

**US50 Course Manual  
for NORD-50 Software Courses**

**NORSK DATA A.S**





**US50 Course Manual  
for NORD-50 Software Courses**

*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.

The information described in this document is protected by copyright. It may not be photocopied, reproduced or translated without the prior consent of Norsk Data A.S.

Copyright © 1979 by Norsk Data A.S.



## PREFACE

This manual is meant to serve as a supplement to the existing documentation on NORD-50 software. The manual does not intend to be complete or self-contained in any way, and is primarily written to be part of the course material in a related course.

## TABLE OF CONTENTS

+ + +

<i>Section:</i>		<i>Page:</i>
1	INTRODUCTION .....	1-1
2	HARDWARE SUMMARY .....	2-1
3	NORD-50 MONITOR .....	3-1
4	SUBSYSTEMS FOR NORD-50 .....	4-1
4.1	NORD-50 FORTRAN .....	4-1
4.2	NORD-50 Assembler .....	4-2
4.3	NORD-50 Loader .....	4-9
5	FILE HANDLING FROM NORD-50 PROGRAMS .....	5-1
6	4 NORD-50'S SYSTEM (F16 Configuration) .....	6-1
7	PROGRAMMING EXAMPLES .....	7-1
7.1	Running a Simple NORD-50 Program .....	7-1
7.2	Sharing Data Between NORD-10/S and NORD-50 .....	7-4
7.3	Calling an Assembly Routine from FORTRAN .....	7-8

## 1

## INTRODUCTION

The NORD-50 processor is constructed to be part of an integrated computer system under control of a NORD-10/S (NORD-100). The characteristics of the NORD-50 processor includes high execution speed, high data precision and large physical address space.

These qualities, combined with the flexible interaction with SINTRAN III on the NORD-10/S, makes the NORD-50 Computer System especially suited for applications in the fields of science and industry.

## 2

## HARDWARE SUMMARY

The NORD-50 Computer System consists of:

- NORD-10/S (or NORD-100) CPU
- NORD-50 CPU
- Multiport Memory System
- I/O System and Peripherals

Figure 2.1 illustrates a NORD-50 Computer System configuration.

The two processors communicate through a NORD-10/S I/O bus in the I/O system. Both processors are connected to the Multiport Memory System and may both use the shared memory. Therefore, this area also provides means of communication between the processors. In addition, each processor may have its private memory, inaccessible to the other.

The peripherals of the I/O system must be accessed through the NORD File System, which is part of SINTRAN III on the NORD-10/S. For this reason, I/O to the NORD-50 CPU must normally go through the NORD-10/S CPU. This is a rather slow method. Therefore, there exists a feature for optimized file transfer to disk and magnetic tape, where the initiation of the file transfer is performed in the file system, while the transfer itself only involves the I/O system and shared memory. This feature is referred to as direct memory access (DMA).



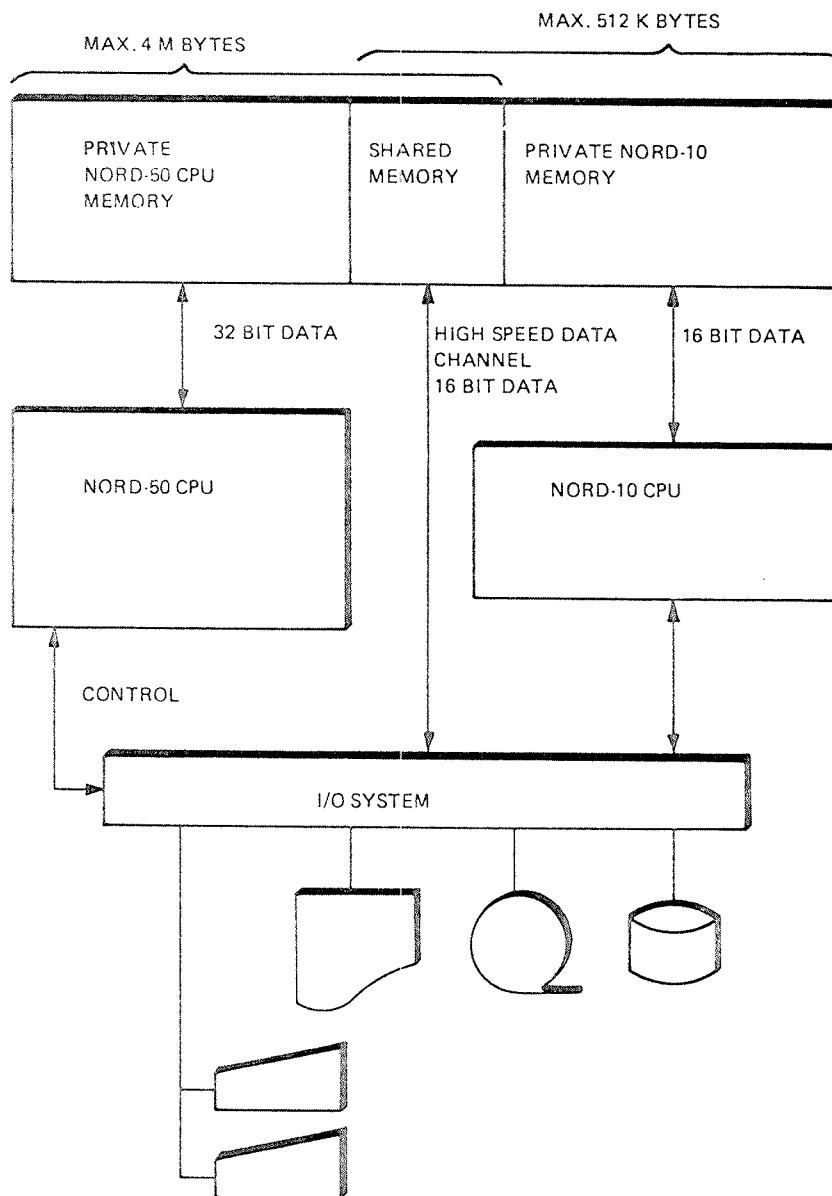
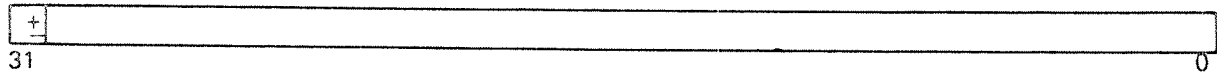


Figure 2.1: NORD-50 Computer System Parts

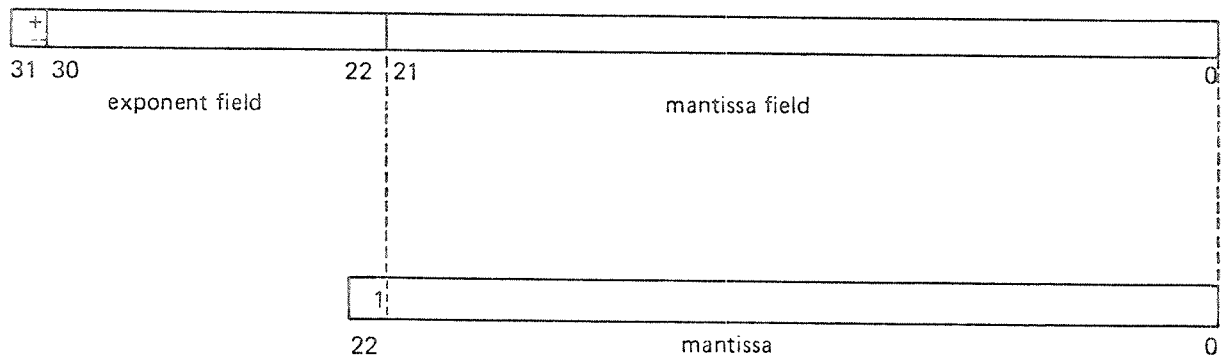
The NORD-50 CPU is a 32 bit processor. It has an extensive instruction set. Instruction look-ahead and specialized units for arithmetical and logical operations give increased performance. Data elements include fixed point and single or double floating point precision, the latter given a precision of 16 decimal digits. The range of floating point data is from  $10^{-76}$  to  $10^{76}$ . The format of data elements is illustrated in Figure 2.2.



### Fixed Point Format

Negative numbers are represented in two's complement.

Range:  $-2,147,483,648$  to  $2,147,483,647$



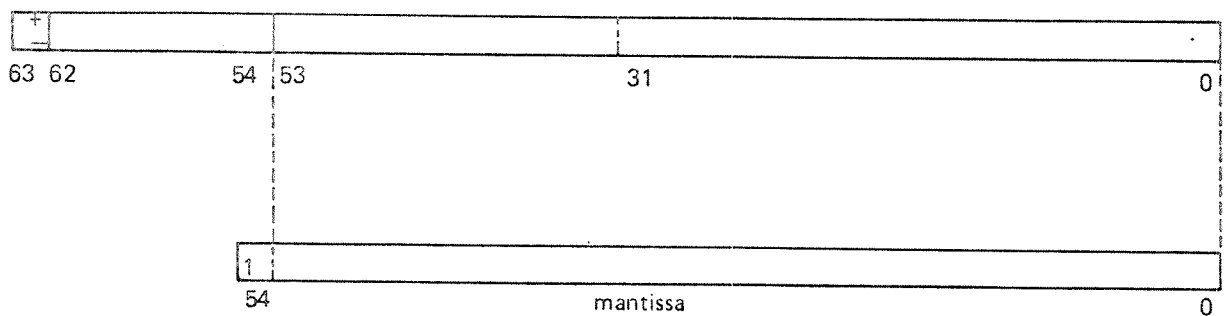
### Single Precision Format:

Bit 31 holds the sign of the mantissa. The exponent is biased.  $400_8$  is added to it before it is stored in the exponent field.

Although the mantissa field has 22 bits only, the mantissa has 23 bits because the most significant bit is always assumed to be 1. Mantissa is always in the range 0.5 to 1.0.

Range:  $10^{-76}$  to  $10^{76}$

Precision: 6-7 decimal digits.



### Double Precision Format

Range:  $10^{-76}$  to  $10^{76}$

Precision: 16 decimal digits.

*Figure 2.2: Format of Data Elements*

The processor's register block contains, logically, 64 registers of 32 bits each. These may be used as general single word (fixed point) registers, single precision floating point registers, base registers, index registers and modification registers. They may also be used in pairs to form double precision floating point registers of 64 bits. Certain restrictions on the usage of the registers are enforced by the hardware logic of the register block.

The hardware structure of the register block is illustrated in Figure 2.3.

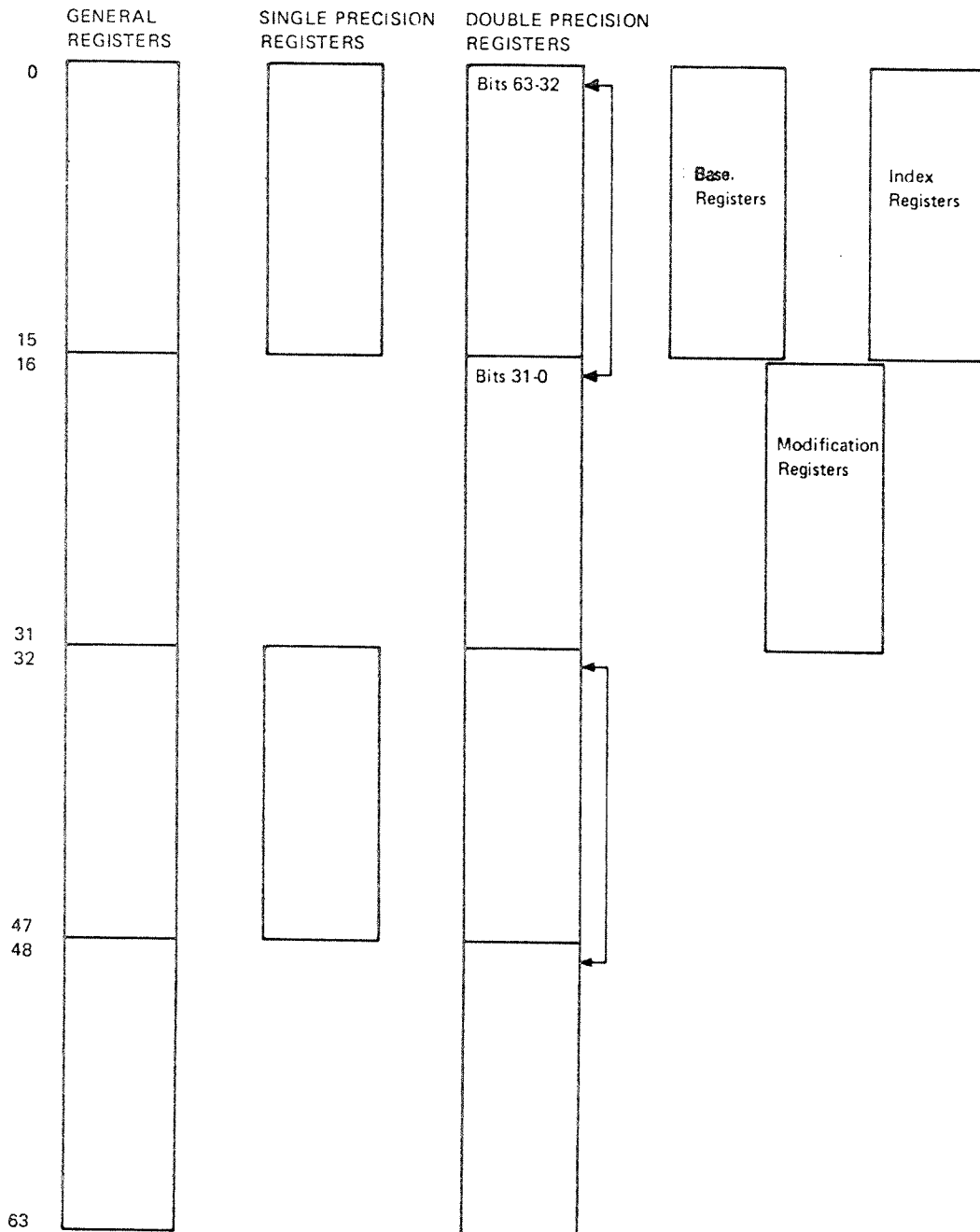


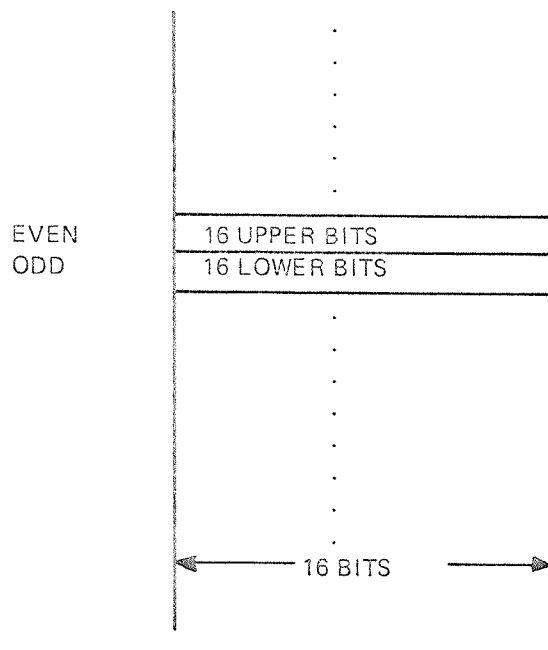
Figure 2.3: Hardware Structure of the Register Block

The figure illustrates that the number of physical registers is 192. However, registers in the same horizontal position have the same contents. The reason for this design is to open for parallel operations on, virtually, one register.

The first register in the register block, register 0, has a special function. Its contents is always 0 and it is therefore used for a number of "clear" operations.

In addition to the register block, the NORD-50 CPU contains 3 auxiliary, internal registers: the Overflow Register holding the upper 32 bits after a multiplication, the Remainder Register holding the remainder after a division and the Carry Register.

The memory of a NORD-50 Computer System consists of 16 bit words. Two consecutive words must therefore be used to form a 32 bit NORD-50 word. A word at an even memory address constitutes the most significant part of this 32 bit word, while the word at the subsequent odd memory address constitutes the least significant part. See Figure 2.4.



*Figure 2.4: Memory*

Memory is organized in units called memory banks. By hardware construction even and odd addresses are located in separate banks. A memory access from the NORD-50 CPU (i.e., accessing a 32 bit word) will therefore affect two separate banks which are accessed in parallel to reduce memory cycle time.

The maximum physical memory available to the NORD-50 CPU is 1 M words of 32 bits (i.e., 4 MB). The NORD-50 address 0 must be within the NORD-10 memory space, so that the two processors have some shared memory.

The NORD-50 CPU is controlled from the NORD-10/S through an I/O bus. Functionally, NORD-10/S performs this control as if the NORD-50 CPU was an I/O device. It issues IOX instructions to transfer data between the A register of the NORD-10/S CPU and the interface and communication registers in the NORD-50 CPU. (See NORD-50 Monitor User's Guide and System Documentation, Figure 1.2.)

The IOX instruction transfers a 16 bit value to/from the A register of the NORD-10/S. To communicate with a 32 bit register in the NORD-50 CPU, two IOX instructions must be executed. The argument of the IOX instruction must be set to select upper or lower part of the NORD-50 register.

## 3

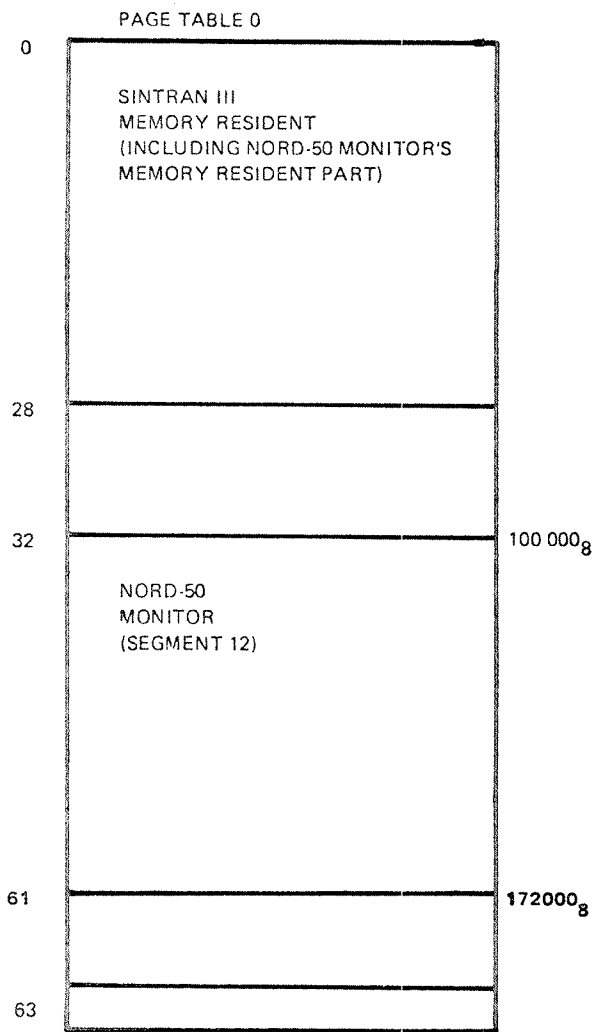
## NORD-50 MONITOR

The NORD-50 Monitor is an optional integrated part of SINTRAN III. Its task is to supervise the operation of the NORD-50 and provide the programmer with efficient tools to control the supervision.

The bulk of the NORD-50 Monitor resides on the NORD-50 Monitor segment (segment no. 12). The segment contains the NORD-50 Monitor Command Processor, various routines performing the commands, and tables and buffers holding data. Segment 12 is a demand segment with logical address space from  $100000_8$ .

Some of the NORD-50 Monitor resides in SINTRAN's memory resident area. This includes the "Enter NORD-50 Monitor" routine (called from the background processor when the command @NORD-50 is issued), the NORD-50 data field (having the same function as data fields for I/O devices), and some I/O and file handling routines which have to be memory resident by the requirements of the I/O system. See Figure 3.1.





Memory resident part of NORD-50 Monitor consists of:

- NORD-50 Data field
- Enter NORD-50 Monitor routine
- Interrupt drivers (levels 3 and 12)
- Subroutines for RFILE, WFILE, ABSTR and MAGTP

*Figure 3.1: NORD-50 Monitor, Virtual Memory Layout*

The "Enter NORD-50 Monitor" routine checks if the NORD-50 is already in use. (A NORD-50 may only execute one program at a time.) If not, the NORD-50 data field is reserved and the NORD-50 Monitor segment is activated by calling the NORD-50 Monitor Command Processor.

The absence of a memory management system in the NORD-50 disallows a dynamic swapping of NORD-50 memory during program execution. Therefore, all parts of a program must be in memory prior to its execution, and it must remain there throughout the execution. This also applies to the pages in shared memory. Being a part of the NORD-10/S memory, the pages in shared memory are normally subject to swapping outside the programmer's control.

A way to prevent SINTRAN from swapping a specific physical area is to load a segment and "fix" it in memory using the FIXC monitor call. This is performed automatically by the NORD-50 Monitor when preparing for the execution of a program. In Figure 3.2 the NORD-50 program resides in the shaded region.

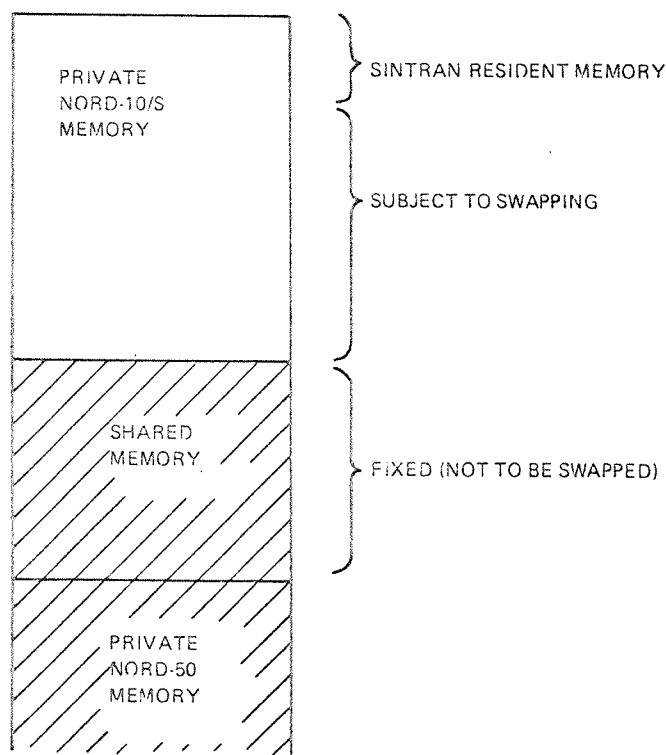
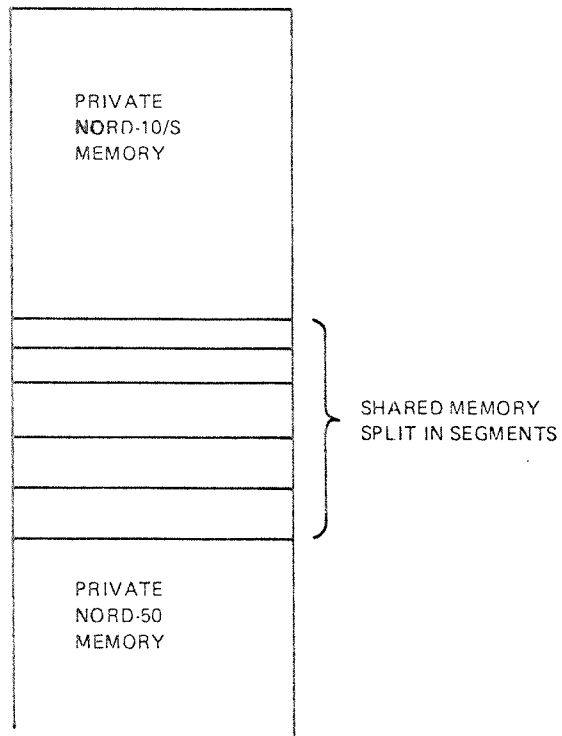


Figure 3.2: Physical Memory Organization

Some NORD-50 programs will not occupy the entire shared memory. In such cases it is ineconomical to fix it all. A better utilization of shared memory is obtained by splitting it into partitions, where each partition is represented by a segment. See Figure 3.3.

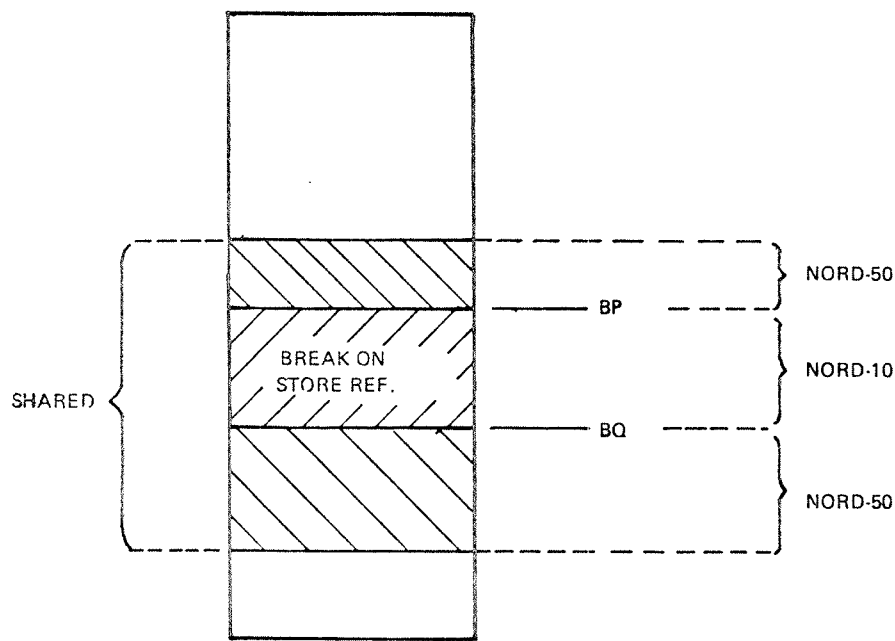


*Figure 3.3: Shared Memory Split in Segments*

The segments may be of different sizes. Each segment is fixed separately if the corresponding memory space is required by a NORD-50 program.

An important note should be made at this point:

The finer the partition (i.e., the smaller the segments), the better the utilization of shared memory will be. But, increased administrative overhead is the price we have to pay. Figure 3.3 illustrates how shared memory should be utilized.



BP and BQ define break area set by:

NORD-50 Loader

- \*EXIT (stored in block 0 of program)
- \*BREAK-CONDITIONS

NORD-50 Monitor

- \*BREAK-CONDITIONS

*Figure 3.4: Shared Memory Allocation*

In the discussion above we have assumed all segments in shared memory to be *dynamic* segments. A dynamic segment has the quality that it will only be fixed when required. A *static* segment, however, will be fixed the first time it is required, and will remain fixed whatever program is being run.

An RT common area may also be part of shared memory. The organization of physical memory is mapped by the NORD-50 Memory Segment table. This table is illustrated in Figure 3.5.

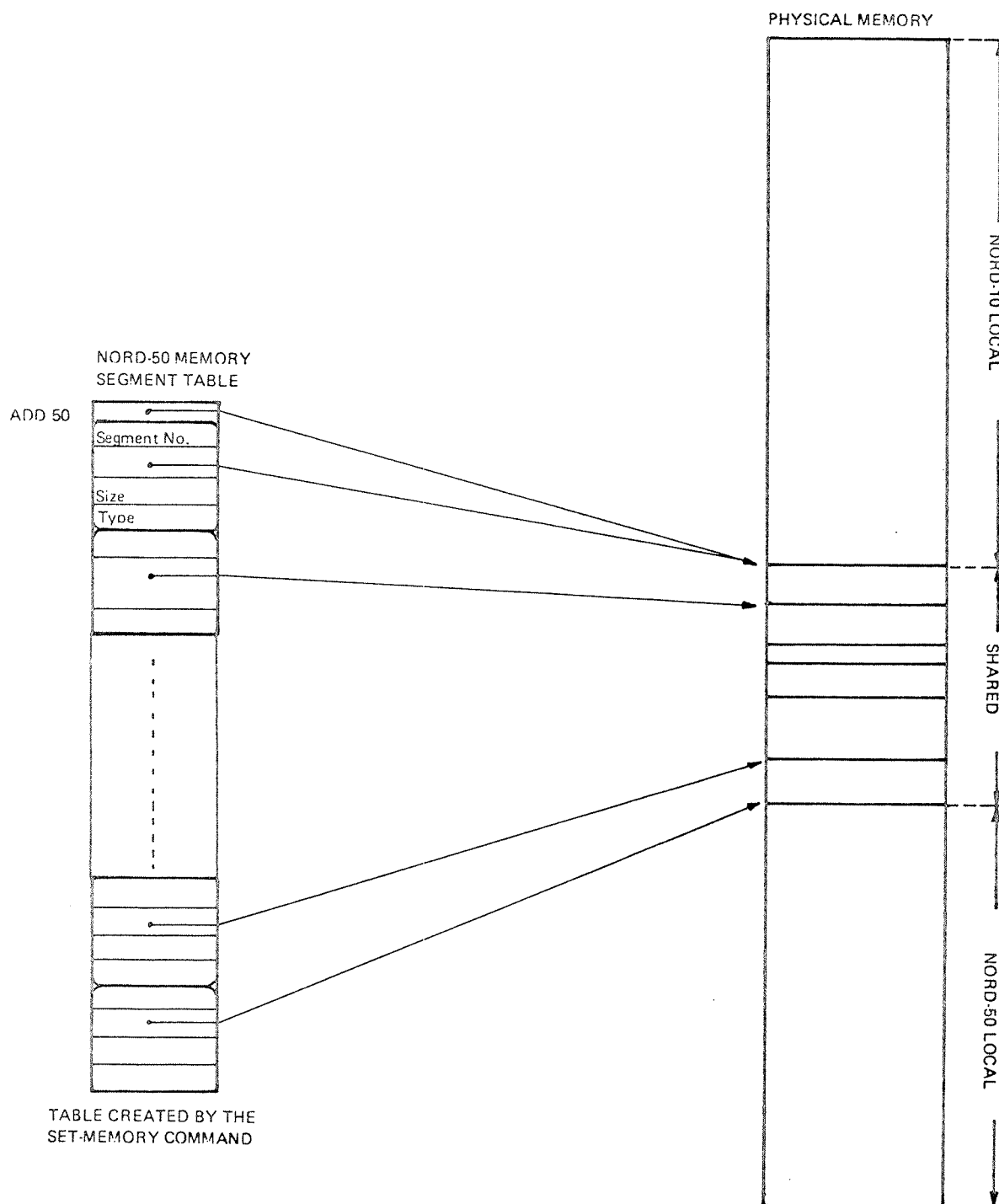


Figure 3.5: NORD-50 Memory Segment Table

Figure 3.6, 3.7 and 3.8 illustrate the format of an executable program and how the NORD-50 Monitor loads and starts the program.

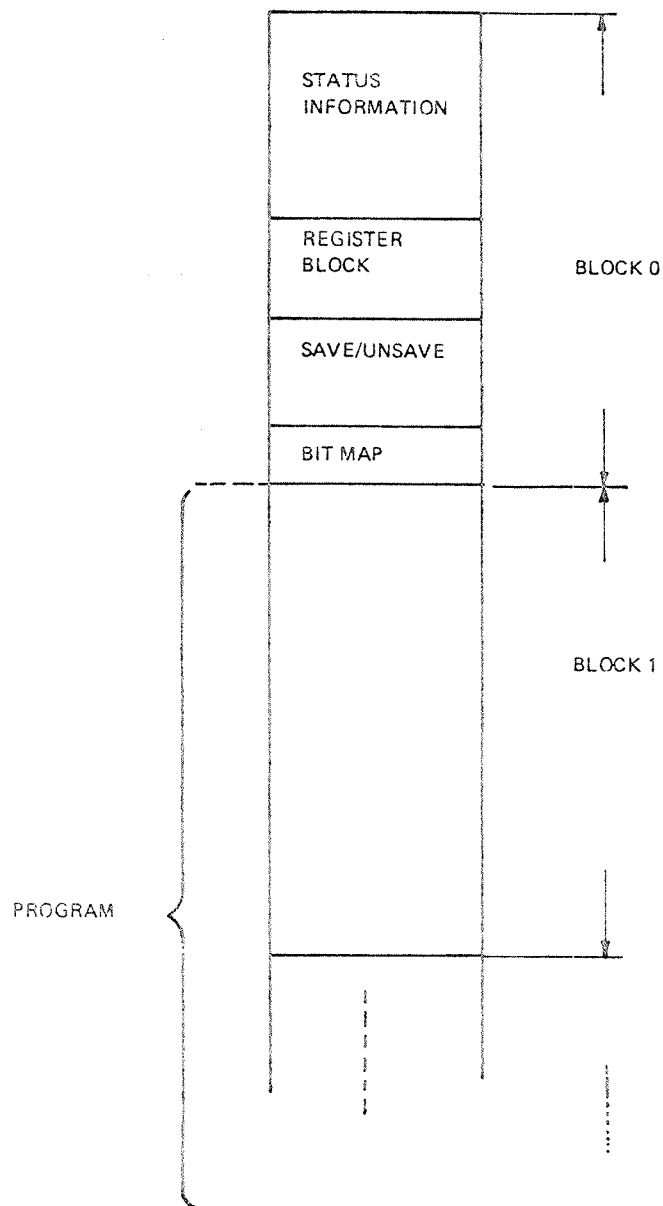


Figure 3.6: Executable Program



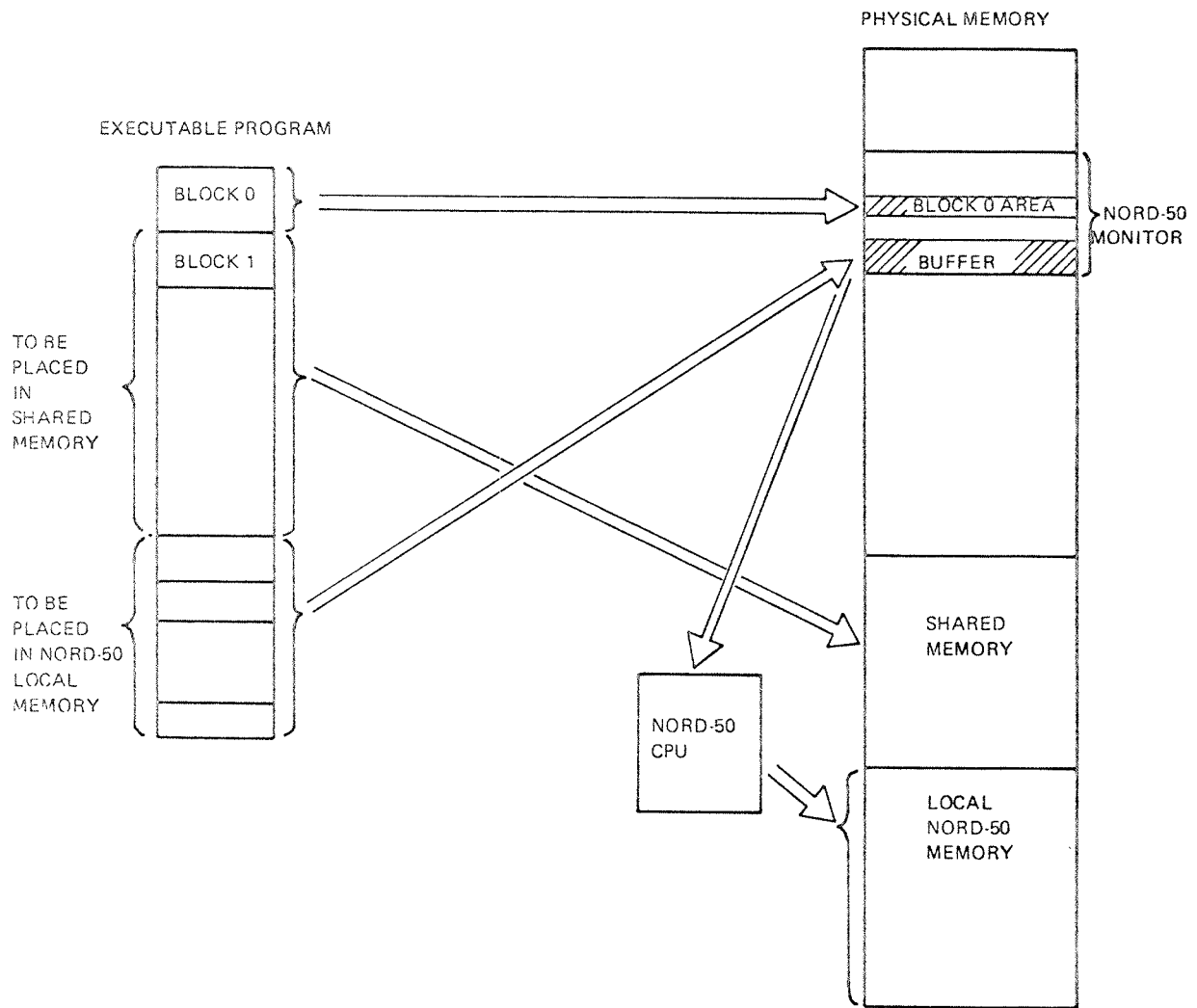


Figure 3.7: Operations at PLACE/LOAD

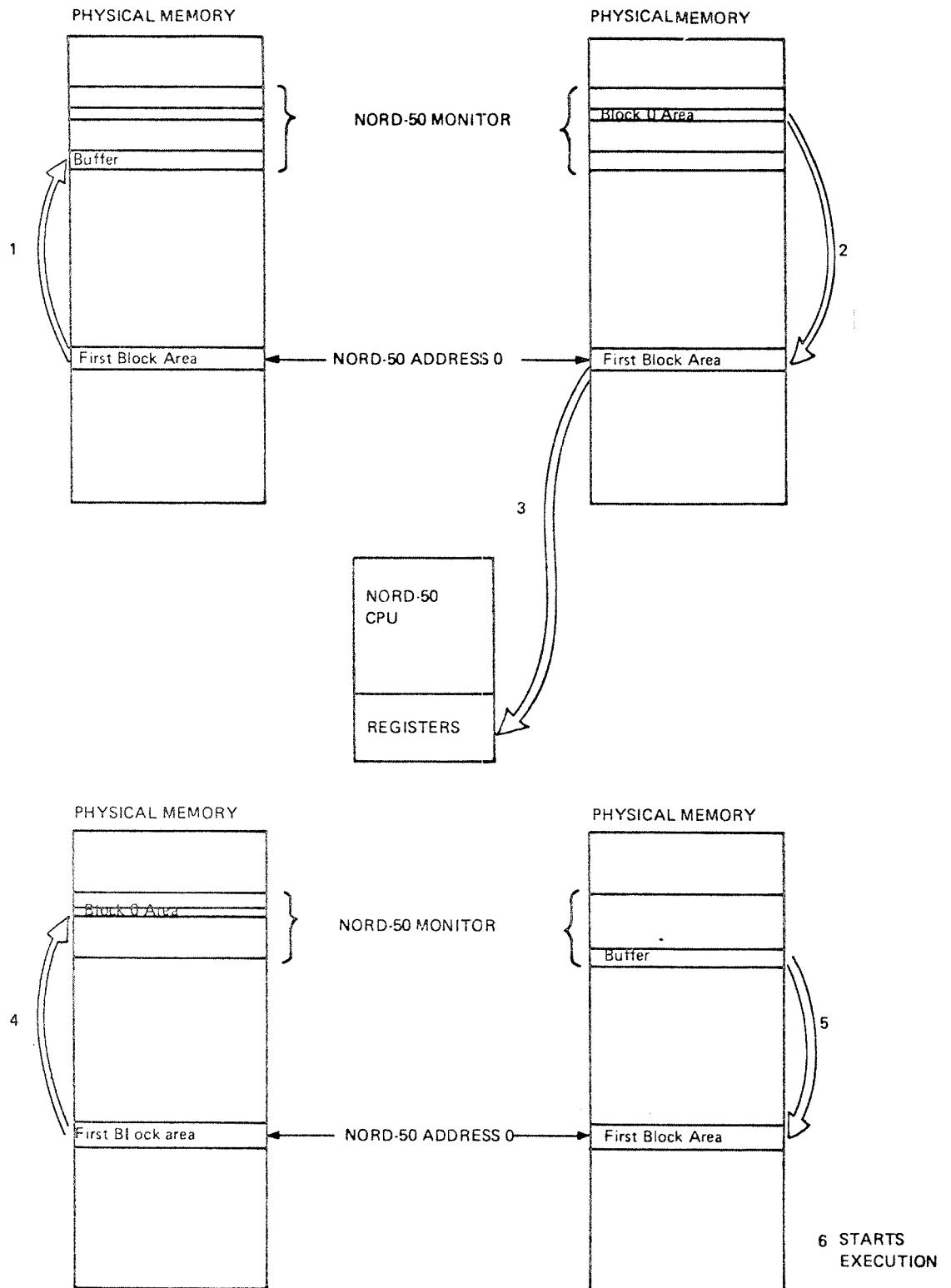


Figure 3.8: Operations at RUN

## 4 SUBSYSTEMS FOR NORD-50

### 4.1 *NORD-50 FORTRAN*

The NORD-50 FORTRAN is source program compatible with the NORD-50 FORTRAN. Nevertheless, differences imposed by hardware may produce unexpected deviations unless the programmer is cautious.

- Standard NORD-50 FORTRAN has 16 bit integers while NORD-50 FORTRAN has 32 bit integers. The programmer should be aware of this difference in range, and also take precautions when using integers in equivalence clauses. The \$DOUBLE-INTEGGER-MODE command to the NORD-10 FORTRAN computer or the DOUBLE INTEGER type declaration in the source program will overcome the problem by forcing allocation of 32 bit integers in the NORD-10.
- Standard NORD-10 floating point format (48 bits) is incompatible with NORD-50 floating point representation. However, an option in NORD-10 configurations offers 32 bit floating point format, compatible with NORD-50 single precision floating point. Unless this option is selected, equivalence clauses with real type elements will cause problems.

For further details see NORD-10 FORTRAN System Reference Manual (ND-60.074.02) and NORD-50 FORTRAN Reference Manual (ND-60.095.02).

## 4.2

*NORD-50 ASSEMBLER*

The NORD-50 Assembler is documented in a separate manual (ND-60.075.01). The NORD-50 Reference Manual (ND-05.003.01) may also be helpful when writing assembly programs.

A summary of all instructions, with the relevant operands for each of them follows.

*Summary of Instructions*

## Memory Reference Instructions:

Mnemonic:		Action:
RTJ	R, D, B, X, I	Return jump
EXC	R, D, B, X, I	Remote execute
MIN	R, D, B, X, I	Memory increment
CRG	R, D, B, X, I	Skip if (R) $\geq$ (Ea)
CRL	R, D, B, X, I	Skip if (R) < (Ea)
CRE	R, D, B, X, I	Skip if (R) = (Ea)
CRD	R, D, B, X, I	Skip if (R) $\neq$ (Ea)
JRP	R, D, B, X, I	Jump if (R) $\geq$ 0
JRN	R, D, B, X, I	Jump if (R) < 0
JRZ	R, D, B, X, I	Jump if (R) = 0
JRF	R, D, B, X, I	Jump if (R) $\neq$ 0
JPM	R, D, B, X, I	Modify (R) and jump if (R) $\geq$ 0
JNM	R, D, B, X, I	Modify (R) and jump if (R) < 0
JZM	R, D, B, X, I	Modify (R) and jump if (R) = 0
JFM	R, D, B, X, I	Modify (R) and jump if (R) $\neq$ 0
ADD	R, D, B, X, I	Add (Ea) to (R)
SUB	R, D, B, X, I	Subtract (Ea) from (R)
AND	R, D, B, X, I	Logical AND between (Ea) and (R)
LDR	R, D, B, X, I	Load (R) with (Ea)
ADM	R, D, B, X, I	Add (R) to (Ea)
XMR	R, D, B, X, I	Exchange (Ea) and (R)
STR	R, D, B, X, I	Store (R) in (Ea)
MPY	R, D, B, X, I	Multiply (R) by (Ea)
DIV	R, D, B, X, I	Divide (R) by (Ea)
LDD	R, D, B, X, I	Load (FD) with (Ea, Ea + 1)
FTD	R, D, B, X, I	Store (FD) in (Ea, Ea + 1)
FAD	R, D, B, X, I	Add (Ea) to (F)
FADD	R, D, B, X, I	Add (Ea, Ea + 1) to (FD)
FSB	R, D, B, X, I	Subtract (Ea) from (F)
FSBD	R, D, B, X, I	Subtract (Ea, Ea + 1) from (FD)
FMU	R, D, B, X, I	Multiply (F) by (Ea)
FMUD	R, D, B, X, I	Multiply (FD) by (Ea, Ea + 1)
FDV	R, D, B, X, I	Divide (F) by (Ea)
FDVD	R, D, B, X, I	Divide (FD) by (Ea, Ea + 1)

R = register  
 D = displacement  
 B = base register  
 X = index register  
 I = indirect addressing

### *Inter Register Operations*

#### Shift Instructions:

Mnemonic:		Action:
SLR	DR, SR, SC	Left rotational shift
SRR	DR, SR, SC	Right rotational shift
SLA	DR, SR, SC	Left arithmetical shift
SRA	DR, SR, SC	Right arithmetical shift
SLL	DR, SR, SC	Left logical shift
SRL	DR, SR, SC	Right logical shift
SLRD	DR, SR, SC	Left rotational double register shift
SRRD	DR, SR, SC	Right rotational double register shift
SLAD	DR, SR, SC	Left arithmetical double register shift
SRAD	DR, SR, SC	Right arithmetical double register
SLLD	DR, SR, SC	Left logical double register shift
SRLD	DR, SR, SC	Right logical double register shift

DR = destination register  
 SR = source register  
 SC = shift count  
      $0 \leq SC \leq 31$  for single register operations  
      $0 \leq SC \leq 63$  for double register operations

## Miscellaneous Operations:

Mnemonic:		Action:
BST	DR, SR, BN	Bit set
BCM	DR, SR, BN	Bit complement
BCL	DR, SR, BN	Bit clear
BSZ	DR, SR, BN	Bit skip on zero
BSO	DR, SR, BN	Bit skip in one
FIX	DR, SR	Convert floating to integer
FIR	DR, SR	Convert floating to rounded integer
FIXD	DR, SR	Convert double precision floating to integer
FIRD	DR, SR	Convert double precision floating to rounded integer
FLO	DR, SR	Convert integer to floating
FLOD	DR, SR	Convert integer to double precision floating
RIN	DR,, ER	Register input
ROUT	,SR, ER	Register output

BN = bit number

$0 \leq BN \leq 31$

ER = external register

2 — OR (overflow register)

3 — RR (remainder register)



## Arithmetic Operations:

Mnemonic:		Action:
RAD	DR, SRA, SRB	Register add
RSB	DR, SRA, SRB	Register subtract
RMU	DR, SRA, SRB	Register multiply
RDV	DR, SRA, SRB	Register divide
RAF	DR, SRA, SRB	Floating register add
RSF	DR, SRA, SRB	Floating register subtract
RMF	DR, SRA, SRB	Floating register multiply
RDF	DR, SRA, SRB	Floating register divide
RAFD	DR, SRA, SRB	Double precision floating register add
RSFD	DR, SRA, SRB	Double precision floating register subtract
RMFD	DR, SRA, SRB	Double precision floating register multiply
RDFD	DR, SRA, SRB	Double precision floating register divide
RAS	DR, SRA, SRB	Register add set carry
RAA	DR, SRA, SRB	Register add add carry
RASA	DR, SRA, SRB	Register add add and set carry
RSS	DR, SRA, SRB	Register subtract set carry
RSA	DR, SRA, SRB	Register subtract add carry
RSSA	DR, SRA, SRB	Register subtract add and set carry

SRA = source register A

SRB = source register B

## Test and Skip:

## Mnemonic:

## Action:

SGR	DR, SRA, SRB	Subtract registers and skip if result $\geq 0$
ASG	DR, SRA, SRB	Add registers and skip if result $\geq 0$
SLE	DR, SRA, SRB	Subtract registers and skip if result $< 0$
ASL	DR, SRA, SRB	Add registers and skip if result $< 0$
SEQ	DR, SRA, SRB	Subtract registers and skip if result $= 0$
ASE	DR, SRA, SRB	Add registers and skip if result $= 0$
SUE	DR, SRA, SRB	Subtract registers and skip if result $\neq 0$
ASU	DR, SRA, SRB	Add registers and skip if result $\neq 0$
SGF	DR, SRA, SRB	Subtract floating registers and skip if result $\geq 0$
ASGF	DR, SRA, SRB	Add floating registers and skip if result $\geq 0$
SLF	DR, SRA, SRB	Subtract floating registers and skip if result $< 0$
ASLF	DR, SRA, SRB	Add floating registers and skip if result $< 0$
SEF	DR, SRA, SRB	Subtract floating registers and skip if result $= 0$
ASEF	DR, SRA, SRB	Add floating registers and skip if result $\neq 0$
SUF	DR, SRA, SRB	Subtract floating registers and skip if result $\neq 0$
ASUF	DR, SRA, SRB	Add floating registers and skip if result $\neq 0$
SGD	DR, SRA, SRB	Subtract double precision and skip if result $\geq 0$
AGFD	DR, SRA, SRB	Add double precision and skip if result $\geq 0$
SLD	DR, SRA, SRB	Subtract double precision and skip if result $< 0$
ALFD	DR, SRA, SRB	Add double precision and skip if result $< 0$
SED	DR, SRA, SRB	Subtract double precision and skip if result $= 0$
AEFD	DR, SRA, SRB	Add double precision and skip if result $= 0$
SUD	DR, SRA, SRB	Subtract double precision and skip if result $\neq 0$
AUFD	DR, SRA, SRB	Add double precision and skip if result $\neq 0$

## Logical Operations:

Mnemonic:		Action:
RND	DR, SRA, SRB	Register AND
RNDA	DR, SRA, SRB	Register AND, use complement of (SRA)
RNDB	DR, SRA, SRB	Register AND, use complement of (SRB)
RXO	DR, SRA, SRB	Register exclusive OR
RXOA	DR, SRA, SRB	Register exclusive OR, use complement of (SRA)
RXOB	DR, SRA, SRB	Register exclusive OR, use complement of (SRB)
ROR	DR, SRA, SRB	Register OR
RORA	DR, SRA, SRB	Register OR, use complement of (SRA)
RORB	DR, SRA, SRB	Register OR, use complement of (SRB)
SZR	DR, SRA, SRB	Set all zeros
RNAB	DR, SRA, SRB	Register AND, use complement of SRA and SRB
ROAB	DR, SRA, SRB	Register OR, use complement of SRA and SRB
RXAB	DR, SRA, SRB	Register exclusive OR, use complement of SRA and SRB

## Argument Instructions:

Mnemonic:		Action:
XORA	DR, ARG	Exclusive OR
ANDA	DR, ARG	AND
ORA	DR, ARG	OR
SETA	DR, ARG	Set register
SECA	DR, ARG	Set register to complement
ADDA	DR, ARG	Add
ADCA	DR, ARG	Add complement
DDP	DR, ARG	Skip if (DR) $\geq$ ARG
DDN	DR, ARG	Skip if (DR) < ARG
DDZ	DR, ARG	Skip if (DR) = ARG
DDF	DR, ARG	Skip if (DR) $\neq$ ARG
DSP	DR, ARG	Skip if (DR) $\geq$ -ARG
DSN	DR, ARG	Skip if (DR) < -ARG
DSZ	DR, ARG	Skip if (DR) = -ARG
DSF	DR, ARG	Skip if (DR) $\neq$ -ARG

ARG = argument

### 4.3 *NORD-50 LOADER*

The NORD-50 Loader is nearly identical to the NORD-10 Relocating Loader, both in design and operation. The NORD-50 Loader is documented in the NORD-50 Loader User's Guide (ND-60.083.02).

## 5

## FILE HANDLING FROM NORD-50 PROGRAMS

The NORD-50 has neither an I/O system nor a file system. Therefore, normal peripheral access from NORD-50 programs must go through SINTRAN's file handling routines in the NORD-10/S. This has two advantages:

- The NORD-50 user is provided with a powerful I/O system and file handling tools compatible to those of the NORD-10/S user.
- Distribution of file processing activities allows a parallel NORD-50/NORD-10 execution giving increased performance.

Unconsidered programming, however, may lead to a reduction in performance if the NORD-50 frequently awaits termination of I/O operations taking place in the NORD-10/S. In order to avoid awkward program design some facts on the file processing strategies of the NORD-50 should be observed.

At INBT/OUTBT one character at a time will be transferred between SINTRAN's resident buffer and the communication register of the NORD-50.

RFILE/WFILE is either performed indirectly, through a buffer, or directly to the programs data area.

At indirect RFILE/WFILE, the N50FIO (loaded from the FORTRAN library) will contain a buffer to be used in read/write operations. If the buffer resides in shared memory, physical I/O may go directly to memory. The user should therefore arrange his program so that the buffer is placed in shared memory. This will be the result when normal loading is performed, because the loader will always load N50FIO first. If the buffer resides in private NORD-50 memory, physical I/O goes to a buffer in the NORD-50 Monitor and a word by word copying will take place between the buffer and the NORD-50 communication registers.

Direct file transfer will automatically take place if the file has been opened in a special modus. When RFILE/WFILE is used in the NORD-50 program, an optimized disk transfer (with ABSTR) will take place. Certain limitations on direct file transfer exists. See NORD-50 Monitor User's Guide and System Documentation (ND-60.076.02). Among the limitation is the requirement that the file be contiguous. The following table illustrates the access mode to be used in the various situations to obtain the optimal file transfer.

NORD-10/S	NORD-50 indexed file	NORD-50 contiguous file
R	RX	D or DC
W	WX	D or DC

By using the ACCESS = 'READ' and ACCESS = 'WRITE' clauses in open statements, the most optimal access mode will be selected automatically.

Note that the RFILE/WFILE calls have a flag. If set, the NORD-50 program will continue in parallel with the file transfer. The WAITF call may be used to check if the transfer has terminated.



## 6

## 4 NORD-50'S SYSTEM (F16 Configuration)

In the F16 Configurations one NORD-10/S and 4 NORD-50s are used. Physical memory is organized as shown in Figure 6.1.

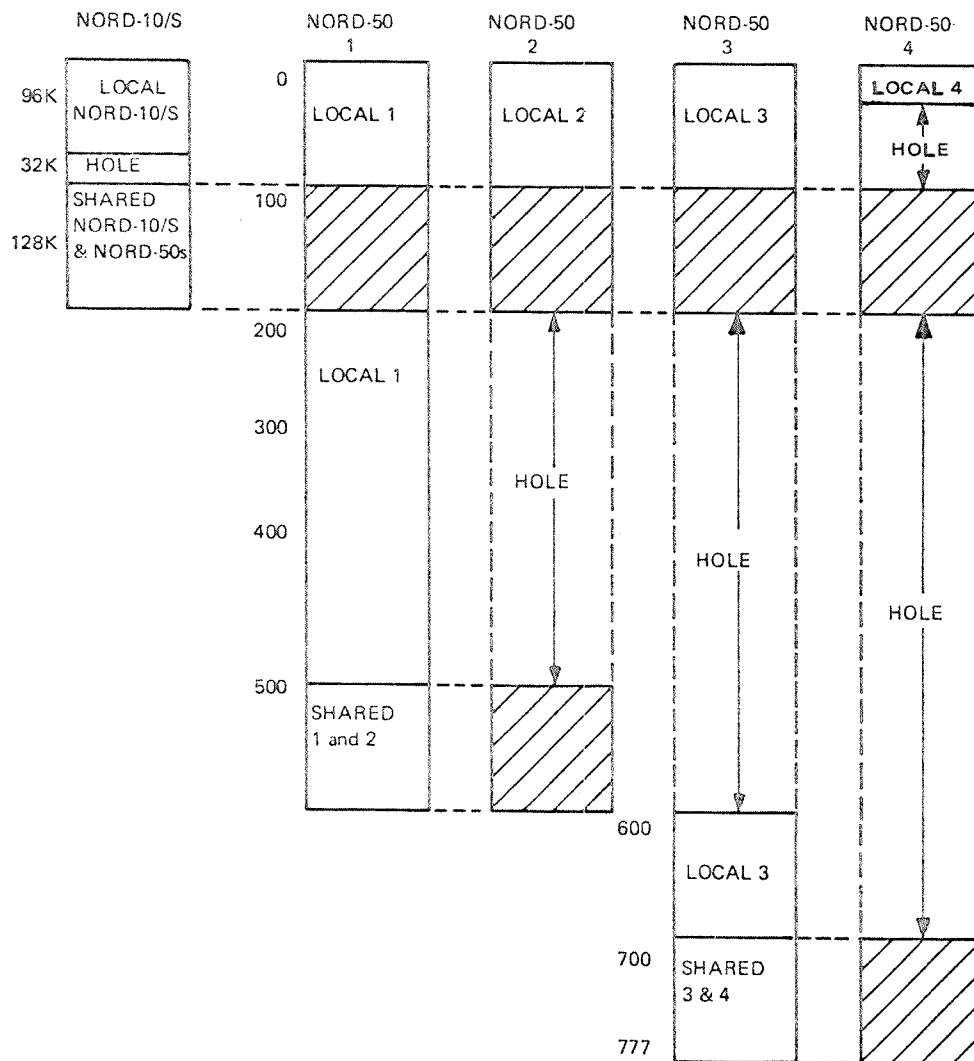


Figure 6.1: F16 Configuration Memory Organization (NORD-50 addresses are given in octal K words (32 bits).)

There are 4 independent NORD-50 Monitors, one for each NORD-50. When the command @NORD-50 is issued, the first available NORD-50 is reserved. If a specific NORD-50 is required, the NORD-50 number must follow as parameter to the command (e.g. @NORD-50 3).

The NORD-50 monitors use reentrant routines on segment 12. In addition, they each have a 5 K data segment, segments 15, 16, 17 and 20 respectively.

## 7 PROGRAMMING EXAMPLES

### 7.1 RUNNING A SIMPLE NORD-50 PROGRAM

Running a simple NORD-50 program:

Figure 7.1 shows the operations needed to prepare and run a FORTRAN program on the NORD-50.

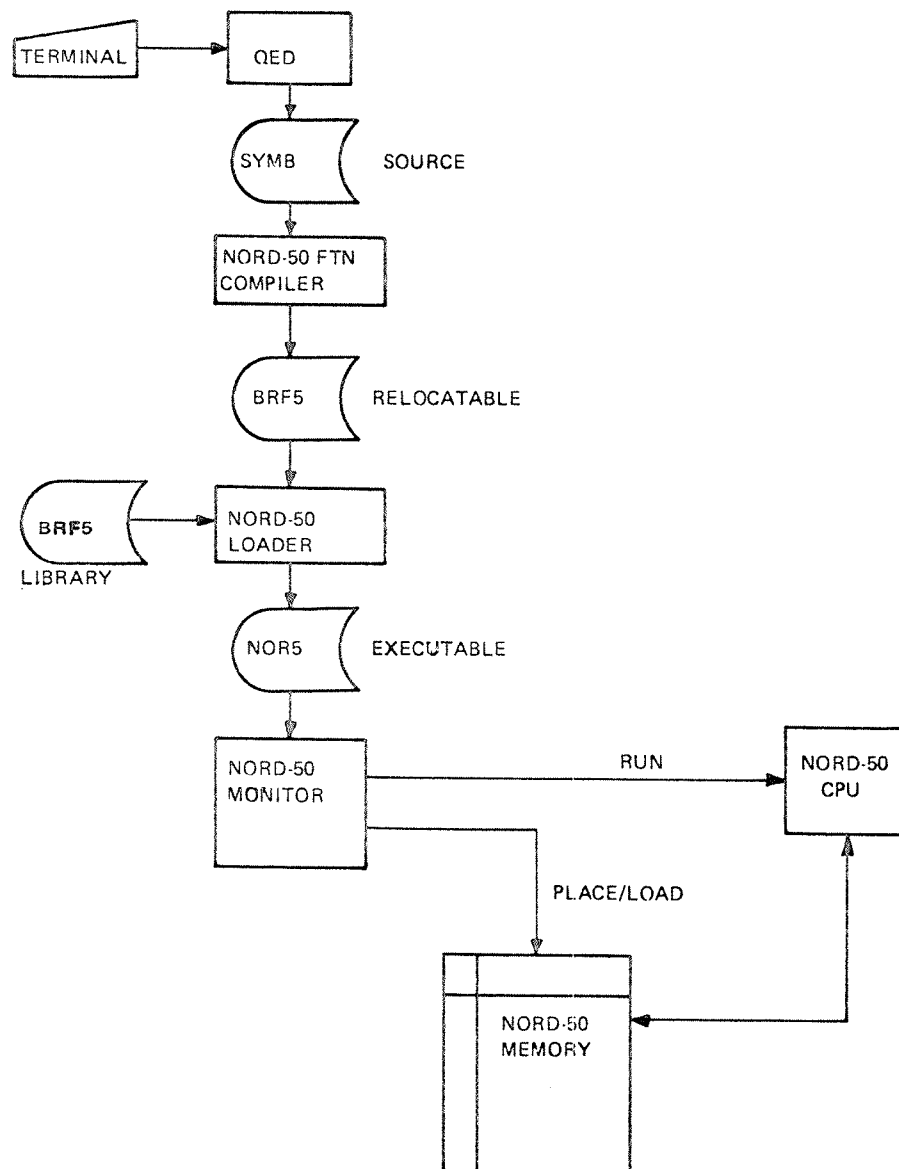


Figure 7.1: Preparing and Running a NORD-50 Program

Note that QED, compiler and loader operations are done on the NORD-10.

The example below illustrates how to compile, load and execute a NORD-50 FORTRAN program.

First, the compiler is invoked and compilation requested.

@N50-FORTRAN

— NORD-50 FTN COMPILER 2159C —

\$ COM SPROG, 1, 100

```

1*      PROGRAM EX1
2*      WRITE (1, 10)
3*  10  FORMAT (*THIS IS A TYPE-OUT*)
4*      END

```

4 STATEMENTS COMPILED. OCTAL SIZE = 42

\$EX

The relocatable program was output to the scratch file (file no. 100) and may now be input to the loader which will produce an executable program.

@N50-LOADER

NORD-50 LOADER — K

MEMORY-IMAGE FILE:

Note that the present processing takes place on the NORD-10. Therefore, a "memory image" file is used by the loader when producing an executable program. The above request must be answered by giving the file name of the file to receive output from the loader. Default file type is NOR5.

MEMORY-IMAGE FILE: ENORD

\*LOAD 100

FREE 0016330 0177777

\*EX

FREE 0016330 0177777

The program may now be executed by invoking the NORD-50 Monitor and using the subcommand LOAD.

@NORD-50

NORD-50 MONITOR — J

\*LOAD ENORD

THIS IS A TYPE-OUT

— \*\*\* END \*\*\* — AT: 000017 —

\*EX

All necessary commands are summarized below. For the sake of comparison the similar procedure for a NORD-10 program is presented.

NORD-50:

@N50-FORTRAN  
\$COM SPROG, 1, 100  
\$EX

@N50-LOADER  
MEMORY-IMAGE FILE:ENORD  
\*LOAD 100  
  
\*EX

@NORD-50  
\*LOAD ENORD

NORD-10:

@N10 FTN  
\$COM SPROG, 1, 100  
\$EX

@NRL  
  
\*LOAD 100  
\*DUMP ENORD  
\*EX

@ENORD

## 7.2 SHARING DATA BETWEEN NORD-10/S AND NORD-50

The following example illustrates how data may be stored in shared memory by a RT program in NORD-10 and read by a NORD-50 program. This is obtained by using a labelled COMMON area, TTF, placed on a segment to be put in shared memory. The RT program in NORD-10 stores the value 1 in the integer V1 in the COMMON area. The program looks as follows:

```
PROGRAM N10PUT, 10
COMMON/TTF/V1
DOUBLE INTEGER V1
V1 = 1
END
```

The NORD-50 program reads the value of V1 by outputting it to the terminal. The NORD-50 program looks as follows:

```
PROGRAM N50GET
COMMON/TTF/V1
INTEGER V1
WRITE (1, 10) V1
10 FORMAT (* V1 = *, I1)
END
```

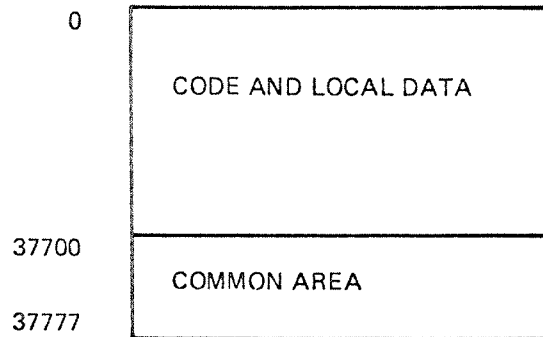
After having compiled both programs, the NORD-10 program is loaded to segment 123 with the following sequence:

```
@RT-LOADER

REAL-TIME LOADER 78.10.18B

*NEW ,,,WP
NEW SEGMENT NO.: 123
*PRESET-COMMON 123, 37700
*NRE (LASSE) N10PUT,,
*END
*EX
```

Segment 123 now looks as follows:



The program is now started with the command @RT N10PUT and V1 will receive the value 1.

The NORD-50 program is now to be loaded by the NORD-50 Loader. However, the normal memory configuration of shared memory must be modified, so that, whenever this program is being run, segment 123 will be part of shared memory. The following sequence will do the job.

```

@N50LDR
NORD-50  LOADER      - M
MEMORY-IMAGE FILE: 100
*LIST-MEMORY*,
SEGMENT NO.  FIRST ADDR.  SIZE  TYPE
200      000000  0010  DYNAMIC
201      010000  0010  DYNAMIC
202      020000  0020  DYNAMIC
203      040000  0020  DYNAMIC
204      060000  0020  DYNAMIC
NONE      100000  0300  LOCAL NORD-50 MEMORY WITH DMA ACCESS

*SET-MEMORY
SEGMENT NO.: 200 10 0
SEGMENT NO.: 201 10 0
SEGMENT NO.: 202 20 0
SEGMENT NO.: 203 20 0
SEGMENT NO.: 123 20 0
SEGMENT NO.: -1 300 4
SEGMENT NO.: 0
*LIST-MEMORY*,
SEGMENT NO.  FIRST ADDR.  SIZE  TYPE
200      000000  0010  DYNAMIC
201      010000  0010  DYNAMIC
202      020000  0020  DYNAMIC
203      040000  0020  DYNAMIC
123      060000  0020  DYNAMIC
NONE      100000  0300  LOCAL NORD-50 MEMORY WITH DMA ACCESS

*SEGMENT-COMMON-DEFINE 123
FREE: 000000 0077737
*LOAD (LASSE)N50GET
FREE: 0016662 0077737
*ENT-DEF
PWF. 0000106 ENTR. 0000232 LEAV. 0000237
EXIT. 0000244 GOER. 0000250 ERR8. 0000262
ERR9. 0000266 ERRAD. 0000547 XCLO. 0001761
XFND. 0002067 BSIZ. 0002267 FIO. 0002603
DATA. 0002730 CLSE. 0003111 N50GET 0016615
TTF 0077740 0000001
FREE: 0016662 0077737
*EX
FREE: 0016662 0077737

```

Now, the NORD-50 Monitor is entered, and the program is executed as follows:

```
@NORD-50
NORD-50 MONITOR — J
*LOAD 100

      V1 = 1
— *** END *** — AT: 0000247 —
*EX
```



### 7.3 CALLING AN ASSEMBLY ROUTINE FROM FORTRAN

The following program system contains a main program, MAIN, written in FORTRAN, and an assembly routine, AROUT, called from MAIN. AROUT is called with one parameter, an integer variable, containing a character in the rightmost byte. This character is output to a terminal.

FORTRAN program:

```
PROGRAM MAIN
INTEGER A
A = 4H    X
CALL AROUT (A)
END
```

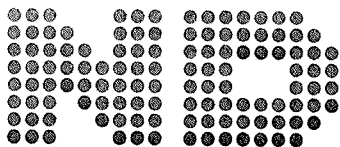
The calling sequence looks as follows in assembly code:

```
RTJ  6, * + 1, 4,, 1
ACN AROUT
1
<address of A>
```

Assembly program:

	REF	AROUT	
AROUT	LDR	11, 0, 6	% R <sub>11</sub> : = address of AROUT
	LDR	32, 2, 6, 0, 1	% R <sub>32</sub> : = A
	STR	32, PAR1, 11	% PAR1: = A
	STOP	2	% OUTBT character
	ACN	PAR0	
	STOP	0	
	ADD	6, 1, 6	% R <sub>6</sub> : = R <sub>6</sub> + no. of parameters
	RTJ	0, 2, 6	% return
PAR0	GCN	1	
PAR1	BSS	1	
	END		

See also NORD-50 FORTRAN Reference Manual (ND-60.095.02), Appendix E.2.



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

## COMMENT AND EVALUATION SHEET

US50 Course Manual for NORD-50 Software Courses

August 1979

Publication No. ND-60.118.01

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 preaddressed form and mail it. Please be specific wherever possible.

FROM .....

.....

.....

**– we make bits for the future**