A MANUAL OF GIER PROGRAMMING

VOLUME I

by

Chr. Andersen and Chr. Gram

(translated by Jytte Søllested and George Alan Lake)

DANISH INSTITUTE OF COMPUTING MACHINERY

Copenhagen, October 1963

FOREWORD

In August 1961 the first Danish GIER manual ("Lærebog i Kodning for GIER") was published. The third revised edition (again in Danish) was published in September 1962. This manual is a translation of the third edition, with only a few minor corrections.

As can be seen from the list of contents, this volume contains a general outline of the features of computers, a description of the structure of GIER, the instruction format, the list of operations; and a section with examples and exercises.

Very little has been said in this volume about peripheral equipment and input-output routines, but these topics will be covered in a forthcoming volume II. This will be a translation of the first (Danish) edition of volume II, published in October 1963, and a.o. contain a full description of the programming language SLIP, which allows the use of symbolic addresses, and of HELP, the system of administrative programs.

We would like to express our indebtedness to the many people who have given us much valuable help in the preparation of this book. First of all, our thanks are due to Mr. Bj. Svejgaard and Mr. T. Krarup, of the Royal Danish Geodetic Institute, who have willingly answered a great many questions.

Acknowledgement is also due to Mr. H. Isaksson, Mr. L. Hansson, and Mr. P. Mondrup, who have participated in many discussions and have suggested many improvements.

Finally, we would like to thank Mrs. Jytte Søllested and Mr. George Alan Lake for their patient work with translation of the Danish edition. Mrs. Søllested has translated and prepared the whole of the manuscript; and Mr. Lake has read, critized, and corrected the English manuscript.

Chr. Andersen   Christian Gram

TABLE OF CONTENTS

1. DIGITAL COMPUTERS

1.1 Introduction

Before we discuss the electronic computer GIER we have thought it reasonable to give a survey of certain characteristics common to electronic digital computers 1). Furthermore we will describe different aspects of these computers.

The survey given in this chapter is by no means complete. There is for instance little reference to the technical aspects, but we hope that the reference given to each item will be sufficient to ensure that readers without advanced knowledge of digital computers nevertheless will be in a position to study for themselves the remaining contents of this manual.

Chapter 2 describes the construction and function of GIER, while chapter 3 and the following chapters are devoted to the information which the user (programmer) must have at his disposal to be able to write programs for GIER.

1) The big family of analogue computers will not be mentioned in this manual at all.

1.2 Structure

By far the most electronic computers existing today have the same basic, logical structure. Each computer has an arithmetic unit, a storage unit, a control unit, and certain peripheral units.

The arithmetic unit has certain characteristics in common with the modern desk calculating machine i.e. it carries out the four basic arithmetic operations and has space for recording a few numbers; these storing spaces in the arithmetic unit are called registers. On a desk calculating machine one can suffice with a few registers as interim results can be written down on paper and replaced in the machine later on. Such a procedure necessitates that the machine stands idle several seconds between each calculation, and bearing in mind that electronic computers are very expensive this procedure is completely useless.

Therefore the electronic computer is provided with a store, in which it can store the interim results (the store thus replaces the paper). The time needed for the computer "to write a number" in the store or "to read a number" from same is approximately 10^{-4} seconds, and calculations in the arithmetic unit are carried out with the same speed. Due to this high speed it is very economic to have extensive calculations carried out on electronic computers.

The store is divided into areas or drawers called cells. A cell consists of a series of physical elements each having a number of "stable states". In most computers each element only has two states, and in that case the elements are called binary (and the computer in question is said to work in binary mode). Very often each cell consists of 40 binary elements and in that case there will be 2^{40} different possibilities for "the contents" of the cell.

Such contents we may call a digital pattern,which corresponds to the contents of each element being called a digit or a bit (binary digit). The two possible values are called 0 and 1 (see chapter 1.5). The computer will often regard a digital pattern as a (real) number, and each cell is thus able to hold one number. Each cell has a number, an address, and the computer is constructed in such a way that it is able a.o. to perform operations such as "store this number in cell so and so" and "read the number in cell so and so and send it to the arithmetic unit".

In order to be able to carry out large calculations, the computer needs a big store with space for many numbers; a big store, however, that is "as fast as" the arithmetic unit is very expensive. For economic reasons many computers are therefore provided with a smaller, but fast working store and a very big store, which is considerably slower to write to and to read from. It is then necessary to conduct the calculations in such a way that the numbers used most often are contained in the fast store while numbers used only infrequently during the calculations are stored in the slower store.

The control unit controls the course of calculations in accordance with predetermined instructions, and the control unit needs detailed information as to what is to be done and in which order things are to be done. The computer cannot invent new operations or methods; under certain circumstances it may be able to choose between different possibilities, but all possibilities must be thoroughly considered and planned in advance by the instigator.

The peripheral units are the computer's contact with the surroundings, and it is through these units that the computer receives information and delivers results. The efficiency of this equipment varies very much from computer to computer, and only little can generally be said.

However, only very few of the computers existing at present are able to read hand- or typewritten numbers or letters. The general principle is that all information for the computer has to be transferred to punched card or paper tape (this is done by means of special typewriters), after which the input unit of the computer reads the cards or the tape respectively.

A punched card is a rectangular piece of cardboard divided into a number of columns (usually 80), and each column has 10 rows numbered 0, 1,...., 9, and two extra rows used for control punching. A hole in a row represents the corresponding number, i.e. the number 15 may be written with a hole in row no. 1 in column no. 1 and a hole in row no. 5 in column no. 2. In addition a combination of 2 holes in the same column may be used to represent letters.

The principles for the use of punched paper tape are similar to the above: Patterns of holes are made in rows across the length of the tape, each pattern representing a certain symbol (number or letter or possibly a typographical sign).

Punched card or paper tape is read by allowing the medium to pass a reading head which is able to sense these patterns of holes and send electric impulses, corresponding to what is sensed, to the computer itself. Reading from peripheral units is often referred to as input. The conveyance of information to the peripheral units is known as output.

Information may be output by means of an electric typewriter which is directly connected to the computer. An ordinary typewriter operates, however, very slowly compared to electronic computers, and very fast printers (which write a whole line at a time) are very expensive. By far the most computers are provided with output units, which punch cards or paper tape, as such equipment has a reasonable price and is 10 times faster than an ordinary typewriter. Another not inessential advantage is that in this way results written on punched cards or paper tape can be read into the computer again if they are to be used for other calculations.

If output is made on punched cards or paper tape it is necessary to use paper-tape-controlled typewriters or the like to translate the results into legible form afterwards. These remedies, however, are considerably cheaper in use than the computer itself.

1.3 Applications. A simple example of coding

To make full use of the high operating speed of the electronic computer it is essential that it works more or less automatically. In other words, a computer must receive information in advance concerning everything which is going to happen during a calculation. The computer must have a series of instructions, and these are then performed in the order, in which they are written. The computer can only perform very elementary operations such as addition, subtraction, multiplication, or division of two numbers, transfers (as mentioned above) and certain other administrative operations. Therefore, any problem, which is to be solved by an electronic computer, must be thoroughly analysed and then redefined in terms of <u>elementary operations</u>, which have to be performed in a predetermined order. <u>Programming</u> is the name given to this conversion of a given operation into a series of elementary operations, which the computer can perform.

For instance a computer cannot immediately perform the following operation:

(1) Solve the equation: $22x + 32 = 479$

because it requires exact information on how to handle the matter. It helps considerably if the operation is formulated as follows:

(2) Calculate: $x = \frac{479 - 32}{22}$.

It will not, however, be ready for machine processing until we can imagine the numbers 22, 32, and 479 placed in the store, for instance in cell no. 101, 102, and 103, and after this give the computer the following series of instructions (the following program):

"read the number in cell 103; send it to the arithmetic unit"

"read the number in cell 102, subtract it from that in the arithmetic unit"

(3)"read the number in cell 101; divide it into that in the arithmetic unit"

"write on typewriter the number in the arithmetic unit"1)

"stop"

These instructions must be stored in advance in the above order and for this purpose the <u>same store</u> is used as for storing numbers. Many computers are constructed in such a way that each cell is able to store either a number or an instruction. It is convenient to use the same store for both things, firstly because it makes the computer more flexible in use - for some operations much storage space is used for numbers and only very little for instructions and vice versa, and secondly it means that the computer is able to modify its own instructions, i.e. the computer can itself change the method of calculation during solution of a problem. However, this can only be done in accordance with a plan fixed by the programmer.

---

1) While the other instructions correspond with real facts, this way of writing the output is a serious simplification of the truth. In fact, at this stage of the program there should have been a whole series of instructions which should control the number of digits and decimals etc.

As we are going to show in the following, each cell in the store can only store digital patterns, and if the contents of a cell are sent to the arithmetic unit this will be understood and treated as a number; but if the contents of the same cell are sent to the control unit, it will be understood as an instruction, which will then be executed. The computer itself can in other words <u>not</u> distinguish between a number or an instruction, and therefore the programmer has to decide and then keep track of which parts of the store are used for numbers and which parts are used for instructions.

The procedure necessary for the computer to solve the above-mentioned problems is now as follows:

1) At first the necessary 5 instructions and 3 numbers are punched either on paper tape or on cards. We might for instance punch the exact digital patterns that represent the instructions of (3) in the store; but as many electronic computers use the binary system it will mean that each instruction and each normal decimal number at first must be translated or converted into a series of zeroes and ones (and apart from the inconvenience herewith it would be nearly impossible to do it correctly and to proof-read it). Therefore, nearly all computers have a special <u>input program</u> (which is placed in the computer at first), which reads paper tapes or punched cards punched with more convenient conventions (for instance a number is written in decimal form), and this input program then translates the information on the paper tape (or punched cards) into the internal language of the computer. As to what is to be punched instea of for instance the instructions in (3) can be found in the <u>operation list</u> of the computer; the operation list first describes which operations the computer is able to perform and secondly gives information of what is to be punched for each single instruction.

In this case one may find that the instructions (3) must be punched as

```
     MOVE  103
     SUB   102
(4)  DIV   101
     WRITE
     STOP
```

as far-reaching mnemotechnical viewpoints must be taken into consideration when choosing the "external" language, after which an input program can be constructed in accordance herewith.

Besides the instructions the numbers 22, 32, and 479 must be punched (as decimal numbers) and then read in by means of the input program, which converts the numbers into the number system of the computer (if this is not the decimal system).

2) When the instructions have been read in, for instance to cell no. 1, 2, ..., 5 and the numbers to cell no. 101, 102, and 103, the computer is started. Hereby the control unit at first reads the contents of cell no. 1, interprets it as an instruction and executes this (in this case the number in cell 103 is read to the arithmetic unit); then the control unit reads cell no. 2, interprets the contents as an instruction, executes this etc. until the control unit interprets the contents in cell no. 5 as a stop instruction and stops the computer.

It is a typical feature in by far the most electronic computers that the instructions are automatically executed in the order, in which they are placed in the store. However, it is often of interest to let the computer repeat a calculation many times (for instance with new data each time), in other words to let the computer return to one of the instructions already executed. For this purpose - and generally to break the natural order of instructions - the so-called jump instructions are used. These instructions are purely administrative, as execution of a jump instruction does not cause any calculation, but only that the next instruction is taken from a new, further specified place in the store.

Example 1.1

As an illustration of the use of jump instructions let us take the above program (4). In this program the instruction WRITE is a serious modification of the truth, as no computer can do with one instruction only (in the case of output of a number), because in some way or another it is necessary to state how many decimals must be included, how the sign must be written, where the decimal point should be placed and so on (and further, if the computer works internally in the binary system it must make convertion into decimal representation). As, however, the writing of results is a function which is repeated in nearly all operations, a program for this special purpose is coded once and for all and stored at a certain place in the store, for instance from cell 1000 and onwards. If it is desired as in program 4 to write out a result, the WRITE-instruction is replaced by a jump instruction as for instance

JUMP 1000

This instruction has the effect that after the instruction DIV 101 has been executed the computer carries on with the instructions in cell 1000, 1001, ..., until the final result has been written on the typewriter. This fixed writing program will often be followed by another jump instruction, which makes the computer return to the place where it left off, and thus continues with the STOP instruction in cell no. 5, i.e. the computer stops.

1.4.Applications. Subroutines

The writing program mentioned in example 1.1 is a subroutine, i.e. a program which is permanent (and tested) and which is able to perform some frequently recurring problems.

The most important subroutines for any computer are: an input program, which - as mentioned in 1.3 - is able to translate from our external instruction language and from our decimal language to the internal representation of the computer, and an output program for writing out results. These routines are used so frequently that they will often have a fixed place in the store.

As other illustrations of subroutines can be mentioned a routine which can evaluate the square root of a given number, a routine which calculates the cosine of a given angle, a routine which can sort a set of numbers in the store into order etc.etc.

All these routines cannot be permanently stored in the computer due to the limited storage capacity, but it is important that there is a good description of the exact function of the subroutines and that there is easy access to the actual code. If it is desired to use the square root routine, the user has only to copy the routine already written into a suitable place in his own program.

Note: One of the characteristics of a good computing centre is that many well written and easy accessible subroutines are available. It makes programming much easier if many standard operations can be performed by means of subroutines.

1.5 The Binary System

Since many computers use the binary system internally because it is easier from an electronic point of view (many electronic components are bistable, i.e. they have two stable states), we will explore in this chapter the peculiarities of the binary system and study the representation of numbers in this system.

1.5.1 Fixed-Point Numbers

Any positive real number can be written as a finite or infinite binary fraction, i.e. as a sum of positive and negative powers of 2, each power having the coefficient of 0 or 1. The binary point is used in exactly the same way as the decimal point, and binary fractions are written so that they look like decimal fractions except that only the digits 0 and 1 are used.

Example 1.2

In the following examples of the representation of real numbers in the decimal system and the binary system the decimal form is written in the left-hand column and the binary in the right-hand column:

0	0
1	1
2	10
3	11
-4	-100
7	-111
8	1000
11.5	1011.1
1.75	1.11
0.4	0.01100110...

Very simple rules can be created for conversion of one representation to the other and vice versa.

In example 1.2 the negative numbers are represented by their absolute value preceded by a minus; the same system might be used in an electronic computer if a special bit was reserved only to show the sign. However, certain parts of the binary arithmetic become simpler if a negative number is represented by a complement, i.e. the positive number which is obtained by adding a fixed power of two to the given negative number (a power big enough to ensure positive representation for all the negative numbers considered).

In many computers the range of numbers, which the computer has been constructed to handle, is from -1 to +1. All numbers in the computer are represented by a fixed number of binary digits - for instance 4o as in GIER - and the built in arithmetic operations of the computer treat these numbers so that the first digit of the number is treated as a digit before the point and all other digits are treated as digits after the point developed as a binary fraction.

A number with the first digit equal to 1 is regarded by the computer as a negative number whose value is obtained by subtracting 2 from the whole binary fraction.

Example 1.3

Let us consider a computer whose cells have a capacity of 40 digits with the representation of numbers as mentioned above.

If the contents of a cell are	they will be treated as	
	(binary no.)	(decimal no.)
0111.........11	0.11......1	= $1-2^{-39} \doteq 1$
0110.........00	0.110.....0	= 0.75
0100.........00	0.100.....0	= 0.5
00010........00	0.0010....0	= 0.125
0000.........01	0.000....01	= $2^{-39} \doteq 1.82\cdot10^{-12}$
000..........00	0.00......0	= 0
111..........11	1.11......1 -10	= $-2^{-39} \doteq -1.82\cdot10^{-1}$
11110........00	1.1110....0 -10	= -0.125
1100.........00	1.100.....0 -10	= -0.5
10100........00	1.010.....0 -10	= -0.75
100..........01	1.000....01 -10	= $-1 + 2^{-39}$
1000.........00	1.000.....0 -10	= -1

The computer will thus be able to handle all numbers with 39 binary digits after the point within the range $-1 \leqq x < 1$.

The reason for choosing just this way of point placing and this representation for negative numbers is that it simplifies parts of the arithmetic.

For instance the same process will have to be carried out for addition, no matter whether the numbers added are negative or positive.

Multiplication will likewise become very simple since the product of two machine numbers (i.e. fixed point numbers in the range $-1 \leqq x < 1$) also gives a machine number unless both factors are equal to -1.

No matter what representation is chosen addition and subtraction may overload the capacity of the computer, i.e. give a result outside the permitted range.

Example 1.4

If we let the machine mentioned in 1.3 add the numbers 0.75 and 0.75 it will do the following calculation:

```
  01100.......0
+ 01100.......0
  -------------
  11000.......0
```

and will interpret the result as the number -0.5.

Addition of the numbers -0.75 and -0.5 runs as follows:

```
  10100.......0
+ 11000.......0
  -------------
  01100.......0,
```

in other words the result 0.75 = -1.25 + 2. Carry is transferred automatically, the carry from the most significant digit disappearing into thin air. This results in the computer calculating modulo 2, i.e. that results outside the range $-1 \leq x < 1$ are restored to this range by the addition or subtraction of 2.

As seen from example 1.4 there is the risk that the computer presents a wrong result, if its capacity is exceeded. One can naturally (and in many cases ought to) plan the computations in such a way that it does not happen, but many computers are, however, provided with a facility which makes it easy to save the situation in cases like the above mentioned:

Like desk calculators electronic computers have a bigger capacity of digits in the computing registers than in the rest of the computer. There is in these registers space for two digits before the point (and the usual number of digits after the point). If, however, the first digit is equal to 0 the contents of the register are treated as a positive number with the value of the binary fraction. If, however, the first digit is equal to 1, the contents are treated as a negative number, whose value is obtained by subtracting 4 from the value of the binary fraction.

Example 1.5

Let a computer have a computing register with 41 positions in which the representation of numbers is as above.

If the contents of the register are	they will be treated by the computer as the number (binary number)			(decimal number)
01110.......0	1.11		=	1.75
01100.......0	1.1		=	1.5
01000.......0	1		=	1
00110.......0	0.11		=	0.75
000010......0	0.001		=	0.125
111110......0	11.111	-100	=	-0.125
11010.......0	11.01	-100	=	-0.75
11000.......0	11	-100	=	-1
10100.......0	10.1	-100	=	-1.5
10010.......0	10.01	-100	=	-1.75
10000.......0	10	-100	=	-2

The range of values will thus be $-2 \leqq x < 2$.

If we are now going to perform the same additions as in example 1.4, but with two digits before the point we get

```
  binary        decimal              binary       decimal
  00.1100...0 = 0.75                 11.010...0 = -0.75
+ 00.1100...0 = 0.75               + 11.100...0 = -0.5
  ------------------                 -------------------
  01.1000...0 = 1.5                  10.110...0 = -1.25
```

in other words the correct results.

It is apparent from example 1.5 (and it can, by the way, easily be proved) that the machine numbers, i.e. the numbers in the range $-1 \leqq x < 1$, are those numbers which have the digits 00 or 11 before the point in the computing register; if the digits before the point are 01, then the number is in the range $1 \leqq x < 2$, and if the digits before the point are 10, the number is in the range $-2 \leqq x < -1$.

If the computer is going to add two numbers taken from the store (with 1 digit before the point) the sign digits of the two numbers must thus be respectively duplicated. Addition is then performed with two digits before the point.

It can easily be ascertained whether the result is a machine number or not: If the two digits before the point are like, the result is again a machine number, and if they are unlike, the result lies outside the range $-1 \leqq x < 1$ (in such a case we say that <u>overflow</u> has occurred).

If the result is a machine number it can be stored in a cell for later use, and it will be apparent from the above that it is sufficient to store one digit before the point and all digits after the point; the extra digit before the point in the computing register is then forgotten.

<u>Example 1.6</u>

It must be noted that if we add more digits before the point so that the numbers are written with for instance h digits before the point, the range can be extended correspondingly. With the provision that numbers whose first digits equal 0 are regarded positive, while numbers whose first digits equal 1 are regarded negative, with a value as that of the binary fraction minus 2^h, then the range will be $-2^{h-1} \leqq x < 2^{h-1}$. Numbers in the basic range $-1 \leqq x < 1$ are here also characterized by the digits before the point being all zeroes for positive numbers and all ones for negative numbers.

<u>1.5.2 Floating-point Numbers</u>

In many of the computations presented to electronic computers, it is possible, by sensible planning, to keep within a range of numbers as described in the previous section, by using suitable scale factors on the data. In order to use such methods it is very pertinent that the largest and smallest absolute values do not deviate more from each other than the digital capacity of the computer can bear. If, for instance, a computer can store numbers with 4o binary digits the difference between the largest absolute value and the smallest significant absolute value (other than 0) must not exceed 2^{39} as otherwise either the largest numbers will come outside the range or the smallest numbers will be treated as zeroes.

If it is desired to solve a problem where this requirement cannot be fulfilled by the data, it will be convenient to let the computer interpret each number as consisting of a mantissa and an exponent. Once again let the cells in a computer contain 4o bits. The computer may, for instance, be constructed in such a way that it (with certain instructions) reads the first 3o bits as a mantissa, and the last lo bits as a binary exponent so that the contents of a cell are treated as a number, the value of which is the numerical value of the first 3o bits multiplied by a power of 2 (exponent). The exponent is the value of the last lo positions.

A frequently used representation of the mantissa and the exponent is as follows:

a) The 3o bits reserved for the mantissa are regarded a positive or negative binary fraction in exactly the same way as described in 1.5 with the point between the first and the second digit.

b) The 10 bits reserved for the exponent are regarded as an integer (written in the binary system); if the first digit is equal to zero, the exponent is regarded as positive; if the first digit is equal to 1, the exponent is regarded as a negative integer whose value is obtained by subtracting $2^{10} = 1024$.

Let the mantissa be z and the exponent v. The contents of the cell are treated as the number

$$y = z \cdot 2^{v},$$

and we say that the number is written in floating point form or as a floating point number.

Considering the absolute values only the range thus available is

$$y = 0$$
$$2^{-29} \cdot 2^{-512} \leqq y < 1 \cdot 2^{511}$$

which is an enormous increase in relation to the one mentioned in 1.5.1. On the other hand the accuracy is decreased in the respect that each number can only be stored with 3o significant digits against the usual 4o digits, but this price is often paid to have the range extended.

If a larger range is required, it can be obtained at the expense of the accuracy, as 11 digits may be reserved for the exponent and only 29 digits for the mantissa. The range will then be, approximately,

$$2^{-1024} < y < 2^{1023}$$

and, in general, if x digits have been reserved for the exponent, the range (considering absolutes values) will essentially be

$$2^{-2^{x-1}} < y < 2^{2^{x-1}}$$

To make full use of the 3o bits in the mantissa it may be necessary to normalize the mantissa, i.e. the exponent must be chosen in such a way that the absolute value of the mantissa is between 0.5 and 1. This can be done for all numbers except zero, since by separation of suitable powers of two all mantissas (or rather absolute values) can be moulded to a certain binade, i.e. a certain range of the form

$$2^{a} \leqq z \leqq 2^{a+1}$$

With only a fairly small decrease in the range it is thus possible to arrange that all 3o positions of the mantiss contain significant digits. In this respect it should be mentioned that most of the computers which are able to deal with floating point numbers automatically normalize the result of an arithmetic operation (and adapt the exponent accordingly) before it is presented.

2. The Structure of GIER

2.1 Introduction

With regard to later reference we have in this chapter collected information about structure and arithmetic of the GIER-computer. The following will be heavy reading and can partly be skipped to begin with, but should be returned to later as the need arises.

2.2 Ferrite Core Store

The working store of GIER is a ferrite core store with 1024 cells, and each cell consists of 42 bits, which are numbered from 0 to 41 incl.; the first 40 bits, nos. 0-39 can be used for storing numbers in two different ways:

1) In fixed-point arithmetic the built-in arithmetic operations work as if the point has been placed between position 0 and position 1; numbers with 0 in position 0 are numbers in the range $0 \leqq x < 1$, while numbers with 1 in position 0 are regarded as numbers in the range $-1 \leqq x < 0$, i.e. as the value of the binary fraction (a number between 1 and 2) minus 2. This representation can be characterized by the fact that the computer computes modulo 2 in the range $-1 \leqq x < 1$ with 39 binary digits after the point.

2) In floating-point arithmetic any number y is imagined to be written in the form:

$$\begin{aligned} &y = z \cdot 2^{v} \\ &\text{where either } 1 \leqq z < 2 \\ &\qquad \text{or } \quad z = 0 \\ &\qquad \text{or } \quad -2 \leqq z < -1^{1)} \end{aligned}$$

z being a binary fraction while v is an integer. The built-in operations work in such a way that the first 10 bits, pos. 0-9, are treated as the exponent v and the rest of the cell as the mantissa z. If bit no. 0 is 0 the exponent is treated as a positive integer, i.e. $0 \leqq v \leqq 511$, and if we have 1 in pos. 0, as a negative integer in the range $-512 \leqq v < 0$. In the mantissa the point is placed between pos. 11 and 12, while pos. 10 indicates the sign. According to the definition of z pos. 10 and 11 will nearly always be unlike: If the mantissa z is positive we have 01 in pos. 10-11, and if z is negative we have 10; only the mantissa 0 has 0 in both positions.

The mantissa is stored in pos. 10-39 and has thus 28 binary digits after the point.

The range that can be handled in GIER's floating-point arithmetic is as follows:

$$-2 \cdot 2^{511} \leqq y \leqq -(1 + 2^{-28}) \cdot 2^{-512}$$

$$y = 0$$

$$1 \cdot 2^{-512} \leqq y \leqq (2 - 2^{-28}) \cdot 2^{511}$$

or in the usual decimal notation

1) It is always possible to choose z in such a way, and consequently the exponent v is defined uniquely except in the case where z = 0.

c. $7.458 \cdot 10^{-155} <$ abs $(y) <$ c. $1.341 \cdot 10^{154}$ .1)

The two extra bits in each cell are used for marking the numbers in the store. Most of the numbers need not be marked, i.e. pos. 40 and 41 are cleared, but any number may either be

a-marked by placing 10 in pos. 40-41,
b-marked by placing 01 in pos. 40-41 or
c-marked by placing 11 in pos. 40-41.

Example 2.1

If a cell contains the bit pattern

0	1	2	3	9	10	11	12	13	39	40	41
0	1	1	0	0	0	0	0	0	0	0	1

it will in fixed-point arithmetic be read as the number 0.75 and in floating-point arithmetic as zero, in both cases b-marked.

1) Note: In GIER's floating-point arithmetic the number 0 can be treated in two ways: Either always with the mantissa and the exponent part equal to 0 or with the mantissa 0 and the exponent generated during the calculation; in the latter case the addition

$0 \cdot 2^{300} + 1.5 \cdot 2^{225}$

will, for instance, give the result $0 \cdot 2^{300}$. Whether the built-in floating-point operations work in one or the other of the two modes depends on a switch in GIER so that the user can choose between these two modes.

Example 2.2

If a cell contains the bit pattern

```
 0 1 2 3   9 10 11 12 13          39 40 41
|1 1 1 1   1  0  1  1  0           0| 0  0|
```

it will in fixed-point arithmetic be read as the number

$-(2^{-10} + 2^{-12}) \doteq -0.00122$

and in floating-point arithmetic as

$1.5 \cdot 2^{-1} = 0.75$,

both without any marking.

Example 2.3

If a cell contains the bit pattern

```
 0 1 2 3   9 10 11 12 13          39 40 41
|0 0 0 0   0  0  0  0  0        0  1| 1  1|
```

it will in fixed-point arithmetic be read as the c-marked number

$2^{-39} \doteq 1.82 \cdot 10^{-12}$

while in floating-point arithmetic it is meaningless because the mantissa does not belong to the permitted set of numbers.

In practice the first or the last number in a group of numbers (possibly both) will often be marked. See the examples in chapter 7.

Instead of a number, a full-word instruction or two half-word instructions can be stored in each cell; one marker-bit is used for distinguishing between these two cases, while the other marker-bit indicates whether the arithmetic basic operations work in the fixed-point or floating-point mode. The method of representation of instructions in the cell are referred to in 4.9.

2.3 Drum Store

Besides the ferrite core store GIER has a magnetic drum store with 32o tracks each of 4o cells built like the cells in the core store. Transfer of data to and from the drum takes place one track (of 40 cells) at a time. As, however, transfer takes place independent of the arithmetic unit, calculations can be made simultaneously with the transfer. It is, however, a prerequisite that during transfer from the drum to core store none of the 40 ferrite cells read to are affected by or used in the simultaneous calculation. Drum transfer takes place in the following manner: Using a select-instruction (a VK-instruction) the number of the actual drum track is chosen (and placed in a special register, the tk-register. Using a read-instruction, a so-called LK-instruction, transfer is started from the chosen track to a section of 40 consecutive cells in the core store (specified by the address in the LK-instruction). The drum rotates with constant speed, and reading from the track starts immediately, no matter which cell in the track is on level with the reading device. GIER computes automatically that cell of the section of the core store to which the data shall be transferred and continues thus until the whole track has been read.1) At the same time execution of the instructions that follow after the LK-instruction continues. Transfer in the other direction takes place in exactly the same manner.

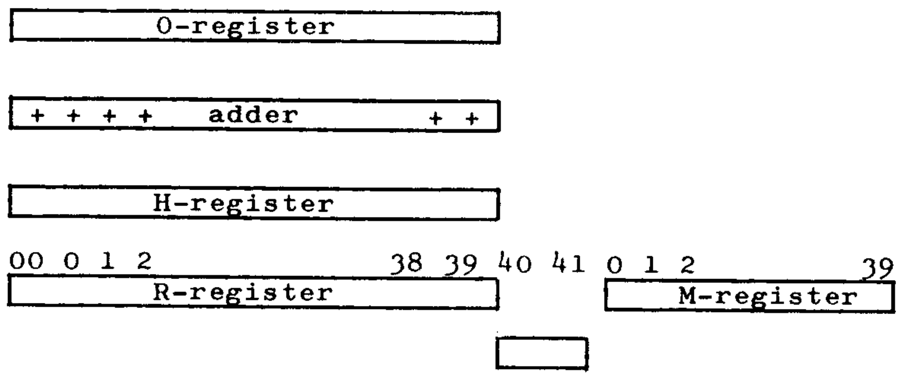
---

1) In many computers transfer is not started until cell no. 1 in the selected drum track is on level with the reading device, in which case there is the possibility that the drum must rotate twice before reading is completed.

2.4 Arithmetic Unit

The central part of the arithmetic unit is the adder with the adjacent, controlling 41-bits-registers, O-register, and H-register. The H-register is of the most interest since the result of an arithmetic operation (and by the way also of an address calculation) is stored here before it circulates in the computer. The H-register (and O-register) differ from the storage cells in that way that they have no marker-bits, but an extra pre-fixed position no. 00. It is thus possible to perform operations with two digits before the point (see 2.5).

The H-register and several other registers in the arithmetic unit are, however, not "available" to the programmer; the programmer uses the accumulator R and the multiplier register M. R consists of 41 positions like the H-register and has besides space for two marker-bits, like the storage cells. The multiplier register M has 4o bit positions like the storage cells, but no marker-bits. The use of these registers will be explained in detail in the next chapter.



2.5 Arithmetic

2.5.1 Fixed-point Addition and Subtraction

Before addition and subtraction one of the operands must be placed in the R-register. The sign of the operand is herewith duplicated, as pos. 0 and 00 in R acquire the same value. When the other operand is taken from the store the sign of this one is also duplicated and the operation is performed with 2 digits before the point (and 39 digits after the point).

This takes place in the H-register after which the result will be transferred to the R-register. As both operands are in the range $-1 \leqq x < 1$, the result will be in the range $-2 \leqq z < 2$, and therefore placed correctly in R with 2 digits before the point, taken modulo 4 within the range $-2 \leqq z < 2$ (see chapter 1 concerning the binary system). If the result is in the interval $-1 \leqq z < 1$, bits no. 00 and 0 will be like (11 for negative numbers and otherwise 00), while they are unlike if the result falls outside this range (01 for results $\geqq 1$ and 10 for results < -1); in the latter case we refer to overflow, because the result cannot be stored in a cell without further treatment.

Information on overflow is stored in a special register (see 2.6). The overflow situation is, however, under control, since there is a GIER operation (a shift) which moves all bits in R one position to the right (at the same time keeping pos. 00 and rounding off pos. 39).

If this operation is performed after an addition or a subtraction pos. 0-39 in R will always (no matter if there was overflow or not) contain half the sum or difference (except possible rounding-off errors), and this number can be stored and treated in the usual way. We note that the M-register is not used in fixed-point addition and subtraction.

2.5.2 Fixed-point Multiplication

The exact result of a multiplication of two numbers with 39 digits after the point is a number with 78 digits after the point, and as the user now and then needs all 78 digits,but often only the first 39, GIER has two built-in multiplication operations: short multiplication and long multiplication.

For both types of operation one factor must be placed in the M-register in advance. The multiplication operation itself takes the other factor from a cell. The two types of operation will now be treated separately.

In short multiplication the product is calculated and then rounded-up to 39 binary digits after the point (if bit no. 40 of the product is equal to 1); afterwards the contents of the R-register are added to the rounded product1). This operation is known as accumulating multiplication.

The result is placed in the R-register, while the first factor remains in the M-register. If both factors are not equal to -1, the product will again be a number in the range $-1 \leqq z < 1$, but in the final addition overflow may occur in exactly the same way as described in section 2.5.1.

If both factors are equal to -1, the multiplication gives the binary number 01.00.....0, which is then added to the contents of the R-register. If the latter contains a negative number (between -1 and 0), the addition gives the correct answer without overflow. On the other hand, if the number in the R-register is in the range $0 \leqq z < 1$, the addition gives the correct answer in the R-register (with two bits before the point), with overflow, however. In other words the result itself cannot be stored in a cell.

The rules for short multiplication are thus:

As long as no overflow occurs in the R-register before the multiplication, the R-register will on completion of the operation contain the correct result - possibly with overflow - in which case the situation can be regulated by means of a right-shift as with ordinary addition or subtraction.

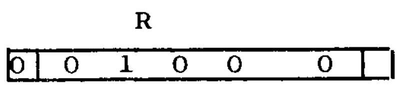
---

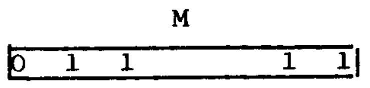
1) If the product itself is required the R-register can be cleared previously.

In long multiplication the product is calculated with 2 digits before and 78 digits after the point; to this are added the contents of the R-register multiplied by 2^{-39} (i.e. shifted 39 places to the right). The result, with 2 digits before the point and the first 39 digits after the point, is placed in the R-register, and with the 39 last digits in pos. 1-39 in the M-register. At the same time pos. 0 in M is cleared. In this case the result will always be correct, and unless there is overflow in R before the operation, overflow can only occur if both factors are equal to -1.

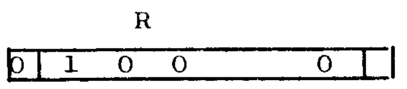
Example 2.4

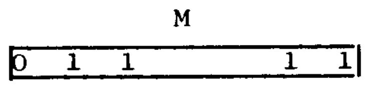
Let the R-register contain the number 0.5 and the M-register the number $+1 - 2^{-39}$.



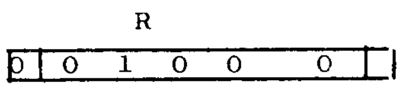


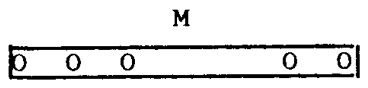
In short multiplication by the number +0.5, the result will be +1 due to the round-off, and R and M look like this:





If we instead use long multiplication by +0.5, the result will be +0.5.





since the actual multiplication gives $0.5 - 2^{-40}$, after which is added $0.5 \cdot 2^{-39} = 2^{-40}$.

2.5.3 Fixed-point Division

GIER has a short and a long division corresponding to the two forms of multiplication: In long division the contents in the long register consisting of R and pos. 1-39 in M are used as the dividend, and in short division the contents of R (or rather the contents of R followed by 39 zeroes) are used as the dividend. The two forms of division work, otherwise, in the same manner. The dividend must be placed in the register (the long register R,M and the short R) in advance, after which the divisor is taken from the store in the actual division operation. <u>Provided that the quotient q is in the range</u> $-2 \leqq q < 2$, the division is performed as follows:

1) Positive divisor d: The computer calculates the quotient that gives a remainder r in the range $0 \leqq r < d \cdot 2^{-39}$.

2) Negative divisor d: The computer calculates the quotient that gives a remainder r in the range $d \cdot 2^{-39} \leqq r < 0$.

In other words the division is made in such a way that the remainder always has the same sign as the divisor.

Then the quotient is placed in the R-register, and the appropriate remainder multiplied by 2^{39} is placed in the M-register.

If the quotient in fixed-point division would fall outside the range $-2 \leqq q < 2$, the division operations will be performed, but the results that appear in the computer will <u>not</u> be the quotient and the remainder. It must be regarded a serious <u>error</u> to use fixed-point division, if the user is not absolutely sure that the quotient will fall between -2 and 2.

<u>Example 2.5</u>

Unless there is overflow after a division the quotient q is in the range $-1 \leqq q < 1$.

If there is overflow and the user is assured that the division is "legal" the quotient will fall between -2 and -1 or between 1 and 2. By means of a right-shift in the R-register it will be possible to go on with half the quotient.

<u>Example 2.6</u>

If the divisor is negative, the remainder must always be < 0 and, in the rare case where the dividend is exactly divisible, the remainder in M must also be equal to the divisor (i.e. < 0), and the quotient adapted accordingly. Whereas, if the divisor is positive the correct quotient is always obtained.

For example let the dividend $9 \cdot 2^{-78}$ be placed in the long register. If this is divided by $3 \cdot 2^{-39}$ by means of long division the quotient $3 \cdot 2^{-39}$ will be put in R and the remainder 0 in the M-register. On the other hand, if $-9 \cdot 2^{-78}$ is divided by $-3 \cdot 2^{-39}$, the quotient in R will be $2 \cdot 2^{-39}$, and in M will be $-3 \cdot 2^{-39}$ (which corresponds to a remainder in the long register of $-3 \cdot 2^{-78}$. It may therefore sometimes be unwise to use negative divisors.

2.5.4 Floating-point Operations

The built-in floating-point operations use the R- and M-registers as one register. The mantissa z of a floating-point number is always placed in pos. 10-39 in R, and at the same time bits nos. 00-9 are made equal to bit no. 10 so that the sign of the mantissa can be read in pos. 00. The exponent v of a floating-point number is placed in pos. 0-9 of the M-register corresponding to its location in pos. 0-9 of a storage cell.

The R-register supplemented with pos. 0-9 in the M-register is called the floating-point accumulator RF, and it is the only arithmetic register accessible to the programmer when using floating-point arithmetic. The remainder of the M-register is used internally during every floating-point operation.

In all four basic arithmetic operations one operand must be placed in RF in advance. During the actual operation the other operand is taken from the store, and the result is placed in RF.

Furthermore, when adding or subtracting, pos. 10-19 of M are used for storing information about possible loss of significant digits.

In addition or subtraction of two floating-point numbers with the same exponent, there is the possibility that the resulting mantissa has a very small absolute value.

As, however, the mantissa and the exponent are adjusted in such a way that the absolute value of the mantissa falls between 1 and 2, the mantissa is sufficed with a string of unsignificant zeroes (to fill out pos. 10-39); the amount of these (or in other words, the amount of left-shifts necessary to normalize the resulting mantissa) is placed in pos. 10-19 in the M-register as an integer with unity in pos. 19 1).

In case of overflow after a floating-point operation, i.e. if the result is outside the floating-point range (see 2.2), GIER gives up and jumps to cell 0, where a stop instruction or a jump to a control program must be placed.

2.6 Control Unit

In the control unit there is a large number of registers, but it is not necessary to know all about them to understand the function of GIER or use GIER; we are here going to mention the most important registers.

The Instruction Register F is a 42 bit register, which contains in pos. 10-41 the current instruction excluding the address. After a normal stop (programmed or push-button controlled) F_{10-41} contains, however, the next instruction to be executed.

---

1) In division the absolute value of the quotient of the mantissa must fall between 0.5 and 2. This means that at the most one binary digit will be lost, and neither here nor in multiplication will information be stored about this, but since pos. 10-39 are cleared before any floating-point operation, information about loss of digits must be stored or used immediately after the operation, in which the loss of digits occurs.

Pos. 0-9 makes up the so-called p-register or index-register. The contents of this are determined by the programmer through special operations, and are used in the calculation of indexed addresses. A number of 10-bit registers are gathered around F for use in determination of addresses:

Control Counter r1 - contains during execution of an instruction and at normal stop the address of the next instruction to be executed.

Address Register r2 - contains the address of the cell in the ferrite core store, on which the control unit is operating. (It is the contents of this register that are used in the relative addressing mentioned in 3.4.3).

Subroutine Register s1 - contains normally the address, from which the last subroutine jump has been performed (see operation list): this address is used in determining subroutine-indexed (s-modified) addresses (see 3.4.4). The register may be used for other purposes by the programmer since it is both possible to store its contents and to replace its contents. Another register s2 is used as an auxilary register for determining addresses, a.o. when s-modification is used together with indirect addressing (determination of addresses is referred to in 3.4-3.6 and 4.2).

Drum Track Register tk - contains the number of the track, to which the ferrite store is connected and thus controls, on which track writing (and reading) takes place. The contents of tk can be changed by a VK-instruction, which selects a new track.

Drum Address Register ta - is used during drum transfers to hold the addresses of cells in the appropriate section of the ferrite core store where transfer is taking place (see 2.3), but has otherwise no interest for the programmer.

Indicator Register in - is a collection of 12 1-bit registers, which can be used independent of each other for storing information about overflow, zero results, sign, and marking for later use. The use of this important register will be explained in detail in 4.6 and after.

Overflow Register 0 - is a 1-bit register which is set equal to 1, when overflow occurs in an arithmetic operation, otherwise 0. The contents of 0 can by means of an indicator operation (see chapter 5) be stored in the indicator for later use, but remains unchanged until the next arithmetic operation (including number shift and cyclic shift in R). Notice that the 0-register corresponds to the overflow situation in the H-register, not in the R-register; however, it only makes difference in the case of two operations (i.e. addition to and subtraction from the storage cells, the AC- and SC-instructions, where the overflow situation for the result will be registered).

A 10-bits peripheral unit register by contains the identification numbers of the peripheral units connected. This register controls which units (see 2.7) will be activated by a standard input/output instruction (LY/SY). The contents in by can be changed using a VY-operation, which selects a new peripheral unit.

2.7 Peripheral Units

For input GIER has a dielectric 8-channel paper tape reader, which reads one character at a time to a cell and to the R-register, with a speed of about 1000 chars./sec. In addition there is a typewriter from which one character can be read in at a time.

For output GIER has an 8-channel paper tape punch, which punches one character at a time with a speed of about 150 chars./sec. The same typewriter as mentioned above can also be used for output, and the speed here is about 8 chars./sec.

Furthermore, GIER can be equipped with a punched card reader and a high speed printer, see volume II.

2.8 Control Panel

A photo at the back of volume II illustrates GIER's control panel, which consists of two rows of display lamps with corresponding push-buttons, ignition key, two sets of start-stop switches, and a volume control for the loud-speaker. The significance and effect of the different lamps and switches are referred to in a later chapter concerning general "operating instructions".

Another photo illustrates the little panel with error lamps and with the HP-button for activating an interrupt function (see volume II).

3. THE GIER INSTRUCTION I

3.1 Introduction

In this chapter we start considering the external instruction code, by which we understand the program code, which is to be written down by the GIER user. The notation is depending on the input program used; the notation used in the following is in accordance with the conventions for the input program SLIP, although the rules here are stricter than actually necessary for SLIP (see volume II). In the first place we will refer solely to the structure of the basic instruction, and reference is limited to the simplest type of instruction, namely the _half-word instruction_.

3.2 The Basic Instruction

An instruction always contains information on the _basic operation_ to be performed and an _address_.

In the simplest case the instruction does not contain any more elements. As mentioned in chapter 2, the instructions are to be placed in cells in the store, and we are therefore going to refer to the _address part_ of an instruction and a cell, as being that part of the instruction or cell which contains the address.

3.3 Basic Operations

Corresponding to the 57 basic operations performed by GIER, 57 pairs of letters have been chosen. If performance of a specific basic operation is required the corresponding pair of letters is written as the first part of the instruction. In chapter 5, there is a list containing all 57 basic operations and the pair of letters which must be used as codes for the various operations. For instance, the operation "subtract from a cell" is written as SC.

We note by the way that the description "subtract from a cell" is not very precise. In chapter 5 we consider the 57 basic operations and for each operation is thoroughly explained,what happens in GIER.

3.4 Address

The address of an instruction can be specified in several different ways, namely as absolute address, indexed address, relative address, and subroutine-indexed address respectively.

3.4.1 Absolute Address

An absolute address is specified by an arbitrary integer h, and the effect is, in most operations, that h - taken modulo 1o24 - is regarded as the address of the operand. h is called the address constant.

Example 3.1

The address 714 in the instruction AR 714 will cause the number in cell 714 to be added to the contents of the accumulator.

Example 3.2

The address -15 in the instruction MK -15 will cause the number in cell -15+1024, i.e. the number in cell 1009 to be multiplied by the contents of the multiplier register after which the contents of the accumulator are added to the product.

3.4.2 Indexed Address

An indexed address is written by means of the letter p followed by an arbitrary integer h. The effect of indexing is, in most operations, that the sum of the contents of the index register and the address constant h - taken modulo 1024-is regarded as the address of the operand. The contents of the index register are always an integer in the range $0 \leq p \leq 1023$.

Example 3.3

Let the contents of the index register be 42. Consequently the address p+24 in the instruction SR p+24 will cause the number in cell 66 to be subtracted from the contents of the accumulator.

Example 3.4

Let the contents of the index register be 127. The instruction SC p-212 will first cause 127 - 212 + 1024 to be regarded as the address of the operand, after which the contents of the accumulator are subtracted from the contents of cell 939.

3.4.3 Relative Address

A relative address is written by means of the letter r followed by an arbitrary integer h. Let n be the number of the cell which contains the instruction with the relative address. The effect of the relative address will normally be that the sum n + h, taken modulo 1024, is regarded as the address of the operand.

Example 3.5

Let the current instruction be AR r - 7, and let this instruction be located in cell no. 72. 72 - 7 = 65 will then be regarded as the address of the operand, i.e. that the number in cell 65 is added to the accumulator.

Example 3.6

In cell 973 the instruction AC r + 566 is placed. In this instruction 973 + 566 - 1024 = 515 will be regarded as the address of the operand, i.e. that the number in cell 515 is added to the number in the accumulator.

3.4.4 Subroutine-indexed Address

A subroutine-indexed address is written by means of the letter s followed by an arbitrary integer h. The effect of a subroutine-indexed address is that the contents of the subroutine index register are added to the number h, after which the sum, taken modulo 1024, is regarded as the address of the operand1). The contents of the subroutine index register are always an integer in the range $0 \leqq s \leqq 1023$. The use and utility of this way of addressing will be mentioned later on.

3.5 Indirect Addressing

In practice the address of the cell that contains the desired operand will often be in the address part of another cell. It is, in fact, possible to get hold of such an operand, very simply by writing a bracket round the address part of an instruction. The procedure can be illustrated by means of the following examples.

Example 3.7

Let the desired operand be in cell 715, and let the number 715 be the value of the address part of cell no. 43. If we now write an instruction with the address part (43), the bracket will indicate that the number 43 is not regarded as the address of the operand,

1) If the address of the desired operand is equal to the contents of the subroutine index register one may write either s or s + 0, which will have the same effect. For relative and indexed addresses the same is the case.

but that the address of the operand is found in the address part of cell 43. The address (43) will in this case have the same effect as the address 715.

<u>Example 3.8</u>

Let the instruction SR (r - 6) be in cell 862. At first the address 862 - 6 = 856 is formed, and due to the bracket the number 856 the number 856 is regarded as the address of the cell, in which the address of the operand is stored. If the address part of cell 856 is 14, the operand will be taken from cell 14.

<u>Example 3.9</u>

Let the contents of the index register p be 38, and let an instruction have the form of MK (p +10). At first the address 38 + 10 = 48 is formed, and due to the bracket it is regarded as the address of the cell containing the address of the operand. If the address part of cell 48 is r + 2, the operand will be taken from cell 50.

<u>Example 3.1o</u>

Indirect addressing is <u>recursive</u>, i.e. if the address found through an indirect address is indirect itself, yet another link will be involved before the resulting address is determined. In that case we talk about a <u>chain of brackets</u> or indirect addresses.

Let the address part of the current instruction be (139), and let the address part of cell 139 be (20). Finally, let us assume that the address part of cell 20 is 888. In this case the original address part (139) will cause the operand to be taken from cell 888.

N.B. If such a chain of brackets is closed, i.e. if it indirectly refers to itself, the instruction will never be fulfilled because GIER will go on looking for the final address.

3.6 Modified Address

Example 3.9 shows that when indirect addressing is used the ultimate address is calculated on the basis of a completely new address part using the r-value corresponding to the location of the new address part. The number thus determined is called the modified address (in chapter 4 it will turn up that it does not need to be the final address):

The modified address is defined as that integer h in the range $0 \leq h \leq 1023$, which is determined by GIER on the basis of the elements of the address part in the last link of the chain of brackets. For all instructions mentioned in this chapter the modified address is the same as the final address, i.e. the number that shows from which cell the operand must be taken.

Example 3.11

Let the instruction AR r - 48 be in cell 2o. The final address is then 20 - 48 + 1024 = 996.

Example 3.12

Let the contents of the index register p be 900, and let the contents of cell 276 be AR 27. The instruction MK (p + 400) has thus the final address 27 (found in cell 276).

3.7 S-Modification of the Basic Operation

After the two letters used for specification of a basic operation the letter S may be added. In this case the accumulator will be cleared before the basic operation is performed[1].

1) i.e. that pos. 00-39 in the R-register are cleared, while the marker-bits (nos. 40-41) remain unchanged.

Example 3.13

The instruction ARS 439 will cause the accumulator to be cleared, after which the contents of cell 439 are added to the accumulator. The effect of the instruction can in short be described as a transfer of the contents of cell 439 to the accumulator.

3.8 F-Modification of the Basic Operation

After the two letters used for specification of a basic operation the letter F may be added. This is, however, only of importance for the arithmetic operations, which are performed in floating-point mode, when F-modified. Other operations are not changed by F-modification.

Example 3.14

The instruction SRF 45 will cause that the contents of cell 45 are subtracted from the contents of the accumulator, both regarded as floating-point numbers.

F-modification and S-modification may be used simultaneously and here the S-modification causes the whole floating-point accumulator RF to be cleared[1]. The order in which S and F are written is unimportant.

Example 3.15

The instruction ARSF 137 will cause that the number in cell 137 regarded as a floating-point number will be tranferred to the floating-point accumulator RF.

---

1) RF consists of the accumulator R, excluding marker-bits, and pos. 0-9 of the M-register.

In conclusion the diagram below shows the possible elements of a half-word instruction:

Operation Code	S F	Address
Always two letters. 57 different possibilities all mentioned in chapter 5.	S or F or S as well as F may be excluded	Always an address. 8 different possibilities: h, p+h, r+h, s+h, (h), (p+h), (r+h) or (s+h), where h stands for an arbitrary integer, while the letters p, r, s must be written as such.

A half-word instruction can be stored so that it fills only half a cell in the store. Two half-word instructions can be stored in the same full-cell. If this way of storing is used, a special condition must be mentioned, namely that the F-modification of a half-word instruction also causes the other half-word instruction in the same cell to be F-modified.

4. THE GIER INSTRUCTION II.

4.1 Introduction

There is nothing to prevent programs being written using half-word instructions, but it will normally be advantageous to use the possibility of extending half-word instructions to full-word instructions. This extension can be done in several ways, namely by addition of an increment or by means of X-modification, V-modification or D-modification. Several of these (or all) possibilities may be used at the same time.

Besides these extensions of the instruction there is a further possibility of extension, i.e. by inclusion of an indicator part. This will be mentioned as the last point.

4.2 The Increment

The increment in an instruction is specified by means of an arbitrary integer. The increment must be noted at the end of the instruction and may be separated from the address part by the letter t. If the increment is written with sign, the separating t is superfluous (see also volume II, chapter 11). The effect of the increment is that it is added (mod 1024) to the modified address before execution of the instruction. It is emphasized that as long as the address is not indirect, the actual instruction is altered, as its address constant will be the sum of the increment and the former address constant.

If the address is indirect the alteration occurs in the instruction containing the last link of the chain of brackets (it is the address part of precisely <u>this instruction</u> that is used in determining the modified address). We refer by the way to the following examples.

<u>Example 4.1</u>

The instruction AR 814 t 2 will prior to execution be altered to AR 816 t 2, after which the contents of cell 816 are added to the contents of the accumulator. In practice instructions with increments are normally used only when these instructions are to be run through several times. With the next execution of the instruction mentioned it will be changed to AR 818 t 2, after which the contents of cell 818 are added to the contents of the accumulator. It may occur that a single instruction with increment - by repeated use - can be made to sum a whole series of numbers.

<u>Example 4.2</u>

Let the instruction SRF r+50 t-1 be in cell 124. When the instruction is executed for the first time it will be changed to SRF r + 49 t-1, and the number in cell 173 is subtracted from the contents of the accumulator. Due to the F-modification the arithmetic is in the floating-point mode. When the instruction is executed for the second time it will be altered to SRF r + 48 t-1, and the number in cell 172 is subtracted from the contents of the accumulator etc. etc.

<u>Example 4.3</u>

Let the address part of cell 289 be 837. The instruction MK (289) + 1 will, when first executed, cause the address part in cell 289 to be increased by 1 to 838, after which the contents of cell 838 is regarded as the multiplicand. (In the case of two half-word instructions being in cell 289 there will be two address parts in this cell. If so, then the address in the first half-word instruction only will be altered). With the next execution of the instruction the address part in cell 289 is increased again (to 839) and so on. Note that the actual instruction MK (289) + 1 will not be affected, but the alteration takes place in the address part of cell 289. Furthermore note that in case of the address in 289 again being indirect, incrementing will not take place until an address part without brackets is found. In other words, the increment alters only the address constant in the last link of a chain of bracketed addresses.

Example 4.4

Let the contents of the index register p be 72. The instruction AR (p + 7) + 2 will then - before performance of the basic operation - cause the address of cell 79 to be increased by 2 each time the instruction AR (p + 7) + 2 is executed (provided that the address of cell 79 is not indirect).

Example 4.5

Let cell 34 and 35 contain the instructions:

```
34: AR (35) +1
35: SC 218  -2
```

The first instruction is executed as AR 219, and at the same time the second instruction is altered to SC 219 -2. The second instruction is then executed as SC 217, at the same time being altered to SC 217 -2.

It should be noted that the workings of certain special half-word instructions depend upon the fact that GIER always gives a half-word instruction the increment number 0 (see operation list in chapter 5).

4.3 X-Modification of the Basic Operation

After the address the letter X can be added. This has the effect that the contents of the accumulator and the multiplier register are exchanged after performance of the basic operation. This means that the contents in pos. 0-39 in the two registers are exchanged,after which bit no. 00 of R is put equal to (the new) bit no. 0.

Example 4.6

The instruction SR 445 X will cause the contents of cell 445 to be subtracted from the contents of the accumulator, after which the contents of the accumulator and multiplier register change places.

4.4 V-Modification of the Basic Operation

After the address the letter V can be added. V-modification does not affect the instruction itself, but, instead, causes the next whole cell, that would otherwise be regarded as an instruction, to be skipped.

Example 4.7

Let the instruction ARS 705 V be in cell 212. After execution of this instruction GIER continues with the instruction in cell 214. The interjacent cell 213 may, for instance, be used for storing interim results.

Example 4.8

A V-modified instruction must always be placed in a whole cell, and the next whole cell is skipped no matter whether it contains a full-word instruction or two half-word instructions (or a number).

4.5 D-Modification of the Basic Operation

After the address the letter D can be added. The effect of the D-modification depends on the actual basic operation. In the operation list the D-modification for each operation is described separately. In general, the final address of a D-modified instruction will be used as the operand in arithmetic operations (contrary to the normal case, where the address indicates where to find the operand).

Example 4.9

The instruction AR 309 D causes the number 309 itself to be regarded as an operand so that 309 will be added to the address constant of the accumulator (positions 00-9), while the positions 10-39 of the accumulator remain unchanged.

Example 4.10

If the address in a D-modified instruction is indirect, the address refers to that cell, the address part of which will be regarded as an operand. Let the instruction in cell 444 have the address part 615. The instruction AR (444) D+7 will then cause the number 622 to be added to the address constant of the accumulator (moreover the address constant of cell 444 will become 622), because the increment is first added in the usual way, after which the D-modified operation is performed.

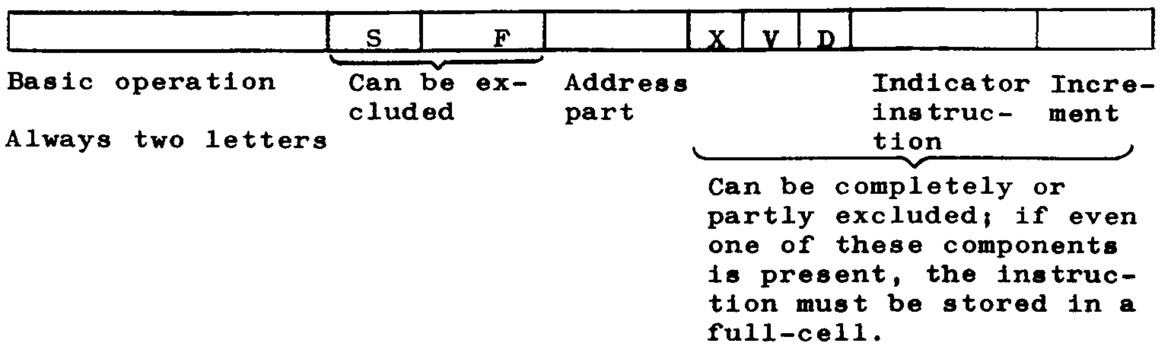
X-, V-, and D-modifications can be used simultaneously (in other words, any two or three of these modifications can be used in the same instruction). The order, in which the letters X, V, and D are written, is of no importance.

Example 4.11

We imagine the instruction ARS 42 XVD +1 stored in cell 100. Execution of the instruction will cause the following things to happen:

1) The instruction is altered to ARS 43 XVD +1
2) The accumulator is cleared
3) The number 43 is added to the address constant of the accumulator (which is zero)
4) The accumulator and the multiplier register are interchanged
5) Next instruction is taken from cell 102.

The following diagram shows the different possibilities for writing an instruction. The indicator-instruction is mentioned in the following pages.


S
F
X
V
D
Basic operation
Always two letters
Can be excluded
Address part
Indicator instruction
Increment
Can be completely or partly excluded; if even one of these components is present, the instruction must be stored in a full-cell.

S and F are interchangeable. X, V, and D can be written in any order.

4.6 Instructions with Indicator-Instructions

The last type of component of an instruction is an indicator-instruction. Before we examine the many different - and often very useful - applications of the indicator, we must, however, point out that in many cases excellent programs can be made without use of the indicator.

Therefore, it is recommended that the reader goes through the operation list very carefully and works out the first examples from chapter 7 before studying the list of indicator-instructions and their effect in chapter 6. Before we describe an indicator-instruction we must remind the reader of the so-called indicator register.

4.6.1 Indicator Register

The indicator register which is part of GIER's control unit (see chapter 2) consists of 12 1-bit registers, each of which can be used for storing information about a certain state or a certain result for later use in the administration of a program.

OA	OB	TA	TB	PA	PB	QA	QB	RA	RB	KA	KB
overflow or zero in R		sign		a- and b-marking						can only be set from the control panel	

The above diagram shows the names of each register, and the text indicates which kind of information can be stored in the various parts.

4.6.2 Indicator-Instruction

The indicator-instruction consists of an indicator operation and an indicator address1). Both components are specified by means of capital letters. The indicator-instruction must (together with X, V, and D-modifications) be written after the address part and before a possible increment.

An instruction can at the most contain one indicator-instruction.

1) in a particular case no indicator address is written (see the list of indicator-instructions chapter 6).

4.6.3 Indicator Operation

The indicator operation which is placed first in the indicator-instruction is written with one of the four letters I, M, N, and L.

I causes a bit of information to be stored in the indicator for later use. The indicator address decides which kind of information to be indicated (see next chapter).

M causes (in some of the basic operations) a certain marking of the operand cell, i.e. that cell in the store, to which the final address refers. The indicator address decides the kind of marking.

N and L cause the actual instruction to be conditional, i.e. the basic operation is only performed if a certain condition (either in the indicator or in the arithmetic unit) is fulfilled. Again, the indicator address indicates the condition.

4.6.4 Indicator Address

The indicator address is written as one or two letters immediately after the indicator operation. As indicator address may be used the names of the twelve registers in the indicator OA, ..., KB, and each of these indicator addresses refer to the corresponding register. The indicator addresses OC, TC, ..., KC refer to both OA and OB register, both TA and TB register etc. etc. in the indicator, respectively. Finally the indicator addresses A, B, and C alone refer to one or two marker-bits in the operand cell or in the R-register.

The detailed effect of all possible indicator-instructions are described in the list of indicator-instructions, section 6.1.

4.7 Internal Instruction Format

A full-word instruction or two half-word instructions may - as mentioned - be placed in a cell in the store. Occasionally it is important to the user of GIER to know how the different components of the instruction(s) are distributed over the 42 bits that form a cell. Furthermore, it may be necessary for the programmer to know the value of the 42 bits corresponding to a certain instruction. Therefore, we finish these chapters about the structure of the instructions by giving a survey of the internal instruction format.

4.7.1 Conversion from the External Instruction to the Internal Instruction Format

During input of an instruction written in accordance with the above-mentioned conventions the instruction is converted before it is stored as a combination of ones and zeroes in the 42 positions of a cell. This conversion is performed by means of the input program SLIP, which has been stored in the computer in advance. The input program treats the single components of an instruction separately. The following diagram shows the bits used for storing the separate parts of a full-word instruction .

4.7.2 One Full-Word Instruction

0..9	10..19	20..25	26	27	28 29	30	31	32	33 34	35 36 37	38 39	40	41
address constant	increment	basic operation	S	( )	r, s and p	X	V	D	indicator operation	K Z O T P Q R	A B C	O	F
										indicator address		(zero)	

Each bit is occupied in accordance with the following rules:

Pos. 0-9	contain the address constant written as an integer in the binary system.
Pos. 10-19	contain the increment written as an integer in the binary system.
Pos. 20-25	are used to indicate the baisc operation contained in the instruction. The six bits permit, in fact, the use of $2^6 = 64$ different operations. As mentioned only 57 of these possibilities are utilized in the standard GIER computer. A survey of the relationship between the basic operations and the bit-patterns can be found in chapter 8.
Pos. 26	has the value 1, when the instruction is S-modified, otherwise the value 0.
Pos. 27	has the value 1, when the instruction is indirect, otherwise the value 0.

Pos. 28 and 29 are utilized as follows:

address	contents of pos. 28-29
absolute	0 0
relative	1 0
indexed	1 1
subroutine-indexed	0 1

Pos. 30	has the value 1, when the instruction is X-modified, otherwise the value 0.
Pos. 31	has the value 1, when the instruction is V-modified, otherwise the value 0.
Pos. 32	has the value 1, when the instruction is D-modified, otherwise the value 0.

Pos. 33 and 34 are utilized to indicate the indicator operation as follows:

Indicator Operation	Contents of pos. 33-34
No indicator operation or I	0 0
M	0 1
N	1 0
L	1 1

Pos. 35, 36, 37, 38, and 39 are used to represent the indicator address as follows:

First part of the indicator address	Contents of pos. 35-36-37
no address	0 0 0
K	0 0 1
Z	0 1 0
O	0 1 1
T	1 0 0
P	1 0 1
Q	1 1 0
R	1 1 1

Second part of the indicator address	Contents of pos. 38-39
no address	0 0
A	1 0
B	0 1
C	1 1

Pos. 40 For full-word instruction this bit must have the value 0

Pos. 41 has the value 1, when the instruction is F-modified, otherwise the value 0.

4.7.3 Two Half-Word Instructions

0...9	10..19	20..25	26	27	28 29	30..35	36	37	38 39	40	41
address constant of 1st instruction	address constant of 2nd instruction	basic operation of 1st instruction	S of 1st instruction	( ) of 1st instruction	r, s, and p of 1st instruction	basic operation of 2nd instruction	S of 2nd instruction	( ) of 2nd instruction	r, s, and p of 2nd instruction	1	F

One can see from the above diagram how each position is occupied. Special attention is drawn to the fact that pos. pos. 40 must have the value 1 in the case of two half-word instructions stored in the cell. As mentioned earlier an F-modification of either of the two instructions will affect both instructions.

The examples below will illustrate how important it is that the user of GIER is familiar with the internal format of instructions.

Example 4.12

At a certain stage of a calculation the user wants to change the address of a certain instruction. This can be done by means of another instruction with the basic operation GA (store address constant). However, it is decisive to know here whether this will affect possible indirect addressing, relative addressing etc.etc. Furthermore, the operation will only have the desired effect when used on a full-word instruction or on the first of two half-word instructions.

Example 4.13

If it is desired to X-modify an instruction of a cell this can be done by placing 1 in pos. 30 in the cell concerned. The basic operation AB (add Boolean) is very suitable for this purpose.

Finally we point out that the internal instruction format is characteristic for the computer, as its circuits have been built with respect to this format. The external instruction code can be selected in many different ways as long as a special input program is made for each selection.

5. OPERATION LIST

5.1 Introduction

In the last section of this chapter we consider the 57 basic operations in GIER one by one and in 5.3 there is a direction for the operation list; first of all, however, we will consider the execution of a complicated GIER-instruction; additionally some of the approximate operation times are mentioned.

In chapter 6 there is a complete list of the possible indicator-instructions and their effects.

5.2 Execution of the Instruction

A GIER-instruction may have many elements (the complicated instruction SCS (r-18) XV IOA +4 is a good illustration). The diagram below specifies the order in which these elements are processed and the time necessary to process them (the basic unit of time is 1 microsecond = 10^{-6} seconds).

Step	Duration
1 Search for conditional indicator operations (N or L)	
Condition not satisfied: Next instruction is begun after	15 µs
Condition satisfied or no N or L operation	no extra time
2 Determination of final address incl. relative, indexed, subroutine-indexed addresses, excl. increment and indirect addressing	27 µs
increment $\neq 0$: additional time	+ 9 µs
indirect addressing, without s-indexing - - per link	+ 12 µs
Indirect addressing, with s-indexing - - - -	+ 26 µs
3 Execution of S-modification	no extra time
4 Performance of basic operation (or F- and D-modification) incl. the necessary marking of overflow	dependant on the basic operation
5 Execution of indicator operations I or M and V-modification	no extra time
6 Execution of X-modification additional time	+ 4 µs
7 Execution of IK indicator operation additional time	+ 6 µs

The time necessary to perform the actual basic operation can, in many cases, only be stated as the average since the numbers themselves affect the operation time: It is faster to multiply with the digit 0 than with the digit 1, and the total operation time for a multiplication is dependant on the amount of ones and zeroes in the multiplicator. Thus the D-modification of an operation may in some cases be faster than the basic operation. With these reservations we give in the below diagram a survey of the time (point 4) necessary to perform the different basic operations.

	Basic Operation µs	F-modification µs
QQ PS-PP VK	c. 2	
GR-GM-GA-GT-GP-GS-GI-GK PM-PA-PT BS-BT-UD-XR LK-SK (see below) HV-HH-HK	c. 9	only GR : c. 9
IS-IT-NS-NT-PI MB HS-VY	c.16	
AR-AC-AN-SR-SC-SN MT-CA-NC HR-LY	c.22	c.66 +2.2·exponent diff. +2.2·no of lost digits
AB-CM	c.27	
TK-TL-NK-NL-CK CL	c.20+2.2 per shift 4.4 per shift	c.30+2.2 per shift
MK-ML	c.155 (average)	c.140 (average)
DK-DL	c.240 (average)	c.190 (average)
SY	c.31	

It will take 2o millisec. before the LK and SK operations are completed, but after 9 microsec. GIER can execute other instructions simultaneously with the drum transfer; however, execution of these other instructions is delayed by 1 millisec. altogether during the drum transfer.

An ordinary addition or subtraction without indirect addressing or increment takes about 50 µs. An ordinary multiplication takes on the average 180 µs, and an ordinary division takes on the average 270 µs. Floating-point multiplication or division is a little faster than the corresponding operations with fixed point, and floating-point addition or subtraction is about twice as slow as the corresponding fixed-point operations.

5.3 Explanation of the Operation List

On the following pages the effect of each basic operation is described in detail. Expressions similar to those used in ALGOL are used in the descriptions, and the meaning of the used names will be established in the following:

c	means the *final address*, determined from an address and possible modification by means of index, s-index, relative and indirect addressing and increment. The cell which c refers to is called the *operand cell*. c, when referring to a storage cell, is regarded as a number in the range $0 \leq c \leq 1023$, but in many other cases as an integer in the range $-512 \leq c \leq 511$ (e.g. when the number c itself is used as the operand).
cell [c]	signifies the contents of cell c, excl. pos. 40-41, regarded as a machine number, i.e. a *fixed-point number* in the range $-1 \leq$ cell[c]<1.
float cell[c]	signifies the contents of cell c regarded as a *floating-point number*.
address constant c and incr c	signify the contents of pos. 0-9 and pos. 10-19, respectively, of cell c regarded as integers.
oper c, LHoper c, and RHoper c	signify the contents of that part of cell c which contains the operation in respectively a full-word, left-hand half-word, and a right-hand half-word.
pos i,c	signifies the contents of pos. i of cell c regarded as a *Boolean variable*[1].

1) The digits 0 and 1 are interpreted here, and in the following, as *false* and *true*, respectively.

R	signifies the contents of the accumulator (pos. 00-39) regarded as a fixed-point number. In certain circumstances, it refers only to pos. 0-39, but in these cases it will be indicated. In the accompanying text R is sometimes used for the accumulator itself.
RF	signifies the contents of the floating-point register, i.e. pos. 00-39 of the accumulator together with pos. 0-9 of the M-register regarded as a floating-point number.
RM	signifies the contents of the long accumulator consisting of the accumulator and pos. 1-39 of the M-register, regarded as a machine number.
Raddr and Rincr	signify the contents of respectively the address part (pos. 0-9) and the increment part (pos. 10-19) of the accumulator.
R00 and Rpos[i]	signify the contents of a particular position in the accumulator regarded as a Boolean variable.
M, Maddr, Mincr, and Mpos[i]	are the corresponding designations for the contents of certain parts of the M-register. Similar designations are also used, when necessary, for the contents of the H-register and the F-register.
sl	signifies the contents of the subroutine register.
rl	signifies the contents of the control counter (this register is increased by 1 during the initial determination of addresses for full-word instructions and right-hand half-word instructions).

r	signifies the address of the cell containing the current instruction.
p	signifies the contents of the index register.
indicator	signifies the contents of the indicator register excl. KA and KB.
indic [j]	signifies the contents of a particular position in the indicator regarded as a Boolean variable. For j = 0,..., 9, 10, and 11 the registers indicated are respectively, OA,..., RB, KA, and KB.

Besides the final address c, the following address designations are used:

D-address or Daddr means the address of that cell which contains the last link of a chain of indirect addresses, i.e. contains the (direct) address which is used in determining the final address.

The modified address is the value of the address before the increment is added. For non-adjustable operations the final and modified addresses are (by definition) the same.

Non-adjustable means that the increment is not used in determining the final address for the basic operation in question. However, most operations are adjustable, which means that the increment is used in determining the address, as described in chapter 4, and this is not mentioned explicitly in the operation list.

a '+' in the column marked 'Registration M' means that markerbits are registered, i.e. the contents of pos. 40-41 of cell c (the operand cell) are transferred to $R_{40,41}$. If the column is empty, $R_{40,41}$ remains unchanged.

a '+' in the column marked 'Registration O' means that overflow is registered, i.e. the overflow register O is set to 1 in case of overflow in H, otherwise to 0. If this column is empty, the O-register remains unchanged.

a '+' in the column marked Marking means that if the instruction is supplemented with an M-indicator operation, the marker-bits (the contents of pos. 40-41) of cell c can be changed. If the column is empty the cell-markings cannot be changed (but an absolute marker-operation will clear $R_{40,41}$, anyway, see chapter 6, below).

a '+' in the column marked 'Registration S' means that if the instruction is supplemented with an IT indicator operation, the sign of the result in the H-register will be stored in the appropriate indicator register. If the column is empty the IT indicator operation will still register the 'sign' of the H-register's contents which, however, will not be the result, but the final address or some other peculiar value.

The different modifications of the basic operations (F, S, D, and V) are referred to, in the operation list, in the following manner:

1. F-modifications are referred to only where the modification is relevant. As long as the F-modification is not mentioned, it is unimportant whether the modification is present or not.

2. S- and V-modifications are not mentioned as they always affect the operations in exactly the way described in chapters 3 and 4. The X-modification is only mentioned

in the few cases where the effect is not quite the same as the usual interchange of R and M.

3. The D-modification is referred to in all the operations where the address is relevant.

The means by which addresses are determined before the execution of each instruction can be written as the following ALGOL-like algorithms (without, however, all the proper declarations).

```
STOP:; comment GIER stops here both after a ZQ (halt) in-
         struction and after the NORMAL STOP button has been
         pressed. Incidentally, this is also the stage where
         activation of the HP button will interrupt the pro-
         gram:

NEW INSTR: r2:= r1;
           s2:= s1;
         if halfword then begin
            if ¬ RH halfword then begin FLHoper:= LHoper [r2] ;
                                        Haddr:=address co [r2] ;
                                        relative:=pos [ 28,r2 ];
                                        subindex:=pos [ 29,r2 ];
                                        indirect:=pos [ 27,r2 ];
                                        RH halfword:=true
                                  end LH halfword

                             else begin FLHoper:=RHoper [r2] ;
                                        Haddr:=incr [r2];
                                        relative:=pos [ 38,r2] ;
                                        subindex:=pos [ 39,r2] ;
                                        indirect:=pos [ 37,r2] ;
                                        RH halfword:=false;
                                        r1:=r1+1
                                  end RH halfword;
                          Fincr:=0
                          end halfword

              else begin    Foper:=oper[ r2] ;
                            Haddr:=address co [ r2] ;
                            Fincr:=incr [r2];
                            relative:=pos [ 28,r2];
                            subindex:=pos [ 29.r2];
                            indirect:=pos [ 27,r2];
                            r1:=r1+1;
                            RH halfword:=false;
                            if conditional indicator operation
                               not satisfied then
                            go to NEW INSTR
                   end fullword;
```

DETERMINE ADDRESS:

```
    if indirect ∨ non adjustable operation then go to MODIFY;
    address co[r2]:=Haddr:=Haddr + Fincr;
```

MODIFY:

```
    if ¬ relative ∧ subindex then

                      begin Haddr:=Haddr + s2;
                            s2:=incr[s2]
                      end subroutine indexed addr;

    if relative ∧ ¬ subindex then Haddr:=Haddr + r2
    if relative ∧   subindex then Haddr:=Haddr + p
    if indirect then begin  r2:=Haddr;
                            Haddr:=address co[r2] ;
                            relative:= pos[28,r2] ;
                            subindex:= pos[29,r2] ;
                            indirect:= pos[27,r2] ;
                            go to DETERMINE ADDRESS
                     end indirect;
    if ¬ Dmodification then r2:=Haddr;
```

After this, performance of the operation itself begins. For instructions without D-modification, the final address is c = r2 = Haddr; for instructions with D-modification the final address is c = Haddr, and the D-address is Daddr = r2.

For arithmetic operations in the D-mode, c is thus calculated using the 11 bits H_{00-9} (with the possibility of overflow) all of which take part in the operation.

If the increment is used during performance of a basic operation, its value is taken from the increment part of the F-register, Fincr.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
AR	cell[c] is added to R and the result is placed in R: R:=R+cell[c];	+	+	+		
AR D	$c \cdot 2^{-9}$ is added to R and the result is placed in R: R:=R+cx2↑(-9);		+	+		c is regarded as an integer in the range $-512 \leq c \leq 511$.

AR F	float cell[c] is added to RF and the result is placed in RF, while the increment part of M is set equal to the number of shifts that were necessary to normalize the result.
	RF:=RF+float cell[c];
	Mincr:=number of shifts;
		+		+		The D-modification must <u>not</u> be used. Overflow causes a jump to cell 0 with the address of next instruction in Raddr. (If the current instruction is a left half word instruction the jump is to the RH half of cell 0; if the actual instruction is a RH half word instruction the jump is to the LH half of cell 0; in both cases Raddr = rl).

AN	The absolute value of cell[c] is added to R and the result is placed in R: R:=R+abs(cell[c]);	+	+	+		The operand -1 is treated correctly, i.e. the absolute value of -1 is +1 (with overflow).
AN D	The absolute value of $c \cdot 2^{-9}$ is added to R and the result is placed in R: R:=R+abs(cx2↑(-9));		+	+		c is regarded as an integer in the range $-512 \leq c \leq 511$.

AN F	The absolute value of float cell[c] is added to RF and the result is placed in RF, while the increment part of M is set equal to the number of shifts that were necessary to normalize the result.
	RF:=RF+abs(float cell[c]);
	Mincr:=number of shifts;
		+		+		The D-modification must <u>not</u> be used.
						Overflow has the same effect as with ARF.

AC	R is added to cell[c] and the result is placed in cell c: cell[c]:=H:=cell[c]+R;	+	+	+	+	The accumulator remains unaltered. We note that ACF has the same effect as AC, i.e. the floating-point numbers are <u>not</u> added correctly.
AC D	Raddr is added to the increment, if any, of the instruction, and this sum is added to the address constant in Daddr: address co[Daddr]:=Haddr:=c+Raddr;		+	+		

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
SR	cell[c] is subtracted from R and the result is placed in R: R:=R-cell[c];	+	+	+		The operand -1 is treated correctly, i.e. subtraction of -1 is treated as addition of +1.
SR D	$c.2^{-9}$ is subtracted from R and the result is placed in R: R:=R-cx2↑(-9);		+	+		c is regarded as an integer in the range -512 ≤ c ≤ 511.

SR F	float cell[c] is subtracted from RF and the result is placed in RF, while the increment part of M is set equal to the no. of shifts that were necessary to normalize the result.
	RF:=RF-float cell[c];
	Mincr:=number of shifts;
		+		+		The D-modification must not be used.
						Overflow causes a jump to cell 0 as with ARF.

SN	The absolute value of cell[c] is subtracted from R and the result is placed in R: R:=R-abs(cell[c]);	+	+	+		The operand -1 is treated correctly, i.e. the absolute value of -1 is +1 (with overflow).
SN D	The absolute value of $c \cdot 2^{-9}$ is subtracted from R and the result is placed in R: R:=R-abs(cx2↑(-9));		+	+		c is regarded as an integer in the range -512 ≤ c ≤ 511.

SN F	The absolute value of float cell[c] is subtracted from RF and the result is placed in RF, while the increment part of M is set equal to the number of shifts that were necessary to normalize the result.
	RF:=RF-abs(float cell[c]);
	Mincr:=number of shifts;
		+		+		The D-modification must not be used.
						Overflow causes a jump to cell 0 as with ARF.

SC	R is subtracted from cell[c[and the result is placed in cell c: cell[c]:=H:=cell[c]-R;	+	+	+	+	The accumulator remains unaltered. We note that SCF has the same effect as SC, i.e. the floating-point numbers are not subtracted correctly.
SC D	Raddr is subtracted from the address constant in Daddr. The increment, if any, of the instruction is added to the result: address co[Daddr]:=Haddr:=c-Raddr;		+	+		

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
MK	M is multiplied by cell[c] and the product is added to R. The result is placed in R: R:=R+Mxcell[c]+2↑(-40);	+	+	+		The multiplication is said to be accumulative. The result is rounded off. The contents of M remain unaltered.
MK D	M is multiplied by $c\cdot 2^{-9}$ and the product is added to R. The result is placed in R: R:=R+Mxcx2↑(-9)+2↑(-40);		+	+		As with MK. c will be regarded as an integer in the range $-512 \leq c \leq 511$.
MK F	RF is multiplied by float cell[c] and the product is placed in RF: RF:=RFxfloat cell[c];	+		+		The D-modification must not be used. The multiplication is not accumulative. Overflow causes a jump to cell 0 as with ARF.

ML and ML F	M is multiplied by cell[c] and the product is added to $R\cdot 2^{-39}$. The result is placed in RM.
	RM:=Rx2↑(-39)+Mxcell[c];
	Mpos[0]:=0;
		+	+	+		Multiplication is performed without rounding off. The result has 78 binary digits.

ML D	M is multiplied by $c\cdot 2^{-9}$ and the product is added to $R\cdot 2^{-39}$. The result is placed in RM:
	RM:=Rx2↑(-39)+Mxcx2↑(-9);
	Mpos[0]:=0;
			+	+		The result has 78 binary digits.

MT	R is multiplied by +1 or by -1 whether pos[0,c] is 0 or 1. The result is placed in R: R:=if pos[0,c]=1 then -R else R;	+	+	+		If the number in c is a fixed-point number, R is multiplied by the sign of the number. If the number in c is a floating-point number, R is multiplied by the sign of the number's exponent.
MT D	$c\cdot 2^{-9}$ is regarded as a fixed-point number and R is multiplied by the sign of c: R:= if H00=1 then -R else R;		+	+		

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

DK	R is divided by cell[c]; the result is placed in R and the remainder - multiplied by 2^{39} - is placed in M. If abs(R)>abs(cell[c]) overflow is registered:
	if abs(R)<2xabs(cell[c]) then
	begin
	  q:=entier(R/cell[c]x2↑39)x
	    2↑(-39);
	  M:=(R-qxcell[c])x2↑39;
	  R:=q;
	  if cell[c]<0 ∧ R/cell[c]x2↑39=
	    entier(R/cell[c]x2↑39)
	  then begin
	    R:=q-2↑(-39);
	    M:=cell[c] end
	end
	else overflow:= true;
		+	+	+		Should only be used if abs(R)<2xabs(cell[c]). If abs(R)>abs(cell[c]) half the quotient in R can be obtained by means of a right shift. The remainder always has the sign of the divisor. After overflow the contents of R and M are useless. 0/0 gives the result $R_{00}=0$, $R_{1-39}=1$, and M = 0.
						See section 2.5.3

DK D	R is divided by $c\cdot 2^{-9}$; the result is placed in R and the remainder - multiplied by 2^{39} - is placed in M:
	if abs(R)<2xabs(cx2↑(-9)) then
	begin
	  q:=entier(R/cx2↑48)x2↑(-39);
	  M:=(R-qxcx2↑(-9))x2↑39;
	  R:=q;
	  if c<0∧R/cx2↑48=entier(R/cx2↑48)
	  then begin
	    R:=q-2↑(-39);
	    M:=cx2↑(-9) end
	end
	else overflow:= true;
			+	+		Should only be used if abs(R)<2xabs($c\cdot 2^{-9}$).
						c is regarded as an integer in the range $-512 \leq c \leq 511$.
						Besides, see note to DK.

DK F	RF is divided by float cell[c]; the result is placed in RF; the remainder is lost: RF:=RF/float cell[c];	+		+		The D-modification must not be used. Overflow causes a jump to cell 0 as with ARF.
DL	RM is divided by cell[c]; the result is placed in R and the remainder - multiplied by 2^{39}- is placed in M. If abs(RM)>abs(cell[c]) overflow is registered: The ALGOL description as for DK with RM instead of R except in the 2 statements R:=q and R:=q-2↑(-39).	+	+	+		Should only be used if abs(RM)<2xabs(cell[c]). Besides, see remark to DK.
DL D	RM is divided by $c\cdot 2^{-9}$, apart from this as DK D with RM instead of R.		+	+		Should only be used if abs(RM)<2xabs($c\cdot 2^{-9}$). c is regarded as an integer in the range $-512 \leq c \leq 511$. Besides, see remark to DK.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

NK	R is normalized, i.e. is shifted until it contains a number, N, satisfying: $-1 \leq N < -0.5$ or $N = 0$ or $0.5 \leq N < 1$. The normalization exponent (number of shifts with reverse sign) is placed as address constant in cell c:
	
	exponent:=0;
	if R ≠ 0 then
	begin if ROO ≠ Rpos[0] then
	  begin for i:=39 step -1 until 1
	    do Rpos[i]:=Rpos[i-1];
	  Rpos[0]:=ROO;
	  exponent:=1 end
	 else
	 E: if Rpos[0]=Rpos[1] then
	  begin
	    R:=2xR;
	    exponent:=exponent -1;
	    go to E end
	end;
	address co[c]:=exponent;
				+	+	The normalization exponent is only positive (+1) if there is overflow in R before execution of the operation. The right shift is performed without rounding off.

NK D	As NK with the change that the normalization exponent is placed in the cell, the address of which is the D-address.			+		

NK F	At first R is normalized, then the normalized number is shifted 10 places to the right. The normalization exponent minus 1 is added to c and the sum is placed in Maddr. (see remark).
	ALGOL-description: As for NK except that the last statement is replaced by:
	for i:= 39 step -1 until 10 do
	  Rpos[i]:=Rpos[i-10];
	for i:= 1 step 1 until 9 do
	  Rpos[i]:=Rpos[0]
	Maddr:=c+exponent -1;
				+		The D-modification must not be used.
						The instruction NK F 0 will have the effect that a fixed-point number in R is transformed to a floating-point number in RF.
						No rounding off. If R = 0, NKF c will have the effect that Maddr:=c.

NL	RM is normalized; M_0 is not included in the shifts: Mpos[0]:=0. Otherwise as for NK.			+	+	
NL D	As NL, however, the normalization exponent is placed in the cell, the address of which is the D-address.			+		

Specifi-cation	Effect	Registration			Marking	Special Remarks
		M	O	S		

TK	R is shifted c places to the left if $c > 0$ and -c places to the right if $c < 0$:
	`if c ≥ 0 then`
	`for k:=1 step 1 until c do`
	`begin R00:=Rpos[0];`
	`  for i:=0 step 1 until 38 do`
	`    Rpos[i]:=Rpos[i+1];`
	`  Rpos[39]:=0`
	`  end`
	`else`
	`begin q:=Rpos[40+c];`
	`  for k:=1 step 1 until -c do`
	`  begin`
	`    for i:=39 step -1 until 1 do`
	`      Rpos[i]:=Rpos[i-1];`
	`    Rpos[0]:=R00`
	`  end`
	`  R:=R+q×2↑(-39) end;`
			+	+		If $R \cdot 2^c$ is in the range $-2 \leq x < 2$, R will contain this number after the operation.
						A right shift is completed by rounding off.
						After the shift operation itself overflow is registered.

TK D	R is shifted as many places to the left (or to the right) as indicated by Daddr. Otherwise as TK.		+	+		Normally of no interest.
TK F	R is shifted (c+Maddr+1) places to the left if $(c+Maddr+1) \geq 0$, and -(c+Maddr+1) places to the right if $(c+Maddr+1) < 0$.					The D-modification must not be used. The instruction TK F 10 will have the effect that a floating-point number in RF will be transformed to a fixed-point number in R (if the number is between -2 and 2). After the shift operation itself overflow is registered. Rounding off may take place.
TL	RM is shifted; M_0 is, however, not included: Mpos[0]:=0. Otherwise as TK, but without rounding off at right shift.		+	+		
TL D	AS TK D, but with RM instead of R.		+	+		Normally of no interest.

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
CK	R is shifted cyclically c places to the left if c > 0 and -c places to the right if c < 0: (see below)		+			Pos. 00 in the R-register is not included in the shift, but is cleared.

```
R00:=0:
if c ≥ 0 then
for k:=1 step 1 until c do
  begin a:=Rpos[0];
  for i:=0 step 1 until 38 do
    Rpos[i]:=Rpos[i+1];
  Rpos[39]:=a end
else
for k:=1 step 1 until -c do
  begin a:=Rpos[39];
  for i:=39 step -1 until 1 do
    Rpos[i]:=Rpos[i-1];
  Rpos[0]:=a
end
```

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
CK D	R is shifted cyclically as many places to the left(or to the right) as indicated by Daddr. Otherwise as CK.		+			Normally of no interest.
CL	RM is shifted cyclically c places to the left if c > 0 and -c places to the right if c < 0. M_0 is not included: Mpos[0]:=0. Otherwise as CK.		+			Pos. 00 in the R-register and pos. 0 in the M-register are not included in the shift, but are cleared.
CL D	As CK D, but with RM instead of R.		+			Normally of no interest.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
AB	cell[c] is Boolean added to R and the sum is placed in R:	+		+		The X-modification has a special effect in this and the following instruction and is described separately.
	for i:=0 step 1 until 39 do					
	Rpos[i]:=Rpos[i] $\vee$ pos[i,c];					
	R00:=R00 $\vee$ pos[0,c];					
AB D	c is Boolean added to R and the sum is placed in R:			+		
	for i:=0 step 1 until 9 do					
	Rpos[i]:=Rpos[i] $\vee$ Hpos[i];					
	R00:=R00 $\vee$ H00;					
AB X	After execution of AB, R is placed in the M-register and ¬R in the R-register:	+		+		
	for i:=0 step 1 until 39 do					
	begin Mpos[i]:=Rpos[i] $\vee$ pos[i,c];					
	Rpos[i]:= ¬Mpos[i] end;					
	R00:=¬(R00 $\vee$ pos[0,c]);					
AB DX	After execution of AB D, R is placed in the M-register and ¬R in the R-register:			+		
	for i:=0 step 1 until 9 do					
	Rpos[i]:=Rpos[i] $\vee$ Hpos[i];					
	for i:=0 step 1 until 39 do					
	begin Mpos[i]:=Rpos[i];					
	Rpos[i]:=- Rpos[i] end;					
	R00:=¬(R00 $\vee$ H00);					

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

MB	cell[c] is Boolean multiplied by R and the product is placed in R:
	
	for i:= 0 step 1 until 39 do
	  Rpos[i]:=Rpos[i] ∧ pos[i,c];
	R00:=R00 ∧ pos[0,c];
		+		+		The X-modification has a special effect in this and the following instruction and is described separately.

MB D	c is Boolean multiplied by R and the product is placed in R:
	
	for i:=0 step 1 until 9 do
	  Rpos[i]:=Rpos[i] ∧ Hpos[i];
	R00:=R00 ∧ H00;
	for i:=10 step 1 until 39 do
	  Rpos[i]:=0;					

MB X	After execution of MB, R is placed in the M-register and the nonequivalence of original R and c is placed in the R-register:
	
	for i:=0 step 1 until 39 do
	begin Mpos[i]:=Rpos[i] ∧ pos[i,c];
	  Rpos[i]:=¬(Rpos[i] ≡ pos[i,c])
	end;
	R00:=¬(R00 ≡ pos[0,c]);
		+		+		

MB XD	After execution of MB D, R is placed in the M-register, and the nonequivalence of original R and c is placed in the R-register:
	
	for i:=0 step 1 until 9 do
	begin Mpos[i]:=Rpos[i] ∧ Hpos[i];
	  Rpos[i]:=¬(Rpos[i] ≡ Hpos[i])
	end;
	R00:=¬(R00 ≡ H00);
	for i:=10 step 1 until 39 do
	  Mpos[i]:=0;
				+		Pos. 10-39 of R remain unaltered after the operation.

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
PM	The contents of cell[c] is placed in M, i.e. M:=cell[c];	+		+		
PM D	$c \cdot 2^{-9}$ is placed in M_{0-9} while the remainder of M is cleared, i.e. M:=cx2↑(-9);			+		
PI	The modified address is placed in the indicator (except KA and KB) with the increment of the instruction as mask; those positions of the indicator that are specified by ones in the increment of the instruction (binary) are <u>not</u> changed: <u>for</u> i:=0 <u>step</u> 1 <u>until</u> 9 <u>do</u> indic[i]:=(indic[i] $\wedge$ Fincr[i]) $\vee$ (Hpos[i] $\wedge \neg$ Fincr[i]);					The instruction is <u>non-adjustable</u>. If PI is a half-word instruction the increment is regarded as zero, i.e. the whole modified address is transferred to the indicator.
PI D	As above, the D-address replacing the modified address.					The instruction is <u>non-adjustable</u>. Normally of no interest.
PP	c is placed in the p-register: p:=c;					
PP D	The D-address is placed in the p-register: p:=Daddr;					Normally of no interest.
PS	c is placed in the subroutine register: sl:=c;					
PS D	The D-address is placed in the subroutine register: sl:=Daddr;					Normally of no interest

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
PA	The increment of the instruction is placed in pos. 0-9 of cell c, while the remainder of the cell remains unaltered: address co[c]:=Fincr;					The instruction is <u>non-adjustable</u>. If the <u>PA-instruction</u> is a half-word instruction, it has the same effect as a full-word instruction with the increment 0.
PA D	The increment of the instruction is placed in pos. 0-9 of the cell specified by the D-address: address co[Daddr]:=Fincr;					The instruction is <u>non-adjustable</u>.
PT	The increment of the instruction is placed in pos. 10-19 of cell c, while the remainder of the cell remains unaltered: incr[c]:=Fincr;					The instruction is <u>non-adjustable</u>. If the <u>PT-instruction</u> is a half-word instruction, it has the same effect as a full-word instruction with the increment 0.
PT D	The increment of the instruction is placed in pos. 10-19 of the cell specified by the D-address: incr[Daddr]:=Fincr;					The instruction is <u>non-adjustable</u>.

Specifi-cation	Effect	Registratio			Mark-ing	Special Remarks
		M	O	S		
GR	R is stored in cell c: cell[c]:=R;				+	R00 irrelevant. In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).

GR D	The address constant and the increment of R are stored in the corresponding positions in the cell specified by the D-address, while the remainder of the cell remains unaltered:
	address co[Daddr]:=Raddr;
	incr[Daddr]:=Rincr;					

GR F	RF is stored in cell c: float cell[c]:=RF;				+	R00-9 is irrelevant. In absolute marking R_{40} and R_{41} are set to one(otherwise unaltered). The D-modification must <u>not</u> be used.
GM	M is stored in cell c: cell[c]:=M;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).

GM D	The address constant and the increment of M are stored in the corresponding positions of the cell specified by the D-address, while the remainder of the cell remains unaltered:
	address co[Daddr]:=Maddr;
	incr[Daddr]:=Mincr;					

Specifi-cation	Effect	Registration			Mark-ing	Special Remarks
		M	O	S		
GA	The address constant of R is stored in pos. 0-9 of cell c, while the remainder of cell c remains unaltered: address co[c]:=Raddr;				+	R00 is irrelevant. In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GA D	The address constant of R is stored in pos. 0-9 of the cell specified by the D-address, while the remainder of this cell remains unaltered: address co[Daddr]:=Raddr;				+	R00 is irrelevant. In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GT	The increment of R is stored in pos. 10-19 of cell c, while the remainder of the cell remains unaltered: incr[c]:=Rincr;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GT D	The increment of R is stored in pos. 10-19 of the cell specified by the D-address, while the remainder of this cell remains unaltered: incr[Daddr]:=Rincr;				+	In absolute marking R_{40} and R_{41} are set to one(otherwise unaltered).

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
GP	The contents of the p-register are stored in pos.0-9 of cell c, while the remainder of the cell remains unaltered: address co[c]:=p;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GP D	The contents of the p-register are stored in pos. 0-9 of the cell specified by the D-address, while the remainder of the cell remains unaltered: address co[Daddr]:=p;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GS	The contents of the subroutine register are stored in pos.0-9 of cell c, while the remainder of the cell remains unaltered: address co[c]:=sl;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GS D	The contents of the subroutine register are stored in pos. 0-9 of the cell specified by the D-address, while the remainder of the cell remains unaltered: address co[Daddr]:=sl;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unchanged).
GI	The contents of the indicator (except KA and KB) are stored in pos. 0-9 of cell c, while the remainder of the cell remains unaltered: address co[c]:=indicator;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).
GI D	The contents of the indicator (except KA and KB) are stored in pos. 0-9 of the cell specified by the D-address, while the remainder of this cell remains unaltered: address co[Daddr]:=indicator;				+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).

GK	The contents of the by-register and the track register are stored in the address and increment positions of cell c, while the remainder of this cell remains unaltered:
	address co[c]:=by;
	incr[c]:=tk;
					+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).

GK D	As above, with storage in the cell specified by the D-address:
	address co[Daddr]:=by;
	incr[Daddr]:=tk;
					+	In absolute marking R_{40} and R_{41} are set to one (otherwise unaltered).

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
IS	The IS-instruction is only effective the first time s is included (if at all) in the address calculation for the instruction immediately following. The effect will thus be that the final address of the IS-instruction is used as the s-value in the following instruction regardless of the present contents of the subroutine register (and this remains unaltered after the operation): s2:=c; <u>comment</u>: This statement replaces the statement s2:=s1 in the address calculation for the instruction immediately following;					The instruction immediately following should (out of regard to the tracer-programs) not be a jump instruction or a V-modification. I-indicator-operation has no effect. X- and V-modification in an IS-instruction have no effect.
IS D	The D-address of the instruction is used as the s-value in the instruction immediately following. Otherwise as the IS-instruction.					See above.
NS	The final address with reverse sign is used as s-value in the instruction immediately following. Otherwise as the IS-instruction. s2:=-c; <u>comment</u>: This statement replaces the statement s2:=s1 in the address calculation for the instruction immediately following;					See above.
NS D	The D-address with reverse sign is used as s-value in the instruction immediately following. Otherwise as the NS-instruction.					See above.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
IT	The final address of the instruction is used as increment in the instruction immediately following regardless of the increment, if any, of this instruction (the increment remains unaltered after the operation): Fincr:=c; comment: This statement replaces the statements Fincr:=0 and Fincr:=increment[r2] in the address calculation for the instruction immediately following;					The instruction immediately following should (out of regard to the tracer-programs) not be a jump instruction or a V-modification. X- and V-modification in the IT-instruction have no effect. I-indicator-operation has no effect. If the IT-instruction is a LH half-word instruction the corresponding RH half-word instruction should be be non-adjustable or have an indirect address; in other cases the address constant of the IT-instruction is changed.
IT D	The D-address of the instruction is used as increment in the instruction immediately following. Otherwise as the IT-instruction.					See above.
NT	The final address with reverse sign is used as increment in the instruction immediately following. Otherwise as the IT-instruction: Fincr:=-c; comment: This statement replaces the statements Fincr:=0 and Fincr:=increment[r2] in the address calculation for the instruction immediately following;					See above.
NT D	The D-address with reverse sign is used as increment in the instruction immediately following. Otherwise as the NT-instruction.					See above.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

BT	The instruction (or instructions) up to and including the following RH half-cell is only executed if c > t, the final address c and the increment t of the instruction both being regarded as numbers in the range -512 ≦ x ≦ 511. Otherwise this instruction (or these instructions) is skipped:
	_if_ c ≤ Fincr _then_
	begin RH halfword:= _false_;
	rl:=rl+1 _end_;
						For a half-word instruction the increment t equals 0. If the BT-instruction is V-modified the described effect applies to the instruction (instructions) in cell r+2, the cell immediately after the BT-instruction _always_ being skipped.

BT D	The instruction has a conditionalizing effect exactly as BT, with the exception that the D-address is compared with the increment:
	_if_ Daddr ≤ Fincr _then_
	begin RH halfword:= _false_;
	rl:=rl+1 _end_;
						V-modification: As with BT.

BS	The instruction (or instructions) up to and including the following RH half-cell is only executed if m > t, the modified address m and the increment t of the instruction both being regarded as numbers in the range -512 ≦ x ≦ 511. Otherwise this instruction (or these instructions) is skipped:
	_if_ c ≤ Fincr _then_
	begin RH halfword:= _false_;
	rl:=rl+1 _end_;
						The instruction is _non-adjustable_.
						For a half-word instruction the increment t equals 0.
						V-modification. As with BT.

BS D	The instruction has a conditionalizing effect exactly as BS, with the exception that the D-address is compared with the increment:
	_if_ Daddr ≤ Fincr _then_
	begin RH halfword:= _false_;
	rl:=rl+1 _end_;
						The instruction is _non-adjustable_.
						V-modification: As with BT.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

CA	The instruction (or the instructions) up to and including the following RH half-cell is only executed if the final address is equal to the address constant in R. Otherwise the instruction (or the instructions) is skipped:
	if c ≠ Raddr then
	begin RH halfword:= false;
	rl:=rl+1 end;
						c is calculated with 11 bits in pos. 00-9 of the H-register, and then the contents of pos. 0-9 are compared with the corresponding bits of R.
						V-modification: As with BT.

CA D	Same effect as CA, with the exception that the D-address is compared with the address constant in R:
	if Daddr ≠ Raddr then
	begin RH halfword:= false;
	rl:=rl+1 end;
						V-modification: As with BT.

NC	The instruction (or the instructions) up to and including the following RH half-cell is only executed if the final address is different from the address constant in R. Otherwise this instruction is skipped:
	if c = Raddr then
	begin RH halfword:= false;
	rl:=rl+1 end;
						c is calculated as in the CA-instruction.
						V-modification: As with BT.

NC D	Same effect as NC with the exception that the D-address is compared with the address constant in R:
	if Daddr =Raddr then
	begin RH halfword:= false;
	rl:=rl+1 end;
						V-modification: As with BT.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

CM
|
O
	The instruction (or the instructions) up to and including the following RH half-cell is only executed if the contents of cell c are not identical with R in those positions where the corresponding positions in M ~~[illegible]~~ contain ones:
	coinc:= true;
	for i:=0 step 1 until 39 do
	  coinc:=coinc ∧
	    (pos[i,c] ∧ Mpos[i]
	    ≡ Rpos[i] ∧ Mpos[i]);
	if coinc then
	begin RH halfword:= false;
	      rl:=rl+1 end;
		+				R00 is not included in the coincidence examination.
						V-modification: As with BT.

CM D	Same effect as CM with the exception that the final address c itself followed by 30 zeroes is compared with R (with M as mask).					R00 is not included in the coincidence examination. V-modification: As with BT.

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
HV	Next instruction is taken from cell c; if cell c contains two half-word instructions, the left one is executed first: RH halfword:= <u>false</u>; rl:=c;					If the instruction is a V-modification, next instruction is taken from cell c+1.
HV D	Next instruction is taken from the cell specified by the D-address. Otherwise as the HV-instruction.					
HH	Next instruction is taken from cell c; if cell c contains two half-word instructions only the right one is executed: RH halfword:= <u>true</u>; rl:=c;					If the instruction is a V-modification, next instruction is taken from cell c+1.
HH D	Next instruction is taken from the cell specified by the D-address. Otherwise as the HH-instruction.					

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		

HS	Next instruction is taken from the cell specified by the modified address c; if this cell contains two half-word instructions the left one is executed first. The address of the HS-instruction is stored in the subroutine register, and for a full-word HS-instruction the previous contents of the subroutine register are stored in the increment part of the full-word instruction:
	if ¬ halfword then
	  increment[r]:=sl;
	sl:=r;
	RH halfword:= false;
	rl:=c;
						The instruction is non-adjustable. If the instruction is a V modification the next instruction is taken from cell c+1. If several subroutines are nested the HS-instructions ought to be full-word instructions except in the outer block.

HS D	The same effect as for HS, except that the next instruction is taken from the cell specified by the D-address.					The instruction is non-adjustable.

HR	Next instruction is taken from cell c. If this cell contains two half-word instructions the left one is executed first. Also the increment of cell s is transferred to the subroutine register:
	sl:=increment[sl];
	RH halfword:= false;
	rl:=c;
						If the instruction is a V-modification next instruction is taken from cell c+1. If several subroutines are nested it is necessary to use one HR-jump corresponding to each HS-jump if the subroutine mechanism is to work correctly.

HR D	The same effect as for HR, except that the next instruction is taken from the cell specified by the D-address.					
HK	If no drum transfer takes place the instruction has the same effect as HS. Otherwise the instruction is blank(however, a S-modification, if any, is always executed).					The instruction is non-adjustable.
HK D	If no drum transfer takes place the instruction has the same effect as HS D. Otherwise the instruction is blank(however, a S-modification, if any, is always executed).					The instruction is non-adjustable.

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
VK	The drum track specified by the final address is connected to the ferrite core store (but no drum transfer takes place): tk:=c;					The VK-instruction is not executed until an eventual running drum transport is completed.
VK D	The drum track specified by the D-address is connected to the ferrite core store (but no drum transfer takes place).					As above.
LK	The drum track connected in advance is transferred to the ferrite core store in 40 consecutive cells so that the first word of the track is stored in cell c. R and M are not affected.					The LK-instruction is not executed until an eventual running drum transport is completed.
LK D	Same effect as LK with the exception that the D-address replaces the final address c.					As above.
SK	The contents of 40 consecutive cells from the ferrite core store are stored on the drum track connected in advance; the first of these cells is cell c, which is stored as the first word on the drum track. R and M are not affected.					The SK-instruction is not executed until an eventual running drum transport is completed. Certain tracks are locked for writing (see volume II). A SK-instruction referring to one of these tracks is neglected.
SK D	Same effect as SK with the exception that the D-address replaces the final address c.					As above.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
VY	The modified address is placed in the by-register with the increment of the instruction as mask; the positions specified by ones in the increment of the instruction (binary) are not changed: for i:=0 step 1 until 9 do by[i]:=(by[i] $\wedge$ Fincr[i]) $\vee$ (Hpos[i] $\wedge$ $\neg$ Fincr[i];					The instruction is non-adjustable. If the VY-instruction is a half-word instruction, the increment is regarded as zero, i.e. the whole modified address is transferred to the by-register.
LY	A symbol is read from a peripheral unit to the address positions of R and cell c in such a way that the bit pattern of the symbol (except of the parity-bit) is placed in pos. 3-9. Pos. 0-2 (and R00) are cleared and pos. 10-39 remain unaltered. Pos. 7-9 of the by-register determine from which peripheral unit to be read: by$_{7-9}$ = 000: tape reader by$_{7-9}$ = 001: typewriter by$_{7-9}$ = 010: punched card reader In tape input blank tape is skipped and 'all holes' are read as the value 127 (without error in the parity check)					When the code is read in from the typewriter, the symbol is transferred directly from the typewriter to the cell and to R. When the code is read from punched tape, LY has the effect that a symbol is transferred from the buffer register bl to the cell and to R, and the next symbol is then read from the punched tape to bl; if there is error in the parity-check GIER stops, and it is then possible to make corrections in the R-register and in the cell. The parity-bit is placed in pos. 2 of the bl-register.
SY	The last 7 digits of the final address c (pos. 3-9) are written in accordance with the flexowriter code (disregarding the parity-bit). When the code is punched on tape the parity-bit is put in the right place. The last 7 digits of c are shifted 10 places to the right and are added to the contents of cell 1023 in pos. 0-19. Pos. 4-5 of the by-register determine the peripheral units, from which output is to be made: by[5] = 1: typewriter by[4] = 1: tape punch					The sum of the bit pattern of all symbols is accumulated in cell 1023 pos. 0-19 and the contents of this may thus be used as an output check sum. The output units can be connected independent of each other and of the input units

Specification	Effect	Registration M	Registration O	Registration S	Marking	Special Remarks
QQ	No basic operation, but all modifications wanted (address calculation, X-modification etc.) are executed.					The value of QQ being 0, the instruction QQ 0 causes clearing of a cell (or a half-cell) during input. The address is irrelevant for execution of the instruction.
ZQ	Having performed all modifications GIER stops. However, an eventual running drum transport is terminated before the stop. By pressing the NORMAL START button GIER continues with the next instruction (or the following if the ZQ-instruction is provided with a V).					When GIER has stopped, the ZQ-instruction has been completed. The address of the ZQ-instruction is irrelevant.

XR	The contents of R and M are exchanged (R00 is not included):
	H:=M;
	M:=R;
	<u>comment</u>: excluding R00;
	R:=H;
				+		The address of the XR-instruction is irrelevant. The sign to be registered is the final sign of R, i.e. the sign of the contents of R <u>before</u> the operation.

Specification	Effect	Registration			Marking	Special Remarks
		M	O	S		
UD	One of the instructions placed in cell c is executed with the current r-value and (if the s-mechanism is used correctly) with the current s-value. The diagram below illustrates the effects of the interplay between placing of the UD-instruction and the instruction (or instructions) of cell c. If the instruction of cell c is a substitution instruction (IS, NS, IT or NT) the whole substitution chain, i.e. the substitution instruction(s) and the instruction affected by the last substitution instruction, is executed. At the same time r is increased correspondingly. If the last instruction of the substitution chain is a jump instruction, this is executed.					X- and V-modifications of the UD-instruction have no effect.
UD D	The same effect as UD, c everywhere being replaced by the D-address.					Normally of no interest. X- and V-modifications of the UD-instruction have no effect.

In cell c	The UD-instruction of cell r is a		
	full-word	LH half-word	RH half-word
Two half-words			
No jump instructions	RH half-word of cell c is performed, and then the instruction of cell r+1.	LH half-word of cell c is performed, and then RH half-word of cell r.	RH half-word of cell c is performed, and then the instruction of cell r+1.
Jump instruction in LH half-word	RH half-word of cell c is performed, and then the instruction of cell r+1	Final address of jump instruction is stored as as address constant in cell r. Then LH or RH half-word of cell r-1 is performed depending on the jump instruction being HV (HS, HR, HK) or HH.	As when no jump instructions are placed in cell c.
Jump instruction in RH half-word	Final address of jump instruction is stored as address constant in cell r. Then LH or RH half-word of cell r-1 is performed depending on the jump instruction being HV (HS, HR, HK) or HH.	As when no jump instructions are placed in cell c.	Final address of jump-instruction is stored as address constant of cell r. Then LH or RH half-word of cell r-1 is performed depending on the jump instruction being HV (HS, HR, HK) or HH.
One full-word			
No jump instruction without V	The instruction of cell c is executed, and then the instruction of cell r+1.		
No jump instruction, with V	The instruction of cell c is executed, and then the instruction of cell r+2.		
Jump instruction	Final address of the instruction of cell c is stored as address constant in cell r. Then LH or RH half-word of cell r-1 is performed depending on the jump instruction being HV (HS, HR, HK) or HH.		

6. INDICATOR INSTRUCTIONS

On the following pages can be found a list of indicator instructions and their effect. In the column 'Special remarks' we have noted certain conditions, to which special attention must be paid.

Specification	Effect	Special remarks
IOA	The register [1]OA gets the same contents as the overflow register after performance of the basic operation: OA:= <u>if</u> overflow <u>then</u> 1 <u>else</u> 0;	Overflow is <u>registered</u> by the <u>arithmetic basic operations</u> (including the shift operations) and their modifications except the F-modification, i.e. the overflow register 0 is set to one if there is overflow in the result of the H-register, otherwise it is cleared. In all other basic operations and their modifications the O-register remains unaltered.
IOB	The register OB gets the same contents as the O-register after performance of the basic operation: OB:= <u>if</u> overflow <u>then</u> 1 <u>else</u> 0;	
IOC	The registers OA and OB both get the same contents as the O-register after performance of the basic operation: OA:=OB:= <u>if</u> overflow <u>then</u> 1 <u>else</u> 0;	
IZA	The register OA is set to one if the contents of the R-register is 0 after performance of the basic operation. Otherwise OA is cleared: OA:= <u>if</u> R=0 <u>then</u> 1 <u>else</u> 0;	ROO is included here. Registration of the zero situation in the R-register can take place in connection with any operation. Registration takes place <u>before</u> performance of a V-modification, if any.
IZB	The register OB is set to one if the contents of the R-register is 0 after performance of the basic operation. Otherwise OB is cleared: OB:= <u>if</u> R=0 <u>then</u> 1 <u>else</u> 0;	
IZC	The registers OA and OB are both set to one if the contents of the R-register is 0 after performance of the basic operation. Otherwise OA and OB are cleared: OA:=OB:= <u>if</u> R=0 <u>then</u> 1 <u>else</u> 0;	

1) The specification 'register' is in this chapter used a.o. about a single position of the indicator containing 12 positions (12 flip-flops).

Specification	Effect	Special remarks
ITA	The TA-register is set to one if the contents of the H-register is negative after performance of the basic operation. Otherwise TA is cleared: TA:=H00;	In the arithmetic basic operations (and the shift operations and a few more) the sign of the H-register is equal to the sign of the sign of the result, and it is thus the sign of the result that is registered. It is necessary to register the sign in the instruction where it 'arises', since the H-register is used in each operation. The sign is read in pos.00 and is thus correct even if overflow occurs. Note besides the difference from the indicator instructions LT and NT where the sign is examined in the R-register.
ITB	The TB-register is set to one if the contents of the H-register is negative after performance of the basic operation. Otherwise TB is cleared: TB:=H00;	
ITC	The registers TA and TB are set to one if the contents of the H-register is negative after performance of the basic operation. Otherwise TA and TB are cleared: TA:=TB=H00;	
IPA	The PA-register gets the same contents as R_{40} after performance of the basic operation: PA:=Rpos[40];	Marking of the operand cell is <u>registered</u>, i.e. transferred to $R_{40,41}$ in the following basic operations: AR, AN, AC, SR, SN, SC, MK, ML, MT, DK, DL, AB, MB, PM, CM, and their S-, F-, X-, and V-modifications, but <u>not</u> D-modifications.
IPB	The PB-register gets the same contents as R_{41} after performance of the basic operation: PB:=Rpos[41];	
IPC	The registers PA and PB get the same contents as R_{40} and R_{41} after performance of the basic operation: PA:=Rpos[40]; PB:=Rpos[41];	
IQA, IQB, ..., IRC	have the corresponding effects for the Q- and R-registers.	
IK, IKA, IKB, IKC	Have all the same effect, i.e. the contents of the p-register and the indicator are exchanged after performance of the basic operation.	The contents of KA and KB are <u>not</u> included in the exchange and remain unaltered.

Specification	Effect	Special Remarks

M	The marker-bits of the operand cell are both cleared. The marker-bits of the R-register are both set to one:
	pos[40,c]:=pos[41,c]:=0;
	Rpos[40]:=Rpos[41]:=1;
		Is normally used only in connection with the basic operations AC, SC, GR, GM, GA, GT, GI, GP, GS, GK, NK, and NL, and their modifications (not GRD and GMD).
		In connection with the other basic operations the marker-bits of the operand cell will remain unaltered; however, the marker-bits of the R-register are still set to one.
		M, MA, MB, and MC are called absolute marking operations.

MA	The marker-bits of the operand cell are set to 10. The marker-bits of the R-register are both set to one:
	pos[40,c]:=1; pos[41,c]:=0;
	Rpos[40]:=Rpos[41]:=1;	

MB	The marker-bits of the operand cell are set to 01. The marker-bits of the R-register are both set to one:
	pos[40,c]:=0; pos[41,c]:=1;
	Rpos[40]:=Rpos[41]:=1;	

MC	The marker-bits of the operand cell are set to 11. The marker-bits of the R-register are both set to one:
	pos[40,c]:=pos[41,c]:=1;
	Rpos[40]:=Rpos[41]:=1;	

MOA	Pos.40 of the operand cell gets the same contents as OA, while pos.41 is cleared. The marker-bits of the R-register remain unaltered: pos[40,c]:=OA; pos[41,c]:=0;	The other registers in the indicator including KA and KB can be used in marking of a cell in exactly the same way as OA and OB. The marker-bits of the R-register always remain unaltered. Note besides the remarks under M. MOA, MOB, MOC, MTA, ..., MKC are called indicator-dependent marking operations.
MOB	Pos.41 of the operand cell gets the same contents as OB, while pos.40 is cleared. The marker-bits of the R-register remain unaltered: pos[40,c]:=0; pos[41,c]:=OB;	
MOC	Pos.40-41 of the operand cell get the same contents as OA and OB, respectively. The marker-bits of the R-register remain unaltered: pos[40,c]:=OA; pos[41,c]:=OB;	

Specification	Effect	Special Remarks
LO (NO)	The current instruction is only executed if ROO $\neq$ RO (or ROO = RO).	Independent of the contents of the indicator.
LOA (NOA)	The current instruction is only executed if the contents of OA are equal to 1 (or 0).	
LOB (NOB)	The current instruction is only executed if the contents of OB are equal to 1 (or 0).	
LOC (NOC)	The current instruction is only executed if the contents of both OA and OB are equal to 1 (or 0).	
LZ (NZ)	The current instruction is only executed if the contents of R are equal to 0 (different from 0).	ROO is included here. Independent of the contents of the indicator.
LZA, LZB, LZC	Same effect as LOA, LOB, and LOC, respectively.	
NZA, NZB, NZC	Same effect as NOA, NOB, and NOC, respectively.	
LT (NT)	The current instruction is only executed if ROO=1 (or ROO=0).	Note the difference from registration of sign where the sign is taken from HOO. Independent of the contents of the indicator.
LTA (NTA)	The current instruction is only executed if TA=1 (or TA=0).	
LTB (NTB)	The current instruction is only executed if TB=1 (or TB=0).	
LTC (NTC)	The current instruction is only executed if TA=TB=1 (or TA=TB=0).	

Specification	Effect	Special Remarks
LA (NA)	The current instruction is only executed if $R_{40}=1$ (or $R_{40}=0$).	Independent of the contents of the indicator.
LB (NB)	The current instruction is only executed if $R_{41}=1$ (or $R_{41}=0$).	
LC (NC)	The current instruction is only executed if $R_{40}=R_{41}=1$(or $R_{40}=R_{41}=0$).	
LPA (NPA)	The current instruction is only executed if PA=1 (or PA=0).	
LPB (NPB)	The current instruction is only executed if PB=1 (or PB=0).	
LPC (NPC)	The current instruction is only executed if PA=PB=1 (or PA=PB=0).	
LQA, NQA, ..., NRC	have the corresponding effects, depending on the Q- and R-bits.	
LKA (NKA)	The current instruction is only executed if KA=1 (or KA=0).	The contents of KA and KB can only be changed by means of push buttons on the control panel (see volume 2).
LKB(NKB)	The current instruction is only executed if KB=1 (or KB=0).	
LKC (NKC)	The current instruction is only executed if KA=KB=1 (or KA=KB=0).	

7. EXAMPLES AND EXERCISES

7.1 Introduction

This chapter consists of a series of examples showing the effect of each instruction, and some examples of how different types of problems are programmed, and accompanying exercises.

In section 7.2 instructions without indicator instructions are considered, and in section 7.3 the main consideration is for instructions with indicator instructions. Section 7.4 contains some examples of programs dealing with both large and small problems, and some exercises. In each section the arrangement is as follows: At first the problem to be considered is described, and then the solution of the problem is demonstrated by the statement of a program. These programs normally contain only few or no comments at all, since the purpose of the examples is to make the reader familiar with the operation list. The instruction UD is not considered in this chapter as it is rather particular.

Incidentally, the reader's attention is drawn to the fact that the problems demonstrated may frequently be solved in many other ways than those described in this manual.

7.2 Instructions without Indicator Instructions

7.2.1 Examples of Addition, Subtraction, Storage, and Use of Stop Instructions

Example 7.1

The machine numbers x, y, z, v are stored in cell no. 100, 101, 102, and 103 respectively. It is assumed that the sum $s = x + y + z + v$ is a machine number. s is required to be stored in cell no. 104, after which the computer should stop.

Program:

```
[m+0]   ARS  100,  AR  101
[m+1]   AR   102,  AR  103
[m+2]   GR   104,  ZQ    0
```

Exercise 7.2

In what way should the program in example 7.1 be changed if the numbers x, y, z, and v are floating-point numbers?

Exercise 7.3

Will the result in example 7.1 be correct if $x = 0{,}75$, $y = 0{,}75$, $z = -0{,}50$, and $v = -0{,}25$? (If so, the interim result $x + y$ is not a machine number).

Exercise 7.4

A program is required to form $|x| + |y| + |z| + |v|$ in cell 105. (The same storage as in example 7.1 with the assumption again that the result is a machine number).

Exercise 7.5

As in exercise 7.4, except that we assume that the numbers are now floating-point numbers.

Exercise 7.6

x - |y| - z + |v| is to be stored in cell 106. (Storage as before).

Example 7.7

Let us imagine that the integers a1, a2, and a3 are placed in cell no. 201, 202, and 203, respectively, with units in pos. 9 (i.e. the numbers are placed in the address positions).

The number a1 - a2 + a3 + 13 is to be stored with the units in pos. 9 of cell 200.

Program:

```
[m+0]   ARS 201, SR 202
[m+1]   AR  203
[m+2]   AR   13 D
[m+3]   GR  200, ZQ   0
```

Exercise 7.8

What condition must be fulfilled to obtain the desired result in example 7.7?

Exercise 7.9

Write a program that forms - |a1| + |a2| - |a3| - 32. (conditions as in example 7.7).

Exercise 7.10

Write a program that stores the numbers 1, 2, 3, 4, and 5 in cells nos. 901, 902, 903, 904, and 905. The numbers must be stored with the units in pos. 9.

Example 7.11

Full-word instructions with address constants a1, a2, and a3 are placed in cells nos. 201-203. None of the instructions have relative or indirect addressing, or are indexed. The number a1 - a2 + a3 + 13 is to be stored with the units in pos. 9 of cell 200.

<u>Program</u>:

```
[m+0]   ARS  (201)  D
[m+1]   SR   (202)  D
[m+2]   AR   (203)  D
[m+3]   AR     13   D
[m+4]   GR    200,  ZQ 0
```

If only one of the instructions in cell 201-203 has indirect addressing the above mentioned program will not function correctly. (Why?) The actual solution of this problem will, in fact, require application of operations not yet examined.

<u>Example 7.12</u>

The machine numbers a and b are placed in R and cell 444 respectively. The sum a + b is to be formed in cell 444.

```
[m+0] AC 444, ZQ 0
```

<u>Example 7.13</u>

The floating-point numbers a and b are placed in RF and in cell 444 respectively. Their sum is to be formed in cell 444:

```
[m+0]   ARF 444, GR 444 [Why not just ACF 444, ZQ 0?]
[m+1]   ZQ    0
```

<u>Exercise 7.14</u>

The numbers a and b are placed as in example 7.12. Form

1) a - b in cell 444
2) b - a in cell 444.

7.2.2 Examples of Multiplication and Division. Placing in the M-register.

Example 7.15

The machine numbers x, y, z, and v are placed in cells 300, 301, 302, and 303, respectively. The quantity $a = xy(z + v)$ is to be calculated and stored in cell 304.

Program 1:

```
[m+0] ARS 302, AR  303
[m+1] GR  304, PM  304
[m+2] MKS 301, GR  304
[m+3] PM  304, MKS 300
[m+4] GR  304, ZQ    0
```

By the use of the X-modification the job can be done with fewer instructions.

Program 2:

```
[m+0] ARS 302
[m+1] AR  303      X
[m+2] MKS 301      X
[m+3] MKS 300, GR 304
[m+4] ZQ    0
```

Note, however, that this program takes up just as much space as the first program. The job can also be done using a somewhat shorter program:

Program 3:

```
[m+0]  ARS 302,  AR  303
[m+1]  XR    0,  MKS 301
[m+2]  XR    0,  MKS 300
[m+3]  GR  304,  ZQ    0
```

But, in fact, program 2 is performed in shorter time than the other ones.

Our reason for demonstrating different solutions is purely that we would like to illustrate different types of instructions. We will, however, not encourage the reader to change the programs to save a couple of cells. It requires a great deal of time to program with the least possible number of instructions.

Exercise 7.16

Same problem as in example 7.15, but with floating-point numbers instead of machine numbers.

Exercise 7.17

What will the contents of cell 304, in example 7.15, be after program no. 1 is completed, if the S-modifications in [m+2] and [m+3] are excluded.

Exercise 7.18

With the same condition of storage as in example 7.15 the quantity $a = (x - y)(v - z)$ is to be formed and stored in cell no. 304.

Example 7.19

Let the machine numbers a and b be placed in M and R respectively. The instruction MK c D will thus cause $ac \times 2^{-9} + b$ to be placed in R if $-512 \leqq c \leqq 511$.

Exercise 7.20

Let a and b in example 7.19 be 0.3 and 0.2 respectively. What is the effect of the instruction MK 896 D?

Example 7.21

The integers a, b, and c are placed in cell no. 400, 401, and 402 respectively with the units in pos. 39. We assume that $0 \leqq ab+c < 2^{39}$. ab+c is to be stored in cell

403 with the units in pos. 39.

<u>Program</u>:

```
[ m+0 ]  PM 400,   ARS 402
[ m+1 ]  ML 401,   GM  403
[ m+2 ]  ZQ   0
```

<u>Exercise 7.22</u>

Why will the program in example 7.21 not work properly if the condition $0 \leqq ab+c < 2^{39}$ is not fulfilled?

<u>Example 7.23</u>

The floating-point numbers a, b, and c are placed in cell no. 400, 401, and 402, respectively. The quantity ab+c is to be stored in cell 403.

<u>Program</u>:

```
[m+0]   ARSF 400,   MKF 401
[m+1]   ARF  402,   GRF 403
[m+2]   ZQ     0
```

Note, that one F in each of the cells [m+0] and [m+1] would be sufficient.

<u>Example 7.24</u>

The machine numbers a, b, and c are placed in cell 12, 13, and 14, respectively. The quantity ac/b is to be stored in cell 11.

<u>Program 1</u>:

```
[m+0]   PM 12,   MKS 14
[m+1]   DK 13,   GR  11     [request:  |a x c| < |b|]
[m+2]   ZQ  0
```

Program 2:

```
[ m+0 ]  ARS 12,   DK  13      [ request:  |a| < |b| ]
[ m+1 ]  XR   0,  MKS  14
[ m+2 ]  GR  11,  ZQ    0
```

Exercise 7.25

Due to the round-off errors the computer does not calculate exactly. Write a program that calculates more accurately than the two mentioned above (use long operations).

Exercise 7.26

Same problem as in example 7.24, but with floating-point numbers instead of machine numbers.

Exercise 7.27

Same problem as in example 7.24, but with integers (stored with the units in pos. 39) instead of machine numbers.

Example 7.28

The machine numbers a and b are placed in cell 100 and 101, respectively, and sign(a) $\times$ b is to be stored in cell 102. (sign(a) = 1 for a $\geqq$ 0 and -1 for a 0).

Program:

```
[m+0]  ARS 101,  MT 100
[m+1]  GR  102,  ZQ   0
```

In the following exercises we assume that the numbers x, y, z, and v are placed in cell 32, 33, 34,and 35, respectively.

Exercise 7.29

The quantity $(x + y)/(z + v)$ is to be stored in cell 31. (x, y, z, and v machine numbers).

Exercise 7.3o

The quantity $\frac{x}{2} + \frac{y}{3} - \frac{|z|}{4} + \frac{|v|}{5}$ is to be stored in cell 30. (x, y, z, and v machine numbers).

Exercise 7.31

Exercise 7.29 with floating-point numbers.

Exercise 7.32

Exercise 7.30 with floating-point numbers, and with the floating-point numbers 2, 3, 4, and 5 stored in some cells.

7.2.3 Examples of Normalization and Number Shift

Example 7.33

We require the machine number a in cell 42 to be normalized. The normalizing exponent must be stored in cell 43, and the normalized number must be stored in cell 44.

Program:

```
[m+0]  ARS  42, NK  43
[m+1]  GR   44, ZQ   0
```

Example 7.34

The machine number x placed in cell 1000 is to be transformed to floating-point number and replaced in cell 1000.

Program:

```
[ m+0 ]  ARS  1000
[ m+1 ]  NKF     0,  GRF  1000
[ m+2 ]  ZQ      0
```

Exercise 7.35

Why is it not possible to pack the instructions in example 7.34 in such a way that they only occupy two whole cells?

<u>Exercise 7.36</u>

How will the program in example 7.34 be affected if 1) the first F in cell [m+1] is excluded? 2) if the second F is excluded? 3) if both are excluded?

<u>Example 7.37</u>

The machine number a is placed in cell 702. We assume that $2^3 \times a$ is a machine number, too. This number must be placed in cell 701.

<u>Program</u>:

```
[m+0]   ARS 702,   TK 3
[m+1]   GR  701,   ZQ 0
```

<u>Example 7.38</u>

We assume that the floating-point number b, placed in cell 98, is in the range $-1 \leq b < 1$. The number is to be transformed to a machine number and stored in cell 99.

<u>Program</u>:

```
[m+0]   ARFS 98,   TKF 10
[m+1]   GR   99,   ZQ   0
```

<u>Exercise 7.39</u>

The machine numbers a and b are placed in cell 100 and 101, respectively. The quotient $\frac{a}{b}$ is to be calculated in the form $c \times 2^d$, where c is a normalized number and d an integer. c and d are to be stored in cell 102 and 103, respectively.

Examples of cyclic shift can be found in section 7.4.

7.2.4 Application of Boolean Operations

<u>Example 7.40</u>

The instruction AB 27 will cause the contents of

cell 27 to be added Boolean to the contents of R.
Let the contents before performance of the operations be

```
              0                    39
cell 27  | 1 0 . . 1 0 1 0 1 1 |

R       |0| 0 0 . . 0 0 1 1 0 0 |
```

After execution of the instruction is obtained

```
        00                         39
R       |1| 1 0 . . 1 0 1 1 1 1 |
```

since ones will appear in those positions where there is a one in at least one of the operands (R and cell).

<u>Example 7.41</u>

The instruction MB 27 causes the contents of cell 27 to be Boolean multiplied by the contents of R.
Let the contents in R and in cell 27 be

```
              0                    39
cell 27  | 1 0 . . 1 0 1 0 1 1 |

R       |1| 0 0 . . 1 0 1 1 0 0 |
```

After execution of the instruction is obtained

```
        00                         39
R       |1| 0 0 . . 1 0 1 0 0 0 |
```

as ones only will appear in those positions where there is a one in both operands (R and cell).

<u>Example 7.42</u>

The effect of X-modification on the instructions AB or MB deviates from the usual interchange of R and M.
Let the contents of R and cell 27 be as follows:

```
              0                    39
cell 27       |1 0 . . 1 0 1 0 1 1|

R          |0|0 0 . . 0 0 1 1 0 0|
```

After the instruction AB 27 X the contents of R and M will be as follows:

```
           00                      39
R          |0|0 1 . . 0 1 0 0 0 0|

M             |1 0 . . 1 0 1 1 1 1|
```

The contents of M are the Boolean sum of the operands, and the new contents of R are ones where the contents of M are zeroes and vice versa. After the instruction MB 27 X R and M will have the following contents:

```
           00                      39
R          |1|1 0 . . 1 0 0 1 1 1|

M             |0 0 . . 0 0 1 0 0 0|
```

The contents of M are the Boolean product of the operands. The new contents of R are ones in those positions where the original contents of R and of cell 27 differ, and zeroes in the other positions.

We point out that the logical operations are performed on 41 positions with R_{00-39} as one operand and the contents of the cell with duplicated sign as the other operand.

Exercise 7.43

Let R have ones in the positions 00, 1, 3, ..., 39, and zeroes in the positions 0, 2, 4, ..., 38. Furthermore, let cell 100 have ones in the positions 0, 2, 4, ..., 38, and zeroes in the positions 1, 3, 5, ..., 39. What are the contents of R and M after performance of the operations:

1) AB 100
2) MB 100
3) AB 100 X
4) MB 100 X

Exercise 7.44

Let R have the same contents as in exercise 7.43. What are the contents of R after performance of the operations:

1) AB 5 D
2) MB 5 D
3) AB 5 DX
4) MB 5 DX

7.2.5 Examples of Placing and Storing Operations

Example 7.45

Using the instruction PM 307 D the number 307 is placed in M with the units in pos. 9. The remainder of M is cleared.

Example 7.46

Using the instruction PP 47 the number 47 is placed in the p-register.

Example 7.47

Using the instruction PS 133 the number 133 is placed in the s-register.

Example 7.48

The instruction PA 36 + 14 causes the increment +44 to be placed in cell 36 with the units in pos. 9, i.e. as an address constant in cell 36.

Example 7.49

The instruction PT 236 + 44 causes the increment + 44 to be placed in cell 236 with the units in pos. 19, i.e. as an increment in cell 236.

Exercise 7.50

Write a program which places the numbers 7, 9, and 13 in pos. 0-9 of the M-register, p-register, and s-register, respectively.

Exercise 7.51

Write a program which places the numbers 22 and -22 with the units in pos. 9 and pos. 19 of cell 7, respectively.

Example 7.52

Let us imagine the instruction PA r+42 D +17 to be placed in cell 278. The instruction will be adjusted to PA r+17 D+17.

Example 7.53

The instruction PA (r+42) D +17 which is imagined placed in cell 278 causes the number 17 to be placed in cell 320 with the units in pos. 9, i.e. as an address constant in cell 320. (Which assumption is made?).

Exercise 7.54

PT r+1 D -48, PT (r+1) D -86, and ZQ 0 are placed in the cells 25,26, and 27, respectively. What are the contents of the three cells after execution of the program?

Example 7.55

Let the address constant and the increment of R be 7 and 20, respectively. The instruction

```
GR p+408 D +12
```

will thus be adjusted to GR p+7 D +20, after execution.

Exercise 7.56

The instruction in example 7.55 is changed to GR (p+408) D +12. The number 1012 is placed in the p-register. What is the effect of the instruction now?

Example 7.57

Let the address constant and the increment of M be 207 and -413, respectively. The instruction

```
GM s+4 D +11
```

will thus be adjusted to GM s+207 D -413, after execution.

Exercise 7.58

What is the effect of the following program:

```
[m+0]   PM r+7   -6
[m+1]   GM (r-1) D -49
[m+2]   ZQ  0
```

Example 7.59

Let the address constant and the increment of R be 39 and 93, respectively. The instruction

```
GA r+4 D -17
```

will then adjust itself to GA r+39 D -17. The instruction GT p-411 D -373 will adjust itself to GT p-411 D +93.

Exercise 7.60

What is the effect of the following program:

```
[m+0]   ARS  r-4  +7
[m+1]   GA  (r+1)  D
[m+2]   GT  (r+2)  D
[m+3]   PP  (r+1)  +32
[m+4]   ZQ  4   +5
```

Example 7.61

Let the contents of the s-register be 378. The instruction GP p+3 will then cause the number 378 to be stored in pos. 0-9 in cell 381. The instruction GP p+3 D will adjust itself to

GP p+378 D.

Example 7.62

Let the contents of the s-register be 788. The instruction GS 47 will then cause the number 788 to be stored in pos. 0-9 in cell 47. The instruction GS 47 D will then adjust itself to GS 788 D.

Exercise 7.63

What is the effect of the following program:

```
[ m+0 ]  PP 42,   PS -21
[ m+1 ]  GP s+1, GS p+7
[ m+2 ]  GS 42 D
[ m+3 ]  GP 19 D
[ m+4 ]  GS (r-3) D
[ m+5 ]  GP (r-2) D
[ m+6 ]  ZQ  0
```

Examples of operations involving the indicator are dealt with in section 7.3 below.

7.2.6 Examples of Substitution Instructions

Example 7.64

Let the s-register contain the number 25. We consider the following programs:

Program 1:

```
[ m+0 ]   IS  20, ARS s+2
[ m+1 ]   AR 400, ZQ    0
```

Program 2:

```
[ m+0 ]  IS 20, ARS 400
[ m+1 ]  AR s+2,ZQ    0
```

In program 1 the IS-instruction is immediately followed by an s-indexed instruction. The effect of program 1 will be that the sum of the numbers in the cells 22 and 400 is obtained in the accumulator. In program 2 the IS-instruction has no effect. The s-register remains unchanged in both cases.

__Exercise 7.65__

What is the effect of the following program, if the contents of the s-register are 200 :

```
[ m+0]   IS   50,    ARS s+10
[ m+1]   AR s+10,    ZQ     0
```

__Exercise 7.66__

What is the effect of the following program stored in cells 10-12:

```
[ 10]   PS   2,   IS 11
[ 11]   ARS  (s+1)
[ 12]   AR s+1,  ZQ   0
```

__Example 7.67__

The instruction NS r+82 D +167 is placed in cell 37. The instruction will cause the number -37 to be used as the s-value in the instruction (the first instruction) in cell 38. (And the instruction itself is adjusted to NS r+249 D +167).

__Example 7.68 The IT-instruction__

We consider the following program:

```
[m+0]   IT      4   +3
[m+1]   ARS   100  +20
[m+2]   ZQ      0
```

The final address in the IT-instruction is 7, which is used as increment in the following instruction. The program will cause the contents of cell 107 to be transferred to R.

Exercise 7.69

What are the contents of the cells [m+0], [m+1] ,and [m+2] in example 7.68 after execution of the program?

Example 7.7o

The instruction NT s-14 D -77 is placed in cell 888. The instruction will cause the number -888 to be used as increment in the instruction (the first instruction) in cell 889.

Exercise 7.71

What is the effect of the following program placed in cell 100 and onwards:

```
[100]   IT    r+1    D +20
[101]   IT    r-1      +40
[102]   SRS   r-2      +60
```

Example 7.72

The operation IT should only be used in a left-hand half-word instruction, if the right-hand half-word instruction is non-adjustable 1) because otherwise the effect is destructive: The increment (i.e. the address adjustment) occurs in the address of the IT-instruction itself (in pos. 0-9 of the cell). If, for instance, we write the instructions

1) A half-word instruction containing an operation for which the increment is active (i.e.it can adjust the modified address), is only adjusted if the preceeding instruction is an IT-instruction, otherwise the increment is automatically set equal to 0.

```
IT  4, AR r+17
```

they will be executed as AR r+21, but after execution of the instructions are changed to

```
IT 21, AR r+17
```

It will be even worse if the IT-instruction has an indirect address: The instructions

```
[ m+0 ]  GR 3 X
[ m+1 ]  IT (r-1), AR r+4
```

will be executed as GR 3 X followed by AR r+7, but in the process the instructions will have been changed to

```
[ m+0 ]  GR 3 X
[ m+1 ]  IT (r+7), AR r+4
```

because the increment in the addition instruction is always made in the address positions 0-9 of this full-cell, i.e. in the address of the IT-instruction.

If, on the contrary, the addition instruction is indexed the increment takes place in the normal way in the final address constant; for instance the instructions below

```
[ m+0 ]  GR  3 X
[ m+1 ]  IT (r-1), AR (r+1)
[ m+2 ]  MK  5
```

will be executed as GR 3 X, then AR 8 and finally MK 8, as increment takes place in this last instruction.

If the IT-operation is placed in a right-hand half-word instruction or in a full-word instruction, it will affect the execution of the instruction (or the left half-word instruction)in the next cell in a more normal way, as increment now takes place in the address positions 0-9 of the cell, which positions just contain the address constant of the instruction.

Finally, if there is a non-adjustable instruction after the IT-instruction, no problems of the placing arises, because no modification takes place in the instruction cell.

Example 7.73 The IS-instruction

As pointed out in the operation list the IS-instruction only influences the address of the instruction that follows immediately after the IS-instruction. However, it also influences the final address in a chain of brackets. For instance the instructions

```
[m+0]    IS 13, AR (r+1)
[m+1]    ZQ s-1
```

are executed as AR 12 independent of the contents of the s-register. On the other hand the IS-instruction works only with the first s-indexing in a chain of brackets.

Thus the instructions

```
[14]    SRS s-2,    AR r+25
[15]    IS   13,    AR (s+1)
```

will cause the effective address in the last instructions to be evaluated from the first address constant in cell 14, i.e. s-2, and here the increment constant in cell 13 replaces s, since the address of the last instruction containing both s and brackets, starts the subroutine mechanism (see address calculation page 63 and IS-instruction page 78).

Note that the IS-instruction substitutes the contents of the auxilary subroutine index register s2 with its final address whereas s2 is normally set equal to s1, the main subroutine index register. s2 is thereafter used exclusively in the address calculations and each time it is used its contents are subsequently set equal to pos.10-19 of the cell whose address is contained in s2.

7.2.7 Conditional Instructions

Example 7.74

The effect of the instruction BT p+37 +19 is that the instruction(s) stored in the following cell will only be executed if the contents of p are such that p+37+19 > 19 (the value of p+37+19 is regarded as a number within the range $-512 \leq x \leq 511$. Furthermore, the BT-instruction is changed to BT p+56+19.

Exercise 7.75

Examine the effect of the instruction in example 7.74 for the different possible values of p.

Example 7.76

The effect of the instruction BS p+37+19 is that the instruction(s) stored in the following cell will only be executed if the contents of p are such that p+37 > 19 (the value of p+37 is regarded as a number within the range $-512 \leq x \leq 511$.

Exercise 7.77

Examine the effect of the instruction in example 7.76 for the different possible values of p.

Example 7.78

The instruction BS s-37 D +50 has no effect unless it is placed in a cell, the address of which is larger than 50.

Exercise 7.79

The instruction in example 7.78 is changed by writing a bracket round s-37. In which cases will the instruction be effective now?

Example 7.80

The effect of the instruction CA 42 +28 is that the instruction(s) stored in the following cell will only be executed when the address constant in R is 70. Besides, the CA-instruction is changed to CA 70 +28.

Exercise 7.81

Let the address constant of R be 70. When is the instruction CA 42 D +28 effective?

Example 7.82

The instruction NC r-19 +39 is only effective when the address constant of R is exactly r+20. (However, under all circumstances the address constant of the NC-instruction will be changed.)

Exercise 7.83

What is the effect of the following program:

```
[m+0]   ARS r+1   V
[m+1]   ARS (s+37) D +114
[m+2]   CA -19   X +56
[m+3]   MKS  37 D
[m+4]   ZQ  0
```

Exercise 7.84

Let the address constant of R be 403. When is the instruction NC p+42 DVX-413 effective.

Example 7.85

Let the contents of pos. 10-19 of the M-register be ones, and otherwise zeroes. The effect of the instruction

CM 708

is then that the contents of pos. 10-19 (the increment) of cell 708 are compared with the contents of the corresponding positions of the R-register. If they are identical the instruction(s) stored in cells up to and including the following right-hand half-cell will be skipped.

7.2.8 Examples of Jump Instructions

Example 7.86

The following fragment of program

```
[ m+0 ]  AR 100,   GR 100
[ m+1 ]  HV r-1
```

will cause the addition instruction and the storage instruction to be repeated over and over again until GIER is stopped manually. On the other hand

```
[ m+0 ]  AR 100,   GR 100
[ m+1 ]  HH r-1
```

will only cause the storage instruction to be repeated again and again until GIER is stopped. An efficient stop-lock can be made by means of the two instructions

```
ZQ 0,   HV r
```

and GIER will stop again as soon as NORMAL START-button is pressed. (This will prevent inadvertent running of a program which is placed immediately after a stop instruction).

Example 7.87

If four machine numbers of cells 100-103 are to be added together and the sum stored in cell 104 this can be done by means of the following program (provided that R is cleared in advance):

```
→[m+0]  AR 99 +1
 [m+1]  BT  3 -1 [next instruction is only executed 3 times]
 [m+2]  HV r-2   [jump back to the addition instruction]
 [m+3]  GR 104, ZQ 0
```

In this case where the full-word instruction is contained in the cell to which the jump is performed we could just as well have used a HH-instruction. The program

```
→[m+0]  AR 99 +1
 [m+1]  BT  3 -1
 [m+2]  HH r-2
 [m+3]  GR 104, ZQ 0
```

will have the same effect (see also example 122 that shows many other ways of programming such a calculation).

Example 7.88

Let the instruction HS r+12 be placed in cell 440 and let the contents of the s-register be 67. The instruction will then cause the next instruction to be taken from cell 452. Furthermore, the contents of the s-register, i.e. 67, are stored as the increment constant of the instruction (this only occurs when the HS-instruction is a full-word instruction). Finally, the location of the HS-instruction, i.e. 440, is placed in the s-register. After execution, the instruction is thus HS r+12 +67.

Example 7.89

Let us consider the instruction HR s+1 in relation to the previous example. Since the s-register contains the number 440, the final address becomes 441. The instruction will then cause the next instruction to be taken from cell 441, and besides the increment, i.e. 67, is transferred to the s-register.

Exercise 7.90

What is the effect of the following program which we can assume to be stored from cell 20 and onwards, if the s-register contains the number 2:

```
[ 20 ]  HS  r+3
[ 21 ]  HS  r+2
[ 22 ]  HR  r+2
[ 23 ]  HR  r-3  +1
[ 24 ]  ZQ    0
```

Example 7.91

A HK-instruction has the same effect as the corresponding HS-instruction provided that no drum transfer is being made. Otherwise the HK-instruction has no effect. (However, a HKS-instruction will always clear the R-register).

Example 7.92

It is desired that GIER shall jump to cell 100 from an instruction in cell 42. This cell contains a full-word instruction without indirect addressing. No drum transfer is being made. How many of the following instructions could be used in cell 42?

```
HV 100        HV 100 D        HV (100) D
HH 100        HH 100 D        HH (100) D
HS 100        HS 100 D        HS (100) D
HR 100        HR 100 D        HR (100) D
HK 100        HK 100 D        HK (100) D
```

7.2.9 Examples of Drum Instructions and Administration of Peripheral Units.

Example 7.93

We consider the following program:

```
[ m+0 ]   VK 17, LK 440
[ m+1 ]   VK 18, LK 480
[ m+2 ]   VK 19, SK 440
[ m+3 ]   VK 20, SK 480
[ m+4 ]   ZQ  0
```

The effect of the program will be that the contents of track no. 17 (i.e. the contents of the 40 cells on the track) are transferred to track no. 19, and the contents of track no. 18 are transferred to track no. 20.

Example 7.94

If we want the sum of the first element of track 200 and the first element of track no. 201 to be stored in cell 1015 (as floating-point numbers) the code can be made as follows:

```
[ m+0] VK 200, LK    974[track 200 is read to cell 974-1013 ]
[ m+1 ] VK 201, LK   975[track 201 is read to cell 975-1014 ]
[ m+2 ] VK    0, ARSF 974
[ m+3 ] ARF 975, GR 1015[addition and storage ]
[ m+4 ] ZQ      0
```

Here the VK 0 instruction ensures that drum transfer is completed before treatment (addition) of the elements is started, and this is a necessary condition.

Exercise 7.95

The number a is placed in cell 12 on track 42 and the number b in cell 27 on track 45. The number $c = |a| - |b|$ is to be stored in cell 5 on track 71. (We assume that the numbers a and b are machine numbers. Calculation of c must not start during drum transfer).

Example 7.96

Before input to the computer or output from the computer take place, the peripheral units to be used must be selected. The diagram below illustrates 8 possibilities for assignment of the apparatus (a + means connected, otherwise not connected):

Input		Output		Can be achieved using the instruct.
Tape reader	Type-writer	Tape punch	Type-writer	
+				VY 0
	+			VY 1
+			+	VY 16
	+		+	VY 17
+		+		VY 32
	+	+		VY 33
+		+	+	VY 46
	+	+	+	VY 49

The VY-instructions mentioned in the diagram determine assignment of the input/output units completely, but it is also possible to select a certain input unit independent of the output and without changing the output situation and vice versa by means of a VY-instruction with an increment as mask. The instruction

VY 0 +48

will, for instance, select the tape reader without influencing the assignment of the output units. The instruction

VY 32 +7

similarly connects the tape punch to the output channel without influencing the assignment of the input units. (The instructions are unaltered after execution because VY is a non-adjustable operation)

Example 7.97

A primitive program to copy a tape might, in its essence, look like this:

```
[m+0 ] VY 32, LY r+1
[m+1 ] SY  0, HH r-1
```

(however, the program ought to be furnished with some sort of administration to start and stop in a reasonable way).

Further reference to LY and SY is made in volume II, in connection with the standard input and output programs.

7.3 Instructions Using the Indicator

In sections 7.3 and 7.4 several of the operations not used in the previous chapters are reviewed with reference to the indicator.

7.3.1 Examples of Placement and Storage

Example 7.98

Let us consider the instruction PI 20 +3 and re-write the modified address, here 20, and the increment, here 3, in the binary system. The numbers are to be written with the units in pos. 9.

pos.	0	1	2	3	4	5	6	7	8	9
20 =	0	0	0	0	0	1	0	1	0	0
3 =	0	0	0	0	0	0	0	0	1	1

The effect of the instruction will thus be that bit no. 8 and 9 in the indicator are <u>not</u> changed (since the number 3 has ones in the corresponding positions), while bits nos. 0-7 acquire the value of 00000101.

Example 7.99

The instruction GI 37 +88 will cause the contents of the indicator (except KA and KB) to be stored in pos. 0-9 of cell 125.

Exercise 7.100

Find the effect of the following program stored in cell 37-39. The s-register contains the number 900.

```
[37]  PI s+179 +33
[38]  GI s+179 D +33
[39]  ZQ  0
```

7.3.2 Examples of Instructions with Indicator Instructions

Example 7.101 Example of Indication of Overflow

The effect of the instruction AR 37 IOB will be, primarily, that the number in cell 37 is added to the contents of the R-register. If the sum, s, is outside the range $-1 \leqq x < 1$, overflow will be registered in OB (i.e. OB:= 1) on account of the indicator instruction. Overflow will also occur in the R-register, and the correct half sum can be formed in R by means of the instruction TK -1.

The effect of the instruction AC 37 IOB will be, primarily, that the sum s = cell[37] + R is formed in the H-register, from which bits nos. 0-39 are transferred to cell 37. If s is in the range $1 \leqq s < 1$, the effect of the indicator instruction will be that OB:=0; otherwise OB:=1, and the contents of cell 37 will be spoilt. It is very difficult to save the situation because the original contents of cell 37 are lost.

The instruction HV 10 IOB will primarily cause a jump to the first half-word instruction (or full-word instruction) in cell 10. Whether overflow will be registered or not depends on the result of the preceding arithmetic operation.

Example 7.102 Example of Indication of the Zero Situation

The effect of the instruction SNS 682 IZC +2 will be, primarily, that the absolute value of the number in cell 684 is transferred with negative sign to the R-register. Then, if R is equal to 0, OA:= OB:= 1, and if R is different from 0, OA:= OB:= 0.

Example 7.103 Example of Indication of the Zero Situation

Let M = 0.125. The effect of the instruction GRS 37 X IZA will be, primarily, that R:= cell[37] := 0; the zero situation is then registered so that OA:=1. Finally R and M are interchanged.

Example 7.104 Example of Indication of Sign

The effect of the instruction MK 10 ITA will be, primarily, that M×cell[10] is added to R and placed in R. Then the sign of the result is registered. If the result is negative, TA:=1, otherwise TA:=0. This happens no matter whether there is overflow or not.

The effect of the instruction ARS 10 X ITA will be, primarily, that cell 10 is transferred to the R-register; the sign of this number is then registered. Finally the number is transferred to M (and the contents of M to R).

Exercise 7.105

Describe the effect of the instruction XR 10 ITC +2.

Example 7.106 Example of Indication of Marking

The effect of the instruction SR 11 IRB will be, primarily, that cell 11 is subtracted from R and that marking of cell 11 is registered, i.e. is transferred to the marker-bits in R, after which the indicator instruction

IRB causes the b-marking to be registered, in other words that RB:=Rpos[41] := pos[41,11].

The effect of the instruction SR 11 D IRB will be, primarily, that the number 11 · 2^{-9} is subtracted from the number in R; the marker-bits of R are thereby not changed, and the indicator instruction IRB will - as above - cause R_{41} to be transferred to RB (irrespective of the marking in cell 11).

Example 7.107 Example of Marking

The effect of the instruction GR 12 MB+1 is that the number in R is stored in cell 13 and that bits nos. 40-41 in cell 13 are put equal to 01 and in R equal to 11.

The effect of the instruction GR 12 D MB+1 is that the contents of R_{0-19} are stored in the corresponding positions of the cell, in which the instruction is placed. Furthermore, R_{40-41}:= 11. Let the contents of R_{0-9} be 16 and let the contents of R_{10-19} be -2; the instruction will then be changed to GR 16 D MB -2, because marking in the cell is not performed by the operation GRD.

Exercise 7.108

Describe the effect of the instruction GT r+3 D MA+1.

Example 7.109

The effect of the instruction GR 12 MTA will be that the number in R is stored in cell 12 and that bit no. 40 in the cell is put equal to TA, while bit no. 41 is cleared. R_{40} and R_{41} remain unchanged.

Example 7.110

The effect of the instruction AC 100 MQC will be, primarily, that the contents of cell 100 are increased by the number in the R-register, and that the marking in cell 100 is registered; in other words, is transferred to the marker-bits of R. The effect of the indicator instruction MQC will be that the two marker-bits in cell 100 acquire the same value as QA, QB. During this marking operation the marker-bits in R are not affected so that they - when the complete instruction has been executed - will contain the original value of the marker-bits in cell 100.

Example 7.111 Example of Conditional Jump

Let the current instruction be HV r-1 LT. If the sign in the R-register is negative, i.e. if bit no. 00 is equal to 1, the jump instruction is executed, and GIER starts executing the first half-word instruction or full-word instruction in the preceding cell. If the sign is positive the instruction is dummy, and GIER executes the following instruction.

Example 7.112 Example of Conditional Instruction

Let the current instruction be GR p+98 NZB -1. If OB = 0 the contents of the R-register are stored in cell p+97, and the address constant of the instruction is changed to p+97. If OB = 1 the instruction is dummy, and neither storage nor change of address take place.

Example 7.113 Example of Conditional Stop

Let the current instruction be ZQ 0 LKA. If the operator has previously put KA equal to 1, GIER stops (when NORMAL START is pressed it carries on with the next instruction independent of the address). Otherwise GIER will perform the next instruction at once.

Example 7.114 Example of Conditional Jump

Let the current instruction be HH 17 V NPB +2. If PB = 0, GIER will jump to the right-hand half-cell no.20 and start executing the second half-word instruction or full-word instruction; the jump instruction is further-more changed to HH 19 V NPB +2.

If PB = 1 the instruction remains unchanged, and GIER starts executing the instruction in the immediately following cell (the V-modification is not effective either).

Exercise 7.115

Describe the effect of the instruction HV s-3 D LRC -3.

7.4 Examples of Programs

Example 7.116 Example of Interchange

The numbers in cell 102 and 103 are required to be interchanged. This can be done as follows:

```
[ m+0 ]  ARS 102,  PM 103   [R:= x,  M:= y ]
[ m+1 ]  GR  103,  GM 102   [storage]
[ m+2 ]  ZQ    0            [stop]
```

Note that this program functions equally well no matter whether the numbers are machine numbers or floating-point numbers (or in any other form of representation). The marker-bits of the cells are <u>not</u> changed by the above instructions.

Exercise 7.117

a) The machine numbers x, y, z, and v are stored in cells 800, 801, 802, and 803. Compute (x+y)(z-v) and store the result in cell 799. In cases of overflow in the interim results or in the product, a jump to cell 0 should be made. b) The same computation is to be made with x, y, z, and v as floating-point numbers.

Exercise 7.118

a) Let a, b, and c be three machine numbers stored in cells 50, 51, and 52, and we assume that $c \neq 0$ and $abs(a) \leq abs(c)$; ab/c is to be stored in the cell situated just before the cell containing the first instruction of the program.

b) Let three machine numbers a1, b1, and c1 be stored as above. If $a1 \times b1/c1$ is a machine number this is to be stored as before; if not, jump to cell 0 if $c1 \neq 0$, and jump to cell 10 if $c1 = 0$.

c) Same problem as in b), a1, b1, and c1 being floating-point numbers.

Exercise 7.119

Let the integers x and y be stored in cell 47 and 48 with the units in pos. 39. The same cells are to be used for storage: the integer quotient x/y is to be stored in cell 47 and the appropriate remainder in cell 48; the remainder must be ≥ 0.

Example 7.120 Example of Address Calculation

The integers N and J are placed as the address constants of cells 859 and 865. It is desired to place NJ-1 as address constant of the cell s+2.

```
[m+0] PM (859) D
[m+1] MKS(865) D     [N×J is placed with the units in
                      pos. 18 of R ]
[m+2] TK 9           [N×J with the units in pos. 9 of R]
[m+3] SR 1 D         [N×J -1 in R0-1 ]
[m+4] GA s+2, ZQ 0   [N×J -1 is stored in cell s+2]
```

Note that D-modification has been used in the instructions which refer to the address constants in cells 859 and 865 in order to avoid that possible contents of the other positions in these cells should influence the address calculation. If it is known that pos. 10-18 of the two cells are cleared (if, for instance, the increments are 0) the two first instructions in the above can be replaced by two half-word instructions

PM 859, MKS 865

because the remaining contents (pos. 19-39) of the two cells cannot spoil the significant result of the multiplications of the address constants.

Exercise 7.121

a) The integers A, B, and C are placed as the address constants of the instructions in cell s-3, s-2, and s-1. A program is required (concluded by a stop instruction) to place A+4B-C as an address constant in the full-cell immediately after the stop instruction.

b) The same problem is to be solved with the increments in cell s-3, s-2, s-1 equal to 0.

Example 7.122 Example of Program Loops

In many programs certain parts of the program are repeated many times, since the same operations are to be performed for many numbers. In order to use the same part of the program it is only necessary to change the addresses in some of the instructions for each iteration, and this can be done by means of increments.

In addition the program must contain a device which controls the number of times the part of the program (loop) has been run through, and in GIER this can be done in many ways.

Finally, the start of a program should contain a number of instructions that reset (initialize) those parts of the instructions which have been changed during a complete cycle. The reason is that the whole program often is to be used several times after each other (for instance with different sets of data).

For instance, let 50 machine numbers be stored in cell 200 and succeeding cells. The sum of the absolute values of these numbers is to be placed in the M-register.

Draft program no. 1:

```
 [ m+0 ]  PAS r+1 +199    [ reset the address constant 199 ]
→[ m+1 ]  AN  199 +1      [ add the absolute values
|[ m+2 ]  BS  (r-1) +248  [ the next two instructions are
|                           not executed until 199+j > 248]
|[ m+3 ]  XR  0, HH r+1   [ exchange R and M, jump to stop ]
└[ m+4 ]  HV  r-3, ZQ 0   [ jump back the first 49 times, stop ]
 [ m+5 ]  HV  r-5         [ by pressing NORMAL START the whole ]
                            program is executed once more]
```

The following program occupies less storage, but the time used for calculation is nearly the same. Addition of the numbers takes place in reverse order.

Draft program no. 2:

```
 [m+0]  PAS r+1 +250      [reset the address constant 250]
→[m+1]  AN  250-1         [add the numbers absolutely]
|[m+2]  BS  (r-1) X+200   [next instruction is performed as
|                          long as 250-j> 200]
└[m+3]  HV  r-2 X         [jump back the first 49 times]
 [m+4]  ZQ  0, HV r-4     [stop, start once again by pressing
                           NORMAL START ]
```

The two cases of X-modification are superfluous the first 49 times, where the running total is simply moved to M and back again, but the 50th time the first X-modification only is performed, whereby the final sum is placed in M.

Draft program no. 3:

```
 [m+0]  PA  r+3
 [m+1]  PAS r+1  +199
→[m+2]  AN  199 +1
|[m+3]  IT  0   +1
└[m+4]  BS  50, HV r-2
 [m+5]  XR  0, ZQ   0
 [m+6]  HV  r-6
```

For all three drafts the calculation time is nearly the same, since the central part of the program, namely the loop, which is to be run through 50 times, is almost like in the three cases.

By using marking in connection with an indicator instruction a loop of only two instructions can, however, be formed; assuming that only the last number in the group of numbers is a-marked, we get

Draft program no. 4:

```
    [ m+0 ] PAS  r+1  +199   [ reset the address constant 199]
  ↱ [ m+1 ] AN   199  +1     [ add the numbers absolutely.
  |                            Marking is registered ]
  └-[ m+2 ] HV   r-1  NA     [ jump back while no a-marking ]
    [ m+3 ] XR     0,  ZQ   0[ move the sum to M, stop ]
    [ m+4 ] HV   r-4         [ repeat the whole program by
                               pressing NORMAL START ]
```

Note: While the three first drafts only work correctly if the array consists of exactly 50 numbers, the fourth draft can be used to sum an array of arbitrary length (provided that it is stored from cell 200 and onwards and that only the last number is a-marked).

A useful tallying device can be formed by using the p-register as follows:

Draft program no. 5:

```
    [ m+0 ] PPS  0, AN p+200
    [ m+1 ] PP   p+1, IT p
    [ m+2 ] BS   50, HH r-2
    [ m+3 ] XR    0, ZQ   0
    [ m+4 ] HV   r-4
```

Merely to demonstrate the mainfold possibilities of the GIER instruction let us show yet another two methods of solving the same problem.

Draft program no. 6:

```
 [ m+0]  PAS r+1 +199
┌>[ m+1]  AN  199 X+1
│ [ m+2]  ARS r-1, NC 249
└─[ m+3]  HV  r-2 X
 [ m+4]  ZQ  0, HV r-4
```

Draft program no. 7:

```
 [ m+0]  PA  r+3 +49
 [ m+1]  PAS r+1 +199
┌>[ m+2]  AN  199 X+1
│ [ m+3]  BT  49 -1
└─[ m+4]  HV  r-2 X
 [ m+5]  ZQ  0, HV r-5
```

The last-mentioned method, where tallying occurs in the BT-instruction, has proved very useful. This method can easily be generalized. For instance, a program loop consisting of 20 cells which are to be run through K+1 times looks like this:

```
  ...
  PA r+20 +K
┌>...
│ ...
│ BT K t-1
└─HV r-20
  ...
```

Exercise 7.123

a) N machine numbers are stored in cells 300, 301, ... Write a program, which places the biggest of these numbers in R, where N is stored as the address constant of cell 299.

b) The same problem when it is known that only the last number is c-marked. The test for this should be made using the indicator instruction LPC, not NPC (or the like) Why?

Example 7.124

The machine numbers x_0, x_1, ..., x_{100} are stored in cells 100-200. Only cell 200 is a-marked. Σa_i is to be formed in cell 250.

Let us solve the problem by converting the machine numbers to floating-point numbers using floating-point addition to form the sum.

```
 [m+0]  PAS r+1 +99
→[m+1]  ARS 99 +1          [ machine number in R ]
|[m+2]  NKF 0, GRF (r-1)   [ replace with floating-point
|                            number]
└[m+3]  HV  r-2 NA         [ all numbers are now floating-
                             point numbers ]
 [m+4]  PAS r+1 +99
→[m+5]  ARF 99 +1          [ program summation]
└[m+6]  HV r-1 NA
 [m+7]  GRF 250, ZQ 0
```

Exercise 7.125

Same problem as in 7.124, but in this case only one loop is to be used.

Exercise 7.126

In the cells 100 to 150, 51 floating-point numbers are stored. Only cell 150 is b-marked. The numbers which can be converted to machine numbers must be converted and stored in their original cells, which are to be a-marked.

Example 7.127 Example of Sorting

The machine numbers a_0, a_1, ..., a_{100} are placed in the cells 200, 201, ..., 300, and only cell 300 is a-marked. The address of the biggest of these numbers must be placed as address constant in cell 400:

```
  [m+0]   PA   r+5 +200           [reset the address constant of
                                   the SR-instruction]
  [m+1]   ARS 200, GR r+2         [place a0 in a working cell]
  [m+2]   PA   400 V +200         [place meanwhile the address
                                   constant 200 in cell 400]
  [m+3]   QQ   0                  [working cell]
┌→[m+4]   ARS r-1                 [R:= working cell]
│ [m+5]   SR   200 IPA +1         [R:= working cell - ai,
│                                  a-marker-bit in indicator]
│┌[m+6]   HV   r+3 NT             [jump forward if R ≥ 0]
││[m+7]   ARS  (r-2), GR r-4      [place new ai in working cell]
││[m+8]   ARS r-3, GA 400         [place address of ai in cell 400]
└└→[m+9]  HV   r-5 NPA            [repeat until first a-marking]
  [m+10]  ZQ   0
```

Example 7.128 Example of Drum Operations

The drum tracks 37 and 38 contain the machine numbers a_0, a_1, ..., a_{39} and b_0, b_1, ..., b_{39}, respectively while the machine numbers c_0, c_1, ..., c_{39} are stored from cell 50 and onwards; only cell 89 is a-marked. The numbers $d_0 = a_0 b_0 + c_0$, $d_1 = a_1 b_1 + c_1$, ...,$d_{39} = a_{39} b_{39} + c_{39}$ must be stored on track 39

```
 [m+0]   VK  37, LK 90
 [m+1]   VK  38, LK 130         [the two trakcs are transferred
                                 to the store ]
 [m+2]   VK  39, PP 0           [drum transfer is completed and
                                 tallying is initialized ]
┌→[m+3]  PP p+1, PM p+89        [tallying in the p-register, M:= ai]
│[m+4]   ARS p+49 IQA           [R:= Ci, a-marking is stored ]
│[m+5]   MK p+129, GR p+89      [di is stored in cells nos. 90,
│                                91, ..... ]
└[m+6]   HV r-3 NQA             [repeat until first a-mark]
 [m+7]   SK 90, ZQ 0            [di is moved to track 39]
```

Note that the location of the instruction VK 39, before the cells 130-169 incl. are used, is very decisive.

Exercise 7.129

Same problem as in 7.128 with the difference that neither the p-register nor marking of the cells are to be used.

Example 7.130 Square Root Calculation

With a jump to cell[m+0] of the following program the square root of the contents of R, regarded as a machine number, will be calculated, as long as the number is positive or zero. At exit the square root is placed in R. With a jump to cell [m+12] the square root of the contents of RF, regarded as a floating-point number, will be calculated, as long as the number is positive or zero. The result is placed in RF before exit from cell [m+19] .

Calculation of the square root y of the non-negative machine number x is accomplished using iteration after the formula

$$y_{n+1} = \tfrac{1}{2}(y_n + \frac{x}{y_n})$$

$$y_0 = x \cdot 2^u$$

where u is selected so that

$$\tfrac{1}{4} \leqq x \cdot 2^{2u} < 1$$

The floating-point number can be written $p \cdot 2^q$. If q is even, $\sqrt{\frac{p}{2}} \cdot 2^{\frac{1}{2}(q+2)}$ is evaluated, and if q is odd $\sqrt{\frac{p}{2}} \cdot 2^{\frac{1}{2}(q+1)}$ is evaluated.

```
Machine numbers  →[m+0]  PA r+8 +4
                  [m+1]  GR r+9, NK r+1
                  [m+2]  SRS 0 D
                  [m+3]  TK -11, GT r+1
                  [m+4]  ARS r+6, TK 0
                 ┌→[m+5]  GR r+6, ARS r+5
                 │[m+6]  DK r+5, AR r+5
                 │[m+7]  TK -1, IT -1
                 └─[m+8]  BT  4, HV r-3 [iteration is performed
                                         5 times]
                  [m+9]  HR s+1         [exit for machine number]
                  [m+10] QQ 0
Floating-point    [m+11] QQ 0
numbers ──────→   [m+12] GR r-2 X
                  [m+13] AR 2 D
                  [m+14] TK -1, GA r+5
                  [m+15] TK 10, CK -19
                  [m+16] GT r+1, ARS r-6
                  [m+17] TK 9, TK 1
                  [m+18] HS r-18
                  [m+19] NKF 0, HR s+1 [exit for floating-
                                        point number]
```

Exercise 7.131

Write comments to the instructions in example 7.130

Exercise 7.132

The program in example 7.130 is to be changed so that a jump to cell s+1 is performed if the given number is negative, otherwise a jump to cell s+2. If the given number is equal to 0 the jump must be performed at once.

Exercise 7.133

Make a program especially adapted to calculate the square root of floating-point numbers.

Example 7.134 Calculation of n!

The following program calculates n! = 1 · 2 · 3 · .....·n as a floating-point number in cell s+1, when n is placed in the address part of cell s-1.

Program:

```
[ m+0 ]  ARS (s-1) DV +1
[ m+1 ]  QQ 0 +256                [1 as floating-point number ]
[ m+2 ]  ARSF r-1, GRF s+1
[ m+3 ]  ARS (s-1) D - 1
[ m+4 ]  HR s+2  LZ
[ m+5 ]  NKF 9, MKF s+1
[ m+6 ]  HH r-4
```

Exercise 7.135

Write comments to the instructions in example 7.134.

Example 7.136 Application of Subroutines

Programs, which are able to solve problems that occur frequently, are normally prepared as subroutines. For an illustration of such subroutines one can refer to example 7.130 (square root) and example 7.134 (n!). For such subroutines two main requirements should be fulfilled:
1) The subroutine should be able to work at any place of the store. 2) The user should be able to jump to the subroutine from anywhere in his program, and the subroutines should automatically ensure that the correct return jump is made. The first requirement is fulfilled by using relative addressing in the subroutine, and the second by using HS- and HR-jump.

Let us illustrate the intercourse between the HS- and HR-jump, respectively, in the main program, i.e. the user's own program and the subroutine, which may, for instance, be the subroutine for n!. Let us assume that the programmer in his program needs 12!

```
Main program,                Subroutine,
stored from 100              stored from 800

[100]  ......                [800]  ......
  :                            :
[117] PA 0 D +12             [804]  HR s+2  LZ
[118] HS 800                        ......
[119] QQ 0
[120]  ......
```

The effect of the instruction in cell [118] is

1) The contents of the s-register are placed as the increment in 118.
2) The contents of the s-register will then become 118.
3) The next instruction is taken from cell 800.

The effect of the subroutine is now that 12! is calculated and stored in cell s+1, i.e.in cell 119, and the effect of the instruction in cell 804 is then:

1) The next instruction is taken from cell s+2, i.e. cell 120.
2) The increment in cell s, i.e. cell 118, is transferred to the s-register, which has now its "old" contents.

<u>Exercise 7.137</u>

A main program requires $\sqrt{13!}$. Place the square root subroutine from 700, and the n!-subroutine from 725. Sketch part of the main program and examine the different jump instructions necessary.

<u>Exercise 7.138</u>

The number 0.05 is stored in cell 799 (as a machine number), and in cell 850 is stored a subroutine that calculates the square root of a number placed in the R-register; the subroutine ends (with the required square root in the R-register) with the instruction HR s+1. Write a program which stores

sqrt (0), sqrt (0.05), sqrt (0.1), ..., sqrt (0.95)

in cells nos. 800, 801, ...

Exercise 7.139

Write two programs which evaluate the quantity $\binom{n}{r} = \frac{n(n-1)\ ...\ (n-r+1)}{1 \cdot 2 \cdot .... \cdot r}$ as a floating-point number in cell s+1, when n and r are placed in the address parts of cell s-2 and cell s-1, respectively:

a) by using the program from example 7.134 as a subroutine
b) by performing the calculation in the following order

$$\frac{n}{r} \cdot \frac{n-1}{r-1} \cdot .... \cdot \frac{n-r+1}{1}$$

When using the first method the number range of the computer will be exceeded even for moderate values of n, whereas the second method can be used for considerably larger values.

Exercise 7.140

A number between 1 and 5 is placed in the address positions of R. Write a program that causes a jump to

cell	100	if	Raddr	is	1
-	120	-	-	-	2
-	140	-	-	-	3
-	160	-	-	-	4
-	180	-	-	-	5

Exercise 7.141

Four integers a,b,c, and d are placed in cell 100 with the units in pos. 9, 19, 29, and 39, respectively.

The numbers are to be placed in cell 101, 102, 103, and 104, respectively with the units in pos. 39.

Exercise 7.142

Data are stored as in exercise 7.141. Form ac + bd in R with the units in pos. 39.

Exercise 7.143

A hundred machine numbers are stored from cell 100. Mark each cell that contains a positive number with an a, and each cells containing zero with a b.

Exercise 7.144

Place the largest of the numbers A and B in cell 5, where A is the smallest of the numbers in cell 1 and cell 2, and B is the smallest of those in cell 3 and 4.

Exercise 7.145

The machine numbers a and b are stored in the cells 100 and 101. Form $-|a| / (|a| + |b|)$ in R. The product must be normalized, and the exponent of normalization must be placed in M_{0-9}.

Exercise 7.146

Same problem as in exercise 7.145. The product is now required in floating-point form.

Exercise 7.147

Two floating-point numbers x and y are stored in cells nos. 20 and 21. $|x| > |y|$. Form $\frac{y}{x}$ as machine number in cell 22.

Exercise 7.148

A set of machine numbers are stored with the first number in cell 200. Some of the numbers are a-marked. The last number is b-marked (and possibly also a-marked). Form the sum of the a-marked numbers in cell 199.

<u>Exercise 7.149</u>

A set of machine numbers are stored from cell 100. Only the last one is b-marked. The numbers must be treated as follows: Negative numbers are halved and positive numbers are doubled up. It is assumed that overflow will not occur.

<u>Exercise 7.150</u>

The same problem as in exercise 7.149, but with floating-point numbers.

<u>Exercise 7.151</u>

Place 1 in each bit of the R-register without the use of stored constants. (It can be done by means of a program that fills two whole-cells incl. stop).

<u>Example 7.152</u>

The machine numbers a_0, b_0, a_1, b_1, ..., a_n, b_n are stored from cell 400. Only b_n is a-marked. $\Sigma\, a_i b_i$ is to be formed in R.

<u>Program</u>:

```
  [m+0]   PAS  r+1   +399
┌→[m+1]   PM   0     +1
│ [m+2]   MK   (r-1) +1
└─[m+3]   HV   r-2   NA
  [m+4]   ZQ   0
```

<u>Exercise 7.153</u>

The same problem as in example 7.152, but with floating-point numbers.

<u>Exercise 7.154</u>

The same problem as in example 7.152, but the product is to be stored unabridged in R,M.

<u>Exercise 7.155</u>

The same problem as in example 7.152. Form $\sum (b_i)^2$ in cell 399 and $\sum (a_i)^2$ in cell 398.

<u>Exercise 7.156</u>

The same problem as in exercise 7.155, but with floating-point numbers.

<u>Exercise 7.157</u>

The machine numbers a_0, a_1, a_2, ..., a_n are stored from cell 100. a_n is b-marked. The number x is placed in R. $y = a_n x^n + a_{n-1} x^{n-1} + \ldots\ldots + a_1 x + a_0$ is to be stored in cell 99. Use the algorithm $y = (\ldots((a_n x + a_{n-1})x + \ldots + a_0)$.

<u>Example 7.158</u>

Let the contents of R be

$$R = \begin{array}{|c|l|} \multicolumn{1}{c}{00} & \multicolumn{1}{l}{0 \qquad\qquad\qquad\qquad 39} \\ \hline 1 & 1\ 0\ 1\ 0\ .\ .\ .\ 1\ 1\ 0\ 1 \\ \hline \end{array}$$

The effect of the instruction CK 3 will be that the contents of R become

$$R = \begin{array}{|c|l|} \multicolumn{1}{c}{00} & \multicolumn{1}{l}{0 \qquad\qquad\qquad\qquad 39} \\ \hline 0 & 0\ .\ .\ .\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \\ \hline \end{array}$$

<u>Exercise 7.159</u>

Let the contents of RM be

$$\begin{array}{|c|l|} \multicolumn{2}{c}{R} \\ \multicolumn{1}{c}{00} & \multicolumn{1}{l}{0 \qquad\qquad 38\ \ 39} \\ \hline 0 & 0\ 1\ 0\ .\ .\ .\ 1\ 0 \\ \hline \end{array} \quad \begin{array}{|c|l|} \multicolumn{2}{c}{M} \\ \multicolumn{1}{c}{0} & \multicolumn{1}{l}{1 \qquad\quad 38\ \ 39} \\ \hline 1 & 0\ .\ .\ .\ 1\ 1 \\ \hline \end{array}$$

What are the contents of R,M after execution of the instruction CL -1?

Example 7.160 Example of Simple Input

For each input/output symbol (i.e. letters, digits, or sign) there is a corresponding integer. And contrary, for each integer in the range $0 \leqq x \leqq 64$ there are two corresponding symbols, namely one in lower case and one in upper case. The symbol which is represented by a particular integer, at any given time, is defined by the case symbol last used.

The numbers 49, 50, and 51 correspond to: In the lower case, a, b, and c, respectively; and, in the upper case, A, B, and C, respectively.

Let us now read a symbol from the tape reader into GIER. If it is an A (or a), a B (or b), a C (or c) a jump must be made to cell 100, cell 120, and cell 140, respectively. Otherwise a new symbol is to be read in and treated in the same way etc.

Program:

```
   [m+0 ]  VY   0
 ┌→[m+1 ]  LYS  0 D  [a symbol is read to R and to this cell]
 │ [m+2 ]  CA   49, HV 100
 │ [m+3 ]  CA   50, HV 120
 │ [m+4 ]  CA   51, HV 140
 └─[m+5 ]  HV   r-4
```

Example 7.161 Example of Output

A program is required which will direct the typewriter to begin a new line, "write" SPACE 10 times and finally write 3/7 1960.

Program:

```
[ m+0 ] VY 16, SY 64     [64 is the value of the symbol for
                          CR ( carriage return) in both
                          upper and lower case ]

[ m+1 ] PA r+1 +10
[ m+2 ] BT 10 -1
[ m+3 ] SY 0, HV r-1     [0 is the value of the symbol for
                          SPACE in both upper and lower case ]
[ m+4 ] SY 58, SY 3      [lower case, write the digit 3]
[ m+5 ] SY 60, SY 3      [upper case, write /]
[ m+6 ] SY 58, SY 7
[ m+7 ] SY  0, SY 1
[ m+8 ] SY  9, SY 6
[ m+9 ] SY 16, ZQ 0      [16 is the value of the symbol for 0]
```

Exercise 7.162

Read 2o symbols (letters or digits) from the tape into GIER and write out the symbols on the typewriter in reverse order with SPACE between each symbol. It is assumed that all symbols are in the same case.

Exercise 7.163

The same problem as in exercise 7.162, but there may be case shifts between the symbols corresponding to the occurrence of capital and small letters.

Example 7.164 Example of Overflow

The machine numbers a_0, a_1, ..., a_{100} are stored from cell 100 to 200. Only cell 200 is a-marked. The sum $s = \Sigma a_i$ is to be formed in cell 201; if overflow occurs during the addition the necessary right-hand shifts are to be performed.

Program:

```
  [ m+0 ]  PA r+4  + 0
  [ m+1 ]  PAS r+2 +99
┌→[ m+2 ]  GR 201
│ [ m+3 ]  ARS 0 IPA +1
│ [ m+4 ]  TK 0, AR 201
│ [ m+5 ]  QQ (r-1) LO -1
│ [ m+6 ]  TK -1 LO
└ [ m+7 ]  HV ı-5 NPA
  [ m+8 ]  GR 201, ZQ 0
```

If overflow occurs during the calculation, the program will shift the partial sum as well as the following addends to the right. The final result given by the program will thus not be s, but s $\cdot$ 2^{-q}, where the exponent -q will be the address constant in cell [m+4].

Exercise 7.165 Exercise with Drum Operations and Boolean Operations.

2000 growths are examined for the existence of 40 kinds of plants. The different kinds are numbered from 0 to 39, and if the existence of a particular kind is denoted by a one, and non-existence by a zero, a storage cell can exactly hold the information about one growth. A collection of information is stored in 2000 cells on the drum on 50 consecutive tracks beginning with track no. 60. Only the last cell on the last track is b-marked. A program is required to:

a) Place in the positions of cell 400 a zero corresponding to each kind of plant which does not exist at all among the 2000 growths; in the remaining positions a one is to be placed

b) Place in the positions of cell 401 a one corresponding to each kind of plant that exists in all 2000 growths, and zero in the remaining positions.

c) Store in cell 402 the number of growths,for which at least the plant-kinds nos. 0, 1, 2, ..., 10 exist; the number is to be stored as an integer with the units in pos. 39.

Exercise 7.166

The same problem as in the previous exercise, but without the use of a- or b-marking.

8. TABLES

8.1 The Numerical Equivalents of the Basic Operations

In the diagram below the basic operations are arranged according to their decimal values, i.e. the contents of pos. 20-25 (30-35, respectively) regarded as an integer.

Value	Operation
0	QQ
1	ZQ
2	AR
3	SR
4	AN
5	SN
6	AC
7	SC
8	MB
9	AB
10	MT
11	MK
12	ML
13	DK
14	DL
15	NK
16	NL
17	HR
18	TL
19	CK
20	CL
21	GR
22	GA
23	GT
24	TK
25	CA
26	GM
27	PM
28	XR
29	GI
30	PS
31	PP

Value	Operation
32	PA
33	PT
34	HK
35	PI
36	IS
37	IT
38	CM
39	BT
40	NS
41	NT
42	GP
43	NC
44	
45	
46	
47	
48	
49	BS
50	HS
51	VY
52	LK
53	SK
54	GK
55	VK
56	HV
57	
58	SY
59	LY
60	HH
61	GS
62	
63	UD

As can be seen from the diagram 7 out of 64 possible combinations are not utilized in the standard GIER.

8.2 Indicator Operations etc.

Overflow is registered in the overflow register after the operations:

$$\left.\begin{array}{l}\text{AR - AN - AC}\\ \text{SR - SN - SC}\\ \text{MK - ML - MT}\\ \text{DK - DL}\\ \text{TK - TL - CK - CL}\end{array}\right\}\ \text{incl. all modifications except F-modifications}$$

Sign may be registered in the indicator after the operations

$$\left.\begin{array}{l}\text{AR - AN - AC - AB}\\ \text{SR - SN - SC}\\ \text{MK - ML - MT - MB}\\ \text{DK - DL}\\ \text{NK - NL - TK - TL}\\ \text{XR}\end{array}\right\}\ \text{incl. all modifications}$$

Any operation can be furnished with ITA, ITB or ITC, but only in the above operations the sign for the arithmetic result is registered.

Marking is registered in R_{40-41} in the operations:

$$\left.\begin{array}{l}\text{AR - AN - AC - AB}\\ \text{SR - SN - SC}\\ \text{MK - ML - MT - MB}\\ \text{DK - DL}\\ \text{PM}\\ \text{CM}\end{array}\right\}\ \text{incl. the S-, F-, X-, and V-modifications}$$

The marker-bits of R remain unchanged in the D-modification of above-mentioned operations, and in all other operations.

A cell can be marked in the operations:

$$\left.\begin{array}{l}\text{AC - SC}\\ \text{GR - GM - GA - GT - GP - GS - GI - GK}\\ \text{NK - NL}\end{array}\right\}\ \text{incl. all modifications}$$

In any absolute marking operation $R_{40,41}$:= 11, as well, whereas, if the marking is dependant on the indicator, $R_{40,41}$ remain unchanged. Cells cannot be marked in the operations GRD and GMD.

8.3 The Flexowriter Punched Tape Code

Symbol		Code
Lower Case	Upper Case	
a	A	`, oo  .  o,`
b	B	`, oo  . o ,`
c	C	`, ooo . oo,`
d	D	`, oo  .o  ,`
e	E	`, ooo .o o,`
f	F	`, ooo .oo ,`
g	G	`, oo  .ooo,`
h	H	`, oo o.   ,`
i	I	`, oooo.  o,`
j	J	`, o o .  o,`
k	K	`, o o . o ,`
l	L	`, o   . oo,`
m	M	`, o o .o  ,`
n	N	`, o   .o o,`
o	O	`, o   .oo ,`
p	P	`, o o .ooo,`
q	Q	`, o oo.   ,`
r	R	`, o  o.  o,`
s	S	`,  oo . o ,`
t	T	`,  o  . oo,`
u	U	`,  oo .o  ,`
v	V	`,  o  .o o,`

Symbol		Code
Lower Case	Upper Case	
w	W	`,  o  .oo ,`
x	X	`,  oo .ooo,`
y	Y	`,  ooo.   ,`
z	Z	`,  o o.  o,`
æ	Æ	`, ooo .   ,`
ø	Ø	`, o oo. oo,`
0	∧	`,  o  .   ,`
1	∨	`,     .  o,`
2	×	`,     . o ,`
3	/	`,   o . oo,`
4	=	`,     .o  ,`
5	;	`,   o .o o,`
6	[	`,   o .oo ,`
7	]	`,     .ooo,`
8	(	`,    o.   ,`
9	)	`,   oo.  o,`
,	ᵒ	`,  ooo. oo,`
.	:	`, oo o. oo,`
-	+	`, o   .   ,`
<	>	`,  oo .  o,`
_	|	`,    o.oo ,`

The following hole combinations correspond to the same 'typographical' symbol in both lower case and upper case.

Typographical Symbol	Code
Car. Return	`,o    .   ,`
Space	`,    o.   ,`
TAB	`,  ooo.oo ,`
LC	`, oooo. o ,`
UC	`, oooo.o  ,`
Tape Feed	`, oooo.ooo,`

Typographical Symbol	Code
Stop Code	`,    o. oo,`
Punch Off	`,  o o.ooo,`
Punch On	`, o  o.o  ,`
Punch Address	`,o    .   ,`
Aux Code	`,    o.o  ,`

The key for _ and | does not advance the carriage. Punch Address and Aux Code only insert their respective codes when pressed simultaneously with any other key.

8.4 Numerical Representation of the Typographical Symbols

In the following table the characters have been arranged according to the numerical equivalent of the hole combinations (after removal of the parity check hole). The table shows for each symbol its numerical equivalent which a) is placed in pos. 3-9 of R and the cell, when it is read into GIER by a LY-instruction ; and b) is to be used as the address constant in an SY instruction when the symbol is to be printed (compare operation list, peripheral units).

Value	Symbol	
	Lower Case	Upper Case
0	Space	
1	1	∨
2	2	×
3	3	/
4	4	=
5	5	;
6	6	[
7	7	]
8	8	(
9	9	)
10	not used	
11	Stop code	
12	not used	
13	å	Å
14	_	|
15	not used	
16	0	∧
17	<	>
18	s	S
19	t	T
20	u	U
21	v	V
22	w	W
23	x	X
24	y	Y
25	z	Z
26	not used	
27	,	⏨
28	not used	
29	red ribbon	
30	Tab.	
31	Punch Off	

Value	Symbol	
	Lower Case	Upper Case
32	-	+
33	j	J
34	k	K
35	l	L
36	m	M
37	n	N
38	o	O
39	p	P
40	q	Q
41	r	R
42	not used	
43	ø	Ø
44	Punch On	
45	not used	
46	not used	
47	not used	
48	æ	Æ
49	a	A
50	b	B
51	c	C
52	d	D
53	e	E
54	f	F
55	g	G
56	h	H
57	i	I
58	Lower Case	
59	.	:
60	Upper Case	
61	not used	
62	black ribbon	
63	Tape Feed	
64	Car. Return	

The symbols å, Å, red ribbon, black ribbon exist only on the on-line typewriter and not on the flexowriter, and thus the instructions SY 13, SY 29, and SY 62 are only effective when output is to the typewriter.

INDEX

(Op) refers to the descriptions in the operation list