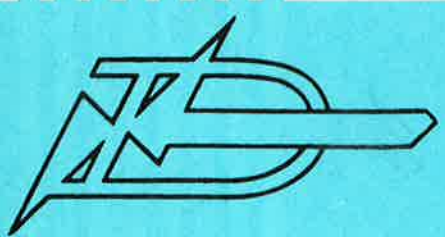
NORD
COMPUTER SYSTEMS

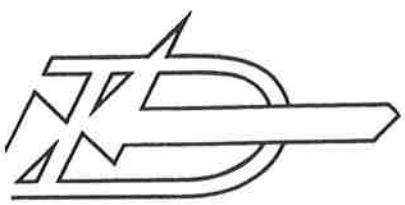
BINARY RELOCATING LOADER



A/S NORSK DATA-ELEKTRONIKK
Økernveien 145, Oslo 5 - Telefon (02) 21 73 71

NORD
COMPUTER SYSTEMS

BINARY RELOCATING LOADER



A/S NORSK DATA-ELEKTRONIKK

REVISION RECORD

Revision	Notes
3/70	Original Printing
1/73	Second Edition

Publication No. ND-60.030.02
January 1973

A/S NORSK DATA-ELEKTRONIKK
Økernvn. 145, Oslo 5 - Tlf. 21 73 71

INTRODUCTION

The loaders are used to read BRF output from the MAC assembler and the NORD FORTRAN compilers into core, and make the loaded program(s) executable.

The loader exists in the following versions:

BRL	- Loader for BRF output from MAC assembler and Mini FORTRAN compiler.
NORD FORTRAN II Loader	- Loader for BRF output from NORD FORTRAN II compiler and MAC assembler.
NORD FORTRAN IV Loader	- Loader for BRF output from NORD FORTRAN IV compiler, NORD FORTRAN II compiler, Mini FORTRAN compiler and MAC assembler.
SINTRAN II All Core Loader II	- NORD FORTRAN II Loader with some extensions for running in a SINTRAN II system.
SINTRAN II All Core Loader IV	- NORD FORTRAN IV Loader with some extensions for running in a SINTRAN II system.
SINTRAN II Real-Time Loader	- Loader for BRF output from NORD FORTRAN IV compiler, NORD FORTRAN II Compiler, Mini FORTRAN compiler and MAC assembler. This loader may only be used in a SINTRAN II mass-storage system.

This manual describes the use of all these versions of the loader except the NORD FORTRAN IV Loader and the SINTRAN II Real-Time Loader. In the ordinary NORD FORTRAN IV Loader, the command structure is different, but a special version of this loader with the same command structure is available.

TABLE OF CONTENTS

--ooOoo--

--ooOoo--

1 GENERAL

1.1 Binary Relocatable Format

Output from the NORD FORTRAN compiler and the BRF-MAC assembler is in Binary Relocatable Format, abbreviated to BRF.

To execute the object program it must be read into memory by a loader.

1.2 Binary Format

BRF is organized in eight bit bytes and is not bound to a certain 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 bytes called an S-group and contains a symbol consisting of one to five 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>

BRF is a sequence of BRF-groups.

1.3 Relocatable Format

A program is a set of instructions and data which, when it is interpreted, will perform an algorithm. A program may be in different 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 examples 1, 2 and 3 below).

Example 1	Example 2	Example 3
Program PER written in assembly code	Program PER written in machine code and placed in location 10	Program PER writt machine code and pl 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 put in the control byte. The program PER will, in BRF, look like Figure 1.2

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

Figure 1.1 Example of BRF

To make it more clear, we organize the BRF-groups by columns (see Examples 2 and 3).

Mnemonics	Control bytes	P-groups	
BEG	17		← Control-bytes
LF	1	125001	
LR	2	4	
LF	1	157	← P-groups
LF	1	751	
LF	1	151000	
END	21	100575	

Figure 1.2 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. Control number 1 (LF) means that the corresponding P-group shall 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.

1.4 Relocation of Internal Addresses

Suppose that the loader has filled core up to location 621 and is going to load the object program described in Figure 1.1.

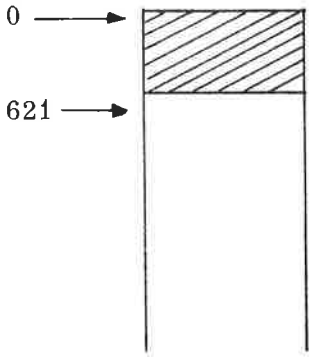


Figure 1.3
Image of core before loading

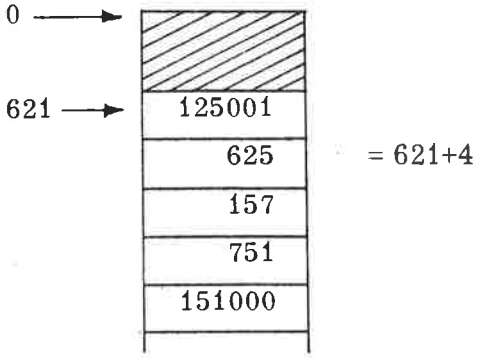


Figure 1.4
Image of core after loading

When the loader reads control number 17 (BEG), the current location (in this case, 621) 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 1.4

1.5 Program Units

A FORTRAN program is composed of one main program and none 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.

1.6 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:

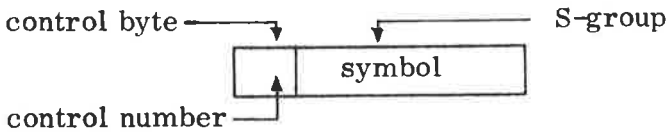


Figure 1.5 S-group with control number

Control number	Mnemonic	Meaning
14	MAIN	Symbolic start address
15	LIBR	A warning that this symbol will appear later as an entry point in the succeeding (library) subprogram
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 subprograms are loaded when the LIBR symbol has been referenced by a REF group and is not already defined as an 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 get any other information, the program units are loaded consecutively, starting at a system implemented address. However, the program units may be put in any other way 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 last. If a library subprogram A makes reference to another library subprogram B, then A must appear first (without any consequences for the user, however).

1.7 Linking of Program Units

The loader has a symbol table which has one entry for each program unit. Each entry consists of two words (=4 bytes) 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. There are two free bits in the symbol, and if the memory size is less than 32K, at least one bit in ADR.

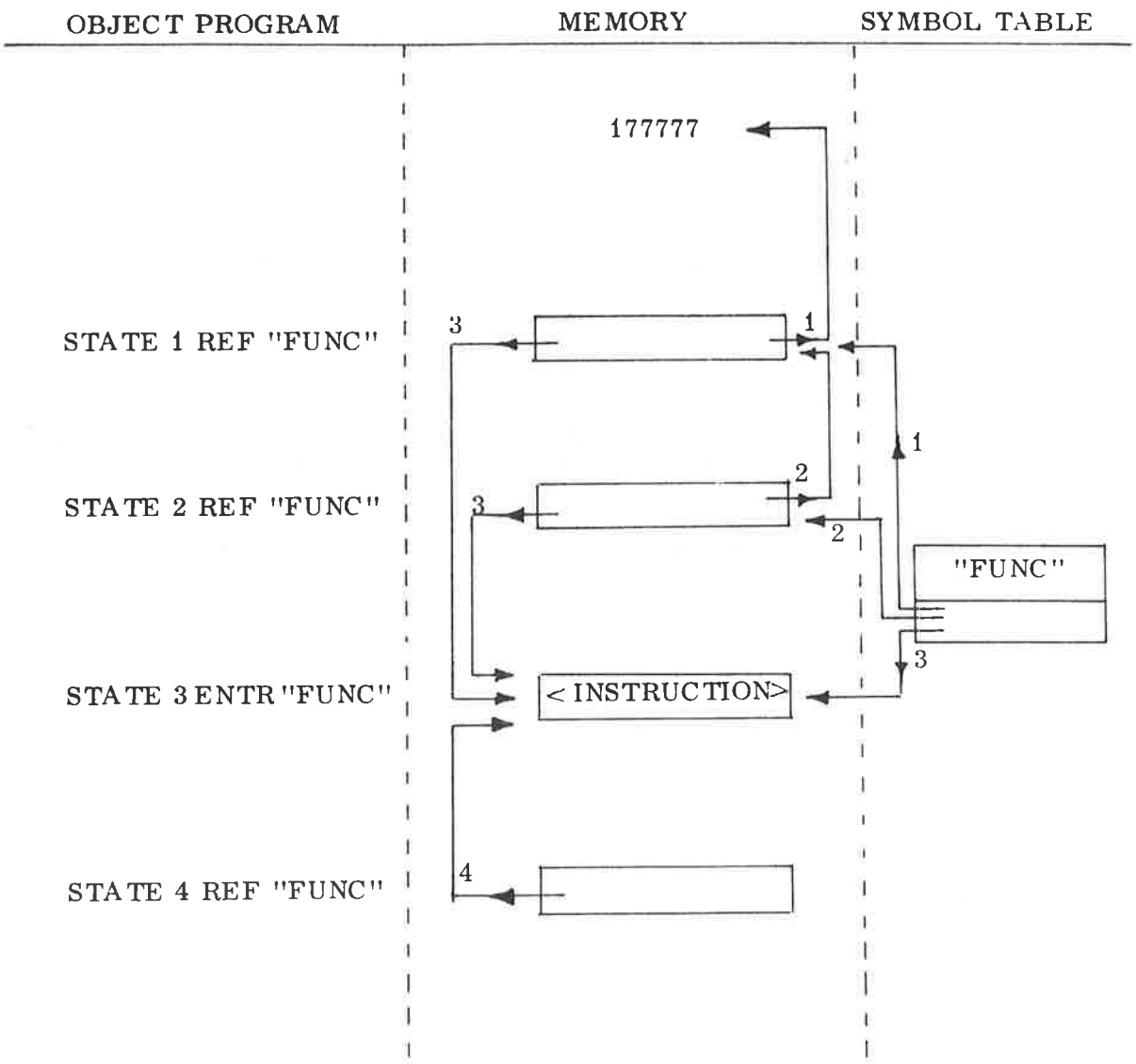


Figure 1.6 The link-method

In Figure 1.6, the numbers close to the arrows are the states after the corresponding input from the object program. REF "FUNC" is a symbolic external reference to the entry point FUNC (STATE 1, 2 and 4). ENTR "FUNC" is the entry point FUNC (STATE 3). A rectangle is a machine word, and if an arrow is starting inside the rectangle, then the machine word contains a pointer to the location to which the arrow points. If there are several arrows out from a rectangle, then the arrow is valid which has the highest number not exceeding the current STATE number. For instance, in STATE 2, the arrows 3 and 4 are dummy.

1.8 COMMON Block

The memory area in which the loader puts the program is a contiguous 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 13 (COMN). The COMMON block address is found by subtracting this length from the upper bound. Thus there is indeed no waste space if the program reaches the COMMON block.

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 the COMMON block cannot be expanded by the succeeding program units.

Data which are in COMMON are referenced by indirect addressing. Such addresses are preceded by the control number 3 (LC), which tells the loader to add the COMMON block address.

In order to allow multiple COMMON blocks, modification of the loader is straight forward, if the restriction not to expand COMMON is accepted. The COMN group is extended to the format

< COMN > <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, then the allocation of the blocks would be as in Figure 1.7.

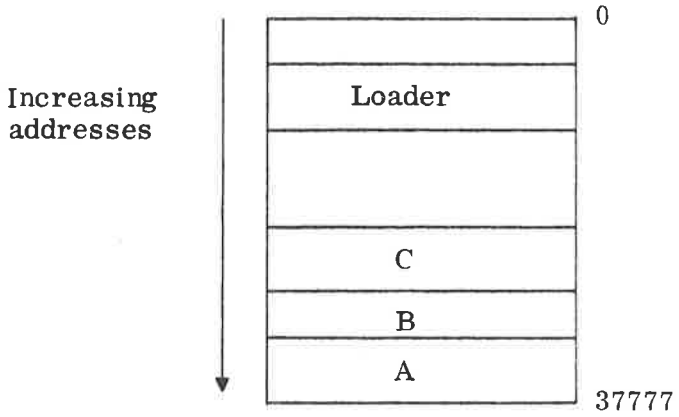


Figure 1.7 Multiple COMMON Blocks

The LC-group then has to be extended to the format

LF <P-group> <ADS> <S-group>

with the interpretation: The value of the S-group is added to the previously read P-group.

1.9 Checksum

In order to detect read errors during loading, a checksum is put behind each END control byte. Here is everything from the BEG control byte to the END control byte 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 sixteen bit numbers. (In Figure 1.1, all the numbers are given as octal numbers, in two's complement modulo sixteen bits.)

1.10 Fix-up Facility

BRF and the loader is designed to allow single-pass, sequential compiling as discussed in Section 1.3. This implies that the loader is able to fix up 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 shall 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 this gives the four possiblities.

2 THE BRF-LOADER

2.1 Terminology

The legal control numbers are sequential numbers starting at zero and are interpreted as commands to the loader. (The numbers themselves are used directly as jump parameters in the loader program). They are listed in a table (Figure 2.1) below, 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 put. PB is the program base of the current program unit. CDB is the COMMON data base (COMMON block address). W_1, W_2 are the contents of the first respective second P-group.

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.

The control number list may easily be extended to meet future needs. The table contains only those used by the FORTRAN compiler.

2.2 Loader Parameters

The following symbols are parameter names in the loader:

CLC	current location counter
PB	program base to current program
CDB	COMMON data base
SA	start address
ADR	address to a segment which is contained in the program table TAB
LB	lower bound
UB	upper bound
CL	COMMON length
SKPF	skip flag
MANF	manual flag
MAXN	maximum control numbers plus one
w1, w2	first and second word after a control byte
CHSM	checksum
PRIO	priority of real time programs

TABLE OF CONTROL NUMBERS

Contr. no.	Mnemonic	Interpretation
0	FEED	Neglect
1	LF	$W_1 \rightarrow ((CLC))$, $(CLC) + 1 \rightarrow (CLC)$
2	LR	$W_1 + (PB) \rightarrow ((CLC))$, $(CLC) + 1 \rightarrow (CLC)$
3	LC	$W_1 + (CDB) \rightarrow ((CLC))$, $(CLC) + 1 \rightarrow (CLC)$
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	$W_1 + (PB) + (W_2 + (PB)) \rightarrow (W_2 + (PB))$
10	SFL	$W_1 \rightarrow (CLC)$
11	AFL	$W_1 + (CLC) \rightarrow (CLC)$, fill zeroes
12	SRL	$W_1 + (PB) \rightarrow (CLC)$
13	COMN	
14	MAIN	
15	LIBR	
16	ENTR	
17	BEG	$(CLC) \rightarrow (PB)$
20	REF	
21	END	
22	INHB	
23	EOF	
24	LNF	$<W_0> <W_1 \ldots . W_{W_0}> \rightarrow (CLC), \ldots (CLC+W_0-1)$
25	RT	$W_1 \rightarrow$ PRIO
26	ASF	<symbol> < number >
27	ADS	<symbol> + (CLC-1) → (CLC-1)
30	ASG	<symbol> < number >
31	ADG	<symbol> + (CLC-1) → (CLC-1)

2.3 Error Messages

The loader error messages have the format ERR Ldd where dd is a two-digit error number as explained below.

01	common,expanded
02	double defined entry point
03	checksum error
04	erroneous program
05	illegal control number
06	overlap
07	no start address
08	symbol table full
09	undefined symbols
10	undefined common or global block
11	undefined label (system error)
12	illegal character in octal number

3 LOADER MONITOR

The loader is activated and controlled from an on-line Teletype through a command program called the loader monitor.

The echo modus of the loader may be changed from non-echo to echo mode or reverse, by typing control K on the Teletype. (If standard I/O - STIO1 - is used by the loader.)

3.1 The Commands

An Automatic mode.

This command loads program units from the device with the logical device number n until the control byte EOF is read, and then returns control to the loader monitor.

Cn Print vale of current location.

This command will print the value of current location, (CLC), on the device with the logical device number n.

Example:

```
L * C 1↵
CLC : 005000
L *
```

D Deposit new value (octal) into the specified address. Type the address terminated by /, then the contents of the location will be printed, and then the user may type the new value he wants to deposit, or just give carriage return if no change is wanted.

Example:

Deposit the instruction JMP * -1 (124377) into location 302

```
L * D 302/125000 124377
L *
```

En Examine contents of locations.

This command will print the contents of the specified locations in octal format on the device with the logical device number n.

Example:

```
L * E 1 10 15↵
000010/001234
001235
000000
000000
000014
000015
L *
```

In this example the contents of locations 10 through 15 are printed on the device with the logical device number 1.

This command is equal to the MAC command)PRINT.

F Fix loader symbol table and set lower bound equal to current location.

I Define new size of loader symbol table.

Example:

```
L * I 100↵
L *
```

The loader symbol table will hold 100 (octal) symbols in this example.

The loader symbol table is placed immediately after the loader program, and will be expanded upwards. This command will also set CLC equal to the new LB.

We will advise the users to use the I command before any program units are loaded.

CORE LAY OUT

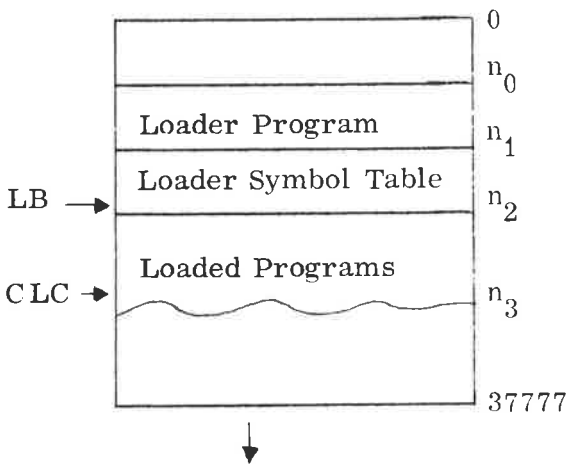


Figure 3.1 (cont.)

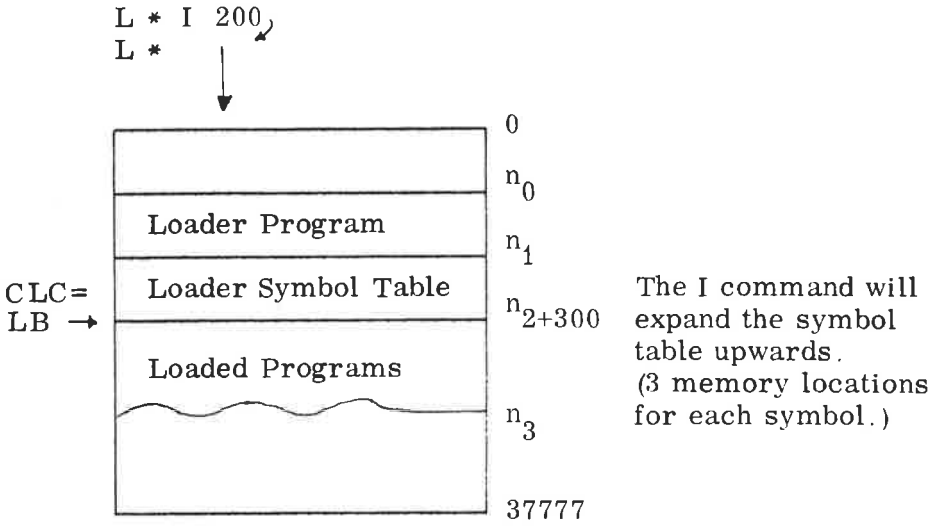


Figure 3.1

Note: Take care not to expand the loader symbol table into loaded programs.

L — Set start load address.

Example:

```
L * L 5000
```

In this example the next program to be loaded will be loaded from location 5000 (octal) and upwards.

Mn — Manual mode.

Load one program unit (until END) from the device with the logical device number n, and return control to loader monitor.

Nn — List undefined symbols.

This command will list all undefined symbols in the loader symbol table on the device with the logical device number n.

Example:

```
L * N 5
  PER  U006000
PRINT  U006010
 OLER  U006070
L *
```

These three lines will be printed by the line printer

R — Reset loader.

S — Start execution of the loaded program.

U Define upper address for loader area (upper bound).

Example:

```
L * U 70000↵
L *
```

In this example upper bound will be set to address 70000.

Wn Write defined symbols.

This command will list all defined symbols in the loader symbol table on the device with the logical device number n.

Example:

```
L * W 3↵
  TOR = 005000
 NILS = 005010      } These lines will be punched
NILS2 = 005011      } on paper tape
  ALF = 005400
@
*  : 007600   ←     } Value of current location
C  : 070000   ←     } Lower address of common area
L *
```

This symbol list, until the character @, may be read into the MAC assembler's symbol table, and used for linking of binary programs and BRF programs, or for debugging purposes.

Xn Define symbols.

This command reads symbols and values from the device with the logical device number n into the loader symbol table. The symbol list must be terminated by the character * or @ .

Example:

```
L *X 1↵
SYMBL = 001000
  PER = 050000      } Read from the Teletype
 SINU = 050100
*
L *
```

This command may also read a symbol list produced by the MA assembler's)LIST command.

Y Define only undefined symbols.

If the command X is used after the command Y, only undefined symbols will be defined by the command X and the other symbols will be skipped.

Example:

```
L * N 1↵
SYMBL U004000
   PER U005000
  NILS U005500
L * W 1↵
* : 006000
C : 077777
L * Y↵
L * X 1↵
SYMBL = 010000
   OLE = 123456
*
L * W 1↵
SYMBL = 010000
@
* : 006000
C : 077777
L * N 1↵
   PER U005000
  NILS U005500
L *
```

In this example the symbol OLE will not be defined by the command X, because it was not undefined in loader symbol table.

Zn Undefined symbol.

This command will read symbols from the device with the logical device number n, and make then undefined in the loader symbol table. The symbol list must be terminated by the character @ or *. In the location where the symbol will be undefined, the loader will deposit the value -1.

Example:

```
L * Z 1↵
SYMBL = 10
   OLE = 20
*
L *
```

In this example the symbol SYMBL will be undefined in address 10 and the symbol OLE in address 20, and the contents of address 10 and 20 will be -1.

If we now use the N command, the result will be:

```
L * N 1↵
SYMBL U000010
   OLE U000020
L *
```

The commands X, Y and Z are not standard, but they are available as an option.

3.2 Map of Memory after Loading

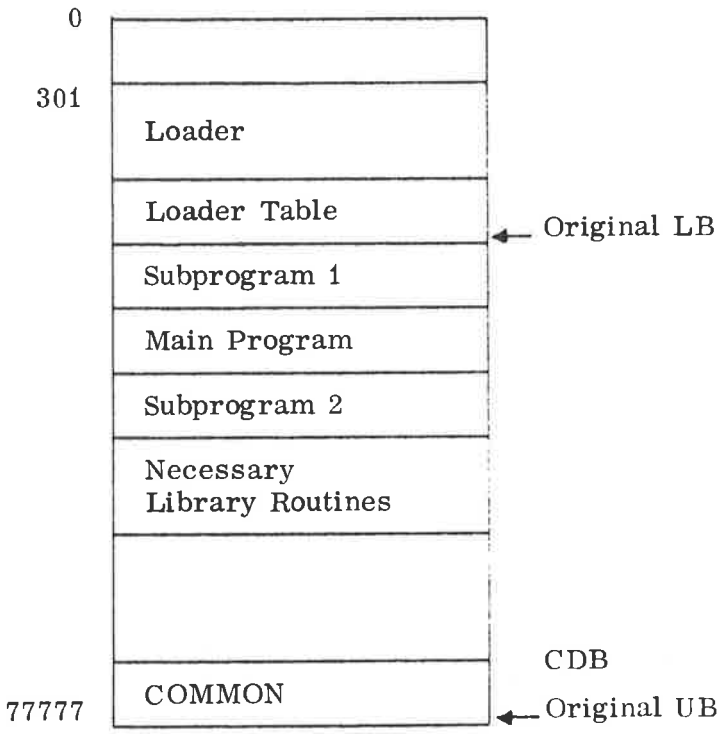


Figure 1.2

APPENDIX A

SINTRAN II ALL CORE LOADER

This is an expanded version of the ordinary loaders (BRL, FORTRAN II and FORTRAN IV loaders) for running under a SINTRAN II system. This loader can handle the RT byte (25_8) and generate an RT-description for RT-programs.

The loader exists in two versions:

> SINTRAN II ALL CORE LOADER II, which can handle BRF output from the MAC assembler and the FORTRAN II compiler.
>
> SINTRAN II ALL CORE LOADER IV, which can handle BRF output from the MAC assembler and the FORTRAN IV compiler.

When the loader is entered, the A-register must contain the logical device number of the calling device, i.e. the logical device number of the device which has invoked the loader. It will be suitable to start the loader with a command in the SINTRAN II operator communication, using the same RT-description as MACD and other low-priority RT-programs. The entry point for the loader is BRL.

The loader starts by printing a message and a question on the calling device:

```
SINTRAN II ALL CORE LOADER II
RESET LOADER?
```

Then the operator must answer with Y for yes or N for no. If the operator answers Y, then the loader will be reset (executing the R-command) and the loader prints the question:

```
NO. OF RT-PROGRAMS:
```

and the operator must then answer with a suitable number. (See the description of the Q-command.)

All commands which are described in Section 3.1 in this manual, except the S-command, are included in this version of the loader.

The following commands are modified:

I Define new size of loader symbol table.

This command is modified such that it will execute the Q-command in addition to its previous function.

F Fix loader symbol table.

This command will only save the current symbols in the loader symbol table. These symbols will not be cleared from the loader symbol table with the R-command. The F-command will not change CLC, PB or LB.

Additional commands:

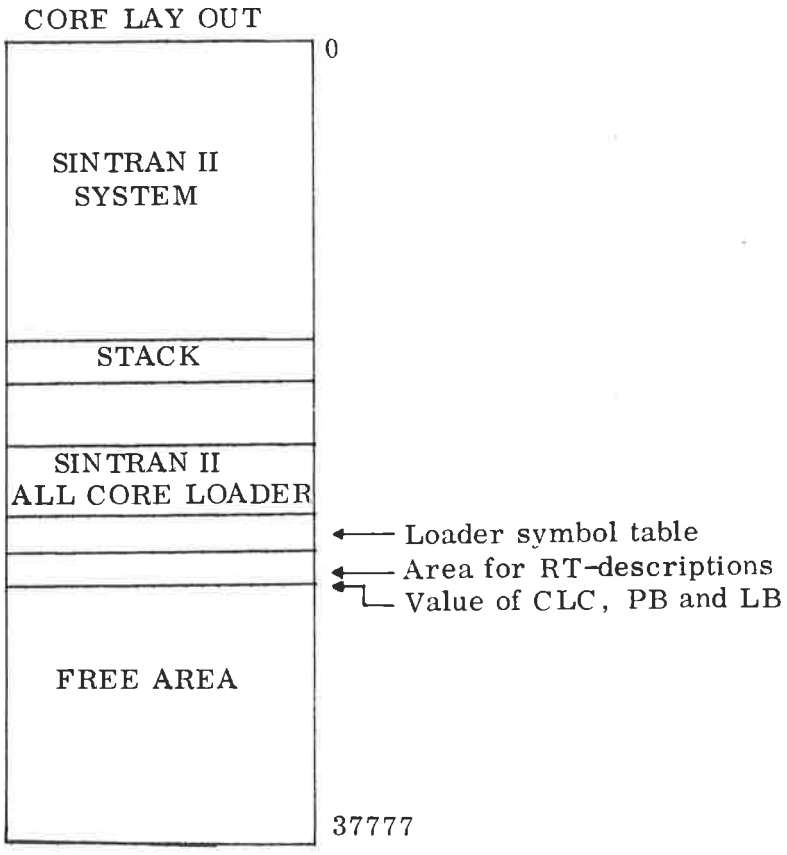
Q Reserve area for RT-descriptions.

This command will ask for number of RT-programs and will then reserve 9 locations for each RT-program immediately after the end of the loader symbol table.

Example:

L * Q 10_8

In this example the loader will reserve 110_8 locations for RT-descriptions immediately after the loader symbol table, and CLC, PB and LB will be set equal to the end of the RT-description area.



ND-60.030.02

T Terminate loader.

This command will terminate the loader. The loader will print the message EXIT LOADER!, and then call RTEXT in SINTRAN II monitor.

The SINTRAN II All Core Loader will generate an RT-description for every RT-program loaded. The priority and the start address of the RT-program will be set into the RT-description and the other locations of the RT-description will be zero.

RT-description generated by the loader:

Location no.	
0	0
1	PRIORITY
2	0
3	0
4	0
5	0
6	START ADDRESS
7	0
8	0

Additional error messages:

20 No octal number has been read in connection with a command.

21 Too many RT-programs loaded. The area for RT-description is already full.

22 Wrong priority of an RT-program. Priority equal to zero or priority is greater than 255 (377_8). This is only a warning message, the priority is set equal to zero, and the loading will continue.

A/S NORSK DATA-ELEKTRONIKK
Erich Mogensens vei 38, Oslo 5 - Tlf. 21 73 71

COMMENT AND EVALUATION SHEET

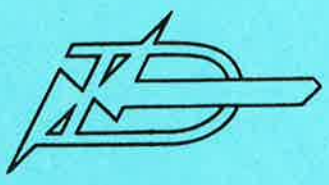
Publication No. ND-60.030.02 Binary Relocating Loader
January 1973

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 _____

_____

_____



A/S NORSK DATA-ELEKTRONIKK
Økernveien 145, Oslo 5 - Telefon (02) 21 73 71