012 ₁₀+8
```

The space before the exponent is due to the fact that 2 digits have been allocated to the exponent, since bE = 2.

12.2.4 Scale factors.

If one wishes a number to be multiplied by a scale factor which is a power of ten, before the number is edited - for instance, if a series of fixed point numbers are to be multiplied by 100 under output - it can be achieved by the following modification of the routine, independent of the layout selected:
If the required scale factor is 10^H, the integer H is placed in the address part of cell 14 of the routine before the usual entry takes place. The initial value of this address is 0, but it is unaffected by the routine; therefore, if once one has set a particular scale factor in cell 14 it will remain there and be effective for all following calls of the routine, until one changes the contents of cell 14 or reads in the routine from the drum afresh.

Example 12.5.

A fixed point number is to be multiplied by 1000 before editing and only those digits before the decimal point are to be printed. Let the editing routine be placed in cells 900-1022, let the number be stored in cell [a2] and let cell [a5] contain the layout

a5: QQ 3.3 + 3.7 + 1.9 + 1.14 + 3.23
[b = h = g1 = 3, f1 = n = 1, remainder = 0]

The part of the program for outputting the number could be written as follows:

```
---, ARS a2
PA 914 t3      [set scale factor 10^3]
HS 903         [edit number]
QQ a5, PA 914  [reset scale factor to 10^0]
---
```

If cell [a2] contained the number -0.03456... , for instance, this would have been printed as

| -35|

where the vertical lines show where printing begins and ends.

If one wishes a number to be multiplied by a scale factor which is a power of 2, before editing, this can be done by appropriately modifying the contents of pos.0-9 of the M register - which holds the base-2 exponent for floating point numbers - before entering the routine as for normal editing of floating-point numbers. It should be noted, here that when editing floating-point numbers the routine does not assume that R contains a correctly normalised mantissa; the routine simply regards the contents of R as a binary number with the point between pos.11 and 12 (and R00 as sign indicator), and edits it with appropriate consideration for the exponent in M_{0-9}.

Example 12.6.

The fixed-point number in cell [a2] is to be edited with a scale factor 2^{15}. This can be done by putting the number in R and an exponent 4 (namely, 15 minus 11) in M_{0-9}, and then jumping to the entry for editing of floating-point numbers. Using the same allocation of the core store as in the previous examples, the coding will be:

```
---, ARS a2   [R:= the number to be edited]
PM 4 D        [set the exponent = 4]
HS 904        [edit as fl.pt. number]
QQ a5, ---
```

12.2.5 Examples of layouts; special facilities.

In the table below is shown the effect of editing different numbers with 4 different layouts; these have been selected to illustrate, among other things, the effect of f1, which governs the printing of signs, and n, which governs the printing of "whole number zeros". Notice that the routine always prints correct values (rounded-off within the limits indicated), but if the number has a value outside the expected range, the result may occupy more positions than was planned.

The top half of the table shows the values of each parameter together with the corresponding layout in GIER ALGOL. (No entry means that the parameter value is 0). The values of numbers on entry to the routine are given at the bottom LH corner, and on their right are shown the printed results, where the vertical lines indicate where printing begins and ends for each number.

If one wishes to print only the exponent of a number this can be achieved by selecting b = h = d = 0 , i.e. printing of mantissa with 0 significant digits and 0 digits before and after

b	4	4	4	4
h	2	2	4	2
d	2	2		6
f1	1		2	1
n	1			1
bE				1
f2				2
g1, ..., g5	g1 = g2 = 2	g1 = g2 = 2	g1 = 4	g1 = 2, g2 = g3 = 3
corr. GIER-ALGOL layout	{-nd.dd}	{dd.dd}	{+dddd}	{-nd.dd0 000$_{10}$+d}
0.012345...	0.01	.01		0.012 35
1.2345...	1.23	1.23	+1	1.235
-1.2345...	-1.23	-1.23	-1	-1.235
-123.456...	-123.46	-123.46	-123	-0.001 235$_{10}$+5

the decimal point. There is the one disadvantage that all numbers are treated as positive, i.e. (dependent on f1) a space or a plus is printed before the exponent regardless of the number's sign in GIER.

As will be seen from the table above, the routine is normally disposed to print numbers with the decimal point "under each other" for a given layout. However, it is possible to print numbers whose position is justified to the first significant digit at the price of the number of printing positions not being the same for all numbers; if one in fact selects $h = 0$ but $b > 0$, every number whose absolute value is $\geq$ 1 will exceed the limits imposed by the layout with the result that printing is left-justified. The address constant in cell 0 of the routine will incidentally be increased by the number of positions that are used.

Example 12.7.

On output of a series of integers, the first significant digits are required to be printed below each other and all digits in the number are to be printed. Therefore, h is set = 0 and b = 15, the maximum allowable; since only integers are involved d is set = 0, and n = 1, f1 = 1 (standard treatment of signs), bE = f2 = 0 (no exponent), g1 = 15 and g2 = g3 = g4 = g5 = 0 (no grouping). This layout will for the numbers 0, −1, 12, −123 and 1234, for instance, result in the following output:

```
| 0|
|-1|
| 12|
|-123|
| 1234|
```

12.3 Printing of text using the sub-routine in HELP.

12.3.1 Function.

For each entry into this routine, one text line read in by SLIP is printed out, i.e. the routine prints a copy of what has been read in between t and the first semi-colon which follows it (not including these symbols).

Prior to entry into the routine the required output unit must be selected by means of a VY instruction in the main program; the routine assumes that the selected output unit has been set in Lower Case on entry but always leaves it in Lower Case.

On exit from the routine both the accumulator, R, and the register, M, will be different, while the other registers employed will be re-set to their values on entry.

12.3.2 Location of the routine; entry and exit.

The routine occupies 13 cells and is stored in cells 27-39 on track 16 in the permanent part of HELP. It can be placed anywhere in the core store and the user must himself program the transfer of track 16 to the core store; the start address of the routine may thus not be greater than 1010 (and during transfer of track 16 from the drum 40 cells will always be affected).

Entry from the main program is made by jumping with an HS instruction to cell 0 of the routine, which requires as a program parameter the address of the first cell in which the text is stored. In the same way as with the editing routine, the program parameter and subroutine jump may be written, either as to half-word instructions

QQ <text address>, HS <entry address>

or as a whole-word instruction followed by a whole-word or half-word instruction

HS <entry address>

QQ <text address>, - - -

The text address may be indirect, relative or, s- or p-indexed, since the routine uses the indirect address (s+0) or (s+1) to get at the stored text.

Exit from the routine is always made using the instruction HR s+1, at which stage the registers R and M are changed, while the indicator register and p register are re-set; if the HS entry instruction is programmed as a half-word instruction, the subroutine index-register is not re-set correctly (cf. the corresponding remarks in section 12.2.2 above).

Example 12.8.

If one wishes to include a sequence which types "that was - GIER - that was.", in a program it could be written as follows

```
    ---
    VK 16, LK 983     [read track 16]
    VY 1.5 t7         [input unchanged, output to typewriter]
    VK 0, SY 58       [wait until track 16 has been read,
                                         select Lower Case]
    HS 1010
    QQ b2, ---
    ---
b2: t̲that was - GIER - that was.;
```

If the instruction SY 58 were replaced by SY 60 [select Upper Case] , the sequence would type "THAT WAS + GIER - that was.", because the text read in and stored by SLIP does not begin with a Lower Case symbol but with the symbol for t, alternatively T. The first Lower Case symbol comes between R and -.

12.3.3 Example of the use of HELP output-routines.

In order to illustrate the use of the output routines in a more coherent and realistic example, we will go through part of a program, which, in connection with some calculation, prints a table with the following format:

```
        HEADLESS. 28.4.1772

   t      x      y        teta
  0     ————   ————    ————
  0.1   ————   ————    ————
  0.2   ————   ————    ————
  --    ---
  2.0   ————   ————    ————
  divergence = ————
```

The horizontal lines represent the calculated values of x, y and teta, and finally, the divergence; x, y and teta are calculated

as fixed point numbers and are to be printed with the layout {-n.dddd}, while the divergence (also fixed pt.) is to be printed with the layout {+n₁₀+d} ; t is calculated as a floating pt. number and, as can be seen is to be printed as {n.d} .

Let t be stored in cell no. c0, and let corresponding values of x, y and teta be stored in the cells c1, c1+1 and c1+2. The main features of the program should then be as follows:

```
           b a3, b4, c1
           ---                 [includes definition of the label c1]
           VK 16, LK 860       [put text routine into cells 887-899]
           VK 33, LK 900
           VK 34, LK 940
           VK 35, LK 980       [put number routine into cells 900-1019]
           VY 1.4 t7           [select punch for output]
           SY 64, SY 64        [2 times CR]
           HS 887
           QQ b0, SY 64        [punch leading and CR]
           VK 0                [wait until drum transfer is completed]
           PA a3 t20           [re-set line counter]
  ┌-> a0:  ---
  |        ---                 [calculation of a value of t, x, y and
  |                                                              teta]
  |        SY 64, ARSF c0      [punch CR]
  |        HS 904              [punch a value of t]
  |        QQ b2, SY 0
  |        PA a2 tc1-1         [re-set address of x]
  | ┌-> a1: SY 0, SY 0
  | |  a2: ARS c2-1 t1         [R:= x or y or teta]
  | |      HS 903              [punch R]
  | |      QQ b3, ARS a2
  | └--    NC c1+2, HV a1      [jump to a1 the first 2 times]
  |    a3: BT 20 t-1
  └----    HV a0               [jump to a0 (fresh line) the first 20
                                                              times]
           SY 64, SY 64        [2 times CR]
           HS 887              [punch the final text]
           QQ b1, ARS c1+3
           HS 903              [punch divergence]
           QQ b4, SY 64        [punch a final CR]
           SY 11, ---          [punch Stop Code]
           ---
      b0:  t      HEADLESS. 28.4.1772

       t          x        y        teta;
      b1:  t divergence = ;
      b2:  QQ 2.3 + 1.7 + 1.13 + 1.14 + 1.23 + 1.27        [layout {n.d}]
      b3:  QQ 5.3 + 1.7 + 4.13 + 1.9 + 1.14 + 1.23 + 4.27  [layout {-n.dddd}]
      b4:  QQ 1.3 + 1.7 + 2.9 + 1.14 + 1.17 + 2.19 + 1.23  [layout {+n₁₀+d}]
           ---
```

Note that in this program the label c1 must be defined before the central part of the program is read in, (to be precise, before the PA instruction in cell a1-1 is read in).

The punching of a Stop Code at the end of the output tape has simply the effect that, when the tape is read by an off-line typewriter, it will stop automatically at this symbol.

13. UTILITY PROGRAMS; THE HELP SYSTEM.

13.1 A system of utility programs.

While testing programs it is very important to have a system of utility programs, which can assist debugging. In order that these programs can give effective assistance to the operator, it is essential that they are easy to call and control during a run of the program which is being debugged; the HELP system is therefore organized in such a way that it can be operated directly via the HP button and on-line typewriter. Furthermore it is also useful to be able to operate the system according to a pre-determined plan for which purpose it has been made possible to feed the control information via punched tape and through special instructions in the program.

Such a system should be able to perform many different service functions flexibly but at the same time it must not be too complicated to operate: There should not be too many utility programs as otherwise it would become difficult to remember about them all, and likewise the number of different parameters for each routine should not be too large.

One must thus make a compromise between these wishes although there are a certain minimum of facilities that the system should provide. They are:

1) Initialisation of the computer's store.
2) Input and output of programs and data etc. both in symbolic (readable) form, and condensed (binary) form (for optimum speeds of input/output).
3) Amendments to a loaded program.
4) Dumps of store and registers, and changes in the store during a run.
5) Tracing of a running program giving output which indicates the workings of the program on the one hand administratively and on the other hand in its treatment of the data.
6) The possibility for introducing or removing a utility subroutine at any stage of a run, without corrupting the running program.

As will be evident from the following sections, the HELP system generally fulfills these requirements and has furthermore the advantage of being adaptable to the incorporation of new utility programs. In section 13.4, utility programs which constitute the standard system are discussed, and the end of this section is devoted to a description of the way in which the system can be arbitrarily extended; in section 13.5, sources of different errors and the resulting messages are mentioned, together with their treatment. But before this, the central mechanism of the HELP system will be described together with the different ways of entering the system.

13.2 The HELP administrator.

From the programmer's point of view HELP can be considered as a collection of sub-routines which can be entered at an arbitrary stage of a program-run via the HP button or by means of special instruction in the program (programmed entry). Furthermore, HELP will be entered automatically if overflow occurs when using floating-point arithmetic.

The utility programs are controlled by a common administrator (administrative routine), and the loader SLIP may be regarded, in this context, as a sub-routine for the administrator: All input - both of programs and of controlling information - is made via SLIP, and all input just as all activation of utility programs is concluded by an exit via the administrator. When HELP is entered, the exit address is normally set equal to the address of the instruction from which entry took place. Thus, if the exit address has not been changed in the meantime, GIER will continue at the place where the course of the program was interrupted by entry into HELP.

The cost of using the facilities of the HELP system is a restriction in the available storage space, as the system itself must be stored somewhere. In the standard version of GIER with 1 drum, the following parts of the store are used:

Cell 0-9 and cell 1023 for those parts of the administrator which must always be present in the core store.

Track 0 for the basic administrator.

Tracks 1-37 for a number of standard routines, including SLIP and the output routines mentioned in section 12.

Track 38 for working storage, used for instance after operation of HP button.

The remainder of the standard utility routines are normally stored on

tracks 39-57 or on tracks 275-293; in the case of the latter the ALGOL compiler placed on tracks 39-190 can become a part of the HELP system. These utility routines may, in fact, be placed anywhere on the drum (see section 13.4).

Tracks 294-319 are used as Core Store Image in which HELP, on entry, dumps the whole of the contents of the core store and all registers and from which the core store and registers are restored on exit from HELP.

The storage space available for programming is thus cells 10-1023 and tracks 58-293 on the drum, and if absolutely necessary tracks 39-293.

It will be apparent during discussion of the different utility routines below, that, if a program does not use all of the space on the drum, it may be expedient to allocate a few of the tracks after no.57 and a few before the Core Store Image, for use with utility programs.

13.2.1 The HP button.

In chapter 11, the function of the HP button is mentioned very briefly; the effect of pressing the HP button is described here in more detail as follows:

1) Execution of the current instruction is completed, including any drum transfers or peripheral unit functions which have been activated. (If GIER is already stopped, step 2 is taken immediately).

2) Pos.0 of the by register is set = 1, the HP button thereby being set out of function so that depression of the button will have no effect.

3) The contents of cells 0-39 are stored on track 38 and track 0 is transferred to cells 0-39.
4) The contents of the register r1 (the Control Counter) are stored as the address constant in cell 0, and GIER jumps to cell 1.

Assuming that track 0 contains the basic administrator the effect after this is as follows:

5) A check total is made of the locked tracks 0-31. (If this is not correct, "FEJL" is typed; see further in section 13.4 below).
6) CR, "hp-knap" followed by the exit address and CR is typed (in red). An h after the exit address means that the instruction just completed was in a LH half-word, and that GIER, on exit from HELP, will continue with the corresponding RH half-word. Addresses $\geq$ 512 are typed as their negative complements.
7) The contents of the core store and all relevant registers at the time of entry into HELP are dumped to tracks 294-319. The first 10 cells on track 294 will however always be loaded with the standard program necessary for HELP to work correctly.
8) A special bit pattern (GK, VY r) is placed in pos.20-39 of cell 1023 in the core store having the effect that depression of the HP button will not cause the core store and registers to be dumped.

   Pos.0 of the by register is thereafter set = 0, making the HP button once again active.

9) Entry to SLIP is made; this routine then awaits input from the typewriter, acting as if the following block had been read in:

```
b k=294, i=0
i=10
m n s
```

(see also chapter 11).

Remarks: A) Note that there are two different forms of inhibition: The first, by means of by[0] , sets the HP button out of function during execution of steps 3-8 which last altogether approx. $\frac{1}{2}$ sec.; this is necessary to ensure that an untimely depression of the button does not spoil the effects of the storage dump etc. In the same way the HP button is also set out of function for approx. $\frac{1}{2}$ sec. during restoration of the core store immediately before exit from HELP (see below).

The other form of inhibition is active after the core store has been dumped (step 8 above) until the core store is restored on exit from HELP; this "switch" (governed by the contents of cell 1023) prevents the contents of the Core Store Image from being overwritten by depression of the HP button, before restoration of the core store. In addition the exit address stored in cell 9 is not changed so long as the inhibition is active in cell 1023.

B) The dumping and restoration deal with the contents of cells 10-1023 of the core store while cells 0-9 always have fixed contents; cells 0-5 contain the common entry mechanism used by programmed entry to HELP, and cells 6-9 contain the common exit mechanism used in all exits from HELP. The actual code is as follows:

```
→ [0]  IT -1, IT -1     [entry after floating-point overflow]
→ [1]  IT -2, PT 0      [entry for HP patches and programmed
                                                    call of HELP]
→ [2]  GK 1, VY 529     [entry to SLIP input; inhibition of
                                                         button]
  [3]  VK 318, SK 960
  [4]  VK 25, LK 960
  [5]  VK 317, HV 960   [jump to SLIP etc.]
→ [6]  LK 960, VK 0     [common exit from HELP]
  [7]  QQ 0 N           [used by HP patches]
  [8]  VY 0 t511        [release button inhibition]
  [9]  QQ, HV (r)       [exit from HELP]
```

The exit address is placed in cell 9 as an address constant; if entry occurred by means of a LH half-word instruction, the jump instruction is changed to HH (r) *).

C) The HP button has the desired effect if only track 0 is intact. If the contents of tracks 1-31 are also intact SLIP etc. will work properly, otherwise the word FEJL ("error") is typed.

D) If the lamp marked "HP-knap spærret" (HP button inhibited) is not lit and the HP button does not however function, this is usually because GIER is attempting to execute an instruction which can not be completed. Such an instruction may include an undefined basic operation, a drum operation on a non-existent track, a closed chain of indirect addresses or GIER may be waiting for input via a peripheral unit. In each case, pressing the RESET button will enable the HP button to function again.

E) While the button is inhibited, GIER will remember if the HP button has been pressed, and as soon as the inhibition is

---

*) and to assist program manipulation cell 9 is b-marked.

removed, the normal function of the button will be performed. The HELP administrator ensures therefore that if the button is pressed rapidly a number of times after each other it will have the same effect as one depression of the button. Furthermore the administrator ensures that if the HP button is pressed during restoration of the core store prior to exit, HELP will be entered once more but the exit address will remain unchanged (and will not always, as one might have feared, be set to 9 because the removal of the inhibition of the button took place in cell 8).

F) Similarly, the HELP administrator ensures that if one presses the HP button while GIER is within the HELP system (for instance, during input), hardly any damage will be done thanks to the inhibiting bit pattern in cell 1023: The Core Store Image will be unaffected, including the exit address in cell 9, and as usual SLIP will be reset to begin input to cell 10 in the core store (in the Core Store Image, in actual fact). One may however run the risk that some of the information last input will be lost when the HP button is pressed during input: Input is always loaded to the drum either within the Core Store Image or elsewhere; during input it is however collected in the core store and only transferred to the drum when 40 cells have been filled up, or when a definition of i , a block head or a block tail, has been read in. In all other cases depression of the HP button during input will cause that information which has been read in but not yet transferred will be lost; in the worst case the contents of just less than two tracks can be lost (because SLIP buffers information in 2 track-sections of the core store).

G) The registers whose contents are dumped and restored together with the Image are all those of interest to the programmer: the R register, M register, Overflow register, Indicator, p register, s register, by register and tk register.

13.2.2 Exit.

Exit from HELP (or SLIP) occurs either when e followed by an address has been read in, or when more e symbols than b symbols have been read in. When all input has been placed on the drum, the HP-button function is inhibited, after which all registers are restored and the Core Store Image is read back to the core store; GIER jumps to cell 6 in which the last track of the Image is read into the store; after this the button-inhibition is removed (in cell 8) and exit always takes place from cell 9 where the exit address has been placed as the LH address constant, while the RH half-word contains either the instruction HV (r) or HH (r) depending on whether the return jump is to be made to a LH or RH half-cell.

While the effect of the HP button is quite independent of the contents of the core store, and cells 0-9 always acquire the correct contents, programmed and automatic entries into HELP only work correctly if cells 0-5 are intact. After such an entry the contents of cells 6-8 (in actual fact cells 0-8) will be set as prescribed, but cell 9 will only contain the exit address (as address constant).*) Therefore, cell 9 must also be

*) Pos.33 (and pos.41) will possibly be changed to indicate HV or HH operation as appropriate.

intact if an exit from HELP is to function correctly. (The reason for not restoring cell 9 is that certain utility programs insert special instructions here).

Example 13.1.

The message (in red), on the typewriter,

hp-knap 25h

means that the HP button has been pressed while GIER was in the process of executing the LH half-word instruction in cell 25 or a RH jump instruction with final address 25.

Example 13.2.

If one presses the HP button during a run and after this types e CR , the run will continue undisturbed.

13.2.3 Programmed entry.

Programmed entries into HELP may be made in 3 ways of which the first has approximately the same effect as the HP button:

A) The instruction HSF 2 causes the following:

1) The HP button is inhibited by setting by[0] = 1.
2) The message CR "hsf 2" followed by the exit address and CR is typed (in red). Exit will always occur to the next LH half-word or whole word as if it had been made via the normal sub-routine return jump HR s+1.
3) The core store and all relevant registers are dumped to tracks 294-319. Cells 0-8 always acquire the fixed, prescribed contents, and the exit address is placed in cell 9.

4) The bit pattern to inhibit dumping is placed in cell 1023, while the button-inhibition is removed and a jump is made to SLIP, (just as in step 8 and 9 when the HP button is used (section 13.2.1)).

B) The instruction HS 2 causes entry into HELP (or SLIP) without any entry message and without fresh selection of peripheral unit; the functions performed are thus as steps 1, 3 and 4 in A) above with the exception that the by register is undisturbed (apart from pos.0).

Example 13.3.

If the instruction HS(F) 2 is placed in the LH half-word of a cell, the corresponding RH half-word instruction is skipped on exit from HELP. Furthermore it will cause a malfunction of the subroutine mechanism since the address constant in this RH half-word will be placed in the subroutine index register; this is exactly the same effect as if the exit was made by means of the instruction HR s+1.

C) The third way of making programmed entries into HELP is by using the instruction HS 1 followed by the program parameters for a HELP routine. This causes - without an entry message or possibilities for intervention via the typewriter - the following:

1) The HP button is inhibited while the core store and registers are stored on tracks 294-319.
2) Dump inhibition is registered in cell 1023, and the button inhibition is removed.
3) The program parameters after the HS instruction select and control the execution of a HELP routine.

4) Restoration is performed as usual and exit is made to the first cell after the program parameters.

A more detailed description of this is given in section 13.3.3 below.

13.2.4 Floating-point overflow.

If overflow occurs during any of the floating-point operations (ARF, ANF, SRF, SNF, MKF or DKF), GIER makes a jump to cell 0; assuming that cells 0-5 are intact, the following then occurs:

1) The HP button is inhibited.
2) The message CR, "fl.overløb" followed by the address of the instruction which has caused overflow, and CR, is typed (in red). If the offending instruction is a RH half-word instruction, an h is typed after the address.
3) The core store and all relevant registers are dumped to tracks 294-319. Cells 0-8 always acquire the fixed, prescribed contents, and the address typed out previously, is placed in cell 9.
4) Dump inhibition is registered in cell 1023, the button inhibition is removed, and a jump is made to SLIP just as in steps 8 and 9 when the HP button is used (section 13.2.1).

Note that if exit is made from HELP in this situation without changing the exit address, GIER will repeat the instruction which caused overflow; this is usually meaningless because, for instance, the address positions of the accumulator have been changed (cf. description of ARF in the Operation List, Vol.I page 64).

13.3 Activation of HELP routines.

13.3.1 HELP-routine call-line. Control parameters.

Selection of HELP routines is made by means of a HELP-routine call-line which is an auxiliary line (cf. SLIP syntax) of the form

h <name of HELP routine>

when the routine is to be incorporated by the operator using typewriter or paper tape input. It has the form

h <name of HELP routine>/<address constant>

when entry to the routine is programmed (in connection with the instruction HS 1).

A HELP-routine call-line is usually followed by a number of lines with control parameters for the routine in question. The rules for these parameters will be dealt with as each HELP routine is described separately in section 13.4 below, but they all have the format of normal instruction lines or number lines, and are read in the usual way by SLIP.

The control information has exactly the same form regardless of the way in which the HELP routine is selected; but it is emphasized that the representation of individual parameter lines has been selected purely for mnemotechnical reasons and that each line may be written as an arbitrary line of SLIP coding as long as it creates the desired bit pattern on input.

13.3.2 Activation of routine from typewriter or tape.

Once HELP has been entered by means of the HP button, the instruction HSF 2 or floating-point overflow, a HELP routine may be activated by typing

h <HELP-routine name>

Appropriate parameters for the routine in question

e CR

When e CR is typed, the checktotal of the selected routine is verified, and the routine is then executed straight away; since a HELP-routine call-line has approximately the same syntactical level as a drum-block head, e CR causes this dummy block to be abolished leaving HELP in exactly the same situation as before the call-line: The parameters just read in are not loaded anywhere (at any rate, they are inaccessible to the programmer), and the serial address and track no. are unchanged.

Activation via paper tape: If one does not wish to type the control information, one may read a tape with the same format but concluded by s (or possibly with e CR): After entry into HELP as above, l is typed after which the tape is read in; the routine is executed, and the concluding s transfers control back to the typewriter whereby HELP is in the same situation as after typing e CR , above.

Example 13.4.

After the HP button has been pressed, a print-out of cells 100-108 in instruction format is required. When the message, for instance,

hp-knap -501

has been typed in red by HELP, indicating that exit will occur to cell 523, one should type

```
┌ h tryk         [select gen.output routine (tryk ≡ output)]
│ gp 100 t 108   [initialises "tryk" for output of cells
│                                                  100-108]
└ e CR           [starts execution of routine]
```

When this output has been made, the situation is again as it was before the HP button was pressed and if one types a further e CR , GIER will resume the original program at cell 523 as if nothing had happened.

If a tape was placed in the reader, with the following information

```
h tryk
gp 100 t 108
e
e CR
```

the typing of 1 would have exactly the same effect as the typed information above.

13.3.3 Programmed activation of a HELP routine.

If one wishes to interrupt the course of a program at certain places in order to take measures which are dependent on the progress of a run, one should put in the instruction HSF 2 in these places; each time GIER reaches this instruction, one will have an opportunity to direct events from the typewriter.

If, on the other hand, one wishes to take predetermined measures at a particular place in a program, one might just as well insert the control information in the appropriate part of the program, in the following manner:

```
---
HS 1   [execute the HELP routine which follows]
h <HELP-routine name>/<no. of prog. parameters>
Appropriate control parameters for the routine in question
---
```

<no. of prog. parameter> means, here, the number of cells (that is, lines, usually), which the control parameters following, occupy; the control parameters have exactly the same format as in section 13.3.2 above.

However one should not write <u>e</u> CR as a termination of the control information, as this type of HELP-routine call-line with stroke is not on the same level as a block head.

Everything read in is loaded as standard SLIP information, of which the call-line is "translated" into one cell of information *). Specification of the number of program parameters enables HELP to assemble the exit to the first cell after the control parameters.

<u>Example 13.5.</u>

If, after pressing the HP button, one types

```
i = 150
hs 1
h tryk /1
gp 100 t 108
ar 45
```

it will be read in and stored in cells 150-153. When GIER during execution of the program reaches cell 150, the contents of cells 100-108 will be printed out, after which GIER will continue in cell 153.

If, on the other hand, one types

```
i = 150
hs 1
h tryk
gp 100 t 108
e
ar 45
```

HELP will immediately print out the contents of cells 100 - 108 while the instructions hs 1 and ar 45 will be stored

---

*) The line is loaded as the half-word instructions NC, QQ where the address constant for NC is the sum of the values of the letters in the HELP-routine name, while the address constant for QQ is the number of program parameters. The part of the line which follows the stroke is read in as a normal increment.

in cells 150-151 (and if GIER attempts to execute the instruction in cell 150, an error message will be typed because the next cell does not have the expected contents, namely an NC instruction etc.)

13.3.4 Corrections.

As it must be evident from the above, one does not need a special utility program to make corrections to the core store, since SLIP information can be read directly via typewriter or punched tape into the core store (actually, the Core Store Image). It should be remembered that by using the control lines

x <track address> and z <track address>

one can re-establish the labels used during input with the same values, which can be a great help when reading-in corrections (cf. example 11.38).

However corrections to the contents of registers can only be accomplished (in a reasonable way) using the HELP routine "ret" (Danish for "correct"), which is described below.

If, at any time during the selection or execution of a HELP routine, one regrets taking this measure, one may depress the HP button. As soon as the instruction in progress is completed, the usual buttonwise entry to HELP is made, but, thanks to the dump-inhibition pattern in cell 1023, the Core Store Image (and thereby the exit address) remain unaffected since the previous entry *) In this way the HELP routine originally selected is

---

*) This is not applicable to the use of the HP button during execution of a tracer program, since once this program is started it works outside the HELP system, and the dump-inhibition is therefore also removed.

forgotten completely and SLIP is once more in a position to accept information from the typewriter to cell 10 or for selection of a new HELP routine.

Example 13.6.

After entering HELP, by one way or another, one has typed, for instance,

```
h tryk
gp 100 t 405
e
```

after which HELP begins printing out the contents of 306 cells on the typewriter. This is perhaps not what was wanted so as soon as the mistake is realized, one may press the HP button, after which the print-out is stopped, and HELP is again ready to accept other information as if "tryk" had not been selected. One could then type the correct control information, for instance

```
h tryk
gp 400 t 405
e
```

which causes a print-out of the contents of 6 cells.

Example 13.7.

If one starts to type the same information as in example 13.6 but realizes the mistake before e has been typed, i.e. after

```
h tryk
gp 100 t 405
```

has been read in, one has 2 possibilities (besides that of activating the "tryk" routine with the undesired parameter):

One may press the HP button and thereafter type an arbitrary character (to release GIER from the LY instruction for the completion of which it is waiting), causing a return to the "ready" situation before "tryk" was selected.

One may instead, however, correct the information just read in by concluding the instruction line with a CR and then typing

```
i = i-1          [set the serial address back]
gp 400 t 405     [the correct parameters]
e
```

after which the HELP routine will be executed.

13.4 Standard HELP routines.

13.4.1 Location of HELP routines.

As mentioned previously HELP normally occupies tracks 0-57, but one may if necessary use tracks 39-57 for other purposes either by doing without the routines which are normally placed on these tracks or by placing them elsewhere on the drum.

The permanent part of HELP, tracks 0-37 includes the HELP administrator, SLIP, routines for output of instructions, numbers and text. (The 2 last-named are described in section 12), and the HELP routines h start, h kontrol and h slip which are described below.

The remainder of the standard version of HELP consists of the following utility programs. (The usual locations on the drum are also given).

h tryk and h sam	occupy	9	tracks,	usually	located	on	tracks	39-47
h kompud	-	4	-	-	-	-	-	48-51
h ret	-	1	-	-	-	-	-	52
h hent and h gem	-	1	-	-	-	-	-	53
h hp ind and h hp ud	-	3	-	-	-	-	-	54-56
h læs hp	-	1	-	-	-	-	-	57

In consideration of the requirements of the standard version of the ALGOL compiler which is usually loaded to tracks 39-160, a compatible version of HELP is available in which the above-named routines are placed on tracks 191-209 (in the same order), while the permanent part is still loaded to tracks 0-37. During compilation of ALGOL program tracks 191-209 will however be overwritten. All these routines can be loaded anywhere on the drum and not necessarily in continuation of each other; each routine is available on a separate tape and can be read to an arbitrary drum track by typing c <start track>CR and l (see also below). In the same way, additional HELP routines may be read in in unlimited quantities overwriting, if desired, routines already loaded - the routine "læs hp" ("read hp") should however be intact, as it is used during the input.

All the time the HELP system is in use, a catalogue is kept of routines read in and their location. Every time a HELP routine is called, an investigation is made to see if the name exists in the catalogue; if the search proves fruitless an error message "tomt hp" (meaning "empty hp") is written, after which HELP is again ready for input. If the name is found, the check total is verified and if it is in error "sum-fejl" (meaning "checktotal error") is typed, after which HELP is yet again ready for input (see also section 13.5).

There are a number of points, regarding the catalogue of HELP routines, which should be noted:

1) HELP does not ascertain whether a new HELP routine is read in to a track already used. It will simply extend the catalogue with the data for the new routine. It is only when

one tries to call the routine read in previously that HELP will make a protest because of the erroneous check total. Thus, if one wishes to read in new HELP routines on top of routines read in earlier, everything will go smoothly until one tries to use the old routines.

2) If one loads a new routine with the same name as an existing routine, the new routine will be loaded to the required location and the catalogue will be amended so that it corresponds to the new routine's location and check total.

3) The catalogue may be reported (with verification, of check totals) at any time using the "kontrol" routine but it can only be initialised by the "start" routine (which includes "kontrol"). "start" deletes all non-standard routines from the catalogue and verifies the check totals for the standard routines (see below).

13.4.2 "kontrol" (check) and "start".

1) As mentioned above one can check which HELP routines have been read in to the unlocked part of the drum, using the routine called "kontrol". This routine can be used in two ways, namely in the presence or absence of a parameter, the actual form of which is quite arbitrary, the effect being:

Without parameter: Check totals for every HELP routine listed in the catalogue are verified, and a list of the starting track and name of each routine is typed out; in the case of a non-standard HELP routine only the starting track no. is typed out.

If a check total does not agree, the message "sumfejl" ("check total error") is typed out.

With parameter, for instance, QQ: The same check is performed but messages are only typed where the check total does not agree.

The first item in the catalogue concerns the permanent part of HELP on the unlocked tracks 32-37. For programming reasons, tracks 28-37 are however treated as one HELP routine with the name "uaflåsede" (unlocked), (this name should not be changed by the user) and the first line in the typed list is therefore "28 uaflåsede". If all the standard HELP routines are located in their usual positions and the catalogue has not been extended, the list obtained will be as follows:

```
28  uaflåsede
43  sam
39  tryk
48  kompud
57  læs hp
53  hent og gem
54  hp ind
55  hp ud
52  ret
sumfejl 45 algol
```

The ALGOL compiler is listed in the catalogue as a HELP routine on level with the remainder, but since it shares tracks with some of them, it cannot be intact at the same time as the HELP routines (in this version); the message "sumfejl 45 algol" is therefore obtained *).

---

*) HELP verifies, in fact, only track 45 containing the basic entry to the compiler. The remainder is checked by the compiler itself when it is called.

If the standard HELP routines are loaded to tracks 275-293 and the part of the ALGOL compiler on track 45 is intact, the list obtained will now be as follows:

```
28  uaflaasede
279 sam
275 tryk
284 kompud
293 læs hp
289 hent og gem
291 hp ind
290 hp ud
288 ret
45  algol
```

2) "start" is an <u>initialising</u> program, which sets GIER's store and registers in a well-defined state before commencement of new jobs. "start" may be used in two ways, namely in the presence or absence of a parameter, the actual form of which is quite arbitrary, the effect being:

Without parameter: The available part of the store (i.e. cells 10-1022 and tracks 58-319) is filled with the instruction HSF 2 in each word; all registers are set to zero (excepting the by register and r1 register which are set to 17 and 10, respectively; the HELP routine catalogue is initialised, i.e. all non-standard routines are deleted and the standard routines are catalogued as being on tracks 39-57; the list of HP patches is destroyed (see the section on "hp ind"). After this the routine "kontrol" is executed.

With parameter, for instance, QQ: The same effect as for "start" without parameter except that the catalogue check message reports only those routines for which the check total does not agree.

Note that if one of the standard HELP routines is corrupted, "start" cannot restore it but will instead type the message "sumfejl" and the start track no. and name.

The whole message is thus as shown at the beginning of this section, with "sumfejl", where relevant, preceding one or more lines.

Example 13.8.

Let us assume that while testing a program, the routines "kompud" and "ret" are not required whereas a tracer program called "hop" and filling 5 tracks is needed as a HELP routine.
After one has typed

h start
e

and ensured that the standard version of HELP is intact, one should place the tape containing the routine "hop" in the reader, and thereupon type

c 48
l

The routine will be loaded to tracks 48-52, and can be used in precisely the same way as the other HELP routines. On the other hand, attempts to use "kompud" or "ret" will cause error messages, because the check total does not agree.
If one later makes a call of "kontrol" by typing

h kontrol
e

the following report will be obtained:

```
28  uaflaasede
43  sam
39  tryk
sumfejl 48 kompud
57  læs hp
53  hent og gem
54  hp ind
55  hp ud
sumfejl 52 ret
sumfejl 45 algol
48
```

13.4.3 "tryk" (output).

The HELP routine "tryk" can be used to print (punch) the contents of arbitrary parts of the store and registers in many different ways, and the format of the control parameters is rather complicated; for instance, several parameters are packed together in one cell. Thus "tryk" can be used to print the contents of the store and registers as instructions (again, in different ways), and as decimal numbers, in 4 different ways, (with many more options) but not in binary form (this is catered for by "kompud"), octal form or any such form.

The parameters are written in the form of instructions; there are 3 distinct types of parameters: the general output delimiters (1 whole-cell), the basic trim (1 whole-cell) and the number editing trim (2 whole-cells); these types are described in more detail below. "tryk" can be activated with an unlimited number of parameters, so that one can obtain a print-out of several parts of the store in different ways, if required; there should however be at least one set of general output delimiters (describing what is to be output).

A) <u>The general output delimiters</u> may consist of two half-word instructions, or one whole-word instruction having one of the following forms

<output format>,<name of register>
<output format><start address>t<end address>
<output format><track no.>.39+<start address>t<end address>

where <output format> and <name of register> are SLIP operation codes, and the other quantities are address constants.

Each set of general output delimiters begins then with an operation code indicating the format of the output i.e. whether the output is to be in the form of instructions, fixed or floating point numbers, integers or packed integers (within each group the output format can be further defined by means of the base trim and the number-editing trim). The meaning of the LH operation is shown in the table below.

<output format>	The contents of cells to be output as
gp	instructions
gpr	instructions, with indication (in square brackets) of the equivalent absolute adress where the address is relative
gr	floating pt.numbers, with 5 significant digits
grp	– – – , – 10 – –
grn	– – – , layout and scale factor as indicated in the number-editing trim *)
gm	fixed pt.numbers, with 6 decimals
gmp	– – – , – 12 –
gmn	– – – , layout and scale factor as indicated in the number-editing trim *)
gi	integers (units in pos.39), with 13 digits
gin	– (– – –), layout and scale factor as indicated in the number-editing trim *)
ga	packed integers, i.e. 4 integers from each cell in the range $0 \leq t \leq 1023$

*) See section C) below.

When numbers are output the marking of the respective cells (or the R register) is indicated as a, b, c or a comma, so that (punched) output can be read in again by SLIP.

The remainder of a set of general output delimiters indicates what is to be output, and each set may be concerned with the contents of one of the following: registers, the core store or the drum.

Output from registers is indicated by general output delimiters of the form

<output format>,<name of register>

or

,<name of register>

where <name of register> can be one of the operation codes shown in the table on the opposite page. If <output format> is omitted, the output format previously used is retained.

Output from the core store is obtained using general output delimiters of the form

<output format><start address>t<end address>

which means that cells from <start address> to <end address> inclusive, are to be output in the form indicated. If the end address is omitted (or set equal to zero) only one cell is output.

Output from the drum is obtained using general output delimiters of the form

<output format><k>.39+<b>t<s>

which means that cells from no.<b> to no.<s> on track no.<k> are to be output in the form indicated. The addresses <b> and <s> may be > 39 since then these are simply regarded as being on

<name of register>	Output from	Output format
gr grf gm	R register (incl.marker-bits) RF register (- - -) M register	dependent on <output format>.
gs gp ga gk	s register p register exit address ("r1 register") tk register	integers, $-512 \leq i \leq 511$. Independent of <output format>.
gi	indicator reg. incl. KA and KB and overflow register	1) pos.0-9 of indicator as integer, $-512 \leq i \leq 511$ 2) - 0-9 - - - 5 2-bit groups *) 3) KA, KB as one 2-bit group *) 4) overflow indicated by the letter O otherwise space.
sy	by register	1) as integer, $-512 \leq i \leq 511$ 2) positions of the bits equal to 1
cl	the cell in which the user's program will continue (cell no.[exit address]).	dependent on <output format>.
ar arf	all of the above excluding RF all of the above, R and M being treated as RF	only the treatment of R and M are dependent on <output format>.

*) Each 2-bit group is output as one symbol where point indicates 00
a - 10
b - 01
c - 11

neighbouring tracks, but both <b> and <s> must be within the range $0 \leq t \leq 1023$. If <s> is omitted (or set equal to 0), only cell <b> on track <k> is output. (NB. If a reference address $\neq$ 0 has been used, the output will begin with cell no.[<b>-<reference address>] on track <k>, derived from the fact that if the first cell of the track <k> corresponds to cell no.<reference address>, <b> becomes the address in the core store of cell <b> on track <k>; see also item B), the basic trim.)

B) <u>The basic trim</u> which is valid for all succeeding general output delimiters until a new basic trim is given has the form

bt<s><integer>.39+<output unit code>t<reference address>

The operation code bt can be regarded as the mnemonic for <u>b</u>asic <u>t</u>rim while the remaining constituents have the following effect:

If <s> is empty, each line of the output is preceded by the address in the core store, in square brackets. If <s> is a clearing flag, these addresses are omitted from the output.

<integer> indicates how many numbers are to be printed on one line. If <integer> is equal to 0 or the term <integer>.39 is omitted completely, the previous value (initially 3) is used. This term has no effect on output in the form of instructions since this is always with one whole-word per line.

<output unit code> selects the output unit(s): 1.5 for typewriter, 1.4 for punch and 3.5 (or 1.4 + 1.5) for both units. If <output unit code> is omitted the medium last used is selected; this is initially the typewriter.

<reference address> is an integer and should normally only be used for output from the drum. It indicates the core store address corresponding to the first cell on the first track (similar to the serial address in a drum-block head when a program is read in); after this the start and end addresses in the general output delimiters will refer to addresses in the core store, and <u>not</u> to the cells on the drum track. If <reference address> is omitted, it will be set equal to 0 *).

If none of these special facilities are required, the basic trim can be omitted completely in which case the <u>initial trim</u>

bt 3.39 + 1.5 t0

is effectual, so that 3 numbers/line are output preceded by addresses; the typewriter is selected as output unit and the reference address is 0 which means that during output from the drum, addresses refer "directly" to the cells on a track.

C) <u>The number-editing trim</u> governs all the succeeding sets of general output delimiters which begin with grn, gmn or gin until a new number-editing trim is given. It has the form of 2 whole-word instructions

---

*) The reference address is used to find the required cells address on the drum. It is in fact cell no. [core store address - <reference address>] on the first track (this number may well be > 39).

During output from the core store, the reference address is also subtracted from the start and end addresses, and a reference address ≠ 0 will therefore cause a "displacement" of the output from the core store.

nt <power of 2>t<power of 10>

<layout>

The operation code nt can be regarded as the mnemonic for number-editing trim while the remaining constituents have the following effect:

<power of 2> is an integer which causes the contents of a cell in the store or a register to be multiplied by the power of 2 indicated, before it is edited. If <power of 2> is omitted this corresponds to a multiplication factor of 2↑0.

<power of 10> is an integer causing multiplication by a power of 10 before editing. If <power of 10> is omitted this corresponds to a multiplication factor of 10↑0.

<layout> is an instruction line consisting of QQ followed by as many as 12 scaled integers. The layout determines the local typography of each number (number of significant digits, decimals, exponent digits etc., etc.) after exactly the same rules as for the general routine for editing of numbers, described in section 12.2, especially 12.2.3, above (it is in fact the same routine that is used by "tryk").

Example 13.9.

After entry into HELP a print-out of the contents of the register, R and M is required, in the form of fixed pt. numbers with 12 decimals, and a print-out of the s-register. One may type

```
h tryk
gmp , gr
gmp , gm
,gs
e
```

after which the following will be typed, for instance,

```
[ R]  0.123 456 789 101a
[ M] -0.345 000 678 321
[ s] 345
```

(After this GIER will again **wait** for typewriter input).

Example 13.10.

After entry into HELP a print-out is wanted of the contents of all registers (and incidentally the cell to which HELP will exit), where RF contains a floating pt. number. One may type

```
h tryk
grf , arf
e
```

after which the following will be typed, for instance,

```
97 ar r-5 ,gr 34
 1.2346₁₀7a
345    10 61 82=.bb.a . 0 -239= .1.5.9
```

The arrangement in this case **is** as follows:

<exit address><contents of cell>

<RF register>

<s reg.><p reg.><tk reg.><indicator><overflow reg.><by reg.>

Thus in the example s = 345, p = 10 and tk = 61. The indicator (pos.0-9) contains the bit pattern 00 01 01 00 10 corresponding to the number 82, and the succeeding point indicates that KA = KB = 0. The letter O indicates, that overflow has been registered, and finally by = -239, i.e. positions 0, 1, 5 and 9 are equal to 1 (but pos.0 is never shown in the output).

Example 13.11.

If one wishes to have a print-out of the program in cells 10-12 one may (after entering HELP) type

```
h tryk
gpr 10 t 12
e
```

after which the following will be typed, for instance,

```
[10] arf r+7 [17] t1
[11] mkf  94     , dkf s-3
[12] hv r+31 [43] NT
```

If the output is to be punched and without addresses before each line, one would need to include a basic trim by typing

```
h tryk
bt s 1.4
gpr 10 t 12
e
```

It is important here that one types the basic trim before the general output delimiters, as it will otherwise have no effect.

Example 13.12.

One wishes to make a print-out of the floating pt. numbers stored on tracks 110-114 with a layout {-nddd.000}; the output is to be made via the punch with 8 numbers per line and without addresses. After entry to HELP the following should be typed

```
h tryk
bt s 8.39 + 1.4              [no addr.; 8 no./line; punch]
nt                                     [no scale factor]
qq 4.3+4.7+3.13+1.9+1.14+4.23+3.27              [layout]
grn 110.39 + 0 t 199 [200 cells from tracks 110 onwards]
e
```

(The symbols +0 in the last parameter could just as well have been omitted).

13.4.4 "kompud" (condensed dump).

The HELP routine "kompud" makes dumps of specified parts of the store, in a condensed form punched on tape, so that it can be read by SLIP at optimum speed. The punched tape is supplied with all necessary control information and can be re-read by typing 1 after a call of HELP. What happens on conclusion of the re-input is dependent on some of the parameters given to "kompud" (see sub-section B below).

"kompud" requires as parameters one or more cells of packed integers to specify the required sections of the store to be dumped, to specify the treatment of "undumped" parts of the store on re-input and to specify the required terminating action on re-input. These specifications are made using 2 types of parameters:

A) Output delimiters. These are packed integers of the form

<first track>/<first cell>/<last track>/<last cell><mark>

where <mark> is either a, b or nothing while the remaining quantities are integers. These parameters causes dumping of a section of the store according to the following rules:

<first track> is the number of the first track; if <first track> is omitted (or equal to 0), it means that a section of the core store is to be dumped.

<first cell> in the number of the first cell to be dumped in relation to cell 0 on the first track; if <first track> = 0, <first cell> is the address of the first cell in the core store to be dumped.

<last track> is the number of the last track; if <last track> is omitted (or equal to 0), it means that the last track is the same as the first track. If <first track> = 0, <last track> must also = 0 and dumping is made from the core store.

<last cell> is the number of the last cell to be dumped in relation to cell 0 on the last track; when dumping from the core store <last cell> is the address of last cell in the store, to be dumped.

The quantities <start track> and <last track> must be within the range $1 \leq t \leq 319$ (and track nos. ≥ 294 refer naturally to the core store image. The expressions first track and last track must be regarded to some extent as reference addresses only, in that if the quantities <first cell> or <last cell> are > 39, the routine will calculate internally the actual address of the cell on a succeeding track. Cell numbers should, however, be within the range $0 \leq c \leq 1023$ (although in fact, numbers outside this range will be taken modulo 1024; cf. input of packed integers, section 11.5.1).

If the section of the store in question includes cells which contain the instruction HSF 2 (cf. the HELP routine "start", which fills the whole store with HSF 2), these are skipped during the dump; by <u>marking</u> the packed integers which constitute the output delimiters, it is possible to indicate the two ways in which sections of the store, so skipped, are to be treated on re-input:

<mark> = empty causes SLIP to fill up with HSF 2 all cells within the section of the store that was skipped during the dump. The section of store will thus be regenerated completely.

<mark> = a causes SLIP to skip all cells (within the section of the store) which were skipped during dumping.

The last possibility

<mark> = b causes the section of store in question to be ignored during dumping and, on re-input, to be filled with HSF 2 .

Note that

<mark> = c indicates another type of parameter (see B below).

It may often be useful to specify a section of store in terms of symbolic addresses, but this is not possible when using packed integers; it may thus be convenient to indicate the output parameters in the form of an instruction with 4 scaled terms i.e.

qq <first track>.9+<first cell>.19<last track>.29+<last cell>.39

One should however remember that marking in this case must be indicated by half-word marking or F marking.

An unlimited number of sections of store can be specified by means of a cell of packed integers (or an instruction) for each of them; certain general cases of dumps covered in B) below, require no output delimiters.

B) Exit specification and general dumps. This parameter is a c-marked cell with packed integers.

<u addr.>/<e addr.>/<exit code>/<dump code> c

There can only be one parameter of this type and by definition it must be c-marked.

The first 2 quantities may be arbitrary integers, <exit code> consists of max. 3 digits, each digit being 0 or 1, and <dump code> may be either 0, 1 or 2 (and under special circumstances 3 or 4 - see example 13.15). The effect of these parameters is as follows:

Re-input of a condensed tape is always terminated by a jump to SLIP, which continues reading tape in the usual way (the values of i and k are discussed below); the digits in the <exit code> causes the tape to be punched with one or more of the conventional SLIP control lines, as follows:

<exit code> = 100: The condensed tape is to be terminated with the control line u<u addr.>. On re-input the exit address from HELP is thus set equal to <u addr.> (see chapter 11), after which input from tape continues via SLIP.

<exit code> = 10: The condensed tape is to be terminated with s , so that SLIP waits for typewriter input after re-input.

<exit code> = 1: The condensed tape is to be terminated with the control line e<e addr.>, so that GIER jumps immediately to cell [<e addr.>] after re-input.

If <exit code> is set to 11, 101, 110 or 111 the above-named effects are combined and the control lines are punched in the same order as the digits are written, i.e. first u<u addr.>, then s and finally e<e addr.> .

If <exit code> is set to 0 or 100 the condensed tape can not be used alone for re-input, as it not concluded "properly" - the tape reader will (literally) "cry out for more tape". Unless the punched tape is supplied with additional codes in some way or another, one of the other alternatives should be selected.

The last quantity <dump code> is used to specify whether the dump parameters are explicit or specific:

<dump code> = 0 means that the dump is to be governed by parameters of the type mentioned in A) above.

<dump code> = 1 indicates that all registers and the whole of the core store are to be dumped; cells containing HSF 2 at dump time will be regenerated with HSF 2 on re-input.

<dump code> = 2 indicates that all registers, the available part of the drum, i.e. tracks 58-293, and the whole of the core store are to be dumped; cells containing HSF 2 at dump time will be regenerated with HSF 2 on re-input.

Since the Core Store Image extends as far as cell 23 on track 319 and the contents of all registers are stored on cells 24-33 of this track, the specification

<dump code> = 1 corresponds to the output delimiters 294//319/33

<dump code> = 2 corresponds to the output delimiters 58//319/33.

Example 13.13.

If one requires a condensed dump of the core store, so that after re-input, GIER will jump immediately to cell 10, one may type

```
h kompud
10/1/1 c
e
```

Example 13.14.

If one requires a condensed dump of cells 10-94, cells 300-451 and tracks 100-114, so that after re-input, GIER will wait for typewriter input ready to jump to cell 305, one may type

```
h kompud
10//94
300//451
100//114/39
305//110/c
e
```

When the tape so produced is re-input (by simply typing l), the exit address is set equal to 305, and GIER then awaits typewriter input. This will give one time to, for instance, set a data tape in the reader before one types e to start execution of the program.

The order in which the parameters are presented to "kompud" is quite arbitrary and in the above instance one could just as well have typed

```
h kompud
300//451
305//110/c
100//114/39
10//94
e
```

for instance, the only difference being the order on the tape produced: The dumps are made in the same order as the parameters are given.

The serial address and track number when a condensed tape has been read in. Let i_0 , k_0 be the values of the serial address and track number in the last drumblock head before the condensed tape is read in; let i_1 , k_1 be the current values

just before reading of the condensed tape starts. Then, if i, k are the values just after the loading is finished, the following rule holds: If $40k + i \geq 40k_o - i_o$, then the serial address and track number keep their current values i, k; otherwise they are set "back" to the values i_1, k_1 which were effective just before input of the dump.

In other words, this rule means that, if input terminates <u>within</u> the drum block which was entered before input was started, one may read program in continuation of the dump, without further ado. But if input terminates <u>outside</u> (that is to say, before) this drum block, i and k will be set back as if the condensed input had not appeared. For this purpose the core store is regarded as a drum block with $k = 294$ and $i = 0$.

Immediately prior to entry into HELP-SLIP a block is established with $k = 294$ and $i = 0$. Input of condensed tapes dumped from the core store will in this situation always terminate within this block, and input of sections of the drum other than the Core Store Image will terminate outside this block. Thus, input to the core store can always be made in automatic continuation of re-input of core store dumps.

<u>The format of the condensed tape</u>. The condensed tape which is produced by "kompud" is made up in the following way:
1) an <u>introduction</u> consisting of a start combination, definition of the starting addresses i.e. serial address and track number, 47 space symbols and an end combination; 2) the <u>main part</u> consisting of a number of blocks, each consisting of a word indicating the number of cells, to be skipped and the contents of a number of consecutive cells (each word consisting of 6 charac-

ters); 3) a check total and the numbers of characters on the condensed tape; 4) a termination, if requested by means of the exit specification mentioned in B).

The reason why the introduction contains so many spaces (which are dummy symbols here) is that by starting reading at any of these spaces, the dump will be re-input to the instantaneous values of the serial track number and address. The loading will thus be "linearly displaced" in the store because all the addresses in the condensed tape are relative to the start address. If, for instance, one has made a dump of cells 100-155 and wish to read them in to cells 248-303, one should simply define the serial address i = 248 via the typewriter and commence input of the tape after the first ten characters.

Example 13.15.

It is possible to punch a condensed tape without any introduction: if the first parameter for "kompud" is 3c , the starting address (and the 47 spaces) are omitted from the condensed tape.

If, for instance, one wishes to prepare a library routine, which is loaded to cells 850-899, so that it can be read anywhere in the store, one should type

```
h kompud
3c
850//889
10/c
e
```

The tape so punched has therefore no introduction and is terminated with s ; it will be read to the instantaneous serial address and on completion of input GIER will wait for typewriter input.

If one wishes to be able to dump and re-input cells containing HSF 2 exactly like any other cells (due to the

requirements of the GIER ALGOL "gierproc" routine, for instance), one may include the parameter 4c as the first parameter (if both 3c and 4c are to be used simultaneously 3c comes first).

13.4.5 "gem" (preserve), "hent" (retrieve) and "sam" (compare).

"gem".

Activation of "gem" causes re-dumping of the Core Store Image (including contents of registers) - 26 tracks in all - to a section of the drum store and since the Core Store Image is identical with the core store at the time HELP is called, "gem" does in fact "preserve" the contents of the core store etc.

"gem" may be activated without parameters or with any number of parameters of the form

sk <b>,

where b is an integer in the range $39 \leq b \leq 319$. The effects are as follows:

Without parameter: The whole of the Core Store Image (tracks 294-319) is copied to tracks 268-293, i.e. the last 26 free tracks on the drum.

sk <b> The Core Store Image is copied to track <b> and the 25 tracks thereafter. If <b> = 0 (or <b> is omitted) the Image will be copied to tracks 268-293.

If one wishes to copy only a part of the Image or to copy different parts on different sections of the drum, this can be achieved using "hent", which is described below.

"hent".

Whereas "gem" always copies from the Core Store Image to other sections of the drum, "hent" has been designed to perform transfers in the opposite direction; the parameters are, however, so flexible that it is possible to perform any drum-to-drum transfers with this routine and the facilities lacking in the "gem" routine are obtainable in "hent" using appropriate parameters.

"hent" may be activated without parameters or with any number of parameters of the form

sk<a>.39 + <b>t<c>

where <a> , <b> and <c> are integers within the ranges $0 \leq a \leq 319$, $39 \leq b \leq 319$ and $39 \leq c \leq 319$. The effects are as follows:

Without parameters: The contents of the 26 tracks 268-293 are copied to the Core Store Image, being precisely the reverse of "gem" without parameters.

sk<a>.39 + <b>t<c>: The contents of <a> tracks are transferred from track <b> onwards to track <c> onwards. As exceptions, <a> = 0 has the same effect as <a> = 26 ; i.e. 26 tracks are transferred; <b> = 0 has the same effect as <b> = 268 ; i.e. copying is made from the "Preserved Image" just in front of the Core Store Image; <c> = 0 has the same effect as <c> = 294 , i.e. transference is made to the Core Store Image.

If more than one set of parameters are given, transference takes place in the order in which the parameters are presented.

"sam".

"sam" compares two drum sections (each of 26 tracks) cell by cell, and outputs the deviations in a selected format.

The routine requires 2 types of parameters, namely the section delimiters of the form

sk <a>t<b>

and the output format which has the same form as the output delimiters and trims for the HELP routine "tryk". Their effect is as follows:

A) Section delimiters.

sk <a>t<b> : "sam" compares the two 26-track sections of the drum which begin with track <a> and track <b> , respectively. As exceptions, <a> = 0 has the same effect as <a> = 294 (i.e. this section is the Core Store Image), while <b> = 0 has the same effect as <b> = 268 (i.e. this section is the "Preserved Image" immediately preceding the Core Store Image).

If this parameter is omitted, the sections compared are the Core Store Image and the section immediately preceding it (i.e. the same effect as sk 0 t 0).

B) Output format. For each set of section delimiters there must be an output format consisting of one or more sets of parameters formed after the same rules as for "tryk", section 13.4.3, with

the following exceptions: a) all addresses refer to the relative positions of cells with an image of the core store, and therefore reference to drum tracks has no effect, i.e. any indication of reference address or track numbers is ignored; b) if a basic trim is included in the parameters, the indication of the number of words per line is ineffectual since the output always has the following form *):

	<a>	<b>
<address>	<contents of cell>	<contents of cell>
<address>	<contents of cell>	<contents of cell>
......	......	......

where the situation of the third column is dependent on the positioning of the tab stop since "sam" uses the tab instruction SY 30 between the second and third column (and only here).

If one forgets to specify the output delimiters, "sam" has no effect.

One may specify several consecutive comparisons between pairs of sections on the drum using parameters of type A) sk <a>t<b> , but a complete output format must follow each parameter of this type.

Example 13.16.

A typical usage of the three routines "gem", "hent" and "sam" might be as follows:

1) When a program (which does not use tracks 268-293) has been loaded to the core store,

---

*) Addresses are output even though a basic trim with s flag is used.

```
h gem
e
```

is typed, after which the required run of the program is made.

2) Whatever happens during a run, good or bad, the original situation can be re-established by pressing the HP button and typing:

```
h hent
e
```

3) If, however, one concludes a run in one way or another and thereafter, for instance, wishes to find out what, if anything, has been changed of the instructions in cells 10-150 and 600-1022, and of the working locations in cells 151-559, one may press the HP button and type:

```
h sam
bt 1.4
gp 10 t 150
gp 600 t 1022
gr 151 t 559
e
```

Since the section delimiter has been omitted, the Core Store Image (i.e. the situation _after_ the run) is compared with the "Preserved Image" (i.e. the situation _before_ the run); the basic trim causes output to be made via punch, and the last parameters cause diverging cells in section 10-150 and 600-1022 to be output as instructions and in cells 151-599 in the form of floating pt. nos. with 5 significant digits.

13.4.6 "ret" (correction).

After any normal entry into SLIP-HELP one may type directly anything whatsoever into any cell in the store whatever by first defining i and k appropriately and thereafter typing the required contents. If, however, one wishes to _change the contents of one of the registers_, it must be done using "ret". This routine which requires as control information, the name of

the register to be changed followed by the desired contents, must be supplied by a pair of parameters of the form

<name of register><overflow code>

<SLIP line>

where <name of register> may be, as for the HELP routine "tryk",

gr	for the R register,	grf	for the RF register,
gm	– – M – ,	gs	– – s – ,
gp	– – p – ,	gk	– – track – ,
gi	– – indicator register and		
sy	– – by register		

<overflow code> may only be used when correcting the R register. It controls the flow of information partly to R and partly to the overflow register:

<overflow code> = 0 or nothing: After the specified number (or program constant) is read in, R00 is set = Rpos[0] and the overflow register is cleared.

<overflow code> = 1: After the specified number is read in, R00 is set = ¬ Rpos[0] and the overflow register is cleared.

<overflow code> = 10: After input R00 is set = Rpos[0] and the overflow register is set to 1.

<overflow code> = 11: After input R00 is set = ¬ Rpos[0] and the overflow register is set to 1.

The two most applicable possibilities are thus <u>gr</u> for input to R <u>without overflow</u>, and <u>gr 11</u> for input to R of something <u>with overflow</u>. <SLIP line> may be an instruction line, or a number line possibly preceded by a control code for number-input (<u>f</u> or <u>m</u>):

The contents of the registers R and M may be changed by input of an instruction line or a number which may be either a fixed pt. number, packed integers or a floating pt. number (stored in a cell).

The RF register may only be set equal to a floating point number which is then placed as in arnf instructions.

The remaining registers can be set equal to an integer (modulo 1024) which is read in as an integer (with control code m) or as an address in an instruction line of the form qq <pre-defined address>.

The rules for writing SLIP lines are otherwise exactly as described in chapter 11.

Note that for each call of "ret" the contents of only one register may be changed.

Example 13.17.

A program contains a jump to SLIP using the instruction HS 2 ; the number $5_{10}-20$ is to be put in the RF register after which the run is to continue. When this jump to SLIP is made, one should type

```
h ret
grf        [put in RF register]
f          [the floating pt.no.]
5₁₀-20     [5₁₀-20]
e
e          [continue with program]
```

Example 13.18.

The bits TA and PA of the indicator are to be set to 1 while the remainder are to be cleared. At the same time the drum track register is to be set equal to 140. One could then type:

```
h ret
gi
qq 1.2 + 1.4   [set pos.2 and 4 of the indicator to 1]
e
h ret
gk
140            [set integer 140 in the track register]
e
```

The control code m which might otherwise precede the integer 140, is unnecessary because entry to SLIP-HELP always invokes m automatically.

13.4.7 "hp ind" (patch).

This HELP routine inserts "patches" which incorporate calls of other HELP routines in running programs, i.e. "hp ind" will arrange for another HELP routine ("tryk", for instance) to be called when a certain instruction in a running program is executed. "hp ind" in fact, replaces the instruction(s) in the cell in question by a jump to HELP (HSF 1) which is conditioned to execute the selected HELP routine, thereafter to execute the displaced instruction(s) and finally to continue with the running program.

One may insert several "HELP" patches in the same program, but not more than 32 at a time.

The selected HELP routines are executed each time GIER comes to the cell in question but one may suppress this action the first n times (where n is any number ≥ 0). A patch may be removed using "hp ud" (de-patch - see next section).

"hp ind" requires 2 types of control information as follows:

A) Drum storage information (which may be omitted) has the format

qq <start cell>.29+<free track>t<reference address>.29 +<track number>

The first 2 parameters are concerned with the location on the drum of the necessary information for insertion of patches, while the last 2 parameters are only used if the displaced instruction (from the running program) is situated in a drum block. All the drum storage information may be omitted if the patch is to be inserted in a core store block and at the same time if the necessary information may be stored on tracks 58 onwards or in continuation of earlier patch information (depending on the previous use of "hp ind" regarding drum storage information).

The meaning of the 4 parameters is as follows:

<free track> is the address of the first of the tracks (often only one) to be used for storage of the succeeding patch and call information. If this parameter is omitted, or is set equal to 0, the information is stored on the drum in continuation of the information last stored in a call of "hp ind"; if the parameter is omitted with the first use of "hp ind", the information is stored automatically on track 58 onwards (the first available tracks). *)

<start cell> is the number of the first cell, on the track selected above, to be used for storage of the patch and call information which follows. Thus, these two parameters indicate the cell on the drum to

---

*) The information to be stored is the patch and call information described under B) below; the number of cells occupied on the drum is equal to the number of lines of information (including the line containing the patch information). (See example 13.18 below).

which storing begins. If <start cell> = 0 or is omitted completely storing begins at the start of the selected track. <start cell> may also be > 39 since the appropriate address on one of the next tracks is calculated in the usual way.

<track number> is only given if the patch is to be inserted instead of an instruction in a drum block. In this case <track number> is the number of the first track in this block (i.e. the value of k as in the drum-block head). If this parameter is omitted it means that the patch is inserted in the core store.

<reference address> is also only given if the patch is made in a drum block. <reference address> will then indicate the starting address of the block when in the core store (i.e. the value of i at the time the drum-block head is read in; cf. description of drum blocks section 11.6.4). If this parameter is omitted, the reference address = 0.

B) <u>Patch and call information</u> consists of the patch information

<u>h</u> <name of HELP routine>/<n>.29+<patch address>

followed by the necessary call information for the selected HELP routine presented in the usual way.

The patch information tells <u>which</u> routine is to be called via the patch, <u>where</u> the patch is to be made and <u>when</u> the routine is to be activated:

<name of HELP routine> is the name of the selected HELP routine.

<patch address> is the address in the core store of the instruction in the running program, which is to be displaced by the patch (also if the patch is to be inserted in a drum block; the core store address of the appropriate instruction must still be given). The patch always displaces one cell and the instruction or the two instructions, which were originally situated there are executed after the selected HELP routine has been executed.

<n> is the number of times which cell [<patch address>] is to be run through before the HELP routine is activated; after this the HELP routine is executed every time cell [<patch address>] is run through. If <n> is omitted or set = 0, the HELP routine is activated at the very first time (and every successive occasion) when cell [<patch address>] is passed.

One may insert patches in almost any part of a running program as long as one respects the following limitations:

1) Each call of "hp ind" can only be used to insert one patch for one HELP routine in the running program. On the other hand one may call "hp ind" several times with the same patch address (and different HELP routine names) in order to activate several routines in succession at one place.

2) There may be no more than 32 patches at any one time (because HELP's Catalogue of patch addresses is limited to 32); if, however, one removes patches (using the HELP routine "hp ud")

during the course of a program the space occupied becomes free, whereby it is possible to operate with more than 32 patches but not all at the same time.

3) Since the displaced instruction(s) are transplanted from their original location in the store to be executed in some other location (in cell 7, in fact, where it is executed immediately before the return jump in cell 9) certain types of instructions can not be displaced by patches. cell [<patch address>] may thus <u>not contain</u> any of the following:

a) <u>Instructions which are modified</u> by other instructions in the program (the inserted jump will be ruined and the intended effect will never be achieved).

b) An instruction with <u>increment which modifies its own address constant</u> (the instruction is always executed as it was at the time of displacement - modification by increment will <u>not</u> be recorded).

c) An <u>HS instruction</u> after which the s register is used for any other purpose than a return jump of the form HR s+1 , HH s or such-like (since the HS instruction is put in cell 7, the contents of s will thus be 7). An HS jump to a subroutine which calls HELP may not be displaced by a patch.

d) An instruction with <u>V modification</u>.

e) A <u>conditionalising instruction</u> (BS, BT, CA, NC, CM) which governs the execution of an instruction in the next cell (as it will be applied to cell 8). On the other hand the cell may contain a LH half-word conditionalising instruction, as it is only applied to the RH half-word which is also displaced to cell 7.

f) Instructions which are involved with <u>substitution linkage</u>

(IS, IT, NS, NT or instructions immediately following these), because the effect of a substitution is dependent on the two instructions being executed immediately after each other. (It may be alright if the LH half word is the substitution instruction and the RH half word an otherwise harmless instruction).

g) Instructions whose contents are used as constants by other instructions (including reference via indirect addressing).

Indirect, relative or indexed addresses within the displaced instruction are allowed and will be treated correctly.

4) One may set patches in different drum blocks at the same time, but the appropriate core store addresses must be different.

Example 13.19.

In a program in the core store a print-out of the contents of cell 400 and the R register is required each time cell 157 is passed. When cell 219 is passed, a comparison of the core store with the original contents is to be made together with a print-out of the contents of cells 500-509.

When the program has been read in, one may type the following:

```
h gem
e               [by which the Image is preserved for later
                                                comparison]
h hp ind
h tryk / 157    [insert patch activating "tryk", in cell 157]
gm 400          [output cell 400 as fixed pt. no.]
gm, gr          [output R register as fixed pt. no.]
e               [terminate "hp ind"]
h hp ind
h sam / 219
gp 0 t 1023     [changes anywhere in core store are out-
                                        put as instructions]
e               [terminate "hp ind"]
h hp ind
h tryk / 219
gm 500 t 509    [output cells 500-509]
e               [terminate "hp ind"]
e               [terminate call of HELP, enter the program]
```

This will cause cells 157 and 219 to be displaced from the program and replaced by the instruction HSF 1; all the typed information will be stored on track 58 (filling, in fact, 10 cells). During a run the patches will be activated each time cell 157 or 219 is passed.

The patch catalogue will consist of 3 addresses since cell 219 is featured twice.

Example 13.20.

During a run of a program one wishes to output the contents of the RF register each time the computer comes to the instruction in cell 74 which belongs to a drum block that has been read in with the block head b k=125, i=50 . The output routine is to be first activated on the 16.th occasion the instruction is met.

After the program has been read in one may type:

```
h hp ind
qq t 50.29 + 125       [drum storage information]
h tryk/15.29 + 74      [15 dummy runs, cell 74]
gr, grf                [output RF as floating pt.no.]
e                      [terminate "hp ind"]
e                      [terminate call of HELP, enter pro-
                                                     gram]
```

This causes displacement of the 24.th cell of the drum block and output of the contents of RF starting with the 16.th time this cell is passed. On the other hand cell 74 in the core store may be passed many more times, if it is in a section of the store which is used for other program blocks.

In the above case the 4 first cells on track 58 are used for storage of the necessary information about the patches. If this space is not available but, for instance, the last 10 cells on track 153 are free, one may type, instead of the above,

```
h hp ind
qq 30.29 + 153 t 50.29 + 125 [storage in cell 30, track 153]
h tryk/15.29 + 74
gr, grf
e
e
```

If, later in the same run - i.e. without the HELP routine "start" having been used - one calls "hp ind" without giving <free track> and <start cell>, the patch information will be stored on track 153, cell 34 onwards in continuation of the above.

13.4.8 "hp ud" (de-patch).

This routine removes patches which have been inserted by "hp ind", the original instruction(s) being re-instated in their correct place while references thereto are deleted from the patch catalogue. Parameters for "hp ud" include the insertion address and all references to this address are thereby deleted from the catalogue.

In full, "hp ud" parameters have the following format

qq <reference address>.39 + <insertion address>t<track no.>

where

<insertion address> is the address in the core store of the cell from which the patch is to be removed.

<track no.> and <reference address> are only to be given when the cell in question belongs to a drum block; in this case these values correspond to those for "hp ind", that is, the values of k and i at the time the drum block was read in.

It must be emphasized that if one has inserted several patches at the same address (as in example 13.19), they are all removed with one application of "hp ud".

Primarily, "hp ud" may be activated manually during a run when one reckons that a patch has served its purpose; but if one

can plan the removal of a patch beforehand, one can arrange for "hp ud" to be activated after a given number of passages using a counter controlled by "hp ind" (see the following example).

Example 13.21. Controlling "hp ud" using "hp ind".

If one wishes to make a check output of all registers when the instruction in cell 843 (belonging to a modest core store block) is passed, but only from the 2.nd to the 10.th passage, one may type the following after the program has been read in

```
h hp ind
h tryk / 1.29 + 843      [output from 2.nd passage onwards
                                              of cell 843]
gm, ar                   [output all registers,
                                 M and R as fixed pt.nos.]
e
h hp ind
h hp ud / 10.29 + 843    [activate on 11.th passage of
                                              cell 843]
qq 843                   [removes patches in cell 843 incl.
                                           "hp ud" patch]
e
```

Example 13.22. Programmed control.

This illustration of programmed removal of patches should not necessarily be regarded as a typical example. However, let us consider a program at whose start patches were inserted in cells 314 and 471 (in core store blocks). They may be removed again at an appropriate stage of the program by coding:

```
---
[m  ] HS 1             [jump to HELP]
[m+1] h hp ud / 1      [execute "hp ud" with 1 parameter-
                                                  cell]
[m+2] qq 314           [remove patch from cell 314]
[m+3] HS 1             [fresh jump to HELP]
[m+4] h hp ud / 1
[m+5] qq 471           [remove patch from cell 471]
---
```

13.4.9 "slip".

Occasionally it may be useful to make a jump to SLIP-HELP after a certain part of the program has been executed several times, in order to avail oneself of unspecified HELP routines etc. This is impossible with the mechanisms introduced so far, as even though one can jump to SLIP-HELP from the program (with HSF 2 or HS 2) this will happen already on the <u>first</u> time such an instruction is met; one may also cause HELP routines to be executed after a certain number of passages via "hp ind" but these routines must be specified beforehand.

Therefore, a very primitive little HELP routine called "slip" has been made which simply performs the indicated <u>selection of input unit</u> after which it jumps to SLIP (as by HS 2); the routine is primarily intended to be used as auxiliary routine for "hp ind". One parameter of the form

qq <input unit code>

is required, where <input unit code> must be = 17 if one wishes to make input from typewriter and otherwise <input unit code> = 16, giving input from the tape reader. The output is as usual typewriter.

Example 13.23.

After cell 127 has been run through 100 times, one wishes to have the possibility of using SLIP-HELP without being able to specify the actions to be taken beforehand. One should thus insert a patch activating SLIP-HELP via "slip" on the 100.th passage by typing, after the program has been read in

```
h hp ind
h slip / 99.29 + 127   [on 100.th passage, jump to SLIP]
qq 17                  [which awaits input from typewriter]
e
```

13.4.10 Incorporation of other HELP routines.

As mentioned in section 13.4.1 it is extremely easy to change the location and number of HELP routines, as long as tracks 0–38 remain uncorrupted. In addition the HELP routine "læs hp" (read HELP routine) which is normally stored on track 57 must be intact. "læs hp" can not be called as other HELP routines can, but it is taken into use when one reads in a HELP routine tape by typing

c<first track>
l

At the same time as the program is read in, its name and location are registered in the catalogue of HELP routines; if it has the same name as a routine which has been read in earlier the original routine is deleted from the catalogue, (cf. the remarks in section 13.4.1).

During input of new HELP routines, track 57 (or rather, the track on which "læs hp" is stored) must not be corrupted, but as soon as input is over this track may be used for anything whatsoever.

Example 13.24.

If one insists on reading a HELP routine occupying 4 tracks to tracks 54–57, it cannot be done without moving "læs hp". Therefore, one must first read the HELP-routine tape for "læs hp" on to another unused track, let us say, 117 by typing:

c 117
l

When "læs hp" has been read in, one can place the desired HELP-routine tape in the reader and type

c 54
l

Note that the tapes "HJÆLP—uaflåsede" (HELP-unlocked tracks) and "HJÆLP-uaflåsede udenom ALGOL" (HELP-unlocked tracks co-existent with ALGOL), containing all the standard HELP routines, are read in by typing 1 only. The tape "HJÆLP" (HELP), which contains the whole system excluding track 0, is read in by typing a space (after "FEJL" message, see section 13.5.3, below).

13.4.11 Rules for preparation of HELP routines.

Routines which are to be introduced into the HELP system must be prepared with due consideration for the few rules which must necessarily be fulfilled so that routines can be administered correctly by HELP. These rules are concerned partly with the way in which a routine must be programmed and partly with the layout of the final tape.

A) The program.

1) Length and location: The program must not occupy more than 12 tracks and must be designed for execution when located in cell 0 onwards. The first 2 cells must be as follows:

[cell 0] a QQ instruction with address 55 x (no.of tracks-1)*)

[cell 1] "complementary check total"

By "complementary check total" is meant that the bit pattern is such that the total (accumulated by AR instructions) of all cells in the program (a whole number of tracks) is equal to 0.

2) Entry and exit: The entry to a HELP routine is made from the HELP administrator using an HV instruction; the address of the entry is not restricted and is specified on the tape which

*) The contents of the RH half-word in cell 0 is irrelevant.

inputs the routine (see below). The usual exit from the routine should be made with the instruction HV 694 , and the contents of the registers at this point are immaterial. If the routine, for some reason or other, cannot fulfill its mission (perhaps because of errors in the parameters), an error message should be typed terminating with the execution of the instructions VY 17, HH 695 ; this will put HELP in the same situation as after s .

3) Parameters: When entry to a routine is made, the HELP administrator has read in the parameters placing them in cells 642, 643, ...; there is not room for more than 38 parameters. The number of parameters is held as the increment in cell 641 (which incidentally contains the line h <HELP-routine name>/<no. of parameter> , stored in the same way as mentioned in section 13.3.3). On exit from the routine the contents of cells 641-679 are irrelevant.

B) The tape.

The tape containing a HELP routine to be read in as described in section 13.4.10 above must have the following structure:

```
b a0
a0: b k=a0, i=0
    ---
    --- }  punch-out (usually condensed) of the HELP rou-
    ---    tine without introduction or termination
    e
h læs hp
h <HELP-routine name>/<entry address>.29 + a0
e
e
s
```

where <entry address> is the required starting point (most likely to be cell 2), while <HELP-routine name> is the required name of

the program. The name may be selected freely, although the sum of the values of the letters should be different from those of existing HELP routines.

13.5 Error messages from HELP.

Apart from the error messages (and after effects) mentioned in section 11.8, due to syntactical errors and such-like during input via SLIP, there are a number of error messages which may occur when using the HELP system as a whole; they are mentioned below.

13.5.1 Wrong control parameters.

During input (typed or punched) of parameters for a HELP routine one may encounter the error message

gal information

(Danish for "wrong information"). This means that one or more of the parameters is not in accordance with the requirements of the routine in question. When the terminating e is read in the HELP routine may be executed inadequately (or not at all) after which one can select the routine once again and supply the correct parameters.

One may also, as soon as the error message has been recognized, press the HP button and select the routine once again.

If, while typing the control parameters, one realizes that something has been typed incorrectly, one may correct the error in the same way as with normal input via SLIP: complete the cur-

rent line; type, if necessary, CR and a space which will give a report of the serial address i ; set i back to the erroneous line; type the line correctly; advance i as appropriate and continue where one left off.

One may of course press the HP button but this will involve starting from scratch with the routine.

13.5.2 "tomt hp" (not in catalogue) and "sumfejl" (error in check total).

If one calls a HELP routine and gets the reply

tomt hp

it means that there is no entry in the catalogue of a routine with the name or rather with a name in which the sum of the values of its letters is the same as the selected name).

sumfejl

means that the routine in question is registered in the catalogue but that the check total does not agree, i.e. the tracks on which the routine is stored have been corrupted.

In both cases, GIER will await typewriter input and one is strongly advised to type

<u>h</u> kontrol
<u>e</u>

in order to ascertain which HELP routines are intact and where they are located. After this one may possibly read the corrupted routines in again and continue. A single HELP routine is read in by typing

<u>c</u> <start track>
<u>l</u>

whereas the whole of that part of HELP which is stored on unlocked tracks is contained in both "HJÆLP-uaflåsede" and "HJÆLP-uaflåsede udenom ALGOL" and these can be read in by just typing <u>1</u> ; input is terminated by a standard jump to HELP, and the interrupted run may be continued since input only affects those tracks reserved for HELP.

<u>13.5.3 Errors in the locked tracks.</u>

A) If, after pressing the HP button or after any other entry to HELP, one obtains the message

FEJL

it means that there is <u>an error in one of the locked tracks 1-31</u>. The situation can be resolved by the following procedure: 1) Unlock tracks 1-31 (a switch in the main cabinet); 2) Place the "HJÆLP" tape (containing the whole of the HELP system located on tracks 1-57) in the reader and type a space, after which the tape will be read in; 3) Lock tracks 1-31. As in the case above, the situation in the rest of GIER is unchanged and the interrupted run may be continued.

B) If pressing the HP button does not even cause the message FEJL, it may be because the HP button is inhibited (pos.0 in the by register set to 1). Pressing the RESET button will release the interrupt. If even this does not work, there is a computer error or an error in track 0, and in all cases one can only continue running after technical assistance has been called for.

During a normal run with GIER errors on track 0 should not occur since writing is inhibited by a very inaccessible contact pin in the main cabinet. If the HP button does not function it is therefore an indication of a serious error in GIER.

14. LIBRARY ROUTINES.

This chapter dealing with the way in which library routines should be prepared and presented has been deleted. The reader is referred to the publications of "GIER System Library" which is responsible for the coordination of user activity with respect to library routines.

15. EXERCISES.

This chapter has been deleted.

16. SUMMARIES AND TABLES.

16.1 Numerical Representation of the Typographical Symbols.

The table below is a slightly extended version of the table in section 8.4 of Volume I; it shows for both off-line typewriter, on-line typewriter and line printer the correspondance between the address in an SY instruction and the symbol output. For the line printer the code and character set are as currently available with Anelex mark 4-1000 (cf. section 9.4.3).

	Off-line typewriter		On-line typewriter		Line Printer	
	LC	UC	LC	UC	LC	UC
0	Space		Space		Space	
1	1	∨	1	∨	1	£
2	2	×	2	×	2	×
3	3	/	3	/	3	/
4	4	=	4	=	4	=
5	5	;	5	;	5	;
6	6	[	6	[	6	[
7	7	]	7	]	7	]
8	8	(	8	(	8	(
9	9	)	9	)	9	)
10	not used		not used		Space	
11	Stop Code		not used		Stop	
12	not used		not used		Space	
13	not used		å	Å	Å	
14	_	|	_	|	*	'
15	not used		not used		%	&
16	0	∧	0	∧	0	↑
17	<	>	<	>	<	>
18	s	S	s	S	S	
19	t	T	t	T	T	
20	u	U	u	U	U	
21	v	V	v	V	V	
22	w	W	w	W	W	
23	x	X	x	X	X	
24	y	Y	y	Y	Y	
25	z	Z	z	Z	Z	
26	not used		not used		ǂ	
27	,	⏨	,	⏨	,	⏨
28	not used		not used		Space	
29	not used		Red ribbon		not used	
30	Tab		Tab		Tab	
31	Punch Off		not used		$	

	Off-line typewriter		On-line typewriter		Line Printer	
	LC	UC	LC	UC	LC	UC
32	–	+	–	+	–	+
33	j	J	j	J	J	
34	k	K	k	K	K	
35	l	L	l	L	L	
36	m	M	m	M	M	
37	n	N	n	N	N	
38	o	O	o	O	O	
39	p	P	p	P	P	
40	q	Q	q	Q	Q	
41	r	R	r	R	R	
42	not used		not used		Page Change *)	
43	ø	Ø	ø	Ø	Ø	
44	Punch On		not used		α	
45	not used		not used		Vertical Tab *)	
46	not used		not used		Space	
47	not used		not used		Space	
48	æ	Æ	æ	Æ	Æ	
49	a	A	a	A	A	
50	b	B	b	B	B	
51	c	C	c	C	C	
52	d	D	d	D	D	
53	e	E	e	E	E	
54	f	F	f	F	F	
55	g	G	g	G	G	
56	h	H	h	H	H	
57	i	I	i	I	I	
58	LC		LC		LC	
59	.	:	.	:	.	:
60	UC		UC		UC	
61	not used		not used		Space	
62	not used		Black Ribbon		not used	
63	Tape Feed		not used		not used	
64	CR		CR		CR	

LC = Lower Case; UC = Upper Case

*) "Vertical Tab" means that the line printer can be made to space a predetermined number of lines when it receives the instruction SY 45. "Page Change" via SY 42 causes the line printer to start on a fresh page. See also section 9.4.2.

16.2 Entries and Layouts when Editing numbers.

Below is given a short summary of the different entries, the layout format and the meaning of each parameter when editing numbers using the standard routine in HELP. These are described in more detail in section 12.2.

Entry: HS <addr.>
QQ <address of parameter word>, ...
or QQ <address of parameter word>, HS <addr.>

<addr.> = m+0 : R_{0-9} as integer ≥ 0
m+1 : R_{0-9} – integer, $-512 \leq h \leq 511$
m+2 : R – integer
m+3 : R – fixed pt.no.
m+4 : RF – floating pt.no.

Layout:

QQ b.3+h.7+d.13+f1.9+n.14+bE.17+f2.19+g1.23+g2.27+g3.31+g4.35+g5.39

Parameter	Meaning
$0 \leq b \leq 15$	No. of significant digits
$0 \leq h \leq 15$	No. of digits before the point
$0 \leq d \leq 15$	No. of decimals
$0 \leq f1 \leq 3$	Printing of sign of mantissa:
	Plus sign before positive Nos.: f1 = 2
	– – replaced by space: f1 = 1
	– – omitted completely: f1 = 0
	Sign in first printing pos.: f1 = 3
$0 \leq n \leq 1$	Printing of zeroes: Nought before point: n = 1
	Space – – : n = 0
$0 \leq bE \leq 7$	No. of significant digits in exponent
$0 \leq f2 \leq 3$	Printing of sign of exponent: as for f1
$0 \leq g1 \leq 15$	No. of positions in 1st group of digits
$0 \leq g2 \leq 15$	– – – – 2nd – – –
$0 \leq g3 \leq 15$	– – – – 3rd – – –
$0 \leq g4 \leq 15$	– – – – 4th – – –
$0 \leq g5 \leq 15$	– – – – 5th – – –

16.3 Underlined letters in SLIP.

Below is given a list of all the underlined letters in SLIP; detailed descriptions are given in chapter 11, particularly section 11.6.5.

b <declarations>	: Core-store-block head (b alone is dummy)
b i=<pre-def.addr.>	: Core-store-block head
b k=<pre-def.addr.>	: Drum-block head
c <integer>	
d <name>=<pre-def.addr.>,...	: Definition of serial addr. and/or label (d is superfluous)
e	: Termination of block
e <pre-def.addr.>	: Termination of input
f	: Floating pt. numbers
h <HELP-routine name>	: Call of HELP routine
l	: Input via tape
m	: Fixed pt. numbers
n	: Cancel automatic relative-addressing
r	: Establish automatic relative-addressing
s	: Input via typewriter
t	: Text
u <pre-def.addr.>	: Definition of exit address
x x <pre-def.addr.>	: Dump table of labels
z z <pre-def.addr.>	: Restore table of labels

All other underlined letters (except g) have the same effect as s.

16.4 Error messages from HELP and SLIP.

The table below shows all the error messages produced by HELP and SLIP with a short description of their meaning. Further explanation is given in sections 11.8 and 13.5.

Message in red	Meaning
1 <serial address>	Syntactical error
2 <serial address>	Post-defined label used incorrect
3 <serial address>	Undeclared label used
4 <serial address>	Error in declaration
5 <serial address>	Unused punched code
6 <serial address>	Check definition does not agree
7 <serial address>	Syntactical error in number
8 <serial address>	Range of numbers exceeded
9 <serial address>	Too many labels or blocks
<label><addr.1><addr.2>	Label not defined at end of block. Last reference in the address part of cell [<addr.1>] and in the increment part of cell [<addr.2>]. (Typed in black).
gal information	Wrong parameters. A HELP routine has been called with erroneous parameters.
sumfejl	Check-total error. Check total for selected HELP routine does not agree.
tomt hp	Not in catalogue. Selected HELP routine is not registered in the HELP-routine catalogue.
FEJL	Error in one of the locked tracks (Nos.1-31).

16.5 The effect of HELP routines and types of parameters.

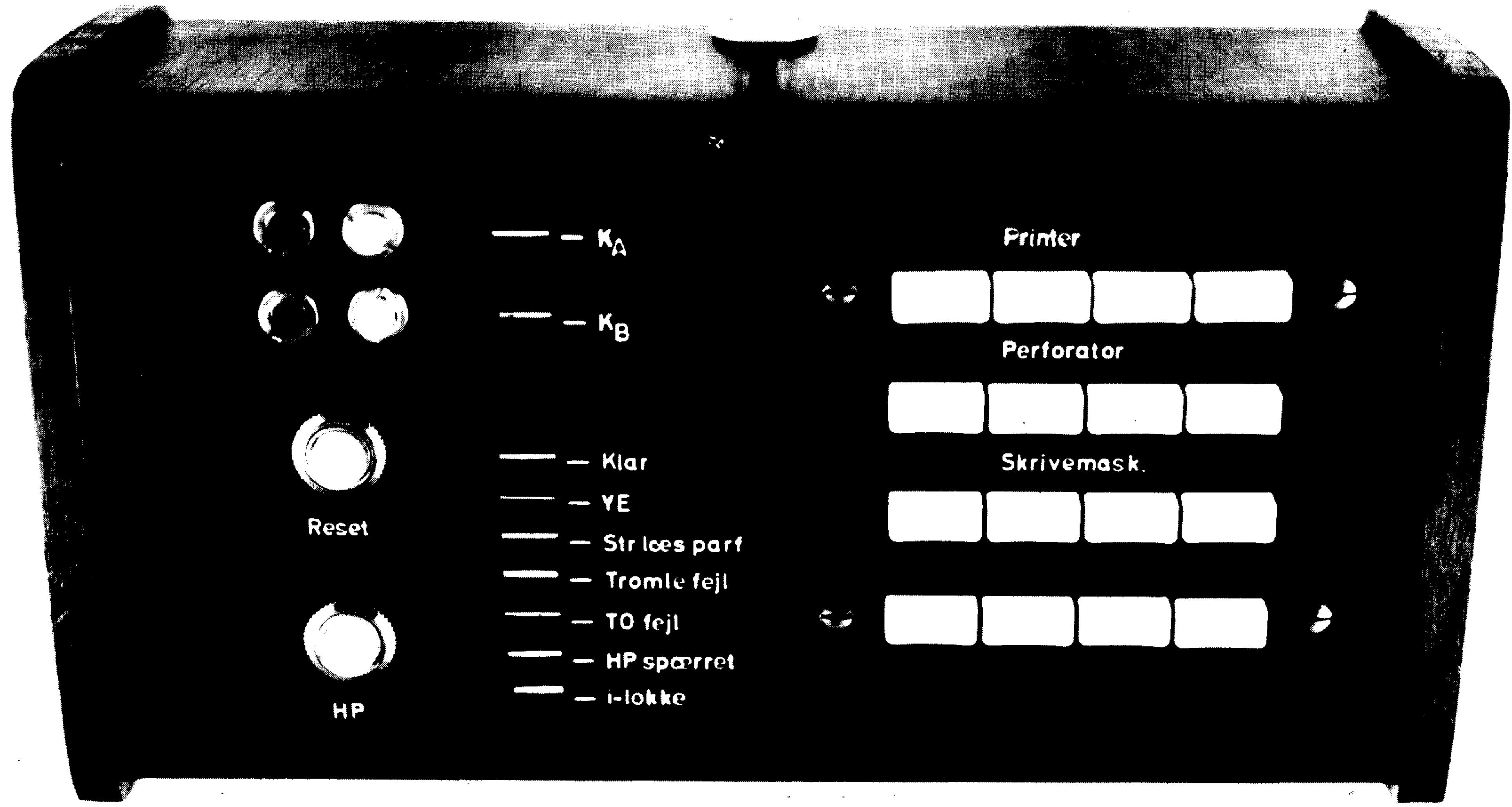
The table below shows the effect of each routine and the types of parameter which can be included in a call of each rou-

Name	Effect
h start	Clears store, initialises HELP-routine catalogue, verifies check-totals
h kontrol	Verifies check totals
h tryk	Print-out of store and registers
h kompud	Condensed dump of store
h gem	Preserves Core Store Image
h hent	Restores Core Store Image
h sam	Comparison of sections of store
h ret	Input to registers
h hp ind	Inserts HELP-routine patches
h hp ud	Removes HELP-routine patches
h slip	Entry to SLIP
h hop	Traces running program, reports all jumps

tine. The specific effect of each parameter is not mentioned here (but is described in chapter 13). The table covers all the standard HELP routines and the tracer-routine "hop".

Parameters
None or one in any form
None or one in any form
<output format>,<name of register> <output format><track>.39+<start addr.>t<end addr.> bt<s><integer>.39+<output unit code>t<ref.addr.> { nt<power of 2>t<power of 10> <layout>
3c 4c <first track>/<first cell>/<last track>/<last cell><mark> <u addr.>/<e addr.>/<exit code>/<dump code>c
None or sk<start track>
None or sk<start track> sk<no.of tracks>.39+<start track 1>t<start track 2>
sk<start track 1>t<start track 2> output delimiters as for h tryk
<name of register> <SLIP line>
qq<free cell>.29+<free track>t<ref.addr.>.29+<start track> h (name of HELP-rout.>/<n>.29+<insertion addr.> parameters for the selected HELP-rout.
qq<ref.addr.>.39+<insertion addr.>t<start track>
qq<input unit code>
bt<core store addr. for "hop">t<output unit code> qq<start addr.>t<end addr.>

CONSOLE



AUXILIARY CONSOLE

INDEX

INDEX

Address 61
Address in RH half-word 68
Address, syntax 108
Address with post-defined labels 62
Address with pre-defined labels 65
Anelex line printer 20
Auxiliary console 50
Auxiliary console, photo 205
Available storage space in HELP 55, 135

b (in layout) 117
b 80
Base 10 exponent (in layout) 116
Basic operation 59
Basic trim for "tryk" and "sam" 160
bE (in layout) 118
bl lamp 41
Blocks 80
Blocks, drum 89
Block head 80
Block, syntax 108
Block tail 82
bs lamp 42
Buffer stores in card reader 27
by lamp 42

Call line for HELP routines	144
Card-reading speeds	26
Cards, "80 column"	26
Card reader	26
Catalogue of HELP routines	151, 153
Check (HELP routine "kontrol")	152
Check on reading of drum tracks	7
Compare (HELP routine "sam")	173
Condensed Dump (HELP routine "kompud")	165
Condensed tape	165
Consecutive stacking, card reader	30
Console	38
Console lamps	38
Console, photo	204
Constant lines, syntax	110
Control codes	94
Control information in SLIP	57
Control lines, syntax	111
Control parameters for HELP-routines	144
Core Store Image	135
Correction (HELP routine "ret")	177
Corrections to core store	148
Corrections to register	148, 177
d (in layout)	117
Definition line	85
De-patch (HELP routine "hp ud")	173
Digits before the point (in layout)	116
Digits (in layout), number of	116
Drum block	89
Drum-block head	90
Drum blocks, restrictions	92
Drum error	8, 52
Drumfree jumps	10
Drum store	1
Dummy information in SLIP	58

e	82
Editing, special facilities	125
Entry, manually-controlled	101
Entry, program-controlled	102
Error marking, card reader	29
Error messages (HELP)	195, 201
Error messages (SLIP)	104, 201
Error types (SLIP)	105, 202
Errors in the locked tracks	195
Execution of single instructions	46
Exit address	95, 103, 140, 167
Exit from HELP	140
Exit from SLIP	103
Exponent (in layout)	116
f1 (in layout)	117
f2 (in layout)	118
f	94
"FEJL"	195, 201
Final address when reading cards	31
Fixed-point numbers	74
F lamp	41
Floating-point numbers	74
Floating-point overflow	143
g1...g5 (in layout)	119
g	97
"gem", HELP-routine	173, 202
Grouping of digits (in layout)	116
h (in layout)	117
h	97, 144
HELP, error messages	194
HELP, location of	134
HELP routine: "gem"	173, 202
"hent"	174, 202

HELP routine: "hop" 202
"hp ind" 180, 202
"hp ud" 187, 202
"kompud" 165, 202
"kontrol" 152, 202
"læs hp" 190
"ret" 177, 202
"sam" 175, 202
"slip" 189, 202
"start" 154, 202
"tryk" 156, 202
HELP routines 150
, activation via paper tape 145
, catalogue of 151, 153
, call-line 144
, location of 150
, preparation of 191
, programmed activation 146
HELP-unlocked 191
HELP-unlocked co-existent with ALGOL 191
HELP, utility programs 132
"hent", HELP-routine 174, 202
HJÆLP, error messages 194
HJÆLP-uaflåsede 191
HJÆLP-uaflåsede udenom ALGOL 191
h lamp 39
H lamp 40
HP button 50, 51, 135
HP button out of function 136
"hp ind" HELP routine 180, 202
HP-knap 50, 51, 135
HP-knap spærret lamp 52
HP patches 178
"hp ud" HELP routine 187, 202
HSF 2 102, 142
HS 1 142

i, serial address 68, 78
"i-løkke" lamp 52
Incorporation of HELP routines 190
Increment 71
Increment, syntax 110
Indicator lamps 39
Information, lines of (SLIP) 57
Initialisation (HELP routine "start") 152
Initialisation parameters for "tryk" 116
in lamp 41
Input from tape 12
Input from typewriter 17
Input of condensed tape 170
Input of numbers and text using SLIP 74
Input program SLIP 54
Instantaneous values of i and k 68, 78
Instruction line, syntax 109
Instructions, loading of 59
Instructions, separate parts of 42
Integers 74
Integers, packed 75
Interrupt button 50, 51

k, serial number 68, 78
Kanal 0 4, 195
KLAR lamp 40, 51
"kompud", HELP routine 165, 202
"kontrol", HELP routine 152

l 94
Labels 62, 108
Labelling 86
Label, local 80
Labels, pre-defined 62
Label, post-defined 62
Labels, values of 106
Lamps on auxiliary console 51

Layout	116
Lines in SLIP	57
Line printer	20
Line printer code	25
Lines, syntax	108
L lamp	40
Loading of instructions	59
Local label	80
Location of HELP	134
Locked tracks	4, 195
Locked tracks, error on	195
Locking drum tracks (3 drums)	5
Lower Case, input from typewriter	17
Lower Case, output to typewriter	18
"læs hp" HELP routine	190
m	94
$M_1 \ldots M_4$ lamps	39
Manually-controlled entry	101
MIKROTEMPI START	44
MIKROTEMPI STOP	43, 44, 51
M lamp	40
n (in layout)	118
n	95
NORMAL START	43
NORMAL STOP	44
nought, printing of, (in layout)	116
numbers, editing of, by HELP	113
numbers, input of	74
O lamp	39, 40
On-line typewriter	17
Operation times for LY instructions	20
Output	112
Output (HELP routine "tryk")	156
Output unit, choice of	21

Packed integers 74
Page change, line printer 24
Paper tape activation of a HELP routine 145
Paper tape punch 16
Paper tape reader 12
Parity bit 14
Parity check, input 13
Parity check, reading to tracks 5, 7
Parity error 14
Patches (HELP routines "hp ind" and "hp ud") 180
Peripheral units 12
Post-defined labels 62
Pre-defined labels 62
p register 42
Preserve (HELP routine "gem") 173
Printer code 25
Program 99
Program-controlled entry into SLIP 102
Programmed activation of a HELP routine 146
Programmed entry into HELP 141
Programming of HELP routines 190
Program, syntax 108

r 95
r1 lamp 41
r2 lamp 41
RC 2000 tape reader 15
Reading of cards 26
Re-definition 88
Registers 38
Relative address with labels 63, 67
Reports (SLIP) 106
RESET 44
RESET lamp 51
"ret" HELP routine 177, 202
Retrieve (HELP routine "hent") 173
RH half-word, address in 68
R lamp 40

s 94
s1 lamp 41
s2 lamp 41
"sam" HELP routine 175, 202
Scale factor (in layout) 123, 126
Scaling 68
Separation of address part and increment 72
serial address, i 68, 78
serial track number, k 68, 78
SF lamp 40
Significant digits 116
Signs (in layout), printing of 116
Simultaneous drum transfers 7
SLIP 54, 59
"slip" HELP routine 189, 202
SLIP syntax 108
Sorting of cards 30
Stacking, consecutive 30
Stacking of cards 29
Start 46
Start buttons 42, 43
"start" HELP routine 154, 202
Step-by-step running 47
Stop 43
Stop buttons 43
str.læs.par.fejl 52
"sumfejl" 194, 201
Summary, error messages 201
HELP routines 202
layout 199
line printer code 198
numbers, editing of 199
numerical representation of typographical symbols 198
off-line typewriter 198
on-line typewriter 198
underlined letters in SLIP 200
Syntax for SLIP code 108

t	72
<u>t</u>	94
ta lamp	41
text, input of	77
Text, printing of, using HELP	127
text line, syntax	110
Three drums	5
tk lamp	41
T lamp	39
TO lamp	40
"tomt hp"	<u>194</u>, 201
Track 0	4, 195
Track address	4
Tracks, registers of locked	5
TR lamp	40
Tromlefejl	52
"tryk", HELP-routine	<u>156</u>, 202
Typewriter	17
<u>u</u>	<u>96</u>, 168
Upper Case, input from typewriter	17
Upper Case, output to typewriter	18
Utility programs, HELP	132
Value of labels	106
<u>x</u>	<u>96</u>, 98
YE lamp	39
<u>z</u>	<u>96</u>, 98