ND-500
Loader/Monitor

ND-60.136.03

NORSK DATA A.S

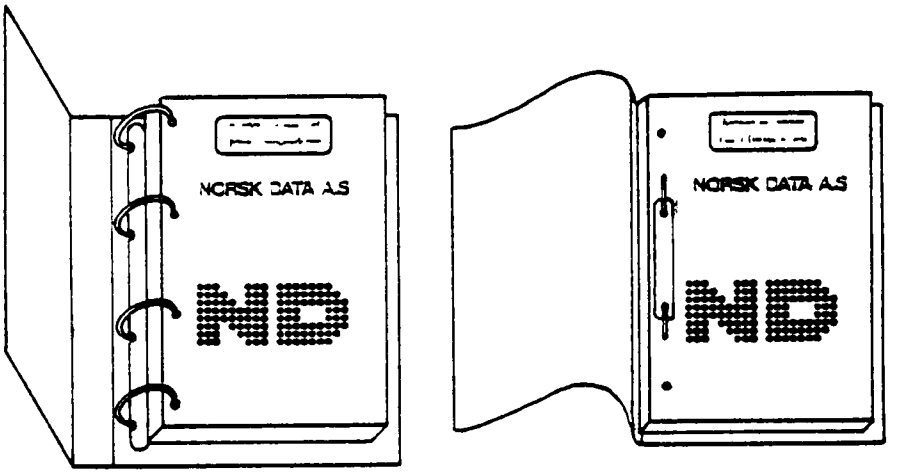
ND-500
Loader/Monitor

ND-60.136.03

This manual is in loose leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.



A Ring Binder B Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name .....
Company .....
Address .....
.....
City .....

NOTICE

PRINTING RECORD

Printing	Notes
01/81	Version 01
07/81	Version 02
01/82	Version 03

ND-500 Loader/Monitor
Publ.No. ND-60.136.03
January 1982

NORSK DATA A.S
P.O. Box 4, Lindeberg gård
Oslo 10, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the ND Bulletin and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

PREFACE

THE PRODUCT

This manual describes

Linkage-Loader	ND-10319 version C
ND-500 Monitor	ND-10320 version B

The ND-500 Monitor is an extension of the Sintran III operating system which provides for program execution on the ND-500 computer. The Linkage-Loader runs as an ND-500 program, while the ND-500 Monitor is an integral part of the Sintran III VSE/500 operating system.

The ND-500 memory management system is described in the ND-500 CPU Reference Manual. However, Sintran III does not utilize the hardware fully, as a process may consist of one domain only.

THE READER

This manual is written for programmers and operators who want to load and run programs on the ND-500. It also describes the Monitor commands available to the system supervisor for maintaining proper control over ND-500 resources.

PREREQUISITE KNOWLEDGE

The reader is assumed to have some previous knowledge of the ND-500, the ND-100 and the Sintran III operating system. Depending on the intended use of the ND-500 computer, this may vary from knowing how to compile a simple program with only a rudimentary knowledge of the memory management system (for a programmer with timesharing/background requirements only) to familiarity with the hardware configuration, ND-100 segment file and RT loader structure (for the system supervisor).

Necessary information is found in the following manuals:

ND-500 CPU Reference Manual (ND-05.009)
SINTRAN III Reference Manual (ND-60.128)
SINTRAN III System Supervisor (ND-60.103)
SINTRAN III RRT Loader (ND-60.051)

THE MANUAL

This manual will give the reader information about how to link relocatable modules to make an executable ND-500 program (domain), and how to execute programs on the ND-500 under the Sintran III operating system.

The manual should be used for reference; it is not intended to be a textbook in loader and monitor use. Each command description is independent of others, and can be read without knowing other commands described. However, the first chapters contains some introductory information about the ND-500 system.

A thorough understanding of the ND-500 memory management system and trap handling is required to fully utilize the ND-500. A detailed description may be found in the ND-500 CPU Reference Manual ND-05.009 chapters 4 and 6.

TABLE OF CONTENTS

Section	Page
1. INTRODUCTION	3
1.1. Use of the Linkage-Loader and Monitor	3
1.1.1. Compilation	3
1.1.2. Loading	4
1.1.3. Execution	5
1.1.4. Multi-segment domains	6
1.2. Command and parameter format	8
1.3. Command syntax	8
1.4. Naming rules	10
1.5. The description file	10
1.6. The function of the Linkage-Loader	11
1.7. The function of the Monitor	12
2. MEMORY MANAGEMENT SYSTEM	15
2.1. Logical memory structure	15
2.2. Capabilities	17
3. TRAPS	19
3.1. Trap handler calling	19
3.2. Use of trap handlers	20
4. STANDARD EXCEPTION HANDLER LIBRARY	21
4.1. ND-500 traps table	23
4.2. The EXCEPT routine	24

Section Page

4.3. The EXCDEF routine 29

4.4. The EXCTERM routine 31

4.5. The PRITRAC routine 32

4.6. The PRIMESS routine 33

4.7. The GETMESS/PGETMESS routine 34

4.8. The RDEFVAL routine 35

4.9. The RCURVAL routine 36

5. COMMUNICATION BETWEEN ND-500 AND ND-100 39

5.1. Monitor calls 39

5.2. Communicating through the process flags 39

5.3. Communicating through RTCOMMON 40

5.4. Communicating through an RT segment 40

5.5. Communicating through files 41

6. LOADER COMMANDS 42

6.1. Domains 42
6.1.1. SET-DOMAIN 42
6.1.2. END-DOMAIN 43
6.1.3. CLEAR-DOMAIN 43
6.1.4. DELETE-DOMAIN 43
6.1.5. LIST-DOMAIN 44
6.1.6. WRITE-DOMAIN-STATUS 44
6.1.7. RENAME-DOMAIN 44
6.1.8. COPY-DOMAIN 45
6.1.9. RELEASE-DOMAIN 46

6.2. Segments 47
6.2.1. OPEN-SEGMENT 47
6.2.2. CLOSE-SEGMENT 49
6.2.3. LINK-SEGMENT 49
6.2.4. LIBRARY-SEGMENT-LINK 50
6.2.5. APPEND-SEGMENT 50
6.2.6. SET-SEGMENT-NUMBER 51

Section		Page
6.2.7.	CLEAR-SEGMENT	51
6.2.8.	DELETE-SEGMENT	52
6.2.9.	RENAME-SEGMENT	52
6.2.10.	LIST-SEGMENT	52
6.2.11.	WRITE-SEGMENT-STATUS	53
6.3.	Commands to load NRF code	54
6.3.1.	LOAD-SEGMENT	54
6.3.2.	RELOAD-SEGMENT	55
6.3.3.	LIBRARY-SEGMENT-LOAD	55
6.3.4.	OMITTED-SEGMENT-LOAD	56
6.3.5.	SELECTED-SEGMENT-LOAD	56
6.3.6.	TOTAL-SEGMENT-LOAD	57
6.4.	COMMON segments	58
6.4.1.	COMMON-SEGMENT-OPEN	58
6.4.2.	COMMON-SEGMENT-CLOSE	59
6.4.3.	COMMON-SEGMENT-APPEND	59
6.4.4.	COMMON-SEGMENT-NUMBER	59
6.5.	Auto-link segments	60
6.5.1.	SET-AUTO-LINK-SEGMENT	60
6.5.2.	DELETE-AUTO-LINK-SEGMENT	61
6.5.3.	LIST-AUTO-LINK-SEGMENTS	61
6.6.	Auto-load files	62
6.6.1.	SET-AUTO-LOAD-FILE	62
6.6.2.	DELETE-AUTO-LOAD-FILE	63
6.6.3.	LIST-AUTO-LOAD-FILE	63
6.7.	Label and reference handling	64
6.7.1.	PROGRAM-REFERENCE	64
6.7.2.	DATA-REFERENCE	65
6.7.3.	DEFINE-ENTRY	65
6.7.4.	DEFINE-COMMON	66
6.7.5.	LIST-ENTRIES-DEFINED	66
6.7.6.	LIST-ENTRIES-UNDEFINED	67
6.7.7.	LIST-MAP	67
6.7.8.	SYSTEM-ENTRIES-ON	67
6.7.9.	GLOBAL-ENTRIES	68
6.7.10.	KILL-ENTRIES	68
6.8.	Areas shared with ND-100 processes	69
6.8.1.	MATCH-RTCOMMON	69
6.8.2.	MATCH-COMMON-RT-SEGMENT	70
6.8.3.	LINK-RT-PROGRAM	70
6.9.	Miscellaneous commands	71
6.9.1.	LOW-ADDRESS	71
6.9.2.	HIGH-ADDRESS	71
6.9.3.	ENTRY-ROUTINES	72
6.9.4.	SET-IO-BUFFERS	72
6.9.5.	LIST-OCTAL	73

Section Page

6.9.6. LIST-SYMBOLIC 73
6.9.7. LIST-MODE 74
6.9.8. DISASSEMBLE-MODE 74
6.9.9. CHECK-SYNTAX-MODE 74
6.9.10. RESET 74
6.9.11. RENAME-DEFAULT-DIRECTORY-AND-USER 75
6.9.12. SUPPRESS-DEBUG-INFORMATION 75

6.10. NRF editor 76
6.10.1. NEW-NRF-MODULES 76
6.10.2. FETCH-NRF-MODULES 76
6.10.3. APPEND-NRF-MODULE 77
6.10.4. DELETE-NRF-MODULES 77
6.10.5. LIST-NRF-ENTRIES 78
6.10.6. LIST-NRF-CODE 78
6.10.7. WRITE-NRF-EOF-AFTER-MODULE 79
6.10.8. INSERT-NRF-MESSAGE 79
6.10.9. PREPARE-NRF-LIBRARY-FILE 80

7. COMMANDS AVAILABLE IN THE NLL AND THE MONITOR 81

7.1. Utility commands 81
7.1.1. HELP 81
7.1.2. OUTPUT-FILE 81
7.1.3. @ (Sintran-III command) 82
7.1.4. CC 82
7.1.5. ABORT-BATCH-ON-ERROR 82
7.1.6. EXIT 83

7.2. Trap handling 84
7.2.1. LOCAL-TRAP-ENABLE 85
7.2.2. LOCAL-TRAP-DISABLE 85
7.2.3. SYSTEM-TRAP-ENABLE 86
7.2.4. SYSTEM-TRAP-DISABLE 86

7.3. VALUE-ENTRIES 86

8. MONITOR COMMANDS 87

8.1. Commands for running an ND-500 program 87
8.1.1. PLACE-DOMAIN 87
8.1.2. RUN 88
8.1.3. RECOVER-DOMAIN 88
8.1.4. GO 89
8.1.5. CONTINUE 89

8.2. Standard domains 90

Section Page

8.2.1. DEFINE-STANDARD-DOMAIN 90
8.2.2. DELETE-STANDARD-DOMAIN 91
8.2.3. LIST-STANDARD-DOMAINS 91

8.3. Commands for opening and connecting files 92
8.3.1. OPEN-FILE 92
8.3.2. CLOSE-FILE 93
8.3.3. LIST-OPEN-FILES 93
8.3.4. Error returns 94

8.4. Direct file transfer 95
8.4.1. Direct file transfer with RFILE and WFILE (disk) 95
8.4.2. Direct file transfer with MAGTP (magnetic tape) 96

8.5. Macro commands 97
8.5.1. DEFINE-MACRO 97
8.5.2. Macro subcommands 98
2.1. IF-ERROR-MACRO-STOP 98
2.2. IF-ERROR-FULL-STOP 98
2.3. NOLIST 98
2.4. LIST 98
8.5.3. EXECUTE-MACRO 99
8.5.4. RESUME-MACRO 100
8.5.5. ERASE-MACRO 100
8.5.6. DUMP-MACRO 100
8.5.7. LIST-MACRO 101

8.6. Debugging commands 102
8.6.1. DEBUG-PLACE 102
8.6.2. BREAK 102
8.6.3. TEMPORARY-BREAK 103
8.6.4. STEP 103
8.6.5. LOOK-AT commands 104
5.1. LOOK-AT-PROGRAM 106
5.2. LOOK-AT-DATA 107
5.3. LOOK-AT-STACK 107
5.3.1. Subcommands PREVIOUS and NEXT 108
5.4. LOOK-AT-RELATIVE 108
5.5. LOOK-AT-REGISTER 108
8.6.6. SET-MEMORY-CONTENTS 109
8.6.7. MAIN-FORMAT 109
8.6.8. EXTRA-FORMAT 110
8.6.9. TRACE 110
8.6.10. GUARD 111
8.6.11. BRANCH-TRACE 112
8.6.12. CALL-TRACE 112
8.6.13. EXHIBIT-ADDRESS 113
8.6.14. DEBUG-STATUS 114
8.6.15. ENABLED-TRAPS 114
8.6.16. STATUS 114
8.6.17. RESET commands 114
17.1. RESET-DEBUG 114
17.2. RESET-BREAKS 115

Section		Page
	17.3. RESET-LAST-BREAK	115
	17.4. RESET-TRACE	115
	17.5. RESET-GUARD	115
	17.6. RESET-CALL-TRACE	116
	17.7. RESET-BRANCH-TRACE	116
8.7.	Commands for performance measurement	117
	8.7.1. Histogram commands	117
	1.1. SET-HISTOGRAM	117
	1.2. START-HISTOGRAM	118
	1.3. STOP-HISTOGRAM	118
	1.4. PRINT-HISTOGRAM	118
	1.5. RELEASE-HISTOGRAM	118
	8.7.2. Monitor call logging	119
	2.1. START-MONCALL-LOG	119
	2.2. PRINT-MONCALL-LOG	119
	2.3. STOP-MONCALL-LOG	119
	8.7.3. Process logging	120
	3.1. START-PROCESS-LOG-ALL	120
	3.2. START-PROCESS-LOG-ONE	120
	3.3. PRINT-PROCESS-LOG	120
	3.4. PROCESS-LOG-ALL	121
	3.5. PROCESS-LOG-ONE	121
	3.6. RELEASE-LOG-BUFFER	122
	8.7.4. SWAPPING-LOG	122
	8.7.5. LIST-EXECUTION-QUEUE	122
8.8.	Process communication flags	123
	8.8.1. GET-FLAG	123
	8.8.2. SET-FLAG	123
8.9.	Memory allocation	124
	8.9.1. Demand paging	124
	8.9.2. "Fixing" in memory	125
	8.9.3. Limiting the number of pages in memory	125
	8.9.4. "Fixing" programs in memory	126
	8.9.5. Fixing segments scattered in memory	126
	8.9.6. Fixing segments in contiguous memory	127
	8.9.7. Fixing segments in an absolute location	128
	8.9.8. Fixing segments shared by several processes	128
	8.9.9. Unfixing a segment	129
	8.9.10. The swapping strategy	130
	8.9.11. SET-SEGMENT-LIMITS	132
	8.9.12. FIX-SEGMENT-SCATTERED	132
	8.9.13. FIX-SEGMENT-CONTIGUOUS	133
	8.9.14. FIX-SEGMENT-ABSOLUTE	133
	8.9.15. UNFIX-SEGMENT	134
8.10.	Miscellaneous commands	135
	8.10.1. AUTOMATIC-ERROR-MESSAGE	135
	8.10.2. RESET-AUTOMATIC-ERROR-MESSAGE	135
	8.10.3. The "Escape" key	135
	8.10.4. TIME-USED	135

Section	Page
8.10.5. WHO-IS-ON	136
8.10.6. VERSION	136
8.10.7. SET-PRIORITY	136
8.11. Commands for the System Supervisor	138
8.11.1. SET-ND-500-UNAVAILABLE	138
8.11.2. SET-ND-500-AVAILABLE	138
8.11.3. STOP-ND-500	139
8.11.4. Memory configuration	139
4.1. DEFINE-MEMORY-CONFIGURATION	139
4.2. MEMORY-CONFIGURATION	140
8.11.5. Memory administration	140
5.1. GIVE-ND-500-PAGES	140
5.2. TAKE-ND-500-PAGES	141
8.11.6. Microprogram maintainance	142
6.1. MICRO-STOP	142
6.2. MICRO-START	142
6.3. LOAD-CONTROL-STORE	142
6.4. COMPARE-CONTROL-STORE	143
6.5. LOOK-AT-CONTROL-STORE	143
6.5.1. Subcommands EDIT and ORIN	144
6.5.2. Subcommands OCTAL and SYMBOLIC	144
6.5.3. Subcommands GROUP and WORD	144
8.11.7. LOOK-AT commands	144
7.1. LOOK-AT-RESIDENT-MEMORY	144
7.2. LOOK-AT-PHYSICAL-SEGMENT	145
7.3. LOOK-AT-HARDWARE	145
8.11.8. Process management	146
8.1. ATTACH-PROCESS	146
8.2. LOGOUT-PROCESS	146
8.3. ABORT-PROCESS	146
8.4. PROCESS-STATUS	147
8.11.9. Inspecting system tables	147
9.1. LIST-TABLE	147
9.2. LIST-ACTIVE-SEGMENTS	148
9.3. LIST-SEGMENT-TABLE-ENTRY	148
9.4. LIST-PROCESS-TABLE-ENTRY	148
8.11.10. Swap files	149
10.1. DEFINE-SWAP-FILE	149
10.2. DELETE-SWAP-FILE	149
10.3. LIST-SWAP-FILE-INFO	150
10.4. LOAD-SWAPPER	150
10.5. START-SWAPPER	150
8.11.11. SET-SYSTEM-PARAMETERS	151
8.11.12. LIST-SYSTEM-PARAMETERS	151
8.11.13. MASTER-CLEAR	151
9. SINTRAN-III MONITOR CALLS	153
10. THE N500M MONITOR CALL	161

Section Page

11. DESCRIPTION FILE LAYOUT 165

12. THE ND RELOCATABLE FORMAT 169

12.1. DESCRIPTION 169

12.2. NRF control numbers 170

12.3. Summary of NRF control numbers 175

13. LINKAGE-LOADER ERROR MESSAGES 176

14. ND-500 MONITOR ERROR MESSAGES 182

15. EXAMPLES OF LINKAGE-LOADER AND MONITOR USAGE 198

15.1. Executing an ND-500 domain 198

15.2. Using libraries 199

15.3. Using files 201

15.4. Macros 202

15.5. Debugging 203

15.6. System Supervisor: Installing NLL 205

INDEX 207

1. INTRODUCTION

1.1. Use of the Linkage-Loader and Monitor

An ND-500 program goes through two main steps before it is ready for execution:

- compilation, transforming a human readable program into machine code

- loading, combining the user program and subroutines with library routines, and assigning the program a specific position in memory

1.1.1. Compilation

The compilation is performed by a compiler specific to the language of the source program: a Fortran compiler, a Pascal compiler, a Planc compiler or a Cobol compiler. The compiler may run in the ND-100, even if the compiled program will be running in the ND-500. More commonly, even the compiler runs in the ND-500.

A compiler running in the ND-500 is operated in exactly the same way as an ND-100 compiler. However, in order to start it, the name of the ND-500 monitor must preceed the compiler name:

```
@ND-500 FORTRAN
```

"ND-500" is (an abbreviation of) the name of the monitor. The monitor is a rather complex system controlling the ND-500 computer, but for the beginner, "ND-500" may be viewed simply as a message to the operating system requesting program execution in the ND-500 rather than the ND-100.

"FORTRAN" is the name of the Fortran compiler. Generally, compilers have the name of the language they compile, followed by the machine the code is made for. The full name of the Fortran compiler is FORTRAN-500, but in most installations, an abbreviation is unambiguous.

A program is compiled with the COMPILE command:

```
@ND-500 FORTRAN
ND-500 ANSI 77 FORTRAN COMPILER - NOVEMBER 24, 1981

FIN: COMPILE TESTPROG,"TESTPROG:LIST", "TESTPROG"

- CPU TIME USED: 3.2 SECONDS.  750 LINES COMPILED.
- NO MESSAGES
- CODE SIZE=3644  DATA SIZE=403  COMMON SIZE=0  STACK SIZE=654
FIN: EXIT
```

The default file type of the code file generated by all ND-500 compilers is :NRF, while the default file type of ND-100 compilers is :BRF. Thus, if the same file is compiled for for both computers, the code files may be given the same name without causing a name conflict.

1.1.2. Loading

After compilation, there is no difference between programs and subroutines in different languages, and they are all loaded using the same loader. The loader is called by the command

```
@ND-500 LINKAGE-LOADER
```

"ND LINKAGE-LOADER" is often simply called "NLL".

NLL will create a program ready for execution. In the ND-500, a program is termed a domain. A domain usually has a name, used when starting execution. It may also be "unnamed" - actually, it then has the standard name SCRATCH-DOMAIN. Any permanent domain should be given a name; each time a file is loaded to an unnamed domain, it will overwrite the previous contents of SCRATCH-DOMAIN. It may, however, be convenient to omit the naming of the domain during the debug phase of a program.

A domain is named before anything is loaded to it, by the NLL command

```
NLL: SET-DOMAIN "DOMAIN-NAME"
```

The double quotes indicate that this is a new domain, used exactly like double quotes to create a new file (however, the domain is not a file). If no double quotes are used, an existing domain will be overwritten. (This is strictly true only for simple use of NLL.)

The code file generated by the compiler is loaded by the command

```
NLL: LOAD-SEGMENT TESTPROG
```

Several files may be named in the LOAD-SEGMENT command; for example, the main program and subroutines may be compiled separately, to different code files. Also, several LOAD-SEGMENT commands may be used in succession.

After all files required have been loaded, NLL is left through the command

```
NLL: EXIT
```

A number of operations are performed in the EXIT command: references to the required libraries are set up, the default handling of errors is defined, the appropriate files are updated and the file access on new files set.

As mentioned, the domain is not a file, nor is any program file explicitly specified. This does not imply that there are no files used - the loaded code is stored in files manipulated by NLL. These have types :PSEG, :DSEG and :LINK, and (by default) names chosen by NLL. In addition, there is a file called DESCRIPTION-FILE:DESC. For all practical purposes, these files are invisible to the programmer - he will always identify his program by its domain name.

1.1.3. **Execution**

A user domain is started by typing the name of the ND-500 monitor followed by the domain name:

```
@ND-500 DOMAIN-NAME
```

This is exacly like the way a compiler running in the ND-500 is started; "ND-500" is a message informing Sintran that the program is to be executed in the ND-500 computer rather than the ND-100.

Communication with the user through the terminal is exactly as for an ND-100 program, as are file access and access to various devices. Pushing the escape or break key will interrupt the program and return control to Sintran.

The user may also type "ND-500" without following it by a domain name. This will start the monitor, and give control to the command processor of the monitor:

```
@ND-500
ND-500 MONITOR  81.11.14/81.11.04
N500:
```

Execution of a domain may now be started simply by typing its name:

```
N500: DOMAIN-NAME
```

After execution, control returns to the command processor of the monitor, rather than to Sintran, and another domain may be executed. As an alternative to first starting the compiler by @ND-500 FORTRAN, then NLL by @ND-500 LINKAGE-LOADER and finally the program by @ND-500 DOMAIN-NAME, they may be run by the command sequence

```
@ND-500
N500: FORTRAN
FTN:  <compiler commands>
FTN:  EXIT
N500: LINKAGE-LOADER
NLL:  <loader commands>
NLL:  EXIT
N500: DOMAIN-NAME
      <program input/output to terminal>
N500: EXIT
@
```

Even if execution is interrupted by the escape or break key, return will be to the monitor. All files are then kept open, and execution may be resumed by the CONTINUE command.

The command processor of the monitor will interpret and execute a large set of commands, described in chapter 8 of this manual. The majority of these are highly specialized commands and commands for the system supervisor. Regardless of the complexity of the program system, it is executed simply by stating its name.

1.1.4. Multi-segment domains

A set of subroutines used in several domains will, if loaded by the LOAD-SEGMENT command together with the main program, occupy space in each and every domain it is used. In order to save file space (and also memory space if the two domains are executed concurrently), these routines may be grouped together and put on a segment, a "slice" of the addressing area that may be treated independently of the other slices (segments).

If only one segment is used, that segment is usually "unnamed" - it is given a standard name by NLL, which may be ignored by the user. A segment used by several domains should be given a more descriptive name. This is done by explicitly opening a segment (after the domain has been named):

```
NLL: SET-DOMAIN "TWO-SEG-DOMAIN"
NLL: OPEN-SEGMENT "SUBROUTINES", P
```

A new segment is created by enclosing the name in double quotes, as shown above. If the quotes are not included, new information will overwrite what is already loaded to the segment.

"P" is an attribute code that allows this segment to be used by other domains as well. Now, the subroutines that are common to several segments is loaded by a LOAD-SEGMENT command:

```
NLL: LOAD-SEGMENT SUBR-FILE
```

When all common subroutines have been loaded (possibly from several files), the subroutine segment is finished by the command

```
NLL: CLOSE-SEGMENT
```

after which the main program (and possibly non-common subroutines) is loaded as usual. But before the EXIT command, the user should link the subroutine-segment to the main program by the command

```
NLL: LINK-SEGMENT SUBROUTINES
```

There may be more than one subroutine segment, each of them opened with the OPEN-SEGMENT command and terminated with a CLOSE-SEGMENT, and they should all be listed as parameters to the LINK-SEGMENT command.

The complete set of commands for loading a two-segment program is therefore, complete with the response from NLL:

```
NLL: SET-DOMAIN "TWO-SEGMENTS"
NLL: OPEN-SEGMENT "SUBROUTINES" P
NLL: LOAD-SEGMENT SUBR-FILE
Program:.........44660B P    Data:...........17344B D
NLL: CLOSE-SEGMENT
Segment no.........30         is linked
NLL: LOAD-SEGMENT MAINPROG
NLL: LINK-SEGMENT SUBROUTINES
Segment no.......... 1       is linked
NLL: EXIT
Segment no.........30         is linked
```

(Segment no 30 contains the Fortran library, and will in most installations be linked automatically, as the example above).

When loading the second and following domain using the routines in the SUBROUTINES segment, the files are already loaded. The OPEN-SEGMENT, LOAD-SEGMENT SUBR-FILE and CLOSE-SEGMENT commands are omitted. Only the main program segment is loaded, followed by the LINK-SEGMENT command.

A slight problem occurs with the segment numbers: each segment has a fixed number between 0 and 31, which must be unique within the domain. By default, new segments are given the first number available, starting at 1; thus the SUBROUTINES segment above is number 1. When a segment is created in a new domain that will also be linked to the SUBROUTINES segment, another segment number should be selected for the main program and other segments. This is done by the commands

```
NLL: SET-DOMAIN "SECOND-DOMAIN"
NLL: SET-SEGMENT-NUMBER 2
NLL: LOAD-SEGMENT SECOND-MAIN
Program:..........766B P     Data:.............244B D
NLL: LINK-SEGMENT SUBROUTINES
Segment no......... 1        is linked
NLL: EXIT
```

If the SUBROUTINES segment will be linked to a high number of main programs, it may be more convenient to set the segment number of the SUBROUTINES segment, leaving segment number 1 (the default value) for the various main segments.

If two or more subroutine segments are used by one domain, they must all have different segment numbers.

1.2. Command and parameter format

Normally, the user communicates with NLL and the Monitor through a terminal. The terminal is called the communication device. In a batch or mode job the communication device is the command input file for input and the output file for output.

Information returned from command execution is usually written to the communication device. Such output may be directed to another file or device by the OUTPUT-FILE command. The current file used for output is called the output device, whether this is the same as the communication device or another file.

Commands to NLL and the Monitor may be given in upper or lower case letters. Commands and parameters are terminated by comma, space or carriage return. If required parameters are not supplied, they are prompted for with the names of the parameters. Parameters may be left out by typing two successive commas in the command line, or pressing CR (Carriage Return) in response to the prompt. If a parameter is not supplied, the default value is used if it exists.

Numeric parameters are specified in octal, unless the number is followed by a D, indicating decimal format, or H, indicating hexadecimal format. If a hexadecimal number does not start with a digit, it must be preceeded by a (redundant) leading zero to avoid confusion with alphanumeric symbols.

1.3. Command syntax

When describing the commands available in the Monitor and NLL the following rules are applied:

- The command name is used as a section header.
- All parameters are enclosed in < > brackets.
- If a parameter that is asked for has a default value, its name is also enclosed in ( ) brackets.
- The names of optional parameters that are not asked for are enclosed in [ ] brackets.
- If more than one value may be specified the right enclosing bracket is followed by an ellipsis, as in <>... .

All command, domain and segment names may be abbreviated as long as they are unambiguous. Most of the command names follow two rules:

- The first word in the command describes the action.
- The second word in the command describes the subject the action is going to be taken upon.

If, for instance, the command HELP is used in the following way:

HELP -SEGMENT

all commands concerning segment manipulation are printed on the output device.

New domains and segments are created by surrounding the name with double quotes (" "). Double quotes are only valid in commands with a name as a parameter. These commands are: SET-DOMAIN, OPEN-SEGMENT and COMMON-SEGMENT-OPEN. If the double quotes are not used, the named object (domain or segment) is assumed to exist.

NLL will prompt for required but missing parameters. Multiple parameters will be asked for the first time, and the full range of Sintran III editing characters is available. If the first character of a command line is '@', the command is taken to be a Sintran III command. The character '&' means that the input line continues on the next line.

In interactive mode, all list output can be temporarily stopped by typing any character on the input device. The output is resumed when another character is typed. In order to terminate the listing, an '@' may be typed.

The ND-500 Monitor is started by typing ND-500-MONITOR in response to the Sintran III prompt. A domain name may follow on the same line, implying a RECOVER-DOMAIN command with this name as parameter. If no domain name was specified on the call line, the ND-500 Monitor will prompt for commands with "N500: ". The rules for parameter specification are the same as in NLL. Wherever a parameter from a list of valid values is expected, "HELP" may be written. This will cause the possible choices for the parameter values to be printed on the communications device. (Obviously, this does not apply where an arbitrary string, such as a domain name, may be specified.)

1.4. Naming rules

Segments and domains are referred to by name in NLL and the Monitor. The name of a segment is equal to the name of the segment files. The program, data and symbol files of a segment have the same name, but are distinguished by their types: :PSEG, :DSEG and :LINK respectively. As segment names coincide with file names, the segment name syntax follows the Sintran III file name syntax, and the segment name must be unique in the current user's file catalog. The file type may not be modified.

Domain names may - like segment names - contain alphanumerics and hyphens, and may coincide with segment names. Maximum length is 16 characters, and lower case characters are converted to upper case.

1.5. The description file

The names of segments and domains are found in a file called DESCRIPTION-FILE:DESC. Each named object (segment or domain) has an entry in this file, containing all information needed by NLL and the Monitor. For example, the domain entry - one for each domain - contains the name of the domain, a table of the segment files of which the domain consists, information about the relationships to other domains, the size and the start address of the domain, and information relevant to the internal operation of the Monitor.

Every user of NLL has his own description file, which is created and initialized the first time the user starts NLL.

Although all domains of a user are described in one file, the same user can access NLL from several terminals simultaneously; NLL will see to it that access conflicts are resolved. If attempts are made to modify the same domain from two terminals simultaneously, one of the users will get an error message.

A word of warning:

The contents of the description file at any time reflects the state of the segment and domain definitions of the current user. The user should take great care to never make any modifications to the segment files or the description file, except through NLL. Otherwise inconsistencies may arise, and it may be necessary to rebuild the description file, thereby losing all information about previously loaded segments.

1.6. The function of the Linkage-Loader

The output from language processors (compilers, assemblers) is in the form of relocatable modules. The term 'relocatable' means that the modules are not assigned a fixed position, but may be placed anywhere in memory. Modules are not dependent on being placed in any specific sequence.

NLL is a subsystem able to convert relocatable object files in ND Relocatable Format (NRF) created by language subsystems, to independent executable programs, or processes.

A process is a set of instructions to be executed in a sequential manner, and its associated data. The simplest process possible consists of one segment in one domain; a more complex process may consist of up to 32 segments. A segment is built by NLL, on three separate files: one file contains the instructions: the program segment; another contains the data: the data segment; the third contains the names and values of all labels and optional debugging information, and is called the :LINK file.

A domain is an addressing space, divided into segments. Domains and segments are described in detail in the ND-500 CPU Reference manual ND-05.009.

Information about intermodule references, symbols and labels is coded in the file that is output from compilers. The format of this code is such that procedure calls or references to global data are made through symbols, that is, alphanumeric (symbolic) names assigned to an instruction or data item. These symbols are made by the language processor (often based on user assigned names in the source program), and are referred to as 'labels'.

At execution time, references are made to addresses rather than to labels. The conversion from relocatable symbols to machine addresses is done by NLL. NLL will maintain a table, called the loader table, where symbols are entered as they are encountered.

A symbol may refer to a machine address or it may be a data value. If the first occurence of a symbol is its definition, then the loader will enter the symbol name into the loader table together with the address where it is defined or together with its data value. In either case, the value associated with the symbol is simply called the symbol's value. Whenever a reference to the label is later encountered, the symbolic reference is replaced with the value found in the loader table.

If a reference is made to a label before it has been defined, space is left open in the loaded code for later insertion of its value. The symbol is entered in the loader table, but rather than containing a value of the label, the table contains a reference to where the symbol is used. As soon as a definition of the label is read, the loader will fill in the now defined value wherever a reference has been made.

If two definitions of one label are encountered, the loader cannot distinguish between them, and an error message is issued. In such cases, the first definition of the label always applies.

Before the program is ready for execution, the loader must ensure that all symbolic references are replaced with numeric values/addresses. To achieve this, it may be necessary to load libraries, either by the user or automatically. The loader is able to distinguish between a required and a not required module in a library.

A note on the terminology:

In this manual, the term 'reference' is used to describe a symbol that has been entered into the loader table, but has not yet been defined. An 'undefined entry' is equivalent to a reference. The term 'label' is used for a symbol which has been assigned a value in the loader table; it may have been referenced or not. A 'defined entry' is equivalent to a label. 'Symbol' is the general term for all symbolic (alphanumeric) names, but is mostly used for names not yet in the loader table.

The term 'loading' is sometimes used in the sense 'bring into memory for execution'. Another interpretation is 'to fetch relocatable program units and link together to an executable program'.

In this manual, as in all ND software, the latter usage is adopted. The bringing of a program into memory for execution will in most cases be completely invisible to the user of the program; he may consider the program file to be directly executable.

In the cases where the program is read into physical memory, different terms are employed to describe this depending on the specific situation, such as 'starting execution' or '"fixing" a segment'.

1.7. The function of the Monitor

The ND-500 computer has no capabilities to communicate directly with the "outer world". Nor does it have an elaborate operating system administering users' processes and system resources.

Such tasks are executed by the ND-100 CPU. The functions performed are manifold; some of the more important ones are:

The user will always communicate with the ND-100 machine. When he enters the Monitor, he enters a program that has the capability of transforming the user requests into orders to the ND-500. For example, when execution of a program is started through the RECOVER-DOMAIN command, the Monitor will open the files required, reserve a scratch area for data that is modified during execution, create a table entry in a system table identifying the user of the system resources and so on. When all administrative work is complete, a message is sent to the ND-500 requesting execution of the program.

During execution, the program may request input or output of data, may request system information (such as the time of day etc.) or other services that the operating system provides. Such requests are not handled in the ND-500, but are transferred to the Monitor. The Monitor will initiate an I/O operation, obtain the requested information or perform the operation required, before the result of the request is returned to the program in ND-500.

If an error occurs in the ND-500 and is not taken care of by the user program, the error is reported to the Monitor, and it may take recovery actions, or possibly abort the job with an error message sent to the user. If one program monopolizes the CPU for a certain period of time, the Monitor will intervene, and temporarily suspend the program, letting other programs execute in the meantime.

In the debugging phase of a program, the Monitor may act as a supervisor of the user program, providing the user with commands to inspect and modify the program during execution. As the code required to fetch information about the user program is a part of the Monitor, the program being debugged may be compiled and loaded exactly like a production program. This guarantees that the results produced are unaffected by debugging instructions.

The Monitor also performs a number of system oriented tasks, such as book-keeping of resource usage, preventing unauthorized users from executing privileged functions etc. Because all communication with the ND-500 is channeled through the Monitor, the interface between the user and his program may look exactly as if there were only one CPU (except for the starting of the Monitor). Letting the ND-100 perform all administrative tasks also frees the ND-500 for user programs. The two processors may work in parallell, with the ND-500 executing a user program while the ND-100 prepares the next job.

2. MEMORY MANAGEMENT SYSTEM

The maximum program size that ND-500 is able to handle is too large to handle as one unit, both for man and machine. A logical subdivision is done by splitting a domain into segments, where each segment is of a more managable size, and the interface between the segments is clearly defined. This subdivision is handled by the machine by its memory management system. The architecture of this system will to some degree affect large programs and programs with special communication requirements.

Understanding the information in this chapter is not required for running most ordinary programs. Nevertheless, it provides the background information necessary in order to understand the use of all commands described in the manual.

2.1. Logical memory structure

An ND-500 addressing space is called a DOMAIN. A domain contains an executable program that can be started through the ND-500 Monitor. For practical purposes a domain may be considered equivalent to a program.

The address range of a domain may vary from 2k bytes up to 4 gigabytes, equivalent to a 32 bit address space. Instructions and data are, however, kept fully separated, and, in fact, a domain contains one area for instructions and another for data. These cover the same address range, but as instructions may never be read as data, or data executed as instructions, no conflicts arise.

A domain is divided into SEGMENTS. A domain comprises from one to 32 segments: the uppermost five addressing bits select the segment. The instruction and the data part of the segment (in the program and data areas of the domain) are termed the instruction and the data segment, respectively.

A segment is a set of files, cataloged under the Sintran III file system. The instruction segment and the data segment have the same name, but types :PSEG and :DSEG, respectively. In addition, there is a :LINK file. The :LINK file is not opened when the program is executed but is used during the loading process and by the symbolic debugger. These three files together are called the segment, unless a qualification of program, data or :LINK file is made.

The files may be indexed or contiguous, but will by default be indexed.

A domain consists of a table of segments, and is not a separate entity in the file system. The segment tables for all domains belonging to one user are kept in a file called DESCRIPTION-FILE:DESC.

Domains and segments are referred to by symbolic names. Internally, a numerical index is employed, but the user will not have to be concerned about this index; NLL will obtain the domain or segment number from the description file. The domain name follows the syntax of and may coincide with file names. Domain names are stored solely in

the description file. Segment names are the names of the :PSEG, :DSEG and :LINK files making up the segment. These names are also stored in the description file, where the position in the segment table for the domain determines the segment number.

When required, the domain and segment numbers can be obtained by executing the NLL commands LIST-DOMAIN or LIST-SEGMENT.

The reasons for splitting a domain in several segments are many:

- The more time critical parts of a program may be kept permanently in memory (fixed segments), while other parts may be regular demand segments

- A segment may be part of several domains. Thus, file space is required for one copy only, rather than including the data or routines (for example the Fortran library) in every domain.

- At run time, the Monitor will recognize a program segment used by several users concurrently, and keep only one copy in memory, thereby reducing swapping.

- Different segments may be given different protection against other users.

- Two programs running concurrently may communicate through a shared data segment (Normally, however, each program would have his own copy of the data segment).

- Program modularization is simplified.

- Modifications of routines or data in one segment will not require a reloading of the whole domain (unless it has been marked as sensitive to modifications).

- No swap file space is required for the program segment; it is read directly from the :PSEG file and never written back. Thus, swap file space is saved and no unnecessary rewrites are performed.

A segment will always be declared in one domain. If other domains need routines or data in this segment, references are defined by linking this segment to the other domain through the NLL command LINK-SEGMENT. The linked segment may also belong to another user, for example user SYSTEM may have a segment with library subroutines that other users may link to.

Linking is possible only if the segment has no external references to other segments in the domain where it was created, unless all these segments are also linked.

ND-500 hardware allows a segment to be used as an "indirect" segment. Call to an indirect segment implies a change of control to another domain, and is used for building a program system consisting of several domians. This mechanism is not used under the Sintran III operating system.

The indirect segment concept is, however, used for operating system requests: "monitor calls" are calls to routines on a system segment used as an indirect segment. Thus, monitor calls look exactly like regular routine calls, and parameters are transferred through the same mechanisms. By convention (although not by necessity), segment number 31 (octal 37) is used for interfacing to the operating system.

2.2. Capabilities

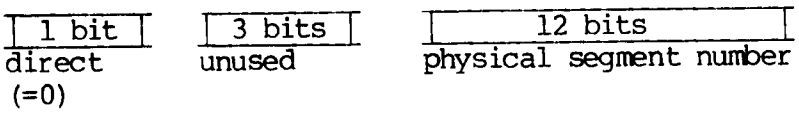
During execution, ND-500 will keep a 16 bit descriptor, called a capability, for each logical segment in use. This capability contains information about access rights, location in physical segments and sharing with other processes.

Each data segment may be individually protected against write access and access to subroutines parameters. If the segment is used concurrently by several processes, the capability will inform the CPU that data accesses should go directly to memory rather than through the cache. This is done to prevent that one process' updating of a variable is immediately observed by the other processes; the cache is not neccessarily cleared when another process gains access to the CPU, and the one process may also be running in ND-100.

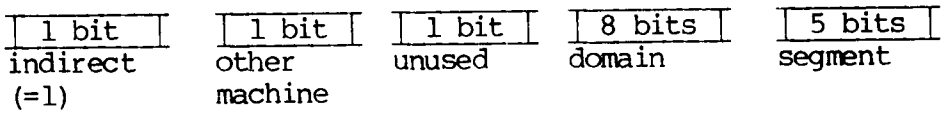
A program segment is identified as a direct or indirect segment. A direct segment is part of the current domain, while an indirect segment is part of another domain (in Sintran III: in the ND-100). This mechanism is used by the operating system to implement a set of monitor calls: Logically, the routines are addressed within the address space of the current domain, but when such a routine is called, the microprogram will recognize the segment as indirect, and transfer control to the appropriate domain. ND-100 is in this respect considered another domain. The capability contains an explicit indication that the other domain is in another machine.

Program segment capability:

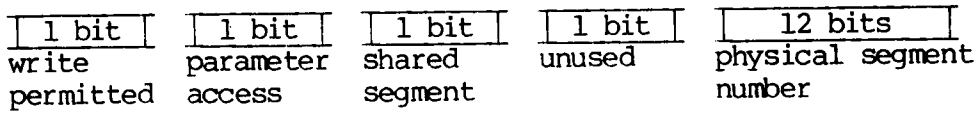
a) Direct segment



b) Indirect segment



Data segment capability:



Both data and program segment capabilities also indicate which physical segment that is addressed. A physical segment is a part of (physical and virtual) memory; a logical address is translated to a physical address in the physical segment.

Two logically separate segments may map onto the same physical segment. This will appear as the capabilities of the two segments pointing to the same physical segment. The physical segment number is determined when the segment is brought into memory for execution. Sharing a segment in this manner may reduce swapping, and it may be used for communicating data between processes.

When routines on a program segment is started, the Monitor will normally check whether the physical segment has already been fetched by some other process. If it has, no new copy is required, and the second segment maps onto the physical segment already in memory. (This relies upon program segments being read-only - if any modification (patching) is done to the program segment, the user will receive his own private copy.)

Data segments will not unless explicitly specified be mapped onto the same physical segments, as one process' modification of a location will have an immediate effect for other processes' use of the value. Sharing a physical segment is, however, the most direct way of transferring data between processes. When accessing data in a shared segment, the cache is bypassed in order to ensure data consistency. If multiple CPUs have access to the memory, the multiport will ensure that a write or read operation of one location will not be interrupted by another process. (Higher level protection and synchronizing mechanisms may be implemented in software based on this hardware mechanism.)

If a logical data segment is mapped directly onto the file where it is stored (rather than to a copy on a swap file), modifications to the data will be permanent. By using a file as a segment any file may be manipulated; the cataloged file will be directly addressed as a part of the logical address space. Compared to ordinary file access, the overhead is reduced drastically, and addressing can be done easily and directly within the logical address space. Obviously, only one process at a time may modify a permanent file, or the two processes must have agreed upon a synchronization protocol.

3. TRAPS

Trap conditions are special situations detected by hardware, possibly requiring special handling. Examples of such situations are division by zero, protect violation or illegal index.

Some trap conditions may be completely ignored. Others require some form of handling, while still others are so serious that they are reported directly to the operating system. These three groups are labeled ignorable, non-ignorable and fatal, respectively.

Trap conditions may be handled by a routine in the current domain, or propagated to the ND-100. The presence of a local trap handler routine is signalled by setting the bit in the OTE register (Own Trap Enable) corresponding to the trap condition. This register has one bit for each possible trap condition.

If the OTE bit is cleared, the trap is propagated to the ND-100 if the MTE bit (Mother Trap Enable) is set, signifying that the ND-100 has a trap handler. Otherwise the trap is ignored.

ND-100 may limit the ND-500 domains' freedom to modify bits in the OTE register (and thereby the handling of the trap condition), by clearing the corresponding bit in the TEMM register (Trap Enable Modification Mask). Fatal traps may never be locally enabled in ND-500.

3.1. Trap handler calling

When a trap condition occurs, the calling of a handler is determined by the setting of the MTE and OTE registers. If the affected bit is reset in both registers, no trap handler is called and the trap condition ignored.

If the OTE bit is set, a routine in ND-500 is called. This routine may be written by the user, or may be loaded or linked from a library of standard trap handlers.

If the MTE bit is set and not the OTE bit, ND-100 will take care of the trap condition.

When a trap condition is taken care of in ND-500, the address of the trap handler is found in a table pointed to by the THA (Trap Handler Address) register. The n'th entry in this table contains the address of the handler for the n'th trap condition. One handler may take care of several traps, or each trap condition may be handled by a separate routine.

The routine may perform any operation, including calling subroutines, but if a trap condition occurs during the execution of this routine, the trap is unconditionally reported to ND-100. The reason for this is that the local data area for a trap handler is fixed in the space above the table containing the trap handler start addresses; trap handlers are thus not reentrant.

At the call of the trap handler the local data area will be initialized with information about the trap and the state of the process when the trap occurred. The layout of this information is described in the ND-500 CPU Reference Manual ND-05.009.

3.2. Use of trap handlers

Writing a handler for a trap condition will require a familiarity with the instruction set and call mechanisms of the ND-500. Reading the values in the local data area of the trap handler (containing the register block and data about the trap) is most easily done in assembler, but may in principle be done in any language.

Most often the user will want to handle the error on a more abstract level. A standard trap handler library will take care of the low level trap handling, and call an exception handler routine. These will present hardware and software detected errors to the user in a uniform way. The standard routines may perform all the error handling or take care of a subset of errors, they provide mechanisms for entering the address of user written routines into the table of handlers and for setting and resetting bits in the OTE register.

Trap handler routines and enabling/disabling of traps may be defined at load time or before execution is started. These settings act as default values that may be modified by the program at execution time.

The standard library will also provide routines for errors detected by software. Such errors are usually very dependent on the application (for example errors in the correspondence between the IO-list and the FORMAT statement in Fortran), and rely upon instructions generated by the compiler. The standard way of reporting errors that occured in a routine is to set the K bit in the status register and leave an error code in the I1 register.

The combination of hardware trap handling and handling of software detected errors allows a uniform interface to the environment, regardless of the mechanism used for detecting the error.

The term used to cover both hardware and software detected errors is exception, consequently the standard library is termed a standard exception handler library.

4. STANDARD EXCEPTION HANDLER LIBRARY

The term exception covers in addition to all defined hardware traps, special situations and errors detected by software. An exception handler is a routine to be activated when an exception occurs, and to take appropriate recovery actions.

A set of standard routines for use with Fortran or Planc has been developed. These are available in a standard library, and will be linked automatically if the user so desires.

For each error condition, the user may determine:

1) The number of times each error message is to be printed.
2) The number of times an error may occur before the program is abnormally terminated.
3) Whether a user-supplied exception handler is to be activated upon detection of an error.
4) Whether traceback of routine stack frames is to be printed when the error occurs or when the program terminates, (In case of traps, this includes a register dump).
5) Printout of error statistics when the program terminates.

The library consists of the following routines:

EXCEPT - disable/enable handling of specified exception,
EXCDEF - reset handling of exception to default,
EXCTERM - define action to be taken upon program termination,
PRITRAC - print traceback of routine instances (subroutines),
PRIMESS - print error message,
GETMESS - return error text (Fortran),
PGETMESS - return error text (Planc),
RDEFVAL - read default exception handling parameters values,
RCURVAL - read current exception handling parameters values.

In the following descriptions, the header of these routines is described, giving the number and types of the arguments. These routines are supplied with the standard ND Fortran library. Except where designated as returned values, all parameters are read-only input values.

Where routines are used as parameters, the name of the routine is supplied in the actual parameter list. The compiler will generate the appropriate reference to the entry point of the routine.

Traps and exceptions will be handled in the ND-500, providing they are locally enabled. There are default settings for all traps. If no local handling has been specified, or the trap has been disabled, then some traps may be handled as a system trap in the ND-100. The Monitor will then handle the trap in a standard manner, depending on the type of trap. System traps may also be disabled, but the user's right to modify trap handling may be restricted.

Handling of traps may be determined at load time or before execution through the commands LOCAL-TRAP-ENABLE, LOCAL-TRAP-DISABLE, SYSTEM-TRAP-ENABLE and SYSTEM-TRAP-DISABLE. These commands are available both in NLL and the Monitor, and to set default values to be used if no action is taken by the program.

4.1. ND-500 traps table

The following is a list of defined hardware traps, their corresponding bit number in the status, OTE, MTE and TEMM registers, and the name of the trap. For a more detailed explanation, see the ND-500 CPU Reference Manual ND-05.009.

Bit no.		Name	D	msg	err
9	11B	OVERFLOW		10	unl
11	13B	INVALID OPERATION	*	10	unl
12	14B	DIVISION BY ZERO	*	10	unl
13	15B	FLOATING UNDERFLOW		10	unl
14	16B	FLOATING OVERFLOW		10	unl
15	17B	BCD OVERFLOW		10	unl
16	20B	ILLEGAL OPERAND VALUE	*	10	unl
17	21B	SINGLE INSTRUCTION TRAP		0	unl
18	22B	BRANCH TRAP		0	unl
19	23B	CALL TRAP		0	unl
20	24B	BREAKPOINT INSTRUCTION TRAP		0	unl
21	25B	ADDRESS TRAP FETCH		0	unl
22	26B	ADDRESS TRAP READ		0	unl
23	27B	ADDRESS TRAP WRITE		0	unl
24	30B	ADDRESS ZERO ACCESS	*	10	unl
25	31B	DESCRIPTOR RANGE		10	unl
26	32B	ILLEGAL INDEX	*	1	0
27	33B	STACK OVERFLOW	*	1	0
28	34B	STACK UNDERFLOW	*	0	0
29	35B	PROGRAMMED TRAP		0	unl
30	36B	DISABLE PROCESS SWITCH TIMEOU	*	1	0
31	37B	DISABLE PROCESS SWITCH ERROR	*	1	0
32	40B	INDEX SCALING ERROR	*	1	0
33	41B	ILLEGAL INSTRUCTION CODE	*	1	0
34	42B	ILLEGAL OPERAND SPECIFIER	*	1	0
35	43B	INSTRUCTION SEQUENCE ERROR	*	1	0
36	44B	PROTECT VIOLATION	*	1	0

The "D" column refers to the default enabling of traps used by the standard exception handler library discussed in the next chapter. "*" indicates that the trap is enabled if the default trap library settings are used.

msg = default maximum number of error messages
err = default number of exceptions prior to abnormal termination
unl = unlimited number

4.2. The EXCEPT routine

The EXCEPT routine is used to modify the current exception handling conditions.

PLANC specification:

```
TYPE RTYP = ROUTINE REFERENCE VOID,VOID (INTEGER)
ROUTINE  REFERENCE VOID, VOID (INTEGER, INTEGER, RTYP POINTER, &
         INTEGER, INTEGER, BITS POINTER):                      &
EXCEPT   (EXCNO,EXCFUN,EXCROUT,NOMSG,NOEXC,EXCARR)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
  SUBROUTINE EXCEPT(EXCNO,EXCFUN,EXCROUT,NOMSG,NOEXC,EXCARR,
+                   EXCNOL,EXCNOH)
  INTEGER EXCNO,EXCFUN,EXCROUT,NOMSG,NOEXC,EXCNOH,EXCNOL
  LOGICAL EXCARR(EXCNOL:EXCNOH)

<standard library routine>

  END
```

Parameter values:

EXCNO	Exception number or exception number group:	
	0	default group of traps to be set (see section 4.1)
	4	LOGICAL array (EXCARR, EXCNOL and EXCNOH must be present, Fortran)
	5	BITS POINTER (EXCARR must be present, Planc)
	11B:44B	specific trap number
	400B	all FTN errors
	401B:457B	specific FTN error
	other	illegal
EXCFUN	Function:	
	-1	disable exception(s) indicated by EXCNO and ignore all other exceptions. Further, the parameters EXROUT, NOMSG and NOEXC will be ignored.
	0	enable exception(s) indicated by EXCNO as TRUE, set new handler/values, and disable all other exceptions which are indicated as FALSE. For EXCNO values 11B:44B or 401B:457B, only the single values 11B:44B or 401B:457B, only the single exception thus specified, is enabled.
	1	enable exception(s) indicated by EXCNO, do not modify handler/values, and ignore all other exceptions.
	other	illegal

EXCROUT User defined exception handler routine
- ><0 routine address (supplied by routine name in the source program)
- 0 no routine supplied

NOMSG Number of messages allowed before program is aborted
- -1 any number of messages allowed
- >= 0 number of messages allowed (<2**31-1)
- other illegal

NOEXC Number of traps before program is aborted:
- -1 any number of traps allowed
- >= 0 number of traps allowed (<2**31-1)
- other illegal

EXCARR LOGICAL array (Fortran) or BITS POINTER (Planc) containing TRUE or FALSE for exceptions to be handled

EXCNOL (Fortran) Low limit of EXCARR

EXCNOH (Fortran) High limit of EXCARR

The handling of one or several exception conditions may be modified, selected through the EXCNO parameter. If this parameter is 4 (Fortran) or 5 (Planc), the EXCARR parameter selects a set of exceptions to be handled. If the EXCFUN parameter is zero and EXCARR is present, the elements set to TRUE in this array will cause the corresponding exception to be enabled, while FALSE will cause it to be disabled.

The EXCROUT parameter specifies the name of a user supplied routine to be executed when the exception occurs. The routine should conform to the following formal specification:

In Fortran:

```
SUBROUTINE name(ierno)
INTEGER ierno
    .
<user written exception handler>
    .
END
```

In Planc:

```
ROUTINE REFERENCE VOID, VOID (INTEGER): name (ierno)
    .
<user written exception handler>
    .
ENDROUTINE
```

The parameter <ierno> will transfer the error number to the exception handler. If the EXCROUT parameter is zero, the standard exception handler routine from the library is used.

After an error has occurred, the sequence of operations is as follows; the steps marked with an asterisk apply to traps only:

Note: the details are slightly different in Planc

1) * If the exception is a trap, the trap routine is activated.

2) A system provided exception handler is called.

3) This handler updates the occurence counter for this type of exception and activates the user exception handler if one has been specified.

4) If the traceback condition (see note 1) is true, the system outputs:
   * - register dump
     - traceback printout

5) If the message occurrence limit (NOMSG) has not been exceeded, or if the traceback condition (see note 1) is true, an error message is printed.

6) If the error count is less than or equal to the allowed number of occurences for this exception type, control is returned to normal FORTRAN error handling,

   otherwise, the program is abnormally terminated with error statistics, if specified.

Note that in Fortran the STACK UNDERFLOW trap condition is handled by special software mechanisms and must, in order to ensure correct termination of the I/O activities, always be enabled.

note 1: the traceback condition is evaluated by the following expression:

```
thiserror >< 'STACK UNDERFLOW' and
( ( TRACEBACK=2 and
    ( thiserror.NOMSG = unl or
      thiserror.numerrors in 0:thiserror.NOMSG )  )
  or
  ( TRACEBACK >= 1 and
    ( thiserror.NOEXC >< unl and
      NOT thiserror.numerrors in 0:thiserror.NOEXC )  )
)
```

where

thiserror.numerrors is the current value of the number of exceptions of this type which have occurred.

EXAMPLES, Fortran

1. Enable DIVISION BY ZERO trap using default exception values:

```
C DIVISION BY ZERO is trap number 12
    CALL EXCEPT(12,1)
```

2. Enable OVERFLOW and allow maximum 2 error messages and 10 overflow errors before abnormal termination. Activate the user defined routine MYROUT each time the overflow trap occurs.

```
      CALL EXCEPT(9,0,MYROUT,2,10)
```

3. Disable error handling for exponential functions, Fortran error numbers 431B, 432B, 433B, 437B:

```
      LOGICAL ERRARRAY(431B:437B)
      DATA ERRARRAY/.FALSE. , .FALSE. , .FALSE. , .TRUE. ,
     +              .TRUE. , .TRUE. , .FALSE./

      CALL EXCEPT(4,-1,0,0,0,ERRARRAY,431B,437B)
```

4. Manipulation of some exception settings.

Assume the following is the current table settings for exceptions:

exc. no. (octal)	EXCROUT setting	msg	err	setting
...				
431	A	10	unl	enabled
432	A	10	unl	enabled
433	A	10	unl	enabled
434	0	10	20	disabled
435	0	10	unl	enabled
436	0	10	unl	disabled
437	0	10	50	enabled
...				

If the following call were executed,

```
      CALL EXCEPT(4,0,MYROUT,5,-1,ERRARRAY,431B,437B)
    C ERRARRAY as declared in previous example
```

then the table settings would be changed as follows,

exc. no. (octal)	EXCROUT setting	msg	err	setting
...				
431	A	10	unl	disabled
432	A	10	unl	disabled
433	A	10	unl	disabled
434	MYROUT	5	unl	enabled
435	MYROUT	5	unl	enabled
436	MYROUT	5	unl	enabled
437	0	10	50	disabled
...				

4.3. The EXCDEF routine

EXCDEF is used to set the default exception handling values for a given set of exceptions. This is functionally equivalent to calling EXCEPT with the default parameter values for each of the traps specified, but is more convenient and relieves the programmer from knowing the defaults.

PLANC specification:

```
ROUTINE REFERENCE VOID, VOID(INTEGER, BITS POINTER):        &
        EXCDEF(EXCNO, EXCARR)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
SUBROUTINE EXCDEF(EXCNO, EXCARR, EXCNOL, EXCNOH)
INTEGER EXCNO, EXCNOL, EXCNOH
LOGICAL EXCARR(EXCNOL:EXCNOH)

<standard library routine>

END
```

Parameter values:

EXCNO — Exception number or exception number group:

0	default setting (see section 4.1)
4	LOGICAL array (EXCARR and EXCNOH present, Fortran)
5	BITS POINTER (EXCARR present, Planc)
11B:44B	specific trap number
400B	all FTN errors
401B:457B	specific FTN error
other	illegal

EXCARR — LOGICAL array (Fortran) or BITS POINTER (Planc) containing TRUE for exceptions to be handled, FALSE for those that should remain as they are

EXCNOL — (Fortran) Low limit of EXCARR

EXCNOH — (Fortran) High limit of EXCARR

The EXCARR parameter selects a set of exception conditions, like in the EXCEPT routine. Alternatively, one specific exception may be selected through the EXCNO parameter.

EXAMPLES, Fortran:

1. Reset handling of all traps and Fortran errors to default:

```
C All traps
      CALL EXCDEF(0)
C All Fortran errors
      CALL EXCDEF(400B)
C set default program termination conditions
      CALL EXCTERM(0,1,20)
```

This setting is identical to the setting at the beginning of execution of a Fortran program.

2. Reset special error handling for exponentiental functions, error numbers 431B, 432B, 433B and 437B, but keep possible special handling of other exceptions:

```
      LOGICAL ERRARRAY(431B:437B)
      DATA ERRARRAY/.TRUE.,.TRUE.,.TRUE.,.FALSE.,
     +              .FALSE.,.FALSE.,.TRUE./

      CALL EXCDEF(4,ERRARRAY,431B,437B)
```

4.4. The EXCTERM routine

EXCTERM may be called to determine the printing of traceback and error statistics information. If it has been called more than once, the last call applies at program termination.

PLANC specification:

```
ROUTINE REFERENCE VOID,VOID(INTEGER,INTEGER,INTEGER): &
                  EXCTERM(TRACEBACK,PRSTAT,NOLEV)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
SUBROUTINE EXCTERM(TRACEBACK,PRSTAT,NOLEV)
INTEGER TRACEBACK,PRSTAT,NOLEV

<standard library routine>

END
```

Parameter value:

TRACEBACK — traceback print, for all errors:
- 0 : no traceback (default)
- 1 : traceback upon abnormal termination
- 2 : traceback upon error
- other : illegal

PRSTAT — error statistics print at program termination, for all errors:
- 0 : no statistics output
- 1 : print statistics (default)
- other : illegal

NOLEV — maximum number of levels to process when a traceback is provided.
- > 0 : maximum number of stack levels to print, default 20
- other : not valid

4.5. The PRITRAC routine

PRITRAC is a utility routine to print a traceback of routine instances (stack frames). The routine is called from a user handler, or automatically upon abnormal termination of the job if traceback has been selected (in the EXCEPT call referring to the exception condition raised).

PLANC specification:

```
ROUTINE REFERENCE VOID, VOID (BOOLEAN READ): PRITRAC (TRAP)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
SUBROUTINE PRITRAC (TRAP)
LOGICAL TRAP

<standard library routine>

END
```

Parameter value:

TRAP	TRUE if called while a trap is being handled. FALSE should be set for any other condition.

Note that the default maximum number of stack levels to be printed is 20.

4.6. The PRIMESS routine

The PRIMESS routine will print the error message, corresponding to the parameter value, on the standard output device (unit 1).

PLANC specification:

```
ROUTINE REFERENCE VOID, VOID (INTEGER): PRIMESS(EXCNO)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
SUBROUTINE PRIMESS (EXCNO)
INTEGER EXCNO

<standard library routine>

END
```

Parameter values:

EXCNO exception number

The parameter (EXCNO) must be in the range 11B:44B (traps) or 401B:457B (FORTRAN errors).

4.7. The GETMESS/PGETMESS routine

PGETMESS/GETMESS will return the error text corresponding to the specified exception number.

PLANC specification:

```
ROUTINE VOID,BYTES POINTER(INTEGER): PGETMESS(EXCNO)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
      FUNCTION GETMESS(EXCNO)
C this function must be declared to be of type character in the
C calling program
      INTEGER EXCNO
      CHARACTER *(*) GETMESS

      <standard library routine>

      END
```

Parameter values:

EXCNO the number of an exception condition

EXCTEXT (Fortran; out-value in Planc:)
Return parameter containing the error text

EXCNO must be the number of a defined exception condition, in the range 11B:44B (traps) or 401B:457B (Fortran error).

4.8. The RDEFVAL routine

PLANC specification:

```
ROUTINE  REFERENCE VOID,VOID (INTEGER, INTEGER WRITE,
      INTEGER WRITE, INTEGER WRITE, INTEGER WRITE, INTEGER WRITE):
RDEFVAL  (EXCNO, NOMSG, NOEXC, TRACEB, PRSTAT, NOLEV)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
SUBROUTINE RDEFVAL(EXCNO, NOMSG, NOEXC, TRACEB, PRSTAT, NOLEV)
INTEGER EXCNO, NOMSG, NOEXC, TRACEB, PRSTAT, NOLEV

<standard library routine>

END
```

Parameter values:

EXCNO exception number

NOMSG default number of messages allowed
(returned value)

NOEXC default number of exceptions allowed
(returned value)

TRACEB default (=0) traceback setting (all EXCNOs)
(returned value)

PRSTAT default (=1) error statistics setting (all EXCNOs)
(returned value)

NOLEV default (=20) maximum number of levels to be printed during a traceback
(returned value)

RDEFVAL may be called to read the default values of the exception parameters corresponding to a given exception number (EXCNO).

4.9. The RCURVAL routine

RCURVAL may be called to read the current values of the exception parameters corresponding to a given exception number (EXCNO).
PLANC specification:

```
TYPR RTYP = ROUTINE REFERENCE VOID,VOID (INTEGER)
ROUTINE  REFERENCE VOID,VOID (INTEGER, RTYP POINTER,                &
         INTEGER WRITE, INTEGER WRITE, INTEGER WRITE,               &
         INTEGER WRITE, INTEGER WRITE, INTEGER WRITE):              &
RCURVAL  (EXCNO,EXCROUT,NOMSG,NOEXC,TRACEB,PRSTAT,NOLEV,EXCCOUNT)

<standard library routine>

ENDROUTINE
```

FORTRAN specification:

```
  SUBROUTINE RCURVAL(EXCNO,EXCROUT,NOMSG,NOEXC,TRACEB,PRSTAT,NOLEV,
+                    EXCCOUNT)
  INTEGER EXCNO,EXCROUT,NOMSG,NOEXC,TRACEB,PRSTAT,NOLEV,EXCCOUNT

<standard library routine>

  END
```

Parameter values:

EXCNO exception number

EXCROUT address of current user exception handler or zero (supplied as a routine name in the source program)

NOMSG current number of messages allowed before abnormal termination (returned value)

NOEXC current number of exceptions allowed before abnormal termination (returned value)

TRACEB traceback setting (for all EXCNOs), see section 4.4 (returned value)

PRSTAT status report print upon end of program (for all EXCNOS) (returned value)

NOLEV current setting of maximum number of levels to be printed during traceback (returned value)

EXCCOUNT current exception count (returned value)

FORTRAN EXCEPTIONS:

dec	oct		msg	err	
257	401	FATAL FORMATTING SYSTEM ERROR	1	0	
258	402	TOO LOW PARENTHESES LEVEL IN FORMAT	1	0	
259	403	ILLEGAL CHARACTER IN FORMAT	1	0	
260	404	ILLEGAL TERMINATION OF FORMAT	1	0	
261	405	OUTPUT RECORD SIZE EXCEEDED	10	unl	
262	406	FORMAT REQUIRES GREATER INPUT RECORD	10	unl	
263	407	INTEGER OVERFLOW ON INPUT	10	unl	
264	410	INPUT RECORD SIZE EXCEEDED	10	unl	
166	412	BAD CHARACTER ON INPUT	10	unl	
267	413	REAL OVERFLOW ON INPUT	10	unl	
268	414	REAL UNDERFLOW ON INPUT	10	unl	
270	416	REAL OVERFLOW ON OUTPUT	10	unl	
271	417	FORMAT SPECIFICATION DOES NOT APPLY	1	0	
272	420	OVERFLOW IN EXPONENT ON INPUT	10	unl	
274	422	TOO MANY FILES OPENED	-	-	ND-100 only
275	423	EXCEPTION NUMBER OUT OF RANGE=	10	unl	
276	424	MIXING OF BINARY/ASCII ILLEGAL	1	0	
277	425	NO MORE BUFFERS AVAILABLE	1	0	
281	431	ZERO BASE AND NEGATIVE EXPONENT	10	unl	
282	432	BASE LESS THAN ZERO IN EXPONENTIATION	10	unl	
283	433	OVERFLOW IN EXPONENTIATION	10	unl	
284	434	NEG. ARG. IN SQUARE ROOT	10	unl	
285	435	TOO LARGE ARG. IN SINE	10	unl	
286	436	TOO LARGE ARG. IN COSINE	10	unl	
287	437	TOO LARGE ARG. IN EXP-FUNCTION	10	unl	
288	440	ZERO OR NEG. ARG. IN LOGARITHM	10	unl	
289	441	BOTH ARGS. ZERO IN ARCTAN	10	unl	
293	445	TOO LARGE ARG. IN HYPERB. TAN	10	unl	
294	446	TOO LARGE ARG. IN HYPERB. SINE	10	unl	
295	447	TOO LARGE ARG. IN HYPERB. COSINE	10	unl	
302	456	ILLEGAL ARG. IN ARC-SINE/COSINE	10	unl	
303	457	ILLEGAL ARG. IN TAN	10	unl	

msg = default maximum number of error messages
err = default number of exceptions prior to abnormal termination
unl = unlimited number

Numbers not listed are currently not used. All Fortran errors are default enabled.

All languages:

The hardware traps are listed in section 4.1.

5. COMMUNICATION BETWEEN ND-500 AND ND-100

There are several ways of transmitting information between the ND-500 and the ND-100; the selection of a method depends on the transmission speed required, the requirements and privileges of the sending and the receiving process, and above all, the amount of data to be transmitted.

5.1. Monitor calls

This is the simplest and, for the programmer, most direct way of communicating through the operating system mechanisms, or with the operating system itself. A monitor call will look exactly like a regular subroutine call to a routine in an indirect segment. The services provided are the same as in a ND-100 system. Monitor calls are used in connection with semaphores, internal devices, reserving and using external devices, file I/O and for starting and stopping RT programs in the ND-100.

When a monitor call is executed, the ND-500 process is suspended and a twin process in the ND-100 is started to execute the call on behalf of the ND-500 process. Some monitor calls may allow the ND-500 process to continue while the call is executed if the function code is selected accordingly.

The starting and stopping of a ND-100 process is rather time consuming, and monitor calls should be used for small amounts of data only, or for setting up other communication channels. The overhead is essentially constant regardless of the number of bytes transferred, as long as this number is moderate.

All Sintran III VSE/500 systems are delivered with the OUTST monitor call (MON 162). This is used by the standard libaries, rather than OUTBT. The programmer using monitor calls explicitly is advised to utilize OUTST if possible. OUTST will cause activation of the twin process for each string to be transferred, while OUTBT will activate it for each byte transferred.

5.2. Communicating through the process flags

Each process running in ND-500 has two 32 bit words assigned for communication purposes. These are termed the input flag and the output flag. Monitor calls and commands are available to read and write these flags. The flags are not used by the monitor, and may contain any bit pattern the user desires.

The input flag of a process is used for signalling to a running ND-500 process. This flag may be written by an ND-100 process or through commands, and read by the process itself. The output flag is used for returning data or status, and is written by the process. This flag may be read by ND-100 or through commands, but may not be written.

There is no queueing - if another value is written in the flag word before it is read, the first value is overwritten.

5.3. Communicating through RTCOMMON

This is the fastest method of communication, as reading and writing is directly to the location accessed by the ND-100, and this part of memory is always resident. The only limiting factor is the size of the RTCOMMON area.

The RTCOMMON area is accessed from the ND-500 as a part of the regular memory space. The mapping onto the RTCOMMON is done at load time through the MATCH-RTCOMMON command, used before any loading to the segment is done.

No modification of the size of RTCOMMON should be done after the segments referring to it have been loaded. If such modifications are done, the segments must be reloaded. Segments using RTCOMMON can not, in general, be moved to another machine after loading.

If the RTCOMMON area is used from ND-500, it must be contiguous. In other words, if the system supervisor through the SINTRAN-SERVICE-PROGRAM command DEFINE-RTCOMMON-SIZE expands RTCOMMON beyond what was specified at system generation, this area must be adjecent to the initially allocated area.

5.4. Communicating through an RT segment

An ND-100 RT-program may share data with an ND-500 process through a segment in one of the ND-100 SEGFILs. The segment must be fixed in a continous area in memory before the ND-500 process referring to it is started.

This is the most efficient way of transferring larger amounts of data between the two processors. Access to the area should be protected by semaphores; this is done through monitor calls.

The symbols defined by the ND-100 RT-LOADER are available to the ND-500 process after the MATCH-COMMON-RT-SEGMENT command has been given. This command should be given after these symbols have been defined in the ND-100, but before any loading to the ND-500 segment is done.

If two (or more) ND-100 segments are matched with one ND-500 segment, they must be fixed in memory at physical addresses with a fixed distance equal to the distance between them in ND-500 address space.

5.5. Communicating through files

All files are common to the two processors, and the same regulations apply to processes running in the two CPUs as to processes running in the same CPU. Files provide for transfer of arbitrarily large amounts of data, but are significantly slower than the other methods.

In order to speed the file access, the file may be opened with direct transfer (open modus 8 or 9). This puts some restrictions on the application program, but allows the transfer to go directly to memory, circumventing a major part of the file system. However, the user must do most of the bookkeeping himself, and the file system provides no structuring of the disk pages. The transfer speed will, however, when the block size is large, approach the hardware speed of the disk.

Programs using direct transfer may also allow a higher number of simultaneously opened files. Direct transfer is also available for magnetic tape.

When direct transfer is used, the Monitor will automatically fix the memory buffer in a contiguous part of memory before the first transfer, and it will remain fixed until the program terminates.

6. LOADER COMMANDS

Although the set of commands available in NLL is large, most users need only a couple of them. The most important are

SET-DOMAIN	-	name an executable domain
LOAD-SEGMENT	-	load a file containing relocatable code
EXIT	-	return to Sintran III

The EXIT command is described in the next chapter.

Various error messages may be returned from NLL during or after command interpretation and execution. These error messages are listed in chapter 13, with short explanations and references to where they may occur.

6.1. Domains

6.1.1. SET-DOMAIN

SET-DOMAIN (<domain name>)

<domain name> - the name of the domain to be set as the current domain, 1 to 16 alphanumeric characters or hyphen. Default name is SCRATCH-DOMAIN.

The domain with name <domain name> is set to be the current domain. The subsequent segment handling, loading and linking will be done in the current domain. <domain name> cannot include the directory and user name; loading may be done only in the domains of the current user.

If a domain is already set when the SET-DOMAIN command is executed, it is closed by an implicit END-DOMAIN.

The default domain name is SCRATCH-DOMAIN. If a LOAD-SEGMENT command is given when there is no current domain, an implicit SET-DOMAIN SCRATCH-DOMAIN is performed, and the segment SCRATCH-SEGMENT. This will delete all information previously loaded, using default names. Thus, a domain to be permanently retained will usually be given another name, to prevent it from being destroyed when default domain name is used.

A user may have a maximum of 256 domains. New domains are specified by enclosing the domain name in double quotes.

6.1.2. **END-DOMAIN**

END-DOMAIN

Finishes operation upon the current domain. END-DOMAIN automatically executes the command CLOSE-SEGMENT. END-DOMAIN will automatically be executed by the commands SET-DOMAIN, and EXIT.

6.1.3. **CLEAR-DOMAIN**

CLEAR-DOMAIN <domain name>

<domain name> - the name of the domain to be cleared, 1 to 16 alphanumeric characters or hyphen.

All segments which the domain <domain name> consist of are deleted from the domain. The segment (:PSEG, :DSEG, :LINK) files are retained.

This command may not be executed when a domain is set. <domain name> cannot be SCRATCH-DOMAIN, and must belong to the current user. (Individual segments in SCRATCH-DOMAIN may be cleared by the CLEAR-SEGMENT command.) It is assumed to exist in the description file of the current user. The domain continues to exist, but no longer comprises any segments.

6.1.4. **DELETE-DOMAIN**

DELETE-DOMAIN <domain name>

<domain name> - the name of the domain to be deleted, 1 to 16 alphanumeric characters or hyphen.

All segments in the domain are deleted, the domain itself is then also deleted. The segment (:PSEG, :DSEG, :LINK) files are retained. This command may not be executed when a domain is set. <domain name> cannot be SCRATCH-DOMAIN, and it must belong to the current user.

6.1.5. LIST-DOMAIN

LIST-DOMAIN (<domain name>)

<domain name> - the name or abbreviation of names of domains to be listed. Default is all domains created by the current user.

Writes all domains with names matching <domain name> and their start addresses (if any) on the output device. Default is all domains created by the current user.

This command can only be used to list domains belonging to the current user. To list domains belonging to other users, use the command LIST-SEGMENT, prefixing the <segment name> parameter with the user name in parentheses.

6.1.6. WRITE-DOMAIN-STATUS

WRITE-DOMAIN-STATUS [(<domain name>)]...

<domain name> - the name of the domain about which information is requested. Default is the current domain.

Prints all the available information about the domain or domains specified. These are assumed to exist in the description file of the current user. If no parameters are given, the current domain-status is printed. Note that during a linking/loading session the domain-entry or the segment-entry is not fully updated until the commands END-DOMAIN and CLOSE-SEGMENT, respectively, are executed.

6.1.7. RENAME-DOMAIN

RENAME-DOMAIN <new domain name>, <old domain name>

<new domain name> - the new name of the domain, 1 to 16 alphanumeric characters or hyphen.

<old domain name> - the name of an existing domain

Renames the domain <domain name>. The domain is assumed to exist in the description file of the current user.

6.1.8. COPY-DOMAIN

COPY-DOMAIN <destination domain>, <source domain>

<destination domain> - the name of a domain to receive a copy of the <source domain>. May not be prefixed with directory or user name.

<source domain> - the name of the domain to be copied. May be prefixed by directory and/or user name.

Copies the entire <source domain> to <destination domain>. <source domain> may be prefixed with a user name or directory:user name in parentheses. If the destination domain already exists, the segments on this domain must have the same names as the segments on the source domain or default names, and they will be overwritten with the segments from <source domain>.

If the segments do not exist, they will be created. They will be given the names they have in <source domain>, unless these names were default names; the segments then will be given new default names according to the <destination domain> number. If the destination domain does not exist, <destination domain> must be enclosed in double quotes.

To move a domain from one installation to another, the domain to be moved must be described in a description file being moved with it. The user to which the copying is done must enter NLL to create a description file (if it is not already created) and then copy the domain(s) by prefixing the source domain with the relevant directory and/or user name.

As the description file contains the name of the user, if the Sintran III commands @RENAME-DIRECTORY and @RENAME-USER is used, the NLL-command RENAME-DEFAULT-DIRECTORY-AND-USER must be used to update the description file.

Domains making references to RTCOMMON or Sintran III/ND-100 segments should not be copied to other machines.

6.1.9. **RELEASE-DOMAIN**

RELEASE-DOMAIN <domain name>

<domain name> - the name of the domain to be released

This command is used if an error in the system has occurred (e.g. a system crash) leaving a domain in an open state with no user attached to it. The domain will therefore be unavailable for further use. This may also occur if a loading process was not terminated before a Sintran III command was executed that did not return control to NLL.

RELEASE-DOMAIN will force the domain to be closed even if the user issuing the command is not the one who are using it or have been using it.

RELEASE-DOMAIN should be used with great care, and if used inappropriately it may cause inconsistencies in the description file. In any case, the contents of the released domain must be considered unpredictable, and it should be reloaded before being used again.

6.2. Segments

This section describes commands that manipulate segments as a whole, either before or after the actual load operation. The commands that cause code to be loaded are described in the next section (Commands to load NRF code).

Commands in this section are mainly used in connection with multi-segment domains. The OPEN-SEGMENT command may be used to select a non-default name of the segment, but the rules for default names ensure that segment names never collide. If there is only one segment per domain, the user need not be concerned about segment names at all, and thus, need not use any of the commands in this section. The opening and closing of segments are done automatically.

6.2.1. OPEN-SEGMENT

OPEN-SEGMENT (<segment name>), (<segment attributes>)

<segment name> - the name of the segment to which subsequent loading should be done, 1 to 16 alphanumeric characters or hyphen. Default is SCRATCH-SEG-01 if current domain is SCRATCH-DOMAIN.

<segment attributes> - a string of the characters CDEMOPRSW. See below. Default is CW.

Prepares the segment <segment name> for loading, i.e. set <segment name> as the current segment. If the segment does not exist when this command is executed, the segment name must be enclosed in double quotes. If the segment was already contained in the current domain, all old information about the segment is erased. (To add more code to an already loaded segment, use the command APPEND-SEGMENT.)

The scratch domain is used if there is no current domain.

The <attributes> specifies the use of the segment, and consists of a string of option letters. The options are:

R Read Only data segment. May not be combined with W.

W Write allowed data segment. Default value. May not be combined with R.

O Use original data segment file for swapping. Modifications to data will be permanent. May not be combined with C.

C Copy data segment to swap file. Default value. May not be combined with O.

E Empty data segment. The data segment will be dynamically assigned at execution time.

P Shared program segment. May be included in another domain by the command LINK-SEGMENT. Only the program segment will be shared.

This command is only necessary if the segment will be linked to another domain than the current one.

M Other Machine. The program segment capability will at execution time indicate that the segment is located in another CPU. Monitor calls which are executed in ND-100 can be defined as an indirect segment in another machine.

D Shared Data. A linked segment will by default have only the program segment shared. This attribute declares the data segment as shared. If both program and data segments should be shared, PD must be specified.

F File

If an NRF file is loaded when there is a current domain (set by SET-DOMAIN) but no current segment (set by OPEN-SEGMENT), an implied command:

```
OPEN-SEGMENT SEGMENT-Dxxx-Syy R
```

is executed, where xxx is the number of the current domain and yy the logical segment number used. If the segment does not exist, it is created.

All information on the segment is deleted. The segment number may be forced by the command SET-SEGMENT-NUMBER; otherwise the first free segment number, starting at 1, is used. If the segment exists, the segment number will be retained.

If an NRF file is loaded when there is neither a current segment nor domain, two implied commands are executed:

```
SET-DOMAIN  SCRATCH-DOMAIN,,
OPEN-SEGMENT  SCRATCH-SEGMENT-01, CW
```

Code previously loaded to SCRATCH-SEGMENT-01 will be deleted. Thus, to prevent the contents of a segment from being destroyed next time anything is loaded to the segment using the default name, the segment should be explicitly named.

Note that the default name depends on the domain and segment numbers. Therefore, as long as each program is loaded to a different domain, default segment name may be used in each of the domains without interfering with segments in other domains.

OPEN-SEGMENT automatically executes CLOSE-SEGMENT if a current segment is open and COMMON-SEGMENT-CLOSE if one or more COMMON segments are open.

6.2.2. CLOSE-SEGMENT

CLOSE-SEGMENT [<Y/N>]

<Y/N> - Y will cause a load map to be written to the output file after all linking and loading is complete, N will suppress this. Default is N.

Terminates loading to the current segment. After this command has been executed, there is no current segment.

If the segment was not opened by APPEND-SEGMENT, a trap handler vector is allocated. If there are undefined references, the auto-link segments will automatically be linked. If there are still undefined references, the defined auto-load files (see SET-AUTO-LOAD-FILE) will automatically be loaded. Auto-link segments and auto-load files defined by the current user are first linked/loaded, and then those defined by user SYSTEM.

If there still are undefined references, an error message will be given. In a batch or a mode job all undefined references will be written to the output device and the command will be executed. In interactive mode a warning will be given and the command not executed. The second time the command is given it will always be executed.

The segments will be closed, all labels will be saved on the :LINK file in numerically sorted order, all other necessary information will be saved on the description file, and the correct file access will be set on the files involved. The KILL-ENTRIES and GLOBAL-ENTRIES commands may be used before the segment is closed to restrict the selection of labels saved on the :LINK file.

CLOSE-SEGMENT is automatically executed by END-DOMAIN, SET-DOMAIN, EXIT or OPEN-SEGMENT.

6.2.3. LINK-SEGMENT

LINK-SEGMENT <segment name> ...

<segment name> - the name of the segment to be linked to the current segment.

Links all modules on the segment <segment name> to the current segment. Routines and data areas defined on the segments listed will satisfy references on the current segment.

The linking can be done before or after loading the current segment; all symbols defined on the specified segments will be available until loading to the current segment is terminated (by CLOSE-SEGMENT). The <segment name>s specified must be already loaded segments.

The segments which are linked will be a part of the current domain and must have no external references to other segments if it is part of another domain. This means that if the linked segment originally is a part of another domain it cannot, itself, have linked and common segments. It can, however, have indirect segments. Linked segments may have linked segments in the current domain. Logically, a segment linked to more than one domain may be treated as if there were several identical copies of the segment, one in each domain.

There are no restrictions on external references if the linked segments are parts of the current domain. It is also possible to make two-way references between segments within one domain.

If a segment in another domain is linked, the segment number will be the same in the two domains. The segment number must therefore be available when the linking is done, except if the segment has been previously linked - a second LINK-SEGMENT command may be used to define new references since the first linking was done.

6.2.4. LIBRARY-SEGMENT-LINK

LIBRARY-SEGMENT-LINK <segment name> ...

<segment name> - the name of the segment to be linked to the current segment.

A LINK-SEGMENT command will make all labels in the specified <segment name> available in the current domain. This may cause name conflicts, and can make the space requirements for the name table grow very large.

LIBRARY-SEGMENT-LINK will define only those symbols actually referenced. Otherwise it works exacly as LINK-SEGMENT.

6.2.5. APPEND-SEGMENT

APPEND-SEGMENT (<segment name>) (<segment attributes>)

<segment name> - the name of an existing segment, to which more code will be added. Default is SCRATCH-SEG-01.

<segment attributes>- a string of the characters CDEMOPRSW. Default is the current attributes of the segment.

This command prepares <segment name> for further loading. All previously defined and referenced symbols are available, and the new code can be appended to the old code. <segment name> must exist when this command is executed.

<segment attributes> have the same meaning as for OPEN-SEGMENT. If a non-default value is specified, the attributes are changed, otherwise the existing attributes will not be modified.

Be aware that only the first 20 characters of a symbol will be saved on the :LINK file, thus, if the symbol name is longer than 20 characters it will not match with the full, un-truncated symbol when loading to the segment is resumed at a later time with this command.

The common and link segments defined when the segment was previously closed are not automatically restored, and must be explicitly defined by the user. In order to avoid clearing the common segment, COMMON-SEGMENT-APPEND should be used.

6.2.6. SET-SEGMENT-NUMBER

SET-SEGMENT-NUMBER (<segment number>)

<segment number> - a number in the range 0:37B to be the logical segment number of the next current segment. Default is 1.

Specifies explicitly the logical segment number for the program segment within a domain. This command can be used in connection with the command OPEN-SEGMENT. If the command SET-SEGMENT-NUMBER is not issued, the first free segment number is used.

In most cases, the user need not be concerned about the segment number used. However, if the next free segment number (i.e. the default segment number) is already used by a segment that will later be linked to the domain, the segment number must be set to another value.

6.2.7. CLEAR-SEGMENT

CLEAR-SEGMENT <segment name>

<segment name> - the name of an existing segment that is to be cleared.

The segment <segment name> will be cleared and readied for loading new code, i.e. all information about labels, start address, low address, and size is deleted. Pages allocated to the segment files will also be released, but the file will be retained.

6.2.8. DELETE-SEGMENT

DELETE-SEGMENT <segment name>

<segment name> - the name of an existing segment that is to be removed.

All information and the files making up the segment <segment name> are deleted. The space on the domain which the segment was a part of is released.

This command is not legal if a domain is set, in which case an END-DOMAIN command must be executed before the segment is deleted.

6.2.9. RENAME-SEGMENT

RENAME-SEGMENT <new segment name>, <old segment name>

<new segment name> - the new name to be given to the segment, 1 to 16 alphanumeric characters or hyphen.

<old segment name> - the name of an existing segment in the description file of the current user.

Renames the segment <segment name>. If the segment to be renamed is not in the default directory and/or belongs to another user than the current user, the entire directory and user name must be specified unabbreviated and in parentheses as a prefix to <new segment name>.

6.2.10. LIST-SEGMENT

LIST-SEGMENT (<domain name>), (<segment name>)

<domain name> - the name or abbreviation of names of the domain to be searched. Default is all domains of the current user.

<segment name> - the name or abbreviation of names of segments to be listed. Default is all segments in the selected domains.

All segment names matching <segment name> in the domains with name matching <domain name> are written on the output device, together with some segment information. If a list of another user's segments is wanted the segment name must be prefixed by the user name in parentheses. The domain name may not be prefixed by a user name!

This command will list the domain names as well as the segment names.

6.2.11. WRITE-SEGMENT-STATUS

WRITE-SEGMENT-STATUS [(<segment name>)]...

<segment name> - the name or abbreviation of names about which information is requested. Default is current segment.

Prints all the available information about the segment or segments specified, belonging to the current user or the user specified in parentheses as a prefix to <segment name>. No parameter means that the current segment status is printed. Note that the current segment entry is not fully updated before the command CLOSE-SEGMENT is executed.

6.3. Commands to load NRF code

NRF files conatins code in the format described in chapter 12, produced by language compilers and assemblers.

A file in NRF format may be structured three different ways:

a) Normal, as default output from ASSEMBLER-500, PLANC-500, FORTRAN-500, PASCAL-500, COBOL-500 etc. Modules are located in strict sequential order, and defined labels are indicated by DEF or DDF control numbers.

b) Slow library files, output from the compilers and assembler mentioned above, when compiled/assembled in library mode (refer to the manual for the language in use). Labels defined in the file will appear with LIB control numbers. The term library file refers to a file in this format.

c) Fast library files, as (slow) library files but preceeded by an index table containing the name and byte address within the file of each label defined in the file. This format is obtained by transforming a file in format (b) with the PREPARE-NRF-LIBRARY-FILE command (section 6.10.9.).

The command normally used to load NRF code is the LOAD-SEGMENT. The other commands in this chapter are required only if it is neccessary to force the loading of a library module that would normally not be loaded, or to prevent a module from being loaded.

6.3.1. LOAD-SEGMENT

LOAD-SEGMENT <file name>...

<file name> - the name of a file in NRF format. Default file type is :NRF.

This command loads the NRF code into the current program segment, data segment, and optional common segments. The current segment is the last one specified in an OPEN-SEGMENT or APPEND-SEGMENT command. If no current segment or current domain exists, a scratch domain and a scratch segment will be opened and used by NLL. Default attributes will be used.

If a current domain exists and no segment has been opened with OPEN-SEGMENT, a default segment will be used. See OPEN-SEGMENT.

If the current segment was opened with the command APPEND-SEGMENT rather than LOAD-SEGMENT, and in addition a routine vector was allocated on this segment by the ENTRY-ROUTINES command, the LOAD-SEGMENT command will work like RELOAD-SEGMENT: the new code will be appended to the existing code; previously defined entry points will not cause a "double definition" error, but the routine vector will be updated to point to the new version. Any new entry points will be

entered into the routine vector after the already defined ones.

6.3.2. RELOAD-SEGMENT

RELOAD-SEGMENT <file name> ...

<file name> - the name of a file in NRF format. Default file type is :NRF.

This command will load NRF code to a segment like LOAD-SEGMENT, but modules already loaded to the segment will be replaced with the modules with the same identification in <file name>. The code loaded to the data segment is not replaced, but a warning message is given. This command should be used after an APPEND-SEGMENT command in order to avoid clearing the segment before loading.

This command is useful while debugging large segments and changes are made in a single or a small number of modules. Loading the entire segment is avoided; only the modules that have actually been modified needs to be reloaded.

The new version of the modules are loaded at the current load address of the segment. The space occupied by the old version of the module is not released, and it is the responsibility of the user to load the entire segment to clean this up after the debugging phase is complete.

6.3.3. LIBRARY-SEGMENT-LOAD

LIBRARY-SEGMENT-LOAD <file name>...

<file name> - the name of a file in NRF format. Default file type is :NRF.

This command will load only modules containing referenced symols from a file of structure (a), (b) or (c), as described above. For type (b) and (c) the effect will be exactly as with the LOAD-SEGMENT command.

All symbols defined in <file name> will be considered library symbols, regardless of whether they are actually defined as such in the :NRF file or not. Thus, NRF modules in the file will be loaded only if there are references that can be defined by loading the module. Modules containing no symbol definitions, definitions of already defined symbols or definitions only of symbols not referenced, will not be loaded. This command allows a file to be used as a library even if it has not been compiled/assembled in library mode.

If an NRF module in <file name> contains several symbol definitions, of which one or more are referenced, and others which are already defined, the module is loaded. The first definition of the already defined symbols will then apply, but a warning message will inform that a redefinition was attempted.

6.3.4. OMITTED-SEGMENT-LOAD

OMITTED-SEGMENT-LOAD <file name>, <entry>...

<file name> - the name of a file in NRF format. Default file type is :NRF.

<entry> - the name of a symbol defined in <file name>.

This command will load all modules of an NRF file of structure (b) and (c) containing any referenced symbol(s), except for those modules containing definitions of the specified ones. These modules will not be loaded during this load operation. (Subsequent load commands may cause these modules to be loaded.)

The <entry> does not have to be a library symbol (LIB contol number); it will be omitted from loading regardless of symbol definition type.

This command is commonly used to prevent a standard version of a routine from being loaded, in order to load a different non-standard version from another file. If <entry> is not defined in <file name> or if no symbols are specified, this command will act as LIBRARY-SEGMENT-LOAD.

6.3.5. SELECTED-SEGMENT-LOAD

SELECTED-SEGMENT-LOAD <file name>, <entries>...

<file name> - the name of a file in NRF format. Default file type is :NRF.

<entries> - the name of a symbol defined in <file name>.

The complement of OMITTED-SEGMENT-LOAD: this command will load only those modules containing definitions of the specified <entry>s from files of structure (b) and (c). Other modules will not be loaded. Symbols do not have to be library entries; no modules except those referenced in <entry> will be loaded regardless of symbol definition type.

<entry> does not have to be referenced prior to the use of this command.

This command is used to load a selected routine without necessarily loading all routines in the same file, even if these routines are referenced. If the specified symbol is not found in <file name>, no action is taken.

6.3.6. TOTAL-SEGMENT-LOAD

TOTAL-SEGMENT-LOAD <file name>...

<file name> - the name of a file in NRF format. Default file type is :NRF.

This command will load all modules of an NRF file of structure (a), (b) or (c) except for those modules already loaded. For structure (a) it will act as LOAD-SEGMENT except for modules already loaded, which will be skipped with no warning message given.

6.4. COMMON segments

Fortran COMMON areas may either be placed in the same segment as other data, or they may be put in their own segment(s). Arguments for using separate COMMON segments are similar to other segmenting: a different protection, memory allocation or sharing is desired for the COMMON segments than for other data areas.

Own data segments for the COMMON areas may be defined by the commands below. These segments will not have corresponding program segments (unless the segments have been previously opened with OPEN-SEGMENT, in which case the existing program segment is ignored). Common segments are special only in the sense that they facilitate selective loading and linking of common blocks.

These commands apply mainly to Fortran programs. Common blocks defined by a Fortran program may be referenced by some languages. A label is defined on a common segment if the NRF language code of the loaded module is FORTRAN and data mode is set (DMO control number, see chapter 12), or if the label has been explicitly defined on a common segment by the DEFINE-ENTRY or the DEFINE-COMMON command.

6.4.1. COMMON-SEGMENT-OPEN

COMMON-SEGMENT-OPEN (<segment name>), (<attributes>)

<segment name> - the name of a segment to be used for common areas in subsequent loading. Default name is COMMON-SEGMENT.

<attributes> - a string of the letters ROPLMEWC. Default is WCL.

Prepare a common-data-segment as an additional current data-segment.

The <attributes> have the same meaning as for OPEN-SEGMENT, section 6.2.1., where the option letters are explained.

The default segment when loading a common block is the last one specified in a COMMON-SEGMENT-OPEN command. However, common areas can be placed on any segment by defining the common label on those segments prior to loading the file containing the common block. See the DEFINE-COMMON command (section 6.7.4.).

Each program/data segment pair may have up to four common segments. Any data segment may be opened as common segment to any program segment. COMMON-SEGMENT-OPEN will clear the segment - if code is to be added to an already loaded common segment or it will be linked to, COMMON-SEGMENT-APPEND should be used.

6.4.2. COMMON-SEGMENT-CLOSE

COMMON-SEGMENT-CLOSE

Loading to all currently defined common segments is terminated. After this command, there is no current common segment. Loading of common areas may continue, but they will be located in the current data segment unless they were already defined on one of the common segments.

This command is not required before opening a new program segment, common segment or EXIT.

6.4.3. COMMON-SEGMENT-APPEND

COMMON-SEGMENT-APPEND (<segment name>)

<segment name> - name of an existing segment, to which more common blocks will be added. Default is COMMON-SEGMENT.

Common blocks defined in NRF files to be loaded will be located after the data already loaded to <segment name>. <segment name> must exist when the command is executed. The access of the segment is not changed. Only the data segment is affected.

This command may also be used to link a previously loaded common segment to another program segment, as an alternative to LINK-SEGMENT. With respect to the data segment these two commands are identical, but LINK-SEGMENT will also link to a program segment, if it exists. This could cause unintened linking of accidentally synonymous entry points. The COMMON-SEGMENT-APPEND will if used on a normal segment consisting of both a program and a data segment, link to the data segment only.

6.4.4. COMMON-SEGMENT-NUMBER

COMMON-SEGMENT-NUMBER <segment number>

<segment number> - a number in the range 0:37B to be the logical segment number of the current common segment. Default is 33B.

Same as SET-SEGMENT-NUMBER except that COMMON-SEGMENT-NUMBER applies to the last common segment specified in a COMMON-SEGMENT-OPEN or COMMON-SEGMENT-APPEND command. The user will normally not be concerned with the segment number used.

6.5. Auto-link segments

An auto-link segment is linked if there are still undefined references after the specified files are loaded when the CLOSE-SEGMENT command is executed. If undefined references still exist after the auto-link segments defined by the current user have been linked, the auto-link segments defined by SYSTEM are linked. Auto-link segments are linked before the auto-load files are loaded.

Auto-link segments are language sensitive, and will be linked only if one or more module of the language(s) associated with it are already loaded.

6.5.1. SET-AUTO-LINK-SEGMENT

SET-AUTO-LINK-SEGMENT <segment name>, <language> ...

<segment name> - the name of a segment to be automatically linked at CLOSE-SEGMENT if undefined references remain.

<language> - a combination of FORTRAN, ASSEMBLER, PLANC, COBOL or PASCAL.

Defines the segment with name <segment name> as an auto-link segment. The auto-link segment specified will be valid until the command DELETE-AUTO-LINK-SEGMENT is used. The auto-link segment applies only to the user who has defined it. Auto-link segments defined by user SYSTEM apply to all users however, after the auto-link segments of that user have been linked.

The <language> name may be abbreviated as long as it is unambiguous.

The buffer containing the auto-link segment names can hold a maximum of six entries. This does not include auto-link segments defined by user SYSTEM. If the segment name is abbreviated in this command, it is not expanded before the name is saved. Thus, to avoid ambiguity with segments defined at a later time, the name should not be abbreviated.

It is not checked whether the segment exists at the time when it is defined as an auto-link segment by this command. If the segment is not present when the automatic linking is performed, it is ignored; no error message is issued.

6.5.2. DELETE-AUTO-LINK-SEGMENT

DELETE-AUTO-LINK-SEGMENT

All the user-defined auto-link segments are deleted permanently. The auto-link segments defined by user SYSTEM will also be removed, but only until NLL is reentered.

After a DELETE-AUTO-LINK-SEGMENT new permanent auto-link segments may be defined with the SET-AUTO-LINK-SEGMENT command.

6.5.3. LIST-AUTO-LINK-SEGMENTS

LIST-AUTO-LINK-SEGMENTS

Writes on the output device all the auto-link segments in the sequence they will be linked. Both the user's own and SYSTEM's auto-link segments will be listed.

6.6. Auto-load files

An auto-load file is an NRF file which is automatically loaded when the command CLOSE-SEGMENT is executed and any undefined references exist. The auto-load files are loaded after the defined auto-link segments are linked.

The auto-load files will be loaded in the sequence that they are specified. If after loading all the user-defined auto-load files there are still undefined references, the auto-load files of user SYSTEM will be loaded.

The auto-load files are language dependent. Only those files specified as auto-load files for the language used, will be loaded. If a system consists of modules of different languages, auto-load files will be loaded for all languages used, in the order they have been specified with the SET-AUTO-LOAD-FILE command.

6.6.1. SET-AUTO-LOAD-FILE

SET-AUTO-LOAD-FILE <file name>, <language>

<file name> - the name of a file to be automatically loaded if undefined references remain at CLOSE-SEGMENT.

<language> - combination of FORTRAN, ASSEMBLER, PLANC, COBOL or PASCAL

An auto-load file may be specified with more than on language parameter, indicating that the file should be loaded if routines written in either language have been loaded.

File names defined by the command SET-AUTO-LOAD-FILE, will be stored permanently for the current user, and will only be removed by use of the command DELETE-AUTO-LOAD-FILE.

The buffer containing the auto-load file names can hold a maximum of six entries. This does not include auto-load files defined by user SYSTEM.

6.6.2. DELETE-AUTO-LOAD-FILE

DELETE-AUTO-LOAD-FILE

All the auto-load files defined by the current user are deleted permanently. The auto-load files defined by user SYSTEM will also be removed, but only until NLL is reentered.

After a DELETE-AUTO-LOAD-FILE new permanent auto-load files may be defined with the SET-AUTO-LOAD-FILE command.

6.6.3. LIST-AUTO-LOAD-FILE

LIST-AUTO-LOAD-FILE

Lists all the auto-load files in the sequence that they will be loaded. Both the user's own and SYSTEM's auto-load files will be listed.

6.7. Label and reference handling

References may be of four kinds:

- a program reference in the program segment. This occurs for example when a routine is called and the routine address is a part of the instruction operand.
- a data reference in the program segment. Any instruction operating on a variable data item will make this kind of reference.
- a program reference in the data segment. If a jump address or subroutine address is found in the data segment (referenced through a general operand specifier, in assembler terms), this occurs.
- a data reference in the data segment. A data value contains the address of another data value, a displacement etc.

If the references can be defined at compile (or assembly) time, the user will not be aware of them. However, if the referenced item is not located within the NRF module of the reference, a symbolic name is associated with it. A value to be given to the symbol may be defined either by another NRF module, or by the user from the terminal.

The user may also make references of all the four kinds mentioned above. This is mainly used for forcing specific library modules to be loaded.

Wherever a numeric parameter is called for in the commands below, this parameter may be a decimal or octal number, or it may be a previously defined symbol (either defined by a command or by loading an NRF module). The symbols #PCLC and #DCLC are available to indicate the current program location counter (load address) and current data location counter, respectively.

6.7.1. PROGRAM-REFERENCE

PROGRAM-REFERENCE <symbol>, (<address>), (<space>)

<symbol> - the name of a defined or undefined symbol.

<address> - the address where the reference is made in the program segment. Default is 0. Symbolic as well as numerical addresses are legal.

<space> - P or D, indicating a symbol defined in the program or the data segment, respectively. Default is P.

If <symbol> is not present in the loader table, it will be entered as an undefined program label reference at <address>. If <symbol> is present but as an undefined reference, <symbol> will be referenced once more in the <address> specified.

If <symbol> is an already defined label, its value will immediately be put into the <address> specified. Otherwise, as soon as it is defined, it will be put into all addresses from where it has been referenced.

Default <address> causes no modification of any memory location if the symbol was undefined when the command was given, and is later defined.

6.7.2. DATA-REFERENCE

DATA-REFERENCE <symbol>, (<address>), (<space>)

<symbol> - the name of a defined or undefined symbol.

<address> - the address where the reference is made in the data segment. Default is 0. Symbolic as well as numerical addresses are legal.

<space> - P or D, indicating a symbol defined in the program or the data segment, respectively. Default is D.

If <symbol> is not present in the loader table, it will be entered as an undefined data label reference at <address>. If <symbol> is present but as an undefined reference, <symbol> will be referenced once more in the <address> specified.

If <symbol> is an already defined label, its value will immediately be put into the <address> specified. Otherwise, as soon as it is defined, it will be put into all addresses from where it has been referenced.

Default <address> causes no modification of any memory location if the symbol was undefined when the command was given and is later defined.

6.7.3. DEFINE-ENTRY

DEFINE-ENTRY <label>, (<value>), (<space>)

<label> - the name of a not yet defined symbol.

<value> - the value assigned to the label. Default is 0.

<space> - P or D, representing a symbol defined in program or data memory, respectively. Default is P.

<label> will be entered into the loader table as a defined symbol. The value will be equal to <value>. If the default value is used, no modification of the current load address is done.

If the entry is already defined, an error message is issued.

6.7.4. DEFINE-COMMON

DEFINE-COMMON <symbol name>, (<size>), (<value>)

<symbol name> - the name of an undefined symbol in a common block.

<size> - the size of the common area to be defined. Default is undefined size.

<value> - the value of the common symbol. Default is undefined value, but on the current data segment or last common segment specified.

The common label will be entered into the loader table as a symbol defined in a common data block. If <value> is zero the common label will be allocated from the current data load address. If <value> plus <size> is larger than the current data load address on the segment where symbol is being defined, the current data load address is adjusted upwards to this value.

The common block is placed on the current data segment, or if common segments are open, on the last common segment specified in a COMMON-SEGMENT-OPEN or COMMON-SEGMENT-APPEND command.

Default <size> will cause the size to be determined the first time it is defined during loading of code. If the default <size> is used, <value> may not be specified.

Default <value> will cause the actual allocation of the common block to be done at the current load address when a definition of the symbol is loaded from an NRF file. If <size> is specified, the common block will have this size regardless of the defined size of the first occurence of the common block. This can be used to override the limitation that the first definition of a common block must be the largest one.

6.7.5. LIST-ENTRIES-DEFINED

LIST-ENTRIES-DEFINED (<sort criterium>)

<sort criterium> - NUMERICAL or ALPHABETICAL. Default is NUMERICAL.

All defined labels together with their values and space (P or D) and the current load address will be written to the output device. The <sort criterium> determines whether the list is sorted according to symbol name or to their numerical value.

If the command SYSTEM-ENTRIES-ON is given before the LIST-ENTRIES-DEFINED command, all entries are listed. Otherwise only user defined entries are listed.

6.7.6. LIST-ENTRIES-UNDEFINED

LIST-ENTRIES-UNDEFINED (<sort criterium>)

<sort criterium> - NUMERICAL or ALPHABETICAL. Default is NUMERICAL

All undefined entries (references) in the loader table, together with their referenced address and space (P or D), will be written on the output device. If a symbol is referenced several places, it is written once for each reference, each with the address of the reference.

The <sort criterium> determines whether the list is sorted according to symbol name or to the numerical address of the references.

6.7.7. LIST-MAP

LIST-MAP

Writes the load map on the output device. This includes the addresses of all undefined references followed by the addresses or values of defined labels, both sorted in numerical order.

6.7.8. SYSTEM-ENTRIES-ON

SYSTEM-ENTRIES-ON

The command LIST-ENTRIES-DEFINED will not print the system defined labels. System defined labels will have their first character equal to # or [. If a list including system defined entries is desired, the command SYSTEM-ENTRIES-ON must be issued before the command LIST-ENTRIES-DEFINED. When system entries are printed, the language is included. Program entries containing an entry point specifying a fixed data area (INIT, ENTM, ENTF, ENTFN and ENTT instructions) rather than stack allocation will be followed by a slash and the address of the local data area.

LIST-ENTRIES-UNDEFINED will print the referenced system entries without using the command SYSTEM-ENTRIES-ON.

The SYSTEM-ENTRIES-ON command applies to the next LIST-ENTRIES-DEFINED only, and the command must be given every time a list of system defined labels is required.

6.7.9. GLOBAL-ENTRIES

GLOBAL-ENTRIES <label> ...

<label> - name of symbol to be retained on :LINK file.

The entries in the loader table, except those referred to in this command, are removed from the loader table before the table is written on the :LINK file. This is useful if only a subset of the routines on the segment should be made global. This command must be issued before the segment is closed.

If the GLOBAL-ENTRIES command has not been executed, all entries in the loader table will be retained on the :LINK file. In either case, all symbols will be truncated to 20 characters.

6.7.10. KILL-ENTRIES

KILL-ENTRIES <symbol> ...

<symbol> - the name of an entry to be removed from the loader table.

If present, the symbol(s) specified will be removed from the loader table. The entry may be defined or undefined. This command is used to resolve name conflicts, avoid loader table overflow and to selectively prohibit symbols from being saved on the :LINK file.

6.8. Areas shared with ND-100 processes

The following commands are used to define sharing of segments with ND-100 processes. The programmer must have experiencece with ND-100 real time programming in order to utilize these commands, as he is responsible for the synchronizing with ND-100 processes and the protection of common areas.

Readers who do not need to communicate with ND-100 processes may skip this section.

6.8.1. MATCH-RTCOMMON

MATCH-RTCOMMON

All RT-COMMON labels defined by the RT-LOADER (see the RT loader manual ND-60.051) will be defined as common labels in the loader table. The addresses are transformed to ND-500 addresses. The RTCOMMON area will start at the next free page boundary in the current common segment if any is defined; otherwise it will be located in the data segment.

The MATCH-RTCOMMON command should be used before the program modules referring to the RT-COMMON area are loaded.

The names of defined labels will be reformatted from the BRF format (6 bits per character) to NRF format (ASCII bytes), addresses will be converted to byte addresses and an offset representing the relative ND-500 address is added.

The MATCH-RTCOMMON command applies to ND-100/ND-500 communication. A domain using RTCOMMON should not be copied to other machines with the COPY-DOMAIN command. The size of RTCOMMON must not be changed after the domain is loaded; that will require reloading. The RTCOMMON area must be contiguous.

6.8.2. MATCH-COMMON-RT-SEGMENT

MATCH-COMMON-RT-SEGMENT <segment number>

<segment number> - the number of an ND-100 segment.

All segment common labels defined by the RT-LOADER on the segment specified will be defined as common labels in the loader table. The MATCH-COMMON-RT-SEGMENT command should be used before the program modules referring to the segment common are loaded.

The MATCH-COMMON-RT-SEGMENT command applies to ND-100/ND-500 communication. A domain making references to ND-100 segments should not be copied to another machine with the COPY-DOMAIN command. A maximum of five ND-100 segments, RTCOMMON inclusive, is available.

6.8.3. LINK-RT-PROGRAM

LINK-RT-PROGRAM

Defines the RT programs defined by the RT-LOADER. The command should be used after the program modules referring to the RT program names are loaded. Only RT program names which are referenced in the loader table are defined by the command.

The LINK-RT-PROGRAM command applies to ND-100/ND-500 communication. A domain making references to RT programs should not be copied to another machine with the COPY-DOMAIN command.

6.9. Miscellaneous commands

6.9.1. LOW-ADDRESS

LOW-ADDRESS (<address>), (<space>)

<address> - address in the range 0:777777777B. Default value is 4.

<space> - P, D or C or combinations of these, indicating program, data or common address, respectively. Default is PD.

The lower load address for subsequent loading to the current segment is set. If C is specified, the load address is set on the last common-segment specified in a COMMON-SEGMENT-OPEN or COMMON-SEGMENT-APPEND command.

If the load address is set to a higher value than the current load address, a hole may remain in the file if the affected pages have never been assigned to the segment file. If a NO SUCH PAGE condition occurs at execution time, the Monitor will zero fill the page in memory. If the page has been used at an earlier time, the old contents will be used, and may for practical purposes be considered unpredictable.

6.9.2. HIGH-ADDRESS

HIGH-ADDRESS (<address>), (<space>)

<address> - address in the range 0:777777777B. Default value is 777777777B.

<space> - P, D or C or combinations of these, indicating program, data or common address, respectively. Default is PD.

This command sets the highest address available on a segment. If any loading above the specified upper high address is attempted, a warning message is issued and the loading process interrupted.

6.9.3. ENTRY-ROUTINES

ENTRY-ROUTINES (<number of entries>)

<number of entries> - the maximum number of routines to be loaded on the current segment. Default is 200B.

A library segment will at the start of the segment have a "routine vector": when a routine is called from another segment, control goes via this vector - the first routine on the segment represents the first routine, the second element the second routine and so on. If the routines are modified and change their relative position, no relinking of other segments is necessary as long as the routine vector is updated and the routine number stays the same.

This command will allocate space for a routine vector of the specified size, and must be given before any loading to the segment. <number of entries> should be at least the maximun number of routines that will be loaded to the segment.

All manipulation of the routine vector is done by NLL, and the user need not be concerned about how the link from other segments is set up.

The entries in the routine vector are initialized to zero, but will be filled in by NLL as code is loaded to the program segment.

6.9.4. SET-IO-BUFFERS

SET-IO-BUFFERS (<number>)

<number> - the number of 2k byte buffers to be used by the Fortran library for sequential I/O for file buffering. Default is 16.

This command should be used only when a Fortran library segment is created, or for any reason the Fortran library is loaded to the main segment. From ordinary programs, the Fortran library will be linked to the main segment, and the I/O buffers will already have been allocated in the data segment of the library.

The command specifies a number of input/output buffers for more efficient handling of sequential files in Fortran. Two system labels will be defined, at the lower and upper limits of the buffer area, and the current load address will be increased by the size of the buffer area. The total size of all buffers will be <number> * 2048 bytes. The user should choose an appropriate number of buffers; the normal number is one for each simultaneously opened sequential file. If a Fortran library segment is being created, 16 buffers should be specified.

The labels defined by this command will be used by the Fortran I/O system to determine the location and the size of the buffer. No other use of the area is made.

6.9.5. LIST-OCTAL

LIST-OCTAL <low address>, <high address>, <space>

<low address> - the address from which listing should start. Default is 0.

<high address> - the address up to which listing should continue. Default is <low address>+200B.

<space> - P or D, indicating program or data memory, respectively. Default is D.

The contents of the locations between <low address> and <high address> will be written on the output device in octal format, together with the byte address.

6.9.6. LIST-SYMBOLIC

LIST-SYMBOLIC (<low address>), (<high address>), (<space>)

<low address> - the address from which listing should start. Default is 0.

<high address> - the address up to which listing should continue. Default is <low address>+200B.

<space> - P or D, indicating program or data segment, respectively. Default is P.

The contents of the locations between <low address> and <high address> will be written on the output device in a disassembled format, together with the byte address.

6.9.7. LIST-MODE

LIST-MODE

Everything that is loaded is written on the output device in octal format as it is being read from the NRF file. LIST-MODE will be terminated by DISASSEMBLE-MODE.

6.9.8. DISASSEMBLE-MODE

DISASSEMBLE-MODE

Everything that is loaded is written on the output device in disassembled format as it is being loaded from the NRF file. Disassemble-mode will be terminated by LIST-MODE.

6.9.9. CHECK-SYNTAX-MODE

CHECK-SYNTAX-MODE

If this command is executed, the following commands up to EXIT are checked for syntactic correctness in the command processor only. They will not be executed.

This is helpful for checking a batch or mode job before it is started.

6.9.10. RESET

RESET

Removes all symbols from the loader table and resets load addresses to the initial low addresses (which is 4 for both program, data and common segments).

Observe that NRF code is loaded directly to the segment files. Thus, RESET cannot be used to discard loaded code and revert the segment files to the state they were before loading was started.

6.9.11. RENAME-DEFAULT-DIRECTORY-AND-USER

RENAME-DEFAULT-DIRECTORY-AND-USER <(new directory:new user)>

<(new directory:new user)> - the new unabbreviated directory and user name, including parentheses and colon.

If the default directory and/or the user must be renamed with the Sintran III commands @RENAME-DIRECTORY and @RENAME-USER, this command must be used in order to make the domain and segment descriptions in the description file consistent with the new Sintran III names. An exact match with the user and directory name is required, including the parentheses and the colon.

6.9.12. SUPPRESS-DEBUG-INFORMATION

SUPPRESS-DEBUG-INFORMATION (<ON/OFF>)

<ON/OFF> - ON if debug info should be suppressed, OFF if it should be retained. Default is ON.

If the parameter is specified as ON, all debug information in subsequently loaded files will be discarded, rather than saved on the :LINK file. If the command is given with the parameter OFF, copying of the debug info to the :LINK file will be resumed (the initial state of NLL).

The primary purpose of this command is to reduce the size of the :LINK file. It may also be used if the Symbolic Debugger will be used, when parts of the system are already completely debugged so that no further debugging of these parts will be done. Suppressing the debug info will then prevent breakpoints, line or routine tracing in the selected parts.

6.10. NRF editor

The NRF editor commands manipulate modules of an NRF file, that is, the information delimited by BEG and END control numbers. Control numbers and mnemonics are described in chapter 12. A module is identified by any of the DEF, DDF or LIB symbols defined within it. Modules are treated as indivisible units; specifying one (of several) symbols in a module denotes the entire module.

These commands are mainly used by system supervisors and system programmers who have to maintain libraries of NRF code. A familiarity with the NRF format is desirable in order to use these commands.

6.10.1. NEW-NRF-MODULES

NEW-NRF-MODULES <new modules file>, <NRF file>

<new modules file>- the name of an NRF file containing the new modules to replace the old ones. Default file type is :NRF.

<NRF file> - the NRF file to be updated. Default file type is :NRF.

The NRF modules in <NRF file> with the same identification as the NRF modules in the <new modules file> will be replaced by the NRF modules in the <new modules file>. The various NRF modules in <NRF file> will have their same relative position within the file after the NEW-NRF-MODULES command as before. NRF modules in the <new modules file> not found in <NRF file> will be skipped and a warning message given. NRF modules without symbolic names cannot be replaced.

6.10.2. FETCH-NRF-MODULES

FETCH-NRF-MODULES <source file>, <destination file>
(<first module>), (<last module>)

<source file> - the name of an NRF file containing the modules to be appended.

<destination file>- the name of an NRF file to be appended to.

<first module> - the first module from the source file to be appended to the destination file. Default is the first module in the source file.

<last module> - the last module from the source file to be appended to the destination file. Default is the last module in the source file.

The NRF modules in the <source file> starting with the <first module>, including every module up to the <last module> will be appended to <destination file> after the last NRF module in the <destination file>.

6.10.3. APPEND-NRF-MODULE

APPEND-NRF-MODULE <source file>, <destination file>
(<after module>)

<source file> - the name of an NRF file containing the modules to be appended.

<destination file>- the name of an NRF file to be appended to.

<after module> - the module in the destination file after which the new modules will be appended. Default is after the last module.

All NRF modules in the <source file> will be appended to <destination file> after the specified NRF module in the <destination file>.

6.10.4. DELETE-NRF-MODULES

DELETE-NRF-MODULES <file name>, (<first module>), (<last module>)

<file name> - the name of an NRF file. Default file type is :NRF.

<first module> - a symbol defined in the first module to be deleted. Default is the first module in the file.

<last module> - a symbol defined in the last module to be deleted. Default is the last module in the file.

The specified NRF modules will be deleted from <file name>. <first module> is the first module which will be deleted, and then all NRF modules following and including <last module> will be deleted.

6.10.5. LIST-NRF-ENTRIES

LIST-NRF-ENTRIES <file name>

<file name> - the name of an NRF file. Default file type is :NRF.

This command will list all DEF, DDF and LIB symbols in <file name> on the output device, together with their byte address in the file.

6.10.6. LIST-NRF-CODE

LIST-NRF-CODE <file name>, (<first module>), (<last module>)

<file name> - the name of an NRF file. Default file type is :NRF.

<first module> - the name of a symbol defined in the file, identifying the first module to be listed. Default is the first module in the file.

<last module> - the name of a symbol defined in the file, identifying the last module to be listed. Default is the last module in the file.

All NRF information in the specified modules in the <file name> will be listed in the following format: location counter, NRF control number, name of NRF control number. In addition symbolic names will be written in ASCII format. Binary information will be written in both disassembled ND-500 format (if program code) and octal format.

6.10.7. WRITE-NRF-EOF-AFTER-MODULE

WRITE-NRF-EOF-AFTER-MODULE <file name>, (<module>)

<file name> - the name of an NRF file. Default file type is :NRF.

<module> - the name of a symbol defined in the NRF file, identifying the last module still valid. Default is to insert the EOF control number in front of the first module in the file.

Write the NRF control number 260 (EOF) after the specified NRF module in <file name>. If the default value for the parameter <module> is used, the EOF byte is written as the first byte on the <file name>.

6.10.8. INSERT-NRF-MESSAGE

INSERT-NRF-MESSAGE <file name>, (<module>), <message>

<file name> - the name of an NRF file. Default file type is :NRF.

<module> - the name of a symbol defined in the file, identifying a module in front of which the message will be located. Default is the first module in the file.

<message> - any character string excluding space up to the first carriage return.

This command inserts the message in the NRF <file name> before <module>. If the file is prepared with the PREPARE-NRF-LIBRARY-FILE command, the default <module> is in the front of the address table in <file name>.

The specified message will be written on the output device when the file is loaded. If the file is a library file headed by an address table, a message in front of the address table will be written; all other messages (defined by this command) are located outside NRF modules, and will not be written.

In the <message>, a dollar sign will be converted to Carriage Return and Line Feed.

6.10.9. PREPARE-NRF-LIBRARY-FILE

PREPARE-NRF-LIBRARY-FILE <file name>

<file name> - the name of an NRF file. Default file type is :NRF.

This command will set up an address table in front of <file name> containing all LIB symbols together with their byte addresses in <file name>. This will convert the file from structure "b", slow library file, to "c", fast library file, as described in the chapter on 'Commands to load NRF code'. <file name> must be the output of a compilation with library mode set. If a library is prepared by this command, conditional loading is done much more efficiently.

The address table is invalidated by all commands modifying the contents of the NRF file, and the table must be rebuilt if a sequential search of the file is to be avoided.

7. COMMANDS AVAILABLE IN THE NLL AND THE MONITOR

These commands may be issued either during the loading of the program, or at run time, before the program is executed. Some of the commands, those defining trap handling, will define defaults if used in NLL. These defaults may be overridden in the Monitor. If the command is given in the Monitor, it applies to the current job only, and will not permanently influence the properties of the segment or domain.

Some commands behave slightly differently in NLL and the Monitor. Such differences are explained under each command.

7.1. Utility commands

7.1.1. HELP

HELP (<command name>)

<command name> - any command abbreviation, ambiguous or non-ambiguous. Default is all commands available.

All commands matching <command name> are written with their parameters on the output device. Parameters enclosed in brackets [] are optional parameters that will not be prompted for if not supplied.

7.1.2. OUTPUT-FILE

OUTPUT-FILE (<file name>)

<file name> - the name of the file where output is desired. Default is the communication device.

This command is used to define an output device different from the current one (initially the communication device). Most output will go to <file name>, but commands, parameter prompt and error messages will continue to appear on the communication device. The <file name> is used as an output device until EXIT or a new OUTPUT-FILE command is given.

7.1.3. @ (Sintran-III command)

@command

If a line starts with the @ character, the remainder of the line is asumed to be a Sintran III command and executed through the COMND monitor call.

Be aware that control will not return to NLL or the monitor after execution of the command if another subsystem or user program was called by the command. Also, if an error occurs during the execution of the command, control will not return to the calling program.

In NLL, if a loading is in progress when the Sintran III command is executed and control does not return, the description file may have been left in an inconsistent state. It may be necessary to use the RELEASE-SEGMENT command to gain access to the segment that was current, and the contents will be unpredictable.

The Monitor will check the command issued before it is submitted to Sintran III, and will allow only a subset of Sintran III commands.

7.1.4. CC

CC any text

Comment; whatever follows on the same line as the CC command is ignored and treated as a comment. This command is primarily useful for making comments in a batch or mode job.

7.1.5. ABORT-BATCH-ON-ERROR

ABORT-BATCH-ON-ERROR <ON/OFF>

<ON/OFF> - ON if batch jobs should terminate if an error occurs, OFF if only the current command should be terminated.

If an error occurs in a batch or mode job and this command has been executed with the parameter OFF, only the current command is aborted and the next command in the batch input file is executed. If the command has not been executed or executed with the parameter ON, the entire job is terminated. The error message will be written on the batch output file, and in NLL the commands CLOSE-SEGMENT and END-DOMAIN will be executed (if required).

This command may be specified several times, switching the batch termination on and off before and after critical sequences.

7.1.6. EXIT

EXIT

Returns to the Sintran III command processor.

In NLL, if not explicitly done the CLOSE-SEGMENT and END-DOMAIN commands are executed if a segment is opened or a domain set.

In the Monitor this command releases the allocated ND-500 resources. If the buffer used by the histogram and logging commands was reserved, it will be released.

If NLL was started after entering the Monitor, return will be to the Monitor. Otherwise, return will be to Sintran command mode.

In the Monitor, this command is also used to return from the LOOK-AT commands.

7.2. Trap handling

The ND-500 trap mechanisms may be used to detect and handle exceptional conditions occuring at execution time. The user may optionally specify a routine to take care of the trap, or it may be handled by a standard library routine.

Some of the traps are by default system enabled, others are locally enabled and handled by the library routines. The default settings are discussed in chapter 4., Standard Exception Handler Routines.

The names of the trap conditions and the label of the standard handlers are:

trap name	label
OVERFLOW	#OVERFLW
INVALID-OPERATION	#INVALOP
DIVIDE-BY-ZERO	#INVALDI
FLOATING-UNDERFLOW	#FLTUFLW
FLOATING-OVERFLOW	#FLTOFLW
BCD-OVERFLOW	#BCDOFLW
ILLEGAL-OPERAND-VALUE	#ILLOPER
SINGLE-INSTRUCTION-TRAP	#SINGINS
BRANCH-TRAP	#BRANCTR
CALL-TRAP	#CALLTRA
BREAK-POINT-INSTRUCTION-TRAP	#BRKPNTR
ADDRESS-TRAP-FETCH	#ADDRFTC
ADDRESS-TRAP-READ	#ADDREAD
ADDRESS-TRAP-WRITE	#ADDWRTE
ADDRESS-ZERO-ACCESS	#ADDZERO
DESCRIPTOR-RANGE	#DESCRIR
ILLEGAL-INDEX	#ILLINDX
STACK-OVERFLOW	#STKOFLW
STACK-UNDERFLOW	#STKUFLW
PROGRAMMED-TRAP	#PROGTRA
DISABLE-PROCESS-SWITCH-TIMEOUT	#DISPSWT
DISABLE-PROCESS-SWITCH-ERROR	#DISPSWE
INDEX-SCALING-ERROR	#INXSCAL
ILLEGAL-INSTRUCTION-CODE	#ILINCOD
ILLEGAL-OPERAND-SPECIFIER	#ILOPSPE
INSTRUCTION-SEQUENCE-ERROR	#INSEQUE
PROTECT-VIOLATION	#PVIOLAT

7.2.1. LOCAL-TRAP-ENABLE

LOCAL-TRAP-ENABLE <label> <trap condition> ...

<label> - the name of a user written or library exception handler routine. Default is the standard handler in the library for the specified <trap condition>.

<trap condition> - one of the trap names above or an unambiguous abbreviation.

The bit in the OTE register corresponding to the specified <trap condition> will be set, thereby causing the trap condition to be reacted upon if it occurs. The <trap condition> parameter must be one or more of the names in the table above. Abbreviations are legal as long as they are non-ambiguous.

The <label> is inserted in the table of exception handler routines. This table may contain different labels for each trap condition, or one routine may be used by several traps. The default trap handler has a label as specified in the table above. NLL will cause the standard handlers used to be loaded from the standard library. The Monitor allows the <label> to be specified either as an absolute address or as a defined program label. This label must be present in the :LINK file of the segment. If the <label> is omitted and an exception handler routine is defined, it is not modified. If no handler was defined, the standard library handler is used. This requires that the standard routine was previously loaded.

The trap handler allocated by NLL is an array located at the most recently modified segment (OPEN-SEGMENT or APPEND-SEGMENT) in the domain.

7.2.2. LOCAL-TRAP-DISABLE

LOCAL-TRAP-DISABLE <trap condition> ...

<trap condition> - one of the trap names above or an unambiguous abbreviation or ALL.

The bit in the OTE register corresponding to the specified <trap condition> is cleared, thereby disabling trap handling for that trap condition. If ALL is specified, all traps will be locally disabled. This is mainly used in order to override the default setting before a new selection of traps is enabled.

The routine defined in the exception handler table is not cleared. If the OTE bit is later set (by program or by using the LOCAL-TRAP-ENABLE command in the monitor before execution is started), the routine defined in the LOCAL-TRAP-ENABLE command acts as the default exception handler.

7.2.3. SYSTEM-TRAP-ENABLE

SYSTEM-TRAP-ENABLE <trap condition> ...

<trap condition> - one of the trap names above or an unambiguous abbreviation.

The <trap condition> specified will be handled by the Monitor residing in the ND-100 when the conditon occurs. It will be given a standard treatment, which varies with the kind of trap.

If a local trap handler is defined and the local trap enabled, it will be used rather than the system trap handler. System trap handling is used only for those trap conditions that are locally disabled or have no local trap handling defined.

7.2.4. SYSTEM-TRAP-DISABLE

SYSTEM-TRAP-DISABLE <trap condition> ...

<trap condition> - one of the trap names above or an unambiguous abbreviation.

The <trap conditions> specified will not be reacted upon by the system when the conditon occurs.

A number of trap conditions may not be system disabled. If a modification of these traps are attempted, an error message is issued and the command ignored.

7.3. VALUE-ENTRIES

VALUE-ENTRIES <label>...

<label> - the name of a defined symbol.

Prints the values of the labels specified on the output device. The value is printed in octal format. The label will also be identified as a program or as a data segment label.

8. MONITOR COMMANDS

Most commands in this chapter need not be known to the ordinary ND-500 user. The one command used for executing an ND-500 program, the RECOVER-DOMAIN command, is implicit if match with no other commands is found. Thus, in order to start execution of a domain, it is sufficient to give the domain name as a command.

A domain name may also be specified on the same line as the command to start the Monitor. If a domain is executed this way, control will return to Sintran III immediately after execution is complete. Otherwise, the EXIT command must be used in the Monitor.

Various error messages may be returned from the ND-500 Monitor during command and program execution. These error messages are listed in chapter 14, with short explanations and references to where they may occur.

8.1. Commands for running an ND-500 program

8.1.1. PLACE-DOMAIN

PLACE-DOMAIN <domain name>

<domain name> - the name of a domain in the description file of the current user or the user specified in parentheses as a prefix to <domain name>.

An executable ND-500 domain is made ready for execution. The specified <domain name> is searched for on the description file of the current user. If no match is found, the description file of user SYSTEM is scanned. A user name prefixing <domain name> is valid. The syntax is equal to the file system syntax.

If the specified domain is found, some initialization is performed. The start address is moved into the program counter register. The child trap enable register of ND-100, the own trap enable register of the domain and the trap handler address register are initialized. Each logical segment is mapped on a physical segment.

The program segment will normally map directly onto the :PSEG file. Several users may be using the same physical segment, although the segments may be logically different. It is assumed that the program segments are read only. This means that breakpoints cannot be used, and patching is not possible. The DEBUG-PLACE command will permit modifications.

The data segment is initially mapped on the :DSEG file. Upon page fault the required page is read from the file. When modifications are made, the affected pages are not written back to the :DSEG file but to a scratch area on a swap file. This copy is used for later references. Each concurrent user of the data segment has his own copy of modified

pages on the swap file, and is thus independent of other users. The physical segment corresponding to the data segment is therefore a mixture of unmodified pages in the :DSEG file and modified pages in the swap file.

8.1.2. RUN

RUN

The current domain is started in its start address.

The command must have been preceeded with a PLACE-DOMAIN or DEBUG-PLACE command in order to bring the domain into memory. Return will be to the Monitor after execution has completed.

8.1.3. RECOVER-DOMAIN

RECOVER-DOMAIN <domain name>

<domain name> - the name of a domain in the description file of the current user, user SYSTEM or if user name specified, of that user.

The PLACE-DOMAIN and RUN commands are performed as one by using the command RECOVER-DOMAIN. The words RECOVER-DOMAIN can be left out. The domain name itself becomes a pseudo command. The procedure for looking up the command or domain is then as follows:

1) A search is made in the list of basic commands. If a match is found, the corresponding command is executed.

2) If no command is found, the list of standard domains are searched (see section 8.2.). If there is any such standard domain, it is started.

3) If the search among the standard domains was unsuccessful, a search is made in the domains of the current user. If a domain with the specified name is found, it is started as with a RECOVER-DOMAIN command.

4) If no domain with the specified name is found, the domains of user SYSTEM are searched. If a domain with a matching name is found, the domain is started.

5) If no domain is found, the specified string is assumed to be a macro name, and a temporary macro is searched for. If any matching macro is found, it is processed. (See section 8.5. for a discussion of macros.)

6) If no match is found among the temporary macros, the name is assumed to be the name of a permanent macro. If a file with the specified string as name and type :MACR exists, it is taken as a permanent macro and processed. The file system will ensure that

if a file with the specified name is not found under the current user, the directory of user SYSTEM is searched.

7) If none of the above lead to a successful match, the error message NO SUCH COMMAND OR DOMAIN is printed on the communication device, and no further action will result from the entered input.

If a domain has been given the name of or a legal abbreviation of a command or standard domain, the words RECOVER-DOMAIN may not be left out.

8.1.4. GO

GO <address>

<address> - an address within the domain.

Starts the execution of an ND-500 program at the specified address.

8.1.5. CONTINUE

CONTINUE

The execution is restarted at the current program counter. There is one exception: if a program has stopped normally (by MON 0 or a stack underflow trap) the execution is started at the original start address.

If the execution has stopped because of a breakpoint, the original instruction will be restored. If the breakpoint is a permanent breakpoint, a single instruction is performed, and the original instruction is replaced by a breakpoint instruction before the execution is started.

If the execution has stopped because an escape character was typed, the execution will be restarted where it stopped. Files will remain opened after an escape, and the program will continue as if nothing had happened.

8.2. Standard domains

The search procedure employed when a command is issued will not find domains belonging to user SYSTEM until the name has been ruled out as a monitor command or as the name of a domain belonging to the current user. In particular if the file system is heavily loaded, the opening and reading of the description file may take some time, and increase the system load even more.

To speed the search for commonly used systems, like compilers or the NLL loader, these domains may be defined as standard domains. The names of standard domains are entered in a table that is searched by the monitor before the description file of a user is opened. This will reduce startup time.

Standard domain names are stored in a segment used by the monitor, therefore the name table will not survive a cold start ( )HENT/22!). After a warm start (masterclear/load), the name table is intact.

Standard domains in many respects resemble "reentrant subsystems" in the ND-100, and essentially, the rules are the same. Standard domains may only be defined and deleted by user SYSTEM, but they may be listed by any user.

8.2.1. DEFINE-STANDARD-DOMAIN

DEFINE-STANDARD-DOMAIN <standard domain name> <domain name>

<standard domain name>- the name to be used when calling the domain. May be the same as the domain name, but may not include user name. It should not be a legal abbreviation of a monitor command.

<domain name> - name of an already loaded domain, belonging to any user

When any user issues <standard domain name>, or an unambiguous abbreviation of it, as a command, <domain name> will be started. If the user has a private domain that would otherwise have been started, the name must include the user name in parentheses.

DEFINE-STANDARD-DOMAIN is permitted for user SYSTEM only.

8.2.2. DELETE-STANDARD-DOMAIN

DELETE-STANDARD-DOMAIN <name>

<name> - name of an existing standard domain

The specified standard domain is deleted from the name table of standard domains. The domain will not be deleted, but will no longer be a standard domain.

DELETE-STANDARD-DOMAIN is permitted for user SYSTEM only, and may not be issued while the standard domain is in use.

8.2.3. LIST-STANDARD-DOMAINS

LIST-STANDARD-DOMAINS

The names of all standard domains and the segments comprising them are listed ont the output device.

This command is permitted for all users.

8.3. Commands for opening and connecting files

In most programs, files are dynamically opened and closed during program execution. In some cases it is desirable to open files explicitly through commands. This occurs in particular where Fortran programs are transported from other machines where all files must be opened through operating system commands prior to program execution, or where transportation to such machines is probable. These commands may, however, be used to open files for programs in any language allowing a file to be identified by its open file number.

The commands below are similar to the Sintran III commands with the same names, but will affect files opened for use by the ND-500.

8.3.1. OPEN-FILE

OPEN-FILE <file name>, <connect file number>, <access mode>

<file name> - the name of a file to be used by a program. Default file type is :DATA.

<connect file number> - the file number used in the program.

<access mode> - see table below.

Opens a file and connects it to a file number used in the program. If <connect file number> is 0 a file number is returned that must be used from the program.

Default number base of <connect file number> is the main format - initally octal. If a decimal number is specified, it must be followed by a D. Unit numbers in Fortran programs are decimal.

The opened file will be associated with a Sintran file number, usually ranging from 100B and upwards, in a manner equivalent to ND-100 operation. However, the monitor maintains a connect number table, allowing programs to access the file either through the Sintran file number or through the user selected connect number.

Access modes:

W	0	sequential write (OUTBT,OUTST)
R	1	sequential read (INBT)
WX	2	random read/write (RFILE/WFILE)
RX	3	random read (RFILE)
RW	4	sequential read/write (INBT/OUTBT,OUTST)
WA	5	sequential write append
WC	6	random read/write with read/write access allowed from other users (contiguous files only).
RC	7	random read with read access allowed from other users (contiguous files only).

D 8 direct transfer
DC 9 direct transfer with the file closed, modus 9.
READ 10 The system will select the access mode R, RX or D. The most optimal access mode which can be used for the file/device is selected. The following is a list of file/devices and the corresponding access mode selected by the system:

terminal:	R
tape reader:	R
indexed file:	RX
contiguous file:	D
magnetic tape:	D

WRITE 11 The system will select the access mode RW, WX or D, as for READ access above.

8.3.2. CLOSE-FILE

CLOSE-FILE <connect number>

<connect number> - the connect number of a file open from a ND-500 program or through the OPEN-FILE command.

Closes a file and disconnects the file number.

<file number> > 0 close the file open with the given number
= -1 close all files temporary open
= -2 close all open files
= -3 close all files open from the ND-500 program or by the OPEN-FILE command in the Monitor.

8.3.3. LIST-OPEN-FILES

LIST-OPEN-FILES

Lists all files opened from a ND-500 program or by the OPEN-FILE command in the Monitor. The list will appear on the output device.

Files opened locally in the ND-100 will not be listed.

8.3.4. Error returns

Monitor calls from the ND-500 may return error codes outside the range used by ND-100. These are

Code	Error message
1000B	ND-500 OPEN FILE TABLE IS FULL
1001B	FILE IS NEITHER CONTIGUOUS NOR MAG. TAPE
1002B	ND-500 OPEN FILE TABLE FOR DIRECT TRANSFER IS FULL
1003B	ERROR IN MONITOR CALL
1004B	ODD BYTE ADDRESS
1005B	ODD BYTECOUNT
1006B	TOO BIG BYTECOUNT
1007B	BYTECOUNT NOT MODULO SECTOR SIZE IN DIRECT TRANSFER
1010B	ADDRESS OUTSIDE FILE LIMITS IN DIRECT TRANSFER
1011B	BLOCK ADDRESS NOT MODULO SECTOR SIZE IN DIRECT TRANSFER
1012B	HARDWARE STATUS ERROR IN DIRECT TANSFER
1013B	ILLEGAL MONITOR CALL NUMBER
1014B	DC ACCESS NOT LEGAL ON MAG. TAPE
1015B	WRONG NUMBER OF PARAMETERS IN MON. CALL
1016B	BYTE POINTER NOT MODULO SECTOR SIZE IN DIRECT TRANSFER
1017B	DATA AREA CANNOT BE PLACED INSIDE A 64K SINTRAN III SEGMENT

These error messages may also be written with the ERMSG monitor call (MON 64). Explanations of these error messages are found in chapter 14, which contains all error messages that may be issued by the ND-500 Monitor.

File system error codes known in ND-100 are explained in Sintran III Reference Manual, ND-60.128.

8.4. Direct file transfer

8.4.1. Direct file transfer with RFILE and WFILE (disk)

Direct file transfer is a feature for optimized disk transfer to the ND-500. It allows very high transmission speeds between disk and memory, moving a maximum of one disk cylinder per request for disk transfer. (One monitor call may cause several disk transfers if the amount of data exceeds one cylinder.)

The file is opened by the OPEN-FILE command, modus D or DC or from the ND-500 program by the monitor call OPEN, modus 8 or 9. In modus 8, the file is kept open; in modus 9 the file is closed during the file transfer.

The modus 9 feature allows the user to work on a larger number of files than the maximum number of files that can be concurrently open in the Sintran III file system.

The standard calls RFILE, WFILE and WAITF are used in the ND-500 program, but there are some limitations to the parameters. See the Sintran III Reference Manual ND-60.128 for a description of these monitor calls.

The actual file transfer is performed by the monitor call ABSTR. The file system is bypassed and the mass storage device may be used in an optimal way.

The monitor calls RMAX and SMAX may be used if the file is opened (modus 8).

The limitations to the use of the standard Sintran III file system are:

- The file must be contiguous.
- Only the monitor calls OPEN, CLOSE, RFILE, WFILE, WAITF, RMAX and SMAX may be used.
- The default logical block size is equal to the hardware sector size.
- The word count in RFILE or WFILE must be a multiple of the hardware sector size of the mass storage device.
- The maximum byte pointer is not updated when DC is used.
- The data area to be transferred to or from must be contiguous in physical memory. (This is, however, automatically done by the Monitor at execution time if required.)

8.4.2. Direct file transfer with MAGTP (magnetic tape)

Direct file transfer is a feature for optimized magnetic tape transfer to the ND-500. It allows data records of arbitrary size to be transferred, bringing the transmission speed close to the maximum speed of the hardware.

The file is opened by the OPEN-FILE command, modus "D" or from the ND-500 program by the monitor call OPEN, modus 8. The modus "DC" (or 9) may not be used for MAGTP.

The monitor call MAGTP may be used in a standard way from the ND-500, but the actual transfer is performed by the ABSTR monitor call in ND-100 and it goes directly from the interface into the ND-500 memory via the DMA channel. The only limitation to the block size is the maximum size of contiguous physical memory that may be allocated.

8.5. Macro commands

Macros provide a convenient mechanism for executing the same set of commands repeatedly. This is particularly useful for programs requiring certain initialization commands to be given before execution starts, for initialization after a system restart, or for executing a set of debug commands. Each user may in fact build his own set of commands from the elementary commands available in the Monitor.

It is not possible to supply input to a program in a macro body.

Macros may be saved in files, or they may be temporary, vanishing when the Monitor is left.

8.5.1. DEFINE-MACRO

DEFINE-MACRO <macro name>
<macro body>
END-MACRO

With this command it is possible to compose new commands from the original commands or other macros.

Macros defined by this command are temporary. Permanent macros may be prepared by a text editor on a file. The file must be of type :MACR.

Every line following the DEFINE-MACRO command is taken as the macro body until the END-MACRO is encountered. END-MACRO must be written on a new line.

It is possible within the macro body to define parameters that are replaced by the actual parameters when the macro is called. A parameter is defined by

PARAMETER <parameter name>, <default value>, <prompting text>

If spaces or commas should be part of the <parameter name>, <default value> or <prompting text>, they may be enclosed in apostrophes. Otherwise, apostrophes are permitted but not required.

The first actual parameter supplied in the macro call line replaces <parameter name> used in the first PARAMETER definition; the second actual parameter replaces <parameter name> used in the next PARAMETER definition and so on. Excessive parameters are ignored.

When the macro is called, the parameters which are not specified are asked for by typing the prompting text on the communication device. If the actual parameter is empty the default value is used when expanding the macro.

8.5.2. Macro subcommands

A monitor call, MACROE (MON 400), for signalling error return from a program to the Monitor is implemented. There is a flag which is raised when the executing program is terminated by this monitor call or by a trap. The error flag is set to zero when a program is terminated normally.

Two commands may be used within a macro to test the error flag:

8.5.2.1. IF-ERROR-MACRO-STOP

IF-ERROR-MACRO-STOP

Causes the currently executing macro to abort if the error flag is set. If the error flag is reset, the processing of the macro continues.

8.5.2.2. IF-ERROR-FULL-STOP

IF-ERROR-FULL-STOP

Equal to IF-ERROR-MACRO-STOP except that all active macros are aborted if the error flag is set.

By default, the expansion of the macro is printed on the output device. This printing may be suppressed by

8.5.2.3. NOLIST

NOLIST

The printing of macro expansions is suppressed.

8.5.2.4. LIST

LIST

The printing of macro expansions to the output file is reinstated. This is the default mode when a macro expansion is started. Macro subcommands may not be abbreviated.

8.5.3. EXECUTE-MACRO

EXECUTE-MACRO <macro name>, [<parameters>]...

<macro name> - the name of an existing (temporary or permanent) macro.

<parameter> - actual parameter to replace a formal parameter in the macro. If several parameters are supplied they are separated by comma or space. The parameter may contain any character except space or comma.

The macro with the specified name is processed. Formal parameters are substituted with actual parameters. If the actual parameters are not supplied, they are prompted for with <leading text> specified in the PARAMETER definition (see the DEFINE-MACRO command).

The words EXECUTE-MACRO can be left out. The procedure used for looking up a command or macro is as follows:

1) A search is made in the list of basic commands. If a match is found, the corresponding command is executed.

2) If no command is found, the list of standard domains are searched (see section 8.2.). If there is any such standard domain, it is started.

3) If the search among the standard domains was unsuccessful, a search is made in the domains of the current user. If a domain with the specified name is found, it is started as with a RECOVER-DOMAIN command.

4) If no domain with the specified name is found, the domains of user SYSTEM are searched. If a domain with a matching name is found, the domain is started.

5) If no domain is found, the specified string is assumed to be a macro name, and a temporary macro is searched for. If any matching macro is found, it is processed. (See section 8.5. for a discussion of macros.)

6) If no match is found among the temporary macros, the name is assumed to be the name of a permanent macro. If a file with the specified string as name and default type :MACR exists, it is taken as a permanent macro and processed. The file system will ensure that if a file with the specified name is not found under the current user, the directory of user SYSTEM is searched.

7) If none of the above lead to a successful match, the error message NO SUCH COMMAND OR DOMAIN is printed on the communication device, and no further action will result from the entered input.

Temporary macros may be defined within permanent macros. Such temporary macros will be erased when the processing of the permanent macro is finished.

If a macro is given the name of, or a legal abbreviation of a command, a standard domain or a domain or a domain belonging to the current user or SYSTEM, EXECUTE-MACRO may not be left out.

Input to the program may not be supplied in a macro body.

8.5.4. RESUME-MACRO

RESUME-MACRO

The last aborted macro is resumed at the line following the one where the macro was interrupted.

8.5.5. ERASE-MACRO

ERASE-MACRO <macro name>...

<macro name> - the name of an existing temporary macro.

The named temporary macros are erased. Permanent macros are erased through the Sintran III command @DELETE-FILE <macro name>:MACR.

8.5.6. DUMP-MACRO

DUMP-MACRO <macro name>

<macro name> - the name of an existing temporary macro.

The named temporary macro will be written to a file with the name of the macro, i.e. the macro is made permanent and can at a later time be executed by using the macro name as a command. If the file does not exist, it will be created. The default type of the file is :MACR.

8.5.7. LIST-MACRO

LIST-MACRO (<macro name>)...

<macro name> - a macro name or abbreviation of names of the macros to be listed. Default is all macros defined.

The names and contents of the macros with names matching the specified name are listed on the output device.

Only temporary macros are listed. Permanent macros may be listed by the Sintran III command @LIST-FILES <macro name>:MACR, TERMINAL.

8.6. Debugging commands

Before any debugging commands are used, a program must be moved into the user's virtual memory. This is done by the PLACE-DOMAIN or DEBUG-PLACE command, or implicitly by a RECOVER-DOMAIN. If patches to the program segment are to be done, DEBUG-PLACE must be used.

After program termination, either normal or error termination, the program is still in the virtual memory space and may be inspected or modified before restart. The program may be restarted by either one of the commands RUN, CONTINUE, GO or STEP. After error termination, the P register contains the address of the instruction following the last instruction executed. Depending on the kind of error, continuing execution from this address may be menaingless.

8.6.1. DEBUG-PLACE

DEBUG-PLACE <domain name>

<domain name> - the name of an existing domain.

The program segments as well a the data segments will be copied to the swap file. This allows patches to be done to the program segment. Patches are not permanent. In order to do permanent patches, LOOK-AT-PROGRAM must be used.

Otherwise, this command works exactly like PLACE-DOMAIN.

8.6.2. BREAK

BREAK <address>, [<count>]

<address> - the program address where a breakpoint is to be set.

<count> - one plus the number of times the breakpoint should be ignored before a break is performed. Default value is 1.

This command sets a breakpoint at the specified address. If a positive number is specified for the count argument, the breakpoint will be passed <count>-1 times before reaction.

When the breakpoint is reached, execution terminates and control is passed to the command processor.

After a breakpoint has been reached, program or data locations or the registers may be displayed or modified. The display format may be changed at will. Control flow or data location tracing may be initiated and terminated. The next instruction to be executed is by default the instruction pointed to by the P register, but this may be overridden by the GO command or the optional <execution start>

parameter of the STEP command.

When execution is continued by the STEP or CONTINUE command, the original instruction is restored and a single step is performed followed by a reinsertion of the breakpoint. If a non-default execution start address was selected, the original instruction in the break address is not executed, and the breakpoint instruction is retained.

It is possible to set new breakpoints as long as the Monitor has memory space to store information about them. New breakpoints are given a number for identification purposes.

8.6.3. TEMPORARY-BREAK

TEMPORARY-BREAK <address>, [<count>]

<address> - the program address where the breakpoint is to be set.

<count> - one plus the number of times the breakpoint should be ignored before a break is performed. Default is 1.

Similar to BREAK except that when the breakpoint is reached, the original instruction is permanently restored, and will not cause a break next time the instruction is executed.

8.6.4. STEP

STEP [<step start>], [<execution start>], [<count>]

<step start> - the program address where single step execution should start. Default is the current value of the program counter.

<execution start> - the program address where execution should start. Default is the current value of the program counter.

<count> - one plus the number of times the address specified as <step start> should be passed before single step execution is started. Default is 1.

Single step. If no parameter is given, the instruction pointed to by the program counter is disassembled and shown on the output device. By typing carriage return, this instruction will be executed. The next instruction will then be disassembled and shown on the output device and will be executed when another carriage return is given.

Typing anything else than a single carriage return causes return to the command processor of the Monitor.

If the <step start> parameter is given, normal execution is started from the current program counter, and single step is provided when the <step start> address is reached. If, in addition, the <execution start> parameter is given, the execution is started at the specified address rather than from the current program counter. The <step start> address will be passed <count>-1 times before single step is provided; the default value will start single step execution as soon as the indicated <step start> address is reached.

This command may be used immediately after a domain has been placed in memory by the PLACE-DOMAIN (or DEBUG-PLACE) command. More commonly it is used when the program is in a temporary halt state after a breakpoint has been detected. A break is then inserted immediately before the program address where the tracing should start. From this point on, single instruction execution is started. If desired the contents of any register or data location may be inspected after each instruction executed. Any intermediate command (other than CR) will require that STEP be respecified in order to continue single step execution. Default parameters to the STEP command will cause the next instruction in sequence to be executed.

8.6.5. LOOK-AT commands

By this set of commands it is possible to display and modify register and locations in program and data memory.

An address in the current segment is specified by its 27 bit segment relative address. An address in an arbitrary segment may be specified as

<segment no>'<segment relative address>

Generally, modification of program or data is not permanent. The modifications are made on a copy of the original :PSEG or :DSEG file. However, LOOK-AT-PROGRAM will make permanent modifications to the segment.

The LOOK-AT commands have a set of subcommands as follows:

cr carriage return causes display of the next item (register, instruction, memory cell).

EXIT Return to the Monitor command processor.

Special notation used with the slash (/, indirect) command:

m = address or register name.
n = number of bytes.
cr = carriage return.

m/cr Take the value of m as the address and display this location. m may also be a register name.

/cr Take the contents of the current location as next address and display this location. If the current location is a register, displaying of the memory is started. Specifying the P or the L register cause the program memory to be displayed, while the rest of the registers cause the data memory to be displayed.

m,n/cr Take the value of m as next address and display n bytes. m may also be a register name.

,n/cr Same as /cr except that n bytes are displayed.

Dumping of register, memory or segment to file:

m,n <output file> cr . Same as m,n/cr except that the output is written to the specified file. The file is closed upon exit from LOOK-AT.

,n <output file> cr . Same as ,n/cr except that the output is written to the specified file. The file is closed upon exit from LOOK-AT.

HELP Listing of all LOOK-AT subcommands

n cr Modifications of memory or registers are done by typing the new value in the current main format (octal, hexadecimal or decimal as set by the MAIN-FORMAT command) followed by carriage return. It is possible to use other formats than the main one by typing B, H or D before the carriage return for octal, hexadecimal or decimal respectively.

'XXX'cr Modifying the data memory or a data segment by ASCII characters may be done by enclosing the ASCII string in quotes.

CODE Modification of program memory is possible by the command CODE followed by an ND-500 assembler instruction. The instruction will be assembled and stored starting at the current location. Program memory may also be modified numerically by first typing BY, and thereafter modifying bytes in the main format (See the MAIN-FORMAT command).

BYTE
HALFWORD
WORD
FLOAT
DOUBLEFLOAT

ASCII When displaying data memory it is possible to use byte, halfword, word single or double precision float or ASCII characters as a display unit. Changing from one unit to another is done by simply typing BYTE, HALFWORD, WORD, FLOAT,

DOUBLEFLOAT or ASCII.

PERMIT-DEPOSIT In order to avoid unintended modification of the memory or a register, the command PERMIT-DEPOSIT must be typed before the depositing of a new value can take place.

EXTRA-FORMAT <format> ... In a LOOK-AT command it is possible to temporarily specify that memory locations shall be displayed in the the indicated formats in addition to the main format by the EXTRA-FORMAT command. This command is similar to the global EXTRA-FORMAT command, except that the extra formats are only valid within LOOK-AT.

ABSOLUTE <address> When relative addresses are displayed (LOOK-AT-STACK and LOOK-AT-RELATIVE), new addresses (number followed by a slash) are taken as relative addresses. However, displaying from an absolute address can be done by the ABSOLUTE command.

NEW-SEGMENT <segment no> The specified segment number will be set as current segment. Addresses specified without a segment number will be in the new current segment. The segment number is valid only while in LOOK-AT mode, and must be respecified next time LOOK-AT mode is entered.

In a LOOK-AT command it is possible to change to one of the other LOOK-AT commands by typing one of the subcommands below. This is equivalent to EXITing from LOOK-AT and reenter to inspect or modify another area (program, data or registers), but EXTRA-FORMAT need not be respecified, and it is faster. These subcommands are:

DATA <address>
PROGRAM <address>
REGISTER <register name>
<register name>
STACK
RELATIVE <relative to>

8.6.5.1. LOOK-AT-PROGRAM

LOOK-AT-PROGRAM <address>, [<domain>]

<address> - the segment address from where inspection should start.

<domain> - the name of an existing domain. Default is inspection of the domain currently in memory.

Displays and modifies program memory or program segments. The display is started at the specified <address>.

If <domain> is specified, the program segment file is displayed and may be modified. Only one segment may be displayed and modified at a time.

If <domain> is not specified, the default is the domain currently in memory. The memory image is inspected, rather than the original segment from which it was loaded. If any modifications are made, the domain must have been placed in memory by the DEBUG-PLACE command, otherwise no modification is legal.

8.6.5.2. LOOK-AT-DATA

LOOK-AT-DATA <address>, [<domain>]

<address> - the segment address from where inspection should start.

<domain> - the name of an existing domain. Default is inspection of the domain currently in memory.

This command is similar to LOOK-AT-PROGRAM except that the data memory or data segment is involved. Modification is always permitted.

8.6.5.3. LOOK-AT-STACK

LOOK-AT-STACK

The current local data field is displayed. This is the memory area pointed to by the current B register, and contains the subroutine call information, such as address local data field of calling routine (PREVB), return address (RETA), number of arguments to the routine (N), the current top of stack (SP) and an auxillary location for language processes (AUX) not used by hardware. At the next addresses are the addresses of the routine arguments, and the local variables of the routine.

The standard locations are labeled with the symbolic names above. For the argument addresses and the local variables two addresses are given, the global address and the address relative to the start of the local data field.

8.6.5.3.1. Subcommands PREVIOUS and NEXT

PREVIOUS

Display the previous local data field, i.e. the local data field of the procedure calling the current one. Several PREVIOUS commands may be given, each descending one more level in the call sequence. It is not possible to move beyond the data field of the main program (the lowermost stack frame).

NEXT

Display the next local data field, i.e. the local data field of the procedure called by the current one. Valid only after PREVIOUS. It is not possible to move beyond the data field of the routine currently being executed (the uppermost stack frame).

8.6.5.4. LOOK-AT-RELATIVE

LOOK-AT-RELATIVE (<relative to>)

<relative to> - B, R, I1, I2, I3, I4 or a numeric address. Default is R.

Start listing of data memory relative to either the contents of the R, B, I1, I2, I3 or I4 register or an address. Both global and relative address are displayed.

8.6.5.5. LOOK-AT-REGISTER

LOOK-AT-REGISTER [<register name>]

<register name> - the name of one of the registers. Default is P.

The specified register is displayed in current main format. If carriage return is typed, the next register in the sequence below is displayed. Registers identified as MIC are used by the microprogram and are not available to the user. Register sequence:

```
P,    L,    B,    R,    I1,   I2,   I3,   I4,
A1,   A2,   A3,   A4,   E1,   E2,   E3,   E4,
ST1,  ST2,  PS,   TOS,  LL,   HL,   THA,  CED,
CAD,  MIC,  MIC,  MIC,  MIC,  OTE1, OTE2, CTE1,
CTE2, MTE1, MTE2, TEMM1,TEMM2
```

8.6.6. SET-MEMORY-CONTENTS

SET-MEMORY-CONTENTS <from address>, <up to address>,
<value>, (<datatype>)

<from address> - the lower limit for modification of memory.

<up to address> - the upper limit for modification of memory.

<value> - the value to be written in the specified area.

<datatype> - BYTE, HALFWORD, WORD, FLOAT or DOUBLEFLOAT indicating the size and type of the specified <value>. Default is WORD.

The data memory is filled with the specified value from the first address specified up to the second specified address, inclusively.

8.6.7. MAIN-FORMAT

MAIN-FORMAT <format>

<format> - OCTAL, HEXADECIMAL or DECIMAL or abbreviation of one of these.

When displaying registers, memory contents, or segments with the LOOK-AT commands, the specified <format> is used. When the Monitor is started, octal is used as the main format.

The default MAIN-FORMAT may be modified by using the MAIN-FORMAT command, then leaving the Monitor by the EXIT command. The memory image can then be copied to a file by using the Sintran III command @DUMP. The :PROG file created by the @DUMP command will be equivalent to the existing monitor, but the default MAIN-FORMAT is as specified before the @DUMP. Refer to the Sintran III Reference Manual ND-60.128 for a description of the @DUMP command.

8.6.8. EXTRA-FORMAT

EXTRA-FORMAT <format> ...

<format> - one of the formats listed below or an unambiguous abbreviation of one of them.

With all commands displaying memory or segment contents it is possible to have the locations displayed in various formats in addition to the format specified in the MAIN-FORMAT command. Data and instructions are then displayed in both the format(s) specified in this command as well as the main format. The alternatives are:

BYTE The displayed location is divided into bytes and they are displayed in the main format.

HALFWORD Similar to BYTE, except halfwords are displayed. This is effective only when displaying words or doublewords as main format.

WORD Similar to BYTE, except words are displayed. This is effective only when displaying doublewords as main format.

FLOAT Single precision floating point format.

DOUBLEFLOAT Two consecutive words are displayed in double precision floating point format.

ASCII ASCII format.

OCTAL Number base for BYTE, HALFWORD and WORD display.

HEXADECIMAL Number base for BYTE, HALFWORD and WORD display.

DECIMAL Number base for BYTE, HALFWORD and WORD display.

8.6.9. TRACE

TRACE <address>, <datatype>

<address> - the address of the variable to be traced (lowermost byte).

<datatype> - BYTE, HALFWORD, WORD, FLOAT or DOUBLEFLOAT or abbreviation of one of these, indicating the size of the data element to be traced.

Whenever the location starting at the specified address is modified during program execution, its new value is displayed on the output device.

This command uses the low and high limit registers, LL and HL of the ND-500 exclusively, i.e. the previous command using these registers (GUARD or TRACE) will be discontinued.

8.6.10. GUARD

GUARD <address>, <datatype>, [<limit1>, [<limit2>]]

<address> - the address of the variable to be guarded (lowermost byte)

<datatype> - BYTE, HALFWORD, WORD, FLOAT or DOUBLEFLOAT or abbreviation of one of these, indicating the size of the data element to be traced.

<limit1> - the lower limit of the legal value range or upper limit of prohibited range.

<limit2> - the upper limit of the legal value range or lower limit of prohibited range.

If no limits are given, any modification of the location specified in this command causes a guard violation error and gives control back to the command processor whenever the specified "guard area" is modified. The "guard area" starts at <address>, and <datatype> determines the size, from one to eight bytes.

If one or two limits (forming a legal range) are specified, the new value of the guard area is checked against this range. If the value is outside the range, it is a conditional guard violation and the control is transferred to the command processor. If <limit1> <= <limit2>, then the permitted range is <limit1> <= n <= <limit2>. If <limit1> > <limit2> the new value n is legal if n < <limit2> or n > <limit1>.

If the variable has a value outside the permitted range at the time the command is given, this is not trapped. The check is made on assignments (store operations) to the variable only.

If only <limit1> is specified, then <limit2> is set equal to <limit1>, allowing the variable to take the specified value only.

This command will cause a considerable load on the ND-100 if frequent modifications of the guarded area are made.

This command uses the LL and HL registers exclusively to delimit the start address and uppermost address of the guarded variable. The previous command (GUARD or TRACE) using these registers will be discontinued.

8.6.11. BRANCH-TRACE

BRANCH-TRACE [<start address>], [<mode>], [<file name>]

<start address> - the program address where tracing should start from.

<mode> - DUMP or COMPARE. Default is DUMP.

<file name> - the file to which the tracing should be dumped or compared.

This command initiates tracing of the program counter upon branch trap conditions.

When only one or no parameters are specified, the tracing is written to the output device.

If <start address> is specified, the tracing is started when the execution reaches the specified address. Otherwise the tracing is started immediately.

In order to catch difficult bugs, it is possible to save the tracing and compare it to another run, possibly with another data set or on another machine. <mode>=DUMP will dump the tracing to the specified file and <mode>=COMPARE will compare the trace with the dumped trace on the specified file. Any differences detected will be printed on the output device.

The information is stored on <file name> in a binary format, and is not readable like the symbolic output obtained if no <file name> is specified. If a symbolic dump to a file is wanted, use the OUTPUT-FILE command before BRANCH-TRACE.

8.6.12. CALL-TRACE

CALL-TRACE [<start address>], [<mode>], [<file name>]

<start address> - the program address where tracing should start from.

<mode> - DUMP or COMPARE. Default is DUMP.

<file name> - the file to which the tracing should be dumped or compared.

This command initiates tracing of the program counter upon call trap conditions.

When only one or no parameters are specified, the tracing is written to the output device.

If <start address> is specified, the tracing is started when the execution reaches the specified address. Otherwise the tracing is started immediately.

In order to catch difficult bugs, it is possible to save the tracing and compare it to another run, possibly with another data set or on another machine. <mode>=DUMP will dump the tracing to the specified file and <mode>=COMPARE will compare the trace with the dumped trace on the specified file. Any differences detected will be printed on the output device.

The information is stored on <file name> in a binary format, and is not readable like the symbolic output obtained if no <file name> is specified. If a symbolic dump to a file is wanted, use the OUTPUT-FILE command before CALL-TRACE.

All routine calls including run time library routines are traced.

8.6.13. **EXHIBIT-ADDRESS**

EXHIBIT-ADDRESS <program address>, <data address>, (<data type>)

<program address> - the instruction that causes the specified variable to be displayed when executed.

<data address> - the address of the variable to be displayed.

<data type> - BYTE, HALFWORD, WORD, FLOAT or DOUBLEFLOAT, indicating the size of the variable to be displayed. Default is WORD.

With this command a breakpoint is set in the specified <program address>. When the execution reaches this breakpoint, the <data address> and its contents are written to the output device. The data type of the variable may be specified.

Several variables may be traced simultaneously with this command, as long as the Monitor has room for information about the breakpoints.

8.6.14. DEBUG-STATUS

DEBUG-STATUS

Lists information about previously used debug commands. Enabled traps, breakpoints, and the use of the LL and HL registers are listed.

8.6.15. ENABLED-TRAPS

ENABLED-TRAPS

Lists the contents of the own trap enable register (OTE) of the current domain and the mother trap enable register. Enabled traps, either in the current domain or in ND-100, are listed on the output device.

8.6.16. STATUS

STATUS

Lists the contents of the status register. Some of the status bits have no corresponding bit in the trap enable registers. These bits are always listed with name and value. If other status bits are set, their names and values are listed.

8.6.17. RESET commands

In order to clear the effect of previously used debugging commands, the ND-500 Monitor has several reset commands. These are:

8.6.17.1. RESET-DEBUG

RESET-DEBUG

will clear the effect of all previously used debugging commands.

8.6.17.2. RESET-BREAKS

RESET-BREAKS <break number> ...

The breakpoints with the specified numbers are removed by using this command. If the last active breakpoint is removed, the breakpoint bit in the ND-500 CTE register is reset.

If no <break number> is specified, every active breakpoint is removed permanently by typing RESET-BREAKS. If one or more <break number>s are specified, only those breakpoints are removed.

'Breakpoint' includes, in addition to those set by the BREAK command, breakpoints set by the EXHIBIT-ADDRESS command.

8.6.17.3. RESET-LAST-BREAK

RESET-LAST-BREAK

When a breakpoint is encountered during execution, this breakpoint may be removed and the original instruction restored by executing this instruction.

8.6.17.4. RESET-TRACE

RESET-TRACE

The tracing specified in the TRACE command is discontinued.

8.6.17.5. RESET-GUARD

RESET-GUARD

The guarding of the area specified in the GUARD command is discontinued.

8.6.17.6. RESET-CALL-TRACE

RESET-CALL-TRACE

Dumping or comparing with previous dump of routine calls is discontinued.

8.6.17.7. RESET-BRANCH-TRACE

RESET-BRANCH-TRACE

Dumping or comparing with previous dump of branch conditions is discontinued.

8.7. Commands for performance measurement

Performance measurement commands serve two main purposes: the HISTOGRAM-commands and MONCALL-LOG commands are used to evaluate one program in order to detect bottlenecks in time critical sequences, while the LOG-commands measure the load on the system in order for the system supervisor to set the system parameters properly.

The histogram and log commands all use the same buffer, and there is only one buffer in the system. Therefore, only one user may use these commands at a time, and he must either release the buffer explicitly or leave the monitor (implicitly releasing the buffer) before any other user may use it.

If a user attempts to execute any of the log or histogram commands while the buffer is in use, an error message is issued.

8.7.1. Histogram commands

8.7.1.1. SET-HISTOGRAM

SET-HISTOGRAM <start address>, <max. address>, (<no. of intervals>)

<start address> - the lower address of the area to be measured.

<max. address> - the upper address of the area to be measured.

<no of intervals> - the number of equally sized intervals between <start address> and <max address> in the range 1:64 decimal. Default is 64 decimal.

This command will reserve and clear the histogram buffer.

A subsequent START-HISTOGRAM will start sampling the accesses to the instruction bank between the <start address> and the <max. address>. This area is divided into <no. of intervals> equally sized intervals. The maximum size of an interval is 32767 bytes.

8.7.1.2. START-HISTOGRAM

START-HISTOGRAM

The sampling of the program counter will be started. The sampling may be started and stopped any number of times before the histogram is printed. The buffer is not cleared before sampling is started; samples will be added to what is already in the buffer.

Samples are taken every 20 ms.

8.7.1.3. STOP-HISTOGRAM

STOP-HISTOGRAM

This command stops the histogram sampling.

8.7.1.4. PRINT-HISTOGRAM

PRINT-HISTOGRAM

This command prints the histogram on the output device. If sampling has been started and stopped several times, the histogram will represent the sum of all samples since SET-HISTOGRAM. The histogram buffer is not cleared by PRINT-HISTOGRAM.

8.7.1.5. RELEASE-HISTOGRAM

RELEASE-HISTOGRAM

This command releases the histogram buffer. This means that other users may use the HISTOGRAM, the PROCESS-LOG, the MONCALL-LOG and the SWAPPING-LOG commands.

If the buffer is not released through this command, it will automatically be released when the user leaves the Monitor.

8.7.2. Monitor call logging

8.7.2.1. START-MONCALL-LOG

START-MONCALL-LOG

This command will clear the log buffer, and reserve it for the user issuing the command. All monitor calls executed from the ND-500 will be logged. A count of the number each monitor call has been executed can later be printed.

Roughly speaking, the load on the ND-100 CPU imposed by the ND-500 is proportional to the number of monitor calls executed from ND-500. (Obviously, this general rule applies to CPU load only, not to file system and channel load.) Isolating programs that perform a disproportionate number of monitor calls may help increasing ND-100 throughput.

8.7.2.2. PRINT-MONCALL-LOG

PRINT-MONCALL-LOG

A count of monitor calls executed since START-MONCALL-LOG is printed on the output device. Each monitor call number up to 777B is listed with an individual count. Parts of this range is not valid as monitor call numbers, and will always appear with a count of zero.

This command does not release the buffer, nor does it clear it. Further monitor calls will add to the count already in the buffer.

8.7.2.3. STOP-MONCALL-LOG

STOP-MONCALL-LOG

The log buffer is released, and no further logging of monitor calls will be done.

Other users may use the HISTOGRAM, the PROCESS-LOG, the MONCALL-LOG and the SWAPPING-LOG commands. If the buffer is not released through this command, it will automatically be released when the user leaves the Monitor.

8.7.3. Process logging

8.7.3.1. START-PROCESS-LOG-ALL

START-PROCESS-LOG-ALL

This command will clear the process-log buffer and reserve it for the user issuing the command.

Logging the CPU usage of the active processes is started. Samples are taken every 20 ms, and the measurements are represented as percents of the total CPU capacity. The result of the logging may be presented by the PRINT-PROCESS-LOG command.

This command is allowed for user SYSTEM only.

8.7.3.2. START-PROCESS-LOG-ONE

START-PROCESS-LOG-ONE <process number>

<process number> - the process identifier found by the WHO-IS-ON command.

Logging of one specified process is started. The percentage of the time spent by the process in the states 1) Idle, 2) Waiting for swapper, 3) Using swapper, 4) In monitor call, 5) Active, and 6) Waiting for CPU, are logged. The 'active' entry (5) is equal to the entry that would appear in the START-PROCESS-LOG-ALL command for the specified process.

This command is allowed for user SYSTEM only.

8.7.3.3. PRINT-PROCESS-LOG

PRINT-PROCESS-LOG <first process>

<first process> - the lowest numbered process to be printed. Default is 0.

The accumulated measurements from the last START-PROCESS-LOG-ALL or START-PROCESS-LOG-ONE are printed on the output device. The buffer is not cleared, and the logging is continued, adding subsequent measurements to the printed values. In order to clear the buffer, the START-PROCESS-LOG-ALL or START-PROCESS-LOG-ONE should be used to start the next logging period.

This command is allowed for user SYSTEM only.

8.7.3.4. PROCESS-LOG-ALL

PROCESS-LOG-ALL <interval> <first process>

<interval> - the time in seconds between each report.

<first process> - the lowest numbered process to be logged. Default is 0.

The logging of CPU usage in percent of total capacity is started and written to the output device every <interval> second. The buffer is cleared between each report; displayed results are not cumulative.

A sample is taken every 20 millisecond, and for the report to have a reasonable accuracy, the interval should be at least 10 seconds. The logging is stopped by pressing the escape key.

This command is allowed for user SYSTEM only.

8.7.3.5. PROCESS-LOG-ONE

PROCESS-LOG-ONE <process no> <interval>

<process no> - the identifying number of the process, found by the WHO-IS-ON or PROCESS-STATUS command.

<interval> - the time in seconds between each report.

The logging of the specified process is started, and the log printed every <interval> seconds. The buffer is cleared between each report; displayed results are not cumulative. The report contains the same measurements as measured by the START-PROCESS-LOG-ONE command.

A sample is taken every 20 milliseconds, and for the result to have a reasonable accuracy, the interval should be at least 10 seconds.

The logging is stopped by pressing the escape key.

This command is allowed for user SYSTEM only.

8.7.3.6. RELEASE-LOG-BUFFER

RELEASE-LOG-BUFFER

The buffer used for the SWAPPING-LOG and PROCESS-LOG-commands is released, allowing other users to use these commands, the HISTOGRAM- and MONCALL-LOG-commands.

If the buffer is not released through this command, it will be released when the user leaves the Monitor.

8.7.4. SWAPPING-LOG

SWAPPING-LOG <interval>

<interval> - the period in seconds between each report.

This command will clear the log buffer and reserve it for the user issuing the command. The buffer is the same as the one used in the PROCESS-LOG, MONCALL-LOG and HISTOGRAM commands, which means that only one user at a time can use any of these commands.

Logging of swapping is started, and will be written to the output device every <interval> seconds. The logging is stopped by pressing the escape key.

Each report will include values for the last interval, the average per interval since logging was started and the total. For each of these, a count of page faults, transfers, the total free space etc. will be listed.

This command is allowed for user SYSTEM only.

8.7.5. LIST-EXECUTION-QUEUE

LIST-EXECUTION-QUEUE <interval>

<interval> - time in seconds between each report

The currently executing program, its priority, the queue of jobs for the ND-500 and their priorities are listed on the output device every <interval> seconds.

8.8. Process communication flags

A simple mechanism for communication between an ND-100 process and an ND-500 process is implemented.

To each process two 32 bit words are assigned, the input and output flags. The owner process may read its own input flag and write into its output flag by the monitor calls Read input flag (MON 402) and Write output flag (MON 403). When the Monitor is entered, both flags are initially zero. The flag word is not used by the monitor, and may contain any information as determined by the process(es).

A ND-100 program may use functions 100B and 101B in the Sintran III monitor call N500M (MON 60) to communicate with an ND-500 process.

From a terminal the same functions are performed by the commands GET-FLAG and SET-FLAG described below.

Note that there is no queueing of flags; if the input flag of a process is modified twice before the owner reads the flag, the first value is lost.

8.8.1. GET-FLAG

GET-FLAG <process no.>

<process no.> - an unsigned value in the range 0:3777777777777B.

The output flag (32 bit word) of the specified process is written on the output device in the current main format. If the specified process is connected to a terminal, this command must be given from another terminal.

8.8.2. SET-FLAG

SET-FLAG <value>

<value> - an unsigned value in the range 0:3777777777777B.

The specified <value> (32 bit word) is written into the input flag of the specified process. If the specified process is connected to a terminal, this command must be given from another terminal.

8.9. Memory allocation

System performance depends on how the active process uses its memory and how the entire available memory is administered. When performance is critical it is possible to allocate memory explicitly to the process by several commands described below.

The total execution time of a process may vary within wide limits, depending on the amount of physical memory that the process is allotted and the allocation strategy employed.

All commands for memory allocation are reserved for user SYSTEM if executed in the Monitor. If fixing has been specified by a non-privileged user in NLL, this specification is ignored at execution time unless user SYSTEM executes the domain.

Explicit allocation is very rarely needed. Whenever hardware considerations require it (direct transfer files, communication with RTCOMMON or with an ND-100 segment), this will automatically be taken care of by the Monitor at execution time.

In general, physical memory is significantly smaller than the sum of the logical sizes of all processes submitted for execution. Physical memory may even be smaller than the size of each one of the processes.

To overcome space problems, a memory management system is used, mapping the logical address spaces onto physical memory through a translation mechanism. Each logical address space is divided into pages, or blocks of 2k bytes (2048 bytes). Page boundaries will always be at physical addresses which are multiples of 2048 (4000B).

8.9.1. Demand paging

It is not necessary for all the pages of a segment to be in memory when the process starts executing. If access to a page not in memory is attempted, this is detected by hardware as a page fault, and the running process is suspended until the page has been copied from disk.

Due to the translation mechanism, the page brought into memory may be placed wherever there is room for it. Thus, several users may have fractions of their programs scattered in memory.

Whenever the memory is full, and there is no room for a page that is needed, another page must be removed to free the physical page. If the page to be removed has not been modified (as is normally the case for program segments) the page does not need to be written back. If it has been modified, it must be written back to the disk before another one takes its place in memory. This process is called swapping, and is in the ND-500 performed by a system process called the swapper, running in the ND-500 independently of any terminal.

The algorithm used to select a page for removal attempts to find the page that has the least probability of being used again, and will roughly speaking pick the page that has remained unused in memory for the longest time.

This allocation strategy, called demand paging, is the default strategy used, to achieve optimal utilization of physical memory.

8.9.2. "Fixing" in memory

Certain I/O operations require that the data area to be transferred to or from is located in a contiguous area in memory. The Monitor will recognize such requirements, and will allocate an area before the first I/O transfer is started. The memory area will remain reserved ("fixed") until the program completes execution. The user need not be concerned with this at all, as the operation is completely automatic and transparent.

If several processes use exceedingly large areas for I/O operations requiring the data area to be fixed in memory, this will affect system performance to some degree, as it limits the number of pages available for swapping.

8.9.3. Limiting the number of pages in memory

In order to ease the load on the swapper, the user may specify the minimum and maximum number of pages to be kept in memory during execution of the segment. The segment will still be treated as a demand segment, but memory requests from other processes will never cause the number of pages to drop below the specified lower limit. Thus, the number of page faults during execution is reduced.

Initially, at the start of program execution, no pages are in memory. The minimum number of pages does not apply until that number of pages have been brought into memory as a result of page faults. However, none of these pages will be swapped out.

The maximum number of pages is used to indicate the approximate size of the working set. As soon as the number of pages in memory exceeds this limit, the least recently used pages are marked for swapping.

This can be used with advantage for processes passing through a data set in a sequential manner, or processes having a large amount of initializing code and a small working set as soon as the initialization is done. Although the gain in speed is lower than it would be with fixed allocation, the penalty in reduced performance of the ND-500 system as a whole is far less.

The minimum and maximum number of pages in memory are set by the SET-SEGMENT-LIMITS command.

8.9.4. "Fixing" programs in memory

The suspension of a process while a page transfer takes place makes response time of a process more irregular than if the entire process could be in memory at the same time. Another aspect of swapping is that the busier the CPU gets, the more crowded memory will get (in general), and the more time is spent at swapping.

Ordinary interactive processes, or processes operating on permanently stored data can usually tolerate the delay imposed by demand paging. However, programs interacting with I/O-devices will often depend on short and well defined response times or rely upon data areas being in memory at any time for performing I/O operations.

Processes with such requirements may be exempted from swapping, and be allowed to keep all or some of their pages in memory continously. If the segment remains in memory even before and after execution, controlled by explicit commands, it is said to be fixed in memory. Several degrees of fixing are possible.

Even though part of a segment is declared to be a fixed segment, this does not have to apply to the entire segment. The commands available have parameters for specifying the lower and upper bound of the segment area affected. The remainder of the segment will be treated as a demand segment.

Under normal circumstances, the Monitor will detect the conditions requiring special memory allocation, and perform the allocation at execution time. Thus, the user does not have to be concerned about the commands to perform explicit allocation. For time critical tasks, however, they may be required.

8.9.5. Fixing segments scattered in memory

This kind of allocation will scatter pages throughout memory, exactly like demand paging, but these pages will not be candidates for swapping, and will not be removed when the program completes its execution.

Start-up times will be significantly shorter than for demand segments, as the segment will already be in memory and no disk access is required. If the process is restarted after completion before the segment is unfixed, no extra disk accesses are introduced.

The effective memory size available for demand paging processes will be reduced, causing an increased swapping for these segments, but the Monitor is free to place the fixed area wherever seems most suitable at the time the process is submitted for execution.

Fixing a segment scattered in memory is done by the FIX-SEGMENT-SCATTERED command.

8.9.6. Fixing segments in contiguous memory

Some I/O operations, namely all DMA operations, require that the area to be transferred to or from is a contiguous memory area, as the DMA device does not use the translation mechanisms to transform the logical address to a physical one.

In order to permit DMA transfers crossing page boundaries, the affected area may be allocated in a contiguous area in physical memory. Usually this will apply to data areas only, but may be used for any segment.

Program execution will not be any faster than with a fixed scattered segment; the translation mechanism to convert from logical addresses is not bypassed (and consists mostly of tailor made hardware, working at the speed of the CPU, causing no delay in addressing in most cases). The main advantage over fixed scattered is the ability to perform direct transfer file access, see chapter 8.4.

Allocating a contiguous area require that the swapper clears a memory area of the requested size without regard to whether the removed pages are active or not. Consequently, the allocation of a contiguous segment is rather costly. Obviously, a contiguous area of the requested size must also be available. If a high number of users request (any kind of) fixed allocation, this is not always the case, and the request is refused.

A system parameter determines the maximum number of pages that may be fixed by each user, the maximum for all users and the maximum for the entire system, including areas fixed implicitly by the Monitor. If any of these limits are exceeded, an error message will be returned.

The maximum number of contiguously fixed pages may also be limited by a system parameter.

Observe that when performing DMA through the file system, e.g. to direct transfer files, the fixing is performed automatically by the Monitor, and does not require explicit specification by the user.

Fixing a segment in a contiguous area is done by the FIX-SEGMENT-CONTIGUOUS command.

8.9.7. Fixing segments in an absolute location

This allocation strategy is even more demanding on system resources, as it includes an explicit specification of the physical memory area to be used. No further gains in speed can be achieved with absolute fixing in memory, but such allocation may be required when communicating with special I/O devices.

Communication with ND-100 may go through shared memory. In particular, the RT-COMMON area will always be contiguous in a fixed location, and ND-500 processes accessing this area will require that the logical ND-500 addresses map onto the RT-COMMON area. All communication with fixed segments in ND-100 will, however, be taken care of by the Monitor, and the user will not have to be concerned about it.

Fixing a segment in an absolute location is done by the FIX-SEGMENT-ABSOLUTE command.

8.9.8. Fixing segments shared by several processes

Obviously, if two processes want to communicate through a data segment, they must access the same physical location when they address the same logical location. If the first process to start execution requests demand paging scattering pages throughout memory, it is impossible for the second process to request contiguous allocation and map to the same addresses. (No kind of fixing is necessary in order to share a segment; it may be a demand segment if fixing is not required for other reasons.)

This also applies to fixed contiguous vs. fixed absolute, or any other combination. The process first bringing the segment into memory must allocate it with the highest grade allocation required by any of the processes accessing the segment. The highest grade allocation is fixed absolute, then follows fixed contiguous, fixed scattered and demand paging as the lowest grade.

If only a part of the segment is fixed, the first process to fix the segment must fix the maximum area requested by any process accessing the segment.

8.9.9. Unfixing a segment

After the process using a fixed segment completes execution, the segment is not automatically released. The cause for this is that processes using fixed segments often are either periodical or they are restarted as a result of an external event, and the time spent moving the segment into memory would often be too long.

Therefore, a segment must be released explicitly through the UNFIX-SEGMENT command. This function is also available as an ordinary monitor call. If the segment is not unfixed, the Monitor will perform the unfixing when the last user having fixed the segment leaves the Monitor.

The command will not necessarily have an immediate effect. If there are still one or more processes using the segment, it will not be removed from memory until all of them have completed.

8.9.10. The swapping strategy

A certain knowledge of the swapping strategy is required by the system supervisor in order to properly set the parameters determining the operation of the swapper. Programmers may also want to know how system software affects the performance of their programs and utilization of the available resources.

The swapper maintains two page lists for each physical segment: one for the socalled active pages, another for the passive pages. Active and passive refers to whether the page has been accessed since the swapper evaluated the list.

Pages in these two lists are not ordered according to use or priority. The swapper will move pages from the passive list to the active list as a result of an explicit request. Pages are moved from the active to the passive list several pages at a time, with longer intervals. This reduces the administration overhead compared to using sorted lists, but reduces the resolution in the process of selecting a victim for swapping. The moving of pages from the active to the passive list is described as cleaning the segment.

If a page is contained in either the active or the passive list, it is present in physical memory, otherwise it is swapped out to disk. A page fault will occur if it is not in the active list. When this occurs, the passive list is first searched to determine if the page is in memory. If it is found in this list, the page is moved to the active list. Otherwise, it must be fetched from disk before it is entered in the active list. Generally, a disk access will cause the requesting process to be suspended and another one activated, while moving a page from the passive to the active list will allow the requesting process to continue immediately afterwards. When a page fault occurs, this is handled by the swapper process. An available memory page is found and entered into the active pages list of the segment. If no pages are available, one process is selected as "victim", and all pages of all its segments are passivated. One of the pages are allocated to the requesting process. All these pages will be considered available next time a page fault occurs.

The victim for loosing its pages is selected according to a "round robin" scheduling among processes of the lowest priority in the system. All timesharing processes are treated as if they have a fixed priority of 20B. When a process is selected, it will be suspended for five seconds.

When a process looses all its pages, these will remain in memory, and if the process using it is activated, these pages are moved to the active list without the need for reading them from disk.

If the minimum and/or maximum number of pages in memory has been set for a segment, the swapper will ensure that the number of pages in the active list stays within the specified bounds. If the sum of active and passive pages exceed the maximum, the pages in the passive list are immediately swapped out if they have been modified and the memory pages returned to a freepool.

Before a page is taken from a still active segment, the swapper will check the freepool. This pool will contain pages never used by any segment and pages previously used by now terminated processes.

The swapper will not touch pages fixed in memory, until the last user having fixed the segment unfixes it or logs out. At that time the pages are returned to the freepool.

8.9.11. SET-SEGMENT-LIMITS

SET-SEGMENT-LIMITS <segment no.>, <type>, <min no.of pages>,
<max no.of pages>

<segment no.> - the number of an existing segment

<type> - P or D, indicating program or data segment, respectively.

<min no.of pages> - the minimum number of pages of the specified segment to remain in memory throughout process execution.

<max no.of pages> - the maximum number of pages of the specified segment to remain in memory throughout process execution.

Specifies the minimum number of pages of a segment that must be in memory before execution of instructions on the segment starts, and the maximum number of pages allowed in memory at one time.

When execution of the segment is started, no page of the segment will be a candidate for swapping until the minimum number of pages specified is brought into memory. At no time during the execution will the number of pages in memory be less than the specified minimum.

Normally, due to page faults, the number of pages in memory will increase during the execution of the domain. If this number exceeds the specified maximum, one or more of the pages not used for some time will be marked for swapping.

Pages may be swapped even if the maximum limit has not been reached, but at no time will the number of pages in memory be below the specified minimum.

8.9.12. FIX-SEGMENT-SCATTERED

FIX-SEGMENT-SCATTERED (<segment name>), (<type>),
(<lower addr>), (<upper addr>)

<segment name> - the name of an existing segment.

<type> - P or D, signifying program or data segment.

<lower addr> - the lower boundary of the area to be fixed. Default is the lowest address on the segment.

<upper addr> - the upper boundary of the area to be fixed. Default is the uppermost address of the segment.

In NLL, the default segment is the current segment. Segment may be specified either by name or by logical segment number.

<lower addr> will be rounded down, <upper addr> will be rounded up to the nearest page boundary. In NLL both may be defined symbols or addresses. The Monitor accepts addresses only.

The segment or part of segment specified is declared to be retained in memory after it has been loaded for execution, until it is explicitly released through the Monitor command UNFIX-SEGMENT. The pages belonging to the segment may be scattered throughout physical memory.

8.9.13. FIX-SEGMENT-CONTIGUOUS

FIX-SEGMENT-CONTIGUOUS (<segment no.>), (<type>),(<lower addr>), (<upper addr>)

<segment no.> - the number of an existing segment.

<type> - P or D, signifying program or data segment.

<lower addr> - the lower boundary of the area to be fixed. Default is the lowest address on the segment.

<upper addr> - the upper boundary of the area to be fixed. Default is the uppermost address of the segment.

<lower addr> will be rounded down, <upper addr> will be rounded up to the nearest page boundary.

The segment or part of segment specified is declared to be allocated in a contiguous area of memory, and to be retained in memory until it is explicitly released through the Monitor command UNFIX-SEGMENT.

8.9.14. FIX-SEGMENT-ABSOLUTE

FIX-SEGMENT-ABSOLUTE (<segment no.>), (<type>), (<phys. addr>), (<lower addr>), (<upper addr>)

<segment no.> - the number of an existing segment.

<type> - P or D, signifying program or data segment.

<phys. addr> - the address in physical memory where the segment should start.

<lower addr> - the lower boundary of the area to be fixed. Default is the lowest address on the segment.

<upper addr> - the upper boundary of the area to be fixed. Default is the uppermost address of the segment.

<lower addr> will be rounded down, <upper addr> will be rounded up to the nearest page boundary.

The specified segment or part of segment is declared to be allocated in a contiguous area in memory, starting at the physical address specified. It will remain in memory until explicitly released through the Monitor command UNFIX-SEGMENT.

8.9.15. UNFIX-SEGMENT

UNFIX-SEGMENT , <type>

 - the name of a segment which has previously entirely or in part been fixed in memory through one of the above commands.

<type> - P or D, indicating program or data segment, respectively.

The area occupied by a segment, or part of segment, previously specified as fixed in memory, is unfixed. The freed space may be used by other segments. The command has no effect before every process that has fixed the segment has released or unfixed it.

8.10. Miscellaneous commands

8.10.1. AUTOMATIC-ERROR-MESSAGE

AUTOMATIC-ERROR-MESSAGE

Error messages caused by monitor calls will automatically be written to the communication device. MON 64 (ERMSG) will then be unnecessary after every monitor call in the ND-500.

8.10.2. RESET-AUTOMATIC-ERROR-MESSAGE

RESET-AUTOMATIC-ERROR-MESSAGE

Reverses the effect of the AUTOMATIC-ERROR-MESSAGE command.

8.10.3. The "Escape" key

By pressing the Escape key during the execution of an ND-500 program the execution is stopped and the control is given to the ND-500 command processor.

No files are closed and no resources released. Execution may be resumed by the CONTINUE command, possibly after executing other monitor commands. If execution is not resumed, resources are released when the user leaves the monitor.

8.10.4. TIME-USED

TIME-USED

This command prints the ND-500 and ND-100 CPU time and clock time elapsed from the moment that the ND-500 Monitor was entered.

8.10.5. WHO-IS-ON

WHO-IS-ON

A list of users currently logged on the ND-500 is printed on the output device.

8.10.6. VERSION

VERSION

The version numbers of the currently active subsystem (background part of the monitor), system part (Sintran part of the monitor), swapper and microprogram is written to the output device.

8.10.7. SET-PRIORITY

SET-PRIORITY <ND-100 mon call priority>, <max % of ND-100 time>, <ND-500 priority>

<ND-100 mon call priority> - the priority of the ND-100 process executing monitor calls on behalf of the ND-500 process, in the range 0:70B. Default is 70B.

<max % of ND-100 time> - the maximum percentage of ND-100 CPU time the ND-100 process may use over a two second period. Default is 50%.

<ND-500 priority> - the priority of the ND-500 process, in the range 0:377B. Default is dynamic modification by the time slicing mechanism.

whenever an ND-500 process executes Sintran III monitor calls, a twin process running in the ND-100 is started. The required parameters for the call are transferred to this process, and the call is executed in the ND-100 before the results (if any) are returned to the ND-500 process.

When a monitor call is executed, the priority of the ND-100 twin process is determined by the parameter <ND-100 mon call priority>.

The <max % of ND-100 CPU time> parameter specifies the maximum percentage of ND-100 CPU time the ND-500 process may use over a two second period through its twin process executing monitor calls. If the percentage is exceeded, the <ND-100 mon call priority> is reduced to 20B.

Be aware that the measured CPU time spent in monitor call handling includes activty on interrupt level 4 and 1. Other hardware levels (for ND-500 monitor calls: 14, 12, 3 and possibly 11 and 10) are not measured. The measured CPU load will be a smaller or larger fraction of the actual CPU load. The ND-100 may be saturated even though the sum of all "max percentages" is significantly below 100%.

If <ND-500 priority> is zero, the process will be time sliced with other processes with priority varying between 20B and 61B. If <ND-500 priority> is non-zero, the process will run on a fixed priority as specified. The default handling of the ND-500 process is timeslicing with no fixed priority. A priority specified in the source program is ignored.

This command is allowed for user SYSTEM only.

8.11. Commands for the System Supervisor

These commands are allowed for user SYSTEM only, and most of them require that no other users are logged in on the ND-500. New users may be prevented from logging in by the command SET-ND-500-UNAVAILABLE.

These commands will interpret and display addresses as octal values regardless of the format set by the MAIN-FORMAT command. However, decimal or hexadecimal addresses may still be entered by trailing the parameter with D or H respectively.

The user RT has no special privileges in the ND-500, and is treated as a regular public user. This applies both to commands and the ND-500 instruction set available. However, monitor calls executed in the ND-100 are treated in the same manner as for ND-100 programs, giving the user RT higher privileges than public users.

8.11.1. SET-ND-500-UNAVAILABLE

SET-ND-500-UNAVAILABLE

No user may log on to the ND-500 until the SET-ND-500-AVAILABLE command is given. SET-ND-500-UNAVAILABLE must be used before any modification of system parameters is done, to ensure that no user interrupts critical operations. If any command that requires exclusive access to ND-500 is executed, this command is implicitly attempted, and an error message issued only if others are using ND-500. If ND-500 has been implicitly set unavailable, it will be impossible for others to use it until SET-ND-500-AVAILABLE is executed or the user reserving the ND-500 leaves the monitor.

This command will not force a log-out of those already logged in, but will prevent new users from logging on. Logged in users must log out explicitly.

8.11.2. SET-ND-500-AVAILABLE

SET-ND-500-AVAILABLE

Other users may now log in. This command has the reverse effect of SET-ND-500-UNAVAILABLE, and should be issued as soon as exclusive use of the ND-500 is no longer required. An implicit SET-ND-500-AVAILABLE is executed when the user setting it unavailable leaves the monitor.

8.11.3. STOP-ND-500

STOP-ND-500

The ND-500 CPU is stopped. When a user attempts to start an ND-500 process after this command has been executed, the microcode will automatically be reloaded, the swapper process placed in memory and started ("warm start" of ND-500).

If the ND-500 should be stopped and then started with no need for restarting running jobs, the MICRO-STOP command should be used.

8.11.4. Memory configuration

In an ND-500 computer system the processors may be connected to either local memory (memory that can be addressed from only one processor) or to a multiport memory system (shared memory). By processor, in this context, is meant the disk, the ND-100 CPU, the ND-500 CPU program channel or the ND-500 data channel.

There are two restrictions which must be noticed when configuring an ND-500 computer system. First, the physical addressing range for program and data memory may not overlap if the memory addressed is not the same physical memory.

Secondly, if the disks have access to a memory cell, it is assumed that the ND-100 CPU also has access to that memory cell, and vice versa.

The ND-500 system has itself limited capability to investigate its own memory configuration. Therefore the memory configuration must be defined by the command DEFINE-MEMORY-CONFIGURATION.

Note: Local ND-500 memory is not legal in the ND-500 multiuser Monitor.

8.11.4.1. DEFINE-MEMORY-CONFIGURATION

DEFINE-MEMORY-CONFIGURATION <ND-100 page# for ND-500 phys.addr 0>

The operating system is given information about the physical memory configuration.

The parameter is ND-100 page number for which the ND-500 physical address is zero, i.e. the difference between the ND-500 and ND-100 physical addresses for the same physical cell in common memory.

The information about the size of the system, and the access to the different memory parts of the system is given as subcommands to this command. The information given by this command is saved and will survive a normal restart ("warm start") of the system.

The subcommands will request the information

- size in number of pages for the memory part
- Does ND-100 have access to the part?
- Does ND-500 have access to the part as program?
- Does ND-500 have access to the part as data?
- Is this the last memory part?

Default is access for both CPUs, both P and D for ND-500.

When Sintran III is restarted by the MACM)HENT / 22! commands ("cold start"), the memory configuration information is lost. For convenience a permanent macro with the memory configuration definition should be made.

8.11.4.2. MEMORY-CONFIGURATION

MEMORY-CONFIGURATION

Information about the current memory configuration is printed on the output device.

8.11.5. Memory administration

When the ND-500 is started the first time, every page of ND-100/ND-500 shared memory belongs to ND-100. Memory is administered through the commands GIVE-ND-500-PAGES and TAKE-ND-500-PAGES.

8.11.5.1. GIVE-ND-500-PAGES

GIVE-ND-500-PAGES <no. of pages>

<no. of pages> - the number of pages to be used by ND-500.

The specified number of pages are taken from the ND-100 and released to the ND-500. If ND-500 already has pages, the specified number of pages is added to those ND-500 had previously.

All system tables are located in memory belonging to the ND-100. Thus, the number of pages specified will all be available for user processes.

8.11.5.2. TAKE-ND-500-PAGES

TAKE-ND-500-PAGES <no. of pages>

<no. of pages> - the number of pages to be returned to ND-100.

The specified number of pages are taken from the ND-500 and given to the ND-100. The number specified should be less than or equal to the number given to ND-500 previously with the GIVE-ND-500-PAGES command, otherwise the number of pages actually released is returned.

8.11.6. Microprogram maintainance

Using these commands require a detailled knowledge of the ND-500 microprogram format and hardware. This chapter is not assumed to give sufficient information; the reader must as a minimum be familiar with Test Micro Program Descriptions for ND-500 (ND-30.103).

8.11.6.1. MICRO-STOP

MICRO-STOP

The execution of the ND-500 microprogram is stopped, and may be resumed through the command MICRO-START. The ND-500 will stop completely, but the contents of all registers are retained. It is not necessary to restart programs running in the ND-500.

8.11.6.2. MICRO-START

MICRO-START <address>

<address> - the octal control store address where execution of the microprogram should start.

The execution of the ND-500 microprogram is started at the specified address.

8.11.6.3. LOAD-CONTROL-STORE

LOAD-CONTROL-STORE (<file name>), (<start address>), (<no. of words>)

<file name> - the name of the file from which the microprogram is read. Default is CONTROL-STORE:DATA.

<start address> - the octal address where the first microprogram word should be loaded in control store. Default is 0.

<no. of words> - the number of words to be compared with the file contents after loading. Default is 20000B (entire control store).

The ND-500 microprogram is loaded to the control store from the specified file. The first microprogram word on the file is loaded into the control store at the specified start address. Every microprogram word (144 bits, 18 bytes) loaded into successive words.

When the loading is finished, the first words of the file are compared with the corresponding contents of the control store. The number of words to be compared is specified through the <no. of words> parameter. If unequality is found, the error message CONTROL STORE UNSUCCESSFULLY LOADED is written to the output device.

8.11.6.4. COMPARE-CONTROL-STORE

COMPARE-CONTROL-STORE (<file name>), (<start address>), (<no. of words>), (<max.no. of faults>)

<file name> - the name where the microprogram is stored. Default is CONTROL-STORE:DATA.

<start address> - the octal address where the comparison should start. Default is 0.

<no. of words> - the number of words to be compared. Default is 20000B (entire control store).

<max.no.of faults>- the maximum number of unequalities accepted between the file contents and the loaded micro-program before the comparison is aborted. Default is 7 (the number of messages that will fit on a VDU screen).

The current ND-500 microprogram is compared to the microprogram residing on the the specified file, <file name>. The comparison starts at the specified microprogram address, <start address>. This word is compared to the first word on the file, etc. Four words will be modified after the microcode is loaded and will always be different.

Upon unequality the address and the two differing control store words are written to the output device. The comparison lasts until <no. of words> are compared or <max. no. of faults> are found.

8.11.6.5. LOOK-AT-CONTROL-STORE

LOOK-AT-CONTROL-STORE (<address>)

<address> - an octal address in control store, range 0:20000. Default is 0.

Examine and modify the ND-500 microprogram.

The display is started at the specified <address>. One control store word and the corresponding address are displayed on one line. On carriage return, the next control store word is displayed. A control store word consists of 144 bits which are grouped into nine 16 bit words.

The next control store word to be displayed may be specified by typing its address followed by a slash and carriage return.

8.11.6.5.1. Subcommands EDIT and ORIN

By default, the control store is disassembled and displayed symbolically. Symbolic modifying of the control store is performed by either the subcommand EDIT or ORIN. By EDIT the current control store word is cleared and the disassembled string is then put into the terminal input buffer. It is then possible to modify the disassembled string by the Sintran III line editing features. At carriage return the modified string is assembled and written into the control store. By ORIN the next terminal input is assembled and a logical OR of the entered instruction and the old contents is stored into the current control store word.

8.11.6.5.2. Subcommands OCTAL and SYMBOLIC

By the subcommand OCTAL it is possible to have the control store displayed in octal format. The display is returned to the symbolic mode by typing the command SYMBOLIC.

8.11.6.5.3. Subcommands GROUP and WORD

By typing GROUP only one 16 bit word is displayed. On carriage return the next 16 bit word is displayed. Within GROUP modus it is possible to modify the displayed 16 bit word by typing the new octal value followed by a carriage return. By typing WORD the display of nine 16 bit words continues.

8.11.7. **LOOK-AT commands**

8.11.7.1. **LOOK-AT-RESIDENT-MEMORY**

LOOK-AT-RESIDENT-MEMORY <address>

<address> - the octal physical address to be inspected.

Equal to LOOK-AT-DATA except that physical memory is examined and modified.

The subcommands are described in chapter 8.6.5.

8.11.7.2. LOOK-AT-PHYSICAL-SEGMENT

LOOK-AT-PHYSICAL-SEGMENT <address>, <phys. segm no.>

<address> - the octal segment relative address to be inspected.

<phys. segm no> - the number of the physical segment to be inspected

Equal to LOOK-AT-PROGRAM or LOOK-AT-DATA, except that a physical segment is inspected and modified directly.

The subcommands are described in chapter 8.6.5. Some of the subcommands are not valid in LOOK-AT-PHYSICAL-SEGMENT.

8.11.7.3. LOOK-AT-HARDWARE

LOOK-AT-HARDWARE <hardware register name>

Display the contents of the specified internal ND-500 CPU register or ND-100/ND-500 interface register.

The <hardware register name> may be one of

INTERFACE
Display the interface registers

Carriage Return
Display the hardware registers (approx 80 registers)

A,XD
Display the registers starting with name A,XD

Register name
Display the specified register

MEMORY-MANAGEMENT-REGISTER
Display the 40 MMS registers.

8.11.8. Process management

8.11.8.1. ATTACH-PROCESS

ATTACH-PROCESS <process no>

<process no> - the number of the process with which communication is desired. Default is the current process connected to the terminal.

Subsequent commands LOOK-AT, RUN etc will be routed to the specified process. The process should not be connected to any other terminal.

This command is currently used for debugging purposes, attaching to the swapper process.

8.11.8.2. LOGOUT-PROCESS

LOGOUT-PROCESS <process>

<process> - the number of a currently running process.

The ND-500 process specified will be aborted and its reserved resources released. Also, the user will be forced to leave the ND-500-MONITOR.

This is the normal command to remove a user from the ND-500 system. A proper cleanup of the area used by the logged out process is done; it is therefore safer than ABORT-PROCESS. LOGOUT-PROCESS resembles the Sintran III command @STOP-TERMINAL for ND-100 processes.

8.11.8.3. ABORT-PROCESS

ABORT-PROCESS <process>

<process> - the number of a currently running process.

The process specified will be aborted and its reserved resources released. The user will be forced to leave the monitor.

This command should be used with care, as no cleanup of the system tables and queues is performed. It should be employed only in case of a system hangup, where there is no other way stop a process.

8.11.8.4. PROCESS-STATUS

PROCESS-STATUS

A summary of the status of all active processes is printed on the output device. The information includes for each active process the terminal number of the user having reserved the process, the user name, the status of the process (idle or active), and the amount of ND-500 CPU time used and login time since the Monitor was entered.

8.11.9. Inspecting system tables

8.11.9.1. LIST-TABLE

LIST-TABLE <table name>

<table name> - the name of one of the system tables.

This command has a number of subcommands used for searching through the system tables. Detailed system knowledge is required in order to utilize the information obtained through this command. The subcommands are:

PROC-TAB - List the table of active processes.

HW-SEGM-TAB - List the table of physical segments in use.

SW-SEGM-TAB - List the segment table used by software.

MEMORY-MAP - List the memory map.

LAST-N500-MSG - List the ringbuffer containing the last 64 messages to ND-500

N500-MSG - List the messages to ND-500 from a specified process.

FOLLOW-LINK - Follow the link to the next element in the table.

FOLLOW-TABLE - List the next element in the table.

<octal value>/ - List the specified entry in the current table.

cr - List the next element in the current table.

EXIT - Return to the command processor.

8.11.9.2. LIST-ACTIVE-SEGMENTS

LIST-ACTIVE-SEGMENTS <process no.>

<process no> - the number of an active process.

This command will list all the segments currently in use by a process, the correspondance between logical and physical segments and the name of the process.

The <process no.> parameter may also take the values OWN or -1, indicating the user's own process, ALL or -2 indicating all active processes.

8.11.9.3. LIST-SEGMENT-TABLE-ENTRY

LIST-SEGMENT-TABLE-ENTRY <segm. no>

<segm. no> - a physical segment number.

The information in the physical segment table will be printed oh the output device. This information includes the segment name and type, the owner process, the size of the segment, the segment attributes and allocation in the swap file, and the current use of the segment by the active processes.

<segm. no> equal to ALL or -1 indicates all segments.

8.11.9.4. LIST-PROCESS-TABLE-ENTRY

LIST-PROCESS-TABLE-ENTRY <process no.>

<process no> - the number of an active process.

The process description of the specified process is printed on the specified file. OWN or -1 indicates the user's own process, ALL or -2 indicates all active processes.

The returned information includes the process segment, the program and data capabilities.

8.11.10. Swap files

A segment may either be swapped out on its original file or a system selected swap file. This is determined by the attribute specified for the segment in the OPEN-SEGMENT command.

System selected swap files are contiguous files used as scratch area for modified pages of a segment. As long as no modifications are done, pages are read from the original segment file, but if a page is modified this page is copied to the swap file and used for further swapping. For each segment that may need a scratch area, a contiguous area is allocated. The segment may not be expanded during execution.

To define a file as a swap file for the ND-500 the file must be created with the Sintran III command @CREATE-FILE. Then the Monitor command DEFINE-SWAP-FILE must be used to inform the Monitor that this file should be used for swapping.

8.11.10.1. DEFINE-SWAP-FILE

DEFINE-SWAP-FILE <file name>

<file name> - the name of an existing contiguous file.

The file specified is defined as a swap file for ND-500 segments. The file must be a contiguous file, and must be created before this command is used. The file may belong to any user, but user SYSTEM must have at least read and write access (RW) to it.

There may be several swap files in the system; the Monitor will assign a swap area to a process on whichever file has sufficient free space left. Definition of swap files will survive a warm start, but not a cold start.

8.11.10.2. DELETE-SWAP-FILE

DELETE-SWAP-FILE <file name>

<file name> - the name of a file previously defined as an ND-500 swap file.

The specified file is de-allocated as an ND-500 swap file. The file is not deleted from the file directory, but will no longer be used by the Monitor as swap area for ND-500 segments.

8.11.10.3. LIST-SWAP-FILE-INFO

LIST-SWAP-FILE-INFO <swap file no.>

<swap file no> - the number of the swap file, starting at 0, or ALL.

Information about the swap file is printed on the output device. This information includes both file system statistics and the current usage of the file. If the parameter is given as ALL, information about all swap files defined is printed.

8.11.10.4. LOAD-SWAPPER

LOAD-SWAPPER <file name>

<file name> - name of binary file where the swapper is located. Default file name is (SYSTEM)SWAPPER.

The swapper process is loaded into ND-500 memory. Normally, this is done automatically when the first ND-500 process is initiated by the monitor, but this command may be useful to load a new copy if there are reasons to believe that the existing one is corrupted, or to load a non-standard version of the swapper process.

The file type may not be specified but will always be :PSEG and :DSEG. The swapper will always run as process number zero.

8.11.10.5. START-SWAPPER

START-SWAPPER

The swapper process, loaded into memory by the LOAD-SWAPPER command, is started.

8.11.11. SET-SYSTEM-PARAMETERS

SET-SYSTEM-PARAMETERS <no of phys. segm>,
<max pages fixed>

<no of phys segm> - the maximum number of physical segments in the ND-500. Maximum is 2000B.

<max pages fixed> - the maximum number of pages fixed for the system as a whole.

A modification of the number of physical segments will have no effect until the system is restarted. The number of segments include all physical segments including the process segments used internally by the monitor. Reducing the number of physical segments will reduce the space needed by system tables and release memory pages for swapping.

The maximum number of pages fixed for the system as a whole includes pages implicitly fixed by the Monitor before doing direct transfer I/O.

8.11.12. LIST-SYSTEM-PARAMETERS

LIST-SYSTEM-PARAMETERS

The values of all parameters specified by the SET-SYSTEM-PARAMETERS command are printed on the output device.

8.11.13. MASTER-CLEAR

MASTER-CLEAR

Brings the ND-500 out of any hang-up state by sending a hardware master clear signal to the ND-500 interface. This will cause the ND-500 to stop immediately and reset the interface. This is equivalent to pressing the MCL button on the ND-500 front panel.

This command is used before a complete restart of the ND-500, and the contents of registers are unpredictable.

9. SINTRAN-III MONITOR CALLS

Monitor calls are requests to the operating system for I/O services, system information and a number of special functions. Normally, a compiler will translate certain source statements, like the Fortran OPEN, into a monitor call, and thus hide the monitor call from the user. For the assembler programmer, and for the programmer requiring functions not offered by the compiler, direct access to the monitor calls may be neccessary.

Most of the monitor calls in Sintran III are available for ND-500 programs through the ND-500 Monitor. The arguments of the monitor calls are, with a few exceptions, the same as in Sintran III. For detailed information of the Sintran III monitor calls, see the Sintran III Reference Manual ND-60.128.

In the ND-500 a Sintran III monitor call is performed by a CALL or CALLG instruction. CALL and CALLG are general subroutine call instructions. A routine call where the five leftmost bits of the subroutine address are set - the segment number is 37B - is a Sintran III monitor call. That is, to the user program the Sintran III monitor call functions appear to be regular routines on a link segment. The 27 rightmost bits of the subroutine address are the monitor call number.

Single parameters to a Sintran III monitor call must always be a 32 bit word residing in the data memory. When Sintran III requires byte or halfword parameters, these are the rightmost byte or halfword of the 32 bit parameter. Observe that the address of a 32 bit word is addressed by its leftmost byte, thus, the address a one or two byte variable (e.g. BYTE, INTEGER1 and INTEGER2 in Planc) cannot be used directly in the argument list, but must be incremented by 3 or 2, respectively.

Arrays are addressed by the lowermost word in the array, and the array elements are always of size 32 bits.

String parameters - such as filenames etc. - must be a descriptor parameter (DESC). A descriptor is a two word element, the first containing the highest array index, starting at zero, the second the address of the element with index 0.

During the execution of monitor calls, errors may occur. If an error has occurred, the K flag is set; otherwise it is reset. If an error code is returned to the program, it may be found in the I1 register. If an error occurs in a monitor call not returning an error code, the I1 register will be set to -1. The K flag is also set.

Monitor calls returning a value will leave the value in the I1 register. Monitor calls not returning a value may destroy the I1 register, even if no error occurs or no error is possible. All other registers will at return contain the values they had before the monitor call.

When an error has occurred, an error message may then be written on the communication device by using the error code as argument in the monitor calls ERMSG (MON 64) or QERMS (MON 65). Error messages from monitor calls are always written to the communication device if the command AUTOMATIC-ERROR-MESSAGE is given.

The following is a list of the available monitor calls with their arguments. When the function of the monitor call is identical in both the ND-500 and the ND-100, the description of the monitor call is found in the Sintran III Reference Manual ND-60.128. If the function is not exactly the same, the differene is described here.

Mon.call no.	name	Comments
0B	LEAVE	All files opened by the ND-500 program or by the OPEN-FILE command in the Monitor will be closed.
1B	INBT	<unit> <byte>
2B	OUTBT	<unit> <byte>
3B	ECHOM	<unit> <echo mode no> <bitmap>
4B	BRKM	<unit> <break mode no> <bitmap> <max. no. of chars>
11B	TIME	The result will be in the I1 register.
12B	SETCM	<string descriptor>
13B	CIBUF	<unit>
14B	COBUF	<unit>
16B	MGTTY	<unit> <terminal type>
17B	MSTTY	<unit> <terminal type>
21B	M8INB	<unit> <no. of bytes read> <buffer descriptor>
22B	M8OUT	<unit> <buffer descriptor>
23B	B8INB	<unit> <no. of bytes read> <buffer descriptor>
24B	B8OUT	<unit> <buffer descriptor>
26B	LASTC	<unit> The result will be in the I1 register.
27B	RTDSC	<RT description> <buffer> Number of connected devices to the RT description will be returned in the I1 register.
30B	GETRT	The result will be in the I1 register.
31B	EXIOX	<in-value> <device no.> The result will be in the I1 register.
32B	MSG	<descriptor of the string>
35B	IOUT	<format> <value> <format>= 2: Binary format. <format>= 8: Octal format. <format>= 10: Decimal format. <format>= 16: Hexadecimal format.
36B	NOWT	<unit> <input/output> <flag>
37B	AIRDW	<no. of channels> <channel buffer> <data buffer> <error indicator>
40B	SPCLO	<unit> <text string descriptor> <no. of copies> <flag> If <unit>= -3 then all files opened by the ND-500 program will be closed. Other values of <unit> will give standard action.
41B	ROBJE	<unit> <buffer>
43B	CLOSE	<unit> If <unit>= -3 then all files opened by the ND-500 Monitor and the ND-500 program will be closed. Other values of <unit> will give standard action.
44B	RUSER	<descriptor of user name string> <buffer>
45B	GTYPR	<unit> <typring> <status> <Sintran III open file number> The TYPRING word in the datafield, a word containing flag bits and the open file number corresponding to the specified <unit> is returned
50B	OPEN	<unit> <access code> <file name string descriptor> <file type string descriptor> The parameter <unit> is the open-file number the program assigns to the specified file. If <unit> = 0 then the ND-500 Monitor will return an open-file number in the parameter <unit>.
52B	TERMO	<unit> <mode>

53B	RSEGM	<segment no> <buffer>
54B	MDLFI	<file name string descriptor>
55B	RSPQE	<unit> <buffer addr.>
56B	PASET	<buffer> Five 32 bit words may be set. These are independent of the status words for the corresponding ND-100 call
57B	PAGET	<buffer> Five 32 bit words set by PASET may be read.
60B	RWPM	<function> <ND-500 program address> <data> Read/write program memory. <function>=0 : read, <function>=1 : write. <data> is always 4 bytes Observe that MON 60 (N500M) executed from ND-100 has functions different from the RWPM call
62B	RMAX	<unit> <no. of bytes in file>
63B	B4INW	<unit> <buffer>
64B	ERMSG	<error code>
65B	QERMS	<error code>
66B	ISIZE	<unit> The result will be in the I1 register.
67B	OSIZE	<unit> The result will be in the I1 register.
70B	COMND	<descriptor of command string> Only a subset of Sintran III commands are legal
73B	SMAX	<unit> <max. byte pointer>
74B	SETBT	<unit> <byte pointer>
75B	REABT	<unit> <byte pointer>
76B	SETBS	<unit> <block size in bytes>
77B	SETBL	<unit> <block number>
100B	RT	<RT description> Available for users SYSTEM and RT only.
101B	SET	<RT description> <no. of time units> <time unit no.> Available for users SYSTEM and RT only.
102B	ABSET	<RT description> <second> <minute> <hour> Available for users SYSTEM and RT only.
103B	INTV	<RT description> <no. of time units> <time unit no.> Available for users SYSTEM and RT only.
104B	HOLD	<no. of time units> <time unit no.>
105B	ABORT	<RT description> Available for users SYSTEM and RT only.
106B	CONCT	<RT description> <unit> Available for users SYSTEM and RT only.
107B	DSCNT	<RT description> Available for users SYSTEM and RT only.
110B	PRIOR	<RT description> <priority> The old priority of <RT description> will be in the I1 register. Available for users SYSTEM and RT only.
111B	UPDAT	<min> <hour> <day> <month> <year> Available for users SYSTEM and RT only.
112B	CLADJ	<no. of time units> <time unit no.> Available for users SYSTEM and RT only.
113B	CLOCK	<buffer>
114B	TUSED	The result will be in the I1 register.
115B	FIX	<segment no.> Available for users SYSTEM and RT only.
116B	UNFIX	<segment no.> Available for users SYSTEM and RT only.

117B	RFILE	<unit> <flag> <buffer> <block no.> <no. of bytes to read> If <flag> ≠ 0 then the ND-500 program will be restarted immediately after the parameters are accepted (not waiting for the transfer to be terminated). The status of the transfer may later be checked by the monitor call WAITF.
120B	WFILE	<unit> <flag> <buffer> <block no.> <no. of bytes to write>. If <flag> ≠ 0 then the ND-500 program will be restarted immediately after the parameters are accepted (not waiting for the transfer to be terminated). The status of the transfer may be checked later on by the monitor call WAITF.
121B	WAITF	<unit> <flag> If the previous RFILE/WFILE/MAGTP on <unit> was called with <flag> ≠ 0, then the status of this transfer will be returned into the I1 register.
122B	RESRV	<unit> <input/output> <flag> Available for users SYSTEM and RT only.
123B	RELES	<unit> <input/output> Available for users SYSTEM and RT only.
124B	PRSRV	<unit> <input/output> <RT description> Available for users SYSTEM and RT only.
125B	PRLS	<unit> <input/output> Available for users SYSTEM and RT only.
126B	DSET	<RT description> <basic time unit> Available for users SYSTEM and RT only.
127B	DABST	<RT description> <basic time units> Available for users SYSTEM and RT only.
130B	DINTV	<RT description> <basic time units> Available for users SYSTEM and RT only.
131B	ABSTR	<unit> <function> <buffer> <block no.> <number of blocks> The <buffer> parameter must be specified as an ND-100 physical address. Available for users SYSTEM and RT only.
134B	RTEXT	Same effect as LEAVE
135B	RTWT	Available for users SYSTEM and RT only.
136B	RTON	<RT description> Available for users SYSTEM and RT only.
137B	RTOFF	<RT description> Available for users SYSTEM and RT only.
140B	WHDEV	<unit> <input/output> The result wil be in the I1 register.
141B	IOSET	<unit> <input/output> <RT description> <value> The result will be in the I1 register.
142B	ERMON	<error code> <sub. error number>
143B	RSIO	<mode> <input unit> <output unit> <user index>
144B	MAGTP	<function> <buffer addr.> <unit> <max. no. of bytes> <actual no. of bytes read> If 1000B is added to <function>, the ND-500 program will be restarted immediately after the parameters are accepted (no waiting for the monitor call to be terminated). The status of the execution may later be checked by the monitor call WAITF.

145B	ACM	<unit> <function> <buffer> <dma addr.> <no. of bytes>
147B	CAMAC	<value> <status> <crate number> <station number> <subaddress> <function> Available for users SYSTEM and RT only.
150B	GL	<value> <flag> Available for users SYSTEM and RT only.
151B	GRTDA	<descriptor of RT program name> Available for users SYSTEM and RT only.
152B	GRTNA	<RT descriptor> <buffer> Available for users SYSTEM and RT only.
153B	IOXN	<data> <IOX-code> Available for users SYSTEM and RT only.
154B	ASSIG	<log. unit number> <graded LAM number> <crate number> Available for users SYSTEM and RT only.
157B	ENTSG	<page table> <interrupt level> <start address> Available for users SYSTEM and RT only.
160B	FIXC	<physical start page> Available for users SYSTEM and RT only.
161B	INSTR	<unit> <string descriptor> <max. no. of bytes> <terminator> The function value will be in the I1 register.
162B	OUTST	<unit> <string descriptor> The function value will be in the I1 register.
164B	WSEG	<segment no> Available for users SYSTEM and RT only.
165B	DIW	<no. of registers> <buffer with logical units> <data buffer> <error indicatior> Available for users SYSTEM and RT only.
166B	DOLW	<no. of registers> <buffer with logical units> <data buffer> <error indicator> Available for users SYSTEM and RT only.
213B	MUIDI	<descriptor of user name string> <directory index> <user index>
214B	GUSNA	<descriptor of user name string> <directory index> <user index>
215B	DROBJ	<object entry buffer> <directory index> <user index> <object index>
216B	DWOBJ	<object entry buffer> <directory index> <user index> <object index>
217B	GUIOI	<unit> <directory index> <user index> <object index>
220B	DOPEN	<unit> <access code> <descriptor of file name string> <descriptor of file type string>
221B	CRALF	<descriptor of file name string> <start address of file> <number of pages in file>

400B MACROE Subsystem error return. Will set an error indicator that may be tested by the IF-ERROR-MACRO-STOP and IF-ERROR-FULL-STOP commands.

401B DISASS <program address>, <descriptor of returned string> <max no. of characters>
One ND-500 instruction starting at the specified <program address> is disassembled. The returned string will be truncated to <max no. of characters> if required. The actual number of characters in the returned string is returned in the length part of the <descriptor of returned string>.

402B RFLAG <value>
Reads a 32 bit flag array set by the SET-FLAG Monitor command or by a monitor call in a Sintran III program. See section 8.8.1.

403B WFLAG <value>
Writes a 32 bit flag array that may be read by a Sintran III program or by the GET-FLAG Monitor command. See section 8.8.1.

404B IOFIX <first addr> <size of area in bytes>
Specify to the Monitor the data area the program will use for file system I/O.

405B USTBRK <function> <address>
<function>=0 : enable, <function>=1 : disable user handling of the escape input. If enabled, control will be transferred to <address> if the user presses the escape or break key.

406B RWRTC <function> <RTCOMMON addr> <# bytes> <buffer>
<function>=0 : read, <function>=1: write RTCOMMON
The specified RTCOMMON address is an ND-100 virtual word address.

407B TPSTRA <p1>..<p7>
Return from N500M, function RUNN (12B), to ND-100. Stop reason is given the value 65, and the seven parameters are transferred to ND-100. Interpretation of the parameters is up to the ND-100 program issuing N500M.

410B FIX <type> <first addr> <length> <phys ND-100 addr>
<type>=0 : fix scattered
<type>=1 : fix contiguously, address returned
<type>=2 : fix contiguously at given address
<first addr> is the logical ND-500 address, <length> in bytes, but the physical address is specified/returned as an ND-100 word address

411B UNFIX <address>
Only the segment number of the address is significant. All fixed areas in the given segment are unfixed.

412B FSCNT <file no> <log. segment no> <type> <segment no>
Connect file as a segment. File must be opened through MON 50 or the OPEN-FILE command.
<log. segment no>=0 will select the first free segment and return in <segment no>.
<type>=0 : file contains initial data
<type>=1 : uninitialized, empty file
<type>=2 : primarily sequential access
<type>=3 : combination of 1 and 2
Specifying <type>=2 will reduce swapping, as long as access is sequential

413B FSDCNT <file no> <segment no>
The file is no longer accessed as a segment, but is not closed. All pages are, however, flushed to the file.
CLOSE will imply FSDCNT.

414B BCNAF <function> <address> <data> <status>
Special CAMAC monitor call.

415B BCNAF1 <function> <address> <data> <status>
Special CAMAC monitor call.

416B WSEGN <log segment no>
Write all modified pages back to segment file

417B MXPISG <log segment no> <segm. type> <# pages>
Set max pages in memory for a segment.
<segm. type>=0 : data segment
<segm. type>=1 : program segment

10. THE N500M MONITOR CALL

This chapter is intended as background information only, and is included for readers with a thorough knowledge of Sintran III. The N500M monitor call is primarily used by the Monitor itself, and will normally not be used by application programs. Programmers who want to use the functions listed below, are advised to consult Norsk Data for further details.

The ND-500 Monitor is divided into three separate parts that run in the ND-100. The first part runs on page table 2 as the subsystem called ND-500-MONITOR, and the second runs on page table 0. The third part is the ND-500 driver routine residing in the resident and the "paging off" area.

The subsystem ND-500-MONITOR communicates with the page table 0 part through a special monitor call, MON 60 (N500M). N500M has several functions, and accepts parameters according to the specified function. Observe that the monitor call number 60 is used from ND-500 for functions different from N500M executed on the ND-100.

The parameters to N500M are specified in same way as a "Fortran monitor call", (the A-register pointing to a list of parameter addresses). The first parameter is the function code which must be a 16 bit word. The rest of the parameters are either arrays or 32 bit words. Skip return indicates successful completion, direct return that an error occured.

The following is a list of the functions available in N500M:

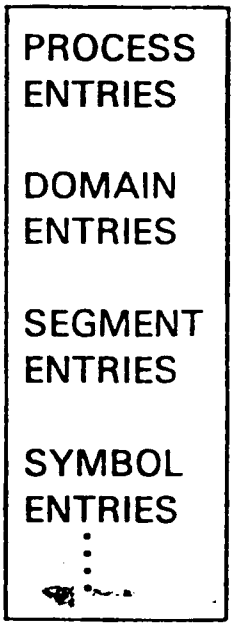
Function no.	name	explanation	parameters
0B	RREG	Read register	<reg.no> <value>
1B	WRWG	Write register	<reg.no> <value>
2B	RPROG	Read program memory	<no.of bytes> <ND-500 addr> <data area> <no.of bytes returned>
3B	RDATA	Read data memory	<no.of bytes> <ND-500 addr> <data area> <no.of bytes returned>
4B	WPROG	Write program memory	<no.of bytes> <ND-500 addr> <data area>
5B	WDATA	Write data memory	<no.of bytes> <ND-500 addr> <data area>
6B	PLACE	Place segment	<file name> <segment base> <size in bytes> <segment type>
7B	SWLOD	Load swapper	<swapper segment name>
10B	RREGB	Read ND-500 Registers	<register block>
11B	WREGB	Write ND-500 Registers	<register block>
12B	RUNN	Start program	<stop reason> <returned trap info> <clear time used>
13B	CNCFI	Connect file	<file name> <access code> <default type> <connect no.> <returned connect no.>
14B	CLSFI	Close file	<file no.>

15B	RESRV	Reserve ND-500-process	<start addr. after escape> <version string of PT0>
16B	RELIS	Release ND-500-process	
17B	LISOP	List open files	
20B	TIMUS	Time used	
21B	WHO	Who is on	
22B	ERRFL	Set error flag	<value>
23B	REACS	Read Control store	<CS addr.> <no of 16 bit words> <data-area>
24B	WRICS	Write Control store	<CS addr.> <no of 16 bit words> <data-area>
25B	MICST	Start micro program	<micro program start address>
26B		Data memory examine	<addr.> <value>
27B		Data memory deposit	<addr.> <value>
30B		Prog. memory examine	<addr.> <value>
31B		Prog. memory deposit	<addr.> <value>
32B	ABSMR	Absolute memory read	<no.of bytes> <ND-500 addr.> <data area> <no.of bytes returned>
33B	ABSMW	Absolute memory write	<no.of bytes> <ND-500 addr.> <data area>
34B	MSTOP	Stop micro program	
35B	MSTCL	Master clear	
37B		Load control store	<CS addr><no of words><file name>
40B	DEFM	Define memory config.	<start page> <no.of memory parts> <part array>
41B	RSTAT	Read comm. status	<status (bits 16:31: ND-500, bits 0:15: ND-100)> <MAR (memory address register)>
43B	SPRES	Reserve for spec. use	
44B	SPREL	Release after spec use	
46B	DEFSW	Define swap file	<file name>
47B	DELSW	Delete swap file	<file name>
50B	TESTF	Test function	<I1> <I2> <I3> <I4>
51B	RIFRG	Read interface reg	<register value/I4>
52B	G500P	Give ND-500 pages	<number of pages>
53B	T500P	Take ND-500 pages	<number of pages>
54B	STSWP	Start swapper	
55B	SPLAC	Start place	
56B	EPLAC	End place	
57B	MPVER	Microprogram version	<version number/I4>
60B	LIMEM	List memory config.	<array (regblk start/I2, ND-100 procadr/I4, ND-500 null/I2, memparts/I2(0:17B), accesstab/BY(0:17B)>
61B	RESER	Reserve N500 and N500 memory	<no. of pages> <first page no.>
62B	HIDEF	Define histogram	<start address> <interval size> <no. of intervals>
63B	HISTA	Start histogram	
64B	HISTP	Stop histogram	
65B	HISTR	Read histogram	<array>
66B	HIREL	Release histogram	
67B	SPRTE	Search for proc.entry	<process name> <record>
70B	GPRTE	Get process entry	<process> <record>
71B	SSGTE	Seach for phys.segm.	<name of phys.segment> <array>
72B	GSGTE	Get physical segment	<phys.segment no.> <array>

73B	RPHSG	Read physical segment	<phys.segment no.> <address> <no. of bytes> <array>
74B	SPRNM	Set process name	<process name>
75B		User SYSTEM test (skip	if SYSTEM)
76B	TOSWP	Send msg. to swapper	<record>
77B	RPROC	Read last message	<process no.> <record>
100B	RFLAG	Read process flag	<process no.> <flag>
101B	SFLAG	Set process flag	<process no.> <flag>
102B	GPSGE	Release ND-500 system	
103B	RSYSP	Read system param.	<parameter array>
104B	WSYSP	Write system param.	<parameter array>
105B	SPRIO	Set priority	<ND-100 monitor call priority> <max percent of ND-100 CPU time>
106B		Link to process	<process no.>
110B		Write physical segment	<segm no.> <ND-500 address> <no. of bytes> <data area>
111B		Start process log one	<process no.>
112B		Stop logging	
113B		Read log info	<data area>
114B		Release log facility	
115B		Start log all active processes	
117B		Abort process	<process no.>
120B		Set output device	<unit>
121B		Read from swapper process' data memory	<no. of bytes> <ND-500 address> <data area> <no. of bytes read>
122B		Logout process	<process no>
123B		Release all memory reserved by this process through function RESER (61B).	
124B		Start moncall log	
125B		Print moncall log	<array of 1K 16 bit words>
126B		Stop/release moncall log	
127B		Define standard domain	<array>
130B		Place standard domain	<name>
131B		Delete standard domain	<name>
132B		List standard domain	
133B		List execution queue	

11. DESCRIPTION FILE LAYOUT

This chapter will give an overview over the information stored in the description file. It is meant to be a general presentation, and does not pretend to give a complete description. The format of the information in the description file may be slightly modified in later versions of NLL and the Monitor, but the main structure is fixed.



The description file contains all necessary information about processes, domains and segments created by the user owning the description file. Each process, domain and segment has its own entry in the file. This means that each new segment opened, linked or indirectly linked will be assigned an entry in the description file. When a segment is deleted, the corresponding segment entry is removed, ie. linked out of the segment link of a domain.

The description file is an indexed file. It can therefore be expanded dynamically when new segments are created and segment entries reserved. The size of the process and domain entry area in the file is fixed, in order to speed the search for a new domain number.

In addition to the three major information blocks shown in the figure to the left are miscellaneous information, such as a list of the users auto-load files and the name of the Monitor. This information is stored in gaps between the major blocks.

Process entry - size: 1 byte

[ ]	Domain number of the first domain in the process (1 byte)

Domain entry - size: 56 bytes

[SEGLINK]	Link location for the first segment in the domain (4 bytes)
[DNAME]	Domain name (16 bytes)
[CHILDDOMAINS]	Numbers of the child domains of which this domain is the mother (6 bytes)
[MOTHER]	Domain number of mother domain (1 byte)
[CHILDINDEX]	First free loacation in the child domain area (1 byte)
[PROCPRIOR]	Priority (1 byte)
[FLAG]	Flag bits (1 byte)
[STADR]	Start address (4 bytes)
[ENABLEINT]	Bit mask indicating enabled traps (4 bytes)
[THA]	Trap handler address (4 bytes)
[SYSENABL]	Bit mask indicating system enabled traps (4 bytes)
[PBITMAP]	Bit map of used program segments (4 bytes)
[DBITMAP]	Bit map of used data segments (4 bytes)

Segment entry - size: 192 bytes

[SEGLINK]	Link to next segment in the domain (4 bytes)
[SNAME]	Segment name (directory:user)filename (54 bytes)
[SEGTYPE]	Flags indicating type of segment (4 bytes)
[COMSEGNO]	Number of shared Sintran III segments (2 bytes)
[COMSEGADDR]	Array containing logical address of all shared Sintran III segments within the data segment (10 bytes)
[COMSEGSIZE]	Array containing the size of all shared Sintran III segments (5 bytes)
[N100SEGNO]	Array of actual Sintran III segments (5 bytes)
[PLOG]	Logical number of this program segment (5 bits)
[DLOG]	Logical number of this data segment (5 bits)
[HDDINDEX]	No of link, indirect and common segments from other domains to this user (3 bits)
[PLB]	Logical low bound for program segment (4 bytes)
[PSIZE]	Size in bytes of program segment (4 bytes)
[DLB]	Logical low bound for data segment (4 bytes)
[DSIZE]	Size in bytes of data segment (4 bytes)
[DEBUGINFO]	Size of debug info on the :LINK file (4 bytes)
[DLINKDATE]	Last date written when segment was linked (4 bytes)
[ABSFIXAD]	Address if the segment should be fixed in absolute address in memory (2 bytes)
[LOWLOGFIX]	Lower page no. in fixed area (2 bytes)
[UPPLOGFIX]	Upper page no. in fixed area (2 bytes)
[MINPAGES]	Minimum number of pages in memory (2 bytes)
[MAXPAGES]	Maximum number of pages in memory (2 bytes)
[INDPLOG]	Logical program segment number in indirect domain (5 bits)
[INDDLOG]	Logical data segment number in indirect domain (5 bits)
[ADDSEGLINK]	Pointer to linked/common/indirect segment from other domains of the same user (4 bytes)
[INDDOMAIN]	Domain no of the indirect domain (1 byte)
[ADDTYPE]	Type of this segment (2 bits)
[ADDPSEGNO]	Logical program segment no within this domain (5 bits)
[ADDDSEGNO]	Logical data segment no within this domain (5 bits)
[INDPSEGNO]	Logical program segment no within indirect domain (5 bits)
[INDDSEGNO]	Logical data segment no within indirect domain (5 bits)
[LINKDATE]	Last date written when linking took place (4 bytes)

The fields from ADDSEGLINK to LINKDATE occur 5 times.

Symbol entry - size: variable

```
[ELINK      ] Link to next symbol in link (4 bytes)
[SL         ] Length of symbol (1 byte)
[NLE        ] Numeric length (3 bits)
[OPER       ] Operation type (+, -, *, /) upon this symbol
[IDENT      ] Language code (1 byte)
[CW         ] Type of symbol (see below) (1 byte)
[VAL        ] Value of symbol (4 bytes)
[SIZE       ] Size of common block (4 bytes)
[SS         ] Symbol name (max 255 bytes)
```

CW bits:

Bit no	Name	Explanation
0	UDEF	false = undefined element
1	DREF	false = program memory reference true = data memory reference
2	DSYM	false = program label true = data label
3	CLAB	true = common label
4	DMPF	true = symbol is written (used in list handling)
5	GLOB	true = the symbol will not be deleted when the loader table is saved
6	SELECT	true = module must be loaded
7	OMIT	true = module must not be loaded

12. THE ND RELOCATABLE FORMAT

12.1. DESCRIPTION

The ND Relocatable Format (NRF) is organized as a sequence of so-called NRF-groups where each group is composed of a control byte (5 + 3 bits) alone or followed by a varying number of trailing information bytes. The trailing information is either a numeric field, a symbolic field or both in a sequence;

<NRF-group>::=<control field> <numeric field> <symbolic field>

The control field (5 bits) contains an NRF control number in the range 0-37B. The control numbers denote a set of particular loader actions.

The numeric field (N) consists of a numeric length (NL - 3 bits) specification followed by zero to seven 8 bit bytes, as indicated by NL.

<Numeric field>::=<numeric length><byte>... etc.

Note that the numeric field is always present although the length may be zero (the control number + numeric length make up an 8 bit byte). A zero numeric length may in some cases be interpreted as an "all zeros" case of the numeric field.

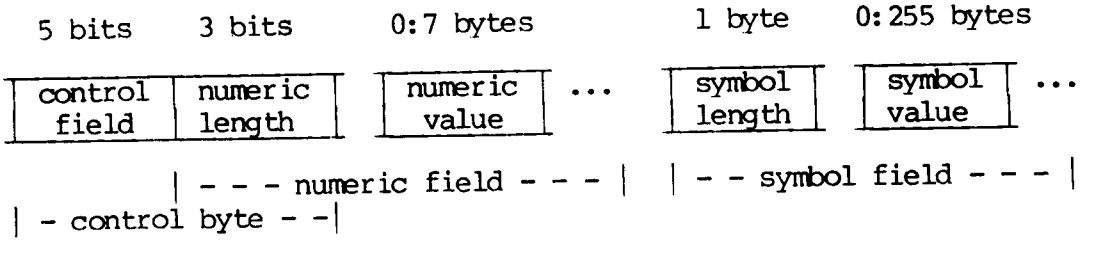
The numeric field is signed, with negative values in 2's complement form.

If a numeric field is present where it has no meaning, the number of bytes specified in the NL field are read and ignored.

The symbolic field (S) consists of a symbol length (8 bits) followed by 1-255 characters which constitute the symbol. Each character is represented in 8 bit ASCII code with the parity bit cleared. All characters are valid, including non-printing control characters. For two symbols to be equal, both the length and all characters must be equal.

<Symbol field>::=<symbol length><ch1>... etc.

The symbolic field is valid only in a subset of control codes.



In NLL there are two major byte pointers: the program byte pointer (PP) and the data byte pointer (DP). These byte pointers will normally point at the next "free" byte address in the program- and data-areas, and may be referenced during the loading session as #PCLC and #DCLC.

The size of an address is termed the address length, ADL, and is determined by the third byte in the information trailing the BEG control byte.

The byte pointer (BP) may be "coupled" to PP, DP or a "free" pointer (XP) by the control numbers PMO, DMO, and FMO. The "free" mode is useful when there is a need for modification of previously loaded information. For this mode neither PP nor DP are affected or changed. The "free" mode is reset by either PMO or DMO. Initially - after a BEG control number - the mode is PMO.

12.2. NRF control numbers

NUL 0 Ignored by NLL if numeric length is also 0. A non-zero numeric length is illegal.

BEG 1 A program system is composed of one or more modules. The BEG control number signifies the start of a module. Examples of modules are:

a) the outermost MODULE/ENDMODULE of Planc
b) PROGRAM/END, SUBROUTINE/END and FUNCTION/END of Fortran

The first byte of the trailing numeric field contains the real-time priority, the second contains the language code: ASSEMBLY=0, FORTRAN=1, PLANC=2, COBOL=3, PASCAL=4. The third byte contains the address length (ADL), default value is 4. Before an NRF module is loaded, the load address is adjusted upwards to a multiple of the address length. This applies to both the program byte pointer and the data byte pointer.

When a BEG control number is loaded, subsequent loading will be to the program segment until a DMO control number is loaded.

END 2 End of module. The trailing bytes' information contains the checksum in 2's complement form. The checksum is calculated by adding the binary byte values from BEG to END, trailing fields included, into a word, ignoring overflow. This sum is supplied in the END numeric field. The numeric length of the END control number specifies the size of the checksum in bytes (default 2 bytes - 16 bits). If numeric length is 0, no checksum test is performed.

MSA 3 Main Start Address. The current byte address is defined to be the main start address of the loaded module(s). If more than one MSA is loaded in the domain, a warning message is issued, and the first defined MSA applies.

LIB 4 The symbol in the symbol field is searched for in the loader table. If the symbol is present and not defined (i.e. only references exist) the rest of the module will be loaded. If the symbol is defined or not present the rest of the module is skipped.

When more than one LIB appears in a sequence, the module will be loaded when at least one of the symbols is undefined. For LIB, NL has no meaning.

DEF 5 Program Label Definition. Depending on NL this control number is interpreted as follows:

a) NL=0. The symbol in the symbol field will be entered into the loader-table with the current value of the program byte pointer (PP).

b) NL≠0. The symbol in the symbol field will be entered into the loader table with the value found in the numeric field with possible sign extensions if NL<ADL.

All previous references to the symbol will be defined.

REF 6 Program Reference. The symbol in the symbol field will be referenced in the address which corresponds to the current byte pointer in either program or data memory. When NL=0 the symbol value will occupy the next ADL bytes (one word). When NL≠0 the symbol value will occupy the NL next bytes.

BP will be incremented by NL (ADL if NL=0), to make room for later insertion of the symbol value. NL≠ADL or 0 must be used with care due to possible overflow bits which are lost (if the symbol value is greater than can be held in NL bytes).

When the symbol is defined, the sum of the numeric value in the REF group and the symbol value will be inserted in the NL bytes where the REF control number occurred.

LRF 7 Similar to REF if the symbol is already defined when the LRF control number is loaded. The value zero will be stored into the reference/byte(s) when either
a) the symbol is undefined (references only), or
b) the symbol is not present.

DDF 10 Data Label Definition. Similar to DEF but applies to data-memory and current data byte pointer (DP).

DRF 11 Data Label Reference. Similar to REF, but applies to data-labels.

RMV 12 Remove Symbol. The symbol in the symbol field is removed from the loader table. This directive is used to prevent the loader from overflowing, and by language processors to avoid name conflicts between local labels in different modules, used within the module only.

SLA 13 Set Load Address. The current byte pointer will be set to the contents of the numeric field. If the symbol length is non-zero, the symbol value will be added. The load-mode (program or data-mode) is unaltered

AJS 14 Adjust. The current byte-pointer will be adjusted with the (signed) number contained in the numeric field. If the symbol length is non-zero the symbol value will be added. The load mode (program or data-mode) is unaltered.

PMO 15 Set program mode. The program byte pointer (PP) will be set to the current value + the (signed) number in the numeric field.

DMO 16 Set data mode. The data-byte-pointer (DP) will be set to the current value + the (signed) number in the numeric field.

FMO 17 Set "free" mode. The current byte pointer + the (signed) number in the numeric field will be moved to the "free" byte pointer. Loading will be to the data or program segment determined by the current mode when the FMO control number is read. If the symbol field is filled ($S \neq 0$) the symbol value will be used instead of the current byte pointer. The program- and data pointers (PP and DP) are left unmodified and the loading may be resumed from PP or DP by using PMO or DMO.

REP 20 Repeat. The subsequent NRF-group will be repeated the number of times specified in the numeric field. If the next group is a compound group, the entire compound group will be repeated the specified number of times.

LDI 21 Load immediately. The NL trailing bytes will be stored into the NL next bytes according to the current byte pointer. The current byte pointer will be incremented by NL.

ADI 22 Add immediately. The value of the numeric field is added into the NL next bytes according to the current byte pointer. The current byte pointer will be incremented by NL.

APA 23 Add program address. The program byte pointer (PP) value + the number in the numeric field is stored into the next word (ADL bytes). The current byte pointer will be incremented by ADL.

ADA 24 Add data address. The data byte pointer (DP) value + the number in the numeric field is stored into the next word (ADL bytes). The current byte pointer will be incremented by ADL.

IHB 25 Execution inhibit. The NRF is incomplete due to errors during the compilation phase.

EOF 26 End of file. End of NRF file.

DBG 27 Debug. Start/stop of debug information. NLL will copy the information between two DBG control numbers to the :LINK file rather than to the :PSEG and :DSEG files. This information is used by the Symbolic Debugger.

LBB 30 Library module bytepointer. The library module in the symbolic field which begins in the byte address in the numeric field will be loaded if the symbol is present in the loader table, but undefined. This may increase the speed of library loading considerably.

MSG 31 Message. The ASCII string in the symbolic field is printed on the output device. The string is printed only if the MSG control number is actually loaded. A MSG control number in a library file not within an NRF module will not be printed unless it is located ahead of the address table in the file. If it is located within a module, it will be printed only if that module is actually loaded. The numeric field is ignored if present.

MIS 32 Miscellaneous. Sub control number in the numeric field.

CGR0 0 Start of compound group. Compound groups are used mainly in connection with the REP control number. Any sequence of control numbers may follow, up to the next MIS CGR1 control number. Compound groups may be nested to any level.

CGR1 1 End of compound group. If compound groups are nested, only the innermost nest is terminated; each level of nesting requires a matching CGR1.

ADD 2 The value of the next referenced symbol (REF, LFR or DRF control byte is added to the location pointed to by the current byte pointer. The size of the numerical values to be added is determined by the numerical length (NL) of the reference.

The current byte pointer should point to an already loaded value; usually "free mode" (FM0 control byte) will be effective, in order to set the byte pointer appropriately. The next referenced symbol must be defined prior to the reference (but need not immediately follow the ADD control byte), otherwise the ADD control byte has no effect. There is no distinction between REF, LRF and DRF references.

SUB 3 The value of the next referenced symbol will be subtracted from the location at the current load address. Otherwise, it acts as ADD.

MUL 4 The value in the current load address will be multiplied by the next referenced symbol. Otherwise, it acts as ADD.

DIV 5 The value in the current load address will be divided by the next referenced symbol. Otherwise, it acts as ADD.

LDN 33 Load immediately the number of bytes found in the numeric field.

> 33 Illegal control number.

12.3. Summary of NRF control numbers

ADL = address length
BP = current byte pointer
NL = numeric length
NV = numeric value

Control no	Trailing info	Comment
NUL=0		Ignored by NLL if NL=0, otherwise illegal
BEG=1	N	Start of module, priority, language, address alignment(ADL)
END=2	N	End of module,checksum
MSA=3	N	Main start address is at BP+NV
LIB=4	N,S	Conditional load. Load if S undef but referenced
DEF=5	N,S	Program label definition (BP=:(S) or NV=:(S))
DEF=6	N,S	Prog ref. BP+NL=:BP, if NL=0 then ADL=:NL
LRF=7	N,S	Reference if S defined, otherwise 0 (BP+ADL=:BP)
DDF=10	N,S	Data-label definition
DRF=11	N,S	Data-label reference
RMV=12	N,S	Remove symbol S
SLA=13	N,S	Set load address (NV=:BP), mode unaltered
AJS=14	N,S	Adjust byte-pointer (BP+NV=:BP), mode unaltered
PMO=15	N	Program mode (PP+NV=:BP)
DMO=16	N	Data mode (DP+NV=:BP)
FMO=17	N,S	Free mode ((S)+NV=:BP or BP+NV=:BP)
REP=20	N	Repeat next group NV times
LDI=21	N	Load immediately NV=:(BP:BP+NL), BP+NL=:BP
ADI=22	N	Add immediately (BP:BP+NL)+(NV)=:(BP:BP+NL), BP+NL=:BP
APA=23	N	Add program address. PP+NV=:(BP:BP+NL),BP+ADL=:BP)
ADA=24	N	Add data address. DP+NV=:(BP:BP+NL),BP+ADL=:BP)
IHB=25	N	Run inhibit (compiler errors)
EOF=26	N	End of NRF file
DBG=27	N	Start/stop debug information
LBB=30	N,S	Library module bytepointer. Module where S is defined starts at byte NV in NRF file.
MSG=31	N,S	S is printed on terminal during loading
MIS=32	N	Miscellaneous (sub control no in numeric field):
		GCR0=0 Start of compound group
		GCR1=1 End of compound group
		ADD =2 Add referenced value
		SUB =3 Subtract referenced value
		MUL =4 Multiply by referenced value
		DIV =5 Divide by referenced value
LDN=33	N	Load NV bytes immediately following

13. LINKAGE-LOADER ERROR MESSAGES

ALREADY DEFINED

A shared ND-100 segment or RTCOMMON was already defined.

AMBIGUOUS COMMAND

An abbreviated command has several possible matches.

ATTEMPT TO CREATE TOO MANY SCRATCH-SEGMENTS IN SCRATCH-DOMAIN

No more than 31 scratch segments are allowed in SCRATCH-DOMAIN.

ATTEMPT TO CREATE TOO MANY SEGMENTS IN THIS DOMAIN

A domain may contain no more than 31 segments.

CHECKSUM ERROR

Due to hardware or software errors the checksum supplied in the numeric field of the END group does not match the checksum calculated by NLL.

DATA AREA FULL

The load address specified in the HIGH-ADDRESS command has been reached in the data segment.

DOMAIN ALREADY EXISTS

A domain name specified in double quotes in the SET-DOMAIN command already exists in the description file of the current user. If any loading to (new or existing) segments in the existing domain should be done, repeat the command with the domain name unquoted. Otherwise specify a different name for the domain.

ERROR IN INITIALIZING THE DESCRIPTION FILE

This error may occur the first time NLL is used, and may indicate a hardware or file system error (e.g. lack of sufficient space). If no explanation for the error is found, please report to Norsk Data.

ERROR IN OPENING RTFIL

The Sintran III file system error message will indicate the reason why (SYSTEM)RTFIL:DATA cannot be opened during a LINK-RT-PROGRAM or MATCH-COMMON-RT-SEGMENT command. Appropriate action must be taken according to the file system error message.

FATAL ERROR

If this error occurs, please report to Norsk Data, preferably with a copy of the description file at the time of the error and an as complete as possible list of commands executed prior to the error. The contents of the description file may be invalid, and no further loading should be performed without rebuilding the description file. Be aware that this will destroy all information about previously loaded segments.

ILLEGAL ATTRIBUTE CODE

An attribute code unknown to NLL was encountered in an OPEN-SEGMENT or COMMON-SEGMENT-OPEN command.

ILLEGAL CHARACTER IN PARAMETER

The rules of the Sintran III file system apply to segment names, i.e. a name may consist of alphanumerics and hyphens. In general, the same rules apply to domain. Either a non-alphanumeric/hyphen character was encountered in a name, or a double quote indicating a new name was not matched by another. Where user names may be specified, mismatching parentheses may also be a source of this error.

ILLEGAL CONTROL BYTE

An NRF control number larger than 32B, or zero with a non-zero N field, was encountered in an NRF file.

ILLEGAL TRAP MNEMONIC

The trap name specified in either SYSTEM-TRAP-ENABLE, LOCAL-TRAP-ENABLE, SYSTEM-TRAP-DISABLE or LOCAL-TRAP-DISABLE was not one of the names in the table on page 84.

INSUFFICIENTLY COMPILED PROGRAM

An IHB control number (250B) was encountered in an NRF file, indicating that errors occurred during program compilation.

LOADER TABLE OVERFLOW

Too many labels have been defined. The segment must be reloaded, and before the loader table overflows, the entries that are no longer needed should be removed with the KILL-ENTRIES command. Alternatively, the loading may be terminated prematurely with CLOSE-SEGMENT (ignoring the error message - the command must be specified twice), and restarted with APPEND-SEGMENT.

NO MORE AUTO-LOAD-FILE BUFFER-SPACE AVAILABLE

A maximum of six auto-load files may be specified by each user.

NO MORE DOMAINS AVAILABLE FOR THIS USER

The description file can hold a maximum of 256 domains.

NO SUCH (AMBIGUOUS) DOMAIN ON THE SPECIFIED USER

Either the domain name is not registered in the description file, or more than one domain has a name that matches the specified one.

NO SUCH (AMBIGUOUS) SEGMENT IN THIS DOMAIN

Either the segment is not registered in the current domain, or more than one segment has a name that matches the specified one.

NO SUCH COMMAND

The command is not known to NLL. Check the list of available commands with the HELP command.

NO SUCH MODULE

A module identifier specified was not found in the specified file in one of the NRF editor commands DELETE-NRF-MODULE, LIST-NRF-CODE, FETCH-NRF-MODULES, WRITE-NRF-EOF-AFTER-MODULE or INSERT-NRF-MESSAGE.

NOT DELETE ACCESS

It is not legal to delete processes, domains or segments in other than the current user's description file. No prefixes (directory or user name) are allowed.

NOT IMPLEMENTED

A command available only in the multisegment version of the Linkage-Loader was attempted executed in the single segment version.

NOT IN THE LOADER TABLE

The label specified in the VALUE-ENTRY command has not been loaded or defined since NLL was entered, or it has been killed with the KILL-ENTRIES command, or it has been deleted from the loader table implicitly at an CLOSE-SEGMENT or END-DOMAIN.

NOT LINK ACCESS

The list of segments in the LINK-SEGMENT command includes one or more segments declared without link access. This means that the segment is either

- opened without shared program segment (P attribute)
- opened without shared data segment (D attribute)
- part of another domain but have linked segments, or
- not opened with random read access from this user.

NOT OCTAL NUMBER

NLL expected an octal number, and a number containing 8 or 9, or non-numeric characters, was entered.

PROGRAM AREA FULL

The load address specified in the HIGH-ADDRESS command has been reached in the program segment.

<file name> RESERVED BY ANOTHER USER

One or more of the files involved in a SET-DOMAIN command is currently being loaded or opened from another terminal or a batch process. To avoid inconsistencies NLL does not allow loading in a domain while it is being used by others.

ROUTINE VECTOR TOO SMALL

Insufficient space for routine labels was reserved by the ENTRY-ROUTINES command. The segment must be cleared and reloaded after executing ENTRY-ROUTINES with a higher <number of entries>.

SEGMENT ALREADY EXISTS

A segment name specified in double quotes in the OPEN-SEGMENT command already exists in the description file of the current user (segment names must be unique in the description file, even if they belong in different domains). If extensions to the existing domain should be made, use the APPEND-SEGMENT command. If the old contents should be replaced with new code, use the OPEN-SEGMENT command with the segment name unquoted. If the existing information should be kept unmodified, re-issue the command with a different (quoted) segment name.

SEGMENT NOT AVAILABLE OR AMBIGUOUS (OPEN-FILE FAILED)

The segment name specified in the OPEN-SEGMENT, APPEND-SEGMENT, LINK-SEGMENT, COMMON-SEGMENT-OPEN or COMMON-SEGMENT-APPEND command either does not exist or its name was abbreviated too much, or that for some other reason NLL did not succeed in opening one or more of the files affected by the last command.

*** SPECIAL USER BREAK ***

The 'escape' key was depressed on the terminal, interrupting any ongoing activity in NLL.

THIS COMMAND SHOULD BE DONE BEFORE LINKING

After a segment has been linked to other segments, the APPEND-SEGMENT may not be executed.

THIS COMMAND SHOULD BE DONE BEFORE LOADING

The ENTRY-ROUTINES command may not be performed after code has been loaded, as this code may be overwritten if a routine vector is built.

UNDEFINED SYMBOL

A parameter to a command was specified by a symbolic value unknown to NLL. Use LIST-ENTRIES-DEFINED to check which symbols are defined.

WARNING: LAST COMMAND WAS NOT EXECUTED

This message occurs after another error message, referring to an illegal operation.

WARNING: MODULE INDEX-TABLE IS NOT CORRECT

The address table of an NRF library file has been invalidated by an NRF editor command. The address table can be rebuilt by a PREPARE-NRF-LIBRARY-FILE command. If the address table is invalidated, the file will be loaded, but a sequential scan of the entire file is required.

SEGMENT HAS ALREADY LINKED SEGMENTS

To correct this situation, the segment which is trying to be linked must be loaded again without any linked segments. If the linking is done from another user, the segment must be deleted (DELETE-SEGMENT) before a new linking is tried.

LOGICAL SEGMENT NUMBER ALREADY USED

If it is absolutely necessary that this segment has the specified segment number (as in LINK-SEGMENT to a global library segment), the segment which has this number should be deleted and loaded again with another segment number (SET-SEGMENT-NUMBER).

14. ND-500 MONITOR ERROR MESSAGES

ADDRESS OUTSIDE FILE LIMITS IN DIRECT TRANSFER

This error message is returned from file access monitor calls to files opened in mode 8 or 9 (direct transfer), error code 1010B. The entire area to be transferred must be within the file.

ADDRESS OUTSIDE PROGRAM SEGMENT

ADDRESS OUTSIDE DATA SEGMENT

These errors are usually the result of an error in a user program, causing an address to exceed the size of the program or data segment. Usually, the address referenced can be found by using the LOOK-AT commands after the program has error terminated, inspecting the program instructions immediately preceeding the location pointed to by the P register. Common causes of this error are indexing errors or careless use of equivalenced variables.

ALWAYS SYSTEM ENABLED

An attempt was made to modify a fatal trap condition by the LOCAL-TRAP-ENABLE command.

AMBIGUOUS COMMAND

An abbreviated command has several possible matches.

AMBIGUOUS PARAMETER

A parameter has several possible matches. Reissue the command with HELP as "parameter" in order to have the list of legal parameters printed on the output device.

BIT NOT MODIFIABLE

A trap condition specified in a LOCAL-TRAP-ENABLE command may not be local enabled or disabled. This applies to fatal and non-ignorable trap conditions.

BLOCK ADDRESS NOT MODULO SECTOR SIZE IN DIRECT TRANSFER

This error message is returned from monitor calls to files opened in mode 8, 10 or 11 (direct transfer), error code 1011B. The limitations in accessing files opened in this mode are discussed in chapter 8.4.

BOTTOM OF STACK

While in the LOOK-AT-STACK command, the stack area displayed was that of the main program when the PREVIOUS subcommand was executed.

BREAK AT

Debug message, indicating that a user defined breakpoint was encountered. The Monitor halts and awaits further commands.

BUFFER FULL

One of the internal buffers used in expanding the macro overflowed. Simplifying the macro or using less extensive parameter substitution may help avoid the problem.

BYTECOUNT NOT MODULO SECTOR SIZE IN DIRECT TRANSFER

This error is returned from monitor calls to files opened in mode 8 or 9 (direct transfer), error code 1007B. The limitations in file addressing with direct transfer files are discussed in chapter 8.4..

BYTE POINTER NOT MODULO SECTOR SIZE IN DIRECT TRANSFER

This error is returned from monitor calls to files opened in mode 8 or 9 (direct transfer), error code 1016B. If the byte pointer is explicitly modified (through the SETBT monitor call, MON 74) on files opened with direct transfer, care should be taken that the limitiations discussed in chapter 8.4. are respected.

CONTROL STORE NOT SUCCESSFULLY LOADED

After a LOAD-CONTROL-STORE command the first 100 words of the micro-program store were compared to the file from which it was loaded, and an unequality found.

CURRENT MACRO ABORTED

A program signalled an error exit through monitor call MACROE (MON 400), or a trap condition not handled locally occurred. The current macro is aborted, but the macro calling the current one is continued, as if the IF-ERROR-MACRO-STOP command has been executed.

DC ACCESS NOT LEGAL ON MAG.TAPE

This error is returned from the OPEN monitor call (MON 50) when attempting to open a file with open mode 9 (direct transfer, file closed), error code 1014B. Magnetic tape files may be accessed in open mode 8 (direct transfer), but the file must be open during the transfer.

DEFINE-MEMORY-CONFIG. COMMAND IS REQUIRED

The Monitor needs information about the physical memory before any operation in the ND-500 is attempted. This information is lost after a)HENT system restart.

DEPOSIT NOT PERMITTED

The PERMIT-DEPOSIT command must be executed prior to a modification of memory or a register.

DIFFERENT PROGRAM TRACE FOUND AT nnnnn

Will be issued during execution of a BRANCH-TRACE or CALL-TRACE command with <mode> equal to COMPARE, when there are differences between the previously dumped trace and the current run.

ERROR IN LINKING TO RTCOMMON

This message is issued after a PLACE-DOMAIN or RECOVER-DOMAIN together with a second error message specifying what kind of error was discovered. Usually, the reason for the error is a modification of RTCOMMON from the time the domain was loaded to execution time.

ERROR IN MACRO

A syntax error was discovered in a submitted macro.

ERROR IN MEMORY CONFIGURATION

The Monitor has detected discrepancies between the memory configuration specified and the location of physical memory accessible to ND-500. Reissue the DEFINE-MEMORY-CONFIGURATION command with the correct parameter.

ERROR IN MONITOR CALL

This error message is returned after certain monitor calls, error code 1003B, and indicates an unclassified error from the Sintran III operating system. The cause may be either internal errors in the monitor call routine or errors in the parameters that could not be

classified otherwise.

FATAL ERROR FROM MICROPROGRAM

This indicates an internal error in the ND-500 microprogram that should be reported to Norsk Data.

FATAL PIT-0 ERROR. ERROR CODE: nnnn

This indicates an error in the operating system that should be reported to Norsk Data with as many details as possible about the system status at the time of the error.

FATAL ERROR FROM SWAPPER

This is an internal error in the swapper process, and the error should be reported to Norsk Data.

FILE IS NEITHER CONTIGUOUS NOR MAG. TAPE.

If a file is opened with direct transfer, open mode 8 or 9, the file must be either contiguous or a magnetic tape file. This error is returned as error code 1001B from the monitor call OPEN (MON 50). The file can be read if another open mode is selected.

FIXED SEGMENT HAS NO PAGES IN MEMORY

This is an internal error that should be reported to Norsk Data. The segment number specified in the message refers to a ND-100 segment number.

HARDWARE STATUS ERROR IN DIRECT TRANSFER

This error message is returned from monitor calls using direct transfer, error code 1012B, indicating that the transfer did not complete successfully.

HISTOGRAM ALREADY IN USE

Only one user at a time may use the histogram facility for performance measurement. Until the user currently using the histogram buffer executes a RELEASE-HISTOGRAM, no other user may execute a SET-HISTOGRAM.

HISTOGRAM NOT USED BY YOU

The PRINT-HISTOGRAM, START-HISTOGRAM and STOP-HISTOGRAM commands may be used only by the user who has reserved the histogram buffer.

ILLEGAL ADDRESS

The address specified in a LOOK-AT-command was outside the program or data segment.

The error may also be returned from the N500M monitor call (MON 60), indicating errors in the parameters. If no explanation for the error is found, please report to Norsk Data.

ILLEGAL CHARACTER

The rules of the Sintran III file system apply to segment names, i.e. a name may consist of alphanumerics and hyphens. In general, the same rule applies to domain and process names. Either a non-alphanumeric/hyphen character was encountered in a name, or a double quote indicating a new name was not matched by another. Where user names may be specified, mismatching parentheses may also be a source of this error.

ILLEGAL FILE NUMBER IN LOAD

Error message after executing the N500M monitor call (MON 60). If the application program uses N500M, the parameters should be checked. Otherwise, this indicates an internal error that should be reported to Norsk Data.

ILLEGAL FORMAT

8 or 9 was encountered in an octal number, or non-numeric characters in an octal, decimal or hexadecimal number (for hexadecimal, A:F are legal).

ILLEGAL FUNCTION CODE IN MON 60

Check the list of valid function codes in chapter 10 against the location pointed to by the ND-100 A register. If this monitor call was not used by the application program, the error message indicates an error in the Monitor that should be reported to Norsk Data.

ILLEGAL FUNCTION IN MON 61

The MON 61 call (FIXC5) is not normally used by application programs, and the error message indicates an error in the Monitor that should be reported to Norsk Data. The available functions will at a future time be documented in the Sintran III Reference Manual ND-60.128 for programmers who want to use this call.

ILLEGAL LOGICAL SEGMENT TYPE

Normally this indicates an internal error that should be reported to Norsk Data. If the user calls N500M (MON 60), the parameters may be erroneous.

ILLEGAL MICRO FUNCTION

This is an internal error message from the operating system or the ND-500 driver, that should be reported to Norsk Data.

ILLEGAL MONTIOR CALL NUMBER

This error message is returned from monitor calls from the ND-500, error code 1013B, and compares to the similar error message in ND-100.

ILLEGAL REGISTER NUMBER

The number specified in the N500M monitor call (MON 60) indicating a register, was outside the range allowed. This error should normally not occur if the application program does not use N500M, and should be reported to Norsk Data. If the application program uses N500M the arguments should be checked.

ILLEGAL PARAMETER

Many commands, e.g. MAIN-FORMAT, take only a limited set of different parameters. This error message indicates that a parameter not in the valid set was specified. Reissuing the command and giving the "parameter" HELP will cause the list of valid parameters to be listed on the output device.

ILLEGAL STATUS IN MESSAGE TO ND-500

This is an internal error message from the ND-500 driver that should be reported to Norsk Data.

ILLEGAL STOP REASON

This indicates an internal error in the ND-500 microprogram. Please report to Norsk Data.

INDIRECT NOT POSSIBLE

The "/" (indirect) subcommand in the LOOK-AT-commands is valid only when main format is word, as all addresseses are words and no reasonable interpretation can be applied to byte or halfword addresses.

INSUFFICIENTLY LOADED DOMAIN

A RECOVER-DOMAIN or PLACE-DOMAIN command was executed on a domain which still has undefined references or is not linked to the appropriate link segments.

LAST BREAK NOT FOUND

The last execution halt was not due to a breakpoint set by the BREAK command.

LL AND HL CHANGED

This message is a warning only, indicating that previous use of the LL and HL registers for debugging purposes will no longer have any effect. It is issued when another command using the LL and HL registers is given (TRACE and GUARD commands).

LOGGING FACILITY ALREADY RESERVED

Another user has already reserved the logg buffer for either histogram-commands or log-commands. The user having the buffer reserved must execute the RELEASE-LOG-BUFFER or RELEASE-HISTOGRAM command or interrupt the SWAPPING-LOG by pressing the escape key before another user can use the log commands.

LOGGING FACILITY NOT RESERVED BY YOU

The PRINT-PROCESS-LOG and RELEASE-LOG-BUFFER commands are valid only after the log buffer has been reserved for te current user through the START-PROCESS-LOG-ALL or the START-PROCESS-LOG-ONE command.

MACRO STACK ERROR

Macro calls were too deeply nested or recursive. The problem will usually be avoided by breaking down a complicated call sequence into a simpler one, with fewer levels. If the error persists, contact Norsk Data and keep a listing of all macros in use when the error occurred.

MACRO(S) ABORTED

A program signalled an error exit through monitor call MACROE (MON 400), or a trap condition not handled locally occurred. The current macro including the macro(s) calling the current one, is aborted, as if the IF-ERROR-FULL-STOP command has been executed.

MEMORY NOT AVAILABLE FOR ND-500 SEGMENT

If an explicit request for memory allocation was issued, the request could not be satisfied. This occurs when segments are shared with ND-100 or RTCOMMON.

ND-500 DMA ERROR

A hardware error has occured in the DMA transfer to or from the ND-500. Consult the system supervisor; or if the error persists, call Norsk Data.

ND-500 INTERFACE ERROR

A hardware error has occured in the ND-100/ND-500 interface. Consult the system supervisor; or if the error persists, call Norsk Data.

ND-500 OPEN FILE TABLE FOR DIRECT TRANSFER IS FULL

This error message is returned from the OPEN monitor call (MON 50), error code 1002B. The message indicates that the number of files opened with open mode 8 or 9, direct transfer, exceeds the maximum determined at system generation time. If there is room in the open file table for files opened with other access modi, one or more of the direct transfer files may be accessed in other modi, otherwise the number of concurrently opened files must be reduced.

ND-500 OPEN FILE TABLE IS FULL

This error message is returned from the OPEN monitor call (MON 50), error code 1000B. The message indicates that the total number of files open exceeds the allowed maximum. The limit is a system generation parameter, and sets a limit on the number of files opened with DC access (open mode 9). For files opened with other access modes, the Sintran III limitation of 16 files still applies.

ND-500 POWER FAIL

A power failure response was received from the ND-500 interface, indicating that power has not been turned on to ND-500, or some serious hardware problem has occurred.

ND-500 RESERVED FOR SPECIAL USE

The SET-ND-500-UNAVAILABLE command has been issued. Other users will have to wait until the SET-ND-500-AVAILABLE command is issued.

ND-500 STOPPED

The microprogram in the ND-500 is not running, and must be restarted by the system supervisor before any programs can be run.

ND-500 TIMEOUT

The ND-500 computer does not respond to requests from ND-100. Consult the system supervisor; if the error persists, call Norsk Data.

NO FREE PHYSICAL SEGMENT

The pool of physical segments is empty, and the job has to be run when the load on the system is lower. The number of physical segments is a system generation parameter.

NO FREE SWAP FILE ENTRY

The Monitor has run out of table space for the swap file entries, and the job must be rerun when the load on the system is lower. The table space available is a system generation parameter.

NO MEMORY AVAILABLE FOR ND-500 BUFFERS

There is no memory available for ND-500 buffers in memory bank 0-3 (each process needs 1K 16 bit words). Consult the system supervisor.

NO MORE BUFFER AREA

The number of macros, breakpoints and other debug commands requiring information to be kept in memory is exceedingly large. The most common cause for this error message is a "wild" recursive macro call generating temporary macros or breakpoints. If this is not the case, some macros must be deleted by ERASE-MACRO or breaks reset by RESET-BREAKS.

NO ND-500 PROCESS AVAILABLE

When an ND-500 job is submitted, a free process is allocated from a pool of available processes. This error message indicates that the pool is empty, and the job cannot start until a process is freed. The number of processes is a system generation parameter, and may be less than the number of terminals in the system, which means that not all terminals can run ND-500 jobs at the same time.

NO PAGE AVAILABLE FOR THE CONTEXT BLOCK

It is impossible to allocate memory to the ND-500 context blocks and segment table. This is caused either by an error in the memory configuration or because no free memory for ND-500. This error occurs only when the ND-500 is initially started or after a reconfiguring of memory.

NO RTCOMMON DEFINED

References have been made to the RTCOMMON area, which is non-existent on the machine. NLL will not allow references to RTCOMMON if it not defined, but a modification of the size or removal of RTCOMMON between the time of loading and execution time will cause errors to occur. Domains with references to RTCOMMON should under no circumstances be moved to another machine.

NO SUCH COMMAND

The command is not known to the ND-500 Monitor. Check the list of available commands with the HELP command.

NO SUCH COMMAND OR DOMAIN

The command specified is not known to the ND-500 Monitor, and is not found as a domain name in the description file of the current user.

NO SUCH DOMAIN

A domain name specified in a RECOVER-DOMAIN or DUMP-DOMAIN command is unknown in the description file of the current user.

NO SUCH MACRO

The macro name specified in the EXECUTE-MACRO command is not found in the list of temporary macros or as a permanent :MACR file.

NO SUCH SEGMENT

An unknown segment name was specified as a parameter. If there is any doubt with regard to which segments are available, use the NLL command LIST-SEGMENTS, or use the Sintran III command @LIST-FILES. (Be aware, however, that a file is not necessarily a segment file even though its type is :PSEG or :DSEG!)

NO SWAP FILE PART AVAILABLE

The Monitor has run out of table space for swap file parts. The job will have to be rerun after the load on the swap file has decreased.

NO WELL DEFINED PROGRAM IN MEMORY

A RUN, CONTINUE or GO command was specified before any PLACE, DEBUG-PLACE or RECOVER-DOMAIN command was executed.

NOT EXISTING BREAKPOINT

The breakpoint number specified in RESET-BREAK is unknown to the system.

NOT IMPLEMENTED

An attempt was made to use a feature that is not yet available in the monitor but will be implemented at a later stage.

NOT IN SEGMENT MODE

It is not possible to switch to another LOOK-AT-command from the LOOK-AT-PHYSICAL-SEGMENT command.

NOT REQUIRED ACCESS TO SEGMENT

One of the segments in the domain that was started or placed in memory does not have the required file system access rights. The default access will permit other users to execute the code on a program segment, but not to modify it. If the data segment is swapped from the original file, the file access of the data segment must also be set to RW (read and write) for other users to execute the domain(s) containing the segment. The access is modified through the Sintran III @SET-FILE-ACCESS command.

NUMERIC INPUT NOT ALLOWED IN DISASSEMBLE MODE

When in a LOOK-AT-command in disassemble mode, numeric deposit cannot be done. Change to a numeric MAIN-FORMAT in order to patch numerically.

ODD BYTE ADDRESS

This error is returned from file access monitor calls, error code 1004B.

ODD BYTECOUNT

This error is returned from file access monitor calls, error code 1005B.

OTHER USERS ALREADY LOGGED ON N500

Some of the system supervisor commands require exclusive access to ND-500. The SET-ND-500-UNAVAILABLE command will not force a logout of the users already logged on; these must be logged out explicitly before the system supervisor commands are used.

POWER FAIL DETECTED IN LOADING CS

A power failure occured during the loading of the microprogram to the control store. The loading of the control store must be restarted from the beginning.

POWER OFF

No response from ND-500. If power to the ND-500 is turned on, a hardware error has occured and service should be called for.

POWER UP AFTER POWER FAIL

This is an informative message to explain possible delays while the control store is being loaded and the Monitor initialized.

RTCOMMON NOT CONTIGUOUS

The RTCOMMON area may not be fractioned when shared with an ND-500 segment. This is usually detected at load time, and if the error occurs at run time it indicates modification of RTCOMMON after the affected segment has been loaded.

RTCOMMON SIZE DOES NOT MATCH THE ACTUAL RTCOMMON SIZE

This occurs after a PLACE-DOMAIN or RECOVER-DOMAIN (implicit or explicit), indicating that modification of RTCOMMON has been made after the linking of the domain took place. The affected segments must be reloaded and the domain relinked.

RTCOMMON SPECIFIED IN DOMAIN

This occurs after a PLACE-DOMAIN or RECOVER-DOMAIN (implicit or explicit). In general, any modification of the size or redefintion of RTCOMMON invalidates previously loaded domains using RTCOMMON. If an attempt is made in NLL to load segments referring to RTCOMMON is a system where there is none, an error message is issued at load time.

RTCOMMON's PHYSICAL ADDRESS DOES NOT MATCH THE PHYSICAL ADDRESS OF THE DOMAIN

This error message is issued at PLACE-DOMAIN or RECOVER-DOMAIN, and indicates that RTCOMMON has been modified since the domain using it was loaded. The segments containing RTCOMMON references must be reloaded and the domain relinked before the domain can be executed.

SEGMENT FIXED BUT NOT CONTIGUOUSLY

Segments shared between the ND-100 and the ND-500 must be fixed contiguously in memory. The number indicated in the message refers to the ND-100 segment number.

SEGMENT FIXED IN WRONG PHYSICAL ADDRESS

If an ND-500 segment is shared with more than one ND-100 segment or RTCOMMON, the physical address of the ND-100 segments cannot be modified after loading of the domain.

SHARED SEGMENT OUTSIDE ND-500 MEMORY

The segment shared between ND-100 and ND-500 is placed in private ND-100 memory located below ND-500 address zero. The segment must be released and fixed in an address accessible to ND-500.

SHARED SEGMENT DOES NOT OVERLAP ND-500 SEGMENT

Modification of the ND-100 segment or explicit setting of load addresses may cause parts of the ND-100 segment to be located beyond the limits of the ND-500 segment. The ND-500 segment must be reloaded.

SHARED SEGMENT FIXED, BUT NOT CONTIGUOUSLY

A segment shared between ND-100 and ND-500 has been fixed scattered in memory. The segment must be unfixed and fixed in a contiguous area before the ND-500 process will run.

SEGMENT NOT MODIFIABLE

An attempt was made to modify a segment declared as a read-only segment. Default segment attributes will make the program segment read-only, while pages in the data segment will if they are modified be copied to a swap file. This may be modified by using non-default segment attributes.

SWAP FILE ALREADY DEFINED

The <file name> in the DEFINE-SWAP-FILE command is already defined as an ND-500 swap file.

SWAP FILE IS IN USE

The DELETE-SWAP-FILE command may not be executed while an ND-500 process has its swap area allocated in the specified <file name>.

SWAP FILE IS NOT CONTIGUOUS MASS STORAGE FILE

The <file name> in the DEFINE-SWAP-FILE command is indexed, or it is not a mass storage (disk) file.

SWAP FILE NOT FOUND

The <file name> specified in the DELETE-SWAP-FILE command is not an ND-500 swap file, or the file name in the DEFINE-SWAP-FILE is unknown under.

SWAPPING SPACE NOT AVAILABLE

A large enough continuous are for the segments requiring swap file space was not available. The job must be rerun after other jobs have released enough space to fit in the rejected segment(s).

TOO BIG BYTECOUNT

This error message is returned from file access monitor calls, error code 1006B, indicating that the specified byte count is larger than can be represented in 17 bits. This is a limitation in the ND-100 file access monitor calls, where the byte count is in number of 16 bit words, represented in a single (16 bit) integer.

TOO BIG DATA SEGMENT

The sum of the start address and the length of the data segment gives an address above 777777777B (27 bits address space).

TOO BIG HISTOGRAM INTERVAL

The highest histogram interval allowed is 32767 bytes. Use a higher <number of channels> (if less than maximum) or a smaller range from <start address> to <max address>.

TOO BIG PROGRAM SEGMENT

The sum of the start address and the length of the program segment gives an address above 777777777B (27 bits address space).

TOO BIG VALUE

A numeric constant exceeding the legal range for the data type in question (e.g. a byte value >255) was entered. If it is desirable to enter the larger value, the main format should be changed to halfword or word as appropriate.

TOO MANY SHARED AREAS

The Monitor has run out of table space to store information about segments shared between ND-100 and ND-500. The job will have to be rerun at a time when the load on the system is lower. The size of the tables is a system generation parameter.

TOP OF STACK

While in the LOOK-AT-STACK command, the stack area was that of the currently executing procedure when the NEXT subcommand was executed.

TRYING TO LINK TO A DEMAND SEGMENT

This occurs after PLACE-DOMAIN or RECOVER-DOMAIN. A demand segment may not be shared between ND-100 and ND-500. Normally, this is discovered at load time by NLL, but if the error occurs at run time it indicates that modifications have later been done to the ND-100 segment.

TRYING TO LINK TO A NON-EXISTING SINTRAN III SEGMENT

This occurs when ND-500 shares a segment with ND-100 and the segment has been cleared in the ND-100 SEGFIL after the loading took place. The segment must be rebuilt and the ND-500 domain reloaded/relinked.

UNKNOWN BREAK AT nnnnn

A break instruction was encountered in the program segment. Breaks used for debugging purposes must be under full control by the Monitor; i.e. they should be inserted by the BREAK or TEMPORARY-BREAK commands.

UNKNOWN TRAP

This indicates an error in the ND-500 microprogram. Please report to Norsk Data.

WRONG NUMBER OF PARAMETERS IN MONITOR CALL

This error is returned from ND-500 monitor calls, error code 1015, and indicates that either excessive or insufficient parameters were transmitted.

15. EXAMPLES OF LINKAGE-LOADER AND MONITOR USAGE

The examples shown in this chapter are relatively small and incomplete as problem solutions. The intention is to give the beginner a certain familiarity with ND-500 operation and give a general impression of the user/Monitor interface.

In the examples, abbreviations of commands are used to some degree, to show how a more experienced user will write the commands. In some cases, all parameters are supplied in the command line, in other cases NLL or the Monitor prompts for them after CR is pressed. Remember that some parameters will not be prompted for if not supplied, rather, a default value is used. User input is always underlined.

NLL is available both as an ND-500 program and as an ND-100 program. In most examples, the ND-500 version is used, but the user interface is exactly the same for the ND-100 version.

15.1. Executing an ND-500 domain

Most compilers and the loader will execute on the ND-500 and must be started through the Monitor. This can be done in two ways:

Either, the domain name may be given as a parameter to the Monitor at the time of the call. To start the compiler FORTRAN, executing in ND-500:

```
@ND-500-MONITOR FORTRAN
$  <Fortran compiler commands>
$EX
@
```

Or the monitor may be started first, after which the domain is started by typing its name:

```
@ND-500-MONITOR
N500: FORTRAN
$ <Fortran compiler commands>
$EX
N500: EX

@
```

The two methods are essentially equivalent, but if the domain name was a parameter to the Monitor, control will return to Sintran III rather than to the Monitor upon program exit. Calling the Monitor first is used mainly if other monitor commands should be given before or after the domain is executed.

In most installations, the name of the Monitor may be abbreviated and still be unambiguous. The following is a complete example of compiling a Pascal program, loading it and executing it, all programs executing in the ND-500, and the name of the Monitor is abbreviated to ND-500:

```
@ND-500 PASCAL
PASCAL/ND-500   VERSION A  81-05-08
$COM PASPROG,,"PASPROG"

  NO ERRORS
   1 NON-STANDARD WARNINGS
   0.34 SECONDS COMPILATION TIME

$EX

@ND-500 LINK
ND-Linkage-Loader 81.07.14
NLL:SET-DOMAIN "PASCAL-TEST"
NLL:OP-SEGM "SEGMENT-ONE",,
NLL:LOAD PASPROG PASCAL-LIB
Program:.............450 P     Data:............352 D
Program:...........16644 P     Data:.........402420 D
NLL:EX

@ND-500 PASCAL-TEST

I execute, therefore I am.

I have been executed, therefore I am not.

@
```

15.2. **Using libraries**

A user may find it tiresome to specify loading of a library every time he loads a program, if that library is not specified as an auto-load library by user SYSTEM. A user may also have his own libraries; containing for example mathematical or statistical routines.

In the following example, all auto-load files are deleted in order to make sure no obsolete entries remain in the table of auto-load files. PLANC-LIB is then defined as an auto-load file for Planc programs. This would not be neccessary if user SYSTEM had defined it as an auto-load file, unless the user wants to force loading of the libraries in another sequence.

For both Planc and Fortran the user's own STAT-LIB is defined as auto-load file. On this file is built a routine table by the PREPARE-NRF-LIBRARY-FILE command, in order to increase the speed of loading, and the identifying text STAT-LIB-JULY-1981 is inserted at the top of the file.

Finally, the defined auto-load files are listed, in order to confirm that the file names are correct. The version of the loader executing in the ND-500 is used, therefore the Linkage-Loader is called up through the Monitor:

```
@ND-500 LINKAGE-LOADER
ND-Linkage-Loader 81.07.14
NLL:DELETE-AUTO-LOAD-FILE
NLL:SET-AUTO-LOAD (SYSTEM)PLANC-LIB PLA
NLL:SET-AUTO-LOAD STAT-LIB PLA
NLL:SET-AUTO-LOAD STAT-LIB FOR
NLL:PREP-NRF-LIB STAT-LIB
NLL:INSERT-NRF-MESSA STAT-LIB,,STAT-LIB-JULY-1981$
NLL:LIST-AUTO-LOAD
(PACK-ONE-1:SYSTEM)PLANC-LIB  - PLANC
(PACK-ONE-1:PROJECT)STAT-LIB  - PLANC
(PACK-ONE-1:PROJECT)STAT-LIB  - FORTRAN
NLL:EX

@
```

Now assume that the routines F22 and F23 in STAT-LIB have been recompiled to the file UPDATES. The new modules should replace the old ones in the library, and the routines should be reloaded to the domain DOMANE (without reloading the entire segment). The segment in DOMANE has the name SEG-X. The new versions of F22 and F23 use another routine in STAT-LIB that was not previously loaded, therefore STAT-LIB is automatically loaded at EXIT (which implies execution of an CLOSE-SEGMENT), and the identification of the library is printed:

```
@ND-500 LINKER
ND-Linkage-Loader 81.07.14
NLL:NEW-NRF-MODULES UPDATES STAT-LIB
NLL:CC  REBUILD MODULE INDEX TABLE:
NLL:PREP-NRF-LIB STAT-LIB
NLL:CC THE PREP-OPERATION DESTROYS THE MESSAGE!
NLL:INSERT-NRF-MESSA STAT-LIB,,STAT-LIB-JULY-1981$
NLL:SET-DOMAIN DOMANE
NLL:APP-SEG SEG-X,,
NLL:RELOAD-SEG UPDATES
Program:.....46114 P     Data:.....73466 D
NLL:EX
```

15.3. Using files

DOMANE accesses two files, one for random input as file number 10, another for sequential output as file number 12. The input file REC:DATA is contigous, and the record size is 1024 bytes, thus the file may be accessed in the direct transfer open mode. The output file REPORT:LIST is an ordinary sequential file and is opened with Write access:

```
@ND-500
ND-500 MONITOR  81.05.21/81.05.15
N500:OPEN-FILE REC 10 D
N500:OPEN-FILE REPORT:LIST 12 W
N500:DOMANE
N500:EXIT
@
```

15.4. Macros

DOMANE is executed often, always using the same output file, but with different input files. In order to reduce the number of commands required to execute the program, a macro called XQT is defined and saved as a permanent macro. When executed, it will request the input file name, but supply all other parameters automatically:

```
@ND-500
ND-500 MONITOR  81.05.21/81.05.15
N500:DEF-MAC XQT
PARAMETER INFILE,NO-DEFAULT,Input-file=
OPEN-FILE INFILE 10 D
OPEN-FILE REPORT:LIST 12 W
DOMANE
EXIT
END-MACRO
N500:DUMP-MAC XQT
N500:EX
@

@ND-500
ND-500 MONITOR  81.05.21/81.05.15
N500:XQT
Input-file=NEW
N500: OPEN-FILE NEW 10 D
N500: OPEN-FILE REPORT:LIST 12 W
N500: DOMANE
N500: EXIT

@
```

Observe that the user did not enter the commands to open the files and start the domain, but these commands are always echoed to the output device.

15.5. Debugging

The following Planc program fragment:

```
MODULE PLCTEST
REAL PRECISION(15) ARRAY: ARR(1:10)
REAL PRECISION(15): TOTAL
INTEGER ARRAY: STACK(0:100)
INTEGER: I

PROGRAM: SUM
INISTACK STACK

DO WHILE I<10
  TOTAL+ARR(I)=: TOTAL
  I+1=: I
ENDDO

ENDROUTINE
ENDMODULE
```

will provoke a PROTECT VIOLATION error message from the Monitor if loaded to a segment using default values. After compilation and loading to domain PLCTEST:

```
@ND-500 PLCTEST

PROTECT VIOLATION
AT PROGRAM ADDRESS          40B

@
```

In order to catch the error, the program is placed in memory, using the DEBUG-PLACE command in order to permit modifications. Then single step execution is started, and one instruction at a time is executed by pressing CR.

```
@ND-500-MONITOR
ND-500 MONITOR  81.05.21/81.05.15
N500:DEB-PLA
DOMAIN: DOMANE
N500: STEP
P 10000000004B: INIT 00000000134B, +00000000024B, 000624B
P 10000000021B: W COMP2 00000000760B, 12B
P 10000000030B: IF >= GO 042B--> 01000000072B
P 10000000032B: D1:= 00000000124B
P 10000000040B: W1:= 00000000760B
P 10000000046B: D1 + 37777777774B(W1)

PROTECT VIOLATION
AT PROGRAM ADDRESS          40B

N500:
```

Obviously, something went wrong when access to an array element was attempted. The index value was loaded from address 760, and this value is inspected:

```
N500: LOOK-AT-DATA
Address: 760
D       760B:          0B EX
N500:
```

A base address of 37777777774B and a displacement of 0 will generate an access to a negative segment address. This is certainly not legal, as we then "overflow" into another segment (in our case, a non-existing one). The intention was to let the index variable run from 1 to 10, rather than from 0 to 9, and we therefore deposit the initial value 1 (old initial value: 0) in location 760, and we modify the upper limit of the test from 12B to 13B:

```
N500: LOOK--DATA 760
D 000000760B          0  PERMIT-DEP
D 000000760B          0  1
D 000000764B          0  EX,,
N500: LOOK--PROG 21
P 10000000021B: W COMP2 00000000760B, 12B PER
P 10000000021B: W COMP2 00000000760B, 12B BYTE
P 10000000021B:  56B
P 10000000022B: 304B
P 10000000023B:   0B
P 10000000024B:   0B
P 10000000025B:   1B
P 10000000026B: 360B
P 10000000027B:  12B 13
P 10000000030B: 314B EX
N500:
```

Now, the modified version of DOMANE in memory may be started by a by a RUN command.

```
N500: RUN
N500: EX

@
```

15.6. System Supervisor: Installing NLL

The first time NLL is installed, user SYSTEM should define auto-link segments to be used if a user attempts to close a segment while undefined references exist. Usually, the run time libraries for different languages are loaded to a segment named LIBRARY-DOMAIN, but if an installation makes heavy use of other special libraries, for example a collection of mathematical or statistical functions, it may be convenient to load even this library to the library segment.

Different run time libraries may be loaded to different segments, but as long as no symbol conflicts occur, they may all be put on the same segment. This will reduce the probability of segment number conflicts.

When a library segment is created, all traps should be locally disabled, in order to inhibit the automatic allocation of a trap handler vector.

Because default segment numbers grow from low to high, autolink segments should preferably be numbered from above. The system supervisor may also choose to define the :NRF files as library files. If, for example, a user defined auto-load file (loaded after the linking has been performed) makes further references to a standard library, the reference will be defined by the automatically loaded files.

If an entered command is not in the command list, is not the name of a domain belonging to the user issuing the command, and is not the name of a temporary or permanent macro, the domains of user SYSTEM will be searched. ND-500 systems such as compilers running on the ND-500 and the ND-500 version of NLL is usually a domain belonging to SYSTEM. Such systems are delivered either as a domain that should be copied by COPY-DOMAIN, or as an NRF file that should be loaded like another program. The example below shows how PASCAL is loaded.

In order to speed the search for a compiler or other standard system, all such domains owned by SYSTEM should also be defined as standard-domains.

```
@ND-500 LINKAGE-LOADER
NLL: SET-AUTO-LOAD (SYSTEM)FORTRAN-LIB FORTRAN
ND Linkage-Loader 80.05.18
NLL: SET-DOMAIN "LIBRARY-DOMAIN"
The "DESCRIPTION-FILE" will now be initialized
NLL: SET-SEGMENT-NUMBER 29
NLL: OPEN-SEGMENT "PLANC-LIB", P
NLL: LOCAL-TRAP-DISABLE ALL
NLL: ENTRY-ROUTINES 400
NLL: TOTAL-SEGMENT-LOAD PLANC-LIB
NLL: SET-SEGMENT-NUMBER 30
NLL: OPEN-SEGMENT "FORTRAN-LIB", P
NLL: LOCAL-TRAP-DISABLE ALL
NLL: ENTRY-ROUTINES 500
NLL: SET-IO-BUFFERS 16
```

```
NLL: TOTAL-SEGMENT-LOAD FORTRAN-LIB
NLL: SET-AUTO-LOAD (SYSTEM)PLANC-LIB PLANC
NLL: SET-AUTO-LOAD (SYSTEM)NAG FORTRAN
NLL: PREPARE-NRF-LIBRARY NAG
NLL: END-DOMAIN
NLL: SET-AUTO-LINK FORTRAN-LIB FORTRAN
NLL: SET-AUTO-LINK PLANC-LIB PLANC
NLL: EXIT

@
```

The System supervisor should also ensure that all terminals that will be using ND-500 systems have a 128k background segment. This can be changed by the Sintran III command

```
@CHANGE-BACKGROUND-SEGMENT-SIZE <term no.> 128
```

<term no.> can be found by the @WHO command. The segment size information is lost after a "cold start" ( )HENT / 22!), but the command to change it may be included in the HENT-MODE file.

If the installation runs the accounting system, the @START-ACCOUNTING command may be used to log ND-100 and ND-500 CPU time, terminal time, mass storage transfers and number of spooling pages printed. The last parameter of the command, <ND-500> is used in ND-500 systems only, and is answered with Y if resources used by ND-500 should be logged, N otherwise.

I N D E X

abbreviation, general rules . . . . . 9.
command name . . . . . 9.
domain name . . . . . 88.
macro name . . . . . 97.
ABORT-BATCH-ON-ERROR command
description . . . . . 82.
aborting
batch job . . . . . 82.
macro . . . . . 98.
ABORT-PROCESS command
description . . . . . 146.
access
mode in OPEN-FILE command . . . . . 92.
right, in segment capability . . . . . 17.
access conflicts, description file . . . . . 10.
active
processes . . . . . 117, 146-147.
users . . . . . 136.
actual macro parameter . . . . . 97.
ADA, NRF control number . . . . . 172.
adding code to loaded segment . . . . . 50, 55.
address
alignment . . . . . 170.
length . . . . . 170.
overlap . . . . . 139.
physical memory . . . . . 139.
range . . . . . 15.
table in NRF file . . . . . 80, 173.
ADDRESS-TRAP-FETCH trap . . . . . 84.
ADDRESS-TRAP-READ trap. . . . . 84.
ADDRESS-TRAP-WRITE trap . . . . . 84.
ADDRESS-ZERO-ACCESS trap. . . . . 84.
ADI, NRF control number . . . . . 172.
AJS, NRF controlnumber . . . . . 150.
allocation of memory . . . . . 124.
alphanumeric label . . . . . 11.
ampersand (&) . . . . . 9.
APA, NRF control number . . . . . 172.
APPEND-NRF-MODULES command
description . . . . . 77.
APPEND-SEGMENT command
description . . . . . 50.
reference . . . . . 48, 54.
architecture, memory management system . . . . . 15.
ASCII . . . . . 105.
characters in NRF symbols . . . . . 169.
format, LOOK-AT commands . . . . . 104.
assembler language code . . . . . 170.
assembling in library mode . . . . . 56.
at, commercial (@) . . . . . 8, 82.
ATTACH-PROCESS command
description . . . . . 146.
attribute of segment . . . . . 47.
auto-link segment . . . . . 60.
auto-load file . . . . . 62.
AUTOMATIC-ERROR-MESSAGE command

description . . . . . 135.
BCD-OVERFLOW trap . . . . . 84.
BEG, NRF controlnumber . . . . . 76, 170.
body of macro . . . . . 97.
BRANCH-TRACE command
description . . . . . 112.
reference . . . . . 116.
BRANCH-TRAP . . . . . 84.
BREAK command
description . . . . . 102.
reference . . . . . 103, 115.
breakpoint . . . . . 89, 102.
BREAK-POINT-INSTRUCTION-TRAP . . . . . 84.
BRF format . . . . . 69.
buffer, histogram and log commands . . . . . 117.
byte
format in patching . . . . . 107, 110.
parameter monitor call . . . . . 153.
pointer during loading . . . . . 170, 172.
CALL instruction . . . . . 112, 153.
called routine, local data field . . . . . 108.
CALLG instruction . . . . . 153.
calling routine, local data field . . . . . 108.
CALL-TRACE command
description . . . . . 112.
reference . . . . . 116.
CALL-TRAP . . . . . 84.
cataloged file as segment . . . . . 18.
CC command
description . . . . . 82.
CGR0, NRF controlnumber . . . . . 173.
CGR1, NRF controlnumber . . . . . 173.
characters legal in names . . . . . 10.
checksum . . . . . 170.
CHECK-SYNTAX-MODE command
description . . . . . 74.
CLEAR-DOMAIN command
description . . . . . 43.
clearing
histogram buffer . . . . . 117.
process log buffer . . . . . 119.
CLEAR-SEGMENT command
description . . . . . 51.
CLOSE-FILE command
description . . . . . 93.
CLOSE-SEGMENT command
description . . . . . 47.
implicit . . . . . 43, 83.
closing file after escape . . . . . 89.
COBOL
language code . . . . . 170.
cold start . . . . . 139.
command
abbreviation . . . . . 8.
input file . . . . . 8.
list (HELP) . . . . . 81.
output file . . . . . 8.
syntax . . . . . 9, 74.
terminator . . . . . 9.
comment . . . . . 82.

commercial at (@) . . . . . 9, 82.
common
  block . . . . . 58.
  label . . . . . 66.
communication
  between processes 16, 19
  device . . . . . 8, 81.
  with ND-100 . . . . . 39, 69-70.
  with the ND-500 process . . . . . 123.
COMND monitorcall (MON 70) . . . . . 82.
COMPARE-CONTROL-STORE command
  description . . . . . 143.
compilation errors . . . . . 173.
compiling a library . . . . . 54, 79.
compound NRF group . . . . . 172, 175.
conditional loading . . . . . 55, 171, 173.
configuration, physical memory . . . . . 139.
connecting file . . . . . 92.
CONTINUE command
  description . . . . . 89.
  reference . . . . . 102.
control
  byte, NRF . . . . . 169.
  characters in NRF symbol . . . . . 169.
control store (microprogram memory) . . . . . 142-144.
control number, NRF . . . . . 169.
COPY-DOMAIN command
  description . . . . . 45.
  reference . . . . . 69, 70.
CPU
  time used in ND-100 . . . . . 136.
  utilization . . . . . 120, 121.
creating segment . . . . . 47.
data
  byte pointer . . . . . 170, 172.
  memory LOOK-AT . . . . . 107.
  mode NRF control number . . . . . 172.
  transfer between ND-100 and ND-500 . . . . . 69.
DATA-REFERENCE command
  description . . . . . 65.
DBG, NRF control number . . . . . 173.
#DCLC, Data Current Location Counter. . . . . 64, 170.
DDF, NRF control number . . . . . 76, 171.
debug information
 in :LINK file . . . . . 8, 15.
 in :NRF file . . . . . 173.
debugging . . . . . 102-116.
DEBUG-PLACE command
  description . . . . . 102.
  reference . . . . . 87.
DEBUG-STATUS command
  description . . . . . 114.
decimal format . . . . . 9, 110, 138.
DEF, NRF controlnumber . . . . . 76, 78, 171.
default
  domain name . . . . . 8, 42, 44.
  macro parameter . . . . . 97.
  main format . . . . . 109.
  segment attributes . . . . . 47.

segment name . . . . . 8, 54.
segment number . . . . . 51.
DEFINE-COMMON command
description . . . . . 66.
defined symbols . . . . . 66, 171.
DEFINE-ENTRY command
description . . . . . 65.
DEFINE-MACRO command
description . . . . . 97.
DEFINE-MEMORY-CONFIGURATION command
description . . . . . 139.
DEFINE-SWAP-FILE command
description . . . . . 149.
DELETE-AUTO-LOAD-FILE command
description . . . . . 63.
DELETE-DOMAIN command
description . . . . . 43.
DELETE-NRF-MODULES command
description . . . . . 77.
DELETE-SEGMENT command
description . . . . . 52.
DELETE-SWAP-FILE command
description . . . . . 149.
deleting a macro . . . . . 100.
demand paging . . . . . 124.
description file . . . . . 10, 75, 165.
description of NRF format . . . . . 169.
DESCRIPTION-FILE:DESC . . . . . 10.
descriptor addressing in monitor calls . . . . . 153.
DESCRIPTOR-RANGE trap . . . . . 84.
destroying I1 register contents . . . . . 153.
difference in physical address ND-100/ND-500 . . . . 139.
direct transfer files . . . . . 95.
DISABLE-PROCESS-SWITCH-ERROR trap . . . . . 84.
DISABLE-PROCESS-SWITCH-TIMEOUT trap . . . . . 84.
disabling trap . . . . . 20, 84-86.
disassemble instruction . . . . . 103.
DISASSEMBLE-MODE command
description . . . . . 74.
disconnecting file . . . . . 93.
disk access from ND-500 . . . . . 139.
display
control store (micro program) . . . . . 143.
hardware register . . . . . 145.
DIVIDE-BY-ZERO trap . . . . . 84.
DMO, NRF control number . . . . . 170, 172.
domain . . . . . 6, 15.
status . . . . . 44.
name . . . . . 10.
number . . . . . 16, 165.
double definition . . . . . 56.
double quote (") . . . . . 9.
doublefloat format LOOK-AT . . . . . 105.
DRF, NRF controlnumber . . . . . 171.
:DSEG file . . . . . 9, 15.
DUMP-MACRO command
description . . . . . 100.
EDIT subcommand
description . . . . . 144.
empty macro parameter . . . . . 98.

I N D E X

ENABLED-TRAPS command
description . . . . . 114.
enabling trap . . . . . 85.
END, NRF control number . . . . . 76, 175.
END-DOMAIN command
description . . . . . 42.
implicit . . . . . 83.
END-MACRO command
description . . . . . 97.
End of NRF file . . . . . 77.
ENTRY-ROUTINES command
description . . . . . 72.
EOF, NRF control number . . . . . 79, 173.
ERASE-MACRO command
description . . . . . 100.
ERMSG monitorcall (MON 64) . . . . . 135, 153.
error
abort of NLL or Monitor. . . . . 82.
codes from file system . . . . . 94, 153.
during compilation . . . . . 173.
message from monitor calls . . . . . 135.
in monitorcall . . . . . 153.
termination of macro . . . . . 98.
escape key . . . . . 135.
examining
control store (microprogram) . . . . . 142.
physical memory . . . . . 145.
physical segment . . . . . 145.
resident memory . . . . . 145.
EXCDEF exception library routine . . . . . 29.
EXCEPT exception library routine . . . . . 24.
exception . . . . . 20, 23.
exception handler library . . . . . 21-36.
EXCTERM exception library routine . . . . . 31.
EXECUTE-MACRO command
description . . . . . 99.
implicit . . . . . 88.
executing a domain . . . . . 88.
execution interrupt . . . . . 135.
EXHIBIT-ADDRESS command
description . . . . . 113.
reference . . . . . 115.
EXIT command
description . . . . . 83, 104.
EXTRA-FORMAT command
description . . . . . 110.
within LOOK-AT . . . . . 105.
FETCH-NRF-MODULES command
description . . . . . 76.
file
buffers for sequential Fortran I/O . . . . . 72.
close . . . . . 93.
open . . . . . 92.
file access monitor call error returns . . . . . 94.
fixed priority . . . . . 136.
flag
for communication with an ND-500 process . . . . . 123.
float format in LOOK-AT commands . . . . . 105.
FLOATING-OVERFLOW trap. . . . . 84.
FLOATING-UNDERFLOW trap . . . . . 84.

FMO, NRF control number . . . . . 170, 172.
forcing logout of other users . . . . . 146.
formal macro parameter . . . . . 99.
format in LOOK-AT commands. . . . . 109, 110.
Fortran
   file number . . . . . 92.
   language code. . . . . 170.
   sequential I/O . . . . . 72.
free
   bytepointer during loading . . . . . 170.
   mode . . . . . 172.
GET-FLAG command
   description . . . . . 123.
GIVE-ND-500-PAGES command
   description . . . . . 140.
GLOBAL-ENTRIES command
   description . . . . . 68.
GO command
   description . . . . . 89.
GROUP subcommand
   description . . . . . 144.
GUARD command
   description . . . . . 111.
   reference . . . . . 115.
halfword
   format in LOOK-AT commands . . . . . 110.
   parameter to monitor calls . . . . . 153.
hardware registers LOOK-AT . . . . . 145.
HELP
   description . . . . . 81.
   in LOOK-AT . . . . . 104.
hexadecimal
   command parameters . . . . . 8.
   format in LOOK-AT. . . . . 110.
HIGH-ADDRESS command
   description . . . . . 71.
high limit register . . . . . 111.
histogram
   commands . . . . . 117.
HL register . . . . . 111.
I registers . . . . . 153.
IF-ERROR-FULL-STOP subcommand
   description . . . . . 98.
IF-ERROR-MACRO-STOP subcommand
   description . . . . . 98.
ignorable trap . . . . . 19.
IHB, NRF control number . . . . . 173, 176.
illegal control number . . . . . 173, 176.
ILLEGAL-INDEX trap . . . . . 84.
ILLEGAL-INSTRUCTION-CODE trap . . . . . 84.
ILLEGAL-OPERAND-SPECIFIER trap . . . . . 84.
ILLEGAL-OPERAND-VALUE trap . . . . . 84.
implicit
   CLOSE-SEGMENT . . . . . 43, 47, 83.
   END-DOMAIN . . . . . 42, 83.
   EXECUTE-MACRO . . . . . 88, 99.
   OPEN-SEGMENT . . . . . 54.
   RECOVER-DOMAIN . . . . . 88, 99.
INDEX-SCALING-ERROR trap. . . . . 84.
inhibit execution . . . . . 151.

initialization of traphandler data field . . . . . 20.
initializing
  data memory . . . . . 109.
input to program in macro body . . . . . 97.
INSERT-NRF-MESSAGE command
  description . . . . . 79.
instruction set . . . . . 20.
INSTRUCTION-SEQUENCE-ERROR trap . . . . . 84.
interface
  register LOOK-AT . . . . . 145.
intermodule reference . . . . . 11.
INVALID-OPERATION trap . . . . . 84.
I/O
  buffer for sequential file access . . . . . 72.
  file open and close commands . . . . . 92-94.
K flag . . . . . 153.
KILL-ENTRIES command
  description . . . . . 68.
label . . . . . 8, 11.
  definition . . . . . 11, 171.
language
  code . . . . . 170.
  sensitivity 60,62
layout of description file . . . . . 165.
LBB, NRF controlnumber . . . . . 173.
LDI, NRF controlnumber . . . . . 172.
length of NRF symbol . . . . . 169.
LIB, NRF controlnumber . . . . . 56, 76, 171.
LIBRARY-SEGMENT-LOAD command
  description . . . . . 55.
  reference . . . . . 56.
line continuation . . . . . 9.
:LINK file . . . . . 9, 15.
LINK-RT-PROGRAM command
  description . . . . . 70.
LIST-ACTIVE-SEGMENTS command
  description . . . . . 148.
LIST-AUTO-LINK-SEGMENT command
  description . . . . . 61.
LIST-AUTO-LOAD-FILE command
  description . . . . . 63.
LIST-DOMAIN command
  description . . . . . 44.
LIST-ENTRIES-DEFINED command
  description . . . . . 66.
  reference . . . . . 67.
LIST-ENTRIES-UNDEFINED command
  description . . . . . 67.
LIST-MACROS command
  description . . . . . 101.
LIST-MAP command
  description . . . . . 67.
LIST-MODE command
  description . . . . . 74.
LIST-NRF-CODE command
  description . . . . . 78.
LIST-NRF-ENTRIES command
  description . . . . . 78.
LIST-OCTAL command
  description . . . . . 73.

LIST-OPENED-FILES command
description . . . . . 93.
LIST-PROCESS-TABLE-ENTRY command
description . . . . . 148.
LIST-SEGMENT command
description . . . . . 52.
LIST-SYMBOLIC command
description . . . . . 73.
LIST-TABLE command
description . . . . . 147.
LL register . . . . . 111, 115.
load immediately, NRF control number . . . . . 172.
LOAD-CONTROL-STORE command
description . . . . . 142.
loader table . . . . . 11.
loader table overflow . . . . . 68.
LOAD-SEGMENT command
description . . . . . 54.
local
data field . . . . . 108.
memory . . . . . 139.
trap handling . . . . . 19, 85.
LOCAL-TRAP-DISABLE command
description . . . . . 85.
LOCAL-TRAP-ENABLE command
description . . . . . 85.
logging
all processes . . . . . 120.
one process . . . . . 121.
LOOK-AT commands
description . . . . . 104.
subcommands . . . . . 108, 144.
LOOK-AT-CONTROL-STORE command
description . . . . . 143.
LOOK-AT-DATA command
description . . . . . 107.
LOOK-AT-HARDWARE command
description . . . . . 145.
LOOK-AT-PHYSICAL-SEGMENT command
description . . . . . 145.
LOOK-AT-PROGRAM command
description . . . . . 106.
reference . . . . . 144.
LOOK-AT-REGISTER command
description . . . . . 108.
LOOK-AT-RELATIVE command
description 108
LOOK-AT-RESIDENT-MEMORY command
description . . . . . 144.
LOOK-AT-STACK command
description . . . . . 107.
LOW-ADDRESS command
description . . . . . 71.
low limit register . . . . . 111, 115.
LRF, NRF control number . . . . . 171.
:MACR file . . . . . 97,98.
macro
body . . . . . 97.
commands . . . . . 97-101.
main

format . . . . . 109.
program data field . . . . . 107.
MAIN-FORMAT command
description . . . . . 109.
MASTER-CLEAR command
description . . . . . 151.
MATCH-COMMON-RT-SEGMENT command
description . . . . . 70.
MATCH-RTCOMMON command
description . . . . . 69.
memory
administration . . . . . 140.
allocation . . . . . 124.
configuration . . . . . 139-140.
memory management system . . . . . 35.
microprogram . . . . . 143.
maintainance . . . . . 142.
registers . . . . . 108.
MICRO-START command
description . . . . . 142.
MICRO-STOP command
description . . . . . 142.
MIS, NRF control number . . . . . 173.
MMS (Memory Management System) . . . . . 13.
modularization . . . . . 16.
module . . . . . 8, 11, 170.
MON 60 N500M . . . . . 161.
MON 64 ERMSG . . . . . 135.
MON 65 QERMS . . . . . 135.
monitor call
arguments . . . . . 153.
priority . . . . . 136.
MSA, NRF control number . . . . . 170.
MSG, NRF control number . . . . . 173.
multiple definition . . . . . 56.
name . . . . . 11.
conflicts . . . . . 68.
syntax . . . . . 9.
ND-100
communication . . . . . 39-51, 69-70.
monitor calls . . . . . 136.
ND Relocatable Format (NRF) . . . . . 169.
negative values in NRF code . . . . . 169.
nested compound group . . . . . 173.
NEW-NRF-MODULES command
description . . . . . 76.
NEXT command
description . . . . . 99.
non-printing characters . . . . . 169.
non-reentrant traphandler . . . . . 19.
NRF . . . . . 169-174.
editor . . . . . 76.
library file . . . . . 80.
symbol . . . . . 64, 170.
NRF file maintainance . . . . . 76.
NUL, NRF control number . . . . . 170.
numeric field, NRF . . . . . 169.
numeric length, NRF . . . . . 169.
numeric parameters . . . . . 8.
octal . . . . . 8.

listing . . . . . 73, 78.
OMITTED-SEGMENT-LOAD command
description . . . . . 56.
omitting
EXECUTE-MACRO . . . . . 99.
RECOVER-DOMAIN . . . . . 88.
OPEN-FILE command
description . . . . . 92.
OPEN-SEGMENT command
description . . . . . 47.
implicit . . . . . 54.
optional parameter . . . . . 9.
ORIN subcommand
description . . . . . 144.
OUTBT monitorcall (MON 2) . . . . . 39.
output device . . . . . 7, 81.
output flag . . . . . 123.
OUTPUT-FILE command
description . . . . . 81.
OUTST monitorcall (MON 162) . . . . . 39.
OVERFLOW trap . . . . . 84.
overhead monitorcall . . . . . 39.
page fault . . . . . 124.
pages, giving to ND-500 . . . . . 140.
parameter . . . . . 97.
addresses . . . . . 153.
reference . . . . . 98.
terminator . . . . . 9.
parity . . . . . 169.
microprogram . . . . . 124.
#PCLC (Program Current Location Counter) . . . . . 64, 170.
percentage of CPU time used in ND-100 . . . . . 136.
performance measurment . . . . . 117.
PERMIT-DEPOSIT subcommand
description . . . . . 105.
physical segment . . . . . 145.
PLACE-DOMAIN command
description . . . . . 87.
PMO, NRF control number . . . . . 170, 172.
PREPARE-NRF-LIBRARY-FILE command
description . . . . . 80.
reference . . . . . 54.
PREVIOUS subcommand
description . . . . . 108.
PRIMESS exception library routine . . . . . 33.
PRINT-HISTOGRAM command
description . . . . . 118.
PRINT-PROCESS-LOG command
description . . . . . 120.
reference . . . . . 118.
priority, monitorcall . . . . . 136.
PRITRAC exception library routine . . . . . 32.
PROCESS-LOG-ALL command
description . . . . . 121.
PROCESS-LOG-ONE command
description . . . . . 121.
PROCESS-STATUS command
description. . . . . 147.
program . . . . . 170.
label . . . . . 64.

mode . . . . . 172.
reference . . . . . 64, 171.
program counter sampling . . . . . 118.
PROGRAMMED-TRAP . . . . . 84.
PROGRAM-REFERENCE command
description . . . . . 64.
PROTECT-VIOLATION . . . . . 84.
QERMS monitorcall (MON 65) . . . . . 153.
read only segment . . . . . 47.
RECOVER-DOMAIN command
description . . . . . 88.
implicit . . . . . 88.
REF, NRF control number . . . . . 171.
references, undefined . . . . . 67.
register LOOK-AT . . . . . 108.
RELEASE-DOMAIN command
description . . . . . 46.
RELEASE-HISTOGRAM command
description . . . . . 118.
RELEASE-LOG-BUFFER command
description . . . . . 122.
RELOAD-SEGMENT command
description . . . . . 55.
RENAME-DEFAULT-DIRECTORY-AND-USER command
description . . . . . 75.
RENAME-DOMAIN command
description . . . . . 44.
RENAME-SEGMENT command
description . . . . . 52.
REP, NRF controlnumber . . . . . 172, 173.
RESET command
description . . . . . 74.
RESET-AUTOMATIC-ERROR-MESSAGE command
description . . . . . 135.
RESET-BRANCH-TRACE command
description . . . . . 116.
RESET-BREAKS command
description . . . . . 115.
RESET-CALL-TRACE command
description . . . . . 116.
RESET-DEBUG command
description . . . . . 114.
RESET-GUARD command
description . . . . . 115.
RESET-LAST-BREAK command
description . . . . . 115.
RESET-TRACE command
description . . . . . 115.
RESUME-MACRO command
description . . . . . 100.
RMV, NRF control number . . . . . 172.
RT
program ND-100 . . . . . 39, 70.
segment ND-100 . . . . . 69.
RTCOMMON ND-100 . . . . . 40, 69.
RUN command
description . . . . . 88.
S field, NRF . . . . . 169.
sampling
CPU usage . . . . . 117-119.

search procedure, command processor . . . . . 88, 99.
SELECTED-SEGMENT-LOAD command
  description . . . . . 56.
SET-AUTO-LINK-SEGMENT command
  description . . . . . 60.
SET-AUTO-LOAD-FILE command
  description . . . . . 62.
SET-DOMAIN command
  description . . . . . 42.
  implicit . . . . . 42.
SET-FLAG command
  description . . . . . 123.
SET-HISTOGRAM command
  description . . . . . 117.
  reference . . . . . 118.
SET-IO-BUFFERS command
  description . . . . . 72.
  reference . . . . . 49.
SET-MEMORY-CONTENTS command
  description . . . . . 109.
SET-ND-500-AVAILABLE command
  description . . . . . 138.
SET-ND-500-UNAVAILABLE command
  description . . . . . 138.
SET-PRIORITY command
  description . . . . . 136.
SET-SEGMENT-NUMBER command
  description . . . . . 51.
  reference . . . . . 47.
setting K flag . . . . . 153.
sign extension . . . . . 171.
single step execution . . . . . 103.
SINGLE-INSTRUCTION-TRAP . . . . . 84.
@ sintran command
  description . . . . . 82.
SLA, NRF control number . . . . . 172.
slash (/) in LOOK-AT . . . . . 105.
STACK-OVERFLOW trap . . . . . 84.
STACK-UNDERFLOW trap . . . . . 26, 84.
standard trap handler . . . . . 19.
START-HISTOGRAM command
  description . . . . . 118.
START-PROCESS-LOG-ALL command
  description . . . . . 120.
START-PROCESS-LOG-ONE
  description . . . . . 120.
STATUS command
  desctiption . . . . . 114.
status register . . . . . 114.
STEP command
  description 103
STOP-HISTOGRAM command
  description . . . . . 118.
  program execution . . . . . 116.
string parameter to monitor call . . . . . 153.
sub controlnumber NRF . . . . . 173.
supervisor commands . . . . . 138.
SWAPPING-LOG command
  description . . . . . 122.
symbol

definition . . . . . 65.
global . . . . . 68.
length, NRF file . . . . . 169.
value . . . . . 86.
syntax
check of commands . . . . . 74.
of names . . . . . 9.
system defined auto-load/auto-link files. . . . . 60, 62.
SYSTEM-ENTRIES-ON command
description . . . . . 67.
system supervisor commands . . . . . 138.
SYSTEM-TRAP-DISABLE command
description . . . . . 86.
SYSTEM-TRAP-ENABLE command
description . . . . . 86.
TAKE-ND-500-PAGES command
description . . . . . 141.
TEMPORARY-BREAK command
description . . . . . 103.
test, checksum in NRF code . . . . . 170.
time slicing of ND-500 processes . . . . . 136.
TOTAL-SEGMENT-LOAD command
description . . . . . 57.
TRACE command
description . . . . . 110.
reference . . . . . 115.
traceback print after error . . . . . 32.
trap 19-20, 84-86
twin process . . . . . 136.
two's complement . . . . . 169, 170.
unconditional load . . . . . 57.
undefined references . . . . . 67.
user interrupt . . . . . 135.
user written trap handler . . . . . 19, 23.
VALUE-ENTRIES command
description . . . . . 86.
VERSION command
description . . . . . 136.
WHO-IS-ON command
description . . . . . 136.
WORD subcommand
description . . . . . 144.
WRITE-DOMAIN-STATUS command
description . . . . . 44.
WRITE-NRF-EOF-AFTER-MODULE command
description . . . . . 79.
WRITE-SEGMENT-STATUS command
description . . . . . 53.

– we make bits for the future

NORSK DATA A.S BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY PHONE: 30 90 30 TELEX: 18661