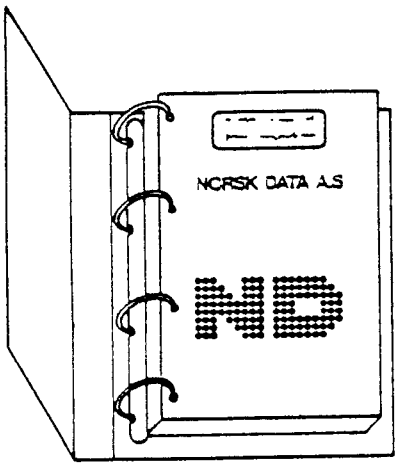
ND Relocating Loader

ND-60.066.04

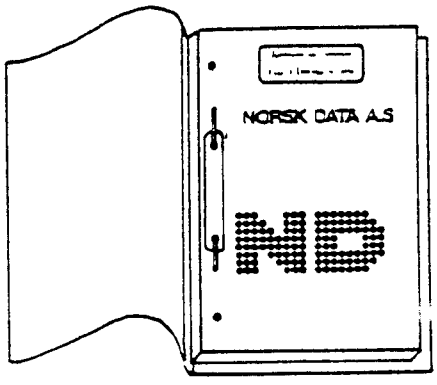
This manual is in loose leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

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:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

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 .....

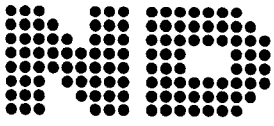
NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

PRINTING RECORD

Printing	Notes
04/77	Version 01
01/82	Version 02
07/82	Version 03
02/83	Version 04

ND Relocating Loader
Publ. No. ND-60.066.04



Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, 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 Customer Support Information (CSI) 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 and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:

THE PRODUCT

This manual describes the ND Relocating Loader running under SINTRAN III. It now incorporates the BRF Editor (formerly ND-60.085).

ND 10/ND 100 Relocating Loader ND-60.066

The binary relocating loader is used to read BRF output from the MAC assembler and from the ND compilers (Fortran, COBOL, PLANC, BASIC, PASCAL etc.) into memory and make them executable.

THE READER

This manual is written for programmers using the ND Relocating Loader. For real-time programs see:

Real-time Loader ND-60.051

PREREQUISITE KNOWLEDGE

No previous knowledge of the ND Relocating Loader is assumed in this manual. However, some basic knowledge of both the SINTRAN commands and the principles of compilation is recommended.

THE MANUAL

This manual describes the basic Loader commands in chapter 1. Chapters 2 and 3 cover Binary Relocatable Format and the BRF Editor respectively.

RELATED MANUALS

SINTRAN III Reference Manual	ND-60.128
MAC Users Guide	ND-60.096
Real-time Loader	ND-60.051

TABLE OF CONTENTS

Section Page

A P P E N D I X

1 FUNCTION OF THE NRL

The Nord Relocating loader is a subsystem which is able to convert the output from the language processors (compilers, assemblers) into executable programs running under SINTRAN III. The object files created by the language subsytems are in binary relocatable format (described in detail in chapter 2), otherwise known as BRF. The NRL relocates this output and changes its format so that it can be loaded to actual addresses, the property of relocatability being lost in the process. The NRL maintains a symbol table in which all intermodule references, symbols, and labels appear together with their defined addresses. All of these are resolved by the NRL before execution of the program can proceed.

1.1 LOADING

In this manual the term loading means "to fetch relocatable program units and link them together to form an executable program".

When loading the NRL can operate in four different modes where:

1) The BRF code is loaded directly into a current address space for immediate execution. This is known as basic loading.

2) The address space on a file contains a memory image which cannot be executed directly, but must be converted to one of the other forms. This is called image-file loading.

3) A file of type :PROG is ready for execution using the @RECOVER command. This is referred to as prog-file loading.

4) The program is too large for an address space. Overlay mode will be used to enable different parts of the program to have the same address. This mode is called overlay loading.

The relocating loader is recovered from the operating system by typing:

```
NRL
```

When an asterisk (*) is printed on the terminal, the loader is ready to accept commands from the user.

NOTE ON THE FORMAT OF NRL COMMANDS:

All NRL commands may be abbreviated provided that no ambiguity results. Parameter delimiters are: space, comma, or carriage return. Missing parameters will be asked for but default values may be specified by giving two commas or carriage return. The character "control L" (octal 14) given in a command or parameter line, will cancel the command. The line editing characters "control Q" for deleting the current line and "control A" for deleting one character on the current line, are available in the NRL. Alphabetical characters may be upper or lower case.

The appearance of square brackets ([]) in the command formats denotes the presence of an optional parameter. In the command descriptions, the terms "octal address" or "octal address/value" must be entered as an unsigned octal number of six digits or less. No trailing B is allowed and the maximum value 177777. All commands may be abbreviated so long as they remain unique in the normal SINTRAN manner.

1.1.1 Commands for Loading and Executing a BRF-Program

Loader input is obtained from one or more files or library files. The loading is initiated by the command:

*LOAD <file name>[<file name>...]

Each of the files specified will be loaded until end-of-file is detected, then control is transferred to the loader command processor (which types a *) which is then ready to accept another command.

To obtain the entry-point addresses of the loaded program, we use the command:

*ENTRIES-DEFINED [<file name>]

which will give a printout of the entry-names along with their octal addresses in memory. If no file/device name is specified, the printout will appear on the users terminal. Also, referenced (not defined) entry-points may be requested by the command:

*ENTRIES-UNDEFINED [<file name>]

The octal addresses which appear on this map denote the last reference address.

If a program has been loaded and some references still remain, a run-time library system file should be loaded as well. If any of these routines are necessary for the execution they will be selected by the loader and connected with their corresponding references. The names of these libraries are provided with the respective ND compiler and library products. There is a facility for loading libraries automatically at the end of a load operation (see the AUTOMATIC command below).

```
*AUTOMATIC
<library file 1>
                .
                .
                .
                .
<library file n>
```

The specified library files will be loaded when the RUN, DUMP and BPUN commands are used, if undefined references exist in the loader table. The loading from the libraries will terminate when all references are defined or when the library files have been scanned twice. If this results in the necessary definitions, the specified command will be performed, otherwise an error message will be written.

Example:

```
*AUTO
FTNLIBR
USER-LIBRARY
.
*
```

The command lines are terminated by a dot (.).

The pre-automatic mode buffer is not cleared by the RESET comand, and thus the loader may be initiated and dumped for later recovery with the automatic sequence intact. The buffer may be cleared by typing:

```
*AUTO
.
```

The example below shows a procedure for generating a subsystem having the automatic sequence initiated with the user's library files.

```
@PLACE-BINARY NRL-1935I:BPUN
@GO 0
RELOCATING LOADER LDR-1935I
*AUTOMATIC
USER-LIB
FTNLIBR
.
*EX
@DUMP "NRL-1935I",0,1
@
```

There should be no undefined entry-points remaining and the program may be started with the command:

```
*RUN
```

When the program has been executed, control is transferred to the operating system (as indicated by the @ prompt).

To leave the loader and return to the operating system the user writes:

```
*EXIT
```

The loader can be reentered by using the system command:

@CONTINUE

1.1.2 Load-Address Control Commands

In order to load the program at an address which differs from the current value, use the command:

*SET-LOAD-ADDRESS <octal address>

Subsequent loading will then be performed from the address specified.

Note that during basic loading, the loader itself occupies the lower part of the address space. The load address in this case must not be set lower than the upper address of the loader (which can be found from the ENTRIES-DEFINED command if it is issued as the first command).

The absolute upper limit may be redefined with:

*UPPER-LIMIT <octal address>

but care should be taken that no overlapping occurs when manipulating load-addresses.

1.1.3 Commands Affecting the Symbol-Table

The symbol table contains a list of symbols and their defined addresses. Whenever a defintion for a symbol is found in the input, the value of the current load address is the value of the symbol. Whenever a reference is found in the input to a name in the symbol table, the corresponding value is placed in the referenced position. If the referenced symbol has not yet been defined, it is stored in the symbol table awaiting resolution by a later definition. Symbols may be up to seven characters long.

Symbolic table entries may be created, renamed or deleted by the user. An entry is created by:

*DEFINE <symbol name><octal value/address>

Symbol names may be renamed by:

*RENAME <old symbol name><new symbol name>

An entry is deleted from the synbol table by:

*KILL <symbol name>

The associated address/value of an entry may be examined by typing:

*VALUE <symbol name>

The loader then prints the octal number on the terminal.

The associated address/value of an entry may be entered into a memory location by the command:

*REFERENCE <symbol>[<octal address>]

It doesn't matter if the referenced entry is present in the table or not, as the correct address will be filled in when the symbol value is defined.

If the message:

LOADER TABLE OVERFLOW

is given it means that there is no more room for entries. The table-length may be expanded through the command:

*SIZE <number of entries (octal)>

However, the old table contents are lost. This means that the load procedure must be repeated from the beginning using an appropriate table length. This command should be issued before any command that affects the symbol table or before any loading is performed, since it redefines the table and the current load address.

All table contents are removed by typing:

*RESET

However, all entries present may be protected from later removal (through RESET) by typing:

*FIX

The RESET will then merely remove all symbols entered after the time at which the table was fixed.

The current location when the FIX command is issued, becomes the lower bound address on the next RESET.

The user is advised not to fix the table when there are undefined references.

Fixed entries are not listed when using the commands:

ENTRIES-DEFINED and ENTRIES-UNDEFINED.

1.1.4 Saving and Dumping of Binary Programs

The loaded program may be saved in binary form in two ways. Firstly by:

*DUMP <destination file name>[<start address><restart address>]

This command saves the loaded program on the specifed file. The program may be retrieved with the RECOVER command. It then starts at the specified start address. The restart address indicates where the program should be started with the CONTINUE command. The dump limits may be set by the BOUNDARIES command. Default boundaries range from the lowest to the highest address accessed by the loader since the last recovery. The main entry will act as default start and restart addresses. The default file type is :PROG.

Secondly, using the command:

*BPUN <destination file name><start address><bootstrap address>

the program area (default or specified by the BOUNDARIES command) will be dumped in absolute binary form on the destination file preceded by an octal coded bootstrap. The main start entry of the program may be specified in symbolic or octal form. The bootstrap address (octal number) specifes where the bootstrap program (44 octal locations) will be located if the program is loaded into a stand-alone NORD-10/ND-100. Default destination type: BPUN. Default boundaries range from the lowest to the highest address accessed by the loader since the last recovery.

The SINTRAN command PLACE-BINARY can be used to place a BPUN file into the user's address space.

When a dump area other than the default area is preferred, it may be specified by:

*BOUNDARIES <lower address><upper address>

1.1.5 Auxiliary Memory Examination Commands

*OCTAL-DUMP <lower address><upper address>[<file name>]

The contents of the locations between the lower and upper addresses will be dumped on the specified file, with eight consecutive locations to a line. Each word will be written as a six-digit octal number. If no file name is specified the contents are dumped to the terminal.

*ASCII-DUMP <lower address><upper address>[<file name>]

The contents of the locations between the lower and upper addresses will be dumped on the specified file, eight consecutive locations (16 characters) to a line. Non-printable characters appear as a space. If no file name is specified the characters are dumped to the terminal.

*DEPOSIT <octal address>[<new contents>]

*DEPOSIT <symbol name> [,<±octal displacement>] [,<new contents>]

The new contents (octal value) are put into the octal address specified, or into the address of the symbol name, plus or minus the displacement. If the last parameter is missing, the old contents are displayed as two ASCII characters and as an octal number; they may be changed by typing the new contents on the same line. By typing CR the next location will be displayed automatically. The termination character is point (.).

1.1.6 Image-file Loading

Programs may be loaded directly onto a memory image file instead of into main memory. The loader is put into this special mode by the command:

*IMAGE-FILE <file name>

where the file name denotes the memory image file having IMAG as its default type.

The IMAGE-FILE must be the first command to be given after the loader recovery.

The DUMP and BPUN commands apply to memory images as well as to pure memory-loaded systems. SET-LOAD-ADDRESS may now set any address starting at zero.

Image-loaded programs may only be executed by applying the RECOVER or PLACE-BINARY SINTRAN III commands. Hence the DUMP or BPUN NRL commands must be used before exit from NRL.

1.1.7 Prog-file Loading

Programs may be loaded onto a file in absolute binary form instead of into main memory. The loader is put into this special mode with the command:

*PROG-FILE <file name>

where file name is the name of the file onto which the program is loaded. The default file type is PROG. PROG-FILE should be the first command given after the loader recovery, while the last command to be given must be EXIT. Programs loaded in this way may only be executed after the application of the SINTRAN III RECOVER command. The DUMP and BPUN commands cannot be used in this mode.

1.2 OVERLAY SEGMENTATION OF PROGRAMS

The NRL supports two different overlay systems. One system is tree-structured, and it can be one or more levels deep. The other system consists of a root segment (or link) and one overlay level.

1.2.1 The Multi-level Overlay System

In order to use the NORD-10/ND-100's overlay capability, the user must have a good understanding of the way in which his program operates and of the relationship between the modules within it. He should organize his overlay structure (described below) so as to retain in memory the links that contain the more-commonly used modules, and place the infrequently-used modules in links which can overlay one another. For example, a specialized error-recovery procedure would need to be in memory only when the specialized error occurred. Each link should be a collection of functionally-related modules and be as self-contained as possible, calling other links as infrequently as possible. In particular, references to links which would cause the overlaying of existing links should be minimal.

A tree-like structure, called an overlay structure, can be used to illustrate the dependencies among the overlay links. In a tree structure, each link has only one immediate ancestor but it may have more than one immediate descendent. The link containing the parts of the program which are always required and which must, therefore, always be in memory during execution, is called the root link. Since the root link receives control at the start of execution, it does not have an ancestor. The remaining links branch away from the root link and are structured according to their interdependencies.

Links which do not have to be in memory at the same time are termed independent links. For example, two modules which do not reference each other or pass data directly to each other, are independent links. When such a link is no longer required in memory, it can be overlaid by another when it is brought in. On the other hand, a dependent link must have the link upon which it depends in memory at the same time and cannot therefore overlay it. Every link is dependent on the root link.

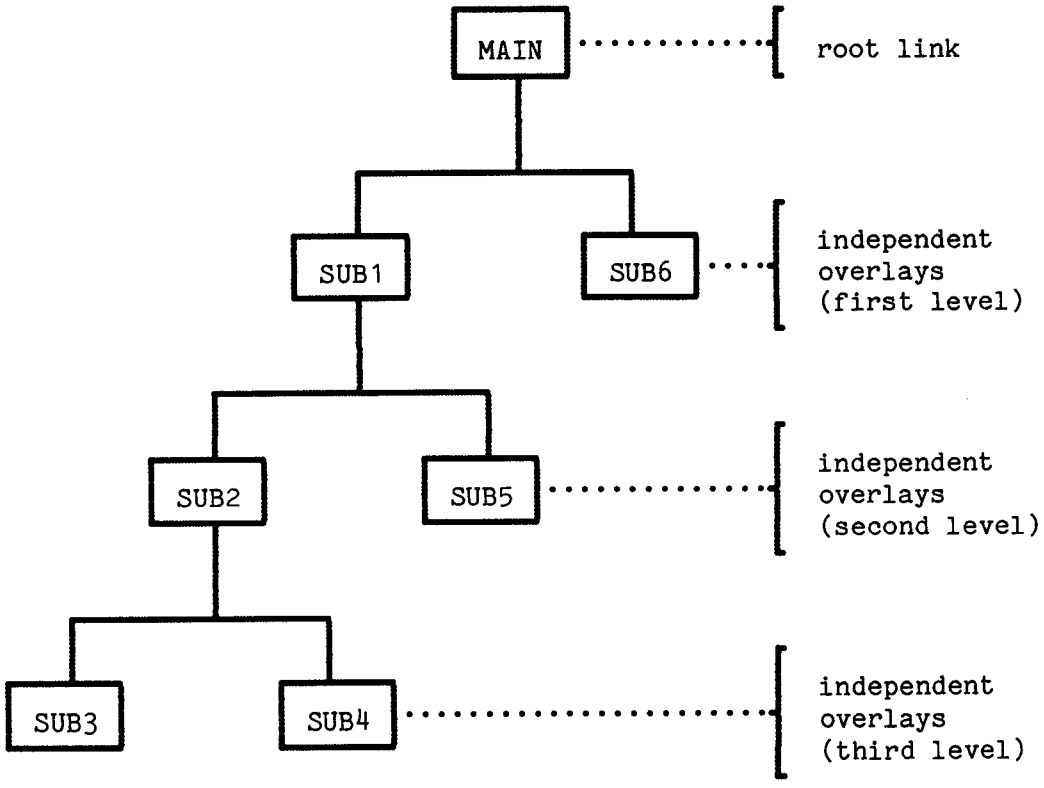
As an illustration, assume the user to have a FORTRAN program which consists of a main program, MAIN, and six subroutines, SUB1, SUB2, SUB3, SUB4, SUB5, and SUB6. The subroutines are related as follows:

1) SUB1 and SUB6 are called directly from the main program and are independent of each other.

2) SUB2 and SUB5 are called directly from SUB1 and are independent of each other.

3) SUB3 and SUB4 are called directly from SUB2 and are also independent of each other.

After analyzing these relationships, the user can draw the following tree structure to illustrate the subroutine dependencies:



By studying the dependencies in the diagram, the user can see that when a specific link is executed, all links between it and the root link must be in memory.

For example, SUB4 depends on SUB1 and SUB2 so that these two links must be in memory for the execution of SUB4. This chain of links is referred to as "the path of the link currently being executed". The action of bringing these links into memory is termed "path loading" and the chain of links branching away from the root link is known as "the extended path". In the previous example therefore, the path of SUB4 is MAIN, SUB1, and SUB2. There are three extended paths of SUB1:

1) SUB2, SUB3

2) SUB2, SUB4

3) SUB5

Links may communicate with other links if they lie on a common path or extended path. The communication is through references to global symbols. References from the current link to a global symbol in another link on the path are called backward references, while references from the current link to a global symbol in another link on the extended path are called forward references. Since all links from the current link back to the root link must be in memory, a backward

reference does not cause any links to be brought into memory. However, with a forward reference, the referenced link may not be in memory and must therefore be fetched, possibly overlaying a link already there.

1.2.2 Designing an Overlay Structure

The first step to be taken when designing an overlay structure is to draw a tree-like diagram showing the functional relationships among the modules within the program. The tree begins with the root link containing the main program and which remains in memory throughout execution. The remainder of the program is contained in the overlay links that support the root link.

Links that are functionally related lie on the same path. Links that can overlay each other are at the same level on different paths but, in this case, are not functionally related.

The user should remember several points when drawing his overlay structure:

1) References that will cause overlaying of existing links should be minimized.

2) Independent links cannot reference each other; communication is by way of a common link.

3) As a general rule, calls should be forward references, while returns should be backward references.

4) If data is modified during execution, the modification is destroyed once the link is overlaid. If therefore, data required by another link is modified, then the data must be returned to this other link before the one containing the changed data is overlaid.

5) When a link is to be overlaid , there should be no addresses or references to it remaining.

6) Modules or data areas used by several links should be explicitly loaded into a link that is common to all links using these modules or data areas. For example, a COMMON data area should be in a link immediately before the first link which references it. Moreover, COMMON should be positioned in such a way that it never gets re-initialized after the first call.

Tree-structured overlay systems can be one or more levels deep. The amount of memory required is at least the amount needed for the longest path. The length of the longest path is not the minimum requirement however, since special tables must be included when a program is divided into links.

The root link and the COMMON area defined within it reside in memory throughout the entire execution, while the overlays and the COMMON area defined only within them, reside on a random read-only file. This file is specified with the PROG-FILE command.

1.2.3 Overlay Handler Subroutines

The following subroutines are available to the user program and must be called when designing a multi-level overlay structure.

CALL OVLINIT

The initialization routine is used to open the overlay file and must be called before any of the overlays can be loaded.

CALL OVERLAY (<name>[,<par 1>,... ,<par n>])

where <name> is the name of the subprogram to be given control on an overlay; it is written as a hollerith constant having a length of seven characters. <par 1> to <par n> are the actual parameters to the subroutine.

The specified overlay will always be loaded into memory, and control will be transferred to the start address of the routine <name>.

CALL OVRECAL (<name>[,<par 1>,... ,<par n>])

For the above subroutine the parameters have the same meaning as for OVERLAY. The specified overlay will only be loaded if it is not already in memory. Thus successive calls to the same overlay will cause it to be loaded on the first call or if it has been overlaid since the last call. Control will be transferred to the start address of the routine <name>.

Calls to the three subroutines have been shown in FORTRAN format. For COBOL users the CALL statements would appear as follows:

CALL "OVLINIT"

CALL "OVERLAY" [USING <name> [<par 1>,...,<par n>]]

CALL "OVRECAL" [USING <name> [<par 1>,...,<par n>]]

where "name" is a literal of length seven characters, the name of the routine being left justified within it.

1.2.4 Loader Commands for an Overlay System

The following loader commands must be used when creating an overlay system.

*PROG-FILE <file name>

The BRF information will be loaded into the specified file instead of directly into main memory.

*OVERLAY-GENERATION [<no. of overlay entries>]

This command specifies that a multi-level overlay system is to be generated. <no. of overlay entries> is the maximum number of overlay entries specified in the OVERLAY-ENTRY commands. The default is 128.

For two-bank loading (see section 1.4.2) the command SET-MODE DATA must be issued before the OV-GEN command in order to place the run-time table in the correct bank.

Note: The root link must be completed by loading the appropriate language library.

*OVERLAY-ENTRY [(<level>)] <entry name 1>[,... ,<entry name n>]

This command specifies that the next overlay link is to be generated. <level> is the overlay level, it has a default value of 1. <entry name 1> to <entry name n> give the names of the subprograms called by the OVERLAY and/or OVRECAL routines from the previous level. The root link is level 0.

NOTE: The level number must be in parentheses.

*EXIT

This command dumps the root link, the COMMON area, and the last overlay link, onto the file specified in the PROG-FILE command. The execution of the overlay system must be started by the RECOVER command.

EXAMPLE:

```
@FTN
NORD 10/ND-100 FORTRAN COMPILER FTN-2090I
$COM MAIN,1,MAIN
         1*        PROGRAM MAIN
         2*        WRITE (1,*)  'START MAIN'
         3*        CALL OVLINIT
         4*        CALL OVERLAY (7HSUB1   ,1)
```

```
         5*        CALL OVERLAY (7HSUB6   ,6)
         6*        WRITE (1,*)  'END MAIN'
         7*        END


7 LINES COMPILED . OCTAL SIZE=   137
CPU-TIME USED IS 0.2 SEC.
$COM SUB1,1,SUB1
         1*        SUBROUTINE SUB1(N)
         2*        WRITE (1,*)  'SUBROUTINE' ,N, ' CALLED'
         3*        CALL OVERLAY (7HSUB2   ,2)
         4*        CALL OVERLAY (7HSUB5   ,5)
         5*        END


5 LINES COMPILED . OCTAL SIZE=     124
CPU-TIME USED IS 0.2 SEC.
$COM SUB2,1,SUB2
         1*        SUBROUTINE SUB2(N)
         2*        WRITE (1,*)   'SUBROUTINE' ,N, ' CALLED'
         3*        CALL OVRECALL (7HSUB3   ,N+1)
         4*        CALL OVRECALL (7HSUB3   ,N+1)
         5*        CALL OVRECALL (7HSUB4   ,N+2)
         6*        CALL OVRECALL (7HSUB4   ,N+2)
         7*        END


7 LINES COMPILED .  OCTAL SIZE=     161
CPU-TIME USED IS 0.3 SEC.
$COM SUB3,1,SUB3
         1*        SUBROUTINE SUB3 (N)
         2*        WRITE (1,*)   'SUBROUTINE' ,N, ' CALLED'
         3*        END


3 LINES COMPILED .  OCTAL SIZE=     56
CPU-TIME USED IS 0.1 SEC.
$COM SUB4,1,SUB4
         1*        SUBROUTINE SUB4(N)
         2*        WRITE (1,*)   'SUBROUTINE' ,N, ' CALLED'
         3*        END


3 LINES COMPILED . OCTAL SIZE=     56
CPU-TIME USED IS 0.1 SEC.
$COM SUB5,1,SUB5
         1*        SUBROUTINE SUB5(N)
         2*        WRITE (1,*)   'SUBROUTINE' ,N, ' CALLED'
         3*        END


3 LINES COMPILED . OCTAL SIZE=    56
CPU-TIME USED IS 0.1 SEC.
$COM SUB6,1,SUB6
         1*        SUBROUTINE SUB6(N)
         2*        WRITE (1,*)   'SUBROUTINE' ,N, ' CALLED'
         3*        END


3 LINES COMPILED . OCTAL SIZE=    56
```

```
CPU-TIME USED IS 0.1 SEC.
$EX
@NRL
RELOCATING LOADER LDR-1935H
*PROG-FILE MAIN
*OVERLAY-GENERATION 7
*LOAD MAIN,FTNLIBR
FREE: 013561-177777
*OVERLAY-ENTRY (1) SUB1
*LOAD SUB1
OVERLAY 1 LEVEL 1 COMPLETED. AREA: 013361-013703
   SUB1=013561
*OVERLAY-ENTRY (2) SUB2
*LOAD SUB2
OVERLAY 2 LEVEL 2 COMPLETED. AREA: 013704-014063
   SUB2=013704
*OVERLAY-ENTRY (3) SUB3
*LOAD SUB3
OVERLAY 3 LEVEL 3 COMPLETED. AREA: 014064-014140
   SUB3=014064
*OVERLAY-ENTRY (3) SUB4
*LOAD SUB4
OVERLAY 4 LEVEL 3 COMPLETED. AREA: 014064-014140
   SUB4=014064
*OVERLAY-ENTRY (2) SUB5
*LOAD SUB5
OVERLAY 5 LEVEL 2 COMPLETED. AREA: 013704-013760
   SUB5=013704
*OVERLAY-ENTRY (1) SUB6
*LOAD SUB6
OVERLAY 6 LEVEL 1 COMPLETED. AREA: 013561-013635
   SUB6=013561
*EXIT
@MAIN

START MAIN
SUBROUTINE          1 CALLED
SUBROUTINE          2 CALLED
SUBROUTINE          3 CALLED
SUBROUTINE          3 CALLED
SUBROUTINE          4 CALLED
SUBROUTINE          4 CALLED
SUBROUTINE          5 CALLED
SUBROUTINE          6 CALLED
END MAIN
```

1.3 THE OPEN COMMAND

The loader OPEN command (used only with FORTRAN) provides the user with the ability to open a file and connect it to a chosen unit number when the OPEN statement is omitted from his program. This command has the form:

*OPEN <file name><decimal unit no.><access>

where

file name — is a 1 to 16 character file or device name acceptable to the SINTRAN III file system. The default file type is SYMB.

decimal unit no. — is a number in the range 1-99 chosen by the user and which may appear in his I/O statements.

access — is one of the following:

SEQUENTIAL

DIRECT

SPECIAL (ND FORTRAN only)

and this determines the access method for the connection of the file. The default is SEQUENTIAL. The first two should be used if the file is to be accessed through FORTRAN READ/WRITE statements. SPECIAL should be used when the FORTRAN monitor calls RFILE, WFILE or MAGTP are employed. In ND FORTRAN the following values are also acceptable:

W	Sequential output (WRITE statements)
R	Sequential input (READ statements)
WX	Random input or output (for RFILE/WFILE use)
RX	Random input (for RFILE use)
RW	Sequential input or output (READ/WRITE statements)
WA	Sequential output appending to an existing file (WRITE statements)
WC	Random input or output to contiguous files (for RFILE/WFILE use)
RC	Random input from contiguous files (for RFILE use)

Note: This command may only be applied after the FORTRAN library has been loaded.

1.4 TWO-BANK PROGRAMS

To overcome address space constraints in the ND-100, a two-bank system can be utilised where the compiler (PLANC, COBOL, FORTRAN-77) is capable of generating separate output for the code and for the data part. For each program, the address space is limited to 64K. A two-bank program uses a seperate address space for code and data, thus it may have 64K words of code and 64K words of data.

1.4.1 Two-bank Systems versus One-bank Systems

Since the ND-100 is capable of addressing data through an alternative page table, programs may in principle consist of 64K code and 64K data. Programs where code and data are separated in this way are called "two-bank" programs, whereas "normal" programs (i.e., those whose code and data share a single address space of 64K) are called "one-bank" programs.

Two-bank object programs may be generated by an option in the various compilers and can be loaded by NRL. The following should be noted:

- Two-bank programs must be linked with the two-bank version of the appropriate run-time/library system. The naming conventions are: language name - 1BANK/2BANK, e.g. PLANC-1BANK, FORTRAN-2BANK, COBOL-2BANK etc.
- Care must be taken when linking assembly or NPL routines with two-bank systems.
- The code parts of two-bank systems are in principle completely read-only.
- Overlay tree structures are still available, and both code and data parts are brought in when a link is required.

1.4.2 Two-bank Loading

Two BRF control numbers, PMO and DMO, are used to put the loader into program or data mode. IMAGE-FILE or PROG-FILE should be used when building two-bank program systems, and they must be linked together with the two-bank, run-time versions.

Progams compiled in two-bank mode are by default loaded into two 64K banks, but may optionally be loaded into one 64K address space. In the former case, the program system executes with all data access via the alternate page table. In the latter case, execution is the same as for programs compiled in the default one-bank mode except that the user has explicit control for indicating where the code (read-only) and the data (modifiable) is to be placed in memory. This control is obtained with the command:

```
*SET-DATA-LOAD-ADDRESS <address>
```

The data, which would normally be loaded into the 64k data bank, will now be placed at the specified address and upwards from it. (The one-bank version of the run-time system must be used.)

All loader commands (SET-LOAD-ADDRESS, DEPOSIT etc.) will apply to either the code or the data bank so long as the loader is first put into the appropriate mode using the command:

```
*SET-MODE <mode>
```

where <mode> is either PROG (the initial setting) or DATA. The current mode will be printed on the terminal if no argument is supplied to the command. SET-MODE DATA must be issued before the OV-GEN command for two-bank overlaid programs.

Note: One-bank and two-bank programs may not be mixed. The BPUN command does not apply to two-bank systems.

The following applies until the two-bank RECOVER command is implemented in the operating system:

Recovery of a two-bank system starts by reading the data bank contents from the PROG-file into the alternate memory bank, and so for this reason, the name as specified in the PROG-FILE or DUMP command, is stored in the program file. If the file is to reside in user SYSTEM, or will always be accessed by the user who dumps it, then no user name is required. Otherwise the user name should be specified so that the correct file can be found by RECOVER to load the data bank.

Where the extent of the code plus data is less than 64K, a copy of the data may be kept beyond the code part so as to avoid access to the PROG-file. The data is instead initialized by moving it from one bank to the other. The command:

```
*DATA-BANK-COPY
```

duplicates the data area in the PROG segment above the code. (Segments in the RT-system are described in the SINTRAN III Real-Time Guide (ND-60.133) and in the latest version of the RT-Loader (ND-60.051)). This command should be given before the command DUMP or EXIT.

2 BINARY RELOCATABLE FORMAT

The output from the language processors (compilers,assemblers) is in the form of relocatable modules which means that they have not been assigned a fixed position in memory. This is possible since the code is in binary relocatable format (BRF) in which information about intermodule references such as procedure calls, references to global data etc., will have been coded in symbolic form. The symbols, usually called labels, are alphanumeric names assigned by the compiler to an instruction or to a data item. Relocatability is maintained until these labels are transformed into machine addresses by the Nord Relocating Loader.

2.1 STRUCTURE OF BRF

BRF code is organized in eight-bit bytes and is not bound to any particular data medium (magnetic tape, disc, etc.). The information contained in the object program may be classified as:

- Control information which is held in a control byte and is interpreted as a loader command.
- Programmed information which is held in two bytes containing a sixteen-bit word and is termed a P-group.
- Symbolic information which is held in four (six in FORTRAN,COBOL, etc.) bytes termed an S-group containing a symbol of one to seven six-bit characters.

For further information see the MAC USERS GUIDE.

A BRF group is defined to be:

```
          <control byte>
or        <control byte><P-group><P-group>
or        <control byte><S-group>
or        <control byte><S-group><P-group>
```

BRF is a sequence of BRF-groups.

A program is a set of instructions and data which, when it is interpreted, will perform an algorithm. A program may be in various forms. It may be written in FORTRAN, assembly code, machine code, etc. By means of special programs (i.e., compilers, assemblers, loaders and so forth) the progam may be changed from one form to another. Conceptually, however, we will regard the program to be of the same type before and after the tansformation. We say that a program is written in relocatable format or, more briefly, that the progam is relocatable, if it does not have to occupy a specific memory position

Thus a FORTRAN program and an assembly progam (with only symbolic addresses) are relocatable programs, while a program in binary form is generally not relocatable. See the three following programs:

Example 1	Example 2	Example 3
Program PER written in assembly code	Program PER written in binary form (placed in location 10)	Program PER written in binary form (placed in location 20)
PER, JMP I *+1	125001	125001

```
      OLE            14              24
      157           157             157
      751           751             751
OLE,  WAIT       151000          151000
```

The binary-form program in example 2 is bound to location 10 and cannot be moved to location 20 without changes. As we see, the machine code is not in relocatable format, since there is no information about which words contain addresses (internal addresses) that have to be modified depending on the placement of the program. In BRF, this information is placed in the control byte. The program PER will, in BRF, look like the figure below.

17	1	125001	2	4	1	157	1	751	1	151000	21	100575

Fig. 1. Example of BRF

More specifically, we organize the BRF groups by columns (see examples 2 and 3).

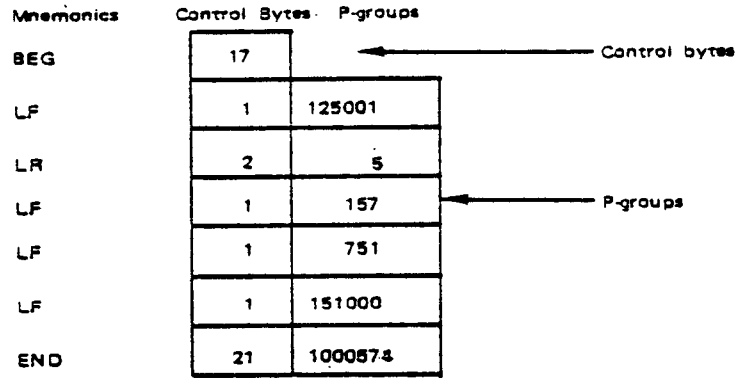


Fig. 2. Example of BRF

The contents of the control byte will form the control number. Control number 17 (mnemonic BEG) marks the beginning of the program. In FORTRAN the 17 (BEG) is followed by 32 (LONGF) which indicates that all S-goups contain six bytes instead of four. Control number 1 (LF) means that the corresponding P-group will be loaded unmodified, while control number 2 (LR) means that the corresponding P-group contains an address, which is given relative to the beginning of the program. Control number 21 (END) is followed by a checksum.

Statement numbers (labels) in FORTRAN and BASIC are represented by S-groups where the two first and the two last bytes are zero. The third and fourth byte contain the numerical label value.

2.2 RELOCATION OF INTERNAL ADDRESSES

Suppose the loader has filled memory up to location 621 and is going to load the object program shown in Figure 2.

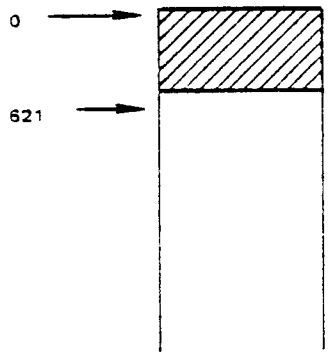


Fig. 3. Memory Image Before Loading

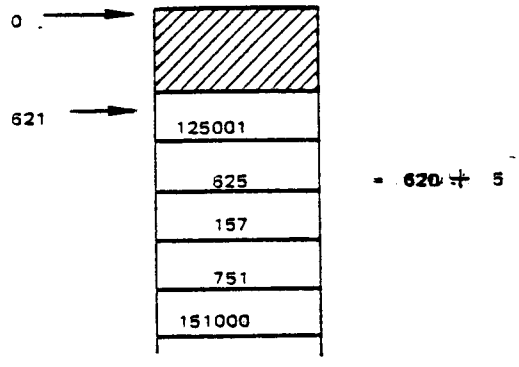


Fig. 4. Memory Image After Loading

When the loader reads control number 17 (BEG), the current location-1, in this case 620, is taken as the program's first address (the so-called "program-base"). This program-base is added to those P-groups which are preceded by the control number 2 (LR). The result is shown in fig.4.

2.3 PROGRAM UNITS

A program is composed of one main program and one or more subprograms. Those subprograms which are part of the system are called library subprograms and are available for users. A common name for main programs and subprograms is program unit.

The address (or addresses) of a program unit where the execution begins is called the entry point. If the program unit is a main program, the entry point is called a start address. A word containing the address of an entry point (of another program unit) is termed an external reference.

2.4 SEPARATE COMPILING/ASSEMBLING

When a compiler compiles a program, each program unit is translated without any information about other program units. Therefore, the program units need not be compiled at the same time. This is termed separate compiling. Thus, the object program consists of one or more BRF program units. The information necessary to link these together to an executable program, namely the entry points and the external references, is symbolic, and is placed in the S-groups. The meaning of the S-group is determined by the preceding control number in the following way:

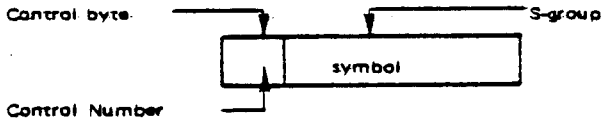


Fig. 5. S-group with Control Number

Control Number	Mnemonic	Meaning
14	MAIN	Symbolic start address
15	LIBR	Library subprogram entry point
16	ENTR	Symbolic entry point
20	REF	Symbolic external reference

The object program units begin with control number 17 (BEG), end with control number 21 (END) and may contain at least one of the control numbers 14 (MAIN) or 16 (ENTR). A library subprogram has a LIBR group at the beginning of the program unit. Only the necessary library subprogams are loaded when the LIBR symbol has been referenced by a REF group and is not already defined as a symbolic entry point. If not needed, the object program is only check-read to the END group, without losing control of the BRF syntax.

If the loader does not receive any other information, the program units are loaded consecutively, starting at a system-defined address. However, the program units may be loaded elsewhere by means of the control numbers.

10 (SFL) Start (continue) loading at the location in the P-group.

11 (AFL) Continue at the current location + the relative

address in the P-groups.

12 (SRL) Continue at the current program base + the relative address in the P-group.

The main program and the subprograms may be read in an arbitrary sequence, i.e., if a program unit "A" makes reference to another program unit "B" it does not matter which of them is loaded first. The (necessary) library subprograms are loaded last. If a library subprogram A makes reference to another library subprogram B, then A must appear first.

2.5 LINKING OF PROGRAM UNITS

The loader has a symbol table where each entry consists of three words for the symbol (the S-group) and one word (ADR) for the address.

ADR may have different meanings; if a symbol is not in the table, then formally ADR = 0. If a symbolic entry point has been read, then ADR is the memory address of the entry point. If only symbolic external references to a symbol have been read, then the ADR is a pointer to the last location at which the symbol was referenced. This location contains a pointer to the preceding reference to the same symbol. The first reference location contains the word 177777 (octal) to mark the end of this list. One bit in the table entry is necessary to discriminate between the two interpretations of ADR.

The link structure of referenced symbols (not defined) may be visualised as in the figure below.

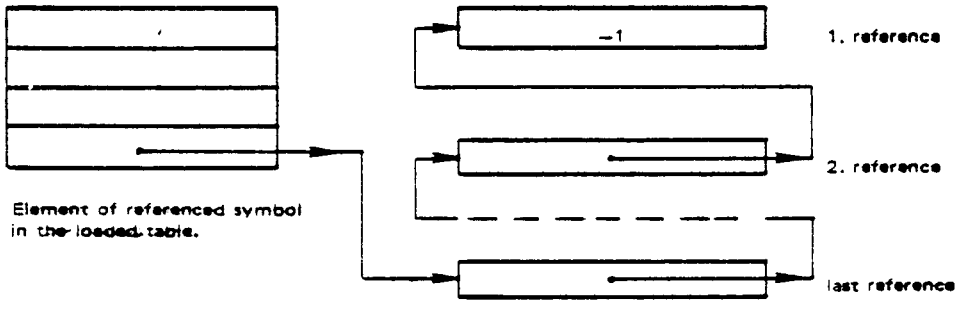


Fig. 6. Symbol Reference Link

2.6 COMMON BLOCKS

The memory area in which the loader puts the program is a continuous area from a lower address up to the upper bound. The program units therefore, normally grow upwards. For one-bank programs, COMMON blocks are allocated from the upper bound downwards. Thus the COMMON block address is found by subtracting the length from the upper bound and reducing the upper bound appropriately.

For two-bank programs, COMMON blocks are allocated like all other data areas, i.e., from the present data load address upwards.

The COMMON block address must be known before the addresses referencing COMMON are loaded. Therefore the COMMON block address which uniquely specifies the maximum block length, is defined by the first program unit using COMMON data. This is the explanation of the restriction that a COMMON block cannot be expanded by the succeeding program units.

Data in COMMON is referenced by indirect addressing. Such addresses are followed by the control number 27 (ADS) which tells the loader to add the COMMON block address.

The COMMON block lengths cannot be expanded. The ASF group has the format:

<ASF><S-group><P-group>

where the S-group contains the name of the COMMON block, and the P-group contains the block length. Thus, if the COMMON blocks A,B, and C are declared in the object program in that order, the allocation of the blocks would be as in the figure below.

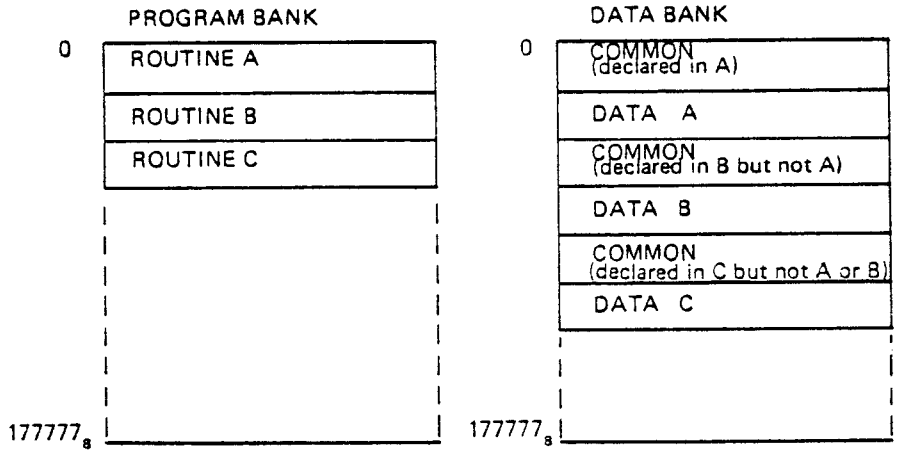


Fig. 7. Two-bank COMMON Blocks

The ADS-group has the format:

<ADS><S-group>

with the interpretation that the value of the S-group is added to the previously loaded address (P-group).

2.7 CHECKSUM

In order to detect read errors during loading, a checksum is placed behind each END control byte. Here, everything from the BEG control byte to the END control byte is added together, complemented and put in a P-group. The control bytes are regarded as eight bits, the P-group as sixteen bits, and the S-group as two or three sixteen bit numbers. (In fig. 2 all the numbers are given as octal numbers.)

2.8 FIX-UP FACILITY

BRF and the loader are designed to allow single-pass, sequential compiling as discussed in Section 2.1. This implies that the loader is able to fix words which have already been loaded. This is done by the four control numbers 4 (AFF), 5 (ARF), 6 (AFR), 7 (ARR) which all have two P-groups. The second P-group contains an address, and the first P-group has contents which will be added to the address. Both the address and the contents of the first P-group (which may be an address) may be relocated with the program base, and thus gives four possibilities.

2.9 DESCRIPTION OF THE BRF-CONTROL NUMBERS

The legal control numbers are sequential numbers starting at zero and are interpreted as commands to the loader. They are listed in the following table together with their mnemonics and their interpretation. The terminology needs some explanation.

CLC is the current location counter. It contains the address where the next word is to be placed. PB is the program base of the current program unit. CDB is the COMMON data base (COMMON block address). W1 and Wn are the contents of the first to the n'th P-group, respectively.

If "a" is an address or an address expression, then (a) is the content of this address. The expression X -> (Y) means that the value of X will replace the contents of Y.

Control Number (octal)	Mnemonic	No. of Words	Interpretation
0	FEED	0	Neglect
1	LF	1	W1->((CLC)),(CLC)+1->(CLC)
2	LR	1	W1+(PB)->((CLC)),(CLC)+1->(CLC)
3	LC	1	W1+(CDB)->((CLC)),(CLC)+1->(CLC)
4	AFF	2	W1+(W2)->(W2)
5	ARF	2	W1+(PB)+(W2)->(W2)
6	AFR	2	W1+(W2+(PB))->(W2+(PB))
7	ARR	2	W1+(PB)+(W2+(PB))->(W2+(PB))
10	SFL	1	W1->(CLC)
11	AFL	1	W1+(CLC)->(CLC), fill zeros
12	SRL	1	W1+(PB)->(CLC)
13		-	Not Used
14	MAIN	2(3)	Symbol in S-group will become the main entry
15	LIBR	2(3)	Conditional loading
16	ENTR	2(3)	Symbol in the S-group is assigned value of CLC
17	BEG	0	(CLC)->(PB) First control byte of a unit

20	REF	2(3)	Symbol in S-group is referenced in CLC
21	END	1	W1 contains the BRF-checksum
22	INHB	0	Warns that compilation errors have occurred
23	EOF	0	End of loading
24	LNF	1+W1	W1,W2,...,WwO>->(CLC),...,(CLC+WO-1)
25	RT	1	W1 contains real-time priority
26	ASF	3(4)	<symbol><number> Defines common length. Value of symbol in loader table = common start address.
27	ADS	2(3)	<symbol>+(CLC-1)->(CLC-1) Adds common address
30		-	Not used
31		-	Not used
32	LONGF	0	Flags six bytes S-group
33		-	Not used
34	INL	2	W2->(W1+(PB))
35	DBL	3	Wi->(W1+(PB)+i-2) (i = 2 to 3)
36	RLL	4	Wi->(W1+(PB)+i-2) (i = 2 to 4)
37	CXL	7	Wi->(W1+(PB)+i-2) (i = 2 to 7)
40	INC	4(5)	W5->(W4 + ADR)
41	DBC	5(6)	Wi->(W4 + ADR + i-5) (i = 5 to 6)
42	RLC	6(7)	Wi->(W4 + ADR + i-5) (i = 5 to 7)
43	CXC	9(10)	Wi->(W4 + ADR + i-5) (i = 5 to 10)
44	BYL	2	W2(bit 0-7)->(W1+(PB))(bit 0-7) if W2 bit 15=0 W2(bit 0-7)->(W1+(PB))(bit 8-15) if W2 bit 15=1
45	BYC	5	W5(bit 0-7)->(W4 + ADR)(bit 0-7) if W5 bit 15=0 W5(bit 0-7)->(W4 + ADR)(bit 8-15) if W5 bit 15=1
46	NWL	1	W1 contains line number. (Not in use.)
47	DBG	0	Debug mode on/off
50	PMO	0	Program bank mode

51	DMO	0	Data bank mode
52	LRP	1	Same as LR but PB of program bank
53	LRD	1	Same as LR but PB of data bank
54	DIC	-	Dictionary table follows. Each element contains name (3 words) and byte pointer (2 words). End of table marked by -1.

With reference to the control numbers 40,41,42,43, and 45, the W1, W2, and W3 contain a common block name. At load time the symbol must be defined; its value is referred to as ADR.

3 THE BRF EDITOR

The BRF Editor is a SINTRAN subsytem for handling files containing BRF code (output from compilers, the MAC assembler, etc.). The code itself was described in chapter 2; the files and the information held on them can be manipulated by the commands described in this chapter. Thus one can perform such operations as combining files, modifying libraries, etc. Note the following:

- The first BRF unit number in a file has the unit number of one. The unit number must be specified in decimal.
- The BRF Editor will check all units for syntax errors and checksum error.
- The current location counter is the address at which the code on the BRF file will be placed at load time.
- The default values for "first unit number" and "last unit number" are the first BRF unit on the file and the last BRF unit on the file respectively. All files used as parameters (except "output file") have the default type :BRF.
- The units in commands can be identified by any of the names found in the units MAIN or ENTR codes.

3.1 Symbol Handling - Basic

The command:

*LIST-ENTRIES <input file> <output file>

will list all ENTRY, MAIN and LIBR symbols found on the input file onto the output file. The output will appear in the order: BRF unit number, symbol name, and symbol type (ENTR, MAIN or LIBR).

*APPEND-FILE <source file> <destination file>

The BRF units in the source file will be appended to the destination file after the last BRF unit in the destination file. The BRF control number 23 octal (EOF) will be placed at the end of the destination file.

*APPEND-UNIT <source file> <destination file> <unit>

The BRF units in the source file will be inserted in the destination file after the unit identified by <unit>.

*FETCH-UNITS <source file> <destination file> (<first unit>) (<last unit>)

The BRF units in the source file, specified by the parameters first

and last unit together with those units occurring between them, will be appended to the destination file following the last BRF unit which appears in it.

*DELETE-UNITS <file> (<first unit>) (<last unit>)

The specified BRF units will be deleted from the file. <first unit> will be the first unit deleted, then all BRF units following, including <last unit> will be deleted.

3.2 Commands for Updating

*EXCHANGE-UNITS <source file> <destination file>

Those BRF units in the destination file which have the same identification as those in the source file, will be replaced by the BRF units in the source file.

The various BRF units in the destination file will ahve the same relative position within the file after the EXCHANGE-UNITS command is given as they had before it.

BRF units in the source file which are not found in the destination file, will be skipped and a warning message will be issued, after which the exchanging will continue.

BRF units without symbolic identification cannot be replaced.

*WRITE-EOF-AFTER-UNIT (<unit>) <destination file>

The control byte 23 octal (EOF) will be inserted after the specified unit in the destination file. The default value for the unit is zero, i.e., the EOF byte would be written as the first byte on the file.

3.3 Additional Symbol Commands

*MAKE-LIBRARY-UNITS <file>

A BRF control byte LIBR will be inserted at the beginning of each BRF unit in the file, assuming that the unit does not already have one. The first ENTR symbol in a unit will be the LIBR symbol in the unit. Only one LIBR is inserted in each unit.

*MAKE-LIBRARY-FILE <source file> <destination file>

This command will copy the source file to the destination file and insert a BRF unit containing a dictionary table of all the BRF units.

The dictionary table is the first BRF unit in the destination file. Each element in the dictionary table consists of five words, 3 words for the unit name and 2 words for the byte pointer of the unit. Selective loading (search for referenced library units) from a file with a dictionary table will be faster than loading the same file without a dictionary table.

*RENAME-SYMBOL <old symbol> <new symbol>

This command,together with the CHANGE-FILE command, may be used to change the names of symbols in a BRF-code file. <old symbol> is the current name of the symbol while <new symbol> specifies the new one. The symbols (which may be ENTR, REF, COMMON, MAIN, or LIBR) which have been redefined are actually changed when the CHANGE-FILE command is given. To reset any of the specified symbols he CLEAR-TABLES command should be used.

*CHANGE-FILE <file>

All the symbols on the file <file> which have been the subject of a RENAME-SYMBOL command will now be changed.

*CLEAR-TABLES

This command will reset all outstanding RENAME-SYMBOL commands.

3.4 Other Functions

*LIST-BRF <input file> (<first unit>) (<last unit>) (<output file>)

All the BRF information regarding the <first unit> and all the other units up to and including <last unit> on the <input file> will be listed on the <output file>. The information given is as follows:

1. BRF control number (octal)
2. Name of the BRF control number
3. All symbolic names (REF, ENTR, LIBR, MAIN, ASF, ADS etc.)

Note the following:

All "binary" information will be written as octal numbers and such information belonging to the BRF control numbers 1 and 24 (octal) will in addition be "disassembled", i.e., listed as MAC assembly code.
If the command SET-CLC (described below) is given, then the value of the current

location counter will be written first on
on each line.

*SET-CLC (<value>)

This has the effect of writing the value of the current location counter on each line when using the LIST-BRF command. <value> specifies the first value of the current location counter and must be given in octal form. Its default value is zero.

*RESET-CLC

This command will discontinue the appearance of the current location counter when using the LIST-BRF command.

*RESET

The BRF Editor will be reset.

*HELP

A list of the available commands will be obtained together with their required parameters.

*EXIT

Control leaves the BRF Editor and returns to the operating system.

A P P E N D I X A

LOADER COMMAND SUMMARY

The loader is controlled from the terminal by the set of commands listed below. The command words may be abbreviated and the parameters (if any) are separated by a space or a comma.

*ASCII-DUMP <lower address><upper address>[<file name>]

The contents of the locations betwen the upper and lower addresses will be dumped on the specified file, eight consecutive locations (16 characters) to a line. Non-printable characters appear as spaces. If no file name is specified the characters are dumped on the terminal.

*AUTOMATIC
<library file 1>
.
.
.
.
<library file n>

The specified library files will be loaded when the RUN, DUMP and BPUN commands are used, if undefined references exist in the loader table. The loading from the libraries will terminate when all references are defined or when the library files have been scanned twice. If this results in the necessary definitions, the specified command will be performed, otherwise an error message will be written.

Example:

*AUTO
FTNLIBR
USER-LIBRARY
.
*

The command lines are terminated by a dot (.).

The pre-automatic mode buffer is not cleared by the RESET comand, and thus the loader may be initiated and dumped for later recovery with the automatic sequence intact. The buffer may be cleared by typing:

*AUTO
.

*BOUNDARIES <lower address><upper address>

This command is used to specify the dump area in connection with the BPUN and DUMP commands.

*BPUN <destination file name><start addr><bootstrap addr>

The program area (default or specified by the BOUNDARIES command) will be dumped in absolute binary form on the destination file preceded by an octal coded bootstrap. The main start entry of the program may be specified symbolically or in octal form. The bootstrap address (octal number) specifies where the bootstrap program (44 octal locations) will be located if the program is loaded into a stand-alone NORD-10/ND-100. Default destination type is BPUN. Default boundaries range from the lowest to the highest address accessed by the loader since the last recovery.

*DATA-BANK-COPY

This command will duplicate the data area which is beyond the code part in a PROG segment. The copy so formed may be kept in order to avoid access to the PROG-file.

*DEFINE <symbol><octal value>

The symbol will be entered into the loader table. Its value will be equal to the octal number specified.

Example:

*DEF XXX 777

*DEPOSIT <octal address> [<new contents>]

*DEPOSIT <symbol name> [,<x octal displacement>] [,<new contents>]

The new contents are put into the octal address specified or into the address of the symbol name plus or minus the displacement. If the last parameter is missing the old contents are displayed as two ASCII characters and as an octal number. They may be changed by typing the new contents on the same line. By typing CR the next location will be displayed automatically. The termination character is a dot (.).

*DUMP <destination file name>[<start address><restart address>]

This command saves the loaded program on the specified file. The program may be retrieved with the RECOVER command, when it commences at the specified start address. The restart address specifies where the program should be started with the CONTINUE command. The dump limits may be set by the BOUNDARIES command. Default boundaries range from the lowest to the highest address accessed by the loader since the last recovery. The main entry will act as default start and restart addresses.

*ENTRIES-DEFINED [<file name>]

All symbols (defined) present in the loader table will be printed on the terminal. In addition the current and the upper bound are displayed with the following format:

FREE: <current location><upper bound>

Default file name is the terminal.

Example:

E-D
XXX = 017777
FREE: 020000-17777

*ENTRIES-UNDEFINED [<file name>]

This command is similar to ENTRIES-DEFINED. However, only undefined symbols are printed. The default file name is the terminal.

*EXIT

Control is returned to the operating system.

*FIX

The current contents of the loader table are fixed (will not be removed by RESET) and the current location will later act as the lower bound reset address. The fixed entries do not appear in any listing of the entries.

*HELP

List the available loader commands on the terminal.

*IMAGE-FILE <file name>

The BRF information will be loaded into the file specified instead of directly into main memory. The default file type is IMAG.

Example:

IM-FI MIRROR

*KILL <symbol>

If present, this symbol will be removed from the loader table.

Example:

*KILL BUG

*LOAD <file name>[<file name>...]

The file(s) specified will be loaded until the end-of-file is encountered. The default file type is BRF.

Example:

LOAD SUB1, SUB2

*OCTAL-DUMP <lower address><upper address> [<file name>]

The contents of the locations between the lower and upper addresses will be dumped on the specified file, eight consecutive locations to a line. If no file name is specified the contents are dumped at the terminal.

Example:

*OCTAL-DUMP 0 3

000000: 000000 000000 000000 000000

*ON-ERROR-EXIT

When this command is given, the loader will give control to the operating system after the first error message. (This is useful for MODE and BATCH jobs.)

*OPEN <file name><decimal unit no.><access>

Note: This command may only be applied when the Fortran library is loaded.

where

file name is a 1 to 16 character file or device name

acceptable to the SINTRAN III file system. The default file type is SYMB.

decimal unit no. is a number in the range 1-99 chosen by the user and which may appear in his I/O statements.

access is one of the following:

SEQUENTIAL

DIRECT

SPECIAL (ND FORTRAN only)

and this determines the access method for the connection of the file. The default is SEQUENTIAL. The first two should be used if the file is to be accessed through FORTRAN READ/WRITE statements. SPECIAL should be used when the FORTRAN monitor calls RFILE, WFILE or MAGTP are employed. In ND FORTRAN the following values are also acceptable:

W	Sequential output (WRITE statements)
R	Sequential input (READ statements)
WX	Random input or output (for RFILE/WFILE use)
RX	Random input (for RFILE use)
RW	Sequential input or output (READ/WRITE statements)
WA	Sequential output appending to an existing file (WRITE statements)
WC	random input or output to contiguous files (for RFILE/WFILE use)
RC	Random input from contiguous files (for RFILE use)

Note: This command may only be applied when the FORTRAN library is loaded.

*OVERLAY-ENTRY [(<level>)] <entry name 1>[,... ,<entry name n>]

This command specifies that the next overlay link is to be generated. <level> is the overlay level and the default value is 1. <entry name 1> to <entry name n> are the names of the subprograms called with the OVERLAY and/or OVRECAL routines from the previous level. The root link is level 0.

*OVERLAY-GENERATION [<no. of overlay entries>]

The above command specifies that a multi-overlay system is to be generated. The parameter is the total number of overlay entries given in the OVERLAY-ENTRY commands. The default value is 128.

Note: The root link must be completed by loading the FORTRAN Library.

*PROG-FILE <file name>

The BRF information will be loaded onto the file specified instead of directly into main memory. The default file type is PROG.

*REFERENCE <symbol>[<octal address>]

1) If the symbol is not present in the loader table a -1 will be put into the specified address and this address will be referenced in the table. The specified octal address must be an unused memory address. If no address is given then the symbol will be treated as a referenced symbol only. It is impossible to reference an undefined symbol in 177777 (octal).

2) If the symbol is present but already referenced (undefined) the address specified will be linked into the reference chain.

3) If the symbol is defined, its value will be put into the address specified.

An example follows on how to load the routines SUB1 and SUB3 from the file LIBSUB compiled in library mode:

```
*E-D
FREE: 013665-177777
*REF SUB1,13665
*REF SUB3,13666
*SET-LOAD-ADDRESS 13667
*LOAD LIBSUB
  .
  .
  .
```

*RENAME <old symbol name><new symbol name>

The old symbol name in the loader table will be replaced by the new one. The defined or not-defined state and the value are left unchanged.

*RESET

The loader variables and tables are initialized (symbols removed).

*RUN

The loaded program will be started at its main entry (defined by control byte 14).

*SET-COMMON-ADDRESS <label>,<address>,<size>

A common block with the name <label> is defined at the specified address and with the specified size. The user should make sure that this area is not overlapped by code or data.

*SET-DATA-LOAD-ADDRESS <address>

This command is used for two-bank loading. The data, which would normally be loaded into the 64K data bank, will now be placed at the specified address and upwards from it. (The one-bank version of the run-time system must be used.)

*SET-IO-BUFFERS <no. of 1K buffers>

This command will specify buffer space for the NORD-10 FORTRAN buffered I/O. The parameter is the number of buffers of 2K bytes (= 2000 octal words) each. The number of buffers is limited to a maximum of eight. The buffer space is reserved in the user's program area by the loader. This command is only for FORTRAN programs loaded together with the NORD-10 FORTRAN Library Run-time System. This command must be given before the FTN library is loaded.

EXAMPLE:

*SET-IO-BUFFERS 3

*SET-LOAD-ADDRESS <octal address>

Subsequent loading will start from the address specified.

*SET-MODE <mode>

This is a command for two-bank loading where <mode> is either PROG (the initial setting) or data. All loader commands (SET-LOAD-ADDRESS, DEPOSIT etc.) will now apply to either the code or the data according to the mode specified. If no argument is given the current mode will be taken.

*SIZE <octal number>

If the message LODER-TABLE OVERFLOW is given the loader table may be expanded by this command. The octal number specifies the number of entries in the table. Old table contents are lost. The default size is 300 octal entries. With this command the loader will be reset automatically. The commands PROG-FILE and IMAGE-FILE will change the default size to 2000 octal entries.

*UPPER-LIMIT <octal address>

The load area upper limit is set to the value specified.

*VALUE <symbol>

If defined, the value of the symbol specified will be printed on the terminal.

Example:

*VAL NAME
000777

*X-LOAD <file name1>[<file name2>....]

Exclusive load. Library sequences headed with defined symbols are skipped while all other units on the specified file(s) will be loaded until end-of-file is encountered. The default type is BRF. This command is somewhat special and is used for system generation.

Example:

X-LOAD LIBRA

A P P E N D I X B

THE LOADER ERROR MESSAGES

AMBIGUOUS

The last command word has been abbreviated so that ambiguity results.

AT UPPER LIMIT

The current load address has reached the absolute upper limit or the beginning of the common area.

AUTO-BUFFER FULL

No more space for AUTOMATIC commands.

BRF-CHECKSUM ERROR

The BRF-file contents have been corrupted as a result of hardware or software errors occurring during reading or writing.

COMAND-BUFFER FULL

Too many characters in a loader command. The maximum no. is 64.

COMMON BLOCK EXCEEDS AVAILABLE MEMORY

The specified block is too large.

COMMON BLOCK EXPANDED

The length of an already-defined common block has been declared to be larger in a subsequently-loaded program.

DOUBLY DEFINED

The symbol being defined (either by loading a file or by the DEFINE command) has already been assigned a value.

ILLEGAL OVERLAY LEVEL

The overlay level should be 1 or increased by 1.

ILL-BRF CONTROL NO.

Non-interpretable information has appeared on the BRF file due to hardware or software errors.

ILL-FILE NO.

The specified file no. in the OPEN command must be in the range 1-99.

INSUFFICIENT PROGRAM

Error diagnostics have occurred during the compilation process.

LINKED FROM ILLEGAL OVERLAY LEVEL

Forward references should only be made to the next level.

LOADER-TABLE OVERFLOW

The loader symbol table is full.

NO IMAGE-FILE/PROG-FILE SPECIFIED

The command IMAGE-FILE or PROG-FILE must be used when creating Overlay or Multi-segment systems, or when the routines are compiled with the compiler commands DEBUG-MODE ON and/or SEPARATE-DATA ON.

NO MAIN ENTRY

The user is trying to start a program having no main module.

OPEN-CONNECT TABLE MISSING

The FORTRAN run-time system library must be loaded prior to this command.

OVERLAY SEGMENT-TABLE OVERFLOW

Too many overlay segments. The table size can be expanded by increasing the maximum number of overlay entries in the command OVERLAY-GENERATION.

SET-MODE DATA NOT SPECIFIED BEFORE OVERLAY-GENERATION

When creating an overlay system of two-bank compiled routines, the command SET-MODE DATA must be given before the command OVERLAY-GENERATION.

UNDEFINED REFERENCES ON OVERLAY

A new overlay link is specified before the previous link has benn completed.

UNDEFINED SYSTEM/LIBRARY ENTRIES ON ROOT SEGMENT

The appropriate language library must be loaded into the root link.

-WARNING- MIXED ONE/TWO-BANK ROUTINES

It is not allowed to mix routines compiled with the compiler command SEPARATE-DATA OFF with routines compiled with SEPARATE-DATA ON. There is an exception in the case of routines written MAC and NPL.

Note: In addition to the messages listed above, some of the file-system diagnostics may appear on your terminal.

A P P E N D I X C

INDEX

********** SEND US YOUR COMMENTS!!! **********

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you
- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

********** HELP YOURSELF BY HELPING US!! **********

Manual name: ND Relocating Loader

Manual number: ND-60.066.04

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

Do you have suggestions for improving this manual ? _____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for ? _____

_____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

Answered by _____ Date _____

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway