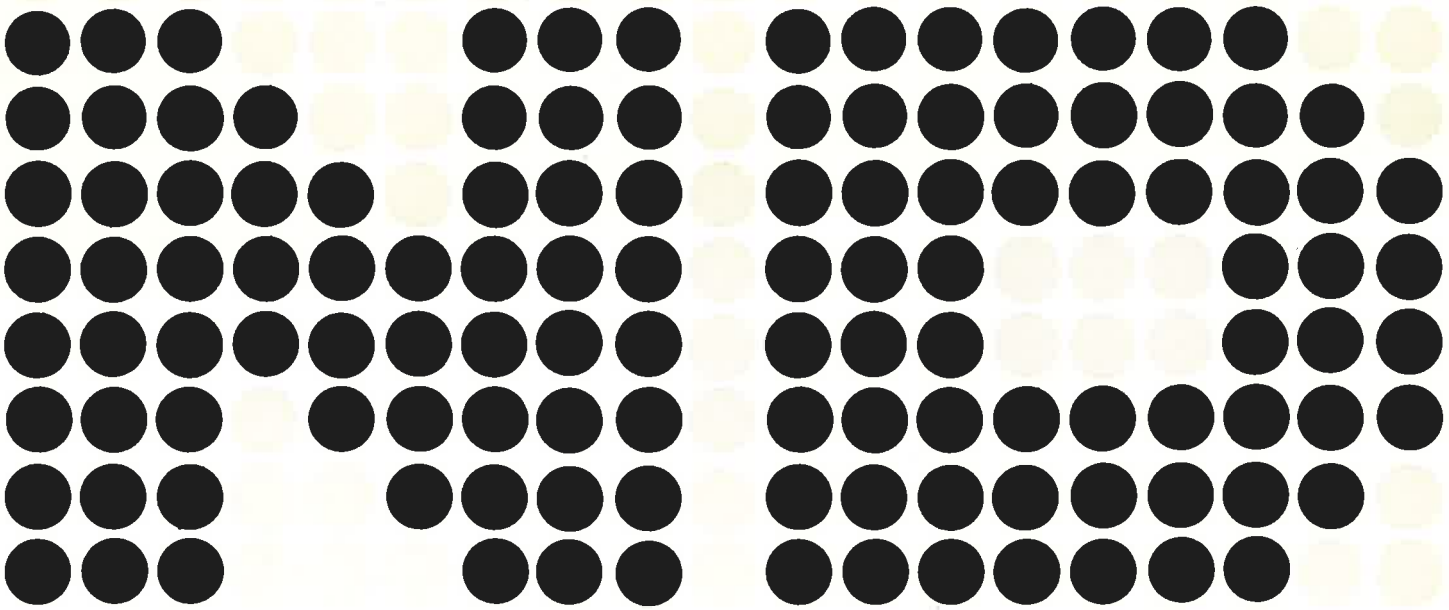
NORD-10

Relocating Loader

A/S NORSK DATA-ELEKTRONIKK



NORD-10
Relocating Loader

REVISION RECORD

Revision	Notes
3/70	Original Printing
1/73	Second Edition
2/76	Total Revision, superceding all previous issues

Publication Number ND-60.030.03
February 1976

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf.: 21 73 71

TABLE OF CONTENTS

+++
+

The binary relocating loader is used to read BRF output from the MAC assembler and the NORD FORTRAN/BASIC compilers into memory and make the program(s) executable.

This manual describes the use of the NORD-10 Relocating Loader running under the operating system SINTRAN III.

1 GENERAL

Programs transformed (by an assembler or compiler) into Binary Relocatable Format (BRF) must be read and processed by a loader in order to be executed by the machine. Relocatable programs may be loaded anywhere in the memory according to default system addresses or according to load-addresses specified by the user.

By the load process, the BRF-programs are transformed into an absolute binary format and the quality of relocatability is lost.

There are three main types of loading:

- Basic Loading

This is the most common method of loading, whereby the program is loaded directly into the users memory space.

- Memory-Image Loading

The program is loaded onto a file where it resides in absolute binary form.

- Overlay-Segment Loading

Parts of the program are loaded into the same memory area and, when completed, they are written on different areas on a file.

1.1 *HOW TO LOAD AND EXECUTE A BRF-PROGRAM*

Loader input is obtained from one or more files/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 the control is transferred to the loader command processor (types *) which is then ready to accept another command. The bracket contents denote optional parameters.

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

*ENTRIES-DEFINED [<file name>]

which will give you a printout of the entry-names along with their octal addresses in memory. If no file/device name is specified, the printout will appear at your 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 you have loaded a FORTRAN program and some references still remain, the FORTRAN-runtime/library system-file should be loaded. If any of these routines are necessary for the execution they will be selected by the loader and connected with their corresponding references.

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

*RUN.

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

If you wish to leave the loader and enter the operating system you simply write:

*EXIT.

You may re-enter the loader by using the system command:

@CONTINUE.

1.2 *LOAD-ADDRESS CONTROL COMMANDS*

If you wish to load your program at an address other than the preset values you may obtain this by typing:

*SET-LOAD-ADDRESS <octal address>.

Subsequent loading will then be performed from the address specified.

Also, the absolute upper load limit may be redefined with:

*UPPER-LIMIT <octal address>.

Be certain that no overlapping may occur when manipulating load-addresses.

1.3 *COMMANDS AFFECTING THE SYMBOL-TABLE*

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

*DEFINE <entry name> <octal value/address>.

Symbol names may be renamed by:

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

and an entry is deleted by:

*KILL <symbol name>

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

*VALUE <symbol>

The loader then prints the octal number on the terminal.

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

*REFERENCE <symbol> <octal memory address>.

It doesn't matter if the referenced entry-point 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 you must repeat the load procedure beginning with an appropriate table length.

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 moment when the table was fixed.

Also, the current location when fixing will later act as the lower bound reset-address.

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

Fixed entries are not listed through the commands: ENTRIES-DEFINED and ENTRIES-UNDEFINED.

1.4 *SAVING AND DUMPING OF BINARY PROGRAMS*

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

*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. It then starts in 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.

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

The program-area (default or specified by the BOUNDARIES-command) will be dumped binary on the destination file with an octal coded bootstrap ahead. The main start entry of the program may be specified symbolically or octally. The bootstrap address (octal number) specifies where the bootstrap program (44_8 locations) will be located, if the program is loaded into a stand-alone NORD-10 or into the users SINTRAN III memory by the PLACE command. Default destination type: BPUN. Default boundaries range from the lowest to the highest address accessed by the loader since the last recovery.

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

*BOUNDARIES <lower address> <upper address>.

1.5 *AUXILIARY MEMORY EXAMINATION COMMANDS*

DEPOSIT <octal address> <new contents>

The new contents are put into the octal address specified. If the last parameter is missing the old contents are displayed and may be changed by typing the new contents on the same line. By typing CR the next location will be displayed automatically. Termination character is point (.).

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

The contents of the locations between lower and upper address will be dumped on the specified file, 8 consecutive locations on each line. If no file-name is specified the contents are dumped on the terminal.

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

The contents of the locations between lower and upper address will be dumped on the specified file, 8 consecutive locations (16 characrers) on each line. Non-visual characters appear as space. If no file-name is specified the characters are dumped on the terminal.

1.6 *MEMORY-IMAGE LOADING*

Your program(s) may be loaded directly into a memory-image file instead of into main memory.

The loader is put into this special mode by the command:

*IMAGE-FILE <file name>

whereby, the file name denotes the memory-image file and has IMAG as default type.

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

The DUMP- and BPUN-commands apply to memory-images as well as to pure memory-loaded systems.

1.7 *OVERLAY SEGMENTATION OF FORTRAN PROGRAMS*

As a program may be too large to fit in the available memory space, the programmer may decide to divide his program into several overlay-segment modules. When the program system is generated in this way, only certain portions (root-segment + one overlay-segment) of the executing program need to be in memory concurrently. The various overlay-segments reside in the same area of memory at different times, and during time of execution they are loaded automatically (in binary form) by the runtime system when the control is transferred to one of its entry-points. The overlay structure consists of a main program (referred to as the root segment) and one level of associated overlay segments.

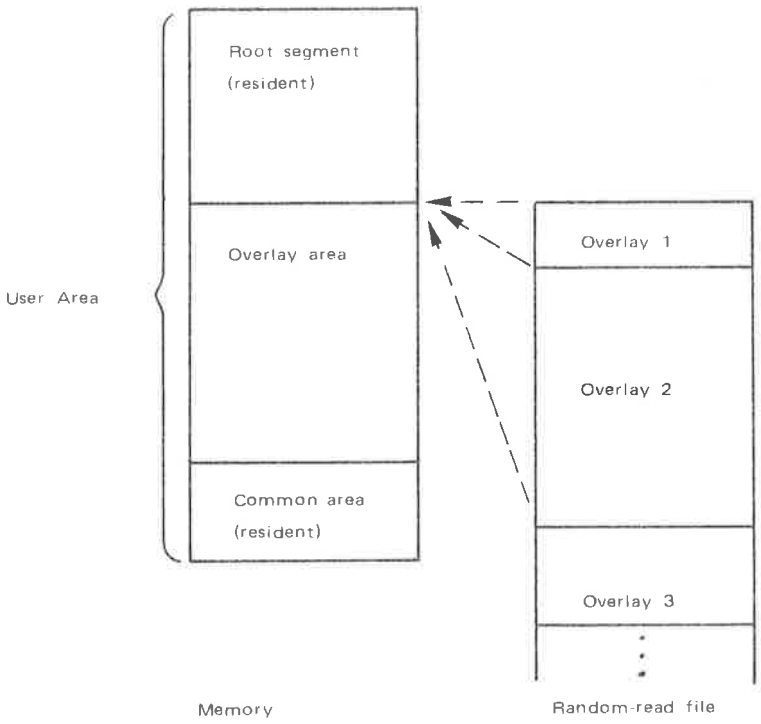


Figure 1.1. *The Overlay Structure*

The root segment and the common area reside in memory throughout the entire execution, while the overlays reside on a random-read file. This file is specified with the OVERLAY-FILE command which also acts as an overlay-modus setting. The OVERLAY-FILE command should, therefore, be the first directive given after recovering the loader from the operating system. Default type of the overlay-file is OVLY.

Example:

```
*OV-FI OVLAY1
```

Note:

The scratch file 100 may be used as the overlay-file by giving the command OV- FI 100.

The root-segment is generated by loading the main program, along with some (user selected) frequently used function/subprograms. The root segment should be completed by loading the FORTRAN runtime and library-file (if not permanently present since system generation).

Usually, when the root-segment is completed, some undefined subprograms are referenced. Such referenced subprograms may be grouped into overlay-segments in various ways. In generating overlays, the programmer should organize his program to retain the commonly used subprograms in the root segment and the less used routines in the overlay-segments, which reside in memory only temporarily, one at a time. The set of subprograms on an overlay-segment is specified by the loader-command:

*OVERLAY-ENTRY <name 1> <name 2>...<name N>

where the names refer to subprograms called from the root segment.

When this command is given, the specified subprograms can be loaded from one or more BRF files. It is recommended that the overlay subprograms be kept on a separate BRF file compiled in library mode (refer FORTRAN Reference Manual). In this way, the specified set of subprograms may be selected and loaded into the overlay independently on the compilation sequence.

When all specified entry-points are defined and no other undefined references occur on this overlay, the message:

OVERLAY COMPLETED

is given.

When all specified entry-points are defined but other references occurred during the load process, the message:

UNDEF REFERENCES ON OVERLAY

is given. The file(s) containing these entry-points may then be loaded in order to complete the overlay.

When an overlay is completed another one may be specified and created according to the outline above.

An overlay system is considered to be complete if no undefined reference occurs on the entry-map (UNDEFINED-ENTRY command). It may then be started by the RUN command or saved by the DUMP command (to be retrieved later).

The user should consider the following restrictions:

- Only one level of overlays is possible, thus the root segment may reference any other root segment or overlay subprogram, while an overlay subprogram may only reference subprograms in its associated overlay or in the root segment.

- The FORTRAN-debugging option cannot be used in connection with overlays.

An example of overlay generation

In the following example the root segment is compiled into the file ROOT:BRF, and the subprograms into LIBSUB:BRF (in library mode) in the sequence SUBR1, SUBR2, SUBR3, SUBR4. To generate a program system with SUBR1, SUBR4 on overlay 1 and SUBR2, SUBR3 on overlay 2, the following command sequence will apply:

```
*OV-FI OVERLAY-SYSTEM
*LOAD ROOT
*OV-ENT SUBR1 SUBR4
*LOAD LIBSUB
OVERLAY COMPLETED
*OV-ENT SUBR2 SUBR3
*LOAD LIBSUB
OVERLAY COMPLETED
*
```

2 THE RELOCATING LOADER

2.1 *BINARY RELOCATABLE FORMAT*

BRF is organized in eight bit bytes and is not bound to any particular data medium (magnetic tape, drum, etc.). The information contained in the object program may be classified as follows: Control-information is held in a control-byte and interpreted as a loader-command; programmed information is held in two bytes containing a sixteen bit word and is called a P-group; and symbolic information is held in four (six in STANDARD FORTRAN) bytes called an S-group and contains a symbol consisting 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, and so forth. By means of special programs (i.e., compilers, assemblers, loaders, etc.), the program may be transformed from one form to another, but conceptually we will regard the program to be the same before and after the transformation. We say that a program is written in relocatable format or, more briefly, that the program is relocatable, if the program is not predestined to lie in a specific place in memory. Thus, a FORTRAN program and an assembly program (with only symbolic addresses) are relocatable programs, while a machine program is in general not relocatable (See example 1, 2, and 3 following).

Example 1		Example 2	Example 3
Program PER written in assembly mode		Program PER written in machine code and placed in location 10	Program PER written in machine code and placed in location 20
PER,	JMP I + 1	125001	125001
	OLE	14	24
	157	157	157
	751	751	751
OLE,	WAIT	151000	151000

The machine 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 a relocatable format, because 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 Figure 2.1.

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

Figure 2.1: *Example of BRF*

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

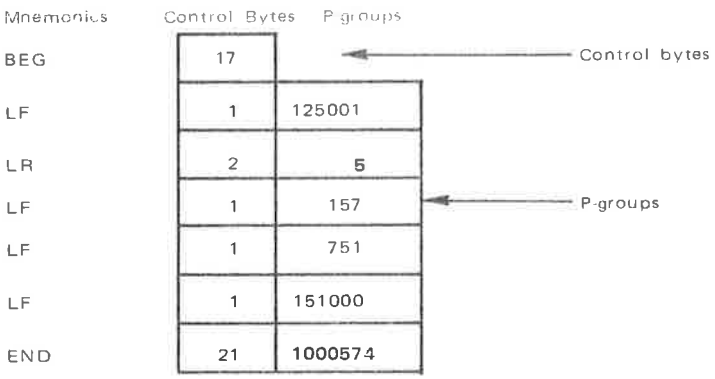


Figure 2.2: *Example of BRF*

The contents of the control byte are called the control number. Control number 17 (mnemonic BEG) marks the beginning of the program. In Standard FORTRAN the 17 (BEG) is followed by 32 (LONGF) which tells us that all S-groups 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 the memory up to location 621 and is going to load the object program described in Figure 2.1.

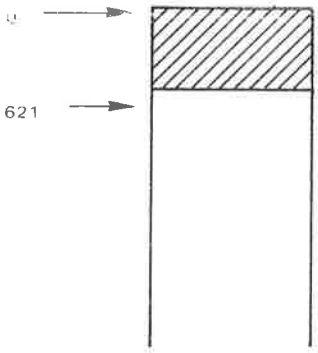
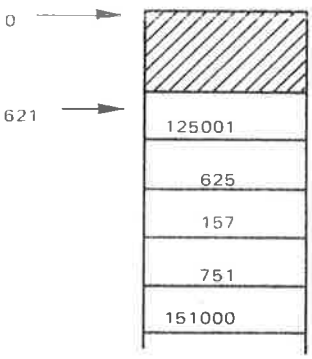


Figure 2.3:
Memory Image Before Loading



= 620 + 5

Figure 2.4:
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 Figure 2.4.

2.3 *PROGRAM UNITS*

A FORTRAN program is composed of one main program and one or more subprograms (in FORTRAN, the subprograms are called SUBROUTINE subprograms and FUNCTION 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 an 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 called an external reference.

2.4 *SEPARATE COMPILING/ASSEMBLING*

When the FORTRAN compiler compiles a program, each program unit is translated without any information about the other program units. Therefore, the program units need not be compiled at the same time. This is called 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:

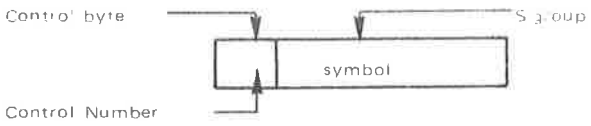


Figure 2.5: *S-group with Control Number*

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), ends 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 subprograms 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 implemented 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 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 references to a program unit 'B' it does not matter which of them is loaded first. The (necessary) library subprograms are loaded at last. If a library subprogram 'A' makes reference to another library subprogram 'B' then 'A' must appear first (without any consequences for the user).

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 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_8 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 visualized as in Figure 2.6.

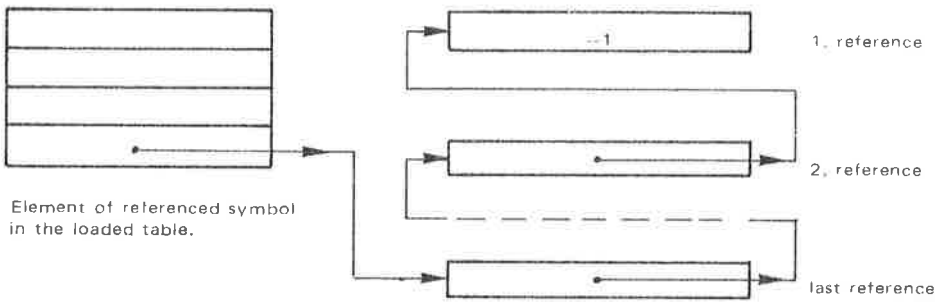


Figure 2.6: *Symbol Reference Link*

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 while the COMMON block is allocated in the topmost part of the available space. The length of the COMMON block is given in the object program, and the corresponding control number is 26 (ASF). The COMMON block address is found by subtracting this length from the upper bound.

The COMMON block address must be known before the addresses referencing COMMON are loaded. Therefore, the COMMON block address which uniquely specifies the maximum COMMON 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 which are in COMMON are 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 this succession, the allocation of the blocks would be as in Figure 2.7.

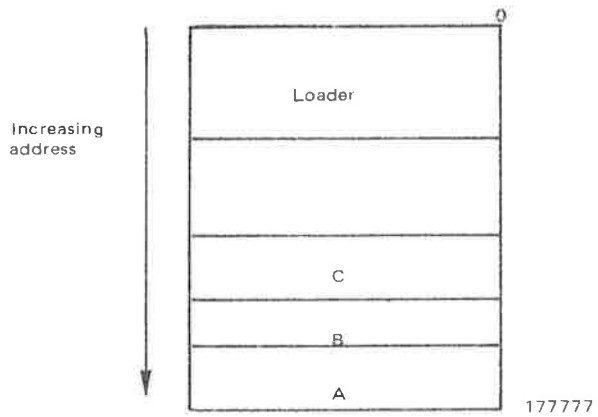


Figure 2.7: *Multiple COMMON Blocks*

The ADS-group has the format:

<ADS> <S-group>

with the interpretation: 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 Figure 2.1 all the numbers are given as octal numbers in two's complement modulo sixteen bits.)

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 contains a content which will be added into the address. Both the address and the content (which may be an address itself) 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 Table 2.1 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). W_1 to W_2 are the contents of the first to the n'th P-group, respectively.

If 'a' is an address or address expression, then (a) is the content of this address. The expression X → (Y) means that the value of X shall replace the content of Y.

Control Number	Mnemonic	Interpretation
0	FEED	Neglect
1	LF	$W_1 \rightarrow ((CLC))$, (CLC) + 1 → (CLC)
2	LR	W_1 + (PB) → ((CLC)), (CLC) + 1 → (CLC)
3	LC	W_1 + (CDB) → ((CLC)), (CLC) + 1 → (CLC)
4	AFF	$W_1 + (W_2) \rightarrow (W_2)$
5	ARF	W_1 + (PB) + $(W_2) \rightarrow (W_2)$
6	AFR	$W_1 + (W_2 + (PB)) \rightarrow (W_2 + (PB))$
7	ARR	W_1 + (PB) + $(W_2$ + (PB)) → $(W_2$ + (PB))
10	SFL	W_1 → (CLC)
11	AFL	W_1 + (CLC) → (CLC), fill zeros
12	SRL	W_1 + (PB) → (CLC)
13	COMN	
14	MAIN	Symbol in S-group will become the main entry.
15	LIBR	Conditional loading
16	ENTR	Symbol in S-group is assigned value of CLC

Table 2.1.: *Table of Control Numbers*

Table 2.1. continued

Control Number	Mnemonic	Interpretation
17	BEG	(CLC) → (PB) First control byte of a unit
20	REF	Symbol in S-group is referenced in CLC
21	END	W_1 contains the BRF-checksum
22	INHB	Warns that compilation errors have occurred
23	EOF	End of loading
24	LNF	$<W_0><W_1 \ldots W_{W_0}>$ → (CLC), . . . (CLC + $W_0 - 1$)
25	RT	W_1 contains real-time priority
26	ASF	<symbol> <number> Defines common length
27	ADS	<symbol> + (CLC −1) → (CLC −1) Adds common address
30		Not used
31		Not used
32	LONGF	Flags six bytes S-group
33		Not used
34	INL	$W_2 \rightarrow (W_1 + (PB)$
35	DBL	$W_i \rightarrow (W_1 + PB + i - 2)$ (i = 2 to 3)
36	RLL	$W_i \rightarrow (W_1 + PB + i - 2)$ (i = 2 to 4)
37	CXL	$W_i \rightarrow (W_1 + PB + i - 2)$ (i = 2 to 7)
40	INC	$W_5 \rightarrow (W_4 + ADR)$
41	DBC	$W_i \rightarrow (W_4 + ADR + i - 5)$ (i = 5 to 6)
42	RLC	$W_i \rightarrow (W_4 + ADR + i - 5)$ (i = 5 to 7)
43	CXC	$W_i \rightarrow (W_4 + ADR + i - 5)$ (i = 5 to 10)

Table 2.1.: *Table of Control Numbers*

With reference to the control numbers 40, 41, 42, and 43, the W_1, W_2 and W_3 contain a common block name. At load time the symbol must be defined and its value is referred to as ADR.

Control Number	Belonging Word	Used By		
		MAC	Standard FTN	Basic
0	0		x	x
1	1	x	x	x
2	1	x	x	x
3	1	x		
4	2			
5	2			
6	2	x	x	x
7	2		x	x
10	1	x		
11	1		x	x
12	1	x		
13	1			
14	2 (3)	x	x	x
15	2 (3)	x	x	x
16	2 (3)	x	x	x
17	0	x	x	x
20	2 (3)	x	x	x
21	1	x	x	x
22	0	x	x	x
23	0	x	x	x
24	$1 + W_1$			
25	1	x	x	x

Table 2.2: *The BRF-subset Produced by MAC, FORTRAN and BASIC*

Table 2.2, continued

Control Number	Belonging Word	Used By		
		MAC	Standard FTN	Basic
26	3 (4)	x	x	
27	2 (3)	x	x	
30	--			
31	--			
32	0		x	x
33	--			
34	2		x	x
35	3		x	
36	4		x	
37	7		x	
40	4 (5)		x	
41	5 (6)		x	
42	6 (7)		x	
43	9 (10)		x	

Table 2.2: *The BRF-subset Produced by MAC, FORTRAN and BASIC*

APPENDIX A

LOADER COMMAND SUMMARY

APPENDIX 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 space or comma.

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

The contents of the locations between lower and upper address will be dumped on the specified file, 8 subsequent locations (16 characters) on each line. Non-visual characters appear as space. If no file-name is specified the characters are dumped on the terminal.

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 binary on the destination file with an octal coded bootstrap ahead. The main start entry of the program may be specified symbolically or octally. The bootstrap address (octal number) specifies where the bootstrap program (44_8 locations) will be located if the program is loaded into a stand-alone NORD-10. Default destination type: BPUN. Default boundaries range from the lowest to the highest address accessed by the loader since last recovery.

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 EDMUN 777

DEPOSIT <octal address> <new contents>

The new contents are put into the octal address specified. If last parameter is missing the old contents are displayed and may be changed by typing the new contents on the same line. By typing CR the next location will be displayed automatically. Termination character is a full stop (.).

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, and then starts in 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 last recovery.

ENTRIES-DEFINED [<file name>]

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

FREE: <current location>–<upper bound>

Default file name is the terminal.

Example:

```
E–D
EDMUN = 000777
  FREE = 02000-177777
```

ENTRIES-UNDEFINED [<file-name>]

This command is similar to ENTRIES-DEFINED. However, only undefined symbols are printed.

Default file name is the terminal.

EXIT

The control is left 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 entry list-out.

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 the main memory. Default file type is IMAG.

Example:

IM-FI DREAM

KILL<symbol>

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

Example:

KILL EDMUN

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

The file(s) specified will be loaded until end – of – file is encountered. Default file-type is: BRF.

Example:

LOAD SUB1, SUB2

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

The contents of the locations between lower and upper address will be dumped on the specified file, 8 subsequent locations on each line. If no file-name is specified the contents are dumped on the terminal.

Example:

OCTAL-DUMP 0 3

000000: 000000 000000 000000 000000

OVERLAY-ENTRY <entry name> [<entry name>.....]

Specifies the subprograms on the next overlay. These units may be called from the root-system or from the actual overlay itself.

OVERLAY-FILE <file no./name>

Specifies the overlay-file and the loader is put into overlay-mode.

APPENDIX B

THE LOADER ERROR–MESSAGES

APPENDIX B

THE LOADER ERROR-MESSAGES

AMBIGUOUS

The last command word is abbreviated until an ambiguity has occurred.

AT UPPER LIMIT

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

BRF CHECKSUM ERROR

The BRF-file contents are damaged due to hardware or software errors occurring when it was written or read.

COMMON BLOCK EXPANDED

The length of an already defined common block is declared 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.

ILL BRF-CONTROL NO

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

INSUFFICIENT PROGRAM

Error-diagnostics have occurred during the compilation process.

LOADER-TABLE OVERFLOW

The loader symbol table is filled.

NO MAIN ENTRY

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

NO OVERLAY- FILE SPECIFIED

The command OVERLAY-FILE should be given first.

OVERLAY ENTRY-TABLE OVERFLOW

Too many entries in the overlay-system. Table size can only be expanded by generating a new loader version.

OVERLAY SEGMENT-TABLE OVERFLOW

Too many overlay segments. Table size can only be expanded by generating a new loader version. Default size: 16 overlays.

AUTO-BUFFER FULL

No more room for automatic-commands.

In addition to the messages listed above, some of the file system diagnostics (I/O-errors) may appear at your terminal.

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf. 21 73 71

COMMENT AND EVALUATION SHEET

NORD-10 Relocating Loader Manual
ND-60.030.03

In order for this manual to develop to the point where it best suits your needs, we must have your comments, corrections, suggestions for additions, etc. Please write down your comments on this pre-addressed form and post it. Please be specific wherever possible.

FROM: _____

_____

_____