stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 34/75 MARCH

P. TEN HAGEN & H. NOOT

A DIGISET SIMULATOR

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

ACM -Computing Reviews- Category: 8.2

A DIGISET SIMULATOR

by

P. ten Hagen & H. Noot

ABSTRACT

In this report, it is shown, that the Digiset-40 T1 photo-typesetter can be used with good results as a device for producing two dimensional hard-copy output generated by a computer program.

The hardware characteristics and instruction set of the Digiset are described in sufficient detail, to enable the use of this report as a Digiset manual.

A Digiset simulation program is given, which translates "shorthand" Digiset code in machine code, performs a syntactical analysis of this code and produces papertape output that can be used as input to the Digiset. The program has some possibilities for the simulation of Digiset output on a line printer or plotter.

A character set is designed in order to use the Digiset as a line-drawingmachine. Examples of line drawings made with the simulator and/or with the real Digiset, are shown and discussed.

KEY WORDS & PHRASES: *Computer graphics, Digiset, Photo-typesetting, Line drawings, Simulation.*

CONTENTS page

ABSTRACT

APPENDIX A

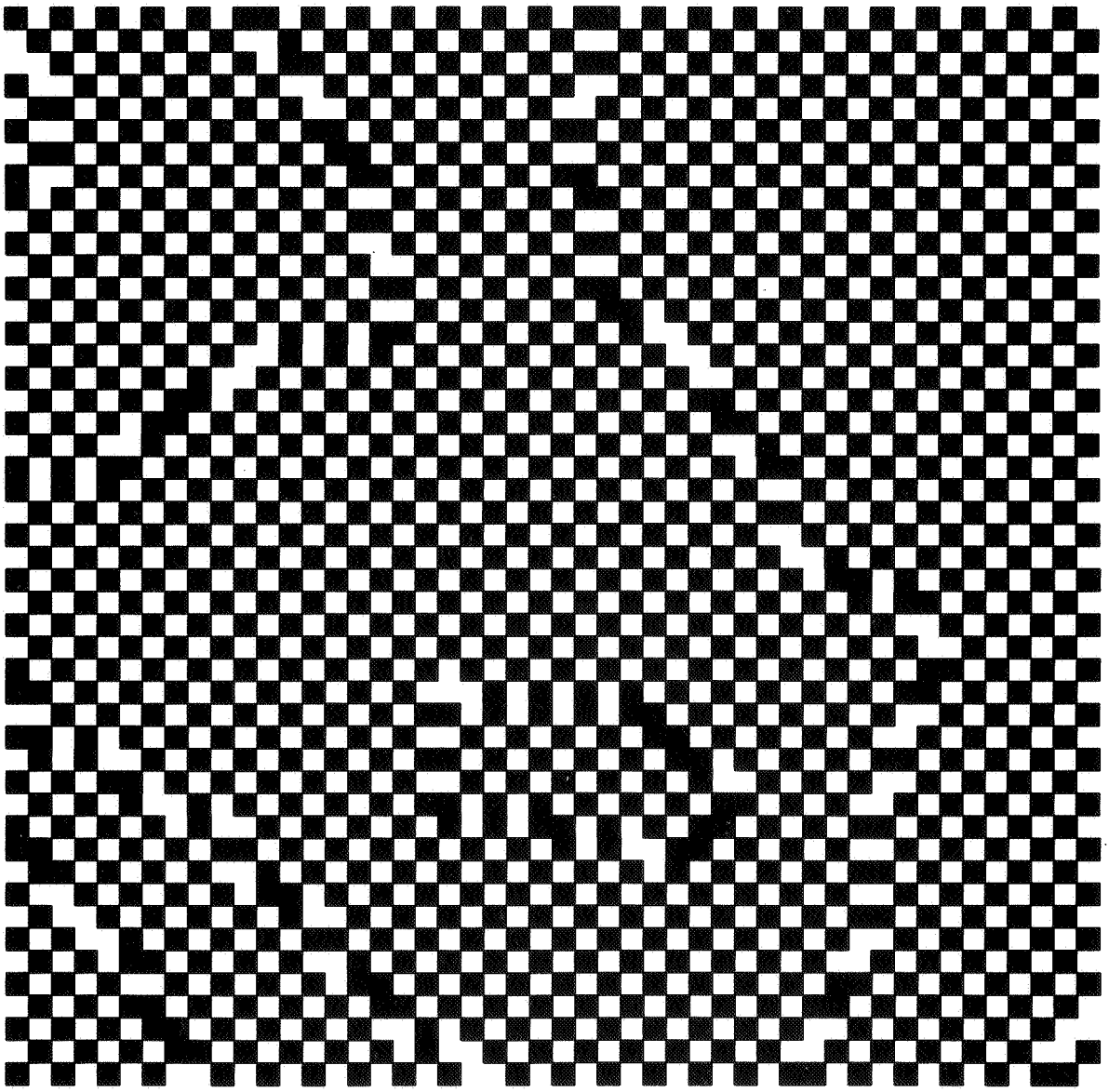
The Digiset simulator in ALGOL 60

APPENDIX B

1. Syntax of the Digiset machine code
2. Syntax of the shorthand code
3. Syntax of the Digiset code

APPENDIX C

Sample programs



"Crystal Structure"
after L. Geurts and L. Meertens,
produced on the Digiset.

0. INTRODUCTION

The underlying report contains a description of an ALGOL 60 program that simulates most of the behaviour of a Digiset-40 T1 electronic photo-typesetter.

The writing of the program is part of a careful study of the possibilities of the photo-typesetter for on-line computer applications (computer graphics). The program in its present state is used to execute programs written or generated in Digiset code. This allows us to continue our study, without using the real apparatus, or alternatively, using it with better results, in the following ways:

- The design of standard Digiset programs (e.g. characters or microprograms) for on-line applications, like the production of line drawings, grayscale pictures and other computer graphics.
- To provide a testing facility for software that generates Digiset programs.
- The program checks the syntactic (and some of the semantic) correctness of Digiset programs. This is necessary because the Digiset itself does not provide any error detection whatsoever.

The report is divided in three parts:

In the first part (chapter 1) a short description of the Digiset-functions and internal organisation is given. The description is meant to allow further reading of the report, as well as the use of the Digiset and the Digiset simulator, without knowledge of the rather technical manuals.

In the second part (chapter 2) the simulator is discussed. First the set of input/output specifications is given. This information is sufficient for any programmer who wants to run an exercise on the simulator. The section also describes how the output is to be interpreted in order to get an idea about the real Digiset performance. The next section gives a sketch of the structure of the ALGOL 60 program (which is added as appendix A). Discussion of the simulator follows in three levels of detail. At each level there exists an error detection mechanism that gets full attention.

In the third part (chapter 3), a selection of exercises is given as an illustration of the possibilities of both simulator and simulated

machine. The collection of exercises will extend far beyond the contents of this report since every step in the development of the computer graphics project will be accompanied, and partly described by such a set of exercises.

The reader should be aware of the following typographic conventions throughout the report: Notions that are used to describe the Digiset are underlined at first occurence. Usually some kind of definition is then contained in the context (e.g. typefount). Names of Digiset instructions are denoted in capital letters (e.g. KUR, SEQ, UAT), and are the same as those from the official manuals (german mnemonics!). Names of identifiers from the ALGOL 60 program are denoted in a different type (e.g. *inpcore*, *first white element*).

1. THE DIGISET

1.1. Survey of the functioning of the Digiset

The Digiset photo-typesetter generates hard-copy representation of graphical information by displaying drawings on the screen of a cathode ray tube, which are recorded on photographic paper of film. For a general overview of the Digiset hardware, see fig. 1.1.1.

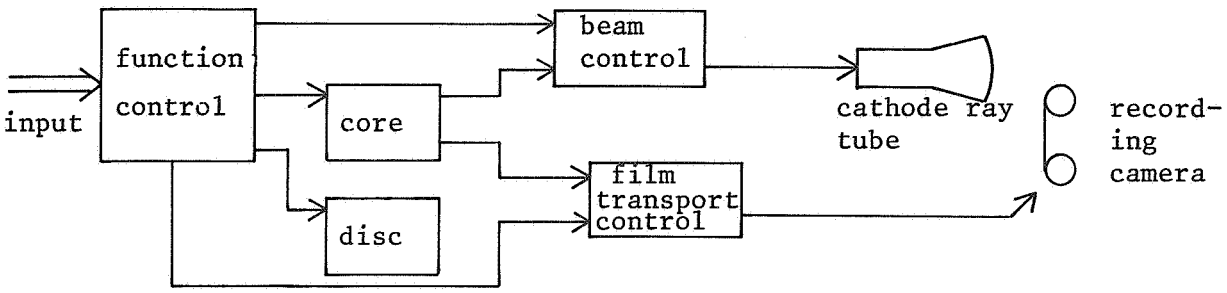


fig. 1.1.1. Flow of control in the Digiset.

The dimensions of the area of photographic material that can be exposed without transporting it, the usable recording area, are shown in fig. 1.1.2. In the same figure, the screen coordinates are shown, which

will be used in this report. The unit of length, is the point:
1 point = 0.376065 mm.

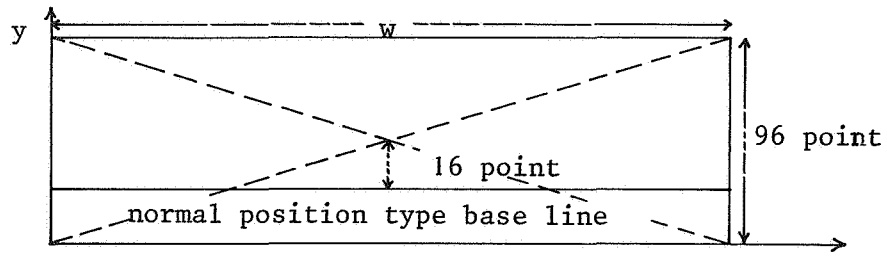


fig. 1.1.2. Usable recording area.

The width w of the recording area depends on the photographic material used: w ≤ 310 mm.

The recording takes place with a camera which can rewind the photographic material it contains. This offers the possibility of repeated exposure of the same piece of film, alternated with exposure of different pieces. The recording of drawings which are larger then the usable recording area is made quite easy this way.

The Digiset is equipped with a core store of 32 to 64 segments of 256 words. A word consists of 3 bytes of 8 bits each. The disc store of the Digiset is divided into 3247 addressable sectors, containing 744 non addressable words.

Drawings are composed from elementary characters like lines, text symbols etc. which are stored in the core- or disc store. Only characters stored in core can immediately be displayed, characters stored on the disc have to be transferred to core first. The input of the core address of a character (a character call) causes this character to be displayed. Positioning and transformation of characters are brought about by instructions which enter the Digiset through its input point. The Digiset permits the use of another programming unit for describing and drawing pictures, the microprograms. These are sequences of character calls and instructions which are stored and can be called like characters.

1.2. Storage of characters and microprograms

In order to be able to call characters and microprograms from core with a single one byte instruction, indirect addressing is necessary.

Therefore, the core store is divided into segments which can be addressed as a whole by a segment instruction which remains valid until replaced by a different one. Words in a segment thus selected, can be addressed by a one byte word address, the primary address. To make use of this mechanism, microprograms and characters (which must be defined on the same recording grid, see 1.3.) are grouped together in typefounts of atmost 251 elements. These elements are stored in the image area of core, which may consist of several parts of different segments. The data of single characters or microprograms have to be loaded in contiguous parts of the image area however. The segment- and word address of the beginning of such an area are each stored in one word (the secondary address word) of a selected core segment, the address area of the typefount. A character thus stored, can be addressed by the word address of its secondary address, its so called primary address, as long as the appropriate segment instruction is valid.

The three bytes of the secondary address are used as follows (see table 1.2.1.)

byte 1	byte 2	byte 3	
word address	rel.seg.address	0,1 or 32	character
"	"	2	microprogram

table 1.2.1. The secondary address word.

The first byte of the secondary address word contains the word address of the beginning of the character data in the image area.

The second byte contains the relative segment address of the character. This address has to be added to the segment address of the address area to find the segment address of the beginning of the character (or microprogram) data. The relative segment address has to be positive or zero. In the last case the first byte of the secondary address word has to be greater than 62, i.e. if the image area lies in the same segment as the address area, it has to lie above the word with address 62 (and above the address area).

The third byte of the secondary address word has value 2, if the addressed element is a microprogram. In the other case, it has the values

0,1 or 32 (see 1.3).
A special role is reserved for the secondary address word with primary address 62. This word contains the typeface constant which specifies among other things the dimensions of the recording grid on which the character is displayed (see 1.3).

The disc can only be loaded with complete typefounts i.e. a typeface constant followed by characters and/or microprograms. If elements of these typefounts are needed for typesetting, they have to be transported to core first. It is possible to transport typefounts as a whole as well as single characters or microprograms. While the disc is loaded with typefounts, the first core segment serves as an input buffer.

1.3. The coding of characters

Characters are defined on a square, in which a recording grid composed of rectangular cells is inscribed (see fig. 1.3.1a.)

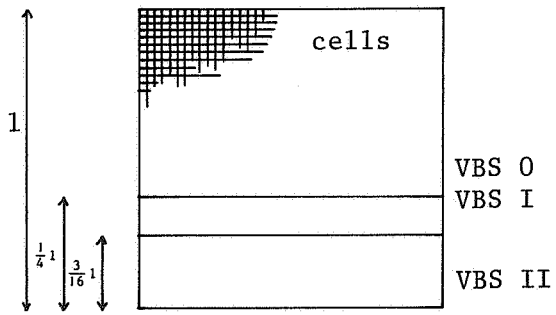


fig. 1.3.1a. Recording grid.

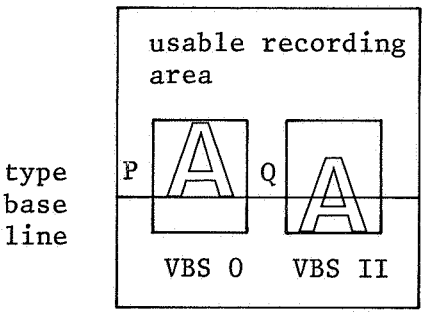


fig. 1.3.1b. Character position.

In this square, three different character base lines (denoted by VBSII, I and 0) are defined which lie at the base of the square, at 3/16 1 and 1/4 1 above this base respectively, if 1 is the length of an edge of the square. If a character is displayed it is positioned in such a way, that the VBS 0 line coincides with the current position of the type base line on the screen. This type base line intersects the current beam position and runs parallel to the x-axis of the screen coordinate system.

When a character is displayed, the current beam position moves from point P to point Q in fig. 1.3.16. The amount by which a character can extend below the type base line is determined by its character base line (see fig. 1.3.16.).

The cells, and hence the grids are classified according to their size in three types, the B-, C- and D type (see table 1.3.1.).

	length in x direction	length in y direction
B type	4/50 pt	4/120
C type	4/50 pt	4/60
D type	4/25 pt	4/60

table 1.3.1. Cell types and cell sizes.

According to the length of an edge of the square, recording grids are distinguished in grid types I, II, III, IV, and V.
The size of these grids is given in table 1.3.2. below.

	B		C		D		cell type
grid type	x	y	x	y	x	y	
I	50	120	50	60	25	60	number of cells in x- and y direction
II	100	240	100	120	50	120	
III	200	480	200	240	100	240	
IV	400	960	400	480	200	480	
V	800	1920	800	960	400	960	

table 1.3.2. Size of recording grids.

The distance (expressed in numbers of cells) from the base of the square to the character base lines, for different cell- and grid types is given in table 1.3.3. below.

	B					C, D					cell type
character base line	I	II	III	IV	V	I	II	III	IV	V	grid type
VBS II	0	0	0	0	0	0	0	0	0	0	distance of character base line above base of recording grid in number of cells
VBS I	22	45	90	180	360	11	22	45	90	180	
VBS 0	30	60	120	240	480	15	30	60	120	240	

Table 1.3.3. Position of character base lines.

Which character line is used in the definition of a character by its character data, is specified in the third byte of its secondary address word, see table 1.3.4.

decimal value of third byte of secondary address word:	1	0	32
character base line	VBS 0	VBS I	VBS II

Table 1.3.4. Secondary address word and character base line.

Remark: The data of tables 1.3.1. through 1.3.3. are only valid when the GROa instruction has been given, otherwise they must be multiplied by the appropriate enlargement factors (see GRO and DAN instructions: 1.4.).

Although characters are defined on a rectangular grid, they can be displayed as italics. In this case, the recording grid is deformed to a parallelogram, as shown in fig. 1.3.2.

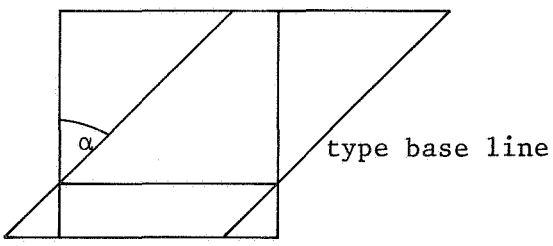


fig. 1.3.2. Italic characters.

Their exist three italic positions, corresponding with values of α (fig. 1.3.2.) of 22.5°, 25° and 27.5°.

The cell type, grid type and italicity of a character are specified in the typeface constant. This constant consists of one word, the second and third byte of which are zero. The value of the first byte is given by the following formula:

$$\text{byte 1 typeface constant} = N0 + 4N1 + N2$$

in which N0 specifies the cell type, N1 the grid type and N2 the italicity of the character. N0, N1 and N2 can have the values listed in table 1.3.5.

N0	cell type	N1	grid type	N2	italicity
0	B	0	I	0	0°
32	C	1	II	1	22.5°
64	D	2	III	2	25°
		3	IV	3	27.5°
		4	V		

Table 1.3.5. Constants, defining the typeface constant.

We only give a short description of the Digiset machine code for characters, since as input to the simulator, a much simpler, but just as powerful code can be used (see 2.1.), that will be translated in machine code before the simulation starts. This translated machine code can be used as input to the real Digiset.

Characters are coded as a sequence consisting of image lines, empty image line commands (LBL commands) and image line repetition commands (BLW commands). An image line is a column of the recording grid, in which for each cell it is defined whether it is black or white. An empty image line contains only white cells. The specification of the white and black elements in a non-empty image line consists of a sequence of alternating white and black elements. These elements are integer values, which denote the length of the pieces of the same colour in cell units (see fig. 1.3.3.). An image line has to start with a white element and to end with a black

element. An image line is displayed by alternately recording white- and black elements, starting from the character base line. However, the recorded length of the first white element, is one less than its coded length. This makes it possible to define characters that begin at the character base line, by giving their first white elements unit length.

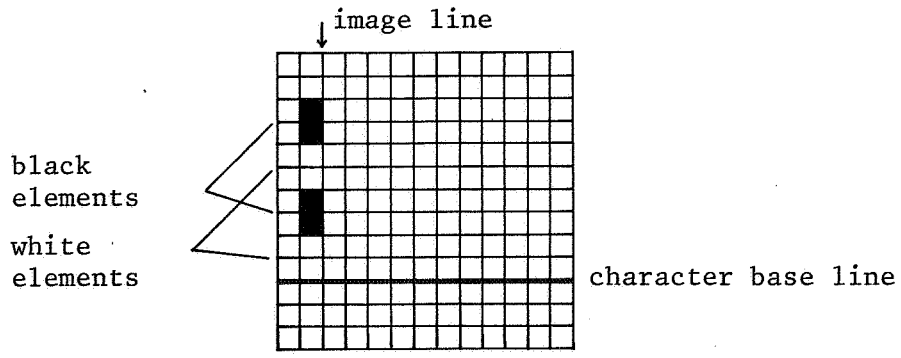


Fig. 1.3.3. Character coding.

The height of line elements can be specified in one byte if it is smaller than 2^7, otherwise two bytes can be used. In this coding one bit (of the first byte) is used to indicate whether the line element is the last one of an image line or not. In case of two byte coding a second bit of the first byte indicates whether both available bytes are really used, or only the first one. This last possibility can be used for line elements shorter than 2^6 cell units enclosed by lines longer than 2^7-1 units. In table 1.3.6. the modes in which line elements of different length can be coded, are shown.

length of element in cell units	element can be coded in one byte	element can be coded in two bytes	
		both bytes used	second byte zero
$1 > 127$		yes	
$63 < 1 \leq 127$	yes	yes	
$1 \leq 63$	yes		yes

Table 1.3.6. Coding modes for line elements.

The LBL- and BLW commands have parameter parts in which the number of empty lines that have to be recorded, respectively the number of times the image line following the command has to be repeated, is specified. This parameter part can have values between 0 and 255. In addition, one bit in these commands is used to indicate the mode in which the image lines following the command are coded, i.e. to indicate, whether they are coded in one- or two-byte mode. In order to change the coding mode without introducing empty lines or repetitions, the LBL- or BLW commands with zero-valued parameter parts are used.

The ambiguity in the coding possibilities for line elements gives rise to the problem of finding the shortest possible coding for a character. If for instance, a character contains both lines with some long elements, as well as lines with only medium elements, everytime these latter lines occur, the question arises if the coding mode has to be changed in order to minimize the memory area occupied by the character.

The answer to this question depends on:

1) The number of medium lines not separated by long lines.
2) The position of LBL-and BLW commands (if present) relative to the position of the medium lines in the coding sequence.
3) The position of the medium lines with respect to the beginning or end of the coding sequence.

The translation of the character code used as input to the simulator in the representation of machine code used during simulation, is such that the shortest possible code is generated.

1.4. The instruction set of the Digiset

The term instruction is used in this report, to denote all commands that control the functioning of the Digiset, with the exlusion of character data. Instructions enter the Digiset through its input point, or they are elements of microprograms. In the first case they are executed on-line, in the second, when the microprogram is activated. Instructions can be devided in direct instructions (consisting of one byte) and indirect instructions (of up to three bytes).

1.4.1. Direct instructions

The direct instructions are: NUL, KEN, UAK, ZWR and UAT.

NUL: This instruction does not cause any action of the Digiset.

KEN: The KEN instruction causes the three bytes following it to be ignored. These bytes may not contain the instruction sequences UAK SPE or KEN KEN KEN however. In the simulator input, KEN may not occur in character- or microporgram data. The KEN instruction is only of use, when a magnetic tape unit provides the input, a situation not discussed here.

UAK: This instruction must precede every indirect instruction.

ZWR: The ZWR instruction is a spacing instruction. It causes the recording position to be shifted in the + x direction by an amount that must have been devised with the indirect instruction ZWQ.

UAT: This instruction serves as an end marker for character- and microprogram data, and for instruction sequences controlling data transfer from disc to core. It has always to be given twice in succession.

All other one byte commands are interpreted as primary addresses and cause the character or microprogram thus addressed, to be displayed or executed. The existence of this 5 direct instructions explains, why a typefount can have at most 251 elements although a core segment has room for 256 secondary addresses. The five values 0, 28, 62, 252 and 255 denote direct instructions and hence can not be used as primary addresses, without causing ambiguities.

1.4.2. Indirect instructions

The indirect instructions which are preceded by the UAK instruction can be devided in the following categories:

- Instructions which control data transfer in the Digiset and between the Digiset and the outside world.
- Layout instructions
- The START and STOP instructions.

1.4.2.1. Data transfer instructions

The data transfer instructions are: UBL, SEG, SPE, PLA , PLE and TRE.

UBL: The UBL instruction causes the Digiset to ignore all input until the

instruction sequence UAK START is encountered in the input. When the simulator is used, UBL may not occur in microprogram- or character data in the input stream.

SEG: By means of the SEG instruction, a core segment is addressed. This address is given in the quantification byte following the instruction, which can have values between 0 and 64. All word addresses that occur in the input or in microprograms refer to the segment, last addressed with SEG. There is, however, one exception to this rule. If a segment instruction S0 is followed by a different segment instruction S1, which itself precedes a microprogram call, which is not separated from S1 by a character call, it is only used for the selection of the microprogram. Inside the microprogram S0 is valid again, until other SEG instructions are given there. After the execution of the microprogram, S0 becomes active again, regardless of the SEG instruction given within the microprogram. If one wishes S1 to remain active during and after the microprogram call it has to be given twice in succession.

SPE: This instruction is used in two ways:
In the first place, it has to precede the indirect instructions PLA, PLE and TRE (see below). In the second place, it causes the input in core of the character, microporgram or type-face constant (with primary address 62) that follows the SPE instruction.
The instruction sequences are in these cases:

<UAK> <SPE> <62> <type-face constant> <UAT> <UAT> seq. 1.4.1.

<UAK> <SPE> <primary address> <secondary address>
<character data> <UAT> <UAT> seq. 1.4.2a.

<UAK> <SPE> <primary address> <secondary address>
<microprogram data> <UAT> <UAT> seq. 1.4.2b.

PLA: These instructions control the data transfer from the input point to the disc and from the disc to the core store respectively.

TRE: If we denote seq. 1.4.1. by TC and seq. 1.4.2a,b. by CM, a typefount is defined as:

<typefount>:: = <TC> <alphabet>
<alphabet> :: = <CM> | <alphabet> <CM>

The instruction sequence for the transfer of a typefount from the input point to the disc is:

<UAK> <SPE> <UAK> <PLE> <disc address> <UAT> <UAT>
<typefount> <UAK> <SPE> <UAK> <TRE> <UAT> <UAT> seq. 1.4.3.

This is the only way in which the instructions PLE and TRE can be used. It should be noticed, that during the input of data to the disc, segment 0 of core store is used as a buffer. Thus a PLE instruction causes the loss of the data stored there.

The transfer of a typefount from disc to core is brought about by the following instruction sequence.

<UAK> <SPE> <UAK> <PLA> <disc address> <UAT> <UAT> seq. 1.4.4.

The disc address that always occupies two bytes specifies the number of the disc sector where the typefount begins. Typefounts are always stored without gaps on the disc. For the format of the disc address see fig. 1.4.1.

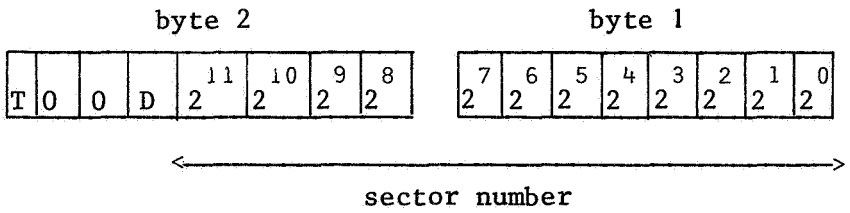


fig. 1.4.1. The disc address.

The bit D serves to distinguish the two discs that can be simultanuously used. In the simulator only one disc is simulated and bit D is always set zero. In seq. 1.4.3. bit T is always set to zero. If bit T is set in the disc address of seq. 1.4.4., this sequence causes the transfer of only the disc addresses of the elements of the typefount to the selected address area in core. Every time such an element is called up for typesetting it is transferred from the disc to the segment of core, that lies immediately above the segment that contains

the address area with disc addresses. After this transfer it is displayed or executed. Data transfer from disc to core is the only data transfer possible during the execution of microprograms.

1.4.2.2. Layout instructions

The layout instructions that are part of the Digiset machine code are: VERA, VERO, VERU, ZWQ, RUCK, TAB, ZEV, KUR1, KUR2, KUR3, NOR, DAN, GROa through GROm, WID, SCHN and DREHO. The last two instructions are not incorporated in the Digiset simulator and will not be discussed here. Some of the instructions can be followed by at most two quantification bytes. In this case, the first byte always contains the least significant bits.

VERO: The instructions VERO and VERU cause the type base line to move up
VERU: and down respectively, by an amount specified in the next two quantification bytes. The unit in which this displacement is measured is 1/32 point. The VERA instruction cuases the type base line to move back to its neutral position (see fig. 1.1.2.). For reasons to be explained later, the VERA instruction has in the simulator the effect, of moving the type base line to the centre of the usable recording area.

ZWQ: The instruction ZWQ fixes the amount of spacing in the + x direction caused by the direct instruction ZWR. It is followed by two quantification bytes, which specify the displacement in units of 1/50 point.

RUCK: The RUCK instruction cuases displacement in the - x direction of the recording beam. It has two quantification bytes, which specify the displacement in units of 1/50 point.

TAB: By means of the TAB instruction a line perpendicular to the x - axis is defined in absolute screen coordinates. The two quantification bytes specify the constant C in the line equation x = C, in units of 1/50 point.

ZEV: The ZEV and ZER instructions displace the type base line in the same
ZER: way and have the same quantification bytes as the instructions VERO respectively VERU. In addition, the recording beam is moved to the point of intersection of the displaced type base line and the line defined by TAB.

KUR1: The instructions KUR1, KUR2 and KUR3 cause the image lines to be

KUR2: rotated clockwise round the points of intersection of these lines
KUR3: with the type base line. The rotation takes place over angles of 27.5°, 25° and 22.5° respectively. A KUR instruction remains valid until replaced by a different one, or a NOR instruction. When this last instruction is given, the italicity of the displayed character becomes the one, specified by the typeface constant.

GRO: These instructions cause the elementary rectangles of the recording
DAN: grid to be enlarged by a definite factor. When a GRO instruction is used enlargement takes place both in the x- and y direction. In case of a DAN instruction, enlargement in the x direction is defined, while the enlargement in the y direction remains the one, defined by the GRO instruction last given. The enlargement factor is fixed by selecting a particular GRO instruction, or by the quantification byte of the DAN instruction, according to table 1.47.

GRO instruction	DAN quantification	enlargement factor
GROa	8	1.
GROb	9	1.125
GROc	10	1.25
GROd	11	1.275
GROe	12	1.5
GROf	14	1.75
GROg	16	2
GROh	18	2.25
GROi	20	2.50
GROj	22	2.75
GROk	24	3
GROl	28	3.5
GROm	32	4

Table 1.4.7. Enlargement factors.

An enlargement remains valid until a new GRO or DAN instruction occurs. When a GRO instruction is given, the typeface constant stored in the segment addressed at this time, is evaluated. This typeface constant

will be used for the display of characters called after the GRO instruction. From, this it follows that, before the first character call a GRO instruction has to be given and before the first GRO instruction, a SEG instruction.

WID: The WID instruction causes a character call or microprogram call following it, to be repeated the number of times specified in the two quantification bytes of the instruction.
Remark: The character is displayed (or the microprogram executed) once more, than the number of times it is repeated.
A microprogram that contains a WID instruction may not be repeated itself by a WID instruction.

1.4.2.3. Start and stop instructions.

START: A Digiset program has to begin with the instruction sequence: UAK
STOP : START and ends with UAK STOP. In addition, the START instruction is used to terminate the effect of the UBL instruction (see 1.4.2.1.).

2. THE SIMULATOR

2.1. Input to the simulator

The input to the simulator has the following form:

<parameter setting> <Digiset code representation>

The first part of this sequence will be discussed in section 2.3. after the description of the simulator output.
A sequence of Digiset instructions and parameters is recognized by the simulator if:

- The represenation of a byte is separated from the representation of another one by a semicolon.
- Instructions are represented by their names (see 1.4.).
- Microprogram data are enclosed between square brackets.
- Parameters and addresses are represented by decimal numbers.

Examples:

- UAK; START; UAK; DAN; 8; UAK. ZWQ; 10; 0; ...
- UAK; SPE; UAK; PLA; 21; 0; UAT; UAT; 63; 68; 112; ...
- UAK; UBL; byte; ..... byte; UAK; START; ....

The instruction sequences of microprograms are coded in the same way.

Character data, as defined in seq. 1.4.2a., are coded as follows:

<character data>:: = <character body>;

<character body>:: = <char element> | <char element> <char body>

<char element>:: = (repeated empty line>) | (<repeated line>)

<repeated empty line>:: = <number of empty lines>

<repeated line>:: = <number of lines>, <first white element>,
<black white list>, <last black element>

<black white list>:: = <black element>, <white element> |
<black element>, <white element>, <black white list>

<first white element>:: = <positive integer>

<last black element>:: = <positive integer>

<white element>:: = <positive integer>

<black element>:: = <positive integer>

<number of lines:: = <positive integer>

<number of empty lines>:: = <non zero integer>

<number of lines> (respectively <number of empty lines>) denotes the number of times the line (respectively the empty line) has to be displayed in succession. If <number of empty lines> is positive the empty lines are displayed in the recording direction, otherwise in the opposite direction. Thus an empty line command with a negative parameter gives backspacing within a character. The value of <first white element> is the length in cell units plus one of the first white element of the image line. All other white or black elements denote exactly the length of the corresponding parts of an image line in cell units.

Example: (3) (4,2,1);

This character is built up out of three empty image lines, followed by 4 image lines each consisting of a white element of length one followed by a black element of length one. Characters coded this way, are translated by the simulator in a representation of Digiset machine code (see 1.3.).

This translation takes place in such a way, that the resulting code is the shortest possible one. Hence there is no need to consider the "shortest code problem" (see 1.3.) while using the simulator.

Characters and microprograms from a single typefount that is loaded without being interrupted by the input of elements from a different typefount, can be coded in shorthand code, without specifying secondary word addresses and relative segment addresses. These two addresses must then be replaced by the single pseudo address 256. In this case, the data are loaded consecutively, starting at word address 0 of the segment, following the address area of the typefount. The secondary word addresses and segment addresses are then calculated by the simulator.

2.2. Simulator output

The simulator can produce its output on the plotter, on the lineprinter and on paper tape. When the lineprinter is used, every cell of a C type recording grid corresponds to one print position on the lineprinter paper. This situation fixes the scale of lineprinter drawings. The C type grid has been chosen, because its cells are almost square. It is not possible to display characters coded on B- or D type grids with the lineprinter. This would not make sense anyhow, for it would change the scale of characters, while the scale of layout instructions would remain the same. The height of the part of the usable recording area that is simulated on the lineprinter is 144 C type cell units, while the height of the real recording area is 1440 units (see fig. 2.2.1.). However, the length can be chosen freely. (see 2.3).

The instructions GRO, KUR, NOR and DAN cannot be used in lineprinter drawings. In addition, italic characters are displayed upright.

When using the plotter, every C type cell is displayed as a rectangle of 12 x 1mm, B- and D type cells can be used too, and are displayed on the same scale. (B type cell 12 x 0.5mm, D type 2.4 x 1mm). Black elements are displayed black, by shading them.

The maximal height of a plotter drawing is 275 C type cell units, the maximal length 1250 units. The length of the real usable recording area is 9900 units (see fig. 2.2.1.).

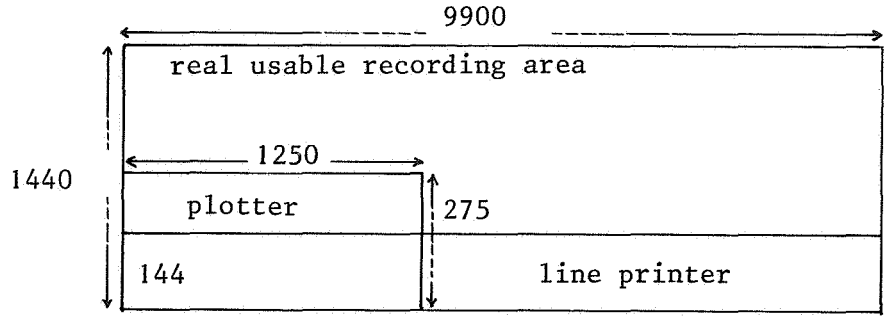


Fig. 2.2.1. Simulated recording area.
Units: C type cell units.

When using the plotter, every instruction described in this report, can be used. The range of the parameters of the positioning instructions that can be simulated is much smaller than the range for the real Digiset, because of the smaller recording areas. For the same reason the effect of the VERA instruction is changed into positioning the type base line as a line, that lies halfway between the base and the top of the simulated recording area.

Apart from drawings, the output contains a numbered listing of all input elements. This listing is used for error identification (see 2.4.).

Furthermore, the simulator can produce a paper tape with the Digiset machine code that corresponds to the short hand code used as input. In this way, tapes that can directly be used as input to the Digiset are produced.

2.3. Specification of simulator properties

The input to the simulator has to be preceded by some simulation parameters. For most simulations it is not necessary to supply the full capacity of the real machine. Program time and space can be saved by specifying the smallest possible configuration. Output time can be reduced by directing output to the lineprinter instead of to the plotter for small exercises. The format of the specification is:

$<N_1>$; $<N_2>$; $<N_3>$; $<N_4>$; $<N_5>$; $<N_6>$;
in which $N_1,\ldots,N_5$ are integers.

N_1 specifies the number of core segments ($1 \leq N_1 \leq 64$), N_2 the number of disc sectors ($1 \leq N_2 \leq 3274$). N_3 is an upper bound for the number of image lines

that can occur in a character ($1 \leq N_3 \leq 15000$) and N_4 is an upper bound for the number of bytes that the input would occupy, if coded in Digiset machine code ($4 < N_4$).
If $N_5 = 1$, the output is plotted, else it is displayed on the lineprinter. In the latter case, the simulated recording area has a length of N_5 image lines ($1 \leq N_5 \leq 15000$).
The syntax of $<N_6>$ is:

$<N_6>$:: = p; | <empty>

When $<N_6> = p$, a papertape with Digiset machine code is produced.

2.4. Errors

If the input to the simulator contains syntactic errors or semantic errors that are discovered during simulation (an attempt to draw outside the recording area, for example), an error message is given. The same happens if the simulation parameter setting contains errors.

The error detecting mechanism has the property that input errors are discovered before they can cause the simulation program to be stopped by the machine on which it runs. In this way it is possible to localize errors, without knowledge of the functioning of the simulation program.

Errors can be classified in three types:

1) Parameter setting errors.
2) Errors discovered during translation of the input in Digiset machine code.
3) Errors discovered during execution of the translated Digiset program.

All errors of type 1 or 2 are discovered in one scan, after which they are listed, and the simulation stops. If a type 3 error occurs, an error message is given, and the output which has already been generated is displayed, after which the program stops.

An error message for type 2 or type 3 errors consists of:
- A description of the error.
- The number of the input element that causes the error. This number

corresponds to the number of the element in the listing of the input.
- If the error is caused by a microprogram instruction, the number of this instruction, relative to the beginning of the microprogram, is also given.
- If the error is caused by character data, the number of the image line that contains the error, is also given.

Examples:

1) input listing: 31 (10) (20,0,1) (3,1,10)
error message: zero valued line element
number input element 31
line number 2

2) input listing: 11 PLA;
12 10;
13 0;
14 UAK;
error message: PLA not followed by UAT; UAT;
number input element 14

If a parameter setting error occurs, the error message consists only of a description of the error.

There is one important semantic error which does not always cause an error message. If a core area is addressed that has not been loaded with correct character or microprogram data, it is nevertheless possible that the data present can be interpreted as microprogram or character data. Only if this is not the case, an error message will be given.

2.5. Description of the simulator

The simulation program will be discussed with the aid of fig. 2.5.1. In this figure, the block structure of the program is shown by rectangles. If a block is a procedure body, this fact is indicated by the text in the block. In fig. 2.5.1. from left to right three levels of detail can be seen. We will describe the program by discussing these levels in the same order.

2.5.1. Level 1

The simulator consists of the simulation parameter setting followed by the simulation program proper. In the parameter setting part the output mode is specified, as well as upper bounds for some array indices. This part contains the procedure *parameter error*, which gives error messages in case of errors in the specification parameters of the input.

The block that contains the real simulator, begins with the declaration of the integer arrays *dtape, core, disc* and the boolean array *lineprinter output*. Array *dtape* will be filled with the input in a representation of Digiset machine code. This representation, which is a result of translation of the input will be used throughout the simulation. It is defined as follows:
Every instruction or character byte is coded as a decimal number of at most eight digits. Each digit corresponds to one bit of machine code and can have the values zero or one. Instruction quantification bytes and addresses, which occur in the simulator input as decimal numbers with values between 0 and 255, are left unchanged, however. The arrays *disc* and *core* simulate the disc and core memory of the Digiset. The boolean array *lineprinter output* represents the usable recording area, in case the output has to be displayed on the lineprinter. Every array element corresponds to one cell of a C type grid. It gets the value "true" when a cell is found to be part of the output drawing, otherwise the value "false". At the end of the simulation for every element with value "true", a $ sign is printed.

The procedures which follow the array declarations can be classified as auxiliary procedures and procedures that bear the names of Digiset instructions and simulate them.

The machine code table links the names of the instructions with the representation of their machine code, while the enlargement factor table contains factors used in the GRO and DAN instructions.

The simulation starts with the translation of the input into machine code and the storage of this code in the arra *dtape*. Thereafter this translated input is processed by the procedure interpret (see 2.5.2.1.).

2.5.2. Level 2

2.5.2.1. Auxiliary procedures

The integer procedures *read dtape*, *read core* and *read disc* read one byte from *dtape*, *core* or *disc* respectively, every time they are called. They increase array indices in such a way, that the next time they are called they will read the next byte (unless indices are reinitialized between calls). The integer procedures *rdtod* (*bdisc*), *rdtoc* (*mic*) and *rdtodoc* (*brddisc*) read from *dtape* or *disc*, from *dtape* or *core* and from *dtape* or *disc* or *core* respectively, depending on the values of their boolean parameters. They are used in procedures that must have access to different data sources.

The procedure *error* (*string, mic, char, stop*) prints its string parameter when it is called. This string contains the description of the error which caused the procedure call. In addition, the number of the input element that caused the error is printed. If the boolean parameters *mic* and/or *char* are "true" the number of the microprogram instruction and/or the number of the image line which contains the error is/are also printed. If the parameter *stop* is "true", the simulation stops and the output sofar obtained is printed, otherwise the simulation continues.

The procedure *print relative* (*init linenumber, init printpos, deltax, deltay, linenumber max, print*) is used to store drawings in the array *line printer output*. If *init linenumber* and/or *init printpos* are "false" *deltax* and/or *deltay* are absolute screen coordinates, otherwise they are relative to the point of the drawing that is defined by the previous execution of *print relative*. If the boolean *print* has the value "true" the array element corresponding to the point of the drawing defined by the parameters of *print relative* gets the value "true". The variable *linenumber max* is used to check whether the print lies inside or outside the usable recording area. In the last case, an error message is given.

Procedure *plotrelative* (*initial value, deltax, deltay, ipen, kurn*) is analogous to *print relative*. However, it causes direct movements of the plotter pen instead of storing the output. The coordinates *deltax* and *deltay* are absolute or relative simultanuously depending on the value of the boolean *initial value*. The integer *ipen* is used to indicate, whether

the pen movement has to take place with pen up or down. The value of the integer *kurn* indicates which of the KUR or NOR instructions is valid at the time of the procedure call. If this is a KUR instruction, movements in the y direction occuring during character display are changed into sloped movements. In addition an error message is given if pen movements are towards points outside the recording area.

Procedure *interpret (instruction, bool rddisc, mic, plotter)* is used to interpret machine code bytes, not belonging to character data. It interprets *dtape* data, microprogram data from *core* and *disc* data during execution of the PLA instruction. The procedure is called with the decimal value of an instruction or address byte as value of its parameter *instruction*. If this value is the coding for UAK, another byte is read from *dtape, disc* or *core* depending on the values of the booleans *boolrddisc* and *mic*. This reading takes place with procedure *rdtodoc (boolrddisc, mic)*. This second byte will cause a call from the procedure that corresponds with the instruction that is coded by it. If the value of *instruction* does not designate the UAK instruction, a direct instruction is executed, or a character or microprogram is called. The boolean *plotter*, which has the value "true" when the output has to be plotted, is passed to the instruction simulating procedures.

2.5.2.2. Digiset instruction procedures

The procedures *seg, dan, gro, kur, nor, ken, ubl, init, zwr, ruck, vero, veru, vera, zer, zev, zwq* and *tab* simulate the effect of the Digiset instructions with the same name. This simulation is straightforward, and will not be discussed here.

The procedure *stop* stops the simulation. If line printer output has been generated, *stop* causes a print-out of the array *lineprinter output*.

The procedure *inpcore* stores typefount elements in *core*. It reads the data from *dtape* or *disc* with procedure *rdtod (boolrddisc)*. These data are stored in *core* by the local procedure *storec (x)*. Data are read and stored until the first UAT UAT instruction sequence not belonging to UAK PLA byte byte UAT UAT is recognized. During storage, the values of secondary address word bytes and typeface constant bytes are checked. If these bytes have non allowed values an error message is given. Procedure *inpcore* is used

for direct storage into *core*, during execution of the PLA instruction and when a typefount element is addressed that has to be brought in from *disc* by the single character transfer method.

The procedure *spe* first reads the next byte from *dtape, disc* (during execution of *pla*) or *core* (during execution of a microprogram). If this byte is not an UAK byte, procedure *inpcore* is called, otherwise the next instruction byte is read, to determine whether procedure *pla, ple* or *tre* has to be called. In case of non allowed data transfer instruction (for instance, PLE within a microprogram) an error message is printed.

The procedure *call typefount element* is used, whenever interpret encounters a byte, which is a core address. In this procedure, first it is determined whether the data of the addressed element still have to be brought in from *disc* (single character transfer) and whether the element is a character or a microprogram. In the last case, the data are read by procedure *interpret* which causes the execution of the microprogram. If the element is a character, the position of the character base line is calculated first. It is determined by the typeface constant and the third byte of the secondary address word. The y coordinate of the character base line is stored in the variable *y position*. Then the character data are read which cause calls of the procedures *lbl, blw* and *image line*. Procedure *lbl* causes the appropriate number of empty image lines to be displayed, while *blw* assigns values to *repeat* (the number of repetitions) and to *line repetition* ("true"). Procedure *image line* displays the line a number of times equal to the value of *repeat*. If the addressed data cannot be interpreted as character data clearly a wrong core area is addressed. In this case, the procedure *address error* prints an error message.

2.5.2.3. Input translation

In the block labelled input translation (fig. 2.5.1.), the input is translated into the representation of Digiset machine code, already discussed. This is done by procedure *code switch*. The input is read symbolwise by integer procedure *skip*, which delivers different values for different symbols read and which skips blanks and "new card" symbols. As a side effect, *skip* produces a listing of the input.

The procedure *read input element* scans input strings enclosed between

semicolons, by successive calls of *skip*. These strings can contain a quantification byte, an address, an instruction name or a character definition (which is recognized by an opening parenthesis). In case of a quantification byte or an address, the corresponding numerical value is stored in *dtape*. If an instruction name is read, its machine representation is stored in *dtape* as an integer consisting of only zeros and ones. A character definition is processed by the procedure *character* (see 2.5.3.), which delivers a sequence of bytes stored in *dtape*. As a side effect of *read input element*, all strings are enumerated. The procedure *listingnumber* prints the number of each input string on a new line, prior to its listing by *skip*. Input strings that cannot be recognized or processed in this way (because of the occurence of non allowed symbols, for instance), cause error messages. The translation continues, until the STOP instruction is recognized in the input.

2.5.3. Level 3

Procedure *pla*, which is called from *spe*, transfers a typefount from *disc* to *core*. First the disc address of the typefount is evaluated, and from its first address byte it is determined whether the transfer mode is single character or not. In the last case, procedure *interpret* scans the typefount data in *disc*, which results in the transfer of the typefount to *core*, by *spe* calls, causing *inpcore* calls. When the instruction sequence UAK SPE UAK TRE UAT UAT is recognized, the transfer stops. In the case of single character transfer, *pla* scans the typefount for UAK SPE instructions. If these instructions are not followed by UAK, they mark the beginning of a typefount element and are followed by a primary address. The sector address (for the first character) or the sector address -1 for every other character), is then copied in the secondary address word of *core* that has the primary address just found.

Procedure *ple* checkes typefount data, after which they are stored in *disc*, or an error message is given and the simulation stops. Data are stored bytewise by the local procedure *stored* (x), which is called with the machine code representation of the byte to be stored as value of x. Procedure *ple* starts with evaluating the disc address and checking whether the typefount begins with a correct typeface constant. Then procedure

element onto disc is called. In this procedure, it is checked (by procedure *check*) if the next instructions are UAK SPE and whether the following instructions are UAK TRE UAT UAT or a legitimate primary and secondary address. If the TRE sequence is found, the transfer is ended. The secondary address is used to assign value "true" or "false" to the boolean *microprogram*, which determines whether the following data are interpreted as microprogram or character data. Then procedure *uatuat (microprogram)*, which is local to *element into disc*, is called. If *microprogram* is "false", all subsequent data are store, until the first UAT UAT instructions are encountered, otherwise all data are stored until the first occurence of UAT UAT not being part of UAK PLA byte byte UAT UAT. Finally, *uatuat* calls *element into disc* again.

The procedure *image line (cellwidth, cellheight, shading, remainder, repeat, y position, plotter, line repetition, kurn)* is used to display an image line of a character. The parameters *cellwidth* and *cellheight* specify the size of the recording grid, taking into account the GRO and DAN instructions last given. The integers *shading* and *remainder* specify the way in which black elements are shaded. These parameters, which are only used when the output is plotted, are calculated using the typeface constant evaluated during the last call of *gro*. The integer *repeat* and the boolean *line repetition* obtain their values from procedure *blw*. The integer *kurn* which specifies the italicity of a character, gets its value from procedure *kurn* or from the typeface constant. The boolean *plotter* specifies, whether the output is on the plotter or the lineprinter. Finally, the real variable *y position*, specifies the position of the character base line. It is calculated by procedure *call typefount element*, which is global to *image line*. The data defining an image line are read element by element from core by the integer procedure *read line element*. These data are decoded into the length of successive line elements and are displayed by alternating calls of the procedures *black* and *white*. These procedures use the auxiliary procedures *plotrelative* or *printrelative* (see 2.5.1). If a line has to be repeated, the length of its elements are stored at the same time by procedure *store repeated line (x)* in the integer array *repeated line*. The integer parameter x passes the value to be stored. When the last black ele-

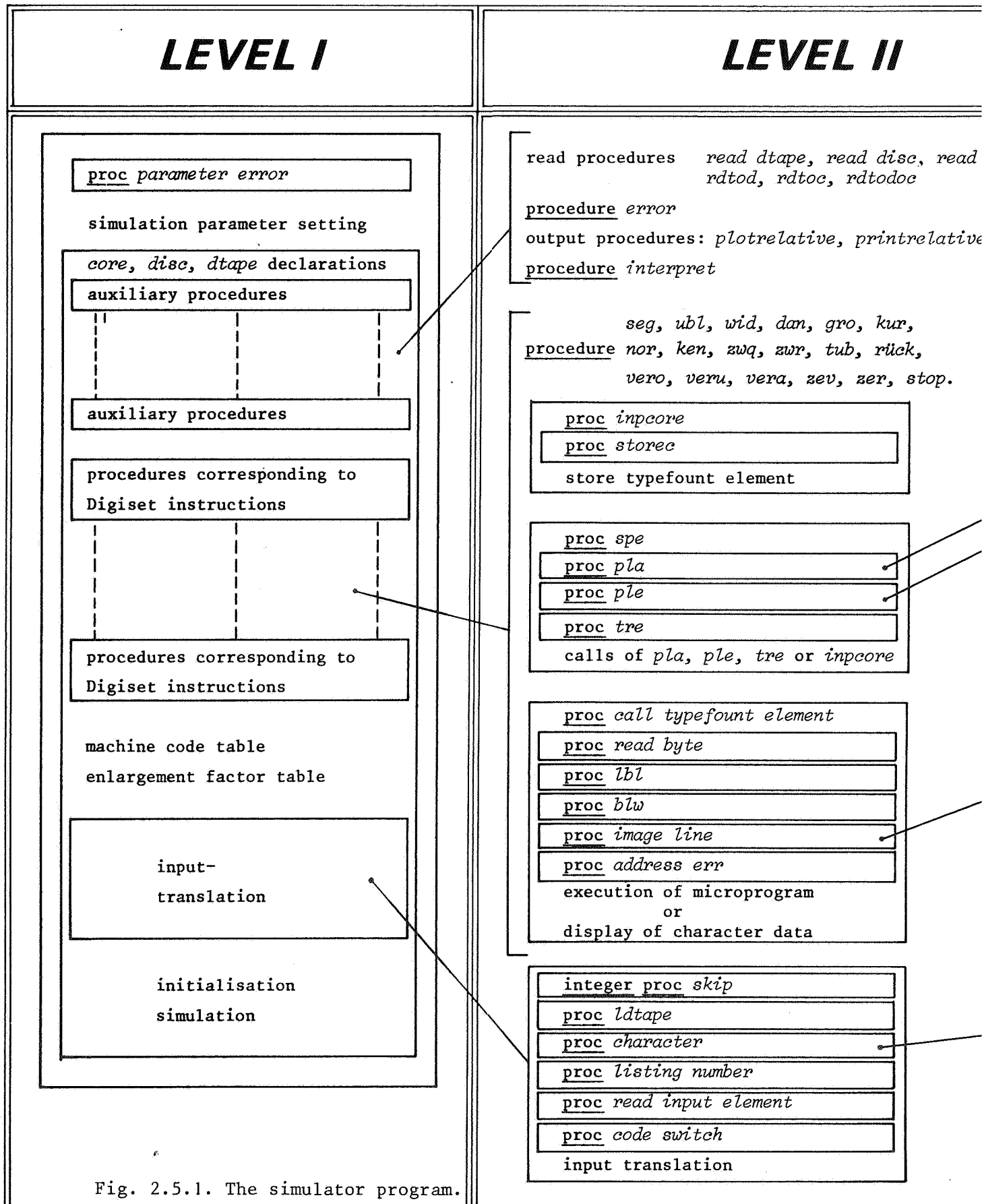
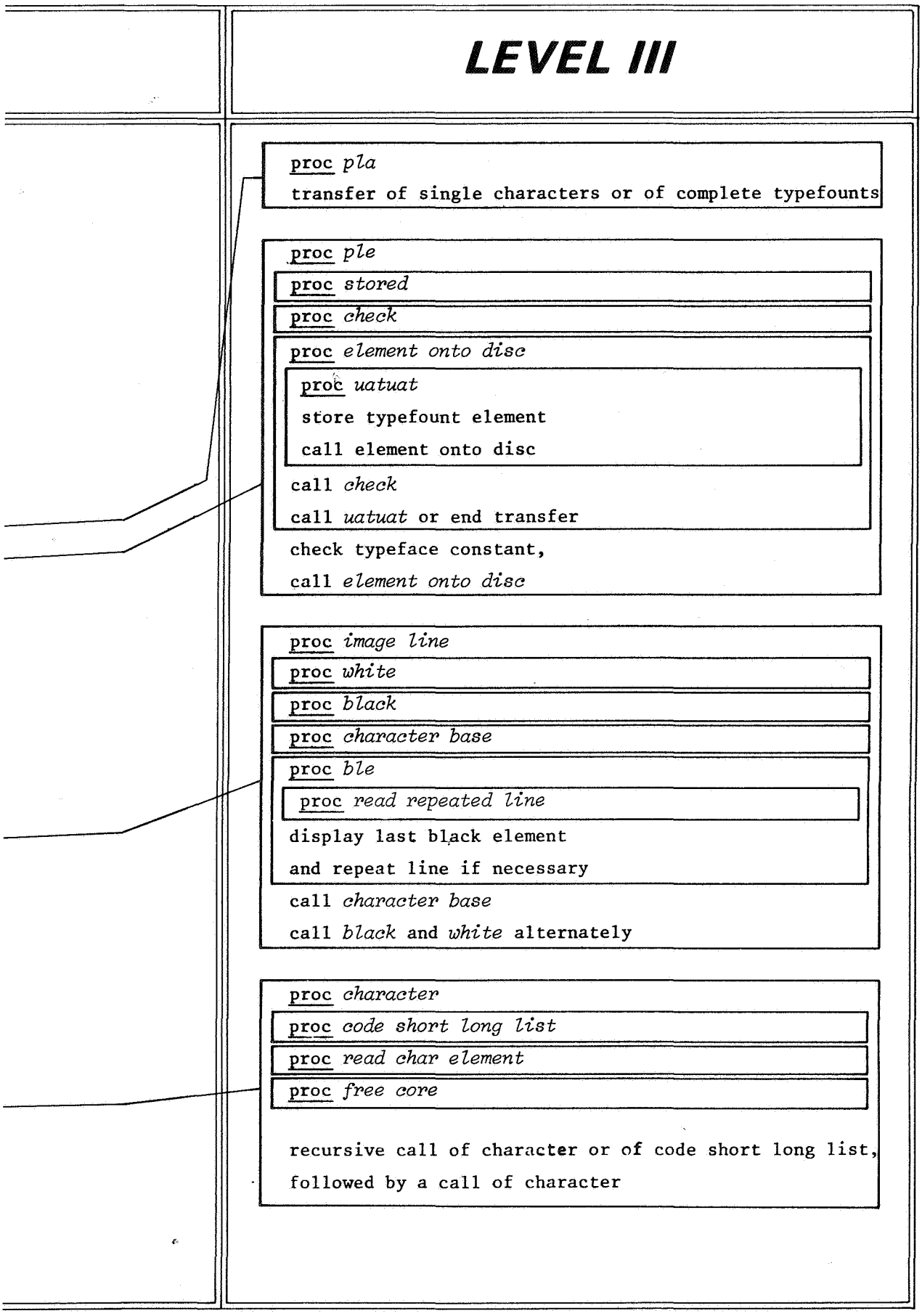


Fig. 2.5.1. The simulator program.

LEVEL III


proc pla
transfer of single characters or of complete typefounts
proc ple
proc stored
proc check
proc element onto disc
proc uatuat
store typefount element
call element onto disc
call check
call uatuat or end transfer
check typeface constant,
call element onto disc
proc image line
proc white
proc black
proc character base
proc ble
proc read repeated line
display last black element
and repeat line if necessary
call character base
call black and white alternately
proc character
proc code short long list
proc read char element
proc free core
recursive call of character or of code short long list,
followed by a call of character

ment of the line is encountered, procedure *ble* is called. This procedure displays the last element and repeats the whole line as many times as required. When repetition has to take place, *ble* uses the data from the array *repeated line*, which it reads with integer procedure *read repeated line*. Every time an image line is displayed, procedure *character base* is called from *ble*, which changes the recording position to the point on the character base line where the next image line has to begin.

Procedure *character (k, start)* is used to translate character input into the shortest possible machine code. For this purpose image lines of the character to be coded, are split by the procedure into possibly empty groups of short lines, followed by one long line. These groups can be coded independently from each other, because of the fact that their short lines for which coding is ambiguous, are enclosed between long lines, that have to be coded in two-byte mode. This process proceeds as follows (see fig. 2.5.2.). Procedure *character (k, start)* reads the k-th line of a sequence of the type just described and stores the length of the line elements and the number of times the line has to be repeated in array AA. Number of repetitions is stored in array element AA [*k,1*] and the length of the j-th line element in AA [*k,j*]. This reading and storage is done by the local procedure *read char element (k,j)* which recursively calls itself, thereby increasing *j* by one. If the line just read is a long line or the last line of the character, the sequence of *k* lines (whose data are stores in AA) is coded by procedure *code short long list (k, start)*. We will not discuss this procedure here because of the fact that this would require detailed knowledge of Digiset machine code. After execution of *code short long list*, it is checked whether there are still lines to be coded. If this is the case, *character (1, "false")* is called, and the whole process starts all over agaian, otherwise *character* is left. If the last line read by *character (k, start)* is a short line or an empty line however, *character (k+1, start)* is called and hence the next line is read. The translation starts with a call of *character (1, "true")*. The value "true" of the boolean *start* indicates that the coding of the character still has to begin, and hence, that the coding mode is still undefined.

Furthermore, *character* contains the procedure *free core (word pointer,*

seg pointer), which is used to calculate the secondary word address and relative segment address of typefount elements, that are input without specified addresses (see 2.1.).

3. EXERCISES

All exercises are generated in shorthand code, either by hand or by computer programs.
Although the syntax of Digiset programs is very simple (cf. appendix B), most exercises have past the simulator several times before the input was syntactically accepted. This convinced us of the fact that a syntactic check on hand written code is necessary, and that a syntactic check on computer generated code is a very useful testing facility for that computer program. Nevertheless we only give as examples simulator output produced by correct input.

Exercises on the lineprinter serve mainly to study single characters and alphabets. The mutual connection, transformation and positioning of characters can be judged on a plotter simulation.

After a succesful run on the simulator, most exercises have also been carried out on the Digiset itself. These pretested exercises proved to be failsafe: each run on the Digiset was faultless. For comparision both results are published here.

The exercises are instructive as:
- examples of Digiset programs
- illustrations of the working principles of the Digiset
- illustrations of the performance of simulator and Digiset
- an illustration of the capacity needed for storage of pictures as well as alphabets.

3.1. Exercise 1:

Define an alphabet containing one character, and display the character. The character chosen is an alto clef taken from music notation.

Each Digiset (sub)program contains the following possibly empty,

sequences of instructions:

- initialisation
- input of character data
- display of characters, mixed with lay out instructions (positioning etc.).

The program together with some explaining comment is listed below:

```
UAK;                          / begin of program
START;
UAK;                          / select segment Ø
SEG;
Ø;
UAK;                          / select base line
VERA;
UAK;                          / input to core segment
SPE;
62;                           / address of typeface constant
32;                           /\
Ø;                            /  ) typeface definition
Ø;                            /
UAT;                          /\ ) end of input to pa 62
UAT;                          /
UAK;                          / input to pa 63
SPE;
63;
1;                            / word address
1;
Ø;                            / character type
(7,1,86)(4,82,5)(4,1,86)      / image line information
(1,42,3)(1,41,5)(1,40,7)      / # of lines followed by
(1,39,9)(1,38,11)(1,9,6,22,   / pairs of white and
13,22,6)(1,7,10,19,15,19,10)  / black values.
(1,3,17,7,33,7,17)(1,2,18,7,12,9,12,7,18)
(1,2,17,8,10,13,10,8,17)(1,1,17,10,8,15,8,10,17)
(1,1,5,4,7,13,7,13,7,13,7,4,5)
(1,1,5,5,5,17,5,11,5,17,5,5,5)
```

```
(1,1,6,27,6,7,6,27,6)(1,1,6,27,7,5,7,27,6)
(1,1,7,27,7,3,7,27,7)(1,2,6,27,7,3,7,27,6)
(1,2,7,25,8,3,8,25,7)(1,3,7,21,10,5,10,21,7)
(1,3,9,17,11,7,11,17,9)(1,4,9,13,13,9,13,13,9)
(1,5,33,11,33)(1,5,31,15,31)(1,6,29,17,29)
(1,7,27,19,27)(1,8,25,21,25)(1,9,22,25,22)
(1,11,18,29,18)(1,12,15,33,15)(1,14,10,39,10)
(1,18,2,47,2);
UAT;                        / end of input for pa 63
UAT;                        / now the alto clef is defined
                            / as character #63 in segment Ø.
UAK;                        / position below base line,
VERU;                       / into usable recording
12Ø;                        / area.
Ø;
UAK;
ZWQ;                        / define space width
2ØØ;
Ø;
UAK;                        / define enlargement factor
GROa;
63;                         / display character
UAK;
STOP;                       / end of program
```

The result of this program for the lineprinter is given in fig. 3.1.1. Each line on the printer corresponds with one image line. Fig. 3.1.2. is the plotter simulation. Its shape is exactly equal to the character on the first line of the Digiset result (see fig. 3.1.3). The plotter simulation and the first character of the Digiset output have the same enlargement factor. For the Digiset exercise we have repeated the character over the whole range of enlargement factors (first line), followed by the whole

fig. 3.1.1. Alto clef character,
line printer simulation.

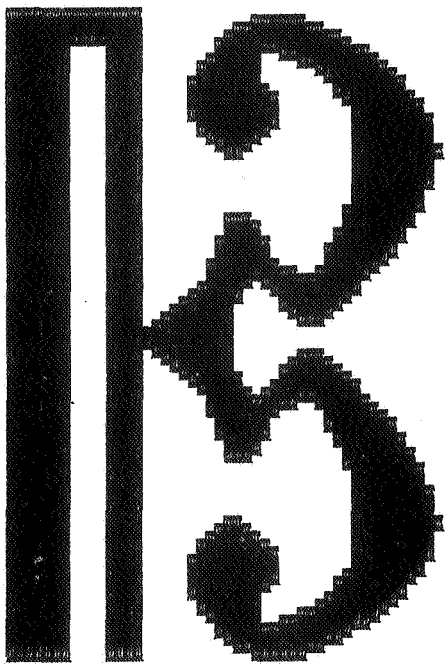


fig. 3.1.2. Alto clef character,
plotter simulation.

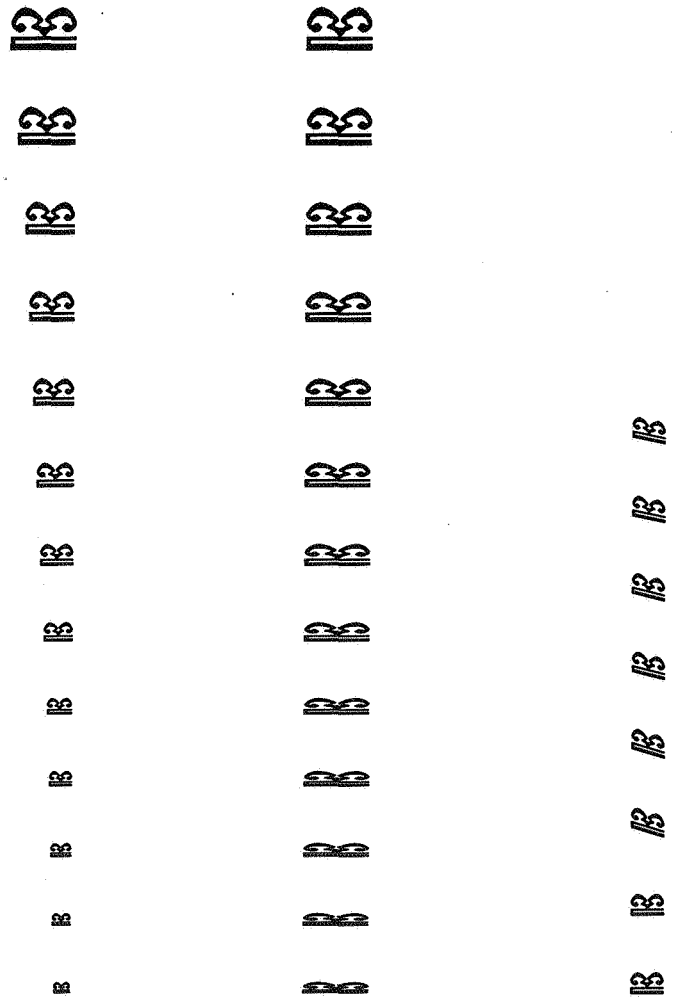


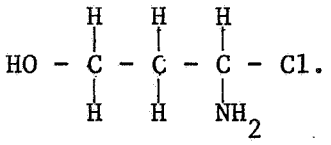
fig. 3.1.3. Alto clef series,
Digiset output.

range of width enlargements with constant maximal vertical enlargement (second line), followed by enlargement factor 2 (GROg) combined with the four available italicities.

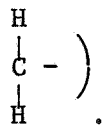
The complete program of the lineprinter exercise takes about 300 bytes in Digiset code, namely: 20 bytes for initialisation, 270 bytes for character definition and 10 bytes for positioning and display.

3.2. Exercise 2

The chemical formula



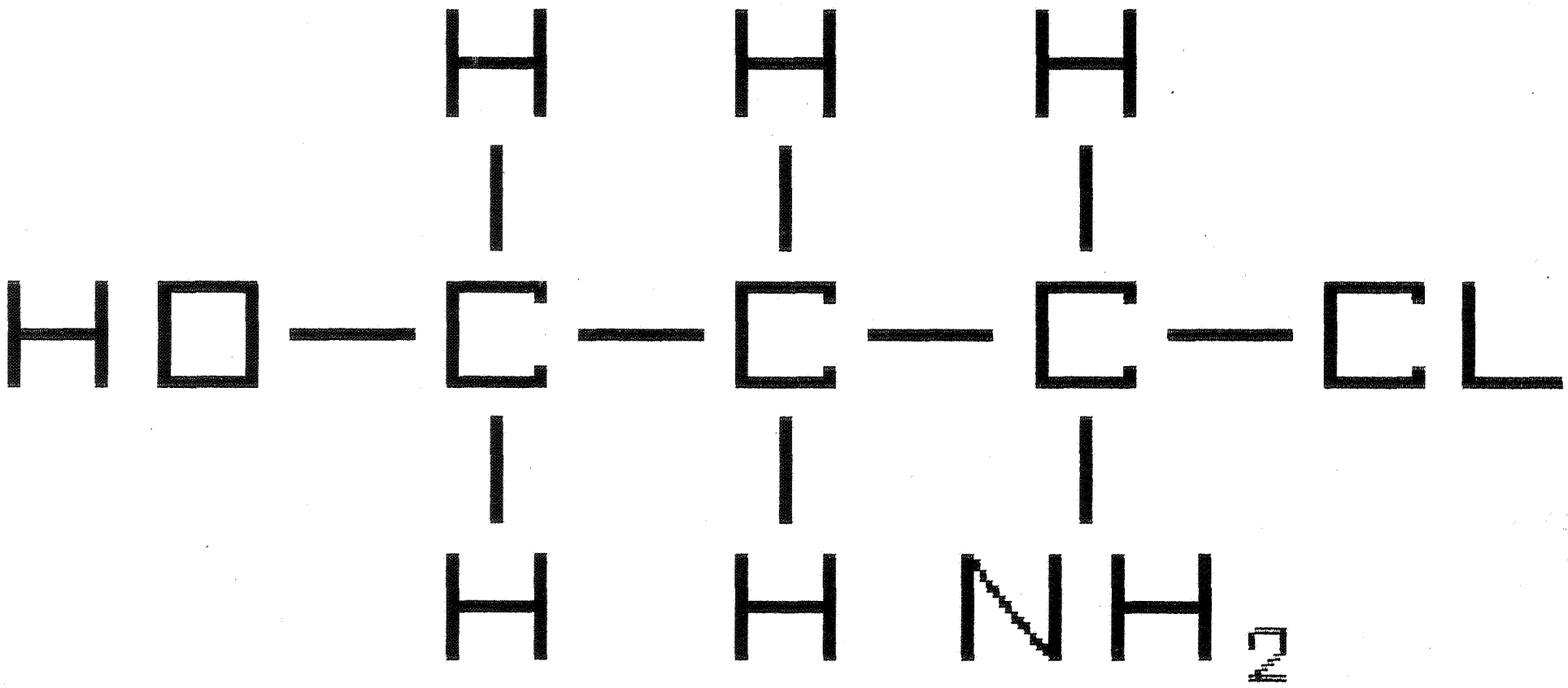
The formula is built up with the following alphabet:

- characters: O, H, N, C, 2, L, - , | ;
- microprogram: - CH_2 (namely:  .

Although the smallest possible charactersize was chosen, the lineprinter result still extends over 5 pages. Therefore only a part of the result is shown here (see fig. 3.2.1.). The plotter simulation takes a considerable amount of plottime. This is about the size of the largest picture one can produce on the plotter (see fig. 3.2.2.).

As far as actual drawing is concerned, the usefulness of the simulator lies mainly in the field of judging individual characters and character connections. Apart from this the best profit from the simulator is obtained by using it for syntactic pre-checking of the Digiset input, and for semantic checking by monitoring the Digiset reactions on the input without actually drawing. This produces error messages of the type: outside recording area, non existent character called, etc.

fig. 3.2.1. Chemical formula, fragment of line printer simulation. (reduced size)



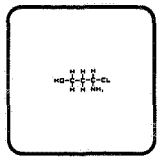


fig. 3.2.2b. Digiset output corresponding to plotter simulation

fig. 3.2.2a. Chemical formula, plotter simulation

The Digiset program takes about 35Ø bytes, this includes the character definitions (15Ø bytes) and the microprogram (5Ø bytes). The microprogram is called twice in succession by means of the WID command. Note that the chemical formulae HO - $(CH_2)_n$ - C^H_{NH} - CL, with n = 1,2,..., 2 ↑16 could have been drawn with a program that differs from the one listed below by not more than two bytes.

```
UAK; START;                    / begin of program
UAK; SEG; Ø;                   / define core segment
UAK; TAB; Ø; Ø;                / define tab position
UAK; VERA;                     / select type base line (y=Ø)
UAK; ZEV; Ø; Ø;                / select tab position (x=Ø)
UAK; SPE; 62;                  / define typeface constant:
32; Ø; Ø;                      / grid CI = 5Ø * 6Ø
UAT; UAT;                      / end of typeface data.
UAK; SPE; 63;                  / #63 = letter O
1; 1; 1;                       / address + character base line
(4)(2,5,2Ø)
(12,5,2,16,2)
(2,5,2Ø)(4);                   / 24 image lines
UAT; UAT;
UAK; SPE; 64;                  / #64 = letter H
6,1,1;
(4)(2,5,2Ø)
(12,14,2)(2,5,2Ø)(4)
UAT; UAT;
UAK; SPE; 65;                  / #65 = letter N
1Ø; 1; 1;
(4)(2,5,2Ø)(1,22,2)
(1,2Ø,2)(1,18,2)(1,17,2)
(1,15,2)(1,14,2)(1,12,2)
(1,11,2)(1,1Ø,2)(1,8,2)
(1,7,2)(1,6,2)(2,5,2Ø)
(4);
UAT; UAT;
```

```
UAK; SPE; 66;                    / #66 = letter C
21; 1; 1;
(4)(2,5,2Ø)(12,5,2)
(2,5,4,12,4)(4);
UAT; UAT;
UAK; SPE; 67;                    / #67 = cipher 2
27; 1; 1;
(2)  (1,1,1,5,3)(1,1,2,7,1)
(1,1,1,1,1,6,1)(1,1,1,2,1,5)
(1,1,1,3,1,4,1)(1,1,1,4,5)
(2);
UAT; UAT;
UAK; SPE; 68;                    / #68 = -
36; 1; 1;
(1)  (2Ø,14,2)(1);
UAT; UAT;
UAK; SPE; 69;                    / #69 = |
41; 1; 1;
(2,1,2Ø);
UAT; UAT;
UAK; SPE; 70;                    / #7Ø = microporgram
                                 /       for  -  CH2
43; 1; 2;                        / address + code for microprogram
[                                / start of microprogram
64;                              / display H
UAK; VERO; 6Ø; Ø;                / ↑ (move)
UAK; RUCK; 52; Ø;                / ← (move)
66;                              / display C
UAK; VERO; 6Ø; Ø;                / ↑
UAK; RUCK; 52; Ø;                / ←
69;                              / display |
UAK; VERO; 42; Ø;                / ↑
UAK; RUCK; 52; Ø;                / ←
64;                              / display H
```

```
UAK; VERU; 1Ø2; Ø;          / ↓
68;                         / display -
UAK; VERU; 1Ø2; Ø           / ↓
];                          / end of program
UAT;
UAT;                        / end of microprogram input
UAK; SPE; 71;               / letter L
8Ø; 1; 1;
(2)(2,5,2Ø)(14,5,2)(2);
UAT; UAT;                   / end of text data
                            / start of actual program
UAK; GROa;                  / define size + typeface
UAK; VERU; 3Ø; Ø;           / position in common recording
                            / area for lineprinter, plotter
                            / and Digiset.
64; 63; 68;                 / display H O -
UAK; VERU; 1Ø2; Ø;          / ↓
UAK; WID; 1; Ø;             / display - CH2 - CH2
7Ø;
65; 64; 67;                 / display NH2
UAK; VERO; 6Ø; Ø;           / ↑
UAK; RUCK; 188; Ø;          / ←
69;                         / display |
UAK; RUCK; 52; Ø;           / ←
UAK; VERO; 42; Ø;           / ↑
66;                         / display C
UAK; VERO; 6Ø; Ø;           / ↑
UAK; RUCK; 52; Ø;           / ←
69;                         / display |
UAK; VERO; 42; Ø;           / ↑
UAK; RUCK; 52; Ø;           / ←
64;                         / display H
UAK; VERU; 1Ø2; Ø;          / ↓
68; 66; 71;                 / display - CL
UAK; STOP;                  / end of program
```

3.3. Exercise 3; a characterset for a table of prime numbers

There exists a table of all prime numbers below 10^7 (see LEHMER [*]). One could produce a table of primes up to 10^9 in a volume of the same size, by making use of a special characterset.

The idea for this dense notation for the distribution of prime numbers originates with A. VAN WIJNGAARDEN.

A sequence of 5 consecutive numbers beginning with a 5-fold contains at most 2 primes (except for the sequence Ø, 1, 2, 3, 4). Moreover, the only candidates are the 2nd and 4th number or the 3rd and 5th number, depending on whether or not the first number is even. Therefore, it takes two bits to characterise a prime distribution in such a quintuple.

Let: Ø Ø, denoted by - , indicate no primes;
1 Ø, denoted by \ , indicate the left candidate is prime;
Ø 1, denoted by / , indicate the right candidate is prime;
1 1, denoted by | , indicate both candidates are prime.

Next, consider all characters that one obtains by taking all combinations of 4 elements from the set { -, \ , / , | }. This produces 256 characters. Each character can be interpreted in the obvious way as a notation for a prime distribution in a sequence of 2Ø consecutive numbers. By leaving out all 3-folds, one obtains 165 valid characters.

For a correct interpretation it is still necessary to know whether the first number in the sequence is even or not. It is possible to construct the table in such a way that each character represents a sequence that starts with a 20-fold (which implies an even five fold!).

We can for instance form blocks of 5 characters for each hundred.

```
e.g.: \ - - /
      ` - - \
      \ - \ -
      - | / \
      - | | -
```

fig. 3.3.1.

[*] P.N. LEHMER, list of prime numbers from 1 to 10.006.721, Hafner Publishing Co., New York 1956.

The size of these blocks can be reduced to a square of 2 by 2 mm. including spaces, and we would still have good readability. On a page in A4 - format we can write down 10^4 of these blocks, that is the distribution in a sequence of 10^6 numbers. Hence for the primes up to 10^9 we would need a volume of 1000 pages.

We will now as an illustration compare two ways of building up a character set for this book.

First solution: Observe that each character is build up from elements of the set:

- , \ , / and |.

We will define these four shapes as Digiset characters. The 165 valid characters for the 20-folds will be defined as microporgrams. Each micro-program will contain four character calls to one of the four primitive shapes. Since we intend to group five characters in a column (see fig. 3.2.1.), we can include in the microprogram the movement of the recording beam down to the beginning of the next row. For each block we need 5 microprogram calls followed by a move to the beginning of the next block.

The grouping of the blocks can be done in the same way for both solutions, and will not be discussed here.

The program fragment listed below contains the character definition of the four basic shapes, and two of the 165 microprograms.

```
UAK; START;                    / begin of program
UAK; SEG; Ø;                   / core address
UAK; SPE; 62;                  / store typeface constant
Ø; Ø; Ø;                       / 4-pt grid, 5Ø * 12Ø lines (type B)
UAT; UAT;
UAK; SPE; 1;                   / #1 = basic shape: -
1; 1; 1;                       / word #1 in segment #1
(1)(8,7,5)(1);                 / 1Ø image lines
UAT; UAT;
UAK; SPE; 2;                   / #2 = basic shape  \
```

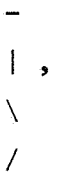
```
2Ø; 1; 1;
(1)(1,14,4)(1,12,6)
(1,1Ø,6)(1,8,6)(1,6,6)
(1,4,6)(1,2,6)(1,2,4)
(1);
UAT; UAT;
UAK; SPE; 3;                    / #3 = basic shape   /
4Ø; 1; 1;
(1)(1,2,4)(1,2,6)(1,4,6)
(1,6,6)(1,8,6)(1,1Ø,6)
(1,12,6)(1,14,4)(1);
UAT; UAT;
UAK; SPE; 4;                    / #4 = basic shape   |
6Ø; 1; 1;
(4)(2,2,16)(4);
UAT; UAT;
                                / by way of example we only give
                                / two of the 165 microprograms.
UAK; SPE; 6;                    / microprogram for:
8Ø; 1; 2;                       /    - - - -
[                               / (i.e. no primes).
1; 1; 1; 1;                     / display: - (4*)
UAK; RUCK; 16Ø; Ø;              / move to the beginning of
UAK; VERU; 43; Ø;               / the next row
];                              / end of microprogram
UAT; UAT;
.                               / other definitions
.
.
UAK; SPE; pa;                   / microprogram for:
wa; sa; 2;                          / / | \ .
[
3; 3; 4; 2;
UAK; RUCK; 16Ø; Ø;
```

```
UAK; VERU; 43; Ø;
];
UAT; UAT;                          / 20 bytes
```

It follows that for the coding of the alphabet about 3500 bytes are needed. The actual program takes 6 bytes for each hundred numbers, that is 60.000 bytes / page (1 page covers 10^6 numbers). One page would take about 1 minute Digiset time.

Second solution: The set of 165 characters is directly coded. This takes an average of 40 bytes a character, or ca. 7000 bytes for the alphabet. By displaying the characters "upright",

e.g.



each block of 100 also takes 6 calls.

Except for the alfabet, both solutions require an equal amount of coding space. However, the second solution may be faster, due to the fact, that a character contains less image lines and displaying of a character requires only one microprogram call. This gain in speed blances the much simpler coding of the alfabet of the first solution.

As an example we will now give the coding for two of the characters.

```
UAK; SPE; 1;                       / character for:    |
                                                       |
                                                       |
                                                       |
wa; sa; 1;                         / address + type
(4)
(2,3,2Ø,5,2Ø,5,2Ø,5,20)
(4);
```

```
UAT; UAT;
UAK; SPE; n;                    / character for:    /
                                                    |
                                                    -
                                                    \
wa; sa; 1;
(1)
(1,21,2,12,5,                   36,2
(1,18,5,12,5,                   36,5)               ( \
(1,15,5,15,5,                   39,5)
(1,12,5,18,5,12,2Ø,             10,5)
(1,9,5,21,5,12,2Ø               13,5)
(1,6,5,24,5,                    48,5)
(1,3,5,27,5,                    51,5)
(1,3,2,3Ø,5,                    54,2)
(1);
UAT; UAT;
```

This exercise has been run on the plotter (see fig. 3.3.2.), and on the Digiset itself (see fig. 3.3.3.). The plotter result is only partly reproduced.

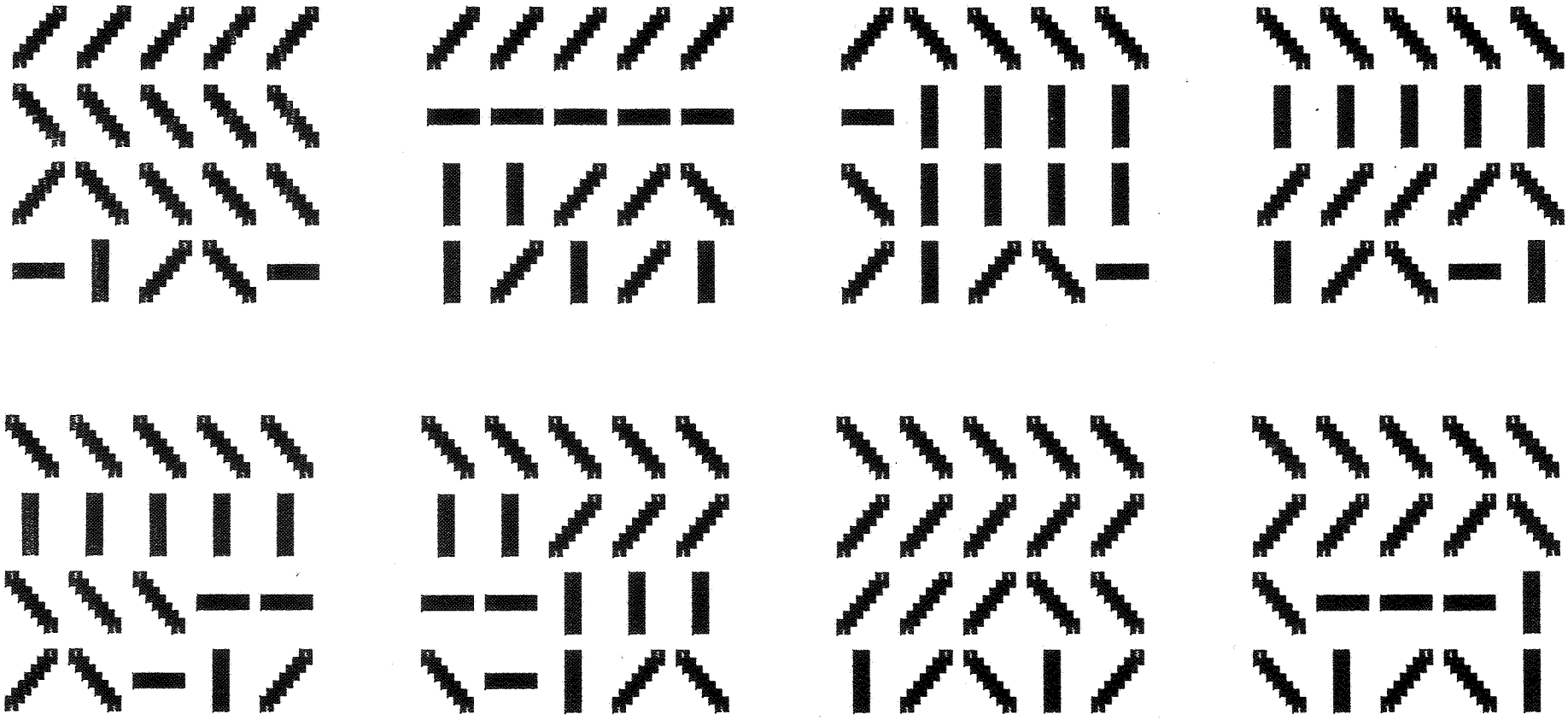


fig. 3.3.2. A character set for a table of prime numbers, plotter simulation of a fragment.

fig. 3.3.3. A character set for a table of prime numbers.

3.4. Exercise 4; an alphabet for line drawings

This exercise proves that the Digiset can be used as a machine that generates line drawings.

First the set up of the alphabet and the way in which a vector is drawn is outlined. Next an ALGOL 60 program is given, that actually geneates a sub alphabet in hand code.

The second part of the exercise shows some line drawings produced by ALGOL 60 programs that use character calls as primitive actions.

3.4.1. The alphabet

Each Digiset character must be defined within a square, the so-called reference square or unit square. The description of the character in the unit square is then transformed into a description on an existing square raster (the recording grid). The size of the recording grid and the distance of the grid points in x- and y direction can be chosen from a fixed list of alternatives. The reference point of the raster, that is the grid point that will coïncide with the actual position of the recording beam when the character is displayed, also must be specified in the definition.

A character consists of a piece of straight line, that enters and leaves the unit square entirely through the vertical or entirely through the horizontal sides (see fig. 3.4.1.).

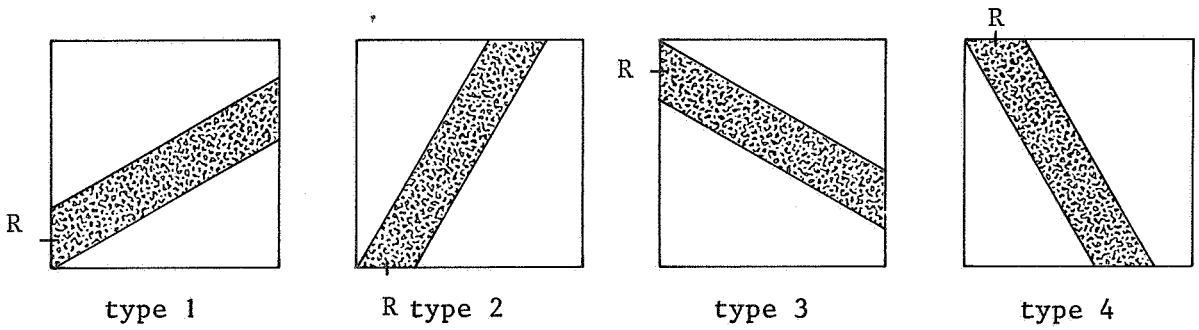
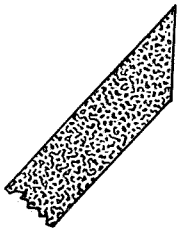


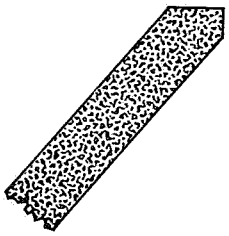
fig. 3.4.1. character types.

Each line piece starts in the lower left- or upper left corner. Hence there are four types of characters, namely two for each quadrant. Each type has a different reference point, at the position marked R. In case a line is cut at the corner diagonal to R, the square is a little extended

in one direction, such that a straight cut off can be made, e.g.:



in stead of



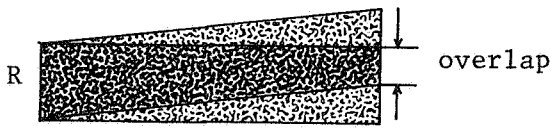
The straight cuts enables the connection of two characters of the same type without gaps or overlaps. Both would be visible and give poor quality.

We propose to draw an arbitrary straight line by approximating it with the two characters with the closest direction. The method of approximating is called threading. Due to some overlap in the directions, covered by the different types, each threading can be obtained of characters from one type only. (So we have no gaps there). Moreover, the threading is calculated, that gives the best approximation.

In order to make straight lines of arbitrary length, we must have at our disposal a sequence of charactersets which contain line pieces of decreasing length.

The alphabet chosen contains the following charactersets:

- The area between $-\pi/2$ and $+\pi/2$ is divided into 6Ø directions. There exist 6Ø line pieces, one for each direction, so that the mutual difference is less than or equal to 3°. The line pieces are defined as 8 point characters with a line width of .8 point (=Ø.Ø3 mm). This means that the overlap of two adjacent line pieces is about .5 line width:



  Each type contains 16 characters: 15 to cover all directions and one type overlapping character.

- A set of 34 characters consisting of:
  3Ø 4-point characters, each one consisting of a line piece differing from its neighbour by 6°, but with the same line width of Ø.Ø3 mm;

4 type-overlapping characters.

- A set containing:
  15 2-point characters, differing 12° and 4 overlapping characters.
- A set with 4 1-point characters and 2 overlapping characters.
- 1 .5-point character (the square point).

The complete characterset thus contains 124 characters. For each character a microprogram is added, that contains the character call followed by a positioning of the recording beam at the end of the line piece. Now one can draw a line simply by calling the appropriate microprograms one after another.

The 124 characters together with the 124 microprograms can be contained in one typefount. In this way a typefount can be defined for each line width desired.

In appendix C an ALGOL 6Ø program is listed that generates the 64 8-point line pieces mentioned above, together with the microprograms. This part of the typefount suffices for our testing purposes. The code for a line piece is shown for a few characters only (the complete set takes 18 pages).

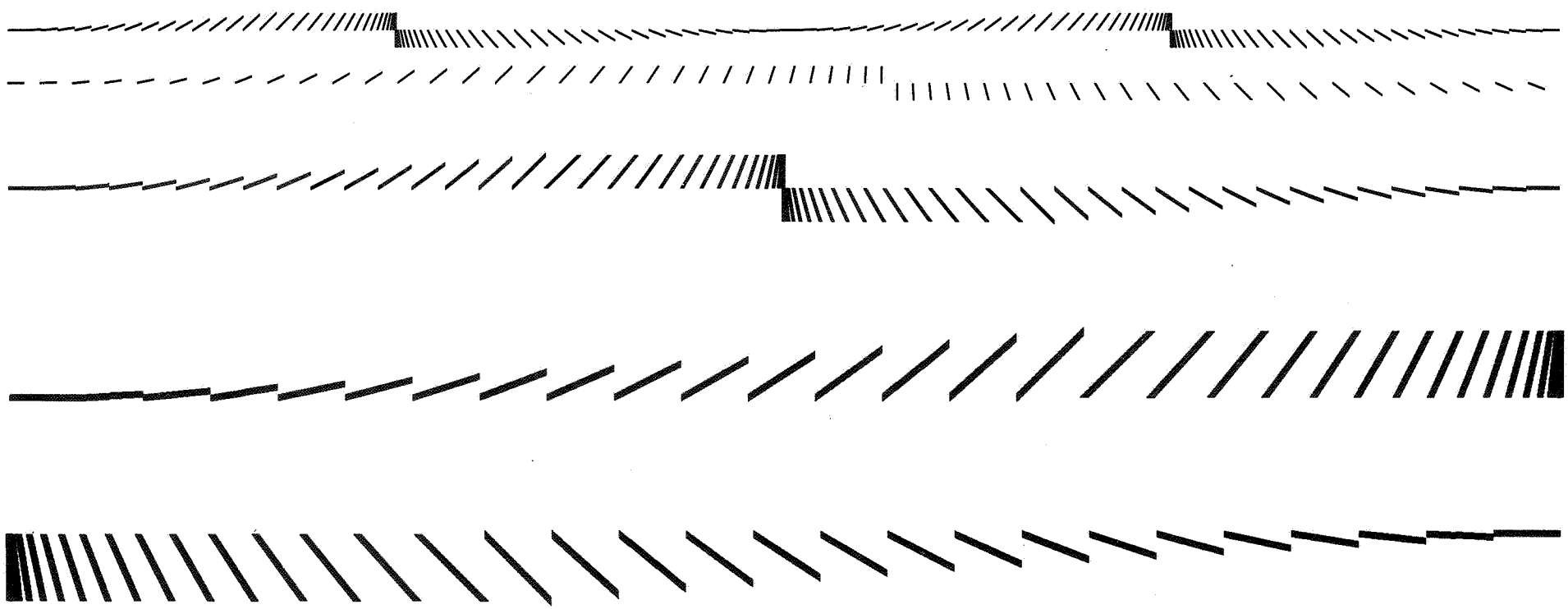


fig. 3.4.2. A character set for line drawings (line alfabet).

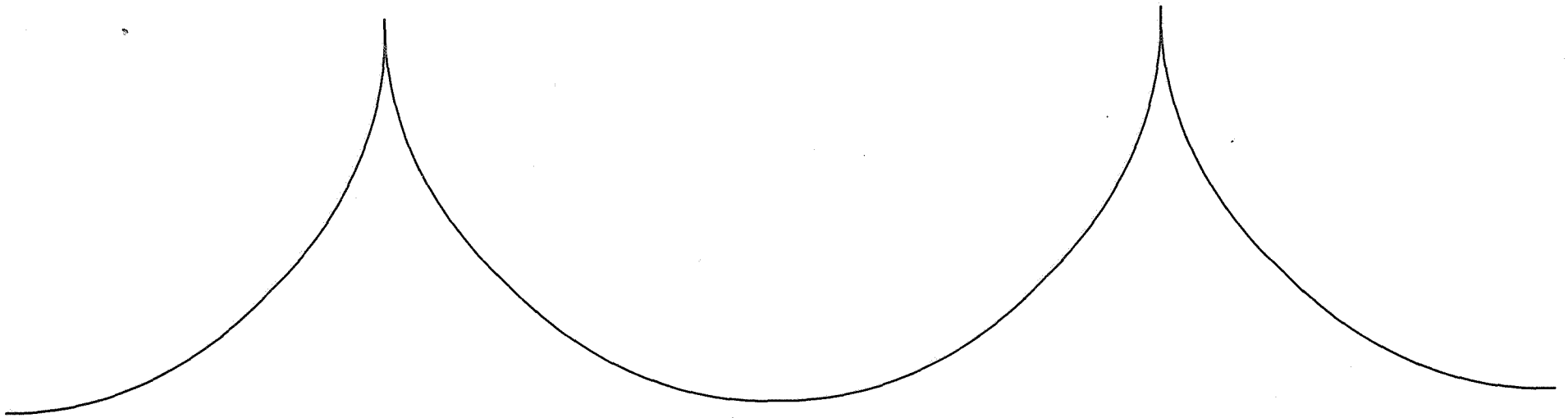
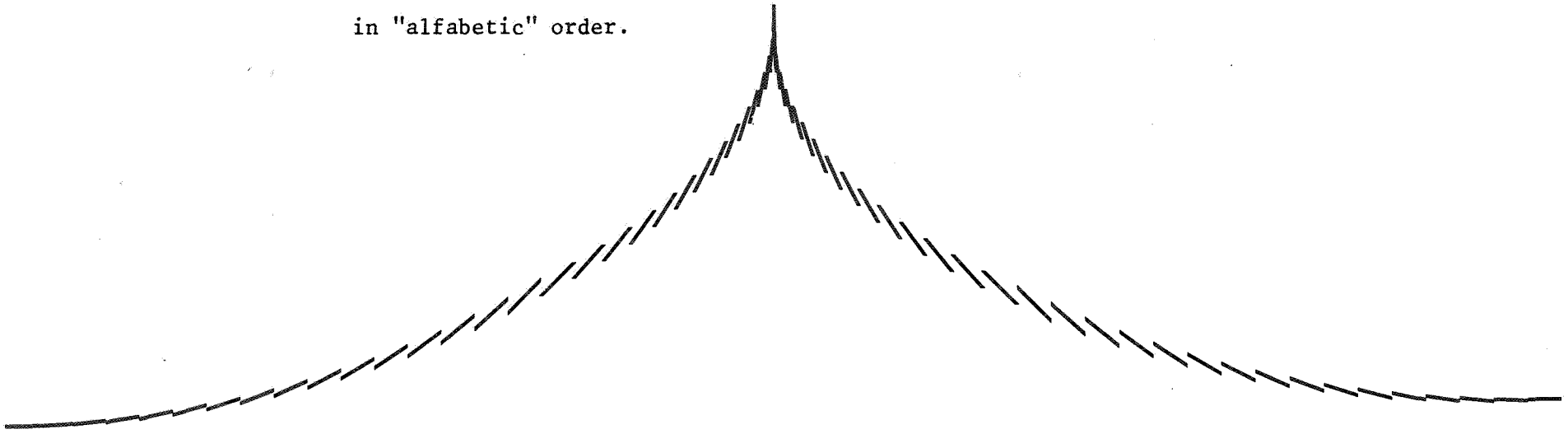


fig. 3.4.3. The line alfabet microprograms in "alfabetic" order.



In figure 3.4.2 the result of the printout of the alphabet on the Digiset is given in the following ways:

- on the first line the characterset is printed twice in real size (GROa)
- on the second line the characterset is printed once with a space after each character
- on the third line the characterset is enlarged by a factor 2
- on the 4th and 5th line the set is enlarged by a factor 4.

In figure 3.4.3. the characterset is printed twice by calling the microporgram for each character. This is repeated once on the second line with enlargement factor 2.

3.4.2. The line drawings

Each "sundown"-like drawing of figure 3.4.4. is the result of the program listed below, executed once for each character in the alphabet. The characters are called through their microprograms. In this way, one execution of 6 instructions produces a line consisting of 1Ø equal line pieces.

```
UAK; WID; 9; Ø;          / repeat microprogram * 1Ø times
mp;                      / primary address of microprogram
org;                     / move back to the starting point
```

The microprogram "org" stands for the sequence:

```
UAK; ZEV; Ø; Ø;          / move to Ø x position
UAK; VERA;               / move to Ø y position
```

Each one of the five subpictures shows the directions available within the alphabet. The black kernel at the origin has a radius of about two line pieces. Moreover, the shape of the kernel is exactly equal for all sub-pictures. Hence the shape is determined by the overlap only, and is not visibly infuenced by inaccuracies.

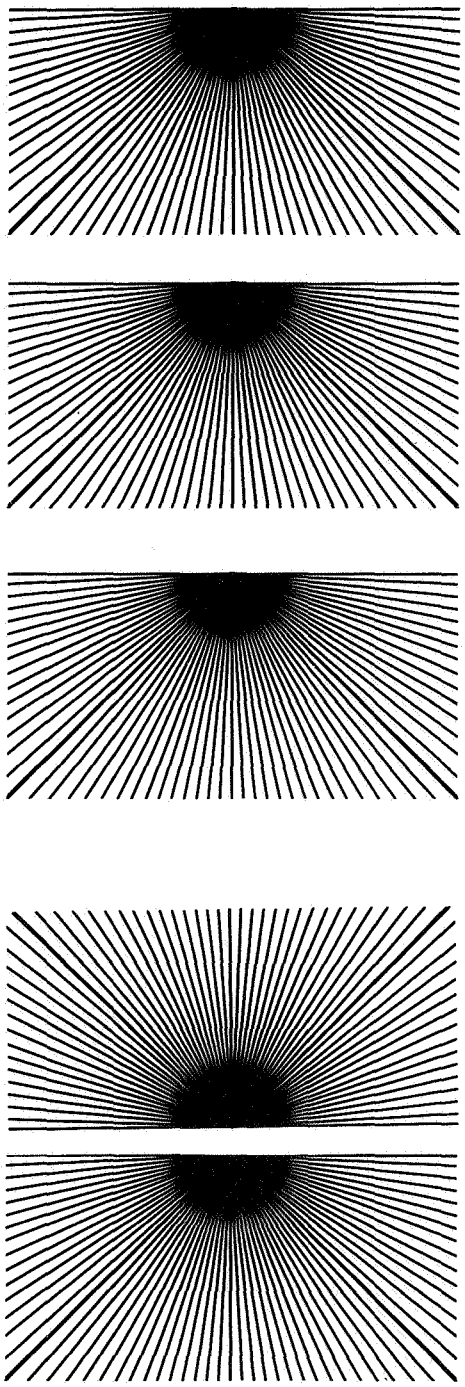


fig. 3.4.4. Repeated microprograms

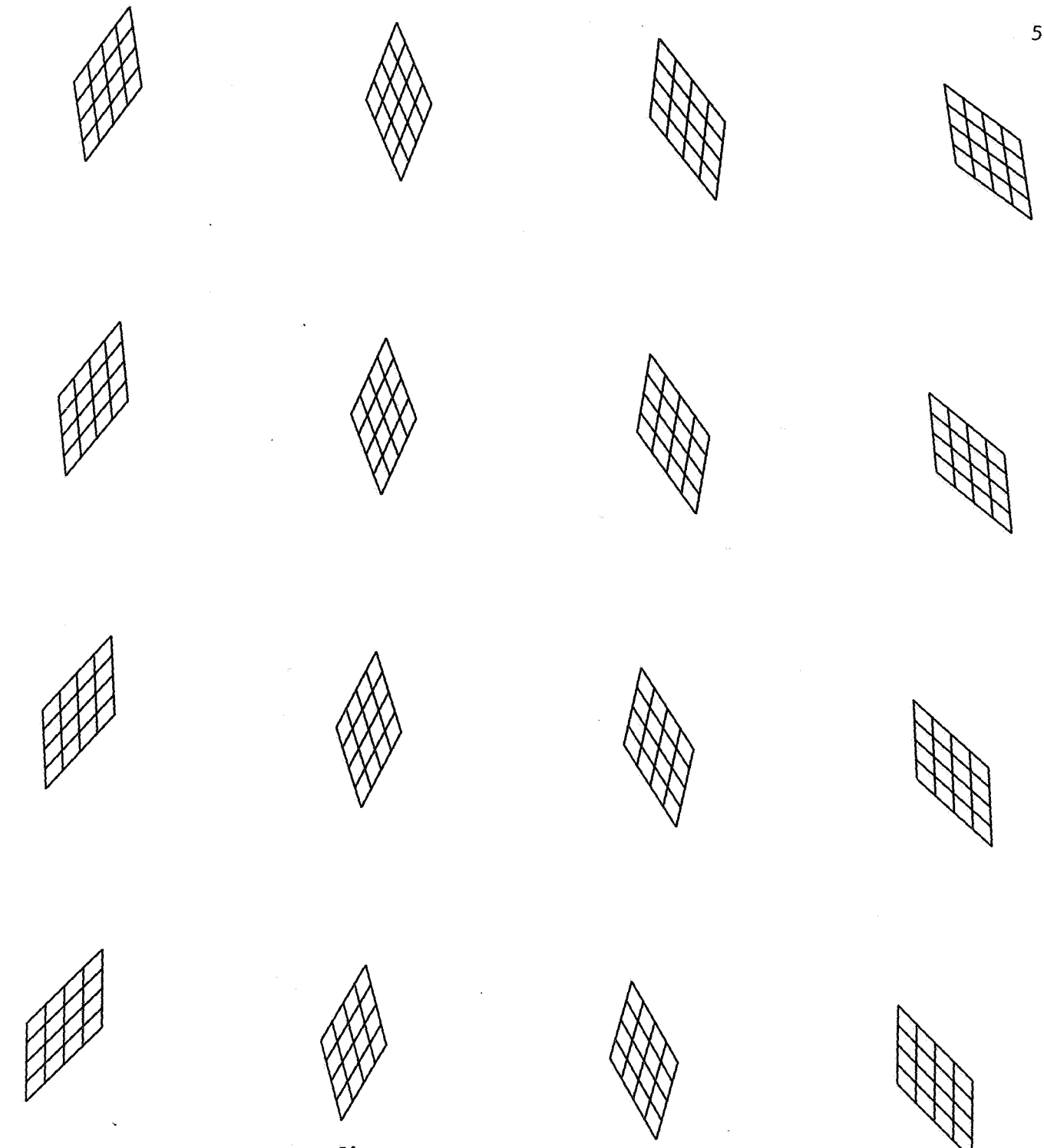
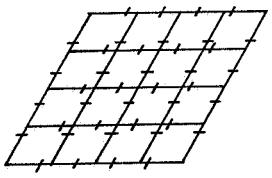


fig. 3.4.5. Diamonds.

The "diamond" shapes of figure 3.4.5. are obtained by combining two different characters of the alphabet as follows:



The length of the sides is five line pieces, hence nowhere inside the diamond, the connection between two pieces is covered by a crossing line.

None of the diamonds shows any irregularity caused by an improper connec - tion or improper positioning of its constituing pieces. The precision obtained, indeed equals the theoretical precision of the addressable points.

The ALGOL 60 program that generated this figure is also listed in appendix C. It is an example of the programming effort necessary for drawing on the Digiset. One should remember that the characterset needs to be generated only once. Hence we can say that the program listed is really all that is needed for this line drawing.

The short hand code program for one diamond is listed below. The x- and y direction are $\pi/4$ and $\pi/2$ respectively.

```
UAK; TAB; a1; a2;          / define new origin
org;                       / move to origin
UAK; WID; 4; Ø; X;         / draw "X" 5 times (first line)
org;
UAK; VERU; 32; 1;          / move to start of 2nd line
UAK; WID; 4; Ø; X;         / second line
org;
UAK; VERU; 64; 2;          / start of 3rd line
UAK; WID; 4; Ø; X;         / third line
org;
UAK; VERU; 96; 3;
UAK; WID; 4; Ø; X;         / fourth line
org;
UAK; VERU; 128; 4;
UAK; WID; 4; Ø; X;         / fifth line
```

```
org;
UAK; WID; 4; Ø; Y;          / draw "Y" 5 times (first line)
org;
UAK; ZWQ; 194; 1; ZWR;      / change x position
UAK; VERU; 32; 1;           / change y position
UAK; WID; 4; Ø; Y;          / second line
org;
 .                            .
 .                            .
 .                            .
UAK; WID; 4; Ø; Y;          / fifth line
org;                        / 116 bytes
```

APPENDIX A

The Digiset simulator in ALGOL 60.

B2665F.001,NOOTTENHAGEN ,R100

```
      'BEGIN'
 1        'INTEGER' CORESEGMENTS, DISCSECTORS, MAXLINENUMBERCHAR,
 2            DTAPELENGTH, LINENUMBERMAX,A;
 3        'BOOLEAN' PLOTTER, BOOLERROR, PUNCH;
 4
 5        'PROCEDURE' PARAMETER ERROR (STRING); 'STRING' STRING;
 6        'BEGIN' PRINTTEXT ("PARAMETER ERROR"); SPACE (5);
 7            PRINTTEXT (STRING); NLCR; BOOLERROR:= 'TRUE'
 8        'END' PARAMETER ERROR;
 9
10        BOOLERROR:= 'FALSE'; CORESEGMENTS:= READ;
11        'IF' CORESEGMENTS < 1 ∨ CORESEGMENTS > 64
12        'THEN' PARAMETER ERROR
13               ("NUMBER OF SEGMENTS OUTSIDE ADMISSIBLE RANGE");
14        DISCSECTORS:= READ;
15        'IF' DISCSECTORS < 1 ∨ DISCSECTORS > 3274
16        'THEN' PARAMETER ERROR ("NUMBER OF DISCSECTORS OUTSIDE RANGE");
17        MAXLINENUMBERCHAR:= READ;
18        'IF' MAXLINENUMBERCHAR > 15000
19        'THEN' PARAMETER ERROR
20               ("MAX.NUMBER OF IMAGE-LINES OUTSIDE ADMISSIBLE RANGE");
21        DTAPELENGTH:= READ; LINENUMBERMAX:= READ;
22        'IF' LINENUMBERMAX < 1 ∨ LINENUMBERMAX > 15000
23        'THEN' PARAMETER ERROR
24               ("SIMULATED RECORDING AREA NOT CONTAINED IN REAL
25               RECORDING AREA");
26        PLOTTER:= LINENUMBERMAX = 1;
27        A:= RESYM; PUNCH:= A = 25;
28        'IF' ¬ (PUNCH ∨ A = 91)
29        'THEN' PARAMETER ERROR
30               ("NON-ALLOWED PUNCH OPTION")
31        'ELSE' 'IF' PUNCH 'THEN'
32        'BEGIN'
33           'IF' ¬(RESYM = 91 ∧ RESYM = 91)
34            'THEN' PARAMETER ERROR
35                   ("WRONG NUMBER OF SPECIFICATION-PARAMETERS");
36          'FOR'A:= RESYM 'WHILE' A = 93 'DO';
37            'IF' A 'NE' 119
38            'THEN' PARAMETER ERROR
39                   ("PARAMETERS NOT ON SEPARATE CARD")
40        'END';
41        'BEGIN'
42            'INTEGER' CDO, CDAN, CGROA, CGROB, CGROC, CGROD, CGROE, CGROF,
43                CGROG, CGROH, CGROI, CGROJ, CGROK, CGROL, CGROM, CKEN,
44                CKUR1, CKUR2, CKUR3, CNOR, CNUL, CPLA, CPLE, CRUCK, CSCHN,
45                CSEG, CSPE, CSTART, CTAB, CTRE, CUAK, CUAT, CUBL, CVERA,
46                CVERO, CVERU, CWID, CZER, CZEV, CZWQ, CZWR, CSTOP,
47                ZWQ QUANTIFICATION, TAB QUANTIFICATION, KURN, SECTADDR,
48                PA, SEGM, SEGMF, SECSEG, SECPA, THIRDBYTE, BYTE, SHADING,
49                REMAINDER, GRIDTYPE, PRINTPOSITION, LINENUMBER, SECTBYTE,
50                INSTR NUMBER MIC, INSTR NUMBER, LINECOUNT,
51                BYTENUMBER INPUT, A, B, I, J, N, R, S, T;
52            'REAL' CELLWIDTH, CELLHEIGHT, WIDTH, HEIGHT, FACTOR;
53            'BOOLEAN' MIC, BUFFERSEG, BOOL RDDISC, BOOL WID, CHAR,
54                BOOL INPCORE, BOOL CHAR TO DISC, BOOL ZWQ;
55            'INTEGER' 'ARRAY' DTAPE [1 : DTAPELENGTH],
```

```
56              CORE [0 : CORESEGMENTS, 0 : 255, 1 : 3],
57              DISC [1 : DISCSECTORS, 1 : 744];
58
59          'REAL' 'ARRAY' ENLARGEMENT FACTOR [0 : 12];
60          'BOOLEAN' 'ARRAY' LINEPRINTEROUTPUT
61              [1 : LINENUMBERMAX, 1 : 144];
62
63          'INTEGER' 'PROCEDURE' READ DTAPE;
64          'BEGIN' READ DTAPE:= DTAPE [BYTENUMBER INPUT];
65              BYTENUMBER INPUT:= BYTENUMBER INPUT + 1;
66              'IF' ¬ BOOL INPCORE ∧ ¬ BOOL CHAR TO DISC
67              'THEN' INSTR NUMBER:= INSTR NUMBER + 1;
68
69          'END' READ DTAPE;
70
71          'INTEGER' 'PROCEDURE' READ DISC;
72          'BEGIN'
73              'IF' SECTADDR > DISCSECTORS
74              'THEN' ERROR
75                      ("INSTRUCTION CAUSES ADDRESSING OF NON-EXISTENT
76                      DISC-SECTOR", MIC, 'FALSE', 'TRUE');
77              READ DISC:= DISC [SECTADDR, SECTBYTE];
78              'IF' SECTBYTE < 744 'THEN' SECTBYTE:= SECTBYTE + 1 'ELSE'
79              'BEGIN' SECTADDR:= SECTADDR + 1; SECTBYTE:= 1 'END'
80          'END' READ DISC;
81
82          'INTEGER' 'PROCEDURE' READ CORE;
83          'BEGIN'
84              'IF' SECSEG > CORESEGMENTS
85              'THEN' ERROR
86                      ("NON-EXISTENT CORE-SEGM.ADDRESSED",
87                      MIC, 'FALSE', 'TRUE');
88              'IF' MIC ∧ ¬ CHAR
89              'THEN' INSTR NUMBER MIC:= INSTR NUMBER MIC + 1;
90              READ CORE:= CORE [SECSEG, SECPA, BYTE];
91              'IF' BYTE < 3 'THEN' BYTE:= BYTE + 1 'ELSE'
92              'IF' SECPA < 255 'THEN'
93              'BEGIN' SECPA:= SECPA + 1; BYTE:= 1 'END'
94              'ELSE'
95              'BEGIN' SECSEG:= SECSEG + 1; SECPA:= 0; BYTE:= 1 'END'
96          'END' READ CORE;
97
98          'INTEGER' 'PROCEDURE' RDTOD (BOOL RDDISC);
99          'BOOLEAN' BOOL RDDISC;
100         'IF' BOOL RDDISC 'THEN' RDTOD:= READ DISC
101         'ELSE' RDTOD:= READ DTAPE;
102
103         'INTEGER' 'PROCEDURE' RDTOC (MIC); 'BOOLEAN' MIC;
104         'BEGIN'
105             'IF' MIC 'THEN' RDTOC:= READ CORE
106             'ELSE' RDTOC:= READ DTAPE
107         'END' RDTOC;
108
109         'INTEGER' 'PROCEDURE' RDTODOC (BOOL RDDISC, MIC);
110         'BOOLEAN' MIC, BOOL RDDISC;
111         RDTODOC:=
112         'IF' BOOL RDDISC 'THEN' READ DISC 'ELSE' RDTOC (MIC);
113
114         'PROCEDURE' ERROR (STRING, MIC, CHAR, STOP); 'STRING' STRING;
115         'BOOLEAN' MIC, CHAR, STOP;
```

```
116          'BEGIN' 'INTEGER' N, M;
117              NLCR; PRINTTEXT (STRING); NLCR;
118              PRINTTEXT ("NUMBER INPUT-ELEMENT"); SPACE (33);
119              PRINT (INSTR NUMBER - 1); NLCR;
120              'IF' MIC 'THEN'
121              'BEGIN' PRINTTEXT ("INSTRUCTION-NUMBER MICROPROGRAM");
122                  SPACE (22); PRINT (INSTR NUMBER MIC); NLCR;
123              'END';
124              'IF' CHAR 'THEN'
125              'BEGIN' PRINTTEXT ("LINENUMBER CHARACTER"); SPACE (33);
126                  PRINT (LINECOUNT); NLCR;
127              'END';
128              'IF' STOP 'THEN'
129              'BEGIN' NEW PAGE;
130                  'IF' ¬ PLOTTER 'THEN'
131                  'BEGIN'
132                      'FOR' N:= 1 'STEP' 1 'UNTIL' LINENUMBERMAX 'DO'
133                      'FOR' M:= 1 'STEP' 1 'UNTIL' 144 'DO'
134                      'IF' LINEPRINTEROUTPUT [N, M] 'THEN' PRSYM (133)
135                      'ELSE' SPACE (1)
136                  'END';
137                  EXIT
138              'END'
139              'ELSE' BOOLERROR:= 'TRUE'
140          'END' ERROR;
141
142          'PROCEDURE' PRINTRELATIVE
143          (INIT LINENUMBER, INIT PRINTPOS, DELTAX, DELTAY,
144           LINENUMBERMAX, PRINT);
145          'INTEGER' DELTAX, DELTAY, LINENUMBERMAX;
146          'BOOLEAN' PRINT, INIT LINENUMBER, INIT PRINTPOS;
147          'COMMENT' THE GLOBAL PARAMETERS LINENUMBER AND PRINTPOSITION
148              ARE CHANGED;
149          'BEGIN'
150              'IF' INIT LINENUMBER 'THEN' LINENUMBER:= DELTAX
151              'ELSE' LINENUMBER:= LINENUMBER + DELTAX;
152              'IF' INIT PRINTPOS 'THEN' PRINTPOSITION:= DELTAY
153              'ELSE' PRINTPOSITION:= PRINTPOSITION + DELTAY;
154              'IF' PRINTPOSITION < 1 ∨ PRINTPOSITION > 144
155              'THEN' ERROR ("Y OUTSIDE PRINT-RANGE", MIC, CHAR, 'TRUE');
156              'IF' LINENUMBER < 1 ∨ LINENUMBER > LINENUMBERMAX
157              'THEN' ERROR ("X OUTSIDEPRINT-RANGE", MIC, CHAR, 'TRUE');
158              'IF' PRINT
159              'THEN' LINEPRINTEROUTPUT [LINENUMBER, PRINTPOSITION]:=
160                     'TRUE'
161          'END' PRINTRELATIVE;
162
163          'PROCEDURE' PLOTRELATIVE
164          (INITIAL VALUE, DELTAX, DELTAY, IPEN, KURN);
165          'INTEGER' DELTAX, DELTAY, IPEN, KURN; 'BOOLEAN' INITIAL VALUE;
166          'BEGIN' 'OWN' 'INTEGER' X, Y; 'REAL' TG;
167              'IF' INITIAL VALUE 'THEN' 'BEGIN' X:= 0; Y:= 1375 'END'
168              'ELSE'
169              'IF' KURN = 0 ∨ DELTAY = 0 'THEN'
170              'BEGIN' X:= X + DELTAX; Y:= Y + DELTAY;  'END'
171              'ELSE'
172              'BEGIN'
173                  'IF' KURN = 1 'THEN' TG:= 0,5205671 'ELSE'
174                  'IF' KURN = 2 'THEN' TG:= 0,4663077
175                 'ELSE' TG:= 0,4142136;
```

```
176                 Y:= Y + DELTAY; X:= X + TG * DELTAY;
177             'END';
178             'IF' ¬ (0 'LE' X ∧ X 'LE' 15000)
179             'THEN' ERROR ("X OUTSIDE PLOTRANGE", MIC, CHAR, 'TRUE');
180             'IF' ¬ (0 'LE' Y ∧ Y 'LE' 2750)
181             'THEN' ERROR ("Y OUTSIDE PLOTRANGE", MIC, CHAR, 'TRUE');
182             PLOT (X, Y, IPEN)
183         'END' PLOTRELATIVE;
184
185         'PROCEDURE' INTERPRET (S, MIC, BOOL RDDISC, PLOTTER);
186         'INTEGER' S; 'BOOLEAN' MIC, BOOL RDDISC, PLOTTER;
187         'BEGIN' 'INTEGER' R, SECPAMIC, SECSEGMIC, BYTEMIC, N, M;
188             'IF' S = CUAK 'THEN'
189             'BEGIN' R:= RDTODOC (BOOL RDDISC, MIC);
190                 'IF' R = CSPE 'THEN' SPE (MIC) 'ELSE'
191                 'IF' R = CSTOP 'THEN' STOP (PLOTTER) 'ELSE'
192                 'IF' R = CSEG 'THEN' SEG (RDTOC (MIC), SEGM, SEGMF)
193                 'ELSE'
194                 'IF' R = CDAN
195                 'THEN' DAN
196                        (RDTOC (MIC), REMAINDER, SHADING, FACTOR,
197                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
198                 'ELSE'
199                 'IF' R = CGROA
200                 'THEN' GRO
201                        (0, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
202                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
203                 'ELSE'
204                 'IF' R = CGROB
205                 'THEN' GRO
206                        (1, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
207                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
208                 'ELSE'
209                 'IF' R = CGROC
210                 'THEN' GRO
211                        (2, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
212                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
213                 'ELSE'
214                 'IF' R = CGROD
215                 'THEN' GRO
216                        (3, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
217                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
218                 'ELSE'
219                 'IF' R = CGROE
220                 'THEN' GRO
221                        (4, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
222                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
223                 'ELSE'
224                 'IF' R = CGROF
225                 'THEN' GRO
226                        (5, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
227                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
228                 'ELSE'
229                 'IF' R = CGROG
230                 'THEN' GRO
231                        (6, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
232                         CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
233                 'ELSE'
234                 'IF' R = CGROH
235                 'THEN' GRO
```

```
236                     (7, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
237                      CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
238                 'ELSE'
239                 'IF' R = CGROI
240                 'THEN' GRO
241                     (8, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
242                      CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
243                 'ELSE'
244                 'IF' R = CGROJ
245                 'THEN' GRO
246                     (9, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR,
247                      CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT, PLOTTER)
248                 'ELSE'
249                 'IF' R = CGROK
250                 'THEN' GRO
251                     (10, REMAINDER, SHADING, GRIDTYPE, KURN,
252                      FACTOR, CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT,
253                      PLOTTER)
254                 'ELSE'
255                 'IF' R = CGROL
256                 'THEN' GRO
257                     (11, REMAINDER, SHADING, GRIDTYPE, KURN,
258                      FACTOR, CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT,
259                      PLOTTER)
260                 'ELSE'
261                 'IF' R = CGROM
262                 'THEN' GRO
263                     (12, REMAINDER, SHADING, GRIDTYPE, KURN,
264                      FACTOR, CELLWIDTH, WIDTH, CELLHEIGHT, HEIGHT,
265                      PLOTTER)
266                 'ELSE'
267                 'IF' R = CNOR 'THEN' KURNOR (0, KURN, PLOTTER) 'ELSE'
268                 'IF' R = CKUR1 'THEN' KURNOR (1, KURN, PLOTTER) 'ELSE'
269                 'IF' R = CKUR2 'THEN' KURNOR (2, KURN, PLOTTER) 'ELSE'
270                 'IF' R = CKUR3 'THEN' KURNOR (3, KURN, PLOTTER) 'ELSE'
271                 'IF' R = CUBL 'THEN'
272                 'BEGIN'
273                     'IF' MIC
274                     'THEN' ERROR
275                             ("UBL IN MIC", 'TRUE', 'FALSE', 'TRUE')
276                     'ELSE' UBL
277                 'END'
278                 'ELSE'
279                 'IF' R = CRUCK
280                 'THEN' RUCK (RDTOC (MIC), RDTOC (MIC), PLOTTER)
281                 'ELSE'
282                 'IF' R = CVERO
283                 'THEN' VERO (RDTOC (MIC), RDTOC (MIC), PLOTTER)
284                 'ELSE'
285                 'IF' R = CVERU
286                 'THEN' VERU (RDTOC (MIC), RDTOC (MIC), PLOTTER)
287                 'ELSE'
288                 'IF' R = CVERA 'THEN' VERA (PLOTTER) 'ELSE'
289                 'IF' R = CZWQ
290                 'THEN' ZWQ
291                     (RDTOC (MIC), RDTOC (MIC), ZWQ QUANTIFICATION,
292                      BOOLZWQ)
293                 'ELSE'
294                 'IF' R = CTAB
295                 'THEN' TAB
```

```
296                         (RDTOC (MIC), RDTOC (MIC), TAB QUANTIFICATION,
297                          MIC)
298                 'ELSE'
299                 'IF' R = CZER
300                 'THEN' ZER
301                         (RDTOC (MIC), RDTOC (MIC), TAB QUANTIFICATION,
302                          PLOTTER)
303                 'ELSE'
304                 'IF' R = CZEV
305                 'THEN' ZEV
306                         (RDTOC (MIC), RDTOC (MIC), TAB QUANTIFICATION,
307                          PLOTTER)
308                 'ELSE'
309                 'IF' R = CWID
310                 'THEN' WID (RDTOC (MIC), RDTOC (MIC), MIC, BOOL WID)
311                 'ELSE'
312                 'IF' R = CPLA ∨ R = CPLE ∨ R = CTRE
313                 'THEN' ERROR
314                         ("PLA,PLE OR TRE NOT PRECEDED BY UAK;SPE;",
315                          MIC, 'FALSE', 'TRUE')
316                 'ELSE' ERROR
317                         ("INSTRUCTION CANNOT BE PRECEDED BY UAK", MIC,
318                          'FALSE', 'TRUE')
319             'END'
320             'ELSE'
321             'IF' S = CZWR
322             'THEN' ZWR (PLOTTER, BOOLZWQ, ZWQ QUANTIFICATION)
323             'ELSE'
324             'IF' S = CKEN 'THEN'
325             'BEGIN'
326                 'IF' MIC
327                 'THEN' ERROR ("KEN IN MIC", 'TRUE', 'FALSE', 'TRUE')
328                 'ELSE' KEN
329             'END'
330             'ELSE'
331             'IF' S = CNUL
332             'THEN' ERROR
333                     ("NUL-INSTRUCTION NOT SIMULATED", MIC, 'FALSE',
334                      'TRUE')
335             'ELSE'
336             'IF' S = CUAT
337             'THEN' ERROR
338                     ("UAT-INSTRUCTION NOT ALLOWED IN THIS PLACE", MIC,
339                      'FALSE', 'TRUE')
340             'ELSE'
341             'BEGIN'
342                 'IF' MIC 'THEN'
343                 'BEGIN' SECPAMIC:= SECPA; SECSEGMIC:= SECSEG;
344                     BYTEMIC:= BYTE;
345                     CALL TYPEFOUNT ELEMENT
346                     (S, PLOTTER, CELLHEIGHT, CELLWIDTH, SHADING,
347                      REMAINDER, GRIDTYPE, KURN);
348                     SECSEG:= SECSEGMIC; SECPA:= SECPAMIC;
349                     BYTE:= BYTEMIC;
350                 'END'
351                 'ELSE' CALL TYPEFOUNT ELEMENT
352                         (S, PLOTTER, CELLHEIGHT, CELLWIDTH, SHADING,
353                          REMAINDER, GRIDTYPE, KURN)
354             'END'
355         'END' INTERPRET;
```

```
356
357         'PROCEDURE' STOP (PLOTTER); 'BOOLEAN' PLOTTER;
358         'BEGIN' 'INTEGER' N, M;
359             'IF' ¬ PLOTTER 'THEN'
360             'BEGIN'
361                 'FOR' N:= 1 'STEP' 1 'UNTIL' LINENUMBERMAX 'DO'
362                 'FOR' M:= 1 'STEP' 1 'UNTIL' 144 'DO'
363                 'IF' LINEPRINTEROUTPUT [N, M] 'THEN' PRSYM (133)
364                 'ELSE' SPACE (1);
365
366             'END';
367             EXIT
368         'END' STOP;
369
370         'PROCEDURE' SEG (BYTE, SEGM, SEGMF);
371         'INTEGER' BYTE, SEGM, SEGMF;
372         'BEGIN' SEGMF:= SEGM; SEGM:= BYTE;
373             'IF' SEGM > CORESEGMENTS
374             'THEN' ERROR
375                    ("NON-EXISTENT CORE-SEGMENT", MIC, 'FALSE', 'TRUE')
376         'END' SEG;
377
378         'PROCEDURE' DAN
379         (BYTE, REMAINDER, SHADING, FACTOR, CELLWIDTH, WIDTH,
380          CELLHEIGTH, HEIGTH, PLOTTER);
381         'INTEGER' BYTE, REMAINDER, SHADING;
382         'REAL' FACTOR, CELLWIDTH, WIDTH, CELLHEIGTH, HEIGTH;
383         'BOOLEAN' PLOTTER;
384         'BEGIN' 'REAL' FACTORDAN;
385             'IF' ¬ PLOTTER 'THEN'
386             'BEGIN'
387                 'IF' MIC
388                 'THEN' INSTR NUMBER MIC:= INSTR NUMBER MIC - 1
389                 'ELSE' INSTR NUMBER:= INSTR NUMBER - 1;
390                 ERROR
391                 ("DAN INSTRUCTION AND LINEPRINTEROUTPUT", MIC,
392                  'FALSE', 'TRUE');
393
394             'END';
395             'IF' 7 < BYTE ∧ BYTE < 13
396             'THEN' FACTORDAN:= ENLARGEMENT FACTOR [BYTE - 8]
397             'ELSE'
398             'IF' 13 < BYTE ∧ BYTE < 25
399             'THEN' FACTORDAN:= ENLARGEMENT FACTOR [BYTE / 2 - 2]
400             'ELSE'
401             'IF' BYTE = 28 'THEN' FACTORDAN:= ENLARGEMENT FACTOR [11]
402             'ELSE'
403             'IF' BYTE = 32 'THEN' FACTORDAN:= ENLARGEMENT FACTOR [12]
404             'ELSE' ERROR ("QUANTIFICATION DAN", MIC, 'FALSE', 'TRUE');
405             FACTOR:= 1; CELLWIDTH:= WIDTH * FACTORDAN;
406             CELLHEIGHT:= HEIGTH; SHADING:= ENTIER (CELLWIDTH / 4);
407             REMAINDER:= CELLWIDTH - 4 * SHADING;
408             'IF' REMAINDER = 4 'THEN'
409             'BEGIN' SHADING:= SHADING + 1; REMAINDER:= 0 'END'
410         'END' DAN;
411
412         'PROCEDURE' GRO
413         (N, REMAINDER, SHADING, GRIDTYPE, KURN, FACTOR, CELLWIDTH,
414          WIDTH, CELLHEIGTH, HEIGTH, PLOTTER);
415         'INTEGER' N, REMAINDER, SHADING, GRIDTYPE, KURN;
```

```
'REAL' FACTOR, CELLWIDTH, WIDTH, CELLHEIGTH, HEIGTH;
'BOOLEAN' PLOTTER;
'BEGIN' 'INTEGER' R;
    'IF' SEGM = - 1
    'THEN' ERROR
           ("GRO NOT PRECEDED BY SEG", MIC, 'FALSE', 'TRUE');
    R:= CORE [SEGM, 62, 1];
    'IF' ¬ ((0 'LE' R ∧ R 'LE' 19) ∨ (32 'LE' R ∧ R 'LE' 51) ∨
          (64 'LE' R ∧ R 'LE' 83))
    'THEN' ERROR
           ("TYPEFACE-CONST. HAS NON-ALLOWED VALUE",
           MIC, 'FALSE', 'TRUE');
    'IF' PLOTTER 'THEN'
    'BEGIN' KURN:= R - 4 * ENTIER (R / 4);
        'IF' 0 'LE' R ∧ R 'LE' 19 'THEN'
        'BEGIN' WIDTH:= 12; HEIGTH:= 5 'END'
        'ELSE'
        'IF' 32 'LE' R ∧ R 'LE' 51 'THEN'
        'BEGIN' WIDTH:= 12; HEIGTH:= 10; R:= R - 32 'END'
        'ELSE'
        'IF' 64 'LE' R ∧ R 'LE' 83 'THEN'
        'BEGIN' WIDTH:= 24; HEIGTH:= 10; R:= R - 64 'END'
        'ELSE' ERROR
               ("TYPEFACECONSTANT", MIC, 'FALSE', 'TRUE');
        FACTOR:= ENLARGEMENT FACTOR [N];
        CELLWIDTH:= WIDTH * FACTOR;
        CELLHEIGHT:= HEIGTH * FACTOR;
        SHADING:= ENTIER (CELLWIDTH / 4);
        REMAINDER:= CELLWIDTH - 4 * SHADING;
        'IF' REMAINDER = 4 'THEN'
        'BEGIN' SHADING:= SHADING + 1; REMAINDER:= 0 'END'
    'END'
    'ELSE'
    'BEGIN'
        'IF' ¬ N = 0
        'THEN' ERROR
               ("NOT GROA AND LINEPRINTEROUTPUT",
               MIC, 'FALSE','TRUE')
        'ELSE' KURN:= 0;
        'IF' 32 'LE' R ∧ R 'LE' 51 'THEN' R:= R - 32
        'ELSE' ERROR
               ("B-OR D-GRID AND LINEPRINTEROUTPUT", MIC,
                'FALSE', 'TRUE')
    'END';
    GRIDTYPE:= 2 ** ENTIER (R / 4);
'END' GRO;

'PROCEDURE' KURNOR (N, KURN, PLOTTER); 'INTEGER' N, KURN;
'BOOLEAN' PLOTTER;
'BEGIN'
    'IF' PLOTTER 'THEN' KURN:= N
    'ELSE' ERROR
           ("KUR/NOR-INSTRUCTION AND LINEPRINTEROUTPUT", MIC,
            'FALSE', 'TRUE')
'END' KURNOR;

'PROCEDURE' KEN;
'COMMENT' THE KEN-INSTRUCTION IS ONLY SIMULATED IN
    TEXT-DATA,NOT IN CHARACTER-OR MICROPROGRAM-DATA;
'BEGIN' 'INTEGER' R, S, T;
```

```
476             R:= READ DTAPE; S:= READ DTAPE; T:= READ DTAPE;
477             'IF' ((R = CUAK) ∧ (S = CSPE)) ∨ ((S = CUAK) ∧ (T = CSPE))
478                  ∨ (R = CKEN ∧ S = CKEN ∧ T = CKEN)
479             'THEN' ERROR
480                    ("UAK SPE OR KEN KEN KEN IN KEN", 'FALSE', 'FALSE',
481                     'TRUE')
482         'END' KEN;
483
484         'PROCEDURE' UBL;
485         'COMMENT' THE UBL-INSTRUCTION IS ONLY SIMULATED IN
486             TEXT-DATA,NOT IN CHARACTER-OR MICROPROGRAM-DATA;
487         'BEGIN' 'INTEGER' Y, Z;
488             Y:= READ DTAPE;
489             'FOR' Z:= READ DTAPE 'WHILE' ¬ ((Y = CUAK) ∧ (Z = CSTART))
490             'DO' Y:= Z
491         'END' UBL;
492
493         'PROCEDURE' WID (BYTE1, BYTE2, MIC, BOOLWID);
494         'INTEGER' BYTE1, BYTE2; 'BOOLEAN' BOOLWID, MIC;
495         'BEGIN' 'INTEGER' N, M, PA;
496             'IF' BOOLWID
497             'THEN' ERROR
498                    ("REPEATED MICROPROGRAM CONTAINS WID-INSTRUCTION",
499                     'TRUE', 'FALSE', 'TRUE');
500             BOOLWID:= 'TRUE'; N:= BYTE1 + 256 * BYTE2;
501             PA:= RDTOC (MIC);
502             'IF' PA < 1 ∨ PA = 28 ∨ PA = 62 ∨ PA = 251 ∨ PA > 254
503             'THEN' ERROR
504                    ("INSTRUCTION NOT ALLOWED AS WORD-ADDRESS", MIC,
505                     'FALSE', 'TRUE');
506             'FOR' M:= 0 'STEP' 1 'UNTIL' N
507             'DO' CALL TYPEFOUNT ELEMENT
508                  (PA, PLOTTER, CELLHEIGHT, CELLWIDTH, SHADING,
509                   REMAINDER, GRIDTYPE, KURN);
510             BOOLWID:= 'FALSE'
511         'END' WID;
512
513         'PROCEDURE' ZWQ (BYTE1, BYTE2, ZWQ QUANTIFICATION, BOOLZWQ);
514         'INTEGER' BYTE1, BYTE2, ZWQ QUANTIFICATION; 'BOOLEAN' BOOLZWQ;
515         'BEGIN'
516             ZWQ QUANTIFICATION:= 3 * RDTOC (MIC) + 768 * RDTOC (MIC);
517             BOOLZWQ:= 'TRUE';
518         'END' ZWQ;
519
520         'PROCEDURE' ZWR (PLOTTER, BOOLZWQ, ZWQ QUANTIFICATION);
521         'BOOLEAN' PLOTTER, BOOLZWQ; 'INTEGER' ZWQ QUANTIFICATION;
522         'BEGIN' 'INTEGER' R;
523             'IF' ¬ BOOLZWQ
524             'THEN' ERROR
525                    ("ZWR-INSTRUCTION BEFORE FIRST ZWQ-INSTRUCTION",
526                     MIC, 'FALSE', 'TRUE');
527             'IF' PLOTTER
528             'THEN' PLOTRELATIVE
529                    ('FALSE', ZWQ QUANTIFICATION, 0, - 2, 0)
530             'ELSE' PRINTRELATIVE
531                    ('FALSE', 'FALSE', ZWQ QUANTIFICATION / 12, 0,
532                     LINENUMBERMAX, 'FALSE')
533         'END' ZWR;
534
535         'PROCEDURE' RUCK (BYTE1, BYTE2, PLOTTER);
```

```
536           'INTEGER' BYTE1, BYTE2; 'BOOLEAN' PLOTTER;
537           'BEGIN' 'INTEGER' RU;
538               RU:= 3 * BYTE1 + 768 * BYTE2;
539               'IF' PLOTTER
540               'THEN' PLOTRELATIVE ('FALSE', - RU, 0, - 2, 0)
541               'ELSE' PRINTRELATIVE
542                      ('FALSE', 'FALSE', - RU / 12, 0, LINENUMBERMAX,
543                       'FALSE')
544           'END' RUCK;
545
546           'PROCEDURE' VERO (BYTE1, BYTE2, PLOTTER);
547           'INTEGER' BYTE1, BYTE2; 'BOOLEAN' PLOTTER;
548           'BEGIN' 'REAL' VE;
549               VE:= 4,6875 * BYTE1 + 1200 * BYTE2;
550               'IF' PLOTTER 'THEN' PLOTRELATIVE ('FALSE', 0, VE, - 2, 0)
551               'ELSE' PRINTRELATIVE
552                      ('FALSE', 'FALSE', 0, VE / 10, LINENUMBERMAX,
553                       'FALSE')
554           'END' VERO;
555
556           'PROCEDURE' VERU (BYTE1, BYTE2, PLOTTER);
557           'INTEGER' BYTE1, BYTE2; 'BOOLEAN' PLOTTER;
558           'BEGIN' 'REAL' VE; 'INTEGER' R;
559               VE:= 4,6875 * BYTE1 + 1200 * BYTE2;
560               'IF' PLOTTER
561               'THEN' PLOTRELATIVE ('FALSE', 0, - VE, - 2, 0)
562               'ELSE' PRINTRELATIVE
563                      ('FALSE', 'FALSE', 0, - VE / 10, LINENUMBERMAX,
564                       'FALSE')
565           'END' VERU;
566
567           'PROCEDURE' VERA (PLOTTER); 'BOOLEAN' PLOTTER;
568           'BEGIN'
569               'IF' PLOTTER
570               'THEN' PLOTRELATIVE
571                      ('FALSE', 0, 1375 - PLOT (0, 0, 16), - 2, 0)
572               'ELSE' PRINTRELATIVE
573                      ('FALSE', 'TRUE', 0, 73, LINENUMBERMAX, 'FALSE')
574           'END' VERA;
575
576           'PROCEDURE' TAB (BYTE1, BYTE2, TAB QUANTIFICATION, MIC);
577           'INTEGER' BYTE1, BYTE2, TAB QUANTIFICATION; 'BOOLEAN' MIC;
578           TAB QUANTIFICATION:= 3 * RDTOC (MIC) + 768 * RDTOC (MIC);
579
580           'PROCEDURE' ZER (BYTE1, BYTE2, TAB QUANTIFICATION, PLOTTER);
581           'INTEGER' BYTE1, BYTE2, TAB QUANTIFICATION; 'BOOLEAN' PLOTTER;
582           'BEGIN' 'REAL' ZE;
583               ZE:= 4,6875 * BYTE1 + 1200 * BYTE2;
584               'IF' PLOTTER 'THEN'
585               'BEGIN' PLOTRELATIVE ('FALSE', 0, + ZE, - 2, 0);
586                   PLOTRELATIVE
587                   ('FALSE', TAB QUANTIFICATION - PLOT (0, 0, 15), 0,
588                    - 2, 0)
589               'END'
590               'ELSE' PRINTRELATIVE
591                      ('TRUE', 'FALSE', TAB QUANTIFICATION / 12 + 1,
592                       ZE / 10, LINENUMBERMAX, 'FALSE')
593           'END' ZER;
594
595           'PROCEDURE' ZEV (BYTE1, BYTE2, TAB QUANTIFICATION, PLOTTER);
```

```
596         'INTEGER' BYTE1, BYTE2, TAB QUANTIFICATION; 'BOOLEAN' PLOTTER;
597         'BEGIN' 'REAL' ZE;
598             ZE:= 4,6875 * BYTE1 + 1200 * BYTE2;
599             'IF' PLOTTER 'THEN'
600             'BEGIN' PLOTRELATIVE ('FALSE', 0, - ZE, - 2, 0);
601                 PLOTRELATIVE
602                 ('FALSE', TAB QUANTIFICATION - PLOT (0, 0, 15), 0,
603                  - 2, 0)
604             'END'
605             'ELSE' PRINTRELATIVE
606                    ('TRUE', 'FALSE', TAB QUANTIFICATION / 12 + 1,
607                     - ZE / 10, LINENUMBERMAX, 'FALSE')
608         'END' ZEV;
609
610         'PROCEDURE' INPCORE (SINGLE CHARACTER);
611         'BOOLEAN' SINGLE CHARACTER;
612         'BEGIN' 'INTEGER' SECSEG, SECPA, BYTE, R, S, T, U, V, W, N;
613
614             'PROCEDURE' STOREC (X); 'INTEGER' X;
615             'BEGIN'
616                 'IF' SECSEG 'LE' CORESEGMENTS
617                 'THEN' CORE [SECSEG, SECPA, BYTE]:= X
618                 'ELSE' ERROR
619                        ("TYPEF.ELEMENT INTO NON-EXISTENT CORE-SEGM.",
620                        MIC, 'FALSE', 'TRUE');
621                 'IF' BYTE < 3 'THEN' BYTE:= BYTE + 1 'ELSE'
622                 'IF' SECPA < 255 'THEN'
623                 'BEGIN' SECPA:= SECPA + 1; BYTE:= 1 'END'
624                 'ELSE'
625                 'BEGIN' SECSEG:= SECSEG + 1; SECPA:= 0; BYTE:= 1
626                 'END'
627             'END' STOREC;
628
629             SEGMF:= SEGM; 'IF' SEGM = 0 'THEN' BUFFERSEG:= 'FALSE';
630             'IF' PA = 62 ∧ ¬ SINGLE CHARACTER 'THEN'
631             'BEGIN' SECSEG:= SEGM; SECPA:= 62; BYTE:= 1;
632                 R:= RDTOD (BOOL RDDISC); S:= RDTOD (BOOL RDDISC);
633                 T:= RDTOD (BOOL RDDISC);
634                 'IF' ¬ BOOL RDDISC 'THEN'
635                 'BEGIN'
636                     'IF' ¬ ((0 'LE' R ∧ R 'LE' 19) ∨
637                           (32 'LE' R ∧ R 'LE' 51) ∨
638                           (64 'LE' R ∧ R 'LE' 83))
639                     'THEN' ERROR
640                            ("BYTE ONE TYPEF.CONST. NON ALLOWED VALUE",
641                            'FALSE', 'FALSE', 'TRUE');
642                     'IF' S 'NE' 0
643                     'THEN' ERROR
644                            ("SEC.BYTE TYPEFACECONST.NOT EQUAL ZERO",
645                             'FALSE', 'FALSE', 'TRUE');
646                     'IF' T 'NE' 0
647                     'THEN' ERROR
648                            ("THIRD BYTE TYPEFACECONST.NOT EQUAL ZERO",
649                             'FALSE', 'FALSE', 'TRUE')
650                 'END';
651                 STOREC (R); STOREC (S); STOREC (T);
652                 'IF' ¬ ((RDTOD (BOOL RDDISC) = CUAT) ∧
653                       (RDTOD (BOOL RDDISC) = CUAT))
654                 'THEN' ERROR
655                        ("TYPEFACECONSTANT NOT FOLLOWED BY UAT UAT",
```

```
656                         'FALSE', 'FALSE', 'TRUE')
657                 'END'
658                 'ELSE'
659                 'BEGIN' BYTE:= 1;
660                     'IF' SINGLE CHARACTER 'THEN'
661                     'BEGIN' SECPA:= 0; SECSEG:= SEGM + 1;  'END'
662                     'ELSE'
663                     'BEGIN'
664                         CORE [SEGM, PA, 1]:= SECPA:= RDTOD (BOOL RDDISC);
665                         CORE [SEGM, PA, 2]:= RDTOD (BOOL RDDISC);
666                         SECSEG:= CORE [SEGM, PA, 2] + SEGM;
667                         'IF' SECSEG > CORESEGMENTS
668                         'THEN' ERROR
669                                ("TYPEF.EL, INTO NON-EXISTENT CORE-SEGM",
670                                MIC, 'FALSE', 'TRUE');
671                         CORE [SEGM, PA, 3]:= THIRDBYTE:=
672                         RDTOD (BOOL RDDISC);
673                         'IF' ¬ BOOL RDDISC 'THEN'
674                         'BEGIN'
675                             'IF' ¬ (THIRDBYTE = 0 ∨ THIRDBYTE = 1 ∨
676                                  THIRDBYTE = 2 ∨ THIRDBYTE = 32)
677                             'THEN' ERROR
678                                    ("THIRDBYTE SEC.ADDRESS-WORD", 'FALSE',
679                                     'FALSE', 'TRUE');
680                             'IF' SECSEG = SEGM ∧ SECPA < 63 'THEN'
681                             'BEGIN' INSTR NUMBER:= INSTR NUMBER - 2;
682                                 ERROR
683                                 ("IMAGE-AREA BELOW TYPEFACECONSTANT",
684                                  'FALSE', 'FALSE', 'TRUE')
685                             'END'
686                         'END'
687                     'END';
688                     'IF' THIRDBYTE = 1 ∨ THIRDBYTE = 0 ∨ THIRDBYTE = 32
689                     'THEN'
690                     'BEGIN' R:= RDTOD (BOOL RDDISC); BOOLINPCORE:= 'TRUE';
691                         'FOR' S:=RDTOD(BOOL RDDISC)
692                               'WHILE' ¬ ((R = CUAT) ∧ (S = CUAT))
693                         'DO' 'BEGIN' STOREC (R); R:= S 'END';
694                         STOREC (R); BOOLINPCORE:= 'FALSE';
695                         'IF' ¬ BOOL RDDISC
696                         'THEN' INSTR NUMBER:= INSTR NUMBER + 2;
697                         'IF' RDTOD (BOOL RDDISC) = CUAT
698                         'THEN' STOREC (CUAT)
699                         'ELSE'
700                         'IF' BOOL RDDISC 'THEN'
701                         'BEGIN'
702                             'IF' SECTBYTE > 1
703                             'THEN' SECTBYTE:= SECTBYTE - 1
704                             'ELSE'
705                             'BEGIN' SECTBYTE:= 744;
706                                 SECTADDR:= SECTADDR - 1
707                             'END'
708                         'END'
709                         'ELSE'
710                         'BEGIN' BYTENUMBER INPUT:= BYTENUMBER INPUT - 1;
711                             INSTR NUMBER:= INSTR NUMBER - 1;
712                         'END'
713                     'END'
714                     'ELSE'
715                     'BEGIN' R:= RDTOD (BOOL RDDISC);
```

```
716                     'FOR' S:=
717                           RDTOD (BOOL RDDISC)
718                           'WHILE' ¬ R = CUAT ∨ ¬ S = CUAT
719                     'DO'
720                     'BEGIN'
721                         'IF' R = CUAK ∧ S = CPLA 'THEN'
722                         'BEGIN' STOREC (R); STOREC (S);
723                             'FOR' N:= 1 'STEP' 1 'UNTIL' 4
724                             'DO' STOREC (RDTOD (BOOL RDDISC));
725                             R:= RDTOD (BOOL RDDISC);
726                         'END'
727                         'ELSE' 'BEGIN' STOREC (R); R:= S 'END'
728                     'END';
729                     STOREC (R)
730                 'END'
731             'END'
732         'END' INPCORE;
733
734         'PROCEDURE' SPE (MIC); 'BOOLEAN' MIC;
735         'BEGIN'
736
737             'PROCEDURE' PLA (MIC); 'BOOLEAN' MIC;
738             'BEGIN'
739                 'INTEGER' BYTE1 SECTADDR, BYTE2 SECTADDR,
740                     REMEMBERSECTADDR, R, S, N;
741                 'BOOLEAN' SINGLE CHARACTER;
742                 SINGLE CHARACTER:= 'FALSE';
743                 BYTE1 SECTADDR:= RDTOC (MIC);
744                 BYTE2 SECTADDR:= RDTOC (MIC);
745                 'IF' BYTE2 SECTADDR > 127 'THEN'
746                 'BEGIN'
747                     'IF' SEGM = CORESEGMENTS
748                     'THEN' ERROR
749                             ("SINGLE-CHAR.TRANSFER TO LAST CORE-SEGM.",
750                             MIC, 'FALSE', 'TRUE');
751                     BYTE2 SECTADDR:= BYTE2 SECTADDR - 128;
752                     SINGLE CHARACTER:= 'TRUE'
753                 'END';
754                 REMEMBERSECTADDR:= SECTADDR:=
755                 BYTE1 SECTADDR + 256 * BYTE2 SECTADDR;
756                 'IF' SECTADDR > DISCSECTORS
757                 'THEN' ERROR
758                         ("NON-EXISTENT DISC-SECTOR ADDRESSED", MIC,
759                          'FALSE', 'TRUE');
760                 'IF' ¬ (RDTOC (MIC) = CUAT) ∨ ¬ (RDTOC (MIC) = CUAT)
761                 'THEN'
762                 'BEGIN'
763                     'IF' MIC
764                     'THEN' INSTR NUMBER MIC:= INSTR NUMBER MIC - 1
765                     'ELSE' INSTR NUMBER:= INSTR NUMBER - 1;
766                     ERROR
767                     ("PLA NOT FOLLOWED BY UAT;UAT;", MIC, 'FALSE',
768                      'TRUE')
769                 'END';
770                 SECTBYTE:= 3; BOOL RDDISC:= 'TRUE';
771                 'IF' SINGLECHARACTER 'THEN'
772                 'BEGIN' 'IF' SEGM = 0 'THEN' BUFFERSEG:= 'FALSE';
773                     SECTBYTE:= 6;
774                     'FOR' N:= 1, 2, 3
775                     'DO' CORE [SEGM, 62, N]:= READ DISC;
```

```
776                     'COMMENT' TYPEFACECONSTANT;
777                     'FOR' N:= 1, N + 1 'WHILE' BOOL RDDISC 'DO'
778                     'BEGIN' R:= READ DISC;
779                         'FOR' S:=READ DISC
780                               'WHILE' ¬ (R = CUAK ∧ S = CSPE)
781                         'DO' R:= S;
782                         R:= READ DISC;
783                         'IF' ¬ R = CUAK 'THEN'
784                         'BEGIN'
785                             'IF' REMEMBERSECTADDR = SECTADDR
786                             'THEN' CORE [SEGM, R, 1]:= SECTADDR
787                             'ELSE' CORE [SEGM, R, 1]:= SECTADDR - 1;
788                             CORE [SEGM, R, 2]:= - 1;
789                         'END'
790                         'ELSE' 'IF' READ DISC = CTRE 'THEN' TRE
791                     'END'
792                 'END'
793                 'ELSE'
794                 'FOR' S:= READ DISC 'WHILE' BOOL RDDISC
795                 'DO' INTERPRET (S, 'FALSE', 'TRUE', PLOTTER)
796             'END' PLA;
797
798             'PROCEDURE' PLE;
799             'BEGIN' 'INTEGER' SECTBYTES;
800
801                 'PROCEDURE' STORED (X);
802                 'BEGIN'
803                     'IF' SECTADDR > DISCSECTORS
804                     'THEN' ERROR
805                             ("NON-EXISTENT DISC-SECTOR ADDRESSED",
806                              'FALSE', 'FALSE', 'TRUE');
807                     DISC [SECTADDR, SECTBYTES]:= X;
808                     'IF' SECTBYTES < 744
809                     'THEN' SECTBYTES:= SECTBYTES + 1
810                     'ELSE'
811                     'BEGIN' SECTADDR:= SECTADDR + 1; SECTBYTES:= 1;
812                     'END'
813                 'END' STORED;
814
815                 'PROCEDURE' CHECK;
816                 'BEGIN' R:= READ DTAPE; S:= READ DTAPE;
817                     'IF' ¬ (R = CUAK) ∨ (¬ (S = CSPE))
818                     'THEN' ERROR
819                             ("TYPEF.EL. DOES NOT BEGIN WITH UAK SPE",
820                              MIC, 'FALSE', 'TRUE')
821                     'ELSE' 'BEGIN' STORED (R); STORED (S);  'END'
822                 'END';
823
824                 'PROCEDURE' ELEMENT ONTO DISC;
825                 'BEGIN' 'INTEGER' R, S, T, N; 'BOOLEAN' MICROPROGRAM;
826
827                     'PROCEDURE' UATUAT (MICROPROGRAM);
828                     'BOOLEAN' MICROPROGRAM;
829                     'BEGIN' 'INTEGER' R, S, T, U;
830                         R:= READ DTAPE; STORED (R);
831                         'IF' ¬ R = CUAT 'THEN' UATUAT (MICROPROGRAM)
832                         'ELSE'
833                         'BEGIN' R:= READ DTAPE; STORED (R);
834                             'IF' ¬ R = CUAT
835                             'THEN' UATUAT (MICROPROGRAM)
```

```
836                             'ELSE'
837                             'BEGIN'
838                                 'IF' ¬ MICROPROGRAM 'THEN'
839                                 'BEGIN'
840                                     'IF' READ DTAPE = CUAT
841                                     'THEN' STORED (CUAT)
842                                     'ELSE' BYTENUMBER INPUT:=
843                                             BYTENUMBER INPUT - 1;
844                                     BOOL CHAR TO DISC:= 'FALSE';
845                                     INSTR NUMBER:= INSTR NUMBER + 2;
846                                     ELEMENT ONTO DISC
847                                 'END'
848                                 'ELSE'
849                                 'BEGIN'
850                                     'IF' SECTBYTES > 6 'THEN'
851                                     'BEGIN' R:= T:= SECTADDR;
852                                         S:= SECTBYTES - 6;
853                                         U:= SECTBYTES - 5;
854                                     'END'
855                                     'ELSE'
856                                     'BEGIN' R:= SECTADDR - 1;
857                                         S:= 744 + SECTBYTES - 6;
858                                         'IF' SECTBYTES > 5 'THEN'
859                                         'BEGIN' T:= SECTADDR;
860                                             U:= SECTBYTES - 5
861                                         'END'
862                                         'ELSE'
863                                         'BEGIN' T:= SECTADDR - 1;
864                                             U:= 744 + SECTBYTES - 5
865                                         'END'
866                                     'END';
867                                     'IF' ¬ (DISC [T, U] = CPLA ∧
868                                           DISC [R, S] = CUAK)
869                                     'THEN' ELEMENT ONTO DISC
870                                     'ELSE' UATUAT (MICROPROGRAM)
871                                 'END'
872                             'END'
873                         'END'
874                     'END' UATUAT;
875
876                     CHECK; R:= READ DTAPE; STORED (R);
877                     'IF' R = CUAK 'THEN'
878                     'BEGIN' R:= READ DTAPE; S:= READ DTAPE;
879                         T:= READ DTAPE;
880                         'IF' ¬ (R = CTRE) ∨ ¬ (S = CUAT) ∨
881                              ¬ (T = CUAT)
882                         'THEN' ERROR
883                                 ("SPE UAK NOT FOLLOWED BY TRE UAT UAT",
884                                 'FALSE', 'FALSE', 'TRUE')
885                         'ELSE'
886                         'BEGIN' STORED (R); STORED (S); STORED (T)
887                         'END'
888                     'END'
889                     'ELSE'
890                     'BEGIN'
891                         'IF' R < 1 ∨ R = 28 ∨ R = 62 ∨ R = 251 ∨
892                              R > 254
893                         'THEN' ERROR
894                                 ("INSTR. NOT ALLOWED AS WORD-ADDRESS",
895                                 'FALSE', 'FALSE', 'TRUE');
```

```
896                         R:= READ DTAPE;
897                         STORED (R); S:= READ DTAPE;
898                         'IF' S = 0 ∧ R < 63
899                         'THEN' ERROR
900                                ("IMAGE-AREA BELOW TYPEFACECONSTANT",
901                                 'FALSE', 'FALSE', 'TRUE');
902                         'IF' S > CORESEGMENTS
903                         'THEN' ERROR
904                                ("REL.SEG.ADDRESS TOO LARGE",
905                                 'FALSE', 'FALSE', 'TRUE');
906                         STORED (S); T:= READ DTAPE;
907                         'IF' ¬ (T = 0 ∨ T = 1 ∨ T = 2 ∨ T = 32)
908                         'THEN' ERROR
909                                ("THIRDBYTE SECONDARY ADDRESS-WORD",
910                                 'FALSE', 'FALSE', 'TRUE');
911                         STORED (T); MICROPROGRAM:= T = 2;
912                         BOOL CHAR TO DISC:= ¬ MICROPROGRAM;
913                         'IF' BOOL CHAR TO DISC
914                         'THEN' INSTR NUMBER:= INSTR NUMBER + 1;
915                         UATUAT (MICROPROGRAM)
916                     'END'
917                 'END' ELEMENT ONTO DISC;
918
919                 SECTADDR:= READ DTAPE + 256 * READ DTAPE;
920                 'IF' SECTADDR > DISCSECTORS
921                 'THEN' ERROR
922                        ("NON-EXISTENT DISC-SECTORADDRESSED", 'FALSE',
923                         'FALSE', 'TRUE');
924                 SECTBYTES:= 1; BUFFERSEG:= 'TRUE';
925                 'FOR' N:= 1, 2 'DO'
926                 'BEGIN' R:= READ DTAPE;
927                     'IF' R = CUAT 'THEN' STORED (R)
928                     'ELSE' ERROR
929                            ("PLE NOT FOLLOWED BY UAT UAT", 'FALSE',
930                             'FALSE', 'TRUE')
931                 'END';
932                 CHECK; R:= READ DTAPE;
933                 'IF' R = 62 'THEN'
934                 'BEGIN' STORED (R); R:= READ DTAPE; S:= READ DTAPE;
935                     T:= READ DTAPE;
936                     'IF' ¬ ((0 'LE' R ∧ R 'LE' 19) ∨
937                           (32 'LE' R ∧ R 'LE' 51) ∨
938                           (64 'LE' R ∧ R 'LE' 83))
939                     'THEN' ERROR
940                            ("BYTE ONE TYPEF.CONST.NON-ALLOWED VALUE",
941                             'FALSE', 'FALSE', 'TRUE');
942                     'IF' S 'NE' 0
943                     'THEN' ERROR
944                            ("SEC.BYTE TYPEFACECONST.NOT EQUAL ZERO",
945                             'FALSE', 'FALSE', 'TRUE');
946                     'IF' T 'NE' 0
947                     'THEN' ERROR
948                            ("THIRD BYTE TYPEFACECONST.NOT EQUAL ZERO",
949                             'FALSE', 'FALSE', 'TRUE');
950                     STORED (R); STORED (S); STORED (T);
951                 'END'
952                 'ELSE' ERROR
953                        ("TYPEFOUNT WITHOUT TYPEFACE-CONSTANT",
954                        'FALSE', 'FALSE', 'TRUE');
955             'FOR' N:= 1, 2 'DO'
```

```
956                 'BEGIN' R:= READ DTAPE;
957                     'IF' R = CUAT 'THEN' STORED (R)
958                     'ELSE' ERROR
959                             ("TYPEFACECONSTANT NOT FOLLOWED BY UAT UAT",
960                              'FALSE', 'FALSE', 'TRUE')
961                 'END';
962                 ELEMENT ONTO DISC;
963             'END' PLE;
964
965             'PROCEDURE' TRE; BOOL RDDISC:= 'FALSE';
966
967             PA:= RDTODOC (BOOL RDDISC, MIC);
968             'IF' PA = CUAK 'THEN'
969             'BEGIN' R:= RDTODOC (BOOL RDDISC, MIC);
970                 'IF' R = CPLA 'THEN' PLA (MIC) 'ELSE'
971                 'IF' MIC
972                 'THEN' ERROR
973                         ("INSTRUCTION NOT PERMITTED IN MICROPROGRAM",
974                          'TRUE', 'FALSE', 'TRUE')
975                 'ELSE'
976                 'IF' R = CPLE 'THEN' PLE 'ELSE'
977                 'IF' R = CTRE 'THEN' TRE
978                 'ELSE' ERROR
979                         ("INSTRUCTION FOLLOWING SPE NOT PERMITTED",
980                          MIC, 'FALSE', 'TRUE')
981             'END'
982             'ELSE'
983             'IF' MIC
984             'THEN' ERROR
985                    ("INSTRUCTION NOT PERMITTED IN MICROPROGRAM",
986                     'TRUE', 'FALSE', 'TRUE')
987             'ELSE'
988             'IF' PA < 1 ∨ PA = 28 ∨ PA = 251 ∨ PA > 254
989             'THEN' ERROR
990                    ("INSTRUCTION NOT ALLOWED AS WORD-ADDRESS",
991                    MIC, 'FALSE', 'TRUE')
992             'ELSE' INPCORE ('FALSE')
993         'END' SPE;
994
995         'PROCEDURE' CALL TYPEFOUNT ELEMENT
996         (PRIMARY ADDRESS, PLOTTER, CELLHEIGHT, CELLWIDTH, SHADING,
997          REMAINDER, GRIDTYPE, KURN);
998         'INTEGER' PRIMARY ADDRESS, SHADING, REMAINDER, GRIDTYPE, KURN;
999         'REAL' CELLWIDTH, CELLHEIGHT; 'BOOLEAN' PLOTTER;
1000        'BEGIN'
1001            'INTEGER' B0, B1, B2, B3, B4, B5, B6, B7, INDEX, REPEAT,
1002                SECPAMIC, SECSEGMIC, BYTEMIC, LINESTART, N, R, S, T;
1003            'REAL' YPOSITION;
1004            'BOOLEAN' SINGLE BYTE, LINE REPETITION, BOOLUAT,
1005                FIRST WHITE ELEMENT;
1006
1007            'INTEGER' 'PROCEDURE' READ BYTE;
1008            'BEGIN' 'INTEGER' N, M; 'INTEGER' 'ARRAY' BB [0 : 7];
1009                M:= READ CORE;
1010                'FOR' N:= 7 'STEP' - 1 'UNTIL' 0 'DO'
1011                'BEGIN' BB [N]:= ENTIER (M / 10 ** N);
1012                    'IF' BB [N] 'NE' 0 ∧ BB [N] 'NE' 1
1013                    'THEN' ADDRESS ERR;
1014                    M:= M - BB [N] * 10 ** N
1015                'END';
```

```
1016                    'IF' BB [7] = 0 'THEN' B7:= 0 'ELSE' B7:= 128;
1017                    'IF' BB [6] = 0 'THEN' B6:= 0 'ELSE' B6:= 64;
1018                    'IF' BB [5] = 0 'THEN' B5:= 0 'ELSE' B5:= 32;
1019                    'IF' BB [4] = 0 'THEN' B4:= 0 'ELSE' B4:= 16;
1020                    'IF' BB [3] = 0 'THEN' B3:= 0 'ELSE' B3:= 8;
1021                    'IF' BB [2] = 0 'THEN' B2:= 0 'ELSE' B2:= 4;
1022                    'IF' BB [1] = 0 'THEN' B1:= 0 'ELSE' B1:= 2;
1023                    'IF' BB [0] = 0 'THEN' B0:= 0 'ELSE' B0:= 1;
1024                    READ BYTE:= B0 + B1 + B2 + B3 + B4 + B5 + B6 + B7;
1025                'END' READ BYTE;
1026
1027                'INTEGER' 'PROCEDURE' READ LINE ELEMENT;
1028                'BEGIN' 'INTEGER' BB;
1029                    BB:= B5 + B4 + B3 + B2 + B1 + B0;
1030                    'IF' SINGLE BYTE 'THEN' READ LINE ELEMENT:= BB + B6
1031                    'ELSE'
1032                    'IF' B6 = 64 'THEN' READ LINE ELEMENT:= BB
1033                    'ELSE' READ LINE ELEMENT:= READ BYTE * 64 + BB
1034                'END' READ LINE ELEMENT;
1035
1036                'PROCEDURE' LBL (ONE BYTE, CELLWIDTH); 'BOOLEAN' ONE BYTE;
1037                'REAL' CELLWIDTH;
1038                'BEGIN' 'INTEGER' R, S;
1039                    SINGLE BYTE:= B0 = 0;
1040                    'IF' B6 = 64 'THEN' S:= - 1 'ELSE' S:= 1;
1041                    'IF' ONE BYTE 'THEN' R:= (B5 + B4 + B3) / 8
1042                    'ELSE' R:= READ BYTE;
1043                    'IF' R > 0 'THEN' LINECOUNT:= LINECOUNT + 1;
1044                    'IF' PLOTTER
1045                    'THEN' PLOTRELATIVE
1046                           ('FALSE', CELLWIDTH * R * S, 0, - 2, 0)
1047                    'ELSE' PRINTRELATIVE
1048                           ('FALSE', 'FALSE', R * S, 0, LINENUMBERMAX,
1049                            'FALSE')
1050                'END' LBL;
1051
1052                'PROCEDURE' BLW (ONE BYTE, REPEAT); 'INTEGER' REPEAT;
1053                'BOOLEAN' ONE BYTE;
1054                'BEGIN' SINGLE BYTE:= B0 = 0;
1055                    'IF' ONE BYTE 'THEN' REPEAT:= (B6 + B5 + B4 + B3) / 8
1056                    'ELSE' REPEAT:= READ BYTE;
1057                    LINE REPETITION:= 'TRUE'
1058                'END' BLW;
1059
1060                'PROCEDURE' IMAGE LINE
1061                (CELLWIDTH, CELLHEIGHT, SHADING, REMAINDER, REPEAT,
1062                 YPOSITION, PLOTTER, LINE REPETITION, KURN);
1063                'INTEGER' SHADING, REMAINDER, REPEAT, KURN;
1064                'REAL' CELLWIDTH, CELLHEIGHT, YPOSITION;
1065                'BOOLEAN' PLOTTER, LINE REPETITION;
1066                'BEGIN' 'INTEGER' INDEX, N; 'REAL' YTOTAL, XPOSITION, Y;
1067                    'BOOLEAN' WHITE ELEMENT;
1068                    'INTEGER' 'ARRAY' REPEATED LINE [1 : 276];
1069
1070                    'PROCEDURE' STORE REPEATED LINE (X); 'INTEGER' X;
1071                    'BEGIN' REPEATED LINE [INDEX]:= X; INDEX:= INDEX + 1
1072                    'END';
1073
1074                    'PROCEDURE' WHITE
1075                    (Y, CELLHEIGHT, PLOTTER, FIRST WHITE ELEMENT, KURN);
```

```
1076                    'VALUE' Y; 'REAL' Y, CELLHEIGHT;
1077                    'BOOLEAN' PLOTTER, FIRST WHITE ELEMENT;
1078                    'INTEGER' KURN;
1079                    'BEGIN'
1080                        'IF' FIRST WHITE ELEMENT 'THEN'
1081                        'BEGIN' Y:= Y - 1; FIRST WHITE ELEMENT:= 'FALSE'
1082                        'END';
1083                        'IF' PLOTTER 'THEN'
1084                        'BEGIN' Y:= Y * CELLHEIGHT; YTOTAL:= YTOTAL + Y;
1085                            PLOTRELATIVE ('FALSE', 0, Y, - 2, KURN)
1086                        'END'
1087                        'ELSE' PRINTRELATIVE
1088                                ('FALSE', 'FALSE', 0, Y, LINENUMBERMAX,
1089                                 'FALSE')
1090                    'END' WHITE;
1091
1092                    'PROCEDURE' BLACK
1093                    (Y, SHADING, REMAINDER, XPOSITION, CELLHEIGHT,
1094                     PLOTTER, KURN);
1095                    'VALUE' Y; 'REAL' Y, XPOSITION, CELLHEIGHT;
1096                    'INTEGER' SHADING, REMAINDER, KURN; 'BOOLEAN' PLOTTER;
1097                    'BEGIN' 'INTEGER' N;
1098                        'IF' PLOTTER 'THEN'
1099                        'BEGIN' Y:= Y * CELLHEIGHT; YTOTAL:= YTOTAL + Y;
1100                            PLOTRELATIVE ('FALSE', 1, 0, - 1, KURN);
1101                            'FOR' N:= 1 'STEP' 1 'UNTIL' SHADING 'DO'
1102                            'BEGIN'
1103                                PLOTRELATIVE ('FALSE', 0, Y, - 1, KURN);
1104                                PLOTRELATIVE ('FALSE', 2, 0, - 1, KURN);
1105                                PLOTRELATIVE ('FALSE', 0, - Y, - 1, KURN);
1106                                PLOTRELATIVE ('FALSE', 2, 0, - 1, KURN)
1107                            'END';
1108                            'IF' REMAINDER = 0 'THEN'
1109                            'BEGIN'
1110                                PLOTRELATIVE ('FALSE', 0, Y, - 2, KURN);
1111                                PLOTRELATIVE
1112                                ('FALSE', - 4 * SHADING - 1, 0, - 2, 0)
1113                            'END'
1114                            'ELSE'
1115                            'IF' REMAINDER = 1 'THEN'
1116                            'BEGIN'
1117                                PLOTRELATIVE ('FALSE', - 1, 0, - 1, 0);
1118                                PLOTRELATIVE ('FALSE', 0, Y, - 1, KURN);
1119                                PLOTRELATIVE
1120                                ('FALSE', - 4 * SHADING, 0, - 2, 0)
1121                            'END'
1122                            'ELSE'
1123                            'IF' REMAINDER = 2 'THEN'
1124                            'BEGIN'
1125                                PLOTRELATIVE ('FALSE', 0, Y, - 1, KURN);
1126                                PLOTRELATIVE
1127                                ('FALSE', - 4 * SHADING - 1, 0, - 2, 0)
1128                            'END'
1129                            'ELSE'
1130                            'IF' REMAINDER = 3 'THEN'
1131                            'BEGIN'
1132                                PLOTRELATIVE ('FALSE', 0, Y, - 1, KURN);
1133                                PLOTRELATIVE ('FALSE', 1, 0, - 1, 0);
1134                                PLOTRELATIVE ('FALSE', 0, - Y, - 1, KURN);
1135                                PLOTRELATIVE ('FALSE', 0, Y, - 2, KURN);
```

```
1136                                PLOTRELATIVE
1137                                ('FALSE', - 4 * SHADING - 2, 0, - 2, 0)
1138                            'END'
1139                        'END'
1140                        'ELSE'
1141                        'FOR' N:= 1 'STEP' 1 'UNTIL' Y
1142                        'DO' PRINTRELATIVE
1143                             ('FALSE', 'FALSE', 0, 1, LINENUMBERMAX,
1144                              'TRUE')
1145                    'END' BLACK;
1146
1147                    'PROCEDURE' CHARACTER BASE
1148                    (PLOTTER, YPOSITION, YTOTAL, KURN);
1149                    'REAL' YPOSITION, YTOTAL; 'INTEGER' KURN;
1150                    'BOOLEAN' PLOTTER;
1151                    'BEGIN'
1152                        'IF' PLOTTER 'THEN'
1153                        'BEGIN'
1154                            PLOTRELATIVE
1155                            ('FALSE', 0, YPOSITION - YTOTAL, - 2, KURN);
1156                            PLOTRELATIVE ('FALSE', CELLWIDTH, 0, - 2, 0)
1157                        'END'
1158                        'ELSE' PRINTRELATIVE
1159                               ('FALSE', 'TRUE', 1, YPOSITION,
1160                                LINENUMBERMAX, 'FALSE');
1161                        FIRST WHITE ELEMENT:= 'TRUE';
1162                    'END' CHARACTER BASE;
1163
1164                    'PROCEDURE' BLE
1165                    (CELLWIDTH, CELLHEIGHT, SHADING, REMAINDER, REPEAT,
1166                     YPOSITION, YTOTAL, PLOTTER, LINE REPETITION, KURN);
1167                    'INTEGER' SHADING, REMAINDER, REPEAT, KURN;
1168                    'REAL' CELLWIDTH, CELLHEIGHT, YPOSITION, YTOTAL;
1169                    'BOOLEAN' PLOTTER, LINE REPETITION;
1170                    'BEGIN' 'INTEGER' N; 'BOOLEAN' WHITE ELEMENT;
1171
1172                        'INTEGER' 'PROCEDURE' READ REPEATED LINE;
1173                        'BEGIN'
1174                            READ REPEATED LINE:= REPEATED LINE [INDEX];
1175                            INDEX:= INDEX + 1
1176                        'END';
1177
1178                        R:= READ LINE ELEMENT;
1179                        'IF' LINE REPETITION 'THEN'
1180                        'BEGIN' STORE REPEATED LINE (R);
1181                            STORE REPEATED LINE (- 1)
1182                        'END';
1183                        BLACK
1184                        (R, SHADING, REMAINDER, XPOSITION, CELLHEIGHT,
1185                         PLOTTER, KURN);
1186                        CHARACTER BASE (PLOTTER, YPOSITION, YTOTAL, KURN);
1187                        'IF' LINE REPETITION 'THEN'
1188                        'BEGIN'
1189                            'FOR' N:= 1 'STEP' 1 'UNTIL' REPEAT 'DO'
1190                            'BEGIN' INDEX:= 1; WHITE ELEMENT:= 'TRUE';
1191                                'IF' PLOTTER 'THEN'
1192                                'BEGIN' XPOSITION:= PLOT (0, 0, 15);
1193                                    YTOTAL:= YPOSITION
1194                                'END';
1195                                'FOR' R:= READ REPEATED LINE 'WHILE' R > 0
```

```
1196                                'DO'
1197                                'BEGIN'
1198                                    'IF' WHITE ELEMENT
1199                                    'THEN' WHITE
1200                                           (R, CELLHEIGHT, PLOTTER,
1201                                            FIRST WHITE ELEMENT, KURN)
1202                                    'ELSE' BLACK
1203                                           (R, SHADING, REMAINDER,
1204                                            XPOSITION, CELLHEIGHT,
1205                                            PLOTTER, KURN);
1206                                    WHITE ELEMENT:= ¬ WHITE ELEMENT;
1207                                'END';
1208                                CHARACTER BASE
1209                                (PLOTTER, YPOSITION, YTOTAL, KURN)
1210                            'END';
1211                            LINE REPETITION:= 'FALSE';
1212                        'END'
1213                    'END' BLE;
1214
1215                    LINECOUNT:= LINECOUNT + 1;
1216                    'IF' PLOTTER 'THEN'
1217                    'BEGIN' XPOSITION:= PLOT (0, 0, 15);
1218                        YTOTAL:= YPOSITION
1219                    'END';
1220                    WHITE ELEMENT:= 'FALSE';
1221                    'IF' LINE REPETITION 'THEN'
1222                    'FOR' INDEX:= 1 'STEP' 1 'UNTIL' 276
1223                    'DO' REPEATED LINE [INDEX]:= 0;
1224                    INDEX:= 1;
1225                    'FOR' N:= 1, N + 1 'WHILE' B7 = 0 'DO'
1226                    'BEGIN' WHITE ELEMENT:= ¬ WHITE ELEMENT;
1227                        Y:= READ LINE ELEMENT;
1228                        'IF' LINE REPETITION
1229                        'THEN' STORE REPEATED LINE (Y);
1230                        'IF' WHITE ELEMENT
1231                        'THEN' WHITE
1232                               (Y, CELLHEIGHT, PLOTTER,
1233                                FIRST WHITEELEMENT, KURN)
1234                        'ELSE' BLACK
1235                               (Y, SHADING, REMAINDER, XPOSITION,
1236                                CELLHEIGHT, PLOTTER, KURN);
1237                        READ BYTE;
1238                    'END';
1239                    BLE
1240                    (CELLWIDTH, CELLHEIGHT, SHADING, REMAINDER, REPEAT,
1241                     YPOSITION, YTOTAL, PLOTTER, LINE REPETITION, KURN)
1242                'END' IMAGE LINE;
1243
1244                'PROCEDURE' ADDRESS ERR;
1245                ERROR
1246                ("ADDRESSED CORE-AREA NOT LOADED WITH ERROR-FREE
1247                CHARACTER-DATA", MIC, 'FALSE', 'TRUE');
1248
1249                'IF' BUFFERSEG ^ SEGM = 0
1250                'THEN' ERROR
1251                       ("CHAR.CALL FROM SEG.0 WHICH HAS BEEN USED AS
1252                       BUFFER FOR DISC-INPUT", MIC, 'FALSE', 'TRUE');
1253                'IF' SEGM = - 1
1254                'THEN' ERROR
1255                       ("CHARACTERCALL BEFORE FIRST SEG INSTRUCTION",
```

```
1256                    'FALSE', 'FALSE', 'TRUE');
1257                'IF' CORE [SEGM, PRIMARY ADDRESS, 2] = - 1 'THEN'
1258                'BEGIN' SECTADDR:= CORE [SEGM, PRIMARY ADDRESS, 1];
1259                    SECTBYTE:= 1; R:= READ DISC; S:= READ DISC;
1260                    'FOR' T:=READ DISC 'WHILE'
1261                          ¬ (R = CUAK ∧ S = CSPE ∧ T = PRIMARY ADDRESS)
1262                    'DO' 'BEGIN' R:= S; S:= T 'END';
1263                    SECTBYTE:= SECTBYTE + 2; THIRDBYTE:= READ DISC;
1264                    BOOL RDDISC:= 'TRUE'; INPCORE ('TRUE');
1265                    BOOL RDDISC:= 'FALSE'; SECPA:= 0; SECSEG:= SEGM + 1;
1266                'END'
1267                'ELSE'
1268                'BEGIN' SECPA:= CORE [SEGM, PRIMARY ADDRESS, 1];
1269                    SECSEG:= CORE [SEGM, PRIMARY ADDRESS, 2] + SEGM;
1270                    'IF' SECSEG > CORESEGMENTS 'THEN' ADDRESS ERR;
1271                    THIRDBYTE:= CORE [SEGM, PRIMARY ADDRESS, 3];
1272                    'IF' ¬ (THIRDBYTE = 0 ∨ THIRDBYTE = 1 ∨ THIRDBYTE = 2
1273                          ∨ THIRDBYTE = 32)
1274                    'THEN' ADDRESS ERR
1275                'END';
1276                FIRST WHITE ELEMENT:= 'TRUE';
1277                BYTE:= 1; LINE REPETITION:= 'FALSE';
1278                'IF' THIRDBYTE = 2 'THEN'
1279                'BEGIN' 'INTEGER' REMEMBERSEGMF;
1280                    REMEMBERSEGMF:= SEGMF;
1281                    SEGM:=SEGMF;
1282                    'IF' MIC
1283                    'THEN' ERROR
1284                           ("MICROPROGRAM CONTAINS MICROPROGRAM-CALL",
1285                            'TRUE', 'FALSE', 'TRUE');
1286                    MIC:= 'TRUE'; INSTR NUMBER MIC:= 0;
1287                    'FOR' S:= READ CORE 'WHILE' ¬ S = CUAT
1288                    'DO' INTERPRET (S, 'TRUE', 'FALSE', PLOTTER);
1289                    MIC:= 'FALSE'; SEGMF:= SEGM:= REMEMBERSEGMF;
1290                'END'
1291                'ELSE'
1292                'BEGIN'
1293                    SEGMF:=SEGM;
1294                    'IF' KURN = - 1
1295                    'THEN' ERROR
1296                           ("CHARACTER-CALL BEFORE FIRST GRO-INSTRUCTION",
1297                            MIC, 'FALSE', 'TRUE');
1298                    CHAR:= 'TRUE'; LINECOUNT:= 0;
1299                    R:=
1300                    ('IF' THIRDBYTE = 0 'THEN' 3.75 'ELSE' 15) * GRIDTYPE;
1301                    'COMMENT' ROUNDING OFF;
1302                    LINESTART:=
1303                    'IF' THIRDBYTE = 1 'THEN' 0
1304                    'ELSE' ('IF' PLOTTER 'THEN' 10 * FACTOR 'ELSE' 1) * R;
1305                    'IF' PLOTTER 'THEN'
1306                    'BEGIN' YPOSITION:= PLOT (0, 0, 16) - LINESTART;
1307                        PLOTRELATIVE ('FALSE', 0, - LINESTART, - 2, KURN)
1308                    'END'
1309                    'ELSE'
1310                    'BEGIN' YPOSITION:= PRINTPOSITION - LINESTART;
1311                        PRINTRELATIVE
1312                        ('FALSE', 'FALSE', 0, - LINESTART, LINENUMBERMAX,
1313                         'FALSE')
1314                    'END';
1315                    BOOLUAT:= 'FALSE';
```

```
1316                'FOR' N:= 1, N + 1 'WHILE' ¬ BOOLUAT 'DO'
1317                'BEGIN' READ BYTE;
1318                    'IF' B7 = 128 'THEN'
1319                    'BEGIN'
1320                        'IF' B2 = 4 'THEN'
1321                        'BEGIN'
1322                            'IF' B1 = 0 'THEN' LBL ('TRUE', CELLWIDTH)
1323                            'ELSE'
1324                            'IF' B1 = 2 ∧ B3 = 0 ∧ B4 = 0 ∧ B5 = 0
1325                            'THEN' LBL ('FALSE', CELLWIDTH)
1326                            'ELSE' ADDRESS ERR
1327                        'END'
1328                        'ELSE'
1329                        'BEGIN'
1330                            'IF' B1 = 0 'THEN' BLW ('TRUE', REPEAT)
1331                            'ELSE'
1332                            'BEGIN'
1333                                'IF' B3 = 0 ∧ B4 = 0 ∧ B5 = 0 ∧ B6 = 0
1334                                'THEN' BLW ('FALSE', REPEAT)
1335                                'ELSE'
1336                                'IF' B3 = 8 ∧ B4 = 16 ∧ B5 = 32 ∧
1337                                     B6 = 64
1338                                'THEN'
1339                                'BEGIN'
1340                                    'IF' PLOTTER
1341                                    'THEN' PLOTRELATIVE
1342                                           ('FALSE', 0, LINESTART,
1343                                            - 2, KURN)
1344                                    'ELSE' PRINTRELATIVE
1345                                           ('FALSE', 'FALSE', 0,
1346                                            LINESTART, LINENUMBERMAX,
1347                                            'FALSE');
1348                                    BOOLUAT:= 'TRUE'
1349                                'END'
1350                                'ELSE' ADDRESS ERR
1351                            'END'
1352                        'END'
1353                    'END'
1354                    'ELSE' IMAGE LINE
1355                           (CELLWIDTH, CELLHEIGHT, SHADING, REMAINDER,
1356                            REPEAT, YPOSITION, PLOTTER,
1357                            LINE REPETITION, KURN)
1358            'END';
1359            CHAR:= 'FALSE';
1360        'END'
1361    'END' CALL TYPEFOUNT ELEMENT;
1362
1363    CDO:= 00010010;  CDAN:= 00100001;  CGROA:= 00011110;
1364    CGROB:= 00101111; CGROC:= 00011111; CGROD:= 00111010;
1365    CGROE:= 00100000; CGROF:= 00101010; CGROG:= 00110000;
1366    CGROH:= 00011101; CGROI:= 00101110; CGROJ:= 00111101;
1367    CGROK:= 00111110; CGROL:= 00111111; CGROM:= 00011011;
1368    CKEN:= 00111110;  CKUR1:= 00101001; CKUR2:= 00010000;
1369    CKUR3:= 00010001; CNOR:= 00101000;  CNUL:= 11111111;
1370    CPLA:= 00110110;  CPLE:= 00110101;  CRUCK:= 00100100;
1371    CSCHN:= 00100101; CSEG:= 00101101;  CSTART:= 00100110;
1372    CSPE:= 00101011;  CTAB:= 00111100;  CTRE:= 00010110;
1373    CUAK:= 00011100;  CUAT:= 11111011;  CUBL:= 00101100;
1374    CVERA:= 00111011; CVERO:= 00100010; CVERU:= 00100011;
1375    CWID:= 00110001;  CZER:= 00110100;  CZEV:= 00110011;
```

```
1376          CZWQ:= 00110010;  CZWR:= 00000000;   CSTOP:= 00100111;
1377
1378          ENLARGEMENT FACTOR[0]:=1;        ENLARGEMENT FACTOR[1]:=1.125;
1379          ENLARGEMENT FACTOR[2]:=1.25;     ENLARGEMENT FACTOR[3]:=1.375;
1380          ENLARGEMENT FACTOR[4]:=1.5;      ENLARGEMENT FACTOR[5]:=1.75;
1381          ENLARGEMENT FACTOR[6]:=2;        ENLARGEMENT FACTOR[7]:=2.25;
1382          ENLARGEMENT FACTOR[8]:=2.50;     ENLARGEMENT FACTOR[9]:=2.75;
1383          ENLARGEMENT FACTOR[10]:=3;       ENLARGEMENT FACTOR[11]:=3.5;
1384          ENLARGEMENT FACTOR[12]:=4;
1385          CHAR:= 'FALSE';
1386          'BEGIN'
1387              'COMMENT' IN THIS BLOCK,THE INPUT IS TRANSLATED INTO
1388                  DIGISET MACHINE-CODE,WHICH IS STORED IN THE ARRAY
1389                  DTAPE;
1390
1391              'INTEGER' MEDIUM LINES, NM, BEFORE LEFT COMMAND,
1392                  BEFORE RIGHT COMMAND, AFTER RIGHT COMMAND,
1393                  LEFT COMMAND, RIGHT COMMAND, I, J, INSTR COUNT,
1394                  KENBYTES, WORD POINTER, SEG POINTER, EL START;
1395              'REAL' A;
1396              'INTEGER' 'ARRAY' AA [1 : MAXLINENUMBERCHAR, 1 : 20];
1397              'BOOLEAN' BOOLSTOP, NUMBER, BOOLKEN, BOOLUBL, MICEND,ADRCAL;
1398
1399              'INTEGER' 'PROCEDURE' SKIP (INITIAL VALUE);
1400              'BOOLEAN' INITIAL VALUE;
1401              'COMMENT' THE GLOBAL PARAMETER INSTR NUMBER IS CHANGED;
1402              'BEGIN' 'INTEGER' S;
1403                  'IF' INITIAL VALUE 'THEN'
1404                  'BEGIN' INSTR NUMBER:= 2; SKIP:= 0; NEW PAGE;
1405                      PRINT (1); SPACE (40 - PRINTPOS)
1406                  'END'
1407                  'ELSE'
1408                  'BEGIN'
1409                      'FOR' S:= RESYM 'WHILE' (S = 93) ∨ (S = 119)
1410                      'DO' ;
1411                      SKIP:=S;
1412                      'IF'PRINTPOS =100'THEN'
1413                      'BEGIN'
1414                          NLCR; SPACE (40);
1415                      'END';
1416                      PRSYM(S);
1417                  'END'
1418              'END' SKIP;
1419
1420              'PROCEDURE' FILL DTAPE (X); 'VALUE' X; 'INTEGER' X;
1421              'BEGIN'
1422                  'IF' INSTR COUNT > DTAPELENGTH
1423                  'THEN' ERROR
1424                         ("INPUT CAUSES DTAPE-OVERFLOW",
1425                         'FALSE', CHAR, 'TRUE');
1426                  DTAPE [INSTR COUNT]:= X; INSTR COUNT:= INSTR COUNT + 1;
1427                  'IF' PUNCH 'THEN'
1428                  'BEGIN'
1429                      'IF' CHAR ∨ X 'GE' 10 ** 3 'THEN'
1430                      'BEGIN' 'INTEGER' A, B, N;
1431                          A:= B:= 0;
1432                          'FOR' N:= 7 'STEP' -1 'UNTIL' 0 'DO'
1433                          'BEGIN'
1434                              B:= ENTIER(X / 10 ** N);
1435                              X:= X - B * 10 ** N;
```

```
1436                            A:= A + ('IF' B = 1 'THEN' 2 ** N 'ELSE' 0)
1437                        'END';
1438                        PUHEP(A)
1439                    'END'
1440                    'ELSE' PUHEP(X)
1441                'END';
1442            'END' FILL DTAPE;
1443
1444            'PROCEDURE' CHARACTER (K, START); 'VALUE' K, START;
1445            'INTEGER' K; 'BOOLEAN' START;
1446            'BEGIN' 'INTEGER' J; 'BOOLEAN' SINGLE BYTE, UAT, L127;
1447
1448                'PROCEDURE' CODE SHORT LONG LIST (K, START);
1449                'INTEGER' K; 'BOOLEAN' START;
1450                'BEGIN' 'INTEGER' N;
1451                    'BOOLEAN' FIRST COMMAND, LAST EMPTY LINE,
1452                        LAST REPETITION;
1453
1454                    'PROCEDURE' CODE CHAR ELEMENT (K); 'INTEGER' K;
1455                    'BEGIN' 'INTEGER' BIT0, BIT6;
1456
1457                        'PROCEDURE' CODE IMAGELINE (K); 'INTEGER' K;
1458                        'BEGIN' 'INTEGER' J, BYTE1, BYTE2;
1459
1460                            'PROCEDURE' LINE BYTES (A); 'VALUE' A;
1461                            'INTEGER' A;
1462                            'BEGIN'
1463                                'IF' ¬ SINGLE BYTE 'THEN'
1464                                'BEGIN'
1465                                    'IF' A > 63 'THEN'
1466                                    'BEGIN'
1467                                        BYTE2:= FORM BYTE (13, 6, A);
1468                                        BYTE1:= FORM BYTE (5, 0, A);
1469                                    'END'
1470                                    'ELSE'
1471                                    'BEGIN' BYTE2:= 0;
1472                                        BYTE1:=
1473                                        FORM BYTE (5, 0, A) + 1000000
1474                                    'END'
1475                                'END'
1476                                'ELSE'
1477                                'BEGIN' BYTE1:= FORM BYTE (6, 0, A);
1478                                    BYTE2:= 0
1479                                'END'
1480                            'END' LINE BYTES;
1481
1482                            'FOR' J:= 2,
1483                                J + 1 'WHILE' AA [K, J + 1] 'NE' 0
1484                            'DO'
1485                            'BEGIN' LINE BYTES (AA [K, J]);
1486                                FILL DTAPE (BYTE1);
1487                                'IF' ¬ BYTE2 = 0
1488                                'THEN' FILL DTAPE (BYTE2)
1489                            'END';
1490                            LINE BYTES (AA [K, J]);
1491                            FILL DTAPE (BYTE1 + 10000000);
1492                            'IF' ¬ BYTE2 = 0 'THEN' FILL DTAPE (BYTE2)
1493                        'END' CODE IMAGELINE;
1494
1495                        'INTEGER' 'PROCEDURE' FORM BYTE (NB, NO, A);
```

```
1496                        'VALUE' A; 'INTEGER' NB, NO, A;
1497                        'BEGIN' 'INTEGER' AA, B, N;
1498                            B:= 0;
1499                            A:=
1500                            A -
1501                            ENTIER (A / 2 ** (NB + 1)) *
1502                            2 ** (NB + 1);
1503                            'FOR' N:= NB 'STEP' - 1 'UNTIL' NO 'DO'
1504                            'BEGIN' AA:= ENTIER (A / 2 ** N);
1505                                A:= A - AA * 2 ** N;
1506                                B:= B + AA * 10 ** (N - NO)
1507                            'END';
1508                            FORM BYTE:= B;
1509                        'END' FORM BYTE;
1510
1511                        'IF' SINGLE BYTE 'THEN' BIT0:= 0
1512                        'ELSE' BIT0:= 1;
1513                        'IF' AA [K, 2] = 0 'THEN'
1514                        'BEGIN'
1515                            'IF' AA [K, 1] > 0 'THEN' BIT6:= 0
1516                            'ELSE' BIT6:= 1000000;
1517                            'IF' ABS (AA [K, 1]) < 8
1518                            'THEN' FILL DTAPE
1519                                    (10000100 + BIT0 + BIT6 +
1520                                     FORM BYTE (2, 0, ABS (AA [K, 1]))
1521                                     * 1000)
1522                            'ELSE'
1523                            'BEGIN'
1524                                FILL DTAPE (10000110 + BIT0 + BIT6);
1525                                FILL DTAPE
1526                                (FORM BYTE (7, 0, ABS (AA [K, 1])))
1527                            'END'
1528                        'END'
1529                        'ELSE'
1530                        'IF' AA [K, 1] > 1 'THEN'
1531                        'BEGIN'
1532                            'IF' AA [K, 1] < 16
1533                            'THEN' FILL DTAPE
1534                                    (10000000 + BIT0 +
1535                                     FORM BYTE (3, 0, AA [K, 1] - 1) *
1536                                     1000)
1537                            'ELSE'
1538                            'BEGIN' FILL DTAPE (10000010 + BIT0);
1539                                FILL DTAPE
1540                                (FORM BYTE (7, 0, AA [K, 1] - 1))
1541                            'END';
1542                            CODE IMAGELINE (K)
1543                        'END'
1544                        'ELSE' CODE IMAGELINE (K)
1545                    'END' CODE CHAR ELEMENT;
1546
1547                    'PROCEDURE' CODE LIST (IC, JC); 'INTEGER' IC, JC;
1548                    'BEGIN' 'INTEGER' HC;
1549                        'FOR' HC:= IC 'STEP' 1 'UNTIL' JC
1550                        'DO' CODE CHAR ELEMENT (HC)
1551                    'END' CODE LIST;
1552
1553                    'PROCEDURE' CODE TAIL (N); 'INTEGER' N;
1554                    'BEGIN' SINGLE BYTE:= 'TRUE';
1555                        'IF' UAT 'THEN' CODE LIST (N, K) 'ELSE'
```

```
1556                        'IF' LAST EMPTY LINE 'THEN'
1557                        'BEGIN' CODE LIST (N, K - 2);
1558                            SINGLE BYTE:= 'FALSE';
1559                            CODE LIST (K - 1, K)
1560                        'END'
1561                        'ELSE'
1562                        'IF' LAST REPETITION 'THEN'
1563                        'BEGIN' CODE LIST (N, K - 1);
1564                            SINGLE BYTE:= 'FALSE';
1565                            CODE CHAR ELEMENT (K)
1566                        'END'
1567                        'ELSE'
1568                        'IF' AFTER RIGHT COMMAND > 1 'THEN'
1569                        'BEGIN' CODE LIST (N, K - 1);
1570                            FILL DTAPE (10000101);
1571                            SINGLE BYTE:= 'FALSE';
1572                            CODE CHAR ELEMENT (K)
1573                        'END'
1574                        'ELSE'
1575                        'BEGIN' CODE LIST (N, RIGHT COMMAND - 1);
1576                            SINGLE BYTE:= 'FALSE';
1577                            CODE LIST (RIGHT COMMAND, K)
1578                        'END'
1579                    'END' CODE TAIL;
1580
1581                    FIRST COMMAND:= AA [1, 1] > 1 ∨ AA [1, 2] = 0;
1582                    'IF' K > 1 'THEN'
1583                    'BEGIN' LAST EMPTY LINE:= AA [K - 1, 2] = 0;
1584                        LAST REPETITION:= AA [K, 1] > 1
1585                    'END'
1586                    'ELSE' LAST EMPTY LINE:= LAST REPETITION:=
1587                        'FALSE';
1588                    'IF' FIRST COMMAND 'THEN' CODE TAIL (1) 'ELSE'
1589                    'IF' LEFT COMMAND > 1 'THEN'
1590                    'BEGIN'
1591                        'IF' BEFORE LEFT COMMAND > 1 ∨ START 'THEN'
1592                        'BEGIN' SINGLE BYTE:= 'TRUE';
1593                            FILL DTAPE (10000100)
1594                        'END';
1595                        CODE LIST (1, LEFT COMMAND - 1);
1596                        CODE TAIL (LEFT COMMAND)
1597                    'END'
1598                    'ELSE'
1599                    'IF' MEDIUM LINES > 1 ∧
1600                         (UAT ∨ LAST EMPTY LINE ∨ LAST REPETITION)
1601                    'THEN'
1602                    'BEGIN' SINGLE BYTE:= 'TRUE';
1603                        FILL DTAPE (10000100);
1604                        'IF' UAT 'THEN' CODE LIST (1, K) 'ELSE'
1605                        'IF' LAST EMPTY LINE 'THEN'
1606                        'BEGIN' CODE LIST (1, K - 2);
1607                            SINGLE BYTE:= 'FALSE';
1608                            CODE LIST (K - 1, K)
1609                        'END'
1610                        'ELSE'
1611                        'IF' LAST REPETITION 'THEN'
1612                        'BEGIN' CODE LIST (1, K - 1);
1613                            SINGLE BYTE:= 'FALSE';
1614                            CODE CHAR ELEMENT (K)
1615                        'END'
```

```
1616                        'END'
1617                        'ELSE'
1618                        'IF' MEDIUM LINES > 2 'THEN'
1619                        'BEGIN' FILL DTAPE (10000100);
1620                            SINGLE BYTE:= 'TRUE'; CODE LIST (1, K - 1);
1621                            FILL DTAPE (10000101); SINGLE BYTE:= 'FALSE';
1622                            CODE CHAR ELEMENT (K)
1623                        'END'
1624                        'ELSE'
1625                        'BEGIN'
1626                            'IF' START 'THEN'
1627                            'BEGIN' SINGLE BYTE:= 'FALSE';
1628                                FILL DTAPE (10000101)
1629                            'END';
1630                            CODE LIST (1, K)
1631                        'END'
1632                    'END' CODE SHORT LONG LIST;
1633
1634                    'PROCEDURE' READ CHAR ELEMENT (K, J); 'INTEGER' K, J;
1635                    'BEGIN' 'INTEGER' R, S;
1636                        S:= SKIP ('FALSE');
1637                        'IF' S = 65 'THEN'
1638                        'BEGIN' S:= RESYM; PRSYM (S); S:= - S 'END';
1639                        'IF' S < 10 'THEN'
1640                        'BEGIN' R:= S;
1641                            'FOR' S:= SKIP ('FALSE') 'WHILE' S < 10
1642                            'DO' R:= R * 10 + S;
1643                            AA [K, J]:= R;
1644                            'IF' R = 0
1645                            'THEN' ERROR
1646                                   ("ZERO-VALUED LINE-ELEMENT", 'FALSE',
1647                                    'TRUE', 'FALSE');
1648                            'IF' J = 1 'THEN'
1649                            'BEGIN' L127:= UAT:= 'FALSE'; NM:= 0 'END'
1650                            'ELSE'
1651                            'IF' AA [K, J] > 127 'THEN' L127:= 'TRUE'
1652                            'ELSE' 'IF' AA [K, J] > 63 'THEN' NM:= NM + 1;
1653                            'IF' S = 87
1654                            'THEN' READ CHAR ELEMENT (K, J + 1)
1655                            'ELSE'
1656                            'IF' S = 99 'THEN'
1657                            'BEGIN' AA [K, J + 1]:= AA [K, J + 2]:= 0;
1658                                'IF' - J - 2 * ENTIER (J / 2) = 1
1659                                'THEN' ERROR
1660                                       ("EVEN NUMBER OF LINE-ELEMENTS",
1661                                        'FALSE', 'TRUE', 'FALSE')
1662                            'END'
1663                            'ELSE' ERROR
1664                                   ("LINE-ELEMENT NOT FOLLOWED BY ) OR ,",
1665                                    'FALSE', 'TRUE', 'FALSE')
1666                        'END'
1667                        'ELSE'
1668                        'IF' S = 98 'THEN' READ CHAR ELEMENT (K, 1) 'ELSE'
1669                        'IF' S = 91 'THEN' UAT:= 'TRUE'
1670                        'ELSE' ERROR
1671                               ("SYMBOL FOLLOWING LINE-ELEMENT NOT ALLOWED",
1672                                'FALSE', 'TRUE', 'FALSE');
1673                        'IF' BOOLERROR ^ - UAT 'THEN'
1674                        'BEGIN' LINECOUNT:= LINECOUNT + 1;
1675                            READ CHAR ELEMENT (K, 1)
```

```
1676                        'END'
1677                    'END' READ CHAR ELEMENT;
1678
1679                    CHAR:= 'TRUE'; SINGLE BYTE:= 'FALSE';
1680                    'IF' K = 1 'THEN'
1681                    'BEGIN'
1682                        BEFORE LEFT COMMAND:= BEFORE RIGHT COMMAND:= 0;
1683                        MEDIUM LINES:=0;
1684                        LEFT COMMAND:= RIGHT COMMAND:= 1
1685                    'END';
1686                    READ CHAR ELEMENT (K, 1); LINECOUNT:= LINECOUNT + 1;
1687                    'IF' ¬ BOOLERROR 'THEN'
1688                    'BEGIN'
1689                        'IF' UAT 'THEN'
1690                        'BEGIN'
1691                            AFTER RIGHT COMMAND:=
1692                            MEDIUM LINES - BEFORE RIGHT COMMAND;
1693                            CODE SHORT LONG LIST (K - 1, START);
1694                        'END'
1695                        'ELSE'
1696                        'IF' L127 'THEN'
1697                        'BEGIN'
1698                            AFTER RIGHT COMMAND:=
1699                            MEDIUM LINES - BEFORE RIGHT COMMAND;
1700                            CODE SHORT LONG LIST (K, START);
1701                            K:= 1; START:= 'FALSE';
1702                            CHARACTER (1, 'FALSE')
1703                        'END'
1704                        'ELSE'
1705                        'IF' AA [K, 2] = 0 'THEN'
1706                        'BEGIN'
1707                            'IF' K > 1 'THEN'
1708                            'BEGIN'
1709                                'IF' AA [K - 1, 2] = 0 'THEN'
1710                                'BEGIN'
1711                                    AA [K - 1, 1]:=
1712                                    AA [K - 1, 1] + AA [K, 1];
1713                                    K:= K - 1
1714                                'END';
1715                                'IF' LEFT COMMAND = 1 'THEN'
1716                                'BEGIN' LEFT COMMAND:= RIGHT COMMAND:= K;
1717                                    BEFORE RIGHT COMMAND:=
1718                                    BEFORE LEFT COMMAND:= MEDIUM LINES
1719                                'END'
1720                                'ELSE'
1721                                'BEGIN'
1722                                    BEFORE RIGHT COMMAND:= MEDIUM LINES;
1723                                    RIGHT COMMAND:= K
1724                                'END'
1725                            'END';
1726                            CHARACTER (K + 1, START)
1727                        'END'
1728                        'ELSE'
1729                        'BEGIN'
1730                            'IF' AA [K, 1] > 1 ∧ K > 1 'THEN'
1731                            'BEGIN'
1732                                'IF' LEFT COMMAND = 1 'THEN'
1733                                'BEGIN' LEFT COMMAND:= RIGHT COMMAND:= K;
1734                                    BEFORE RIGHT COMMAND:=
1735                                    BEFORE LEFT COMMAND:= MEDIUM LINES
```

```
1736                                'END'
1737                                'ELSE'
1738                                'BEGIN'
1739                                    BEFORE RIGHT COMMAND:= MEDIUM LINES;
1740                                    RIGHT COMMAND:= K
1741                                'END'
1742                            'END';
1743                            MEDIUM LINES:= MEDIUM LINES + NM;
1744                            CHARACTER (K + 1, START)
1745                        'END'
1746                    'END';
1747                    CHAR:= 'FALSE';
1748                'END' CHARACTER;
1749
1750                'PROCEDURE' LISTINGNUMBER;
1751                'BEGIN' NLCR; PRINT (INSTR NUMBER); SPACE (40 - PRINTPOS);
1752                    INSTR NUMBER:= INSTR NUMBER + 1;
1753                'END' LISTINGNUMBER;
1754
1755                'PROCEDURE' FREE CORE(WORD POINTER, SEG POINTER);
1756                'INTEGER' WORD POINTER, SEG POINTER;
1757                'BEGIN' 'INTEGER' LENGTH;
1758                    LENGTH:= (INSTR COUNT + 1 - EL START) / 3;
1759                    WORD POINTER:= WORD POINTER +
1760                    ('IF' LENGTH = ENTIER(LENGTH) 'THEN' LENGTH
1761                    'ELSE' LENGTH + 1);
1762                    'IF' WORD POINTER > 255 'THEN'
1763                    'BEGIN'
1764                        WORD POINTER:= WORD POINTER - 256;
1765                        SEG POINTER:= SEG POINTER + 1
1766                    'END'
1767                'END' FREE CORE;
1768
1769                'PROCEDURE' UATT CHECK(NUMBER, BOOLSTOP);
1770                'BOOLEAN' NUMBER, BOOLSTOP;
1771                'BEGIN' 'INTEGER' N; 'REAL' 'ARRAY' AA[1:2];
1772                    'FOR' N:= 1, 2 'DO'
1773                    'BEGIN'
1774                        NUMBER:= 'TRUE'; LISTINGNUMBER;
1775                        READ INPUT ELEMENT; AA[N]:= A; A:= 0;
1776                    'END';
1777                    'IF' AA[1] = 39269 ∧ AA[2] = 39269 'THEN'
1778                    'BEGIN'
1779                        FILL DTAPE(CUAT); FILL DTAPE(CUAT);
1780                    'END'
1781                    'ELSE' ERROR("TYPEFOUNT EL. NOT FOLLOWED BY UAT;UAT;",
1782                                 'FALSE', 'FALSE', 'FALSE');
1783                    BOOLSTOP:= AA[1] = 39260 ∧ AA[2] = 1344841
1784                'END' UATT CHECK;
1785
1786                'PROCEDURE' READ INPUT ELEMENT;
1787                'BEGIN' 'INTEGER' S;
1788                    S:= SKIP ('FALSE');
1789                    'IF' S = 98 'THEN'
1790                    'BEGIN'
1791                        'IF' A = 0 'THEN'
1792                        'BEGIN' A:= S; NUMBER:= 'FALSE';  'END'
1793                        'ELSE' ERROR
1794                               ("INSTRUCTION CONTAINS NON-ALLOWED SYMBOL",
1795                                'FALSE', 'FALSE', 'FALSE')
```

```
1796                    'END'
1797                    'ELSE'
1798                    'IF' S = 100 'THEN'
1799                    'BEGIN'
1800                        'IF' A 'NE' 0 'THEN'
1801                        ERROR("INSTRUCTION CONTAINS NON-ALLOWED SYMBOL",
1802                              'FALSE', 'FALSE', 'FALSE');
1803                        READ INPUT ELEMENT
1804                    'END'
1805                    'ELSE'
1806                    'IF' S = 101 'THEN'
1807                    'BEGIN'
1808                        'IF'SKIP('FALSE') 'NE' 91 'THEN'
1809                        ERROR("INSTRUCTION CONTAINS NON-ALLOWED SYMBOL",
1810                              'FALSE', 'FALSE', 'FALSE');
1811                        'IF' NUMBER 'THEN'
1812                        'BEGIN' FILL DTAPE(A); LISTINGNUMBER 'END'
1813                        'ELSE'
1814                        'BEGIN'
1815                            MICEND:= 'TRUE';
1816                            CODE SWITCH;
1817                            MICEND:= 'FALSE'
1818                        'END';
1819                        A:= 0;
1820                        'IF' ADRCAL 'THEN'
1821                        'BEGIN'
1822                            FREE CORE( WORD POINTER, SEG POINTER);
1823                            ADRCAL:= 'FALSE'
1824                        'END';
1825                        UATT CHECK(NUMBER, BOOLSTOP);
1826                        LISTINGNUMBER;
1827                        NUMBER:= 'TRUE'
1828                    'END'
1829                    'ELSE'
1830                    'IF' S = 91 'THEN'
1831                    'BEGIN'
1832                        'IF' NUMBER 'THEN'
1833                        'BEGIN'
1834                            'IF' A 'NE' 256 'THEN' FILL DTAPE(A);
1835                            LISTINGNUMBER;
1836                            'IF' A = 62 ^ DTAPE[INSTR COUNT-2] = CSPE
1837                            ^ DTAPE[INSTR COUNT-3] = CUAK 'THEN'
1838                            'BEGIN' WORD POINTER:= 0; SEG POINTER:= 1 'END'
1839                            'ELSE' 'IF' A = 256 'THEN'
1840                            'BEGIN'
1841                                ADRCAL:= 'TRUE';
1842                                SPACE(40 - PRINTPOS);
1843                                PRINTTEXT("ADDRESS CALCULATION");
1844                                LISTINGNUMBER;
1845                                EL START:= INSTR COUNT + 2;
1846                                FILL DTAPE(WORD POINTER);
1847                                FILL DTAPE(SEG POINTER)
1848                            'END';
1849                            'IF' BOOLKEN 'THEN'
1850                            'BEGIN' KENBYTES:= KENBYTES + 1;
1851                                BOOLKEN:= KENBYTES = 3
1852                            'END'
1853                        'END'
1854                    'END'
1855                   .'ELSE'
```

```
1856                'IF' S < 10 ∧ NUMBER 'THEN'
1857                'BEGIN' A:= A * 10 + S;
1858                    'IF' A > 256
1859                    'THEN' ERROR
1860                           ("INTEGER CANNOT BE CODED IN ONE BYTE",
1861                            'FALSE', 'FALSE', 'FALSE');
1862                    READ INPUT ELEMENT
1863                'END'
1864                'ELSE'
1865                'BEGIN' NUMBER:= 'FALSE'; A:= A * 36 + S;
1866                    READ INPUT ELEMENT
1867                'END'
1868            'END' READ INPUT ELEMENT;
1869
1870            'PROCEDURE' CODE SWITCH;
1871           'BEGIN'
1872                    'IF' A = 468 'THEN' FILL DTAPE (CD0) 'ELSE'
1873                    'IF' A = 17231 'THEN' FILL DTAPE (CDAN) 'ELSE'
1874                    'IF' A = 782362 'THEN' FILL DTAPE (CGROA) 'ELSE'
1875                    'IF' A = 782363 'THEN' FILL DTAPE (CGROB) 'ELSE'
1876                    'IF' A = 782364 'THEN' FILL DTAPE (CGROC) 'ELSE'
1877                    'IF' A = 782365 'THEN' FILL DTAPE (CGROD) 'ELSE'
1878                    'IF' A = 782366 'THEN' FILL DTAPE (CGROE) 'ELSE'
1879                    'IF' A = 782367 'THEN' FILL DTAPE (CGROF) 'ELSE'
1880                    'IF' A = 782368 'THEN' FILL DTAPE (CGROG) 'ELSE'
1881                    'IF' A = 782369 'THEN' FILL DTAPE (CGROH) 'ELSE'
1882                    'IF' A = 782370 'THEN' FILL DTAPE (CGROI) 'ELSE'
1883                    'IF' A = 782371 'THEN' FILL DTAPE (CGROJ) 'ELSE'
1884                    'IF' A = 782372 'THEN' FILL DTAPE (CGROK) 'ELSE'
1885                    'IF' A = 782373 'THEN' FILL DTAPE (CGROL) 'ELSE'
1886                    'IF' A = 782374 'THEN' FILL DTAPE (CGROM) 'ELSE'
1887                    'IF' A = 26447 'THEN'
1888                    'BEGIN' FILL DTAPE (CKEN);
1889                        'IF' ¬ BOOLKEN 'THEN'
1890                        'BEGIN' BOOLKEN:= 'TRUE'; KENBYTES:= 0 'END'
1891                    'END'
1892                    'ELSE'
1893                    'IF' A = 972973 'THEN' FILL DTAPE (CKUR1) 'ELSE'
1894                    'IF' A = 972974 'THEN' FILL DTAPE (CKUR2) 'ELSE'
1895                    'IF' A = 972975 'THEN' FILL DTAPE (CKUR3) 'ELSE'
1896                    'IF' A = 30699 'THEN' FILL DTAPE (CNOR) 'ELSE'
1897                    'IF' A = 30909 'THEN' FILL DTAPE (CNUL) 'ELSE'
1898                    'IF' A = 33166 'THEN' FILL DTAPE (CPLA) 'ELSE'
1899                    'IF' A = 33170 'THEN' FILL DTAPE (CPLE) 'ELSE'
1900                    'IF' A = 1299044 'THEN' FILL DTAPE (CRUCK) 'ELSE'
1901                    'IF' A = 1322555 'THEN' FILL DTAPE (CSCHN) 'ELSE'
1902                    'IF' A = 36808 'THEN' FILL DTAPE (CSEG) 'ELSE'
1903                    'IF' A = 37202 'THEN' FILL DTAPE (CSPE) 'ELSE'
1904                    'IF' A = 48396233 'THEN'
1905                    'BEGIN' FILL DTAPE (CSTART); BOOLUBL:= 'FALSE''END'
1906                    'ELSE'
1907                    'IF' A = 37955 'THEN' FILL DTAPE (CTAB) 'ELSE'
1908                    'IF' A = 38570 'THEN' FILL DTAPE (CTRE) 'ELSE'
1909                    'IF' A = 39260 'THEN' FILL DTAPE (CUAK) 'ELSE'
1910                    'IF' A = 39269 'THEN' FILL DTAPE (CUAT) 'ELSE'
1911                    'IF' A = 39297 'THEN'
1912                    'BEGIN' FILL DTAPE (CUBL); BOOLUBL:= 'TRUE' 'END'
1913                    'ELSE'
1914                    'IF' A = 1465462 'THEN' FILL DTAPE (CVERA) 'ELSE'
1915                    'IF' A = 1465476 'THEN' FILL DTAPE (CVERO) 'ELSE'
```

```
1916                    'IF' A = 1465482 'THEN' FILL DTAPE (CVERU) 'ELSE'
1917                    'IF' A = 42133 'THEN' FILL DTAPE (CWID) 'ELSE'
1918                    'IF' A = 45891 'THEN' FILL DTAPE (CZER) 'ELSE'
1919                    'IF' A = 45895 'THEN' FILL DTAPE (CZEV) 'ELSE'
1920                    'IF' A = 46538 'THEN' FILL DTAPE (CZWQ) 'ELSE'
1921                    'IF' A = 46539 'THEN' FILL DTAPE (CZWR) 'ELSE'
1922                    'IF' A = 1344841 'THEN'
1923                    'BEGIN' FILL DTAPE (CSTOP); BOOLSTOP:= 'TRUE' 'END'
1924                    'ELSE'
1925                    'IF' A = 98 'THEN'
1926                    'BEGIN'
1927                        LINECOUNT:= 1; CHARACTER (1, 'TRUE'); A:= 0;
1928                        'IF' ADRCAL 'THEN'
1929                        'BEGIN'
1930                            FREE CORE(WORD POINTER, SEG POINTER);
1931                            ADRCAL:= 'FALSE'
1932                        'END';
1933                        UATT CHECK(NUMBER, BOOLSTOP)
1934                    'END'
1935                    'ELSE'
1936                    'IF' BOOLKEN 'THEN'
1937                    'BEGIN' KENBYTES:= KENBYTES + 1; FILL DTAPE (A);
1938                        BOOLKEN:= KENBYTES < 3
1939                    'END'
1940                    'ELSE'
1941                    'IF' BOOLUBL 'THEN' FILL DTAPE (A)
1942                    'ELSE' ERROR
1943                           ("INSTRUCTION DOES NOT EXIST", 'FALSE',
1944                            'FALSE', 'FALSE');
1945                    'IF' BOOLSTOP 'THEN' NEW PAGE
1946                    'ELSE'
1947                    'IF' ¬ MICEND 'THEN' LISTINGNUMBER
1948            'END'CODE SWITCH;
1949
1950             SKIP ('TRUE'); INSTR COUNT:= 1;
1951             BOOLSTOP:= BOOLKEN:= BOOLUBL:= ADRCAL:= 'FALSE';
1952             'FOR' I:= 1, I + 1 'WHILE' ¬ BOOLSTOP 'DO'
1953             'BEGIN' A:= 0; NUMBER:= 'TRUE'; READ INPUT ELEMENT;
1954                'IF' ¬ NUMBER 'THEN' CODE SWITCH
1955             'END';
1956
1957             'IF' BOOLERROR 'THEN' EXIT;
1958        'END' THE INPUT IS TRANSLATED INTO DIGISET MACHINE-CODE;
1959        BOOLWID:= MIC:= BOOL RDDISC:= BOOL INPCORE:=
1960        BOOL CHAR TO DISC:= BUFFERSEG:= BOOLZWQ:= CHAR:= 'FALSE';
1961        SEGM:= SECTBYTE:= KURN:= - 1;
1962        BYTENUMBER INPUT:= INSTR NUMBER:= 1;
1963        'FOR' R:= 1 'STEP' 1 'UNTIL' CORESEGMENTS 'DO'
1964        'FOR' S:= 0 'STEP' 1 'UNTIL' 255 'DO'
1965        'FOR' T:= 1, 2, 3 'DO' CORE [R, S, T]:= 0;
1966        'COMMENT' THIS INITIALISATION IS NEEDED FOR ADDRESS-ERROR
1967            CHECKS;
1968        'IF' PLOTTER 'THEN'
1969        'BEGIN' PLOTRELATIVE ('TRUE', 0, 0, 0, 0);
1970            PLOTFRAME (0, 0, 15000, 2750, 15000, 2750);
1971        'END'
1972        'ELSE'
1973        'BEGIN'
1974            'FOR' LINENUMBER:= 1 'STEP' 1 'UNTIL' LINENUMBERMAX 'DO'
1975            'FOR' PRINTPOSITION:= 1 'STEP' 1 'UNTIL' 144
```

```
1976                'DO' LINEPRINTEROUTPUT [LINENUMBER, PRINTPOSITION]:=
1977                     'FALSE';
1978                PRINTRELATIVE
1979                ('TRUE', 'TRUE', 1, 73, LINENUMBERMAX, 'FALSE')
1980           'END';
1981           'IF' ¬ READ DTAPE = CUAK ∨ ¬ READ DTAPE = CSTART
1982           'THEN' ERROR
1983                  ("PROGRAM DOES NOT BEGIN WITH UAK START", 'FALSE',
1984                   'FALSE', 'FALSE');
1985           'FOR' S:= READ DTAPE 'WHILE' 'TRUE'
1986           'DO' INTERPRET (S, 'FALSE', 'FALSE', PLOTTER)
1987      'END'
1988 'END'
```

APPENDIX B

1. Syntax of the Digiset machine code

The rules of the syntax listed below should be interpreted as follows:

- Non terminals are denoted in small letters, terminals begin with capital letters.
- The notation is in Backus normal form.
  meta characters are:

  ":" for "is defined as"
  "," for "followed by",
  "." for "end of rule",
  "|" for "or".

- In each rule, the first occurence of a non-terminal (except the one defined by that rule) is followed by the number of its defining rule between < and >.
- To emphasize the byte structure of the code, the following redundant notation is used:
  "byte ("should not be read as part of the syntax, but as a comment, that gives information about the byte-structure;")" stands for ",".
  N.B. "byte" not followed by "("is the nonterminal for a number between Ø and 255 in binairy notation.
- Each term in capitals followed by a ";" stands for a unique value in the 8-bit code of the Digiset.

2. Syntax of the shorthand code

The differences between the shorthand code and the Digiset code are the following:

- The numerical values are given in decimal notation followed by a semicolon; for example rule 27:
  pa 62: byte (Ø,Ø,1,1,1,1,1,Ø), should be read as
  pa 62: 62;.

- The definition of single characters is simplified. This is accomplished by dropping the byte structure and putting image lines between brackets. The divergence starts with rule 35. The alternative definitions are marked with an asterisk.
- Each microprogram is embraced by two brackets, in order to improve error recovery. This can be seen from the definition of rule 71 (see rule 71*).

3. Syntax of the Digiset code

```
1  program    : start <2>, body <4>, stop <3>.
2  start      : UAK;, START;.
3  stop       : UAK;, STOP;.
4  body       : inortx <5> | inortx, body.
5  inortx     : input <7> | text <73> | neglect <6>.
6  neglect    : UAK;, UBL;, <"everything except start">, start <2>.
7  input      : segopt <8>, core <1Ø> | disc <12> | segopt, disc to core
                <13>.
8  segopt     : | seg <9>.
9  seg        : UAK;, SEG;, byte <93>.
10 core       : core elmt <11> | core elmt, core.
11 core elmt  : typeface <26> |  char <34> | NUL; | KEN;, byte <93>, byte,
                byte.
12 disc       : uspe <14>, uple <17>, alf <25>, uspe, utre <15>, uatt <16>.
13 disctocore: uspe <14>, upla <18>.
14 uspe       : UAK;, SPE;.
15 utre       : UAK;, TRE;.
16 uatt       : UAT;, UAT;.
17 uple       : UAK;, PLE;, discad <19>, uatt <16>.
18 upla       : UAK;, PLA;, discad <19>, uatt <16>.
19 discad     : byte (cyll <20>, k <22>, sekt <24>), byte (alforch <23>, Ø,
                Ø, cylh <21>).
20 cyll       : BIT 4.
21 cylh       : BIT 5.
22 k          : BIT.
23 alforch    : BIT.
```

```
24 sekt        : BIT 3.
25 alf         : typeface <26>, characterset <33>.
26 typeface    : uspe <14>, pa 62 <27>, tfword <28>, uatt <16>.
27 pa 62       : byte (Ø,Ø,1,1,1,1,1,Ø).
28 tfword      : zbyte <94>, zbyte, byte (Ø, tftype <29>).
29 tftype      : grid <30>, seize <31>, angle <32>.
30 grid        : BIT 2.
31 seize       : BIT 3.
32 angle       : BIT 3.
33 character-
   set         : | char <34>, characterset.
34 char        : micropr <4Ø> | properchar <35>.
35 properchar: uspe <14>, pa <36>, chword <37>, linelist <43>, uatt <16>.
36 pa          : byte <93>,
37 chword      : segad <42>, byte (Ø,Ø,VS1 <38>, Ø, Ø, Ø, VS2 <39>).
38 VS1         : BIT.
39 VS2         : BIT.
40 micropr     : uspe <14>, pa <36>, mpword <41>, comlist <71>, uatt <16>.
41 mpword      : segad <42>, byte (Ø, Ø, Ø, Ø, Ø, Ø, 1, Ø).
42 segad       : byte (Ø, Ø, BIT 6), byte <93>.
43 linelist    : | lspack <44> | slpack <45>.
44 lspack      : longpack <46> | longpack, slpack <45>.
45 slpack      : shortpack <47> | shortpack, lspack <44>.
46 longpack    : long <48> | long, longpack.
47 shortpack   : short <49> | short, shortpack.
48 long        : lbll <53>, lo <51> | blwl <55>, lo <51>.
49 short       : lbls <52>, sh <5Ø> | blws <54>, sh.
5Ø sh          : S <63> | S, sh.
51 lo          : 1 <64> | 1, lo.
52 lbls        : lbl <56> + spack <58>.
53 lbll        : lbl <56> + lpack <59>.
54 blws        : blw <57> + spack <58>.
55 blwl        : blw <57> + lpack <59>.
56 lbl + pack: byte (1, dir <6Ø>, num 3 <61>, Ø, 1, pack)  |
               byte (1,dir, Ø, Ø, Ø, 1, 1, pack), byte <93>.
```

```
57 blw + pack: byte (1, num 4 <62>, Ø, Ø, pack) |
               byte (1, Ø, Ø, Ø, Ø, 1, pack), byte <93>.
58 spack     : Ø.
59 1pack     : 1.
60 dir       : BIT.
61 num 3     : BIT 3.
62 num 4     : BIT 4.
63 s         : swze <65> | swz <67>, s.
64 1         : 1wze <66> | 1wz <68>, 1.
65 swze      : byte (Ø, BIT 7), byte (1, BIT 7).
66 1wze      : ww <69>, byte (1,Ø, BIT 6) | ww, byte (1,1, BIT 6), byte
               <93>.
67 swz       : byte (Ø, BIT 7), byte (Ø, BIT 7).
68 1wz       : ww <69>, zz <7Ø>.
69 ww        : byte (Ø, Ø BIT 6) | byte (Ø, 1, BIT 6), byte <93>.
70 zz        : byte (Ø, Ø, BIT 6) | byte (Ø, 1, BIT 6), byte <93>.
71 comlist   : comlelmt <72> | comlelmt, comlist.
72 comlelmt  : segopt <8>, disc to core <13> | setchar <75> |
               layout <78> | seg <9> | iter <77>.
73 text      : texdat <74> | texdat, text.
74 texdat    : setchar <75> | domic <76> | layout <78> | seg <9> | iter <77>.
75 setchar   : pa <36>.
76 domic     : pa <36>.
77 iter      : UAK;, WID;, byte <93>, byte, domic | UAK;, WID;, byte <93>,
               byte, setchar.
78 layout    : relseize <79> | unsquare <81> | italic <82> | position <84> |
               UAK;, SCHN; | UAK;, DØ;.
79 relseize  : UAK;, gro <80>.
80 gro       : GROa; | GROb; | GROc; | GROd; | GROe; | GROf; | GROg; |
               GROh; | GROi; | GROj; | GROk; | GROl; | GROm.
81 unsquare  : UAK;, DAN;, byte <93>.
82 italic    : UAK;, knor <83>.
83 knor      : NOR; | KUR 1; | KUR 2; | KUR 3;.
84 position  : horp <85> | verp <86> | mixp <88> | defp <92>.
```

```
85 horp      : ZWR; | UAK;, RUCK;, byte <93>, byte.
86 verp      : UAK;, VERA; | UAK;, VERO;, tilt | UAK;, VERU;, tilt.
87 tilt      : byte (Ø, Ø, Ø, Ø, BIT 4), byte <93>.
88 mixp      : UAK;, ZER;, shift <89> | UAK;, ZEV;, shift.
89 shift     : byte (m <9Ø>, sv <91>), byte <93>.
90 m         : BIT 3.
91 sv        : BIT 5.
92 defp      : UAK;, ZWQ;, byte <93>, byte | UAK;, TAB;, byte, byte.
93 byte      : BIT 8
94 zbyte     : byte (Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø).

35* properchar    : uspe <14>, pa <36>, chword <37*>, linelist <43*>,
                    uatt <16>.
37* chword         : addressvalue <94*>, byte (Ø, Ø, vs1 <38>, Ø, Ø, Ø, vs2
                    <39>).
43* linelist       : ; | image line <95*>, linelist.
71* comlist        : [, actual comlist <96>, ].
94* addressvalue   : undefined <97*> | segad <42>.
95* imageline      : empty line <98*> | nonempty line <99*>.
96* actual comlist: comlelmt <72> | comlelmt, actual comlist.
97* undefined      : 256.
98* emptyline      : (, <signed integer>,).
99* nonemptyline   : (, <unsigned integer>, wzpack <100*>,).
1ØØ* wzpack        : wz <1Ø1*> | wz, wzpack.
1Ø1* wz            : white <1Ø2*>, black <1Ø3>.
1Ø2* white         : <unsigned integer>,
1Ø3* black         : <unsigned integer>.
```

APPENDIX C

1. An ALGOL 60 program for the generation of a character set for line drawings (fig. 3.4.2).
2. Main program for the generation of fig. 3.4.5.

D2665F.001,NOOTTENHAGEN,T300,R300,K5

```
      'BEGIN''COMMENT'2665F,PTENHAGEN,HNOOT,GENEREREN VAN L'JNALFABET;
    1 'INT'H,V,PUT,PA,WID,ALFB;
    2 'REAL'VD,HD,D,STEP,FRAC,VFAC,HFAC;
    3 'ARRAY'ALFTAB[1:256,1:3];
    4
    5 'PROC'PRCSYM(S);'BEGIN'PRSYM(S);'IF'PUT=1'THEN'CSYM(S)'END';
    6
    7 'PROC'ABSFIXTC(N,M,V);'VAL'V;'REAL'V;
    8 'BEGIN'ABSFIXT(N,M,V);'IF'PUT=1'THEN'ABSFIXC(N,M,V)'END';
    9
   10 'PROC'FIXTC(N,M,V);'VAL'V;'REAL'V;
   11 'BEGIN'FIXT(N,M,V);'IF'PUT=1'THEN'FIXC(N,M,V)'END';
   12
   13 'PROC'PRCTEXT(S);'BEGIN'PRINTTEXT(S);'IF'PUT=1'THEN'CTEXT(S)'END';
   14
   15 'PROC'CTEXT(S);'STRING'S;
   16 'BEGIN''INT'K,SYM;K:=0;
   17 'FOR'SYM:=STRINGSYMBOL(K,S)'WHILE'SYM'NE'255'DO'
   18 'BEGIN'CSYM(SYM);K:=K+1'END'
   19 'END';
   20
   21 'INT''PROC'NEXTPA;NEXTPA:=PA:='IF'PA'LE'0∨PA'GE'ALFB'THEN'1'ELSE'
   22   PA+('IF'PA=27∨PA=61∨PA=123∨PA=127'THEN'2'ELSE'1);
   23
   24 'PROC'GENTYPEFOUNT(CELLTYPE);'STRING'CELLTYPE;
   25 'BEGIN''INT'CELL;
   26 'REAL'INCR,LWB,UPB,ANGLE,PI;
   27 CELL:=STRINGSYMBOL(0,CELLTYPE);
   28 PRCTEXT("UAK;SPE;62;");
   29 'IF'CELL=11'THEN''BEGIN'H:=400;V:=960;PRCTEXT("12;")'END''ELSE'
   30 'IF'CELL=12'THEN''BEGIN'H:=400;V:=480;PRCTEXT("44;")'END''ELSE'
   31 'IF'CELL=13'THEN''BEGIN'H:=200;V:=480;PRCTEXT("76;")'END''ELSE'
   32 'BEGIN'NLCR;PRINTTEXT("GRIDTYPE ERROR");EXIT'END';
   33 ALFB:=256;
   34 HFAC:=H*FRAC;STEP:=1/HFAC;
   35 PRCTEXT("0;0;UAT;UAT;");
   36 PI:=3.1415926535898;INCR:=PI/60;UPB:=PI/4+10-4;
   37 'FOR'ANGLE:=0'STEP'INCR'UNTIL'UPB'DO'GEN LINE CHAR PLUS MIC(ANGLE,1);
   38 LWB:=UPB-10-4;UPB:=PI/2+10-4;
   39 'FOR'ANGLE:=LWB'STEP'INCR'UNTIL'UPB'DO'GEN LINE CHAR PLUS MIC(ANGLE,2);
   40 INCR:=-INCR;LWB:=LWB-10-4;UPB:=UPB-10-4;
   41 'FOR'ANGLE:=UPB'STEP'INCR'UNTIL'LWB'DO'GEN LINE CHAR PLUS MIC(ANGLE,3);
   42 UPB:=PI/4;LWB:=-10-4;
   43 'FOR'ANGLE:=UPB'STEP'INCR'UNTIL'LWB'DO'GEN LINE CHAR PLUS MIC(ANGLE,4);
   44 ALFB:=PA;
   45 'END';
   46
   47 'PROC'GEN LINE CHAR PLUS MIC(ANGLE,TYPE);'VAL'ANGLE,TYPE;'REAL'ANGLE;
   48 'INT'TYPE;
   49 'BEGIN''REAL'TGA,COSA,SINA,B,P1,P2,I,DH,DV;
   50 'INT'DISPL,DISPL1,DISPL2;
   51 SINA:=SIN(ANGLE);COSA:=COS(ANGLE);TGA:=SINA/COSA;
   52 'IF'TYPE=1∨TYPE=4'THEN'
   53 'BEGIN'VD:=TGA;HD:=1;B:=1+.1*STEP;
   54 DV:=D/COSA;
   55 GEN CHAR HEAD(TYPE);
```

```
'FOR'I:=STEP'STEP'STEP'UNTIL'B'DO'
GEN BL2('IF'TYPE=1'THEN'VD*I-.5*DV'ELSE' 1-I *VD-.5*DV,
    DV,I>B-STEP,TYPE,0)
'END''ELSE'
'BEGIN'VD:=1;DH:=D/SINA;HD:=1/TGA;B:=HD+DH+.1*STEP;
DISPL:=DH*HFAC;
DISPL1:=DISPL2:=DISPL'/'2;
'IF'DISPL=2*ENTIER(DISPL/2)='1'THEN'DISPL2:=DISPL2+1;
GEN CHAR HEAD(TYPE);GEN LBL( - DISPL1/HFAC);
'FOR'I:=STEP'STEP'STEP'UNTIL'B'DO'
'BEGIN'P1:='IF'I'LE'DH'THEN'0'ELSE'(I-DH)*TGA;
 P2:='IF'I'LE'HD'THEN'I*TGA'ELSE'1;
  GEN BL2('IF'TYPE=2'THEN'P1'ELSE'1-P2,P2-P1,I>B-STEP,TYPE,DISPL2/HFAC)
'END';
'END';
GEN LMIC(VD,TYPE);INALFTABLE(ANGLE,HD,VD);
'END';

'PROC'GEN LBL(WID);'VAL'WID;'REAL'WID;
'BEGIN'PRCSYM(98);FIXTC(3,0,HFAC*WID);PRCSYM(99)'END';

'PROC'GEN BL2(W,Z,F,TYPE,DISPL);'REAL'W,Z,DISPL;'BOOL'F;'INT'TYPE;
'BEGIN''OWN''INT'OBL,OWH;'INT'BL,WH;

'PROC'CODE(WH,BL);'INT'WH,BL;
'BEGIN'PRCSYM(98);ABSFIXTC(3,0,WID);PRCSYM(87);ABSFIXTC(3,0,WH);
  PRCSYM(87);ABSFIXTC(3,0,BL);PRCSYM(99);
'IF'PRINTPOS>('IF'PUT=1'THEN'61'ELSE'125)'THEN'PRCSYM(119);
'END';
WH:=(FRAC*W+('IF'TYPE=1'THEN'.0625'ELSE''IF'TYPE=2'THEN'0'ELSE'.025))*
            V+1;
BL:=Z*V*FRAC;
'IF'BL=0'THEN''BEGIN'BL:=1;WH:=WH-1'END';
'IF'WID=0'THEN''GOTO'NEW;
'IF'WH=OWH^BL=OBL'THEN'WID:=WID+1'ELSE'
'BEGIN'CODE(OWH,OBL);
  NEW:  OWH:=WH;OBL:=BL;WID:=1
'END';
'IF'F'THEN''BEGIN'
              CODE(WH,BL);
              'IF'TYPE=2^TYPE=3'THEN'GEN LBL( - DISPL);
              PRCTEXT(";UAT;UAT;")
              'END';
'END';

'PROC'GEN LMIC(VD,TYPE);'REAL'VD;'INT'TYPE;
'BEGIN''INT'VDI,VDIR;
PRCSYM(119);PRCTEXT("UAK;SPE;");ABSFIXTC(3,0,PA+127);PRCSYM(91);
ABSFIXTC(3,0,256);PRCSYM(91);ABSFIXTC(3,0,2);PRCTEXT(";[");
ABSFIXTC(3,0,PA);PRCSYM(91);
'IF'TYPE'LE'2'THEN'PRCTEXT("UAK;VERO;")'ELSE'PRCTEXT("UAK;VERU;");
VDI:=VD*VFAC;'IF'VDI'GE'256'THEN'
'BEGIN'VDIR:=VDI'/'256;ABSFIXTC(3,0,VDI-VDIR*256);PRCSYM(91);
  ABSFIXTC(3,0,VDIR)
'END''ELSE'
'BEGIN'ABSFIXTC(3,0,VDI);PRCTEXT(";0")'END';
PRCTEXT("];UAT;UAT;")
'END';

'PROC'GEN CHAR HEAD(TYPE);
```

```
116   'BEGIN'PRCSYM(119);PRCTEXT("UAK;SPE;");ABSFIXTC(3,0,NEXTPA);
117   PRCTEXT(";256;");ABSFIXTC(3,0,'IF'TYPE=1'THEN'0'ELSE'
118                 'IF'TYPE=2'THEN'1'ELSE'32); PRCSYM(91);
119   WID:=0
120   'END';
121
122   'PROC'INALFTABLE(A,H,V);'VAL'A,H,V;'REAL'A,H,V;
123   'BEGIN'ALFTAB[PA,1]:=A*180/3.14159265;
124   ALFTAB[PA,2]:=H*7.2;ALFTAB[PA,3]:=V*7.2
125   'END';
126
127   'PROC'DUMP ALFTAB;
128   'BEGIN''INT'L,U;U:=PA;PA:=-1;
129   NLCR;PRINTTEXT("  PA    ANGLE      HD        VD");
130   'FOR'L:=1,L+1'WHILE'L'LE'66'DO'
131   'BEGIN'NLCR;ABSFIXT(3,0,L);ABSFIXT(3,2,ALFTAB[L,1]);
132     ABSFIXT(3,2,ALFTAB[L,2]);ABSFIXT(3,2,ALFTAB[L,3]);
133   'END'
134   'END';
135
136   'PROC'BYTE(X);'VAL'X;'INT'X;
137   'BEGIN'ABSFIXTC(3,0,X);PRCSYM(91)'END';
138
139   'PROC'BBYTE(X);'VAL'X;'REAL'X;
140   'BEGIN''INTEGER'X1,X2;
141   X1:=ENTIER(X/256);
142   X2:=X-X1*256;
143   'IF'X2=256'THEN''BEGIN'X1:=X1+1;X2:=0;'END';
144   BYTE(X2);BYTE(X1);
145   'END';
146
147   'PROC'BBCOM(S,V);'VAL'V;'STRING'S;'REAL'V;
148   'BEGIN''IF'PRINTPOS>('IF'PUT=1'THEN'59'ELSE'123)'THEN'PRCSYM(119);
149   PRCTEXT(S);BBYTE(V)
150   'END';
151
152   'PROC'GENWID(N,PA);'INT'N,PA;
153   'IF'N>0'THEN'
154   'BEGIN''IF'N>1'THEN'BBCOM("UAK;WID;",N-1);BYTE(PA)'END';
155
156   'PROC'GENZWQR(X);'VAL'X;'REAL'X;
157   'IF'X>.01'THEN''BEGIN'BBCOM("UAK;ZWQ;",X*50);PRCTEXT("ZWR;")'END''ELSE'
158   'IF'X<=.01'THEN'BBCOM("UAK;RUCK;",-X*50);
159
160   'PROC'GEN VD(X);'VAL'X;'REAL'X;
161   'IF'X=0'THEN' 'ELSE''IF'X>0'THEN'BBCOM("UAK;VERO;",X*32)'ELSE'
162     BBCOM("UAK;VERU;",-X*32);
163
164   'PROC'MOV(XP,YP);'VAL'XP,YP;'REAL'XP,YP;
165   'BEGIN'GEN ZWQR(XP);GEN VD(YP)'END';
166
167   'PROC'REPOSET;
168   'BEGIN''IF'PRINTPOS>('IF'PUT=1'THEN'59'ELSE'123)'THEN'PRCSYM(119);
169   PRCTEXT("UAK;VERA;UAK;ZEV;0;0;")
170   'END';
171
172   'INT''PROC'NEWPA(X);'VAL'X;'INT'X;
173   'BEGIN'PA:=X;NEWPA:=NEXTPA'END';
174
175   PUT:=0;
```

```
176   FRAC:=.225;
177   VFAC:=1024*FRAC;
178   D:=.1;
179   PA:=-1;
180   GEN TYPE FOUNT("C");
181   DUMP ALFTAB;
182   'END'
```

```
'BEGIN''COMMENT'EXERCISE4;
'INTEGER'PX,PY,I,J,T,TB,AXIS;'REAL''ARRAY'HD,VD[1:66];
'REAL'DX,DY;
'INTEGER''PROCEDURE'REMAINDER(A,B);'INTEGER'A,B;
REMAINDER:=A-B*ENTIER(A/B);
'INTEGER''PROCEDURE'NEXTAXIS;
NEXTAXIS:='IF'AXIS=65'THEN'1'ELSE'
                        (
                        'IF'AXIS=15∨AXIS=27∨AXIS=48∨AXIS=61'THEN'AXIS+2
                        'ELSE'AXIS+1
                        );
PUT:=1;
   PRCTEXT("UAK;START;UAK;SEG;0;UAK;GROA;UAK;TAB;0;0;UAK;VERA;");
BBCOM("UAK;ZEV;",450*32);
'FOR'I:=1'STEP'1'UNTIL'66'DO'
'BEGIN'HD[I]:=ALFTAB[I,2];
       VD[I]:='IF'I<34'THEN'ALFTAB[I,3]'ELSE'-ALFTAB[I,3];
'END';
PX:=34;PY:=50;
'FOR'T:=1'STEP'1'UNTIL'46'DO'
'BEGIN'
    'IF'REMAINDER(T,5)=1'THEN'
    'BEGIN'TB:=72;BBCOM("UAK;ZEV;",32*144)'END''ELSE'TB:=TB+144;
    BBCOM("UAK;TAB;",50*TB);
DX:=1.25*HD[PX];DY:=1.25*VD[PX];
'FOR'I:=0,1,2,3,4'DO''BEGIN'REPCSET;MOV(DX*I,DY*I);GEN WID(5,PY+127)'END';
    DX:=1.25*HD[PY];DY:=1.25*VD[PY];
'FOR'I:=0,1,2,3,4'DO''BEGIN'REPCSET;MOV(DX*I,DY*I);GEN WID(5,PX+127)'END';
AXIS:=PX;PX:=NEXTAXIS;AXIS:=PY;PY:=NEXTAXIS
'END';
BBCOM("UAK;ZEV;",450*32);PRCTEXT("UAK;STOP;");
'END'
'END'
```