NIXDORF
COMPUTER

Nixdorf 8870 Quattro

Business BASIC 7.0

Edition: 01.06.85 incl. Rev. 1 Order no. 34822.00.9.93

Introduction 1

Syntax Elements 2

Summary of BASIC Statements and Commands 3

BASIC Commands 4

Data Formats and Data Representation 5

BASIC Statements 6

BASIC Functions 7

CALL Statements 8

Keyboard Input/Screen Output 9

Magnetic Disks 10

File Processing 11

Printers 12

Punched Card Readers 13

Paper Tape Readers/Punches 14

Magnetic Tapes 15

Magnetic Tape Cassettes 16

Program Structure 17

Testing Aids 18

TAMOS Interfaces 19

Program Maintenance 20

Programming Examples 21

Error Handling 22

Index 23

Modifications Sheet

Modifications sheet

This sheet lists all modifications made to this manual since the appearance of the first edition. It should be replaced by the new sheet provided whenever further modifications are announced.

First edition	01.06.85	Rel. 5.1
Revision 1	01.10.86	Rel. 7.0

- New sections

  9.9
  8.19-1
  8.19-2
  8.28-1

- Various supplements and corrections

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

*
*
*
*

Significant modifications to the previous version of this document are marked with * in the left margin of the pages.

The date of issue is indicated on each page.

Errors / Suggestions for Improvement

Errors / suggestions for improvement

If you have noticed any errors while using this section of the systems literature, or should you have suggestions for the improvement of the manual, please send your written comments to the following address:

NIXDORF COMPUTER AG
Abt. ZSI
Fürstenallee 7
D-4790 Paderborn

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

	Contents	Page
1	Introduction	1-1
1.1	BASIC and Business BASIC	1-1
1.2	Hints on using the manual	1-3
1.3	BASIC programming system	1-3
1.3.1	Creating a new BASIC program	1-4
1.3.2	Modifying a BASIC program	1-4
1.3.3	Testing a BASIC program	1-4
1.3.4	Running a BASIC program	1-4
1.3.5	Storing BASIC programs on disk	1-5
1.3.5.1	Selecting the appropriate intermediate code	1-5
1.3.5.2	Dumping and loading source code	1-5
2	Syntax elements	2-1
2.1	Description of the metalanguage	2-1
2.2	List of syntax elements	2-4
3	Summary of BASIC statements and commands	3-1
3.1	Summary of BASIC statements	3-1
3.2	Summary of BASIC commands	3-8
4	BASIC commands	4-1
4.1	ANALYSE command	4-2
4.2	AUTOLN command	4-4
4.3	COMMENT command	4-4
4.4	CREF command	4-6
4.5	DEBUG command	4-6
4.6	DELETE command	4-8
4.7	DUMP command	4-9
4.8	EDIT command	4-10
4.9	HASH command	4-11
4.10	HELP command	4-12
4.11	LIST command	4-12
4.12	LOAD command	4-13
4.13	NEW command	4-14
4.14	PARTRENUM command	4-14
4.15	PLIST command	4-15
4.16	RENUMBER command	4-19
4.17	RUN command	4-20
4.18	SAVE command	4-21
4.19	SIZE command	4-22
4.20	TRACE command	4-23
4.21	UNCOMMENT command	4-25

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

5	Data formats and data representation	5-1
5.1	Representation of numeric data	5-2
5.1.1	Simple numeric variables	5-2
5.1.1.1	Structure of BCD integers	5-2
5.1.1.2	Structure of floating-point numbers	5-3
5.1.1.3	Arithmetic overflow	5-4
5.1.2	Vectors/matrices	5-5
5.1.3	Numeric constants	5-6
5.2	Representation of alphanumeric data	5-7
5.2.1	String variables	5-7
5.2.2	String literals	5-8
5.3	Declaration (dimensioning) of variables	5-9
5.3.1	Structure of the DIM and global statements	5-10
5.3.2	Dimensioning simple numeric variables	5-11
5.3.3	Dimensioning vectors/matrices	5-13
5.3.4	Dimensioning string variables	5-16
5.4	Addressing variables	5-17
5.4.1	Simple numeric variables	5-17
5.4.2	Vectors/matrices	5-17
5.4.3	String variables	5-18
5.5	Hexadecimal constants	5-21
6	BASIC statements	6-1
6.1	Declaration of data fields and functions	6-2
6.1.1	DIM statement	6-5
6.1.2	DEALLO statement	6-5
6.1.3	GLOBAL statement	6-6
6.1.4	DEF statement	6-7
6.2	Data manipulation and arithmetic	6-9
6.2.1	LET statement	6-9
6.2.1.1	Numeric expressions	6-14
6.2.1.2	Combining values of different precision	6-15
6.2.1.3	Calling user-defined functions	6-15
6.2.1.4	Calling numeric BASIC functions	6-16
6.2.1.5	Masks	6-17
6.2.1.6	String expressions	6-21
6.2.1.7	CLEAR statement	6-21
6.2.2	DATA statement	6-22
6.2.3	READ statement	6-23
6.2.4	RESTOR statement	6-24
6.2.5	RANDOM statement	6-25
6.2.6	MAT statement	6-26
6.2.6.1	Assigning a simple matrix variable	6-27
6.2.6.2	Matrix addition and subtraction	6-28
6.2.6.3	Matrix multiplication	6-29
6.2.6.4	Scalar multiplication of a matrix	6-30
6.2.6.5	INV function	6-31
6.2.6.6	MIV function	6-32
6.2.6.7	TRN function	6-33

Contents

6.2.6.6	MIV function	6-32
6.2.6.7	TRN function	6-33
6.2.6.8	ZER function	6-34
6.2.6.9	CON function	6-35
6.2.6.10	IDN function	6-36
6.2.7	MAT READ statement	6-37
6.3	Controlling the program flow	6-38
6.3.1	GOTO statement	6-38
6.3.2	GOSUB statement	6-39
6.3.3	RETURN statement	6-40
6.3.4	ON statement	6-41
6.3.5	IF statement	6-42
6.3.6	IF ERR 0 statement	6-44
6.3.7	IF ESC statement	6-45
6.3.8	FOR statement	6-46
6.3.9	NEXT statement	6-47
6.3.10	CHAIN statement	6-48
6.3.11	LINK statement	6-49
6.3.12	STOP statement	6-51
6.3.13	END statement	6-51
6.4	Input / output statements	6-52
6.4.1	Keyboard input/screen output statements	6-53
6.4.1.1	INPUT statement	6-54
6.4.1.2	MAT INPUT statement	6-56
6.4.1.3	PRINT statement	6-57
6.4.1.4	MAT PRINT statement	6-60
6.4.2	Logical IOCS statements	6-61
6.4.2.1	OPEN # statement	6-61
6.4.2.2	ENV # statement	6-69
6.4.2.3	READ # statement	6-71
6.4.2.4	WRITE # statement	6-73
6.4.2.5	PRINT # statement	6-76
6.4.2.6	MAT READ # statement	6-81
6.4.2.7	MAT WRITE # statement	6-84
6.4.2.8	SEARCH # statement	6-87
6.4.2.9	CLOSE # statement	6-93
6.4.2.10	BUILD # statement	6-94
6.4.2.11	KILL statement	6-97
6.5	Special statements	6-98
6.5.1	Remarks in BASIC programs	6-98
6.5.1.1	REM statement	6-98
6.5.1.2	Remarks in any statements	6-99
6.5.2	Inter-task communication	6-100
6.5.2.1	SIGNAL 1 statement	6-101
6.5.2.2	SIGNAL 2 statement	6-102

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

6.5.3	SIGNAL 3 statement	6-103
6.5.4	CALL statements	6-104
6.5.5	LGERR statement	6-105
7	BASIC functions	7-1
7.1	Mathematical functions	7-2
7.1.1	Trigonometric functions	7-2
7.1.1.1	SIN, COS and TAN functions	7-2
7.1.1.2	ATN function	7-3
7.1.2	Transcendental functions	7-3
7.1.2.1	SQR function	7-3
7.1.2.2	EXP function	7-4
7.1.2.3	LOG function	7-4
7.1.3	Arithmetic functions	7-4
7.1.3.1	ABS function	7-5
7.1.3.2	SGN function	7-5
7.1.3.3	INT function	7-5
7.1.3.4	NOT function	7-6
7.1.3.5	FRA function	7-6
7.1.3.6	RND function	7-7
7.2	Number manipulation functions	7-8
7.2.1	MAN function	7-9
7.2.2	CHR function	7-9
7.2.3	IXR function	7-10
7.3	Special functions	7-11
7.3.1	SPC function	7-11
7.3.2	LEN function	7-13
7.3.3	CHF function	7-13
7.3.4	DET function	7-14
7.3.5	KEY function	7-15
7.3.6	LKY function	7-16
7.3.7	CKY function	7-17
7.3.8	CHN function	7-17
7.3.9	FLN function	7-18
7.3.10	PRN function	7-18
8	CALL statements	8-1
8.1	Table of available subroutines	8-2
8.2	CALL 1	8-5
8.3	CALL 2	8-9
8.4	CALL 3	8-10
8.5	CALL 4	8-11
8.5.1	Output of messages to other ports	8-11
8.5.2	Reading/writing the archive ID and LU number	8-12
8.6	CALL 6	8-15
8.7	CALL 7	8-17
8.8	CALL 20	8-17
8.9	CALL 21	8-20

Contents

	8.10	CALL 22	8-22
	8.11	CALL 23	8-24
	8.12	CALL 24	8-27
	8.13	CALL 25	8-28
	8.14	CALL 26	8-30
	8.15	CALL 28	8-31
	8.16	CALL 29	8-33
	8.17	CALL 31	8-36
	8.18	CALL 32	8-37
	8.19	CALL 35	8-41
*	8.19-1	CALL 43	8-44-1
*	8.19-2	CALL 44	8-44-3
	8.20	CALL 51	8-45
	8.21	CALL 59	8-47
	8.22	CALL 60	8-49
	8.23	CALL 61	8-51
	8.24	CALL 62	8-53
	8.25	CALL 63	8-56
	8.26	CALL 64	8-57
	8.27	CALL 65	8-59
	8.28	CALL 66	8-61
*	8.28-1	CALL 68	8-67
	8.29	CALL 70	8-67-1
	8.30	CALL 72	8-71
	8.31	CALL 73	8-72
	8.32	CALL 74	8-73
	8.33	CALL 80	8-75
	8.34	CALL 84	8-79
	8.35	CALL 90	8-82
	8.36	CALL 91	8-86
	8.37	CALL 93	8-88
	8.38	CALL 96	8-90
	8.39	CALL 97	8-93
	8.40	CALL 98	8-96
	8.41	CALL 99	8-98
	9	Keyboard input/screen output	9-1
	9.1	INPUT statement	9-2
	9.2	MAT INPUT statement	9-3
	9.3	PRINT statement	9-3
	9.4	MAT PRINT statement	9-3
	9.5	Display functions	9-4
	9.5.1	TAB function	9-4
	9.5.2	Backspace function	9-5
	9.5.3	Mark cursor position function	9-6
	9.5.4	Back to marked cursor position function	9-6
	9.5.5	Carriage return function	9-7
	9.5.6	Line deletion function	9-7
	9.5.7	Line insertion function	9-8
	9.5.8	Start background function	9-8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

9.5.9	Start foreground function	9-9
9.5.10	Clear screen function	9-9
9.5.11	Clear foreground function	9-10
9.5.12	Ring keyboard bell function	9-10
9.5.13	Tab to next foreground field function	9-11
9.5.14	Clear foreground field function	9-11
9.5.15	Cursor home function	9-12
9.5.16	Dark function	9-12
9.5.17	Hexadecimal output function	9-13
9.5.18	Duplicate field function	9-13
9.5.19	Duplicate limited characters function	9-14
9.5.20	Duplicate characters function	9-15
9.5.21	Display memory function	9-16
9.5.22	Cursor position function	9-17
9.5.23	Define window function	9-17
9.5.24	No typewriter keyboard function	9-18
9.5.25	Typewriter keyboard function	9-18
9.5.26	Activate 8-bit transfer function	9-19
9.5.27	Deactivate 8-bit transfer function	9-19
9.5.28	Table of available display and device functions	9-20
9.6	Programming release keys and latching keys	9-23
9.6.1	KEY function	9-23
9.6.2	LKY function	9-24
9.6.3	CKY function	9-25
9.7	Input via the OCR A/B reader, BCR, SCR, IDCU	9-26
9.7.1	Input procedure	9-26
9.7.2	Particular features of the individual units	9-28
9.7.2.1	Particular features of the OCR A/B reader	9-28
9.7.2.2	Particular features of the BCR	9-30
9.7.2.3	Particular features of the SCR	9-31
9.7.2.4	Particular features of the IDCU	9-31
9.8	ASCII table	9-34
* 9.9	I/O macro	9-99
* 9.9.1	Calling the I/O macro	9-101
* 9.9.2	Handling of fields	9-105
* 9.9.2.1	Handling of pre-assigned values and inputs	9-105
* 9.9.2.2	Handling of release keys	9-108
* 9.9.3	Field description parameters	9-113
* 9.9.3.1	Column number	9-114
* 9.9.3.2	Line number	9-114
* 9.9.3.3	Skip status	9-114
* 9.9.3.4	Input/output type	9-120
* 9.9.3.5	Maximum input length (no. of digits/bytes)	9-121
* 9.9.3.6	Minimum input length (no. of digits/bytes)	9-122
* 9.9.3.7	Number of decimal places	9-122
* 9.9.3.8	Type of sign	9-122
* 9.9.3.9	Output mask	9-123

Contents

* 9.9.3.10	Allowability of hard keys	9-124
* 9.9.3.11	Allowability of softkeys	9-124
* 9.9.3.12	HELP text definition	9-124
* 9.9.3.13	Minimum input value	9-125
* 9.9.3.14	Maximum input value/allowed characters	9-125
* 9.9.3.15	USING mask	9-126
* 9.9.3.16	Pre-assigned value	9-126
* 9.9.3.17	Attributes	9-127
* 9.9.3.18	Automatic ? function	9-129
* 9.9.4	Field description maintenance utility	9-130
* 9.9.4.1	Creating a field description (CRE)	9-133
* 9.9.4.2	Amending a field description (AME)	9-137
* 9.9.4.3	Deleting a field description (DEL)	9-137
* 9.9.4.4	Reviewing a field description (REV)	9-137
* 9.9.4.5	Listing summarized field descriptions (LST)	9-138
* 9.9.4.6	Printing summarized field descriptions (PRT)	9-139
* 9.9.4.7	Reorganizing the file (REO)	9-139
* 9.9.4.8	Comparing two parameter files (COM)	9-139
* 9.9.5	CALL 9	9-141
* 9.9.6	Example of the I/O macro	9-143
10	Magnetic disks	10-1
10.1	Input/output concept	10-1
10.1.1	File organization	10-1
10.1.2	Record pointers	10-2
10.1.3	Channel concept	10-2
10.1.4	File definitions	10-2
10.1.5	File names	10-3
10.1.6	Concept of logical units	10-4
10.2	Structure of disk files	10-5
10.3	File sharing	10-6
10.4	Creating disk files	10-9
10.5	Accessing disk files	10-10
10.5.1	Opening files	10-10
10.5.2	Accessing files	10-11
10.5.3	Closing files	10-11
10.6	Deleting files	10-12
10.7	CHF function	10-12
11	File processing	11-1
11.1	Types of file organization	11-1
11.2	Formatted files	11-2
11.2.1	Creating formatted files	11-3
11.2.2	Accessing formatted files	11-4
11.3	Contiguous files	11-8
11.3.1	Creating contiguous files	11-8
11.3.2	Accessing contiguous files	11-9
11.3.3	End-of-file (EOF) pointer	11-15

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

11.4	Indexed files	11-16
11.4.1	Creating indexed files	11-17
11.4.2	Accessing indexed files	11-19
11.4.3	Information about indexed files	11-20
11.4.4	Reading the data area sequentially	11-21
11.4.5	Reading the data area in the order of the directory	11-21
11.4.6	Random access via a key	11-22
11.4.7	Inserting a data record with keys	11-22
11.4.8	Deleting a data record with keys	11-23
11.4.9	Deleting an indexed file	11-23
11.4.10	Status messages	11-24
11.4.11	Closing the file	11-24
11.4.12	Problems when processing indexed files	11-25
11.5	Text files	11-28
11.5.1	Creating text files	11-29
11.5.2	Accessing text files	11-30
11.6	Sequential files	11-35
12	Printers	12-1
12.1	Accessing printer files	12-2
12.1.1	Opening printer files	12-3
12.1.2	Output to printer	12-8
12.1.2.1	Forms control	12-9
12.1.2.2	Cut-form processing	12-10
12.1.2.3	Form feed	12-11
12.1.2.4	Paper cassette selection	12-12
12.1.2.5	Elongated printing	12-12
12.1.2.6	Character density	12-13
12.1.3	CHF function	12-16
12.1.4	Closing the printer	12-16
13	Punched card readers	13-1
13.1	Accessing punched card files	13-1
13.1.1	Opening punched card reader files	13-1
13.1.2	Input via the punched card reader	13-2
13.1.3	Closing the punched card reader	13-2
13.2	Example	13-2
14	Paper tape readers/punches	14-1
14.1	Input via the paper tape reader	14-1
14.2	Output via the paper tape punch	14-3
14.3	Code tables for paper tape reader/punch	14-5
15	Magnetic tapes	15-1
15.1	Organization of data on magnetic tapes	15-1
15.2	Accessing magnetic tape files	15-6
15.2.1	Opening the magnetic tape unit with the OPEN # statement	15-6

Contents

15.2.2	READ # statement	15-7
15.2.3	WRITE # statement	15-7
15.2.4	ENV # statement	15-8
15.2.4.1	Transferring code tables	15-8
15.2.4.2	Testing the status	15-9
15.2.4.3	Rewinding	15-10
15.2.4.4	Rewinding and switching off	15-10
15.2.4.5	Writing a tape mark	15-10
15.2.4.6	Forward-spacing by n blocks	15-11
15.2.4.7	Backspacing by n blocks	15-11
15.2.4.8	Forward-spacing by n tape marks	15-12
15.2.4.9	Backspacing by n tape marks	15-12
15.2.5	CLOSE # statement	15-13
15.2.6	Opening the magnetic tape unit with CALL 70	15-13
15.2.7	Processing	15-14
15.2.7.1	Managing the storage medium	15-14
15.2.7.2	Forward-spacing and backspacing the tape	15-15
15.2.7.3	Reading/writing a block	15-16
15.2.8	Closing the tape unit	15-17
15.2.9	Ascertaining the size of the tape buffer	15-17
15.3	Error handling	15-18
15.3.1	Error handling by the tape interface	15-18
15.3.2	Error status codes when using CALL 70	15-19
16	Magnetic tape cassettes	16-1
16.1	Organization of data on tape cassettes	16-1
16.2	Accessing cassette files	16-3
16.2.1	Opening the magnetic tape cassette unit	16-3
16.2.2	Processing	16-4
16.2.2.1	Managing the storage medium	16-4
16.2.2.2	Forward-spacing and backspacing the cassette tape	16-4
16.2.2.3	Reading/writing a block	16-5
16.2.3	Closing the magnetic tape cassette unit	16-5
16.3	Error handling	16-6
17	Program structure	17-1
17.1	Executing a program	17-1
17.2	FOR-NEXT loops	17-3
17.3	Subroutine techniques	17-4
17.4	Subroutines in machine code	17-6
17.5	Segmentation of programs	17-6
17.6	Programming to optimize run-time	17-7
17.6.1	BASIC optimizer	17-7
17.6.2	Arrangement of programs	17-7
17.6.3	Converting old programs	17-9
17.6.4	Other important information	17-12

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Contents

18	Testing aids	18-1
18.1	BASIC error indications	18-1
18.2	IF ERR 0 statement	18-2
18.3	IF ESC statement	18-4
18.4	Direct execution of statements	18-5
18.5	Single-step test run and execution log	18-7
18.6	Program test run	18-9
18.7	STOP statement	18-11
18.8	ESC key	18-12
18.9	Dumping variables	18-13
18.10	Stopping at variables	18-14
19	TAMOS interfaces	19-1
19.1	Preparing programs for inclusion in selectors	19-1
19.2	Starting an application program	19-3
19.3	Displaying the header and message line	19-5
19.4	Terminating an application program	19-6
19.5	Logbook and message files	19-8
19.6	Common area	19-10
19.6.1	Transfer of standard parameters by TAMOS	19-10
19.7	Error handling	19-12
19.8	Job spooler	19-13
19.8.1	Entering a job in the job spool file	19-13
19.9	Parameters for the print spooler	19-19
19.9.1	Forms control	19-19
19.9.2	Spooler control	19-20
19.10	Starting programs in a phantom port	19-21
19.11	Messages from a phantom port	19-22
20	Program maintenance	20-1
20.1	Program listing	20-2
20.2	Unload remarks	20-4
20.3	Load remarks	20-6
20.4	Compare programs	20-8
20.5	Release list	20-11
20.6	Saving a program under another file type (optimization)	20-13
20.7	Program analysis	20-15
21	Programming examples	21-1
21.1	Programming examples for processing indexed files	21-1
21.1.1	Reading a data record directly	21-2
21.1.2	Inserting a key with the associated data record	21-2
21.1.3	Deleting a key and deallocating the associated data record	21-3
21.1.4	Sequential processing from the next higher key onwards	21-3

Contents

21.2	Example of an input routine	21-4
21.3	Copying a contiguous file to magnetic tape	21-7
21.4	Printing a text file	21-9
21.5	Intermediate storage of a printer file in a text file	21-11
21.6	Example of CHF function	21-12
21.7	Reading a punched card file into a contiguous file	21-12
21.8	Use of CALL 97	21-13
21.9	Standard base part for application programs in TAMOS	21-15
21.9.1	Introduction and handling	21-15
21.9.2	Description of the individual routines	21-17
21.9.3	The base part	21-22
22	Error handling	22-1
22.1	BASIC errors	22-1
22.2	Workstation error messages	22-8
23	Index	23-1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

1

Introduction

1 Introduction

1.1 BASIC and Business BASIC

The BASIC programming language (Beginners' All-purpose Symbolic Instruction Code), which was developed in the mid-1960s at Dartmouth College, USA, serves as a foundation for Business BASIC. As its name suggests, this programming language has two significant characteristics:

1. BASIC is easy to learn due to the fact that the language is symbolic and has a clear layout.

2. BASIC is suited for all purposes, i.e. it is a powerful programming language with a wide range of functions.

The following must be stated in order to illustrate the third significant characteristic of BASIC.
If a problem is to be solved by a computer, the various stages required to provide the solution must be input in the form of a sequence of instructions, the program. These instructions, their structure and use are prescribed by the grammar of the programming language. A distinction is made between two types of programming language:

"Machine languages" and "Symbolic languages".

Machine languages are complicated and, in addition, differ from computer to computer, i.e. they are not directly transferrable.

For this reason the majority of programs are written in a symbolic language (machine-independent), and this language is later translated into machine language.
Several machine instructions are normally formed from each instruction of a symbolic language, thus providing a further reason for easier programming in a symbolic language.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Introduction

In most other high-level programming languages (PL/1, ALGOL, COBOL etc.) an editor is used to enter the program in source code and a compiler is used to generate object code. The object code is generally machine code and can be executed directly by the computer. It is not possible to convert this machine code back into source code so users have to keep and manage both forms of the program.

In BASIC the steps leading to an executable program are contained implicitly in various commands. The editor includes some of the features normally associated with conventional compilers such as lexical analysis, syntax checks and the generation of a so-called intermediate code. The BASIC intermediate code is not machine code and has to be executed by a program interpreter. This intermediate code can be converted back to source code by means of commands so users only need one form of the program.

The major advantages of an interpreter are as follows:

- Interactive programming
  A program or parts of the program can be executed, modified, re-executed, deleted or extended immediately after they have been entered. On entry, every statement is checked for lexical and syntax errors.

- Saving of storage space and less file handling.
  Only the intermediate code, which can be interpreted directly, is stored.

The BASIC language was originally intended for use in scientific fields, i.e. priority was given to solving mathematical problems as opposed to the input/output and manipulation of large amounts of data. The numerous computing facilities as well as the simple language grammar were retained when BASIC was further developed to its more commercial form, Business BASIC.
Its ability to support a great number of peripherals has also enabled it to be used in the commercial sector. These peripherals are, in essence, magnetic disks but magnetic tapes, magnetic tape cassettes, punched card readers, floppy disks, needle and high-speed printers, data communication equipment, paper tape readers, tape punches and plotters are all supported as well.

	Introduction

1.2 Hints on using the manual

This manual is intended for use in the following ways:

1. As accompanying documentation for participants in an introductory course on BASIC.

2. As self-training matter for experienced programmers wishing to learn BASIC on their own.

3. As a reference work for experienced BASIC programmers. In this respect, the index at the back of the manual is helpful.

1.3 BASIC programming system

A BASIC program is processed by means of the BASIC processor in the BASIC command mode which can be reached from TAMOS via the EXPERT selector. The BASIC processor is a system program which permits interactive programming supported by various commands. The BASIC command mode is indicated by the fact that the cursor is located at the beginning of a display line. The programmer is expected to input a command or statement (interactive mode). The term "input" in this document covers the actual keying in of characters and their acknowledgement with the CR key (called the ENTER key on some keyboards).

It is possible to exit the BASIC processor by simultaneously pressing the CTL and C keys. If the program has not been saved by means of the the SAVE command after it has been entered or modified, the warning SAVE?! is output and it is not possible to exit the processor until the program has been saved or CTL and C have been pressed again.

With the aid of the syntax analyser - a component of the BASIC processor - the BASIC statements are checked for correct syntax during the editing phase and any errors that are detected are indicated on the display. A statement is displayed with a length of up to 251 characters and the cursor is positioned at the location of the error.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Introduction

1.3.1 Creating a new BASIC program

The program in the memory partition in question can be erased by entering the NEW command and a new program can then be input. After the new program has been input, it should be saved by entering the SAVE command if it is intended to be used again later! The SAVE command causes the new program to be stored in a program library on disk.

1.3.2 Modifying a BASIC program

An existing program must be modified if errors are detected during testing or if it is due for maintenance. First it is loaded into the memory partition by calling it via the EXPERT selector. The program name and logical unit (LU) number must be specified.

Subsequently, statements can be deleted, overwritten, added or be listed with the EDIT command and modified. Finally, do not forget to save the new program version with the SAVE command.

1.3.3 Testing a BASIC program

Before processing of a BASIC program is concluded, there is generally a test to see if it meets requirements.

Various testing aids are available for this purpose and to detect errors. These are described in the "Testing Aids" chapter of this manual.

1.3.4 Running a BASIC program

A BASIC program is made to run by calling it via the EXPERT selector (specifying its program name) or from the command mode by entering the RUN command if the desired program is already in main memory.

Introduction

1.3.5 Storing BASIC programs on disk

1.3.5.1 Selecting the appropriate intermediate code

As described above, it is sufficient to have just the one intermediate code in BASIC and execute programs. This intermediate code is a so-called B file (BASIC program of file type B) and is generated immediately after the source code has been edited and stored with the SAVE command. The BASIC optimizer can be used to generate a better intermediate code. The optimizer can be called with the SAVE command and results in the generation of a D file (BASIC program of file type D).

Attention:

Only D files should be used in future as the programs are executed much faster than when they are in B files.

1.3.5.2 Dumping and loading source code

It is possible to regenerate source code from intermediate code by means of the DUMP command and regenerate intermediate code from source code by means of the LOAD command.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Syntax Elements

2 Syntax elements

This chapter introduces the elements of the Business BASIC language which are required for the formulation of valid statements.
The rules required to form these elements are specified with the aid of a metalanguage.

2.1 Description of the metalanguage

A metalanguage which closely resembles the Backus Naur Form (BNF) will be used in the following to describe the scope of the language.

The symbols of this language and their meaning are:

< > (Opening/closing angle brackets)

The name of a class of language elements in abbreviated form is enclosed in angle brackets, e.g. <NExpr> as an abbreviation for "numeric expression".

If it is necessary to draw a distinction between identical classes of language elements (e.g. numeric expressions) in the syntax description, this is done by indexing the abbreviations in angle brackets (e.g. <NExpr1>).

::= (...is defined as...)

The language element to the left of this definition symbol is defined by the syntactical expression on the right of it.

| (either...or...)

The language elements located before and after this symbol preclude one another; e.g. <Le>|<Di> means "either a letter or a digit" must be specified here.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Syntax Elements

{ } (Opening/closing braces)

Language elements enclosed in braces must appear at this point (mandatory specification).

[ ] (Opening/closing square brackets)

Language elements enclosed in square brackets can appear at this point, i.e. they can also be omitted here (optional specification).

The following applies to braces and square brackets:

Specifications located below one another exclude one another.
Example:
```
... <Le> ...
    <Di>
```

means that either a letter or a digit must be written at this point if braces are indicated.
(Accordingly, when square brackets are indicated, this means "can" not "must".)

$\{\}_2^8$ $[]^4$ (Minimum/maximum multiple specification)

A superscript index to the upper right of a closing brace or square bracket defines the maximum number of times the mandatory or optional specification can occur in succession at this point.

A subscript index to the right of a closing brace defines the minimum number of times the mandatory specification must occur in succession at this point.

If no index is specified, this means the same as specification "1".
The specification "n" as a superscript index means "as often as required" ("n times").

Example: ... $\{$<Di>$\}_1^4$ ...

This means that at least one digit must be written at this point but a maximum of four may be written here.

...[<Le>]^4...

This means that letters need not necessarily be written at this point (optional specification); if they are, however, then a maximum of four may be written.

All other characters outside the angle brackets are end symbols and must always be written.

Note:

If one line is not sufficient for representing a syntax element, it is continued - indented - on the next line.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Syntax Elements

2.2 List of syntax elements

- <Le> ::= A|B|C .... X|Y|Z

  <Le> is the set of letters in the alphabet.

- <Di> ::= 0|1 ... 8|9

  <Di> is the set of decimal digits.

- <NOper> ::= +|-|*|/|↑

  <NOper> are operators for addition (+), subtraction (-), multiplication (*), division (/) and exponentiation (↑).

- <RelOper> ::= <|>|=|>=|<=|<>

  The relational operators mean (from left to right): "less than", "greater than", "equal to " "greater than or equal to", "less than or equal to" and "unequal to".

- <Format> ::= 1%|2%|3%|4%

  <Format> specifies the storage space for variables to be declared.

- <Integer> ::= $[+|-]\{<Di>\}^{n}$

  An integer is a sequence of decimal digits, possibly with a sign.

  Examples: 9 -123456789012345 0815

Syntax Elements

- <DecNo> ::= $[+|-]\left\{\begin{matrix}\{<Di>\}^{n} \\ .\{<Di>\}^{n}\end{matrix}\right.\quad\left.[.\ [\ \{<Di>\}^{n}\ ]]\right\}\quad[E[+|-]\ \{<Di>\}_{1}^{2}\ ]$

  A decimal number is either an integer or a sequence of decimal digits preceded or followed by a point, or within which a decimal point is located at any fixed place.
  Furthermore, by appending the letter E and a one- or two-digit integer, it is possible to multiply the value of the number before the E by 10 to the power of the integer after the E.

  Examples: 99 1. .6789E-5 0.05

- <LNo> ::= $\{<Di>\}^{5}$

  A line number is a number which comprises 1 to 5 decimal digits. Statement numbers from 1 to 65535 are possible.

- <NVarName> ::= <Le>|<Le><Di>
  <SVarName> ::= <NVarName)$
  <VarName> ::= <NVarName>|<SVarName>

  A numeric variable name consists of a letter or a letter followed by a digit. A string variable name consists of a numeric variable name followed by a $ sign. A variable name is a numeric or string variable name.

  Examples: A0 P Z9 E$ E1$

- <ProgName> ::= $[<LU>/]$ <Le> $[<Le>|<Di>|.|-]^{13}$

  A BASIC program is identified by its name and the number of the logical disk unit (LU) on which it is stored. LU 0 does not need to be specified. A program name must begin with a letter followed by up to 13 letters, digits, periods or hyphens.

- <FileName> ::= <ProgName>

  <FileName> is syntactically identical to a program name but signifies a data file.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Syntax Elements

- <DriverName> ::= $ [<Le>|<Di>|.|-][13]

  A driver file is defined by the driver name which must begin with a $ sign followed by up to 13 letters, digits, periods or hyphens.

- <File> ::= <ProgName> [!] / <DriverName>

  A file can be either a disk or driver file. If an existing disk file has to be replaced, it is necessary to specify "!".

  Examples: CL1 TEST-S.O $LPT

- <NVar> ::= <NVarName> [(<NExpr> [,<NExpr>])]

  Numeric variables represent the decimal values of numbers. They comprise a numeric variable name which can be followed by one or two arithmetic expressions (indexed expressions) separated by a comma and enclosed in parentheses.

  Examples: AO P(1) Z9(15,0) F(A*A,1+Y)

- <SVar> ::= <SVarName> [(<NExpr> [, <NExpr>])]

  String variables represent stings. Their syntax is almost identical to that of arithmetic variables; in order to identify string variables, however, a string variable name must be specified. The indexed expressions specify positions within the string defined by <SVarName>.

  Examples: AO$ P$(1) Z9$(2,4) F$(X,X+10)

- <MVar> ::= <Le>|<Le><Di>

  Matrix variables have the same syntax as that of variable names and they designate matrices in their entirety. The matrix variable must be identified as a matrix by means of dimensioning prior to or on its first occurrence.

Syntax Elements

- <NExpr> ::= [+|-]<Operand>[<NOper><Operand>]^{n}

  A numeric expression specifies the rule to be used to calculate a numeric value and is formed by connecting operands by means of arithmetic operators. Only the first operand can have a sign (+ or -) without requiring parentheses.

  Meaning:

  <Operand> ::= { FN<Le>(<NExpr>) | (<NExpr>) | <NVar> | <NFct> | <DecNo> }

  This means that the following can occur as operands:

  - User-defined functions
  - Numeric expressions (enclosed in parentheses)
  - Numeric variables
  - Numeric functions
  - Decimal numbers

- <NFct>

  Numeric functions always return a numeric value.
  The chapter entitled "BASIC Functions" describes this in greater detail.

- <NumPar> ::= <NVar>|<DecNo>

  Numeric parameters are primarily used as the parameters in CALL statements.

- <Char>

  <Char> is the entire set of valid display characters.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Syntax Elements

- <OctNo>

  <OctNo> is a 3-digit octal number ranging from 000 to 377. Octal codes represent the ASCII code of the characters in the character set.

- <SLit> ::= "[<Char> except " and ← or ← <OctNo> ←]"

  A string literal is a defined string enclosed in double quotation marks and containing the display characters except the double quotation mark and left arrow. A double quotation mark in a string literal is represented by two successive single quotation marks (''). A left arrow can only be entered as an octal number.

- <String> ::= <SVar>|<SLit>

  <String> denotes a string variable or string literal.

- <SExpr> ::= <String>[,<String>]^{n}

  A string expression comprises a sequence of string constants and string variables separated by commas.

- <LogExpr> ::=

  $$\left\{\begin{array}{l}\text{<NExpr> [<RelOper><NExpr>]} \\ \text{<String> [,<String>][<RelOper><String>[,<String>]]}\end{array}\right\}$$

  The result of the evaluation of a logical expression is true or false, in accordance with the rules of logic.

- <DisFct>

  Display functions are the functions required for preparing the display and positioning the cursor, as described in the chapter entitled "Keyboard Input/Screen Output".

- <DevFct>

  Device functions correspond syntactically to display functions (<DisFct>) and are used for programming drivers to control peripheral devices.

- <ConChar>

  Control characters in data communication statements to specify the job. The control characters correspond syntactically to display functions (<DisFct>) except that, instead of a 2-character literal, a 3-character literal is enclosed in single quotation marks (refer to Nixdorf 8870 Data Communication programming manual).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

3

Summary of BASIC Statements and Commands

3 Summary of BASIC statements and commands

3.1 Summary of BASIC statements

BUILD # <NExpr>, [+] <String> [[, # <NExpr>],[+] <String>]^n

Creating magnetic disk files

CALL <NExpr> { , <NumPar> / , <String> }^12

Calling machine code subroutines

CHAIN <String>

Calling any program or processor

CLEAR <VarName> [,<VarName>]^n

Clearing the values of variables

CLOSE # <NExpr>[, # <NExpr>]^n

Closing data channels (terminating the data communication connection; see Nixdorf 8870 Data Communication programming manual)

CNTRL # <NExpr>; { <ConChar> / <ConChar>,<ConChar> } ,<SVar>,<NVar>

Controlling the editing of data and receive enable (DNT) or controlling the flow of data (SNA); see Nixdorf 8870 Data Communication programming manual

---

DATA [<Format>,] <DecNo> [,<DecNo>]^n

Storing numeric constants

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Summary of BASIC Statements and Commands

DEALLO <VarName> [, <VarName>]^{n}

Deallocating variables and fields

DEF FN <Le> (<Le>)=<NExpr>

Defining user functions

DIM <FieldDescr> [,<FieldDescr>]^{n}

where

<FieldDescr>::= { <NVarName> [(<NExpr>[,<NExpr>])] | <SVarName>(NExpr>) | <Format> }

Dimensioning variables and fields

END

Terminating a program

ENV # <NExpr>;<NVar> [,<NVar> | ,<SVar>]^{n}

Transferring parameters to OPEN #, READ # or WRITE # statement (also see Nixdorf 8870 Data Communication manual)

FOR <NVar>=<NExpr> TO <NExpr> [STEP <NExpr>]

Starting a program loop

GET # <NExpr>;[<ConChar>,] <SVar>,<SVar>,<NVar>,<NVar>,<NVar>

Receiving data and status information (see Nixdorf 8870 Data Communication programming manual

---

3

Summary of BASIC Statements and Commands

GLOBAL	<FieldDescr> [,<FieldDescr>]^{n} where: <FieldDescr>::= {<NVarName>[(<Integer>[,<Integer>])] <SVarName (<Integer>) <Format>} Dimensioning global variables
GOSUB	<LNo> Branch to subroutine
GOTO	<LNo> Unconditional branch
IF	<LogExpr> <Statement> Conditional execution of <Statement>
IF ERR 0	[<Statement>] Execution of <Statement> in event of error
IF ESC	[<Statement>] Execution of <Statement> when the ESCAPE key is pressed
INIT #	<NExpr>;<SVar>,<SVar>,<NVar> Establishing the physical connection to the required partner using the partner ID (see Nixdorf 8870 Data Communication programming manual)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

	Summary of BASIC Statements and Commands
INPUT	[<SLit>, / <DisFct>, / TAB(<NExpr>[,<NExpr>]), / <NVar>, / <SVar>,]^{n} { <NVar> / <SVar> } Input via keyboard
KILL	<String> [, <String>]^{n} Deleting files
[LET]	{ <NVar>= { <NExpr> / <String> } / <SVar>= { <SExpr> / <NExpr> [USING <String>] } } Assigning a variable (LET must be written in full in an IF statement)
LINK	[;<VarName>,] <String> Calling a BASIC program
LGERR	Recording BASIC errors in the system logbook
LOFF #	<NExpr>;<SVar>,<SVar>,<NVar> Clearing down the logical connection (see Nixdorf 8870 Data Communication programming manual)
LON #	<NExpr>;<SVar>,<SVar>,<NVar> Establishing the logical connection (see Nixdorf 8870 Data Communication programming manual)

Summary of BASIC Statements and Commands

3

MAT <MVar>= { <MVar> | <MVar>{+,-,*}<MVar> | (<NExpr>)*<MVar> | INV(<MVar>) | MIV(<MVar>) | TRN(<MVar>) | ZER[(<NExpr>[,<NExpr>])] | CON[(<NExpr>[,<NExpr>])] | IDN[(<NExpr>,<NExpr>)] }

Matrix allocation

MAT INPUT <MVar>[(<NExpr>[,<NExpr>])]
[,<MVar>[(<NExpr>[,<NExpr>])]]ⁿ

Keyboard input into a matrix

MAT PRINT <MVar> [, | ;]

Displaying a matrix on the screen

MAT READ <MVar>[(<NExpr>[,<NExpr>])]
[,<MVar>[(<NExpr>[,<NExpr>])]]ⁿ

Reading a constant to a matrix

MAT READ #<NExpr>[,<NExpr>[,<NExpr>]]; <VarName> [;]

Reading from a file to a variable

MAT WRITE #<NExpr>[,<NExpr>[,<NExpr>]]; <VarName> [;]

Transfer from a variable to a file

NEXT <NVar>

End of a program loop

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Summary of BASIC Statements and Commands

ON <NExpr> {GOTO | GOSUB} <LNo>[,<LNo>]ⁿ

Computed branch

OPEN # <NExpr>[;<String>], <String>
[[, #<NExpr> [;<String>]], <String>]ⁿ

Opening data channels (opening a DNT or SNA driver, see Nixdorf 8870 Data Communication programming manual)

{PRINT | ;} [USING <String>;] [<NExpr> | <String> | <DisFct>][{;|,} {<NExpr> | <String> | <DisFct>}]ⁿ [;|,]

Output to display (PRINT must be written in full in an IF statement)

{PRINT | ;} # <NExpr> [, <NExpr>[, <NExpr>]] ;

[USING <String>;] [<NExpr> | <String> | TAB(<NExpr>) | <DevFct>] [{;|,} {<NExpr> | <String> | TAB(<NExpr>) | <DevFct>}]ⁿ [;|,]

Output to magnetic disk files or drivers (PRINT must be written in full in an IF statement)

PUT # <NExpr>;[<ConChar>,] [<ConChar>,] <SVar>,<SVar>,<NVar>

Transmitting data and status information (see Nixdorf 8870 Data Communication programming manual)

RANDOM <NExpr>

Controlling the RND function

READ <NVar>[,<NVar>]ⁿ

Reading constants

3

Summary of BASIC Statements and Commands

READ # <NExpr>[,<NExpr>[,<NExpr>]];{<NVar>|<SVar>}[,<NVar>|,<SVar>]^n [;]

Reading from magnetic disk files and drivers

REM [<Char>]^n

Writing remarks in the program (it is also possible to write remarks after any statement by using the /* combination)

RESTOR [<LNo>]

Resetting the constant pointer

RETURN [<NExpr>]

Returning from subroutines

SEARCH # <NExpr>,<NExpr>,<NExpr>;<SVar>,<NVar>,<NVar>[: | ;]

Processing directories in indexed files

SIGNAL {1,<NExpr>,<NExpr>,<NExpr> | 2,<NVar>,<NVar>,<NVar>[,<NExpr>] | 3,<NExpr>}

Processing the signal node queue and wait statement

STOP

Program interrupt

WRITE # <NExpr>[,<NExpr>[,<NExpr>]];{<NExpr>|<String>}[,<NExpr>|,<String>]^n

Writing to magnetic disk files or drivers

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Summary of BASIC Statements and Commands

3.2 Summary of BASIC commands

[<LNo>] ANALYSE <File> [<LNo>] [↑<Integer>]

Listing the program marked for optimization on the printer or in a text file

---

[<LNo>] AUTOLN[<Integer>]

Automatic line numbering

[<LNo>] COMMENT [<ProgName> [!]] [<LNo>] [REM [<LU>]]

Loading remarks into a program

CREF { <LNo> | <VarName> | #<Integer> | #? | STAT }

Displaying cross-references

[<LNo>] DEBUG [<File>]

Testing the statement sequence in single-step mode or listing executed statements on printer or in a text file

---

{ <LNo> | [<LNo>] DELETE [<LNo>] }

Deleting statements

[<LNo>] DUMP <File> [<LNo>] [↑<Integer>]

Listing the program on printer or in a text file

[<LNo>] EDIT

Displaying a statement

Summary of BASIC Statements and Commands

HASH [<LNo>]

Calculating and displaying the hash total

HELP [<Integer>]

Displaying a BASIC error in plain language

[<LNo>] LIST [<LNo>]

Listing statements on the display

LOAD <ProgName>

Loading a program from a text file

NEW

Erasing the current BASIC program in the partition

<LNo> PARTRENUM <LNo>

Partly renumbering the statements in a program

[<LNo>] PLIST <File> [<LNo>] [,<Integer>] [REM [<LU>]]

Listing the program with cross-references on printer or in a text file

---

[<LNo>] RENUMBER [<Integer>]

Renumbering the statements in a program

[<LNo>] RUN [<LNo> [,<LNo>]^{4}]

Starting the current program

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Summary of BASIC Statements and Commands

SAVE [,D,|,B,][#<PartS>#] [<ProgName> [!]]

Saving the program in memory to disk (<PartS>: partition size in KB)

SIZE

Displaying the program size

[<LNo1>] TRACE [<LNo2>] { {,<VarName>}^5 | {,<Condition>}^2 }

TRACE [<File>]

where:

<Condition>::= { <NVarName1><RelOper> {<NVarName2> | <DecNo>} | <NVarName>{=|<>}<DecNo1>:<DecNo2> | <SVarName1><RelOper> {<SVarName2> | <SLit>} }

Stopping the program at specified variable values. Output of all variable values.

[<LNo>] UNCOMMENT [<ProgName> [!]] [<LNo>] [REM [<LU>]]

Unloading remarks from a program

BASIC Commands

4 BASIC commands

The BASIC processor provides the following BASIC commands to aid programming in Business BASIC:

- ANALYSE
- AUTOLN
- COMMENT
- CREF
- DEBUG
- DELETE
- DUMP
- EDIT
- HASH
- HELP
- LIST
- LOAD
- NEW
- PARTRENUM
- PLIST
- RENUMBER
- RUN
- SAVE
- SIZE
- TRACE
- UNCOMMENT

These commands are called from the BASIC command mode which is indicated by the position of the cursor at the beginning of a line.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

4.1 ANALYSE command

The ANALYSE command supports the user in converting programs in preparation for optimization. It analyses programs statically and marks any code that is unfavourable for optimization (compare SAVE, D,...).

Syntax:

[<LNo1>] ANALYSE <File> [<LNo2>] [↑<Integer>]

<LNo1> : Line number of the first statement to be analysed. If this specification is omitted, output starts at the first statement in the program.

<File> : Name of the text file to receive the output or the name of the printer on which the program is to be printed out.

<LNo2> : Line number of the last statement to be analysed. If this specification is omitted, output goes up to the end of the program.

↑<Integer> : Lines per page (24-99). If this specification is omitted, the standard page length of 48 lines is used.

The program analysis outputs a program listing without remarks. The referenced line numbers are not indicated in the listing; nor is any statement cross-reference list output. Instead the following marks are appended to the program listing.

DA All DEALLO statements which are not located directly before LINK in the program (deallocate).

These variables are retained for use under the same name in the successor program.

LC All DEALLO statements which are before LINK in the program (local).

Such variables have to be defined as local variables when a program is converted.

Furthermore the variable cross-reference list is output with the following marks. If several of them apply, they are set one after another.

FP All explicitly dimensioned variables which are not at the start of the program (false position).

VA All DIMensioned variables which do not have an integer as the field description or which are in a DIM statement together with such variables (variable argument).

DA All variables in DEALLO statements which are not located directly before LINK (deallocate).

LC All variables in DEALLO statements which are before LINK (local).

- Example:

  ANALYSE $LPT ↑ 72

  generates an analysis listing of the program which you are in. The list is output to the printer $LPT using 72 lines per page.

The function of the command can be selected for one or more programs via the TAMOS selector (see "Program Maintenance" chapter).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

4.2 AUTOLN command

The AUTOLN command causes line numbers to be generated automatically on the display.

Syntax:

[<LNo>] AUTOLN [<Integer>]

<LNo> : Start of automatic line numbering at this line number

<Integer> : Step

If the <LNo> specification is omitted, automatic line numbering starts at 10.
If the <Integer> specification is omitted, a step of 10 is used.

4.3 COMMENT command

The COMMENT command reloads remarks into a program that had been unloaded with the UNCOMMENT command.

Syntax:

[<LNo1>] COMMENT [<ProgName> [!]]
[<LNo2>] [REM [<LU2>]]

<LNo1> : Line number of the first statement from which remarks are to be reloaded. If this specification is omitted, loading starts from the beginning of the program.

<ProgName> : Remarks are loaded into the program in the partition and this program is saved under the name <ProgName>. The actual name can be up to 12 characters long as the remarks are taken from a text file of the same name which is prefixed by the Y. character combination. If <ProgName> is not specified, the name of the program in the partition is used.

! : An "!" is necessary if a program already exists under the name <ProgName> and is intended to be overwritten.

BASIC Commands

<LNo2> : Line number of the last statement up to which remarks have to be loaded. If this specification is omitted, loading goes up to the end of the program.

REM [<LU2>]: Number of the logical unit on which the text file containing the remarks is stored. If <LU2> equals "0", this specification can be omitted.

A text file that has been generated by means of UNCOMMENT must not be loaded back into the reduced program with the aid of the LOAD command as this could result in the overwriting of program lines.

If a reduced program is modified, this may by mistake involve the use of line numbers which were used for unloaded remarks. If the remarks are reloaded later with the COMMENT command, such remarks are appended to the modified lines in the form of /* remarks instead of REMs.

Attention:

The COMMENT command always causes the text file containing the remarks to be deleted, even if not all remarks are loaded because <LNo1> and/or <LNo2> was specified. If there is a risk of deleting remarks by mistake, a copy of the text file should be made before using the COMMENT command.

The function of the command can also be called for one or more programs via the TAMOS selector (cf. the "Program Maintenance" chapter).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

4.4 CREF command

The CREF command causes the cross-references associated with a statement number, variable or channel number to be displayed. If STAT is specified, the program statistics are output.

Syntax:

```
        { <LNo>      }
        { <VarName>  }
CREF    { #<Integer> }
        { #?         }
        { STAT       }
```

Indexation is not allowed for <VarName> as it is irrelevant for cross-references.
If #? is specified, all channel numbers are displayed which have been defined by variables or numeric expressions.

4.5 DEBUG command

The DEBUG command enables the programmer to test any sequence of statements in single-step mode.

Syntax:

[<LNo>] DEBUG [<File>]

<LNo> : Line number of the statement at which the test run is to commence.
If <LNo> is not specified, DEBUG starts with the first statement of the program.

<File> : Specification of a printer or a text file as an output medium. The STOP AT message is displayed if this specification is omitted.

The <LNo> specification is only expedient if an initial RUN or DEBUG command has already been executed.

DEBUG without specification of <File> initializes the single-step test (DEBUG mode), and displays the line number of the next statement and the statement itself in the form:

STOP AT xxx <Statement>

The BASIC processor remains in the DEBUG mode and the operator can initiate execution of the displayed statement and output of the next statement by pressing the CR key.

Input of a command other than DEBUG or TRACE or a BASIC statement with a line number cancels the DEBUG mode. Direct execution of statements or pressing the ESC key do not cancel the DEBUG mode. Specifying <FileName> in the DEBUG command causes the program to be executed (without stops unlike the single-step test) and the executed statements are output to the specified file or printer.

- Example:

  300 DEBUG

  Starting the DEBUG mode at line number 300 and displaying this statement. The statement is executed when the CR key is pressed and the next statement is displayed.

ERROR # 6 "NO SUCH LINE NUMBER"

is displayed when the line number specified in the DEBUG command does not exist.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

4.6 DELETE command

The DELETE command enables any sequence of statements to be deleted from a BASIC program.

Syntax:

```
{ <LNo>                                  }
{ [<LNo1>]  DELETE  [<LNo2>]             }
```

<LNo1> : Line number of the first statement to be deleted.
If this specification is omitted, deletion takes place from the first statement onwards.

<LNo2> : Line number of the last statement to be deleted.
If the specification <LNo2> is omitted, deletion takes place up to and including the last statement.

If only DELETE is specified, the entire program is erased from the main memory partition. In this form, it creates the same state as the NEW command.

If only <LNo> is specified (without DELETE), only this one statement is deleted.

When large program parts are deleted, execution can take several seconds. The end of the function is indicated by the fact that the cursor is positioned at the start of the next screen line.

- Examples:

  - 490

    Deleting the statement with the line number 490.

  - 100 DELETE 200

    Deleting the statements from 100 to 200 (inclusive).

4.7 DUMP command

The DUMP command dumps the BASIC program in the main memory partition to a text file on disk or to a printer.

Syntax:

[<LNo1>] DUMP <File> [<LNo2>] [↑<Integer>]

<LNo1> : Line number of the first statement to be dumped. If this specification is omitted, the dump commences at the first statement in the program.

<File> : The name of the text file to be used for the dump or the name of the printer on which the program is to be printed.

<LNo2> : Number of the last statement to be dumped.

<Integer> : Number of lines per page.
Note that the first 3 lines are always blank. A default value of 48 lines per page is used if this specification is omitted.

If "↑" and the number of lines per page are specified, the dump is carried out with page separation. If this specification is missing, the dump is without page separation. This applies both for printouts and for dumps to a text file.

Page separation is done by storing the octal code "←214←" (top of form) in the case of a text file.

The top of form code is output by the BASIC processor as soon as the specified page length minus three lines has been reached.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Example:

  500 DUMP $LPT ↑48

  Printing out the program from the statement with line number 500 on the needle printer with a page length of 48 lines and a change of page after 45 lines.

Error messages which appear in the message line are described in the "Error Messages" manual.

4.8 EDIT command

The EDIT command causes a single BASIC statement, which is in memory, to be displayed as a keyboard pre-input so that it can be manipulated directly. Up to 251 characters are displayed. The cursor is located at the beginning of the screen line.

Syntax:

[<LNo>] EDIT

If there is no statement with the specified line number, the statement with the next higher line number is duplicated.

If a statement's line number is changed, the statement exists under the old and under the new line number.

A maximum of 80 characters are transferred to the pre-input buffer on loadable workstations with the old design (RAP).

4.9 HASH command

This command causes the hash total of the BASIC program in the partition to be calculated without using a text file. The hash total is identical to the one which the LIBR processor calculates using a text file.

Syntax:

HASH [<LNo>]

The hash total is calculated up to and including the statement with the specified line number. All statements up to 49999 are included in the calculation if the line number is omitted.

The result is stored right-justified in ASCII in bytes 4 to 10 of the port's common area and is displayed in decimal notation.

The hash total for a B file is the same as for a D file.

The following must be noted:

BASIC sources in the form entered differ from sources which have been optimized and then deoptimized (entry of a B file - SAVE,D,... - DUMP ...). This means that the hash total of the first entered version can differ from sources which result from deoptimization. Therefore a SAVE,D... is required first to make sure that the hash totals match. Of course these differences in the source program have no effect on semantics.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

4.10 HELP command

The text pertaining to the error number specified (ERROR #) or the last error number which occurred is output with the HELP command.

Syntax:

HELP [<Integer>]

<Integer> : Error number.

If an error number is specified, the text which corresponds to this number appears.
If no error number is specified, the text pertaining to the error number which last occurred appears.

- Example:

  HELP 1

  Output: SYNTAX ERROR

If the number entered has not been allocated, the following message is issued:

NO SUCH MESSAGE NUMBER

4.11 LIST command

The LIST command enables the BASIC program in the main memory partition of the calling port to be displayed in ascending order.

Syntax:

[<LNo1>] LIST [<LNo2>]

<LNo1> : Line number of the first statement to be output. If this specification is omitted, output commences with the first statement in the program.

<LNo2> : Line number of the last statement to be output. If this specification is omitted, output will continue up to the very last statement.

If both specifications <LNo1> and <LNo2> are omitted, the entire program is displayed.

The output to display can be terminated by pressing the ESC key. The program is listed on the display in roll-up mode. Rolling can be interrupted by pressing the space bar and continued by pressing the CR key.

- Example:

  100 LIST

  Displaying the program commencing at the statement with the line number 100 to the end of the program or until terminated by pressing the ESC key.

4.12 LOAD command

The LOAD command loads a BASIC program or part of a BASIC program, which was stored in a text file with the DUMP command, into main memory.

Syntax:

LOAD <ProgName>

<ProgName> : Name of the text file from which the program is to be loaded.

If, at the time of execution, another program is in the main memory partition, the programs are merged in accordance with their line numbers. If identical line numbers occur in both program parts, the numbers concerned and the associated statements of the program which is already in memory are replaced by those of the text file.

The LOADed program is translated into the same code (file type B or D) as the program in the partition. If no program is in the partition, a program of type B is generated.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

This facility of merging program parts in the memory with program parts in the text file makes it easier to include standard routines in application programs.
Format checks are made during execution of the LOAD command.
If errors are detected, the corresponding error number is displayed.
The routine, however, continues to run but does not load the incorrect statements into main memory.

4.13 NEW command

The NEW command erases and initializes the main memory partition of the port in question.

Syntax:

```
NEW
```

Warning:

Every input in the BASIC command mode which starts with NEW erases the port's memory partition.

4.14 PARTRENUM command

The PARTRENUM command enables line numbers to be partly renumbered. It is used when it is no longer possible to insert a statement between two statements.

Syntax:

```
<LNo1>  PARTRENUM <LNo2>
```

<LNo1> : Line number at which partial renumbering is to start.

<LNo2> : Line number at which partial renumbering is to finish.

The step cannot be specified for partial line renumbering. This is calculated automatically in accordance with the following formula.

Step = (<LNo2> - <LNo1>) / (No. of lines - 1)

The calculated value is rounded off to an integer. The result is the maximum constant line number step of the statements between the two line numbers which were specified.

4.15 PLIST command

This command causes a program listing with cross-references and statistics to be output to printer or to a text file.

Syntax:

[<LNo1>] PLIST <File>

[<LNo2>] [↑<Integer>] [REM [<LU>]]

<LNo1> : Line number of the first statement to be output. The output starts at the first statement in the program if this specification is omitted.

<File> : The name of the text file for the output or the name of the printer on which the program is to be printed.

<LNo2> : Line number of the last statement to be output. The output finishes with the last statement in the program if this specification is omitted.

<Integer> : Number of lines per page (24-99).
A default value of 48 lines per page is used if this specification is omitted.

REM [<LU>] : The number of the logical disk unit on which the text file containing the remarks is located. If <LU> equals 0, this specification can be omitted.

If REM is not specified and the current BASIC program has already been saved (i.e. the program name already exists), LU 0 is scanned for a text file with the associated remarks.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

When the program text is output, remarks which have been unloaded are indented to column 81 as standard and may use up several lines in certain circumstances.

Since this form of output is not always desirable, it can be suppressed for remarks which have been specified in a REM statement. The remark is input in the form REM @[<Char>]^n and the output then starts in column 1.

Special REM statements can be used to define line and page feeds for the printout:

REM↑ [<Char>]^n = Line feed and remark
REM↑↑ [<Char>]^n = Page feed and remark

If an error occurs when the PLIST command is executed (e.g. paper runs out), an error message is output and the user can choose between restarting and terminating the output.

Explanation of the printout:

Three header lines containing the following information appear on each page:

Line 1 : Date, program name, programmer and page number
Line 2 : Branch (= application area), release level, from master, to master
(These are read from the file header.)
Line 3 : LINE, BASIC STATEMENT, LINE and COMMENT

List of statements

The statements are listed together with their line numbers. Remarks are listed separately on the right of the printout. The specification "X=>" before a line number indicates that this number is referenced in the program. "X" is the number of references.

BASIC Commands

Variable cross-references

The following details are output in a table:

- Variable name (VAR)
- Variable size (SIZE) for strings : length in bytes
  for vectors : number of elements
  for matrices : number of rows, columns

  "******" is output under SIZE for single indices greater then 999, run-time dimensioning and complex expressions. The exact dimensioning can be seen in the DIM statement under the following LINE heading (dimensions of variables).

  The following is marked specially when the size of variables is specified: multiply dimensioned variables (MULTDIM), implicitly dimensioned variables (IM DIM) and non-dimensioned variables (NO DIM = strings which are dimensioned in a calling program and which are passed on to the program in question via LINK).

- Line number of dimensioning (LINE)
- Variable type (TYPE) for strings : $
  for numeric variables : 1% - 4%

  This is ascertained statically from the source program so variances are possible at interpretation time (refer to DATA AREA in the program statistics).
  Global variables are also marked G in the specification of the variable type.

- Line numbers of statements which reference these variables. The line numbers may be marked M or R.
  -- M In this line the variable is on the left of a LET statement (it is modified).
  -- R The variable is referenced in this line and is on the right of a LET statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

The following program statistics appear beneath the table:

```
UNDIMENSIONED VARS :
*-------------------------------------------------------------*

VARIABLES REFERENCED = xxx OUT OF 340
SIMPLE VARIABLES = xxx  STRING VARIABLES = xxx
DIMENSIONED ARRAYS = xxx  TOTAL REFERENCES = xxx
PROGRAM B-FILE SIZE xxxxx WORDS
PROGRAM D-FILE SIZExxxxx WORDS + DATA AREA xxxxx WORDS
                                        = RUNSIZE xx KB
COMPILED VARIABLES = xxx OUT OF xxx TOTAL USED
*-------------------------------------------------------------*
```

D-FILE SIZE and COMPILED VARIABLES are only output for programs which have been saved as a D file (optimized).

Line number cross-references

The following is output from left to right in a table:

- Referenced line numbers (STMT)
  "U" indicates line numbers which are referenced but do not exist in the program.

- Line numbers of statements in which the reference occurs. All these line numbers are marked S or T.
  -- S Subroutine branch
  -- T GOTO statement

Channel number cross-references

The following is output in a table:

- Referenced channel numbers (CHNL)
  In the last place, "?" indicates channels which are not allocated until run-time (e.g. A), whose channel number exceeds 99 or which consist of several elements of an expression.
- Line numbers of statements in which the reference occurs.

The function of the command can also be called from the TAMOS selector for one or more programs (see the Program Maintenance chapter).

4

4.16 RENUMBER command

The RENUMBER command renumbers the lines of a BASIC program.

RENUMBER is generally used when there have been extensive modifications to BASIC programs, in particular, when there are no gaps between the line numbers as a result of inserting new statements, and when it is therefore impossible to insert any more.

Syntax:

[<LNo>] RENUMBER [<Integer>]

<LNo> : The line number to be given to the first statement in the program.
If <LNo> is not specified, the first statement receives the line number <Integer>.

<Integer> : Step

If the step is not specified, a default value of 10 is used.

If renumbering results in a line number above 65535, a step of 1 is used to number the statements.

• Example:

```
1000 RENUMBER 5
```

After renumbering, the line numbers begin at 1000 and have a step of 5 (i.e. 1000, 1005, 1010, 1015, 1020, etc.).

Error:

If a non-existent line number is referenced in a statement, the following message is issued:

```
ERROR # 6 AT <LNo>
```

The non-existent line number which was referenced is then replaced by the line number of the following statement. If no statement follows, the line number is set to 0. RENUMBER is continued.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

4.17 RUN command

The RUN command enables any section of the BASIC program located in the main memory to be executed.

Syntax:

[<LNo1>] RUN [<LNo2> [,<LNo2>]^{4}]

<LNo1> : Line number of the statement at which execution is to start.

This specification is only admissible if the program has already been executed up to <LNo1> in the DEBUG mode or if an initial RUN command has already been executed and the program has not been modified subsequently.
If this specification is omitted, the program is started at the first statement (initial RUN).

<LNo2> : Line numbers at which the program is to stop (stop condition) and enter the DEBUG mode.

When one of these line numbers is reached, the following message is issued:

STOP AT <LNo2> <Statement>

At this point all specified stop conditions are cancelled. This statement has not yet been executed. The system is in the DEBUG mode.
If <LNo2> is not specified, the program is executed up to its logical end.

- Example:

  RUN 300,400,500

  Starting a program at the beginning. The program is stopped when line number 300, 400 or 500 is reached (depending on which one it reaches first) and testing in the DEBUG mode can be continued.

Possible errors:

If a line is specified under <LNo1> which does not exist, the message:

ERROR # 6

is issued and the system remains in the BASIC command mode.

All further error messages which can occur during execution of the program are described in the chapter entitled "Error Handling". With the exception of errors which cannot be intercepted, the program is not terminated abnormally.

4.18 SAVE command

The SAVE command is used to save a BASIC program. The program is unloaded from main memory to magnetic disk.

Specification of ,B, or ,D, defines the file type in which the program is to be saved. If this is omitted, the program is saved in the file type in which it is stored in the partition. A new program, or one which has been LOADed into an empty partition, is created as a B file in the partition. If better run-time behaviour is required, only programs of type D should be used!

Syntax:

SAVE [,D,|,B,] [#<PartS>#] [<ProgName> [!]]

,B, : The program is saved as a B file.

,D, : The program is saved as a D file.

<PartS> : Partition size in KB
If this specification is omitted, the old value is used for existing files and a default value is used for new files.

<ProgName> : Name of the disk file under which the program is going to be saved. This specification can be omitted if the program has already been unloaded beforehand to disk and is due to be saved under the same name after modifications have been made.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

! : The specification "!" is necessary if an existing program file has to be replaced.

The SAVE command enters the BASIC program's hash total up to statement number 49999 in the header. This hash total is the same as the one which the LIBR processor calculates and displays using the associated text file.

The optimizer function of the command can also be called from the TAMOS selector for one or more programs (see "Program Maintenance" chapter).

4.19 SIZE command

The SIZE command is used to determine the current size of the program.

Syntax:

SIZE

The size of the program is output in the following form:

```
SIZE: xxxx WORDS, TOTAL: yyyy WORDS (zz KB) OUT OF
                                        uuuu WORDS (vv KB)
```

xxxx = The number of words occupied by statements and data areas (only variables which have already been dimensioned).

yyyy = The total number of words in the program including the auxiliary areas (stacks and list of variables).

zz = The minimum size of the partition in which the program can run.
This specification is, however, only realistic after all dimensioning has been carried out in the program.
The value shown here can be specified when the SAVE processor is called.

uuuu, vv = Partition size specified in the SAVE command.

4

4.20 TRACE command

The TRACE command is needed to test BASIC programs and is used as a supplement to the facilities provided by DEBUG. The STOP functions only take effect in programs of type B.

Syntax:

$$\left\{\begin{array}{l}\left[\text{<LNo1>}\right]\ \ \text{TRACE}\ \ \left[\text{<LNo 2>}\right]\ \ \left\{\begin{array}{l}\{,\text{<VarName>}\}^{5}\\ \{,\text{<Condition>}\}^{2}\end{array}\right\}\\ \\ \text{TRACE}\ \ \ \text{<File>}\end{array}\right\}$$

where:

$$\text{<Condition>::=}\left\{\begin{array}{l}\text{<NVarName1><RelOper>}\ \left\{\begin{array}{l}\text{<NVarName2>}\\ \text{<DecNo>}\end{array}\right\}\\ \text{<NVarName>}\ \{=\mid\text{<>}\}\ \text{<DecNo1>:<DecNo2>}\\ \text{<SVarName1><RelOper>}\ \left\{\begin{array}{l}\text{<SVarName2>}\\ \text{<SLit>}\end{array}\right\}\end{array}\right\}$$

The length of <SLit> is restricted to 16 characters.

The TRACE command provides three different functions:

- The first function is to stop a program when any values are assigned to certain variables. This corresponds to the syntax version:

  $$\left[\text{<LNo1>}\right]\ \ \text{TRACE}\ \ \left[\text{<LNo2>}\right]\ \ \{,\text{<VarName>}\}^{5}$$

  After the program has been started with RUN (or resumed with <LNo>RUN), it is stopped when a value is assigned to one of the variables specified as <VarName> between the first occurrence of <LNo1> and the first occurrence of <LNo2> (inclusively in each case). This happens with the statements LET, READ, INPUT, CALL, DIM, DEALLO and the corresponding MAT variants as well as with the data communication statements INIT, CNTRL, PUT, LON, LOFF in respect of the header variables and GET. With regard to CALL, TRACE works when the specified variables are referenced.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Control is passed to the DEBUG mode and the statement involved is output together with the associated variable names and their values. This statement has already been executed! The program can be resumed with <NextLNo> RUN. The TRACE command stays in effect, even beyond CHAIN and LINK. If <LNo1> or <LNo2> is not specified, the program section processed by the TRACE command is limited by the first and last statement respectively in the program.

A TRACE command is cancelled by another TRACE command, [CTL Y and] ESC, the end of the program or a modification of the program text.

With regard to assignment to indexed variables, only the component addressed in the assignment (or the referenced substring) is output. There is no output in the event of a stop after MAT... statements. Components of the matrix can however be displayed by means of PRINT or MAT PRINT. If a variable is DEALLOcated, this is indicated by <VarName>="deallocated".

- The second function of the TRACE command is to stop at specific values or ranges of values. This corresponds to the syntax version:

  [<LNo1>] TRACE [<LNo2>] {,<Condition>}²

  Program execution is stopped when one of the addressed variables in the particular part of the program is assigned a value that satisfies <Condition>. The stop condition <NVarName>= <DecNo1>:<DecNo2> is satisfied when the value of <NVarName> or the corresponding components lies between <DecNo1> and <DecNo2> (inclusive) or ...<>... if this is not the case.
  Apart from this, the rules for the first function apply.

- The third function outputs all variables dimensioned in the program together with their values to display, printer or to a text file after the program run. This corresponds to the syntax version:

  TRACE [<File>]

  <File> must be the name of a text file or a printer for the output. If <File> is not specified, the display is used for the output.

  The output is three characters for the variable name and 4, 6, 10 or 14 characters for numeric variables in exponential notation and is of any length for string variables. All numeric variables are output first and are followed by the string variables. A non-printable character in a string is replaced by a blank.

  The TRACE command does not cancel the DEBUG mode.

4.2.1 UNCOMMENT command

The UNCOMMENT command unloads remarks from a program and stores them in a text file. They can be reloaded by means of the COMMENT command.

Syntax:

[<LNo1>] UNCOMMENT [<ProgName> [!]]
[<LNo2>] [REM [<LU2>]]

<LNo1> : Line number of the statement at which unloading of remarks has to start. If this specification is omitted, unloading starts from the beginning of the program.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Commands

<ProgName> : Remarks are unloaded from the program in the partition and the reduced program is saved to <ProgName> which includes the logical unit number. The actual name can be 12 characters long at most because the remarks are written to a text file of the same name but with Y. as a prefix. If <ProgName> is not specified, the name of the program in the partition is used.

! : An "!" is necessary if a program called <ProgName> already exists and is intended to be overwritten.

<LNo2> : Line number of the statement at which unloading of remarks has to finish. If this specification is omitted, unloading goes on to the end of the program.

REM [<LU2>]: Number of the logical unit to which the text file containing the remarks has to be written. This specification can be omitted if <LU2> equals 0.

Remarks are unloaded from REM statements and from statements which have remarks appended by means of /*.
If the branch address of a GOTO or GOSUB is a REM statement, this branch address is changed to the line number of the next statement which is not unloaded.
A REM01 statement is not unloaded.
The function of the command can also be called for one or more programs from the TAMOS selector (see "Program Maintenance" chapter).

Data Formats and Data Representation

5 Data formats and data representation

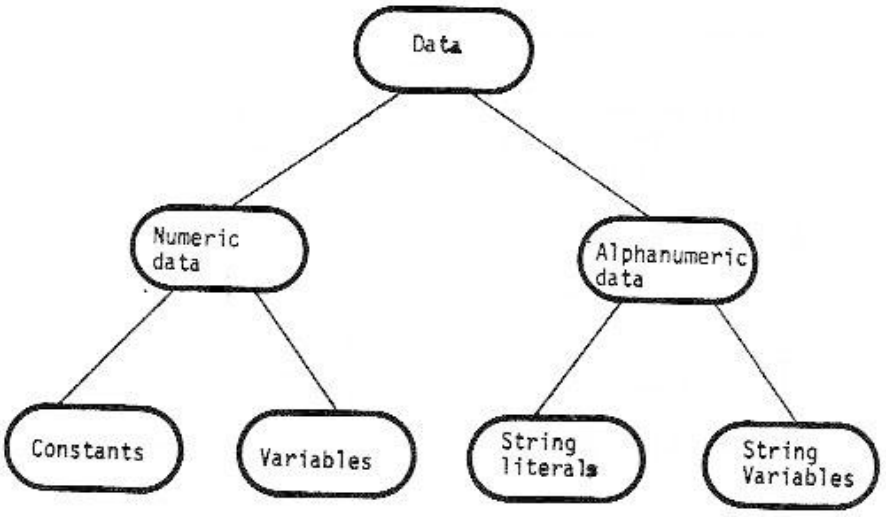
The Business BASIC language distinguishes between numeric data and alphanumeric data. Alphanumeric data are also termed strings or character strings. This is a practical distinction due to the fact that numeric data normally serve as a basis for arithmetic processes, whereas alphanumeric data are used for displaying and printing purposes, in particular, and as search and sort criteria.

Both numeric and alphanumeric data are again divided into variable and constant data. The term variable refers to single or several memory words in the data area of the user partition into which data is placed or in which alterations are made. The variable should therefore not be equated with its value.

Data defined on creation of the program and which are not altered during execution of the program are termed constants. Constants do not have names; they are only represented by their value.

Designations:

- (Simple) variable = Numeric variable
- Constant = Numeric constant
- String variable = Alphanumeric variable
- String literal = Alphanumeric constant



© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Data Formats and Data Representation

5.1 Representation of numeric data

The structure of the main memory influences the representation of numeric data. The memory of the 8870 system has a word structure.

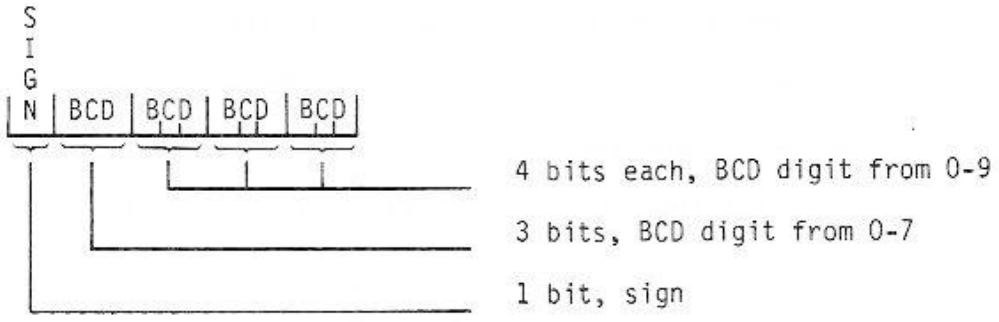
1 word = 2 bytes = 16 bits

5.1.1 Simple numeric variables

A numeric variable occupies 1, 2, 3 or 4 words in memory (format of variables).
The values of 1-word variables are stored as signed BCD integers whereas the values of 2- to 4-word variables are represented as floating-point numbers.

"BCD" is the abbreviation for binary-coded decimal and means that each decimal digit is coded individually in binary form.

5.1.1.1 Structure of BCD integers



The values which can be represented are therefore all integers between -7999 and +7999. It is not possible to represent a decimal point.

- Examples:

BCD integer	Decimal number
0 011 1000 0111 0101	+3875
1 000 0010 1001 0110	-296

Data Formats and Data Representation

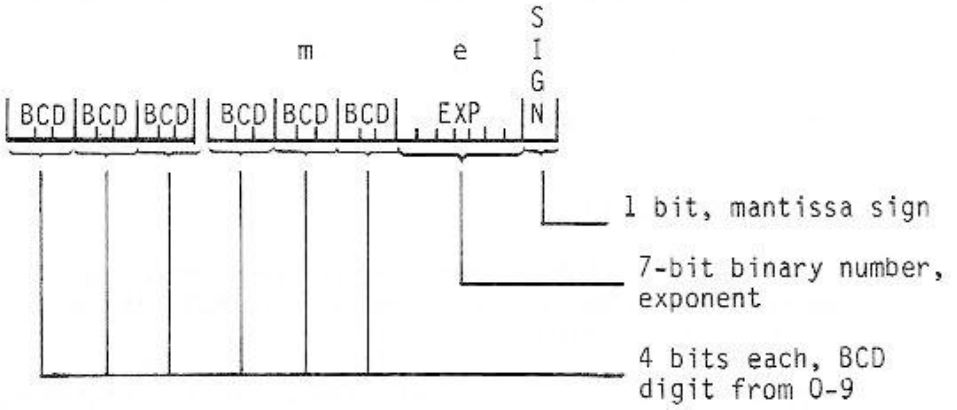
5.1.1.2 Structure of floating-point numbers

Floating-point notation is a semi-logarithmic representation of numbers. The numbers are represented in the form:

$m * 10 \uparrow e$ when $0.1 < ABS(m) < 1$

m = Mantissa
e = Exponent (or characteristic)

Structure of a 2-word floating-point number:



The exponent is coded as a binary number in the following manner:

```
0000001   =   -63
   :
0111111   =    -1
1000000   =     0
   :
1111111   =    63
```

The decimal point of the mantissa is always assumed to be before the first BCD digit.

- Examples:

Floating-point number	Decimal number
0111 0010 0101 0100 0110 0001 1101011 0	+0.725461E+43
0110 0101 0000 0000 0000 0000 0011101 1	-0.65E-35

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Data Formats and Data Representation

In the case of 3- and 4-word variables, these two words are preceded by one or two words respectively with 4 BCD digits each. The precision of the mantissa is thus increased to 10 or 14 significant digits respectively.

The maximum and minimum value which can be represented is:

```
2-word variables:          +/-0.999999E+63
3-word variables:      +/-0.9999999999E+63
4-word variables: +/-0.99999999999999E+63
```

The least difference to 0 which can be represented is:

```
+/-0.1E-63
```

for 2-, 3- and 4-word variables.

5.1.1.3 Arithmetic overflow

If a value is returned during processing which can no longer be represented in the variable which is to hold it, BASIC error #15 (arithmetic overflow) is issued.

This error can be intercepted by means of the statement IF ERR 0.

After this has occurred, the variable contains the maximum/minimum value that can be represented (see above).

If, in a 2-word to 4-word variable, a value results consisting of more significant digits (digits of the mantissa) than the variable can hold, the overflow digits are truncated from the right without an error message being issued.

5.1.2 Vectors/matrices

Vectors:

Vectors are linear data fields consisting of between 2 and n elements. A vector element has the same structure as a numeric variable and can be 1 to 4 words in length. All elements in a vector have the same format.

Matrices:

Matrices are two-dimensional data fields comprising:

2 to n rows and
2 to m columns.

A matrix element has the same structure as a numeric variable and can be 1 to 4 words in length. All elements in a matrix have the same format.

Note:

If the term <NVar> is used in the syntax, it designates:

- a simple numeric variable or
- a vector element or
- a matrix element.

Matrix variables <MVar> on the other hand designate matrices in their entirety.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Data Formats and Data Representation

5.1.3 Numeric constants

A numeric constant is any value in the range:

0 and +/-0.99999999999999E+63 in floating-point notation
or
maximum 63 digits in decimal notation.

If no sign is specified, the constant is positive.

A numeric constant always occupies 4 memory words, i.e. it is represented with the greatest degree of precision possible.

- Examples:

  - Verifying whether the contents of the numeric variable X lie within 1 and 10.

    ```
    90 IF X>=1 IF X<=10 GOTO 210
    ```

  - Performing an arithmetic operation with a numeric constant.

    ```
    50 LET U=D*3.14159
    ```

Data Formats and Data Representation

5.2 Representation of alphanumeric data

Alphanumeric data are represented in modified ASCII. Each character occupies

1 half word = 1 byte = 8 bits.

Due to the fact that ASCII only comprises 7 bits, the bit on the left is always set to 1 when a character is represented.

- Example:

Alphanumeric representation of the "AB" character string.

A	B
11000001	11000010

A list of all ASCII characters can be found in the "Keyboard Input/Screen Output" chapter.

5.2.1 String variables

Unlike numeric variables, no rules exist with respect to the size of string variables. They can be freely defined by the programmer and have a length of 1 to n characters. The maximum length of a string is only restricted by the size of the partition.

Data are always stored left-justified in string variables.

Further forms of representing characters in strings and how this is possible by means of various CALL statements is discussed in the "CALL Statements" chapter.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

5.2.2 String literals

An alphanumeric constant (string literal) is a sequence of optional characters enclosed in "". In addition to specifying the character itself, it is also possible to specify the octal code of characters in "←" (see Code Table).

• Examples:

- Input with check for input of Y or N.

```
:
20 INPUT TAB(X,Y), "PRINT Y/N",A$
30 IF A$="Y" GOTO 100
40 IF A$="N" GOTO 660
```

- Storing the alphanumeric constant "ABCD" and the function code for the acoustic signal in a string.

```
:
50 LET X$="ABCD←207←"
```

After the LET statement has been executed, a delimiter is located in X$ at position 6.

Note:

If either of the two characters " or ← is to be specified in a string literal, this must be done in octal code. In addition, it is possible to create the " character with the aid of two single quotation marks '.

5.3 Declaration (dimensioning) of variables

Storage space is reserved for a variable (dimensioning) when the program is executed. Here, a distinction is made between:

- Explicit dimensioning by means of a DIM or GLOBAL statement
- Implicit dimensioning when the variable is addressed for the first time.

The following must be explicitly dimensioned:

- Name and length of string variables

Implicit or explicit dimensioning is possible with the following:

- Name of numeric variables
- Precision of simple numeric variables
- Precision of vector elements
- Precision of matrix elements
- Number of elements in a vector
- Number of rows and columns in a matrix.

However, every variable used should (for reasons of optimization and documentation) be explicitly dimensioned as a matter of principle and a distinction should be made between local variables (DIM) and global variables (GLOBAL).

A maximum of 340 variables can be dimensioned.

A valid variable name comprises a letter or a letter and a digit.
Examples: A, F9, XO, Z.
A string variable name is a valid variable name followed by a "$",
Examples: A$, B5$, F7$, X$.

After dimensioning has taken place, numeric variables and vector and matrix elements contain the value 0. Strings are filled with delimiters (code: all 8 bits set to 0).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

GLOBAL statements are used to declare those variables that have to be passed on to a LINKed program. When a program contains GLOBAL statements, only those variables named in the GLOBAL statements are passed on.

Strings and simple numeric variables are only allowed to be redimensioned in a program when they have occurred in a DEALLO statement beforehand! Variables which have been declared in a GLOBAL statement must not occur in a DEALLO statement.

On the other hand, matrices and vectors may be redimensioned without a DEALLO statement if the number of elements is not increased. The format of the elements cannot, however, be changed (specification of a format is ignored); nor are the elements initialized with 0. If a matrix has been reduced, it is not possible to increase it back to the old number of elements by means of redimensioning.

The use of a DEALLO statement and other redimensioning functions prevents the generation of variable addresses. Such redimensioning should only be used for programs in which the run-time is not critical.

5.3.1 Structure of the DIM and GLOBAL statements

DIM <FieldDescr> [,<FieldDescr>]^{n}

where:

<FieldDescr>::= { <NVarName> [(<NExpr>[,<NExpr>])] | <SVarName> (<NExpr>) | <Format> }

GLOBAL <FieldDescr> [,<FieldDescr>]^{n}

where:

<FieldDescr>::= { <NVarName>[(<Integer>[,<Integer>])] | <SVarName> (<Integer>) | <Format> }

5

5.3.2 Dimensioning simple numeric variables

There are two methods of dimensioning numeric variables:

- Explicit dimensioning:
  Dimensioning of variables with the DIM or GLOBAL statement.

- Implicit dimensioning:
  Dimensioning when variable names occur for the first time in statements such as LET, FOR, IF, INPUT etc.

The size of the numeric variables is controlled by a format switch which has the following four settings:

```
1%  =  1-word integer
2%  =  2-word floating-point number
3%  =  3-word floating-point number
4%  =  4-word floating-point number
```

Storage space is reserved for a numeric variable in accordance with the state of this switch on dimensioning.

The standard state of this switch is "2%" and it can only be changed with the aid of a DIM or GLOBAL statement. If the format is specified in such a statement (1% to 4%), the switch adopts the corresponding state. This remains valid for any subsequent dimensioning until it is changed by a new format specification.

Note:

The setting of the switch can be influenced by a dynamic program run.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Data Formats and Data Representation

- Example of explicit dimensioning:

```
10 GLOBAL A,1%,B
20 DIM    C, 2%,D
30 DIM    3%,E,F
```

Variables A and D are dimensioned for 2 words, B and C for 1 word, E and F for 3 words.

The last format specification (3%) remains valid until it is changed by a further DIM statement.

- Examples of implicit dimensioning:

```
- :
   20 INPUT A
```

A is dimensioned as a 2-word variable due to the fact that no DIM or GLOBAL statement has occurred and the format switch is set to the standard state of 2%.

```
- 10 DIM 1%,A,B,C,3%
  :
  50 LET X = A*3.14159
  :
  70 DIM 2%
  :
  90 INPUT D
```

The DIM statement reserves space for the 1-word variables A, B and C. The variable X, which was not dimensioned in the DIM statement, is dimensioned implicitly as a 3-word variable due to the fact that the format specification of 3% is the one in use at the time. Prior to dimensioning D, the format switch is set to 2% so that D is dimensioned as a 2-word variable.

5.3.3 Dimensioning vectors/matrices

When vectors/matrices are dimensioned, the number of elements or rows and columns must be defined, in addition to the format specification. They can be both explicitly and implicitly dimensioned. The same rules apply to the format of the vector and matrix elements as to simple numeric variables.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Explicit dimensioning of vectors:

The number of elements which the vector is to contain is defined by specifying the number of the last vector element in parentheses following the variable name.

- Example:

```
10 DIM A(24)
```

The elements of a vector are numbered from 0 to this specification. This means that the number of elements of a vector corresponds to the number of the last element + 1 (in the example: 25 elements).

The precision of the elements is defined by the current setting of the format switch which can be changed by means of the same statement.

- Example:

```
10 DIM 4%,A(24),2%
```

Explicit dimensioning of matrices:

The number of rows and columns in a matrix are defined by appending the numbers of the last row and the last column to the variable name.

- Example:

```
10 DIM B(24,79)
```

Numbering of the rows and columns also commences at 0. This means that the number of columns/rows corresponds to the numbers specified + 1.
The number of elements in a matrix is calculated from the number of rows x number of columns (in the example: 2000 elements).

Data Formats and Data Representation

Note:

All matrix statements and functions start processing matrices from column 1 and row 1. The elements in row 0 and column 0 must, if necessary, be addressed directly and can, for example, be used for column totals, etc.

Implicit dimensioning of vectors/matrices:

Vectors and matrices which are addressed in any statement and which have not yet been dimensioned by means of a DIM statement are dimensioned by RUN when they occur for the first time:

- An implicitly dimensioned vector comprises 11 elements (0 to 10).

- An implicitly dimensioned matrix comprises 11 rows and 11 columns (each 0 to 10).

The precision of the elements is determined by the current setting of the format switch.

Clearly it is not possible to address an element or a row and column greater than 10 in the case of implicit dimensioning.

Note:

If, with implicit dimensioning, the element 0 for vectors or the element in row 0 and column 0 for matrices is addressed first, only a simple variable is dimensioned.

- Examples for vector/matrix dimensioning:

  - Dimensioning a vector with 10 elements (0 to 9) with the name V1. Each element is to have a length of 2 words. The format switch has not yet been changed.

    ```
    10 DIM V1(9)
    ```

  - Dimensioning a matrix with 6 rows and 8 columns with the name M1. Each element is to have a length of 4 words.

    ```
    10 DIM 4%,M1(5,7)
    ```

    The format specification of 4% applies to all elements of the matrix M1 and for all subsequent dimensioning until a new format specification occurs in a DIM statement.

  - The statement sequence

    ```
    10 DIM 1%,A,B,C,3%,X
    :
    50 LET X1(8) = X
    ```

    dimensions the vector X1 with 11 elements (0 to 10). All elements have a format of 3 words.

  - The statement sequence

    ```
    10 DIM A,B,1%,C
    :
    50 LET M1(3,8)=C
    ```

    dimensions the matrix M1 with 11 rows and 11 columns. Each of the elements have a format of 1 word.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

5.3.4 Dimensioning string variables

String variables can only be dimensioned by using the DIM or GLOBAL statement.

Dimensioning defines the name of the string and the maximum number of characters (bytes) which the string can hold.

This size is appended in parentheses to the string variable name in the dimensioning statement.

- Example:

```
10 DIM A$(100),B$(30)
```

Dimensioning of string variables

- A$ with a length of 100 bytes, and
- B$ with a length of 30 bytes.

The format switch used for dimensioning numeric variables is not changed as a result of string dimensioning.

5.4 Addressing variables

5.4.1 Simple numeric variables

Simple numeric variables are addressed by specifying the name of the variable.

- Example:

```
50 LET A=B+C
```

5.4.2 Vectors/matrices

In order to address the individual elements of a vector, it is necessary to specify the name of the vector and the number of the element (0 to n) to be selected.
In order to address the individual elements of a matrix, it is necessary to specify the number of the row (0 to n) and the number of the column (0 to n) of the element to be selected.

If a vector or matrix is addressed without specifying the indices, then access is to:

- element 0 for vectors and
- element 0,0 for matrices
  If a matrix is addressed with only one index, the second index is defaulted to zero.

  Matrices are always addressed as entire units by matrix statements (MAT), however, without row 0 and column 0. In this case the name is specified without indices (<MVar>).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Example:

  Addressing the element in row 2 and column 4 of a matrix with 4 rows and 5 columns with the aid of a LET statement:

  50 LET X(2,4)= ...

Row 0 to 3 \ Column 0 to 4	0	1	2	3	4
0	0,0	0,1	0,2	0,3	0,4
1	1,0	1,1	1,2	1,3	1,4
2	2,0	2,1	2,2	2,3	**2,4**
3	3,0	3,1	3,2	3,3	3,4

5.4.3 String variables

The following rules must be observed when addressing strings:

- Data in strings are always stored left-justified and processed from left to right.
- The bytes in a string are numbered from left to right, commencing with 1.
- Each item of information written to a string is followed immediately by a delimiter.

Processing delimiters

The maximum number of characters that a string can hold is defined during dimensioning. An additional byte is reserved internally for the delimiter.
Due to the fact that a string need not be filled with data throughout its entire dimensioned length, RUN writes delimiters to the string as follows:

- In all bytes during dimensioning
- After the last character entered subsequent to an INPUT statement
- After the last character transferred subsequent to a LET statement (exception: see below).

If the string variable contains data throughout its entire dimensioned length, the string delimiter is located outside the area that can be addressed by the program.

Addressing strings

Strings can be addressed in the following ways:

- Addressing a string from byte 1 up to the first delimiter by specifying the string name.

  Example:

  90 PRINT A$

- Addressing a string from a byte address to be specified in parentheses up to the next delimiter.

  Example:

  50 IF A$(19)="END" GOTO 690

- Addressing a substring within byte addresses to be specified in parentheses.

  Example:

  60 LET A$(3,7) = X$

As far as strings are concerned, index 0 is interpreted as follows: A$(0) as A$, A$(0,n) as A$(1,n), A$(n,0) as A$(n). Moreover, A$(1) is the same as A$.

String processing is terminated when:

- a delimiter is reached in the source string,
- the end of the destination string is reached and
- the specified byte end address is reached in the source or destination string.

After a transfer has taken place, a delimiter is always stored in the destination string after the last character transferred.

Exception:

In the case of operations with start and end indices in the destination string, no delimiter is stored if no existing delimiter is overwritten.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Data Formats and Data Representation

If, in the case of a LET or INPUT statement,

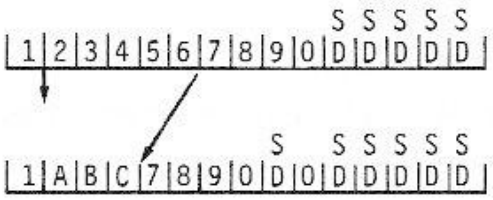
- the destination string is addressed with start and end indices,
- the number of characters to be transferred is less than the area in the destination string intended to hold them, and
- no delimiter is overwritten,

the characters located to the right of the addressed end byte up to the next string delimiter are shifted left and stored adjacent to the last character that was transferred.

- Example:

```
20 DIM A$(15)
30 LET A$="1234567890"
:
90 LET A$(2,6)="ABC"
```

A$ before and after execution of statement 90:



5.5 Hexadecimal constants

A hexadecimal constant is allowed in a BASIC statement wherever a string literal is permitted (e.g. in LET or INPUT statements).

It has the form:

'H {xx}^{n}'

where x represents a digit (0-9) or a letter (A-F).

- Example:

```
LET A$ = 'H7B3F'
```

Note:

The hexadecimal value 00 constitutes a delimiter and is not accepted by the system as a hexadecimal constant. A string variable can be filled with delimiters by means of the CLEAR statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6 BASIC statements

Statements form the basic units of a programming language and can be compared with sentences in a natural language.
The syntax of the statements of the BASIC programming language have already been introduced in the "Summary of Statements", and this chapter provides an explanation of the structure.

Each program consists of a sequence of statements to be executed. Each statement which is included in a program must be preceded by a line number. The input from the BASIC command mode is in the form:

```
<LNo> <Statement>
```

The line number must be between 1 and 65535.

Input of each statement is concluded by pressing the CR key.

A syntax check is first carried out on the statement.
In the event of an error, the message

```
ERROR # xx
```

is displayed. Entering HELP will produce a plain language description of the error. The cursor is positioned to the location in the line where the error was detected.

If a statement has the correct syntax, it is included in the program in accordance with its line number and further statements can be input. It is therefore not necessary to enter the statements in their final order of appearance. They are automatically sorted in ascending order of line numbers. It is advisable to leave a gap when allotting line numbers (such as incrementation in steps of 10) so that statements can be inserted easily later. Examples can be found in the "Programming Examples" chapter.

If a statement which has been accepted is to be corrected, it must be re-entered with the same line number or be processed with the EDIT command.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

If a statement which has been included in a program is to be deleted, the line number of the statement must be entered and acknowledged (cf. DELETE command).

In order to support program testing in interactive mode, individual BASIC statements can be executed directly after input. Direct execution can be initiated by entering the statement without a line number, thereby not permitting the statement to be included in the program. When a statement is executed directly, the program in main memory at the time is not changed.

To facilitate description, the statements have been split up into 5 groups:

- Declaration of data fields and functions
- Data manipulation and arithmetic
- Controlling the program flow
- Input/output statements
- Special statements

6.1 Declaration of data fields and functions

Storage space is allocated to variables (including vectors, matrices and string variables) by means of the DIM and GLOBAL statements and is deallocated with DEALLO. User functions can be defined by means of the DEF statement. They apply locally to the program in which they are defined.

Particular attention should be paid to the distinction between local and global variables.

Local variables are only relevant in a particular program and are not transferred directly to LINKed programs. In particular, this means that programs which neither call others nor are called by others only contain local variables.

BASIC Statements

Local and global variables are declared with the DIM and GLOBAL statements respectively. GLOBAL statements must be the first statements in the program (exception: REM statements) and must be followed by any DIM statements. This ensures that the program run-time can be optimized.

The terms "local" and "global" variables refer to the calling of programs and transfer of variables by means of the LINK statement (not to transfer via the common area or files).

Global variables are transferred directly from the calling program to the called program by means of the LINK statement. (Variables cannot be transferred with the CHAIN statement.)

There are two methods of defining global variables:

1. All variables dimensioned explicitly (with DIM) or implicitly, which are not DEALLOcated, are global if there is no GLOBAL statement.

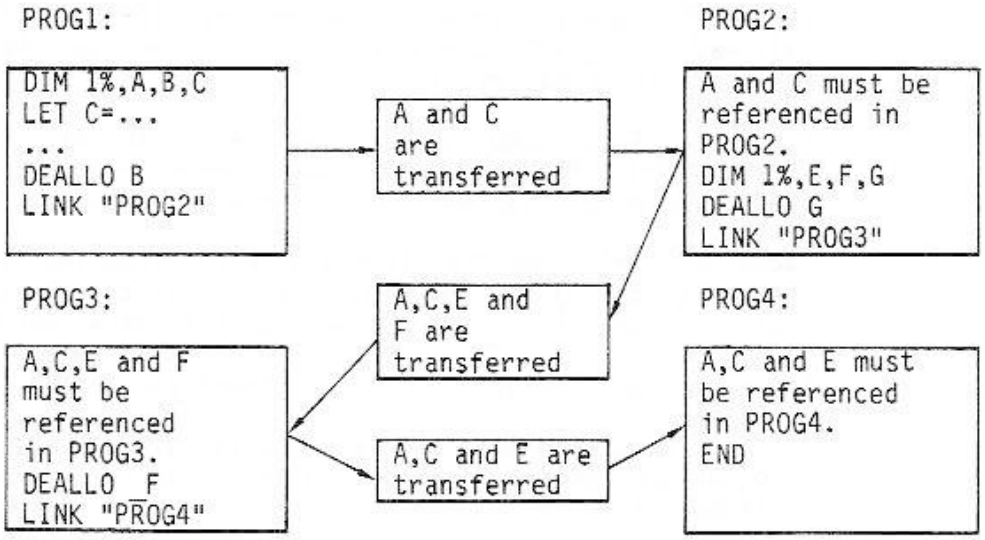
2. All variables contained in GLOBAL statements are global.

The two methods are incompatible with each other! When programs are LINKed, the user must use method 1 or method 2, not both. Method 2 is preferable because the program can be optimized to improve run-time.
Both methods work for B and D files.

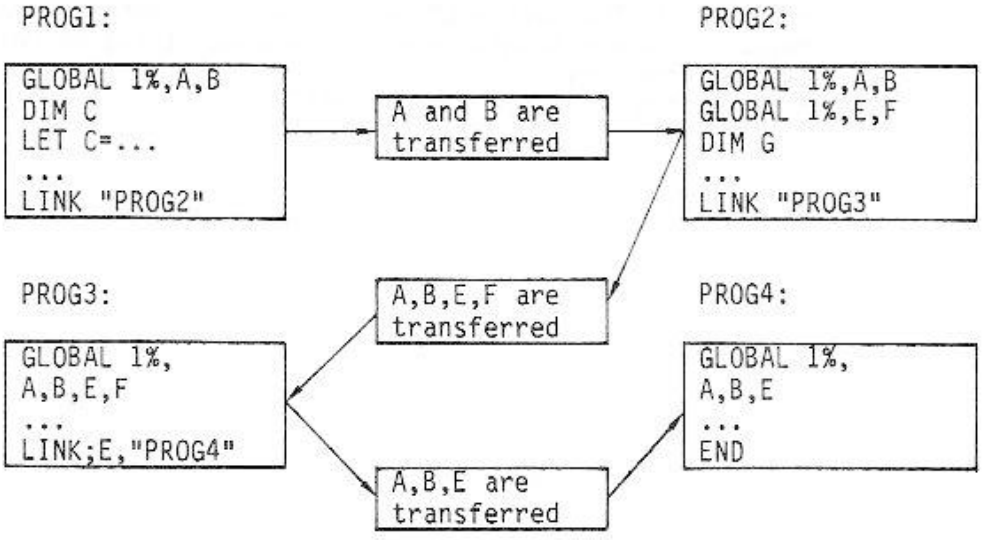
© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

- Example of method 1:



- Example of method 2:



The statement LINK;E,"..." in PROG3 causes the values of the global variables A to E (inclusive) to be transferred to the successor program.

6.1.1 DIM statement

The statement reserves storage space for:

- Numeric variables
- Vectors/matrices
- Strings

The DIM statement defines a name for each variable with which it can be addressed while the program is running.

Syntax:

DIM <FieldDescr> [,<FieldDescr>]^{n}

where

<FieldDescr> ::= { <NVarName> [(<NExpr> [,<NExpr>])] | <SVarName> (<NExpr>) | <Format> }

6.1.2 DEALLO statement

The DEALLO statement deallocates variables that have been declared explicitly in a DIM statement or implicitly. (The name and storage space can be reallocated.) The variables can subsequently be redimensioned. Variables that have been declared in GLOBAL statements must not occur in DEALLO statements.

The second function of the DEALLO statement is to deallocate local variables before programs are called with the LINK statement. Global variables cannot be deallocated. These programs do not contain any GLOBAL statements. Transferred variables must be referenced at least once in the called program. This procedure is no longer recommended as it affects run-time adversely. It is better to declare global variables with the GLOBAL statement, thus making the DEALLO statement unnecessary (cf. previous chapter "Declaration of Data Fields and functions").

Syntax:

DEALLO <VarName> [,<VarName>]^{n}

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

The time required to execute the statement depends on the location of the variables concerned in the list of variables, since deallocation creates a gap which must be closed again. In order to reduce the execution time for this statement, the variable names in the DEALLO statement should be specified in the opposite order to dimensioning.

- Example:

```
100 DIM A$(100),1%,A,B,B$(200)
:
190 LET X=A*B+3
:
500 DEALLO X,B$,B,A,A$
```

Note:

If any errors occur when this statement is being executed, deallocation is terminated at the variable causing the error.

6.1.3 GLOBAL statement

Global variables in a program are declared and dimensioned with the GLOBAL statement. A program containing GLOBAL statements only transfers the variables mentioned in the GLOBAL statements to any LINKed program. The variables are transferred in the order in which they are written. The number of global variables to be transferred can be restricted to an initial range in the LINK statement.

The LINKed program must declare the transferred variables in GLOBAL statements in the same order and size as specified in the calling program. In order to increase the number of global variables, the LINKed program must declare the additional variables in other GLOBAL statements.

Variables that have been declared in GLOBAL statements cannot be DEALLOcated.
Only REM statements are allowed before and between GLOBAL statements.
Dynamic dimensioning is not possible with a GLOBAL statement.

Syntax:

GLOBAL <FieldDescr> [,<FieldDescr>]^{n}

where:

<FieldDescr>::={ <NVarName> [(<Integer> [,<Integer>])] | <SVarName> (<Integer>) | <Format> }

6.1.4 DEF statement

The DEF statement is used to define user functions. Typing and storage space can be saved with the aid of user functions if equivalent numeric expressions (formulas) are to occur several times in the program.

Syntax:

DEF FN<Le1>(<Le2>)=<NExpr>

The keyword DEF must be followed by FN and a letter, a letter enclosed in parentheses, an equals sign and a numeric expression.
The letter following FN identifies the function.
The letter <Le2> enclosed in parentheses is the formal argument of the function which is replaced by the actual argument at the point at which the function is called. Only one argument is permissible.

- Example:

```
:
30 DEF FNS(X)=2*X+1
:
180 LET A=FNS(5)
```

Here, X is the formal and 5 is the actual argument. A is given the value 11.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

The formal argument may only comprise one letter and this letter may have already been given as a variable name.

The DEF statement must always have been executed once prior to the actual function being called for the first time.

User functions can be nested, i.e. further user-defined functions may be called in the numeric expression of the DEF statement. The nesting depth is restricted to 8.

- Example:

```
30 DEF FNX(X)=FNY(X+3)*FNZ(8.5)
```

It is possible to redefine a user-defined function.

6.2 Data manipulation and arithmetic

There follows a description of the statements used to carry out data manipulation and arithmetic operations.

These statements are:

- LET
- MAT
- MAT READ
- DATA
- RANDOM
- READ
- RESTOR

6.2.1 LET statement

The LET statement assigns the value of the expression on the right of the "=" to the variable on the left.

The "=" character should therefore not be interpreted as an equals sign in the mathematical sense but as:

....."receives the value of"....

If the type and format of the destination variable does not match that of the expression to be calculated, the data is converted to the format of the destination variable.

Syntax:

```
[LET] { <NVar>=<NExpr>
      { <NVar>=<String>
      { <SVar>=<SExpr>
      { <SVar>=<NExpr> [USING<String>]
```

The keyword LET can be omitted, except in instances when the LET statement occurs in an IF statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

Numeric variable as the destination variable

The term numeric variable signifies a simple numeric variable, a vector element or a matrix element. When the value of a numeric expression is assigned to the destination variable, the former is calculated and transferred directly.

If a string variable or a string literal is to be assigned to a numeric variable, the string is transferred until

- the end of the string is reached or
- a non-numeric character occurs.

Exceptions: "+" or "-" at the first position
"E" indicating exponent
Decimal sign

A numeric value in exponential notation is converted to a number with non-exponential notation before the transfer, if possible. If no numeric character is transferred, the destination variable is given the value 0.

● Examples:

```
10 LET A=5
20 B=B+1
:
70 LET C(J)=(X-A)/2+J
:
100 X="3.75E+02VALUE"   (Result: X=375)
```

In order to clarify this, let us split up a LET statement into its individual steps using the statement with line number 20 as an example:

1. Load the contents of B into an arithmetic register.
2. Add 1 to the contents of the arithmetic register.
3. Store the contents of the arithmetic register in variable B again.

String variable as the destination variable

A string variable is understood to be a complete string variable or a contiguous part of a string variable, i.e. a string variable with one or two indices.
A string variable can be assigned the value of a string expression. Value assignment of string expressions is only possible for string variables as destinations.

If a string variable is to be assigned a numeric expression, the value of the numeric expression is first calculated. The value is then converted character for character via a mask into ASCII and transferred to the string variable. The mask is defined via the USING specification or a standard mask is employed. It has the following structure:

- Sign (1 byte); a blank is stored for a "+" and a minus sign is stored for a "-" in the destination string.
- The value of the expression including decimal point and exponent (in the case of floating-point notation) is transferred directly after the sign.
- A blank is transferred immediately after the last digit transferred.

If there is no mask specification (USING), the transfer is in floating-point format if a value results which is:

- greater than 99999999999999
- less than 0.1

Note:

Transfer to a string is left to right and is terminated:

- when the dimensioned number of bytes in the destination string is reached,
- when the delimiter in the source string or the end of a string literal is reached,
- when the end criterion in the last element is reached if a string expression is used as source,
- after the closing blank or the last digit (with USING) of the value in a numeric expression has been transferred.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Processing delimiters

The maximum number of characters which a string is able to hold is determined in the course of dimensioning. An additional byte is reserved internally for the delimiter.
Due to the fact that the entire dimensioned length of a string need not be filled with data, RUN writes a delimiter after the last character transferred subsequent to a LET or INPUT statement.

If the entire dimensioned length of the string variable is filled with data, the delimiter is located outside the area which can be addressed by the program.

Exception:

If no delimiter is overwritten, a delimiter is not stored for operations with start and end indices in the destination string.

If, in the case of a LET or INPUT statement, the destination string is addressed with start and end indices, and the number of characters to be transferred is less than the area in the destination string intended to hold them, the characters located to the right of the addressed end byte up to the first delimiter are shifted left and stored adjacent to the transferred characters.

- Examples:

  - Transferring the literal "STATUS: " and the contents of M$ to the variable A$.

```
:
770 LET M$="FILE NOT FOUND"
:
1000 LET A$="STATUS: ",M$

A$ after transfer:

STATUS:  FILE NOT FOUND
```

- Filling the variable M$ with: ....5....0....5...

```
:
50 DIM M$(50)
:
570 M$="....5....0",M$
```

M$ after transfer:

....5....0....5....0....5....0....5....0....5....0

Due to the fact that the transfer is from left to right, the literal is stored on byte 1 to 10 in M$. M$ is subsequently interpreted as the destination and source string.
In this example:
- byte 1 is transferred to byte 11,
- byte 2 is transferred to byte 12, etc.

until the dimensioned end of M$ (destination) has been reached.

- Filling all bytes of M$ with blanks.

```
1000 LET M$=" ",M$
```

- Storing the string literal "$$$" in characters 3 to 5 of A$ and B$ with and without creating a delimiter respectively.

```
490 LET A$(3,5)="$$$"     570 LET B$(3)="$$$"
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.1.1 Numeric expressions

Syntax:

<NExpr>::= [+ | -] <Operand> [<NOper> <Operand>]^{n}

where

<Operand>::= { <NVar> | (<NExpr>) | <DecNo> | FN<Le>(<NExpr>) | <NFct> }

A numeric expression specifies the rules for calculating a numeric value and is formed by linking operands by means of numeric operators. The first operand can include a sign without parentheses. If the second or a subsequent operand has a sign, additional parentheses are needed!

When an expression is calculated, the normal mathematical hierarchy applies:

1. Calculation of expressions in parentheses
2. Calculation of exponentiation
3. Calculation of multiplication and division
4. Calculation of addition and subtraction

Expressions in parentheses are calculated on the basis of the same hierarchy. If several sets of parentheses are nested in one another, the parentheses are processed "from the inside outwards". Superfluous sets of parentheses are ignored.

- Example:

```
Numeric expression:  (2*(8-4))/(2↑2*2)
after the 1st step:  (2*  4  )/( 4 *2)
after the 2nd step:    8      /    8
after the 3rd step:           1
```

- Examples of permissible numeric expressions:

```
-(C+1)        Z*(-4)          +99          A(3,3)*A(3,1)↑2

B1*2/(-3.14)+INT(X+.5)       -(Y1/2.)*FNS(X+5)*FNS(A3)
```

6.2.1.2 Combining values of different precision

Any values of different precision can be combined with one another as required.

- Example:

```
10 DIM 1%,A,2%,B,3%,C,4%,D
:
50 LET E=A*B+C/D
```

Intermediate results are always formed with the greatest precision possible.

However, the representation of the result depends on the precision of the destination variable.

6.2.1.3 Calling user-defined functions

Syntax:

FN<Le>(<NExpr>)

The user-defined function addressed by FN<Le> must have been defined in a DEF FN statement prior to being called.

When a user-defined function is called, the value of <NExpr> is used for the formal argument on the right-hand side of the definition, and the value of the numeric expression which results in this way is calculated.

- Example:

```
10 DEF FNY(Z)=5*Z+17*(3-Z)
:
40 LET X=20
50 LET A=20+FNY(X-18)
```

X-18 replaces the formal argument Z when the function is called, and the numeric expression which results is as follows:

5*(X-18)+17*(3-(X-18))

The value of this expression is 27, thus meaning that the value 47 is assigned to variable A.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.1.4 Calling numeric BASIC functions

Business BASIC provides the programmer with a series of functions (<NFct>) which return a numeric function value and which can be called in the following manner:

<NFct>::= { <FctName>(<NExpr>) | LEN(<SVar>) | DET(<MVar>) }

Specification of parentheses can be omitted if <NExpr> is not a compound numeric expression. The parentheses can also be omitted for the LEN function.

The functions provided by the BASIC system are as follows:

SIN	COS	TAN	ATN	SQR	LOG	EXP	ABS	SGN
INT	FRA	RND	NOT	MAN	CHR	IXR	SPC	DET
CHF	LEN	KEY	LKY					

A detailed description of BASIC functions is given in the "BASIC Functions" chapter.

6.2.1.5 Masks

A mask is a string literal or a string variable used to edit the value of numeric expressions in the form of a character string.

Editing by means of masks is possible in the following statements:

- LET
- PRINT
- PRINT #

Whereas for the PRINT and PRINT # statements editing takes place directly into the output buffer, the LET statement requires a destination variable (string variable) to be specified. The following mask characters are used to edit the values:

+ - $ * , . ↑ &

These mask characters each initiate a particular editing function. All characters other than these control characters are taken over one for one, but may only be specified at the beginning or at the end of the mask.

Masks are processed from left to right. The number of characters transferred depends on the number of characters in the mask. If the number of digits in the numeric expression is greater than the number of characters in the mask which can hold digits, only asterisks (*) are transferred to the destination string or the output buffer.

If masks are stored in string variables, these are only processed until a delimiter occurs.

- Example:

```
10 DIM A$(10)              /* MASK FIELD DIMENSIONING
:
50 LET A$="+##,###.##"
:
90 LET X$=A USING A$(2)    /* EDITING WITHOUT SIGN
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Mask characters	Function
#	Transfer of digits. Leading zeros are replaced by blanks.
*	Transfer of digits. Leading zeros are replaced by asterisks (*).
&	Transfer of digits. Leading zeros are output.
+	Transfer of digits with sign. When they occur, "+" signs must be the first character in the mask. The sign of a value explicitly precedes the digits as "+" or "-" (on one of the "+" fields).
-	Transfer of digits with " " sign for positive and "-" for negative values. When they occur, "-" signs must be the first character in the mask.
$	Transfer of digits preceded by a $ sign. Attention: If a value has the same number of digits as the number of mask characters, a) the leading $ sign is suppressed if a sign occurs as the first mask character and b) a $ sign, and otherwise asterisks, are transferred if no sign occurs as a mask character.
↑	Transfer of the exponent in the form E±xx. If an exponent is to be output, exactly four ↑ must occur at the end of the mask and at least one digit character (#, *, &, $, + or -) before (in the case of more digits than digit characters, the mantissa is rounded down). Attention: It is not permissible to specify four ↑ without preceding digit characters.

Mask characters	Function
.	Inserting the decimal point. A decimal point may only occur once in a mask. If the number to be transferred contains more decimal places than the mask has for digits after the decimal point, the last positions are truncated. If a mask has no decimal point, only the integer part of the value is transferred.
,	Insertion of the separator ",". A comma may occur several times in a mask. If a separator occurs and no digit has been transferred before, the current replacement character is transferred instead of the separator. When the replacement characters "*" and "#" are combined, the "," is not allowed as a separator.

Notes:

If the system has been configured with a decimal comma instead of a decimal point, the meanings of "." and "," are not interchanged; however, a decimal comma and the separator "." are actually transferred.

Attention should be paid to the following restrictions when setting the control characters:

- "+" or "-" signs must be located at the beginning of a mask and they are mutually exclusive.
- $ characters may not be located after # , * or &.
- * characters may not be located after # .
- Only the decimal point is allowed as a separator when "*" and "#" are combined.
- The "," character can only be used to separate the integer part.
- & and * are mutually exclusive.
- & and # are mutually exclusive.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

- Examples:

Statement: 50 LET A$=A USING B$

Mask (B$)	Variable A	A$ after LET
Time: ##.##	17.15	Time: 17.15
	1715	Time: **.**
	1.10	Time: 1.10
++++.++	0.20	+.20
+++#.##	0.20	+0.20
-$#####	50.50	$ 50
	-1000	-$1000
	10000	10000
-&&&.&&	-0.20	-000.20

6.2.1.6 String expressions

Syntax:

<SExpr>::= <String> [,<String>]ⁿ

String expressions may only occur in LET statements if a string variable is on the left-hand side of the "=" sign.

A string expression concatenates the contents of several string variables and literals. The string variables are transferred to the destination variable in accordance with their actual length. A delimiter is not stored between the individual string variables and literals which are specified.

- Examples:

```
10 DIM A$(10),B$(5),C$(12)
20 LET A$="ABCDEFG"
30 LET C$=A$,"123456"
40 PRINT C$          ------> Display: ABCDEFG12345

50 LET B$="$",B$
60 PRINT B$          ------> Display: $$$$

70 LET B$="←261→ ←262←←200←←263←",A$
80 PRINT B$          ------> Display: 123A ("←200←" causes
                             nothing to be displayed)
```

6.2.1.7 CLEAR statement

Syntax:

CLEAR <VarName> [,<VarName>]ⁿ

The CLEAR statement clears the values of variables and initializes the variables in accordance with the type.

The value 0 is given to simple numeric variables and to all elements of vectors and matrices. String variables are filled with delimiters throughout their dimensioned length.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.2 DATA statement

Syntax:

DATA [<Format>,] <DecNo> [, <DecNo>]^{n}

Numeric constants are stored in the data area of the BASIC program with the aid of the DATA statement.

At the same time, the precision of these constants can be defined by means of a format specification. This specification can only be made once in a DATA statement and only applies for constants of this one statement. If no specification is made with respect to format, 2% is assumed. A maximum of 63 constants can be specified.
DATA statements are interpreted and constants transferred at the time of loading. Repeated execution of these statements at program run time has, therefore, no effect. The constants from these statements are transferred from left to right. The statements are processed in the order in which they occur.

Note:

The format specification for constants in DATA statements is independent of the format switch (in DIM statements) for numeric variables.

- Example:

  Storing the 1-word constants 10, 20 and 30, as well as the 2-word constants 10000 and 20000.

```
:
980 DATA 1%,10,20,30
990 DATA 10000,20000
```

BASIC Statements

6.2.3 READ statement

Syntax:

READ <NVar> [,<NVar>]^{n}

Constants created by means of the DATA statements are transferred to optional numeric variables with the aid of this statement. The constants are stored in the data area of the partition.

Processing is carried out sequentially. A pointer is kept internally which, before the first READ statement, points to the first constant. The pointer is incremented for each constant transferred and points to the next constant.
This pointer can be reset by means of the RESTOR statement.

- Example:

  Transferring the constants defined in line numbers 30 and 40 to vectors A and B.

```
:
30 DATA 1%,10,20,30,40,50
40 DATA 1%,3,6,9,12,15
:
1000 FOR I=0 TO 4          /*5 CYCLES
1010 READ A(I)             /*TRANSFER CONSTANT
1020 NEXT I                /*END OF LOOP
1030 FOR I=0 TO 4          /*5 CYCLES
1040 READ B(I)             /*TRANSFER CONSTANT
1050 NEXT I                /*END OF LOOP
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.4 RESTOR statement

Syntax:

RESTOR [<LNo>]

The pointer kept internally for the constants created by means of DATA is reset to the first constant with the aid of the RESTOR statement.

If RESTOR is specified with a line number, the pointer points to the first data element of the first DATA statement which occurs after this line number. The READ statement which follows the RESTOR statement commences transfer of the constants from the current pointer position.

- Example:

  Transferring the constants defined in line numbers 30 and 40 to variables. The internal pointer is subsequently reset to the first constant of statement 40, and this is transferred to variable X.

```
:
30 DATA 1,2,5
40 DATA 125,135
:
60 READ A0,A1,A2,B,C
70 RESTOR 40
80 READ X
```

6.2.5 RANDOM statement

The RANDOM statement enables sequences of random numbers to be generated under the control of the RND function. When working with random numbers, the problem arises as to how the program will react when confronted with various sequences which influence the program run. For this reason, it is possible to control the sequence of random numbers by means of the RANDOM statement.

Syntax:

RANDOM <NExpr>

<NExpr>: Function code controlling the sequence of random numbers.

<NExpr>	Effect
0	The sequence of random numbers generated cannot be predicted. A different sequence of numbers is generated for each program run.
<>0	Generation of a particular sequence of random numbers.

Different values lead to different sequences of random numbers. Identical values result in identical sequences.

The RND function without RANDOM has a different effect to a call with RANDOM:

- The same sequence of random numbers is always generated when the program is loaded and executed.
- If the program is executed several times without being reloaded, the sequence of random numbers cannot be predicted.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.6 MAT statement

The MAT statement assigns values to matrices in their entirety.

Matrices are two-dimensional data fields comprising two to n rows and two to n columns. A matrix element corresponds to a numeric variable in its format and can have a length of 1 to 4 words. All elements in a matrix have the same format.

Note:

The rows and columns of a matrix are numbered commencing at 0, transfer in the case of MAT statements from row 1 and column 1, however.

Syntax:

```
MAT <MVar>={ <MVar>
           { <MVar> {+} <MVar>
           {        {-}
           {        {*}
           { (<NExpr>)*<MVar>
           { INV(<MVar>)
           { MIV(<MVar>)
           { TRN(<MVar>)
           { ZER[(<NExpr>[,<NExpr>])]
           { CON[(<NExpr>[,<NExpr>])]
           { IDN[(<NExpr> ,<NExpr> )]
```

6.2.6.1 Assigning a simple matrix variable

Syntax:

MAT <MVar1>= <MVar2>

The elements of the destination matrix are given the value of the corresponding elements of the matrix <MVar2>.

The <MVar1> matrix need not be dimensioned. A dimensioned matrix can be given another dimension; however, the number of elements may not be increased. After the MAT statement has been executed, <MVar1> has the same dimension as <MVar2>.

- Example:

  Transferring the contents of matrix A to matrix B. Matrix A comprises 3 rows and 3 columns. Matrix B comprises 4 rows and 3 columns.

```
:
50 MAT B=A
```

Matrix A	Matrix B before	Matrix B after
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.2.6.2 Matrix addition and subtraction

Syntax:

MAT <MVar1>=<MVar2> $\{\pm\}$ <MVar3>

The elements of the destination matrix are given the value of the sum/difference of the corresponding elements of <MVar2> and <MVar3>. <MVar2> and <MVar3> must have identical dimensions. If <MVar1> has not been dimensioned, it is given the same dimensions as <MVar2> and <MVar3>. If <MVar1> has already been dimensioned, the number of elements must be greater than or equal to the number of elements of <MVar2> and <MVar3>.

The same matrix may be addressed by means of <MVar1> and <MVar2> and/or <MVar3>.

- Example:

  Adding matrix A to itself and assigning it to itself.

```
:
50 MAT A=A+A
```

Matrix A before

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Matrix A after

$$\begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$$

6.2.6.3 Matrix multiplication

Syntax:

MAT <MVar1>=<MVar2>*<MVar3>

Multiplication of matrices does not comprise element-for-element multiplication of the corresponding matrix elements. The scalar product of the I row of <MVar2> multiplied by the J column of <MVar3> is assigned to the (I,J) element of the destination matrix (see also the following example).

The two matrices <MVar2> and <MVar3> need not be square; however, the number of columns of <MVar2> must be equal to the number of rows of <MVar3>. <MVar1> must not be the same as <MVar2> or <MVar3>.

Matrix <MVar1> receives the following dimensions as a result of multiplication:

- Number of rows as in <MVar2>
- Number of columns as in <MVar3>.

If <MVar1> has already been dimensioned, the number of elements in it may not be increased as a result of this.

- Example:

  The matrices B and C are multiplied by one another. The result of this operation is stored in matrix A.

```
10 DIM B(2,3),C(3,2)
:
50 MAT A=B*C
```

Matrix B Matrix C Matrix A

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 3 \end{pmatrix} \quad \begin{pmatrix} 14 & 13 \\ 10 & 11 \end{pmatrix}$$

Calculation of the elements of A is explained by means of the example A (1,2). A (1,2) is the scalar product of the 1st row vector of B and the 2nd column vector C:

$$A(1,2)=\sum_{I=1}^{3} B(1,I)*C(I,2)=1*2+2*1+3*3=13$$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.2.6.4 Scalar multiplication of a matrix

Syntax:

```
MAT <MVar1>=(<NExpr>)*<MVar2>
```

The product of <NExpr> multiplied by the corresponding elements of <MVar2> is assigned to the elements of the matrix <MVar1>.

<MVar1> need not be dimensioned. If <MVar1> has already been dimensioned, the number of elements in it must not be less than the number of elements in <MVar2>.
After execution, <MVar1> has the same dimension as <MVar2>.

<MVar1> and <MVar2> can be identical.

- Example:

  Multiplying matrix A by the value of the variable B.

```
10 DIM A(3,3)
:
50 LET B=3
:
90 MAT A=(B)*A
```

Matrix A before

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Matrix A after

$$\begin{pmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{pmatrix}$$

6.2.6.5 INV function

Syntax:

MAT <MVar1>=INV(<MVar2>)

The INV function inverts a defined square matrix (the product of an inverted matrix multiplied by the matrix itself results in the identity matrix; see also IDN function). The algorithm used is a variant of the Gaussian elimination procedure with row substitution.

The <MVar1> matrix need not be dimensioned. If <MVar1> has been dimensioned, the number of elements in it must at least equal the number of elements in <MVar2>. After assignment, <MVar1> has the same dimension as <MVar2> (<MVar2> must be square).

Note:

There is only an inverse of a square matrix if its determinant is not zero.

- Example:

  The example calculates the associated inverse of a 2x2 matrix and the product of the two matrices.

```
20 DIM A(2,2),B(2,2),C(2,2)
:
50 MAT B=INV(A)
60 MAT C=B*A
```

Matrix A

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} -2 & 1 \\ 1.5 & -.5 \end{pmatrix}$$

Matrix C

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.2.6.6 MIV function

Syntax:

```
MAT <MVar1> = MIV(<MVar2>)
```

The MIV function uses a more involved algorithm to invert those square matrices that cannot be inverted by the INV function although their determinants are not zero.

- Example:

```
20 DIM 2%, A(2,2),B(2,2)
:
50 MAT B=INV(A)
```

Matrix A

1E-30	1E+34
-1E-30	0

Matrix B

9E+29	-9.99999E+28
1E-34	1E-34

Calculation of a matrix element results in an overflow (ERROR # 15) in this case.
A correct result is returned if the INV function is replaced by the MIV function.

```
50 MAT B=MIV(A)
:
```

Matrix A

1E-30	1E+34
-1E-30	0

Matrix B

0	-1E+30
1E-34	1E-34

Attention:

The MIV function overwrites row 0 of <MVar1> with the exception of the first element. If <MVar1> is defined as a 1% variable, the contents of this first element are also overwritten.

6.2.6.7 TRN function

Syntax:

```
MAT <MVar1>=TRN(<MVar2>)
```

The TRN function assigns the transposed matrix of <MVar2> to matrix <MVar1>. The transposed matrix is formed by interchanging the specifications of the rows and columns of each element.

<MVar1> and <MVar2> must be different.

The <MVar1> matrix need not be dimensioned. If <MVar1> has been dimensioned, the number of elements must at least equal the number of elements in <MVar2>.

The transposed matrix <MVar1> has the same dimensions as <MVar2> but the number of rows and columns have been interchanged.

- Example:

```
20 DIM A(2,3)
:
50 MAT B=TRN(A)
```

Matrix A

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.2.6.8 ZER function

Syntax:

MAT <MVar>=ZER [(<NExpr>[,<NExpr>])]

Creation or assignment of a zero matrix. Each element of a zero matrix has the value 0.

An existing matrix can be re-dimensioned with the aid of this statement if the number of rows and columns is specified in parentheses. The number of elements in it may not be increased.

Matrices which are to be created for the first time must be dimensioned by specifying the number of rows and columns.

If only one numeric expression is specified, a matrix with two columns (including column 0) is formed.

- Example:

```
10 DIM A(3,3)
:
50 MAT A=ZER(2,4)
60 MAT B=ZER(3)
```

Matrix A before

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A after

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

6.2.6.9 CON function

Syntax:

MAT <MVar>=CON[(<NExpr>[,<NExpr>])]

Each of the elements of the matrix <MVar> are set to 1.

If the matrix specified has not yet been dimensioned, it must be dimensioned by specifying the number of rows and columns (in parentheses). An existing matrix can be given another dimension in the same manner. The number of elements in it may not be increased.

If only one numeric expression is specified, a matrix with 2 columns (including column 0) is formed.

- Example:

```
20 DIM A(3,3)
:
50 MAT A=CON(2,4)
```

Matrix A before

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A after

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.2.6.10 IDN function

Syntax:

MAT <MVar>=IDN [(<NExpr>,<NExpr>)]

The IDN function creates an identity matrix; an identity matrix is always square, the elements forming the principal diagonal have the value 1. All other elements have the value 0.

<MVar> need not be dimensioned. If <MVar> is to be re-dimensioned or created, the number of rows and columns must be appended to the function word in parentheses.

<MVar> should be or have been dimensioned as a square matrix since an identity matrix is only meaningful in this case. If an existing matrix is to be re-dimensioned, the number of elements may not be increased.

- Example:

```
20 DIM A(3,3)
:
50 MAT A=IDN
```

Matrix A before

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

Matrix A after

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

6.2.7 MAT READ statement

Syntax:

```
MAT READ <MVar>[(<NExpr>[,<NExpr>])]
                    [,<MVar>[(<NExpr>[,<NExpr>])]]^n
```

Transfer of constants from DATA statements to matrices. Constants are transferred to a matrix row by row and from left to right within the rows.

This statement enables a matrix to be dimensioned or re-dimensioned by specifying the number of rows and columns. The number of elements may not be increased when redimensioning is carried out.
If only one numeric expression is specified, the matrix is dimensioned with 2 columns (including column 0).

- Example:

  Transferring constants from the DATA statement in line 50 to matrices A and B. Matrix A is dimensioned for 4 rows and 4 columns in line 30. Matrix B is dimensioned with the MAT READ statement.

```
:
30 DIM A(3,3)
:
50 DATA 2%,5,5,5,8,8,8,3,3,3,1,2,3,4,5,6,7,8
:
90 MAT READ A,B(2,4)
```

Matrix A

$$\begin{pmatrix} 5 & 5 & 5 \\ 8 & 8 & 8 \\ 3 & 3 & 3 \end{pmatrix}$$

Matrix B

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.3 Controlling the program flow

This section describes the statements which control the flow of a program either directly or as a function of conditions.
The statements are as follows:

- GOTO
- GOSUB
- RETURN
- ON
- IF
- IF ERR 0
- IF ESC
- FOR
- NEXT
- CHAIN
- STOP
- END
- LINK

6.3.1 GOTO statement

Syntax:

GOTO <LNo>

This statement causes an unconditional branch in the program flow to the statement which has the specified line number. Execution of the program is continued at this statement.

• Example:

Branching to line number 50.

```
:
100 GOTO 50
```

6.3.2 GOSUB statement

Syntax:

GOSUB <LNo>

The GOSUB statement causes a branch to a subroutine. The line number of the first statement in the subroutine to be executed must be specified.

Due to the fact that BASIC subroutines are constituent parts of the main program, a special parameter transfer is superfluous. All variables which are current when a subroutine is called can be used and changed.

When the GOSUB statement is executed, the subroutine level counter is incremented and the line number of the statement following GOSUB is stored as the return address.
The GOSUB statement should not be the last statement in a program.

Further subroutines can be called within a subroutine. 32 subroutine levels are available.

A subroutine must be terminated by means of a RETURN statement.

- Example:

Calling a subroutine; the first statement to be executed is at line number 9000.

```
:
100 GOSUB 9000
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

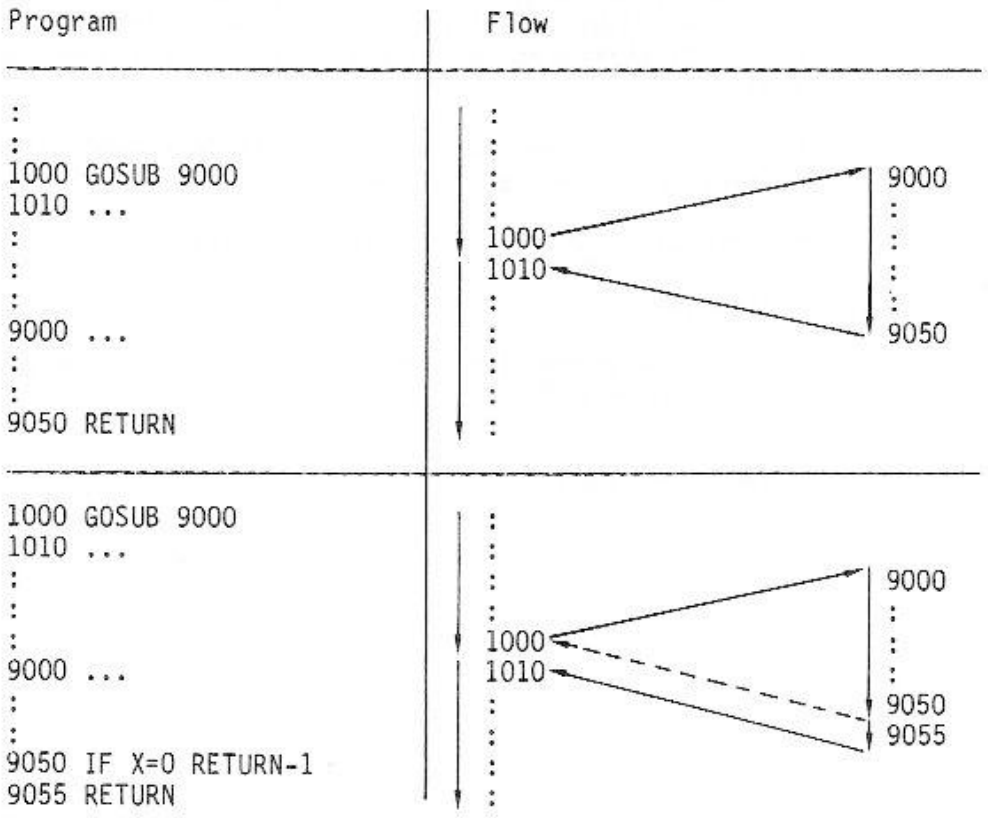
6.3.3 RETURN statement

Syntax:

RETURN [<NExpr>]

Returning from a subroutine branch. Control is returned to the statement which follows the GOSUB statement which called the subroutine.
It is possible to modify the return address by specifying a numeric expression after RETURN. The value of the expression specifies the number of statements by which the return address is to be modified. The line number stored on execution of the GOSUB statement serves as a basis for calculating the return address.

- Examples:



The first example illustrates a subroutine call with an unmodified return branch.

Example 2 shows a return branch to the statement corresponding to the stored return address -1 if the contents of X = 0, i.e. to the subroutine call itself.

6.3.4 ON statement

Syntax:

$$\text{ON } \langle\text{NExpr}\rangle \left\{\begin{matrix}\text{GOTO}\\ \text{GOSUB}\end{matrix}\right\} \langle\text{LNo}\rangle\left[,\langle\text{LNo}\rangle\right]^{n}$$

Multiple branching or subroutine branching facility. The line number of the next statement to be executed is selected from the list of line numbers by means of the value stored in <NExpr>.

The branch can be made by means of the GOTO or GOSUB statement.

The integer part of the value of <NExpr> defines the list element which forms the line number of the next statement to be executed. If the value of <NExpr>is less than 1 or greater than the number of line numbers specified, the program is continued with the next statement.

The length of the list of line numbers does not influence the execution time of the statement.

- Example:

```
:
50 ON X GOSUB 200,300,390
60 ...
```

The statements 200,300,390 form the starting points of one or more different subroutines. The ON statement results in a branch to statement:

- 200 if INT(X)=1
- 300 if INT(X)=2
- 390 if INT(X)=3

When a RETURN statement occurs after the line number concerned, the program continues at statement 60. If INT(X) differs from 1, 2 and 3, statement 60 is executed immediately.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.3.5 IF statement

Syntax:

IF <LogExpr> <Statement>

Conditional execution of a statement. If the value of the logical expression is true, then the next condition is tested or <Statement> is executed. If the value is false, program execution is continued with the statement which follows the IF statement.

Comparison of strings

If only one string expression is specified, there is a check as to whether the first character is a delimiter, i.e. whether it is a null string. If this is not the case, the condition is regarded as satisfied.
When 2 string expressions are compared, comparison goes on until a delimiter occurs. When checking for identity, the string expressions must be identical, including the first delimiter.
Note: A string expression may comprise a maximum of two string variables or literals in an IF statement.

- Example:

  Comparing the contents of the X$ string variable with the combined contents of A$ and B$. If the contents are identical, a branch is made to statement 1440 (GOTO 1440). If they are not identical, the program continues at 1120.

```
:
1110 IF X$=A$,B$ GOTO 1440
1120 ...
```

Comparison of numeric values

If only one numeric expression is specified, there is a check as to whether its value is equal to 0. If this is not the case, the condition is regarded as satisfied.

In the case of two numeric expressions, there is a check as to whether the comparison is true or false and <Statement> is executed or ignored accordingly. When comparing, only the algebraic value is compared, irrespective of the format and the data representation of the expressions to be compared.

If the variable specified is a matrix or a vector, only the elements (0,0) and (0) respectively are compared.

Linking several logical expressions

It is possible to link several conditions by means of a logical "AND" in a statement by specifying the keyword IF several times with a logical expression. All logical expressions must thus be satisfied to enable the statement to be executed.

An "OR" operation is only possible via two separate IF statements.

- Examples:

  - Comparing two string variables for identity. If the condition is satisfied, the program branches to line number 9000.

    ```
    1110 IF A$=B$ GOTO 9000
    ```

  - Checking a string variable as to whether it is a null string. If so, the text "INCORRECT INPUT" appears on the screen.

    ```
    1120 IF A$="" PRINT "INCORRECT INPUT"
    ```

  - Checking a string for the contents "END". If so, a "CHAIN" is made to the TAMOS program "TA.END".

    ```
    1130 IF A$="END" CHAIN "TA.END"
    ```

  - Checking whether A is an integer between 1 and 9. If this is the case, the program branches to line number 9000.

    ```
    1140 IF A>0 IF A<10 IF FRA A=0 GOTO 9000
    ```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.3.6 IF ERR 0 statement

Syntax:

IF ERR 0 [<Statement>]

Intercepting the standard BASIC error indication

ERROR # xx AT <LNo> IN <ProgName>

by means of a specific reaction during program execution. <Statement> is the statement which must be executed when a BASIC error occurs.

In general, this will be a branch to an error handling subroutine in which both the error number and the line number of the statement causing the error can be determined by means of SPC 8 and SPC 10. In this case, the RETURN statement results in a branch being made to the statement following the statement which caused the error.

The BASIC error indication can be cancelled without replacement by means of the statement IF ERR 0 REM or IF ERR 0 /*.

Only one IF ERR 0 statement can be stored during program execution. This means that each IF ERR 0 statement cancels the preceding one.

If BASIC errors are to be displayed in the standard form again, the specific error reaction can be cancelled by means of the statement IF ERR 0 (without <Statement>).

Note 1:

At the time the error is detected, the statement causing this error can already have been partly executed. This can lead to problems particularly in respect of statements containing several operands (e.g. OPEN #, CLOSE #).

Note 2:

The error numbers:

3,6,16,18,19,20,27,31,59,76,77,81,86

cannot be intercepted with the aid of IF ERR 0.

All channels are closed and control is passed to the BASIC processor.

If the program was started from a TAMOS selector, the error number is stored in the common area in the event of the above-mentioned errors and the TA.ABO program is called.

6.3.7 IF ESC statement

Syntax:

IF ESC [<Statement>]

It is not only possible to handle the ESCAPE key via IF ERR 0 and testing error 99 but also directly by means of this statement. When IF ESC <Statement> has been coded and the ESC key or CTL-C combination is pressed, <Statement> is executed.

When IF ERR 0 <Statement> has also been coded, ESC handling is no longer possible in the error routine. If just IF ESC is coded, this ESC key handling procedure is made ineffective again.

Generally, <Statement> will be a GOSUB. If the ESC key is pressed several times, <Statement> is still executed once only, in contrast to IF ERR 0 where the GOSUB stack can overflow!

If a program is called up via CHAIN or LINK, the successor program must contain IF ESC <Statement> if ESCAPE key handling is intended to be used.

GOTO is not allowed as the <Statement>.

REM as the <Statement> suppresses the effect of the ESC key.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Note:

When IF ESC and IF ERR 0 are used in combination, IF ESC should precede IF ERR 0 as ESCAPE might otherwise be intercepted by IF ERR 0 before IF ESC has been executed.

6.3.8 FOR statement

Syntax:

FOR <NVar>=<NExpr1> TO <NExpr2> [STEP <NExpr3>]

Start statement for a program loop.

<NVar> : Control variable.

<NExpr1> : Initial value of the control variable. The value of <NExpr1> is stored in <NVar> when the FOR statement is executed.

<NExpr2> : Final value which, when exceeded in a positive or negative sense, causes an exit from the program loop; the program is continued at the statement following the NEXT statement. The initial value of the control variable is also checked as to whether the final value is exceeded in a positive or negative sense.

<NExpr3> : Step. The value of <NExpr3> is added to the control variable each time the program loop is repeated. If STEP <NExpr3> is not specified, +1 is used as the step.

Note:

Program loops must not overlap but can be nested up to eight times.
<NExpr1> and <NExpr2> can only hold values of up to 6 significant positions as they are stored in 2% format.

6.3.9 NEXT statement

Syntax:

```
NEXT <NVar>
```

The last statement in a program loop. The name of the control variable defined in the associated FOR statement at the beginning of the loop must be specified.

The step defined in the initial FOR statement is added to the control variable in the NEXT statement. When the final value is exceeded in a negative or positive sense, the program loop is finished and the statement following the NEXT statement is executed.
If the final value has not yet been exceeded in a positive or negative sense, a branch is made to the statement following the FOR statement.

- Examples of FOR and NEXT:

  - Ten cycles of a program loop.

```
  :
 ┌1000 FOR I=1 TO 10
 │1005 ...
 │:
 └1100 NEXT I
  1105 ...
```

  - Nested program loop. The outer loop consists of 5 cycles and the inner loop of 10 cycles, making a total of 50.

```
 ┌─1000 FOR I=1 TO 5
 │ 1005 ...
 │ :
 │ ┌1100 FOR M=1 TO 10
 │ │1105 ...
 │ │:
 │ └1200 NEXT M
 │ :
 └─1300 NEXT I
   :
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.3.10 CHAIN statement

Syntax:

CHAIN <String>

Calling an optional BASIC program or processor. The string supplied as a variable or literal must not exceed a length of 127 characters.

The string variable or string literal contains the name of the program/processor which is being called. The program which is calling is terminated and, in addition, all data channels are closed when a processor is called. When a BASIC program is called, channels which are open remain open.

Direct transfer of data from one program segment to another within the partition is not possible by means of the CHAIN statement. The following possibilities exist for transferring data to a subsequent BASIC program:

- Use of the GLOBAL/LINK or DEALLO/LINK statement in place of CHAIN.
- Transfer of data in the common area. A common area with a size of 512 bytes exists for each user. This area can be addressed by means of CALL (see "CALL Statements").
- Transfer of data in disk files. This facility should be used for segment changes in exceptional cases only as it is a relatively time-consuming process.

6.3.11 LINK statement

Syntax:

LINK [;<VarName>,] <String>

The LINK statement calls a BASIC program (segment) and transfers global variables from the calling program to the successor.

There are two variants:

- If the calling program does not contain any GLOBAL statements, all variables dimensioned (with DIM statements or implicitly) which have not been DEALLOcated before the LINK statement are transferred to the successor program. Specification of <VarName> has no effect in this case.

  All variables transferred to the successor must be mentioned there at least once.

- If the calling program contains GLOBAL statements, the variables referenced in these statements are transferred to the successor and must occur there in GLOBAL statements in the same order and with the same precision.

  If <VarName> is specified, only these global variables up to and including <VarName> are transferred.

  Refer to the "Declaration of Data Fields and Functions" chapter.

The name of the program to be called is stored in the string variable or string literal. No more than one program name may be specified.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

Errors:

ERROR	Meaning
44	An illegal file type has occurred.
75	- The file specified is not a BASIC program. - The program is not on the LU specified. This error can be intercepted with IF ERR 0.
76	The capacity of the partition is too small to hold the program (segment). In this case, control remains with the calling program.
77	One or more variables which are to remain available have not been defined in the called program. After the error message, a list of non-defined variables is displayed. The message is stored in the logbook.
81	The program is larger than the active file.
85	Different order of GLOBAL variables.

6.3.12 STOP statement

Syntax:

STOP

The statement interrupts execution of the program with the message:

STOP AT <LNo> STOP

It calls the BASIC processor and the DEBUG mode. The statements following the STOP statement have not yet been executed and no data channels are closed.

The STOP statement is particularly suited for program test runs; after the program has been interrupted by means of the STOP statement, it is possible to continue testing in the DEBUG mode or have the contents of variables displayed in the direct execution mode.

Following the test phase, the STOP statement should be removed from the program.

6.3.13 END statement

Syntax:

END

Execution of a BASIC program is terminated by means of the END statement and all open data channels are closed. The user is then in the BASIC command mode.

If, after the program has finished, another program or processor is to be called, the END statement must be replaced by a CHAIN or LINK statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.4 Input/output statements

The statements which allow input/output on any connected peripherals are as follows:

- INPUT
- PRINT
- MAT INPUT
- MAT PRINT
- OPEN #
- ENV #
- CLOSE #
- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #
- SEARCH #
- BUILD #
- KILL
- CALL

All statements with the exception of CALL are explained in this section. The CALL statements are described in the chapter entitled "CALL Statements".

6.4.1 Keyboard input/screen output statements

The following statements are used for programming workstation input/output:

- INPUT : Input via keyboard
- MAT INPUT : Matrix input via keyboard
- PRINT : Screen output
- MAT PRINT : Matrix output to display

as well as the display functions:

```
<DisFct>::= TAB   (Tabulation, cursor positioning)
            'BS'  (Backspace)
            'MP'  (Mark cursor position)
            'BP'  (Back to marked cursor position)
            'CR'  (Carriage return)
            'LD'  (Line deletion)
            'LI'  (Line insertion)
            'SB'  (Start background)
            'SF'  (Start foreground)
            'CS'  (Clear screen)
            'CF'  (Clear foreground)
            'BEL' (Ring keyboard bell)
            'TB'  (Tab to next foreground field)
            'CFF' (Clear foreground field)
            'CH'  (Cursor home)
            'DRK' (Dark input)
            'X'   (Hexadecimal output)
            'DF'  (Duplicate field)
            'DLC' (Duplicate limited characters)
            'DC'  (Duplicate characters)
            'DM'  (Display memory)
            'CP'  (Cursor position)
            'DW'  (Define window)
            'NTY' (No typewriter keyboard)
            'TY'  (Typewriter keyboard)

"←376←←236←←210←" (Activate 8-bit transfer)
"←376←←236←←211←" (Deactivate 8-bit transfer)
```

The syntax of the device functions (<DevFct>) for programming peripherals can be found in the "Keyboard Input/Screen Output" chapter.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Every display has an indicator (cursor) which is kept internally and which only becomes visible for inputs. Input and output is always made from the current cursor position.

In addition to input and output, the cursor position is changed by the TAB function as well as some other display functions.

6.4.1.1 INPUT statement

Syntax:

```
INPUT [ <SLit>,                          ]n { <NVar> }
      [ <DisFct>,                        ]  { <SVar> }
      [ TAB(<NExpr>[,<NExpr>]),          ]
      [ <NVar>,                          ]
      [ <SVar>,                          ]
```

Keyboard entry of data into optional numeric and string variables. In addition, prompts and display functions can be output.

<SLit> : String literal displayed on the screen as a prompt before the input.

<DisFct> : Any display function (see description of the display functions).

TAB : Tabulation of the cursor (illuminated pointer on the screen) to a particular position. The following can be specified:
- a column in the current line by means of:
  TAB (<NExpr>),
- a column and line by means of:
  TAB (<NExpr>,<NExpr>)

The column specification must range from 0 to 79 and the line specification from 0 to 24.
If only the position within the line is specified (TAB(<NExpr>)), this must be greater than the current position.

<NVar> : Variable which is to receive the input.
<SVar> Several input variables can be specified in one INPUT statement.

BASIC Statements

All operands must always be separated by commas. An input variable must be specified as the last operand.

When an INPUT statement is executed, the program is set to the waiting state, the cursor appears on the screen and the program receives no more CPU time until the input is completed (INPUT mode).
If several variables are to be input, the input of each variable must be concluded by pressing the CR key.

Each display workstation has an input/output buffer with a length of 256 bytes, of which 40 bytes (80 bytes on the DAP 4 workstation) are reserved as a pre-input buffer. This enables up to 40 characters (80 on the DAP 4) and release keys to be input before the INPUT mode occurs. If more characters are entered, an audible signal is emitted and the excess characters entered are not accepted.

As soon as the program is in the INPUT mode, any characters entered are displayed on the screen from the current cursor position onwards. Pre-input characters are displayed at the time of swiching over to the INPUT mode.
If two keys are pressed simultaneously, the acoustic signal is given and neither of the characters is accepted.

If the cursor is in a background field, no characters can be entered. If an input field on the screen is limited by a background character, no input is possible from this position onwards.

The INPUT statement also enables prompts and display functions to be output prior to the actual input. This means that the INPUT statement also undertakes output functions to a certain extent. Prompts cannot, however, be contained in variables; they must be specified as string literals.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Example

  - Output of the prompt "DISCOUNT" from column 0 in line 10.
  - Output of 2 dots to indicate the maximum number of characters to be entered from position 30 onwards.
  - Output of a background blank to limit the input field.
  - Switching to the foreground.
  - Tabulation to the first dot which is output.
  - Input in the variable R.

```
50 INPUT TAB(0,10),"DISCOUNT",TAB(30,10),'MP',"..",
                                          'SB',"%",'SF','BP',R
```

6.4.1.2 MAT INPUT statement

Syntax:

MAT INPUT <MVar>[(<NExpr>[,<NExpr>])]
[,<MVar>[(<NExpr>[,<NExpr>])]]^{n}

This statement enables all the elements of a matrix (without row and column 0) to be entered via the keyboard. Each matrix row must be entered individually (i.e. concluded with CR); the values of the individual elements of a row are separated by commas. A "?" appears on the screen, thus identifying a matrix input.

The matrices to be entered can be dimensioned for the first time or redimensioned with the aid of the MAT INPUT statement by specifying the number of rows and columns; when redimensioning is carried out, the number of elements may not be increased. When just one numeric expression is specified in parentheses after the matrix variable, a matrix with two columns (including column 0) is dimensioned.

- Example:

  Input of values into matrix A. The matrix is dimensioned with 2 rows and 4 columns.

```
:
40 MAT INPUT A(2,4)
```

6.4.1.3 PRINT statement

Syntax:

$$\left\{\begin{matrix}\text{PRINT}\\;\end{matrix}\right\}\ [\text{USING}\quad \langle\text{String}\rangle;]\ \left[\begin{matrix}\langle\text{NExpr}\rangle\\\langle\text{String}\rangle\\\langle\text{DisFct}\rangle\end{matrix}\right]\left[\left\{\begin{matrix};\\,\end{matrix}\right\}\left\{\begin{matrix}\langle\text{NExpr}\rangle\\\langle\text{String}\rangle\\\langle\text{DisFct}\rangle\end{matrix}\right\}\right]^{n}\ \left[\begin{matrix};\\,\end{matrix}\right]$$

PRINT enables

- the contents of string variables
- the value of numeric expressions
- numeric and alphanumeric constants
- display functions

to be output in any order on the screen.

PRINT : Keyword.
; When the statement is input, a semi-colon (;) can be specified instead of PRINT.

USING <String> : This specifies an editing mask which controls the editing of all numeric variables to be output with this statement. The mask is specified in <String>.

USING may only be specified once in a PRINT statement immediately following the keyword, and it must be separated from the following operands by a semi-colon.

The format of masks and the functions of the available mask characters are described in the section entitled "Masks".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Following this, any sequence of

<NExpr> = Numeric expressions
<String> = String variables or string literals
<DisFct> = Display functions

can be specified, and these must be separated by commas or semi-colons.

Output of numeric values

If the output of floating-point or fixed-point numbers is not controlled via masks, this takes place from the current cursor position onwards in the following standard format:

- Sign (1 character); a blank " " is output for plus and "-" for minus.
- Digits of the value, including the decimal point and exponent (for floating-point numbers). The number of characters is determined by the precision of the value to be output. Leading zeros are suppressed.
- A blank " " is appended to the last digit output.

Numeric values are output (without USING specification) in floating-point format if the value is

- greater than 99999999999999
- less than 0.1.

• Examples: The value 0.01 is output as 1E-02.

The value 999999999999999 is output as
9.9999999999999E+14.

Output of strings

Strings are always output from the current cursor position onwards without a leading or trailing blank.

A comma "," and a semi-colon ";" act as separators between and after the operands.

BASIC Statements

Comma separator ","

After the preceding operand has been output, blanks are output until the next greater column position divisible by 15 is reached and the cursor is set to this position.
If this value is greater than 79, a line feed is performed.

Semi-colon separator ";"

After the preceding operand has been processed, the cursor remains at the current position.
After the last operand in the PRINT statement has been output, cursor positioning is carried out in accordance with the same rules.
If neither a comma nor a semi-colon is coded after the final operand, the cursor is positioned at the start of the next line.
An input/output buffer with a length of 256 bytes is allocated to each display workstation in the system. Outputs on the screen do not take place until:

- a program running simultaneously receives its time slice,
- the output buffer is full,
- a PRINT # statement,
- a SIGNAL 3 statement,
- an INPUT statement,
- a CHAIN statement,
- a LINK statement is executed or
- the end of program is reached.

If a character is output in column 79 of line 24, the entire contents of the screen are shifted one line upwards. The data in line 0 is lost and line 24 becomes available.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.4.1.4 MAT PRINT statement

Syntax:

MAT PRINT <MVar>[,|;]

The MAT PRINT statement produces a display of entire matrices (without row and column 0) on the screen. The output appears in rows and from left to right within the rows themselves. Only one matrix can be output with a single statement. The statement always operates with a decimal point.

The final semi-colon or comma determines the format of the separation between the elements of the matrix. If neither of the two characters is specified, a comma is the default value.

, Output of the next element of a row commences at the next position divisible by 15. If 75 is exceeded here, a new screen line is begun.

; Output of the next element immediately follows the preceding element (a sign before the value and a blank following the value are taken into account).

6.4.2 Logical IOCS statements

6.4.2.1 OPEN # statement

Syntax:

OPEN # <NExpr> [;<String1>],<String2>
[[,#<NExpr>[;<String1>]], <String2>]^n

Opening disk files or device channel programs (drivers) on a data channel.

OPEN # : Keyword

<NExpr> : Number of the data channel on which the file or driver is to be opened. The channels are numbered from 0 onwards.
Channels 0 and 1 are occupied when working under TAMOS.

<String2> : Name of the file or driver to be opened. <String2> must contain <FileName> or <DriverName>. It is only necessary to specify the channel number for the first file to be opened with an OPEN # statement. If several files have to be opened on successive channels with one OPEN # statement, it is only necessary to specify the channel number for the first file.

- Example:

```
10 OPEN #2,"$LPT",A$,B$,C$
```

The files (drivers) which are not preceded by a channel number are opened on successive channels (in the example: $LPT on channel 2, A$ on channel 3, B$ on channel 4 and C$ on channel 5).

<String1> : When opening disk files and printers as well as other peripherals, parameters can be defined here for processing the following file (driver).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Disk files:

"L" as the first character in the string means that the file in question can only be accessed after it has been closed by the present user. An OPEN # statement on a locked file results in the following message being issued:

ERROR # 50 AT <LNo> IN <ProgName>

"R" as the first character in the string means that the following error message is issued when a locked record is accessed in the file

ERROR # 123 AT <LNo> IN <ProgName>

When "R" is not present, the program is deactivated until the locked record is unlocked.

"C" as the first character in the string causes a contiguous file to be cleared up to the highest written data record and the EOF pointer to be set to -1. The parameter "C" is ignored for other types of file. If the file concerned is already open, the following error message appears:

ERROR # 50 AT <LNo> IN <ProgName>

The three parameters "L", "R" and "C" can be combined in any sequence.

P Allows locked records to be read. If attempted to update generates BASIC ERROR 46.

- Card reader files:

  The number of columns (1-80) of the cards which are to be processed can be specified in the first two characters of the string.

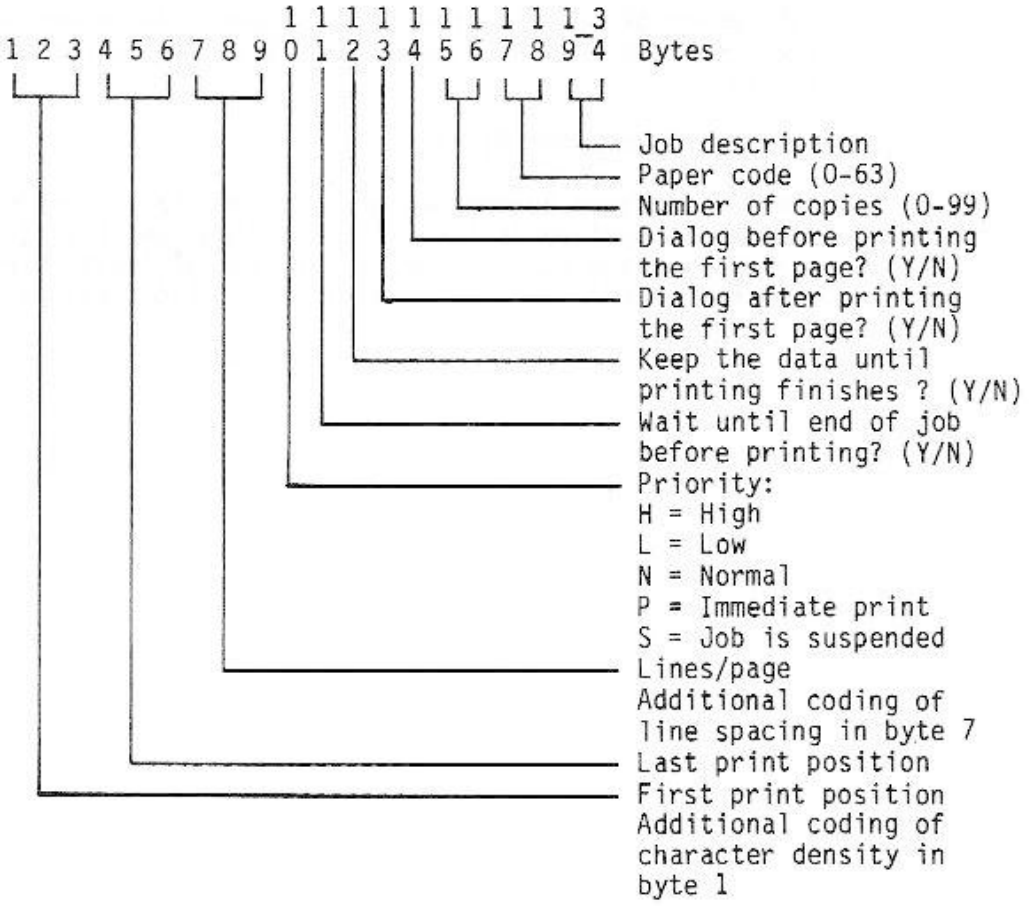
- Printer and plotter files:

  The form control parameters and, if required, spooling control parameters must be defined in the string. In this case, a string variable must be dimensioned for at least 34 bytes. It must be ensured that this string is concluded with a delimiter.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

Format:



The printing areas are not allowed to overlap on needle printers with two paper feeders.

Specification of lower-case letters for the priority (h, l, n, p or s) protects the print job against deletion.

The following default values apply if no specifications are made:

Channel program (driver)	First print position	Last print position	Page length
$LPT	000	176	048
$LPTR	177	177	048
$LPT1	000	135	048
$LPT1S	000	135	048
$LPTS	000	176	048
$LPTRS	177	177	048
$RPLO-15	000	131	048
$RPRO-15	145	177	048
$RPFO-15	000	131	062
$LCP	000	131	048
$LCPS	000	131	048
$ORDP	000	135/*131	048
$ORDPS	000	135/*131	048
$HDD	000	131/*157	048
$HDDS	000	131/*157	048
$HDDSAS	000	131	048
$HDDSASS	000	131	048
$ORDPSR	000	131/*157	048
$ORDPSRS	000	131/*157	048

* Adjustable by means of code table in the driver

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

- Printer names :

$LPT	Needle printer, left feeder
$LPTR	Needle printer, right feeder
$LPT1	Line printer (high-speed printer)
$LPT1S	Second line printer
$LPT2	Printer for USA only
$LPT2S	Second printer (for USA only)
$LPTS	Second needle printer, left feeder
$LPTRS	Second needle printer, right feeder
$RPL0-15	Workstation printer 0-15, left feeder
$RPR0-15	Workstation printer 0-15, right feeder
$RPS0-15	Workstation printer 0-15, form chute
$LCP	Compact printer
$LCPS	Second compact printer
$ORDP	Band printer
$ORDPS	Second band printer
$HDD	System compact printer
$HDDS	Second system compact printer
$HDDSAS	System compact printer via SAS
$HDDSASS	Second system compact printer via SAS
$ORDPSR	Band printer via PSR
$ORDPSRS	Second band printer via PSR

- Other parameters:

Priority	= N
Wait until end of job before printing	= N
Keep data until printing finishes	= N
Dialog after printing the first page	= N
Dialog before printing the first page	= N
Number of copies	= 00
Paper code	= 00

- Job description

If no job description is specified, the third-level selector name is taken if the OPEN comes from a BASIC program and the processor name is taken if the OPEN comes from a processor (e.g. LIBR).
This specification is read from the common area; for this reason, it can have been overwritten by an application program.

Any name can be entered as the printer name (only condition: first character must be a "$" sign; e.g. $LPT). The printer assignment table - which can be changed with the TAMOS program DEVICE ASSIGNMENT - is scanned for an entry for the workstation concerned. If there is an entry for the specified printer name, the printer name is replaced by the appropriate entry (printer driver name).
The parameters are used according to the following priorities:

First/last print position	Lines per page
1. Printer assignment table 2. OPEN # statement 3. Default values	1. OPEN # statement 2. Printer assignment table 3. Default values

- Character density:

Value to be added to the contents of byte 1	Meaning
0	Default value (see "Printers" chapter)
2	10 characters per inch
4	12 characters per inch
6	15 characters per inch
8	16.5 characters per inch

- Example:

  Value in bytes 1-3 : 500
  Meaning: 12 characters per inch and 100 as the first print position

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Line spacing:

Value to be added to the contents of byte 7	Meaning
0	Default value = 6 lines/inch
2	Reserved
4	Reserved
6	6 lines/inch
8	8 lines/inch

- Example:

  Value in bytes 7-9 : 848
  Meaning : 8 lines/inch and page length = 48

- Paper tape files:

  The name of a code or decode table can be specified at this point for the tape punch/reader in the form of a string variable or literal. For further details of this see the chapter entitled "Paper Tape Reader/Punch".

6.4.2.2 ENV # statement

The ENV # statement is needed for programming

- data communications via the 1-channel PLC and
- magnetic tape functions (peripheral driver interface).

It transfers specifications that are needed to handle a data communication line in addition to the statements:

- OPEN #
- WRITE #
- READ #

The number and meaning of parameters are defined by the ENV # function number.

Syntax:

ENV # <NExpr>;<NVar1> $\left[\begin{matrix}\text{,<NVar>}\\\text{,<SVar>}\end{matrix}\right]^{5}$

<NExpr> : Number of the channel on which the line was opened.

<NVar1> : Function number

<NVar> : Parameters, the number, size and meaning of which
<SVar> are described for the individual functions.

In the event of an error, an error code can be returned with the ENV # statement to provide a differentiated analysis of the error.

There are also functions for testing and statistical purposes. The individual functions are distinguished by the function number <NVar1>.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

Overview of ENV # functions (tape interface)

Function	Meaning
0	Transfer code tables
1	Status test
2	Rewind
3	Rewind and switch off
4	Write tape mark
5	Forward-space by n blocks
6	Backspace by n blocks
7	Forward-space by n tape marks
8	Backspace by n tape marks

An exact description of these functions can be found in the "Magnetic Tapes" chapter of this manual. The ENV # functions for programming the 1-channel PLC are listed in the Nixdorf 8870 Data Communication manual.

6.4.2.3 READ # statement

Syntax:

READ # <NExpr1>[,<NExpr2>[,<NExpr3>]];
{<NVar>}[,<NVar>]^n [;]
{<SVar>}[,<SVar>]

Reading data from disk files or device drivers and transferring the data which has been read to optional variables.

<NExpr1> : Number of the data channel on which the file (driver) is open.

<NExpr2> : Number of the record to be read, relative to the beginning of the file (record number 0).

For text file processing, the number of the block (sector) must be specified relative to the beginning of the file (block 0).

If <NExpr2> is not specified, the file is processed sequentially; this also applies if <NExpr2> equals -1. If <NExpr2> equals -2, the record is read which was last read or written on this channel. Text files are excluded from this.

The specification <NExpr2> is of no significance when peripheral units are being processed.

<NExpr3> : Start address within a data record. This specification is only permitted if <NExpr2> is coded.

In the case of formatted files, the number of the field is specified relative to the start of the data record (field 0).

For all other disk files, the address of the byte to be transferred first of all must be specified relative to the start of the data record (byte 0).

The specification <NExpr3> is of no significance when peripheral units are being processed.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

<NVar> : Destination variables.
<SVar> The contents of the data record addressed are transferred to these variables.

The format of the data to be read is only verified for formatted files. With contiguous files, transfer takes place irrespective of the type and format of the destination variables. Data may not be read from text files to numeric variables.

The number of characters to be transferred to the destination variables is dependent on the following:

- The dimensioned format (1-4 words) in the case of numeric variables.
- The occurrence of a delimiter (in text files the 'CR' code also) in alphanumeric data.
- Reaching of the length dimensioned or specified by indices in the case of string variables.

Attention:

Transfer of characters to a string variable is always concluded with a delimiter even if a substring is involved.

; : "No record lock" identifier.
If no semi-colon ";" is coded after the last operand, the record addressed with this statement is protected against access by other users (with disk files only). If, on the other hand, a semi-colon has been coded, there is no record lock.

Specifications for addressing the data to be read (<NExpr1>, <NExpr2> and <NExpr3>) must be separated from the names of the destination variables by means of a semicolon ";".

A string variable is the only permissible destination variable for card reader files, since only alphanumeric data is supplied.

- Examples:

  - 450 READ #C;X,Y,A$;

    Statement for sequential reading. The channel number is specified in the variable C. Data is transferred to the variables X, Y and A$.

  - 450 READ #C;X,Y,A$

    Identical statement with lock of the data record which has been read.

  - 480 READ #C,R,50;A$;

    Random reading in a contiguous file. The channel number is specified in the variable C and the record number in the variable R. Data is transferred to the destination variable A$ from byte 50 of the record addressed.
    The record is not locked.

6.4.2.4 WRITE # statement

Syntax:

```
WRITE # <NExpr1>[,<NExpr2>[,<NExpr3>]];
        {<NExpr4>}[,<NExpr4>]^n  [;]
        {<String>}[,<String>]
```

Writing data to disk files and to peripheral units via device drivers (e.g. printer and line drivers).

<NExpr1> : Number of the data channel on which the file (driver) is open.

<NExpr2> : Number of the record to be written, relative to the beginning of the file (record 0).
When processing text files, the number of the block (sector) must be specified relative to the beginning of the file (block 0).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

If <NExpr2> is not specified, the file is processed sequentially; this also applies if the value of <NExpr2> equals -1.
If the value of <NExpr2> equals -2, the record is written which was last read or written on this channel (with the exception of text files). This enables a data record which has been read and updated to be written back.
The specification <NExpr2> is of no significance when peripheral units are being processed.

<NExpr3> : Start address within the data record addressed. This specification is only permissible if <NExpr2> has been coded.

In the case of formatted files, <NExpr3> is the number of the field relative to the beginning of the data record (field 0), starting from which the data is to be transferred to the record addressed.

The address of the first byte to be written must be specified relative to the start of the data record (byte 0) for all other disk files.

<NExpr3> is of no significance when peripheral units are being processed.

<NExpr4> : Any number of
<String> - numeric expressions,
- string variables and
- string literals
to be output in any order. It is not permissible to specify numeric expressions in the case of printer and text files.

; : "No record lock" identifier. If a semi-colon ";" is not coded after the last operand, the record addressed by this statement is protected from access by other users; if it is not coded, there is no record lock.

This specification is of no significance when peripheral units are being processed.

By specifying WRITE # <NExpr1> ;; the record lock can be cancelled again.

The format of the data to be transferred is only verified in the case of formatted files (not when <NExpr3> = 1). Exactly one output operand is transferred to each field from the addressed field onwards.
In all other instances, data is transferred identically.

A delimiter is stored after string literals and variables when no indices are specified, but not when substrings are output if the length of the substring is actually less than the the dimensioned length of the string. If a delimiter is reached prior to the end of the substring, a delimiter is stored and transfer is terminated.
If an alphanumeric field is not written completely in a formatted file, a delimiter is always stored after the transfer.

The number of characters written to the data record of a contiguous or indexed file are defined by the following criteria:

- The dimensioned format (1-4 words) in the case of numeric variables, 4 words in the case of numeric expressions
- The occurrence of a delimiter in strings
- The length dimensioned or specified by indices in the case of string variables
- The length of string literals

When several variables are written to contiguous or indexed files, the following must be noted:

- When string variables without indices are output, writing is only up to the first delimiter but output of the next operand commences after

  dimensioned length of string variable + 1

  bytes (relative to the first character of the variable).
- Numeric values are always written from the start of a word.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

The specifications for addressing the data to be written (<NExpr1>, <NExpr2> and <NExpr3> must be separated from the names of the variables to be written by a semi-colon ";".

If a delimiter is the first character output to printer, the print buffer is cleared completely before the channel is closed. (WRITE # C;"";)

- Examples:

  - 450 WRITE # C,R+S,5;A;

    Random writing of the contents of the variable A to field 5 of a formatted file. The channel number is specified in variable C. The record is calculated from R+S. The record is not locked.

  - :
    ```
    310 READ # C;X,Y,A$
    320 GOSUB 2000            /* UPDATE DATA
    330 WRITE # C,-2;X,Y,A$;
    340 GOTO 310
    ```

    Sequential reading, updating and writing back in a contiguous file. The channel number is specified in variable C. The variables X, Y and A$ are processed. After reading has been completed, the record concerned is locked and not unlocked until writing has been completed.

6.4.2.5 PRINT # statement

Syntax:

```
{PRINT} # <NExpr1> [,<NExpr2> [,<NExpr3>] ];
{;    }

[USING <String1>;] [<NExpr4>  ][{,} {<NExpr4>  }]n [,]
                   [<String>  ][{;} {<String>  }]  [;]
                   [TAB(<NExpr>)]  [TAB(<NExpr>)]
                   [<DevFct>  ]    [<DevFct>  ]
```

Transferring data to disk files or peripheral units via device drivers (e.g. printers). Numeric values are converted to alphanumeric data (ASCII) with this statement, unlike the faster WRITE # statement.

The keyword PRINT can be replaced by ; on input except in the case of conditional statements.

<NExpr1> : Number of the data channel on which the file is open. If the channel specified has not been opened, all outputs are displayed without an error message being output.

<NExpr2> : Number of the data record to be written, relative to the beginning of the file (record 0).

The number of the block (sector) must be specified relative to the beginning of the file (block 0) when processing text files.

If <NExpr2> is not specified, the file is processed sequentially; this also applies when the value of <NExpr2> equals -1.
If the value of <NExpr2> equals -2, the record is written which was last read or written in this file (exception: text files).

<NExpr2> is of no significance when peripheral units are being processed.

<NExpr3> : Start address within the data record addressed. This specification is only admissible if <NExpr2> is coded.

For formatted files, the number of the field, starting from which transfer of data is to begin, is specified relative to the beginning of the data record (field 0) with the aid of <NExpr3>.
Only alphanumeric fields can be processed in formatted files.

For all other files, <NExpr3> is the address of the byte relative to the start of the data record (byte 0), starting from which the data is written.
It must be noted that ASCII is always output by means of PRINT # . Numeric values are converted automatically to ASCII prior to output.
When PRINT # is used to write directly, only strings with a length of up to 250 characters are allowed to be specified. Longer strings require a WRITE # statement.
<NExpr3> is of no significance when peripheral units are being processed.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

USING <String1> : This specifies an editing mask which controls the conversion of all numeric values to be output with the aid of this statement. The mask is specified in <String>.

USING may only occur once in a PRINT # statement and must be separated from the next operand by a semicolon ";".
The structure of masks and the functions of the available mask characters are described in the chapter entitled "Masks".

<NExpr4>
<String> : - Numeric expressions,
- string literals and string variables whose value or contents are to be output in any order.

TAB(<NExpr>) : For each data channel, the system (internally) keeps a pointer which contains the number of characters output since the last 'CR' code (215 octal) or since the output file was opened. The pointer thus indicates the last byte written +1. The TAB(<NExpr>) specification results in the output of blanks starting from this position up to the position <NExpr> -1 and stores the value of <NExpr> in the pointer. The value of <NExpr> may not be less than the current pointer position.
The TAB pointer is set to 0 when:

- a 'CR' code occurs,
- a statement has been completely processed and no semi-colon is coded behind the last operand (then PRINT # generates the 'CR' code).

The specifications for addressing the data to be written (<NExpr1>, <NExpr2> and <NExpr3>) must be separated by a semicolon ";" from the next operand.

Operands, of which the contents are to be output, must be separated by commas (,) or semi-colons (;).

<DevFct> : Device functions can be specified at this point to control peripherals, e.g. 'ELO' for elongated printing. Other device functions are listed in the "Forms Control" chapter.

Separator ",": After the contents of the preceding variable have been output, blanks are output until the current TAB position can be divided by 15 without leaving a remainder.

Separator ";": The contents of the variable following the ";" are output right next to the last output.

In the case of disk files, the PRINT # statement always sets a record lock for the written data record and a delimiter after the last character transferred.
If a delimiter is output as the first character on printers, the print buffer is cleared completely before the channel is closed (PRINT # C; "";).

Final $\left\{\begin{matrix};\\,\end{matrix}\right\}$: PRINT # always stores a delimiter after the last output operand transferred. If no character has been coded after the last operand, an additional line feed code (215 octal) is output prior to the delimiter. If a semi-colon is coded, this output is omitted. If a comma is specified, blanks are output before the delimiter up to but excluding the next position divisible by 15.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

It must be noted that PRINT # always outputs:

- ASCII and
- a delimiter after the last output operand.

Numeric values are converted to ASCII prior to output.
Floating-point and fixed-point numbers are edited via the following standard mask if USING is not specified:

- Sign (1 character); a blank is output for "+" and a minus sign for "-".
- Digits, decimal point and exponent (in the case of floating-point numbers) are output after the sign. Leading zeros are suppressed.
- A blank is output after the digit which was last output.

If USING is not coded, a number is output in floating-point format when its value is

- greater than 99999999999999 or
- less than 0.1.

● Example:

```
490 PRINT # C; USING "####";TAB(10);K$;TAB(70);"PAGE: ";A
```

Output of the contents of the variable K$ at position 10. Output of the text "PAGE: " from position 70 and subsequent transfer of the contents of variable A to the mask ####. A 'CR' code (215 octal) is then output. Output is to the file which is open on the channel specified in variable C.

For printouts, PRINT # statements should not, if possible, contain any TAB specifications because any errors which may occur with this statement cannot be intercepted with SPC8 or RETURN-1 without shifting print starting positions.

6.4.2.6 MAT READ # statement

Syntax:

MAT READ # <NExpr1> [,<NExpr2> [,<NExpr3>]]; <VarName> [;]

Reading data from formatted, contiguous and indexed files and transferring the data to:

- a matrix,
- a vector,
- a numeric variable or
- a string variable.

<NExpr1> : Number of the data channel on which the file is open.

<NExpr2> : Number of the record to be read, relative to the beginning of the file (record number 0).

If <NExpr2> is not specified, the file is processed sequentially; this also applies if the value of <NExpr2> is -1.
If the value of <NExpr2> is -2, the record is read which was last read or written in this file.

<NExpr3> : Start address within a data record. This specification is only permissible if <NExpr2> is coded.

With formatted files, the number of the field is specified relative to the start of the data record (field 0). Specifying -1 as the field number for formatted files permits processing without a field format check. Transfer starts at the first byte in the addressed record and finishes with the last one or when the dimensioned length of the destination variable is reached.

In the case of all other disk files, the address of the first byte to be transferred must be specified relative to the beginning of the data record (byte 0). If the byte address is odd, an increment is made internally to the next higher even-numbered address due to the system's word-oriented transfer (1 word = 2 bytes).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

<VarName> : Numeric variable, matrix, vector or string variable to which the addressed data record has to be transferred. Only one variable name without indices may be specified. The variable must have been dimensioned already.
The data is transferred until:

- the end of the destination variable or
- the end of the file is reached.

Any delimiters in strings are thus included in the transfer.
When processing a vector or a matrix, the variable is transferred in its entirety, including row and column 0.

; : "No record lock" identifier.
If a semi-colon ";" is not coded after the last operand, the record addressed with this statement is protected from access by other users.
If, on the other hand, a semi-colon is coded there is no record lock.
The record lock only affects the addressed record even if several data records are read with one statement.

The specifications for addressing the data to be read (<NExpr1>, <NExpr2> and <NExpr3>) must be separated from the name of the destination variable by a semi-colon.

- Examples:

  - 20 DIM A(3,3)
    :
    50 MAT READ # C,R;A;

    Reading the matrix A in its entirety from the record whose number is stored in the variable R. The number of the channel on which the file is open is specified in the variable C.

  - 20 DIM A$(512)
    :
    50 MAT READ # C;A$;

    Transferring 512 bytes to the variable A$. The data is read from the record following the record which was last processed in this file. The number of the channel on which the file was opened is specified in the variable C.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.4.2.7 MAT WRITE # statement

Syntax:

MAT WRITE # <NExpr1>[,<NExpr2>[,<NExpr3>]];<VarName> [;]

Transferring data from a string variable, a numeric variable, a vector or a matrix to a disk file (exception: text file).

<NExpr1> : Number of the data channel on which the file is open.

<NExpr2> : Number of the data record to be written, relative to the beginning of the file (record number 0).

If <NExpr2> is not specified, the file is processed sequentially; the same applies if the value of <NExpr2> is -1.

If the value of <NExpr2> is -2, the record is written which was last read or written on this channel.

<NExpr3> : Start address within a data record. This specification is only permissible if <NExpr2> is coded.

With formatted files, the number of the field is specified relative to the start of the data record (field 0). Specifying -1 as the field number for formatted files permits processing without a field format check. Transfer then starts at the first byte of the addressed record.

In the case of all other disk files, the address of the first byte to be transferred must be specified relative to the start of the data record (byte 0). If the byte address is odd, an increment is made internally to the next higher even-numbered address due to the system's word-oriented transfer (1 word = 2 bytes).

<VarName> : Numeric variable, matrix, vector or string variable of which the contents are to be transferred to the data record addressed.

Only one variable name without indices may be specified. The variable must have been dimensioned already.

; : "No record lock" identifier.
If a semi-colon ";" is not coded after the last operand, the record addressed with this statement is protected from access by other users.
If, on the other hand, a semi-colon is coded, there is no record lock.
The record lock only affects the addressed record even if several data records are written with one statement.

Data, including any delimiters, is transferred until:

- the dimensioned length of the variable has been output,
- the end of the file has been reached (contiguous files only),
- the disk capacity is exhausted.

No delimiter is set after a string variable has been transferred if the dimensioned length of the string variable is an even number. Since MAT WRITE # involves a word-oriented transfer, a delimiter is added if an odd number is dimensioned.
If a vector or a matrix is being processed, the element 0 or row and column 0 is also transferred. The variable can be written to several records.

The specifications for addressing the data to be written (<NExpr1>, <NExpr2> and <NExpr3>) must be separated from the name of the variables by a semi-colon.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

- Examples:

  - 20 DIM A$(512)
    :
    50 MAT WRITE # C,-2;A$;

    Writing back 512 bytes from the variable A$ commencing from byte 0 of the last record processed in this file. The number of the channel on which the file is open is stored in the variable C.

  - 20 DIM A(3,3)
    :
    50 MAT WRITE # C,0,X;A;

    Writing the matrix A in its entirety to record 0 starting from the byte address specified in the variable X. The number of the channel on which the file is open is stored in the variable C.

6.4.2.8 SEARCH # statement

Processing of key directories in an indexed file.

Syntax:

```
SEARCH # <NExpr1>,<NExpr2>,<NExpr3>;
                          <SVar>,<NVar1>,<NVar2> [:|;]
```

<NExpr1> : Number of the data channel on which the file is open.

<NExpr2> : Mode; defines the function to be executed.

<NExpr3> : Number of the key directory to be processed (1-15). The specification 0 is only permissible if <NExpr2> is 0 or 1.

<SVar> : String variable in which the key is specified or transferred by the IOCS. <SVar> must be dimensioned for at least the same length as the key.

<NVar1> : Numeric variable, at least 2%, used to transfer the relative record number to the application program or to the IOCS.

<NVar2> : Numeric variable for transferring parameters to the operating system and for transferring status messages to the application program.

: : Setting a record lock for modes 1,2,3 and 5.

; : No record lock (also without ";" or ":").

The functions of SEARCH #

<NExpr2>	Function
0	<NExpr3> <> 0: Defining the directory <NExpr3> with the key length specified in <NVar1>. The directory numbers must be specified in <NExpr3> in ascending order, commencing at 1. <NExpr3> = 0: Initializing all directories which were created with SEARCH mode 0, <NExpr3> <> 0. The number of data records must be supplied in <NVar1>.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

<NExpr2>	Function
1	<NExpr3> <> 0: This function returns in <NVar1> the key length of the directory specified in <NExpr3>. If the directory does not exist <NVar2> is set to 5. <NExpr3> = 0 and <NVar2>= 0: This function returns in <NVar1> the relative record number of the first data record in the data area. <NExpr3> = 0 and <NVar2> = 1: This function returns in <NVar1> the number of free data records. <NExpr3> = 0 and <NVar2> = 2: This function returns in <NVar1> the relative record number of the first free data record, deletes this record from the list of free data records and sets a record lock when ":" is coded. <NExpr3> = 0 and <NVar2> = 3: The data record addressed in <NVar1> is entered in the list of free data records. The data record is only deallocated when no other user has locked the record.
2	This function searches the specified directory for the key specified in <SVar>. The number of characters compared is defined by the actual length of <SVar> (this may therefore not exceed the key length). Identity is recognized if all characters of <SVar> match the key. Identity can therefore also be recognized if the key specified in <SVar> is shorter than that stored in the directory. The process of comparing only up to the first delimiter which occurs enables partial keys to be specified. In cases of identity, the entire key from the directory is returned in <SVar>. The relative record number of the associated data record is returned in <NVar1> and the record is locked, if so defined. If no match is detected <SVar> and <NVar1> remain unchanged and status 1 is returned in <NVar2>.

<NExpr2>	Function
3	The specified directory is searched for the next higher key than the one specified in <SVar>. If a higher key is found, this is returned in <SVar>. The relative record number of the associated data record is returned in <NVar1> and the record is locked if ":" is coded. If there is no higher key, <SVar> and <NVar1> remain unchanged. The value 2 is returned in <NVar2> as a status message.
4	This function inserts the key specified in <SVar> in the specified directory; if necessary, the directory is reorganized. The key must not already exist in the directory specified. Identity is recognized if <SVar> including delimiters matches the key in the directory. The relative record number of the data record to which the key should point must be specified in <NVar1>. If the key already exists in the specified directory, the record number of the associated data record is returned in <NVar1> and status 1 is returned in <NVar2>.
5	The key specified in <SVar> is deleted from the specified directory. The relative record number of the associated data record is returned in <NVar1> and this must be deallocated separately with the aid of SEARCH mode 1. The method of comparing keys corresponds to that used when searching for a key. If the specified key does not exist in the directory, <SVar> and <NVar1> remain unchanged and status 1 is returned in <NVar2>.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

<NExpr2>	Function
6	Directories are deleted and the data records entered in the list of free records, i.e. they are re-concatenated. When this statement is executed, the file is in the same state as when it was created. The contents of <NExpr3>, <SVar>, <NVar1> and <NVar2> are of no significance with this function. The function does not let other users have any CPU time. If the file has been opened by another user, status 15 is reported. After this function has been performed successfully, the number of free data records is returned in <NVar1>.
8	Mode 8 is similiar to mode 3. If "R" has been defined at OPEN time so that a record lock is generated and a locked record is accessed, BASIC error 123 is output. The next higher key is output in <SVar> and the associated record number in <NVar1>. The status variable <NVar2> is set to "0".

9. Initiates a reorganisation
NEXPR1 = channel.
NEXPR2 = 9
NEXPR3 = directory to reorganise.
SVAR1 } Not evaluated
NVAR1 }
NVAR2 Status.

10. Delete one directory.
similar to 6 but only delete 1 directory

Status messages of SEARCH #

The status variable <NVar2> must be checked after each SEARCH # statement has been performed.
The following status messages can be stored in <NVar2> after the various functions of SEARCH # have been performed.

<NVar2>	Meaning
0	Function performed without error.
1	Function not performed successfully.
2	End of directory reached.
3	No free data record available.
4	Not an indexed file.
5	Negative record number, mode > 8, <NVar2> > 3, directory number too high, key field length too small or other serious error. The file may have been destroyed.
6	Illegal sequence in specification of directories (mode 0 only).
7	File is not a contiguous file (mode 0 only).
8	Mode 0, <NExpr3> = 0 already performed.
9	Directory not specified.
10	Too many directories (max. 15 per file).
12	File too small to hold all directories (mode 0 only, <NExpr3> = 0).
15	File opened by another user.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

Rules for programming indexed files

- If a data record is to be inserted which is due to have keys in several directories, the data record must be written first. All keys referring to the data record must subsequently be entered in the directories.

- If a data record has keys in several directories, all keys referring to the data record must first be deleted from the directories. The data record itself can then be deallocated.

- After execution of a SEARCH # statement, the status (<NVar2>) must always be tested.

- The key should also exist in the data record. It can be compared with the key specified and be used for recovery if the index area is lost.

The statements for programming contiguous files are used when processing the data records.

● Example:

Random reading of a data record via a key. The data record is locked. The required key is "AA". The following variable names are used:

C = Channel number
V = Directory number
V$ = Key
N = Relative record number
S = Parameter and status variable

The mode of SEARCH # is specified as a numeric literal in the example. The handling of status messages is only intimated here. Obviously precise action should be taken with respect to status messages in application programs.

```
:
9000 LET V$="AA"
9010 SEARCH # C,2,V;V$,N,S:        /* SEARCH FOR KEY
9020 IF S GOTO 9500                /* STATUS <> 0
9030 READ # C,N;R$                 /* READ DATA RECORD
:
9500 IF S=1 PRINT "RECORD DOES NOT EXIST";
:
```

6.4.2.9 CLOSE # statement

Syntax:

CLOSE # <NExpr>[,#<NExpr>]^{n}

Closing files which have been opened and deallocating the data channel.

<NExpr> specifies the number of the data channel on which the file to be closed is open.
A CLOSE # statement enables any number of files to be closed by specifying the numbers of their data channels.

If no file is open under the channel number specified, BASIC error 49 is issued.

Note:

If an error occurs when several files (channels) are closed with one CLOSE # statement, it is not possible to determine which file caused the error.

- Example:

```
:
490 IF ERR 0 GOTO ...    /* ERROR 49 ANALYSIS
500 FOR I=2 TO 13
510 CLOSE # I
520 NEXT I
530 IF ERR 0 GOSUB ...   /* STANDARD ERROR HANDLING
```

Routine to close channels 2 to 13. Channels 0 and 1, which are used by TAMOS, are not closed.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.4.2.10 BUILD # statement

Syntax:

BUILD # <NExpr>, [+] <String> [[,#<NExpr>], [+] <String>]ⁿ

The statement creates new files or replaces existing files on the disk. The file header block is created, possibly an area is reserved on disk, and the file is entered in the INDEX file.
The disk blocks which have been reserved are entered in the disk map as being in use.
The file is opened at the time of creation and is locked against access by other users.

The file created and opened with the aid of BUILD # must be closed by means of CLOSE # after processing, as the file will otherwise be deleted.

<NExpr> : The number of the channel on which the file is to be created.

\+ : "+" identifies the file to be created as a text file.

<String> : String containing details of the file to be created including file parameters.

The string has the following structure:

"[(<Integer1>{:|?}<Integer2>)] <File> [*|!|#]"

<Integer1> = Number of data records for which a disk area is to be reserved.

<Integer2> = Record length in words (1 word = 2 bytes).

These two specifications are in parentheses and separated from one another by ":" or "?". This specification identifies the file to be created as a contiguous file. Coding "?" prevents the data area from being cleared when the contiguous file is created.

<File> : The name under which the file is to be created on disk or the name of the file to be replaced.

! : Must be specified if an existing file is to be replaced.
If the file to be replaced is on the first ILU and was created with "*", the file is always created on the first ILU.

* : The file is created on the first ILU.

: The file is created on the first ILU even if it already exists on the second ILU.

Notes:

If there is not enough free space on the first ILU, BASIC error 47 is output.

If a file is to be replaced, it is only deleted when the new file (which replaces it) has already been created.

If several files are to be created by means of one BUILD # statement and on successive channels, only the channel number for the first file to be created need be specified. Files whose name is not preceded by a channel number are created on successive channels.

Files created with BUILD # are simultaneously opened for processing. They are protected against access by other users until closed with the aid of the CLOSE # statement.

A file created with BUILD # is deleted if the program is terminated and the channel on which the file was created has not been closed beforehand by means of the CLOSE # statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

• Examples:

```
50 BUILD # 2,"(100:50)9/TEST"
```

Creating a contiguous file on channel 2. All parameters are specified as <SLit> in the statement:

- Number of records : 100
- Record length : 50 (words)
- LU number : 9
- File name : TEST
- New creation

- 50 BUILD # 2,+A$,"(100:50)TEST!",B$, #6,C$

Creating several files with one BUILD # statement.

- Creating/replacing a text file on channel 2. The details of the file are stored in the variable A$.

- Replacing the contiguous file TEST on channel 3. The details of the file are specified as <SLit>.

- Creating/replacing a file on channel 4. The details of the file are stored in the variable B$.

- Creating/replacing a file on channel 6. The details of the file are stored in the variable C$.

- :

```
40 DIM 3%,A,B,C$(50)
50 BUILD # 2,"FORFIL"
60 WRITE # 2,0;A,B;
70 MAT WRITE # 2,0,2;C$;
```

Creating a formatted file on channel 2 with the following data record structure:

Field 0: numeric, 3 words
Field 1: numeric, 3 words
Field 2: matrix field, 25 words.

In all other records of the file, too, the first two fields can only be written with 3% variables by means of WRITE # and the third field by means of MAT WRITE #.

6.4.2.11 KILL statement

Syntax:

KILL <String> [, <String>]^{n}

This statement deletes disk files; the first two characters of the file name are deleted in the INDEX file and the disk blocks allocated are entered in the disk map as being free.

Note:

If the file is still open by other users, the blocks which have been allocated are only deallocated when IPL or INSTALL is executed.

<String> : String variables and string literals which contain the names of the files to be deleted ("<File>").

If KILL is used to delete a file which is open by another user or by the program carrying out deletion, only a delete flag is entered in the file header. A file for which a delete flag has been entered can no longer be opened. It is deleted as soon as the user who last opened it closes the file.

Several files can be deleted by means of a KILL statement. If an error occurs, it is not possible to determine the file which caused it.

- Example:

```
Deleting a file whose name is specified in the variable A$.
:
60 KILL A$
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.5 Special statements

This section describes the following statements:

- REM
- SIGNAL 1
- SIGNAL 2
- SIGNAL 3
- CALL
- LGERR

In addition, the facility for writing remarks in any BASIC statement is described.

6.5.1 Remarks in BASIC programs

In the case of long BASIC programs it is necessary to improve the clarity of the program by means of remarks. It should, however, be noted that remarks occupy storage space as well as having an influence on program execution time.

Two methods exist for inserting remarks in programs:

- REM statement
- "/*" after any statement

6.5.1.1 REM statement

Syntax:

REM [<Char>]^{n}

Any number of characters acting as a remark can follow the keyword.

- Example:

```
100 REM ***** GENERAL ERROR MESSAGE *****
```

Note:

A description of the REM01 line, which every standard BASIC program must contain, can be found in the "Program Maintenance" chapter under "Compare Programs".

6.5.1.2 Remarks in any statements

Syntax:

<Statement> /* [<Char>]^{n}

The remark is separated from the statement by means of the sequence of characters /*.
When the BASIC program is output by means of LIST or DUMP, two blanks are always output between the statement and the /* sequence of characters.

- Example: 10 IF ERR 0 GOSUB 9000 /* ERROR BRANCH

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.5.2 Inter-task communication

The 8870 system is a multi-user system, i.e. several ports can work on the system simultaneously. So that ports can communicate with one another, Business BASIC enables data or messages from the programs of the individual ports (tasks) to be exchanged during processing.

The following statements are available for this:

SIGNAL 1 : Transferring numeric values to the signal node queue for another port.

SIGNAL 2 : Transferring numeric values from the signal node queue which had been put there with SIGNAL 1.

CALL 2 : Transferring the contents of variables to another port's common area.

CALL 3 : Transferring the contents of another port's common area to optional variables.

CALL 4 : Displaying a message at another port.

The system has a list (signal node queue) in which the two values transferred by means of the SIGNAL 1 statement, the number of the transmitting port and the number of the addressed port are stored. Each entry thus comprises four values.

6.5.2.1 SIGNAL 1 statement

Syntax:

SIGNAL 1,<NExpr1>,<NExpr2>,<NExpr3>

Transferring four numeric values to the signal node queue. One value is the number of the transmitting port which is inserted automatically.

The other three values must be specified in the statement as follows:

<NExpr1> : Number of the addressed port.

<NExpr2> : Any numeric values to be transferred to the signal
<NExpr3> node queue. Their value may not exceed 32767.

The values remain in the signal node queue until a SIGNAL 2 statement is executed by the addressed port.

If a port number is specified which has not been configured or if the signal node queue is full, the following appears:

ERROR # 62 (SIGNAL BUFFER IS FULL OR NO SUCH PORT)

An entry in the signal node queue is cancelled if it has been there for two hours without being called by means of a SIGNAL 2 statement.

Note:

Two zeros can be entered in the signal node queue for the user's own port by means of "CTL-B" at any time (i.e. the port need not be in the INPUT mode).

- Example:

  Transferring the values 0 and 10 to the signal node queue.
  The master port is addressed.

```
:
50 LET P=0            /* ADDRESSED PORT
60 LET X=0            /* <NExpr2>
70 LET Y=10           /* <NExpr3>
80 SIGNAL 1,P,X,Y
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

6.5.2.2 SIGNAL 2 statement

Syntax:

SIGNAL 2,<NVar1>,<NVar2>,<NVar3>[,<NExpr>]

Transferring numeric values from the signal node queue to numeric variables.

<NExpr> can be used to specify a period in units of 1/10th of a second indicating how long the transfer of a signal is to be waited for.
If a signal is received before the time has expired, program execution is continued.

The value of <NExpr> must be between 1 and 32767.

The numeric variables <NVar1> to <NVar3> contain the following values after execution of the statement:

<NVar1> : Number of the port from which the signal was transferred to the signal node queue.
If there is no signal, <NVar1> contains the value -1.

<NVar2> : The numeric values transferred to the signal node
<NVar3> queue by means of the SIGNAL 1 statement.
If there is no signal, the values of the variables remain unchanged.

- Example:

A port is waiting for a signal from the master port. Signals from other ports are ignored. The program does not continue to run until a signal has been received.

```
:
100 SIGNAL 2,T,X,Y,100  /* WAIT 10 SECONDS
110 IF T<>0 GOTO 100    /* NOT A SIGNAL FROM MASTER PORT
```

6.5.3 SIGNAL 3 statement

- Syntax:

  SIGNAL 3,<NExpr>

The program is set to a waiting state for the period of time specified in <NExpr>. The program does not receive any CPU time until the time specified has expired. The contents of the input/output buffer of the calling port are output.

The value of <NExpr> must be between 0 and 32767 (inclusive) and specifies the waiting time in units of 1/10th of a second.

- Example:

Output of an error message on the screen. The error message is cleared after 30 seconds and the program continues.

```
:
9000 PRINT M$;
9010 SIGNAL 3,300
9020 PRINT 'LD';
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Statements

6.5.4 CALL statements

Syntax:

CALL <NExpr> { , <NumPar> }[12]
{ , <String> }

Library subroutines in machine code are called by means of CALL statements. These subroutines enable functions to be executed which do not exist in the scope of Business BASIC or which could only be solved with the aid of complex routines.

The value of <NExpr> specifies the number and thus the function of the subroutine.

A list of parameters follows the subroutine number, and this list comprises numeric and string variables or constants in which values from the BASIC program are transferred to the subroutine or from the subroutine to the BASIC program (only in variables). The parameter list has a special structure depending on the CALL subroutine.

It is not possible to address substrings (e.g. A$(2,5)). Vector and matrix elements can only be addressed if they have been dimensioned beforehand.

A detailed description of all available CALL functions is contained in the "CALL Statements" chapter.

6.5.5 LGERR statement

Syntax:

LGERR

This causes BASIC errors which have occurred to be entered in the system logbook TF.LOGFILE in the following form.

ERROR # <Integer> AT <LNo> IN <LU>/<ProgName>

<Integer> : 1- to 3-digit error number as for SPC 8.

<LNo> : 5-digit line number as for SPC 10.

<ProgName> : 14-character programe name as for PRN.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7 BASIC functions

BASIC provides the programmer with a series of functions which, when called, return either a numeric value (<NFct>) or a matrix (<MFct>).

Syntax:

```
<NFct>::= { <FctName> (<NExpr>) }
          { LEN (<SVar>)        }
          { DET (<MVar>)        }
```

where

```
<FctName>::=  SIN  COS  TAN  ATN  SQR  LOG  EXP
              ABS  SGN  INT  FRA  RND  NOT  MAN
              CHR  IXR  SPC  CHF  KEY  LKY

<MFct>::=  { { INV }                                  }
           { { MIV }  (<MVar>)                        }
           { { TRN }                                  }
           { { ZER }  [(<NExpr>[,<NExpr>])]           }
           { { CON }                                  }
           {  IDN     [(<NExpr>,<NExpr>)]             }
```

Furthermore, there are the CKY, FLN and PRN functions which do not return a numeric value and also the CHN function.

The specification of parentheses may be omitted for numeric functions if <NExpr> is not a compound numeric expression. The parentheses can also be omitted for the LEN function.

The matrix functions have been explained in detail in the description of the statements; the effect of the numeric functions is described in the following sections.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.1 Mathematical functions

7.1.1 Trigonometric functions

The available trigonometric functions are as follows:

- SIN (sine)
- COS (cosine)
- TAN (tangent)
- ATN (arc tangent)

7.1.1.1 SIN, COS and TAN functions

The argument <NExpr> for the sine, cosine or tangent function is an angle expressed in radians. If the angle is given in degrees, it must be converted into radians by means of one of the formulas:

```
X * 3.1415926535898 / 180 or

X / 57.295779513085
```

• Examples:

```
:
50 LET H=SIN(B)*L
60 LET C=COS(B)
70 PRINT TAN(B)*D/2
```

BASIC Functions

7.1.1.2 ATN function

The argument <NExpr> for the arc tangent function can be any number.
The arc tangent function returns an angle expressed in radians and is the inverse function of the tangent function:

$$TAN\ (ATN(X)) = X$$
$$ATN\ (TAN(X)) = X \text{ for } -\frac{\pi}{2} < X < \frac{\pi}{2}$$

- Example:

```
50 LET B=ATN(H/D)
```

7.1.2 Transcendental functions

The available transcendental functions are:

- SQR (square root)
- LOG (natural logarithm)
- EXP (exponential function)

7.1.2.1 SQR function

The SQR function returns the square root of <NExpr>.

- Example:

```
50 LET X=SQR(36)
```

Result: X=6

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.1.2.2 EXP function

The EXP function is the exponential function on the basis of e (Euler's number):

EXP(X) = e ↑ X (e = 2.71828182..)

- Example:

  50 LET X=EXP(0)

  Result: X=1

7.1.2.3 LOG function

The LOG function returns the natural logarithm of <NExpr> on the basis of e and is the inverse function of the exponential function:

LOG(X) = ln(X)

- Example:

  50 LET X=LOG(EXP(1))

  Result: X=1

7.1.3 Arithmetic functions

The available arithmetic functions are:

- ABS (absolute value)
- SGN (sign function)
- INT (integer value)
- NOT (0-1 function)
- FRA (fraction)
- RND (random number)

BASIC Functions

7.1.3.1 ABS function

The ABS function returns the absolute value of <NExpr>, i.e. the value of <NExpr> with positive sign.

- Example:

```
50 LET A=ABS(A/B)
```

7.1.3.2 SGN function

The SGN function returns

```
 0 if the value of <NExpr> equals 0.
+1 if the value of <NExpr> is positive.
-1 if the value of <NExpr> is negative.
```

- Example:

```
50 LET X=SGN(A/B)
```

7.1.3.3 INT function

The INT function returns the greatest integer which does not exceed the value of <NExpr>.

- Examples:

<NExpr>	INT(<NExpr>)
0.5	0
-1.999	-2
1.99	1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.1.3.4 NOT function

This function returns:

0 if the argument does not equal 0.
1 if the argument is equal to 0.

- Examples:

  - Storing a 0 or 1 in variable B, depending on the contents of the variable A.

```
:
50 LET B=NOT A
```

  - Rounding up the contents of variable B:

```
:
50 LET B=INT B+NOT NOT FRA B
```

7.1.3.5 FRA function

The FRA function returns the fractional value and the sign of the value of <NExpr>.

- Example:

<NExpr>	FRA(<NExpr>)
+1.1234	+0.1234
-1.5	-0.5

7.1.3.6 RND function

The RND function returns a number generated from a sequence of pseudo-random numbers by the processor. The random number is within the following range:

0 ≤ RND(<NExpr>) < <NExpr>

If 0 is specified as <NExpr>, 1 is inserted.

- Example:

  Forming an integer ranging from +1 to +6. The generated number is stored in the variable A.

```
50 LET A=INT(RND(6))+1
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.2 Number manipulation functions

The following functions are available:

- MAN (mantissa)
- CHR (characteristic)
- IXR (10 ↑ integer no remainder).

These functions enable floating-point numbers to be split up into their components.
A floating-point number is represented in the following form:

$$X = m * 10 \uparrow e \quad \text{with } 0.1 \leq ABS(m) < 1$$

The sign of the number belongs to the mantissa.

- m = Mantissa
- 10 = Radix
- e = Characteristic (exponent)

These functions return the following values:

- MAN (X) = m
- CHR (X) = e
- IXR (CHR(X)) = $10 \uparrow e$
- IXR (X) = 10 ↑ (SGN (X) * INT (ABS (X))

X can be represented with these functions as follows:

```
X = MAN (X) * IXR (CHR (X))
```

BASIC Functions

7.2.1 MAN function

The MAN function returns the mantissa of any number (in the case of an integer, after conversion to a floating point number).

MAN (X) = X/10 ↑ CHR (X)

The result returned by MAN is always a fractional number.

- Example:

  The variable X contains the value 57.60.
  After execution of the statement

```
50 LET Y=MAN(X)
```

  the variable Y contains the value 0.576.

7.2.2 CHR function

The CHR function returns the characteristic (exponent) of a floating-point number. CHR (X) always results in an integer.

This function enables, for instance, the number of digits preceding the decimal point to be determined.

- Example:

  Testing an input for:

  - a maximum of 5 integer positions
  - a minimum of 3 integer positions
  - fractional positions are not permitted.

```
:
100 INPUT X
110 IF FRA(X)=0 IF CHR(X)<6 IF CHR(X)>2 GOTO 130
120 GOTO 100 /* INCORRECT INPUT
130 ...      /* INPUT OK
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

7

BASIC Functions

7.2.3 IXR function

The IXR function returns the value:

IXR (X) = 10 ↑ (SGN (X) * INT (ABS (X)))

- Example:

```
- :
  50 LET X=4.25
  60 LET Y=IXR(X)
  :

  Result: Y=10000
```

```
- :
  50 LET X= -2.6
  60 LET Y= IXR(X)
  :

  Result: Y= 0.01
```

ABS (4.25)

INT4

NIXDORF COMPUTER LTD.

SPC Functions as at 23.01.87

SPC No.	Description.
1	CPU Time since logging-on in seconds.
2	Connection time in minutes.
3	Time passed since 01.01.1980 in hours.
4	??? SYSGEN PARAMETER DATE EXPIRES
5	Own account number.
6	Own port number.
7	DCC/Nixdorf Hardware Indicator (RESULTS IN BASIC ERROR # 29)
8	Number of last BASIC ERROR # that occurred.
9	Current Line Number.
10	Line number where last BASIC ERROR occurred.
11	Decimal RDA printer assignment block.
12	IPL Unit Number.
13	Decimal Memory Address Diagnostic Buffer.
14	0 = Program started from NIROS 1 = Program started from TAMOS
15	0 = DAP w/station. 1 = Phantom Port. 2 = RAP w/station. 3 = DAP4 w/station.
16	System Driver Disc Type. 0 = SMD (Storage Module, 21, 33 or 66 Mb). 1 = CA (Cartridge 5 Mb per disc) 2 = CMD (Phoenix 13Mb per dsic) 3 = CDL 4 = FD 5 = MDL(10) 7 = MDL(20)/MDH 8 = CDH.

SEE MANUAL.

NIXDORF COMPUTER LTD.

SPC Functions as at 23.01.87

SPC No.	Description.
17	Number of PU's.
18	Number of Drives.
19	Decimal BDA of System Histor Block (CONFIG copy flags LU's 0 - 99)
20	0 = Print Spool Idling. -1 = Print Spool Not Active. >1 = No. of Jobs.
21	2 = Local Port. 3 = Remote Port.
22	Number of Free Blocks on LU # 0.
23	Value of Last Character keyed in.
24	???
25	Number of ports generated.
26	Logboot fill status. 0 = Outside fill level. <0 = Inside fill level. >4 = For TAMOS. 65535 = Locked
27	Number of Phantom Ports.
28	Threshold value for subset backup (total half-tracks).
29	No. of changed tracks for subset backup.
30	No. of subset media.
31	Subset backup (over 3rd.Generation) with Delta Tape 0 = Without 1 = With.
32	Number of IOC's.
33	Multi-processor indicator. -1 = Normal CPU (MIROS below 7.0) 1-3 = Processor No. for own port (MIROS 7.0)

7.3 Special functions

The following special functions are available:

- SPC
- KEY
- LEN
- LKY
- TAB
- CKY
- CHF
- DET
- CHN
- FLN
- PRN

7.3.1 SPC function

The SPC function returns the following information, depending on the value specified in <NExpr>:

<NExpr>	Information
0	CPU time used since log-on in units of 1/10th of a second.
1	Connection time in minutes since log-on.
2	Time passed in hours since 1.1.1980.
3	Started hour in units of 1/10th of a second (supplement to SPC 2).
4	System creation date expressed as the difference in hours compared with the base date.
5	User number
6	Number of the port in question.
8	The number of the BASIC error which last occurred. This function returns 0 if no BASIC errors have occurred previously.
9	Current line number.
10	Line number of the statement which caused the last BASIC error. This function returns 0 if no BASIC errors have occurred previously.
12	Number of the physical disk unit from which an IPL was last executed.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

<NExpr>	Information
14	Program start from TAMOS selector: 0 = The program was not started under TAMOS. 1 = The program was started under TAMOS.
15	Type of workstation (port) running the program: 0 = Non-loadable workstation: DAP slave 1 = Phantom port 2 = Loadable workstation: RAP (DAP 3 master) 3 = Loadable workstation: BA13 (DAP 4 master)
20	Print spooling status: -1 : Not active >=0 : Number of active jobs
22	Number of free blocks on LU 0 (system disk)

- Example:

Evaluating BASIC errors in an error handling routine with specific reaction to error 98 and pressing of the ESC key. The routine is written as a subroutine.

```
40 IF ESC REM                        /* ESC KEY INEFFECTIVE
50 IF ERR 0 GOSUB 9005
:
1490 INPUT A
:
:
9005 IF SPC 8=98 GOTO 9030
9010 PRINT TAB (15,24);"BASIC ERROR # ";SPC 8;
"AT LINE NO. : "; SPC 10;
9012 LGERR                           /*ENTRY IN LOGBOOK
9015 SIGNAL 3,20
9020 CHAIN A$
9030 PRINT TAB (15,24);"INPUT MUST BE NUMERIC";
9035 SIGNAL 3,20
9040 PRINT 'LD';                     /* CLEAR MESSAGE LINE
9045 RETURN -1
```

7.3.2 LEN function

The LEN function establishes the position of the first delimiter in string variables and returns this position -1, i.e. the actual length of the string. The string variable must be specified as the argument of the LEN function. This function enables strings to be addressed, for example, according to their actual length.

- Examples:

  - 10 DIM A$(20)
    :
    50 LET A$="THURSDAY"
    60 LET X=LEN(A$) (Result: X=8)
    70 LET Y=LEN(A$(5)) (Result: Y=4)

  - Storing the character ":" behind the last character currently in the string A$.

    :
    110 LET A$(LEN A$+1)=":"

7.3.3 CHF function

The CHF function enables a programmer to determine information on a file which has been opened on a particular channel (alternative to CALL 96).
The argument of the function defines the channel number and the function to be performed.
Different details are returned for the different types of file by means of CHF. The following functions exist (C = channel number):

File type	<NExpr>	Function
Formatted	C	Number of the highest written data record +1.
	C+100	Number of the data record last accessed.
	C+200	Byte address of the last data record processed, relative to the beginning of the block.
	C+300	Record length in words.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

File type	<NExpr>	Function
Contiguous	C	Maximum number of data records which the file can hold.
	C+100	Number of the data record last accessed.
	C+200	Byte address of the last data record processed, relative to the beginning of the block.
	C+300	Record length in words.
* * *	C+400	Number of the data record after the highest written one if the EOF pointer is defined. This is only the case if the file was opened with OPEN "C". Otherwise this function returns the maximum number of data records which the file can hold.
Indexed	C	Number of the highest data record.
	C+100	Number of the data record last accessed.
	C+200	Byte address of the last data record processed, relative to the beginning of the block.
	C+300	Record length in words.
Text	C	Number of the highest written data block +1.
	C+100	Number of the current data block.
	C+200	Number of the byte after the last byte transferred (relative to the beginning of the current block).
Printer	C	Current print line
	C+200	Print spooling CHF (C+200) = 0 : NO CHF (C+200) <>0 : YES

Further information can be called up on indexed file status by means of SEARCH # mode 1.

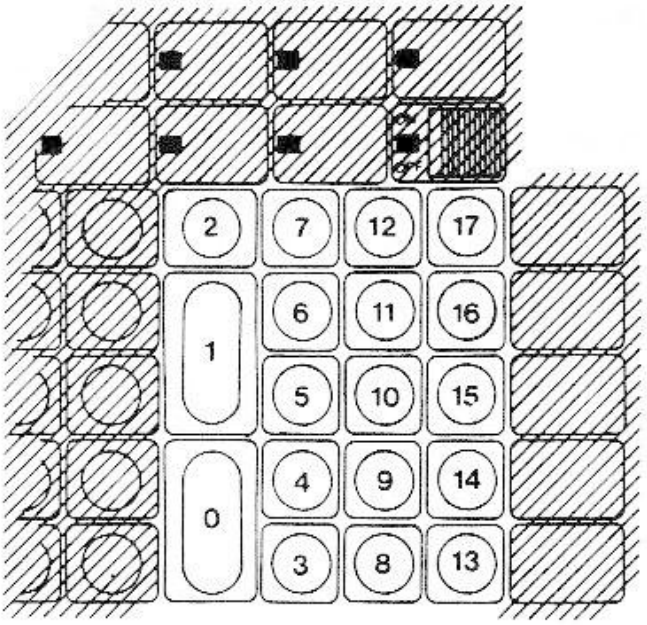
7.3.4 DET function

The DET function returns the determinant value of the matrix last inverted.
A matrix variable (<MVar>) must be specified as the argument; this variable is of no significance, however.

7.3.5 KEY function

The KEY function returns the numeric value of the last release key which was pressed. Only 0 is permissible as the argument of the function.

The values for the keys are:



- Example:

```
:
11  IF KEY (0)=4 GOTO 250
:
250 REM *** A4 PRESSED ***
```

Note:

The "Y" and "N" keys on the numeric keypad have the same effect as "Y" "CR" and "N" "CR" and generate the value 0 as with "CR".

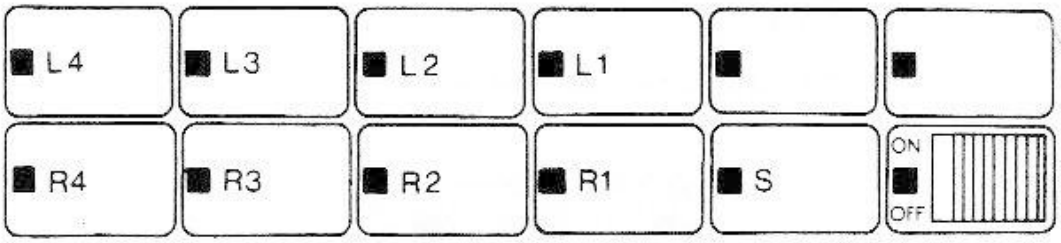
If keys are pressed which are not release keys, KEY (0) has no effect.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.3.6 LKY function

The LKY function returns the value of the set latching key(s).



- LKY (0) returns the total value of all set latching keys.
- LKY (1) returns the value of the "R1" latching key.
- LKY (2) returns the value of the "R2" latching key.
- LKY (3) returns the value of the "R3" latching key.
- LKY (4) returns the value of the "R4" latching key.

The individual values of the keys are:

Latching key	Value set	Value not set
"R1"	256 (2↑8)	0
"R2"	512 (2↑9)	0
"R3"	1024 (2↑10)	0
"R4"	2048 (2↑11)	0

• Example:

```
190 IF LKY (0)=1536 GOTO 250
:
250 REM *** R2, R3  SET ***
:
550 IF LKY (4)=2048 GOTO 660
:
660 REM *** R4  SET ***
```

Note:

If a program is used for remote processing, the latching keys should not be programmed as they can only be used locally.

7.3.7 CKY function

The CKY function clears all latching keys which have been set and sets the latching key values to 0. The argument of the function must be 0.

The function is permissible in the PRINT and INPUT statements only (analogous to TAB functions).

- Example:

```
100 INPUT CKY (0),A
:
190 PRINT CKY (0)
```

7.3.8 CHN function

The CHN function (without parameters) returns the number of the channel which last involved an error.

If no BASIC error has occurred or one has occurred without involving a channel number, CHN returns the value -1. CHN is set to -1 after CHAIN or LINK.
This function is allowed with the statements IF, LET and PRINT. It is not allowed in a numeric expression.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

BASIC Functions

7.3.9 FLN function

The FLN function (without parameters) returns the name of the last file (or driver) which involved an error. This is returned in up to 17 characters in the form <LU>/file (or driver) name.
FLN is reset after CHAIN or LINK.
This function is allowed in the statements IF, LET and PRINT.

7.3.10 PRN function

The PRN function (without parameters) returns the current program name in up to 17 characters in the form <LU>/<ProgName>.
This function is useful for programs which have been CHAINed several times and for error routines.
This function is allowed in the statements IF, LET and PRINT.

CALL Statements

8 CALL statements

CALL subroutines are machine code subroutines which are called up by means of the CALL statements. These subroutines enable functions to be executed which do not exist in the scope of Business BASIC or which could only be solved by means of complex routines.

Syntax:

CALL <NExpr> {,<NumPar> / ,<String>}[12]

<NExpr> : The number of the subroutine to be called (refer to "Table of Available Subroutines).

<NumPar> : Up to 12 numeric parameters (<NVar> or <DecNo>) or
<String> : strings (<SVar> or <SLit>). In many cases, only integers are relevant as numeric parameters (decimal places are then ignored). These quantities are used to specify parameters to the subroutine or to exchange information between the BASIC program and the subroutine.

It is not possible to address substrings (e.g. A$(X,Y)), whereas individual vector and matrix elements can be addressed.

The description of available subroutines indicates restrictions and conditions which do not generally apply.

Errors:

BASIC error	Meaning
36	The called subroutine does not exist.
37	An expression was specified instead of a variable.
38	Incorrect variable type/error detected by the called subroutine.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.1 Table of available subroutines

CALL number	Function
1	Preparing the screen for an input, checking and formatting input data.
2	Moving the contents of variables to a port's common area.
3	Moving data from a port's common area to one or more variables.
4	Outputting a message on another screen or reading/writing the archive ID and LU number.
6	Calculating the check digit of a number in alphanumeric notation.
7	Hash totalling for a string.
20	Reading, setting or resetting a bit in a string.
21	Converting the contents of a numeric variable to a binary number.
22	Converting a binary number and transferring the value to a numeric variable.
23	Searching a string for a character combination.
24	Inverting the character sequence in a string.
25	Converting the date to an industrial date.
26	Selective character replacement in string variables.
28	Converting binary values to other number systems.
29	Reading or modifying the Program Abort Flag and Port Control Area.
31	Code conversion.
32	Date processing.

CALL Statements

CALL number	Function
35	Disk configuration.
51	Supporting screen programming.
59	Converting numbers from and to IBM format.
60	Packing a string.
61	Unpacking a string.
62	Checking whether the contents of a string are numeric. Syntax check on a string.
63	Checking whether the contents of a string comprise a valid numeric value.
64	Checking and converting a date in a string.
65	Sorting keys in a string.
66	Extended code conversion.
70	Programming magnetic tape units.
72	Transferring the contents of variables to a string.
73	Transferring the contents of a string to several variables.
74	Reading an entry in a port's device assignment table.
80	Programming magnetic tape cassette units.
84	Reading, displaying and entering a message in the log-book.
90	Processing string variables.
91	Bit inversion in substrings.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

CALL number	Function
93	Ascertaining disk information.
96	Checking validity of file names and reading information about files.
97	Reading information from file headers.
98	Sending system commands.
99	Transferring/updating date and time of day.

CALL Statements

8.2 CALL 1

Preparing the screen for an input, validating and formatting input data.

A distinction is drawn between two formats:

Format	Syntax
1	CALL <NExpr>, <NVar1>, <String1>, <String2>, <NumPar>, <SVar2>
2	CALL <NExpr>, <NVar1>, <String1>, <SVar1>, <NVar2>, <String3>, <String4>, <SVar2>

Format 1 (screen preparation):

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 1).

<NVar1> : Function number: must be 0 (for format 1). After this CALL has been executed, the variable contains the value 1.

<String1> : Parameter string to control the operation. This variable or literal must be 12 or 13 bytes long (see description).

<String2> : Characters for the input field. If <String1> contains the number 3 in byte 1, the contents of <String2> are output in the input field if the input is alphanumeric. If the input is numeric unpacked (byte 2 of <String1> = 0), the contents of <NumPar> are output.

<NumPar> : Numeric value which is displayed in the input field if the input is numeric unpacked.

<SVar2> : Work area, length > = 18 bytes.

If a mask is output in the input field, a background blank is output after the last character so that the maximum input field length is not exceeded.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Format 2 (validating and formatting the input):

<NExpr>, <String1> and <SVar2> as for format 1.

<NVar1> : Function number (value 1 for format 2).
After execution of the CALL, the variable contains the status of the operation:

-1 = Input was not correct.
0 = Input was correct.
1 = "@" key was pressed to duplicate the existing contents.
2 = "↑" key was pressed to reverse to the last input.
x = The xth-2 three-character function from <String3> was input.

<SVar1> : Input string, dimension >= maximum input length.
After input, the contents of this variable are validated against the control parameters in <String1> and formatted. If the input is correct, the following occurs:
- Numeric inputs are stored right-justified with leading zeros in <SVar1> and any missing decimal places are padded with zeros up to the maximum number of decimal places.
- Alphanumeric inputs are stored left-justified and, if necessary, padded with blanks on the right up to the maximum input length.

The input string is then displayed on the screen without leading zeros.
BASIC error 38 is generated if dimensioning exceeds 13.

<NVar2> : For numeric input : Input value.
For alphanum. input: Number of characters entered.
(However, this does not apply to inputs via a standard function key. Then <NVar2> is not updated.)

<String3> : Contains the character sequences in three bytes each to be interpreted as control information (example: END, SPR etc.).

CALL Statements

<String4> : The bytes in this string specify which control characters are allowed in place of an input. Zero means that the function is allowed and any other character means that it is not.

Assignment of the bytes to the functions:
Byte 1 = "@" key
Byte 2 = "↑" key
Byte 3 = First character sequence in <String3>
:
Byte x = The xth-2 character sequence in <String3>

Example:
<String3> = "SPREND", <String4> = "0110" means:

"@" is allowed.
"↑" is not allowed.
SPR is not allowed.
END is allowed.

Note 1:

If an error occurs, the appropriate message is read from the file open on channel 1 (under TAMOS this is TF.PARAM).

Note 2:

After CALL 1 has been executed, the marked cursor position is the first input field position.

Note 3:

Using CALL 1 format 2 without a previous CALL 1 format 1 causes an error.

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Description of the control parameters in <String1>:

Byte	Value	Meaning
1	0	Do not output an input mask.
	1	Output the input mask in the form of dots.
	2	As for 1, but 'Clear Foreground' beforehand.
	3	Output the contents of <String2>/<NumPar> as an input mask. Note: This byte is set to 0 with format 2 if an error occurs.
2	0	Numeric input.
	1	Alphanumeric input.
	2	Numeric input.
3-4	xx	Column number of cursor position.
5-6	xx	Line number of cursor position.
7-8	xx	Maximum no. of integer places or characters.
9-10	xx	Minimum no. of integer places or characters.
11	x	Number of decimal places.
12	0	No sign allowed.
	1	Sign allowed.
13	0	If input field only has 1-2 characters, supplement with blanks to 3 characters prior to background blank to allow any 3-character control information (END, SPR, etc.) to be input.
	1	Reserve no blanks. If <String1> is 12 bytes long, 0 is adopted.

Possible error: ERROR # 38

Meaning:

- Illegal variable type or dimensioning
- Channel 1 not open
- Illegal values in the parameter string
- Mask for input field too long
- <NVar2> is 1%, input >7999 or <-7999
- Duplication not allowed for numeric input with subsequent packing.

CALL Statements

8.3 CALL 2

Moving the contents of up to 11 variables or constants to any port's common area.

Syntax:

CALL <NExpr>, <NumPar1> { , <String> / , <NumPar2> }[11]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 2).

<NumPar1> : Number of the port into whose common area the contents of the following elements are to be transferred.

<String> : 1 to 11 variables or constants.
<NumPar> The entire dimensioned length of all data to be transferred must not exceed 512 bytes.

Transfer is on a word basis. String variables are transferred in their total dimensioned length, irrespective of delimiters, and are filled with a delimiter to an even number of bytes if the length is odd.

- Example:

  Transferring the contents of the variables B and A$ to the phantom port's common area.

```
10 DIM A$(5),3%,B
:
60 CALL 2,1,A$,B
```

  Positions in the common area: Bytes 1- 5: A$
  Byte 6 : Delimiter
  Bytes 7-12: B

  Possible error: ERROR # 38

  Meaning:

  - The total dimensioned length of the data to be transferred exceeds 512 bytes.
  - The specified port number is invalid.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.4 CALL 3

Moving data from any port's common area to up to 11 variables.

Syntax:

```
CALL <NExpr>, <NumPar>  {,<SVar>}^11
                        {,<NVar>}
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 3).

<NumPar> : Number of the port from whose common area data is to be transferred.

<NVar> : 1 to 11 variables.
<SVar> The data is transferred physically. If the variables specified do not match the data to be transferred with respect to type or length, the result is undefined.

The total dimensioned length of all destination variables must not exceed 512 bytes.

- Example:

  Transferring data from the phantom port's common area to the variables B and A$.

```
:
60 CALL 3,1,B,A$
```

Possible error: ERROR # 38

Meaning:

- The total dimensioned length of the destination variables exceeds 512 bytes.
- The specified port number is invalid.

CALL Statements

8.5 CALL 4

This subroutine performs the following functions:

- Output of messages to other ports
- Reading/writing the archive ID and LU number

8.5.1 Output of messages to other ports

Syntax:

CALL <NExpr>, <String> [, <NumPar>]

<NExpr> : The value of this numeric expression is the number of the subroutine to be called (in this case 4).

<String> : String whose contents are to be output on the screen of the port addressed.

<NumPar> : Addressed port. If the number is not specified, the message is displayed at the master port.

This statement is only executed correctly if the addressed port is in the INPUT mode. Any other status causes BASIC error 38 during execution. The statement must therefore be repeated until it can be executed successfully.
The output appears in the current display line, starting at the first possible input position. Therefore <String> should contain appropriate display functions (e.g. TAB) before the actual message. The receiving port's incomplete input is deleted.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example:

  Output of a message at position 15 in line 24 (message line) at port 2. Output is repeated until the statement is executed without errors.

```
10 DIM M$(50)           /* STRING TO HOLD THE MESSAGE
:
50 LET M$="←207←←376←←211←←376←←221←←217←←230←
BACKUP IN ONE HOUR!←376←←212←"
60 IF ERR 0 GOTO 900    /* ERROR # 38 ??
70 CALL 4,M$,2
80 IF ERR 0 GOSUB 9000  /* STANDARD ERROR HANDLING
:
900 IF SPC8<>38 CHAIN ../* ABORT PROGRAM
910 SIGNAL 3,100        /* WAIT 10 SECONDS
920 GOTO 70             /* TRY AGAIN
```

8.5.2 Reading/writing the archive ID and LU number

Syntax:

```
CALL <NExpr> ,[<NVar1>  ,  <NVar2>,] {<NVar3>  ,}
                                     {<DecNo>  }
             <SVar1>  [,<SVar2>]
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 4).

<NVar1> : 1% numeric variable in which the driver number has to be specified in mixed systems if the function refers to drivers other than the one for the system drive. The driver numbers are allocated as follows:

SMD/CDH/FDU	8
CA	9
CMD	10
CDD	11
FD8	12
MDL	13
FD 5 1/4"	14

<NVar2> : 1% numeric variable in which the driver number has to be stored according to CONFIG after the function has been executed.

CALL Statements

<NVar3> : The physical unit of the storage medium to which the
<DecNo> archive ID has to be written or from which it has to be read.

<SVar1> : String variable, length >= 3 bytes containing the archive ID to be written or read.

<SVar2> : String variable (length >= 2 bytes) which contains the LU number of the addressed storage medium after the function has been executed.

Writing the ID:

If the length of <SVar1> is unequal to 0, the first two bytes of the string are written to the storage medium whose physical unit number is specified.

Reading the ID:

If <SVar1> is a null string, the ID of the storage medium addressed by <NVar1>and <NVar3>/<DecNo> is transferred to the string. The common area of the port in question is overwritten!

- Examples:

  - Archive ID 99 is written to the disk which has physical unit number 1.

```
:
100 LET U=1       /* LOAD UNIT NUMBER
105 LET K$="99"   /* LOAD ID
110 CALL 4,U,K$   /* WRITE ID
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- The ID is read from the disk which has physical unit number 0.

```
:
100 LET U=0     ./* LOAD UNIT NUMBER
105 LET K$=""    /* K$ = NULL STRING
110 CALL 4,U,K$ /* READ ID
```

Possible error: ERROR # 38

Meaning:

- Disk read or write error.
- String variable dimensioned for less than three bytes.
- <NVar1> and/or <NVar2> not dimensioned as 1% variables
- Invalid unit number
- PU number too high

CALL Statements

8.6 CALL 6

Calculating the check digit of a number in alphanumeric notation. Check digits ensure a high degree of data integrity.
CALL 6 provides two different methods of calculating check digits.

Syntax:

CALL <NExpr>, <NumPar>, <NVar>, <String1> [,<String2>]
[,<String3>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 6).

<NumPar> : Function number (1%):

<NumPar>	Check digit
0	B - Z;*F; modB
1	Z;*F;modB

Explanation below.

<NVar> : The check digit is returned (1%) in <NVar> after CALL 6 has been executed.

<String1> : The number for which the check digit is to be calculated must be stored in this string. Only digits and blanks are allowed. Blanks are interpreted as zero.

<String2> : This string (length 1 byte) can be used to specify a radix between 2 and 11, 10 and 11 having to be coded as A and B respectively. If this specification is omitted, A is used.

<String3> : A list of factors can be specified in this string. The factors can be any digits or A (for 10) or B (for 11). If no list of factors is specified, "1" is used.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

8

CALL Statements

Explanation of functions:

B is an integer radix ranging from 2 to 11 and is supplied in <String2>.

Z;*F is formed as follows from the number supplied in <String1> and the optional list of factors supplied in <String3>:

1st digit(number)*1st digit(factors)+2nd digit(number)*2nd digit (factors)+3rd digit(number)*3rd digit(factors)...

If the list of factors is shorter than the number, it is re-started from the beginning when the end has been reached.

The total calculated in this way is divided by the radix B and the remainder is Z;*F;modB.

Therefore, the check digit ranges from 0 to B-1 for function 0 and function 1.

● Example:

The check digit is to be calculated according to function 0 for the number 1234567 and the factor list "0AB" using 8 as the radix.

Z;*F;=1*0+2*10+3*11+4*0+5*10+6*11+7*0=169

169:8= 21 remainder 1, i.e. Z;*F;mod 8=1

The check digit is 8-1=7 according to function 0.

Possible error: ERROR # 38

Meaning:

- Numeric variable not dimensioned with 1%.
- <String2> exceeds 1 byte.
- Character in number string not a digit or blank.
- Illegal character in the radix or list of factors
- Function number less than 0 or greater than 1.

CALL Statements

8.7 CALL 7

Hash totalling for a string.

Syntax:

CALL 7, <NVar1>,<SVar>,<NVar2>

<NVar1> : The number of bytes to be checked is specified in this numeric variable (dimensioning min. 2%).

After the function has been executed, <NVar1> contains the hash total for <SVar>.

<SVar> : String variable for which the hash total is to be formed up to the length specified in <NVar1>. <SVar> can also contain binary values.

<NVar2> : Status variable indicating whether the hash total was formed correctly.

<NVar2> = 0 : Hash total was formed correctly

<NVar2> <> 0 : Error

BASIC error 38 is output in the event of syntax errors.

8.8 CALL 20

Reading, setting or resetting a bit in a string. The bits are numbered consecutively from left to right beginning at 0.
A series of flags can be set in a minimum of space with the aid of this CALL.

Syntax:

CALL <NExpr>, <NumPar1>, <SVar>, <NumPar2> ,<NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 20).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<NumPar1> : Function number:

Number	Function
0	Establishing the first set bit in <SVar> starting from the bit position specified in <NumPar2>. The address established is returned in <NVar>.
1	The bit whose position is specified in <NumPar2> is set (receives the value 1). <NVar> need not be coded with this function.
2	The bit whose address is specified in <NumPar2> is reset (receives the value 0). <NVar> need not be coded with this function.

<SVar> : String variable in which bits are to be processed. The maximum length is 4096 bytes.

<NumPar2> : Bit position in <SVar> (starting from the left with 0, meaning depends on the function specified in <NumPar1>).

<NumPar1>	Meaning of <NumPar2>
0	Position of the first bit from which comparison is to start.
1	Address of the bit to be set.
2	Address of the bit to be reset.

As <SVar> is restricted to a string length of 4096 bytes, the bit position in <NumPar2> can range from 0 to 32767.

<NVar> : (Only necessary if <NumPar1> = 0). The address of the first set bit is returned in this variable. If no set bit is found, <NVar> receives the value -1.

- Example:

Searching the string A$ for the first set bit from position A. The position of the first set bit is expected in B. The first set bit is subsequently reset.

```
:
105 CALL 20,0,A$,A,B
110 IF B=-1 GOTO 9000 /* NO BIT SET IN A$
115 LET F=2           /* RESET BIT FUNCTION
120 CALL 20,F,A$,B    /* RESET BIT
```

Possible error: ERROR # 38

Meaning:

Negative bit address specified.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.9 CALL 21

Converting the contents of a variable or constant to a binary number.

Syntax:

CALL <NExpr>, <NumPar1>, <SVar> [, <NumPar2>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 21).

<NumPar1> : The contents of this decimal number are converted to a binary number of one or two bytes in length. The contents must not exceed 255 (1-byte conversion) or 65535 (2-byte conversion) and must not be negative.

<SVar> : The binary number is stored in this string variable (not followed by a delimiter). <SVar> must be dimensioned for a minimum of two bytes for 2-byte conversion.

<NumPar2> : Byte address starting from which the binary number is to be stored in <SVar>:

- If the byte address is positive, the value of <NumPar2> must range from 1 to the dimensioned length of <SVar>-1 and must be odd. If the value is even, the next lower odd one is used. When a positive byte address is used, the contents of <NumPar1> are converted to a 2-byte binary number.

- If the byte address is negative, the corresponding positive displacement is used and the contents of <NumPar1> are converted to a 1-byte binary number. The byte address must range from 1 to the dimensioned length of <SVar> and can be any value.

If <NumPar2> is not specified or equals zero, 1 is used and 2-byte conversion is performed.

CALL Statements

- Examples:

  - Storing the contents of A as a binary number in A$ starting from byte 9.

```
:
110 CALL 21,A,A$,9     /* CONVERT TO A 2-BYTE BINARY NUMBER
                          AND STORE
```

  - Storing the contents of A as a binary number in A$ at byte 9.

```
:
110 CALL 21,A,A$,-9    /* CONVERT TO A 1-BYTE BINARY NUMBER
                          AND STORE
```

Possible error: ERROR # 38

Meaning:

- Number exceeds 65535 for 2-byte conversion.
- Number exceeds 255 for 1-byte conversion.
- Number is negative.
- Byte address is greater than the dimensioned length of <SVar> for 1-byte conversion or <SVar>-1 for 2-byte conversion.

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.10 CALL 22

Converting and transferring a binary number with a length of one or two bytes from a string to an optional numeric variable.

Syntax:

CALL <NExpr>,[<NumPar>,]<String>, <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 22).

<NumPar> : The byte address in <String> from which the binary number is to be transferred.
- If the byte address is positive, the value of <NumPar> must range from 1 to the dimensioned length of <String>-1 and must be odd. If the value is even, the next lower odd one is used. When a positive byte address is used, 2 bytes are transferred from <String> to a numeric variable.
- If the byte address is negative the corresponding positive displacement is used and 1 byte is transferred from <String> to a numeric variable.
  The byte address must range from 1 to the dimensioned length of <String> and can be any value.

If <NumPar> is not specified or equals zero, 1 is used and a 2-byte conversion is performed.

<String> : The contents of one or two bytes are transferred to a numeric variable from this string, starting at the address specified.

<NVar> : Destination variable which receives the value of the binary number from <String> converted into the appropriate representation.

CALL Statements

- Examples:

  - Transferring the binary number which starts at byte 5 in A$ to the variable F.

```
:
105 CALL 22,5,A$,F      /* TRANSFER 2 BYTES
```

  - Transferring the binary number stored at byte 5 in A$ to the variable F.

```
:
110 CALL 22,-5,$,F      /* TRANSFER 1 BYTE
```

Possible errors:

BASIC error	Meaning
15	Arithmetic overflow. A value exceeding 7999 is to be transferred to a 1% variable.
38	Contents of <NumPar> are greater than the dimensioned length of <String> for 1-byte conversion or <String>-1 for 2-byte conversion.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.11 CALL 23

Searching a string for a substring. Both directions - from left to right and right to left - are possible.

Syntax:

```
CALL <NExpr>, <String1> <String2>, <NumPar1>, <NVar>
                                    [,<NumPar2> [,<NumPar3>]]
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 23).

<String1> : This string specifies the substring to be searched for in <String2>.

<String2> : This string is searched for the substring specified in <String1>. Delimiters do not terminate the search run.

<NumPar1> : The byte address from which searching is to commence in <String2>. The value of <NumPar1> must range from 1 to the length of <String2> minus the length of <String1>.

<NVar> : The start address of the substring in <String2> is returned in this variable after the statement has been executed. If the required substring does not exist in <String2>, <NVar> receives the value -1.

<NumPar3> : This defines how often the search process should be repeated, i.e. if n is specified, the search for the substring will not be terminated until it has been found n + 1 times and only then is the byte address passed on to <NVar2>.
Only positive integers should be specified.
<NumPar2> must not be dimensioned for more than 1%.

CALL Statements

<NumPar3> : Value	Meaning: A string of the same length as in <String1> is sought and
0/4	= Equal to the string in <String1>
1/5	<= Less than or equal to the string in <String1>
2/6	>= Greater than or equal to the string in <String1>
3/7	<> Not equal to the string in <String1>

When strings are compared, the terms "greater" and "less" refer to greater or less than the value of the ASCII code. The characters in the string are compared individually from left to right when the value is 0 to 3 and from right to left when the value is 4 to 7.

- Examples:

Searching the string A$ from byte 1 for the character combination "/*".

Contents of A$: "1234505555/*003004005"

```
:
100 LET B$="/*"       /* COMPARATIVE VALUE
110 CALL 23,B$,A$,1,B /* SEARCH
```

Result: B=11

Searching from left to right for the 6th 3-character substring less than or equal to "ALT" in the specified string, starting at the first byte:

```
110 CALL 23,"ALT","ALTNEU123ALT!456",1,B,5,1
```

Result: B = 13

The following substrings were checked for "less than or equal to": ALT,LTN,TNE,NEU,EU1,U12,123,23A,3AL,ALT,LT!,T!4,!45. The underlined substrings are the ones that were searched for.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Searching the string specified in the previous example for the first substring "ALT", starting at byte 14 and going from right to left:

```
210 CALL 23;"ALT","ALTNEU123ALT!456",14,B,0,4
```

Result: B = 10
(The checked substrings were: T!4,LT!,ALT)

Possible error: ERROR # 38

Meaning:

- Byte address less than 1 or greater than the dimensioned length of <String2>.

CALL Statements

8.12 CALL 24

Inverting the contents of a string. In this way, the last character in the string becomes the first character, the last-but-one becomes the second character, etc.

Syntax:

```
CALL <NExpr>,<SVar>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 24).

<SVar> : The contents of this string are rearranged in such a way that the first character is transferred to the last byte, the second character to the second last byte, etc. If a string delimiter occurs, the string is filled with blanks to the left.

- Example:

  Inverting the contents of A$.

  Contents of A$ before:

```
                        S
|1|2|3|4|5|6|7|8|9|0|D|1|2|3|4|5|
```

```
:
100 CALL 24,A$
```

  Contents of A$ afterwards:

```
| | | | | | |0|9|8|7|6|5|4|3|2|1|
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.13 CALL 25

Converting the date to an industrial date and vice versa. The following possibilities exist:

- Converting the system date to an industrial date.
- Converting a date specified by the application program to an industrial date.
- Converting an industrial date specified by the application program to a date in the format DDMMYY.

Syntax:

```
CALL <NExpr>,<SVar>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 25).

<SVar> : String variable in which the date to be converted is specified. If <SVar> is a null string, the system date is converted. <SVar> must be dimensioned for a minimum of 23 bytes.
After the statement has been executed, <SVar> contains the industrial date in bytes 1 to 5 or a date in the format DDMMYY in bytes 1 to 6.

If a date is specified in <SVar>, two formats are accepted:

- "MMM DD YYYY"
- "DDMMYY"

- Format 1:

The three specifications MMM DD YYYY must be separated by a blank in each case.

MMM = Month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC.
DD = Day
YYYY = Year

CALL Statements

- Format 2:

DDMMYY - date represented in 6 bytes.

DD = Day
MM = Month
YY = Year

The industrial date prepared by means of CALL 25 has the following format:

"YYDDD"

YY = Year
DDD = Day in the year

Finally, CALL 25 enables an industrial date specified by the application program in the above-mentioned format to be converted to an 8-byte date in the format DDMMYY.

In the course of conversion, a check is made as to whether the specified date is permissible. Leap-years are taken into consideration during the check.

• Example:

Converting the system date to an industrial date.

```
:
100  DIM A$(25)        /* DIMENSIONING
:
200 LET A$=""          /* <SVar> = NULL STRING
210 CALL 25,A$         /* CONVERT DATE
```

Possible error: ERROR # 38

Meaning:

- Date specified in wrong format.
- Impossible date
- <SVar> dimensioned for less than 23 bytes.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.14 CALL 26

Selective character replacement in strings.

Syntax:

CALL <NExpr>, <NumPar>,<String1>,<SVar>, <String2>,<String3>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 26).

<NumPar> : Function number = 0

<String1> : String to be modified. The modified contents are transferred to <SVar>, while <String1> remains unchanged.

<SVar> : String variable into which the modified contents of <String1> are to be transferred.

<String2> : Characters which, when they occur in <String1>, are to be replaced by the appropriate characters of <String3>.

<String3> : Characters to be inserted.

<String1> and <SVar> must be dimensioned identically.
<String3> must have the same length as <String2> since the replacement table is formed by means of the two strings. Their length must not exceed 256.

- Example:

  Replacing the characters 1, 3 and 7 by A, B and C in the string A$ and transferring the contents to B$.

```
10 DIM A$(20),B$(20)
:
50 LET A$="1234567890",A$
:
200 CALL 26,F,A$,B$,"137","ABC"
```

  Contents of B$: A2B456C890A2B456C890

CALL Statements

8.15 CALL 28

Converting binary values to numbers in an optional number system with a radix from 2 to 36 and presenting the results in printable format (ASCII).

Syntax:

```
CALL <NExpr>, <String1>, <NumPar1>, <SVar>, <NumPar2>,
     <NumPar3>, <NumPar4>, <NumPar5>, [<NumPar6>]
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 28).

<String1> : Specification of the binary value in one or two words.

<NumPar1> : Specification of the displacement (in bytes) relative to the beginning of <String1> at which the binary value starts in <String1>.

<SVar> : Destination variable for the converted number.

<NumPar2> : Specification of the displacement in <SVar> starting at which the converted number is to be stored in printable format.

<NumPar3> : Specification of the radix (2-36) of the number system into which the binary value is to be converted.

<NumPar4> : Size of the binary value in words (1 or 2).

<NumPar5> : Specification of the minimum number of digits to be stored in <SVar>.
If the converted number results in less digits than specified, leading blanks are inserted. If the value specified was negative, leading zeros are inserted instead of blanks. If the number of digits in the converted number exceeds the specified figure, the entire number is stored nonetheless provided the dimensioned length of <SVar> permits this.

<NumPar6> : =1: Conversion of the byte referenced in <NumPar1>. <NumPar4> is ignored.
=0: No effect.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Note:

Displacements can be positive integers. The value 0 is interpreted as displacement 1, just like the value 1.

Possible errors:

Error	Meaning
15	<SVar> could not hold all the digits which were to be stored. The rest of the number was suppressed.
38	- Not all variables specified - Incorrect parameters specified

CALL Statements

8.16 CALL 29

Format 1 of CALL 29 can be used to influence the Program Abort Flag (PAF) and the Port Control Area (PCA). Format 2 can be used to determine whether a port is inactive or active under TAMOS or SCOPE.

CALL 29 - format 1 (processing the Port Control Area)

The entry in the PAF defines what is to happen after a program aborts. A BASIC program can be split up into individual segments which make different reactions necessary when a program aborts. The PAF and PCA form an area in the TF.PORT file (one for each port) containing the PAF in the first two bytes, the name of the program (<LU>/<ProgName>) to be started in the event of a program abort if PAF equals 2 in the next 18 bytes and user-definable information in the next 492 bytes.

It is possible to read and write the PCA using different types of variables in any sequence (analagous to the common area with CALL 2/3). Numeric variables can be passed on directly, without buffering in string variables, to the restart or repair program. Restrictions analogous to the common area: CALL 29 can use a maximum of 12 variables in all: CALL 29 can use a maximum of 12 variables in all. Including the PAF, 512 bytes can be written to or read from the PCA. The variables are transferred on a word basis.

CALL 29 has the following interface for format 1:

$$\text{CALL 29 , } \left\{\begin{matrix}\text{<NVar1>}\\ \text{<Integer1>}\end{matrix}\right\} \text{ , } \left\{\begin{matrix}\text{<NVar2>}\\ \text{<Integer2>}\end{matrix}\right\} \text{ , } \left\{\begin{matrix}\text{<NVar3>}\\ \text{<SVar>}\end{matrix}\right\}^{10}$$

<NVar1> : Function number
<Integer1> 1 = Read PAF and PCA
2 = Write PAF and PCA

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<NVar2> : Status variable
<Integer2> In function 1, this is the destination variable for the contents of PAF.
In function 2, this is the source variable, i.e. the status which is to be written must be stored here first. Consequences in the event of a program abort:

0 = Return to selector

1 = Choice of restart or return to selector

2 = If possible, the repair program entered in the PCA is called, otherwise second generation recovery

3 = Second generation recovery absolutely necessary

4 = Restart (without any question)

<NVar3> : In function 1, this is the destination variable
<SVar> for the contents of the PCA.
In function 2, the contents of <SVar> are written to the PCA. If a repair program is due to be called, the string contains user-defined information to be passed on to a repair or restart program via the Port Control Area. The first 18 bytes must contain the name of the repair program (<LU>/<ProgName>).
The user-defined information can be up to 510 bytes long and may be split up into several numeric and alphanumeric variables. In accordance with the above-mentioned format, CALL 29 permits 10 variables as user-defined information because 2 of the 12 possible variables have been allocated to the function number and status.

Attention:

A program with this abort handling procedure must only be called via TAMOS.

CALL Statements

CALL 29 - format 2 (reading the TAMOS flag)

In this format, CALL 29 makes it possible to determine whether a port is inactive or active under TAMOS or SCOPE.

CALL 29 has the following interface for format 2:

CALL 29, <NVar1>, <NVar2>

<NVar1> : Function number = 4
<NVar1> contains the port status when CALL 29 has been executed:

-1 = Port inactive
0 = Port active under SCOPE
1 = Port active under TAMOS

<NVar2> : Port number. BASIC error 28 is generated if an illegal port number is specified.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.17 CALL 31

Code conversion with the aid of code tables.

Syntax:

```
CALL <NExpr>, <NVar1>,  <String> , <SVar1>, <SVar2>, <NVar2>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 31).

<NVar1> : Specification of the number of bytes to be converted. After the statement has been executed, this variable contains the number of bytes which were actually converted.

<String> : Source string which must contain the characters to be converted.

<SVar1> : Destination string which contains the converted characters after the statement has been executed correctly.

<SVar2> : The code table to be used for conversion must be specified here. This variable must be dimensioned so as to be large enough to hold the entire code table.

<NVar2> : Status variable
0 = Statement executed without errors
1 = <NVar1> is not positive or too high.

- Example:

```
  :
 50 DIM C$(256)
  :
800 OPEN # 3,"ASCII.EBCDIC"
810 MATREAD # 3,0;C$;       /* READ CODE TABLE
820 CLOSE # 3
830 CALL 31,A,Q$,Z$,C$,N2   /* ASCII TO EBCDIC
```

CALL Statements

8.18 CALL 32

Date processing:

Checking or converting the date, reading or setting the system time.

Syntax:

Format 1 for function 0

CALL <NExpr>, <NumPar1>, <String> [,<NumPar2>], <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 32).

<NumPar1> : Function number (in this case 0).

<String> : String which must contain the date to be checked.

<NumPar2> : Specification of the displacement of the date in <String>.
Value 1 is taken as the default value.

<NVar> : Status variable

Format 2 for functions 1-5 and 10.

CALL <NExpr>, <NumPar1>, <String> [,<NumPar2>], <SVar>
[,<NumPar3>], <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 32).

<NumPar1> : Function number (in this case 1-5 or 10)

<String> : Specification of the date to be converted

<NumPar2> : Specification of the displacement of the date in <String>. Value 1 is taken as the default value.

<SVar> : Destination string for the converted date.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

8

<NumPar3> : Specification of the displacement of the date in <SVar>. Value 1 is taken as the default value.

<NVar> : Status variable.

Format 3 for functions 8 and 9

CALL <NExpr>, <NumPar1>, <SVar1> [,<NumPar2>]
[,<SVar2> [,<NumPar3>]], <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 32).

<NumPar1> : Function number (in this case 8 or 9).

<SVar1> : In fct. 8 : Destination string for the result date
In fct. 9 : Specification of the date

<NumPar2> : Specification of the displacement of the date in <SVar1>.
Value 1 is taken as the default value.

<SVar2> : In fct. 8 : Specification of internal system time representation (STR) which is converted to extended ASCII representation (EAR). The system time is taken as the default value.

In fct. 9 : Result date
Only transfer as the internal system time if this is not coded.

<NumPar3> : Specification of the displacement of the date in <SVar2>.
Value 1 is taken as the default value.

<NVar> : Status variable

CALL Statements

Functions:

Fct. no.	Input	Output	Function description
0	IAR	IAR	Validity check
1	ER	IAR	Conversion, validity check
2	IAR	ER	Conversion
3	IAR	IBR	"
4	IBR	IAR	"
5	IAR	CW	"
6	CW +DAY	IAR	" NIROS 7.01
8	-	EAR	Read internal system time +date
	STR	EAR	Conversion
9	EAR	-	Validity check (with time), set system time
	EAR	STR	Conversion
10	IAR	ER	Conversion, separators suppressed

IAR : Internal ASCII representation
Date representation in the form YYMMDD in 6 bytes of ASCII.

ER : External representation
Preparing a date in the form defined by the format control string in the system, i.e. DD, MM and YY in a certain sequence. This format can either be represented as a 6-byte string or as an 8-byte string with separators.

The following separators are allowed:

ø | , | - | . | / | : | ;

For example: MM-DD-YY or MMDDYY

This format representation applies to function 1:

ER -> IAR

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

IBR : Internal binary representation
Storing a date in 3 bytes in the form YYMMDD where years, months and days are coded in binary notation and stored in 1 byte each.

STR : System time representation
Bytes 1,2: Hours since 1.1.1980 (binary)
Bytes 3,4: Parts of the current hour in units of 1/10th second (binary)

EAR : Extended ASCII representation
The representation is in 12 bytes of ASCII in the form YYMMDDhhmmss.

CW : The date is converted to a calendar week. The output is in 4 bytes of ASCII in the form YYWW.

Status messages in <NVar>:

Status	Meaning
0	Function performed without errors
1	Invalid date detected
2	- Specified displacement less than 1 - Not enough space in the destination string to hold the date - Not enough space in the source string for a valid date
3	For functions 0, 1 and 9 only: The date 999999 was specified. In function 9, the system date is not changed if this is specified.
4	For functions 0, 1 and 9 only 6 blanks were specified as the date. In function 9, the system date is not changed if this is specified.

Possible error: ERROR # 38:

- Incomplete parameter list
- Invalid function number
- Invalid representation

CALL Statements

8.19 CALL 35

Disk configuration

CALL 35 supplies information from the CONFIG file about the currently configured disks.

Syntax:

CALL <NExpr>, <SVar>, <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 35).

<SVar> : String in which the information is stored.

<NVar> : Status variable
0 = Function executed without errors
1 = Dimension of <SVar> is too small

After the function has been executed, the following is returned in <SVar>, depending on the type of drive configured:

- Length of entry for this type in the string.
- Number of physical units per drive of this type.
- Fixed/removable disk identifier for each physical unit.

A distinction is made between the following types:

- 0 = SMD/FDU
- 1 = CA
- 2 = CMD
- 3 = CDD
- 4 = FD 8
- 5 = MDL
- 6 = FD 5 1/4
- 8 = CDH

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<SVar> has the following contents after CALL 35 has been executed:

Bytes	Contents
1,2	First configured drive type
3,4	Entry length in bytes for this type
5,6	- Number of physical units for drive 0
7,k	Fixed/removable disk identifier for PUs (one byte for each PU, F = fixed, W = removable)
k+1,k+2	- Number of physical units for drive 1 (if configured)
k+3,l	Fixed/removable disk identifier for PUs
l+1,m	- Analogous for all other drives of this type
:	:
m+1,m+2	Second configured drive type
m+3,n	Analagous to first drive type
:	:
:	:
n+1,p	All other active drive types
:	:
:	:
p+1	Delimiter

Note: The entry length in bytes for the configured drive type refers to the total entry including the 2 bytes for the drive type and the 2 bytes for the entry length itself.

CALL Statements

- Example:

  There is the following disk configuration:

Drive no.	Type	Physical units/drive
0	SMD	2
1	SMD	2
2	FDU	4
0	CA	1
1	CA	1
0	CMD	2
0	CDD	2
0	FD 8	1
1	FD 8	1
0	MDL	2
4	FD 5	1

  <SVar> has the following contents after execution of CALL 35 with the configuration above:

Bytes	Contents	
1 , 2	00	SMD
3 , 4	18	Entry length
5 , 6	02	PUs on drive 0
7 , 8	WW	Removable disks
9 , 10	02	PUs on drive 1
11,12	WW	Removable disks
13,14	04	PUs on drive 2
15,18	FFFF	Fixed disks
19,20	01	CA
21,22	08	Entry length
23,24	02	PUs
25,26	FW	Fixed/removable
27,28	02	CMD
29,30	08	Entry length
31,32	02	PUs
33,34	FW	Fixed/removable

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Bytes	Contents	
35,36	03	CDD
37,38	08	Entry length
39,40	02	PUs
41,42	FF	Fixed disk
43,44	04	FD 8
45,46	10	Entry length
47,48	01	PU
49,49	W	Removable
50,51	01	PU
52,52	W	Removable
53,54	05	MDL
55,56	08	Entry length
57,58	02	PUs
59,60	FF	Fixed disk
61,62	06	FD 5
63,64	07	Entry length
65,66	01	PU
67,67	W	Removable

CALL Statements

8.19-1 CALL 43

This CALL supports job processing by transferring the parameters and storing them in the job list. In addition the entry date and sender are recorded. The job is classified as waiting.

Syntax:

```
CALL  <NExpr>, <NVar1>, <NVar2>, <NVar3>, <SVar1>,
      <SVar2>, <SVar3>, <SVar4>, <SVar5>, <NVar4>,
      <NVar5> [, <SVar6>]
```

<NExpr> : The value of this numeric expresion is the number of the CALL subroutine (in this case 43).

<NVar1> : This variable contains the function number:

1: Create job
2: Read job

<NVar2> : This variable contains the job number.

For function 1: Return parameter
For function 2: Specification parameter

<NVar3> : Status:

0: Function successful
1: Incorrect parameter array
2: Invalid function number
4: LU not active, bad RDA
5: Job list not found
6: Job does not exist (function 2)
10: Addressee does not exist (function 1)

<SVar1> : Job description (20 characters)

<SVar2> : Program name/number (18 characters)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

```
*           <SVar3>      : Program parameters (40 characters)
*
*                            (1,5)   Run mode
*                            (6,13)  Logical units accessed (like transfer of
*                                    standard parameters by TAMOS)
*                            (14,19) Selector parameters (like transfer by
*                                    TAMOS)
*                            (20,40) Reserved for future use
*
*                            Unused parameters have to be filled with blanks. A
*                            delimiter terminates the transfer.
*
*           <SVar4>      : Job status flag (10 characters)
*
*                            (1,1)   "1": Job cannot be deleted
*                            (2,10)  Reserved for future use
*
*           <SVar5>      : Deadline (10 characters)
*                            Format: YYMMDDhhmm or delimiter
*
*           <NVar4>      : User number or organizational unit number of
*                            addressee
*
*           <NVar5>      : Priority
*
*           <SVar6>      : Freely selectable parameters (350 characters)
*                            These are stored in the application program's
*                            common area behind the TAMOS parameters.
*
*           BASIC error 38 is generated if any of the first three variables
*           is not numeric.
```

CALL Statements

8.19-2 CALL 44

This CALL deletes a job from the job list. The absolute job number has to be specified. To be on the safe side, this function should be preceded by the "Read job" function (see CALL 43) in order to check some characteristics that are significant for the job.

Syntax:

CALL <NExpr>, <NVar1>, <NVar2>, <NVar3>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 44).

<NVar1> : Function number

1: Delete job

<NVar2> : Job number (absolute number in the job list)

<NVar3> : Status:

0: Function successful
1: Invalid function number
4: LU not active, bad RDA
5: Job list not found
6: Job does not exist
9: Job cannot be deleted

BASIC error 38 is generated if any of the three variables is not numeric.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.20 CALL 51

Editing of strings with octal codes to support display programming.

Attention:

CALL 51 is designed for displays with 27 lines of 74 characters each (Hazeltine).

Syntax:

```
CALL <NExpr>, <SVar1>, <SVar2>, <SVar3>, <SVar4>,
          <SVar5>, <SVar6>, <SVar7>, <SVar8>, <SVar9>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 51).

<SVar1> : String variable, length 74 bytes.
The values 1 to 73 (201 to 220, 361 to 364 and 225 to 311 octal) are stored in bytes 1 to 73 on execution. Byte 74 receives the value 0 (200 octal). These values are used to specify the cursor positions (position and line).

<SVar2> : String variable, length 2 bytes.
<SVar2> holds the control characters for cursor positioning (376, 221 octal).

<SVar3> : String variable, length 6 bytes.
<SVar3> is used to hold 6 DEL characters (377 octal).

<SVar4> : String variable, length 2 bytes.
On execution, <SVar4> holds the control characters for "clear screen" (376, 234 octal).

<SVar5> : String variable, length 2 bytes.
<SVar5> is used to hold the control characters for "clear foreground fields" (376, 235 octal).

<SVar6> : String variable, length 2 bytes.
<SVar6> is used to hold the control characters for "start background output" (376, 231 octal).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<SVar7> : String variable, length 2 bytes.
<SVar7> is used to hold the control characters for "start foreground output" (376, 237 octal).

<SVar8> : String variable, length 2 bytes.
<SVar8> is used to hold the control characters for "insert line" (376, 232 octal).

<SVar9> : String variable, length 2 bytes.
<SVar9> is used to hold the control characters for "delete line" (376, 223 octal).

The display functions are selected after CALL 51 has been executed by outputting the appropriate strings by means of the PRINT statement.

Only the DEL characters in <SVar3> constitute a special instance; they are used to prevent the screen being addressed again for a period of between 5 and 6 milliseconds which is required by the following functions:

- Clear screen
- Clear foreground fields
- Insert line
- Delete line

● Example:

Execution of CALL 51 followed by:

- Clear screen
- Place cursor on position 10 in line 2

```
:
150 CALL 51,N$,N1$,N2$,N3$,N4$,N5$,N6$,N7$,N8$
160 PRINT N3$;N2$;N1$;N$(10,10);N$(2,2);
```

CALL Statements

8.21 CALL 59

Converting numeric data to packed decimal numbers in IBM format and vice versa. Only integers can be converted. The two conversion directions are distinguished by the syntax:

Format	Syntax
1	CALL <NExpr>, <String> ,<NVar>
2	CALL <NExpr>, <NumPar> ,<SVar>

Format 1 must be selected when converting from IBM format.

Format 2 must be selected when converting to IBM format.

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 59).

<String> : Packed decimal number in IBM format. Each byte contains two BCD digits and the last tetrad must contain the sign: 1100 = +, 1101 = -. The digits are right-justified to the sign. The first tetrads must contain 0 if a number is to be converted which contains less digits than the available tetrads.

<NVar> : Numeric variable (1-4%) which holds the value converted from IBM to BASIC format. If there are more IBM digits than can be held as significant digits in the variable, the excess digits are lost to the detriment of precision.

<NumPar> : Numeric value to be packed according to IBM format.

<SVar> : String variable to hold the value in packed IBM format. The dimensioned length must not be less than the value defined by <NumPar> (decimal places are not taken into account):

INT ((No. of digits + 2)/2)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Examples:

  - Converting the number

    +0.123456E+06 = 123456

    stored in a 2% variable to packed notation in IBM format.

```
10 DIM A$(4),2%,N              /* DIMENSIONING
 :                             /* PREPARATION OF N
70 CALL 59,N,A$                /* CONVERSION
```

    Contents of A$ after conversion:

    0000 0001 0010 0011 0100 0101 0110 1100

     0   1   2   3   4   5   6   +

  - Converting the number

    +98765432

    stored in packed IBM format in A$ to a 2% variable (N).

    Contents of A$ (length 5 bytes):

    0000 1001 1000 0111 0110 0101 0100 0011 0010 1100

```
11 DIM A$(5),2%,N
 :
70 CALL 59,A$,N
```

    Contents after conversion:

    +0.987654E+08 = 98765400

    Because the number was converted to a 2% variable (which can only hold 6 significant digits), the precision of the last two positions was lost. Conversion to a 3% variable would have been better.

Possible errors: ERROR # 38

- BASIC number too high for IBM format string.
- IBM format string too long for 1% variable.

CALL Statements

8.22 CALL 60

Converting numeric ASCII characters into 4-bit format in accordance with the conversion table. Output is also in a string (packed string).
This subroutine enables numeric data to be stored in a space-saving manner on external storage media.

Syntax:

CALL <NExpr>, <String>, <SVar> [,<NVar>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 60).

<String> : The contents of this string are converted into 4-bit format and transferred to <SVar>. This string may only contain numeric ASCII characters and the special characters ß . , + -.

<SVar> : The contents of <String> converted into 4-bit format are stored in this string variable. Transfer to the destination string is concluded by a null tetrad. <SVar> should be dimensioned for at least half as many bytes as the number of characters converted.
Conversion is terminated when:
- the end of <String> is reached or
- the dimensioned length of <SVar> is reached.

<NVar> : Status variable
0: Executed without errors
1: Dimension of <SVar> too small
2: Unpackable character in <String>

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Conversion table:

Character	4-bit code
Delimiter	0000
0	0001
1	0010
2	0011
:	:
9	1010
∅	1011
+	1100
,	1101
-	1110
.	1111

- Example:

```
:
100 LET A$="123.45"
105 CALL 60,A$,B$
```

Contents of B$ after execution of CALL 60:

0010 0011 0100 1111 0101 0110 0000 0000

Possible error: ERROR # 38

Meaning:

An illegal character has occurred in <String>.

CALL Statements

8.23 CALL 61

Reconverting the contents of a string which was prepared by means of CALL 60 (unpacking).

Syntax:

CALL <NExpr>, <SVar1>, <SVar2> [,<NVar>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 61).

<SVar1> : The contents of this string variable can have been prepared by means of CALL 60. The digits stored in 4-bit format are converted into ASCII and transferred to <SVar2>.

<SVar2> : The numeric characters from <SVar1> converted into ASCII are stored in this string variable. <SVar2> must be dimensioned for as many bytes as the number of characters converted.
Conversion is terminated when:
- a tetrad occurs in which all bits are set to 0 or
- the dimensioned length of <SVar1> is reached.

<NVar> : Status variable
0: Executed without errors
1: Dimension of <SVar2> less than <SVar1>

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example:

  Converting the contents of variable A$ which were prepared by CALL 60 into the variable B$.

  The contents of A$ in binary notation:

  0010 0011 0100 0101 0110 0001 0001 0000

```
:
10 DIM A$(4),B$(10)     /* DIMENSIONING
:
200 CALL 61,A$,B$       /* CONVERSION
```

  Result: B$="1234500"

Possible error: ERROR # 38

Meaning:

Dimension of <SVar2> destination variable too small.

CALL Statements

8.24 CALL 62

Checking whether the contents of a string solely comprise digits (format 1).

Checking the syntax of a string (format 2)

Format	Syntax
1	CALL <NExpr>, <String1> [, <NVar2>]
2	CALL <NExpr>, <String1> ,<NVar1>, <NVar2>, <String2> [, <String3>] [,<NumPar1> [,<NumPar2>]]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 62).

<String1> : String whose contents are to be checked.

<NVar1> : Displacement of the first character to be checked in <String1>.
If the value is 0, checking starts at displacement 1.

<NVar2> : Status variable.

<String2> : Specification of a set of characters which are allowed as the first character in <String1>. There are two possibilities:
1. Listing all permissible characters (without separators)
2. Ascending sequence of characters in octal ASCII code specifying the first and last character. The separator is octal code ←377←.

<String3> : Specification of a set of characters as in <String2> which applies to the other characters in <String1>.
If this specification is omitted, <String2> is used as a substitute.

<NumPar1> : Maximum length of character sequence to be checked. This value must be positive. If it equals 0 or is omitted completely, <String1> is checked up to the end.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<NumPar2> : Minimum length of the character sequence to be checked. This value must be positive. If it equals 0, a null string is accepted. If it is omitted completely, the default value is 1.

Status messages

Status numbers are returned in <NVar2> and have the following meaning:

0: Executed without errors
1: Illegal character in <String1>
2: Delimiter occurred before the minimum length to be checked was reached; the string is too short.
3: Incorrect parameter contents:
- Numeric values greater than 65535 or negative
- Minimum number of characters to be checked exceeds maximum number
- <String1> has more than 2000 characters
- Displacement of first character to be checked exceeds 2000
- Displacement of first character to be checked is not in <String1>
- Minimum number of characters to be checked exceeds 2000

4: <String2> or <String3> is a null string.

Return of displacement

After the syntax check, a displacement is returned in <NVar1>, depending on the status.

Status	Displacement
0	Displacement of last character checked
1	Displacement of first incorrect character
2	Displacement of delimiter
3	Initial value or 0 if this is negative or exceeds 65535
4	Initial value

CALL Statements

- Examples

  - Checking that the contents of A$ are digits only

```
:
150 CALL 62,A$
```

  - Checking contents of A$ for a valid file name

```
 :
 90 LET A$ = "11$ABCDEFGHIJ234AA"
100 LET B$ = "$A←377←Z"
110 LET C$ ="0←377←9.A←377←Z"
120 LET A=3
130 LET C=14
140 LET D=0
150 CALL 62,A$,A,B,B$,C$,C,D
160 PRINT A
170 PRINT B
```

  Result: A=16 : Last character processed
          B= 0 : Status

Possible error: ERROR # 38

Meaning:

Parameters missing or incorrect variable types specified.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.25 CALL 63

Checking whether the contents of a string constitute a numeric value.
A leading sign (- or +) and a decimal point (or comma) at any position are allowed in addition to the digits 0 to 9.

Syntax:

CALL <NExpr>, <String> [,<NVar>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 63).

<String> : String whose contents are then checked as to whether they constitute a numeric value.

<NVar> : Status variable
0: <String> contains a numeric value
1: <String> does not contain a numeric value

- Example:

Checking the contents of A$.

```
:
150 CALL 63,A$
```

Possible error if <NVar> is not coded: ERROR # 38

Meaning:

- An illegal character was found in <String>.
- No digit, only a sign or decimal point was found in <String>.

Note:

A null string does not result in an error message.

CALL Statements

8.26 CALL 64

Checking whether a string contains a valid date. The following format is prescribed for the date:

MM.DD.YY or MM,DD,YY

If the date is found to be valid, it is converted and stored in the string in the format:

YYMMDD

Syntax:

CALL <NExpr>, <SVar> [,<NVar>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 64).

<SVar> : String variable which must be dimensioned for a minimum of 8 bytes. The contents of this string are then checked as to whether they contain a valid date in the above-mentioned format. If the date is valid, it is converted to the format YYMMDD in <SVar>.

<NVar> : Status variable
0: Executed without errors
1: Invalid month specified
2: A separator is neither a point nor a comma or the two separators are different
3: Invalid day specified
4: Invalid year specified

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example:

Checking and converting the date stored in A$.

```
:
10 DIM A$(8)
:
100 CALL 64,A$
```

Contents of A$:

	Before	After
	02.01.78	780201

Possible error if <NVar> is not coded: ERROR # 38

Meaning:

- <SVar> dimensioned for less than 8 bytes.
- A non-numeric character found at one of the positions 1, 2, 4, 5, 7 or 8.
- The 3rd or 6th character is not a point or comma.
- Point and comma mixed in a string.
- Month less than 1 or greater than 12.
- Day less than 1 or greater than 30 or 31. The 29th February is only permissible in leap years.

CALL Statements

8.27 CALL 65

Sorting keys in ascending order (ASCII) which are stored in a string.

Syntax:

```
CALL <NExpr>, <NVar1>, <NumPar1>, <NumPar2>,
                                  <SVar1>, <SVar2>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 65).

<NVar1> : This numeric variable is used to transfer the status from the subroutine to the BASIC program. The contents of <NVar1> must be 0 when the subroutine is called.

<NumPar1> : The number of keys to be sorted (>= 1).

<NumPar2> : The sort key length (>= 1)

<SVar1> : The contents of this string variable are sorted in ascending order in accordance with the parameters specified in <NumPar1> and <NumPar2>. Delimiters are not stored after the sort procedure has been completed. The minimum length of <SVar1> is the product of <NumPar1> * <NumPar2>.

<SVar2> : This string variable acts as a work area. Its minimum length is <NumPar2> + 8.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example:

Sorting the sort keys stored in A$. A sort key has a length of 5 characters. The number of keys to be sorted is 10:

Contents of A$ before:

Sort key length (5)

TOWER1234512355FIELDCYCLESLEEPBRAIN77777TRAINTRUNK...

10 sort keys

```
10 DIM A$(100)                  /* DIM. SORT STRING
15 DIM W$(13)                   /* DIM. WORK AREA
:
100 LET S=0                     /* STATUS TO 0
115 CALL 65,S,10,5,A$,W$        /* SORT
120 ON S GOSUB                  /* ANALYSE STATUS
```

Contents of A$ afterwards:

123451235577777BRAINCYCLEFIELDSLEEPTOWERTRAINTRUNK...

Status messages in <NVar1>

<NVar1>	Meaning
0	Sorting performed without error.
1	Parameter error: - Parameter(s) missing. - Incorrect variable type. - <NVar1> not equal to 0 when called. - <NumPar1> and/or <NumPar2> negative or greater than 32767. - Contents of <NumPar1> or <NumPar2> less than 1.
2	<NumPar1> or <NumPar2> = 0.
3	Dimension of <SVar1> too small.
4	Dimension of <SVar2> too small.
5	A variable name was used more than once.

CALL Statements

8.28 CALL 66

CALL 66 has been available since NIROS 5.1/00 for an extended code conversion function in which one source byte can be converted to up to 13 destination bytes.

The extended code conversion function is called in BASIC programs as follows:

Syntax:

```
CALL 66, <NVar1>, <NVar2>, <SVar1>, <SVaR>; <SVar3>,
                                  <SVar4>, <SVar5>, <NVar3>
```

<NVar1> : The number of source bytes to be processed must be stored in <NVar1> before the function is called. Since the destination string can expand by a factor of 13, it might not be possible to process the entire source string. For this reason, the actual number of bytes processed is stored in <NVar1> after the function has finished.

<NVar2> : The dimensioned (!) length of the destination string must be supplied in <NVar2> before the function is called. <NVar2> contains the number of destination bytes after the function has finished.

<SVar1> : Source string

<SVar2> : Destination string

<SVar3> : Tables which have to be filled before the function is called: bytes 1 - 256 contain the destination code table, bytes 257 - 512 contain the pointer table.

<SVar4> : Table for replacement strings 1 - 36.

<SVar5> : Table for replacement strings 37 - 72.

<NVar3> : Status variable
0 = Function successful
1 = Function unsuccessful because <NVar1> or <NVar2> contained illegal values.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

BASIC error 38 is output in the event of format errors.

It is expedient to read the tables <SVar3>, <SVar4> and <SVar5> from files.

- Example:

  This program example is based on the tables used in the following description of "Functioning of CALL 66".

```
:
100 IF  ERR 0 STOP
110 DIM Q$(20),Z$(260),C1$(512),C2$(512),C3$(512)
120 OPEN #3,"XY.CODE"
130 MAT  READ #3;C1$;
140 MAT  READ #3;C2$;
150 MAT  READ #3;C3$;
160 CLOSE #3
170 INPUT 'CR',"SOURCE  : ",Q$
180 IF Q$="" CHAIN ""
190 LET Q= LEN Q$
200 LET Z=260
210 CALL 66,Q,Z,Q$,Z$,C1$,C2$,C3$,S
220 IF S STOP
230 PRINT 'CR';"DESTIN.  : ";Z$(1,Z);'CR';
240 GOTO  170
```

CALL Statements

Functioning of CALL 66

The function of CALL 66 is described below by means of an example.

Problem:

Texts represented in ASCII have to be processed so that lower case characters are converted to the corresponding upper case characters, umlauts to the respective two characters (e.g. "Ä" = "AE") and "ß" to "SS".

Hex codes:

```
A ... Z  =  C1 ... DA
a ... z  =  E1 ... FA
Ä, Ö, Ü  =  DB, DC, DD
ä, ö, ü  =  FB, FC, FD
ß        =  E0
```

As in CALL 31, a destination code table is needed first. The value of a source byte defines an address in the destination code table, the contents of which determine whether the source byte has to be replaced by a single or several destination bytes.

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example of destination code table:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
3	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
4	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
5	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
6	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
7	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
8	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
9	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
													****			
D	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	FF	FF	FF	DE	DF
		****											****			
E	FF	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
		****														
F	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	FA	FF	FF	FF	FE	FF

If the address in the destination code table contains a value other than 'FF', this value is taken as the replacement character. For instance, the lower case letter "b" equals 'E2' and is replaced by 'C2', the code for "B".

If the address in the destination code table contains the value 'FF' (the address 'DD' in the example above), a pointer to the replacement string must be stored at the same address in a pointer table.

CALL Statements

- Example of a pointer table:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
													****			
D	00	00	00	00	00	00	00	00	00	00	00	00	02*03*00			00
													****			
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

The letter "Ü" = 'DD', for example, has to be replaced by the 3rd string in a separate replacement string table.

The replacement string table consists of substrings of 14 bytes each. The first byte in a substring defines in binary notation the length of the replacement string. Bytes 2 to 14 contain the actual replacement string which can be up to 13 characters long.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example of a replacement string table:

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0	02	C1	C5	00	00	00	00	00	00	00	00	00	00	00	02	CF
													**********			
1	C5	00	00	00	00	00	00	00	00	00	00	00*02*D5			C5*00	
													**********			
2	00	00	00	00	00	00	00	00	00	00	02	D3	D3	00	00	00
3	00	00	00	00	00	00	00	00	01	FF	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

The byte position of the 3rd substring is calculated by the formula (3-1)*14 = 28 = '1C'. Address '1C' contains the length of the replacement string. This is followed immediately by the replacement string. In this way, "Ü" is replaced by 'D5C5' which is "UE".

Maintaining code tables

Coordinates 5/1/2 and 5/1/3 in the EXPERT selector contain the "CREATE CODE TABLES" and "CHANGE CODE TABLES" functions to maintain XCT code tables for the 2-channel PLC.

These code tables must be called XCT0 to XCTZ. These names are reserved for use in data communication but it is possible to get round this and use these functions to process any code tables.

If other code tables are intended to be created, this can be done first with "CREATE CODE TABLES", calling the code table XCTx (x = 0 ... Z). Then the BASIC statements MAT READ and MAT WRITE can be used to transfer the code tables to the actual code table file that is required.

CALL Statements

If a code table created in this way has to be changed, it must first be transferred to an XCTx file by means of MAT READ and MAT WRITE again.

The XCTx files are microprocessor files consisting of one record of 512 bytes. They contain one code table at positions 1 - 256 and another at 257 - 512. These two code tables are called input and output table in the screen mask.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

8.28-1 CALL 68

CALL 68 is used to communicate between the application and the workstation by telling the workstation that a new configuration has to be loaded. It only works with more recent workstations however (DAP 4x, DAP 6 ff.).

Syntax:

CALL <NExpr>, <NVar1>, <NVar2>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 68).

<NVar1> : This variable specifies the required row in the configuration module. Valid range: 1-255

<NVar2> : The status is returned in this variable.

Possible statuses:

0: Loading/reloading successful

1: Configuration file or specified row does not exist

2: Time-out (after 63 retries)

3: Module does not exist on the system disk, wrong module loaded or configuration incomplete.

4: Workstation type is not reloadable (RAP, old DAP, DAP 4)

CALL Statements

8.29 CALL 70

CALL 70 forms the interface between the BASIC interpreter RUN and the channel program $MTX for magnetic tape units.

Only sequential files can be stored on magnetic tape. File management must be controlled by the application program.

A distinction is made between the following four formats:

Format	Syntax
1	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>
2	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>, <NumPar3>, <NVar2>
3	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>, <NumPar3>, <NVar2>, <SVar1>
4	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>, <SVar2>, <SVar3>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 70).

<NumPar1> : Number of the tape unit to be addressed (0 to 3).

<NumPar2> : Number of the function to be executed (see description of functions).

<NVar1> : Numeric variable in which the status is returned after execution of the function. <NVar1> must be 0 before the function is executed.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

8

CALL Statements

<NumPar3> : Number of tape marks, blocks or bytes to be processed (dependent on the function).

<NVar2> : Actual number of tape marks, blocks or bytes processed.

<SVar1> : String variable to hold the block to be read or written.

<SVar2> : String variable to hold an input code table. This table must be loaded by the application program and have a length of 256 bytes.

<SVar3> : String variable to hold an output code table. This table must be loaded by the application program and have a length of 256 bytes.

The format to be selected depends on the required function.

Functions for format 1:

Number	Function
1	Rewind tape to BOT (beginning of tape).
2	Switch tape unit off-line.
6	Write tape mark.
7	Write tape gap.
10	Write tape mark and rewind tape to BOT.
11	Open tape unit for processing without code conversion.
12	Open tape unit for processing without code conversion and rewind tape to BOT.
13	Close tape unit.
14	Close tape unit and switch off-line.

Functions for format 2:

Number	Functions
3	Forward-space tape by the number of blocks specified in <NumPar3>.
4	Backspace tape by the number of blocks specified in <NumPar3>.
8	Forward-space tape by the number of tape marks specified in <NumPar3>.
9	Backspace tape by the number of tape marks specified in <NumPar3>.

Note:

The number of blocks/tape marks is specified in <NumPar3>. After the operation has been performed, <NVar2> contains the actual number of blocks/tape marks which have been processed. This number can be less than the number specified in the statement, but a status <> 0 is returned in <NVar1> in this case.

Functions for format 3:

Number	Function
0	Read block. Transfer is terminated when the dimensioned length of <SVar1> or the number of characters specified in <NumPar3> is reached.
5	Write block. Transfer is terminated when the dimensioned length of <SVar1> or the number of characters specified in <NumPar3> is reached.

Note:

After the function has been executed, the actual block length read or the number of characters actually written is returned in <NVar2>. This value need not match the value specified in <NumPar3>.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Functions for format 4:

Number	Function
11	Open magnetic tape unit for processing with code conversion. <SVar3> need only be specified for output to magnetic tape.
12	As for function 11, but with rewinding of the tape to BOT.

Note:

Conversion tables must be provided in <SVar2> and <SVar3> by the BASIC program and must not be altered after the magnetic tape has been opened. The following standard code tables are available in contiguous files on the system:

File name	Table
ASCII7.ASCII8	Conversion from ASCII without set bit 8 to ASCII with set bit 8.
ASCII8.ASCII7	Conversion from ASCII with set bit 8 to ASCII without set bit 8.
ASCII.EBCDIC	Conversion of ASCII with set bit 8 to EBCDIC.
EBCDIC.ASCII	Conversion of EBCDIC to ASCII with set bit 8.

Returning the status

After a magnetic tape operation has been performed, the status in <NVar1> must be checked. The following values can be returned in <NVar1> by CALL 70:

<NVar1>	Meaning
0	Operation successful
1	Tape unit not available (already opened)
2	Tape unit not ready (off-line)
3	Tape is write-protected (no write-enable ring)
4	Read/write error
5	EOF tape mark read
6	EOT reflector
7	Illegal operation

CALL Statements

8.30 CALL 72

Transferring the contents of up to 11 numeric values or strings to a string. Transfer is carried out irrespective of the type and format of the source data.

Syntax:

CALL <NExpr>, <SVar1> [, <NumPar>]^{11} [, <String>]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 72).

<SVar1> : This string variable is used to hold the variables and constants (up to 11) specified in the following. If a numeric variable is specified here, no error message is issued.

<NumPar> <String> : Maximum of 11 numeric values or strings which are to be transferred to <SVar1>. Transfer is terminated when the dimensioned end of the destination variable or the end of the last source value is reached. Delimiters in source strings do not terminate transfer.
All transferred values are directly adjacent to one another.

- Example:

Transferring the contents of the variables A,B,C,A$,B$,C$ to the string M$.

```
:
10  DIM A$(9),B$(9),C$(9)
15  DIM 2%,A,B,C
20  DIM M$(39)                    /* DESTINATION STRING
:
100 CALL 72,M$,A$,B$,C$,A,B,C     /* TRANSFER
```

Contents of the variable M$ after execution:

Bytes	1...9	10...18	19...27	28...31	32...35	36...39
	A$	B$	C$	A	B	C

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.31 CALL 73

Transferring the contents of a string to a maximum of 11 variables. Transfer is carried out irrespective of the type and format of the destination variables.

Syntax:

CALL <NExpr>, <SVar1> $\left\{\begin{matrix}\text{,<SVar>}\\\text{,<NVar>}\end{matrix}\right\}^{11}$

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 73).

<SVar1> : The contents of this string variable are transferred to up to 11 optional variables.

<SVar>
<NVar> : A maximum of 11 variables to which the string is to be transferred. Transfer is terminated when:
- the dimensioned end of the source string is reached or
- the dimensioned end of the last destination variable is reached.

● Example:

The contents of the string M$ are transferred to the variables A,B,C,A$,B$,C$.

```
:
10 DIM A$(10),B$(10),C$(10)      /*DESTINATION VARIABLES
15 DIM 2%,A,B,C                  /*DESTINATION VARIABLES
20 DIM M$(42)                    /*SOURCE VARIABLE
:
100 CALL 73,M$,A,B,C,A$,B$,C$    /*TRANSFER
```

CALL Statements

8.32 CALL 74

Reading an entry in a port's device assignment table.

Syntax:

CALL <NExpr1>, <NumPar1>, <NumPar2>, <SVar1>, <SVar2>, <SVar3>

<NExpr1> : The value of this numeric expression is the number of the CALL subroutine (in this case 74).

<NumPar1> : Number of the port whose device assignment table is to be read.

<NumPar2> : Number of the entry to be read in the device assignment table. This must range between 0 and 12.
If the value "-1" is used instead of an entry number and a device program name is specified in <SVar1>, the subroutine returns the associated system name and the parameters of the device.

<SVar1> : Minimum of 14 bytes
The program name in the entry is stored here. If the entry is empty, a null string is stored.

<SVar2> : Minimum of 14 bytes
The system name in the entry is stored here. If the entry is empty, a null string is stored.

<SVar3> : Minimum of 10 bytes
All other parameters are stored here. If the entry is empty, a null string is stored.

Possible error: ERROR # 38

Meaning:

* - Incorrect variable type
- Value of <NumPar1> too high or less than 0
- Value of <NumPar2> greater than 12 or less than 0
- String variable incorrectly dimensioned

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

- Example:

Ascertaining the printer assignment ("$LPT") for the phantom port (port 1).

```
  10  DIM N$(14),E$(14),P$(10)
  :
1000  FOR I=0 TO 12
1010  CALL 74,1,I,N$,E$,P$
1020  IF N$="$LPT" GOTO 2000
1030  NEXT I
1040  REM NO PRINTER ASSIGNMENT
  :
2000  REM E$=ASSIGNED PRINTER
          P$=PRINTER PARAMETERS
```

CALL Statements

8.33 CALL 80

CALL 80 forms the interface between the BASIC interpreter RUN and the driver $CAS for the magnetic tape cassette unit.

Only sequential files can be stored on the cassette tape. File management must be controlled by the application program.

A distinction is drawn between three formats:

Format	Syntax
1	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>
2	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>, <NumPar3>, <NVar2>, <SVar>
3	CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar1>, <SVar1>, <SVar2>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 80).

<NumPar1> : Number of the MTC unit (must be 0).

<NumPar2> : Function number (see description of the functions).

<NVar1> : Numeric variable in which the status is returned after the function has been executed. <NVar1> must be 0 prior to execution of the function.

<NumPar3> : Number of bytes to be read/written (depending on the function). The value may not exceed the maximum block length of 512 bytes.

<NVar2> : The actual number of bytes which have been read or written is returned in this numeric variable after the function has been executed.

<SVar> : String variable to hold the data blocks which have been read or written.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

<SVar1> : String variable (256 bytes) to hold the input code table. The code table must be previously loaded by the application program.

<SVar2> : String variable (256 bytes) to hold the output code table. The code table must be previously loaded by the application program.

The format to be selected depends on the function to be executed.

Functions for format 1:

Number	Function
11	Open the tape cassette unit for processing and assign it to the port that opened it. The cover of the cassette unit is opened and "$CAS" waits 105 seconds for the cassette to be inserted and the cover to be closed. The tape is then rewound to the BOT (beginning of tape) mark. If no cassette is inserted during the waiting period, status 5 is issued.
13	Finish processing, rewind the cassette to the physical starting point and eject. The tape cassette unit is then available to other users.
3	Forward-space the cassette tape by one block, i.e. the tape is positioned to the next gap. If there is no futher block behind the current block, status 5 is issued after 25 seconds.
4	Backspace the cassette tape by one block, i.e. position to the last gap. If no block is available (current block = first block), status 5 is issued after 25 seconds.
1	Rewind cassette tape to BOT.
2	Open the tape cassette unit cover and eject the cassette.

CALL Statements

Functions for format 2:

Number	Function
0	Read data block and transfer to <SVar>. If the cassette has been opened with format 3, code conversion is carried out accordingly. Transfer is terminated as soon as the dimensioned length of <SVar> or the number of bytes specified in <NumPar3> is reached (<NumPar3> may not exceed the maximum block length of 512 bytes). After the function has been executed, the number of bytes actually transferred is returned in <NVar2>. After the transfer, the tape is positioned to the gap after the last block read. If errors occur during reading, the process is repeated up to 8 times. If the reading attempts are not successful, status 4 is returned.
5	Write block, i.e. the contents of <SVar> are written to tape starting from the current position. If the cassette has been opened with format 3, the code is converted accordingly. The transfer is terminated as soon as the dimensioned length of <SVar> or the number of bytes specified in <NumPar3> is reached. After the transfer, the number of bytes actually written is returned in <NVar2> and the tape is positioned to the gap after the block last written. In the event of write errors, the process is repeated up to 8 times. If the function is not successful, status 4 is returned.

Function for format 3:

Number	Function
11	The tape cassette unit is opened for processing with code conversion.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Note:

After opening, the variables <SVar1> and <SVar2> in which the code tables are stored must not be altered.

The following standard code tables are available in the system in contiguous files:

File name	Table
ASCII7.ASCII8	Converting from ASCII without set bit 8 to ASCII with set bit 8.
ASCII8.ASCII7	Converting from ASCII with set bit 8 to ASCII without set bit 8.
ASCII.EBCDIC	Converting from ASCII with set bit 8 to EBCDIC.
EBCDIC.ASCII	Converting from EBCDIC to ASCII with set bit 8.

Return of status

After a cassette operation has been performed, the status returned in <NVar1> must be checked.

Status	Meaning
0	Operation successful.
1	The tape cassette unit is not available (already opened by another user).
2	The tape cassette unit is not ready.
3	The tape is write-protected.
4	Read/write error.
5	Time-out.
6	EOT reached.
7	Illegal operation.

CALL Statements

8.34 CALL 84

Reading a message from a message file, displaying and/or entering it in the system logbook.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NVar> [,<NumPar2> [,<NumPar3>]]
    ,<SVar> [,<String>, <NumPar4>, <NumPar5> [,<NumPar6>]]
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 84).

<NumPar1> : Function number

Function	0	1	2	3	4	5	6
Reading a message from a message file	X	X	X	X			
Displaying the message in the message line		X		X			
Storing the message in the system logbook			X	X			
Displaying plain language description in the message line					X		X
Storing a plain language description in the system logbook						X	X

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

<NVar> : Status messages returned:
0: No error occurred
1: Incorrect parameters supplied
2: The message file to be read does not exist or no valid file name was specified in <SVar> or the specified file is not a message file.
3: The LU on which the specified message file is supposed to be located is not active.
4: No file is open on the specified channel or the open file is not a message file.
5: The specified table base and message number combined exceed the file size or the specified message number is greater than the highest error number in the MESSAGES file.

<NumPar2> : Type or area number
(Only for functions 2, 3, 5, 6)
If <NumPar3> is also specified, this is the type number with which the message is to be stored in the system logbook; otherwise it is the application area number.
If <NumPar2> and <NumPar3> are missing, type number 4 is allocated which means that it is a message from application software and the user number is entered as the type specification. If the value of <NumPar2> is out of range (0-255), the value 4 is inserted.
If an application area number is specified, the subroutine calculates the type number for logging purposes. If the type number is not included in the type code list for evaluating the system logbook, the type abbreviation does not exist either.

<NumPar3> : Type specification (0-255)
(Only for functions 2, 3, 5, 6)
The type specification with which the message is to be stored in the system logbook. If this specification is missing or out of range (0-255), the user number is entered.

<SVar> : Message or plain language description
This variable must be dimensioned for at least 55 bytes (50 bytes for message + 1 byte for blank + 4 bytes for message number). The plain language description must be entered here for functions 4-6. The text should not exceed 242 bytes because only the first 242 bytes of a description are entered in the system logbook.

<String> : File name (Only for functions 0,1,2,3)
Specification of the message file from which a message is to be read. This is specified in the form LU/file name. If the logical unit number is not specified, LU 0 is the default value. If a null variable is specified, the MESSAGES file is used.

<NumPar4> : Table base (Only for functions 0,1,2,3)
Specification of the table base relative to which the message number is to be evaluated (see "Logbook and Message File").
If <String> is a null string, this parameter is not evaluated but is defined by the subroutine itself.

<NumPar5> : Message number (Only for functions 0,1,2,3)
Specification of the number for the message which is to be read.

<NumPar6> : Channel number (Only for functions 0,1,2,3)
If the message is to be accessed via an open channel to improve the execution time of the subroutine, the appropriate message file must be opened by the BASIC program and the channel number must be specified with <NumPar6>. If the message file is not open, this specification must be omitted.

Possible error: ERROR # 38

Meaning:

Incorrect parameter or syntax error when calling the subroutine.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.35 CALL 90

Support for string variable processing.

A distinction is drawn between four formats:

Format	Syntax
1	CALL <NExpr>, <NumPar1>, <SVar1>, <NumPar2> {,<NumPar>}^{9}
2	CALL <NExpr>, <NumPar1>, <SVar1>, <NumPar2>, <SVar2>
3	CALL <NExpr>, <NumPar1>, <SVar1>, <NumPar2> [, <NumPar3>]
4	CALL <NExpr>, <NumPar1>, <SVar1>, <NumPar2>, <SVar2> [,<NumPar3> [,<NumPar4>]]

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 90).

<NumPar1> : Function number (see description of the functions).

<SVar1> : String variable containing the destination or source string (depending on the function).

<NumPar2> : Numeric value defining the first byte to be processed in <SVar1>. The value must be odd and positive and must not exceed the dimensioned length of <SVar1>.

<NumPar> : Depending on the function, one to nine numeric values defined as destination variables or source values (variables or constants).

<SVar2> : String variable used as the source string.

<NumPar3> : Last byte to be processed in <SVar1>. The value must be odd and greater than or equal to the value specified in <NumPar2>.

If <NumPar3> is not specified, the string is processed up to its dimensioned end; this is also the case if <NumPar3> is greater than the dimensioned length of <SVar1>.

<NumPar3> for format 4 : First byte in <SVar2> to be transferred. This value must be positive and must not exceed the dimensioned length of <SVar2>. If this specification is omitted, <SVar2> is transferred from the first byte onward.

<NumPar4> : Last byte in <SVar2> to be transferred. This value must not be less than <NumPar3>. If this specification is omitted, <SVar2> is transferred up to the last byte.

Description of functions for the formats of CALL 90

Functions for format 1:

Number	Function
0	Transferring one to nine numeric variables to a string variable. WRITE
1	Transferring the contents of a string variable to one to nine numeric variables. READ

Function for format 2:

Number	Function
2	Transferring the contents of <SVar2> to <SVar1> starting from address <NumPar2>. Transfer is terminated when the dimensioned length of the source or destination string or a delimiter in the source string is reached.

8

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Function for format 3:

Number	Function
3	Filling the string variable <SVar1> with delimiters word by word from byte <NumPar2> to byte <NumPar3> or up to the dimensioned end. If <NumPar2> is an even value, the next lower odd value is used.

Function for format 4:

Number	Function
4	A contiguous part of the source string <SVar2> ranging from byte <NumPar3> to byte <NumPar4> is transferred to the destination string <SVar1> and is stored from byte <NumPar2> onward. Transfer is terminated when the last required byte (<NumPar4>) or the dimensioned end of the source or destination string is reached. Delimiters in the source substring are transferred and do not terminate the transfer unless they indicate the dimensioned end. The source substring is transferred byte by byte. This can have unexpected results if the source string and destination string are identical.
5	A contiguous part of the source string <SVar2> ranging from byte <NumPar3> to <NumPar4> is inserted in the destination string <SVar1> starting at byte <NumPar2>. The original byte <NumPar2> and all following bytes are shifted accordingly to the right.

THE FIRST PARAMETER.

6. AS 5 BUT THE DIMENSIONED LENGTH

CALL Statements

- Example:

  Use of CALL 90, format 4, function 5:

```
10 DIM A$(3),B$(20)
20 LET A$="ABC"  /* SOURCE STRING
30 LET V=1  /* FROM BYTE 1 OF SOURCE STRING
40 LET B=3  /* TO BYTE 3 OF SOURCE STRING
50 LET B$="12345678901234"  /* DESTINATION STRING
60 LET P=7  /* INSERT SOURCE STRING FROM BYTE 7 ON
70 LET PRINT 'CS';"DESTINATION STRING BEFORE : ";B$
80 CALL 90,5,B$,P,A$,V,B
90 PRINT "DESTINATION STRING AFTER  : ";B$
```

  Result:

```
DESTINATION STRING BEFORE : 12345678901234
DESTINATION STRING AFTER  : 123456ABC78901234
```

  Lines 30, 40 and 60 can also be omitted. Line 80 would then read:

```
:
80 CALL 90,5,B$,7,A$,1,3
:
```

Possible error: ERROR # 38

Meaning:

- <NumPar2> is greater than the dimensioned length of <SVar1> or contains an even-numbered value (not for function 3).

- <NumPar3> is less than the value of <NumPar2>.

- <NVar> is greater than the dimensioned length of <SVar1>. <SVar1> is changed.

- <NumPar3> is greater than the dimensioned length of <SVar2> or contains an even-numbered value.

- <NumPar4> is less than <NumPar3>.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.36 CALL 91

Bit functions in strings. There are two different syntax formats:

Format	Syntax:
1	CALL <NExpr>, <NumPar1>, <SVar>, <NumPar2>, <NumPar3> [,<NumPar4>]
2	CALL <NExpr>, <NumPar1>, <String>, <SVar1>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 91).

<NumPar1> : Function number (see description of functions for meaning).

<SVar> : Source and destination string in which bits are to be inverted (function 0).

<NumPar2> : Byte address from which inversion is to take place (function 0).

<NumPar3> : Number of bytes whose contents are to be inverted (function 0).

<NumPar4> : This value added to the start address of the substring to be inverted provides the start address of the next substring to be inverted in <SVar>.
This value must be greater than or equal to the value of <NumPar3>. If the specification is omitted or equals 0, the value is defaulted to 1.

<String> : First source string for functions 1 to 6.

<SVar1> : Second source string as well as destination string for functions 1 to 6.

CALL Statements

Description of functions:

Format 1:

Number	Function
0	Inverting bits in substrings in a string. The substrings have the same length and start addresses with the same step.

- Example:

  Contents of A$: "12345678901234567890123456789O"
  Start address: 16
  Number of bytes to be inverted: 3
  Step between substrings: 5

  The bits in bytes 16 - 18, 21 - 23 and 26 - 28 are inverted with the aid of the statement

  ```
  150 CALL 91,0,A$,16,3,5
  ```

Format 2:

Number	Function
1	Logical AND operation on corresponding bits in the source strings.
2	Logical NAND operation on corresponding bits in the source strings.
3	Logical OR operation on corresponding bits in the source strings.
4	Logical NOR operation on corresponding bits in the source strings.
5	Logical XOR operation on corresponding bits in the source strings.
6	Logical NXOR operation on corresponding bits in the source strings.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

Explanation of logical operations:

Bits in 1st source string	0	0	1	1
Bits in 2nd source string	0	1	0	1
AND	0	0	0	1
NAND	1	1	1	0
OR	0	1	1	1
NOR	1	0	0	0
XOR	0	1	1	0
NXOR	1	0	0	1

Possible error: ERROR # 38

Meaning:

Step between substrings less than substring length (function 0).

8.37 CALL 93

Supplying information about installed disk units to a BASIC program.

Syntax:

```
CALL  <NExpr>, <NumPar1>, <NVar2>, <NVar3>, <NVar4>,
                                    <NVar5>,<NVar6>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 93).

In the description which follows, the first specification applies to multifile structure and the second specification to non-multifile structure.

<NumPar1> : = 1%, specification of the
- number of the installed ILU,
- number of the installed LU,
about which information is wanted.

<NVar2> : = 1%
After execution, this variable contains:
- The number of ILUs set up per LU
- The value 1

CALL Statements

<NVar3> : At least 2%
After execution, this variable contains:
- The total number of addressable blocks on the LU defined by the the specified ILU
- The number of blocks on the LU specified by <NumPar1>

<NVar4> : At least 2%
After execution, this variable contains:
- The first unused RDA
- The number of blocks per LU

<NVar5> : At least 2%
After execution, this variable contains the number of free blocks.

<NVar6> : = 1%, status variable
= 0: processed correctly
= 1: LU was not installed

Possible error: ERROR # 38

Meaning:

Parameter, type or dimension incorrect.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.38 CALL 96

Validity checks for file names and reading information about files.

Format	Syntax
1	CALL <NExpr>, <NumPar1>, <NVar2>, <String>, <NVar3>, <NVar4>
2	CALL <NExpr>, <NVar1>, <NVar2>, <String>, <NVar3>, <NVar4>, <NVar5>, <NVar6>, <NVar7>, <NVar8>, <NVar9>, <NVar10>, <NVar11>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 96).

<NumPar1> : Function number: 1-5 with format 1
<NVar1> 6 with format 2
In format 1, the functions are processed sequentially starting at 1 and going up to the specified function.
In format 2, the LU number is reported.

Function 1:

Formal check of the file name.
<LU>/<FileName> <T> is permissible

where

<T> ::= ; | , | = | ! | & | $ | > | < | - | delimiter (←000←) |
blank (←240←) | 'CR' (←215←) | Zero (←200←)

Leading blanks are allowed but ignored.

Function 2:

Checking whether the LU is active.
After function 1 has been performed, a check is made as to whether the specified LU is installed.

Function 3:
After functions 1 and 2 have been performed, a check is made as to whether the file is entered in the INDEX file. Status 13 is reported if this is not the case.

Function 4:

Not allocated.

Function 5:

After functions 1-3 have been performed, a check is made as to whether the file is open by another port. There are 3 possible states:

- File not open
- File open without lock: status 15
- File open with lock: status 21

Function 6:

Reading file information

<NVar2> : Status variable

<String> : File name - must be dimensioned for at least 15 bytes or be at least 15 bytes long

<NVar3> : After execution of the function, this contains:
- The displacement of the end of the string or the first incorrect character in the string in functions 1-5
- The user number in function 6

<NVar4> : In functions 1-5:
Specification of the LU number; this is only taken into consideration if no LU number is ascertained in the string. The specification -1 is permitted for functions 1 and 2.
After the return from the CALL subroutine, this variable contains the LU number.
In function 6:
After return, this variable contains some file parameters (see CALL 97).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

<NVar5> : After execution, this contains the file size in blocks.

<NVar6> : After execution, this contains the file status. A value other than 0 indicates a formatted file.

<NVar7>
<NVar8>
<NVar10> : Dummy variables which are only necessary to retain compatibility with earlier release levels.

<NVar9> : After execution, this contains the time the file was created in relation to 1.1.1980.

<NVar11> : After execution, this contains the disk address of the file header (decimal).

Status messages

Fct.	Status	Function error-free	Error	Meaning
All	0	X		Function error free
1	1		X	Illegal LU number
	2		X	Separator missing after LU number or illegal
	11		X	Illegal character in file name, no file name specified or file name exceeded 14 characters
2	12		X	LU not active
3	13		X	File does not exist
5	15	X		File open without lock
	21	X		File open with lock
All	63		X	Indefinite error

Possible error: ERROR # 38

Meaning:

- Variables or information missing, incorrect dimensioning of variables or incorrect contents.
- Illegal LU number specified in <NVar4>.
- Non-existent function specified in <NumPar1>/<NVar1>.
- <String> less than 15 bytes for function 6.

CALL Statements

8.39 CALL 97

Reading file information from the file headers using the INDEX file. The following information can be read from all files on the magnetic disks addressed:

- File name
- Account
- User number
- File type
- File size in blocks (sectors)
- File status
- Time of file creation (in hours, relative to 1.1.1980).
- Disk address of the file header.

The INDEX file is accessed via the relative record number within the file starting at 0.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NVar2>, <String>, <NVar3>,
     <NVar4>, <NVar5>, <NVar6>, <NVar7>, <NVar8>, <NVar9>,
                                         <NVar10>, <NVar11>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 97).

<NumPar1> : The logical unit number of the disk from which file information is to be read.

<NVar2> : The relative record number of the record to be read in the INDEX file. If this record is an empty record, the number of the next record in use is automatically inserted.
After successful execution, the contents of <NVar2> are increased by 1. If, however, the file end has been reached, -1 is returned.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

All the following variables are prepared by CALL 97 and contain the file information after execution of the CALL.

<String> : Length at least 15 bytes, to return the file name.

<NVar3> : At least 2%. The specification:

- user number

is returned to the application program in this variable.

<NVar4> : Some file parameters are encoded in this variable (at least 2%). The file type can be established using the formula

FRA (<NVar4> / 32) * 32

and the following table.

File types:
0: Permanent system file
1: System driver/processor
2: BASIC processor/program
3: Stand-alone processor/program
5: Microprocessor file
8: COBOL object file
9: Optimized BASIC program (D file)
10: PASCAL object file
24: Text file
25: Formatted file
26: Contiguous (relative or indexed) file
30: Peripheral driver

<NVar5> : File size in blocks (sectors).

<NVar6> : File status; a value other than 0 designates a formatted file.

<NVar7>
<NVar8>
<NVar10> : Dummy variables only needed to retain compatibility with earlier releases. The value of these variables is irrelevant.

<NVar9> : Time of file creation (relative to 1.1.1980).

<NVar11> : Disk address of the file header (decimal).

CALL Statements

Possible error: ERROR # 38

Meaning:

- <String> is a variable dimensioned for less than 15 bytes.
- LU addressed is not active.
- A numeric variable was dimensioned for less than 2%.
- Variables missing.
- <NumPar1> or <NVar2> negative.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.40 CALL 98

Sending any system command to any port. This enables each port to be logged on and off and any program to be started there.

Syntax:

format 1

CALL <NExpr>, <NumPar>, <String>, <NVar>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 98).

<NumPar> : The number of the port to which the command is to be sent.

<String> : The system command to be started at port <NumPar>. All system commands which can be entered in the system command mode via the keyboard are permissible.

<NVar> : After execution, the status is returned in this variable:

<NVar>	Meaning
0	Function executed without errors.
<>0	The function was not executed (the command was not transferred to the port addressed).

Logging a task on/off:

In order to start a program with a CALL statement, it is necessary to execute the CALL 98 statement twice.
The first time, the octal code ←334← (or the character assigned to octal code ←334← in the country-specific character set, e.g. backslash, Ö etc.) must be sent to the addressed task. This causes the addressed port to be logged off in a defined manner. Any program can subsequently be started by sending a system command.

CALL Statements

Note:

After the CALL 98 statement has been issued, it is necessary to wait 2 seconds (SIGNAL 3,20) before the status is tested in <NVar>.

- Example:

Starting the program called INVOICE in the phantom port (port number 1).

```
:
100 LET S=0                    /* <NVar> TO 0
110 LET A$= "-334-"            /* CODE FOR LOGGING OFF
115 CALL 98,1,A$,S             /* LOG OFF PHANTOM TASK
120 SIGNAL 3,20                /* WAIT 2 SECONDS
125 IF S GOTO ...              /* STATUS TEST
130 LET A$="INVOICE"           /* LOAD PROGRAM NAME
135 CALL 98,1,A$,S             /* START PROGRAM
140 SIGNAL 3,20                /* WAIT 2 SECONDS
145 IF S GOTO ...              /* STATUS TEST
```

8

NIROS 7.0/02 enhanced for phantom port handling. Format 2

CALL 98, SPAR1, NPAR1, NPAR2, SPAR2, NPAR3, SPAR3, SPAR4,

SPAR1 "A.START" [SPAR / NPAR] 3 / 0

NPAR1 function no 0 - Any Phantom
2 - Phantom on own CPU

NPAR2 BEFORE - CALLING PORT NO.
AFTER - STATUS 0 - OK
1 - No free phantom.

SPAR2 - Application programme to be started in phantom

NPAR3 - Common area format SEE P. 4-102
P. 4-101

SPAR3 - TAMOS control Params

SPAR4 - program description

SPAR / NPAR } Additional info (Free format dependant upon space available - dependant upon common area format)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

CALL Statements

8.41 CALL 99

Transferring or updating the system date and system time.

Syntax:

```
CALL <NExpr>, <SVar>
```

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (in this case 99).

<SVar> : String variable, length at least 23 bytes. <SVar> is used to transfer the date and time from the system to the application program or from the application program to the system.

Transferring date/time:

If the current system date and system time are to be transferred to the application program, <SVar> must be a null string.

- Example:

  Transferring the system date and system time.

```
:
100 LET A$= ""      /* <SVar> = NULL STRING
105 CALL 99,A$      /* TRANSFER DATE/TIME
```

The date and the time are subsequently contained in the following form in A$:

```
| M M M  D D     Y Y Y Y     h h : m m : s s |
  1 5 16 18 19   21 22
```

MMM : Month: JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

DD : Day

YYYY : Year

hh : Hour

mm : Minute

ss : Second

CALL Statements

Updating the system date and system time:

<SVar> must be assigned as follows:

| Y Y Y Y , M M , D D , h h , m m , s s |

YYYY : Year

MM : Month

DD : Day

hh : Hour

mm : Minute

ss : Second

Any non-numeric character can be used as a separator.

- Example:

  Updating the current system date. The new date must be entered in the required form.

```
:
100 INPUT "DATE/TIME: ",A$  /*ENTER DATE/TIME
105 CALL 99,A$              /*UPDATE DATE/TIME
```

  Possible error: ERROR # 38

  Meaning:

  - Date/time specified in wrong format.

  - Date or time impossible.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9 Keyboard input/screen output

Display workstations are connected to the central processing unit of the 8870 system by lines.

Display workstations undertake the following functions:

- Logging programs (tasks) on and off
- Dialog between operator and program
- Input and output of data

The display workstations are not subject to the channel concept, i.e. they need not be opened and closed. Each workstation is always assigned to the program which it called and started.

Programming of data input and output is carried out with the BASIC statements:

- INPUT and MAT INPUT and
- PRINT and MAT PRINT

which are supported by display functions (<DisFct>) for controlling the input/output facilities:

A list of the possible display functions is given in the chapter "BASIC Statements" in the section "Keyboard Input/Screen Output Statements".

The screen has an internally-controlled indicator (cursor) which is only visible when inputs are made. Input and output is always carried out from the current cursor position.

The cursor position is changed by the TAB function and certain other display functions apart from when inputs and outputs are being made.

Every display workstation has an input/output buffer with a size of 254 bytes of which 40 bytes (80 bytes on DAP 4) are reserved as a pre-input buffer. This enables up to 40 characters (80 on DAP 4) and, if necessary, release keys to be entered prior to the INPUT mode occurring. If more characters are entered, an acoustic signal is emitted and the excess characters are not accepted. The pre-input buffer is cleared by:
- pressing the ESC key
- output of ←207← by the INPUT or PRINT statement.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

If the program is in the INPUT mode, characters which are entered are displayed immediately from the current cursor position on the screen onwards. Pre-input characters are only displayed when the INPUT mode is activated.

The input/output buffer is emptied after an input is made, i.e. the data which has been entered is transferred by the program if the input is concluded by pressing a release key.
The buffer is emptied after a PRINT statement, i.e. data is displayed on the screen, when:

- a concurrent program receives its time slice,
- the output buffer is full,
- a SIGNAL 3 statement,
- an INPUT statement,
- a PRINT # statement,
- a CHAIN statement,
- a LINK statement is executed and when
- the program end is reached.

9.1 INPUT statement

Syntax:

```
INPUT [ <SLit>,                                  ]^n  { <NVar> }
      [ <DisFct>,                                ]    { <SVar> }
      [ TAB (<NExpr>[,<NExpr>]),                 ]
      [ <NVar>,                                  ]
      [ <SVar>,                                  ]
```

Input of data to numeric or string variables via the keyboard. In addition, prompts and display functions can be output.
A detailed description of the INPUT statement can be found in the chapter "BASIC Statements".

Keyboard Input/Screen Output

9.2 MAT INPUT statement

Syntax:

```
MAT INPUT <MVar>[(<NExpr>[,<NExpr>])]
            [,<MVar>[(<NExpr>[,<NExpr>])]]n
```

This statement enables all elements of a matrix (without row and column 0) to be entered via the keyboard.
A detailed description of the MAT INPUT statement can be found in the chapter "BASIC Statements".

9.3 PRINT statement

Syntax:

```
{PRINT} [USING <String> ;] [<NExpr>  ][{,}{<NExpr>  }]n [;]
{;    }                    [<String>  ][{;}{<String>  }]
                           [<DisFct>  ]  {<DisFct>  }
```

PRINT enables output of

- the contents of string variables,
- the value of numeric expressions,
- numeric and alphanumeric constants and
- display functions

in any order on the screen.

A detailed description of the PRINT statement can be found in the chapter "BASIC Statements".

9.4 MAT PRINT statement

Syntax:

```
MAT PRINT <MVar> [,|;]
```

The MAT PRINT statement results in an entire matrix (without row and column 0) being displayed on the screen. A detailed description of the MAT PRINT statement can be found in the chapter "BASIC Statements".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5 Display functions

Display functions are initiated through output of specific ASCII characters. Due to the fact that these characters cannot be represented with the keyboard, they must be replaced by either two or three symbolic letters in single quotation marks (') or by their octal code.
The lead-in code (←376←) must precede some of the functions in the case of octal representation.

The octal code is included in the following descriptions in addition to the symbolic representation.

9.5.1 TAB function

Syntax:

Column Line

TAB (<NExpr1>[,<NExpr2>])

The TAB display function enables the cursor to be placed anywhere on the screen.

TAB (<NExpr>)

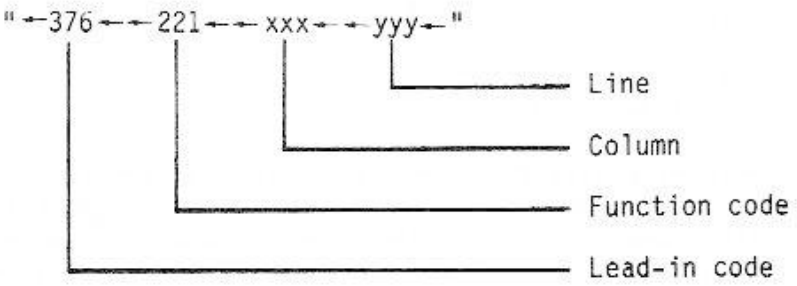
Blanks are output starting from the current cursor position up to the column before the one designated with the value of <NExpr>, and the cursor is positioned to the column designated by <NExpr>. Any text located between these positions is overwritten.
If <NExpr> is less than the current cursor position, the function has no effect. If <NExpr> exceeds the line length, an appropriate number of line feeds are automatically executed.

TAB (<NExpr1>,<NExpr2>)

The cursor is positioned to the column designated by the value of <NExpr1> and the line designated by the value of <NExpr2>. Backward positioning is possible. The contents of the screen remain intact.
The column specification must range from 0 to 79, and the line specification from 0 to 24.

In the case of octal representation of the TAB function, the column and line can only be specified absolutely (not in the form <NExpr>):

Octal code:



The column and line must also be coded in octal notation in this case.

- Example:

  Positioning the cursor to column 10 in line 2.

  50 PRINT TAB(10,2);...

  Octal: 50 PRINT "←376←←221←←212←←202←";...

9.5.2 Backspace function

Syntax:

{'BS' / "←210←"}

Backspacing of the cursor by one character from the current position. A character located at this position is deleted. If the cursor is located in column 0 of any line, it is positioned to column 79 of the preceding line. The function is ineffective if the cursor is located in column 0 of line 0.

It is not possible to change from a foreground field to a background field; if the cursor is located immediately after a background field, the function has no effect and an acoustic signal is emitted.

- Example:

50 PRINT 'BS';

Octal: 50 PRINT "←210←";

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.3 Mark cursor position function

Syntax:

```
{'MP'                    }
{"←376← ←211←"           }
```

The current position of the cursor (column and line) is stored. These stored coordinates can only be changed when this function is called again and are deleted when the port is switched off. If
* a display window is set up, its coordinates are also marked. The
* function also clears an existing display window.

- Example:

```
50 PRINT 'MP';

Octal: 50 PRINT "←376← ←211← ";
```

9.5.4 Back to marked cursor position function

Syntax:

```
{'BP'                    }
{"← 376← ←212←"          }
```

The cursor is set to the position which was stored when the 'MP' (mark cursor position) function was last called. Any display window which has been marked with 'MP' is set up again.

When the port is switched on, the cursor is set to column 0 in line 0 when this function is called.

- Example:

```
50 INPUT 'BP',X

Positioning to the marked display position when X is entered.

Octal: 50 INPUT "←376←←212←",X
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

9.5.5 Carriage return function

Syntax:

```
{'CR'          }
{"← 215←"      }
```

Positioning the cursor to column 0 of the next line (line feed). The contents of the screen remain unchanged. If the cursor is located in line 24, the entire contents of the screen are shifted upwards by one line. The information in line 0 is lost and line 24 becomes free for use. The cursor is subsequently located in column 0 in line 24.

- Example:

  Output of the string literals "FIELD 1" and "FIELD 2" in column 0 of lines 5 and 6.

```
50 PRINT TAB(0,5);"FIELD 1";'CR';"FIELD 2";
```

9.5.6 Line deletion function

Syntax:

```
{'LD'                    }
{"←376← ←223←"           }
```

All lines beneath the line in which the cursor is located are shifted upwards by one line. The line in which the cursor is located is overwritten and line 24 becomes free for use.
If the cursor is in line 24, only this line is deleted.

- Example:

  Deleting line 24 and outputting the text "STATUS :" from column 0.

```
50 PRINT TAB(0,24);'LD';"STATUS :";
```

Keyboard Input/Screen Output

9.5.7 Line insertion function

Syntax:

```
{'LI'                    }
{"←376←←232←"}
```

Inserting a blank line starting from the current cursor position. The line in which the cursor is located and all lines beneath are shifted downwards by one line. The information in line 24 is lost and the cursor is subsequently located in column 0 of the inserted line.

- Examples:

  - Shifting the entire screen contents downwards by one line:

    ```
    50 PRINT TAB(0,0);'LI';
    ```

  - Roll-up from line 3 to line 20:

    ```
    50 PRINT TAB(0,3);'LD';TAB(0,20);'LI';
    ```

  - Roll-down from line 3 to line 20:

    ```
    50 PRINT TAB(0,20);'LD';TAB(0,3);'LI';
    ```

  - Deleting line 11:

    ```
    50 PRINT TAB(0,11);'LD';'LI';
    ```

9.5.8 Start background function

Syntax:

```
{'SB'                    }
{"←376←←231←"}
```

All characters that follow appear on the screen as background characters with reduced illumination.

- Example:

  ```
  50 PRINT 'SB';
  ```

  Octal: 50 PRINT "←376←←231←";

Keyboard Input/Screen Output

9.5.9 Start foreground function

Syntax:

```
{'SF'          }
{"←376←←237←"}
```

All characters that follow appear on the screen as foreground characters with full illumination. This function is only necessary if 'SB' (start background) was specified beforehand.

- Example:

  After the text "DISCOUNT : " is output in the background, switch over to the foreground and input in A.

```
50 INPUT 'SB',"DISCOUNT : ",'SF' ,A
```

9.5.10 Clear screen function

Syntax:

```
{'CS'          }
{"←376←←234←"}
```

The entire screen is cleared. The cursor is subsequently located in column 0 on line 0.

- Example:

```
50 PRINT 'CS';
```

Octal: `50 PRINT "←376←←234←";`

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.11 Clear foreground function

Syntax:

```
{'CF'                    }
{"←376←←235←"            }
```

All foreground fields are cleared and the cursor is subsequently located on the first character of the first foreground field. If there is no foreground field, the cursor is located in column 79 of line 24.

- Example:

```
50 PRINT 'CF';

Octal: 50 PRINT "←376←←235←";
```

9.5.12 Ring keyboard bell function

```
{'BEL'     }
{"←207←"   }
```

Activating the keyboard's audible signal emitter. This function has no effect on the contents of the screen.

- Example:

Output of a string variable starting from column 25 in line 24 and setting the audible signal.

```
50 PRINT TAB(25,24);A$;'BEL';
```

9.5.13 Tab to next foreground field function

Syntax:

```
{'TB'          }
{"←211←"       }
```

The cursor is positioned to the first character of the next foreground field (foreground field, see section 9.5.14). If no foreground field follows, the cursor remains at its current position.

- Example:

  Tabulation to the next foreground field and input of the nth element of a vector.

```
50 INPUT 'TB',X(N)
```

  Octal:

```
50 INPUT "←211←",X(N)
```

9.5.14 Clear foreground field function

Syntax:

```
{'CFF'         }
{"←230←"       }
```

Clearing the foreground field in which the cursor is located. After the function has been executed, the cursor is located at the start of the line concerned or at the first position of the corresponding foreground field.

A foreground field is made up of a sequence of foreground characters and is enclosed by background characters, the start or end of a line.

If, after 'CFF' is employed, a tabulation is to be made in the same line, the TAB function must be used with two parameters.

- Example:

```
50 PRINT TAB(0,14);'CFF';"SALES:";TAB(30,14);
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.15 Cursor home function

Syntax:

```
{'CH'                  }
{"←376←←222←"}
```

The cursor is positioned to line and column 0 provided that it is a foreground field. If this position is part of a background field, the cursor is set to the first position of the first foreground field.

If no foreground field follows, the cursor remains at its current position.

- Example:

```
50 PRINT 'CH';

Octal: 50 PRINT "←376←←222←";
```

9.5.16 Dark function

Syntax:

```
'DRK'
```

Dark input for exactly one variable. This keyboard input is not displayed. The function is specified in the INPUT statement and applies to the next input variable.

- Example:

```
50 INPUT TAB(0,10), "PASSWORD:",'DRK', P$
```

9.5.17 Hexadecimal output function

Syntax:

'X'

This function causes the next string to be output in hexadecimal notation to display or printer.

- Example:

```
10 DIM A$(5)
20 LET A$="12345"
30 PRINT 'X';"AB"; "HEX:"; 'X'; A$; "END";

Result:      C1C2HEX:B1B2B3B4B5END";
```

9.5.18 Duplicate field function

Duplicating an input field on display to the input buffer.

Syntax:

{'DF' / "←376←←236←←200←"} , TAB(A,B) ,TAB(C,D)

A, B, C and D are numeric expressions.

TAB(A,B) : Cursor position after execution
TAB(C,D) : Start of input field

This function causes an input field to be duplicated until a background character is reached. If the input field is not limited by a background character, the characters are transferred to the input buffer until the end of the display or the maximum number (254) is reached.
'DF' can only appear in an INPUT statement.

Attention:

The input field is only duplicated if at least one character has been manipulated. Otherwise, a null string or the value 0 is transferred. Manipulation counts as pressing a key with which data can be input even if this only means that a character has been overwritten by the same character.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.19 Duplicating limited characters function

Duplicating a number of characters from the display memory to the input buffer with limitation of the field length.

Syntax:

```
{'DLC'                          }
{"←376←←236←←201←"}  ,TAB(A,B),TAB(C,D),TAB(E,F)
```

A, B, C, D, E and F are numeric expressions.

TAB(A,B) : Cursor coordinates after execution

TAB(C,D) : Start of duplication field

TAB(E,F) : End of duplication field

This function enables a specified number of characters to be duplicated from an input field with a limited field length. The duplication function is terminated when the end of the display, a background character, the maximum number of characters (254) or the specified number is reached. The number is the difference between the positions TAB(E,F) and TAB(C,D).

- Example:

```
10 INPUT 'DLC',TAB(10,9), TAB(10,9), TAB(19,9),A$
```

Duplicating 10 characters from position 10 in line 9. Afterwards, the cursor is positioned to (10,9). If the transferred data have been updated via keyboard, they are transferred to the application program in the variable A$. This function can only appear in an INPUT statement.

Attention:

The input field is only duplicated if at least one character has been manipulated. Otherwise, a null string or the value 0 is transferred. Manipulation counts as pressing a key with which data can be input even if this means that a character has been overwritten by the same character.

Keyboard Input/Screen Output

9.5.20 Duplicate characters function

Duplicating a number of characters from the display memory to the input buffer without limitation of the field length.

Syntax:

```
{'DC'                          }
{"←376←←236←←202←"}  ,TAB(A,B), TAB(C,D), TAB(E,F)
```

A, B, C, D, E and F are numeric expressions.

TAB(A,B) : Cursor coordinates after execution

TAB(C,D) : Start of duplication field

TAB(E,F) : End of duplication field

This function enables a specified number of characters to be duplicated from an input field. The duplication function is terminated when the end of the display, a background character, the specified number or the maximum number (254) of characters is reached.
The duplication function does not limit the length of the input field. The field length can be extended up to a background character or to the maximum number of characters (254). Only the input and duplicated characters are transferred to the CPU, irrespective of the cursor position.
This function can only appear in an INPUT statement.

Attention:

The input field is only duplicated if at least one character has been manipulated. Otherwise a null string or the value 0 is transferred. Manipulation counts as pressing a key with which data can be input, even if this means that a character is only overwritten by the same character.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.21 Display memory function

Syntax:

```
{'DM'                              }
{"-376←-236←-203-"}  , TAB(C,D), TAB(E,F)
```

C, D, E and F are numeric expressions.

TAB(C,D) : Starting position of transfer

TAB(E,F) : End position of transfer

This function enables the CPU to fetch the data from the display memory at the workstation. All characters (foreground and background) are transferred to the CPU when the next INPUT occurs. Up to 254 characters can be fetched with one read command. If a read command goes beyond the end of the display, the data are transferred up to the end of the display.

- Example:

```
10 INPUT 'DM',TAB(C,D), TAB(E,F), A$
```

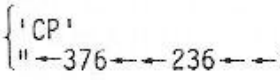
Note:

The characters are transferred without the 8th bit. For this reason, differences can occur when comparing strings even though they are obviously the same. In this case, it is possible to intervene with CALL 20 and reset the appropriate bits.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

9.5.22 Cursor position function

Syntax:



This function enables the CPU to fetch the current cursor position (next write/input position) from the workstation. The cursor coordinates are transferred from the workstation to the CPU when the next INPUT occurs. The cursor coordinates are represented as CCLL:

```
CC = Column position (00 - 79)
LL = Line position   (00 - 24)
```

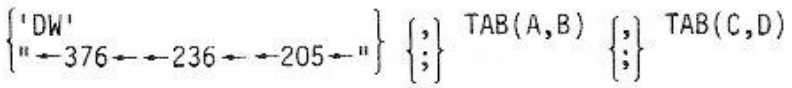
The cursor coordinates can be transferred to a string variable or numeric variable:

```
10 INPUT 'CP', A$
20 LET A= A$(1,2)              /* Column
30 LET B= A$(3,4)              /* Line

10 INPUT 'CP', E
20 LET A = INT(E/100)          /* Column
30 LET B = FRA(E/100)*100      /* Line
```

9.5.23 Define window function

Syntax:



A, B, C and D are numeric expressions.

TAB(A,B) : Start coordinates of window
(column always = 0 in starting line no matter what is specified)

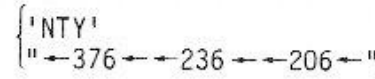
TAB(C,D) : End coordinates of window
(column always = 79 in end line no matter what is specified)

The "," must be used as a separator in an INPUT statement whereas ";" must be used in a PRINT statement. This statement enables the screen (25 lines x 80 columns) to be split up into windows comprising full lines. The column specifications are not evaluated, i.e. A always equals 0 and C always equals 79. The value for the starting line must not exceed the value of the end line of the window. Afterwards all workstation functions only refer to the current window. A window is handled like a small-scale independent display.

The window is erased upon returning to TAMOS or SCOPE.

9.5.24 No typewriter keyboard function

Syntax:

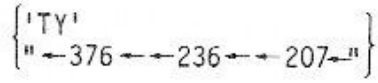


The keyboard mode for upper and lower case, as defined by the shift key, can be switched over with this function. 'NTY' means that upper case is active if the shift key is not pressed.

The program 'TPCODEUTIL' can be used to define which keys are to be affected by the switch-over.

9.5.25 Typewriter keyboard function

Syntax:



The keyboard mode for upper and lower case font, as defined by the shift key, can be switched over with this function. 'TY' means that lower case is active if the shift key is not pressed.

9.5.26 Activate 8-bit transfer function

Syntax:

"←376←←236←←210←"

Bit 8 in the code of a character can be used to control whether the character is displayed with high or low intensity. If 8-bit transfer is activated, a character which has a set bit 8 (=1) is output with high intensity and a character with bit 8 which has not been set is output with low intensity. If 8-bit transfer is deactivated, bit 8 is ignored.

The DM function enables the contents of the screen to be stored with a different degree of intensity. By activating or deactivating the 8-bit transfer function, you can later choose whether the characters are bright on the screen or whether the "read-in" brightness is used.

The SF and SB functions are overridden by this function.

9.5.27 Deactivate 8-bit transfer function

Syntax:

"←376←←236←←211←"

This function terminates brightness control via bit 8.

- Example:

```
10 DIM N$(20), A$(27)
20 INPUT 'CS', TAB(10,5),'SB',"NAME : ",'SF',N$
30 INPUT 'DM', TAB(10,5), TAB(36,5),A$
:
130 PRINT TAB(10,20);"←376←←236←←210←";A$
140 PRINT TAB(10,21);"←376←←236←←211←";A$
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.5.28 Table of available display and device functions

Symbol	Function	Lead-in code	Octal code
TAB (X,Y)	Tabulation X=column, Y=line	376	221 and column and line
'BS'	Backspace one position	376	210
'MP'	Mark cursor position	376	211
'BP'	Back to marked cursor position	376	212
'CR'	Carriage return (line feed)	-	215
'LD'	Line deletion	376	223
'LI'	Line insertion	376	232
'SB'	Start background	376	231
'SF'	Start foreground	376	237
'CS'	Clear screen	376	234
'CF'	Clear foreground	376	235
'BEL'	Ring keyboard bell	-	207
'TB'	Tabulation to the next foreground field	-	211
'CFF'	Clear current foreground field	-	230
'CH'	Cursor home (position to line 0 and column 0)	376	222

Keyboard Input/Screen Output

Symbol	Function	←376←←236←	Octal code	TAB spec.
'DRK'	Dark input (not displayed)	-	-	-
'X'	Hexadecimal output	-	-	-
'DF'	Duplicate field	x	200	x
'DLC'	Duplicate limited characters	x	201	x
'DC'	Duplicate characters	x	202	x
'DM'	Display memory	x	203	x
'CP'	Cursor position	x	204	-
'DW'	Define window	x	205	x
'NTY'	No typewriter keyboard	x	206	-
'TY'	Typewriter keyboard	x	207	-
	Activate 8-bit transfer	x	210	-
	Deactivate 8-bit transfer	x	211	-

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Functions for resetting SAS devices

Symbol	Function	Function code			Octal code	TAB spec.
-	Reset all SAS devices except printers	376	236	237	200	-
-	Reset OCR reader	"	"	"	201	-
-	Reset BCR	"	"	"	202	-
-	Reset IDCU/SCR	"	"	"	203	-
-	Reset POS workstation	"	"	"	205	-

The following device functions (<DevFct>) also refer to workstation peripherals. They are described in the corresponding sections.

Forms control: 'EOL', 'LFB', 'OPN', 'IN', 'FF', 'CR'

Cut-form processing: MCL (<NExpr>)

Paper cassette selection: 'SC1', 'SC2'

Elongated print: 'ELO', 'ELE'

Character density: 'LC1', 'LC2'

Additional functions can be found in the descriptions of the individual devices.

9.6 Programming release keys and latching keys

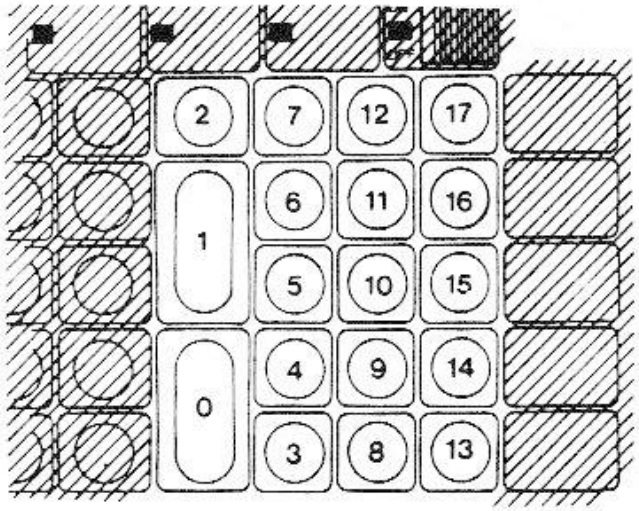
The following functions are available for testing release keys as well as programming latching keys

- KEY
- LKY
- CKY

9.6.1 KEY function

The KEY function returns the numeric value of the release key last pressed. The only permissible argument of the function is 0.

The figures for the release keys are:



• Example:

```
:
110  IF KEY (0)=4 GOTO 250
:
250 REM *** RELEASE KEY 4 PRESSED ***
```

Note:

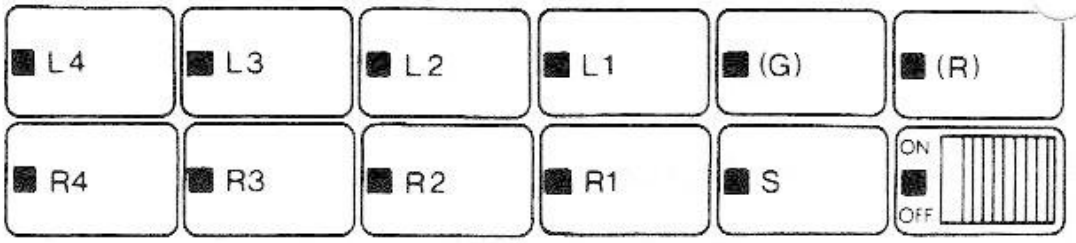
The "Y" and "N" keys on the numeric keypad have the same effect as "Y" "CR" and "N" "CR" on the alphanumeric keyboard and, as with "CR", result in the value 0.
When keys other than release keys are pressed, KEY (0) has no effect.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.6.2 LKY function

The LKY function returns the value of the set latching key(s) (when no latching key is set, the value 0).



- LKY (0) returns the sum of the values of all latching keys set.
- LKY (1) returns the value of the latching key "R1".
- LKY (2) returns the value of the latching key "R2".
- LKY (3) returns the value of the latching key "R3".
- LKY (4) returns the value of the latching key "R4".

The values of set latching keys are as follows:

"R1" : 256 (2↑8)
"R2" : 512 (2↑9)
"R3" : 1024 (2↑10)
"R4" : 2048 (2↑11)

• Example:

```
190 IF LKY (0)=1536 GOTO 250
:
250 REM *** R2, R3 SET ***

550 IF LKY (4)=2048 GOTO 660
:
660 REM *** R4 SET ***
```

9.6.3 CKY function

The CKY function clears all set latching keys and sets the latching key values to 0. The argument of the function must be 0. The function is only permissible in the PRINT and INPUT statements.

- Examples:

```
100 INPUT CKY (0),A

190 PRINT CKY (0)
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.7 Input via the OCR-A/B reader, BCR, SCR, IDCU

In addition to connecting the keyboard, the workstation's keyboard interface can be used to connect a hand-held OCR-A/B reader, a bar code reader (BCR), a swipe card reader (SCR) or an ID card unit (IDCU).

When inputs are required by programs with the appropriate INPUT statements, they can then be made with these devices instead of via the keyboard. The hand-held OCR-A/B reader and the BCR are optical scanners which read OCR-A or OCR-B font and bar code respectively. The SCR and IDCU are units for reading ID cards which have magnetic stripes. The IDCU can also be used to write the magnetic stripes by using a PRINT statement.

9.7.1 Input procedure

Inputs from the keyboard and from these devices can be read into numeric or string variables by means of an INPUT statement. The structure of the data which is transferred is identical. When an INPUT statement occurs, the data is converted to ASCII, is transferred to the application program and possibly displayed. In addition to the actual data, the code for a release key is transferred and this can be tested by means of the KEY function. The data is stored intermediately in a buffer for the device in question.

If a keyboard input is in progress when an input is due from one of the devices, the inputs are rejected until the keyboard input has been completed in the case of SCR and IDCU. The input is transferred to the buffer in the case of the OCR-A/B reader and BCR. Further inputs are also rejected. When the keyboard input has been completed, the reader input is transferred from the buffer and possibly displayed. Then further inputs from the readers are possible.

If an input is in progress from one of the reading devices when a keyboard input is due, the keyboard inputs go into the pre-input buffer until the data from the other device has been transferred in full and a new INPUT statement occurs. The characters which have been keyed in are not displayed during this time.

Keyboard Input/Screen Output

If keyboard and reader inputs alternate, these are processed in the specified sequence. If several readers are connected, however, and there are "mixed" inputs from different devices, it cannot be guaranteed that the inputs are processed in the order that they were made.

The buffer which is assigned to the device in question and the keyboard's pre-input buffer are cleared when a special function (ESC, CTL/C) is detected. If the application program detects an error when checking or processing the input data, it can put the workstation in a defined state by means of the control code "←376←←233←". When a special function is detected, an acoustic signal is given in addition to the steps which have already been mentioned. With regard to the OCR-A reader, only the OCR-A reader buffer is cleared in this case and the acoustic signal is given by outputting octal code "←207←". In this way, the faulty data is available in the pre-input buffer for further processing.
If the cursor reaches a background character when outputting data from the reading devices, the characters which have been output are deleted in addition to the steps which have already been mentioned.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.7.2 Particular features of the individual devices

A utility is used to process code tables for the devices so that the application program can control the way in which the data which has been read in from the devices is transferred.

9.7.2.1 Particular features of the OCR-A/B reader

During a reading operation, the OCR-A/B reader scans a line of a label which can contain several fields of information up to a total length of 26 bytes and outputs the data on the display. An information field has the following structure:

- Function code (one byte).
- Numeric data. The length is determined by the function code.

The numeric information is split up into integer and decimal places in accordance with a code table and is separated by the decimal point or decimal comma.

The control code is mapped to the release keys of the keyboard via the code table and can be processed analogously by the user (function KEY (0)).

In addition, the control code specifies the length and status of the input field:

- standard field or
- protected field.

Unlike standard fields, the length of protected fields cannot be modified.

OCR-A character fields must always be padded with fillers up to the length specified in the control code if the input data requires less space.

Keyboard Input/Screen Output

Example of a code table for control characters and fillers.

Character	Field description/fillers
A	Standard 13 characters
C	Standard 3 characters
D	Filler
M	Protected 7 characters
N	Standard 9 characters
P	Standard 8 characters
R	Standard 10 characters
U	Standard 6 characters
X	Modification +2
Y	Modification -2
+	Protected 5 characters
>	Filler
<Di>/	Modification by <Di>

List of data characters:

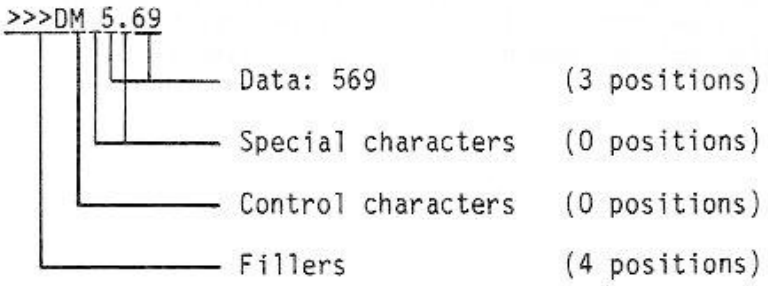
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

List of special characters:

. | , | ß

The special characters are only used to optically divide up the fields and they do not constitute characters, nor are they interpreted as a decimal point or comma.

- Example:



Further examples of OCR-A character fields

UY 02 146 20632033 M123 45.67

DM 1690,90 N3/12 34 56 78

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.7.2.2 Particular features of the BCR

The input data is stored in a buffer. The buffer is big enough to allow a pre-input of up to four 13-character or up to five 8-character labels.

When an INPUT statement occurs, an input field from the buffer is recoded and transferred to the application program. The data from the BCR is always transferred to the application program without a check digit. This means that only 12 items of data are transferred in the case of a 13-character label (7 items of data for an 8-character label). Whether the BCR data is displayed or not is defined via the operating parameter table.

The number of items of data which are transferred depends on the code which is used.

- EAN-13 : 12 characters of information
- EAN-08 : 7 characters of information
- UPC-13 : 12 characters of information
- UPC-E : 6 characters of information

Each BCR input is terminated with the code of a release key. The code of this key is not transferred to the application program in the input variable but it can be ascertained with the KEY function. A release key can be assigned to each code which can be read.
The following key codes have been set as standard:

- EAN-13 : Release key 3 (AT 4)
- EAN-08 : Release key 2 (AT 3)
- UPC-13 : Release key 1 (AT 2)
- UPC-E : Release key 4 (AT 5)

The code table defines the conversion of BCD characters to ASCII, the assignment of release key codes to codes which are in use (EAN and UPC code) and the way the UPC-E code is transferred. It also defines whether the input is displayed.

9.7.2.3. Particular features of the SCR

Two tracks (tracks 2 and 3) can be read on the ID cards. The SCR buffer in the display workstation can hold the contents of both tracks. The code table is used to define whether just track 2 is to be read, just track 3 or both. 37 characters are available for the user on track 2 and 104 characters on track 3.
The code table also defines which release key code is to be transferred in each case.

In addition, it defines the conversion of data characters to ASCII and whether the input is to be displayed.

9.7.2.4 Particular features of the IDCU

After the workstation has been switched on and the IDCU has been initialized, the workstation is in SCR mode. In this state, the IDCU acts just like the SCR as far as the application program is concerned.

It is necessary to switch over to IDCU mode if an ID card is to be written. The application program controls the switch-over by supplying one of the functions "test read", "write" or "eject card". The workstation and the IDCU return to SCR mode whenever a card is ejected.

When programming the IDCU, it must be ensured that there is no switch-over to IDCU mode when a card is being read. An output function must not be issued until it is certain that a previous output function has been completed.

The IDCU acts just like the SCR when reading ID cards.

Writing to an ID card (with a PRINT statement) should always be preceded by a test read of tracks 2 and 3. The data is transferred to the application program when the next INPUT statement occurs. The ID card is not ejected and the data is not output to display.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

- Example : INPUT "←376←←236←←231←", I$

  Octal code←376←←236←←231← means "read without ejecting card or displaying data".

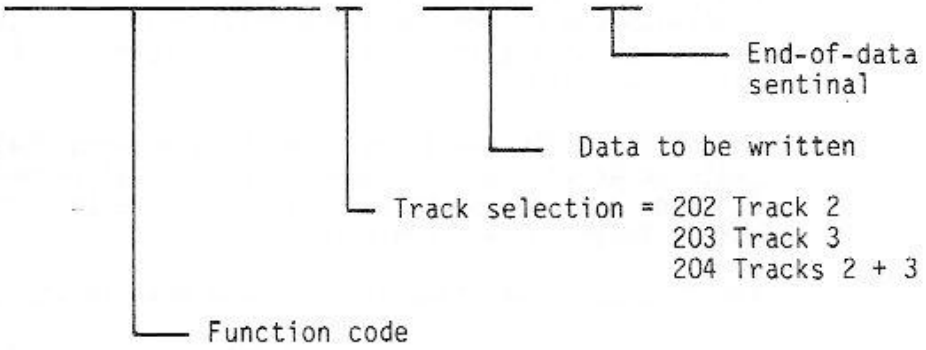
If a read error occurs, two attempts are made to read the track in question. If these attempts are unsuccessful, the card is ejected.

No data is transferred to the application program in the event of an error. However, the "test read" function counts as having been executed. This means that the INPUT statement is still active and must be terminated via the keyboard before a new IDCU function can be started.

An ID card which has been rejected as defective during the "test read" function can only be written without a previous test read.

Data is written to the ID card by means of a PRINT statement with the following structure:

"←376← ←236←←232←←20*← D A T A ←203←"



- Example

  Writing the string O$ to tracks 2 and 3 of an ID card.

  PRINT "←376←←236←←232←←204←";O$;"←203←";

Keyboard Input/Screen Output

If an ID card is to be written without a preceding test read (e.g. in the event of an error) it must be ensured that the ID card in question is inserted in the IDCU immediately after the PRINT statement has been issued.

As soon as a track has been written, this track is read and compared with the data stored in the workstation's buffer. If the contents of the buffer do not match those on the ID card, up to 2 more attempts are made to write the data. If these attempts are unsuccessful, the card is rejected as being defective.

If both tracks are written, the data of the source variables is written sequentially starting with track 2. If track 3 is not used (up to 38 characters), it is automatically marked as being empty through the output of separators.

The "eject card" function is used to eject ID cards which are still in the IDCU after a test read and which are not to be written.

Octal code : "←376←←236←←233←"

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.8 ASCII table

The 7-bit ASCII code is used. The 8th bit of a byte is always set internally to 1.

Character	Octal	Hexadecimal	Bit combination	Key
NUL	2.0.0	8.0	10 000 000	
SOH	2.0.1	8.1	10 000 001	
STX	2.0.2	8.2	10 000 010	
ETX	2.0.3	8.3	10 000 011	
EOT	2.0.4	8.4	10 000 100	
ENQ	2.0.5	8.5	10 000 101	CTL-E
ACK	2.0.6	8.6	10 000 110	
BEL	2.0.7	8.7	10 000 111	
BS	2.1.0	8.8	10 001 000	
TB	2.1.1	8.9	10 001 001	CTL-I
LF	2.1.2	8.A	10 001 010	
VT	2.1.3	8.B	10 001 011	
FF	2.1.4	8.C	10 001 100	CTL-L
CR	2.1.5	8.D	10 001 101	CTL-Z
SO	2.1.6	8.E	10 001 110	
SI	2.1.7	8.F	10 001 111	
DLE	2.2.0	9.0	10 010 000	
DC1	2.2.1	9.1	10 010 001	
DC2	2.2.2	9.2	10 010 010	
DC3	2.2.3	9.3	10 010 011	
DC4	2.2.4	9.4	10 010 100	
NAK	2.2.5	9.5	10 010 101	
SYN	2.2.6	9.6	10 010 110	
ETB	2.2.7	9.7	10 010 111	
CFF	2.3.0	9.8	10 011 000	
EM	2.3.1	9.9	10 011 001	CTL-Y
SUB	2.3.2	9.A	10 011 010	
ESC	2.3.3	9.B	10 011 011	
FS	2.3.4	9.C	10 011 100	
GS	2.3.5	9.D	10 011 101	
RS	2.3.6	9.E	10 011 110	
US	2.3.7	9.F	10 011 111	

Keyboard Input/Screen Output

Notes:

If a combination of keys is necessary to generate a character via the keyboard, this is identified by a multiple specification in the column "Key". The individual key specifications are separated from one another by "-".

Meaning:

CTL- : Hold the control key and press the additional key.

If the desired character is on the upper section of the key, the shift key must be pressed prior to entering the character and held in the locked position.

If lower case letters are to be entered, the shift key must also be pressed, held there and then the appropriate upper case letter entered.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Character	Octal	Hexadecimal	Bit combination	Key
SP	2.4.0	A.0	10 100 000	Space
!	2.4.1	A.1	10 100 001	!
"	2.4.2	A.2	10 100 010	"
#	2.4.3	A.3	10 100 011	#
$	2.4.4	A.4	10 100 100	$
%	2.4.5	A.5	10 100 101	%
&	2.4.6	A.6	10 100 110	&
'	2.4.7	A.7	10 100 111	'
(	2.5.0	A.8	10 101 000	(
)	2.5.1	A.9	10 101 001	)
*	2.5.2	A.A	10 101 010	*
+	2.5.3	A.B	10 101 011	+
,	2.5.4	A.C	10 101 100	,
-	2.5.5	A.D	10 101 101	-
.	2.5.6	A.E	10 101 110	.
/	2.5.7	A.F	10 101 111	/
0	2.6.0	B.0	10 110 000	0
1	2.6.1	B.1	10 110 001	1
2	2.6.2	B.2	10 110 010	2
3	2.6.3	B.3	10 110 011	3
4	2.6.4	B.4	10 110 100	4
5	2.6.5	B.5	10 110 101	5
6	2.6.6	B.6	10 110 110	6
7	2.6.7	B.7	10 110 111	7
8	2.7.0	B.8	10 111 000	8
9	2.7.1	B.9	10 111 001	9
:	2.7.2	B.A	10 111 010	:
;	2.7.3	B.B	10 111 011	;
<	2.7.4	B.C	10 111 100	<
=	2.7.5	B.D	10 111 101	=
>	2.7.6	B.E	10 111 110	>
?	2.7.7	B.F	10 111 111	?
@	3.0.0	C.0	11 000 000	@
A	3.0.1	C.1	11 000 001	A
B	3.0.2	C.2	11 000 010	B
C	3.0.3	C.3	11 000 011	C
D	3.0.4	C.4	11 000 100	D
E	3.0.5	C.5	11 000 101	E
F	3.0.6	C.6	11 000 110	F
G	3.0.7	C.7	11 000 111	G
H	3.1.0	C.8	11 001 000	H
I	3.1.1	C.9	11 001 001	I
J	3.1.2	C.A	11 001 010	J
K	3.1.3	C.B	11 001 011	K
L	3.1.4	C.C	11 001 100	L
M	3.1.5	C.D	11 001 101	M
N	3.1.6	C.E	11 001 110	N
O	3.1.7	C.F	11 001 111	O

Keyboard Input/Screen Output

Character	Octal	Hexadecimal	Bit combination	Key
P	3.2.0	D.0	11 010 000	P
Q	3.2.1	D.1	11 010 001	Q
R	3.2.2	D.2	11 010 010	R
S	3.2.3	D.3	11 010 011	S
T	3.2.4	D.4	11 010 100	T
U	3.2.5	D.5	11 010 101	U
V	3.2.6	D.6	11 010 110	V
W	3.2.7	D.7	11 010 111	W
X	3.3.0	D.8	11 011 000	X
Y	3.3.1	D.9	11 011 001	Y
Z	3.3.2	D.A	11 011 010	Z
Ä	3.3.3	D.B	11 011 011	Ä
Ö	3.3.4	D.C	11 011 100	Ö
Ü	3.3.5	D.D	11 011 101	Ü
↑	3.3.6	D.E	11 011 110	↑
←	3.3.7	D.F	11 011 111	←
ß	3.4.0	E.0	11 100 000	ß
a	3.4.1	E.1	11 100 001	a
b	3.4.2	E.2	11 100 010	b
c	3.4.3	E.3	11 100 011	c
d	3.4.4	E.4	11 100 100	d
e	3.4.5	E.5	11 100 101	e
f	3.4.6	E.6	11 100 110	f
g	3.4.7	E.7	11 100 111	g
h	3.5.0	E.8	11 101 000	h
i	3.5.1	E.9	11 101 001	i
j	3.5.2	E.A	11 101 010	j
k	3.5.3	E.B	11 101 011	k
l	3.5.4	E.C	11 101 100	l
m	3.5.5	E.D	11 101 101	m
n	3.5.6	E.E	11 101 110	n
o	3.5.7	E.F	11 101 111	o
p	3.6.0	F.0	11 110 000	p
q	3.6.1	F.1	11 110 001	q
r	3.6.2	F.2	11 110 010	r
s	3.6.3	F.3	11 110 011	s
t	3.6.4	F.4	11 110 100	t
u	3.6.5	F.5	11 110 101	u
v	3.6.6	F.6	11 110 110	v
w	3.6.7	F.7	11 110 111	w
x	3.7.0	F.8	11 111 000	x
y	3.7.1	F.9	11 111 001	y
z	3.7.2	F.A	11 111 010	z
ä	3.7.3	F.B	11 111 011	ä
ö	3.7.4	F.C	11 111 100	ö
ü	3.7.5	F.D	11 111 101	ü
LEAD-IN	3.7.6	F.E	11 111 110	
DEL	3.7.7	F.F	11 111 111	

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* 9.9 I/O macro

The I/O macro is a tool for processing screen inputs and outputs on a field basis. The I/O macro only needs to be called once to process an input/output field completely. Field editing, input handling and validation are parameter-driven.

The field description parameters are stored in a parameter file which is created and maintained by means of the utility "IOM FIELD DESCRIPTION". The I/O macro provides easy-to-use facilities for creating and maintaining masks without programming effort.

The I/O macro access the parameter file without occupying a data channel. The application merely has to transfer the file name and the number of the required parameter record. Although the I/O macro accesses the TF.PARAM file to check any hard key that has been released, it is not necessary to keep TF.PARAM open.

Some of the parameters stored in the record can be overridden by the I/O macro's call parameters to show the same texts in different screen lines for instance (output data remain constant, for instance, whereas positioning parameters are reset in the program as required).

The I/O macro executes the following functions which can be controlled by parameters in the field description file or by call parameters:

- Positioning
- Validating and displaying a pre-assigned value
- Validating, formatting and displaying an input
- Port-specific attributes (colour, underlining, blinking, etc.)
- Displaying error messages in the form of input descriptions in the message line
- Manual question mark function
- Automatic question mark function in connection with User Administration or HELP
- Temporary deactivation of the automatic question mark function
- Provision of a text and setting up the window for manual or automatic HELP (can be set in HELP itself or User Administration)
- Checking the use of keys
- Displaying the allowed softkeys

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Inputs are validated within the I/O macro itself. Return parameters are only passed on to the application program when a correct input has been acknowledged with an allowed release key.

Depending on the type of field, the formatted input, the value or length of the input, the release key or an abort message can be evaluated as return parameters of the I/O macro.

Section 9.9.2 of this manual outlines the way in which fields are handled for the various inputs/outputs and release keys.

The individual field description parameters are explained in section 9.9.3. This is followed by two sections dealing with the field description maintenance utility (IOM FIELD DESCRIPTION) and CALL 9 which transfers hard key strings and softkey strings to the I/O macro and the workstation.

Keyboard Input/Screen Output

9.9.1 Calling the I/O macro

Syntax:

IOM <SVar1>, <NVar1>, <SVar2>, <SVar3>, <NVar2>, <NVar3>,
<SVar4>, <NVar4>, <NVar5>, <SVar5>

<SVar1> : Name of the parameter file.
If the same parameter file is used in several I/O macro calls, this variable only has be to filled for the first call.

<NVar1> : Record number of the field description in the parameter file (4% variable)

<SVar2> : Override specifications (max. 13 bytes)
The set control bytes in this string specify which parameters in the field description file can be overridden by the IOM statement:

ASCII 1: Byte set
ASCII 0: Byte not set

The set control bytes have the following meaning:

Byte 1 = Column number from <SVar3>
" 2 = Line number from <SVar3>
" 3 = Skip status from <SVar3>
" 4 = Max. input length from <SVar3>
" 5 = Min. input length from <SVar3>
" 6 = Number of decimal places from <SVar3>
" 7 = Hard keys from <SVar3>
" 8 = Softkeys from <SVar3>
" 9 = Attributes from <SVar3>
" 10 = Minimum value for numeric or date input from <NVar2>
" 11 = Maximum value for numeric or date input from <NVar3>
" 12 = Deactivation of automatic question mark function for the input
" 13 = Pre-assigned value from <SVar4> or <NVar4>

The override specifications only have to be made up to the required position (without gaps).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

<SVar3>: Override string (variable length).
The override string contains the following information:

Byte 1/2 = Column number
" 3/4 = Line number
" 5 = Skip status
" 6/8 = Max. input length
" 9/11 = Min. input length
" 12/13 = Number of decimal places
" 14/37 = Allowed hard keys (0 = allowed, 1 = not allowed)
" 38/53 = Allowed softkeys (0 = allowed, 1 = not allowed)
" 54/n = Attributes (for each attribute only the last 2 bytes of the octal control code)

The number of hard keys and softkeys corresponds to the structure of the parameter record and not to the actual number of keys that exist.

The override specifications must be made up to the required position (without gaps).

<NVar2>: Minimum value for numeric or date input (4% variable).

<NVar3>: Maximum value for numeric or date input (4% variable).

<SVar4>: Alphanumeric pre-assigned value/return of alphanumeric input or edited numeric input. This variable must be dimensioned for at least 3 bytes.

Keyboard Input/Screen Output

<NVar4>: Pre-assigned numeric or date value/return of numeric or date input/length of alphanumeric input (4% variable).

<NVar5>: Status variable (1% variable).
If a "2" is entered here as the initial value, an input is expected here irrespective of the current skip status (= type of field handling).
Return values only refer to the position number of release keys in the hard key or softkey string that was transferred by means of CALL 9 and can have the following values:

00 - 29 : Hard key used (position number in hard key string, "0" = "CR")
30 - 49 : Softkey used (position number in softkey string + 29)
60 : Cancellation of input by assigned virtual port.

<SVar5>: I/O macro work area (min. length 1024 bytes).
This variable is not allowed to be used for other purposes in the program.

Note:

Non-overridable parameters:

- Input/output type
- Type of sign
- Date input/output format
- Numeric input/output format
- Characters allowed
- USING mask
- HELP text definition

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Possible error: BASIC ERROR # 38

Meaning:

- Incorrect call
  + <SVar5> less than 1024 bytes
  + Incorrect type of variable
  + Incorrect dimensioning
  + Not enough call variables

- Incorrect record number for the parameter file

- Parameter file does not exist

- Problems reading the parameter file

- Incorrect parameters
  + Incorrect skip status
  + Incorrect input/output type

- TF.PARAM does not exist

- Problems reading TF.PARAM

- Error in the USING mask

- Error reading a numeric variable

- A parameter is overridden by the program but the corresponding specifications are missing in <SVar3>.

Additional error status in NUMES
see 70/82 P. 4-123

9.9.2 Handling of fields

9.9.2.1 Handling of pre-assigned values and inputs

- Alphanumeric input/output

  A pre-assigned value is displayed left-justified at the specified position and the cursor is positioned to the beginning of the field.

  If validation of the pre-assigned value reveals an error, a message is output and the pre-assigned value is displayed and has to be corrected.

  If a value has been pre-assigned and a release key is pressed without a character having been changed, the pre-assigned value counts as an input.

  If characters are changed, the input is accepted up to the end of the field in the same way as for the "duplicate field" function.

  Validation of an alphanumeric input comprises the following:

  - Minimum number of bytes
  - Maximum number of bytes
  - Allowed characters (see field description parameters)

  An error message is output in the event of an incorrect input and this now has be be corrected. In conclusion, the value and length of the input are returned to the application program.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* • Numeric input/output
*
* A pre-assigned value is edited in accordance with the specified
* output mask and displayed left-justified at the specified
* position. The pre-assigned value can be accepted with "CR" or a
* new input can be made with or without thousands separators. The
* validated input is displayed in accordance with the output
* mask.
*
* Validation of a numeric input comprises the following:
*
* - Max./min. number of digits
* - Number of integer places
* - Number of decimal places
* - Max./min. input value
* - Allowed characters (only numerals, thousands separators and
* decimal sign, the latter having to match the one defined in
* the system configuration)
* - Sign
* - Input format if thousands separators were entered.
*
* In conclusion, the formatted input (if a format is specified)
* and the value of the input are returned to the application
* program.

Keyboard Input/Screen Output

* • Date input/output
*
* A pre-assigned date is displayed left-justified without sepa-
* rators and corresponds to the SYSMOD setting (format YYMMDD or
* YYYYMMDD). An error message is output if zero is the pre-assigned
* value. An input can be with or without separators.
*
* Validation of date inputs comprises the following:
*
* - Max./min. input value
* - Allowed separators (Ø|,|-|/|:|;)
* - Input format if the date was entered with separators
* - Validity (see CALL 32)
* - Min./max. number of digits
*
* In conclusion, the date is returned to the application program in
* output format (maybe with separators) and as a numeric value.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* 9.9.2.2 Handling of release keys
*
* Inputs can be completed with any release key ("CR", hard key or
* softkey) which is defined as allowed in the parameter record or
* in the call parameters.
*
* The key that is used can be tested by the program. In this way,
* for instance, it is possible to go to a new program branch by
* entering a softkey.
*
* When an input is made, the release key is checked first and then
* the actual input.
*
* Input fields without a pre-assigned value can be exited by just
* pressing any allowed key other than "CR". In this case, the input
* is not validated and only the release key code is returned to the
* application program.
*
* If a value has been pre-assigned to a field and "CR" is pressed
* as the release key, the pre-assigned value is always interpreted
* as an input and validated.

Keyboard Input/Screen Output

- Handling of hard keys

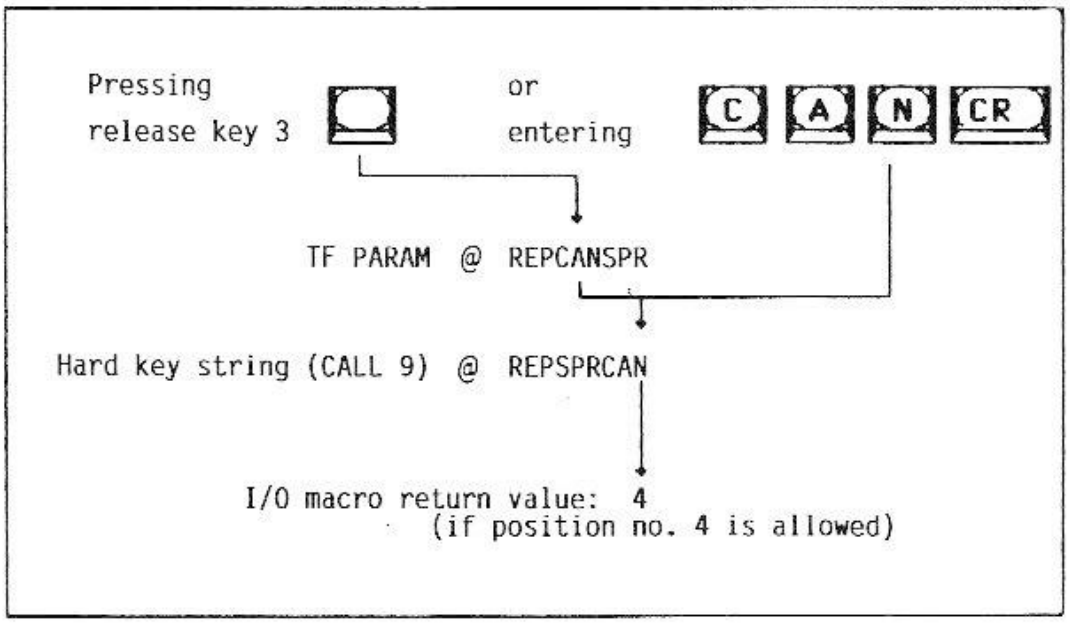
A function which has a hard key label (such as CAN) can be activated by pressing the corresponding hard key or by explicitly keying in the character sequence (in this example "C", "A", "N" and "CR").

Only the assignment of release keys to hard key labels is taken from the hard key string in the TF.PARAM file.

The I/O macro does not use the key code as a parameter but the position of a hard key label in a hard key string. This hard key string is transferred to the I/O macro by means of CALL 9, function 4 (PRINT HARD KEYS).

The sequence of the hard key labels in this string does not have to match the sequence in TF.PARAM. The position defined in the hard key string is always returned to the application program so that release keys can be evaluated irrespective of the assignment of hard key labels.

Thus application programs can use the assignment of the hard key label to the position in the hard key string (CALL 9) without knowing the current assignment of keys.



© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

By means of the sequence in the hard key string, hard keys can be flagged as allowed or not allowed in the parameter file or in call parameters.

When a hard key is pressed, the I/O macro takes the corresponding label from TF.PARAM and looks for it in the hard key string.

When a hard key is pressed without a previous input and without a pre-assigned value in the field, the corresponding function label is displayed in the input field.

When a character sequence is keyed in directly for a function, it is looked for immediately in the hard key string. The position number in the string is used to check whether the hard key is allowed. In the event of an error, the I/O macro displays the allowed hard key labels and a new input is required. The old input is retained and the cursor is located at the beginning of the field. Now it is only necessary to press a key that is allowed.

Keyboard Input/Screen Output

* • Handling of softkeys
*
* Softkeys are part of the new BA26 and BA34 workstations. So
* that application programs can be written irrespective of the
* workstation in question, softkeys are handled differently
* depending on the type of workstation.
*
* - Softkeys for BA26/BA34
*
* CALL 9 tells the workstation that softkeys are going to be
* used (function 0, "OPEN SOFTKEYS") and which ones are in-
* volved (function 1, "PRINT SOFTKEYS").
*
* Function 1 makes the softkey string (up to 9 softkeys of 7
* characters each) available for the BA26 screen. The allowed
* softkeys are displayed when the I/O macro is called.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* - Softkey emulation for the BA13
*
* The I/O macro emulates the handling of softkeys on the BA13
* using line 23 to display the softkeys. This means that one
* line less is available for the application program.
*
* CALL 9, function 1 "PRINT SOFTKEYS" transfers the softkey
* labels to the I/O macro. When the I/O macro is called,
* non-allowed softkeys are displayed in the background and
* allowed softkeys in the foreground.
*
* If no text has been defined for any of the softkeys (7
* blanks), the I/O macro outputs a pre-defined string at this
* position (7 hash signs).
*
* As it is not possible to assign special release keys to the
* softkeys on the BA13, a particular facility has been pro-
* vided. The release key with value KEY (0) = 2 on the right
* side of the keyboard is defined as the mechanism for the I/O
* macro to branch to line 23 where a "?" appears and a softkey
* number (position of the softkey text) is expected to be
* entered.
*
* Entry of a zero causes a return to the last input field. When
* a number between "1" and "9" is entered (without acknow-
* ledgement by a release key), the I/O macro performs a vali-
* dity check.
*
* When an allowed softkey number has been entered, it is dis-
* played in line 23, an audible signal is emitted and the I/O
* macro validates the input for the field in which the softkey
* feature was called or it directly executes the function
* indicated by the softkey text.
*
* An audible signal is also emitted if an illegal softkey
* number is entered but in this case an error message is output
* (illegal command) and a new softkey number is expected. In
* other words, you can only exit the softkey emulation by
* entering "0" or one of the allowed numbers between "1" and
* "9".

9.9.3 Field description parameters

This section describes the field parameters that can be controlled with the I/O macro and that are needed for calling the I/O macro.

The following details always have to be specified when creating a field description record in a parameter file:

- Column number
- Line number
- Skip status
- Input/output type
- Maximum input length (number of digits/bytes)
- Minimum input length (number of digits/bytes)
- Number of decimal places
- Type of sign
- Output mask

The following details can be specified in addition:

- Allowed hard keys
- Allowed softkeys
- HELP text definition
- Minimum input value
- Maximum input value/allowed characters
- USING mask (numeric input/output format)
- Pre-assigned value
- Attributes

The parameters in the parameter file can be overridden when the I/O macro is called except for the following:

- Input/output type
- Type of sign
- Output mask
- Allowed characters
- USING mask
- HELP text definition

Furthermore, the automatic "?" function can be deactivated when the I/O macro is called.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.9.3.1 Column number

The column number must range from "0" to "79".

9.9.3.2 Line number

The line number must range from "0" to "22".

9.9.3.3 Skip status

The skip status enables the type of field handling to be defined according to the following criteria:

- Displaying the pre-assigned value
- Validating the pre-assigned value according to input criteria
- Overwriting the pre-assigned value (input field)

Outline of subfunctions:

Skip status	Display pre-assigned value	Validate pre-assigned value	Input
0	Yes	No	Only upon error
1	No	Yes	Only upon error
2	Yes	Yes	Only upon error
3	Dots only	No	Yes
4	Yes	No	Yes
5	Dots only	Yes	Yes
6	Yes	Yes	Yes
8	Display pre-assigned value without validation or return (used to display prompts)		

Skip status 7 has not been implemented.

Keyboard Input/Screen Output

If an error is detected when the pre-assigned value is output or when this, the return parameters or the input is validated, the following error routine is executed irrespective of the skip status:

An error message is output in line 24. The incorrect value is displayed and the cursor is positioned to the beginning of the field. An input is required and this is validated. An incorrect input remains visible, the cursor is positioned back to the beginning of the field and another input is required.

- Skip status = 0

  A field is formatted as follows:

  - Output of pre-assigned value

  + If the length of the pre-assigned value is less than that of the field, the remaining positions within the field are cleared.
  + If the length of the pre-assigned value is greater than that of the field, then the part that fits is output and the I/O macro branches to the error routine.

  - Dots are output throughout the field if no value has been pre-assigned.

  The pre-assigned value is subsequently interpreted as an input and transferred to the return parameters. If validation of the return parameters reveals an error, there is a branch to the error routine.

  Schematic representation of procedure:

```
 10 Display pre-assigned value
 20 Set return parameters
 30 If no error, go to 100
 40 Error message
 50 Position cursor to beginning of field
 60 Start input
 70 Validate input
 80 If no error, go to 20
 90    Go to 40
100 Return
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Note: Skip status 0 and skip status 2 only differ if an input is required as a result of an override in the I/O macro. In this case, an incorrect pre-assigned value is displayed without an error message with regard to skip status 0. Only when this pre-assigned value is acknowleged as an input is there any validation with an error message.

- Skip status = 1

The pre-assigned value is not displayed (the input field remains unchanged). The pre-assigned value is validated. In the event of an error, it is output and there is a branch to the error routine.

Schematic representation of procedure:

```
 10 Validate pre-assigned value
 20 If no error, go to 110
 30 Error message
 40 Output pre-assigned value
 50 Position cursor to beginning of field
 60 Start input
 70 Validate input
 80 If no error, go to 110
 90     Error message
100     Go to 50
110 Set return parameters
120 Return
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Skip status = 2

  The pre-assigned value is displayed (see skip status 0) and validated. There is a branch to the error routine in the event of an error.

  Schematic representation of procedure:

```
 10 Display pre-assigned value
 20 Validate pre-assigned value
 30 If no error, go to 100
 40 Error message
 50 Position cursor to beginning of field
 60 Start input
 70 Validate input
 80 If no error go to 100
 90     Go to 40
100 Set return parameters
110 Return
```

- Skip status = 3

  Instead of a pre-assigned value, dots are displayed throughout the field and an input is requested. The input is validated and there is a branch to the error routine in the event of an error.

  Schematic representation of procedure:

```
10 Display dots
20 Position cursor to beginning of field
30 Start input
40 Validate input
50 If no error, go to 80
60   Error message
70   Go to 20
80 Set return parameters
90 Return
```

* • Skip status = 4
*
* The pre-assigned value is displayed (see skip status 0) and an
* input is requested. The input is validated and there is a
* branch to the error routine in the event of an error.
*
* Schematic representation of procedure:
*
* 10 Display pre-assigned value
* 20 Position cursor to beginning of field
* 30 Start input
* 40 Validate input
* 50 If no error, go to 80
* 60 Error message
* 70 Go to 20
* 80 Set return parameters
* 90 Return
*
*
* • Skip status = 5
*
* Dots are ouput throughout the length of the input field. The
* pre-assigned value is validated and there is a branch to the
* error routine in the event of an error. Whatever happens, the
* cursor is positioned at the beginning of the field and an input
* is requested. There is a branch to the error routine in the
* event of an error.
*
* Schematic representation of procedure:
*
* 10 Display dots
* 20 Validate pre-assigned value
* 30 If no error, go to 50
* 40 Error mesage
* 50 Position cursor to beginning of field
* 60 Start input
* 70 Validate input
* 80 If error, go to 40
* 90 Set return parameters
* 100 Return

Keyboard Input/Screen Output

- Skip status = 6

  The pre-assigned value is displayed (see skip status 0) and validated. A message is displayed in the event of an error. Whatever happens, the cursor is positioned to the beginning of the field and an input is requested. The input is subsequently validated. There is a branch to the error routine in the event of an error.

  Schematic representation of procedure:

```
 10 Display pre-assigned value
 20 Validate pre-assigned value
 30 If no error, go to 50
 40  Error message
 50 Position cursor to beginning of field
 60 Start input
 70 Validate input
 80 If error, go to 40
 90 Set return parameters
100 Return
```

- Skip status = 8

  This skip status is only used to display prompts. No validation takes place and no return parameters are set.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* 9.9.3.4 Input/output type
*
* There are four different input/output types:
*
* 0 = Alphanumeric with blank elimination
* 1 = Numeric
* 2 = Date
* 3 = Alphanumeric with blank padding
*
* The input/output type cannot be overridden when the I/O macro is
* called.
*
* • Alphanumeric input/output
*
* Alphanumeric inputs can be made in two ways:
*
* - With blank elimination ("0")
*
* Any blanks on the right of a pre-assigned value/input are
* eliminated before validation. The input length that is
* returned to the program then refers to the string that is
* left after this elimination process.
*
* - Padding with blanks ("3")
*
* When the pre-assigned value or input is transferred to the
* return parameters, the field is padded with blanks up to the
* end. If "allowed characters" have been defined, then blanks
* must also count as being allowed. Otherwise an error occurs
* ("illegal character").
*
* Note: The input of "↑" in an alphanumeric field is only
* interpreted as a hard key when input/output type 3 has
* been defined because the upward arrow is stored in the
* hard key string as "↑ ".

- Numeric input/output

  Thousands separators can be entered here. The input/output of thousands separators and the decimal sign is defined by the setting in SYSMOD. A sign can only be entered as the first character. Upon output, it is formatted in accordance with the parameter "type of sign" in the parameter record.

- Input/output of a date

  The date is validated in the same way as for CALL 32. Input of a "0" date is only allowed when exactly one zero is entered. No validation takes place in this case. Otherwise the input complies with the format that is set in SYSMOD. The following separators can be entered:

  ¢ , - / : ;

9.9.3.5 Maximum input length (no. of digits/bytes)

This parameter corresponds to the actual field length in respect of alphanumeric inputs.

For a numeric input, this parameter refers to the number of digits (integer and decimal places) which are allowed to be entered. The sign, thousands separators and the decimal sign do not have to be counted.

With regard to the input of a date, the maximum input length consists of the number of digits plus the number of separators if the latter are allowed.

If the maximum input length is less than three, the field is extended to three characters.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* 9.9.3.6 Minimum input length (no. of digits/bytes)
*
* With regard to a numeric input, this parameter refers to the
* number of digits (integer and decimal places) which have to be
* entered. The sign, thousands separators and decimal sign do not
* need to be counted.
*
* With regard to input of a date, the number of separators has to
* be added if they are allowed.
*
*
* 9.9.3.7 Number of decimal places
*
* The number of decimal places can be the same as the maximum
* number of digits. "0" means that no decimal places are allowed.
*
*
* 9.9.3.8 Type of sign
*
* This parameter is only evaluated for the output of a field when
* no USING mask has been specified:
*
* 0 = No sign allowed
* 1 = Leading sign
* 2 = Trailing sign
*
* The type of sign cannot be overridden.
*
* In inputs, the sign must always be the first character.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

9.9.3.9 Output mask

The following masks can be specified:

Numeric	Date
0 = Thousands separators	YYMMDD
1 = Leading zeros	YYYYMMDD
2 = Thousands separators and leading zeros	YYWW
3 = Without mask	YYYYWW

- Output mask for numeric input/output

  This parameter is evaluated when no USING mask has been specified and only refers to output. In this case, the I/O macro builds a USING mask itself according to the following criteria:

  - The sign is only output for negative numbers
  - A floating sign is used for mask 0 (thousands separators) and 3 (without mask)
  - Allowance is made for the number of decimal places and integer places (max. number of digits minus number of decimal places)

  The advantage of this parameter over the USING mask is that no rigid format has to be specified as the format depends on the input length and on the parameters "min./max. number of digits" and "type of sign".

- Output mask for input/output of dates

  This mask only applies to the return parameters. In the form specified here, the pre-assigned value as well as the minimum and maximum input values have to be defined.

Keyboard Input/Screen Output

* 9.9.3.10 Allowability of hard keys
*
* The hard keys can be defined as allowed or non-allowed in the
* same sequence as provided in the hard key string by CALL 9.
*
* The inputs made in the field description maintenance utility are
* "Y" and "N" but these are stored in the parameter record as "0"
* and "1" and are also overridden in this way.
*
*
* 9.9.3.11 Allowability of softkeys
*
* The softkeys can be defined as allowed or non-allowed in the same
* sequence as provided in the softkey string by CALL 9.
*
* The inputs made in the field description maintenance utility are
* "Y" and "N" but these are stored in the parameter record as "0"
* and "1" and are also overridden in this way.
*
*
* 9.9.3.12 HELP text definition
*
* It is possible to define a HELP text together with the window
* position and size for each input field. The HELP text is speci-
* fied by means of the manual ID and the text number. The I/O macro
* transfers these details to the HELP processor so that information
* can be provided before an input by pressing the HELP key or
* automatically in the so-called training mode.
*
* The definition of the HELP text cannot be overridden when the I/O
* macro is called.

* 9.9.3.13 Minimum input value
*
* A numeric or date input is checked against this value. The same
* applies correspondingly when the pre-assigned value is validated.
*
*
* 9.9.3.14 Maximum input value/allowed characters
*
* A numeric or date input is checked against the maximum input
* value. The same applies correspondingly when the pre-assigned
* value is validated.
*
* The "allowed characters" parameter only applies to alphanumeric
* inputs/outputs. If no specifications are made, all characters are
* allowed. The allowed characters are validated in the same way as
* for CALL 62.
*
* It is possible to enter allowed characters individually or define
* a range of them, the " " character is interpreted as "to".
*
* Example:
*
* A B C D X Y Z
* A D X Y Z
* 0 5 . ,
*
* It is not possible to enter octal codes as this would cause
* errors in the I/O macro.
*
* "N" has to be entered if no check of allowed characters is
* required.
*
* The allowed characters cannot be overridden when the I/O macro is
* called.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* 9.9.3.15 USING mask
*
* Any USING mask specified here corresponds to the editing involved
* in "PRINT USING".
*
* When this parameter has been set, the I/O macro no longer evalu-
* ates the parameters "type of sign" and "output mask". The rigid
* structure of the USING mask can mean that some integer and
* decimal places are not output in certain circumstances if the
* parameters "maximum number of digits/bytes" and "type of sign"
* are not taken into consideration when defining the using MASK.
*
* This parameter cannot be overridden when the I/O macro is called.
*
*
* 9.9.3.16 Pre-assigned value
*
* Any pre-assigned value is handled like an input.
*
* If a date is pre-assigned, the value must be specified in the
* format of the output mask.

9.9.3.17 Attributes

When the I/O macro is called, all active attributes (such as intensity, underlining, blinking and the like) are deactivated. 'SF' (start foreground) and the specified attributes which are possible at the workstation in question are then activated. These attributes are deactivated again before the return from the I/O macro.

There are exceptions, however, in which the output is always without attributes:

- Occurence of a background blank as a field delimiter
- Occurence of error messages
- Output of information in the "?" function

To override the attributes in the I/O macro call, a string with octal control codes must be specified.

Character attributes	Control code	
Activate blinking	←201←	←201←
Activate underlining	←202←	←201←
Activate strike-out	←203←	←201←
Activate subscript	←204←	←201←
Activate superscript	←205←	←201←
Activate reverse video	←206←	←201←
Activate upper line of double height	←207←	←201←
Activate reduced intensity	←210←	←201←
Activate double width	←211←	←201←
Activate character-ignore	←212←	←201←
Activate protected field	←213←	←201←
Activate lower line of double height	←214←	←201←

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Colour controls	Control	
Set frame colour	220	2xx
Set background colour	221	2xx
Set foreground colour	222	2xx
Activate background colour	223	2xx
Activate foreground colour	224	2xx

Colour	Code
Black	200
Red	201
Green	202
Yellow	203
Blue	204
Magenta	205
Cyan	206
White	207
Black + reduced intensity	210
Dark red	211
Dark green	212
Olive	213
Dark blue	214
Purple	215
Dark cyan	216
Grey	217

Keyboard Input/Screen Output

9.9.3.18 Automatic "?" function

The automatic "?" function is activated in the HELP parameter selector or in the user administration facility. When the automatic "?" function has been activated, the allowed elements of the hard key string are displayed as standard before an input.

A call variable of the I/O macro makes it possible to deactivate the automatic "?" function for the current input. In this way, the application program can output its own error messages without them being overwritten immediately by the display of the allowed keys when the I/O macro is called.

If the automatic "?" function has not been activated, it is possible to look up the allowed keys nonetheless by entering "?".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

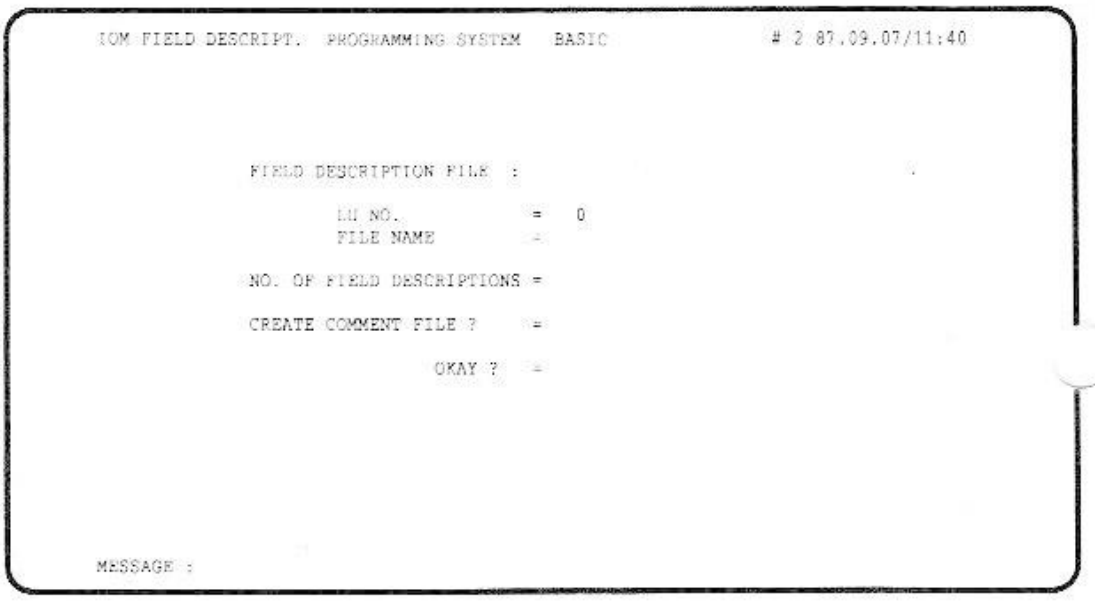
9.9.4 Field description maintenance utility

Parameter files (also called field description files) are created and maintained with the "IOM FIELD DESCRIPT." utility which is entered in a TAMOS selector.

An explanatory comment can be defined for each field description. This comment is stored in a separate comment file, the name of which is generated by appending a ".C" to the name of the parameter file. For this reason, the name of the parameter file must not exceed 12 characters.

Section 9.9.2 provides details of all field description parameters. During parameter entry, you can call up additional information by means of HELP.

The following mask is displayed when the "IOM FIELD DESCRIPT." utility is called:



Keyboard Input/Screen Output

The logical unit number and name of the parameter file have to be entered. If the specified file is not a parameter file, an error message is output. The following information has to be specified in addition for a new file:

- Number of field descriptions to be stored in the file

- Whether a comment file should be created ("Y") or not ("N").

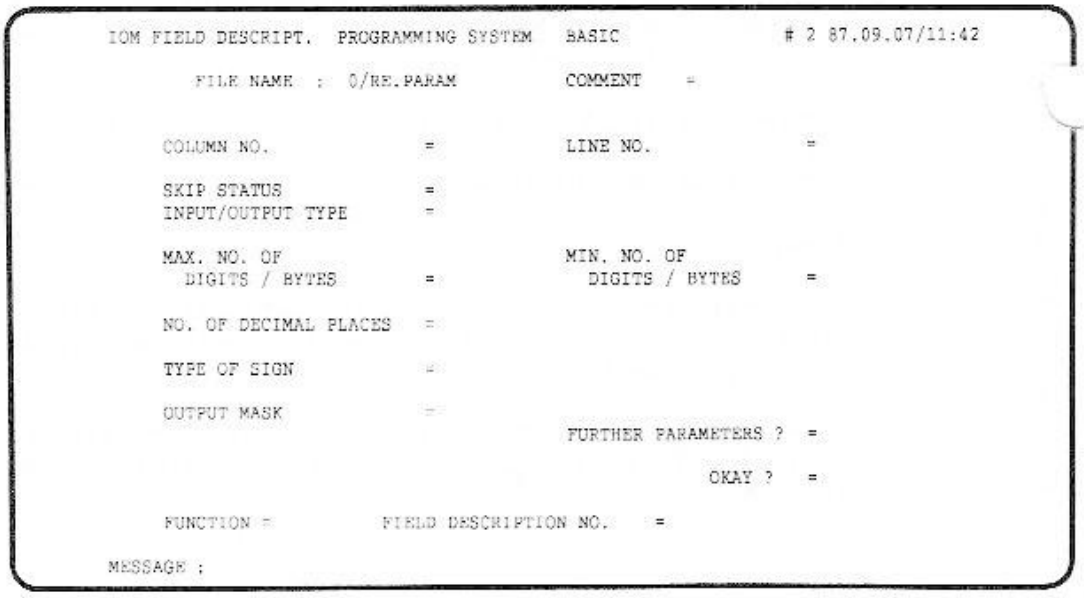
Note:

The parameter file is extended automatically as required. The number of field descriptions specified here is only used to initialize the file.

After answering "Y" in response to "OKAY", the parameter file is created and control is passed to the next mask for the selection of functions.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output



The following functions can be selected (in the "FUNCTION =" field).

- CRE Create a field description
- AME Amend a field description
- DEL Delete a field description
- REV Review a field description
- LST List field description numbers with comments
- PRT Print field description numbers with comments
- REO Reorganize parameter file
- COM Create log file to compare the differences between two parameter files in preparation for Disk Maintenance

The field description number is the record number in the parameter file as specified in the I/O macro call.

Keyboard Input/Screen Output

9.9.4.1 Creating a field description (CRE)

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM  BASIC              # 2 87.09.07/11:42

        FILE NAME  :  0/KB.PARAM         COMMENT    =

     COLUMN NO.              =           LINE NO.              =

     SKIP STATUS             =
     INPUT/OUTPUT TYPE       =

     MAX. NO. OF                         MIN. NO. OF
       DIGITS / BYTES        =             DIGITS / BYTES      =

     NO. OF DECIMAL PLACES   =

     TYPE OF SIGN            =

     OUTPUT MASK             =
                                         FURTHER PARAMETERS ?  =

                                                      OKAY ?   =

     FUNCTION =  CRE     FIELD DESCRIPTION NO.     =

MESSAGE :
```

Input of the field description number is followed by entry of the parameters which are always needed for a field description:

- Comment (only if comment file exits)
- Column number
- Line number
- Skip status
- Input/output type
- Max. number of digits/bytes
- Min. number of digits/bytes
- Number of decimal places
- Type of sign
- Output mask

If "Y" is entered for "FURTHER PARAMETERS?" and "OKAY", control is passed to the next mask where optional parameters can be entered. If "N" is entered for "FURTHER PARAMETERS?", the parameter record is created and the cursor is returned to the "FUNCTION =" field.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

Mask for additional parameters:

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC              # 2 87.09.07/11:46

      FILE NAME  :  0/RE.PARAM          COMMENT    :

   HARD KEY STRING     =
                  +----+----+----+----+----+----+----+----+----+----+----+----+
                  +----+----+----+----+----+----+----+----+----+----+----+----+

   SOFTKEY STRING      =     +----+----+----+ +----+----+----+ +----+----+----+

   HELP ID             =            MANUAL ID    =       TEXT NO.     =
     WINDOW PARAMETERS :            FIRST LINE   =       NO. OF LINES=
   MIN. INPUT VALUE    =
   MAX. INPUT VALUE
    /CHARACTERS ALLOWED=

   INPUT/OUTPUT FORMAT =
        (USING MASK)
   PRE-ASSIGNED VALUE  =
   ATTRIBUTES          =                                 OKAY ?  =


     FUNCTION  : CRE     FIELD DESCRIPTION NO.    : 103

MESSAGE :
```

The following additional parameters can be specified:

- Allowed hard keys
- Allowed softkeys
- HELP text definition
- Min. input value
- Max. input value/allowed characters
- Input/output format (USING mask)
- Pre-assigned value

Entry of "N" for "HARD KEY STRING" or "SOFTKEY STRING" causes "N" to be pre-assigned to all hard keys and softkeys, i.e. they are not allowed unless overridden when the I/O macro is called.

If "Y" is entered for "ATTRIBUTES" or if a value has been pre-assigned, "OKAY = Y" is followed by a branch to the next mask. If, however, "N" was entered for these two fields, the parameter record is created and the cursor is returned to the "FUNCTION =" field.

Keyboard Input/Screen Output

Input mask for attributes:

```
 VDM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC                 # 2 87.09.07/11:49

       FILE NAME  :  0/RE.PARAM          COMMENT    :

            SCREEN CONTROL = N                 COLOUR CONTROL = N        COLOUR
            ------------------                 ------------------        ------
 FLASHING                   =          SET FRAME COLOUR             =
 UNDERLINE                  =          SET BACKGROUND COLOUR        =
 STRIKE OUT                 =          SET FOREGROUND COLOUR        =
 SUBSCRIPT                  =          ACTIVATE BACKGROUND COLOUR   =
 SUPERSCRIPT                =          ACTIVATE FOREGROUND COLOUR   =
 REVERSE VIDEO              =
 DOUBLE HEIGHT (UPPER)      =
 DOUBLE HEIGHT (LOWER)      =
 INTENSITY                  =
 DOUBLE WIDTH               =
 IGNORE CHARACTERS          =
 PROTECTED FIELD            =


                                                        OKAY ?  =

     FUNCTION  : CRE     FIELD DESCRIPTION NO.    : 103

 MESSAGE :
```

The screen and colour control parameters are specified in this mask.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

After answering "Y" in response to "OKAY", the parameter record is created and the cursor is returned to the "FUNCTION =" field. However, this is preceded by the following mask if "Y" is entered in the "PRE-ASSIGNED VALUE" field of the previous input mask:

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC                # 2 87.09.07/11:54

      FILE NAME  :  0/RE.PARAM          COMMENT    :

           SCREEN CONTROL = N                COLOUR CONTROL = Y        COLOUR
           ------------------                -----------------         ------
 FLASHING                  =        SET FRAME COLOUR             = Y     4
 UNDERLINE                 =        SET BACKGROUND COLOUR        = Y     3
 STRIKE OUT                =        SET FOREGROUND COLOUR        = Y     2
 SUBSCRIPT                 =        ACTIVATE BACKGROUND COLOUR   = Y     1
 SUPERSCRIPT               =        ACTIVATE FOREGROUND COLOUR   = Y     0
 REVERSE VIDEO             =
 DOUBLE HEIGHT (UPPER)     =
 DOUBLE HEIGHT (LOWER)     =
 INTENSITY                 =
 DOUBLE WIDTH              =
 IGNORE CHARACTERS         =
 PROTECTED FIELD           =
  PRE-ASSIGNED VALUE  = ...


    FUNCTION  : CRE     FIELD DESCRIPTION NO.    : 104

MESSAGE :
```

You can now enter the required pre-assigned value.

* 9.9.4.2 Amending a field description (AME)
*
* After the number of the parameter record has been entered, the
* field description parameters are displayed and can be amended.
* The procedure and screen layouts are like the "CRE" function.
*
*
* 9.9.4.3 Deleting a field description (DEL)
*
* After the number of the parameter record has been entered, the
* field description parameters are displayed (cf. "CRE" function).
*
* The parameter record is deleted when "Y" is entered in response
* to "OKAY".
*
*
* 9.9.4.4 Reviewing a field description (REV)
*
* After the number of the parameter record has been entered, the
* field description parameters are displayed (cf. "CRE" function).
*
* This function is very handy if an existing record is basically
* going to be copied. When "CRE" together with a free record number
* are specified after this, the parameter values which were shown
* with "REV" are output as a default.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

9.9.4.5 Listing summarized field descriptions (LST)

After a range of parameter record numbers has been entered ("FIELD DESCRIPTION NO. ... TO = ..."), the numbers are listed together with the associated comments. If there is no comment file, a message is output to this effect and the field description numbers are displayed without any comments. If a parameter record does not exist, the message "FIELD DESCRIPTION DOES NOT EXIST" appears. Paging is possible with "CR" when a screen is full.

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC                  # 2 87.09.07/12:40

        FILE NAME  :  0/RE.PARAM          COMMENT    =

         FIELD DESCRIPTION  COMMENT
               103
               104

     FUNCTION :  LST     FIELD DESCRIPTION NO.    =  103    TO  =  104

  MASSAGE :      310                                PRESS <CR>
```

9.9.4.6 Printing summarized field descriptions (PRT)

This function is analagous to the "LST" function. The field description list is output on the printer that is defined as $LPT for the port in question by means of the "DEVICE ASSIGNMENT" program. In addition to the parameters for the range of field descriptions, it is necessary to specify the lines per page.

* 9.9.4.7 Reorganizing the file (REO)

Unused/unusable areas can arise in the parameter file through the deletion or extension of field descriptions. The reorganization process makes the data area contiguous again and updates the control area in the parameter file. The reorganization process entails the creation of an auxiliary file on the LU which is defined for the parameter file. The auxiliary file is called 'UT.WORKpp', where pp is the port number. This file has the same length as the reorganized parameter file. After successful reorganization, the auxiliary file is deleted again.

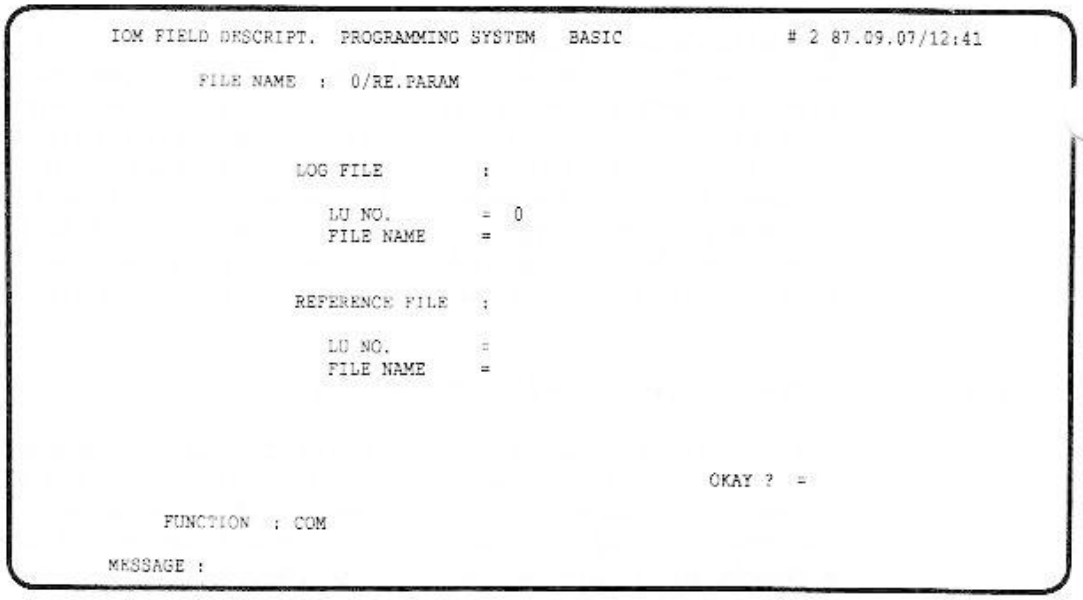
* 9.9.4.8 Comparing two parameter files (COM)

This function is used to make preparations for updating field description files by means of Disk Maintenance. Two parameter files are compared record for record and the varying parameter records are stored in a log file. This file can be transferred to a maintenance file and be used in the destination system as an input for the I/O macro maintenance utility which is called by Disk Maintenance.

After the function has been called, a follow-up mask is used to record the LU numbers and the names of the log and reference files.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output



After "OKAY" has been acknowledged with "Y", the function is performed and control is returned to the function input.

9.9.5 CALL 9

Transferring the current hard keys/softkeys to the I/O macro.

Syntax:

CALL <NExpr>, <NVar1>, <SVar1>, <SVar2>

<NExpr> : The value of this numeric expression is the number of the CALL subroutine (9 in this case).

<NVar1> : Function number:

0: OPEN SOFTKEYS

1: PRINT SOFTKEYS

On the BA13, the softkey string is transferred to the I/O macro work area. On the BA26, the string is also transferred to the workstation.

2: PRINT ALLOWED SOFTKEYS

This function only has an effect on the BA26. It is superfluous when the I/O macro is used because the I/O macro incorporates softkey handling for all types of worstation.

3: CLOSE SOFTKEYS

4: PRINT HARD KEYS

No additional parameters are needed for functions 0 and 3.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

* <SVar1> : This string value has to be pre-assigned as
* follows for functions 1, 2 and 4:
*
* - Function 1:
*
* 9 softkey labels of 7 bytes each
*
* The string must be dimensioned for a length of
* at least 63 bytes.
*
* - Function 2:
*
* 9 details of allowability as follows:
*
* 0: Allowed softkey
* 1: Non-allowed softkey
*
* The string must be dimensioned for a length of
* at least 9 bytes.
*
* - Function 4:
*
* Max. 24 hard key labels of 3 bytes.
*
* Not all hard keys have to be specified, only
* those that are needed.
*
* <SVar2> : In functions 1, 4:
*
* I/O macro work area (min. length 1024 bytes).

9.9.6 Example of the I/O macro

- Problem

  - Adding two numbers to be input on the screen
  - Displaying the total
  - Hard keys and softkeys are transferred to the I/O macro
  - If an input is concluded with an allowed hard key or softkey, the associated function is performed.

- Parameter file

  - Name : PB.PARAMETER
  - Record number: 101

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

- Parameters

The values stored in the parameter record are listed in the following masks.

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC                # 2 87.09.07/12:48

        FILE NAME  :  0/PB.PARAMETER      COMMENT    =

     COLUMN NO.              = 43         LINE NO.              = 10

     SKIP STATUS             = 5
     INPUT/OUTPUT TYPE       = 1

     MAX. NO. OF                          MIN. NO. OF
       DIGITS / BYTES        = 008          DIGITS / BYTES      = 002

     NO. OF DECIMAL PLACES   = 2

     TYPE OF SIGN            = 1

     OUTPUT MASK             = 0
                                          FURTHER PARAMETERS ?  = Y

                                                    OKAY ?     = Y

     FUNCTION =  REV     FIELD DESCRIPTION NO.    = 101

MESSAGE :
```

```
IOM FIELD DESCRIPT.  PROGRAMMING SYSTEM   BASIC                # 2 87.09.07/12:48

        FILE NAME  :  0/PB.PARAMETER      COMMENT    :

   HARD KEY STRING     = N
                +-----+----+----+---+----+----+----+----+----+----+----+----+
                +-----+----+---------+----+----+----+----+---------+----+----+

   SOFTKEY STRING      = N   +---------+---+ +----+----+----+ +----+----+----+

   HELP ID             = N          MANUAL ID    =        TEXT NO.    =
     WINDOW PARAMETERS :           FIRST LINE    =        NO. OF LINES=
   MIN. INPUT VALUE    =          1000.00                        *
   MAX. INPUT VALUE
    /CHARACTERS ALLOWED=          9999.00

   INPUT/OUTPUT FORMAT = N
        (USING MASK)
   PRE-ASSIGNED VALUE  = N
   ATTRIBUTES          = N                                OKAY ?  =  Y


     FUNCTION  : REV     FIELD DESCRIPTION NO.    : 101

MESSAGE :
```

Keyboard Input/Screen Output

- Description of the I/O macro calls

1st IOM call (number 1)

The data in the parameter record is not overridden.

2nd IOM call (number 2)

The line specification as well as the hard key and softkey specifications are overridden (<SVar2> = S2$ in the program). Then the first two hard keys and softkeys are allowed, i.e. in the example END/CAN and ESCAPE/REVERSE. They are overridden by specifications in <SVar3> = S3$ in the program (bytes 14, 15 and 38/39).

3rd IOM call (total)

The values "COLUMN", "LINE", "SKIP STATUS" and "MAX. INPUT LENGTH" are overridden and the "REVERSE VIDEO" attribute is activated (bytes 1,2,3,4,9 in <SVar2> = S2$ in the program).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Keyboard Input/Screen Output

```
10  REM
20  REM ********** DEFINITION OF VARIABLES FOR I/O MACRO CALL
25  REM
30  GLOBAL S1$(17),S2$(13),S3$(80),S4$(50),S5$(1024)
40  GLOBAL 4%,N1,N2,N3,N4,1%,N5
50  REM
60  REM ********** AUXILIARY FIELDS
70  REM
80  DIM B$(72),C$(63),D$(22)
100 DIM 4%,Z
110 IF ERR 0 GOSUB 760
120 LET D$="-"
130 PRINT 'CS'
140 REM
150 REM ********** OUTPUT MASK
160 REM
170 PRINT TAB (29,10);"1ST NO.  : ";
180 PRINT TAB (29,12);"2ND NO.  : ";
190 PRINT TAB (29,13);D$
200 PRINT TAB (29,15);"TOTAL    : ";
210 REM
220 REM ********** FILE NAME AND RECORD NO.
230 REM
240 LET S1$="00/PB.PARAMETER"  /* NAME OF PARAMETER FILE
250 LET N1=101  /* RECORD NO.
260 CLEAR Z
270 CLEAR S2$  /* CLEAR OVERRIDE PARAMETERS
280 CLEAR S3$  /* CLEAR OVERRIDE STRING
290 REM
300 REM ********** PREPARE HARD KEY / SOFTKEY STRING
310 REM             AND TRANSFER
320 REM
330 LET B$="ENDCANSPRCREAMEPRTREV@  "  /* HARD KEY STRING
340 LET C$="ESCAPE REVERSECREATE AMEND  REVIEW DELETE "
350 REM                                   SOFTKEY STRING
360 CALL 9,0  /* OPEN SOFTKEYS
370 CALL 9,1,C$,S5$  /* PRINT SOFTKEYS
380 CALL 9,4,B$,S5$  /* PRINT HARD KEYS
390 REM
410 REM ********** FIRST I/O MACRO CALL
420 REM
430 IOM S1$,N1,S2$,S3$,N2,N3,S4$,N4,N5,S5$
431 REM
432 REM
```

Keyboard Input/Screen Output

```
440 REM ********** PRE-ASSIGN SVAR2/3, SECOND I/O MACRO CALL
450 REM
460 LET S2$="0100001100000" /* OVERRIDE SVAR2 :
470 REM                          LINE, HARD KEYS AND SOFTKEYS
480 LET S3$="0",S3$  /* PRE-ASSIGN ZERO TO SVAR3
490 LET S3$(3,4)="12"  /* NEW LINE NO. = 12
500 LET S3$(16,37)="1111111111111111111111"
505 REM                      BYTE NO. 14, 15 = "00"
510 LET S3$(40,53)="11111111111111"
515 REM                      BYTE NO. 38, 39 = "00"
520 LET S3$(54)=""  /* STORE DELIMITER
530 IOM S1$,N1,S2$,S3$,N2,N3,S4$,N4,N5,S5$
540 REM
550 REM ********** PROGRAM CONTROL THROUGH HARD KEYS / SOFTKEYS
560 REM
570 ON N5 GOTO 740, 260  /* HARD KEY SWITCH
575 REM                       FOR "END/CAN"
580 ON N5-29 GOTO 740, 260  /* SOFTKEY SWITCH
585 REM                          FOR "ESCAPE/REVERSE"
590 REM
600 REM ********** PRE-ASSIGN HARD KEYS / SOFTKEYS
610 REM
620 LET N4=N4+Z  /* NUMERIC PRE-ASSIGNMENT
630 LET S2$="1111000100000"  /* COLUMN, LINE, SKIP STATUS
640 REM                          MAX. INPUT LENGTH, LENGTH
650 LET S3$(1,8)="42158009"
660 LET S3$(54)="_206__201_"  /* ATTRIBUTE : REVERSE VIDEO
670 REM
680 REM ********** THIRD I/O MACRO CALL
690 REM
700 IOM S1$,N1,S2$,S3$,N2,N3,S4$,N4,N5,S5$
710 CALL 9,3  /* CLOSE SOFTKEYS
720 STOP
730 CHAIN ""
744 PRINT TAB (0,24);"END"
750 CHAIN ""
760 REM ERROR HANDLING
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

10 Magnetic disks

10.1 Input/output concept

A logical input/output control system (IOCS) forms the interface between the Business BASIC language and peripheral units.
The IOCS is responsible for data management. The IOCS organizes data as files. This enables the programmer to access data in accordance with purely logical aspects irrespective of their physical arrangement on the storage medium.

Business BASIC provides the following statements for processing files:

- BUILD #
- CLOSE #
- KILL
- MAT READ #
- MAT WRITE #
- OPEN #
- PRINT #
- READ #
- SEARCH #
- WRITE #

10.1.1 File organization

The data in a file is split up into so-called records. The organization of a file determines the method by which the file is created on the medium and the possibilities which exist for accessing data records.
The IOCS supports the following types of file organization on the magnetic disk:

- Formatted files
- Contiguous (relative) files
- Indexed files
- Text files

Both sequential and random access to data is possible with each of these types of organization.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

10.1.2 Record pointers

The IOCS keeps a record pointer for magnetic disk files as soon as a file has been opened. If several users access the same file simultaneously, the pointer is kept for each individual user.

The record pointer contains the current access position in the file, i.e. the relative record number of the last record processed in the file. The contents of the record pointer can be tested by means of the BASIC function CHF.

In the case of text files, the record number of the current data block (1 block = 1 sector) and the byte position within this block which follows the character last processed is kept.

10.1.3 Channel concept

File processing is controlled via data channels. Each program has its own data channels and these are numbered starting from 0.

TAMOS uses the channels 0 and 1.

Exactly one file can be opened on each channel at any one time. Disk files can be opened on different channels at the same time by one or more users (file sharing).

10.1.4 File definitions

File definitions (record length, types of organization, etc.) are only necessary at file creation time when programming with Business BASIC. When a file is processed, the necessary parameters are taken from the file header.

10.1.5 File names

When processing a file, the file name need only be specified once, namely when the file is opened.

- Example:

  OPEN # 2,"INVOICE"

  This statement opens a disk file with the name INVOICE on channel 2.

It is then only necessary to specify the channel number for all further statements used to process the file.

The file name can be specified in a string variable or in the form of a string literal in a BASIC statement.

The file name comprises 1 to 14 letters, digits, periods or hyphens in the case of magnetic disk files. The first character of the file name must be a letter.

<FileName>::= <Le> [<Le> | <Di> | - | .]^13

- Examples:

  CL1-S.O, TEST1, T

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility mode or design.

Magnetic Disks

10.1.6 Concept of logical units

When processing files on magnetic disks, in particular when several drives are connected and when several application areas are simultaneously processed, it is essential that files on different magnetic disks be processed simultaneously.

For this reason, disks are distinguished by means of logical unit numbers (LU numbers) which are recorded on the disk itself and, when working with TAMOS, on the system disk in a so-called archive file. The system disk is always given the logical unit number 0.

If your system only has one magnetic disk, this is split up into at least two physical units which can each hold one logical unit. This ensures that you can swap an application area (one certain logical unit) for another while the operating system remains on the disk.

The logical unit concept is explained in greater detail in the "8870 User Manual" (model-related).

In BASIC programs, the LU number must be specified in the format:

```
<LU>/
```

directly before the file name when a file is opened by means of the OPEN # statement. If a file on logical unit 0 (system disk) is addressed, this specification can be omitted.

- Example:

  Opening the file INVOICE on data channel 2. The file is on logical unit 3.

```
20 OPEN # 2,"3/INVOICE"
```

Magnetic Disks

10.2 Structure of disk files

All data stored on a magnetic disk is organized in files. Each disk file comprises one or more blocks (sectors):

- one file header block and
- 0 to n data blocks.

In the case of formatted files and text files, header extension blocks can exist in addition to the file header block.
User data can be stored in each form of representation supported by BASIC

- ASCII
- BCD integer
- floating-point number

in data files, with the exception of text files. Text files only permit representation in ASCII.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

10.3 File sharing

NIROS permits several users to share disk files, even if the same data record is accessed simultaneously.
To this end, functions must exist to protect against:

- deleting files while several users are using them and
- uncontrolled, simultaneous updating of data records and data corruption connected with this.

Protection against deletion of files while other users are accessing them is described in the section "KILL statement".

Protection against uncontrolled, simultaneous updating is provided by the facility for locking specific data records.

Locking data records

A data record is locked if one of the statements

- MAT READ #
- MAT WRITE #
- READ #
- WRITE #

is executed and no ";" follows the last operand. In addition,

- SEARCH #

sets record locks with modes 1, 2, 3 and 5 if the last operand is followed by a ":".
The PRINT # statement always sets a record lock.

If several data records are read or written simultaneously by means of one of the statements MAT READ #, MAT WRITE #, READ # or WRITE #, the record lock only affects the record which is first read/written.

Magnetic Disks

Only one record can be locked on each data channel. If several records are to be locked simultaneously in a file, the file must be opened simultaneously on several channels and one record locked on each one.

Records cannot be locked in text files.

Effect of the record lock

A locked data record generally cannot be processed (read or written) by other users. If a file has been opened without the parameter "R", a change of task results when a locked record is addressed. Each time the program is activated, an attempt is made to read the record. The record only becomes available after it has been unlocked by the program which is locking it. No status message is issued.
When a file is opened with the parameter "R", BASIC error 123 is issued when a locked record is addressed and specific action can be taken.

Cancelling the record lock

The record lock can only be cancelled by the program which locked the record.

A record lock is cancelled when:

- the locked record is read or written again on the channel on which it was locked, and if ";" follows the last operand in the executing statement,
- another data record is read or written on the same channel,
- the channel on which a record was locked is closed or
- the program is terminated by one of the following conditions:

  - occurrence of an END statement
  - by means of "ESC" and "CTL-C" or "CTL-Y", "ESC" and "CTL-C".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

Deadlock

When operating with record locks, it is possible for two or more programs to lock each other out (deadlock). It should be ensured in the organization and programming phase that access situations do not occur which result in a deadlock.

- Example:

  Program A and program B are both processing the files X and Y. Both programs lock the data records during reading and unlock them when they are written back or when the next record is read.

  Sequence:

Time slice	Program	
1	A	Reading and locking record 15 in file X.
2	B	Reading and locking record 16 in file Y.
3	A	Evaluating the data read and reading record 16 in file Y.
4	B	Evaluating the data read and reading record 15 in file X.

  The two programs attempt to read the data record locked by the other program. In this case, at least one of the two programs must be abnormally terminated to enable the other to continue processing.

Magnetic Disks

10.4 Creating disk files

In addition to the programs in the EXPERT selector, the BASIC statement

- BUILD #

is available for creating disk files. This is described in detail in the chapter "BASIC statements".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

10.5 Accessing disk files

The Business BASIC statements:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

are available for accessing data records in disk files.

All disk files permit both random and sequential access.

The statement:

- SEARCH #

is available for processing the index area of indexed files.

Files are opened and closed by means of the statements:

- OPEN #
- CLOSE #

10.5.1 Opening files

Before data records in a file can be accessed, the file must be opened by means of the statement

OPEN #

on any free channel.

A disk file can be opened simultaneously by a program on several channels. However, only one file can be opened on each channel.

Disk files can be opened simultaneously by several independent programs.

10.5.2 Accessing files

File accessing differs according to the individual types of file organization and is carried out by means of different statements. For this reason, file accessing is described in detail for each of the various file types.

10.5.3 Closing files

After a file has been processed, the channel on which this file was opened must be closed with the aid of the statement:

- CLOSE #

This channel then becomes free for processing another file.

Channels are also closed if:

- an END statement occurs,
- a processor is called with the aid of a CHAIN statement,
- a program is abnormally terminated by "ESC" and "CTL-C" or "CTL-Y", "ESC" and "CTL-C".

Note:

Channels opened by BUILD # can only be closed by the statements CLOSE # and END.

Any number of files can be closed with the aid of one CLOSE # statement.

BASIC error 49 is issued if a channel is to be closed on which no file has been opened.
Several channels should not be closed with one statement as any errors which occur cannot be analysed adequately.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

10.6 Deleting files

BASIC provides the statement

KILL

for the deletion of files. KILL deletes the first two characters of the file name in the INDEX file of the disk. The disk blocks in use are entered as being free in the disk map.

Note:

If the file is still open by another user, the blocks in use are only deallocated at the next IPL or INSTALL.

10.7 CHF function

The CHF function enables the programmer to determine the current size of a file or the current access position within a file.
The argument of the function defines the channel number and the function to be executed.
Various specifications are supplied by CHF and various functions are possible for the different types of disk file:

Formatted files

<NExpr>	Function
Channel number	Number of the highest written data record +1.
Channel number + 100	Number of the data record which was last accessed.
Channel number + 200	Displacement following the last record processed (relative to the start of the block)
Channel number + 300	Record length in words

Magnetic Disks

Contiguous (relative) files

<NExpr>	Function
Channel number	Maximum number of data records which the file can hold.
Channel number + 100	Number of the data record which was last accessed.
Channel number + 200	Number of the first byte of the last data record processed, relative to the start of the block.
Channel number + 300	Record length in words.
Channel number + 400	Number of the highest written data record +1 if the EOF pointer is defined. This is only the case if the file has been opned at some time with OPEN "C". If the EOF pointer has not been defined, this function returns the number of data records which the file can hold.

Indexed files

<NExpr>	Function
Channel number	Number of the highest data record.
Channel number + 100	Number of the data record which was last accessed.
Channel number + 200	Byte address of the last data record processed, relative to the start of the block.

Channel number + 300 Record length in words.

Further information concerning the files status for indexed files can be obtained with the aid of SEARCH # - mode 1.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Disks

Text files

<NExpr>	Function
Channel number	Number of the highest written data block +1. For extended text files, however, the number of header extension blocks must be deducted from this figure.
Channel number + 100	Number of the current data block.
Channel number + 200	Number of the byte after the last byte transferred, relative to the start of the current block.

File Processing

11 File processing

11.1 Types of file organization

The 8870 system has a series of file organization types to provide the most suitable type of organization for each purpose.

The structure of the storage medium restricts the range of possible organization types. The organization types

- formatted file
- contiguous file
- indexed file
- text file

are available on disks. Each of these four file types permits

- random or
- sequential

processing.

However, it is only possible to organize and process the files

- sequentially

on the following peripheral units:

- magnetic tape
- magnetic tape cassette
- printer
- punched card reader
- paper tape reader/punch

due to the properties of the media.

The following sections provide a more detailed explanation of the various types of organization, including the programming of them.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.2 Formatted files

A formatted file consists of a file header and the appropriate number of data blocks required to hold all the data records that have been written. A formatted file does not occupy a contiguous area on the disk. The number of records is not specified when the file is created and any necessary extensions are made via a list of free disk blocks.

For data records with record numbers

> 128 * INT (256 / record length in words)

1 to 128 header extension blocks are created in addition to the file header.
A header extension block enables a maximum of 256 data blocks to be managed.

All the data records in a formatted file have the same format with respect to arrangement and type of the individual data fields. A data record can hold up to 64 different data fields. The format map contains the description of the format and is stored in the file header.

The maximum number of data records for a formatted file depends on the record length.

The number of records must not exceed 65535, however.

Minimum record length:	1 word
Maximum record length:	256 words

Data fields defined with the aid of strings by means of WRITE # not only occupy the space for the string but also an extra byte for a delimiter. Furthermore, the field of a string is padded to make up a whole word length if necessary (this applies to data fields created by means of WRITE # and MAT WRITE # ; see examples in the next section).

Formatted files allow both sequential and random access to data records.

File Processing

11.2.1 Creating formatted files

When a formatted file is created with the aid of BUILD # , only the file header is created.
The format map is created in the header by directly writing record 0 with the WRITE # or MAT WRITE # statement (<NExpr2> = 0). All data records in the file thus have the format specified for record 0.

If fields in record 0 are written with MAT WRITE #, these fields can only be processed in other records by means of MAT READ # and MAT WRITE # .

- Examples:

  - ```
    10 DIM A$(19),B$(10)     /*A and B implicitly dimensioned for 2%
    :
    150 BUILD # 2,"TEST"
    160 WRITE # 2,0;A$,A,B,B$;
    :
    240 CLOSE # 2
    ```

    Creating a formatted file named TEST with the following parameters on channel 2:

    ```
    - LU number          : 0
    - Record format      : Field 0 : String, 19 bytes + delimiter
                           Field 1 : Numeric, 2 words
                           Field 2 : Numeric, 2 words
                           Field 3 : String, 10 bytes + delimiter
                                     + filler
    ```

  - ```
    10 DIM 3%,A,B,C$(50)
    20 BUILD # 2,"FORFIL"
    30 WRITE # 2,0;A,B;
    40 MAT WRITE # 2,0,2;C$;
    :
    140 CLOSE # 2
    ```

    Creating a formatted file on channel 2 with the following data record format:

    ```
    Field 0 : Numeric, 3 words
    Field 1 : Numeric, 3 words
    Field 2 : Matrix field, 25 words (MAT WRITE # does not set a
              delimiter so no filler is needed either)
    ```

The first two fields in all other records of this file can only be processed with WRITE # and READ # and the third field with MAT WRITE # and MAT READ #.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.2.2 Accessing formatted files

Before data records in a file can be accessed, the file must be opened by means of the statement

- OPEN #

on any free channel.

The data records in formatted files are accessed by means of the statements:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

The MAT READ # and MAT WRITE # statements can, however, only be used for fields which were written in record 0 with the aid of MAT WRITE # , i.e. defined accordingly in the format map. Conversely, it is only possible to process such fields with MAT READ # and MAT WRITE #. The PRINT # statement only enables strings to be processed in formatted files.

All the statements permit both sequential and random access to data records in data fields.

- Sequential access

  Data records are accessed sequentially if no relative record number (<NExpr2>) is specified in the statement or if -1 is specified as the record number. If the data records are to be accessed sequentially but only certain fields of the data records transferred, <NExpr2> must be -1. The end of the file can be identified by the occurrence of ERROR # 52 which is also used to indicate records are missing in the file.

File Processing

- Examples:

  - Sequentially reading and transferring the data fields starting from field 0 to the variables A, B, C and X$.

    ```
    1510 READ # C;A,B,C,X$;
    ```

  - Sequentially writing and transferring the variables D, E, F and Y$ to the data records starting from field 3.

    ```
    1520 WRITE # C,-1,3;D,E,F,Y$;
    ```

- Accessing the record last processed

  If the value -2 is specified as the relative record number, the record last read or written is accessed. If -2 is specified immediately after opening the file, this results in BASIC error 51.
  This facility of accessing the data record last read/written makes it easier to update the existing contents of data records.

  - Example:

    Sequentially reading a file, updating and writing back the updated data records.

    ```
    150 READ # 3;A,B,C     /* SEQUENTIAL READING
    160 GOSUB 2000         /* UPDATE VARIABLES
    170 WRITE # 3,-2;A,B,C; /* WRITE BACK
    ```

    After the data record has been read, it is locked against access by other users and is unlocked again after the updated data has been written back.

- Random access

  If a value $\geq$ 0 is specified as the relative record number, this record is accessed directly.

  It is possible to change the access method at any time.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

- Example:

  Random reading of record 100 followed by sequential processing.

```
150 READ # 3,100;A$     /* RANDOM READING
160 GOSUB 2000          /* PROCESSING
170 READ # 3;A$         /* SEQUENTIAL READING
```

- Field processing

  All data records in a formatted file have the same format with respect to arrangement and type of the individual data fields. A check is made each time a record is accessed as to whether the variable types in the statement in question match the associated entry in the file header's format map.
  If they do not match, BASIC error 54 is issued. Data is not transferred if an error is detected.

  Direct access to particular fields in a data record is achieved by specifying the field number relative to the start of the record (field 0).
  The relative field number is specified in <NExpr3>, on condition that <NExpr2> is coded.

  The data is transferred field by field.

  - Numeric variables are converted logically, i.e. the values are converted like in the LET statement (e.g. 1% variables can be transferred to fields for 2% variables and vice versa).
    When this method is used, BASIC error 15 can result if an overflow occurs when transferring to a 1% variable.

  - In the case of strings, transfer is terminated when:

    - a delimiter occurs in the source string with READ # , WRITE # and PRINT # and when
    - the end of the source or destination string is reached with READ # , WRITE # and PRINT #.

Strings which are longer than their destination fields must not be transferred to a formatted file with the aid of the PRINT statement.

The MAT READ # and MAT WRITE # statements process variables in their entirety. Delimiters do not terminate transfer; this is done when the end of the source or destination string is reached.

File Processing

It is possible to process a formatted file without checking the field formats (<NExpr3>=-1 in the MAT READ # and MAT WRITE # statements). Transfer then starts at the first byte of the addressed record and ends at the last byte in the record or when the dimensioned length of the destination variable is reached.

- End-of-file handling

  If a READ # or MAT READ # statement addresses a data record which has not yet been written, BASIC error 52 is issued.

  Note:

  It is not possible to process several data records with one statement in the case of formatted files.

The CHF function enables the programmer to determine the current size of the file or the current access position within a file.

- Closing the file

  After the file has been processed, the channel on which this file was opened must be closed by means of the statement

  CLOSE #

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.3 Contiguous files

A contiguous (relative) file comprises the file header and the appropriate number of data blocks required to hold the maximum number of data records specified upon creation.

A contiguous file always occupies a contiguous area on the disk. The size of this area is established by the number of records and the record length.

The maximum number of data records in contiguous files is limited to 999,999.

```
Minimum record length:      1 word
Maximum record length:     64 K words
```

Contiguous files permit both sequential and random access to data records.

11.3.1 Creating contiguous files

Due to the fact that contiguous files occupy a contiguous area on a magnetic disk, the entire area must be reserved at the time the file is created.

This is done with the appropriate program in the EXPERT selector or in the BUILD # statement by specifying the number of data records and the record length. The specifications

- number of data records and
- record length in words

are separated from one another by means of ":"or "?" and placed in parentheses. They precede the file name.

• Example:

Creating a contiguous file with the name TEST on channel 2:

- LU number : 1
- Number of records : 1500
- Record length : 50 words

```
50 BUILD # 2,"(1500:50)1/TEST"
```

When calculating the record length, it should be noted that string variables occupy the dimensioned number of bytes + 1 and are flush behind any preceding variable.

Numeric variables always commence in a new word. If a string variable and a numeric variable are juxtaposed, this can result in one byte remaining unused.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

11.3.2 Accessing contiguous files

Before the data records in a file can be accessed, the file must be opened by means of the statement

- OPEN #

on any free channel.

Data records in contiguous files are accessed with the statements:

- READ #
- WRITE #
- PRINT #
- MAT READ #
- MAT WRITE #

All statements permit sequential and random access to data records.

- Sequential access

  A data record is accessed sequentially if the statement does not contain any specification of the relative record number (<NExpr2>) or if the relative record number is specified as being -1.
  If data is to be accessed sequentially but transfer is not to start at byte 0, <NExpr2> must be coded and equal -1.

File Processing

- Example:

  - Sequentially reading and transferring data, starting from byte 5, to variable A$. The data record is locked.

    ```
    150 READ # 3,-1,5;A$
    ```

  - Sequentially writing the variables A, B and A$ starting from byte 0. The data is stored as a string on the disk and the numeric variables are edited with a "USING" mask.

    ```
    150 PRINT # 3;USING "######";A,B,A$;
    ```

- Accessing the record last processed

  If the relative record number is specified as being -2, the record which was last read or written is accessed.
  If -2 is specified directly after opening the file, this results in BASIC error 51.
  The feature of accessing the last record processed makes it easier to update the contents of data records.

  - Example:

    Directly reading a record whose relative record number is specified in the variable R. The contents of the record are updated and subsequently written back.

    ```
    150 READ # 3,R;A,B,C     /* RANDOM READING
    160 GOSUB 2000           /* UPDATE CONTENTS OF RECORD
    170 WRITE # 3,-2;A,B,C;  /* WRITE BACK
    ```

File Processing

After the data record has been read, it is locked against access from other users and is not unlocked until the updated data record has been written back.

- Random access

  If a value $\geq$ 0 is specified as the relative record number, the addressed record is accessed directly.
  It is possible to change the access method at any time.

  • Example:

    Random reading of the record 120 followed by sequential processing.

```
150 READ # 3,120;A$       /* RANDOM READING
160 GOSUB 2000
170 READ # 3;A$           /* SEQUENTIAL READING
```

- Field processing

  When data is being processed in contiguous files, the IOCS does not verify the data types unlike in formatted files.
  The data is transferred identically. The programmer must ensure that the destination and source areas have the same field type.
  The possibility of specifying a byte address (displacement) in <NExpr3> relative to the start of the data record addressed means that transfer can begin from any byte (1st byte: byte 0).
  The byte addressed with <NExpr3> can be situated outside the data record addressed.
  If a data record comprises different field types (strings and numeric values), the following must be taken into account when establishing the displacement:

  - Numeric variables always begin in a new word (word boundary).
  - Strings are nearly always concluded with a delimiter (for exceptions, see special points below).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

Special points:

- The WRITE # statement stores delimiters:
  - after transfer of a literal,
  - after transfer of a string without indices,
  - after transfer of a string with indices in its entire dimensioned length and
  - after a substring if a delimiter occurs before the end of the substring.
- The READ # statement expects a delimiter to be stored in strings behind the last character to be transferred, i.e. the character following the last character which was transferred in a string is skipped.

Problems can therefore result if substrings are written by means of WRITE # because the corresponding READ # statement cannot transfer this data correctly.

- Example:

  The data written with the statement
  ```
  50 WRITE # 3;A$(5,9),B$;
  ```
  cannot be read correctly with the statement
  ```
  60 READ # 3;A$(5,9),B$
  ```

  Reason:

  ```
  Dimensions :  A$(9),B$(10)
  Contents   :  A$ = "ABCDEFGHI"
                B$ = "1234567890"
  ```
  After execution of the statement
  ```
  50 WRITE # 3;A$(5,9),B$;
  ```
  the data record contains the following:

  ```
  EFGHI1234567890
  ```

  No delimiter was stored. After execution of the statements
  ```
  55 CLOSE # 3
  57 OPEN # 3, "File"
  60 READ # 3;A$(5,9),B$
  ```
  the destination variables contain the following:

  ```
  A$(5,9)= "EFGHI"
  B$     = "234567890"
  ```

The "1" was not transferred because, after the end of A$ was reached, READ # expected a delimiter in the data record and skipped it.
The contents of this data record can only be transferred correctly if a READ # or MAT READ # statement is used to read to a string variable long enough to hold all the data.

- End-of-file handling:

If a data record is addressed which is located behind the end of the file, BASIC error 51 is issued.

Note:

In contiguous files it is possible to process several data records with one statement.

- Example:

A data record comprises a string with a length of 80 bytes. 5 data records are to be read with one statement.

```
10 DIM A$(80),B$(80),C$(80),D$(80),E$(80)
:
50 READ # 3,R;A$,B$,C$,D$,E$;
```

or:

```
10 DIM A$(400)
:
50 READ # 3,R;A$(1,80),A$(81,160),A$(161,240)...
```

or:

```
10 DIM A$(400)
:
50 MAT READ # 3,R;A$
```

It must be noted that even in such cases the record pointer is only incremented by one. After execution of one of these statements, the record pointer would have the contents R+1.

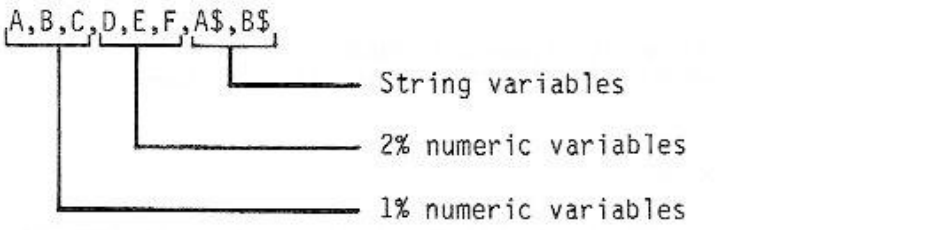
© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

- Rules for working with contiguous files

  - Data records comprising various field types should be structured so that the variables of a particular type are located one after the other.

    • Example:



    This enables capacity to be saved. In addition, this makes it easier to calculate the record length and displacements.

  - If possible, no substrings should be processed.

  The CHF function enables the programmer to determine the current size of the file or the current access position within the file.

- Closing the file

  After processing the file, the channel on which this file was opened must be closed with the statement

```
CLOSE #
```

11.3.3 End-of-file (EOF) pointer

This pointer contains the number of the highest written record +1 if the pointer has been initialized with OPEN # X;"C",... It can be accessed in a BASIC program with the CHF function (channel number + 400).

Value	Meaning
-1	File is empty (after OPEN # X;"C",...)
0	File has never been opened with OPEN # X;"C",...
>0	Current EOF pointer

OPEN # X;"C",... initializes the EOF pointer and sets it to -1.

BUILD # sets the EOF pointer to 0. The EOF pointer can only be activated by specifying CLOSE followed by an OPEN with "C".

WRITE # updates the EOF pointer in the file header if it is greater than 0 and the current record number is greater than EOF.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.4 Indexed files

An indexed file comprises the file header, 1 to 15 key directories and the area for data records. Each key directory is divided into master, coarse and fine blocks to accelerate retrieval.

An indexed file always occupies a contiguous area on a disk. The size of this area is established by the number of data records specified, the record length and the length of the keys.

Calculation of disk storage space:

A Number of disk blocks for data records:
Number of records * Record length / 256

B Number of disk blocks for fine area:
Number of records * 2 / INT(254/(key length +1))

C Number of disk blocks for coarse area:
Number of fine blocks / INT (254/(key length +1))

D Number of disk blocks for master area:
Number of coarse blocks / INT (254/(key length +1))

(Record and key length in words, 1 block = 256 words)

The number of disk blocks for the file (without the header) is the total of A plus, for each directory, B, C and D. The individual results have to be rounded up.

However if the

Total number of blocks * 256 / Record length

exceeds 65533, the calculation must be repeated using

Number of records * 2 / INT (254/key length +2))

in B as the record pointers then require two words.

Minimum record length:	1 word
Maximum record length:	64 K words
Minimum key length:	1 word
Maximum key length:	15 words

File Processing

11.4.1 Creating indexed files

When creating an indexed file with a BASIC program,

- the file area must be created, then
- each key directory must be defined individually, then
- all key directories must be initialized together, and then
- the (free) data records must be chained.

The file area is built by means of the statement

- BUILD #

as with a contiguous file by defining the number of records and the record length in words before the file name. These two specifications must be in parentheses and separated from one another by ":" or "?".
The number of records is made up of the number of data records and the number of records (equal length) which could occupy the index area.

The key directories are defined and initialized and the data records are chained by means of the SEARCH # statement:

SEARCH # <NExpr1>,<NExpr2>,<NExpr3>;<SVar>,<NVar1>,<NVar2>

<NExpr1> : Channel number.

<NExpr2> : Mode: 0 (Defining and initializing)
6 (Chaining the data records)

<NExpr3> : Directory number (irrelevant for mode 6):

<>0: Defining the directory
=0 : Initializing all defined directories

<SVar> : Irrelevant.

<NVar1> : When defining a directory, the key length must be supplied here and when initializing the directories, the number of data records must be supplied (irrelevant for mode 6).

<NVar2> : Status variable.

The directories must be defined in ascending order without any gaps commencing at 1.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Indexed Files

When determining the length of the data records, the rules listed in the section "Creating Contiguous Files" must be adhered to so that the data record length is always sufficient.
If an attempt is made to write beyond the end of a data record in an indexed file by means of WRITE # ,MAT WRITE # or PRINT # , BASIC error 82 is given.

- Example for creating an indexed file:

  Creating the indexed file PUB with 2000 data records each with a length of 30 words and two key directories each with a key length of 6 words.

  Number of blocks for one fine area:
  2000 * 2 / INT (254/(6+1)) = 112 (rounded up)

  Number of blocks for one coarse area:
  112 / INT (254/(6+1)) = 4 (rounded up)

  Number of blocks for one master area:
  4 / INT (254/(6+1)) = 1 (rounded up)

  All in all the index area requires

  2 * 117 = 234 blocks which means 234*256 words and is consequently equivalent to 1997 records of 30 words.

  The file must therefore be created with 3997 records each with a length of 30 words:

```
10 DIM V$(12)
20 IF ERR 0 GOSUB ...        /* ERROR HANDLING
30 BUILD #3,"(3997:30)PUB"   /* FILE AREA
40 LET N=6
50 FOR V=1 TO 2
60 SEARCH #3,0,V;V$,N,S      /* DEFINE DIRECTORY
70 IF S GOTO ...             /* STATUS <> 0
80 NEXT V
90 LET N=2000
100 SEARCH #3,0,0;V$,N,S     /* INIT. DIRECTORIES
110 IF S GOTO ...            /* STATUS <> 0
120 SEARCH #3,6,0;V$,N,S     /* CHAIN DATA RECORDS
130 IF S GOTO ...            /* STATUS <> 0
:
```

11.4.2 Accessing indexed files

Before the data records in a file can be accessed, the file must be opened on any free channel by means of the statement

- OPEN #

Indexed files occupy a contiguous area on the disk as in the case of contiguous files. The reserved area is divided into:

- 1 to 15 directories to hold the keys (index area) and
- the area to hold the data records (data area).

Indexed files permit both sequential and random access to the data records. The following possibilities exist for this:

- Sequential reading of the data area.
- Sequential reading of the index area and random access to the data records.
- Random access in the index area by specifying a key and random access to the associated data record.

In general, several BASIC statements are required to access indexed files.

Data records in indexed files are accessed by means of the statements:

- READ #
- MAT READ #
- WRITE #
- MAT WRITE #
- PRINT #

The same rules apply for accessing data records in indexed files as for contiguous files. It must, however, be noted that the first data record in the data area is not record 0. The number of the first data record relative to the beginning of the file can be established by means of the statement SEARCH # - mode 1.

Using the semicolon in a PRINT # statement behind characters which are to be output leads to errors. WRITE # should be used instead for indexed files.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

In general, data records are accessed via the index area by specifying a key. A data record can be entered in several key directories.
The statement

- SEARCH #

is available for processing index areas.

11.4.3 Information about indexed files

The CHF function and SEARCH mode 1 enable different information to be fetched about the indexed file.

The CHF function enables the highest data record number and the current access position in the data area to be determined.

SEARCH mode 1

SEARCH mode 1 enables the lengths of keys in the directories, the number of the first data record and the number of free data records to be tested.

Statement	Function
SEARCH # C,1,V;V$,N,S	The key length of directory V is returned in N (V$ and S can be any value).
SEARCH # C,1,0;V$,N,S where S=0	The number of the first data record is returned in N (V$ can be any value).
SEARCH # C,1,0;V$,N,S where S=1	The number of free data records is returned in N (V$ can be any value).

11.4.4 Reading the data area sequentially

In order to read the data area sequentially, the number of the first data record must be determined and the first record read with it:

```
LET S=0
SEARCH # C,1,0;V$,N,S
IF S GOSUB ...
READ # C,N;...
```

(C = channel number, V$ can be any value)
All further data records can then be read with

```
READ # C,-1;...
```

11.4.5 Reading the data area in the order of the directory

SEARCH mode 3 makes this sequence possible. A key is specified which is less than any actual key:

```
LET V$=""
SEARCH # C,3,V;V$,N,S
IF S GOSUB ...
READ # C,N;...
```

(C = channel number, V = directory number, N and S can be any value)

After the record has been processed, a branch can be made back to the SEARCH statement without further preparations since the next higher key is supplied in V$.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.4.6 Random access via a key

SEARCH mode 2 enables a record to be read after a key has been specified.
The key must be specified in V$.

```
LET V$="Key" or INPUT V$
SEARCH # C,2,V;V$,N,S
IF S GOSUB ...
READ # C,N;...
```

(V = directory number, N and S can be any value)

11.4.7 Inserting a data record with keys

A search must first be carried out for the first free data record:

```
LET S=2
SEARCH # C,1,0;V$,N,S
IF S GOSUB ...
```

(C = channel number, V$ and N can be any value)

The record number is returned in N and deleted from the list of free data records.
The data record can now be written:

```
WRITE # C,N;...
```

Finally, all keys referring to the data record must be entered in the directories by means of SEARCH mode 4:

```
SEARCH # C,4,V;V$,N,S
IF S GOSUB ...
```

(V = directory number, V$ = key, S can be any value)

11.4.8 Deleting a data record with keys

All keys referring to the data record must first be deleted from the directories by means of SEARCH mode 5:

```
SEARCH # C,5,V;V$,N,S
IF S GOSUB ...
```

(C = channel number, V = directory number, V$ = key, N = number of the record to be deleted, S can be any value)

The data record itself must subsequently be deallocated:

```
LET S=3
SEARCH # C,1,0;V$,N,S
IF S GOSUB ...
```

(V$ can be any value, N = number of the record to be deleted)

11.4.9 Deleting an indexed file

There are two ways of deleting an indexed file:

- Physical deletion of the file by means of KILL:

  CLOSE # C (if file is open)
  KILL <FileName> (= name of indexed file)

- Deletion of the directories and deallocation of all data records by means of SEARCH mode 6, retaining the file structure:

  ```
  SEARCH # C,6,0;V$,N,S
  IF S GOSUB ...
  ```

  (C = channel number, V$, N and S can be any value)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

11.4.10 Status messages

After any SEARCH # statement has been executed, the variable S must be checked as it contains status messages concerning the course of the SEARCH operation.
The following status messages can be stored in S after the various functions of SEARCH # have been executed:

S	Meaning
0	Function executed without errors.
1	Function not executed successfully.
2	End of directory reached.
3	No free data record available.
4	Not an indexed file.
5	Record number negative, mode > 7, S > 3, directory number too high, key field length too small or any other serious error. The file may have been destroyed.
6	Illegal sequence when specifying the directories (only mode 0).
7	File is not a contiguous file (only mode 0).
8	Directory initialization already carried out.
9	No directory specified.
10	Too many directories (max. 15 per file).
12	File too small to hold all directories.
15	File opened by another user.

11.4.11 Closing the file

After the file has been processed, the channel on which the file was opened must be closed by means of the statement

- CLOSE #

11.4.12 Problems when processing index files

Simultaneous processing of an indexed file can cause problems because

- indexed files are generally accessed with several statements

  and

- several time-slices are required for one access.

In order to prevent errors which could result in the file being destroyed, it is recommended that the following rules be observed:

- If a data record with keys is to be inserted into several directories, the data record must first be written. All keys pointing to the data record must then be entered in the directories.

- If a data record with keys is chained in several directories, all keys pointing to the data record must first be deleted from the directories. The data record itself can then be dealloca-ted.

- After a SEARCH # statement has been executed, the status (<NExpr2>) must always be tested.

- The key should also be in the data record. It can be compared with the key specified and can be used for recovery if the index area is lost.

The following examples indicate the sources of possible errors and the methods by which errors can be prevented.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

- Examples:

  - Program 1 reads the record with the key "AA" and updates the data record. Program 2 deletes the key "AA" and enters the associated data record in the list of free records.

Time slice	Program 1	Program 2
1	LET V$="AA" SEARCH # C,2,1;V$,N,S	
2		LET V$="AA" SEARCH # C,5,1;V$,N,S
3	IF S GOTO ..(Error) READ # C,N;R$	
4		IF S GOTO ..(Error) LET S=3 SEARCH # C,1,1;V$,N,S
5	Update record WRITE # C,N;R$	

If this sequence of statements were used, the file would be destroyed after program 1 had written back the data record. The record was entered in the list of free data records in time slice 4 by program 2. Free data records are chained in the first two or first four bytes of the free records. Due to the fact that program 1 wrote back the data record in time slice 5 without displacement, i.e. starting from byte 0, the chain of free records is destroyed.

This error can be avoided if a record lock is set by the SEARCH # statement (in this case in time slice 1) before a data record is updated:

```
SEARCH # C,2,1;V$,N,S:
```

- Program 1 reads the record with the key "AA" and updates the data record. Program 2 deletes the key "AA", updates the data record and inserts the key "BB" for it.

Time slice	Program 1	Program 2
1	LET V$="AA" SEARCH # C,2,1;V$,N,S	
2		LET V$="AA" SEARCH # C,5,1;V$,N,S IF S GOTO ..(Error) WRITE # C,N;R$
3	IF S GOTO ..(Error) READ # C,N;R$ Update record	
4		LET V$="BB" SEARCH # C,4,1;V$,N,S IF S GOTO ...(Error)
5	WRITE # C,N;R$	

Program 1 wanted to update the data record belonging to the key "AA". However, since program 2 had overwritten it with the data record belonging to the key "BB" before program 1 read the data record, program 1 does not receive the desired data and, in addition, updates the data record written by program 2 for the key "BB".

It is possible to prevent this error by storing the key in the associated data record and comparing it with the key searched for in the key directory after reading. Furthermore, a record lock should be set at the first SEARCH statement as in the example on the previous page.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.5 Text files

A text file comprises the file header and the appropriate number of data blocks required to store the actual quantity of data.

If the data occupies more than 128 disk blocks, 1 to 128 extension blocks are available in addition to the file header. A header extension block enables 256 data blocks to be managed.

A text file does not occupy a contiguous area on the disk. Extensions are made by means of a list of free disk blocks.

The contents of a text file consist of a single character string with a maximum length of

16777216 bytes.

In this case, 128 header extension blocks are created.

This character string is split up into pages and lines. Lines are separated by means of CR codes and pages are separated by means of FF codes. The end of the character string, also the end of the text file, is identified by a delimiter.

Text files are designed for sequential processing, but they also enable random access in the following manner:

Random access to specific lines or pages is not possible. It is only possible to access disk blocks directly by specifying the block number relative to the beginning of the file in a READ/WRITE statement as if it were a record number; specific access to data in this block requires specification of a byte address relative to the start of the addressed block.

File Processing

11.5.1 Creating text files

When text files are created with BUILD #, a "+" must be specified before the file details.
When text files are created, they initially comprise the file header block only.

The specifications:

- number of records
- record length

are inadmissible when creating text files.

● Example:

Replacing and simultaneously opening a text file named TEST on LU 2 on channel 2.

```
50 BUILD # 2,+"2/TEST!"
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

11.5.2 Accessing text files

Before the data records in a file can be accessed, the file must be opened by means of the statement

- OPEN #

on any free channel.

Access to data in text files is possible by means of the statements

- READ #
- WRITE #
- PRINT #

Only string literals and/or the contents of string variables can be written to text files (exception: PRINT #). When reading from text files, only string variables are permitted as destination variables.

Special points to be noted:

- When the statement READ # is used, the transfer is terminated when a 'CR' code, a delimiter or the dimensioned end in the destination string is reached.
- When the transfer is terminated because the dimensioned end of the destination string has been reached, the file pointer skips the character in the text file which follows the text that has been transferred even if this character is not a delimiter. If the skipped character is not a 'CR' code, the file pointer must be positioned back by means of a suitable READ # statement using CHF (200+<ChanNo>).
- Specification of <NExpr2> as -2 does not result in the record last processed being accessed, but in transfer continuing if a 'CR' code occurs.
- No numeric variables and no numeric expressions may be specified in the WRITE # statement.
- The function of the separators "," and ";" in the PRINT # statement must be noted.
- A delimiter, which is overwritten again when writing sequentially, is always stored behind the character last transferred by means of PRINT # and, in some cases, by WRITE # (see "Description of the Statements").

File Processing

The record pointer for text files comprises the current block number and the address of the byte following the character last transferred, relative to the start of the block.

- Sequential access

  Sequential access is carried out if the relative record number is not specified in the statement or if -1 or -2 is specified as the relative record number.

  Note:

  If a text file is accessed sequentially, the byte specification must be 0, if coded.

  • Example:

    Sequential reading and display of the lines read. The number of lines to be output is specified in the variable I.

```
:
150 FOR I1=0 TO I-1
160 READ # C;R$;                /* READ TEXT FILE
170 PRINT R$                    /* SCREEN OUTPUT
180 NEXT I1
```

- Accessing the record last processed

  The IOCS does not support this feature for text files. It can only be implemented by means of random access managed by the application program.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

- Random access

Due to the fact that the lines of a text file have different lengths, access to specific lines is not possible. However, in order to be able to access a particular byte in a text file, a record length of 512 bytes is assumed for direct access when <NExpr2> is specified as equal to or greater than 0. The relative record number in text files is therefore the number of the block (sector) relative to the beginning of the file.
It is possible to change the access mode at any time.

• Example:

Random reading from byte 0 in block 3 followed by sequential processing.

```
:
150 READ # C,3;R$       /* RANDOM READING
160 GOSUB 2000
170 READ # C;R$;        /* SEQUENTIAL READING
```

Note:

Due to the fact that each transfer to a text file is concluded with a delimiter (exception: writing of substrings with WRITE #) and that a delimiter also indicates the end of the text file, it is not particularly advisable to write to text files directly. If a delimiter is located in a text file, positioning must be directly to the byte behind the delimiter in order to address data beyond this delimiter.

File Processing

- Example:

```
:
9000 IF ERR 0 GOTO 9100  /*ERROR HANDLING
9010 READ # 3;R$         /*READ TEXT FILE
9020 IF R$="" GOSUB 9200 /*NULL STRING
9030 ....
9040 ....                /*PROCESSING
9050 ....
:
9100 IF SPC 8=52 PRINT "END OF FILE"
9110 CHAIN ""
:
9200 READ # 3,CHF103,CHF203+1;R$;  /*POSITION BEHIND
                                    DELIMITER
9210 IF R$="" GOTO 9200  /*NULL STRING
9220 RETURN
```

- Field processing

There are no fields in text files. It is possible to access specific bytes in a text file by specifying the number of a byte relative to the beginning of the addressed block.
The maximum value which can be specified is 511. This specification is only allowed for random access.

- Example:

Reading to variable R$ from a text file starting at an address specified in the variables X and Y.

```
:
150 READ # C,X,Y;R$;
```

The value of Y must not exceed 511.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

File Processing

- End-of-file handling

  The end of a text file can be established when:

  - a delimiter is read (the destination string is "" or less than the dimensioned length if <NExpr2> equals -2).
  - BASIC error 52 occurs. However, this is only the case if a block to which no characters have been written is addressed directly.

The CHF function enables the programmer to determine the current size of the file or the current access position within the file.

- Example:

  A text file used as a log file should not become larger than 50 blocks. A warning is issued when 40 blocks have been reached. If the 51st block has been created, a flag is set and the current program is terminated at the next interface so that the log file can be printed.

```
:
9000 PRINT # 3,CHF103,CHF203;M$; /*RANDOM WRITE
9010 IF CHF((103)>50 GOTO 9100  /*ABORT?
9020 IF CHF(103)<40 RETURN      /*WARNING?
9030 PRINT TAB(0,24);'LD';"PRINT LOG FILE"
9040 SIGNAL 3,20
9050 RETURN
9100 LET M=1                    /*FLAG FOR ABORT
9110 PRINT TAB(0,24);'LD';"LOG FILE BEING PRINTED";
9120 RETURN
```

- Closing the file

  After the file has been processed, the channel on which the file was opened must be closed by means of the statement

  - CLOSE #

11.6 Sequential files

For the 8870 system, sequential files are restricted to the following media:

- Magnetic tapes
- Magnetic tape cassettes
- Punched cards
- Paper tapes
- Printers

However, the structure and programming of sequential files on these media varies in certain cases so the programming is described in the description of the relevant peripheral.

As the name suggests, sequential files only permit sequential access, i.e. a file must always be processed from the start in its physical sequence.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

12 Printers

Five types of printer can be connected:

- Needle printer 150/210 cps
- Needle printer 100 cps
- Compact printer 100 cps
- Daisy-wheel printer 45 cps
- Line printer 300 or 600 lpm

There are certain rules and restrictions for the programmer, depending on the printer connected.

The main differences between the types of printer are as follows:

- Different number of print positions
- Needle printers can be equipped with two continuous-form feeders and a cut-form chute.
- The first and last print position for needle, compact, high-speed and daisy-wheel printers can be defined by program. The last print position must not exceed the physically last position on the printer.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

12.1 Accessing printer files

A logical Input/Output Control System (IOCS) forms the interface to the Business BASIC language for printers. The IOCS is responsible for managing the data. All data output to printer is organized in the form of files. The only possible form of organization is sequential. A printer file is processed via a data channel.

The same rules apply to output of data to a printer as for output to text files.
A line printer cannot be opened simultaneously by several users or by one user on several channels (this applies to all printers with just one feeder).
In the case of needle printers with dual continuous-form feeders and a cut-form chute, different units can be opened simultaneously by several users due to the fact that each form control is driven as a logical independent device.
The left-hand continuous-form feeder and the cut-form chute can only be opened by one user at a time for reasons of safety. For mechanical reasons, the cut-form chute must be operated on the left side of the printer.
At present it is only possible to operate two feeders in parallel. The following combinations are allowed: left + right continuous-form feeder or cut-form chute + right continous-form feeder.

The transfer of print data and the actual printing are asynchronous. The transferred data are collected until a release code (line feed) or the logical end of line is reached. The data are then sent to the printer.

With regard to printer data, synchronization between the central processing unit and the port is related to the individual instructions into which a statement is broken down. If several fields are output in succession on the same line, this results in an extreme drop in printing speed. For this reason output must be line by line.

Printers

12.1.1 Opening printer files

Before data can be output to a printer, the printer must be opened on any channel by means of the statement

- OPEN # <FileName>

The name of the printer channel program (driver) in question must be specified as <FileName>:

- $LPT = Needle printer, left continuous-form feeder
- $LPTR = Needle printer, right continuous-form feeder
- $LPT1 = Line printer
- $LPT1S = Second line printer
- $LPT2 = Centronics printer (for USA only)
- $LPT2S = Second Centronics printer (for USA only)
- $LPTS = Second needle printer, left continuous-form feeder
- $LPTRS = Second needle printer, right coninuous-form feeder
- $RPLx = Workstation printer x, left continuous-form feeder
- $RPRx = Workstation printer x, right continuous-form feeder
- $RPFx = Workstation printer x, cut-form chute
- $LCP = Compact printer
- $LCPS = Second compact printer
- $ORDP = Band printer
- $ORDPS = Second band printer
- $HDD = System compact printer
- $HDDS = Second system compact printer
- $HDDSAS = System compact printer via SAS
- $HDDSASS = Second system compact printer via SAS
- $ORDPSR = Band printer via PSR
- $ORDPSRS = Second band printer via PSR

All printers can be driven via the logical printer name $<Name>. $<Name> is allocated to a physical printer via the device assignment table, depending on the workstation.

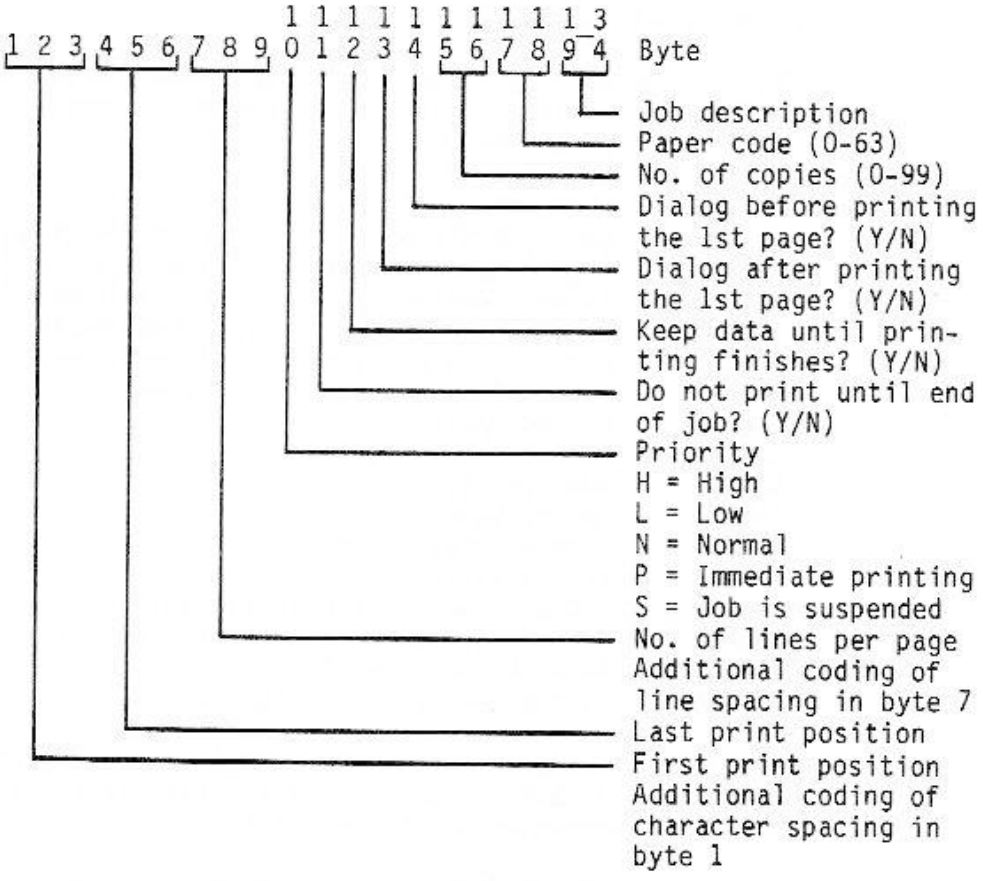
All printers except $RPF can be spooled. The spooler enables a device to be used by several users and guarantees simultaneous operation.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

In addition, parameters for form and spooling control can be specified in the OPEN # statement. These parameters are supplied to the printer channel program in a string literal or string variable which is dimensioned for a minimum of 34 bytes. This string ends with a delimiter.

The string has the following structure:



Specification of lower-case characters for the priority results in protection against deletion.

Printers

The print areas of needle printers with two continuous-form feeders may not overlap.

The following should be noted with respect to the number of lines per page:

- In the case of cut forms, 12.6 mm and 26.2 mm must be deducted from the actual form length for the form header and form footer, respectively.

- The line spacing for the cut-form chute is 5 or 6 lines/inch. The line spacing is only switch-controllable on the line printer (8 or 6 lines/inch).

- Different line spacing figures can be implemented on needle printers, depending on the type.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

If no specifications are made the following default values apply:

Channel program (driver)	First print position	Last print position	Page length
$LPT	000	176	048
$LPTR	177	177	048
$LPT1	000	135	048
$LPT1S	000	135	048
$LPTS	000	176	048
$LPTRS	177	177	048
$RPLx	000	131	048
$RPRx	145	177	048
$RPFx	000	131	062
$LCP	000	131	048
$LCPS	000	131	048
$ORDP	000	135/*131	048
$ORDPS	000	135/*131	048
$HDD	000	131/*157	048
$HDDS	000	131/*157	048
$HDDSAS	000	131	048
$HDDSASS	000	131	048
$ORDPSR	000	131/*157	048
$ORDPSRS	000	131/*157	048

* Adjustable via a code table in the driver

Printers

Other parameters:

Priority	= N
Wait until end of job before printing?	= N
Keep data until printing finishes?	= N
Dialog after printing the first page?	= N
Dialog before printing the first page?	= N
No. of copies	= 00
Paper code	= 00

Job description:

If no job description is specified, the 3rd level selector name is used if OPEN comes from a BASIC program and the processor name (e.g. LIBR) if OPEN comes from a processor.
This specification comes from the common area; it may have been overwritten by an application program.

The parameters are set according to the following priorities for logical printer names:

First/last print position	Lines per page
1. Device assignment table 2. OPEN # statement 3. Default values	1. OPEN # statement 2. Device assignment table 3. Default values

Character spacing:

Value to be added to the contents of byte 1	Meaning
0	Default = 10 characters per inch (16.5 for the 80-position mini compact printer)
2	10 characters per inch
4	12 characters per inch
6	15 characters per inch
8	16.5 characters per inch

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

	Printers

Line spacing:

Value to be added to the contents of byte 7	Meaning
0	Default value = 6 lines per inch
2	3 lines per inch
4	4 lines per inch
6	6 lines per inch
8	8 lines per inch

12.1.2 Output to printer

Output to the printer is effected by means of the statements:

- PRINT #
- WRITE #

Only string literals or the contents of string variables may be output by means of WRITE #.

Special points to be observed are as follows:

- If the specifications <NExpr2> and <NExpr3> are coded, they are ignored.
- No numeric expressions may be specified in a WRITE # statement.
- The different functions of the separators ";" and "," must be noted in the PRINT # statement.
- The TAB(<NExpr>) function is permissible as an output operand in the PRINT # statement.
- If ; or , is not coded after the last output operand in a PRINT # statement, a line feed is carried out after the statement is executed.
- Device functions (<DevFct>) must be coded in octal notation in a WRITE # statement whereas mnemonics are also accepted in a PRINT # statement.

- Pseudo wait
  A WRITE statement just involving a delimiter causes the driver to wait until all print data have been processed. This should be issued before a CLOSE so that any errors that have occurred are reported before the printer is closed.

12.1.2.1 Forms control

Forms are controlled with the aid of form control characters which must be specified as octal code or in symbolic form in the string which is to be output:

Octal code	Symbol	Function
210	'EOL'	End of line without line feed
211	'LFB'	Line feed backwards
212	'OPN'	Open cover for cut-form feeder
213	'IN'	Draw in form to 1st print line
214	'FF'	Eject form or feed in new sheet
215	'CR'	Carriage return with line feed

The functions 'LFB' and 'IN' only apply to the printer $RPFx (ND 02) and the others apply to all printers.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

12.1.2.2 Cut-form processing

The cut-form feeder (1305) which can be attached to the letter-quality printer (TD 01) enables cut forms to be processed under program control.

The following must be noted:

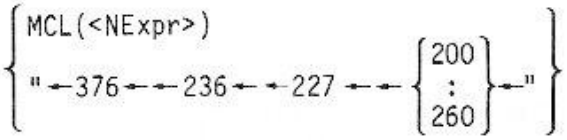
- Continuous and cut forms cannot be processed at the same time.
- Continuous forms must be removed from the tractor when cut forms are processed.
- The paper magazines must not be refilled during printing.
- Print spooling does not support cut-form processing.

There follows a description of printer functions which can be used with the DAP 4 workstation and the $RPL, $RPR and $RPF printer drivers.

Printers

12.1.2.3 Form feed

Syntax:



<NExpr> : Number of feeds of 1/24 inch each (1-48)

Execution of a draw-in instruction causes the specified number of microlines to be fed.
If the printer allows it, microline feeds can also be performed on continuous forms.
Specification of 00 or ←200← causes synchronization to the next full line.

- Example:

```
PRINT # X;MCL(24);

or:

PRINT # X;"←376←←236←←227←←230←";
```

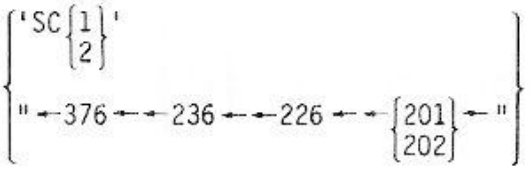
Effect: form feed of 1 inch

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

12.1.2.4 Paper cassette selection

Syntax:



A cassette selection remains in effect until the next switch-over or the next CLOSE. The forms are fed from cassette 1 as standard.

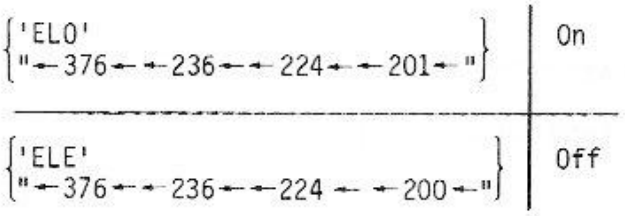
- Example:

```
100 PRINT # X;'SC2';
```

When the next instruction to draw in paper occurs, the form is fed from cassette 2.

12.1.2.5 Elongated printing

Syntax:



This function causes elongated printing on needle printers (the actual characters are elongated) and spaced printing on daisy-wheel and line printers (a space is inserted between each character).

Elongated printing is activated with 'ELO' and is deactivated with 'ELE'.

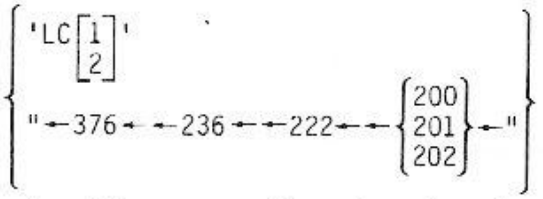
- Example:

```
100 PRINT # X;'ELO';B$;
 :
200 PRINT # X;'ELE';
```

Printers

12.1.2.6 Character density

Syntax:



This function causes the character density to be changed over.

	200	10 cpi
1	201	12 cpi
2	202	15 cpi 16.5 cpi on the mini compact printer

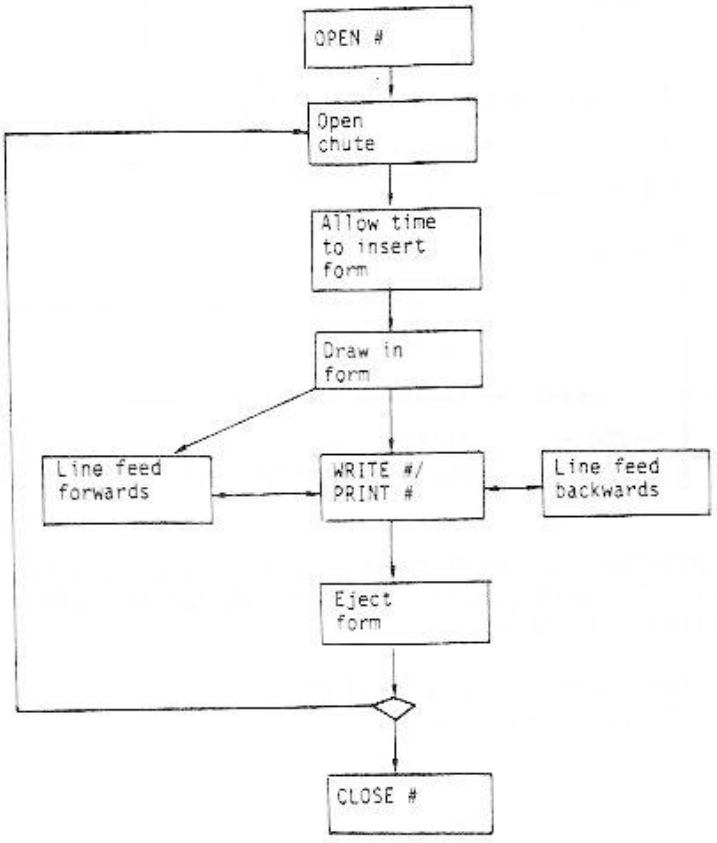
A printer-dependent default value is used after an OPEN statement without parameters if there is no related entry in the device assignment table:

- Mini compact printer 16.5 cpi
- All other printers 10 cpi

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

When printing cut forms by means of the form chute, the processing sequence illustrated in the diagram below must be observed:



In the event of an error, CLOSE # should only be specified after an eject (←214←).

For technical reasons, the line feed backward function cannot be carried out after one of the last lines has been processed.

Printers

A line feed is automatically performed if:

- PRINT # is executed without a closing semi-colon or comma,
- More characters are output without a control code than a single print line can hold.

In general, text files are handled in the same way as printer files, i.e. if printouts are to be made and the printer is in use, the data to be output can be stored intermediately in a text file.

● Examples:

- Intermediate storage of a printer file:

```
1000 IF ERR 0 GOTO 9000     /*ERROR HANDLING
1010 OPEN # 3;P$,"$LPT"     /*OPENING PRINTER
:                           /*PROCESSING
1100 PRINT # 3;R$           /*PRINTOUT
1110 GOTO 1020
:
9000 IF ERR 0 GOTO 9100     /*ERROR DURING BUILD
9010 BUILD # 3,+D$          /*FILE NAME IN D$
9020 IF ERR 0 GOSUB ...     /*STANDARD ERROR HANDLING
9030 GOTO 1020              /*CONTINUE AFTER OPEN
:
9100 LET D$(LEN D$+1)="!"   /*REPLACE FILE
9110 GOTO 9010
```

- Line-by-line printing by means of WRITE #:

```
1000 DIM N$(15),2%,L,Z$(130)
1010 OPEN # 3;"005124072","$LPT"
1020 ...
:
1050 GOSUB 9000
1060 GOTO 1020
:
9000 LET Z$=" ",Z$
9010 IF LEN N$=0 GOTO 9030
9020 LET Z$(21,20+LEN N$)=N$
9030 LET Z$(61,69)=L USING "##,###.##"
9040 LET Z$(72)="DM←215←"
9050 WRITE # 3;Z$;
9060 IF CHF 3>=61 WRITE # 3;"←214←";
9070 RETURN
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Printers

12.1.3 CHF function

A record pointer (line counter) is kept by the IOCS for printer files as soon as a printer file has been opened. The CHF function enables the current print line to be determined. The number of the channel on which the printer is open must be specified as <NExpr>.

CHF(C+200) - where C means the channel number - can be used to establish whether printing is being spooled:

CHF (C+200)	Meaning
0	Printing is not being spooled
<>0	Printing is being spooled

- Example:

  Advance to the next page if the line number equals the specified page length minus 4. The page length is stored in the variable F.

```
:
150 PRINT # C;A$;                    /*PRINTOUT
160 IF CHF C=F-4 PRINT # C;'FF'
```

Note:

It must, however, be noted that the effect of the CHF function on text files is not identical to that on printer files.

12.1.4 Closing the printer

After printing has been completed, the channel on which the printer was opened must be closed again by means of the statement:

- CLOSE #

This is followed by a feed to top of form.

Punched Card Reader

13 Punched card reader

A punched card reader with a speed of 90 cards/minute can be connected to the system.

13.1 Accessing punched card files

A logical Input/Output Control System (IOCS) forms the interface to the Business BASIC language for the punched card reader. The IOCS is responsible for managing the data. All data read in via the punched card reader is organized in the form of files. Punched card files are processed via a data channel.

A punched card file comprises any number of data records of a uniform length. A data record generally corresponds to the number of columns of the punched cards used:

- 80 characters.

They can only be processed sequentially.

The card reader cannot be opened simultaneously by several users or by one user on several channels.
The standard punched card code is converted internally to ASCII by the IOCS.

13.1.1 Opening punched card reader files

Before data is read via the punched card reader, it must be opened on a data channel with the aid of the statement

- OPEN #

The number of columns (1-80) of the punched card to be processed can be specified as a parameter during opening.
The name of the card reader driver

- $CRD1

must be specified as the file name.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Punched Card Reader

13.1.2 Input via the punched card reader

Data is input via the punched card reader by means of the statement

- READ #

The destination variables specified must be strings. If several destination variables are specified, the number of cards which are read is equivalent to the variables which are coded. Transfer is terminated when the end of the destination variables has been reached.

Use of the CHF function on punched card reader files is not evaluated.

13.1.3 Closing the punched card reader

The channel on which the card reader is open is closed by means of the statement

- CLOSE #

13.2 Example

Transfer of a punched card file to a contiguous file on a disk. A /* card is presented as the end criterion for the card file.

```
:
9000 OPEN # 3,"$CRD1"        /*PUNCHED CARD FILE
9010 OPEN # 4,D$             /*CONTIGUOUS FILE
9020 READ # 3,R$             /*READ PUNCHED CARD
9030 IF R$="/*" GOTO 9100    /*END OF FILE?
9040 WRITE # 4;R$            /*WRITE DATA RECORD
9050 GOTO 9020               /*NEXT CARD
9100 CLOSE # 3               /*PUNCHED CARD FILE
9110 CLOSE # 4               /*CONTIGUOUS FILE
9120 CHAIN ""                /*CHAIN SCOPE
```

Paper Tape Reader/Punch

14 Paper tape reader/punch

Paper tapes can be employed for data interchange with other systems or for data backup.
The system permits connection of one paper tape reader and one tape punch for processing both 5- and 8-track tapes.

14.1 Input via the paper tape reader

A logical Input/Output Control System (IOCS) forms the interface to the Business BASIC language for the paper tape reader. The IOCS is responsible for managing the data. All data read in via the paper tape reader is organized in the form of files. A paper tape file is processed via a data channel.

Before data can be read in via the paper tape reader, it must be opened on any channel.

```
OPEN     # <NExpr> [;<String1>], <String2>
     [[, # <NExpr> [;<String1>]], <String2>]^n
```

<NExpr> : Number of the channel on which the paper tape reader is to be opened.

<String1> : If the code is to be converted, the code table must be supplied here. It is advisable to first store the code table in a string variable which is specified here.

In addition to the information for code conversion, this table contains control information for processing.
For the structure of code tables see the section called "Code Tables for Paper Tape Reader/Punch".

<String2> : The driver name

"$PTRA"

must be specified for the paper tape reader.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Paper Tape Reader/Punch

Reading data

Data is input sequentially via the paper tape reader by means of the statement.

READ #

Note:

Only strings may be specified as the destination variables. If a numeric variable is specified as the destination variable, the message

ERROR # 54 (incompatible field type)

is issued.

Only one destination variable should be specified for paper tape reader inputs.

Transfer is terminated when

- a delimiter occurs or
- the length of the string variable as dimensioned or specified by indices has been reached.

Record number and record displacement specifications in the READ # statement are ignored, and the CHF function does not provide definite information.

Closing the paper tape reader

The data channel on which the paper tape reader was opened is closed by means of the statement

- CLOSE #

Note:

The way in which RUN processes delimiters can result in the loss of the last character read into the string if no substrings are used.

Instead of READ # ; A$
use READ # ; A$(1,LENA$-1)

Paper Tape Reader/Punch

14.2 Output via the tape punch

As in the case of the paper tape reader, a logical Input/Output Control System (IOCS) forms the interface to the Business BASIC language for the tape punch. The IOCS is responsible for managing the data to be transferred. Data to be output via the tape punch is organized in the form of files. Processing is carried out via a data channel.

- Opening the tape punch

  Before data can be output via the tape punch, the tape punch must be opened on any free data channel:

  OPEN # <NExpr> [;<String1>], <String2>
  [[, # <NExpr> [;<String1>]], <String2>]^n

  <NExpr> : Number of the channel on which the tape punch is to be opened.

  <String1> : If the code is to be converted, the code table must be supplied here. It is advisable to first store the code table in a string variable which is specified here.

  In addition to the information for code conversion, this table contains control information for processing.
  For the structure of code tables see the section called "Code Tables for Paper Tape Reader/Punch".

  <String2> : The driver name

  "$PTPA"

  must be specified for the tape punch.

- Outputting data

  Data is output sequentially via the paper tape punch by means of the statements

  - PRINT # and
  - WRITE #

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Paper Tape Reader/Punch

It should be noted that only string variables and string literals may be specified in the WRITE # statement.
If a numeric variable or a numeric expression is output in a WRITE # statement, this results in

ERROR # 54 (incompatible field type).

The PRINT # statement automatically converts all data into alphanumeric data.

Only one string variable should generally be specified for output to the tape punch.

Record number and record displacement specifications have no effect; the CHF function does not provide definite values.

The transfer length is determined when

- a delimiter occurs in the source string or the
- end of a string variable or string literal is reached.

However, delimiter handling can be influenced by specifying control information in the code table (see the section called "Code Tables for Paper Tape Reader/Punch").

- Closing the paper tape punch

The data channel on which the paper tape punch was opened is closed with the aid of the statement

- CLOSE #

This causes the trailer to be punched in accordance with the code table. The standard procedure (without code table or when bit 1 of byte 1 in the code table equals 0) involves punching of 256 spaces (only sprocket holes are punched).

Paper Tape Reader/Punch

14.3 Code tables for paper tape reader/punch

If the code is to be converted, a string variable must be specified as <String1> in the OPEN # statement. The conversion table must be stored in this variable before the OPEN # statement is executed.

Paper tape tables always have the following layout:

- Bytes 1 to 4, control information
- Bytes 5 to 260, table

Layout of the input table (paper tape reader)

Byte	Meaning
1	= 0, 8-track paper tape = 1, 5-track paper tape
2	= 0, Read but do not evaluate leader = 1, Read and evaluate leader
3	Escape character for letters (for 5-track tapes only)
4	Escape character for digits (for 5-track tapes only)
5-260	Table

If a table is not specified in the OPEN # statement, bytes 1 and 2 are defaulted to 0 and the paper tape code is transferred directly.

In respect of 5-track paper tapes (telex code), the escape characters must be entered in bytes 3 and 4. In this case, the table is divided into groups of 64 bytes each:

- Bytes 5 - 68 = Letter group
- Bytes 69 - 132 = Digit group
- Bytes 133 - 260 = Free

If one of the escape characters is read, the channel program switches over to the appropriate group. The escape characters are not transferred to the variable.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Paper Tape Reader/Punch

Layout of the output table (paper tape punch)

Bytes	Meaning
1	= 0, 8-track tape = 1, 5-track tape
2	= 0, Punch end sentinel = 1, Do not punch end sentinel; with leader and trailer = 2, Punch end sentinel = 3, Do not punch end sentinel; without leader and trailer
3	Escape character for letters (for 5-track tapes only)
4	Escape character for digits (for 5-track tapes only)
5-260	Table

If a table is not specified in the OPEN # statement, bytes 1 and 2 are defaulted to 0 and ASCII code is punched directly.

In respect of 5-track tapes, a group identifier must be specified in each position in the table (bytes 5-260) in bits 7 and 8.

Layout of a table position for 5-track tapes:

Bit	Meaning
1-5	Code to be punched
6	Not allocated
7-8	= 00, not allowed = 01, character of the letter group = 10, character of the digit group = 11, character occurs in both groups

When changing from one group to another, the channel program outputs the appropriate escape character (bytes 3 and 4).

Paper Tape Reader/Punch

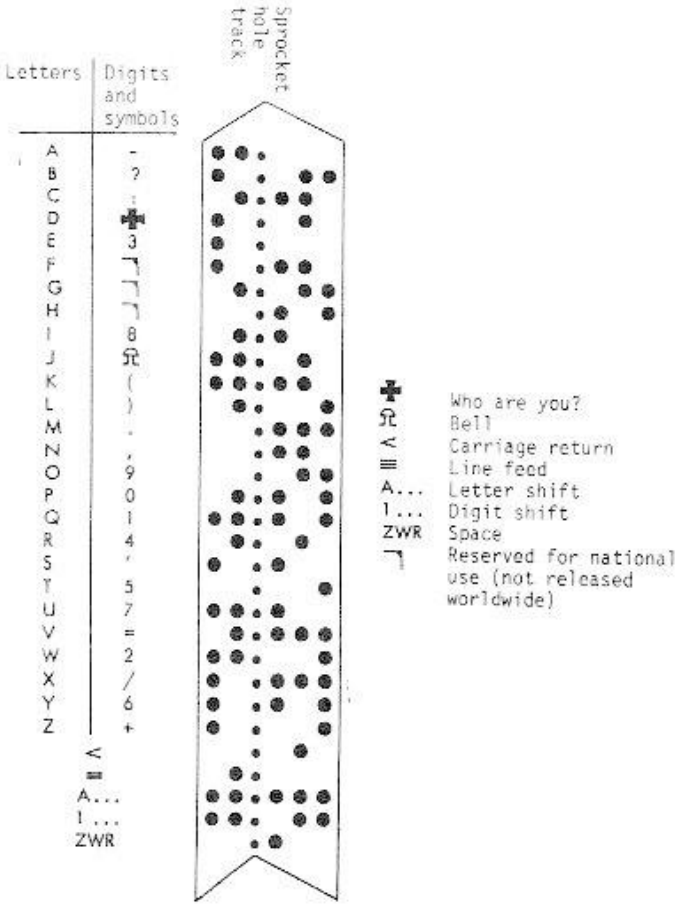
Standard tables

The following standard tables are available:

TT5.PUNCHTABLE: Code table for creating 5-track paper tapes in telex code.
Standard control information entries:
- 5-track identifier (in byte 1)
- Do not punch an end sentinel (in byte 2)
- Punch leader and trailer (in byte 2)

TT5.READTABLE: Code table for reading in 5-track paper tapes in telex code.
Standard control information entries:
- 5-track identifier (in byte 1)
- Read but no not evaluate leader (in byte 2)

Telex code:



© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15 Magnetic tapes

Magnetic tapes can be used for data interchange with other systems and for data backup.

SMT drives can only be connected by means of the new tape controller 1895.01 and support recording densities of 1600 bpi and 3200 bpi but not 800 bpi. The recording density of 3200 bpi is primarily used for data backup and recovery under TAMOS.

In addition to the CALL 70 interface, there is a new tape interface (peripheral driver interface) as from NIROS 5.1 supporting the statements OPEN #, ENV #, READ #, WRITE # and CLOSE #. Permissible driver names are $MT0, $MT1, $MT2 and $MT3. New application software should use this new peripheral driver interface. The recording density (1600/3200 bpi) is specified in the OPEN # statement.

Programs that are intended to run on operating system levels below NIROS 5.1 must continue to use CALL 70 as the tape interface.

15.1 Organization of data on magnetic tapes

All data stored on a magnetic tape is organized into files. The only possible form of organization is sequential. File management must be carried out completely by the application program.

The tape can be used in nearly any manner between the reflective beginning-of-tape and end-of-tape marks. However blocks are not allowed to be updated between the logical beginning and end of tape. From the physical point of view, this area is only divided up into data blocks with any length and interblock gaps. Gaps separate blocks and bridge small defective sections of tape. They are of no significance whatsoever for processing the information.

A block is a contiguous unit of information of any length. The size and content of this information is specified by the program which is processing it.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

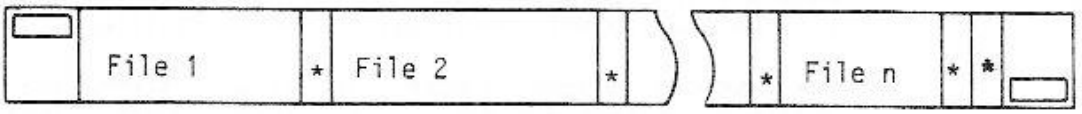
If magnetic tapes are to be interchanged with other systems, certain rules must be observed concerning the logical structure of the storage medium.
The storage capacity of a magnetic tape is divided up logically into one or more files. In turn, the files comprise any number of data blocks.

So-called tape marks are used to divide up the tape logically. These are 1-character blocks with any code at 800 bpi whereas the tape mark block is managed by the hardware from 1600 bpi on. Blocks of one character should therefore not be written by the application program.

Each tape ends logically with a double tape mark. The files, which are each separated by a tape mark, are located between the beginning of the tape and this double tape mark.
In the case of files with labels, the label area of the file is also separated from the data area by a tape mark.

Unlabelled tapes

The following tape structure results in the case of unlabelled tapes:



* Tape mark

Magnetic Tapes

Labelled tapes

A labelled magnetic tape must at least contain the following labels:

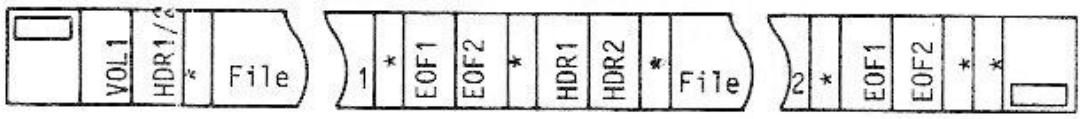
- VOL1 = Volume label
- HDR1 = File header 1
- EOF1 = End-of-file label 1, and, if necessary,
- EOV1 = End-of-volume label 1

The labels HDR2, EOF2/EOV2 can also be obligatory on higher label levels.
If the labels HDR2-9, EOF2-9 or EOV2-9 are used, they must be located immediately after the corresponding label 1. Optionally further user labels can follow.

- UVL1 to UVLn = User volume labels (n=2-9) (immediately follow VOL1)
- UHLa to UHLa = User header labels (immediately follow HDR1 or 2-9)
- UTLa to UTLa = User trailer labels (immediately follow EOF2 or 3-9 or EOV2 or 3-9)

a = any alphanumeric character

The following tape structure results:



Structure of the labels

Each label is a block of 80 characters. If the contents of the labels correspond to the DIN standard 66029, compatability to other systems is ensured.
The code of the data within a file need not be identical to the code of the labels.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

VOL1

This label is used to identify the tape.

Byte	Contents	Value
1-3	Label identifier	VOL
4	Label number	1
5-10	Volume identifier (archive identifier)	
11	Volume accessibility indicator	Ø
12-37	Reserved for future standardization	Ø
38-51	Owner identifier	
52-79	Reserved for future standardization	Ø
80	Standard version indicator	1/2/3

HDR1 / HDR2

The HDR1 and HDR2 labels are used to identify the file which follows.

Byte	Contents	Value
1-3	Label identifier	HDR
4	Label number	1
5-21	File identifier	
22-27	File set identifier	
28-31	File section number	
32-35	File sequence number	
36-39	Generation number	
40-41	Version number	00
42-47	Creation date	ØYYDDD
48-53	Expiration date	ØYYDDD
54	File accessibility indicator	Ø
55-60	Block counter	000000
61-73	System code (remainder Ø)	
74-80	Reserved for future standardization	Ø

Byte	Contents	Value
1-3	Label identifier	HDR
4	Label number	2
5	Record format	F/D/S
6-10	Block length	
11-15	Record length	
16-50	Ø	
51-52	Length of an additional block field inserted at the start of each data block.	
53-80	Reserved for future standardization	Ø

Magnetic Tapes

EOF1 / EOF2

These labels identify the end of a file.

Byte	Contents	Value
1-3	Label identifier	EOF
4	Label number	1
5-54	Corresponds to HDR1	
55-60	Block counter: number of data blocks since the last label group	
61-80	Corresponds to HDR1	

Byte	Contents	Value
1-3	Label identifier	EOF
4	Label number	2
5-80	Corresponds to HDR2	

EOV1 / EOV2

These labels identify the logical tape end for multi-volume files. They are identical to the labels EOF1 and EOF2 apart from the label identifier and the label number.

UVL / UHL / UTL

Optionally, the obligatory labels can be supplemented by the user labels. These labels also have a length of 80 characters; however, their contents are not subject to particular rules just like the contents of HDR3-9, EOF3-9 and EOV3-9, and can be freely allocated by the user. The only condition required is an admissible label identifier.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2 Accessing magnetic tape files

A magnetic tape unit can only be opened by one user at any time.

15.2.1 Opening the magnetic tape unit with the OPEN # statement

The OPEN statement assigns the tape unit to the calling port and locks it for other ports.

Syntax: OPEN #<NExpr1> [;<NExpr2>],<SLit>

<NExpr1> : Number of the channel on which the tape unit is to be opened.

<NExpr2> : Recording density setting. If the numeric expression contains the value 4, a recording density of 3200 bpi is set. If <NExpr2> is not specified, 1600 bpi is used. (Note: the setting comes into effect when the tape is at BOT.)

<SLit> : Driver name. Permissible driver names are $MT0, $MT1, $MT2 and $MT3.

Magnetic Tapes

15.2.2 READ # statement

Data is read from tape by means of this statement.

Syntax: READ # <NExpr>;<SVar>

<NExpr> : Number of the channel on which the tape unit was opened.

<SVar> : String variable to hold the received data.

The length of the string variable, as defined by dimensioning or indexing, specifies the number of characters to be read. Exception: If the length of an indexed string variable is less than dimensioned, one character less than the number specified by indexing is transferred.

Any variance between the required and actual number of characters read can be tested by means of the ENV # statement (function 1).

Note:
The data that has been read is not available until the status has been tested.

15.2.3 WRITE # statement

Data is written to tape by means of this statement.

Syntax: WRITE # <NExpr>;<SVar>

<NExpr> : Number of the channel on which the tape unit was opened.

<SVar> : String variable with the data to be sent.

The length of the string variable, as defined by dimensioning or indexing, specifies the number of characters to be written. Exception: If the length of an indexed string variable is less than dimensioned, one character less than the number specified by indexing is transferred.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2.4 ENV # statement

The ENV # statement provides information needed to operate a data communication line and ascertains the status of the line after execution of an OPEN #, WRITE # and READ # statement.

15.2.4.1 Transferring code tables

Code tables can be transferred to the driver by means of the ENV # statement, function number 0.

Syntax: ENV #<NExpr>;<NVar> ,<SVar1>,<SVar2>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar> : 0 (function number for code conversion)

<SVar1> : Code table for reading from tape (256 bytes)

<SVar2> : Code table for writing to tape (256 bytes)

15.2.4.2 Testing the status

After each OPEN #, READ # and WRITE # statement, the status must be tested by means of the ENV # statement, function 1.

Syntax: ENV #<NExpr>;<NVar1>,<NVar2>,<NVar3>,<NVar4>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Error code with the following meaning:

0 Operation successful
1 Not ready
2 Write-protected
3 Max. gap length exceeded
4 Operation not completed (premature FM, LDP, EOT, 2 FMs detected; see <NVar3>)
5 Permanent tape error
6 Controller defective
7 Time-out, parity error, lost data
8 Power failure
9 Tape recorded at different density

<NVar3> : Tape unit status

0 No status
1 EOT (end of tape)
2 LDP (beginning of tape)
3 RWD (rewinding)
4 FM (tape mark found)
5 FM and EOT (tape mark and end of tape found)

<NVar4> : >= 2% variable. After execution of the statement, variance between the required and the actual number of characters, blocks or tape marks (variance = actual - required). Positive variances are only detected by the PSR controller.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2.4.3 Rewinding

The tape is rewound to BOT by means of the ENV # statement, function 2.

Syntax: ENV # <NExpr>;<NVar1>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 2

15.2.4.4 Rewinding and switching off

The tape is rewound and the tape unit is switched to offline by means of the ENV # statement, function 3.

Syntax: ENV # <NExpr>;<NVar1>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 3

15.2.4.5 Writing a tape mark

A tape mark is written by means of the ENV # statement, function 4.

Syntax: ENV # <NExpr>;<NVar1>

<NEXpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 4

15.2.4.6 Forward-spacing by n blocks

The tape can be foward-spaced by a specified number of blocks by means of the ENV # statement, function 5. The actual number of blocks by which the tape was forward-spaced must be ascertained by a status test.

Syntax: ENV # <NExpr>;<NVar1>,<NVar2>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 5

<NVar2> : Number of blocks by which the tape is to be forward-spaced.

15.2.4.7 Backspacing by n blocks

The tape can be backspaced by a specified number of blocks by means of the ENV # statement, function 6. The actual number of blocks by which the tape was backspaced must be ascertained by a status test.

Syntax: ENV # <NExpr>;<NVar1>,<NVar2>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 6

<NVar2> : Number of blocks by which the tape is to be backspaced.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2.4.8 Forward-spacing by n tape marks

The tape can be forward-spaced by a specified number of tape marks by means of the ENV # statement, function 7. The actual number of tape marks by which the tape was forward-spaced can be ascertained by a status test.

Syntax: ENV # <NExpr>;<NVar1>,<NVar2>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 7

<NVar2> : Number of tape marks by which the tape is to be forward-spaced.

15.2.4.9 Backspacing by n tape marks

The tape can be backspaced by a specified number of tape marks by means of the ENV # statement, function 8. The actual number of tape marks by which the tape was backspaced can be ascertained by a status test.

Syntax: ENV # <NExpr>;<NVar1>,<NVar2>

<NExpr> : Number of the channel on which the tape unit was opened.

<NVar1> : Function code: 8

<NVar2> : Number of tape marks by which the tape is to be backspaced.

15.2.5 CLOSE # statement

This statement closes the channel which was OPENed.

Syntax: CLOSE # <NExpr>

<NExpr> : Number of the channel on which the tape unit was opened.

15.2.6 Opening the magnetic tape unit with CALL 70

The statement

CALL 70

is still available for programming the magnetic tape on release levels before NIROS 5.1. CALL 70 forms the interface between the BASIC interpreter RUN and the channel program $MTX.

The functions for opening the magnetic tape unit include rewinding to BOT, if necessary, and they determine whether or not the code is to be converted.

Syntax:

CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>
[,<SVar1>, <SVar2>]

<NExpr> : 70

<NumPar1> : 0 to 3 (number of the tape unit)

<NumPar2> : Function variable:
11: Opening the tape unit
12: Same as 11 plus rewinding to BOT

<NVar3> : Status variable

<SVar1> : Input code table (256 bytes)

<SVar2> : Output code table (256 bytes)

If the code is to be converted, <SVar1> and <SVar2> must be coded and loaded prior to opening.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2.7 Processing

15.2.7.1 Managing the storage medium

These functions enable the tape to be rewound fast to BOT (beginning of tape). They also enable a tape mark or a gap to be written.
It is necessary to write a gap in the case of permanent tape errors. As from NIROS 5.1, however, the channel program or PSR controller handles errors. The instruction "write gap" then has no effect and is no longer offered by the controller.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>
```

<NExpr> : 70

<NumPar1> : 0 to 3 (tape unit)

<NumPar2> : Function variable:
1: Rewind tape to BOT
6: Write tape mark
7: Write gap (2.5 inch tape gap)
10: Write tape mark and rewind tape to BOT

<NVar3> : Status variable

Magnetic Tapes

15.2.7.2 Forward-spacing and backspacing the tape

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>,
                         <NumPar4>, <NVar5>
```

<NExpr> : 70

<NumPar1> : 0 to 3 (tape unit)

<NVar2> : Function variable:
3: Forward-space tape by <NumPar4> blocks
4: Backspace tape by <NumPar4> blocks
8: Forward-space tape by <NumPar4> tape marks
9: Backspace tape by <NumPar4> tape marks

<NVar3> : Status variable

<NVar5> : Actual number of blocks or tape marks processed

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.2.7.3 Reading/writing a block

Reading or writing a data block, block length: greater than or equal to 2 and less than or equal to the tape buffer of the system. The actual number of characters processed, which is stored in <NVar5>, should always be checked.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>,
                                <NumPar4>, <NVar5>, <SVar>
```

<NExpr> : 70

<NumPar1> : 0 to 3 (tape unit)

<NumPar2> : Function variable:
0: Read block. Transfer is terminated when the dimensioned length of <SVar> or the number of characters specified in <NumPar4> is reached.
5: Write block. End of transfer is the same as when reading the block.

<NVar3> : Status variable

<NumPar4> : Number of characters to be processed

<NVar5> : Actual number of characters processed

<SVar> : String variable to hold the block to be written or which has been read.

Magnetic Tapes

15.2.8 Closing the tape unit

The functions for closing the tape unit also permit the tape unit to be switched to the off-line mode.

Syntax:

CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>

<NExpr> : 70

<NumPar1> : 0 to 3 (tape unit)

<NumPar2> : Function variable:
2: Switch tape unit to off-line
13: Close tape unit
14: Close tape unit and switch to off-line

<NVar3> : Status variable

15.2.9 Ascertaining the size of the tape buffer

Syntax:

CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>, <NVar4>

<NExpr> : 70

<NumPar1> : 0 to 3 (tape unit)

<NumPar2> : 15 (function variable)

<NVar3> : Status variable

<NVar4> : Size of tape buffer after execution of the function

Since the new tape drivers (as from NIROS 5.1) access the user partition directly, no tape buffer is configured under this release. Function 15 (ascertaining the size of the tape buffer) is therefore no longer needed under NIROS 5.1.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tapes

15.3 Error handling

15.3.1 Error handling by the tape interface

BASIC error 73 is output if the call is incorrect (wrong configuration of tape software, wrong function number with ENV #, no status test after incorrect command).

Otherwise attention must be paid to the error codes returned by the ENV # statement (function 1 - status test).

15.3.2 Error status codes when using CALL 70

A status code is returned in the status variable after each tape operation involving CALL 70. Precise action should always be taken to each of these status codes.

The following list provides an overview of the possible status codes, their meaning and further hints:

Status	Meaning	Explanation and action
0	Operation was successful	-
1	Tape unit not available	Tape unit opened by other user. Try again later.
2	Tape unit not ready	Activate the tape unit ("on-line").
3	Tape is write-protected	No write-enable ring; insert and repeat operation.
4	Read/write error	Repeat process, if necessary write gap.
5	EOF read	Terminate reading.
6	EOT marker reached	When reading: terminate reading. When writing: check whether the last block was written completely. Close file.
7	Illegal operation	Check the operation causing the error (<NumPar1> > 3? Unit not opened?).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tape Cassettes

16

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

16 Magnetic tape cassettes

It is possible to connect a magnetic tape cassette unit to the 8870 system for system maintenance via cassette, data backup and for data interchange with other systems.

16.1 Organization of data on tape cassettes

The data stored on tape cassettes is organized in the form of files. The only possible form of organization is sequential. The application program must manage the files and any existing labels.

The cassette tape can be used in almost any manner between the beginning-of-tape mark (BOT) and the end-of-tape mark (EOT). However blocks are not allowed to be updated between the logical beginning and end of tape! From a physical point of view, the only distinction is between data blocks, which can have a length of up to 512 bytes, and interblock gaps. A gap is used to separate two blocks and is of no significance for processing the information.

A block is a contiguous unit of information and the size and contents of this information are specified by the program processing it.

If cassettes are to be interchanged with other systems, certain rules with respect to the logical storage media structure must be adhered to.
The data capacity of a cassette side is logically organized into one or more files. In turn, these files comprise any number of data blocks.

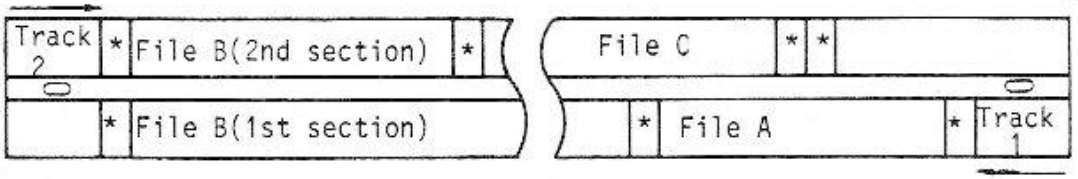
Tape marks are used to divide the tape up logically. These are 1-character blocks with the octal code 377.

Each tape ends with a double tape mark. The files, which are each separated by a tape mark, are located between the beginning of a tape and this double tape mark.
If files have labels, the label area is also separated from the data area by a tape mark.

Magnetic Tape Cassettes

Unlabelled tapes

The following tape structure results in the case of unlabelled tapes:



* Tape mark

Labelled tapes

Two standardized levels exist for label layouts on cassettes:

1. The compact level; the labels have a length of 32 characters and can be created by means of a numeric keypad.

2. The extended level whose labels have a length of 80 characters.

A detailed description of the structure and layout of the headers is contained in the DIN 66229 and DIN 66029 standards.

Magnetic Tape Cassettes

16.2 Accessing cassette files

The cassette unit can only be opened by one user at any one time. The statement

- CALL 80

is available for programming the cassette unit. CALL 80 forms the interface between the BASIC interpreter RUN and the channel program $CAS.

16.2.1 Opening the magnetic tape cassette unit

When opened, the magnetic tape cassette unit is locked against access by other users. The cover of the cassette unit is opened and the cassette channel program waits 105 seconds for the cassette to be inserted and the cover closed. After the cassette has been inserted, it is rewound to BOT.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>
                                    [,<SVar1>, <SVar2>]
```

<NExpr> : 80

<NumPar1> : 0 (MTC unit)

<NumPar2> : Function variable:
11: Open MTC for processing

<NVar3> : Status variable

<SVar1> : Input code table (256 bytes)

<SVar2> : Output code table (256 bytes)

If the code is to be converted, <SVar1> and <SVar2> must be coded and loaded prior to opening.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tape Cassettes

16.2.2 Processing

16.2.2.1 Managing the storage medium

These functions enable the cassette tape to be rewound and the cassette to be ejected.

Syntax:

CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>

<NExpr> : 80

<NumPar1> : 0 (MTC unit)

<NumPar2> : Function variable:
1: Rewind cassette tape to BOT
2: Open cover and eject cassette

<NVar3> : Status variable

16.2.2.2 Forward-spacing and backspacing the cassette tape

The tape is positioned block by block, i.e. from gap to gap.

Syntax:

CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>

<NExpr> : 80

<NumPar1> : 0 (MTC unit)

<NumPar2> : Function variable:
3: Forward-space the tape by one block
4: Backspace the tape by one block

<NVar3> : Status variable

16.2.2.3 Reading/writing a block

Transferring a data block with a length of at least 1 byte and at most 512 bytes. Transfer of a data block is terminated when the dimensioned length of <SVar> or the number of bytes specified in <NVar4> has been reached. The actual number of characters transferred are stored in <NVar5> and has to be checked.
After transfer has been completed, the tape is positioned to the gap after the block which has been transferred.

This procedure is repeated up to 8 times in the event of read/write errors.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>, <NVar4>,
                                        <NVar5>, <SVar>
```

<NExpr> : 80

<NumPar1> : 0 (MTC unit)

<NumPar2> : Function variable:
0: Read block and transfer to <SVar>
1: Write the contents of <SVar> as a block

<NVar3> : Status variable

16.2.3 Closing the magnetic tape cassette unit

This function terminates processing, the tape is rewound and the cassette ejected.

Syntax:

```
CALL <NExpr>, <NumPar1>, <NumPar2>, <NVar3>
```

<NExpr> : 80

<NumPar1> : 0 (MTC unit)

<NumPar2> : Function variable: 13 (close)

<NVar3> : Status variable

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Magnetic Tape Cassettes

16.3 Error handling

A status code is returned in the status variable after each cassette operation. Precise action should always be taken to each of these status codes.

The following list provides an overview of the possible status codes, their meaning and further hints:

Status	Meaning	Explanation and action
0	Operation was successful	-
1	MTC unit not available	MTC unit opened by another user. Try again later.
2	MTC unit not ready	Activate MTC unit (close cover).
3	Cassette is write-protected	Tab on the rear has been broken off.
4	Read/write error	Repeat process, if necessary change cassette.
5	Time-out	When opening: insert cassette and close cover. Otherwise: change cassette.
6	EOT (end-of-tape mark reached)	When reading: terminate reading When writing: check whether the last block has been written completely with the aid of <NVar4>. Close file
7	Illegal operation	Check the operation causing the error (<NumPar1> <> 0? Unit not opened?)

Program Structure

17 Program structure

17.1 Executing a program

A BASIC program which is to be executed must be loaded into a user area of main memory (partition).

There are two ways of doing this:

- The program was previously entered or modified and is started by means of the BASIC command RUN.
- A saved program is called from the EXPERT selector by entering the file name, loaded from the program file into the partition and started.

A BASIC program loaded into the partition for execution has the following structure:

Stacks = Work area for the BASIC interpreter RUN for: - List of user-defined functions - Subroutine stack - FOR-NEXT stack - List of variables
Table of line numbers (in ascending order) with a pointer to the address of the associated statement.
Statements in compressed form (intermediate code).
Variable data area. The area is not created until the run time of the program when the DIM statements are executed and in the event of implicit declaration.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

A program is executed via the table of line numbers. This table is processed sequentially. The statements are accessed via the pointers stored in the table of line numbers.
Sequential processing of this table is interrupted by the statements:

- GOTO
- GOSUB
- RETURN
- CHAIN
- LINK
- IF
- ON
- END
- STOP
- NEXT

and, in certain circumstances, by FOR.

In the event of branch statements, the line number of the next statement to be executed is ascertained in the table of line numbers in accordance with the method of binary searching. The line number is either in the statement (GOTO, GOSUB) or is fetched from the appropriate stacks (FOR-NEXT stack, subroutine stack).

The statements STOP and END stop execution of the program or terminate program execution.

Whereas the GOTO statement causes sequential execution of the statements to be continued starting from the line number specified, the slightly more complex procedures in the case of FOR-NEXT, GOSUB-RETURN, CALL, CHAIN and LINK statements are described in the following sections.

Program Structure

17

17.2 FOR-NEXT loops

The FOR statement identifies the start and the NEXT statement the end of a program loop.

The following is specified in a FOR statement:

1. The control variable
2. The initial value of the control variable
3. The final value
4. The step (optional, default: 1)

Only the control variable is specified again in the NEXT statement to identify the appropriate FOR statement to which it belongs.

When, on program execution, the RUN processor encounters the FOR statement, the control variable is allocated the initial value and tested as to whether the final value has already been exceeded in a positive or negative sense (the latter if the defined step is negative). If this is not the case, processing of the statements continues. When the associated NEXT statement is reached, the step is added to the current value of the control variable and this is tested as to whether the final value has been exceeded in a positive or negative sense. If this is the case, execution is continued at the statement following the NEXT statement.
If this is not the case, the program loop is executed once again with the modified value of the control variable.

• Example:

```
40 FOR I=1 TO 3
50 PRINT "VALUE:";I
60 NEXT I
70 END
```

Procedure:

```
1. I=1; I>3? No, continue            (Statement 40)
2. Display VALUE: 1                  (Statement 50)
3. I=I+1; I>3? No, branch to 50      (Statement 60)
4. Display VALUE: 2                  (Statement 50)
5. I=I+1; I>3? No, branch to 50      (Statement 60)
6. Display VALUE: 3                  (Statement 50)
7. I=I+1; I>3? Yes, branch to 70     (Statement 60)
8. End of program                    (Statement 70)
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

The value of the control variable must not be modified in the loop without being controlled as this would also modify the number of times which the loop is executed.

The initial and final value of the control variable may have a maximum of 6 significant digits as these are stored in 2% format.

Program loops can be nested in one another (up to 8 loops), but they must not overlap one another.

• Example:

```
  :
┌─40 FOR I=1 TO 10
│ :
│ ┌─65 FOR J=1 TO 10
│ │ :
│ └─80 NEXT J
│ :
└─95 NEXT I
  :
  permissible
```

```
  :
┌─40 FOR I=1 TO 10
│ :
│ ┌─65 FOR J=1 TO 10
└─┼─:
  └─80 NEXT I
    :
    95 NEXT J
    :
    not permissible
```

17.3 Subroutine techniques

If the same series of statements is to be executed at different points in the program, the subroutine technique should be used.

The branch to the subroutine is effected by the statement

GOSUB <LNo>

where <LNo> is the line number of the first statement in the subroutine. The line number of the statement following the GOSUB statement is entered simultaneously in the subroutine stack. The GOSUB statement should therefore not be at the end of a program.

Program Structure

The subroutine is then processed sequentially until a RETURN statement occurs. This causes a branch to be made to the line number stored in the subroutine stack.

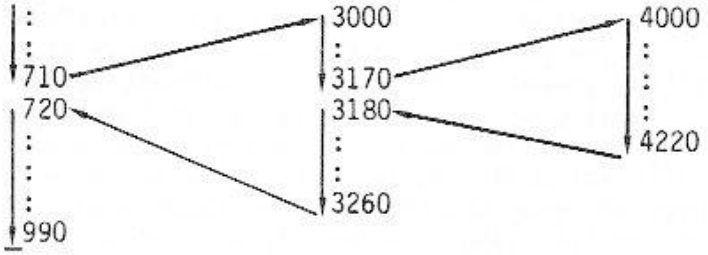
The return address can be modified by specifying a <NExpr> after the keyword RETURN (see RETURN statement).

Further subroutines can be called within a subroutine. The nesting depth of the subroutines must not exceed 32 levels.

- Example:

```
:
710 GOSUB 3000
720 PRINT ...                    Main program
:
990 END or CHAIN ...
3000 ...
:
3170 GOSUB 4000
3180 LET ...                     Subroutine 1
:
3260 RETURN
4000 ...
:                                Subroutine 2
:
4220 RETURN
```

Procedure:



Main program Subroutine 1 Subroutine 2

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

17.4 Subroutines in machine code

The CALL statement is used to fetch library subroutines in machine code.

Subroutines of this type enable functions to be executed which are not available within Business BASIC or which could only be solved by means of time-consuming BASIC routines.

A description of all available subroutines can be found in the chapter "CALL Statements".

17.5 Segmentation of programs

It is possible for a program to start any BASIC programs or processors by means of the CHAIN statement. BASIC programs can also be started with the LINK statement. In addition, the LINK statement involves the transfer of variables to the next program (segment).

Each of these statements enable programs which, in their entirety, would exceed the available memory area to be split up into segments and be executed in succession.

A further feature is that progam suites can be defined. Certain routines can be loaded without the operator having to intervene, depending on the status of the program.

When a CHAIN or LINK statement is executed, the BASIC program specified is loaded from the disk into the user's partition. In the case of the CHAIN statement, the data area is completely overwritten, whereas with the LINK statement variables are passed from the calling program to the successor (memory location, name, dimension, format and value are retained). In programs which do not contain any GLOBAL statements, all dimensioned variables which have not been DEALLOcated are transferred. In programs which do contain GLOBAL statements, all variables declared in these statements are transferred to the successor provided that they have not been excluded from transfer in the LINK statement.

17.6 Programming to optimize run-time

17.6.1 BASIC optimizer

It is always desirable to have programs run as fast as possible. This chapter describes how to write or convert BASIC programs in such a way as to optimize run time.

Explanation:

Optimization involves the generation of an intermediate code which can be interpreted lineally and which contains variable addresses.

This is done by means of the BASIC Optimizer which is called explicitly in the SAVE command by means of the parameter ",D,". The more closely the rules described here are observed, the better the intermediate code and consequently run time.

3 levels of intermediate code can be generated:

- B file SAVE...
- D file SAVE,D,...
- D file with variable addresses SAVE,D,...

Variable addresses are generated when the variables are declared at the beginning of the program as described in the following sections.

There is a special interpreter for D files. The old or new interpreter is called, depending on the file involved.

17.6.2 Arrangement of programs

Unsegmented program

If a program (program code and variable area) is smaller than the maximum available partition size, it is not necessary to segment the program. All variables are local and are declared with DIM statements at the beginning of the program.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

Segmented programs

If an application cannot be implemented in one program for reasons of space, it can be split up into segments to form a program sequence. Segments can be called via CHAIN statements. Since the variable area is erased when the next segment is loaded in this case, variable values can only be passed on via files or the common area. In such segments, the variables are also local and have to be declared at the beginning of each segment by means of DIM statements. Each segment uses the entire data area at run time.

Global and local data area

As the transfer of variables by means of files is detrimental to run time, the variable area is often split up into a global and a local data area. Such splitting should be defined at the beginning of the program development process.

Variables which are referenced in several segments (global variables) must be declared at the beginning of each segment by means of the new GLOBAL statement. The only statement which can precede a GLOBAL statement is REM. It is also important to ensure that the global variables are specified in the same order and size in each segment in the sequence. However it is permissible to increase or reduce the number of global variables from one segment to the next. Additional globals must, however, be declared in a new GLOBAL statement.

The other variables only apply to the respective segment (local variables) and should be declared straight after the GLOBAL statements in the form of DIM statements.

The segments are LINKed in this case. In contrast to CHAIN statements, LINK ensures that the global data area is not erased when the next segment is loaded. The LINK statement provides a facility for reducing the global variable area which is to be passed on to the next segment (see examples later).

IF ESC and IF ERR 0 statements

The statements IF ESC and IF ERR 0 should be located straight after the global and local variables, using constants as the index. This means that errors which can occur with the GLOBAL and DIM statements must be rectified at testing time. The program cannot be interrupted by means of the ESCAPE key when the GLOBAL and DIM statements are being executed.

17.6.3 Converting old programs

The ANALYSE command can be used to help convert programs. Critical situations are marked in the program listing (see the chapters "BASIC Commands" and "Program Maintenance").

Global variables

- Find all global variables which are to be passed between segments in a program sequence. Variables which are not contained in a DEALLO statement before the LINK statement are global.

- Place the global variables at the beginning of the segments in GLOBAL statements. Make sure that the global variables are declared in the same order and dimension. If more global variables are needed in the next segment, the new ones must be declared in a new GLOBAL statement. GLOBAL statements are not executed at run time in D files; only the variable area is initialized accordingly. They are only executed in B files if the program was not called with LINK.

- Delete the DEALLO statement before the LINK statement.

- Place IF ESC and IF ERR 0 statements behind the GLOBAL and DIM statements.

- GLOBAL statements with a variable index are not allowed (i.e. GLOBAL S$(7), not GLOBAL S$(I*J)). The maximum length must be used here. The program sequence cannot be converted if this is impossible.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

- Segments with the old DEALLO/LINK structure are not compatible with segments which have been converted and contain GLOBAL statements in a program sequence!

Local variables

- Find all explicitly and implicitly dimensioned local variables in the respective segments. The ANALYSE command's cross reference list is helpful in this respect.

- From these local variables, find those which are not indexed and those which have a constant as an index and place or re-enter the DIM statements straight after the GLOBAL statements in the respective segment. Make sure that the dimensioning % specifications match!

- Find all DIM statements and implicit dimensioning with a variable index.

- Find multiple dimensioning for the same variables.

- Find all DEALLO statements which are not directly before the LINK statement.

- For the 3 items mentioned above, check whether these cases can be avoided and convert as necessary! If this cannot be done, these DIM statements can remain in the middle of the program but no addresses are generated for the variables.

  Note: If dimensioning by means of a variable index occurs in a DIM statement, no addresses are generated for any of the variables in this statement. For this reason, dimensioning which involves variable indices should be written in a separate DIM statement.

- To improve run time, indexed variables are often cleared by means of the construction DEALLO "variables" and DIM "variables" instead of allocating "0" by means of a FOR/NEXT loop. A CLEAR statement should be used here (see examples).

Program Structure

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- Explanatory examples:

```
- 10 REM DATA AREA DESCRIPTION
  20 GLOBAL 1%,A,B,C,2%,D,E(8),F(6,9)   /* GLOBAL VAR.
  30 GLOBAL A$(12),B$(50)
  40 GLOBAL etc.
  50 DIM 1%,I,J,K(5),3%,L(8,6)   /* LOCAL VARIABLES
  60 DIM etc.
  --- Address generation possible up to here ---
  70 IF ESC GOSUB ...
  80 IF ERR 0 GOSUB ...
  :
  100 INPUT A

  :
  170 DIM 2%,A1(A)   /* VARIABLE AS INDEX!
  180 DIM A2(7,6)
  :
  200 LET A=A8   /* IMPLICIT DIMENSIONING INVALID FOR A8

  :
  250 DEALLO A2   /* USE "CLEAR A2" HERE AND PLACE
  260 DIM A2(7,6) /* STATEMENT 180 IN LOCAL PART

- Global area with changing lengths:

  10 REM DATA AREA DESCRIPTION:
  20 GLOBAL 3%, A,B,C,D   /* GLOBAL AREA
  30 DIM 1%,X   /* LOCAL AREA
  :
  50 ON X GOTO 100,200,300
  :
  100 LINK "PROG1"   /* TRANSFER GLOBAL AREA UP TO D
  :
  200 LINK;B;"PROG2"   /* GLOBAL AREA REDUCED TO D
  :
  200 LINK;C;"PROG3"   /* GLOBAL AREA REDUCED TO C

  10 REM PROG1
  20 GLOBAL 3%, A,B,C,D   /* SAME GLOBAL AREA

  10 REM PROG2
  20 GLOBAL 3%, A,B   /* REDUCED GLOBAL AREA

  10 REM PROG3
  20 GLOBAL 3%, A,B,C   /* REDUCED AND EXTENDED GLOBAL AREA
  25 GLOBAL 3%, E,F   /* USE NEW STATEMENT!
```

Program Structure

17.6.4 Other important information

Optimizing and deoptimizing unsegmented programs

The optimizer and deoptimizer are associated with the SAVE command (see chapter "BASIC Commands"):

Optimizing : SAVE ,D,[#<PartS>#] [<ProgName> [!]]
Syntax errors are reported interactively.

Deoptimizing : SAVE ,B,[#<PartS>#] [<ProgName> [!]]

File type remains: SAVE[#<PartS>#] [<ProgName> [!]]

Batch optimization/deoptimization

The optimizer can be called from the TAMOS selector (see chapter "Program Maintenance") to process a batch of programs (programs contained in a LIBR list). The batch optimizer needs a text file to log program names and to output syntax errors or differences between program lengths between B and D files.

A syntax check is performed in the following cases:

- Entry and modification of a BASIC program
- Optimization of a BASIC program (SAVE, D, ...)
- Loading a program (LOAD <ProgName>) if a BASIC program in the form of a D file is in the user partition.

The following must be noted when SAVE is used without a <PartS> specification but the size is entered in the file header:

Optimization : If the D file becomes larger than the B file, the code length is rounded up to a complete KB value and the old size in the header is increased by this value.
If the D file is smaller than the B file, the old size is retained in the header.

Deoptimization : If the B file becomes larger than the D file, the old size in the header is retained. If the B file is smaller than the D file, the code length is rounded up to a complete KB value and the old size in the header is reduced by this value.

It is therefore advisable to specify the size when saving B files. BASIC error 81 is reported if optimization results in the size exceeding the Active File Size.

Code expansion and program size

Converted BASIC programs for which variable addresses have been generated in the optimization process generally have the same code size as in B file representation. Nonetheless it is necessary to make sure that the maximum partition size is not exceeded. The following outputs are helpful in this respect:

- The previous maximum program size in words is output by QUERY in the RUNTIME SIZE column.

  Note: The program size is entered after RUN whenever the ensuing program size exceeds the previous one (depending on the program path). The program size is set to 0 in the event of SAVE.

The difference between B and D files is output in the course of batch optimization in the text file listing: DIF.: +-xxxx WORDS.

- For D files, the program listing contains the code sizes of B and D files as well as PROGRAM D-FILE SIZE xxxx WORDS.

Matching hash totals

The intermediate code of BASIC programs as entered differs from the intermediate code resulting from optimization and DUMP (entry B file -> SAVE ,D, ... -> DUMP ..). This characteristic means that the hash total for the original format can differ from the intermediate code resulting from deoptimization. In order to make sure that hash totals match, SAVE ,D, ... must always be performed first. Of course these differences in the intermediate code have no effect on semantics.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Structure

To save space and run time, the optimizer eliminates all superfluous parentheses in expressions:

- A(0,0) = A and A(0) = A; also A$(0) = A$ and A$(1) = A$.

- In matrix fields, A(5,0) = A(5); in strings, A$(1,5) = A$, A$(0,5) = A$ and A$ (3,5) = A$(3) if the dimensioned string length corresponds to the second index.

- In GLOBAL and DIM statements, which originally contained no complete specification of precision and for which variable addresses have been generated, the precision is always stored before the first numeric variable dimension when the statement is listed. The maximum length of the statement can be exceeded in this way!

Exception:

- If the monadic operator "-" (only affecting the first operand) occurs, this expression is always parenthesized, e.g. -1+B becomes (-1)+B.

Mixture of B and D files

BASIC programs can run with a mixture of B and D files. Since two interpreters are referenced at run time in this case, an extra partition must be generated to prevent a drop in throughput.

- There are no restrictions for programs without GLOBAL statements (B file -> LINK -> D file -> LINK -> B file).

- All segments in a sequence must be converted if GLOBAL statements exist (otherwise BASIC error 85 occurs at run time).

- It is possible to use a mixture of different D file program sequences and B file program sequences.

Testing Aids

18 Testing aids

The BASIC programming system provides the user with a series of aids for testing new or modified programs with respect to correct syntax and execution.

In detail, these are as follows:

- BASIC error indication when entering statements
- BASIC error indication when executing the program
- Single-step execution of the program and execution log
- STOP statement
- IF ERR 0 statement
- IF ESC statement
- Test run of the program (optionally commencing at a particular line number up to a line number)
- Direct execution of statements
- TRACE command

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

18.1 BASIC error indications

The BASIC processor is capable of detecting certain errors, mainly of a lexical but also of a syntactic nature, when statements are being entered. Furthermore, the statements are tested both individually and in their entirety for syntax and semantic errors on execution. If an error is detected here or on input of the statements, it is indicated to the user in the following form:

```
ERROR # xx  AT <LNo>  IN <ProgName>
```

"AT <LNo> IN <ProgName>" is omitted in the case of errors following the input of a statement. The meaning of error number xx can be found in the list of BASIC errors or displayed by means of the BASIC command

```
HELP [xx]
```

Testing Aids

Error indication during execution of a program aborts the program in serious instances only.
The display otherwise only provides the user with information and program execution continues. It is easy to pinpoint the statement causing the error as a result of the number being displayed. The standard BASIC error indication can be replaced by a user-defined error handling routine with the aid of the IF ERR 0 statement.

- Example:

  The entry

  10 OPEN # 3,"$LPT

  i.e. without closing quotation marks, results in the error message

  ERROR # 4

  being issued on the screen accompanied by an acoustic signal in the keyboard.

  When the command:

  HELP 4 or just HELP

  is entered, the following error text is displayed:

  FORMAT ERROR

18.2 IF ERR 0 statement

If the standard BASIC error indication is to be replaced by specific action to an error, the IF ERR 0 statement must be used.

Syntax:

IF ERR 0 [<Statement>]

When a BASIC error occurs at any point subsequent to this statement, <Statement> is executed.
In general, this will be a branch to an error-handling subroutine or STOP during the test phase.

Testing Aids

During program execution only one IF ERR 0 statement can be active. This means that each IF ERR 0 statement cancels out any similar statement which precedes it.

If BASIC errors are to be displayed again in the form

ERROR # xx AT <LNo> IN <ProgName>

specific reaction to an error can be cancelled by means of the statement IF ERR 0 (without <Statement>).

Note:

The error numbers:

3,6,16,18,19,20,27,31,59,76,77,81,86

cannot be handled with IF ERR 0.

All the channels are closed and control is passed to the BASIC processor.
When the IF ERR 0 statement has not been specified, errors are issued with:

ERROR # xx AT <LNo> IN <ProgName>

and program execution continues.

- Examples:

Error handling in the test phase:

- 10 IF ERR 0 STOP
  :

If an error occurs, the program is always interrupted and

STOP AT <LNo> <Statement>

is displayed, <LNo> being the line number of the statement causing the error.
The user can then display the contents of the variables which may have caused the errors in direct execution mode.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- 10 IF ERR 0 GOSUB ...
  :

  Branch to an error-handling subroutine. Error numbers and statements causing errors can be analysed in the subroutine by means of SPC 8 or SPC 10.

The special functions CHN, FLN and PRN are available for error handling with IF ERR 0.

18.3 IF ESC statement

User-defined reaction to pressing the ESC key.

Syntax:

```
IF ESC  <Statement>
```

<Statement> is executed when the ESC key is pressed.
If the ESC key is pressed in a program in which IF ESC <Statement> has not been coded, the program is interrupted and control is passed to the DEBUG mode (see "ESC Key").
IF ESC REM deactivates the ESC key.
IF ESC without the <Statement> cancels the ESC key handling routine.
If there are several IF ESC statements in a program, only the current one has any effect. If the ESC key is pressed several times, the "ESC activated" flag is only set once and is reset when <Statement> has been executed. The flag is also reset when a program is called up with CHAIN or LINK and IF ESC <Statement> must be recoded in the program which is called if ESC key handling is needed again.

• Example:

```
:
100 IF ESC GOSUB 5000
:

:
5000 PRINT TAB(0,24);"ESC KEY PRESSED";
5100 RETURN
```

18.4 Direct execution of statements

Individual BASIC statements can be executed immediately after input to support program testing in interactive mode. Input of the statement without a line number results in direct execution. This means that the statement cannot be included in the program.

As with all other statements, a syntax check is also carried out here after input.

The program located in the main memory partition at that moment is not changed as a result of direct execution of the statement. The data area of the program is still entirely available. Data can therefore be output (PRINT) or modified (LET) and variables can be created.

The following statements are excluded from this because they cannot be executed directly:

- BUILD #
- CLOSE #
- DATA
- DEF
- END
- ENV #
- FOR
- GOSUB
- GOTO
- MAT READ #
- MAT WRITE #
- NEXT
- ON
- OPEN #
- READ #
- RETURN
- SEARCH #
- STOP
- WRITE #

and, in D files, those statements which reference a user-defined function.

If one of these statements is input, the following message is issued:

ERROR # 55

(Statement is illegal from keyboard.)

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Testing Aids

- Example:

The program

```
 5 IF ERR 0 STOP
10 DIM A(10,10)
20 FOR I=1 TO 10
30   FOR J=1 TO 11
40     LET A(I,J)=I*J+I
50   NEXT J
60 NEXT I
```

encounters a BASIC error with the display text

```
STOP AT 40  LET A(I,J) ) = I*J+I
```

In this case, it is advisable to have the contents of the variables I, J and A(I,J) displayed by entering:

```
PRINT I,J, A(I,J)
```

Testing Aids

18.5 Single-step test run and execution log

If the BASIC system does not detect an error but the program still does not run as desired, a further feature is provided for troubleshooting.

If there is a logical error which the system cannot detect and which the user only recognizes as a result of incorrect results being output, it is extremely difficult to find out which statement has caused the error or at which statement it first occurred during normal program execution.
In this case, the program can be executed statement-for-statement, i.e. after a statement has been executed, execution of the next statement must be initiated by pressing the CR key.

Single-step mode is activated by the BASIC command

[<LNo>] DEBUG

<LNo> : Line number of the statement at which the test run is to be started.
If <LNo> is not specified, DEBUG commences at the first statement in the program.

The <LNo> specification is only meaningful if an initial RUN or DEBUG command has already been executed. DEBUG initiates the single-step test (DEBUG mode), outputs the first statement or the statement specified in <LNo> and issues the message:

STOP AT xxx <Statement>

where xxx is the line number.

The BASIC processor remains in the DEBUG mode and the user can initiate execution of the next statement by pressing the CR key.

The single-step mode is also initiated by a STOP statement or a RUN <LNo> command.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Testing Aids

An execution log can be obtained with the BASIC command:

<LNo> DEBUG <FileName>

The program is executed and the statements involved are logged in the file called <FileName> or on the specified printer.

Input of a command other than DEBUG or input of a BASIC statement with a line number causes the DEBUG mode to be cancelled.

Direct execution of statements and pressing the ESC key do not cancel the DEBUG mode.

- Example:

  300 DEBUG

  Start of the DEBUG mode at line number 300. This statement is not executed until the CR key is pressed. Subsequently the statement with the next line number is displayed.

- Example:

  DEBUG $LPT

  Start of the DEBUG mode at the first statement in the program. The execution log is output to the printer $LPT (see the device assignment table in the supervisor selector).

Testing Aids

18.6 Program test run

It is not necessary to quit the BASIC processor when carrying out a test run for a program which has been entered for the first time or which has been modified. It is sufficient to enter the BASIC command

RUN

from the BASIC command mode. In this case, the BASIC program is executed from the start to the end of the program in the form in which it is stored in the main memory partition. It is possible to modify this command by specifying a start and/or end line number:

[<LNo1>] RUN [<LNo2> [,<LNo2>]^4]

<LNo1>: Line number of the statement from which execution is to be started.

This specification is only admissible if

- an initial RUN command has already been executed, or
- the program was executed in the DEBUG mode up to <LNo1>,

and the program was not subsequently modified. If this specification is omitted, the program is started at the first statement (initial RUN).

<LNo2>: Line numbers at which the program is to be stopped (STOP condition) and control is to be passed on to the DEBUG mode.

When one of the specified line numbers is reached, the message:

STOP AT <LNo2> <Statement>

is displayed for the <LNo2> in question. All specified stop conditions are cancelled at this time. This statement has not yet been executed.

At this point, testing can continue in the DEBUG mode or contents of variables can be displayed in the direct execution mode to check that the program is being executed properly.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Testing Aids

- Example:

```
RUN 300
```

Starting a program at the beginning of the program. The program is stopped when line number 300 is reached.

```
PRINT X,Y,A(X,Y)
```

Checking the contents of the variables X, Y and A(X,Y).

```
300 RUN
```

Continuing to the end of the program.

Testing Aids

18.7 STOP statement

If program execution is to be interrupted several times at the same points during the test phase, it is advisable to insert the statement

STOP

at these points. This statement results in the program being interrupted at this point and

STOP AT <LNo> STOP

being displayed. Control is passed to the DEBUG mode. <LNo> is the line number of the STOP statement. Files which have been opened remain open.

When the STOP statement has been coded, the program is executed with

RUN<LNo>

just as it is without any STOP statements.

After the program has been interrupted, it is possible to continue testing in the DEBUG mode and have the contents of variables output to check that the program is being executed properly.

The program can then be brought to an end by means of

<LNo>RUN

On completion of the test phase, the STOP statements should be removed from the program.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Testing Aids

18.8 ESC key

The ESC key is another way of interrupting the execution of a BASIC program.

After the ESC key has been pressed, program execution is stopped, the DEBUG mode is initiated and the line number of the next statement to be executed is displayed by

```
STOP AT <LNo> <Statement>
```

It is possible to continue testing in the DEBUG mode and have the contents of variables displayed in direct execution mode.

It should be noted that the effect of the ESC key can be cancelled. It is then only possible to interrupt the program by pressing the CTL-Y and ESC keys, provided that the user is authorized to access SCOPE.

If program execution is to be interrupted by means of the ESC key in the testing phase, REM must be specified temporarily in the IF ERR 0 and IF ESC statements.

Generating BASIC error 99 by pressing the ESC key or using the IF ESC statement enables users to have a direct influence on program execution.

- Examples:

`10 IF ESC GOSUB...`	Branch to ESC routine when the ESC key is pressed

`10 IF ESC GOSUB...`
`:`
`230 IF ESC`
	Branch to ESC routine ...
	
	Cancellation of the effect of line 10 - revert to standard reaction to pressed ESC key.

`10 IF ESC REM`	ESC key has no effect.

Testing Aids

18.9 Dumping variables

After the program has been executed or after STOP or ESCAPE, all variables dimensjoned in the program can be dumped with their contents to screen, printer or a text file.

Syntax:

TRACE [<File>]

The output is 3 characters for the variable name, 4, 6, 10 or 14 digits for numeric variables in exponential notation and any length for strings. All numeric variables are output before the string variables. A non-printable character in a string is replaced by a blank.

This command does not cancel the DEBUG mode.

<File> : Specification of a printer or text file. If this specification is omitted, the variables are dumped to screen.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Testing Aids

18.10 Stopping at variables

Two additional formats of the TRACE command enable the program to be stopped when any values are assigned to certain variables or when specific values are assigned to certain variables. The command is entered before starting the program with RUN.

The command

[<LNo1>] TRACE [<LNo2>] {,<VarName>}^{5}

stops the program and initiates the DEBUG mode after the first statement in which a value was assigned to one of the variables <VarName>. This statement has been executed and is displayed. The name and value of the variables involved are also output. The program can be resumed with <LNo> RUN but it must be noted that in this case <LNo> refers to the line number following the statement which was output!

The command

[<LNo1>] TRACE [<LNo2>] {,<Condition>}^{2}

also stops the program and outputs the statement and variables involved. However the <Condition> has to be satisfied in this case.

<Condition>::= { <NVarName1><RelOper> { <NVarName2> / <DecNo> } / <NVarName> { = | <> } <DecNo1>:<DecNo2> / <SVarName1><RelOper> { <SVarName2> / <SLit> } }

<NVarName>=<DecNo1>:<DecNo2> is satisfied when the value (of the respective component) of <NVarName> lies between <DecNo1> and <DecNo2> inclusively. <NVarName> <> <DecNo1>:<DecNo2> is satisfied when the value is out of range.

The length of <SLit> is restricted to 16 characters. If <LNo1> or <LNo2> is specified, the TRACE command only takes effect from the first occurrence of <LNo1> (or up to the first occurrence of <LNo2>) in the program flow.

More details on TRACE can be found in the "BASIC Commands" chapter.

TAMOS Interfaces

19 TAMOS interfaces

19.1 Preparing programs for inclusion in selectors

All application programs included in TAMOS selectors must fulfil a series of interface conditions in order to ensure correct flow.

The interfaces are formed by standard subroutines which the programmer should store in text files by means of the BASIC command DUMP and load into every program with the aid of the BASIC command LOAD, as well as by calling up certain TAMOS programs at the end of the program.

Due to the fact that the standard subroutines require a series of variables, the following variables should not be used in the application part of the programs:

- String variables with the digit 7 (e.g.: H7$, L7$)
- the vectors S and T
  (S only when operating with job spooling)

The variables are filled by the standard subroutines as follows:

Without spooling:

Name	Meaning
H7$(122)	Header line
L7$(126)	Logbook record
M7$(55)	Message
2%:	
T(0)	Port number
T(1)	Message number in TF.PARAM

When working without job spooling, it is therefore necessary to include the following DIM statement at the start of the program:

```
DIM H7$(122),L7$(126),M7$(55),2%,T(1)
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

With job spooling:

Name	Meaning
H7$(122)	Header line
L7$(126)	Logbook record
M7$(55)	Message
D7$(20)	Job description
F7$(25)	User information
P7$(18)	LU/text file or program name
S7$(25)	Paper description
Z7$(160)	User information
2%:	
T(0)	Port number
T(1)	Message number in TF.PARAM
T(2)	Job creation time
T(3)	User number (group/user)
1%:	
S(0)	Job type
S(1)	RUN mode
S(2)	First-level selector coordinates
S(3)	Second-level selector coordinates
S(4)	Third-level selector coordinates
S(5)	Delete, job-skip, abort, restart code
S(6)	Number of issues
S(7)	Paper code
S(8)	First print position
S(9)	Last print position
S(10)	Lines per page
S(11)	Help field
S(12)	First page number

When operating with spooling, the following DIM statements must therefore be included:

```
DIM H7$(122),L7$(126),M7$(55),V7$(4),D7$(20),F7$(25)
DIM P7$(18), Z7$(160),S7$(25),1%,S(12),2%,T(3)
```

TAMOS Interfaces

19.2 Starting an application program

Application programs running under TAMOS must be selected from the appropriate selectors (see "Operator Manual"). TAMOS carries out the following processes in the time between the selection and the actual start of the program:

- An entry with the following data is made in the TF.PORT file:

  - CPU time
  - RUN mode
  - User no.
  - First-level selector coordinates
  - Second-level selector coordinates
  - Third-level selector coordinates
  - Program name (with LU number if not 0)
  - Logical units used

  This entry is made for each port.

- TAMOS provides the port starting the program with the following data in the common area:

  - Description of the 3rd level
  - Description of the 2nd level
  - Description of the 1st level
  - Date
  - The text: MESSAGE:
  - The two selector parameters
  - Program name and number
  - Logical units used

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

- A data record with the following data is written to the logbook (TF.LOGFILE):

  - Program type
  - Specification
  - Port number
  - Starting time
  - Header line
  - Status: START

  TAMOS supplies open data channels 0 and 1 with the following files:

  - TF.LOGFILE (channel 0)
  - TF.PARAM (channel 1)

  They should not be closed by application programs.

TAMOS Interfaces

19.3 Displaying the header and message line

The standard subroutine "Output header line" should be called at the beginning of each application program running under TAMOS.

This subroutine clears the contents of the screen and outputs the header line (line 0) and the message line (line 24).

The header line and message line are output as background characters.

Note:

Programs which only run in the phantom port need not call this subroutine.

- Variables used: H7$(122) = Header line

  T(0) = Port number (2%)

- Subroutine:

```
1100 LET T(0)=SPC 6
1110 CALL 3,T(0),H7$
1120 PRINT USING "##";'CS';'SB';H7$(5,66); TAB(67,0);
       "#";T(0);H7$(67,74); TAB(0,24);H7$(75,89);'SF';
1130 RETURN
```

The format of the header and message lines is described in the section "Transferring Standard Parameters via TAMOS".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

19.4 Terminating an application program

An application program which is started under TAMOS should return to the selector when it has finished. Furthermore, there should be a standard reaction to any errors that may occur. For this reason, every application program which is started under TAMOS should finish by calling up one of the following standard programs by means of CHAIN:

- TA.END
- TA.NCO
- TA.ABO

It is also important to take into account the setting of the Program Abort Flag (PAF).

Program Abort Flag (PAF)

Since programs consist of different critical processing phases, BASIC allows for different handling of program aborts in individual phases with the aid of CALL 29. This CALL can be used to change the setting of the PAF and override the ABORT Y/N entry in the selector. The PAF can have four different values with the following meaning:

0: Return to selector without error handling
1: Program restart (with decision by operator)
2: Calling a repair program
3: 2nd generation recovery

In addition as from NIROS 5.1:

4: Program restart (without decision by operator). When the program is restarted, TAMOS resets the PAF to 1. Appropriate tests should be carried out to ensure that errors are not handled in a continuous loop!

TAMOS Interfaces

TA.END

This program should be called when an application program has been correctly terminated. It closes all channels that were open and registers termination in the logbook (TF.LOGFILE). As from NIROS 5.0, the copy flags in the TAMOS archive file were set after each update and additionally in accordance with the selector entries of the program.
As from NIROS 5.1, the operating system only sets a copy flag after each update.

Note:

If channels 0 and 1 are not open when TA.END is called, then TA.ABO is automatically activated in place of TA.END.

TA.NCO

Application programs which are to be terminated without setting the copy flags must be terminated by calling TA.NCO on release levels up to NIROS 5.0. This is only advisable if an application program is terminated before it has updated data on disk. Since the operating system has assumed control of copy flags as from NIROS 5.1, the only difference between TA.ABO and TA.NCO is now the logbook entry.

TA.ABO

If an application program has to be aborted which has already updated data on disk, the TA.ABO program must be called.

TA.ABO closes all data channels which were open and registers abnormal termination of the program in the logbook. Abnormal termination is indicated at the workstation in the message line as soon as the TAMOS selector appears.

If a program with the selector entry ABORT: NO is abnormally terminated via a TA.ABO call, it is not possible to start further programs due to the fact that TAMOS only permits data recovery (MESSAGE: ONLY FORMATTING OR RECOVERY CAN BE SELECTED).

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

19.5 Logbook and message files

Logbook

System states such as program start, end, abort or system errors are automatically recorded in the logbook. In addition, the user can enter comments in the logbook. BASIC CALL 84 has been provided for automatic entries from application programs. Any BASIC errors that occur can also be entered with the LGERR statement. The entry has a fixed format as follows:

```
ERROR # <Integer>  AT  <LNo>  IN LU/<ProgName>
```

To distinguish application software from system software, the type should be 40 (= USERS) and the specification should be the user number (see CALL 84).

Message files

Message files are contiguous files with a record length of 512 bytes. 50 bytes for the text plus 1 delimiter are always reserved for a message. This means that 10 messages fit into one record. The standard message file in the system is called MESSAGES but it is possible to create any number of message files.

CALL 84 is available for processing message files. It enables messages to be read from the message file, displayed and entered in the logbook.

TAMOS Interfaces

- Example:

```
500  N1 = 3     /* READ/DISPLAY/LOG FUNCTION
510  N2 = 0     /* RESET STATUS
520  S2$= "MESSAGES"
530  S1$= " ",S1$
540  N5 = 1     /* TABLE BASE
550  N6 = 55    /* MESSAGE NO.
560  CALL 84,N1,N2,S1$,S1$,S2$,N5,N6
```

The program fragment reads the message "55 STATEMENT ILLEGAL FROM KEYBOARD" from the MESSAGES file, displays it in line 24 and enters it in the logbook.

It must be noted that the table base together with the error number define the error text. The same message is displayed if the table base is 1 and the message number 50 as when the table base is 2 and the message number 40.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

19.6 Common area

A special memory area of 512 bytes is allocated to each port. This memory area (common area) can be addressed from every port by specifying the port number. The common area is used for various purposes:

- Transfer of standard parameters by TAMOS to the application program
- Data interchange between mutually calling programs or between different ports
- Output of a program's error messages in a background partition with the aid of the MAIL processor

The common area is processed in BASIC by means of the subroutines

CALL 2: Moving data to the common area
CALL 3: Moving data from the common area

All 512 bytes of the common area are available for data interchange. The information can be structured in up to 12 data fields with any representation required. The programmer defines the representation. The common area is addressed and distinguished by means of the port number.

19.6.1 Transfer of standard parameters by TAMOS

TAMOS provides the application program with the following parameters in the form of a string at the following positions in the common area:

1-4	Reserved for error analysis
5-25	Description of the 3rd selector level
26-46	Descritpion of the 2nd selector level
47-66	Description of the 1st selector level
67-74	Date (YY.MM.DD)
75-89	The text: MESSAGE:
90-95	Selector parameters

96-113 (LU/), program name, program number
The LU number is only entered if it was specified when the selector was created. The LU number and program name are stored left-justified whereas the program number is right-justified. The last positions of the program name may be covered by the program number.
114-121 Logical units used

The screen layout to be generated from this is as follows:

Screen header line

1-21	Description of the 3rd selector level
22-42	Description of the 2nd selector level
43-66	Description of the 1st selector level
68-70	Port number (SPC 6)
72-79	Date (YY.MM.DD)

Message line

1-15	Text: MESSAGE:
16-65	Plain language message

The selector parameters and the program number can be used to control program flows.

Additional parameters are transferred in job spooling mode. They are described at the relevant place.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

19.7 Error handling

Handling serious errors which cause the application program to abort.

This subroutine contains the following functions:

- Return -1 to the application program when a BASIC error > 97 occurs, i.e. continuation of the program at the same point although ESC or CTL-C was pressed (SPC 8 = 99) or the repeat of a non-numeric input for a numeric variable (SPC 8 = 98) or when reading a locked data record (SPC 8 = 123).
- Entry of the error message in the logbook (SPC 8 <= 97)
- Calling the TAMOS program TA.ABO

TA.ABO displays a message to the effect that the program was aborted if this program was called from a selector.

- Call:

```
550 IF ERR 0 GOSUB 1000
```

- Subroutine:

```
1000 IF SPC 8>97 RETURN -1
1010 IF ERR 0 GOTO 1030
1020 LGERR     /*LOGBOOK ENTRY
1030 CHAIN "TA.ABO"
```

Line 1000 shows how the input of ESC or CTL-C can be intercepted with the aid of the IF ERR 0 statement. Line 1010 resets the IF ERR 0 switch from line number 1000 to 1030. In the event of another BASIC error during the logbook entry, this prevents a continuous loop from being generated by returns to line 1000.

19.8 Job spooler

The job spooler executes programs which require neither keyboard inputs nor outputs to display. It runs in an additional internal port which does not have a keyboard or screen - the so-called phantom port.

A detailed description of how to handle the job spooler can be found in the "Operator Manual".

⚠ The job spooler's phantom port has the port number 1.

19.8.1 Entering a job in the job spool file

In order to run a program in job spooling mode, the program name and other parameters must be recorded in the job spool file (TF.SPOOLQUEUE) by means of an entry program. Some of these parameters can be taken from the TF.PORT file.

Once the spooler has been started under TAMOS, it checks whether programs are due for execution by reading the entries in the job spool file. If this is the case, they are processed in succession. In addition to the program name, the spooler reads the other parameters which were written to the job spool file by means of the entry program. Some of these parameters are stored in the common area where they are called up by the executing program with the aid of a CALL 3.

Entering a text file job

A spooled text file job involves printing a text file by means of a standard print routine.

The following information must be stored in the job spool file:

- Job type (=0 for text file job)
- Job description (any job name which is displayed at the master port when the job spool file is displayed).
- LU/text file name

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

- The 4-digit information group (XYOZ):

  - Deleting the job and changing the number of issues
    X=0 : Allowed
    X=1 : Not allowed

  - Job skip
    Y=0 : Continue with next job if job aborts
    Y=1 : Stop the job spooler if job aborts

  - Delete code
    Z=0 : Retain text file after printout
    Z=1 : Delete text file after printout

- Number of issues (1 - 7999)
- Paper code: 0 and 1 = Standard paper
  2 to n = Non-standard paper

- Paper description (operator guidance for changing paper)
- Paper format (first and last print position and lines per page).
- Page number of first page to be printed.
- Parameters from the TF.PORT file:
  Job creation time
  RUN mode
  User number
  Program selector coordinates
- User-definable information

These parameters are values provided by the entry program that created the job. If a parameter is needed which differs from the one in TF.PORT, this must be modified accordingly (e.g. RUN mode).

The following restrictions apply to text file jobs which are delete-protected when they are processed with the program "DISP.JOB SPOOLFILE":
- Deleting (DEL) is impossible.
- Changing the number of issues (ISS) is impossible.
- If the job has an error or halt status, it can only be released (REL).

Entering a program job

The following information must be stored in the job spool file:
- Job type (=1 for program job)
- Job description (any job name which is displayed at the master port when the job spool file is displayed)
- LU/program name

TAMOS Interfaces

- The 4-digit information group (WXYZ):

  - Release (REL):
    W=0 : Job may be released if it aborts.
    W=1 : Job may not be released if it aborts.
    The job cannot be deleted, nor can the number of issues be changed.

  - Job skip:
    X=0 : Continue with next job if job aborts.
    X=1 : Stop the job spooler if the job aborts.

  - Abort control code (cf. Program Abort Flag)
    Y=0 : Program may be aborted
    Y=1 : Recovery necessary if program aborts.

  - RUN mode control code:
    Z=0 : Accept RUN mode from entry program
    Z=1 : RUN mode 0

- Number of issues
- Paper code: 0 = Printer not needed
  1 = Standard paper
  >1 = Non-standard paper
- Paper description (or user-defined information)
- Paper format (or user-defined information)
- Page number
- Parameters from the TF.PORT file:
  Job creation time RUN mode
  User number Selector coordinates
- User-defined information

These parameters are values provided by the entry program that created the job. If a parameter needs to differ from the specifications in the TF.PORT file, it must be modified accordingly (e.g. RUN mode).

The phantom port's common area is used by the job spooler to provide the job with parameters from the job spool file prior to starting.

The following restrictions apply to basic programs which are not allowed to be released and which are delete-protected when they are processed with the program "DISP.JOB SPOOLFILE":

- Deleting is only allowed if an error or halt status is set.
- It is not possible to change the number of issues.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

- Variables used:

```
M7$55)      = Message (help field)
D7$(20)     = Job description
P7$(18)     = LU/text or program name
S7$(25)     = Paper description
F7$(25)     = Optional user information
Z7$(160)    = User information
T(0)        = Port number (2%)
T(1)        = Error number (2%)
T(2)        = Job creation time (2%)
T(3)        = User number (2%)
S(0)        = Job type (1%)
S(1)        = RUN mode (1%)
S(2)        = First-level selector coordinates (1%)
S(3)        = Second-level selector coordinates (1%)
S(4)        = Third-level selector coordinates (1%)
S(5)        = Delete code and job skip for text file jobs or
              release, job skip, abort code
              and RUN mode for program jobs (1%)
S(6)        = Number of issues (1%)
S(7)        = Paper code (1%)
S(8)        = First print position (1%)
S(9)        = Last print position (1%)
S(10)       = Lines per page (1%)
S(11)       = Help field (1%)
S(12)       = Page number of first page to be printed (1%)
```

- Call:

```
550 GOSUB 710               /* SUBROUTINE CALL
560 IF T(1)=164 GOTO...     /* SPOOL FILE FULL
```

Note:

Programs can enter themselves in the job spool file. To do this, the routine that stores the parameters in the job spool file should be organized as a subroutine of the program which is to be spooled. All the variables listed above must be dimensioned before the subroutine is called. The variables S(1), T(3), S(2), S(3) and S(4) are taken from the TF.PORT file. S(1) may have to be changed. Fixed values must be assigned to the variables P7$, T(0), T(1), T(2) and S(0) and definable parameters must be assigned to the variables S(5), S(6), S(7), S(8), S(9), S(10) and S(12). The area allocated to the variable Z7$ (max. 160 bytes) can be used to transfer additional parameters to the program which has to be spooled. The variables D7$, S7$ and F7$ contain comments. M7$ and S(11) are help fields only.

TAMOS Interfaces

The parameter arrays used have the following format:

TF.PORT

From displacement 22 + 44 * port number

```
T(2)    = Job creation time (2%)
S(1)    = RUN mode (1%)
T(3)    = User number (2%)
S(2)    = First-level selector coordinates (1%)
S(3)    = Second-level selector coordinates (1%)
S(4)    = Third-level selector coordinates (1%)
```

TF.SPOOLQUEUE

```
D7$     = Job description (20 bytes)
S(0)    = Job type (1%)
T(2)    = Job creation time (2%)
S(1)    = RUN mode (1%)
T(3)    = User number (2%)
S(2)    = First-level selector coordinates (1%)
S(3)    = Second-level selector coordinates (1%)
S(4)    = Third-level selector coordinates (1%)
P7$     = LU/text or program name
S(5)    = Delete code and job skip for text file jobs or re-
          lease, job skip, abort code and RUN mode for program
          jobs (1%)
S(6)    = Number of issues (1%)
S(7)    = Paper code (1%)
S(8)    = First print position (1%)
S(9)    = Last print position (1%)
S(10)   = Lines per page (1%)
S(11)   = Help field (1%)
S(12)   = Page number of first page to be printed (1%)
F7$     = Optional user information (25 bytes)
Z7$     = User information (160 bytes)
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

TAMOS uses the common area to provide the following parameters to the spooled program:

```
H7$      = Header line (122 bytes)
F7$      = User-definable information (25 bytes)
S7$      = Paper description (25 bytes)
S(8)     = Fist print position (1%)
S(9)     = Last print position (1%)
S(10)    = Lines per page (1%)
S(7)     = Paper code (1%)
S(6)     = Number of issues (1%)
S(12)    = Page number of the first page to be printed (1%)
Z7$      = User information (160 bytes)
```

Subroutine:

```
705 REM ***ENTER JOB IN SPOOL FILE***
710 OPEN #2, "TF.PORT"
720 READ #2,0,22+SPC6*44;T(2),S(1),T(3),S(2),S(3),S(4);
725 CLOSE #2
730 OPEN #2,"TF.SPOOLQUEUE"
750 FOR S(11)=0 TO CHF 2-1
760   READ #2,S(11);M7$
770   IF M7$(1,1)=" " GOTO 800 /* EMPTY ENTRY FOUND
780 NEXT S(11)
785 LET T(1)=164
795 GOTO 845
800 LET T(1)=0
810 LET S(11)=0
820 LET T(2)=(SPC 2-INT(SPC2/24)*24)*3600+INT(SPC3/10)
840 WRITE #2,-2;D7$,S(0),T(2),S(1),T(3),S(2),S(3),S(4),
    P7$,S(5),S(6),S(7),S7$,S(8),S(9),S(10),S(11),S(12),
    F7$,Z7$;
845 CLOSE #2
850 RETURN
```

A program running in job spooling mode can use the common area to take the following information from the spooler;

```
H7$,F7$,S7$,S(8),S(9),S(10),S(7),S(6),S(12),Z7$
```

A negative value in S(7) means that there is a change of paper code, in other words appropriate adjusting routines must be carried out for program jobs.

A program should finish by calling one of the following programs:

- TA.END
- TA.NCO
- TA.ABO

Other logical units that have to be accessed by a BASIC program job are taken from the parent program's selector entry.

19.9 Parameters for the print spooler

The parameters for the print spooler are transferred in two main groups:

- Form description
- Spooler control

A string of 34 bytes is transferred internally. The form description takes up 9 of these bytes and spooler control 25.

19.9.1 Forms control

Form control parameters are transferred directly at OPEN time in the first 9 bytes of a parameter string which has to be at least 34 bytes long. If the string is shorter, the missing parameters are replaced by default values.

Besides being transferred from a program, parameters can also be taken from the TAMOS device assignment table or from an internal list of default values.

The parameters are set in accordance with the following priorities:

First/last print position	Lines per page
1. Device assignment table	1. Program
2. Program	2. Device assignment table
3. Default values	3. Default values

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interfaces

19.9.2 Spooler control

Spooler control parameters are transferred at OPEN time, starting at byte 10 in the parameter string.

Parameter format:

1 byte for priority	H	High
	*N	Normal
	L	Low
	S	Suspended Entry in spool file but no printing until manually changed to H,N,L
		If the priorities are specified in lower case letters, the job cannot be deleted from the spool file.
1 byte for print delay	Y	Print after closing the file
	*N	Print during the program run
1 byte for data hold	Y	Retain print data in spooler until printout is acknowledged by the operator.
	*N	Print data are deleted immediately after printing.
1 byte for paper adjustment (after printing)	Y	Call adjustment routine after printing the first page
	*N	No adjustment routine
1 byte for paper adjustment (before printing)	Y	Call adjustment routine before printing the first page
	*N	No adjustment routine
2 bytes for no. of copies	nn	Number of copies without original (00 ≤ nn ≤ 99)
2 bytes for paper code	nn	00 ≤ nn ≤ 63) (not specified = 00)
16 bytes for job description		Optional text

*: Default value

TAMOS Interfaces

19.10 Starting programs in a phantom port

Programs which require neither keyboard inputs nor output to display can be started automatically under the control of the job spooler. To this end, they must be entered in the job spooler by means of the routine described and are started at port number 1, the first phantom port, as standard.

Programs can be started by means of BASIC CALL 98 without the control of the job spooler. This routine terminates the activities in the addressed phantom port in an uncontrolled way. It is up to the programmer to arrange procedures.

In order to start a program with a CALL statement, it is necessary to execute the CALL 98 statement twice. The first time, the octal code ←334← (or the character assigned to octal code ←334← in the country-specific character set, e.g. backslash, Ö, etc.) must be sent to the addressed port. This causes the addressed port to be logged off in a defined manner.
Any program can subsequently be started by sending a system command.

Note:

After the CALL 98 statement has been issued, it is necessary to wait 2 seconds (SIGNAL 3,20) before the status is tested in <NVar>.

- Example:

  Starting the program called INVOICE in the phantom port (port number 1).

```
:
100 LET S=0                 /* <NVar> TO 0
110 LET A$= "←334←"         /* CODE FOR LOGGING OFF
115 CALL 98,1,A$,S          /* LOG OFF PORT#1
120 SIGNAL 3,20             /* WAIT 2 SECONDS
125 IF S GOTO ...           /* STATUS TEST
130 LET A$="INVOICE"        /* LOAD PROGRAM NAME
135 CALL 98,1,A$,S          /* START PROGRAM
140 SIGNAL 3,20             /* WAIT 2 SECONDS
145 IF S GOTO ...           /* STATUS TEST
:
```

19

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

TAMOS Interface

19.11 Messages from a phantom port

Messages from a phantom port can be handled automatically via the MAIL processor (cf. Data Interchange via the Common Area). BASIC CALL 4 can be used if only messages are issued which do not require a response.

The following subroutine covers the following functions:

- Output of a message in the master port's message line.

- Waiting for an input from the master port which is transferred to the phantom port in the common area.

- Variables used:
  M7$(55) = Message and response
  T(0) = Port number (2%)
  T(1) = Number of the message in TF.PARAM

- Call:
```
550 LET T(1)=xx                    /* LOAD MESSAGE NO.
560 GOSUB 1220                     /* SUBROUTINE CALL
or
550 LET M7$="Message"
560 GOSUB 1230                     /* SUBROUTINE CALL
```

- Subroutine:
```
1220 READ#1,INT(T(1)/10)+15,FRA(T(1)/10)*510;M7$;
1230 LET T(0) = -SPC6
1240 CALL 2,T(0),T(0),M7$          /* TRANSFER MESSAGE
1250 READ#1,14,137;M7$(39);
1260 LET M7$="←207←←376←←211←←376←←221←8←230←",
     M7$(39), "←376←←212←"
1270 IF ERR 0 GOTO 1330            /* ERROR ANALYSIS
1280 CALL 4,M7$                    /* DISPLAY MESSAGE
1290 SIGNAL 3,250                  /* WAIT 25 SECONDS
1300 CALL 3,T(0),T(0),M7$          /* FETCH RESPONSE
1310 IF T(0)=-SPC 6 GOTO 1250      /* NO RESPONSE
1320 RETURN
1330 IF SPC8 <> 38 GOTO 1010       /* ERROR HANDLING
1340 GOTO 1280
```

This subroutine displays the message JOB SPOOLER at the master port at intervals of 25 seconds. The actual message is displayed when the MAIL processor is called by means of the "S" key. Since the subroutine uses the IF ERR 0 routine, the condition has to be reset in the main program.

Program Maintenance

20 Program maintenance

This chapter describes the utilities which are available in the TAMOS system to maintain BASIC programs. These utilities are included in the EXPERT selector under BASIC, PROGRAMMING SYSTEM.

```
T A M O S          PROGRAMMING SYSTEM   BASIC              # 0 85.03.04/15:29


                   P R O G R A M   S E L E C T O R


                   1 ... CREATE PROGRAM
                   2 ... EXECUTE PROGRAM
                   3 ... PROGRAM LISTING
                   4 ... UNLOAD REMARKS
                   5 ... LOAD REMARKS
                   6 ... COMPARE PROGRAMS
                   7 ... RELEASE LIST
                   8 ... PROGRAM OPTIMIZER
                   9 ... PROGRAM ANALYSIS

                   SELECT NO. OR "CR"  ......


MESSAGE:
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

20.1 Program listing

```
PROGRAM LISTING     PROGRAMMING SYSTEM   BASIC              * 0 85.03.04/15:38


      DATE                    = 040385          LU NO. FILE NAME
      (<CR> = SYSTEM DATE)
      PROGRAMMER              = GILLESPIE       1  0   INVENTORY
                                                2  0   ACCOUNTS1
      LINES PER PAGE          = 72              3  0   ACCOUNTS2
      REMARKS ON LU NO.       =  0              4  0   NK.TEST
                                                5  0   RECEIVABLES
                                                6  0   ORDERS01
      OUTPUT TO PRINTER       = Y               7 N
      OUTPUT FILE ON LU NO.   =                 8
                                                9
      LIBR LIST ON LU NO.     = N              10
      LIBR LIST NAME          =                11
      PROGRAMS ON LU NO.      =                12


                              OKAY ? = Y


MESSAGE:
```

After BASIC program names have been entered, this utility outputs separate lists of

- statements,
- variables used and cross-references,
- line cross-references in branch statements and subroutine calls and
- channel number cross-references

to printer or to text files.

Inputs:

DATE:
A date of up to six digits can be entered here and this appears in this form in the header of the printout (just CR = system date).

PROGRAMMER:
A text of up to 16 characters which identifies the programmer and is also printed out in the header of the printout.

Program Maintenance

LINES PER PAGE:
Page length (at least 24 lines). The default value is the entry for $LPT in the device assignment table.

REMARKS ON LU NO.:
Optional: Number of the logical unit to which the remarks contained in the programs to be specified were unloaded. The remarks appear in the program listing. They are read from the text files of the same name but with the Y. prefix.

OUTPUT TO PRINTER:
OUTPUT FILE ON LU NO.:
Choosing between the assigned printer and text files which are stored under the same program file name but with the X. prefix. The names of the BASIC program files can have a maximum length of 12 characters when the maintenance utilities are used.

LIBR LIST ON LU NO.:
LIBR LIST NAME:
CR = No LIBR list
If an LU number is specified, the program requests the name of the text file created with LIBR which contains all BASIC programs to be listed (types B and D).

LU NO. FILE NAME
Specification of up to 12 BASIC program names with the associated LU numbers if no LIBR list was defined. Non-installed LUs and non-BASIC program names are rejected.
CR = Branch to OKAY question

The programs are listed in the order in which they were entered.

Control is returned to the selector when the program listing has been output.

An explanation of the printout and direct calling of this function under BASIC can be found in the "PLIST" section of the chapter "BASIC Commands".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

20.2 Unload remarks

```
UNLOAD REMARKS      PROGRAMMING SYSTEM   BASIC            # 0 85.03.04/15:42


                                                  LU NO. FILE NAME

     NEW REMARKS
          ON LU NO.        = 2                    1  0   RECEIVABLES
                                                  2  0   PAYABLES
                                                  3  0   INVENTORY
     DELETE OLD REMARKS    = Y                    4  0   ACCOUNTS2
                                                  5  0   ORDERS01
     LIBR LIST ON LU NO.   = N                    6 N
     LIBR LIST NAME        =                      7
                                                  8
     PROGRAMS ON LU NO.    =                      9
                                                 10
                                                 11
                                                 12


                              OKAY ? = Y

MESSAGE:
```

This utility is used to reduce BASIC programs which contain remarks to the pure program part in order to make storage space as small as possible and execution time as fast as possible. The programs can be put back to their original state at any time with the "LOAD REMARKS" utility.

The following inputs are required when you have selected the "UNLOAD REMARKS" utility from the EXPERT selector:

NEW REMARKS ON LU NO.:
It is necessary to specify the number of a logical disk unit to which the remarks are to be unloaded. The remarks are stored in text files whose file name consists of "Y." and the BASIC program name.

- Example:

  BASIC program name : TEST
  Remark text file name : Y.TEST

Program Maintenance

Since the length of file names is restricted to 14 characters in the system, it must be ensured that the BASIC program names do not exceed 12 characters in length.

Then answer the question

DELETE OLD REMARKS.

You can then provide details on the list containing the program names under

LIBR LIST ON LU NO. and
LIBR LIST NAME.

You can skip these questions by just pressing CR and this takes you to the column

LU NO. FILE NAME

where you can specify the names of the BASIC programs from which the remarks have to be unloaded.

LU number

specifies the logical disk unit on which the program is stored. Names which do not apply to BASIC programs and names of BASIC programs which have already had their remarks unloaded are rejected and an error message is issued. Up to 12 program names can be entered. The list is finished when just CR is entered instead of a program name.

OKAY:

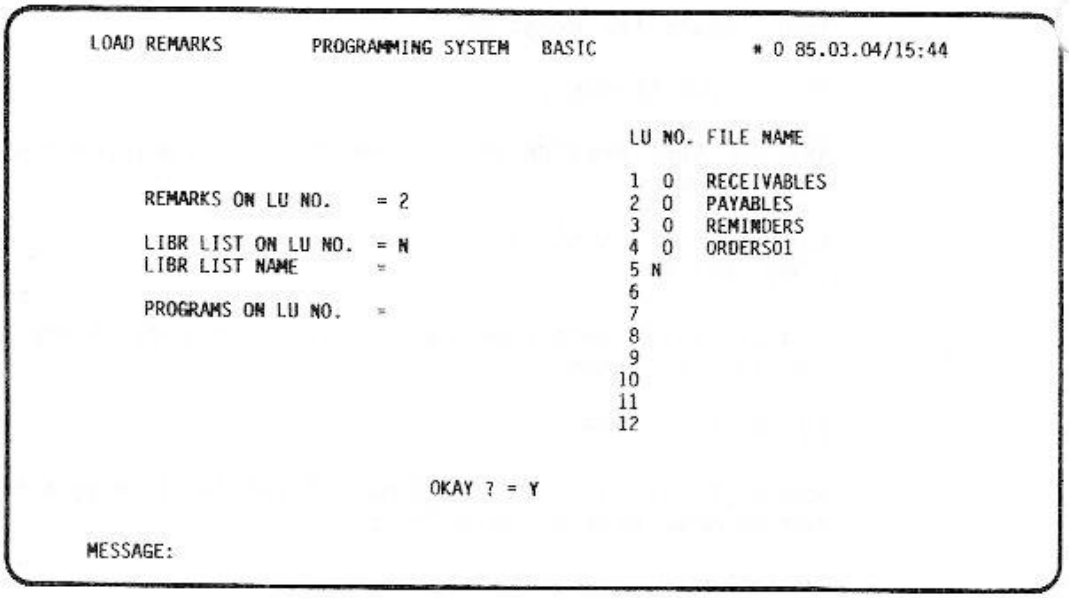
Y	Start unloading the remarks.
N	Return to "NEW REMARKS ON LU NO." and a chance to make corrections.
END	End of program without unloading the remarks.
PHA	The job (i.e. the program specified by the inputs) is entered in the spool file and started by the job spooler.
PHD	Like PHA but parameters can be entered for the job spooler in addition (job description, job skip and paper code).

The UNLOAD REMARKS function can be selected for one program by means of the BASIC command UNCOMMENT. All other explanations of this function can be found in the chapter "BASIC Commands" under UNCOMMENT.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

20.3 Load remarks



Remarks which have been unloaded from BASIC programs are re-inserted in the appropriate programs with the "LOAD REMARKS" utility to make them clearer, for instance when they have to be maintained. The remarks must have been stored on a logical unit beforehand with the "UNLOAD REMARKS" utility.

Inputs:

REMARKS ON LU NO.:
The number of the logical unit on which the remarks to be loaded were stored with the "UNLOAD REMARKS" utility. The remark files are deleted once the remarks have been loaded.

Program Maintenance

If the names of the programs whose remarks have to be loaded back are recorded in a library list, enter the number of the logical unit on which the list is stored and the name of the list under

LIBR LIST ON LU NO. and

LIBR LIST NAME.

Otherwise enter N. This enables you to enter the names of BASIC programs, into which remarks have to be loaded, in the column

LU NO. FILE NAME

Proceed in the same way as for UNLOAD REMARKS. The list of program names is finished by just pressing CR.

OKAY:

Y	Start loading the remarks.
N	Return to "REMARKS ON LU NO.:" and a chance to make corrections.
END	End of program without loading the remarks.
PHA/PHD	See "UNLOAD REMARKS"

The LOAD REMARKS function can also be selected for one program by means of the BASIC command COMMENT.

All other explanations on this function can be found in the chapter "BASIC Commands" under COMMENT.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

20.4 Compare programs

```
COMPARE PROGRAMS    PROGRAMMING SYSTEM   BASIC              # 0 85.03.04/15:43


     ENTER DATE OR END            : 03.05.85
     LINES PER PAGE               : 72
     OLD PROGRAM(S) ON LU.        :  2
     NEW PROGRAM(S) ON LU.        :  0
     PROGRAM NAME OR PART + ',',,,: GL.


     SELECT ON HASHTOTAL Y/N      : Y




MESSAGE:
```

This utility compares different versions of BASIC programs (which must be on different logical disk units) and lists the differing statements on the printer.

Inputs:

ENTER DATE OR END
A date of 6 to 8 characters which appears in this form in the header line of the printout.

LINES PER PAGE
Page length for the paper in the printer (minimum 20, maximum 72 lines).

OLD PROGRAM(S) ON LU
NEW PROGRAM(S) ON LU
The numbers of the logical disk units on which the old and new versions of the BASIC programs are stored.

Program Maintenance

PROGRAM NAME OR PART+',' ,,,
This defines which programs are to be compared. The following inputs are possible (example):

- Individual program	: GL.MAHDRUS04
- Part of a program group	: GL.MAHDRU,
- Entire program group	: GL.
- Several program groups	: GL.,FS.,LW.

If an incomplete program name is entered (concluded with ","), all programs whose names start like this are processed. The same applies if the program name which is entered is less than 4 characters in length (program group).

SELECT ON HASHTOTAL Y/N
This question only has to be answered if program groups have been entered. The file header of every BASIC program contains a hash total which indicates whether BASIC programs are identical (same hash totals) or if they differ.

Y If the old and new hash totals match, this utility continues immediately with the next program.

N The program is included in the printout even if the old and new hash totals match.

Explanation of the printout:

REL x/xx
Exact description (RELEASE/LEVEL) (cf. note).

OLD HASH TOTAL
NEW HASH TOTAL
Hash total of the old and new program version.

REM 01
One remark line in each BASIC standard program must correspond to a certain form and be updated for each new version (cf. note). An error message is output in the list if this line does not exist in the appropriate form.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

OLD-NEW
List of the different, new or omitted statements in the new version compared with the old version.

After the printout has finished, the utility returns to the question "PROGRAM NAME OR PART+',' ,,,:".

Note:

BASIC programs can be assigned to fixed program areas and different masters within them. It is also possible to record the version of a program. These details are respected by the utilities "COMPARE PROGRAMS" and "RELEASE LIST" if they are recorded in a REM statement in the prescribed format:

REM 01 "ProgName" PPFRRT

The abbreviations have the following meaning:

PP : Program area
A 2-digit number identifying the program area. The numbers 51 to 70 are reserved for application programs.

F : From Master
The number of the first master within the program area to which the program belongs.

RR : Release
A 2-digit number identifying the release.

T : To Master
Number of the last master within the program area to which the program belongs.

Program Maintenance

20.5 Release list

```
RELEASE LIST        PROGRAMMING SYSTEM   BASIC              * 0 85.03.04/16:20


ENTER DATE OR END            :  04.03.85
PROGRAM(S) ON LU.            :   0
ON PRINTER Y/N               :  Y
LINES PER PAGE               :  72
PROGRAM NAME OR PART + ',',,,:  ....................................

PROGRAM NAME        AREA     RELEASE     LEVEL              HASHTOTAL




00/PAYABLES         07       0           00                   44773
00/ORDERS01         07       0           00                   63821
                                                             108594

MESSAGE:
```

This utility outputs the release level and hash total of BASIC programs to display or printer.

Inputs:

ENTER DATE OR END
A date of 6 to 8 characters which appears in this form in the header line of the printout (only for output to printer). The default value is the system date in the format DD.MM.YY.

PROGRAM(S) ON LU
Number of the logical disk unit on which the programs to be verified are stored.

ON PRINTER Y/N
Y Output to printer
N Output to display

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

PROGRAM NAME OR PART+',' ,,,
This defines the programs for which the release level and hash total are to be listed. The following inputs are possible (example):

- Individual program : GL.MAHDRUSO4
- Part of a program group : GL.MAHDRU,
- Entire program group : GL.
- Several program groups : GL.,FS.,LW.

If an incomplete program name is entered (concluded with ","), all programs whose names start like this are processed. The same applies if the name which is entered is less than 4 characters in length.

After this input, the following list is computed and output to display or printer:

PROGRAM NAME	AREA	RELEASE	LEVEL	HASHTOTAL
xx/GL.MOS1	07	4	01	xxxxx
xx/GL.MOS2	07	4	01	xxxxx
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
				xxxxx

Explanation of the output:

PROGRAM NAME
Name of the verified program in the form

LU number/Program name

AREA
Program area (see "COMPARE PROGRAMS").

RELEASE/LEVEL
Release (from master)/level (see note on "COMPARE PROGRAMS").

HASHTOTAL
Hash total of the program in question. The grand total of all hash totals is specified at the end of the list.

After this list has been output, the utility returns to the prompt "PROGRAM NAME OR PART+',' ,,,".

Program Maintenance

20.6 Saving a program under another file type (optimization)

```
PROGRAM OPTIMIZER   PROGRAMMING SYSTEM   BASIC              # 0 85.03.04/15:46


          GENERATE PROGRAM CODE = 1
            1 ... OPTIMIZED
            2 ... STANDARD

          ON LOGICAL UNIT       = 0

          LIBR LIST ON LU NO.   = 0
            (N = TOTAL LU)
          LIBR LIST NAME        = NK.DUMMY

          ERROR LOG ON LU NO.   = 0
          ERROR LOG NAME        = NK.ERRORS

                        OKAY ? = Y




MESSAGE:
```

The PROGRAM OPTIMIZER utility can be selected via TAMOS to save a series of BASIC programs of file type B as D files in order to improve run-time.

Inputs:

GENERATE PROGRAM CODE = 1

Press CR to accept the "1", i.e. you want to optimize programs.

ON LOGICAL UNIT =

You enter the number of the logical unit on which the programs to be optimized are stored.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Program Maintenance

```
LIBR LIST ON LU NO. =
   (N = TOTAL LU)
LIBR LIST NAME      =
```

If the names of the programs to be optimized are recorded in a LIBR list, you must enter the number of the logical unit on which the list is stored. Input of "N" means that all programs on the specified logical unit will be optimized. The name of the LIBR list must be specified next.

```
ERROR LOG ON LU NO. =
ERROR LOG NAME      =
```

Here it is necessary to specify the LU number and name of a text file to store messages about program conversion. The structure of the messages is as follows:

```
  <ProgName>                        <ProgName>
                      or
  ERROR xx AT xxxxx                 DIF.: +-xxxx WORDS
```

```
OKAY ?
```

The inputs Y/N/END/PHA/PMD are possible here. These are described in UNLOAD REMARKS at the beginning of this chapter.

The PROGRAM OPTIMIZER utility can also be used to convert D files to B files if you enter "2" (STANDARD) instead of "1" at the beginning.

It is possible to optimize individual programs by means of the BASIC command SAVE,D. The command SAVE,B reverts the program to the standard B file.

Program Maintenance

20.7 Program analysis

```
PROGRAM ANALYSIS    PROGRAMMING SYSTEM   BASIC              # 0 85.03.04/15:46


      DATE                      = 040385          LU NO. FILE NAME
      (<CR> = SYSTEM DATE)
      PROGRAMMER                = TAYLOR           1  0   INVENTORY
                                                   2  0   ACCOUNTS2
      LINES PER PAGE            = 72               3  0   RECEIVABLES
                                                   4  0   PAYABLES
      OUTPUT TO PRINTER         = Y                5  0   REMINDERS
      OUTPUT FILE ON LU NO.                        6  N
                                                   7
                                                   8
      LIBR LIST ON LU NO.       = N                9
      LIBR LIST NAME            =                 10
                                                  11
      PROGRAMS ON LU NO.        =                 12

                                  OKAY ? = Y


MESSAGE:
```

The PROGRAM ANALYSIS utility provides support for program conversion. It analyses programs and outputs a program listing in which statements and variables, that are unfavourable for variable address generation, are marked.

The handling of the utility is basically the same as PROGRAM LISTING under TAMOS except that the question "REMARKS ON LU NO." is not asked.

The function of this utility can also be selected for one program by means of the BASIC command ANALYSE (see the chapter "BASIC Commands").

Once the program listing has been printed, control is returned to the selector.

An explanation of the printout can be found in the chapter "BASIC Commands" under ANALYSE.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

21 Programming examples

This chapter contains a series of programming examples for the following topics:

- Indexed file processing
- Keyboarding with CALL 1 support
- Copying a contiguous file to tape
- Printing a text file
- Intermediate storage of a printer file in a text file
- Restricting the size of a text file
- Reading in a punched card file
- Creating a file overview
- Linking a program into the TAMOS system

21.1 Programming examples for processing indexed files

An indexed file with several key directories, each with a key length of 3 words and a data record length of 41 words, is processed. The following variable names will be used in each of the examples:

```
C  = Channel number
V  = Directory number
V$ = Keys
N  = Relative record number
S  = Parameter and status variable
```

Dimensioning and file opening for all examples:

```
10 REM01 Program name
20 DIM R$(80),B$(17)          /* DATA REC. LENGTH, HELP FIELD
30 DIM V$(6)                  /* KEY
40 DIM 1%,C,V,S,2%,N
50 IF ERR 0 GOSUB ...         /* ERROR DURING FILE OPENING
60 INPUT 'CS',TAB(10,5),"NAME OF INDEXED FILE:",
   TAB(35,5),B$
70 INPUT TAB(10,6),"OPEN ON CHANNEL:",TAB(35,6),C
80 OPEN # C,B$
90 IF ERR 0 GOTO ...
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

21.1.1 Reading a data record directly (in directory 1)

```
100 LET V=1
110 PRINT  TAB (10,9);"KEY:";
120 INPUT  TAB (35,9),V$
130 SEARCH # C,2,V;V$,N,S     /* SEARCH FOR KEY
140 IF S GOTO 1500            /* STATUS <> 0
150 READ # C,N;R$
160                           /* PROCESS DATA RECORD
 :
600 INPUT TAB(10,22),"AGAIN (Y/N)?",TAB(35,22),B$
610 IF B$="Y" GOTO 100
620 CHAIN ""
1500 IF S=1 PRINT TAB(10,20);"KEY DOES NOT EXIST"
1510                          /* STATUS ANALYSIS
 :
1590 GOTO 600
```

21.1.2 Inserting a key with the associated data record

```
100 INPUT 'CS',TAB(10,8),"DIRECTORY NO.:",TAB(35,8),V
110 INPUT TAB(10,9),"KEYWORD:",TAB(35,9),V$
120 SEARCH # C,2,V;V$,N,S
130 IF S=0 GOTO 1560
135 IF S<>1 GOTO 1500
140 LET S=2
150 SEARCH # C,1,0;V$,N,S:
160 IF S GOTO 1500
170                           /* EDIT DATA RECORD
 :
210 WRITE # C,N;R$
220 LET Z=N
230 SEARCH # C,4,V;V$,N,S     /* ENTER KEY
240 IF S  GOTO 1500           /* STATUS <> 0
250 INPUT TAB(10,22),"AGAIN (Y/N)? ",TAB(35,22),B$
260 IF B$="Y" GOTO 100
270 CHAIN ""
1500 IF S=3 GOTO 1530
1510 IF S=1 GOTO 1560
1520 STOP          /*FILE DESTROYED OR NOT AN INDEXED FILE
1530 PRINT 'BEL'; TAB (10,20);"NO FREE RECORDS";
1540 SIGNAL 3,20
1550 GOTO 250
1560 PRINT 'BEL';TAB(10,20);"KEY ALREADY EXISTS";
1570 GOTO 250
```

Programming Examples

21.1.3 Deleting a key and deallocating the associated data record

```
100 INPUT 'CS',TAB(10,8),"DIRECTORY NO.:",TAB(35,8),V
110 INPUT TAB(10,9),"KEYWORD:",TAB(35,9),V$
120 SEARCH # C,5,V;V$,N,S:          /* DELETE KEY
130 IF S  GOTO 1500                 /* STATUS <> 0
140 LET S=3
150 SEARCH # C,1,0;V$,N,S           /* DEALLOCATE RECORD
160 IF S STOP                       /* FILE DESTROYED
170 INPUT TAB(10,22),"AGAIN (Y/N)?",TAB(35,22),B$
180 IF B$="Y" GOTO 100
190 CHAIN ""
1500 IF S=1 GOTO 1520
1510 STOP
1520 PRINT 'BEL';TAB(10,20);"KEY DOES NOT EXIST";
1530 GOTO 170
```

21.1.4 Sequential processing from the next higher key onwards

```
100 INPUT  TAB (10,8),"DIRECTORY NO.:", TAB (35,8),V
110 INPUT  TAB (10,9),"KEYWORD:", TAB (35,9),V$
120 SEARCH # C,3,V;V$,N,S           /* SEARCH FOR HIGHER KEY
130 IF S GOTO 1500                  /* STATUS <> 0
140 READ # C,N;R$                   /* READ DATA RECORD
150                                 /* PROCESS DATA RECORD
 :
250 GOTO 120
1500 IF S<>2 STOP       /*FILE DESTROYED OR NOT AN INDEXED FILE
1510 PRINT 'BEL'; TAB (10,20);"END OF DIRECTORY REACHED";
1520 SIGNAL 3,20
1530 CHAIN ""
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

21.2 Example of an input routine

Many application programs contain input-oriented parts in which alphanumeric or numeric data are requested by means of INPUT statements and these inputs are then processed by other parts of the program. CALL 1 provides an optimum way of handling such routines. Some of the features which this offers are:

- Preparing an input field
  - Defining the position of the field
  - Delimitation through a background blank
  - Specification of values
  - Input mask in the form of dots
- Checking the input
  - For the defined type (alphanumeric, packed or unpacked numeric)
  - For the permissible number of integer and decimal places
  - For the sign
- Specifying standard inputs with special functions
  - To display possible inputs (with ?)
  - To duplicate existing values (with @)
  - To return to previous input fields (with ↑)
  - For other special functions (with 3-character commands which have to be specified)

A typical input consists of the following steps:

- Output of a prompt for the input field
- Definition of parameters for preparing the input field and, possibly, a default value
- Definition of special inputs
- Branch to a subroutine:
  - CALL 1, function 1: input preparation
  - Input
  - CALL 1, function 2: input check:
    - In the event of a special input: corresponding special handling routine
    - In the event of correct input: processing

Programming Examples

In the following example, the input routine creates a distance table which is written to a formatted file, i.e. the name of a town and the associated distance is written to a data record.

The actual input routine has been implemented here as a subroutine which can be inserted in other programs in this form. In particular, this means that some subroutine functions are neither used nor checked by the example given.

After the input, various checks are made which can result in error messages 170 to 176 in the TF.PARAM file.

Other program parts (error handling, TAMOS interface routine and the like) are not shown here.

```
 10 DIM S1$(13),S2$(21),S3$(6),S4$(4),S5$(18),T2$(21)
 20 DIM S$(21),1%,N1,N2,I,E
 30 OPEN # 1,"TF.PARAM"
 31 REM ---- TF.PARAM contains error messages for CALL 1
 40 OPEN # 2, "DIST"
 41 REM ---- The distance table is stored
 42 REM ---- in DIST
 50 REM 50-99: Positioning the file pointer
100 LET S3$="CANEND"
101 REM ---- S3$ specifies possible special inputs
102 REM ---- Here: CAN and END
110 PRINT 'CS';'SB';TAB (10,5);"TOWN:"; TAB (10,7);
    "DISTANCE:"
111 REM ---- Prompts for inputs
120 REM ********** ENTER NAME OF TOWN ********************
130 LET S1$="2125052102000"
131 REM ---- S1$: Parameter string for CALL 1
132 REM ---- 2-1-25-05-21-02-0-0-0 means:
133 REM ---- 2 : Dotted input mask with 'CF'
134 REM ---- 1 : Alphanumeric input
135 REM ---- 25-05 Start position of input field
136 REM ---- 21-02 Max. and min. no. of characters
137 REM ---- 0 : No decimal places
138 REM ---- 0 : No sign
139 REM ---- 0 : 3-character input possible
140 LET S4$="1100"
141 REM ---- S4$ : Selection string for CALL 1
142 REM ---- 1-1-0-0 means:
143 REM ---- 1 : Input @ not allowed
144 REM ---- 1 : Input ↑ not allowed
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

```
145 REM ---- 0 : Input of 1st 3-character command allowed
146 REM ----     (here: CAN)
147 REM ---- 0 : Input of 2nd 3-character command allowed
148 REM ----     (here: END)
150 GOSUB 9000   /*            --> Input and check
160 ON N1+1 GOTO  180, 130, 130, 400, 500
161 REM ---- Reaction to result of input check
162 REM ---- N1+1 = ... means:
163 REM ---- ...1 : Correct input (not special input)
164 REM ---- ...2 : @entered (not allowed here)
165 REM ---- ...3 : 'entered (not allowed here)
166 REM ---- ...4 : CAN entered (cancel entry)
167 REM ---- ...5 : END entered (end of program)
170 LET S$=S2$
171 REM ---- S$ : Stores names of town for later
172 REM ----      writing to DIST file
180 REM ********* ENTER DISTANCE *****************
190 LET S1$="1225070401000"
191 REM ---- Dotted input mask (1), numeric input (2),
192 REM ---- start position (25,07), no. of integer
193 REM ---- places (min. 01, max. 04), no. of decimal
194 REM ---- places (0), no sign (0),
195 REM ---- 3-character input allowed (0)
200 LET S4$="1001"
201 REM ---- 'and CAN allowed, @and END are not
210 GOSUB 9000  /*             --> Input and check
220 ON 11+1 GOTO 230, 190, 130, 450, 190
230 LET E=N2
240 REM ********** WRITE DATA RECORD ******************
250 WRITE # 2;E,S$;
260 GOTO  130  /*              --> New input
350 REM ********** SPECIAL INPUTS **********************
400 REM 400-449: Reaction to CAN for name of town
450 REM 450-499: Reaction to CAN for distance
500 REM 500-549: Reaction to END for name of town
9000 REM ********** INPUT ROUTINE **********************
9010 LET T2$=S2$          /* Saving an existing value
9020 LET N1=0
9030 CALL 1,N1,S1$,S2$,N2,S5$      /* Input preparation
9040 INPUT 'BP', S2$                /* Input at beginning of
field
9050 IF S2$="?" GOSUB 9200         /* Possible inputs ?
9055 IF S2$="?" GOTO 9040          /* New input
9060 IF S1$(2,2)="1" LET N2= LEN S2$
9070 CALL 1,N1,S1$,S2$,N2,S3$,S4$,S5$      /* Input check
9080 IF N1=0 RETURN                        /* Correct input
9090 IF N1>1 RETURN          /* Allowed special input (except @ )
9100 IF N1=1 GOTO 9150                     /* @ allowed and entered
9110 GOTO 9020                             /* N1=-1: incorrect input
9150 LET N1=0                              /* Processing of input @:
9160 LET S1$(1,1)="3"
```

```
9170 LET S2$=T2$                    /* Fetching saved value
9180 CALL 1,N1,S1$,S2$,N2,S5$       /* Checking it
9190 RETURN
9200 PRINT TAB (17,24);'CFF';TAB (17,24);
9210 IF S4$(1,1)="O" PRINT " @/";    /* Display of
9220 IF S4$(2,2)="O" PRINT " ↑/";    /* possible inputs
9230 FOR I=3 TO  LEN S4$             /* in line 24
9240   IF S4$(I,I)="O" PRINT S3$(I*3-8,I*3-6);"/";
9250 NEXT I
9270 PRINT 'BS';'BP';
9280 RETURN
```

21.3 Copying a contiguous file to magnetic tape

The following variables are used:

- U = Magnetic tape unit number
- F = Function
- S = Status
- C = Block/record length
- A = Actual (written) length
- R$ = Data record
- D$ = Name of disk file

The output does not involve code conversion.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

```
10 IF ERR 0 GOSUB 330
20 DIM D$(20)
30 PRINT 'CS';TAB(0,24);"COPY CONTIGUOUS FILE TO TAPE";
40 INPUT 'CR',"FILE NAME:", TAB (30),D$ /* CONTIG. FILE
50 INPUT 'CR',"RECORD LENGTH:", TAB (30),C
60 IF C>2048 GOTO 50                    /* MAX. RECORD LENGTH=2048
70 DIM R$(C)                            /* DIM WITH RECORD LENGTH
80 OPEN # 2,D$                          /* OPEN CONTIGUOUS FILE
90 LET F=12                             /* LOAD FUNCTION
100 CALL 70,U,F,S                       /* OPEN AND REWIND
110 IF S GOTO 220                       /* ERROR DURING OPEN
120 LET F=6                             /* LOAD FUNCTION
125 CALL 70,U,F,S                       /* WRITE TAPE MARK
130 IF S GOTO 200                       /* ERROR
140 LET F=5                             /* LOAD FUNCTION
150 READ # 2;R$                         /* READ DISK RECORD
160 CALL 70,U,F,S,C,A,R$                /* WRITE BLOCK TO TAPE
170 IF S GOTO 200                       /* ERROR DURING WRITE
180 LET R$=" ",R$                       /* CLEAR STRING
190 GOTO 150                            /* NEXT DISK RECORD
200 LET S1=S                            /* SAVE ERROR CODE
210 GOSUB 250                           /* CLOSE TAPE
220 PRINT 'CR';"TAPE ERROR = ";S
230 CLOSE # 2                           /* CLOSE CONTIGUOUS FILE
240 CHAIN ""                            /* CHAIN TO SCOPE
250 LET F=10                            /* LOAD FUNCTION
260 LET S=0                             /* STATUS TO 0
270 CALL 70,U,F,S                       /* WRITE TAPE MARK
280 LET F=13                            /* LOAD FUNCTION
290 LET S=0                             /* STATUS TO 0
300 CALL 70,U,F,S                       /* CLOSE TAPE UNIT
310 LET S=S1                            /* RESET STATUS
320 RETURN
330 IF SPC 8>97 RETURN -1               /* ESC OR INCORRECT INPUT
340 IF SPC 10<110 GOTO 360              /* STILL NO TAPE OPEN
350 GOSUB 250
360 IF SPC 8=51 GOTO 400                /* END OF FILE?
370 PRINT 'CR';"BASIC ERROR # "; SPC 8;"LINE # "; SPC 10
380 IF SPC 10<90 CHAIN ""               /* STILL NO OPEN
390 GOTO 230
400 PRINT 'CR';"END OF FILE"
410 GOTO 230
```

Programming Examples

21.4 Printing a text file

If the output unit used is a line printer, the inputs "FIRST PRINT POSITION" and "LAST PRINT POSITION" are ignored.

```
 10 REM ************************************************************
 11 REM ************ MASK CREATION ROUTINE ***********************
 12 REM ************************************************************
 20 PRINT 'CS'; TAB (10);"PROGRAM : Print Text File"
 30 PRINT TAB (0,2); "Press CR to Accept Print Parameters"
 40 PRINT TAB (10,5); "FILE NAME"; TAB (40); ":";
    'CR'; TAB (10); "STRING LENGTH"; TAB (40); ":";
    'CR'; TAB (10); "PRINTER NAME"; TAB (40); ":"
 50 PRINT TAB (10); "FIRST PRINT POSITION"; TAB (40); ":";
    'CR'; TAB (10); "LAST PRINT POSITION"; TAB (40); ":";
    'CR'; TAB (10); "PAGE LENGTH"; TAB(40); ":"
 60 REM ************************************************************
 61 REM ************* INPUT AND CHECKING ROUTINE *****************
 62 REM ************************************************************
 70 IF  ERR 0 GOSUB  590
 80 DIM D$(20), P$(3), P1$(9)
100 LET P1$="0", P1$
110 INPUT  TAB  (42,5),D$          /* Entry of text file name
120 OPEN # 2,D$
130 INPUT  TAB  (42,6),N           /* Entry of string length
140 IF FRA N GOTO  110
150 IF N<0 GOTO 110
160 IF N>999 GOTO 110
170 DIM A$(N)
180 INPUT  TAB (42,7),D$           /* Entry of printer name
190 INPUT  TAB (42,8),P$           /* Entry of 1st print position
200 CALL 62,P$
210 IF P$="" LET P$="000"
220 LET P1$(4- LEN P$,3)=P$
230 INPUT  TAB (42,9),P$           /* Entry of last print position
240 CALL 62,P$
250 IF P$="" LET P$=131"
260 LET P1$(7- LEN PS,6)=P$
270 IF P1$(1,3) =P1$(4,6) GOTO  190
280 INPUT  TAB (42,10),P$          /* Entry of page length
290 CALL 62,P$
300 IF  LEN P$=1 GOTO 280
310 IF P$="" LET P$="072"
320 LET P1$(10- LEN P$,9)=P$
330 REM ************************************************************
331 REM *********** PROCESSING ROUTINE ***************************
332 REM ************************************************************
340 OPEN # 3;P1$,D$
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

```
350 DIM B$(1)
351 REM ---- B$ : Help variable for EOF test
360 LET X=0
370 LET Y=0
380 READ # 2,X,Y;A$
381 REM ---- File pointer position before READ # :
382 REM ---- Block X, byte Y
390 PRINT # 3;A$;
400 LET X1= CHF 102
410 LET Y1= CHF 202
411 REM ---- File pointer position after READ # :
412 REM ---- Block X1, byte Y1
413 REM ---- X = X1 : File pointer still in old block after
414 REM ----           reading
415 REM ---- X < X1 : Moved over block boundary
420 IF X=X1 LET D=Y1-Y
430 IF X<X1 LET D=((X1-X)*512)+(Y1-Y)
431 REM ---- D : Length moved by file pointer
432 REM ---- D = LEN A$ or D = LEN A$ + 1
440 IF D= LEN A$ GOTO 510
450 REM ---- D = LEN A$ + 1 : File pointer must be
451 REM ---- moved back by 1 character
452 LET Y=Y1-1
460 LET X=X1
461 REM ---- Set X and Y for direct reading
462 REM ---- Y >= 0 : Correctly moved back, file pointer stayed
463 REM ----           in block
470 IF Y>=0 GOTO  380
471 REM ---- Y < 0 : Incorrectly moved back -
472 REM ----          move file pointer
473 REM ----          to last byte in
474 REM ----          previous block
480 LET Y=512
490 LET X=X1-1
500 GOTO  380
510 REM ---- D = LEN A$ : End of character
511 REM ---- transfer for READ # statement
512 REM ---- when CR character is reached
513 REM ---- (then continue reading) or end
514 REM ---- of file reached (then end of program)
515 READ # 2;B$
516 REM ---- B$ = "" : End of file reached
517 REM ---- B$ <> "" : Continue reading! Set X and Y
518 REM ----            for direct reading
520 IF B$="" GOTO 560
530 LET X=X1
540 LET Y=Y1
550 GOTO  380
560 CLOSE # 2, # 3
570 END
```

```
580 REM ************************************************************
581 REM ********** ERROR HANDLING ROUTINE ************************
582 REM ************************************************************
590 IF  SPC 8=38 RETURN -2
591 REM ---- SPC 8 = 38 : CALL 62 detected an illegal input;
592 REM ---- make new input possible
600 IF  ERR 0
610 PRINT  TAB (0,20);"BASIC ERROR NO."; SPC 8;
    "AT LINE NO."; SPC 10
620 IF  SPC 10>100 CLOSE # 2
630 IF  SPC 10>340 CLOSE # 3
640 PRINT "PROGRAM TERMINATED"
```

2.1.5 Intermediate storage of a printer file in a text file

If an error is detected in the program shown in the preceding example, a text file is created and the records to be printed are stored in printable format in the text file.

```
        MASK CREATION ROUTINE and
        INPUT AND CHECKING ROUTINE are
        the same as lines 10-320 in the
        preceding example.

330 REM ************************************************************
331 REM ********** PROCESSING ROUTINE ****************************
332 REM ************************************************************
335 IF ERR 0 GOTO 670          /* Error when opening the printer
340 OPEN # 3;P1$,D$            /* Printer opening
345 IF ERR 0 GOSUB 590         /* Old error handling routine

        Further PROCESSING ROUTINE and ERROR
        HANDLING ROUTINE are the same as lines
        350-650 in the preceding example

660 REM ************************************************************
661 REM **** PRINTER OPENING ERROR HANDLING ROUTINE *************
662 REM ************************************************************
670 IF ERR 0 GOSUB 590
680 INPUT  'CS' , TAB(10,5), "NAME OF TEXT FILE:",D$
690 LET D$(LEN D$+1)="!"       /* Overwrite existing file
700 IF ERR GOTO 740            /* Error when opening text file
710 BUILD # 3,D$               /* Text file opening
720 IF ERR 0 GOSUB 590
730 GOTO 350
740 PRINT TAB(15,24);"NAME IN USE BY OTHER ACCOUNT";
750 GOTO 670
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

21.6 Example of CHF function

A text file used as a log file must not exceed 50 blocks in length. When 40 blocks are reached, a warning is issued. If the 51st block has been created, a flag is set and the current program is abnormally terminated at the next interface in order to print out the log file.

```
10 REM01 Program name
20 DIM M$(50)
30                                      /* PREPARE M$
 :
100 GOSUB 9000                          /* CALL LOG FILE ENTRY
 :
9000 PRINT # 3,CHF103,CHF203;M$         /* RANDOM WRITE
9010 IF CHF 103>50 GOTO 9100            /* ABORT?
9020 IF CHF 103<40 RETURN               /* WARNING?
9030 PRINT  TAB (0,24);'LD';"PRINT LOG FILE!"
9040 SIGNAL 3,20
9050 RETURN
9100 LET M=1                            /*FLAG FOR ABORT
9110 PRINT  TAB (0,24);'LD';"LOG FILE BEING PRINTED";
9120 RETURN
```

21.7 Reading a punched card file into a contiguous file

Transferring a punched card file to a contiguous file on disk. The end criterion for the punched card file is a /* card.

```
10 DIM D$(17),R$(80)
20 INPUT 'CR',"CONTIGUOUS FILE : ",D$
30 OPEN # 3,"$CRD1"                     /* PUNCHED CARD FILE
40 OPEN # 4,D$                          /* CONTIGUOUS FILE
50 READ # 3,R$                          /* READ PUNCHED CARD
60 IF R$="/*" GOTO 100                  /* END OF FILE
70 WRITE # 4;R$                         /* WRITE DATA RECORD
80 GOTO 50                              /* NEXT CARD
 :
100 CLOSE # 3                           /* PUNCHED CARD FILE
110 CLOSE # 4                           /* CONTIGUOUS FILE
120 PRINT "END OF PROGRAM";
130 CHAIN ""                            /* CHAIN SCOPE
```

Programming Examples

21.8 Use of CALL 97

The program draws up a list of all files including the file characteristics of the system disk.

Explanation of values printed out by the program:

```
FILENAME: Name of file
SIZE    : File size in blocks (sectors)
USER    : User number
AGE     : Hours since file creation
HSLA    : Hours since last access
TYPE    : File type
HBA     : Header block address (octal)
```

```
 10 REM  ********** CALL 97 EXAMPLE **********
 20 REM
 30 REM
 40 REM  ********** DIMENSIONING *************
 50 REM
 60 REM
 65 IF ERR 0 GOTO 1000
 70 DIM 1%,J,F1,F9,2%
 80 DIM A$(3)
 90 DIM F$(15)
100 DIM H$(108),L$(125),H1$(108)
110 REM
120 REM ********** PREPARE HEADLINE ***********
130 REM
150 LET H$=" ",H$
160 LET H$(1,8)="FILENAME"
190 LET H$(38,41)="SIZE"
200 LET H$(49,55)="USER"
210 LET H$(63,65)="AGE"
220 LET H$(73,76)="HLSA"
230 LET H$(85,88)="TYPE"
250 LET H$(106,108)="HBA"
260 LET H1$="-",H1$
270 OPEN # 2,"$LPT"
280 REM  ********** PRINT HEADLINE **********
290 LET P9=P9+1
300 PRINT#2;"CALL 97 EXAMPLE";TAB(90);"PAGE :";P9;
    "←215←"
310 PRINT # 2;"←215←←215←"
320 PRINT # 2;H$
330 PRINT # 2;H1$
340 LET Z1=0
350 CALL 97,U,R,F$,A,T,S,Q,C,I,D,L,H
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

```
370 IF R=-1 GOTO 900                    /* END OF FILE?
380 REM  ********** CONVERTING DECIMAL TO OCTAL *****
390 FOR J=1 TO 10
400 IF H<8*J GOTO 420
410 NEXT J
420 FOR J1=1 TO J
430 LET H1= INT (H/8↑(J-J1))*10↑(J-J1)
440 LET H2=H2+H1
450 LET H=H-( INT (H/8↑(J-J1)))*8↑(J-J1)
460 NEXT J1
470 LET H=H2
480 LET H2=0
490 REM  ********** FETCH FILE TYPE FROM T **********
491 REM  ********** T1 = DEC. FILE TYPE    **********
492 REM  ********** T2 = OCTAL FILE TYPE   **********
493 REM  ********** R1 = PROCESSOR TYPE    **********
500 LET T1=T-32* INT (T/32)
510 LET T2=T1-( INT (T1/8)*8)+( INT (T1/8)*10)
520 REM  ********** FETCH CONTROL DIGIT (R,L,I) FROM T
530 LET R1=((T-512* INT (T/512))-T1)/64
580 REM  ********** FETCH ACCOUNT USER NUMBER
610 LET U1=A
620 REM  ******* HOURS SINCE CREATION AND LAST ACCESS
630 LET D= SPC 2-D
640 LET L= SPC 2-L
650 REM  ********** PREPARE PRINT LINE **********
660 LET R1=R1*100
680 LET A$(1,3)=T2 USING "###"
690 LET L$=" ",L$
700 LET L$(1,LEN F$)=F$
740 LET L$(38,41)=S USING "####"
770 LET L$(53,54)=U1 USING "##"
780 LET L$(61,65)=D USING "#####"
790 LET L$(72,76)=L USING "#####"
800 LET L$(85,86)=A$
820 LET L$(104,108)=H USING "#####"
830 PRINT # 2;L$
840 REM  ********** NEW PAGE ???? **********
850 LET Z1=Z1+1
860 IF Z1<34 GOTO 350
870 PRINT # 2;"←214→";
880 GOTO 290
900 CLOSE # 2
910 CHAIN ""
1000 PRINT
1010 PRINT "BASIC ERROR #";SPC8;"LINE #";SPC10
1020 CHAIN ""
```

Programming Examples

21.9 Standard base part for application programs in TAMOS

21.9.1 Introduction and handling

This standard base part can precede any user-written application program. It contains the following routines:

- Transfer and display TAMOS header line
- TAMOS message output
- TAMOS logging routine
- Standard error handling
- Insert routine for indexed files
- Routine for input preparation and checking
- Enter TAMOS job in the job file
- TAMOS communication from phantom to master
- Printer error routine
- Paper change under spooler
- Transfer the TAMOS area for the spooler with printer opening

In order that this base part can be employed universally, it should be stored in a text file on the system disk (logical unit 0) by means of the BASIC command DUMP when it has been entered and tested.

Application programs preceded by the base part must commence with line number 1660 or more. The base part must be inserted by means of the BASIC command LOAD.

Numbering commences at 10 in steps of 10. This enables programs created with this base to be renumbered at any time without changing the branch addresses in the base part.

It is assumed that no changes are made in the base part other than lines 10 and 400 (and 90, if necessary). The program name, area, release, etc. must be entered in line 10 by means of "REM01".

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

If the base part has to be reduced for reasons of capacity , this should be carried out as follows:

- Parts should be deleted from finish to start, if these are not required and are contiguous:

  1310 DELETE 1630 if the program is not a print program.

  1170 DELETE 1630 if the program cannot be spooled and it is not a print program.

  990 DELETE 1630 if a job is not entered in the spool file by means of this program and the conditions as for 1160 are met.

  590 DELETE 1630 if "CALL 1" is not required and the conditions as for 980 are met.

  360 DELETE 1630 if an insert routine for an indexed file is not required and the conditions as for 590 are met.

- If parts between two routines are not needed in exceptional circumstances, these lines should not be deleted, but REM inserted.

When employing the LINK technique, the DIM statements (20-60) of the program to be linked should also be given REM statements. This approach means that the branch addresses in the base part always refer to the same line numbers.

When using a printer, it must be opened on channel 12.
When entering a job in the spool file, channel 13 is used briefly by the base part.

21.9.2 Description of the individual routines

The variables additionally required by the base part are dimensioned prior to the first routine.
If the parts 990 to 1630 are deleted, REM can be inserted in lines 40 to 60.

Build TAMOS header and message lines

This routine is automatically called in line 1650. If the release and level are to be changed, this must be carried out in line 90.

TAMOS message output

Call for outputting a message from TF.PARAM

```
... GOSUB 150
Specification: T(1)  = Message number (2%)
```

Call for outputting a message with a user-defined text

```
... GOSUB 160
Specification: M7$   = Message
```

TAMOS logging

Call:

```
...GOSUB 200
Specification: M7$   = Entry text
```

TAMOS standard error handling

Call:

```
...IF ERR 0 GOSUB 280
No specifications.
```

Insert routine for indexed file

This routine enables a data record and the associated key to be inserted in directory 1.

Call:

```
... GOSUB 370
Specifications: I2  = Channel number
                M7$ = Key
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

The routine "Write data record" must be programmed separately. In order to do this, REM must be deleted in statement 400 and the first line of the routine "Write data record" entered. The relative record number is supplied in T(0).

Return from the insert routine:

- Record inserted correctly : RETURN 1
- File full : Logbook entry, message output and RETURN
- Record already exists : Message output and RETURN
- File error : Message output, logbook entry and program abort

Input preparation and checking routine

This routine enables numeric and alphanumeric values to be entered. The screen is prepared in accordance with the specifications of the command string.
When a ? is entered, all control character and symbol input possibilities are displayed on the screen.

If packing is required in the command string, it is also done with this routine.
If "@" (duplicate) is entered, the value duplicated is displayed on the screen.

Call:

```
...GOSUB 600
Specifications:   M7$ = Alphanumeric value to be duplicated
                  I0  = Numeric value to be duplicated
                  I$  = Command string
                  I2$ = Admissibility flag for control
                        information codes
                  I1$ = Control information codes (3 char-
                        acters each)
```

Result:

```
I0$ = Value entered
I0  = Numeric value entered
I1  = Control function number
```

When control information with a length of 3 bytes is entered, I0$ contains this control information, however.

I1$, I2$ and I$ are not changed. For this reason they need only be changed specifically in the case of several inputs.

Programming Examples

Enter TAMOS job in the spool file

Call:

```
...GOSUB 1000
Specifications: D7$  = Job description
                P7$  = LU/text or program name
                F7$  = Optional user information
                S7$  = Paper description
                S(0) = Job type
                S(5) = Delete code or RUN mode and job skip
                S(6) = Number of issues
                S(7) = Paper code
                S(8) = First print position
                S(9) = Last print position
                S(10)= Lines per page
                S(12)= First page number
```

Result:

```
Enter job in the spool file : T(1)=0
Spool file full             : T(1)=164
```

TAMOS communication from phantom to master

If the application program is executed in the phantom port, a message can be received and a response given at the master port by means of a branch to this routine.

Call with message from TF.PARAM:

```
...GOSUB 1180
Specification: T(1) = Message number in TF.PARAM
```

Call with user-defined text:

```
...GOSUB 1190
Specification: M7$  = Message
```

Result: Response in M7$.

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

Printer error routine

This routine tests errors 26 and 50. In the event of error 50, "PRINTER BUSY OR OFF-LINE" is output on the screen, and "PRINTER OFF-LINE OR OUT OF PAPER" in the case of error 26.
The program can be cancelled with CAN. In this case, "INTERVENTION BY OPERATOR" is output as the message and entered in the logbook. An input other than CAN results in the statement being repeated.

This routine can be run at a normal port and via the spooler, in which case the communication routine is also employed.

Call:

```
... IF ERR 0 GOSUB 1320  before printer opening
... IF ERR 0 GOSUB 1330  before statements for printing
```

If an error other than 26 or 50 occurs, a branch is made to the normal error routine.

Paper change under the spooler

The contents of S7$ (paper description) as well as "INSERTED Y/CAN" is displayed.

Call:

```
... GOSUB 1490
```

Result:

```
Master enters Y   : RETURN
Master enters CAN : Message output, logbook entry and pro-
                    gram abort
Otherwise         : Repetition of the display and input
```

Programming Examples

Transfer of the TAMOS area for the spooler and printer opening

The data provided by the spooler is transferred and a check is made for paper change. In the case of paper change, a branch is made to the routine described above. The printer is then opened with the parameters "page length" and "from/to print position". If an error occurs during opening, a branch is made to the printer error routine.

Call:

```
... GOSUB 1550
```

Result: Under normal circumstances: RETURN
With CAN at paper change or OPEN error: program abort

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

21.9.3 The base part

```
10 REM01 Program name
20 DIM H7$(122),V7$(4),L7$(126),M7$(55),I$(13),IO$(50),
   I1$(18),I2$(8)
30 DIM 4%,IO,1%,I1,I2,2%,T(3),N1,N2,N5,N6
40 DIM D7$(20),P7$(18),S7$(25),F7$(25),S2$(8)
50 DIM 1%,S(12)
60 REM SKIP STANDARD ROUTINE*****************************
70 GOTO 1640
80 REM OUTPUT SCREEN*************************************
90 LET V7$="400/"
100 LET T(0)= SPC 6          /* PORT NO.
110 CALL 3,T(0),H7$          /* READ FROM PORT'S COMMON AREA
120 PRINT 'CS';'SB';V7$;H7$(5,66);TAB(71,0);H7$(67,74);
    TAB(0,24);H7$(75,89);
130 RETURN
140 REM OUTPUT MESSAGE***********************************
150 N1=3                     /* READ, DISPLAY, LOG MESSAGE
152 N5=15                    /* TABLE BASE
154 N6=T(1)                  /* MESSAGE NUMBER
156 S2$="TF.PARAM"
160 CALL 84,N1,N2,M7$,S2$,N5,N6
165 IF NOT N2 GOTO 170
166 PRINT TAB(0,24);"STATUS:";N2;  /* ANALYSE STATUS MESSAGE
170 SIGNAL 3,20
180 RETURN
270 REM ERROR HANDLING***********************************
280 IF SPC 8>97 RETURN -1
290 IF ERR 0 GOTO 350
340 LGERR                          /* LOGBOOK ENTRY
350 CHAIN "TA.ABO"
360 REM INSERT RECORD************************************
370 LET I1=2
380 SEARCH #I2,1,0;M7$,T(0),I1     /* FIRST FREE RECORD
390 IF I1 GOTO 550                 /* NOT THERE
400 REM ON I2 GOSUB ... WRITE DATA RECORD SUBROUTINE
410 LET IO=T(0)
420 SEARCH #I2,4,1;M7$,T(0),I1     /* INSERT KEY
430 IF I1=0 RETURN 1               /* OKAY
440 IF I1=1 GOTO 490
450 LET M7$="CHANNEL, ERROR "
460 LET M7$(14)=I2+I1/10
470 GOSUB 160                      /* MESSAGE OUTPUT
480 GOTO 340                       /* LOGBOOK AND ABORT
490 LET T(1)=263                   /* RECORD ALREADY EXISTS
500 GOSUB 150                      /* MESSAGE OUTPUT
```

Programming Examples

```
510 LET I1=3
520 SEARCH # I2,1,0;M7$,I0,I1 /* DEALLOCATE RECORD
530 IF I1 GOTO 450            /* FILE ERROR
540 RETURN
550 IF I1<>3 GOTO 450
560 LET T(1)=459
570 GOSUB 150
580 GOTO 200
590 REM CALL 1*****************************************
600 LET V7$=I$
610 LET I1=0
620 LET I0$=M7$
630 IF I$(1,2)="32" IF CHR I0<0 GOSUB 890
640 CALL 1,I1,I$,I0$,I0,L7$
650 LET I$(1,2)=V7$
660 LET M7$=I0$
670 INPUT 'BP',I0$
680 IF I0$<>"?" GOTO 770
690 PRINT  TAB (17,24);'CFF'; TAB (17,24);
700 IF I2$(1,1)="0" PRINT "@ /";
710 IF I2$(2,2)="0" PRINT "↑ /";
720 FOR I2=3 TO LEN I2$
730   IF I2$(I2,I2)="0" PRINT I1$(I2*3-8,I2*3-6);"/";
740 NEXT I2
750 PRINT 'BS';'BP';
760 GOTO 670
770 IF P$(2,2)="1" LET I0=LEN I0$
780 CALL 1,I1,I$,I0$,I0,I1$,I2$,L7$
790 IF I1=-1 GOTO 610
800 IF I$(2,2)="0" IF I1=0 CALL 60,I0$,I0$
810 ON I1 GOTO 840
820 LET I$(1,2)=V7$
830 RETURN
840 LET I$(1,1)="3"
850 IF I$(2,2)="2" IF CHR I0<0 GOSUB 890
860 LET I2=0
870 CALL 1,I2,I$,M7$,I0,L7$
880 GOTO 820
890 LET L7$="-",L7$(1,12)," #.######"
900 LET I2=I$(7,8)
910 IF I$(12,12)="0" LET I2=I2-1
920 LET T(0)=I$(11,11)
930 LET M7$=I0 USING L7$(14-I2,15+T(0))
940 LET I2=I$(3,4)
950 LET T(0)=I$(5,6)
960 PRINT  TAB (I2,T(0));M7$;
970 LET I$(1,1)="0"
980 RETURN
990 REM ENTER JOB IN SPOOL FILE***************************
1000 OPEN # 13,"TF.PORT"
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Programming Examples

```
1010 READ # 13,0,22+SPC6*44;T(2),S(1),T(3),S(2),S(3),S(4);
1020 CLOSE # 13
1030 OPEN # 13,"TF.SPOOLQUEUE"
1040 FOR S(11)=0 TO  CHF 13-1
1050   READ # 13,S(11);M7$
1060   IF M7$(1,1)=" " GOTO  1100
1070 NEXT S(11)
1080 LET T(1)=164
1090 GOTO 1150
1100 LET T(1)=0
1110 LET S(11)=0
1120 LET T(2)=(SPC 2-INT(SPC 2/24)*24)*3600+INT(SPC 3/10)
1130 IF  LEN D7$<3 LET D7$=H7$(5)
1140 WRITE # 13,-2;D7$,S(0),T(2),S(1),T(3),S(2),S(3),S(4),
  P7$,S(5),S(6),S(7),S7$,S(8),S(9),S(10),S(11),S(12),F7$;
1150 CLOSE # 13
1160 RETURN
1170 REM COMMUNICATION PHANTOM TO MASTER*****************
1180 READ # 1, INT (T(1)/10)+15, FRA (T(1)/10)*510;M7$;
1190 LET T(0)=- SPC 6                  /* NEGATIVE TASK NO.
1200 CALL 2,T(0),T(0),M7$
1210 READ # 1,14,137;M7$(38);
1220 LET M7$="←377←←207←←376←←211←←376←←221←B←230←",
     M7$(38),"←376←←212←"
1230 IF ERR 0 GOTO 1290
1240 CALL 4,M7$                        /* OUTPUT SPOOL STATUS
1250 SIGNAL 3,250
1260 CALL 3,T(0),T(0),M7$              /* FETCH RESPONSE
1270 IF T(0)=- SPC 6 GOTO 1210         /* NO RESPONSE
1280 RETURN
1290 IF  SPC 8<>38 GOTO 290
1300 GOTO 1250
1310 REM PRINTER ERROR***********************************
1320 IF  SPC 8=50 GOTO 1460            /* PRINTER BUSY
1330 IF  SPC 8<>26 GOTO 280            /* STANDARD ERROR
1340 LET T(1)=183
1350 IF  SPC 6=1 GOTO 1430             /* WITH SPOOLING
1360 GOSUB 150                         /* MESSAGE W/O SPOOLING
1370 INPUT 'SF', TAB (70,24),M7$
1380 IF M7$(1,3)<>"CAN" RETURN -1
1390 LET T(1)=123
1400 GOSUB 150
1410 GOSUB 200                         /* LOGGING
1420 CHAIN "TA.ABO"
1430 GOSUB 1180                        /* MESSAGE VIA SPOOLER
1440 IF ERR 0 GOSUB 1320
1450 GOTO 1380
1460 LET T(1)=182                      /* PRINTER BUSY
1470 GOTO 1350
```

Programming Examples

```
1480 REM PAPER CHANGE UNDER SPOOLER**********************
1490 LET M7$=S7$," INSERTED ? Y/CAN"
1500 GOSUB 1190
1510 LET S(7)= ABS (S(7))
1520 IF M7$(1,1)="Y" RETURN
1530 IF M7$(1,3)="CAN" GOTO 1390
1540 GOTO 1490
1550 LET T(0)=SPC 6          /*TRANS. TAMOS AREA + PRINTER OPEN
1560 CALL 3,T(0),H7$,F7$,S7$,S(8),S(9),S(10),S(7),S(6),S(12)
1570 IF S(7)<0 GOSUB 1490
1580 LET M7$=(S(8)+1000)*1000000+S(9)*1000+S(10)
1590 LET L7$=M7$(3)
1600 IF  ERR 0 GOSUB 1320
1610 OPEN #12;L7$,"$LPT"
1620 IF  ERR 0 GOSUB 280
1630 RETURN
1640 REM START OF PROGRAM*********************************
1650 GOSUB 90
```

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

22 Error handling

This chapter explains the meanings of the BASIC error numbers and describes possible action to be taken to recover errors.
It also contains a similar description for workstation errors in as far as these can be caused by a BASIC program.

22.1 BASIC errors

The errors marked with * cannot be intercepted with an IF ERR 0 statement. If an IF ERR 0 statement is issued, these errors result in a program abort.

Error	Meaning	Possible action or explanation
1	Syntax error	Check the syntax of the statement
2	Illegal string operation	Dimension string/change statement
3*	Storage overflow (program too large)	Split program up into segments or increase partition size by means of SAVE.
4	Format error	Check string literals
5	Illegal character	Check statement
6*	No such line number	A non-existent line number is referred to in a command, a GOTO or GOSUB statement.
7	Renumber aborted by "ESC", program is lost	Only possible to re-enter program or reload from disk
8	Too many variable names, limit is 340	Combine numeric variables in vectors
9	Unrecognizable word	Check the statement causing this
10	Line number, "RUN" is illegal before initial run	Enter RUN without preceding line number

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Error Handling

Error	Meaning	Possible action or explanation
11	Incorrect parentheses closure	Check the number of parentheses
12	Program is list/copy protected	Inform manager
13	Numeric value exceeds 9.9999999999999E+62	Calculation not possible
14	Out of data	Check DATA statements
15	Arithmetic overflow (such as division by zero)	Division by 0 or value can no longer be represented
16*	GOSUBs nested too deep	Restrict subroutine nesting to 32 levels
17	RETURN without GOSUB	Check GOSUB and RETURN statements
18*	FOR/NEXT loops nested too deep	Restrict FOR/NEXT loop nesting to 8 levels
19*	FOR without matching NEXT	Check FOR and NEXT statements
20*	NEXT without matching FOR	Check FOR and NEXT statements
21	Expression too complex (too much function nesting)	Simplify expression by splitting into 2 statements
22	Not enough disk blocks for swap-out	Carry out CLEANUP on the system disk
23	Array size exceeds initial dimensions	Deallocate matrix with DEALLO beforehand
24	Only one dimension allowed for a string variable	Use DEALLO
25	String or array not dimensioned	Insert DIM statement

Error Handling

Error	Meaning	Possible action or explanation
26	Logical unit not active	Inform manager
27*	Syntax error in user-defined function	Check DEF statement
28	Subscript, channel # or signal value out of range	Check statement causing the error
29	Illegal function usage	Call function in separate statement
30	User function not defined	DEF FN statement missing or occurs after the user function call
31*	User functions nested too deep	Restrict nesting depth to 5
32	Matrices have different dimensions	Check dimensioning
33	Operand is not a matrix	Dimension matrix
34	Dimensions are not compatible	Check dimensioning
35	Matrix is not square	Check dimensioning
36	Called subroutine not in storage	Incorrect CALL or SOFTSUB number, otherwise inform manager
37	Expression in operand for CALL	Store the value of the expression in a variable by means of a LET statement
38	Error detected by called subroutine	See "CALL Statements"
39	Formatted output exceeds buffer size	Divide output between two statements
40	Channel in use	Select other channel number

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Error Handling

Error	Meaning	Possible action or explanation
41	Illegal file name	Check name syntax
42	File not found	Inform manager (File on another logical unit?)
44	Not a data file (can't open or replace)	Check file type
45	File is read protected	Inform manager
46	File is write protected	Inform manager
47	Not enough free blocks on logical unit	Select another logical unit
48	Not enough blocks on your account	Inform manager
49	Channel not open	Incorrect channel number/OPEN # statement missing
50	File is open elsewhere	File locked, try again later
51	Illegal record number	End of file?/change record no.
52	Record not written	End of file?/change record no.
53	Illegal item number	Change item number
54	Item types don't match	Change output list
55	Statement is illegal from keyboard	-
56	Can't dump an empty program	Load program
57	String variables cannot be re-dimensioned	Delete dimensioning
58	Error in format string	Check format string

Error Handling

Error	Meaning	Possible action or explanation
59*	RUNMAT and/or SOFTSUBC not in system	Inform manager
60	Too many numbers entered for input	Re-execute program
61	Matrices have different number types	Compare number types in the DIM statement
62	Signal buffer is full or no such port	Check SIGNAL statement/issue later
63	Command is illegal in LOAD mode	Delete command from text file
64	Line number missing in LOAD mode	Insert line number in text file
65	File name in use for a different type file	Select other file name
66	File name in use (file being built or replaced)	Select other file name
67	File name in use and no "!" supplied	Append "!" or select other name
68	File name in use by a different account	Select other file name
69	File is a processor or driver	Check file name
70	Read error/read or write error on floppy disk	Repeat process
71	File is not a peripheral driver	Check file name
72	Overflow CHAIN string	Include part of the command string in a CHAIN statement in the chained segment

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Error Handling

Error	Meaning	Possible action or explanation
73	Illegal parameters/driver not active	See ENV # statement/adapt configuration with SYSMOD/common buffer pool not configured/ status test omitted
74	Error in communication	See "The Programmable Line Controller" ("Data Communications" manual)
75	Not an accessible BASIC program	Not a BASIC program file or read-protected
76*	Partition too small in LINK	Increase partition specification of the start segment
77*	Variable name not defined in linked segment	Dimension variable in the linked segment
78	BASIC statement larger 254 characters	Rephrase statement
79	No free channel available	-
80	Variable name doesn't exist	-
81*	Program is larger than active file	Set active file size
82	Indexed file: attempt to write over record boundary	Shorten file entry or reorganize file
83	Error with a GLOBAL statement: 1. DEALLO for global variables 2. DIM statement between GLOBAL statements 3. GLOBAL statement with implicit index	 Declare variables as local (DIM) or replace DEALLO by CLEAR Place DIM after GLOBAL statements Only use an integer as the index

Error Handling

Error	Meaning	Possible action or explanation
84	Error on SNA data transmission command	See "Data Communications" manual
85	Different sequence of GLOBAL variables	Declare GLOBAL variables in the same sequence as before
86*	BASIC file is destroyed	Inform manager
87	Storage medium is write-protected	-
88	Illegal drive number	-
89	No such message number	-
90	No such message number	-
91	No such message number	-
92	No such message number	-
93	No such message number	-
94	No such message number	-
95	No such message number	-
96	No such message number	-
97	No decimal arithmetic driver active	Inform manager
98	Illegal value entered	Repeat program execution
99	ESCAPE or CTRL-C pressed	-
123	Data record locked	Address data record again later

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Error Handling

22.2 Workstation error messages

A latching key marked "S" and containing a red LED indicator is located on the upper right-hand side of the keyboard; the LED always lights up when the workstation detects an error.
The error can have been caused by the workstation itself or by the workstation printer.

If the "S" key is then pressed, the message line (bottom line) of the screen is cleared and

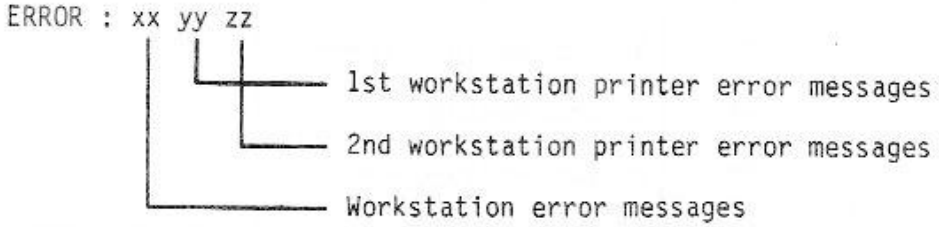
```
ERROR : xx yy zz
```

appears.

The numbers xx, yy and zz can be used to decode the error so that it can be eliminated if necessary.

The message is cleared as soon as the "S" key is pressed a second time. If the screen line contained information before the "S" key was first pressed, the information is retained and now becomes visible again.

The format of the error line is as follows:



Errors which can be caused by the application program are, for instance:

```
ERROR 05 00 00  LEAD-IN code without subsequent code

ERROR 06 00 00  Subsequent code after LEAD-IN > 240
```

Action:

Check the INPUT and PRINT statements for correct coding.

All other ERROR codes are described in the Error Messages manual.

Index

23 Index

	Section
A	
ABS	7.1.3.1
Access position	10.7
Access to cassette files	16.2
Access to contiguous files	11.3.2
Access to files	10.5.2
Access to formatted files	11.2.2
Access to indexed files	11.4.2
Access to printer files	12.1
Access to text files	11.5.2
Addressing	5.4.2, 5.4.3
ANALYSE	3.2, 4, 4.1
Arc tangent	7.1.1.2
Archive identifier	8.5.2
AREA	20.5
Arithmetic functions	7.1.3
Arithmetic overflow	5.1.1.3
ASCII table	9.8
Assignment	6.2.1
Assignment (matrices)	6.2.6.1
Assignment (softkeys)	9.9.5
Asynchronous	12.1
ATN	7.1.1.2
AUTOLN	4.2
B	
B file	1.3.5.1
Background output	9.5.8
Backspace	9.5.2
Bar code reader	9.7
Base part for application programs	21.9
BASIC	1.1
BASIC commands	4
BASIC error indication	18.1
BASIC errors	22.1
BASIC Optimizer	17.6.1
Batch optimization/deoptimization	17.6.4
BCD integer	5.1.1
BCR	9.7
BEL	9.5.12
Binary number	8.9, 8.10
Bit	5.1
BP	9.5.4

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Brightness control 9.5.27
BS 9.5.2
BUILD # 6.4.2.10
Business BASIC 1.1
Byte 5.1

C

CALL 6.5.4
CALL 2 ..... 19.6
CALL 3 ..... 19.6
CALL 4 ..... 19.11
CALL 35 ..... 8.19
* CALL 43 ..... 8.19-1
* CALL 44 ..... 8.19-2
CALL 66 ..... 8.28
* CALL 68 ..... 8.28-1
CALL 70 ..... 15.2.6
CALL 80 ..... 16.2
CALL 97 ..... 21.8
CALL 98 ..... 19.10
CALL statements 8
CALL subroutines 8
Calling BASIC functions 6.2.1.4
Calling user-defined functions 6.2.1.3
* Call the I/O macro 9.9.1
Card file 13.1
Card reader 13
CF 9.5.11
CFF 9.5.14
CH 9.5.15
CHAIN 6.3.10, 17.6.2
Changing paper in spooling mode 21.9.2
Channel concept 10.1.3
Character density 6.4.2.1, 12.1.2.6
Character replacement 8.14
Characteristic 7.2
Checking a string for date 8.26
Checking a string for digits 8.24
Checking a string for numeric contents 8.25
Checking input data 8.2
CHF 7.3.3, 10.7, 21.6
CHN 7.3.8
CHR 7.2.2
CKY 7.3.7, 9.6.3
Class of language elements 2.1
CLEAR 5.5, 6.2.1.7
Clear foreground 9.5.11
Clear foreground field 9.5.14

Index

Clearing latching keys..... 7.3.7, 9.6.3
Clearing the screen..... 9.5.10
CLOSE #..... 6.4.2.9
Closing a file..... 6.4.2.9, 10.5.3
Closing the magnetic tape cassette unit..... 16.2.3
Closing the magnetic tape unit..... 15.2.8
CNTRL #..... 3.1
Coarse blocks..... 11.4
Code conversion..... 8.29, 8.33, 14.1, 14.2, 16.2.1
Code expansion..... 17.6.4
Code tables..... 6.4.2.1, 8.28, 15.2.4.1
Combining values of different precision..... 6.2.1.2
COMMENT..... 4, 4.3
Common area..... 8.3, 8.4, 19.2, 19.6
Communication between phantom and master port 21.9.2
Compact printers 12
Compare programs 20.4
Comparing numeric values 6.3.5
Comparing strings 6.3.5
CON 6.2.6.9
Condition 3.2
Conditional execution 6.3.5
CONFIG 8.19
Constants 5
Contiguous file 11.3
Contiguous file to magnetic tape 21.3
Control variable 6.3.8, 17.2
Controlling the program flow 6.3
Conversion 4.1
Converting a binary number 8.10
Converting into a binary number 8.9
Converting the date 8.13
COS 7.1.1.1
CP 9.5.22
CR 9.5.5, 12.1.2.1
Creating a file 10.4
Creating a file directory 21.8
Creating contiguous (relative) files 11.3.1
Creating formatted files 11.2.1
Creating indexed files 11.4.1
Creating text files 11.5.1
CREF 4.2
Cross-references 20.1
CROSS-REFERENCES 4.15
CS 9.5.10
Current print line 12.1.3
Cursor 6.4.1, 9

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Cursor position 9.5.22
Cursor positioning 9.5.1
Cursor to marked position 9.5.4
Cursor to next foreground field 9.5.13
Cut-form processing 12.1.2.2

D

D file 1.3.5.1
Daisywheel printer 12
Dark input 9.5.16
DATA 6.2.2
Data channel 6.4.2.1, 6.4.2.9, 10.1.3
Date conversion 8.13
DC 9.5.20
Deadlock 10.3
DEALLO 6.1
Deallocating a data record 21.1.3
Deallocating variables 6.1.2, 6.1.3
DEBUG 4.5, 18.5
DEBUG mode 18.5
Decimal number 2.2
Declaration 5.3, 6.1
DEF 6.1, 6.1.4
Define window 9.5.23
Defining user functions 6.1.4
Degree (angle) 7.1.1.1
DELETE 4.6
Delete-protected 12.1.1
Deleting a key 21.1.3
Deleting a line 9.5.6
Deleting files 6.4.2.11, 10.6
Deleting statements 4.6
Delimiter 5.4.3
Delimiter processing 6.2.1
Deoptimization 4.9, 17.6.4
DET 7.3.4
Device channel program 6.4.2.1
Device functions 9.5.28
Device input 9.7.1
Device parameters 8.32
DF 9.5.18
DIM 5.3, 5.3.1, 6.1, 6.1.1, 17.6.2
Dimensioning 5.3
Direct execution of statements 18.4
Directory processing 6.4.2.8
Disk 10

Index

Disk archive identifier 8.5.2
Disk configuration 8.19
Disk requirements for indexed files 11.3
Display functions 9.5, 9.5.28
Display memory 9.5.21
Display workstation 9
Displaying messages 8.5.1
DLC 9.5.19
DM 9.5.21
Drawing in forms 12
Driver 6.4.2.1, 12.1.1
Driver file 3.1
Driver name 2.2
DRK 9.5.16
DUMP 4.7
Dumping programs 4.7
Dumping variables 18.9
Duplicate characters 9.5.20
Duplicate field 9.5.18
Duplicate limited characters 9.5.19
DW 9.5.23

E

EDIT 4.8
Ejecting forms 12.1.2.1
Ejecting the cassette 16.2.3
Elongated printing 12.1.2.5
END 6.3.13
End symbols 2.1
End-of-file handling 11.2.2, 11.5.2
End of line without line feed 12.1.2.1
ENV # 6.4.2.2
EOF pointer 6.4.2.1, 7.3.3, 11.3.3
EOF 1/EOF 2 ..... 15.1
EOL 12.1.2.1
EOV 1/EOV 2 ..... 15.1
Erasing the main memory partition 4.13
ERROR 18.1
Error handling 15.3, 16.3, 19.7, 22
Error interception 6.3.6
Error texts 4.10
ESC 18.8
Execute program 4.17
Execution log 18.5
EXP 7.1.2.2
Explicit dimensioning 5.3.2
Exponent 5.1.1.2

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Extended code conversion 8.28

F

FF 12.1.2.1
* Field description file ..... 9.9 ff
* Field description parameters 9.9.3
* Field description record 9.9.3
Field format 6.4.2.7
Field processing 11.2.2
File 3.2
File access 10.5.2
File closing 6.4.2.9, 10.5.3
File creation 6.4.2.10, 10.4
File deletion 6.4.2.11, 10.6
File header 8.39, 11
* File labels 15.1
File locks 6.4.2.1
File name 10.1.5
File name checks 8.38
File opening 6.4.2.1
File organization 10.1.1
File organization types 11.1
File sharing 10.3
File size 10.7
Filler 11.2.1
Fine blocks 11.4
FLN 7.3.9
Floating-point numbers 5.1.1.2, 7.2
FOR 6.3.8
FOR/NEXT loop 17.2
Foreground output 9.5.9
Form control 12.1.2.1, 19.9.1
Form feed 12.1.2.1
Format of variables 5.1.1
Format switch 5.3.2, 6.2.2
Formatted file 11.2
Formatting the input 8.2
Forward-spacing/backspacing the tape 15.1.7.2
Functions 7

G

Gap 15.1, 16.1
GET # 3.1
GLOBAL 5.3, 5.3.1, 6.1, 6.1.3, 17.6.2
Global data area 17.6.2
GOSUB 6.3.2
GOTO 6.3.1

Index

H

Handling errors 15.3, 16.3, 19.7, 22
* Hard key labels 9.9.2.2
HASH 4.9
Hash total 4.9, 20.4, 20.5
Hazeltine 8.20
HDR1/HDR2 ..... 15.1
Header block 10.2
Header extension block 11.2, 11.5
Header line 19.3, 19.6.1, 21.9.2
HELP 4.10, 18.1
Hexadecimal constants 5.5
Hexadecimal output 9.5.17

I

ID card unit 9.7
IDCU 9.7
Identity matrix 6.2.6.10
IDN 6.2.6.10
IF 6.3.5
IF ERR 0 ..... 6.3.6, 18.2
IF ERR 0 /* 6.3.6
IF ERR 0 REM 6.3.6
IF ESC 6.3.7, 18.3
Implicit dimensioning 5.3.2
IN 12.1.2.1
Index area 11.4.2
Indexed file 11.4
Indexed file disk requirements 11.3
Industrial date 8.13
INIT # 3.1
INPUT 6.4.1.1
Input formatting 8.2
INPUT mode 6.4.1.1
Input preparation 8.2
INPUT routine 21.9.2
Input/output buffer 6.4.1.1
Input/output statements 6.4
Inputs via card reader 13.1.2
Inputs via keyboard 6.4.1.1, 6.4.1.2, 9.1
Inserting a key with a data record 21.1.2
Inserting a line 9.5.7
Inserting in indexed files 21.9.2
INT 7.1.3.3
Integer 2.2

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Interactive programming 1.1
Intercepting errors 6.3.6
Intermediate code 17.6.1
Interrupting execution 6.3.12
Inter-task communication 6.5.2
Inverse matrix 6.2.6.5
Inverted string 8.11
* IOM field description 9.9.4
* I/O macro ..... 9.9.
* I/O macro examples 9.9.6
IXR 7.2.3

J

Job entry 21.9.2
Job spool file 19.8.1
Job spooler 19.1, 19.8

K

Key 6.4.2.8, 11.4
KEY 7.3.5, 9.6.1
Key deletion 21.1.3
Key directory 11.4, 11.4.2
Key directory processing 6.4.2.8
Key insertion 21.1.2
KILL 6.4.2.11

L

Labels (file) 15.1
* Labels (key) 9.9.2.2
LD 9.5.6
LEN 7.3.2
LET 6.2.1
Letters 2.2
LFB 12.1.2.1
LGERR 6.5.5
LI 9.5.7
Line 11.4
Line cross-references 20.1
Line deletion 9.5.6
Line feed 12.1.2.1, 9.5.5
Line feed backwards 12.1.2.1
Line insertion 9.5.7
Line printer 12
Line spacing 6.4.2.1
LINK 6.1, 6.3.11, 17.6.2

Index

Linking logical expressions 6.3.5
LIST 4.11
Listing the program 4.11
LKY 7.3.6, 9.6.2
LOAD 4.12
Load remarks 20.3
Loading 1.3.5.2
Loading a program from a text file 4.12
Local data area 17.6.2
Locking a data record 10.3
LOFF # 3.1
LOG 7.1.2.3
Logbook 19.5
Log file 21.6
Log file entry 21.9.2
Logical expressions (linking) 6.3.5
Logical IOCS 10.1
Logical unit 10.1.6
LON # 3.1
LU number 8.5.2, 10.1.6

M

Machine code subroutines 8, 17.4
Machine language 1.1
Magnetic disk 10
Magnetic tape 15
Magnetic tape buffer size 15.2.9
Magnetic tape cassette 16
Magnetic tape cassette programming 8.33
Magnetic tape interface 15
Magnetic tape programming 8.29
MAN ..... 7.2.1.
Mantissa 5.1.1.2
Mark cursor position 9.5.3
Masks 6.2.1.5
Master 20.4
Master blocks 11.4
MAT 6.2.6
MAT INPUT 6.4.1.2
MAT PRINT 6.4.1.4
MAT READ 6.2.7
MAT READ # 6.4.2.6
MAT WRITE # 6.4.2.7
Mathematical functions 7.1
Mathematical hierarchy 6.2.1.1
Matrix 5.1.2, 6.2.6
Matrix addition 6.2.6.2
Matrix multiplication 6.2.6.3
Matrix subtraction 6.2.6.2
Message file 19.5

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Message line 19.3
Message output 21.9.2
MESSAGES 19.5
Metalanguage 2.1
Microlines 12.1.2.3
MIV 6.2.6.6
Mixture of B and D files 17.6.4
MP 9.5.3

N

Natural logarithm 7.1.2.3
Needle printer 12
NEW 4.13
NEXT 6.3.9
No record lock 6.4.2.3
No typewriter keyboard 9.5.24
Non-interceptable BASIC errors 18.2
Non-interceptable errors 22.1
NOT 7.1.3.4
NTY 9.5.24
Number manipulation 7.2
Numeric constants 5.1.3
Numeric expressions 2.2, 6.2.1.1
Numeric functions 7
Numeric operators 2.2
Numeric variables 2.2

O

Object code 1.1
OCR-A character field 9.7.2.1
OCR-A hand-held reader 9.7
Octal code 5.2.2
ON 6.3.4
OPEN # 6.4.2.1
Opening a file 6.4.2.1, 10.5.1
Opening the cover for forms 12.1.2.1
Opening the magnetic tape cassette unit 16.2.1
Opening the magnetic tape unit 15.2.6
Opening the printer 12.1.1
OPN 12.1.2.1
Optimization 1.1, 4.9
Optimizer 17.6.1
Output of error texts 4.10
Output to display 6.4.1.3, 6.4.1.4, 9.3
Output to printer 12.1.2

Index

P

Packing 8.22
PAF 8.16, 19.4
Page feed 12.1.2.1
Paper cassettes 12.1.2.4
Paper feed 12.1.2.3
Paper tape 14
Paper tape file 14.1
Paper tape punch 14.2
Paper tape reader 14.1
* Parameter file ..... 9.9 ff
Parameters in the common area 19.6
Partition 17.1
PARTRENUM 4.14
PCA 8.16
Phantom port 19.10
Physical unit 10.1.6
PLIST 4.15
Pointer table 8.28
* Pre-assigned value 9.9.2.1
Pre-input buffer 6.4.1.1
PRINT 6.4.1.3
Print buffer 6.4.2.4, 6.4.2.5
Print positions 6.4.2.1
Print spooler 19.9
PRINT # 6.4.2.5
Printer driver name 6.4.2.1
Printer errors 21.9.2
Printer name 12.1.1
Printers 12
Printing a text file 21.4
Printing programs 4.7
PRN 7.3.10
Processing cut forms 12.1.2.2
Processing delimiters 6.2.1
Processing indexed files (examples) 21.1
Processing key directories 6.4.2.8
Program abort handling 19.2
Program analysis 20.7
Program area 20.4
Program execution 4.17
Program group 20.4
Program interrupt 18.7
Program jobs 19.8.1
Program listing 4.15, 20.1
Program loop 6.3.8, 17.2
Program maintenance 20
Program name 2.2
Program optimization 17.6.4, 20.6
Program segmentation 17.5

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Program sequence 17.6.2
Program size 4.19, 17.6.4
Program start 19.2
Program structure 17
Programming examples 21
Programming the display 8.20
Pseudo wait 12.1.2
PUT # 3.1

R

Radian 7.1.1.1
RANDOM 6.2.5
Random number 7.1.3.6
Random number sequence 6.2.5
Random reading in indexed files 21.1.1
READ 6.2.3
READ # 6.4.2.3
Reading file information 8.39
Reading from files 6.4.2.3, 6.4.2.6
Reading in a card file 21.7
Reading the disk archive identifier 8.5.2
Reading the system date 8.41
Reading/writing a block 15.2.7.3, 16.2.2.3
Record lock 6.4.2.4, 10.3
Record pointer 10.1.2
Relational operators 2.2
Relative file 11.3
Release 20.4
Release code 12.1
Release level 20.5
REM 6.5.1.1
REM 01 statement 4.2.1
Remark files 20.3
Remarks 6.5.1
RENUMBER 4.16
Renumbering 4.16
Repair program 8.16
Replacement string table 8.28
Replacing characters 8.13
Representable values 5.1.1.2
Representation of alphanumeric data 5.2
Representation of numeric data 5.1
Restart program 8.16
RESTOR 6.2.4
RETURN 6.3.3
RND 6.2.5, 7.1.3.6
RUN 4.17, 18.6
Run-time optimization 17.6

Index

S

S key 22.2
SAS devices 9.5.28
SAVE 4.18
SAVE,B, ... 4.18
SAVE,D, ... 4.9, 4.18
SAVE?! 1.3
SB 9.5.8
Scalar multiplication of a matrix 6.2.6.4
SCR 9.7
Screen layout 19.6.1
Screen preparation 8.2
SEARCH # 6.4.2.8
Searching for a substring 8.11
Segment 17.6.2
Segmentation of programs 17.5
Selector (inclusion) 19.1
Separators 9.5.23
Sequential files 11.6
Sequential processing (indexed files) 21.1.4
Sequential reading 6.4.2.3, 6.4.2.6
Sequential writing 6.4.2.4, 6.4.2.5, 6.4.2.7
SF 9.5.9
SGN 7.1.3.2
Sign function 7.1.3.2
Signal emitter 9.5.12
Signal node queue 6.5.2
SIGNAL 1 ..... 6.5.2.1
SIGNAL 2 ..... 6.5.2.2
SIGNAL 3 ..... 6.5.3
SIN 7.1.1.1
Single-step mode 4.5
Single-step test 4.5, 18.5
SIZE 4.19
Size of a file 10.7
* Softkey assignment 9.9.5
* Softkey labels 9.9.2.2
Sorting 8.27
Source code 1.1
SPC 7.3.1
Special functions 7.3
Spool file 21.9.2
Spooler control 19.9.2
SQR 7.1.2.1
Square root 7.1.2.1
Standard code tables 8.29, 8.33
Standard error handling 21.9.2
Standard mask 6.2.1, 6.4.1.3
Standard parameters 19.6.1

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

Standard subroutines 19.1
Status test 15.2.4.2
Step 6.3.8
STOP 6.3.12, 18.7
Stop conditions 4.17
Storing 1.3.5
Storing a program in a text file 4.7
String comparison 6.3.5
String expressions 2.2, 6.2.1.6
String inversion 8.12
String length 7.3.2
String literals 5.2.2
String processing 8.35
String variables 2.2, 5.2.1
Subroutine branch 6.3.2
Subroutine technique 17.3
Subroutines in machine code 8, 17.4
Substring 8.10
Summary of CALL statements 8.1
Summary of commands 3.2
Summary of statements 3.1
Swipe card reader 9.7
Symbolic language 1.1
Syntactical expression 2.1
Syntax check 1.1
Syntax elements 2
System date 8.13, 8.4.1
System disk 10.1.6
System name 8.32

T

TA.ABO 19.4
TA.END 19.4
TA.NCO 19.4
TAB 9.5.1
TAMOS flag 8.16
TAMOS interfaces 19
TAN 7.1.1.1
Tape mark 15.1, 16.1
TB 9.5.13
Test run 18.6
Testing 4.20
Testing aids 18
Testing latching keys 7.3.6, 9.6.2
Testing release keys 7.3.5, 9.6
Text file 11.5
Text file job 19.8.1
TF.LOGFILE 19.2
TF.PARAM 19.2

Index

TF.PORT 19.2, 19.8.1
TF.SPOOLQUEUE 19.8.1
TRACE 4.20, 18.9
Transcendental functions 7.1.2
Transfer from a string 8.31
Transfer from the common area 8.4
Transfer to a string 8.30
Transfer to the common area 8.3
Transposed matrix 6.2.6.7
Trigonometrical functions 7.1.1
TRN 6.2.6.7
TY 9.5.25
Typewriter keyboard 9.5.25

U

UHL 15.1
UNCOMMENT 4, 4.21
Unconditional branch 6.3.1
Unload remarks 20.2
Unpacking 8.23
Updating the system date 8.41
User-defined functions 6.1.4
User information 8.16
UTL 15.1
UVL 15.1

V

Validity checks for file names 8.38
Values of different precision 6.2.1.2
Variable 5
Variable name 5.3
Vector 5.1.2
VOL 1 ..... 15.1

W

Wait status 6.5.3
Word 5.1
Workstation errors 22.2
WRITE # 6.4.2.4
Writing the disk archive identifier 8.5.2
Writing to files 6.4.2.4, 6.4.2.5, 6.4.2.7

© Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Index

X

'X' 9.5.17

Z

ZER 6.2.6.8
Zero matrix 6.2.6.8

8

8-bit transfer 6.4.1

$

$PTPA 14.2
$PTRA 14.1

/

/* remarks 3.1, 4.3