LASER

Home-Computer

7

Software System Manual 1

SANYO Video Sales, Hamburg

Wolfgang Radeloff

LASER

Home Computer

Software System
Manual

SANYO Video Sales, Hamburg

Imprint:

COPYRIGHT 1984 by SANYO Video Distribution, Hamburg
Publishing support and sentence: Roland Löhr, Ahrensburg
Design of book covers and graphic works: Cathrin Utescher, Hamburg Print and
production: Kuncke Druck GmbH, Ahrensburg

Introduction

In 12 chapters and a comprehensive annexe, the book you are given here offers all the information that was missing from the BASIC manual of LASER computers.

The system provides the user with a wealth of other services that can now be used with knowledge of the operating system. These include: Compute with 16 significant digits, ON ... GOTO and ON ... GOSUB, FRE(0) and finding variables in memory. Binary files can now be saved and started, the connection of peripherals via the USER port is documented.

In addition, the book answers the questions that have been put to the author repeatedly on the phone by the users. Together with the BASICH manual, a better level of documentation and handling has already been achieved. Questions still outstanding can be dealt with in a following book.

For guidance:

Before working with this book, you should always consider the following three points:

1. There are two software versions for the computer described here (LASER 110/210/310 and VZ 200): LASER-BASIC verse. 1.1 and verse. 2.0. The grave difference is the composition of the image in the video RAM. Verse. 1.1 describes the RAM with normal characters while verse. 2.0 initializes the screen with inverted characters. The book always refers to verse 1.1.

   If you have the BASIC version 2.0 in your device, hold down the CTRL key while the monitor is on, and the screen will look like verse. 1.1 and all examples in the book will work properly.

2. A lot of reference is made in this book to the BASIC-UP extension. If you own this cassette, you should know that 'BASICUP' only works with BASIC programmes created with 'BASIC-UP'.

3. "EXTENDED BASIC", another BASIC TOOLKIT, complements 'BASIC-UP' with an extensive graphic instruction set. CIRCLE, REGT, PAINT, PLOT. TO... , GCLS and LPEN for the light handle are just a few examples.

The book is structured in such a way that the individual chapters are closed in themselves and can be worked through without regard to the order.

Hamburg, July 1984

Table of Contents

1 Numbers, characters, variables. **7**

digit and character formats,
Types of variables
LET, DEFINT, DEFSGN, DEFDBL

2 Fields, organised storage 13

field variables: Tables and files lists
DIM

3 65536 memory is managed. . . . . **19** EPROM

User Programmes
Operating System Pointers
With NEW Deleted Programmes Recover Free Bytes
CLEAR, NEW, FRE (0), FRE (""), MEM, VARPTR

4 PEEK and POKE and the integer format 27

Convert natural integer numbers
in signed integer numbers

5 Comments on PRINT and LPRINT 29

PRINT, ";", ",", PRINT AT, PRINT USING, LPRINT

6 Helping the programmer **38**

Editing, Listing, Deleting, Fault Search,
Structured programmes

7 Text Graphics, Function, and Application 44

- 2000 characters resolution, quarter graphics A quarter learns to walk

8 More graphics and sound 53

The character **"",** INVERS, Cursor control,
COLOUR with the LASER 110,
FASHION and background colour, BUZZER on-off

9 Peripheral control with INP and OUT ..... ・. 57 user port:

Control with INP and OUT
Circuits and Examples in BASIC
The 64KB RAM expansion

10 Programming of joysticks 65
With INP (X) and ON ... GOTO

11 Machine-related programming 69
Creating Programmes
Memory Deployment
Protection against BASIC
CSAVE, CLOAD and CRUN for ML programmes

12 For **the assembler programmer 78**
Keyboard Query, CRUN/CLOAD
Character to screen, string output,
Compare symbol, test next character,
Compare DE/HL Test Variables Type, Control Codes,
Joysticks, BUZZER, BEEP,
printer control

Attachments 86

List of Annexes

1 LASER I/O circuit statement
2 LASER 110/210 schematic (5 diagrams)
3 Plug Occupancy System-Bus, Peripheral-Bus
4 LASER 210 Extension 6 KB, Circuit
5 Printer Interface, Circuit
6 Joystick, Circuit
7 LASER cassette recorder, circuit
8 LASER 110 Upgrade to Colour Computer
9 Variable formats
10 Memory mapping
11 Pointers in the operating
system 12 List of system
variables
13 Advanced ASC II Code, Screen Code
14 Geometric Functions with LASER-BASIC 15
Short notation
16 BASIC text format
17 BASIC Tokens
18 Tape Loading Format
19 Comparison of BASIC dialects

1 numbers, characters, variable

Number and character formats

Variable types

LET,DEFINT,DEFSNG,DEFDBL

Your LASER computer will process numbers that are not writable in the normal way. The number range is, expressed in terms of tens, greater than 119-39 and less than 1139, In the computer such numbers are entered in the form: 1 E38 or 1 E-38. The number 1812 is written in this technical-scientific representation as 1.812E3. Of course, the computer also understands normal spelling. If, however, it is intended to print a number equal to or greater than 9999999.5, it will normally switch to the technical-scientific format (1.2). If the number is less than 0.0099, the technical-scientific format shall also be changed.

```
? 2.33 E 3 <RETURN> 2330
```

1. 1

```
? 3470828 <RETURN> 3.47083 E+06
```

1. 2

Single Precision

This output is made up of six digits and the remaining digits are rounded (1.3).

This type of number representation is called a Floating Point (Single Precision). After power-up, the computer is in this type of presentation. All calculations must pay particular attention to the rounding error!

If the rounding error is not acceptable and a greater accuracy of the calculation result becomes necessary, then 'Double Precision' can be changed. The numbers are defined for input or in the programme in technical science, but the 'E' is defined by a 'D'

(1.4). The number representation is now 16 significant (valid) digits, and the rounding error is negligible for most applications.

```
7 0.1234567 <RETURN> . 123457
                                1.3.

2 1/6 D 0.16666666666666667
                                1.4
```

```
 VARIABLE TYPE TABLE
=
30977  .......A
30978  .......B


31001  .......Y
31002  .......or
                                1. 5
```

variable storage

So far, we have directly regarded the number representation as 'constants'. The programme stores numbers in memory. There, they can be modified and manipulated in the programme run. These stored numbers are called 'variable'. Such numeric stores, called numeric variables, are always created by the programme for single precision numbers without special precautions. In order to store numbers in the format of double accuracy, an intervention in the variable handling of the BASIC interpreter is necessary: From address 30977 the interpreter has created a table of 26 bytes in length. Each table position corresponds to a letter of the alphabet (1.5). By entering key numbers (1.6) using the POKE statement

```
FLAG  VARIABLE  TYPE

2  =  INTEGER NUMBERS

3  =  CHARACTERS

4  =  SINGLE  PRECISION
8  =  DOUBLE PRECISION
                                1.6.
```

can be explained (declared) in groups for a corresponding variable type.

The string variable and the length variable can now be addressed in the programme by omitting the type identifier. Expressions such as **Y** = 12 instead of **Y%** = 12 or A="STRING" instead of AS = "CHAIN OF CHARACTER" are then possible.

However, designators remain the priority. If the group of variables B is defined for adding numbers, the string store BA'/ will still be created primarily for recording strings.

```
10 POKE 30978,8
20BZ = 1 /6D
30 PRINT BZ
RUN .166666666666
66667
                                1. 7
```

Double precision variable

For example, POKE 30978,8 sets up all the variables that start in the name with 'B' to record numbers in 'Double Precision' format. These are names like B, B9, BZ, B(5) or BILD.

Note, however, that assignments to a double-precision variable in the technical-scientific format must be made with the 'D' as an exponent identifier (1.7) ! If in the example (1.7) the D (D0) is omitted, the wrong result is 1/6 = .166666716337204.

However, double-precision operations in the programme should only be performed where the increased accuracy of the result is required. The example (1.8) needs approx. 100 multiplications of double accuracy.

```
10 POKE 30979,8: 'DBL
20 C1 = 1. 5D
30 FOR I=1 TO 100
40 c2 = C1 + 1,234
50 NEXT
60 PRINT C2
++k 12 SECONDS ++k
                                1. 8
```

```
10 C1 = 1. 5
20 FOR I=1 T0 100
30 C2 = C1 1 234 40
NEXT
50 PRINT C2
kk+ 2 SECONDS
```

1.9.

12 seconds, but with Single Precision only 2 seconds (1.9). In addition to the longer run time, the increased memory space must be taken into account for a variable of double accuracy as a disadvantage (1.10).

```
INTEGER          5 BYTES 7
SINGLE PREC:     BYTES 11
DOUBLE PREC:        BYTES
STRING           6 +LONG) BYTES
                              1 1
```

```
AX= 12
```

1.11

integer variable

A third variable format allows only integer numbers in the range of -32768 to +32767. Small memory requirements and fast processing are the advantage of targeted application of these 'integer variables'. Such variables can be set up by means of tables (1.5) and (1.6) or by identifying the name of the variable with a trailing % (1.11).

Changing the Format

Variables can be changed from one format to another. However, restrictions must be observed:

* Integer Float: Cut the decimal places. There is no rounding. Example:
A% = A (1.12).

Single Precision to Double Precision: A previously encountered rounding error will not be reversed. Example: POKE 30978,8: B = **A.**

Double Precision to Single Precision: It is rounded. Example: POKE 30978,8 A = B.

A floating-point format conversion to Integer is also possible with the expression A = INT(B).

```
10 A = 12.25 20
AZ = A
30 PRINT A
RUN
  12
```

1.12

BASIC UP

The BASIC extension 'BASIC UP' eliminates the configuration of the variable type via the table in the system RAM. Instead, the instructions are:

```
DEFINT
[IDEFSN
G
DEFDBL
```

for the initialisation part of the programme.
A list of variable start characters should be attached. Any variables that are the first to list these characters in the name are thus initialised for the corresponding type (1.13). The integer variables with % in the programme text may not be labelled if a corresponding agreement has been reached via the table or BASIC UP.

```
NEW INSTRUCTIONS
IDEFINE A,W,Z
IDEFSNG B,C,M
DEFDBL D,E,M-P
```

1 13

String Strings

The fourth variable type: The string or 'STRING'. This type of variable can be used to assign strings (digits, letters, characters, and graphic characters), displayed inverse or normal, to their storage codes. A '$' character (dollar) is assigned to these variables to identify in the name. These string variables can also be pre-defined via the table, your key number is '3'. See the character codes in Appendix 13.

```
AS = "TEST"
184 169 183 184 1
   STORAGE
                                    1. 14
```

BOOL's variable

This type of variable is the last one mentioned. It records the result of a comparison or a combination of two or more variables. It's also called the Truth Variable.

This type of variable is usually expressed as I F ... and with the instructions according to TH EN ... and ELSE... immediately. For special applications, it can be assigned to a numeric variable (Integer) and evaluated in later IF ...THEN statements.

```
BOOL*SCHE VARIABLE

TRUE        -1 BZ. # 0 0
WRONG:
                                    1 .45


10  A=11:  B=12:  C=12:  D=13
20  X=  B=C:  Y=  A>B:  Z=  A<D
30  PRINT X;Y;Z
RUN
-1 0 -1
                                    1 16
```

Example 1.17: A variable is tested to zero. The programme branches when A is nonzero.

Example 1.18: The programme branches when A is 0.

```
IF ATHENS G0T0 500
                                    1,

IF NOTA THEN GOTO 500                1


                                    1.18
```

2 fields: Organised Storage

Field variable: Lists, tables and files DIM

All types of variables mentioned in chapter 1 can be organised into memory fields. The array gets a name (2.1). The individual space within the field is indicated by an index in brackets (2.2). This index can be a constant, a numeric variable, or a numeric expression (2.3).

The advantage of this organisation: Memory slots are no longer addressed by directly naming their names, but by computer-controlled operations. Example 2.4 prints the contents of 10 variables to the screen.

```
Field variable: Name
=================
AB, D$, E1%
                                2.1.

Index field variable
=====================
AB5), D$@), E1%12)
                                2

Field variable with index constant,
variable, expression
%
AB(5), D$(J), E1%(INT(J/3))
                                3

10 FOR 1=1 TO 10 20
PRINT ABI
30 NEXT I
                                2.4.
```

If a value is assigned to an indexed variable with LET in a programme, the LASER will automatically create an 11-space memory field and place the value assigned with LET in the corresponding space **(2.5).** On the one hand, for fields with less than 11 memory slots, this means non-economic storage management (some seats are not occupied), and on the other hand, the BASIC

```
10 LET ABS)=12.5
RUN
AB(0) = 0
AB(1) =

AB5) = 12.5

(AB10) = 0
```

2 <

Interpreters for fields with more than 11 seats will be assigned an allocation agreement, so for the sake of clarity of the programme, the automatic installation for small fields should be waived.

Instead, the BASIC keyword DIM (DIMensioniere) is used to define the storage organisation in its size for each field. The statement (2.7) creates 16 slots under the variable name AB, starting with AB(O) to AB(15) and ending at zero.

Multiple fields can be initialised with a statement. Named fields are defined by their dimensioning in a list, separated by commas, according to the word DIM (2.8). The D IM statement must always precede assigning values to field variables in the programme. So it belongs in the initialisation part of the programme. A field can only be defined once in the course of the programme using **DIM**. An attempt to size a field under the same name with a different extension is prevented by an error message and a stop of the programme run. (2.9).

DIMensioners

2.6.

```
10 DIM AB15)
```

2.7.

```
10 DIM AB(15),D$(5),E1%(20) 2.8
```

```
10  DIM  A$)
2  DIM A$5)
RUN
?REDIM'D ARRAY ERROR IN 20
```

2 9

An additional error message from the operating system is reported whenever a field variable with an index outside the dimensioned range is accessed (2.10).

```
10 DIM A(5)
20 PRINT A12)
RUN
?BAD SUBSCRIPT ERROR IN 20
```

2. 10

Tables and files

So far only fields in the form of a list have been considered (2.5). The LASER allows for two further organisational forms: These are fields in table form that allow the variables to be arranged by row and column, and fields in the form of a file in which each card is once again created in table form.

The field in table form, also called 'two-dimensional field' or 'matrix', is addressed by the field name with two indexes. The first one will name the row of the table, the second one will name the column (2.11 and 2.12). Accordingly, the organisation requires three index numbers to be entered in files: Column and row of table and number of card. The storage organisation extends here in three dimensions (2.13, 2.14).

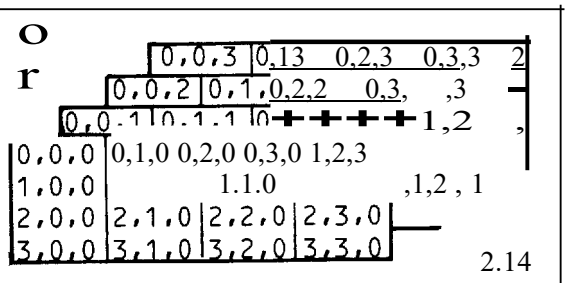
0.0	*0.*1	0.2	l.3
1 , (l)	1 , 1	1, 2	1.3
2.0	2.1	2.2	2.3
3.0	3.1	3.2	3.3

```
1 DIM E1 3,3) 20
E112)=12
30 PRINT E11,2)
RUN
12
```

2 12

```
10 IN Z3,3,3) 20
2%1,0,10=12
30 PRINT Z%1,0,1)
RUN
12
```

2. 13



2.14

Limits only by available memory space

A very high number of elements can be programmed per field, which is limited by the available storage space on the one hand, but on the other hand the LASERBASIC allows only a certain extent of fields. The limit for the corresponding types of variables is given in Table 2.15. They were determined for the LASER 110. The values in this table only describe the maximum allowable dimensioning in the D IM statement, an error message will occur earlier depending on the storage size.

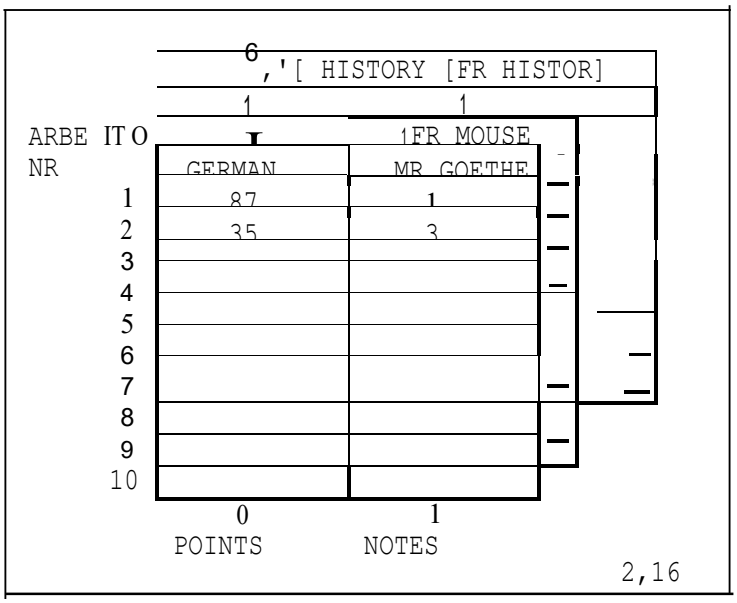
DIM dimensions everything!!

The D IM statement not only allows the creation of fields of the type described, but also creates the entire variable table by naming the variable name according to D IM. (See also Chapter 3 and Annexe 9.) This makes it possible to create a table of variables in the initialisation part of the programme and thus influence the processing speed of the interpreter.

DIM A, Z%, CIS, F(5)

Table 2.15	Limit of field elements			
	INTEGER	SNG.PREC.	DBL.PREC.	STR ING
LIST	17000	8509	4250	11347
TABLE	129129	9191	664	105105
CARTEI	232323	19+19%19	151515	212121

This statement sets the variables A and Z% as the first and second with the value zero in the table of variables, the third with the string variable C1$, and then dimensions the field F with 6 elements.




```
10 'INITIALISATION AND MENUE 20
CLEAR 1000: DIM NO$10.1.6) 40
RESTORS: CLS
50 FOR I= TO 6: READ N0$(0,0,I),N0$(0,1,I)
60: PRINT I-:-1;"=";N0$(0,0,I),NO$C0,1,I)
70 NEXT: PRINT
100 PRINT"WHICH TRAY";: INPUT K
110 IF K<1 OR K>7 THEN PRINT CHR$27);CHR$(27): G0T0100 115
K=K-1
120 PRINT
121  PRINT"  1  =  INPUT"
122 PRINT" 2 = OUTPUT"
129 PRINT" YOUR CHOICE";: INPUT L
```

```
130 IF L<1 OR L>2 THEN PRINT CHR$(27);CHR$(27): GOT0129
140 IF L=1 THEN 300
1 4 1 I F L = 2 TH E N 60 0
300' ENTER
310 CLS: PRINT N0$(0,0,K),N0$(0,1,K): PRINT
320 PRINT" WORK NO. "
330 PRINT’ POINTS "
340 PRINT" NOTE
350 PRINT@85,’; INPUT I
355 IF I>1O THEN 350
360 PR 1NTa117,’’;
INPUTP$ 370 PR 1NT6149,"";:
INPUT NS
380 NOS(I,OK)=PS: NOS(I,1,K)=NS
390 PRINT: PRINT"MENUE YES/NO)". INPUTK$ 400
IF LEFT$(K$,1)="N" THEN 300 ELSE 40
600 'OUTPUT
610 CLS: MP=O: MN=O: Z=O
620 FOR I=0 TO 10: PRINT NO$(I,0,K),NO$(I,1,K): GOSUB 900: NEXT
630 Z=Z-1: IF Z=O THEN 800 ELSE PRINT MP/Z,MN/Z: GOT0800
800 'RETURN TO MENUE
810  PRINT"  MENUE  YES/NO)  ",.  INPUT
K$ 820 IF LEFT$(K$,1)="J" THEN 40 ELSE
END
FORM 900 'AVERAGE VALUES
910 IF NOS1,O,K)>"" AND NO$I,1K)" THEN Z=2+1 920
MP=MP+VAL(NO$I,O,K): MN=MNVAL(NO$1,1,K))
990 RETURN
1000 DATA ENGLISH,MR GOETHE,ENGLISH,FR MOUSE
1020 DATA FRANZOES,FRL PARIS,BIOLOGY,MR BAUM
1040 DATA PHYSICS,MR OHM,MATHE,MR A RIESE 1060
DATA HISTORY,FR HISTOR
```

2/17

The programme 2.17 uses a three-dimensional field { file) **{2.16** for placing 10 censorship in 7 compartments**.** Take special care with the handling of the field:

Row 20 dimensions the field.

Row 50 reads the heading to row 0 of each table.

Row 310 gives the heading of the selected table. Row 380 reads values to a defined **row**.

Row 620 gives the contents of a table.

Row 910 checks if a line is described.

Row 920 adds the contents of a row.

3,65536 Memory Manages

EPROM user programmes, pointers in the operating system;
Recover programmes deleted with NEW; Free Bytes

CLEAR, NEW, FRE (0), FRE (''), MEM, VARPTR

The LASER computers (110, 210, 310, VZ 200) are organised as Z-80 systems. Appendix 10a gives an overview of the physical memory distribution I ung of the computer. Appendix 1 Oe supplements it with the location of the memory extensions. These memory slots are managed by the operating system and the BASIC interpreter, which itself occupy 16 KB memory space **(RÖM 1, ROM 2).**

```
          NAMF GAMF
STACK     EQU 7FFFH
CRTDGE    EQU 4000H
          ORG 787DH
INTVTR    DEFS 3              ;INTERRUPT EXIT LOC

          ORG CRTDGE
           DEFB 0AAH          ; 1ST  PATTERN
           DEFB 055H          ;2ND   PATTERN
           DEFB 0E7H          ;3RD   PATTERN
          DEFB 018H           ;4TH   PATTERN
START      LD SP,STACK         ; RE-INITIAL IZE   STACK
           LD A,0C3H           ;JP 0BJ CODE
           LD (INTVTR) ,A
           LD HL, INTSVC       ; GET  SERVICE   ROUTINE  ADDR
           LD ( INTVTR-:-1) ,HL          ;MOD IFY EX IT LOC



                               ;USERS THIRD INT SERVICE ROUTINE
INTSVC     CALL KEYBRD

          POP HL               ;CLEAR
          POP HL               STACK ;RECOVER HL
          POP EN               REG ;RECOVER DE
          POP BC               REG ;RECOVER BC
          POP AF               REG :RECOVER AF
          EI                   REG ;ENABLE
          RETI                 INTERRUPT
```

EPROM programmes

After power-up, the operating system will check if the bit patterns AAH, 55H, E7H and 18H are present from the addresses 4000H, 6000H or 8000H. If they are detected in one of these areas, the operating system passes control to the user programme beginning after these four bit patterns. Programme 3.1 is an example of how to integrate your own programmes.

Pointer in the **operating system**

The BASIC interpreter manages its RAM range (7AE9H to FFFFH) via a number of 'pointers', which are created together with other system variables after switching on in the RAM range (7800H to 7 A E8H). See Annexe 11.

Pointers are two-byte wide registers in which the BASIC interpreter remembers the addresses of the data and programme text areas it needs to manage.

For the BASIC programmer, the hands HP (start of the BASIC text) and TOM (pointer to the last RAM cell) are of particular interest. A conversion of these pointers creates storage space protected against BASIC to store machine programmes. The Z-80 BASIC versions are usually protected by the TOM (Top Of Memory) pointer. See also chapter 15!

CLEAR

The CLEAR statement deletes all variables and dimensioned fields. The reservation will be cancelled. CLEAR is without parameters. Bookings for strings are not affected.

```
10 CLEAR 1000
```

3.2

```
PRINT PEEK (30898)
```

3.3.

Appendix 11 shows the memory area for storing strings and the corresponding pointer STSP ,String Space). After the computer is turned on, 50 bytes are reserved for string storage. With the CLEAR statement

and the corresponding parameter, the BASIC programmer can set up this string area for the needs of his programme (3.2).

If the programme should run in all possible combinations of LASER computers and memory extensions and if maximum space is required for the filing of strings, then the pointer TOM offers itself as a measure of the size of the string storage area to be dimensioned with CLEAR.

The pointer's MSB (Most Significant Byte) contains the page number of the highest RAM cell detected in the memory test. The statement (3.3) should be used to determine the number of pages that apply to the system (3.4).

If the programme should universally size its string memory area itself, the programme line (3.5) can be used. The constants mentioned in the constants are intended to specify the storage requirements for the device combination with the lowest RAM range.

```
78B2H      30898    (MSB - TOM)
```

```
READ 110        7FH   127
VZ 200          7FH   127
LASER 210       8FH   143
L 110 + 16KB    BFH   191
VZ200+ 16KB     BFH   191
L210 + 16KB     CFH   207
LASER 310       B7H   183
LASER310  + 16KB

                FFH   255
64KB            FFH 255

                              3 4
```

```
20 CLEAR 1000+(PEEK30898)-127)+256 3.5
```

NEW

It is easy to see what the NEW statement does, using the graphic in Annexe 11: It resets all pointers to the value after the computer is turned on. The programme is still present, but the BASIC interpreter has 'forgotten' it.

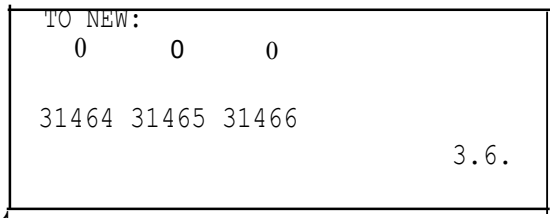
An accidentally deleted programme can be remembered by some POKE instructions to the interpreter. Only the knowledge of how the text tray is organised and managed is necessary. Annexe 16 shows **this** in graphic form.

Each instruction line starts with a 0 byte. This is followed by a two-byte-wide pointer to the beginning of the next line. The line number is then stored in two bytes, followed by the BASIC text of the line. All keywords are encoded into a byte that gets > 127 and entered into memory. These encoded bytes are also called 'tokens'. All other characters are stored with their ASC II code, which is always smaller than 128.

See Appendix 17 for a table of keywords and their code numbers. Annexe 13 shows the ASCII code. After the first statement line, a '0' byte follows the beginning of the second line. The end of the BASIC text is marked by setting the two bytes the pointer of the last line points to 0. A BASIC programme text is thus always completed in memory with a sequence of three times '0'.

If there is no programme in memory, i.e. after NEW, the '0' in cell 74E8H (marking first line) **will** follow **'O' twice more (3.6). The pointer TP (end BASIC text/** beginning variable table) points to NEW to the first byte after the three null bytes.

To restore a deleted programme, just point the pointer in the BASIC text to the second line and point TP to the first byte after the three zero bytes at the end of the earlier BASIC text. In the following, the



3.6.

10 A=12
20 PRINTA

3.7.

short programme 3.7. The programme should be entered, and then the statement (3.8) should make visible the BASIC code generated in the memory. It shall correspond to the code set out in Annexe 16. Especially the values of the pointers are to be observed. The pointer of the first line in the text points to the first byte of the pointer of the second line. All the lines of the BASIC programme are linked together, and the interpreter is 'brimming' through the text using these pointers to process the programme. The TP pointer points to the first byte after the end of the text.

```
FOR 1=0T018:2PEEK31464+I ; :NEXT 3 8
```

Restore programme deleted with NEW

The programme (3.7) will now be deleted with 'NEW'. From the structure of the BASIC text can be inferred to the approximate position of the pointer of the second line. With PEEK instructions from the keyboard, his address is now accurately determined. If the '0' marking the beginning of the second row is found, it is clear that the deleted pointer of the first row must point to cell 31474 (3.9). The address thus determined must now be decomposed into the lower and higher value part {3.10). The two decimal numbers thus obtained are then entered into the two cells of the pointer using the POKEA statement (3.11). This restores the pointer that was deleted by NEW.

```
2PEEK  (31472)
50
2PEEK  (31473)
0
```

3 9

```
? INT  (31474/256)
122
2 31474-(122256)
242
```

3 10

```
POKE 31465,242
POKE    31466,
122
```

3.11

The next step is to restore the TP pointer to the text end. Here, too, the presumed location of the text end in memory must be determined using PEEK instructions. In no case should any loops be formulated to search the memory, since the associated installation of variable stores would further destroy the BASIC text. If the text end (three times '0') is found, the pointer can be restored at address 30969, 30970. The process is the same as in (3.10, 3.11). Note that the PEEK statement as an argument from address 32760 must have negative integer numbers. A conversion utility shows (4.7).

Free Bytes

The LASER-BASIC does not know the FRE (0) statement to calculate the still free RAM space. However, this information should be derived from the values of the pointers that manage the RAM area. Appendix 11 shows that the difference between the FSL and STSP hands is similar to the free range of RAMB. The short programme (3.12), appended or included in the programme in progress, will output the number of bytes remaining free after the call. The stack area at run time of the programme is not recorded.

The free space calculation routine is present in the BASIC interpreter, only the corresponding keyword is not recognised and leads to the output of a '0', since FRE is interpreted as a variable name (3.13).

```
100 ST=PEEK(30881) 256+PEEK30880)
110   FS=PEEK30974)256+PEEK(30973)
120 PRINT "EREE=";ST-FS
```

3 12

```
FRE (0)
```

3,13

```
500 PRINTPRINT(0)
```

3 14

```
POKE 31470,218
```

3-15

FRE (0)

However, you can enable the feature in the following ways:

Enter a dummy line as the first line of the programme (3.14). Annexe 16 shows the structure of the first line. The two PRINT statements will appear as tokens with the code number 178 in the two RAM cells **31469** and 31470. If the token in 31470 is replaced by the token for FR E, the BASIC interpreter will read the line as PRINT FRE (0) and execute it correctly.

The token (encrypted command byte in memory) for FRE is 218. The statement (3.15) replaces the token 'PRINT' with that of 'FRE'. The line will not be listed correctly after this manipulation, but will be executed (3.16, 3.17). The user programme can now be written. All editing functions will include line 500. Other rows can be arranged before and after this row.

The F RE (0) notation of the function returns the free space between the FSL and STSP pointers. The value depends on the size of the programme text, the number and type of the variables, and the string area reserved with CLEAR.

```
LIST
500 PRINT
READY
```

```
RUN                                    3 17
12660
READY
```

The Trash Collector

The FR E ("") spelling determines the space available for strings, revises the string area, and removes unnecessary strings. The function included in the programme creates more space during the running time of the programme (3.18).

```
500 PRINTPRINT(")

POKE 31470,218

100 CLEAR 1000

LIST
100 CLEAR 1000
500 PRINT
READY

RUN
1000
READY
```

< 18

```
3FRE (@) ♫
FRE ("")
MEM
```

3 19

MEM

The statement MEM without parameters returns the free space like FRE (0).

BASIC UP

The BASIC extension BASIC-UP allows the previously described instructions to be included in the programme. However, before each statement, the character [SHIrijYY] must be given (3.19).

Another feature that uses the RAM management pointer is the VARPTR function. In the form (3.21), it provides the address of the variables named as arguments, see also Annexe 9.

VARPTR

Here too, the token 192 for VARPTR can be used without BASICUP. The statement can be implemented using the procedure shown in (3.14-3.18).

The statement (3.22) turns off BASIC-UP. R AM management pointers are re-initialised.

3 20 1

```
VARPTR (A$)
```

[INTB PR VARPTR CA$)
3.21
!'iJSYSTEM
3,22

4 PEEK and POKE and the integer format

Convert natural integer numbers to signed integer numbers

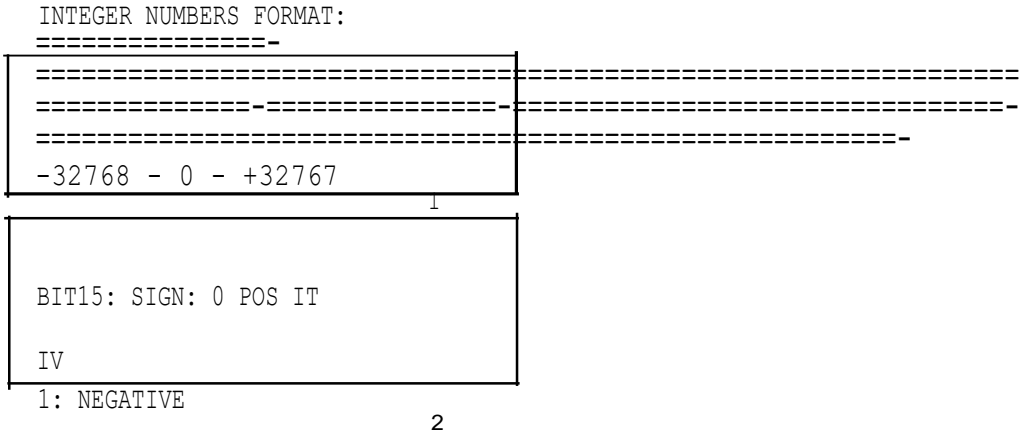
The instructions PEEK (AD) and POKE AD,VALUE require the address specification in integer format. Chapter 1 briefly describes this format: It represents numbers without a decimal fraction that are encoded in two bytes in memory.

With two bytes, corresponding to 16 digits binary, numbers can be represented in the range of 0 to 65535. In this format, numbers are processed internally by the operating system.

The integer format of the BASIC interpreter is divided into negative and positive numbers to support arithmetic (4.1). The sign is the highest value bit 15 (4.2). With the bits 0 to 14, numbers from 0 to 32767 can be displayed. Negative numbers are encoded with set bit 15. They are presented as two-way complement.

From the point of view of the programmer, who wants to read or change the memory cells with the instructions PEEK and POKE, and who therefore needs a number range of 0 to 65535, negative numbers are not much to start with.

The numbers 0 to 32767 (7FFFH) are internally converted into 16 bits



are shown normally and can be used as an address specification (4.3). The next higher number 32768 (8000H) is retained with its value, but the set bit 15 gives a negative sign to avoid the otherwise not very useful value -0.

The following number 32769 (8001 H) is converted as follows: From bits 0 to 14, the two complement is formed via the one complement, bit 15 becomes the character
negative'. Bit 0 to **14** represent the value 32767, with bit 15 then -32767 (4.5).

Based on these findings, natural numbers, hexadecimal numbers and the numerals of the arithmetic integer format can be compared in (4.6). Programme (4.7) is used to convert natural numbers into integer format.

BINARY FIGURES:

```
=======-=
    0 0000 0000 0000 0000
32767  0111  111  111  111
1111
```

or

32768 1000 0000 0000 0000 4.4

32769 (BIT 0-14) 000 0000 0000 0001

```
1ER COMPLEMENT 111 1111 111 110 2ER
COMPLEMENT +1


                 111,111  1111  1111
VALUE            32767
SIGN            1
NUMBER          -32767
                         4.5.
```

0000	0	0
0001	1	1
7FFF	32767	32767
8000	-32768	32768
8001	-32767	32769
FFFE	-2	65534
FFFF	-1	65535

HEXADEZIMAL INTEGER NATi, PAY

4.6.

```
10  K=32768
20 INPUT A
30 IF A<K PRINT A: GOTO 20
40 A=A-K
50 A=(NOT A) +1
60 A=A+K
70 PRINT -A: G0TO 20
                          4.7.
```

5 Comments on PRINT and LPRINT

PRINT, ";", ",", PRINT AT, PRINT USING, LPRINT

The output to screen and printer follows certain rules. These functions are discussed in more detail below. The following table:

1. PRINT
2. THE SEMIKOLON ','
3. THE COMMA ',' AND TAB (X)
4. PRINT AT '@'
5. PRINT USING
6. LPRINT

```
 PRINT 12
  12
 PRINT "HI" HI
 A=: B=5
 PRINT A
  3
PRINT 3+5 8
 PRINT SQR(8)
  2,82843
 AS="HELLO"
 PRINT
 A$ HELLO
 PRINT LEFT$(A$,3)
 HAL
                          5.1.4
```

5.1 PRINT

The PRINT statement prints data (numeric or character strings) to the screen. After PRINT, a function or expression can be called, the result of which is then output to the screen (5.1.1).

In the examples that follow, the character '_' (the underline) is used for a space (SPACE). The first important point: The output to the screen is always completed by the operating system with carriage return and line feed functions. These control commands, taken from typewriter technology, cause the next PRINT statement to output data at the beginning of the following free line.

The 'Line Feeder' statement can also be used in coded form by the BASIC programmer. It corresponds to the function 'CURSOR DOWN'[' (5.1.2). The example (5.1.3) shares both statements, "Line Feeder" and "Waggon Return".

```
ROW ADVANCE: CODE 10
=
INSTRUCTION: PRINT CHR$10);
EXAMPLE:
3"111";CHR$10);"222"
111
   222
```

5. 1. 2

```
CAR RETURN AND ROW
ADVANCE: CODE 13

INSTRUCTION: PRINT CHR$13);
EXAMPLE:
3111";CHR$13);"222"
111
222
```

5. 1.3.

A second important point concerns the issue of numerical data (5.1.4). The following rules apply:

A positive sign is replaced by a space.

For numbers less than zero, the value of zero before decimal is suppressed.

These limitations can be circumvented either by short conversion routines or by the PRINT USING statement. The programme (5.1.5) supplements the positive sign and adds the leading zero for numbers <0. As a subprogramme, the output of numeric values can be done with this programme.

```
10 PRINT 0
20 PRINT -12
30 PRINT 12
40 PRINT 0.37
50 PRINT -0.37
RUN
0-12
12.3
7 -
0.37
```

5.1.4.

```
10 INPUT A: AS=STRSA)+" "
20 IF A=O THEN 90
30 X=ASC(AS): AS=RIGHT$A$,LEN(A$)-1):
   Y=ASC(A$)
40 IF X=32 THEN X=43
50 IF Y=6 THEN AS=CHR$X) +"@"+4$ ELSE AS=CHR$X) +A$
90 PRINT A$: PRINT: GOTO 10
```

5.1. 5

5.2 The semicolon ';'

This character in a PRINT statement suppresses the CR (carriage return) and LF (line feed) control sequences at the end of the output. The next PRINT output will be in the same line immediately after the previous one. (5.2.1) shows that after a numeric value is output, a space is inserted automatically, while strings ・without spaces are lined up.

Within a PRINT statement, functions, constants, variable names, and expressions can take the form of a list

in succession. Separation, if necessary, is done for output within one line by the semicolon (5.2.2, 5.2.3).

```
10 PRINT 2;
20 PRINT "HELLO"; "X"
RUN
 2 HALLOX
```

5. 2.1.

```
PRINT 12;-.13 12 -.13
```

5.2.2.

```
A$="HELLO"
PRINT AS$" LASER"
HELLO LASER
```

5.2.3.

```
PRINT 1,2,3,4
1             2
3             4
```

5.3.1.

5.3 The comma',' and TAB (X)

The comma and the TAB (X) function take over tabulator tasks in PRINTAannotation. The comma directs output to the screen to one of two pre-tabulated positions on a screen line. In a row with columns 0 to 31, these are digits 0 and 16 (5.3.1).

The TAB (X) function generates a tab position within an output line of 64 characters (two screen lines). The argument X of TAB (X) must be in the range of 0 to 255. TAB (100) tabulates in heading 100-64 equal to 36 (5.3.2), while TAB (200) tabulates in column 200-(364) equal to 8. Example (5.3.3) shows the application of this function to right-justified string output.

```
21A100);'X' —0-
31—
—X
```

5.3.2.

```
10 AS$="HAMBURG"
20 FOR I=1 TO LEN(AS)
30 PRINT TAB15-DI; LEFTS(AS,I)
40 NEXT
RUN
             H HA HAM
                HAMB
               HAMBU
              HAMBUR
             HAMBURG
```

5.3.3.

5.4 PRINT AT "

This statement allows the output of numbers and strings to the screen from a position defined by ') '. A constant, numeric expression, or numeric variable can be used as a definition (5.4.1). The value must be in the range of O to 511, corresponding to the 512 positions of the screen (5.4.2).

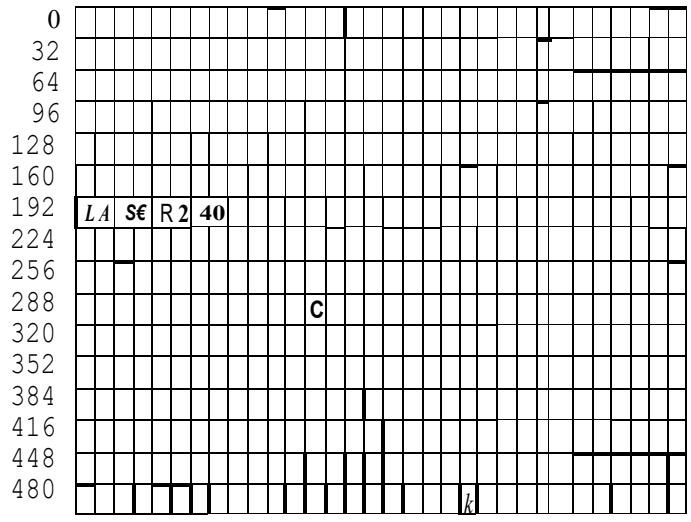
The targeted output to the screen can achieve interesting effects. (5.4.3) prints a flashing line to the screen. Programme (5.4.4) shows the programming of an input mask. Here the shape (5.4.5) is used to place the cursor on a screen position defined with X to place the subsequent INPUT statement. In this programme, also observe the strict separation between the text of the image in DATA lines and the structure of the mask.

```
PRINT 500,"A"
PRINT6 122,"B"
A=150
PR INT A2, 'C'
```

```
10 CLS
20 PRINT 192,"LASER210"
30 FOR I=1 TO 100: NEXT
40 PRINT 192,"AsER21?]"
50 FOR I=1 TO 100: NEXT
60 G0T0 10
```

5.4.1

5.4.3.



5.4.2.

```
10 DIM 2(5,1): COLOUR,1
15 •======= MASK
CONSTRUCTION 20 CLS:
RESTORE
30 FOR A=O TO 448 STEP 64
40 READ A$: PRINT@A,A$: NEXT
50 " ===== INPUT
60 FOR I=O TO 5: PRINT140+I6) ,;:INPUT Z1,@) 70
PRINT148+164), ;:INPUT ZI,1:NEXT
80 PRINT444,
"OK?" :PRINT478,"J" :PRINT476, ;:INPUTJ$ 90 IF
ASC(J$)<>74THEN 20
91 COPY
200 •======= TEXT IMAGE CONSTRUCTION
205 DATA "k CENSORSHIP MANAGEMENT ++++" 210
DATA "+FACH+k POINTS CENSORSHIP"
.220 DATA GERMAN,ENGLISH,MATH,PHYSICS,BIOLOGY,FRENCH.
```

5.4.4.

```
****        CENSORSHIP        ****
            MANAGEMENT
+kFACH#k POINTS CENSORSHIP

GERMAN    ? 23      ? 6

ENGLISH   ? 234     ? 1

MATHE     ? 34      ? 4

PHYSICS   ? 78      ? 2
```

```
BIOLOGY      ? 23?   ?
FRANZOES.    67      Fiv   0K?
                           ? J
                     e?
```

5.4.4.

```
                     2
```

```
100 PRINT@ X; INPUT A$
```

5.4.5.

5.5 PRINT USING

PRINT USING causes the formatted output of numbers. The USING statement can be used in PRINT statements individually or as a last statement with other formatting functions and control characters (**TAB(X),** ',', ';', ' @') (5.5.1). After USING, a string containing an output rule (mask) for the number to be output must always be defined. The basic elements of this mask are the characters **'#'** and '.'. The **'#'** represents a digit position, the dot represents the decimal point as a placeholder (5.5.2).

If the decimal places in the output exceed the places specified in the mask after the point, the round shall be completed (5.5.3).

If the digits before the decimal point of the mask are exceeded by the number to be emitted, the total number is returned with a "%" sign (5.5.4).

```
US ING "###"
```

5. 5.1.

```
PRINT US ING "##.#";8.5 8.50
```

5.5.2.

```
PRINT US ING "###";.34 0
```

5.5.3.

```
PRINT USING
"##.##";315,753 %3.15. 75
```

5.5.4.

```
PRINT US ING "##";-
12 %-12
```

5.5.5.

The negative sign shall also be preceded by a space in the mask. The number -12 must be prepared for output with '###' in the mask (5.5.5).

To control sign output, a '+' in the first position of the mask causes the positive or negative sign to be emitted before the number. The '+' at the end of the mask causes the sign to output after the number (5.5.6).

A '-' sign at the end of the mask allows negative numbers to be marked by a minus sign (5.5.7).

A double '' character in the first two digits of the mask fills all unused positions with the '' character (5.5.8), the so-called protection star, when output.

The double '$' character will cause a dollar sign to be printed immediately before the number. A combination with the character '' is possible (5.5.9).

```
PRINT US ING "+###' ;12
 +12
PRINT  USING  "###+";-12
 12-
                          5.5.6.

PRINT US ING "###-" ;-12
 12-
PRINT  USING  "###-";12
 12
                          5.5.7.

PRINT USING  "wx####.#" ;12
+xx12.00
                          5.5.8.


PRINT US ING "$####"; $12 $12.00
PRINT USING "$####" ;-12 -$12.00
PRINT USING "+$####.##" ; 12
+$12.00
                            5.5.9.
```

A comma on the left of the decimal point separates after three digits in the integer part of the number (5.5.10). Large numbers become easier to read.

An exclamation mark in the first part of the mask causes the first character of a string specified after the mask to be printed. Combination with other characters is possible (5.5.11).

```
POKE 30977,8:'DBL.PREC.
A=112678,989
PRINT USING "#######,.##";A 112,678.99
```

5. 5,10

```
AS="HAMBURG"
PRINT US ING "1####"; A$;12.24
H12.24
```

5.5.11

```
10 FORI=1TO7:READA
20 PRINT,,USING"#######,.##- DM";A
30 NEXT
100 DATA.123,12.999,124375.89,-12,375,1.34E2,1.23E-2
```

```
 DM .12
 13.00 DM
 124.376.00 DM
 12.00 DM
 DM 375.00
 DM 134.00
 DM 0.01
```

5.5.12

All non-functional characters can be placed in the mask at the first or last place. They are then also issued at these positions. Programme (5.5.12) demonstrates this. In principle, one spot in the screen is reserved for each character in the mask.

5.6 LPRINT

All the tax instructions that work with PRINT also work with LPR INT: Print output, together. However, some special features should be taken into account.

The comma tabulates in a line that contains 128 characters. In every 16. Column is a tabulated position. If a printer is used with 80 characters/line, pre-tabulated positions are selected in the first line 5. The distance here is 16 characters.

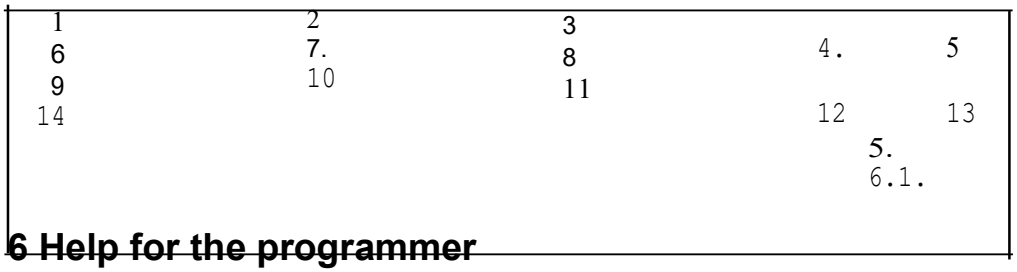
If more than 5 tabulator positions are controlled with an L_PRINT statement, three additional tabs are added in the second line (5.6.1).

PRINT USING works in the form LPR INT USING as on the screen.

Also, the control character ';' will work with LPRINT as usual from the screen output.

The TAB (X) function tabulates from X=O to X=63 in one line. TAB (64) tabulates again in column 0. With LPRINT TAB (X) you can only tabulate in columns O to 63.

```
LPRINT 1.2.3.4.5.6.7.8.9.10.11.12.13.14
```



6 Help for the programmer

STOP, CONT, TRACE ON, TRACE OFF, LIST, DELETE, CAR

A blocky programme structure is required for efficient programme editing and debugging. I.e. the programmer should assign recurring structures in his programme to fixed areas of line numbers. Already at the list. This allows you to access individual sections of the screen. The following framework for a BASIC programme is partly based on programming considerations. For example, finding subprogrammes for the interpreter is much faster when they are at the beginning of the BASIC programme.

```
10       GOTO 20000: JUMP TO MAIN PROGRAMME SUBPROGRAMMES, E.G.
100.     COMPUTING, STRINGMANIPULATIONS LOOP WITH INKEY$
500.     INPUT VON DER FLOPPY
1000     PRINT ON THE FLOPPY
.200     PRINTER:MAIN SUBPROGRAMMES
0.30     FORMAT IERUNGS SUBPROGRAMMES
00       OTHER PERIPHERY ( CASSETTE) FORMATTING
         INSTRUCTIONS FOR OTHER PERIPHERY DATA - READ
5000     ROUTINES
         DATA LINES, COLLECTED
8000     MAIN PROGRAMME********
10000    INITIALISATION OF THE MOST COMMONLY USED
20000    VARIABLES WITH DIM
         MAIN EXPIRY CONTROL MENUE COMMAND LEVEL WITH
         ON ... GOTO 30000,40000, ....
```

6.0.

In a programme built in this way, the LISTKommando can quickly access a part to revise the programme.

LIST

For the sake of completeness, the LIST command is briefly addressed here: LIST (CR) rolls the entire programme across the screen. Fast readers can download the output at the

```
LIST
LIST 10
LIST 100-200
LIST -1000
LIST 100
```

6.1.

```
10 INPUT 'DEZ.ZAHL 0-255';D
20 FOR I=7 TO 0 STEP -1
30 PRINT SGN (D AND 2 tI);
40 NEXT I: PRINT
50 LIST 20-40
RUN
DEC.NUMBER 0-2552 127
0 1 1 1 1 1 1
20 FOR I=7 TO 0 STEP -1 30
PRINT SGN D AND 2I; 40
NEXT I: PRINT
```

6.2.

Press the SPACE key to stop and press again. (6.1) shows the possible parameters of the LIST command. The '-' stands for 'from beginning to ...' or 'to the end'. By the way, LIST can be the last line in the programme. A trial run is then always completed by deleting the currently processed routine (6.2). (note: This small but effective routine converts a decimal number into a binary number!)

BASIC UP: DELETE

Further assistance will again be provided by the BASIC extension: the DELETE statement. It is given with parameters and deletes the statement lines in the programme listing as specified in the parameter part. It is possible to delete a line with DELETE #, but you prefer to enter the line number followed by RETURN. In addition, DELETE # - # deletes a block in the programme from line X to line Y. DELETE -Y deletes from programme start to specified line number.

DELETE should not be used in the programme as an editing instruction. A corresponding error message is then displayed.

```
DELETE 10
DELETE 200 - 300
DELETE - 100
```

6.3.

AUTO - Auto Line Numbers Default

The AUTO X,Y statement automatically specifies the next line number in edit mode, where X is the line number from which the default value starts. Y is the step. AUTO 10.10 generates line numbers starting from 10 in 10-bit spacing, i.e. 10, 20, 30, etc.

If AUTO is entered without parameters, then 10, 10 is automatically set. AUTO 350,50 numbered from 350 with step 50, AUTO 0,5 numbered from 10 with step 5 and AUTO 130 equals 130, 10. The operating system maintains all routines of this function. The word 'AUTO' is not recognised, but the function can be set with three POKE statements

are switched on (6.4). CTRL-BREAK cancels the automatic numbering. If the last POKE statement (6.4) is given again, the AUTO operation will resume with the next number.

```
TURN ON CAR:

(Method 1)
POKE 30946,10
   (STARTING     LSB)
POKE 30947.0
   (STARTING     (MSB)
POKE 30948,10
   (STEP          LSB)
POKE 30949.0
   (STEP          (MSB)
POKE 30945,255
  (LAUNCH,  RESTART)
                          6.4.1.
```

```
TURN ON CAR

(Method 2)
10 PRINT 10,10:   'FIRST  ROW!
(FROM KEYBOARD:)
POKE 31469,183:   '       CAR
(STARTS AUTO FUNCTION:)
RUN
 (RESTART OF CAR FUNCTION
TO BREAK:)
POKE 30945,255
                          6.4.2.
```

BASIC UP CAR X,Y

The AUTO mode can be switched on with the BASIC extension **BASIC-UP** (6.5) and also with CTRL-BREAK.

```
CAR
CAR 350, 50
AUTO, 5
CAR 130
                          6.5.
```

TRACE ON -TRACE OFF,

Gradual processing of the BASIC programme

Another function of the operating system, which can also only be switched on and off with POKE instructions, is the gradual processing of the programme or a programme part with the output of all edited line numbers to the screen.

This feature is especially useful for branching instructions such as IF ... THEN... ELSE and GOTO, GOSUB, ON... GOTO... and ON... GOSUB... test. The instructions for powering this mode of operation ('TRAGE ON') and for turning it off ('TRACE OFF') show (6.6). The example (6.7) will output the programme for converting decimal numbers into binary numbers (6.2) in the TRAGE operation. Here the working methods are clear: Typing from the keyboard and output to the screen is displayed normally, while the number of lines being edited is added to the screen one at a time in the angle brackets. A precise programme analysis is possible.

```
POKE 31003,63: 'TRACE ON
POKE 31003,0 : 'TRACE OFF
```

6.6.

```
RUN
DEZ.NUMB 0-2552 127
ER 0 1     1 1 1        1
{ READY

POKE 31003,63
READY
RUN
<10>DEC.NUMBER 0-255? 127
<20><30> 0 <40><30> 1 <40><30> 1
<40><30> 1 <40><30> 1 <40><30> 1
<40>30> 1 <40> READY
COPY
```

6.7.

STOP and CONT and CTRL/BREAK

The regular BASIC statements STOP and CONT are

also intended for testing the BASIC programmes written by the user.

The CTRL/BREAK key has the same effect as the STOP statement. Both interrupt the programme flow, STOP at defined point in the programme, and CTRL/BREAK at this key combination. CONT stands for CONtinue and continues the programme with the next statement to edit. - How can these instructions be used to detect errors? Here is an example:

In a somewhat structured programme (see the beginning of this chapter!) a bug is suspected in a programme part. A 'STOP' statement is inserted before and at the end of the programme section. When the first STOP statement is reached, the programme is terminated with the message 'STOP in XX', where XX is the line number in which the STOP statement was executed. It is now possible for the programmer to redefine data for the programme part, to continue the programme with the statement 'CONT' and to query the result of the data processing from the keyboard with the second STOP.

A gradual control, not of the programme run as in TRACE ON', but of the processing of all data, is possible. Example (6.8) shows the screen copy of such a work. A programme that was cancelled with the CTRL/BREAK key can also be resumed with CONT. For both types of abort, it must be stated restrictively that after changes in the programme listing, it cannot be started with CONT again. A new programme start with RUN is then necessary with loss of all the data previously generated.

If you want to continue a STOP programme elsewhere, the GOTO XX keyboard can resume the programme run at the location marked XX without data loss.

For example (7.8): In a loop, four field variables are assigned the square of the loop variables. In a STEP -1 reverse loop, the results are

, but not four, but five. The line 35 STOP is entered for testing. The test run then shows that the loop counter is 1 greater than the value defined with TO after the line is finished. The output loop will then add 5 outputs to the screen.

```
10 FOR !=1 TO 4
20 AI=I
30 NEXT I
40 FOR I=I TO 1 STEP -1
50 PRINT AI;
60 NEXT
RUN <CR>

0 16 9 4 1 READY

35 STOP <CR>
RUN <CR>

BREAK IN 35
READY

PRINT I,A(I) <CR>
5       0
```

6.8.

7 Text graphics, function and application

2000 characters Resolution, quarter graphics, a quarter learns to run The LASER screen can display 512 characters in text mode (MODE (0)): 16 lines, 32 characters. Each memory cell of the video RAM can be described with the POKE statement and read with PEEK. In BASIC programmes, a memory cell can be described with the base address 28672 and the coordinates X (position in the row) and Y (line number), where X can take the values 0 to 31 and Y the values 0 to 15. A character position can be described with 28672 + (Y32) +X. In the distribution of the screen shown in the appendix, the selected character position is in row 8 and column 7. The corresponding RAM memory address is then:

28672 + (832) + 7 = 28935

With POKE address,code, every character is to be brought to the screen specifically. (codes: See Annexe 13.)

The example (7.3) brings to 12. Row, 20. Column an **'A'.** Note that column and line numbering starts with 'O' at a time! Also note that the screen code is not the same as the ASCII code!

2000 characters resolution on graphics!

The graphic character set includes 16 characters of quarter graphics. There is a character set for each of the 8 colours of the video controller.

Quarter Graphics increases text resolution from 512 to 2048 points!

In order to use the 816 characters in a targeted manner, the structure and organisation should be described below.

VIDEO RAM:

```
28672 - 29183
7000H - 71FFH
                          7.1.
```

CHARACTER ADDRESSING:

```
28672 + (Y32) + X
                          7.2.
```

```
10 X=20: Y=12
20 8R=28672
30 POKE BR+Y32)+X.1
                          7.3.
```

Each screen position is divided into four rectangles. The combination of set and deleted quarters allows the display of 16 characters. If the screen code of a graphic character is binary, the character is encoded in the four low-value bits, the bits 4-6 describe the colour and Bit 7 displays the graphic.

Now let's turn to bits 0-3. These four bits can be used to encode all 16 graphics characters. Bit 0 describes the bottom right quarter, bit 1 the bottom left quarter. Accordingly, bit 2 will be displayed in the upper right and bit 3 in the upper left. Is used in

If a quarter is set, the corresponding bit position is set to 1.

The graphic character [is represented by the binary number 0100.

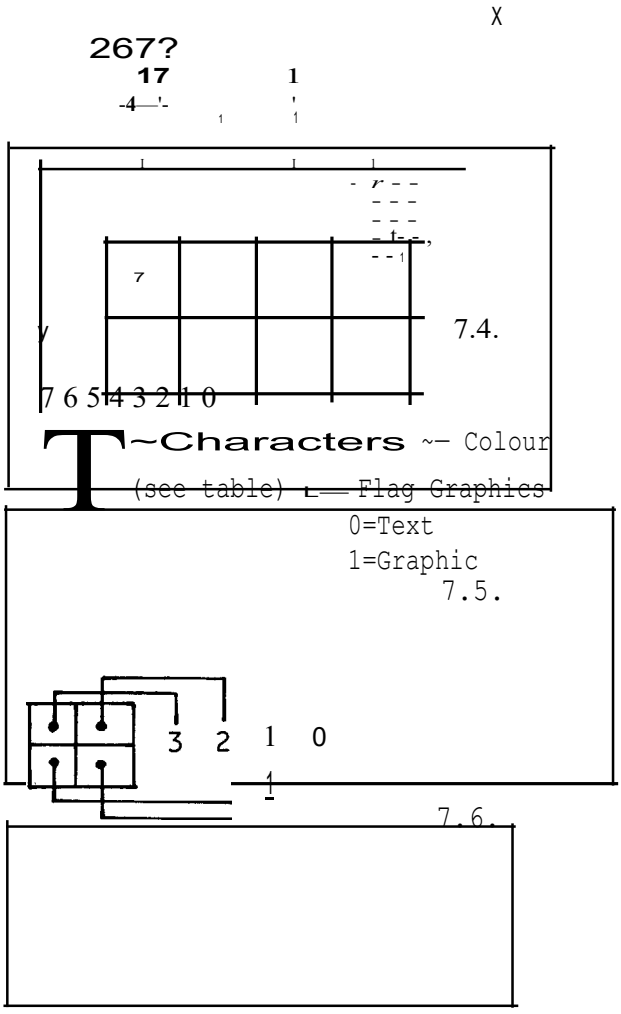
BASIC does not allow binary numbers. In order to calculate the necessary code to create graphics in BASIC, a 'value' is assigned to each quarter (7.7). The character code is calculated as an addition of the values of the set quarters. Example:

]L ++S-S **El**

1+2-3

1 +2+8= 11

Compare to the table below. But with this decimal number we do not have the code number of the graphic character. Add the colour value to the character code and the value for graphic = 128!



Example: Calculation of the code for the ze with the colour 'cyan':

Character code = 1 + 2 + 8 = 11 =
Colour = cyan 80
graphic value = 128

Code to set with POKE = 219 Use the example programme 7.3 for control!

jm

7.7.

No.	Graphics	Code
0	D	0000
1	a	0001
2	liiJ	0010
3	l.	0011
4	[	0100
5	[I	0101
6	lp	0110
7	~	0111
8	e	1000
9	~	1001
10	IJ	1010
11		1011
12	~	1100
13	!I	1101
14	I!	1110
15		1111

7.8.

NO.	COLOUR	CODE	VALUE
0	GREEN	000	0
1	YELLOW	001	16
2	BLUE	010	32
3	RED	011	48
4	BROWN	100	64
5	CYAN	101	80
6	MAGENTA	110	96
7	ORANGE	111	112

7.9.

```
10 GR=128: BR=28672
20 FORC=0T0127 STEP16
30 FORZ=1 T015
40 CH=GR+C+Z
50 POKE BR,CH
60 BR=BR+1
70 NEXT Z,C
```

7.10

A demonstration programme can now be constructed from the previous experience, which calculates the graphic signs in all 8 colours and brings them to the screen (7.10).

To work graphically on the screen, the characters must be manipulated. Each quarter must:

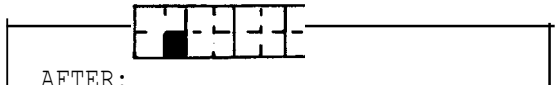
of the graphic character. Example:

A set quarter under the video RAM address BR should be moved to the right by a quarter square (7.11). The following steps are necessary:

1. Read Video RAM under BR.
2. Determine which quarter is set.
3. If the left quarter is set up or down, then .. _
... delete left quarter, ...
right quarter,
... and add to RAM under BR.
4. is set the right quarter up or down, then ...
... Increase BR by one.
.. . delete right quarter, ...
left quarter,
... and under BR.

BEFORE:

BR BR+1 ..



AFTER:

7.
11

y

7,12

```
10 BR=28672
20 POKE BR,200
30 IF BR<29184 THEN
GOSUB 100: GOTO 30 40
END
100 FC=INT PEEK(BR) /16) 16
```

```
11 o ZC=PEEKBR) -FC
120  POKE BR, FC
130  1F ZC=8
      THEN ZC=ZC  AND 7 OR 4
      ELSE ZC=ZC  AND 11 OR 8
      : BR=BR+1
140  POKE BR,  ZC+FC
150  RETURN
                              7.13
```

```
A    PEEK (BR)                7,141
```

A "quarter" learns to walk!

The following programme moves a quarter-square across the screen! **(7.13)**

Row 10 sets the beginning of screen RAM, line 20 writes the first graphic character. The following line represents the main programme until the end of the video RAM is reached, as long as the subroutine calls 100. Lines 100 and 110 read the last character under BR and set graphic and colour code in FC and the character code in ZC. Row 120 then deletes the last character. Row 130 calculates the new character code as a function of the old code and, if necessary, increases the video RAM address BR. Line 140 writes the new character to the screen, in 150 returns to the main programme. - Let's take a closer look at line 130!

Since we initialised the first graphic sign with the code 200 in line 20, we work with the colour 'Brown' and the quarter value = 8. So in ZC we have to expect an 8 in the first run.

if ZC = 8, THEN the character is = ~

OTHERWISE it is a [with who=4.

If it is an 8, then the character is associated with AND operand 7:

	8	1ooo....] o
AND	7	▌
Result	0	0000 ..... □

The sign is erased!

0 0000 .....

~-~—01 00 _ .....—~—

Result 4 0100..... [

The upper right quarter is set!

Now perform the link for the ELSE part of the statement! There, a recognised **4** must become an 8 again. To change from **4** to 8 you also need to increase the VideoRAM address.

```
POKE BR, code
                                    7,15

IF PEEK (BR) AND 1 THEN (set)
   ELSE (not set)
                                    7,16
```

Below is a summary of the BASIC instructions for reading, writing, testing, setting, and deleting graphic characters and graphic districts:

Read a screen line (7.14). Write a screen cell (7.15).

Test whether a quarter is set or deleted (7.16).

AND 1 tests neighbourhoods with value 1 AND 2 tests neighbourhoods with value 2 AND 4 tests neighbourhoods with value 4 AND 8 tests neighbourhoods with value 8

A quarter will be deleted (7.17).

AND 210 deletes 1 AND 195 quarter deletes 2 AND 165 quarter deletes 4 AND 105 quarter deletes 8 quarter

(Colour code and graphic value are not changed!) A quarter is set (7.18).

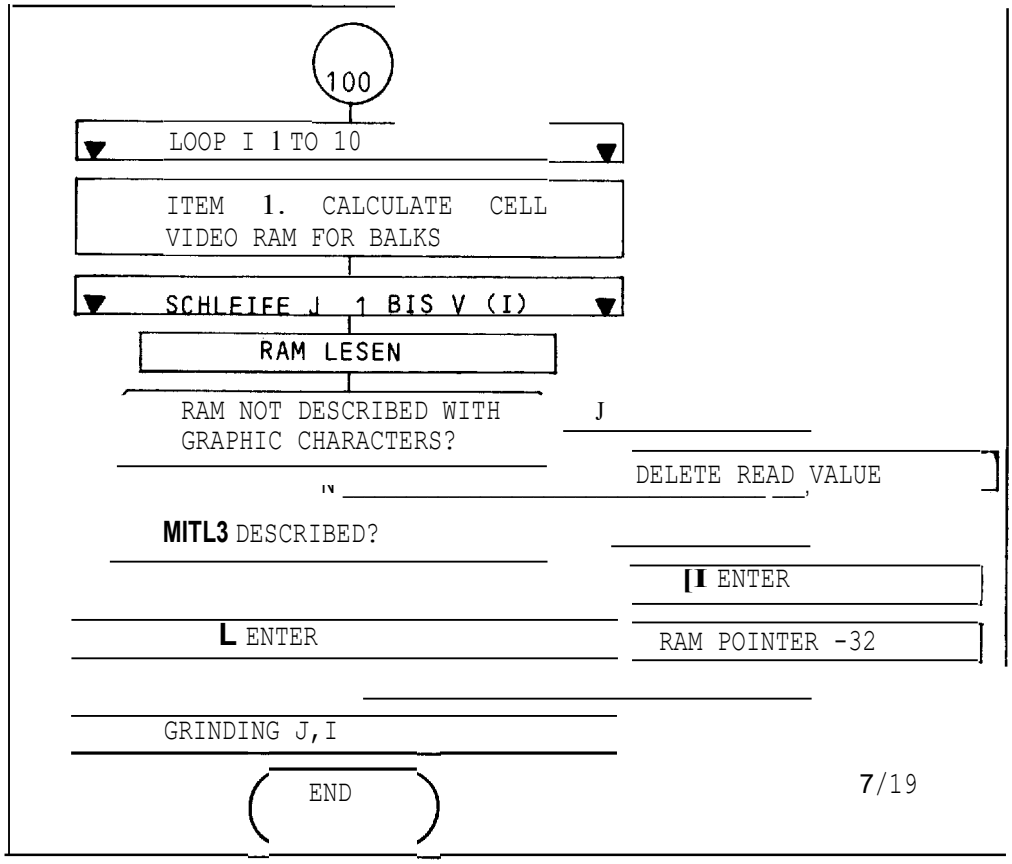
0 R 1 sets quarter with value 1
0R 2 sets quarter with value 2
OR 4 sets quarter with value 4
OR 8 sets quarter with value 8

Of course, you can also set, test and delete several quarters at the same time with corresponding operands.

```
POKE BR, PEEK(BR) AND 210
                              7,17
```

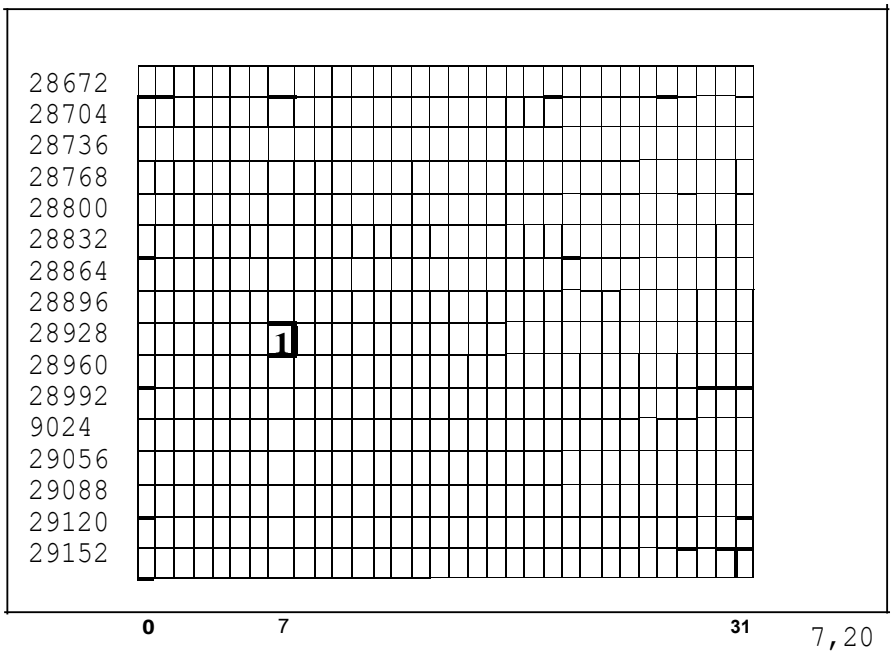
```
POKE BR, PEEK(BR) OR 1
                              7/18
```

The following programme creates a bar chart to graphically display 10 numbers in V(1) with colours stored in field F (1). Now, try to see how this programme works using the flowchart.




```
10 CLS
20 FOR I=1 TO 10: READ VI, FI: NEXTI
30  DATA  2,128,12,144,14,160,15,176,21,192  40
DATA 23,208,8,224,14,240,23,128,2,14 4
```

```
100 FOR I=1 T0 10
110 Y=15: X=4+I2): G0SUB 210
120 FOR J=1 TO VI
130     A=PEEK(PR): IF A=32 0R A=96 THEN A=0
140     IF A AND F(I)+1 THEN POKE PR, A OR
        F(I+5 : G0SUB 200: G0T0 160
150     POKE PR, A OR F(I)+1
      NEXT
J 170 NEXT
I 180 END
200 Y=Y-1
210 PR=28672+Y+32) +X: RETURN
```



8 More graphics and sound

The character *" -",* **INVERS, Cursor control, COLOUR with** the **LASER 110, FASHION** and background colour, **BUZZER on-off**

8.1 The character "-"
8.2 INVERS
8.3 Cursor Control
8.4 COLOUR with the LASER 110
8.5 FASHION and background colour 8.6 BUZZER on-off

8.1 The sign ""

There is a character in the LASER character set that cannot be generated from the keyboard: It's "f;-". However, it can be chained to or chained to strings using the instructions as in (8.1.1) or, as in (8.1.2), placed directly on the screen.

8.2 INVERSE

Characters can be placed on the screen as INVERS constants (8.2.1).

The[CTRL] {INVERS] shortcut sets or clears a bit in the operating system's 7838H flag register. The output of variables and constants to the screen can thus be set or delete the INVERS flag normal or INVERS

&f
&.I NT CHR$ (95)
NT CHR$ (223)

```
POKE  28672,31

POKE 28672,95
```

8.1.1.

```
A$100=A$+CHR$(95)+B$
```

8.1. 2

10 PRINT "HAMBURG"

.9#a'' "au

INVERS a. 1 NVERS out.

8.2.1.

```
POKE 30776,PEEK30776) OR 2 8.2.2

POKE 30776,PEEK (30776) AND 253 8.2.3
```

```
10 A=123,375:F=30776
20 POKE F, PEEKF) OR 2
30 PRINT A,F
40 POKE F,PEEK(F) AND 253
50 PRINT A, F

RUN
[123,375        30776]
 123,375        30776
                      8.2.4
```

and. **(8.2.2)** sets the flag, all the following issues in the programme are represented in INVERS. (8.2.3) deletes the flag, the following output to the screen is brought in normally. The small programme (8.2.4) shows the application.

8.3 Cursor Control

Any cursor control commands that are given by the keyboard when editing the programmes can also be included in the programme. You can create menu controls, input masks, backups, and graphics optimally (8.3.1). The execution of these functions in the programme is programmed with PRINT CHR$ (code), or the code is chained to or in a string with the string concatenation operator **"+"**. The example (8.3.2) backs up an input.

For example (8.3.3), you can specify a commonly used value (default value) to enter different values. This value can then either be applied with RETURN or it will be overwritten.

```
cursor 1 mnks        8
cursor right         9
cursor Up           27
cursor Down        1 0
cursor 'Home'       28
Clear Screen        31
CR (RETURN)        1 3
Insert              21
ruby               127
                        8.3.1.
```

The cursor control codes can be copied from the keyboard and processed with INKEYS.

```
100 INPUT 'ELCHES FACH';K
110 IF K<1 OR K>7 THEN PRINT CHR$(27);CHR$27): G0T0 100 8.3.2

10 PRINT "YOUR CHOICE_2";CHR$8);CHR$8); CHR$(8) 20 INPUT I
RUN
rwe W • 2]
```

8.3.3.

8.4. COLOUR... with the LASER 110

Although the LASER 110 cannot produce colour on the screen, it knows the COLOUR statement. It can be used wisely.

The key combination CTRL/N produces the word 'COLOUR' on the screen. If the instruction is given with the parameter 1, as in (8.4.1), the contrast of the image on the monitor or TV receiver improves.

In addition, graphics can be displayed in different grey values. A table of grey values creates the programme (8.4.2).

Since the LASER 110 works with the same operating system as the LASER 210 Colour computer, the PAL colour coder can be easily retrofitted (Appendix 8). The BASIC statement COLOUR will then work properly and generate the colour as indicated for the colour computers LASER 210,310 and VZ200.

```
COLOUR,1 <RETURN>
```

8.4. 1

```
10 FOR I=1 TO 8 20
COLOUR I
30 PRINT I;' 40
NEXT I                    'SHIFT J 8.4.2
```

8.5 FASHION and background colour

The LASER video controller allows **two** modes of operation:

1. Text and Quarter Graphics
2. High Resolution Graphics

Switching is done in **BASIC** with MODE **(x).** The text operation mode is initialised after power-up. MODE (1) switches to high resolution graphics. MODE (0) switches back to text mode (8.5.1).

The switch is controlled via the 'Output Latch' (8.5.2). Bit 3 switches to text mode, a **1'** turns on the high resolution graphics.

Bit 4 controls the background colour accordingly and selects three foreground colours in the 'high-resolution graphics mode' (see Table 8.5.3).

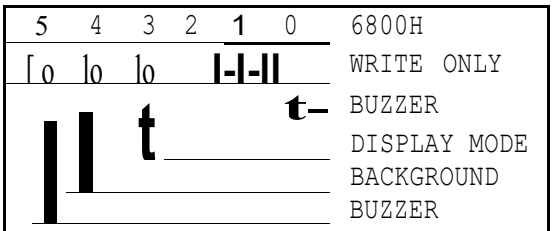
In BASIC, the background colour can be determined directly or from the programme, as in (8.5.4), without the foreground colour being called.

The POKE statement eliminates the need to control the mode and background colour, because the MODE (x) and COLOUR statements provide powerful options for the BASIC programmer.

TEXT AND QUARTERGRAPHIC
~~HIGH-RESOLUTION GRAPHIC~~

FASHION (O)
FASHION 1

8.5.1



8.5.2.

BIT4	TEXT	GRAPHIC
1	GREEN	G~LB GRÜN BLUE RED
1	BROWN	BRi~UN CYAN ORANGE MAGENTA

8.5.3.

```
COLOUR,O     GREEN
COLOUR,1     BRAUN
                                  8.5.4.
```

8.6 BUZZER on-off

Bit 0 and bit 5 of the 8.5. described 'Output Latch' control the buzzer. Here, too, the control will normally be performed using the BASIC statement SOUND. However, the operating system maintains a copy of the Output Latch at 783BH/ 30779. The buzzer can be switched on and off via this tab (8.6.1).

```
POKE  30779,<ll   UZ2ZER 'OFF'
POKE 30779,1      UZZER 'ON' 8.6.1
```

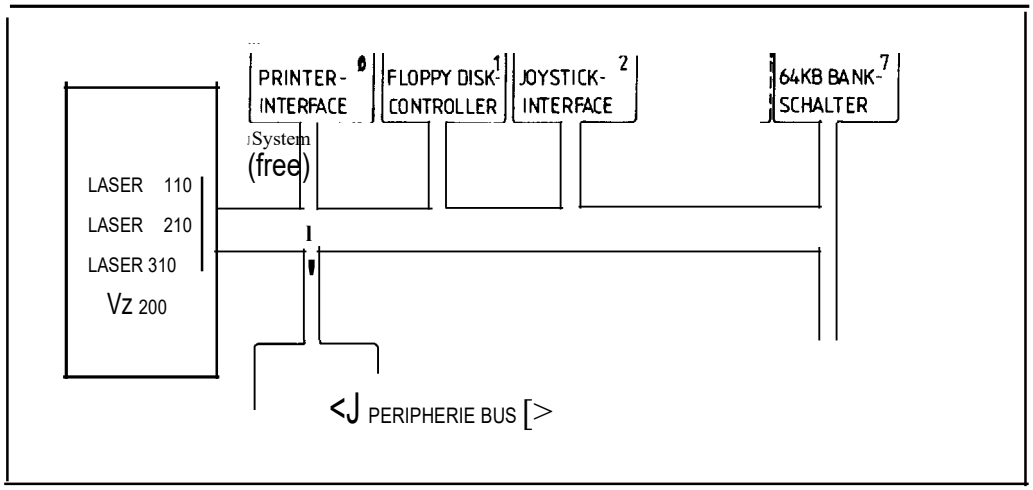
9 Peripheral control with INP and OUT USER port: Control with INP and OUT circuits and examples in **BASIC**

The LASER computer (LASER 110, 210, 310,VZ200) has an interface control system that allows it to exchange data and control commands with external devices.

This data exchange takes place under a total of 256 addresses, 16 of which are assigned to an existing or possible device.
Of these 16 input and output channels, the first eight are reserved for system functionality. The operating system controls

LASER computer input-output channels

Adr. 1 0.	Adr. (16)	Device
0 - 15	OOH - OFH	Printerinterface (Printer)
16 31	10H 1FH	Floppy Disk Controller
32 47	20H - 2FH	Joystick Interface Reserved
48 111	30H - 6FH	for future system extensions 614 KB RAM memory bank switch
112 - 127	70H - 7FH	Free for User Extensions User Port
128 -	80H EFH	Module
240 -	FOH FFH	9.1.



3 • 6

USER-PoRT ' Free for {User; yfrweiterungeri}

9.2.

```
INP (address) A=
INP (X)
```

9.3.

```
OUT address,date OUT
254,130
```

9.4.

these channels printer and floppydisk controller, reads the joysticks and switches the RAM memory banks of the RAME extension (9.1 and 9.2).

The other eight channels are designed for user functions. A USER port module is now available here.

The control of the traffic and the data exchange itself can now be done from an assembler routine or from a BASIC programme written by the user.

The BASIC interpreter provides the INP (X) function for the reading of data. The variable X, also a constant, must be a longer number of 0 to 255. It defines the address at which data should be read from one of the peripherals (9.3).

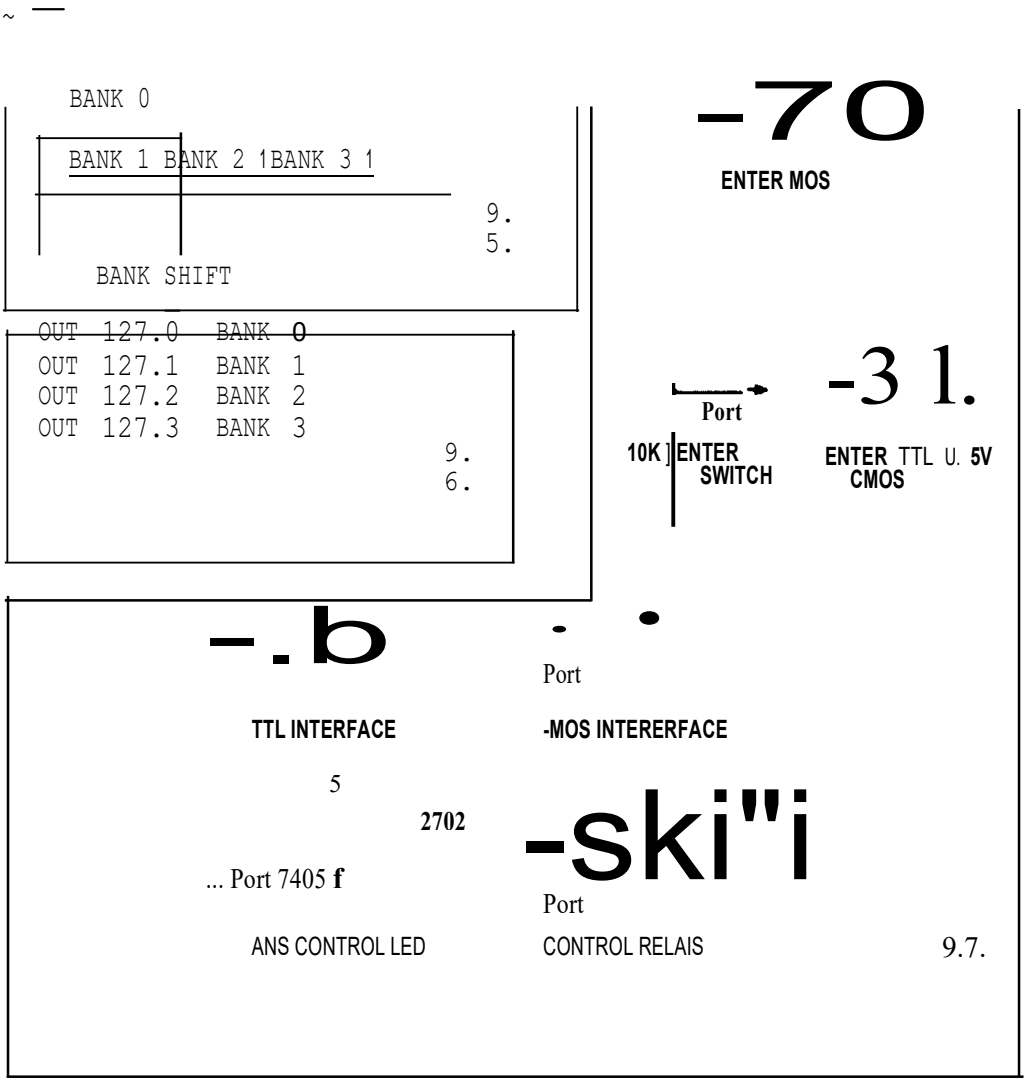
With the OUT X,Y statement, the BASIC programme passes data and control instructions to the device specified by the number X. The transmitted data in the range of 0 to 255 are represented here by variable Y. For X and Y, constants or radio

options to describe the values (9.4).

An example of reading in data from a peripheral device already shows the chapter for programming joysticks.

The 64KB RAM expansion

The 64 KB RAM expansion occupies a memory range of 32 KB. It is divided into four memory banks of 16 KB each (9.5). After the device is turned on, the available RAM area is occupied by the memory banks 'O' and '1'. This makes a total of 32 KB RAM available to the BASIC programmer. A larger memory area is physically not possible, and the BASIC interpreter cannot manage more than 32 KB RAM.



The use of the memory banks '2' and '3' in a BASIC programme is problematic, since Appendix 11 shows that the string area and the BASIC stack would then be in the area of the bank '1'. If Bank '1' is replaced by Bank '2', the BASIC programme will lose all data stored as strings, and the information lost in the stack area about GOSUB return addresses and active FOR-NEXT loops will lead to a collapse of the BASIC interpreter.

Nevertheless, a BASIC programmer can use the existing benches. In the '2' and '3' benches, assembly routines and data stores described in the BASIC programme with POKE and read back with PEEK would be conceivable. By moving the pointer 'Top Of Memory' (see chapter 12), the BASIC programme can run in the bank 'O'.

All benches can be used easily under the control of a machine language (ML) routine. It makes sense to place the main programme in a bank and use the other banks for filing subprogrammes and data. Only the switching mechanism must be used in a controlled manner.

The Bank Switch is connected to the Peripheral Bus as a Write Only Latch. Accordingly, the switch is made in BASIC programmes using the OUT statement (9.6).

The USER port

The user port is device 16 on the peripheral bus. It is a universal circuit for the control of all kinds of electronics and consists mainly of a multi-function chip of type・ 8255. Twenty-four lines can be used as inputs or outputs under programme control. They can be connected to TTL or CMOS circuits and control transistors, LEDs and relays. Combined with the electronics of a model railway or connected to alarm system, heating control or other electronics, the BASIC programme can take control and evaluation. (9.7) shows interface possibilities. The following pages show programme design and application with circuit examples.

The USER port will be set to address 255 for the intended application. The 24 input/output lines are organised into three groups of 8 lines each: Port A, Port B and Port C. The port to be used as input and the port to be used as output is specified by entry in the control register 255. Under three additional addresses, these ports can now be set or read **(9.8).**

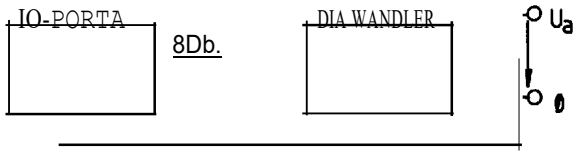
```
OUT 255.X   CONTROL REGISTER   SET
OUT 254.X   PORTC WRITE
INP (254)         READ
OUT 253.X   PORTB WRITE
INP (253)         READ
OUT 252.X   PORTA WRITE
INP (252)         READ
```

9.8.

9.1 Digital/Analogue Converter

A sawtooth output voltage is generated.

```
10 0UT 255,139                  :'CONTROL REGISTER PORT A OUTPUT
20 FOR I=0 T0 255               :'RAMPE IN 256 STEPS
30 0UT 252,I                    :'AN D/A-WANDLER FROM PORT A
40 NEXT I
50 G0T0 20                      :'NEXT IMPULSE
```



9.2 8-bit Analogue/Digital Converter

Analogue values are continuously converted to digital values and displayed.

```
10' PORT B INPUT - PORT CO-C3 OUTPUT - PORT C4-C7 INPUT
20 'PORT A OUTPUT 30 OUT 255,138
40 OUT 254,1: OUT 254.0 50 PRINT
INP (253)                       :'CONFIGURE
60 GOTO 40                      PORTS :'START OF
                                CONVERSION :'SHOW
                                VALUE :'READ NEXT
                                VALUE
```

Port B 8 Datasheet.

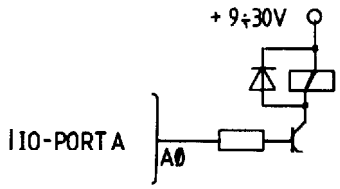
10-P0RT A/DWAND U

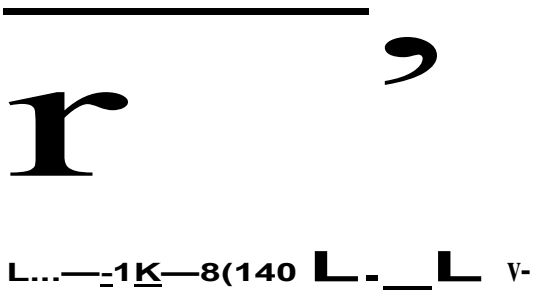
10 SV

F'OrIty 4 Handshakeltg GN

9.3 Switching on/ off relays to control any consumer A lamp is switched on or off.

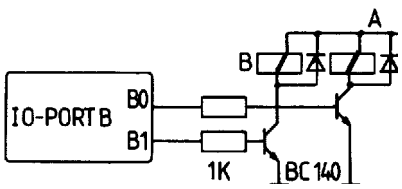
```
10 OUT 255,139                  :'PORT A IS EXIT
20' TURN ON WITH OUT 252,1
30' SHUT OUT WITH 0UT 252,0
```





9.4. A 12V motor is controlled in the ON, OFF, right and left hand functions.

```
10 OUT 255,153                  :'PORT B OUTPUT
20'  ENGINE  RIGHT  RUN  A   OUT 253,1
30' ENGINE LEFT RUN A 40'    OUT 253.3
TURN OFF                     OUT 253.0
```



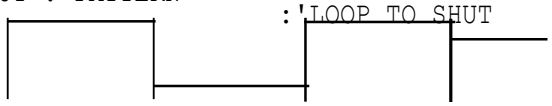
‘tg

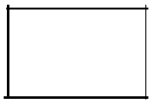
9.5 Running light with 8 lamps

```
10 OUT 255,139
20 L1)=1: L2)=2: L3)=4
22 L(4)=8: L(5)=16: L(6)=32
24 L(7)=64: L(8)=128
30 FOR I=1 TO 8
40 OUT 252,LI
```

: "PORT A OUTPUT :'PATTERN
FOR OUTPUT

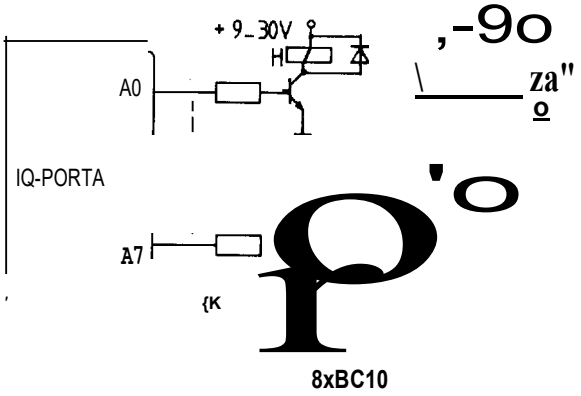
:'LOOP TO SHUT





```
50 FOR V=1 TO 1000: NEXT V 60     :'INSCALE
NEXT I                            DURATION   :   '
70 G0T0 30                        NEXT LAMP
```

NOTE:
FOR LEFT-HAND RUNS, LINE 30 SHALL BE MODIFIED AS FOLLOWS: 30 FOR I=8 TO 1 STEP -1



8xBC10

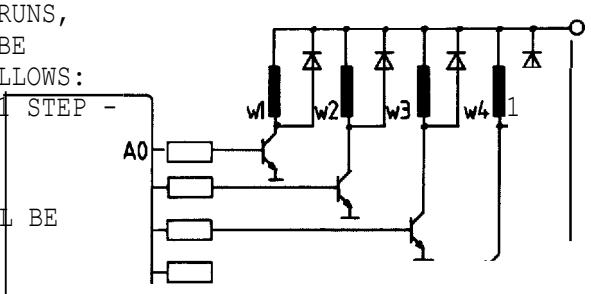
```
: 'PORT A IS
EXIT :'DEFINE PATTERN


: 'ENGINE WRAPPINGS
SHUT :'SPEED
```

9.6 Control of a four-phase stepper motor

```
10 OUT 255,139
20  W(1)=1:  W(2)=22  W(3)=4:  W(4)=8  30  FOR
I=1 TO 4
40 OUT 252,W(I)
50 FOR V=1 TO 250: NEXT V 60 NEXT I: GOTO 30
```

FOR LEFT-HAND RUNS, LINE 30 SHALL BE MODIFIED AS FOLLOWS:
30 FOR I=4 TO 1 STEP -



FOR SEMI-STEP OPERATION, THE FOLLOWING SHALL BE AMENDED: 20

```
W(1)=1:W(2)=3:W(3)=2:W(4)=6:W(5)=4:W(6)=12:
   W(7)=8:W(8)=9
30 FOR I=1 TO 8
```

12V

1-4-x 1-K-1-L

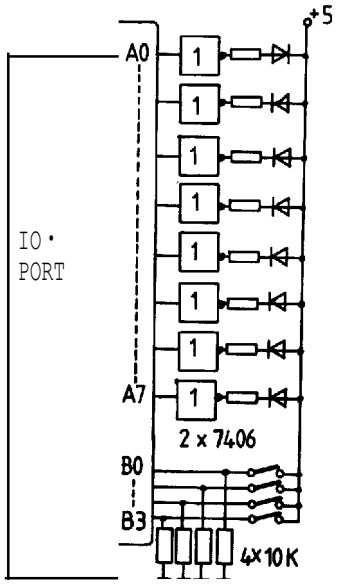
A1
IQ-PORTA A?

A3

9.7 Right/Left Running Light with Visual Age Control

```
 NO ON SWITCH: RUN RIGHT, ANY SWITCH ON: LEFT RUN.

10 OUT 255,130                    :'PORT A=OUTPUT,
20 A=INP (253)                    B=ENTRY :'SWITCH
30 IF A=O THEN 200                :'NO SWITCH: RUN :'OTHERWISE
100 FOR I=128 TO 1 STEP-1         LEFT
110 OUT 252,I                     :'I IS PATTERN 1000 0000 :'BIT
120 I=I/2+1                       SHIFTING, LOOP VAR KORR. : 'NEXT
130 NEXT I:GOT020                 LED ON
200 FOR I=1 TO 128                :'CURRENT CURRENT
210 OUT 252,I                     : 'PATTERN IS FIRST 0000
220 1=I2-1                        0001 :'BIT PUSH RIGHT
230 NEXTI: GOTO 20
```



10 Programming of joysticks

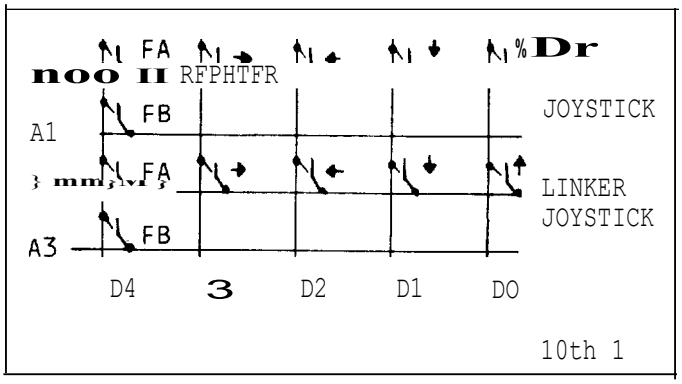
with INP (X), ON ... GOTO, ON... GOSUB...

The organisation of joysticks

The joysticks' switches are arranged in a matrix (10.1). The rows of the matrix are set to 'O' potential, depending on the address given, via the **AO** to A2 address lines. Table (10.2) shows the addresses.

The INP (address) statement can be used to read the state of the data lines. Depending on the specified address, the DO to D4 data lines will indicate which switch was pressed at the time of reading.

This information, obtained with **INP**(address), is encoded in



10th 1

ADDRESS		A3	A2	A1	A0
27H	39	0	1	1	1
2H	43	1	0	1	1
2DH	45	1	1	0	1
2EH	46	1	1	1	0

10/2

27H	39	KEYS FB LEFT JOYSTICK
2BH	43	KEYS FA, DIRECTION LEFT JOYSTICK
2DH	45	TASTER FB RIGHT JOYSTICK
2EH	46	FOOT KEY, DIRECTION RIGHT JOYSTICK

10/3

Form processed as a decimal number by the BASIC interpreter. In the form A=INP (address) it is stored in a variable.

Table (10.3) provides the link between address and function. If the direction information of the right joystick is to be evaluated, the statement 'A= INP (46)' should be included in the programme. Accordingly, the other functions shall be dealt with.

The number measured with the !NP function is in the range of 0 to 255, according to the 8 data lines D0 - D7. Since only the data lines D0 to D4 are used, the with I NP

(address) value read with a MASK in an AND function. The data bits D5 to D7 of the value of the variable are set to '0', the information-carrying bits D0 to D4 are not affected in their value (10.4).

So the four instructions for reading the joysticks are:

```
A = INP (39) AND 31    JOY LEFT,    KEYS  FB
A = INP (43) AND 31    JOY LEFT,    KEYS  Prod. DIRECTION
A = INP (45) AND 31    JOY RIGHT,   KEYS  FB

A = INP (46) AND 31    JOY RIGHT,   KEYS  Prod. DIRECTION
```

D7	D6	O5	D4	D3	D2	D1	D0	
–	–	–	X	X	X	X	X	INP(adr
0	0	0	1	1	1	1	1	AND 31
0	0	0	X	X	X	X	X	Result in A

10/4

Table 10.5. provides the context of the encoded Informa-

actions stored in the variable 'A' here and the joy-key functions.

Using AND masks, you can read individual information at addresses 43 and 46 (Direction and Button A). For example: A variable should contain only the information joystick to the right, button FA pressed or not pressed. Drawing (10.1) shows push button FA on the data line
D4. Table (10.3) returns the address for joystick right, button FA. So the read instructions would be: A = INP(46). The mask would be equivalent to (10.4) binary 0001 0000, decimal is

value 16. Programme (10.6) shows after RUN that the button FA returns 0, otherwise the result will always be 16. So all functions are out of operation (=16), only push button FA shows the value 0.

The conversion of binary numbers to decimal numbers can be done with the table (10.7). For all bits of an eight-digit binary number set to '1', the value should be added according to the table. Example: 1010 0011 binary will be 1+2+32+128 = 163 decimal.

```
ADDRESS FUNCTION RESULT
  JOY              a b

LI
   RE
   ___
39 |45  TAST. FB  15 -5
.4 |46  LI, UP     26 6
3  |    LI,        25 5
   |    RE, EW     22 2
   |    RE, AB     21 1
   |    ADD.       30 10
   |    1.         29 9
   |    LEFT       27 7
   |    RIGHT      23 3
   |    TAST.FA    15 -5
   |    OFF        31 11
   |    ___
   |A=INPX AND 31
(a)|A=(INPX) AND 31)-20
(b)                          10th 5

10 A = INP46) AND 16
20 PRINT A: GOTO 10
                               10/6

BINARY VALUE
```

BIT	VALUE
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

10/7

ON X GOTO... , ON X GOSUB ...

The BASIC function 'ON variable GOTO list,list' is suitable for the evaluation of the direction addresses. But the LASERBASIC does not know the keyword ON. However, because the executing routines are present in the interpreter, the ON statement can be entered in the correct spelling as in Chapter 3, the first line of the programme. Instead of the 'ON' word, another keyword should be written, in (10.8) it is PRINT, whose code is changed from the keyboard to the code for ON with a POKE statement. If the value of the INP function is deducted for direction addresses 20, the result will be in the range of -5 to 11. -5 for push button FA can be excluded with IF. The code for all 8 directions then ranges from 1 to 10. 11 returns pressed for 'no key'. This makes it within the scope of the ON function, which works as a jump distributor depending on the content of the variables named after ON. The variable must contain integer numbers from 1 up. After GOTO or GOSUB, a list of line numbers follows. The variable =1 branches to the first line of the list, with =2 to the second, etc. (10.8).

Programme (10.9) applies the ON function as described. It is intended as a demonstration programme for the joysticks. One point is moved across the screen in the graphic MODE(1). Drawings can be made on the screen. FB button moves the point with the joysticks without drawing.

```
2 PRINT A G0T0 100,200,300
1. ROW IN PROGRAMME)
POKE  31469,161  <RETURN>
```

10/8

```
1 G0T03: '2 ON A G0T0 110,120,130,150,160,170,190,200,210
2
3 CLS: COLOUR2,1 :DIM RI3),AD3"3): RESTORE FOR
I=O TO 3: READ ADD, WI: NEXT: FASHION (1) 10
SET WHERE)1): SET 2)/3))
20 FOR I=O TO 3: RI(I)=(INP(AD(I)AND31)-20:
NEXT 30 IF RIC0)=-5 RESETCW(O),W(1))
31 IF (RI)=-5 RESET 2) .3)
40 C=O: A=RI(1): IF A>O GOSUB 2
```

```
41   C=2: A=RI3):  IF A>O GOSUB 2
50 GOTO 10
110 WHERE+C)=W(0+0)+1: W(1+C)=W(1+C)+1: GOTO 202
120 0+C)=WO+C) +1.     W1+C)=1+C)-1:    GOTO 202
130 WO+C)=WO+C) +1:    GOTO 202
150 WHERE+C)=WHERE+C)- 1+C)=V1+C)+1:    GOTO 202
160 WHERE+C)=0+0)-1:   W(1+C)=W1+C)-1:  GOTO 202
170 WO+C)=W0+0)-1-:    GOTO 202
190 W1+€)=V1+C)+1:     GOTO 202
200 W1+C)=W1+€)-1
202 IFW(O+C)>127THENW(0+C)=127
204 IF WO+C)<O THEN(O+C)=O
206 IFW1+C)>63 THEN W1+C) =63
208 IFW(1+C)<O    THEN1+C)=0
210 RETURN
1000 DATA39.0.43.0.45.127.46.63
                                          10th 9
```

BASIC UP

(10.10) shows the application of the ON function if the BASIC extension programme 'BASIC-UP' has been loaded before.

```
ION A G0T0 100,200,300
ION AGOSUB 100,200,300
                              10/10
```

11 Machine-related programming

Creating Programmes - Storage Deployment - Protection against BASIC CSAVE, CLOAD, and CRUN for ML Programmes

11.1 Introduction

Much has already been written about the use of the LASER computer as a BASIC computer, and it will seem almost self-evident that he understands BASIC. However, this can only be achieved by a combination of the hardware (i.e. the electronic components) and the software (the BASIC interpreter and operating system programmes). All commands entered in BASIC, either directly or as a programme, and any key input. are processed by this combination of software and hardware. So it seems that the device BASIC understands like a language.

By comparison: A separate BASIC programme has been launched with RUN, and

Host no longer understands a BASIC command, only the commands provided in the programme. The programme:

```
10 PRINT "SYNTAX ERROR": PRINT "READY": INPUT COMMANDS 20 IF COMMAND$ <>
"BASIC" THEN GOT0 10
```

demonstrates this in a stark way: After it is started, the computer accepts input, but reacts only to the 'command' BASIC - it turns on BASIC.

A BASIC programme can be interrupted at any time with BREAK. This chapter explains how to interrupt the BASIC interpreter and then directly influence the machine.

11.2 Construction of the computer

The design of the LASER computer corresponds in principle to that of all current microcomputers. The Z80 central processor performs the command. The commands are taken from memory, from which can be read only (the ROM modules) and in which can be written (the RAM memory). The ROMs are memory modules in which the stored information is preserved even after the shutdown. They contain the BASIC interpreter and the operating system. Data and programmes can be stored in the RAM memory. Unlike the ROMs, this information is lost after the power is switched off.

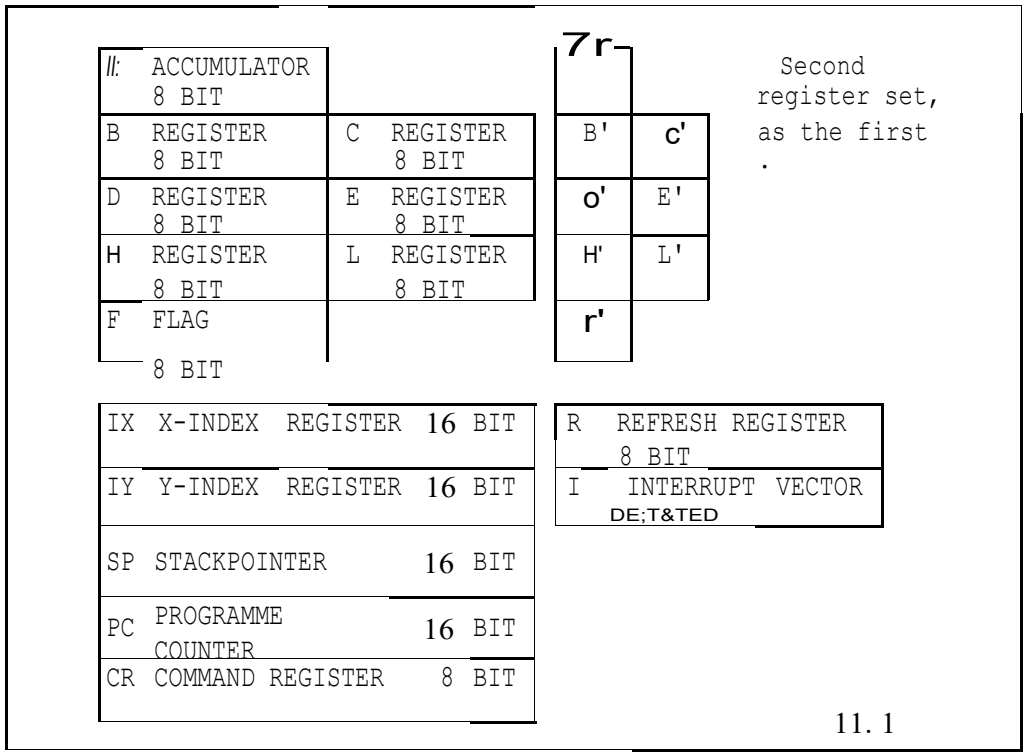
The Z 80 microprocessor can now execute a series of instructions, which in total represent the so-called machine language. In order to be incorporated into the machine-related programming, the description of function, application and programming to be found in the relevant literature should be worked out. Here you can only give a short overview.

11.3 Structure and Function

For the further representations in this chapter, the expressions Bit, Byte, and hexadecimal should be known. To process programmes in machine language, the Z 80 CPU (Central Processing Unit, **ZP**) has a number of internal memories, so-called registers. The usual illustration shows (12.1).

The sequence of a command execution can now be described as follows: The content of the programme counter is output as an address to the memory modules via the address bus (16 lines). The code located on this space is transported to the command register and the programme counter is increased by 1. Depending on the command code, values from the following locations will now be loaded, edited, linked, and

and new operations prepared. The programme counter is increased or changed accordingly. A machine command therefore consists of 1 to 4 memory words, which follow each other. (A keyword = a byte = 8 bits. It can represent 256 different pieces of information that can be expressed by the numbers from 0 to 255.) As long as no jumps occur, the commands and data are transported to the CPU from successive memory locations.



11.4 Transition from BASIC to machine language

To load a machine programme into memory, use the POKE command under the control of the BASIC interpreter. The decimal numbers corresponding to the machine commands and their operands are written continuously into the memory with **POKE**'address, value'.

When setting the start address, ensure that the intended storage area is protected from access by the BASIC interpreter (see Section 6). Then the start address of the programme is entered in the USER vector. This can be done through the programme (11.2). To this end, the statement lines in (11.3) should be added.

The ML programme can then be started with X = USR(X). When the ML programme is finished with 'RET', it returns to the BASIC programme, which continues with the next statement.

```
At play: 11. 2)
 5 ADDRESS      29184          :Start address of the ML-Prgr.
20 READ VALUE%                 :'First value from ML Prgr. list
30 IF VERT?=-1 THEN END        :End of list
O POKE ADDRESS, VALUE?         :Add to memory
50 ADDRESS = ADDRESS +1        :'Address next memory.
60 G0T0 20                     :Continue working in loop
70 —
100 DATA ....,  ..,  ..,  .., :ML-Progr. in decimal values
120 DATA ....,  ..,  ..,  .., :End of list with '...,-1
```

```
10 HB =AD/256            : 'MSB DECIMAL: '
12 LB= AD-(HBZ256) 14    CALCULATE LSB'
POKE 30863,HB%           :'VEKTOR          MSB
16 POKE 30862,L%         SET   :'VEKTOR    LSB
                         SET
```

11.

3

11.5 The USR vector

In the previous programme, this vector was set for the call of a machine programme from a BASIC programme. Here are some more information about the USA command: After the BASIC interpreter is turned on and initialised, this vector is directed to the error output routine. A call to the ML programme will result in

Error Message ?FUNCTION CODE ERROR IN XX.

If this vector is now properly directed to the address of the user routine, the control can be passed to this programme with X = USA (X). In the simplest case, in this function, X is a variable whose content does not play a role in the further programme run, i.e. a DUMMY variable. In other cases, this function can be used to pass a parameter to the machine programme, which can be defined within the brackets as a constant or variable of any kind, including a string. The variable is passed to the BASIC register WRA 1 when the USA function is called. If a String variable or constant is passed, a pointer to the String descriptor block (length and address) can be found in WRA 1. If the control of the programme run with R ET is returned to BASIC as the last instruction of the ML programme, the content of WRA 1 is assigned to the variable in the USA command and can be further processed in the BASIC programme.

The following BASIC programme directs the USA vector to the 'BEEP' routine and puts a string over WRA 1 into AS (11.4).

The addresses for WRA 1 and the related variable type flag are set out in Annexe 12.

```
10 POKE 30862,80 : POKE 30863,52      'USR VECTOR ON BEEP
A$20 = USR C"TEST")
30 PRINT AS
RUN
TEST
                                                    11/4
```

11.6 Why machine language?

Basically, BASIC offers the same possibilities to write programmes as with a Z 80 assembler. The loss of the functions of the BASIC interpreter has advantages and disadvantages.

Fast execution of machine programmes is probably the main advantage. The following BASIC programme inverts the screen (rows 20 to 40). Row 60 places a machine programme in the range reserved from row 10 and starts it in row 70 -80. The screen previously inverse-switched by BASIC is now switched back to normal display at the speed of a ML programme. The difference in execution speed is more than clear.

```
 5 PRINT"BASIC START"
10 REM12345678901234567890 20 FOR 1%=28672 TO
29183
30 A=PEEKKI%)
40 B%=A% AND 64
50 IF B% THEN A%=A% AND 191   :'RESERVED       FOR       ML
ELSE A%=A% OR 64 60 POKE      PROGRAMME   :'SCREEN   INVERSE
I%,A%                         SHIFT :'WITH BASIC SPEED
70 NEXT 1%
80 FOR I%=31493 TO 31505 90 READ A
100 POKE I%,A% 110 NEXT I%
120 POKE 30862.5
130 POKE 30863,123 140 PRINT  :'ML PROGRAMME FROM DATA
"ML START" 150 A%= USR (0)    LINES : • READ AND ...
160 END                       :'ENTER IN REM LINE

                              : 'USR VECTOR LOW
                              SET :'AND HIGH BYTE SET
                              : • M L PROGRAMME ...
                              :' START
```

```
200 DATA 330,112,1,0,2,126,238,64,119 210 DAJA
35,11,120,177,32,246,201

                                        11th 5
```

This difference results from the 'pinginess' of the BASIC interpreter, who spends most of his time looking for bugs and making programming easier for the user; by providing prepared functions.

The time advantage of ML programming is that the user is without protection against possible programme errors. If the machine programme nm in (11.5) is made faulty with POKE 31509,0 and started with A=USR(0), it can be observed what causes a faulty programme.

11.7 Storage space for machine programmes

a) Reservation in BASIC text

As the example (11.5) shows, short ML routines within the BASIC programme can be loaded into space reserved by the REM command. However, the ML programme must not be longer than one BASIC line. If the ML programme is loaded, there will be difficulties in listing the programme. It should be worked very carefully during the editing phase and the BASIC programme should be backed up to cassette before the first test of the ML routine.

b) Reservation in Video RAM

2 KB of RAM is available for refresh memory. They are decoded in the 7000H to 7FFFH address range. In text mode ('MODE (0)') of the video controller, only 512 bytes are used for the screen image, the rest is available for user routines. As the screen RAM may be interfered with in the video due to uncontrolled access of the microprocessor, it is recommended to store interrupt routines and screen information.

c) Store in the 'Free Space' area

Appendix 11 shows that if the BASIC programme is running, there is a free space between the end of the variable tables and the end of the string area. Again, ML routines can be dropped. However, it must be ensured that the programme stored here cannot be overwritten by the dynamically changing BASIC stack.

d) Storage before the BASIC text

If the beginning of the BASIC text is moved to higher ranges, space can be created between the end of the system variables and the beginning of the BASIC text for machine programmes that are before all activities of the BASIC-

```
POK  32768.0 <CR>        :'ANNEXE 16, THREE TIMES '0'
POK  -32767.0 <CR>
POK  -32766.0 <CR>
POK  30884.1 <CR>        :'SET BASIC START POINTER :'TO
POK  30885,128 <CR>      8001H, 32769 :'SET VARIABLE
POK  30969.3 <CR>        START POINTER :' 'TO 8003H,
POK  30970,128 <CR>      32771
                                                11th 6
```

Interpreters are protected. A BASIC programme, which is then written and backed up to tape, is automatically put back in the same place when the cassette is loaded. The necessary transfer of the hands is done by POKE instructions directly from the keyboard. In the following example, the start of the BASIC text should be moved from the usual 7AE8H to 8000H (see also Appendix 11 and 16). In turn, the following instructions shall be given (11.6):

(e) The safest method: At the memory end!

The most commonly used method is to convert the 'Top Of Memory' pointer pointing to the last RAM cell by two POKE commands. This can also be done as the first statement in the BASIC programme. If CLEAR XX is then given to reserve space for strings, all TOM-dependent pointers in the operating system are automatically reset. The statement:

```
10 POKE 30898, PEEKC30898)-2: CLEAR 1000
```

reserves 2256 bytes for ML programmes, rewrites the pointers of the string area and the BASIC stack pointer, and reserves 1000 bytes for string storage. It is important that CLEAR is given with parameters, otherwise the pointer correction will not be performed. If necessary, CLEAR 50 or CLEAR 1.

11.8 Saving the M L programmes on cassette

To save a user-developed ML routine to cassette, enter the programme (11.8). As previously discussed, for this routine (as for the user programme), RAM space should be reserved and the loader should be started with the USR vector set and the call with X=USR(X).

11.9 Users Interrupt Vector

An interrupt pointer under the address 787DH in RAM allows to include user-written routines in the system interrupt. The only interrupt source is the video display generator, which every 20 msek an inter-

```
DI                      ;DISABLE INTERRUPT
LD C, F1 H              ; BINAR PROGRAMME
LD HL, (78A4H)          NUMBER ;RESCUE BASIC START
PUSH HL                 POINTER
LD HL, (78F9H)
PU SH HL                RESCUE BASIC-END-POINTER
LD HL,XXXXXXH
LD (78A4H),HL           STARTADR HERE.
LD HL,YYYH              SETTING ;POINTER ON START BIN
LD (789FH, HL           FILE ;END HERE. SET
LD HL, ZZZZ             ;END BIN FILE POINTER
CALL 34ACH)             ;ADR. PROGR NAME
POP HL                  ; AND SAVE ;MAKE BASIC-END
LD (78F9H),HL           POINTER
POP HL
LD (78A4H)              ;CREATE BASIC START-POINTER ;ALLOW
EI
RET                     INTERRUPT
                        ;RETURN TO BASIC
 Z77ZZ DEFM ' "NAME"
     ' DEFB OOH         ;PROGR. NAME MAX. 15
                        CHARACTERS ;FINAL NAME
```

11th 8

```
LD HL,7AE9H
LD (78A4H),HL           STARTADR.
JP 36CFH                BASIC ;RESTORE
                        ;AND JUMP AFTER BASIC START ;HERE
                        START THE USER ROUTINE
```

11th 9

ruptly for the Z 80 processor on the INT connector. MODE1 of the masked interrupt is used. Since the pulse triggering the interrupt is the vertical synchronous pulse of the video controller, access to the video RAM should only be made during the sync blanking gap in the interrupt programme. Otherwise, there is no need to avoid interferences in the image. If the CPU receives an interrupt pulse and is allowed to execute, then it will. the run continued at 0038H. (11.10) shows the interrupt service routine in ROM 1 of the LASER computers:

In 787DH, the code of the RET command is entered after the computer is turned on. Now a JP command can be implemented from a user routine to a separate interrupt routine.

The user interrupt routine can also be completed with RET to achieve a continuation of the ROM routine.

```
0038     JP 2EB8

2EB8     PUSH  AF        ;INTERRUPT SERVICE    ROUTINE
         PUSH  BC        RESCUE REGISTER
         PUSH  EN
         PUSH  HL
         CALL  787DH     ;USER INTERRUPT
         CALL  SCREEN    ;BUILD SCREEN
         CALL  CURSOR    ; LET CURSOR FLASH
         CALL  KEYBRD    ;CHECK KEYBOARD
         CALL  BUZZER    ;BEEP IF KEY GEDR.
         POP  HL         RESTORE THE REGISTER
         POP  EN
         POP  BC
         POP  AF
         EI
         RETI            ; STOP INTERRUPT
                                                  11/10
```

```
         ORG 787DH            ; USER INT RAM
INTRAM DEFS 3                 JUMP ;RESERVE THREE BYTES
START    EQU 7AE9             ; START USER ROUTINE


         ORG START
         DI                   ; DISABLE INTERRUPT
         LD SP,7FFFH          ; REINITIALISE STACK ;0BJ
         LD A,CH              CODE OF THE JP INSTRUCTION
         LD (I NTRAM) ,       ;IN THE RAM JUMP VECTOR ;START.
         A LD HL,             INTERR.ROUTINE ;IN JUMP
         USRINT               STATEMENT ;INTERRUPT ON
         LD INTRAM+) ,HL EI   ;CONTINUE WITH THE
                              ; ... FRONT PROGRAMME
                              ; CHECK KEYBOARD
                              ; USER INT PROGRAMME
USRINT   CALL  KEYBRD
                              DRIVE. FROM STACK ;CREATE
         POP  HL              REGISTER
         POP  HL              ; TO END THE INT ROUTINE
         POP  EN
         POP  BC              ;INT TURN ON
         POP  AF
         EI                                       11. 11
         RETI
```

12 For the assembler programmer

Keyboard query, CRUN/CLOAD, character to screen, string - output, compare symbol, next character test, compare DE/HL, test variable type, control codes,
Joysticks, Buzzer BEEP, Printer Control

This chapter presents a small collection of useful assembler routines for creating machine code.

Keyboard Query

The keyboard query routine starts at 2EF4H. The keyboard is queried and the key code is placed in register A. The registers AF, BC, DE and H L are amended. The example (12.1) waits for the button RETURN to be pressed.

```
SCAN    CALL 2EF4H          ; KEYBOARD
        OR A                QUERY ;KEY
        JR Z,SCAN           PRESSED? ;NOT
        CP r/JDH            PRESSED ;TEST CR
        JR NZ, SCAN.        ;NO CR
        RET                 ;BACK Z. CALLING PROG. 12/11
```

CRUN and CLOAD

The routine CRUN is called at 372EH, CLOAD at 3656H.' H L is used as a pointer to the programme name. The name is placed in a buffer. The quotation marks are part of the name. The string ends with a '0' byte. Both routines use the system RAM of the LASE R-OS.tems. User programmes should not use this area.

```
NAME    DEFM "TARGET"       ;NAME OF PROG TO LOAD. ;CLOSURE
        DEFB 0

                            ;DISPLAY PROGR.NAME IN HL ;
        LD HL,NAME          RUN CRUN
        J P 372EH                                  12/2
```

```
GAME    DEFB 0              ; CLOAD WITHOUT
                            PROGR.NAME

        LD HL GAME          ;POINT TO NAME ;
        JP 3656H            RUN CLOAD
```

12/3

Character to screen

The character output routine is invoked at Q33AH. The character defined by the ASCII code in A is output to the location of the screen specified by the cursor pointer. Registers will not be changed.

```
        LD A,'A'            ;CODE 'A' IN REGISTER
        CALL 033AH          A ;ISSUE CHARACTERS
        LD A, ODH           ; ISSUE CARRIAGE RETURN
        CALL 033AH
```

12/4

string output

Start address is 28A7H. The HL tab points to the string. It completes with an 'O' byte. All registers are used.

```
        LD HL,MSG           ;HL IS POINTER ON STRING ;
        CALL 28A7H          ISSUE STRING

MSG     DEFM 'READY'        ; STRING
        DEFB ODH            ; CARRIAGE
        DEFB 0              RETURN ;TERMINATO
                            R
```

12/5

RST 08H · Compare Icon

A string that the pointer in HL points to is compared to the character codes after the RST 08H calls. If the match is found, H L is incremented, the control is passed to the following RST 08H statement and the next character is checked. If no match is detected, an error message (SYNTAX ERROR) is reported and the programme branches to re-enter.

```
;COMPARES CHARACTER CHAIN, WHICH HL SHOWS ;WITH
CHAIN 'A=B=C'

        RST O8H             ;TEST WHETHER 'A'
        DEFB 41H            ; HEXCODE FOR
        RST O8H             A ;A FOUND
        DEFB 3DH            ;NOW TEST, WHETHER
        RST O8H             '=' ;OK, NOW TEST,
        DEFB 42H            ;O 'B'
        RST O8H             ;OK, TEST
        DEFB 3DH            OB '='
        RST O8H             ;OK, TEST
        DEFB 43H            ;OK, STRING IS A=B=C


                                                  12/6
```

Test next" icon

A will be loaded with the next character, which the register pair H L points to. The carry flag is set if it is an alphanumeric character. Blanks and the OBH and 09H control codes are ignored and the next character is loaded and tested. String to be tested must be completed with 'O'. H L points to the initial address of the string minus 1 and is incremented with 1 before loading a character.

```
;A CHARACTER CHAIN IDENTIFIED BY HL IS CHECKED WHETHER IT IS
PART OF A VARIABLE ALLOCATION, WHETHER THE '=' ; A CONSTANT
OR VARIABLE NAME FOLLOWS.

        RST O8H             ;TEST, WHETHER '=', YES?: THE
        DEFB 3DH            FOLLOWING
                             ADDRESS CHARACTERS
NEXT    JR NC, VAR          ;ON '=' FOLLOWING CHARACTER
        CALL 1E5AH          IN ;NC IF VARIABLE NAME
        JR SKIP             ; GET VALUE OF
VAR     CP 2BH              CONSTANT ;NEXT JOB
        JR Z, NEXT          NOT NUMERI SCH +, -, ALPHA? ;+,
        CP 3DH              NEXT CHARACTER
        JR Z, NEXT          ;TEST OB -
        CALL 26ODH          ;-, NEXT CHARACTER
                            ;OK, ALPHA CHARACTERS, SEARCH
                            FOR ;VARIABLE NAMES NOW
                                                  12th
                                                  7
```

RST 18 - Compare DE to HL

DE and HL are numerically compared. DE is subtracted from H L. Signed integer numbers can only be processed in positive range. Only the A-register is used. The result of the comparison will be placed in the Status register:

```
                        – HL < DE
                        – HL > EN
CARRY SET               – UNEQUAL
NO CARRY                – EQUAL
NZ
or

IN A STRING (HL SHOWS AT THE BEGINNING), THE FOLLOWING VALUE
IS TESTED AFTER ;'=' WHETHER IT IS IN THE RANGE 100-500 ;S.

        RST 08H             TEST TO '='
        DEFB 3DH            , –
                            ;'=' FOUND, NEXT CHARACTER ; IF NOT
        JR NZ,ERR           NUMERIC
        CALL 1E5AH          ; HOLE BINAERERERT
        LD HL,5OO           ;UPPER LIMIT VALUE
        RST 18H             ;COMPARISON WITH BINERT
        JR C, ERR           ; CARRY, IF VALUE >500
        LD H L, 1           ;LOWER LIMIT TEST
        00 RST 18H          ;COMPARISON
        JR NZ, ERR          ;NO CARRY, MISTAKE!

                                                        12/8
```

RST 20H · Test Variables Type

The routine returns with a combination of numeric values in the A register and status flags. It is orientated to the 'Data Mode Flag' (78AFH) and thus determines the number type in the WRA 1 arithmetic register. The usage should be controlled, because some subprogrammes called can change the type flag and so the value in WRA 1 and the flag no longer match.

```
     TYP       STATUS       A-REG
     E
02   INTEGER   NZ/C/M/E     -1
03   STRING    Z/C/PIE        0
04   SNG.PREC  NZ/C/P/O       1
08   DBL.PREC  NZ/NC/P/E      5


;AFTER AN INTEGER ADDITION, THE DATA TYPE ;IS TESTED
TO DETERMINE THE TYPE AT OVERFLOW

         LD  A,2              ;TYPCODE INTEGER
         LD  78AFH) ,A        ;IN FLAG REGISTER
         LD  BC, (VAL1        ; FIRST VALUE
         LD HL, (VAL2)        ; SECOND VALUE
         CALL 0B2DH           ;DO INTEGER ADDITION
         RST 20H              ;OVERFLOW TEST
         JP  M, OK            ;INTEGER RESULT
                              ;NOT INTEGER
                              ;TEST OTHER TYPES
OK       LD  (SUM),HL             ; RETURN INTEGER RESULT
VAL 1    DEFW 125             ;16 BIT INTEGER
VAL2     DEFW 4235            ;16 BIT INTEGER
SUM      DEFW 0               ;HERE'S THE RESULT     TOWARDS
                                                      12/9
```

Display Control Codes

All cursor functions can be executed in machine programmes.

```
CURSOR  LEFT     08H
CURSOR  ADD.     1BH
CURSOR  DEX.     0AH
CURSOR  RIGHT    09H
RUBOUT           7FH
INSERT           15H
CURSOR HOME      1CH
CLEAR SCREEN     1FH


         LD A,1CH             ;CURSOR
         CALL 033AH           HOME ;DO
         LD A,1FH             ;CLEAR SCREEN;
         CALL 033AH           D0
                                       12/10
```

3 Programming joysticks

In programmes with machine code the joysticks can be read even more easily than under BASIC. The following example reads the joystick matrix and returns the joystick status as a result in tabs B and C. B contains the status of the right joystick, C the status of the left joystick. (See also Chapter 6: Programming the joysticks.)

```
BIT 5         4         3         2         1         0 EW.
      FIRE A FIRE B LINKS                RIGHT DOWN. ;1.
JOYSTK IN A,(2EH) OR OEOH                 READ SERIES
         CPL
         LD B,A
         IN A, (2DH)
         BIT 4,A                 ;2. SERIES
         JR NZ,
         JOYST1 SET
         5.B                     RIGHT JOYSTICK STATUS IN B ;3.
JOYST1   IN A,2BH 0R OEOH CPL    READ SERIES
         LD C,A
         IN A,(27H
         BIT 4,A
         RET NZ                  ;4. READ SERIES
         SET 5.c
         RET
                                 ;LINKER JOYSTICK STATUS IN C
                                                          12th
                                                          11
```

buzzer

Under the address 345CH is the routine that controls the piezo speaker. Before calling, the H L register shall be loaded with a number corresponding to the desired pitch and the SC register with the code of the duration (Duration). All registers are used by the routine.

```
LD HL,259              ; FREQUENZ
LD BC,75               CODE; TON
CALL 345CH             DAUER
                       ;CALL TONROUTINE 12. 12

CALL 3450H             ; CREATE 'BEEP'
                                      12/13
```

The audio frequency encoding is inversely proportional to the audio frequency. Small numbers represent high frequencies and large numbers deep. For example, the 'small C' is encoded with the decimal number 526, the 'C' with the number 259, and the 'high C' with 127. The following routine (10.12) produces a sound 'C' with a sound time of 75 vibrations.

beep routine

The routine used by the operating system, which produces the characteristic 'beep' at every key stroke, starts from address 3450H. All but the HLR register contents are destroyed. In order to produce such a sound, the call is simply given according to (12.13).

printer control

To output a character to the printer, load the character code into the C-Register and call the printer driver routine in 058DH. After the call, the **ASCI** I code of the character is included in the A and C registers for further processing, the contents of all other registers are destroyed. The letter 'a' (ASCI 1 97 decimal) is issued as follows (12.14):

```
LD C,97              ;LOAD CODE IN C-REG
CALL 058H            ; CALL PRINTER DRIVERS
```

12.14

If a CARRIAGE RETURN control code is sent to the printer via the driver, a LINEFEED code is automatically sent. If the driver is called '0' with C-Register, the driver routine checks the printer status and returns the result. Set or delete bit 0 of the A-register. In addition, the BREAK key is tested and the CarryF I ag is pressed.

Printer Status

A routine that checks the printer status starts at 05C4H. When called, it loads the status (1/O port 00H) into the A-register and returns. Bit 0 is '1' when the printer detects the **'BUSY'** signal, ・and '0' when the printer is ready for reception. No other tabs will be changed.

```
TEST     CALL 05C4H          ;TEST PRINTER
         BIT 0,A             READY ;TEST BIT 0
         JR NZ,TEST          :TYPES WHEN BUSY
```

12th
15

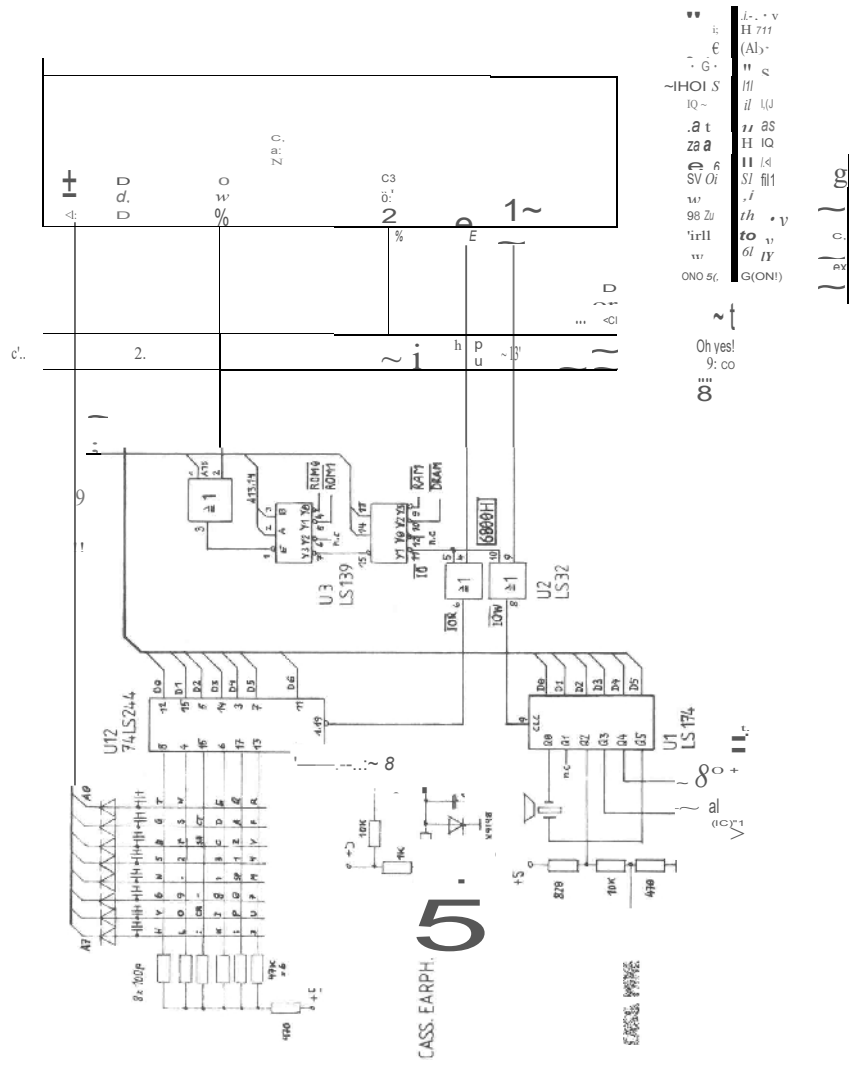
CR/LF to Printer

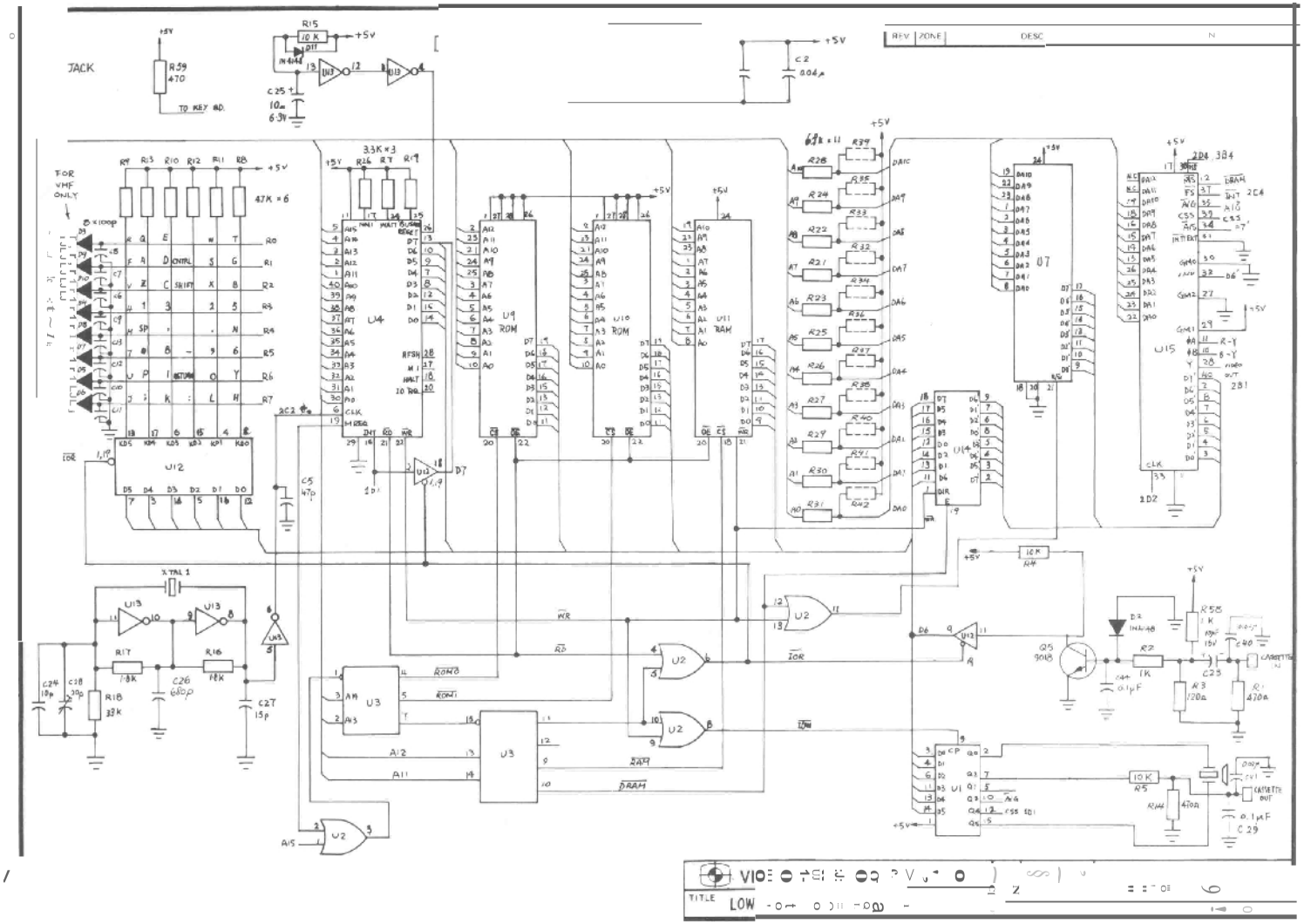
A routine starting from 3AE2H can be called to output the CR/LF combination to the printer. There is no need to set a register in advance. The contents of all registers will be changed. If the BREAK key is pressed while the print process is in progress or while the READY status is in progress, the routine returns with the carry flag set.

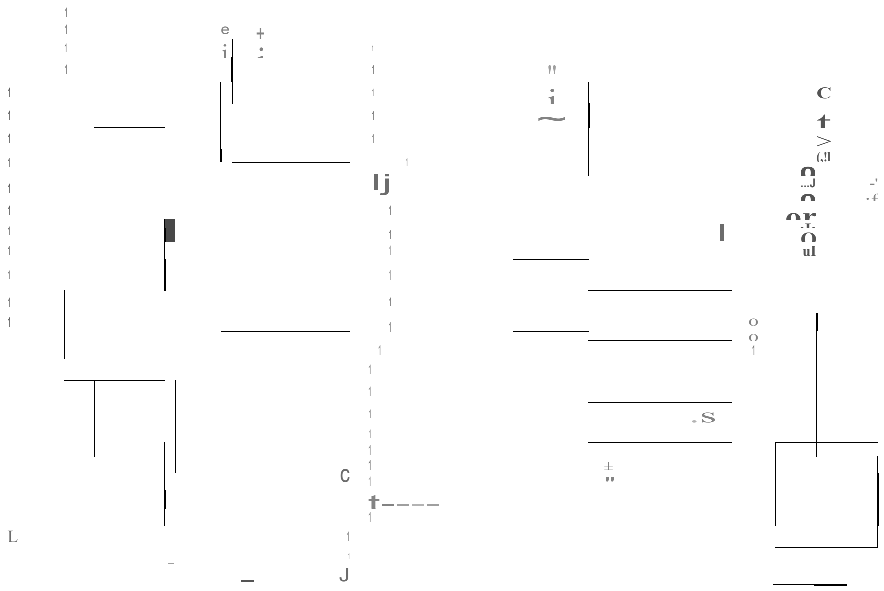
```
CALL 3ÄH              ; SENDS CR/LF TO PRINTER ;TO BRK, IF
JP C, BRK             KEY PRINTED ;NEXT IF NOT
```


Attachments

A table of contents for the attachments is located at the start of the book.

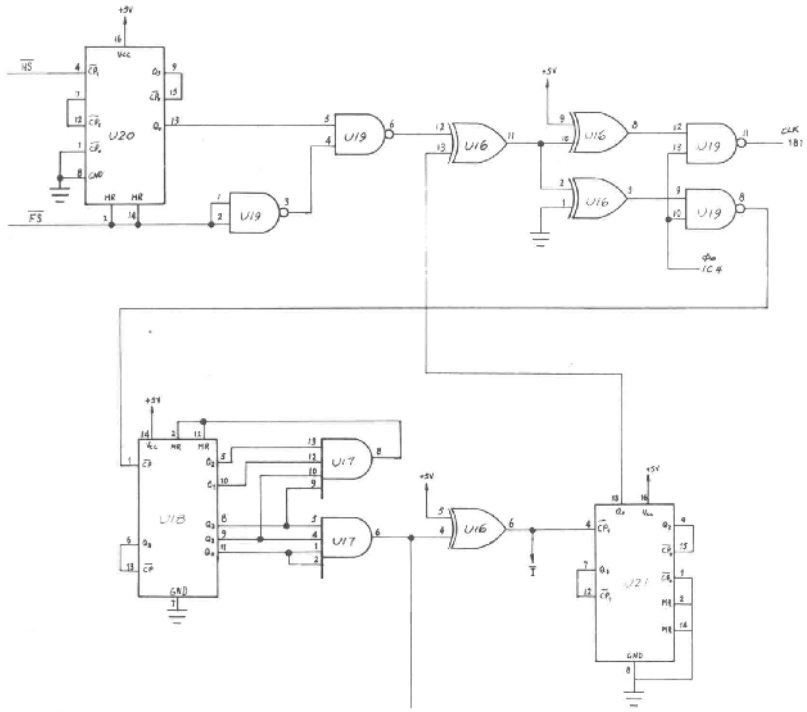

U3
LS139
U2
LS32
U12
74LS244
U1
LS174
CASS. EARPH.


JACK
R59 470
TO KEY BD.
R15 10 K
IN4448
C25 10µ 6.3V
+5V
C2 0.04µ
REV | ZONE | DESC
TOP VIEW
TITLE
LOW
FOR VHF ONLY
47K x 6
3.3K x 3
U4
U9 ROM
U10 ROM
U11 RAM
U12
U15
U7
U14
U1
U2
U3
U13
XTAL 1
C24 10p
C28 39p
R18 33K
R17 18K
R16 18K
C26 680p
C27 15p
C5 470
C29 0.1µF
R55 1K
Q5 9018
D2 IN4148
C40 100µF
C23 0.1µF
R3 120Ω
R1 470Ω
R2 1K
R5 10K
R44 470Ω
CASSETTE OUT
CASSETTE IN
RAM
ROM0
ROM1
WR
RD
IOR
IOW
A12
A11
A15
D7
D6
D5
D4
D3
D2
D1
D0



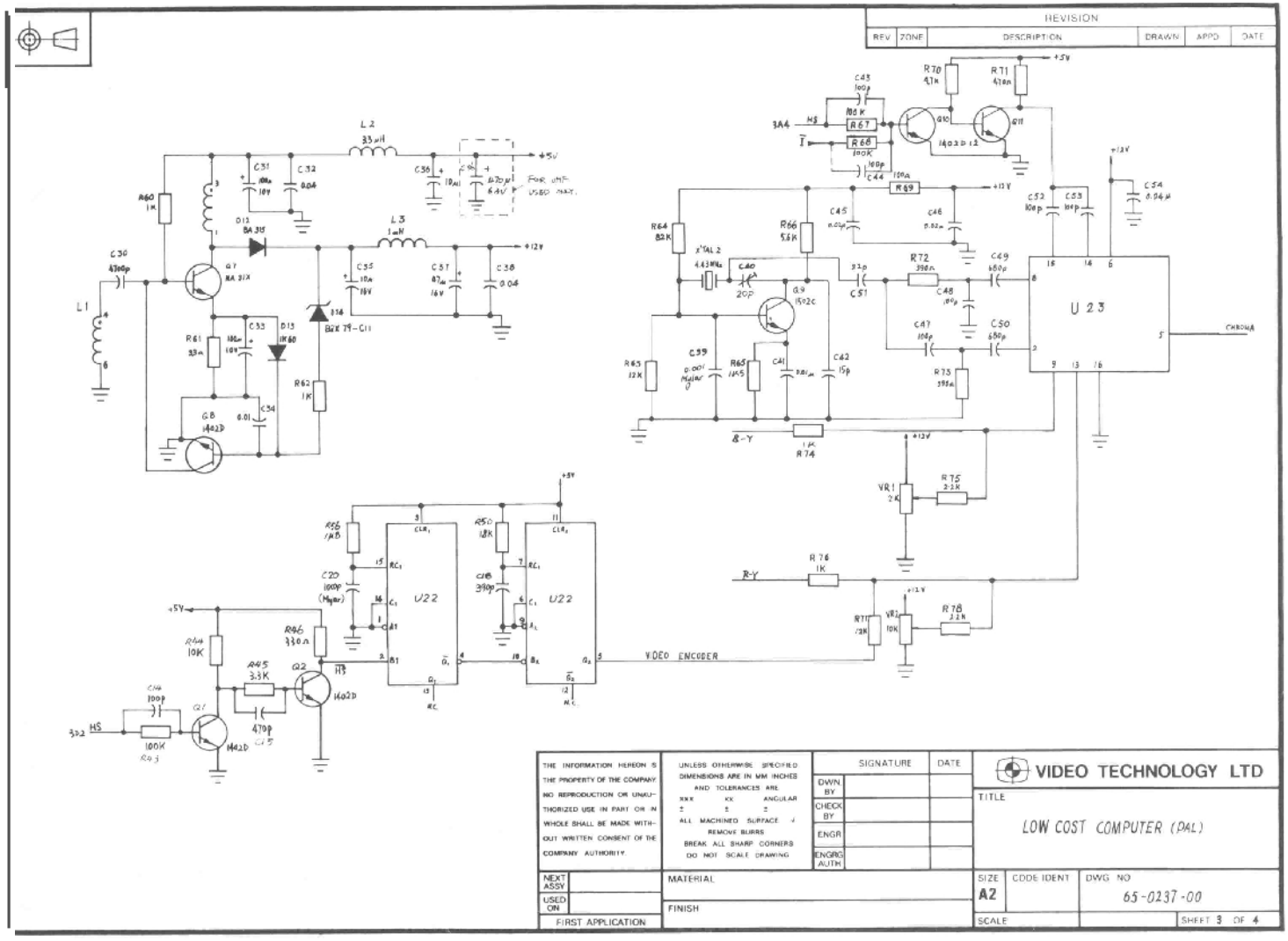
±Z
#4
±

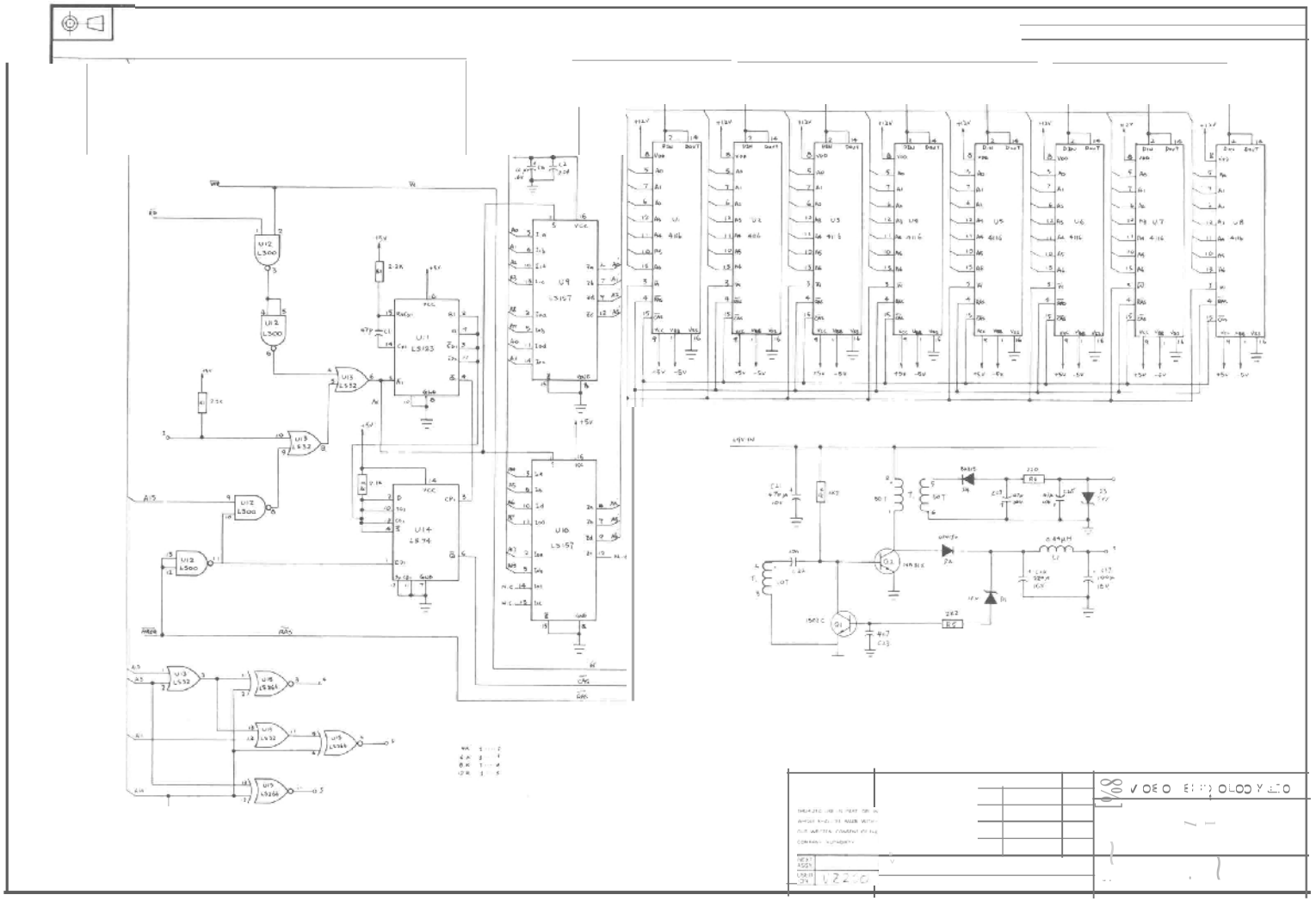
REVISION					
REV	ZONE	DESCRIPTION	DRAWN	APPD	DATE

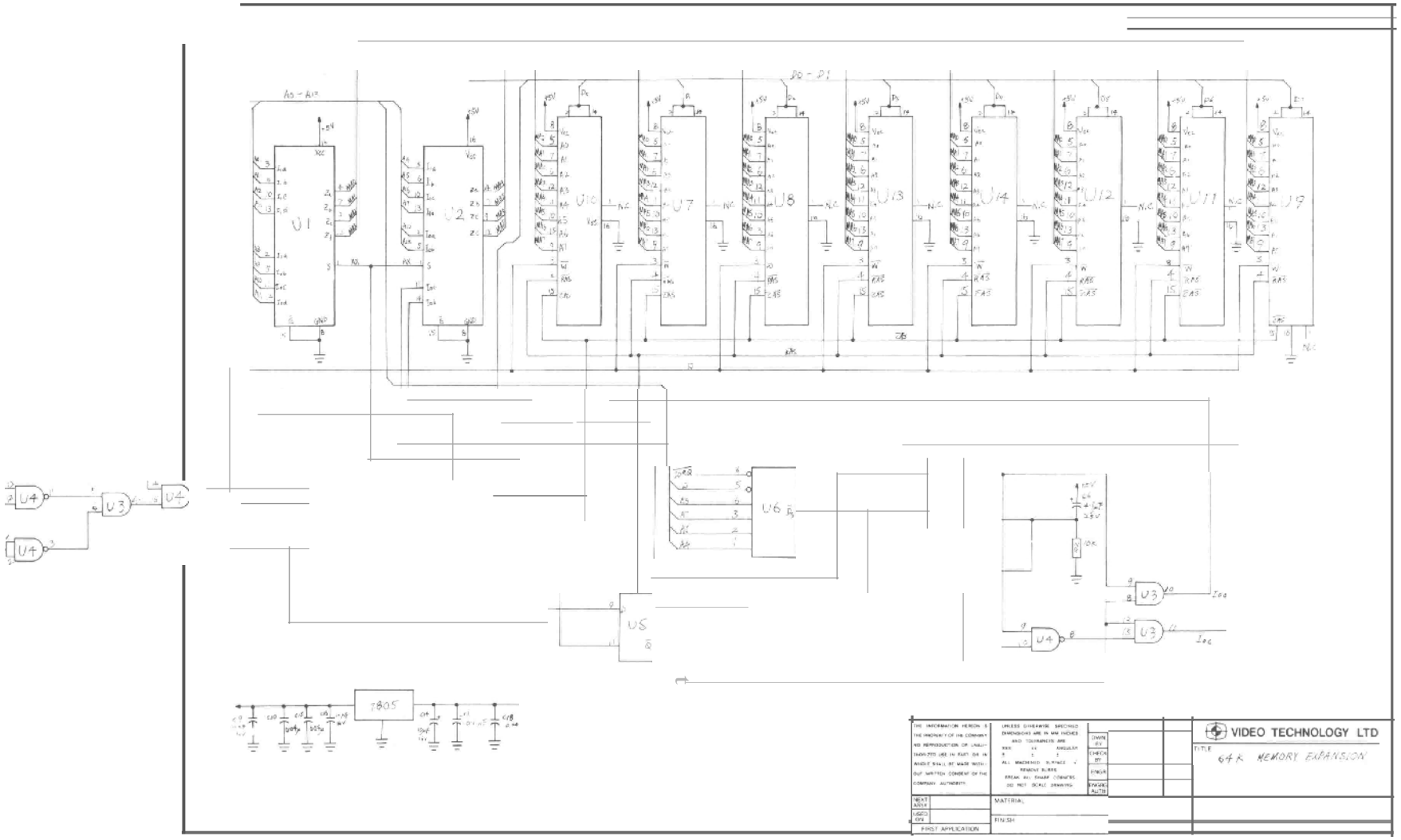


1. '-00' FOR
2. '-01' FOR

THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITH OUT WRITTEN CONSENT OF THE COMPANY AUTHORITY	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE xxx ± xx ± ANGULAR ± ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING	SIGNATURE / DATE: DWN BY; CHECK BY; ENGR; ENGRG AUTH	VIDEO T TITLE: LOW COST COMPUTER
NEXT ASSY; USED ON; FIRST APPLICATION	MATERIAL; FINISH	SIZE A2; CODE IDENT	DWG NO 65-0237-00; SCALE; SHEET 2 OF 4


REVISION
REV | ZONE | DESCRIPTION | DRAWN | APPD | DATE
L2 33 uH
L3 1 mH
L1
C30 4700p
C31 100u 10V
C32 0.04
C36 10u
C37 47u 16V
C38 0.04
C55 10u 16V
C53 10u 16V
C54 0.04 u
D13 BA 315
D15 1N60
Q7 BA 317
Q8 1402D
Q9 1502C
Q10
Q11
Q2 1402D
Q1 1402D
R60 1K
R61 33n
R62 1K
R64 82K
R66 5.6K
R67 180 K
R68 100K
R69
R70 4.7K
R71 470n
R72 390n
R73 390n
R74 1K
R75 2.2K
R76 1K
R77 12K
R78 2.2K
R65 12K
R56 1MB
R50 18K
R46 330 n
R45 3.3K
R44 10K
R43 100K
VR1 2K
VR2 10K
C20 1000p (Mylar)
C18 390p
C39 0.001 Mylar
C41 0.01 u
C42 15p
C51 33p
C52 10p
C49 680p
C50 680p
C47 104p
C48 104p
C45 0.02u
C46 0.02u
C40 1p
20P
X'TAL 2 4.43 MHz
U 23
U 22
U 22
+5V
+12V
-12V
CHROMA
VIDEO ENCODER
R-Y
B-Y
HS
3A4
3A2
FOR UHF USED PART.
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE XXX XX ANGULAR ± ± ± ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING
SIGNATURE DATE
DWN BY
CHECK BY
ENGR
ENGRG AUTH
VIDEO TECHNOLOGY LTD
TITLE
LOW COST COMPUTER (PAL)
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL
FINISH
SIZE A2
CODE IDENT
DWG NO 65-0237-00
SCALE
SHEET 3 OF 4





THE INFORMATION HEREIN IS THE PROPERTY OF VIDEO TECHNOLOGY LTD. NO REPRODUCTION OF UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY.

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
.XXX ±X ANGULAR
.X ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING

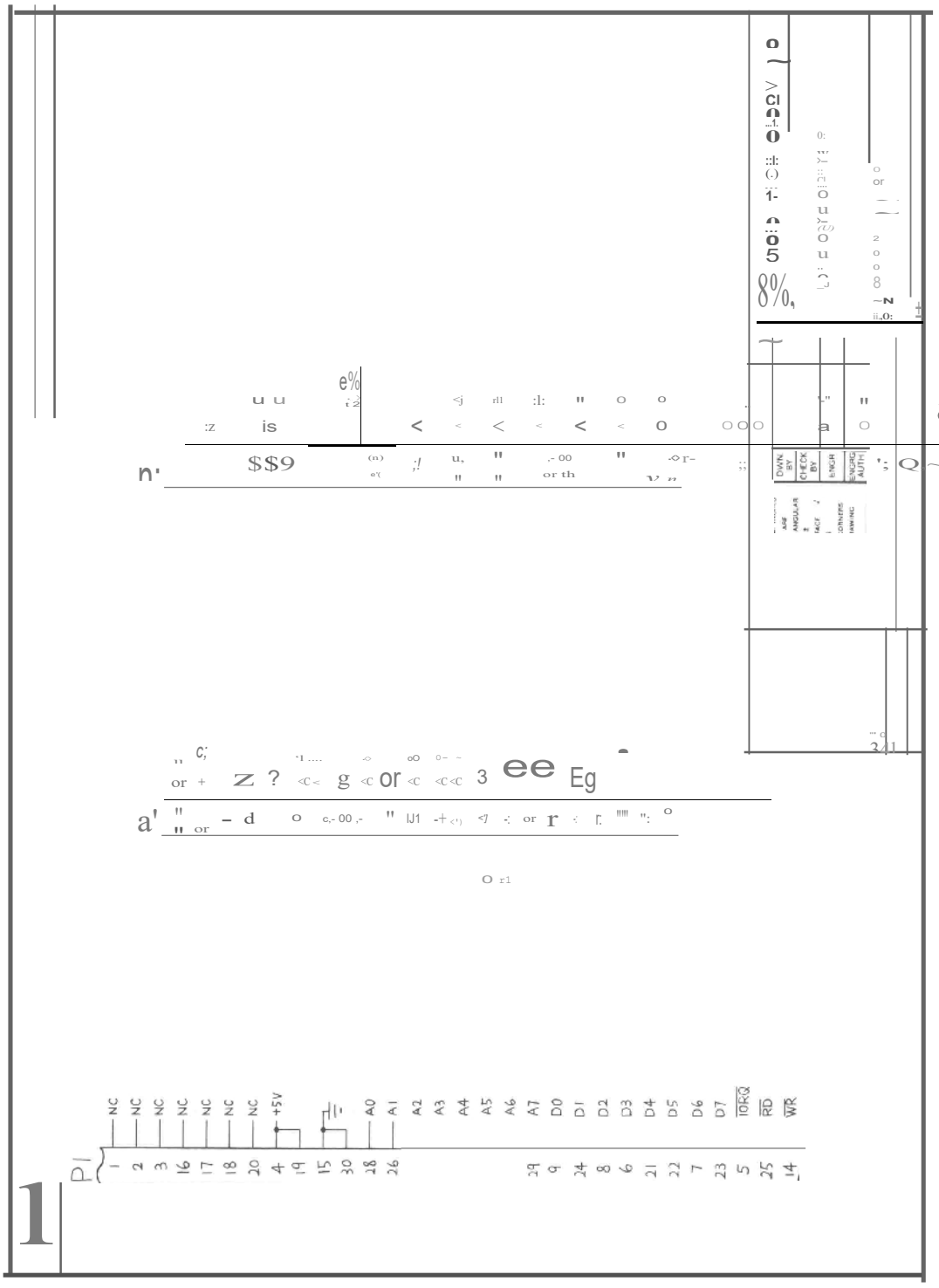
DRAWN BY
CHECKED BY
ENGR
ENGRG AUTH

NEXT ASSY
USED ON
FIRST APPLICATION

MATERIAL
FINISH

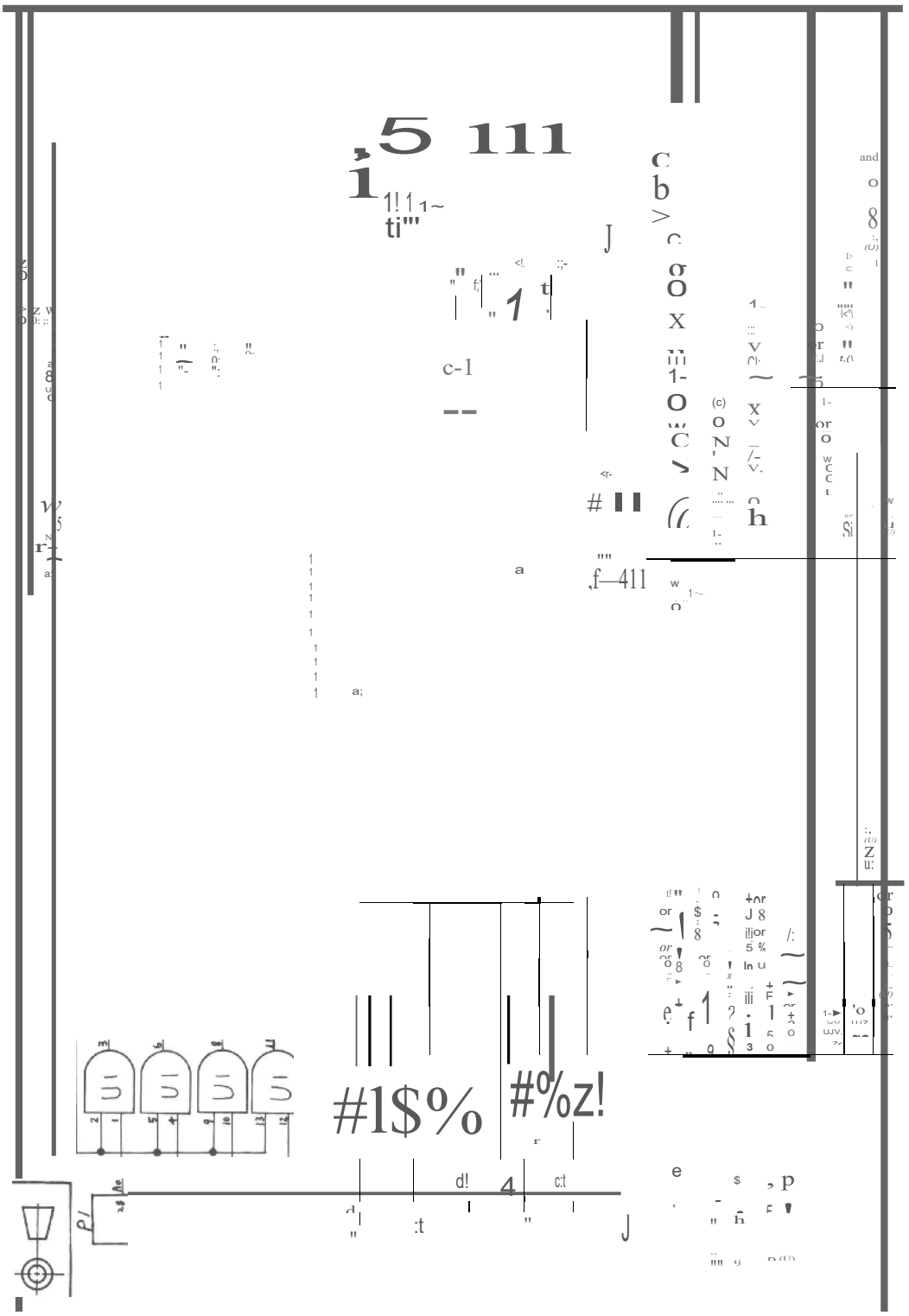
VIDEO TECHNOLOGY LTD

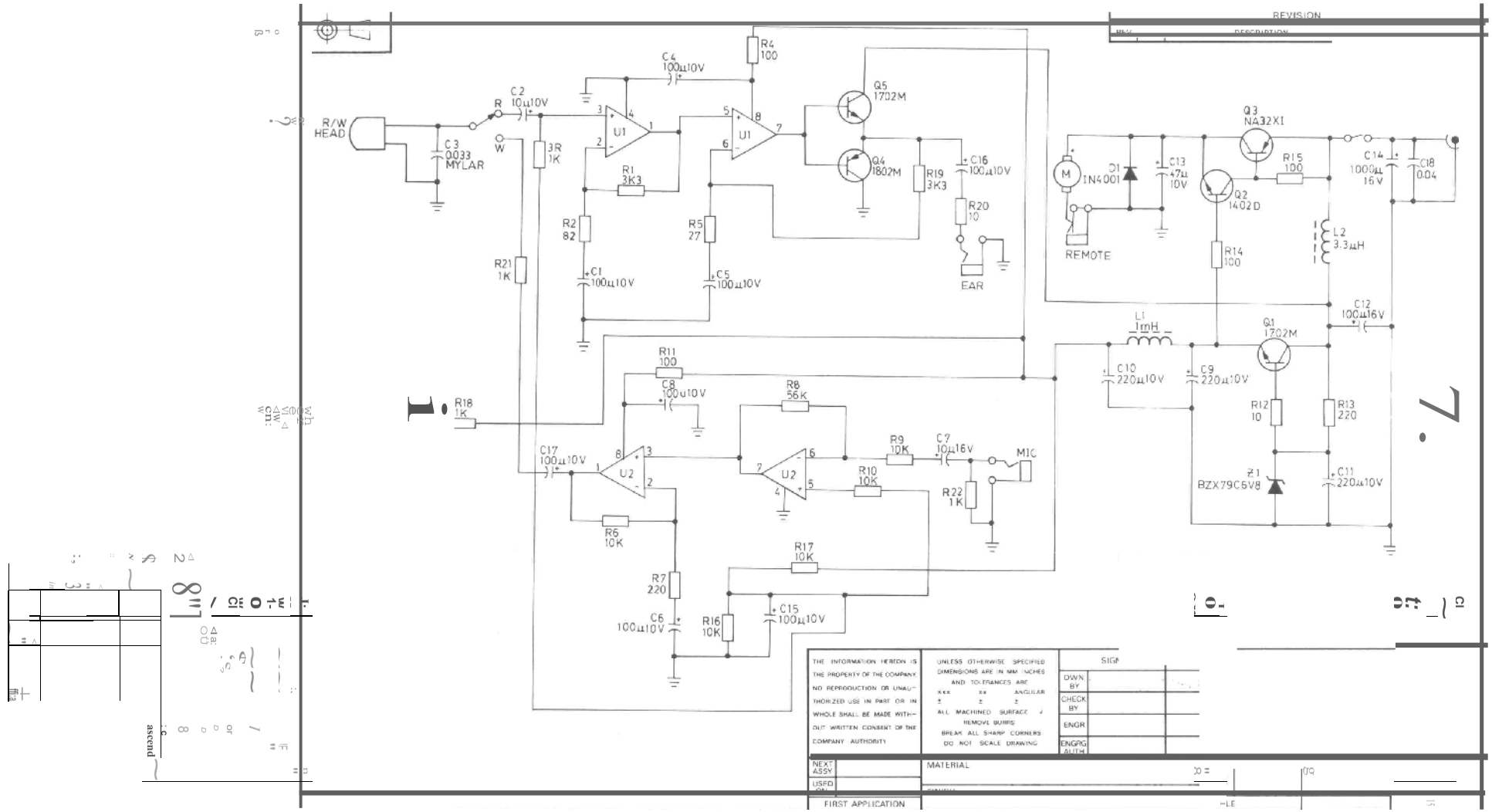
TITLE
64K MEMORY EXPANSION

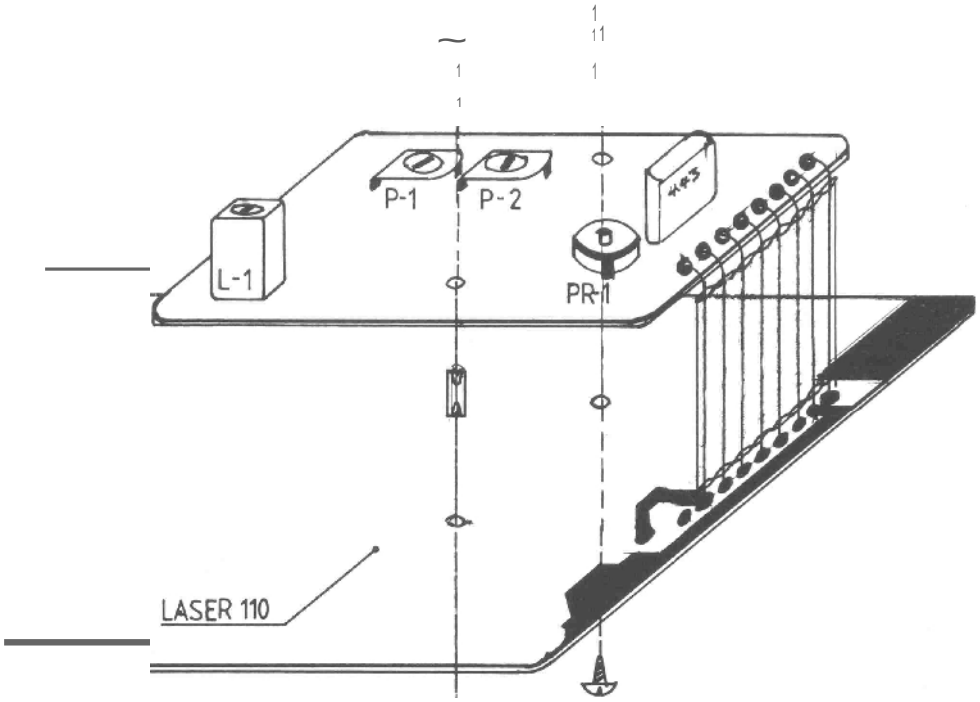

NC
NC
NC
NC
NC
NC
NC
+5V
A0
A1
A2
A3
A4
A5
A6
A7
D0
D1
D2
D3
D4
D5
D6
D7
IORQ
RD
WR
P1
1
2
3
16
17
18
20
4
19
15
30
28
26
29
9
24
8
6
21
22
7
23
5
25
14

6116
U4'
6116
U3'
6116
U2'
U1'
A0-A10
WR
RD
78XX
MREQ
A14
A15
A11
A12
A13

U1a
U1b
U3
4.7
A6
A5
A4
R/W


#1$% #%z!


REVISION
REV
DESCRIPTION
R/W HEAD
C3 0033 MYLAR
R
W
C2 10µ10V
3R 1K
C4 100µ10V
R4 100
U1
R1 3K3
R2 82
R5 27
C1 100µ10V
C5 100µ10V
R21 1K
Q5 1702M
Q4 1802M
R19 3K3
C16 100µ10V
R20 10
EAR
Q3 NA32XI
D1 IN4001
C13 47µ 10V
R15 100
C14 1000µ 16V
C18 0.04
Q2 1402D
L2 3.3µH
REMOTE
R14 100
C12 100µ16V
L1 1mH
Q1 1702M
C10 220µ10V
C9 220µ10V
R12 10
R13 220
Z1 BZX79C5V8
C11 220µ10V
R11 100
C8 100µ10V
R8 56K
R18 1K
C17 100µ10V
U2
R9 10K
C7 10µ16V
MIC
R10 10K
R22 1K
R6 10K
R17 10K
R7 220
C6 100µ10V
R16 10K
C15 100µ10V
THE INFORMATION HEREON IS THE PROPERTY OF THE COMPANY. NO REPRODUCTION OR UNAUTHORIZED USE IN PART OR IN WHOLE SHALL BE MADE WITHOUT WRITTEN CONSENT OF THE COMPANY AUTHORITY
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE
XXX XX ANGULAR
± ± ±
ALL MACHINED SURFACE √
REMOVE BURRS
BREAK ALL SHARP CORNERS
DO NOT SCALE DRAWING
SIGN
DWN BY
CHECK BY
ENGR
ENGRG AUTH
NEXT ASSY
USED ON
FIRST APPLICATION
MATERIAL



LASER PAL COLOUR

BOARD 80 - PAL

INSTALLATION GUIDE

Synchronisation

P-1, P-2
alternately set to green inner field and colourless border
PR-1 to adjust to minimum margin disturbances.
L-1 to~ adjust minimum disturbances in the image.

Annexe 9: Variable formats

```
10 POKE 30978,8 : REM VARIABLE BIN DBL PREC 20
DIM A, AA, BB, Z1$, AA (3,33)
$30.21 = "TEST"
40 AD= PEEK (30970) * PEEK (30969) 45
REM POINTER ON VARIABLE TABLE 50 FOR
I = 0 TO 80
60 PRINT PEEK AD + I; NEXT I
```

AZ INTEGER	2	0 65	0	VALUE
SINGLE PRECISION AA	4	65.65	0 0 0 0	
DOUBLE PRECISION BB	8	66.66	0 0 0 0 0 0 0 0	
STR ING $21	3	49.90	4	22,123 %

VARIA-nY 3 — POINTER ON TEXT

NAME (ASCII) — t L — STRING STRING LENGTH

CONSEQUENCES VARIABLES AD, I

1 FIELD AA 3.3.3)	4	65 65	7 1	3	4 0	4 0	4 0

NUMBER OF BYTES T

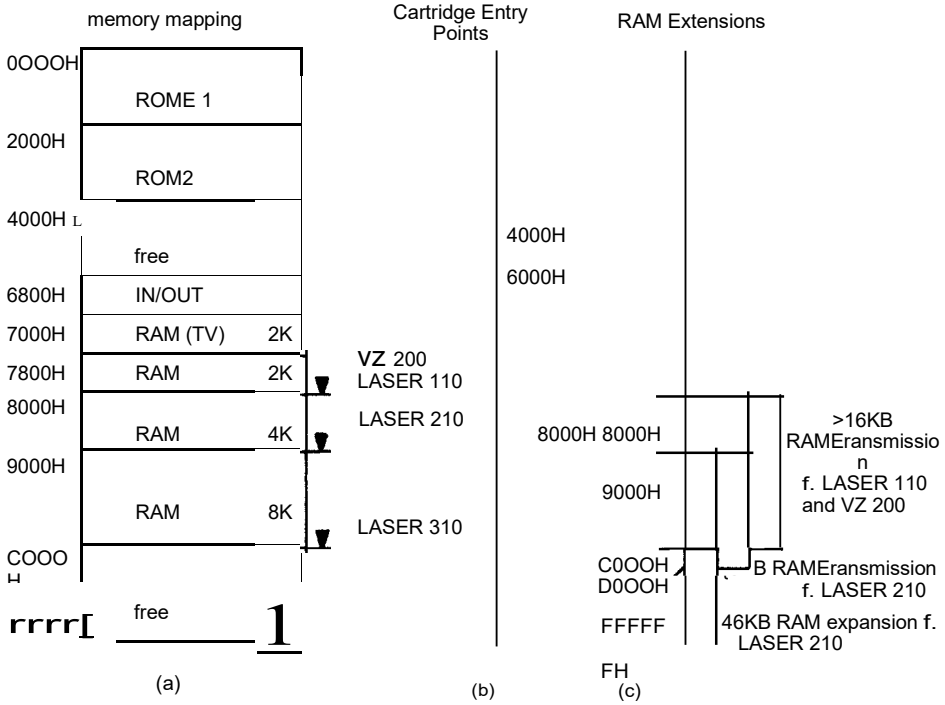
NUMBER DIMENSION — 1

ELEM! ELEM! 2. ELEM!

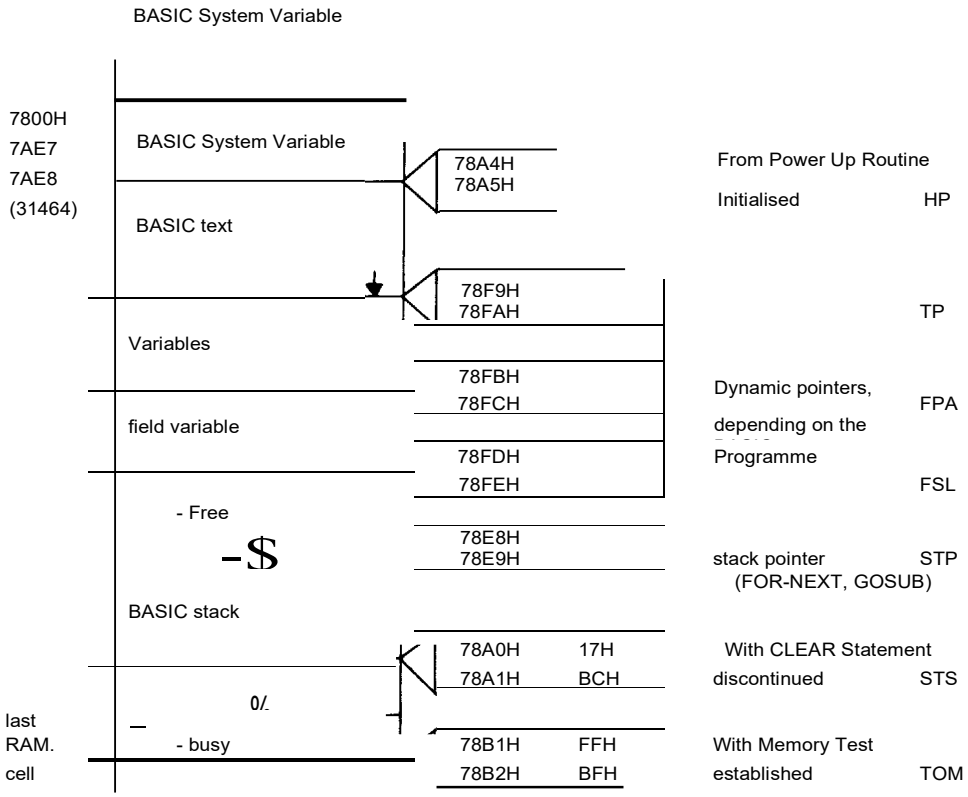
1. DIMENSION 3.

VALUE ..... 64 X 4 BYTES (TOTAL FIELD LENGTH=260 BYTES

Annexe 10


memory mapping
0OOOH
ROME 1
2000H
ROM2
4000H
free
6800H
IN/OUT
7000H
RAM (TV) 2K
7800H
RAM 2K
8000H
RAM 4K
9000H
RAM 8K
COOOH
free
(a)
Cartridge Entry Points
4000H
6000H
8000H
9000H
C0OOH
D0OOH
FFFFF
FH
(b)
VZ 200
LASER 110
LASER 210
LASER 310
RAM Extensions
8000H
>16KB RAMEransmission f. LASER 110 and VZ 200
B RAMEransmission f. LASER 210
46KB RAM expansion f. LASER 210
(c)

Annexe 11



Pointer values apply to the LASER 110 with 16 KB memory expansion. The arrows mark a dynamic border

The string area is defined with CLEAR 1000

HP =Text Head Pointer
TP = Text Tail Pointer
FPA = Field Pointer Top FSL = Free Space List
STP = BASIC Stack Pointer
STSP = String Space TOM = Top of Memory

780C 30732 not used until ... • • 7815
3741
7816 30742 ADDRESS OF KEYBOARD SCAN ROUTINE 7818
39744 not used **until ...** •
781C 30748
7820 3752 POINTS ON CURSOR ADDRESS
783B 3.0779 COPIE DES OUTPUT-LATCH CONTENTS 787D
3845 INTERRUPT EXIT ADDRESS
<CAN BE DIRECTED TO YOUR OWN INTERR.ROUTINE

788E	30862	USR JUMP VECTOR
78903	**30864**	COINCIDENTAL NUMBERS
7893	39867	GENERATOR 'IN', ADDRESS,
7896	38870	VALUE 'OUT', ADDRESS,
7999	30873	VALUE
789A	3087 4	LAST SIGN AFTER 'BREAK'. FLAG: ENTRY
789B	30875	TO RESUME ROUTINE
789C	3876	**DATE** CHARACTERS IN CURRENT PRINT LINE OUTPUT
789D	3 0877	CHANNEL: !=PRINTER, 0=VIDEO, -!=CASSETTE FORMAT OF
789E	30878	DISPLAY ROW
789F	39979	PRINT LINE FORMAT
		reserved
7840	30880	STRING AREA DISPLAY CURRENT
78A2	30882	LINE NUMBER ADDRESS BASIC
7844	30884	START
78A6	30886	CURSOR COLUMN POSITION
78A7	30887	ADDRESS KEYBOARD BUFFER
78A9	30889	FLAG: 0=CASSETTE INPUT, OTHERWISE <>0
784	3089	RANDOM NUMBER GENERATOR
78AB	30891	VALUE OF REFRESH REGISTER LAST
78AC	30892	RANDOM <2BYTES)
78AE	30894	FLAG: 0=SEARCH VARIABLE, !=NEW VAR. ENTER TYPE FLAG
784F	329895	FOR VARIABLE IN ACCU WRA1
		2=INTEGER 3=STRING 4 • SNG.PREC 8=DBL.PREC INTERIM WHILE
78B.0	3,0896	EDITING AN EXPRESSION ADDRESS HciCHSTE RAM CELL <MEM
78B1	3,0897	SIZE>
78B3	3,0899	POINTER ON NEXT OPEN SPACE IN LSPT
78B5	391	LSPT <LITERAL STRING POOL TABLE>
78D2	3093	END LSPT
78D3	3,0931	LENGTH & ADDRESS OF A STRING TRANSPORTED TO THE STRING AREA

78D6 30934 POINTS TO NEXT AVAILABLE SPACE IN THE STRING AREA
78D8 30936 POINTS TO LAST BYTE OF CURRENT INSTRUCTION/ ED IT FLAG
DURING PRINT USING

78DA	30938	ROW NO. OF THE LAST READ DATA STATEMENT 'FOR' FLAG:
78DC	**30940**	59 FOR-LOOP KT. =NOT ACTIVE
78DD	30941	FLAG INPUT: =KEYBOARD OTHERWISE READ
78EN	**3.0942**	READ FLAG: =RAED-ANW. ACTIVE l=INPUT-ANW. ACTIVE ON
		USING: SEPARATOR BETWEEN STRING **L VARIABLE**
78DF	**30943**	PROGR.START-ADR. LOADING TO DOS AUTO FLAG: NO CAR
78E1	**30945**	<2CAR
78E2	**30946**	CURRENT LINE NO. IF INPUT
78E4	**30948**	reserved

79E6 **395 AT** INPUT: CURRENT ROW ADDRESS: ROW NO. THE CURRENT ROW

78E8 3952 POINTER **FR** BASIC-STACK
78E 3954 LINE WHERE LAST ERROR OCCURRED <ERL>
78EC **30956** ADDRESS OF THE COMMAND WHERE ERROR OCCURRED
78F0 AT THE BEGINNING. DER ON ERRORROUTINE
350960 ERROR FLAG: ERROR SETS **255,** RESUME SETS **ADR.**
78F2 **3962** DECIMAL POINT IN BUFFER
78F3 30963 LAST ACT. ROW NO., END, STOP SAVED. LAST BYTE
78F5 30965 WORKED ON ERROR
78F7 30967 POINTER END BASIC/ BEGIN VARIABLE
78F9 3 POINTER END VARIABLE/BEGIN FIELD VARIABLE
0969 78FB POINTS (END FIELD>IBEGINN FREE SPACE
3971 78FD POINTS TO CHARACTERS AFTER LAST
30973 7BFF CHARACTER READ
30975 DEFAULT TABLE VARIABLE TYPE NAMES 26 BYTES (A-Z>
7901.30977 DOES A VARIABLE NAME NOT CONTAIN A TYPE CODE HE WILL BE FROM THE **TAB.** TAKEN. NEW BEARS 04 A CHANGE BY BASIC-UP DEFINT,DEFSNG,DEFDBL TRACE FLAG: 175=TRACE ON =TRACE OFF
CARRY FOR PUSH OPERATIONS
X-REGISTER* WRA1 * LSB DBL. PREC
791B 31003 WRA1 DBL,PREC VALUE
791C 31004 WRA1 DBL.PREC VALUE
791D 31005 WRA1 DBL.PREC VALUE
791E 3106 WRA1 LSB INTEGER/ SNG.PREC
791F 31097 WRA1
7920 31008 WRA1 MSB SINGLE PREC
7921 31 09 WRA1 EXPONENT SNG.PREC
7922 31010 SIGN OF THE YIELD, AT MATH. ~ ARITHM. OPERAT
7923 31011 INTERMEDIATE. DBL.PREC ADDITION
7924 31012 Y • REGISTER # WRA2 % LSB
7925 31013 **RA2** MSB
7926 WRA2 EXPONENT
31,014
7927 not used
31,015 7
92D 31,021 INTERNAL PRINT BUFFER
LAST BYTE PRINT BUFFER
792E DBL.PREC DIVISION <DIVISOR>
31,022
792F 31023
7930 31024
7949 3149
794A
31,050
— OWN ENTRIES—

Annexe 13: Advanced ASCII Code, Screen Code

POKE	ASCH] cHAR		POKE	ASC1I CHAR		POKE	ASH/cHAR		POKE	ASCH] CHAR	
32	32	SPACE	33	33	1	34	34	"	35	35	#
36	36	$	37	37	%	38	38	8	39	39	
4Cf	40	(	41	41	)	42	42		43	43	+
44	44		45	45		46	46		47	47	/
48	48	II	49	49	1	59	50	2	51	51	3
52	52	4	53	53	5	54	54	6	55	55	7
56	56	8	57	57	9	58	58	:	59	59	;
6@	6@	\	61	61		62	62	>	63	63	7
9	64	@	1	65	A	2	66	B	3	67	C
4	68	D	5	69	E	6	70	F	7	71	G
8	72	H	9	73	1	70	74	J	11	75	K
12	76	L	13	77	M	14	78	N	15	79	O
16	8@	p	17	81	Q	18	B2	R	19	83	S
20	84	T	21	85	u	22	86	V	23	87	W
24	88	X	25	89	y	26	99	or	27	91	1
28	92	\	29	93	1	30	94		31	95	e
32	96	SPACE	33	97	1	34	98	"	35	99	#
36	1GII	$	37	1@91	%	38	102	&	39	103	,.
411	1g4	(	41	105	)	42	1f) 6		43	107	+
44	1118		45	1119		46	119		47	111	/
48	112	II	49	113	1	511	114	2	51	115	3
52	116	4	52	117	5	54	118	6	55	119	7
56	128	8	57	1'21	9	58	122	:	59	123	;
6@	124	\	67	125		62	126	>	63	127	RUBOUT
12B	128	┐	129	129	Gl	139	130	G	131	131	1
132	132	ʒ	133	133	[1]	134	134	purl!	135	135	1
136	136		137	137	R	138	138	I]	139	139	livelv
1411	14g	q	141	141	!II	142	142	~	143	143	▬
144	144		145	145	[yyy	146	146	lii.l	147	147	iiiil
148	148	ʒ	149	149	[1]	1511	1511	9	151	151	
152	152	e]	153	153	a	154	154	I]	155	155	livelv
156	156	e	157	157	It	168	168	!!	151	159	▬
16GI	1611		161	161	[yyy	162	162	lii.l	163	163	iiiil
164	164	ʒ	165	165	[1]	166	166	9	167	167	a
168	168	l	169	169		1711	170	%	171	171	livelv
172	172	~	173	173	1	174	174		176	175	▬
176	176	┐	177	177	[yyy	178	178	F	179	179	iiiil
18@	180	ʒ	181	181	[1]	182	182	9	183	183	g
184	184		185	185	a	186	186	I]	187	187	livelv
188	188	ʒ	189	189	!II	1911	1911	F	191	191	▬
64	192	fil|	66	193	A	66	194	Re	67	196	~
68	196	lci|	69	197	F	711	198	[E]	71	199	G
72	200	]8[	73	201	(I)	74	202	O	75	203	6
76	204	[IJ	77	295	f.il	76	206	l!ijl	79	207	(Q)
8	2118	F	81	2119	G	82	210	[Bl	83	211	[S]
84	212	III	85	213	O	86	214	|V!	87	216	Wed
88	216	%	89	217	|Y|	911	218	%	91	219	[D
92	220		92	221	a	94	222		96	223	B3
96	224		97	225	[I]	9B	226	F	99	227	purple
199	228	III	1g1	229	6	1112	230	8g	103	231	8
1@4	232	[D	1@5	233	O	1g6	234	G	197	235	[±1
1118	236	GJ	1@9	237	R	11%	238	┐	111	239	|Z|
112	240	II]	113	241	D	114	242		116	243	@
116	244		117	246	6)	118	246	6	119	247	7
118	248	8	121	249	9	122	250	(TI	123	261	III
124	252	g	126	263	E|	126	264	S	127	266	D

L110	L210
h'grey	green
white	yellow
d'grey	blue
d'grey	red

CONTROL CODE TABLE

ASCII	FUNCTION	ASCII	FUNCTION	ASCII	FUNCTION	ASCII	FUNCTION
Cf	CR/LF	1		2		3	
4		5		**6**		7	
8	CURSOR LEFT	9	CURSOR RIGHT	**14'**	CURSOR **DEX.** LF	11	
12		13	CR/LF	14		15	
16		**17**		18		19	
20		21	INSERT	22		23	
24	CURSOR LEFT	25	CURSOR RIGHT	26		27	CURSOR ADD.
28	CURSOR HOME	29	CR	**3**		31	CLEAR SCREEN

127 RUBOUT

TABLE OF CHARACTERS PRESENTED WITH POKE ONLY

POKE	CHAR	POKE	CHAR	POKE	CHAR	POKE	CHAR
192	⌐	193	L	194	'-J	195	(iiil
196	~	197	[I	198	5	199	~
200	F	201	!!iil	202	I]	203	E
204	~	205	~	206	~	207	▬
208	,,	209	C.	210	¡J	211	(iiil
212	~	213	[I	214	5	215	1
216	!!]	217	~	218	I]	219	
220	~	221	~	222	I	223	▬
224	D	225	[iil	226	.:J	227	(iiil
228	D	229	[I	230	3	231	ɔ
232	1	233	5	234	I]	235	E
236	~	237	~	238	~	239	▬
240	⌐	241	[iil	242	.:J	243	
244	~	245	II	246	5	247	
248	!!]	249	σ	250	IJ	251	Ë
252	~	253	~	254	~	255	▬

Annexe 14: Geometric Functions

Function	BASIC formulation
SECANT	SEC(X)= 1/COS(X)
COSECANT	CSC(X)= 1 /SIN(X)
COTANGENT	COT(X)= 1 /T AN(X)
INVERSE SINE	ARCSIN(X)=ATN(X/SQR(-X・X+I))
INVERSE COSINE	ARCCOS(X)=-ATN (X/SQR(-X・X+l))+pi/2
INVERSE SECANT	ARCSEC(X)=ATN(X/SQR(X+X-1))
INVERSE COSECANT	ARCCSC(X=ATNX/SQR(X+X-1*))* +(SGN(X)-1)+pi/2
INVERSE COTANGENT	ARCOT(X)=ATN(X)+pi/2
HYPERBOLIC SINE	SINE(X)=(EXP(X)-EXP(-X))/2
HYPERBOLIC COSINE	COSH(X)=(EXP(X)+EXP(X))/2
HYPERBOLIC TANGENT	T ANH(X)=EXP(-X)/EXPX)+EXP(-X))" 2+1
HYPERBOLIC SECANT	SECH(X)=2/(EXP(X)+ EXP(-X)
HYPERBOLIC COSECANT	CSCH(X)=2/(EXP(X)-EXP(-X))
HYPERBOLIC COTAN GENT	CQTH(X)=EXP(-X)/(EXP(X)—EXP(-X)+2+1
INVERSE HYPERBOLIC SINE	ARCSINH(X)=LOG(X+SQR(X+X+1)
INVERSE HYPERBOLIC COSINE	ARCCOSH(X)=LOG(X+SQR(X+X-1)
INVERSE HYPERBOLIC TANGENT	ARCT ANH(X)=LOG(J +X)/(l-X))/2
INVERSE HYPERBOLIC SECANT	ARCSECHXJ=LOG((SQR(-X+X+1)+1/X)
INVERSE HYPERBOLIC COSECANT	ARCCSCH(X)=LOG((SGNX)+SQR(X+X+1)/**X**
INVERSE HYPERBOLIC COT ANGENT	ARCCOTH(X)=LOG(X+ 1)/(X-1))/2

Annexe 15: Shorthands INSTRUCTION

CHARACTERS DESCRIPTION

PRINT	?	CAN BE USED IN BASIC TEXT, BUT IS LISTED AS 'PRINT'
REM		REPLACES REM
LET	=	INSTEAD OF LET A=5, A=5 CAN BE WRITTEN
THEN		MAY BE REPLACED OR REPLACED BY COMMA, E.G. IF A=0 GOSUB 1000 IF A<>B PRINT "XXXX" BUT: IF A>5 , 200
		MAY BE ELIMINATED WHEN THERE IS AN IMMENSE SEPARATION OF THE VARIABLES AND CONSTANTS TO BE ISSUED, E.G. PRINT A$ B$ "SEMIKOLON" BUT: PRINT A; B
GOTO		CAN BE ELIMINATED BY THEN, E.G. IF A>S THEN 200

Annexe 16: BASIC text format 10 A = 12

20 PRINT A

FOR I=0 TO 18:?PEEK(31464+I);:NEXT <RETURN>

30884.

[233[122]

HP 78A4H

314641 1

10 11242! 122~ 101

Start

in the row Pointer to Line No. A Token 1 2

the next binary

01 szr s] 0]

1

122*256+242=31474

314731

[o [so!a[zsl 01 1181 651

Token A

31480 PR1~0969

[[e[s["4]3]iq re zur"

128	END	129	FOR	130	RESET
131	SET	132	CLS	133	*
134	%	135	NEXT	136	DATA
137	INPUT	138	DIM	139	READ
14,0	LET	141	GOTO	142	RUN
143	IF	144	RESTORE	145	GOSUB
146	RETURN	147	REM	148	STOP
149	ELSE	15	COPY	151	COLOUR
152	VERIFY	153	<DEFINT>	154	<DEFSGN>
155	<DEFDBL>	156	CRUN	157	FASHION
158	SOUND <ERROR>	159	<RESUME>	16	OUT
161	<ON>	1	*	163	*
164	*	62	*	166	*
167	*	165	*	169	*
170	*	168	*	172	*
173	*	171	<SYSTEM>	175	LPRINT
176	CONT	174	POKE	178	PRINT
179	<DELETE>	177	LIST	181	LLIST
182	CLOAD	**18**	<AUTO>	1 4	CLEAR
1 85	TAB	18	CSAVE	187	NEW
188	USING	3	TO	193	*
191	<ERL>	186	<VARPTR>	193	USR
194	*	189	<ERR>	196	*
197	MEM>	192	POINT	199	%
200	NOT	195	INKE Y$s	202	THEN
203	▪ - ▪	198	STEP	2g5	14°
206	% ?	**21**	· 3°	**28**	'y°
209	·>	204	**AND**	**21**	OR
212	SGN	**27**	INT	**1**	,<
215	FRE>	21g	INP	214	ABS
218	SQR	213	RND	217	%
221	EXP	216		**220**	LOG
224	TAN			223	SIN
227	*			**226**	PEEK
23	LEN	225	**COS**	**229**	*
243	ASC	228	ATN	242	VAL
247				245	LEFT4
6,2				244	%
49,				8.2	%
252		$259	MID	51.	
	RIGHT$			255	
	* —				
		219	STR$		
		222			
			CHR$		
		244			
		.24			
		7			

* UNDOUBLED CODE

(XX> TOKEN IS RECOGNISED BY THE 'BASIC UP' EXTENSION.

Annexe 18: Tape Loading Format

	T: Text File	B: Binary File	D: Data File
SYNC. bytes	255 bytes of 80H	255 bytes of 80H	255 bytes of 80H
HEADER	5 bytes of FEH	5 bytes of FEH	5 bytes of FEH
EXTENSION	1 byte of F0H	1 byte of F1 H	1 byte of F2H
FILENAME	16 bytes (max.)	16 bytes (max.)	16 bytes (max.)
	of ASCI 1	of ASCII	of ASCII
CAP	3 ms Blank	3 ms Blank	3 ms Blank
START ADDRESS	2 bytes of binary	2 bytes of binary	
END ADDRESS	2 bytes of binary	2 bytes of binary	
Programme Content	xx bytes	xx bytes	
Data Content			xx bytes
checksum	2 bytes	2 bytes	2 bytes
End of File	20 bytes of Zeroes	20 bytes of Zeroes	
Marker (EO F)	(00H)		
Terminator			1 byte of 00H

Annexe 19

Comparison of LASER-BASIC to various BASIC dialects Notes on rewriting published programmes

1. LASER-BASIC supplement with "EXTENDED **BASIC":**

ON A GOSUB 100,200,...	DEFINE	VARPTR
ON A GOTO 100,200,...	DEFSNG	POS (X)
FRE (0)	DEFDBL	RANDOM
FR E("")	DEFSTR	
ON ERROR GOTO	RECT	
ERL	CIRCLE	
ERR	PAINT	
RESUME	MEMSIZE = H IMEM	
LPEN	PLOT ... TO...	

2. Sound Commands

Statement	**LASER-BASIC**
500 BEEP	500 SOUND 12, 1
500 CLICK	500 POKE 30862,80: POKE 30863,52 501 X=USR(0)

3. FOR ... **NEXT loops** FOR

I=0 TO V will pass once when V=0 is used. In other BASIC versions, the statements between FOR — NEXT will not be executed.

4. Variable names

Names longer than two characters are allowed in LASER-BASIC. However, only the first two characters will be counted. For example, TI99/4A uses 16 characters; in the SINCLAIR-BASIC, even arbitrary long names are used.

Exampl e: 10 VALUE=5 20 WORLD=12
30 PRINT VALUE; WEL T

A programme run results in 5 and 12 on the mentioned computers. However, the LASER-BASIC issues 12 and 12, since the variable WE has been addressed twice. An adjustment of the variable names should be made if necessary.

5. **Printer Control**

The Statements
200 OPEN 4,4,0
400 PRINT#4,'TEXT""
490 CLOSE4

are against

Replace 400 LPRINT "TEXT", delete

the OPEN and GLOSE statements.

6. Random numbers with **RND (X)**

In some BASIC versions, RND (X) produces random numbers in the range of Obis 1. Subsequent instructions convert these numbers to the desired range. All these instructions must be omitted, in LASER-BASIC argument X is the upper limit of the random number to be generated.

A=RND (49)

generates a number in the range 049.

7. String processing

BASIC DIALECTS	LASER BASIC
	LEFTA.1,)
	LEFT$(A$A)
(SEGA,1,4)	RIGHTSA$,) !
A$(TO 4)	MID$(A$, 6, 5)
A$ (4 TO)	MIDA,6,5)
SEGA.6.5)	MID$(A$,7,1
A$(6 TO 10)	
A$7)	

8. Retain characters from keyboard

GET 10 A$	
10 CALL KEY	AS=INKEYS:AS=LEFT$(A$,1)
	DITO

9th PRINT statements 100

PRINTI;SPCI;I2

	100 PRINT I;
	.11 FOR J=O TO I:PRINT" ";:NEXTI
100 PRINT "	0.1 PRINT 12
100 PRING "	20 CLS ;DELETE SCREEN PRINT
100 INVERS	100 CHR$28) ; CURSOR HOME
	.10 POKE :REM INVERS ON
	0 POKE :REM INVERS OFF
	100
	.10
	0

10 Screen Area

In most cases, a redesign of the screen layout is necessary, depending on the format of the image in rows and columns. The targeted output on the PRINT AT **X,Y** screen**;** shall be assigned to:

```
Z=32X+Y
PRINT@Z,
```

where X and Y is the screen coordinates and Z is the direct number of the screen location. It should be noted that some BASIC interpreters refer to the first screen position with O corresponding to X=O and Y=O, while others use 1 each.

The relative cursor control commands on VC and CBM computers are available on the LASER only with line-length PRINTCHRS(xx);CHRS(xx) ... statements. A new screen would make more sense.

11th Accuracy of figures

See Chapter 1: Double precision variables.

The LASER computer system

LASER 110 Monochrome Video and HF Output Basic Devices
4 KB RAM, 2.5 KB for user programmes.
LASER BASIC V 2.0

LASER 210 Colour Computer, 8 KB RAM, 5.5 KB for user programmes.
LASER BASIC V 2.0

LASER 310 Colour Computer with 18 KB RAM, Typewriter Keyboard, BASIC V 2.0

Accessories

16 KB RAM modules for all basic devices

64KB RAM module with BANK switch (4 banks 16KB)

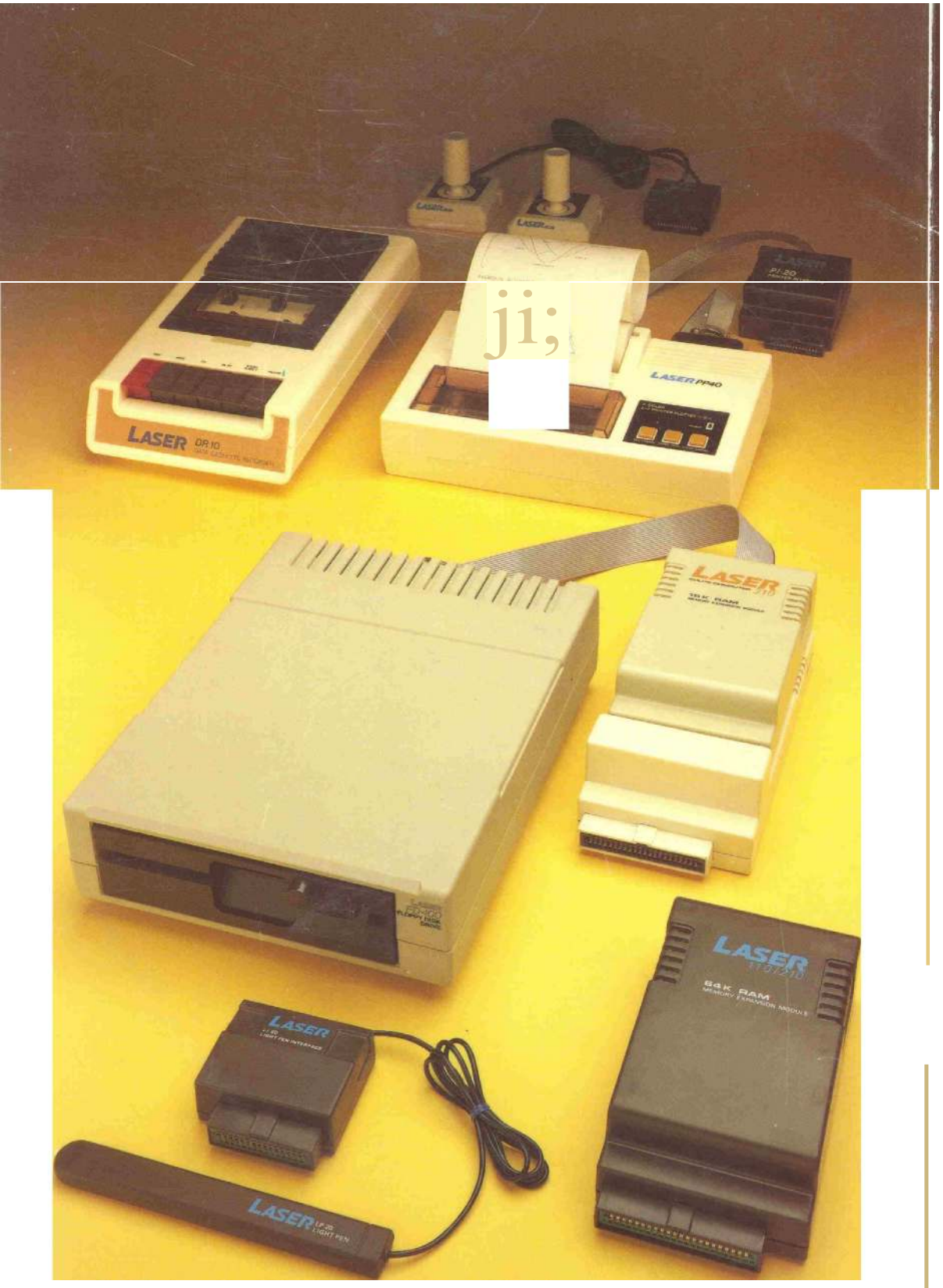
Printer interface with Centronics interface

Joysticks	Control stick for video games, programmable by BASIC
Lightpen	For sophisticated menu technology and Graphics Applications
Floppy Disk Controller	To connect two 5.25" drives
floppy drive	Drive, 90KB capacity, for 5.25" disks, double density
PP4O	four-colour printer/plotter, 40 characters/line. Paper: roll
Software	More than 50 programmes on cassette, including flashbox, Z-80 assembler, game collection

Sales and proof of purchase, software list of
SANYO VIDEO VERTRIEB, long series 29, 2 Hamburg 1

	USER-PORT-Module for 24 IN/OUT lines, interfacebaustein 8255
	Analogue digital converter, User-port operation Digital analogue converter, User-port operation

Sales: **1. Bockstaller, Hadwigstr. 16 7867 Wehr 2, tel.: 07761 - 18 08**



ji;

sl] sd

Software System, Manual 1