NIXDORF
COMPUTER

Nixdorf 8870

Program Tuning

Edition: 01.07.85 Order No. 34698.00.4.93

Modifications Sheet

Modifications sheet

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

First edition: Ø1.Ø7.1985

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 module, please send your written comments to the following address:

NIXDORF COMPUTER AG
Abt. ZSI
Fürstenallee 7
D-479Ø Paderborn

Contents

Contents

	Introduction

1 Introduction

Unfortunately, the past has shown that BASIC application programs have often been written without regard to the speed at which they run and the system load that they cause.

The result is that the system installed at the user's premises does not achieve the necessary performance and easily becomes a problem. This weak performance is particularly noticeable when inefficient programs are in progress at several ports.

To remedy this situation, this document should be given to application programmers before they start programming so that they can use it for reference in their work.

Acceptance testing of software products should not only concentrate on program functionality in future but should also take throughput aspects into account.

This document deals primarily with tuning of application programs. These tuning measures must involve:

- Program flow (organization of program and file system)
- Reduction of program run-time by efficient programming
- Use of system features that increase performance
- Utilization of the simultaneity in the system

Chapter 2 of this document deals with system performance, throughput and load in general. Chapter 3 gives hints on program structuring. Chapters 4 and 5 give the timing of certain BASIC statements and CALLs which are used as a basis for hints on efficient programming.

	System Performance

2 System performance

Just like all other computers, the Nixdorf 887Ø consists of a number of hardware and software components whose interplay under program control constitutes the actual performance of the system. If any major system component becomes a bottleneck (high workload), the performance of the entire system is impaired although the other parts of the system are capable of greater performance (i.e. they are not working at full capacity). The distribution of the workload is very uneven in these cases.

The system can achieve a good level of performance when all components that affect performance have roughly the same workload.

The workload placed on the individual system components is largely determined by the application programs which are used. This means that a program, which puts an unbalanced workload on a resource owing to poor organization or programming, reduces the performance of the entire system.

For these reasons, it is necessary to think about the organization before implementing a program and to choose the BASIC statements very carefully when writing the program.

System Performance

2.1 Throughput and response time

The system load, that is generated by an application program and the way it is distributed, can be determined for each individual port by means of the software monitor "SPE" (System Performance Evaluator) which is stored on the system disks.

The resultant profiles reveal that by far the greatest part of the CPU's performance is consumed by the user (interpretation of the application and arithmetic); for instance about 5Ø% to 6Ø% in interactive applications from the COMET package. When applications are programmed in a favourable way for the system, this will have a positive effect on the response time and throughput of the entire system.

In batch applications, the load on the disk is generally the critical factor. This does not mean, however, that intricate arithmetic and string processing can be used indiscriminately in such applications. Every intricate or even unnecessary BASIC statement is a drain on the system's performance and is detrimental to the entire system.

Hints on system tuning, which consequently involves the configuration, can be found in the "System Tuning" manual (order no.: 34697.ØØ.7.93).

System Performance

2.2 System load through the BASIC interpreter

The user load that has been determined by means of the software monitor mainly arises from the interpretation of BASIC statements, including the arithmetic ($DEC14, $DEC18 or $DEC18H drivers) and CALLs (Discsubs).

In principle, this load can be reduced in two ways:

- Improvement of application programs
- Use of an improved BASIC interpreter

To reduce the load on the CPU, a new interpreter - the BASIC optimizer using RUNO and RUNC - was released with NIROS 5.1.

When the optimizer is used without converting the programs to the generation of fixed variable addresses, the user load is roughly halved. Due to this offloading of the CPU, system throughput is increased by 2Ø% to 3Ø% (measured with various COMET interactive applications in which only optimized programs were used). Converting the programs to the generation of fixed variable addresses achieves a further gain in throughput of about 5% to 1Ø%.

Hints on programming for the BASIC optimizer and on the conversion of existing programs can be found in the NIROS 5.1 release manual and in the NIROS 5.1 BASIC manual.

Hints on Program Structuring

3

3 Hints on program structuring

This chapter gives some hints on the favourable structure of program chains and individual program segments.

- All dimensioning should be done with regard to the generation of fixed variable addresses at the start of the program and not be preceded by other statements (apart from REM).

- Statements such as IF ERR and DEF should not be executed continually (at least not in loops) whenever possible.

- Remarks should be unloaded from the program.

- Function switches should, if possible, be designed with the ON statement and not with IF followed by GOTO or GOSUB.

- In applications which consist of several successive programs for reasons of space, an attempt should be made to concentrate the main loop(s) in just one program. Each program change means that the time slice is passed on to another user and increases the system overhead. Exceptional or special functions should be put in programs of their own in the event of space bottlenecks instead of spreading main loops over several programs.

- Control parameters and screen masks that are needed should be read once at the beginning and then stored in variables - even with LINKed programs - in order to avoid repeated accesses to disk.

Hints on Program Structuring

- In parameter-driven programs, the parameters should be stored in numeric variables, if space allows it, and be stored or read and tested as a vector instead of a control string because conditions with a single-indexed numeric vector are executed faster than with double-indexed strings.

  Not like this:

```
IF P$(2Ø,2Ø)="1" IF P$(14,14)="Ø" GOTO ...
```

  but like this:

```
IF P(2Ø)=1 IF P(14)=Ø GOTO ...
```

- Similar arithmetic expressions, which in the end keep computing the same value, should be avoided in loops. Programmers should target their style at minimizing the system load and not at just making the programs easy to read. This can be done by computing these expressions once before the loop and assigning them to non-indexed variables.

  Not like this:

```
FOR I=1 TO N
  IF A$(B+3,B+3)=X$(I,I) GOTO  ...
  IF B$(2*B+1,2*B+1)=Y$(I,I) PRINT ...
NEXT I
```

  but like this:

```
LET T=B+3
LET H=2*B+1
FOR I=1 TO N
  IF A$(T,T)=X$(I,I) GOTO  ...
  IF B$(H,H)=Y$(I,I) PRINT ...
NEXT I
```

Hints on Program Structuring

- Whenever possible, the control variable should not be indexed in a loop.

  Not like this:

```
FOR H(1)=Ø TO 999
  ...
  LET A=...+H(1)*...
NEXT H(1)
```

  but like this:

```
FOR I=Ø TO 999
  ...
  LET A=...+I*...
NEXT I
```

  (If optimized programs which permit the generation of fixed variable addresses are used, indexing of tables with constants (e.g. A(1Ø)) is not disadvantageous in terms of execution time compared with non-indexed variables.)

Execution Times of BASIC Statements

4 Execution times of BASIC statements

This chapter lists the execution times of individual BASIC statements and gives hints on programming in order to improve execution time. The times that are listed refer to non-optimized programs. The measurements were taken with the 1537 CPU and the decimal arithmetic driver $DEC14 under NIROS 5.1.

4.1 LET statement

Statement	Framework	Time in ms
LET A$="123456789ø"	DIM A$(1ø)	ø.97 - 1.ø2
LET A$ = B$	DIM A$(1ø),B$(1ø)	ø.94 - ø.98
	DIM A$(1øø),B$(1øø)	5.2ø - 5.5ø
LET A$(1ø,11ø)=B$	DIM A$(11ø),B$(1øø)	6.øø - 6.3ø
LET A$ = I	DIM 1%,I I=ø	1.38 - 1.43
	DIM 2%,I I=ø	1.39 - 1.44
	I=99999	2.21 - 2.26
A$ =I USING "#####"	I=99999	2.7ø - 2.9ø
LET I = A$	DIM 2%,I A$="12345"	1.35 - 1.4ø
	DIM 4%,I A$="12345"	1.37 - 1.42
	DIM 2%,I	1.59 - 1.64
	A$="123456789ø"	
LET I = LEN A$	No. of bytes = 5	1.19 - 1.24
	No. of bytes = 1øø	1.9ø - 2.1ø
LET A = 99999	DIM 2%,A	ø.9ø - ø.95
LET A = B	DIM 2%,A,B B=9	ø.94 - ø.99

Execution Times of BASIC Statements

The execution time of these assignments is reduced by a factor of 2 to 2.8 when the BASIC optimizer is used.

In the case of indexed variables, the assignment takes about Ø.4 ms longer in non-optimized programs than without indexing. (In optimized programs with generation of fixed variable addresses, this only applies when the index is not a constant because here indexing with constants does not require any more time.)

4.2 Arithmetic

Statement	DIM	Values	Time in ms
LET A=B+C	2%,A,B	B=C=99999	1.44 - 1.49
LET A=B-C			1.51 - 1.56
LET A=B*C	2%,A		
	1%,B,C	B=C=999	3.6Ø - 3.8Ø
	2%,A,B,C	B=C=999	3.5Ø - 3.7Ø
		B=C=999999	4.5Ø - 4.8Ø
LET A=B*C	2%,A,B,C	B=999999, C=99	3.1Ø - 3.3Ø
LET A=C*B			4.5Ø - 4.8Ø
LET A=B/C	2%,A	B=999, C=2	4.8Ø - 5.ØØ
	1%,B,C	B=999, C=123	6.9Ø - 7.1Ø
		B=777, C=1234	7.ØØ - 7.2Ø
	2%,A,B,C	B=999999, C=99	3.5Ø - 3.7Ø
		B=999999, C=2	**6.2Ø - 6.5Ø**
		B=999999, C=12345	6.4Ø - 6.6Ø
	4%,A,B,C	B=99999999999999	
		C=2	1Ø.2Ø -1Ø.4Ø
		C=12345	7.ØØ - 7.2Ø
		C=1234567899Ø	6.6Ø - 6.8Ø

Execution Times of BASIC Statements

Hints:

- In arithmetic operations, the operands involved should be dimensioned with the same precision whenever possible.

- It is favourable for the execution time of a division if the dividend and divisor are in the same order of magnitude.

- In multiplication, the sequence of operands should be chosen so that the multiplier has as few significant positions as possible.

- Multiplication by 2 should be replaced by addition and, if possible, division by a constant should be replaced by multiplication by the reciprocal.

  Not like this:

```
LET A=2*W
LET B=7*(X/Y+1/4)
LET C=Z/8
```

  but like this:

```
LET A=W+W
LET B=(X/Y+.25)*7
LET C=Z*.125
```

- If space allows, no I% variables should be used because $DEC has to carry out a conversion to a floating point variable before every arithmetic operation. When software arithmetic is used for addition and subtraction, for instance, this results in an execution time which is longer by up to a factor of 2.

When the BASIC optimizer is used, the execution time for arithmetic expressions is reduced by a factor of 1.2 to 2.3, depending on the function and the precision of the variables.

Execution Times of BASIC Statements

4.3 Definition of functions

Statement	Function	ms
DEF	FNA(C)=C*B	Ø.Ø33
	FNA(C)=C*B/(A*B)+C+SIN((A*C)/1ØØ)	Ø.Ø33

The execution time of the DEF statement does not depend on the length and complexity of the function to be defined as only an entry is made in a table. The function is actually executed when it is called in an assignment in the program, e.g. by LET A = FNA(C). Despite the short execution time, it should be ensured that the DEF statement is only executed once at the start of the program for initialization purposes.

Execution Times of BASIC Statements

4.4 Branch and loop statements

Statement	Branch distance	Time in ms
GOTO ...	+/- 1 statement	Ø.Ø4 - Ø.Ø6
	+/- 8ØØ statements	Ø.14 - Ø.16
GOSUB ...	+ 8ØØ statements	Ø.16 - Ø.18
RETURN	- 8ØØ statements	Ø.15 - Ø.17
FOR I=Ø TO 1	Loop length =	1.53 - 1.57
NEXT I	2Ø statements	Ø.61 - Ø.65

The execution time for a branch statement depends on the branch destination, i.e. the number of BASIC lines that have to be skipped. Use of the BASIC optimizer has no effect on the execution times of branch statements.

A FOR statement is only executed once for the entire loop. The execution time of the FOR statement consists of:

- Initialization of the control variable
- Calculation of the end criterion
- Calculation of the step
- Searching for the associated NEXT statement (in certain exceptional cases)

The execution time of the NEXT statement consists of:

- Addition of the step to the control variable
- Comparison with the end condition
- If necessary, return to the start of the loop (statement after the FOR statement)

It is advisable to use a non-indexed 2% variable as the control variable.

Execution Times of BASIC Statements

4.5 IF statement

Statement	Condition	Time in ms
IF A ...	TRUE	Ø.45 - Ø.5Ø
	FALSE	Ø.47 - Ø.52
IF NOT A ...	TRUE	Ø.65 - Ø.69
IF A=Ø ...	TRUE	Ø.96 - 1.ØØ
IF A$(3,3)= " " ...	FALSE	1.6Ø - 1.7Ø
IF A$(3,3)= "3" ...	TRUE	1.69 - 1.74
IF A$(3,3)=B$(23,23) ...	TRUE	2.2Ø - 2.5Ø
IF A < A*B+C-1Ø5+C/B ...	TRUE	8.8Ø - 9.8Ø
IF B$(3,3)="3" IF B$(3,3)=A$(23,23) IF A+B=1Ø1 ...	All conditions TRUE	About 6.ØØ
IF ERR Ø ...		Ø.3Ø - Ø.32

Hints:

- "IF A = Ø ..." should, if possible, be replaced by the FALSE exit of the statement "IF A..." or at least by "IF NOT A...".

- If complex conditions are specified, the time needed to calculate the arithmetic expressions has to be added to the above-mentioned execution times.

Execution Times of BASIC Statements

- If several conditions are combined in a BASIC statement, it should be ensured that the condition that will probably not be satisfied more frequently is in first place or, if the frequency is not known, that the conditions needing less execution time are as far left as possible in order to minimize the average execution time for the overall statement. This is because interpretation of the statement is stopped at the first condition which is not satisfied and the program is continued at the next statement.

  Not like this:

```
IF A+B+C+D+E+F=X IF B=P LET Z= LEN A$
```

  but like this:

```
IF B=P IF A+B+C+D+E+F=X LET Z= LEN A$
```

- In successive IF statements, the same conditions should not be tested again and again.

  Not like this:

```
7ØØØ IF K=4 IF F9>2 IF F9<5 IF V2<>2  GOSUB 8964
7Ø1Ø IF K=4 IF F9>2 IF F9<5 IF V2<>2  GOSUB 8864
```

  but like this:

```
7ØØØ IF K=4 IF F9>2 IF F9<5 IF V2<>2  GOSUB 99ØØ
...
...
99ØØ GOSUB  8964
991Ø GOSUB  8864
992Ø RETURN
```

  Use of the BASIC optimizer reduces the execution time of IF statements, e.g. the statement "IF A=Ø ..." is reduced to Ø.54 - Ø.59 ms (factor 1.3 - 1.8).

Execution Times of BASIC Statements

4.6 Input/output statements

Input/output statements signify here the statements used to process the screen and keyboard.

4.6.1 PRINT statement

An input/output buffer of 254 bytes is created in the central unit for each configured port.

The characters to be output and the screen control characters are stored in the input/output buffer. This buffer is emptied, i.e. the data is sent to the workstation and displayed, when
- the buffer is about 75% full,
- a SIGNAL 3 statement follows,
- an INPUT statement follows or
- the program is exited (program end or change due to CHAIN or LINK).

There are various BASIC statements to output characters on the screen, e.g. PRINT, INPUT, CALL 1 and CALL 4. The most important of them is certainly the PRINT statement.

With the PRINT statement, the output data is stored in the buffer. However there is a check beforehand as to whether a preceding output is still active, i.e. whether the buffer is still in use. If so, there is a change of time slice and the characters are not output until the next time slice is allocated.

The time needed to output the contents of the buffer to the screen is determined by the transmission time (line speed). In most cases, the speed is set to 96ØØ baud. This results in a transmission time of about 1 ms per character, i.e. about 25Ø ms for a completely full buffer.

Execution Times of BASIC Statements

If the volume of data to be output exceeds the buffer size (e.g. to fill a whole 2øøø-character screen), the PRINT statement is executed in several sections. The buffer would have to be emptied 8 times in order to output 2øøø characters and there would be a change of time slice after each section is started. If a lot of workstations are active simultaneously on a large system, it can take a long time until the next time slice is allocated again to output the contents of the next buffer. In this way, the execution time of such a PRINT statement can be several seconds even though the transmission could be finished in 2 seconds as far as the line speed is concerned.

Example: line speed 96øø baud, 16 active workstations of the same priority, time slice 3øø ms. The interval between time slices for a user can be 16 x 3øø ms = 4.8 seconds at most during normal operations. Consequently, the output of 2øøø characters means a maximum waiting time of 8 x 4.8 seconds, i.e. 38.4 seconds. In practice this figure is not reached as a time slice is generally not used in full in interactive applications. One way of improving the response time here would be to define a shorter time slice (e.g. 1øø ms).

This example shows, however, that unnecessarily long outputs to the screen (e.g. clearing the screen by outputting 2øøø blanks) should be avoided.

Execution Times of BASIC Statements

The execution times given here for the PRINT statement are just the times for which the interpreter is active and do not contain transmission time or any waiting times that may occur owing to active outputs or a full output buffer.

PRINT	Comment	Time in ms
Without parameters	Line feed (like 'CR')	0.5 - 0.9
'CS';	Clear Screen Function is performed in the DWS	0.9 - 1.3
String variables:		

B$;	LEN B$=100 bytes	11.2 - 11.6
A$;	LEN A$= 10 bytes	1.8 - 2.3
TAB(0,0);A$;	With tabulation	2.7 - 3.2
TAB(69,24);A$;		3.1 - 3.6
TAB(0,0);A$;	Both in succession:	
TAB(69,24);A$;	Time per statement:	2.7 - 3.9
Numeric variables:		

B;	DIM 2%,B	2.2 - 2.7
	B=123456000	
TAB(30,15);B;		3.2 - 3.5
TAB(30,15);B	Without ";" (i.e.	3.2 - 3.6
TAB(0);B	with line feed)	2.6 - 2.9
TAB(35);B		5.7 - 5.9
TAB(69);B		9.0 - 9.3
USING "##########";B;		3.5 - 3.8
USING "+&&&&&&&&&&";B;		4.0 - 4.3

Use of the BASIC optimizer results, for instance, in an acceleration by factor 1.5 when a 100-byte string is output.

Execution Times of BASIC Statements

Hints:

- Blanks are output for tabulation with TAB(X). This is why the execution time depends on parameter X. With TAB(X,Y), only the parameters are transferred to the workstation and the actual function is executed locally in the workstation without outputting blanks. Therefore it is better to specify the two coordinates (X,Y) for tabulation.

- When PRINT has partly emptied the input/output buffer and no input/output is due for a while, the application program should issue a SIGNAL 3 after the last output instruction for the time being. This utilizes the parallelism between line transmission and the application and it is not necessary to wait for the buffer at the next input or output. This behaviour is beneficial to a port's response time and in particular has a positive effect on the applications' response time on small systems with few ports and many outputs to screen.

- The screen should never be cleared by means of PRINT and blanks. Instead, the 'CS' and 'CF' functions should be used or lines can be overwritten, clearing only the rest by means of blanks (refer to the execution times for PRINT 'CS' and PRINT B$).

  Examples:

  * Clear the screen with 'CS' and reconstruct the header and message lines.

  * Work with a display window ('DW'). Then all screen functions only apply to this window (including 'CS' for instance).

Execution Times of BASIC Statements

4.6.2 INPUT statement

The execution time of the INPUT statement is mainly determined by the following:

- The keyboard operator's waiting and input time.
  During the input (until 'CR' is pressed), the application program is deactivated in the central unit. During this time the CPU serves other ports.

- Utilization of the input/output buffer.
  If the input/output buffer still contains output data, the output is started first and a change of time slice is initiated. After the data has been transmitted, the buffer is free and execution of the INPUT statement is continued. The same thing applies when an output is still active.

With the INPUT statement, it is also possible to output prompting texts on the screen and specify control characters for the workstation, e.g. TAB and 'CS'. After this output data has been transmitted, i.e. when the buffer is free again, the system can accept inputs from the keyboard. Therefore such texts should be sent to the workstation in good time whenever possible so that the buffer is free for the INPUT statement.

Execution Times of BASIC Statements

4.7 File accesses

File accesses are by far the most time-consuming program elements (with the exception of some rarely used arithmetic operations). For this reason, programmers must give great thought to the file concept before writing programs to make sure that the load placed on the disk does not seriously impair the system's performance later. The program should therefore be designed in such a way as to minimize the number of file accesses.

4.7.1 Record structures

A data record generally consists of several fields and programmers should ensure that the sequence of fields in the data records matches the processing sequence whenever possible. If the fields are distributed unfavourably, access will always involve different displacements, resulting in the necessity for several individual accesses.

Example:

Not like this:

```
READ #K,V1,P5(98);E(1)
READ #K,V1,P5(98)+2*P1(98);E(2)
READ #K,V1,P5(98)+2*P1(98)+4*P2(98);E(3)
READ #K,V1,P5(98)+2*P1(98)+4*P2(98)+6*P3(98);E(4)
```

but like this:

```
MAT READ #K,V1,V2;E
```

Two striking errors were made in this example:

- Firstly, the organization of the data record is so unfavourable that each variable has to be read with its own READ statement owing to the different displacements.
- Secondly, repeated recalculation of the displacements puts an unnecessary load on the CPU. At the very least, identical sub-expressions should not be recalculated again and again.

Execution Times of BASIC Statements

4.7.2 Record locks

Another major factor affecting run-time is the way in which record locks are dealt with.

Care should be taken to avoid record locks whenever possible as they make other users wait. Record locks should only be used when a record has to be updated. Particularly with regard to control and parameter files which are accessed from several applications, reading with record locks has negative effects on system throughput.

Record locks should be cancelled as soon as possible.

Refer to section 4.7.5 for more information on this matter.

Execution Times of BASIC Statements

4.7.3 READ/WRITE statements

The execution time for READ and WRITE statements mainly depends on the number of fields and, when a physical access to disk is required, to a great extent on the access time for the attached disk drives.

The times shown in the table below include one physical disk access lasting about 3Ø milliseconds (SMD). The times also cover the conversion of data from the representation used on disk to the one used in memory and vice versa.

Statement	Time in ms
READ:	
1 field	56 - 74
12 fields	78 - 1ØØ
WRITE:	
1 field	88 - 1Ø7
12 fields	1Ø4 - 123

The equivalent execution times in IDC systems are slightly higher than these figures, especially when several fields are processed. This is caused by the logical connection in series of the processors involved (CPU, basic module and disk interface). This must not distract attention from the fact that the IDC does increase performance as a result of the parallel working of the individual processors.

Execution Times of BASIC Statements

4.7.4 SEARCH statement

In order to establish the execution time of the SEARCH statement, two of the possible functions were examined:

Mode 1 (searching for the 1st free data record and deallocating a data record)
Mode 2 (searching for a specific key in a directory)

Mode	Parameters	Time in ms
1	- Searching for and allocating the 1st free record in the file	19 - 23
	- Deallocating a record	26 - 91
2	Searching for various keys (existent and non-existent)	
	- Key length = 6 bytes	18 - 52
	- Key length = 1ø bytes	25 - 64

The considerable spread of the execution times is because data blocks are either found in the buffer pool or have to be read physically from disk.

SEARCH mode 6 (initialization of an indexed file) should only be used in exceptional circumstances if possible. The execution time of this function depends on the size of the file and can be very long because all data records and all directory blocks are written. All other ports on the system have to wait as this process is non-interruptible.

Execution Times of BASIC Statements

4.7.5 OPEN/CLOSE statements

When a disk file is OPENed, careful thought should be given to the use of the parameters "L" (file lock) and "R" (feedback in the event of a lock).

OPEN with the "L" parameter means that all other users who want to access the file have to wait. Therefore this parameter should only be used where it is absolutely essential.

After an attempt to access a locked record, the system waits a few hundred milliseconds as standard and then tries again to access the record in question.

OPEN with the "R" parameter causes a status feedback to the application program in the event of a record lock so that the latter can react to it.

Under no circumstances, however, should this reaction be an immediate attempt to access the same record (loop!) as this would constitute active waiting for the lock to be cancelled (i.e without passive waiting), resulting in a heavy load on the CPU which would reduce the performance of the entire system.

OPEN with the "R" parameter can be used without hesitation when there is no immediate attempt to repeat the READ or WRITE statement after a record lock has been detected. The use of a "standard OPEN" would be preferable in cases where the statement is repeated immediately.

Execution Times of BASIC Statements

Function and parameter	Time in ms
OPEN disk files	82 - 162
OPEN printer	7Ø - 74
CLOSE disk files	121 - 186
CLOSE printer	4Ø - 57

As the figures in the table show, OPEN and CLOSE for disk files are among the statements with particularly long execution times. OPEN and CLOSE should therefore be avoided in frequently executed loops.

In the event of channel bottlenecks, it is better to configure an additional channel. On the other hand, however, the number of configured channels should be kept as low as possible. (This subject is explained in the "System Tuning" manual; order no. 34697.ØØ.7.93.)

Execution Times of BASIC Statements

4.8 CHAIN/LINK statements

The time taken for a change of program via CHAIN and LINK consists of various phases:

- Interpretation of the CHAIN or LINK statement by the BASIC interpreter
- Possibly transmission of data from the input/output buffer to the workstation because the buffer is needed by CHAIN and LINK for other purposes and is emptied.
- Loading of the successor program - from disk unless it is in another partition.
- Change of time slice
- Waiting until the port receives a time slice again (dependent on the current number of users in the system)

The following table indicates the pure interpreter execution time, i.e. the time until the program changes.

Stmnt.	Framework	Time in ms
CHAIN		6.2 - 7.6
LINK	1 global variable A$(1Ø)	6.3 - 7.6
	Many global variables	21.Ø - 39.Ø

These times are increased substantially if the input/output buffer is not empty, e.g. with CHAIN they rise to:

* 2Ø - 25 ms if the buffer contains 1 character
* 27 - 32 ms if the buffer contains 7 characters
* 77 - 82 ms if the buffer contains 5Ø characters

Execution Times of BASIC Statements

Now the time for any loading of the successor program and the time waiting for a new time slice have to be added to these times.

In the final analysis, a complete CHAIN or LINK can take several seconds until the entry point in the successor program is reached in a system with a heavy workload. When organizing the program, programmers should therefore make sure that there are as few program changes as possible in the normal program flow (without special functions and error handling).

Execution Times of BASIC CALLs

5 Execution times of BASIC CALLs

This chapter describes the execution times of some BASIC CALLs which are used frequently. The Discsubs involved were core-resident during the measurement process.

5.1 CALL 1

CALL 1 prepares the screen for an input (function Ø) and checks and displays an input in the correct format (function 1).

CALL 1	No. of chars.	TAB	Dots	Time in ms
Fct. Ø	1	1,1	Without	24 - 25
	1	36,22	Without	24 - 25
	8	36,22	With	42 - 43
	5Ø	36,22	Without	24 - 25
	5Ø	36,22	With	92 - 93
Fct. 1	1	1,1		24 - 25

In time-critical interactive applications at least, programmers should check whether it is necessary to output dots to mark the input position and number of input positions.

Execution Times of BASIC CALLs

5.2 CALL 2 and CALL 3

CALL 2 and CALL 3 move data between the user partition and the common area.

CALL	Parameters	Time in ms
2	5 variables	
	22 bytes in all	3 - 8
	16Ø bytes in all	3 - 8
3	5 variables	
	16Ø bytes in all	3 - 8

5.3 CALL 4

The examination covered the CALL 4 functions for processing the disk archive ID.

CALL 4	Time in ms
Reading the disk archive ID	14 - 18
Writing the disk archive ID	48 - 51

Execution Times of BASIC CALLs

5.4 CALL 21 and CALL 22

CALL 21 and CALL 22 convert binary numbers as part of a string into numeric variables and vice versa.

CALL	Position	Value	Time in ms
21	1	1	1.9 - 2.Ø
	1	12589	2.Ø - 2.1
	49	12589	2.5 - 2.8
22	1	1	3.3 - 3.7
	1	12589	3.2 - 3.6
	49	12589	3.3 - 3.6

5.5 CALL 23

CALL 23 searches for a substring starting at a specified position in a string.

Length of substring	Position of substring	Search fr. position	Time in ms
2	55	1	4.7 - 5.Ø
2	55	5Ø	3.2 - 3.4
6	55	1	4.9 - 5.1
2	Not found	1	5.6 - 5.8

These measurements were based on a string length of 8Ø bytes.

Execution Times of BASIC CALLs

5.6 CALL 26

CALL 26 selectively replaces characters in strings. When, for instance, the character "1" is replaced 8 times by the character "A" in an 8Ø-byte string, the CALL has an execution time of 2.4 - 2.8 ms.

5.7 CALL 6Ø and CALL 61

CALL 6Ø and CALL 61 respectively pack and unpack numeric data in strings.

CALL	No. of digits	Time in ms
6Ø	2Ø	4
	8Ø	12
61	2Ø	3
	8Ø	8

5.8 CALL 99

Reading the system date and system time by means of CALL 99 takes 2.4 -2.8 ms.

	Hint on Data Communication with Code Conversion

6 Hint on data communication with code conversion

With regard to data communication via the PLC, codes should be converted by means of a code table in the PLC whenever possible. Converting codes in the program itself by means of CALL statements puts a considerable load on the CPU and and consequently impairs the system's performance unnecessarily.