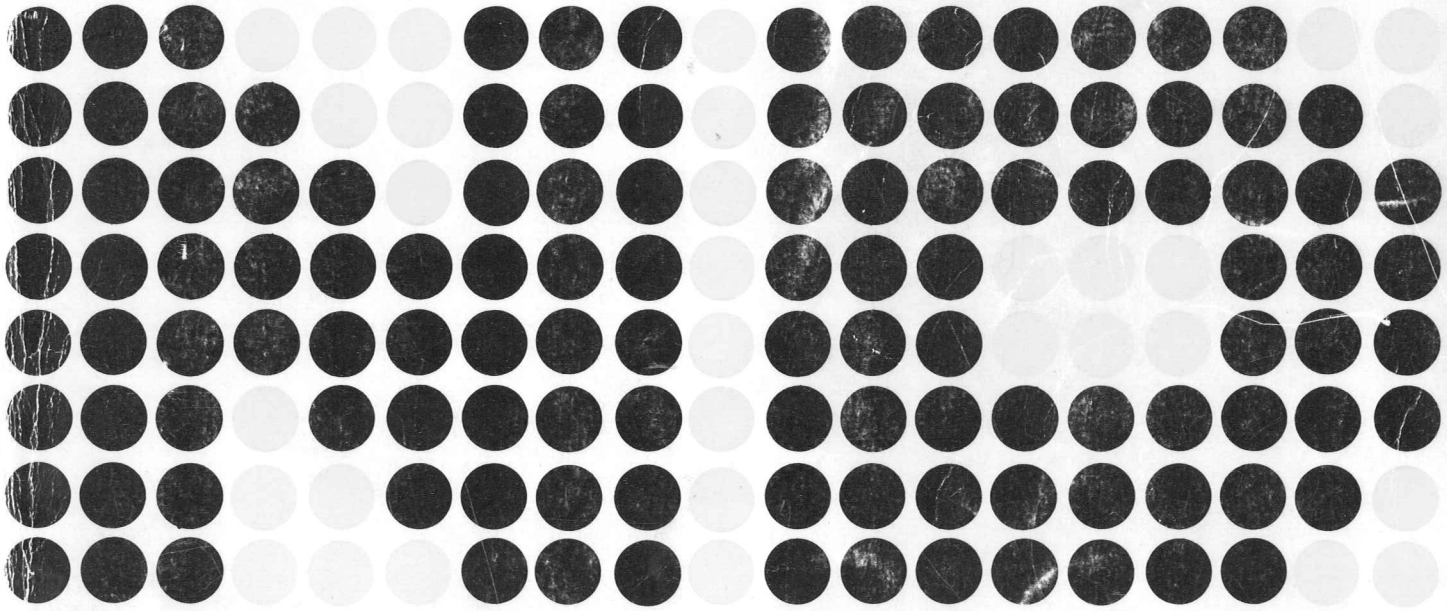
SINTRAN III

Users Guide

A/S NORSK DATA-ELEKTRONIKK



REVISION RECORD

Revision	Notes
01/75	Version 03
	Total revision, superceding all previous versions
01/76	Version 04
	Total revision, superceding all previous versions
03/76	Version 05
	Total revision, superceding all previous versions
06/76	Version 06
	Total revision, superceding all previous versions

Publication No. ND-60.050.06
June 1976

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

PREFACE

Norsk Data Elektronikk A/S is continuously working towards improvements and new extensions of the SINTRAN III operating system. New versions are released 3 or 4 times a year. It is our goal to synchronize the new editions of this manual with the new versions of the operating system, such that all functions of a particular version of SINTRAN III are described in the corresponding edition of the manual, and that all functions mentioned in the manual are contained in the corresponding version of the operating system.

Thus, this version of this manual is the SINTRAN III version of June 1976.

TABLE OF CONTENTS

+ + +

1 INTRODUCTION

This manual is intended to give a brief description of the present operating system, SINTRAN III, which in a version especially generated and strictly suited for each particular NORD-10 configuration, is delivered together with other standard software packages by A/S Norsk Data-Elektronikk.

The reader of this manual will learn all the details about running SINTRAN III, how to make efficient application programs utilizing standard hardware and software features, and will be given a general idea of the structure of SINTRAN III.

Accordingly, this manual is recommended as the necessary documentation for any person (operator or programmer) approaching the problem of utilizing the efficient hardware and software facilities offered by the NORD-10 computer system.

1.1 *GENERAL REMARKS ON SINTRAN III*

The NORD-10 computer system is a medium scale, general purpose computer system which, because of the modular design, is actually a family of computer systems.

SINTRAN III is a multiprogramming, multi-lingual, real-time operating system that supervises the processing of user programs submitted to a NORD-10 computer system. SINTRAN III also relieves the user from program control, input/output and housekeeping responsibilities by monitoring and controlling input, loading, compilation, run preparation, execution and output of user programs.

SINTRAN III offers the user the most efficient facilities because it takes full advantage of the NORD-10 computer hardware resources. Many powerful operating system features are made possible by utilizing the efficient hardware of the NORD-10. This, in turn, makes multi-programming in real-time, timesharing and batch modes possible.

The system is highly modular and may be used for a wide range of NORD-10 configurations. Modularity allows memory resident systems of only 16K, expanding to mass storage systems including 256K main memory, disks, drums, etc. and connections to other NORD computers, thereby, allowing multiprocessing systems.

The philosophy behind SINTRAN III makes it especially suited for:

- Process Control Systems
- Businesss Oriented On-line Systems
- Scientific Engineering Timesharing Systems
- Data Communication Systems
- Data Acquisition Systems

and combinations of these processed concurrently. This, because of the subsystems offered under SINTRAN III, helps to ease the user's implementation of applications.

Operating Modes:

SINTRAN III allows users to run real-time, timesharing and batch programs concurrently.

Time critical real-time processing always has higher priority than timesharing and batch processing. The number of programs that can be processed concurrently depends on factors such as; hardware configuration, operating modes and applications involved.

1.2 *USER CATEGORIES*

The SINTRAN III system may be used by a variety of people, ranging from those who want to run BASIC programs from interactive terminals, to those who want to control complicated equipment through the real-time processing facilities the system offers, and those who are responsible for the control of the computer system itself.

These different users may be categorized into three groups:

- timesharing and batch users,
- real-time users, and
- system supervisors

Each group will have a different background and require different services by the system. This manual has been separated into sections describing functions concerning all user groups to sections with special interest for each group.

Another type of users, often referred to as the "parametric users", perform application oriented tasks such as data-entry or data-inquiry through real-time/timeshared terminal programs. These users will not be discussed in this manual.

1.2.1 *Timesharing and Batch Users*

This user group will normally access the system through a terminal or by submitting jobs for batch-processing from disk files containing the necessary commands and data or card decks to be entered by an operator.

Chapters 2 and 3 of this manual describe all commands and services available together with some examples of how to run various subsystems.

Timesharing and Batch users are known to the system by names which must be presented at the beginning of a terminal session or as the first command in a batch job.

The user name may be protected by a password chosen by the user to prevent unauthorized access to the system.

The user name must be established by those responsible for the system. Timesharing and Batch users access the system in its "background" processing mode, executing programs with a lower priority than time critical real-time programs executing in the "foreground" processing mode. The timesharing and batch processes are normally given processing time on an equal basis in a round-robin fashion.

During processing, the user may take advantage of the many functions provided by the SINTRAN III system, including an efficient FILE MANAGEMENT SYSTEM for saving programs or data in permanent or removable file-directories on disk packs, magnetic tapes and cassettes.

Certain commands related to real-time processing and for use by the system supervisors are not accessible by a timesharing/batch user, and if given, will issue an error message. These commands are described in Chapters 5 and 6.

1.2.2 *The Real-Time Users*

The SINTRAN III system recognizes a special user name RT as a privileged user capable of controlling the real-time processing programs. This user name should also be protected by a password to prevent unauthorized access.

Chapter 5 describes the special commands and services. Chapter 4 describes topics such as; Priority Levels, Memory Management, Program Protection, Program Structures, Reservation of logical peripherals and units, and Communication between RT programs. Chapter 7 gives some examples of RT programs and how to load and activate their execution.

The user RT may also use the Timesharing and Batch facilities for program development and testing purposes.

Certain commands related to system supervision are not accessible by user RT and, if given, will issue an error message.

1.2.3 *The System Supervisor*

The SINTRAN III system recognizes the user name SYSTEM as the user with the most privileged possibilities, capable of creating and deleting users, creating file directories on different devices and entering them so that other users may access files. The user name SYSTEM should also be protected with a password.

Chapter 6 describes special commands and services, together with other information related to this user.

The user SYSTEM may also use the Timesharing and Batch facilities for program development and also has the user RT's capabilities.

1.3 *COMMANDS AND MONITOR CALLS*

The SINTRAN III system provides its users with a large set of control functions which may be activated by a command or from a program by a Monitor Call.

A *Command* may be entered from a terminal or a batch job file to be checked for validity, and cause the appropriate action to take place. The commands consist of three groups:

Background Commands

- calling compilers, assemblers, editors, other subsystems and user programs
- creating, deleting and maintaining files
- reserving and releasing peripheral devices, etc.

These commands may be used by all users.

Real-Time Commands

- starting and stopping RT programs
- loading new RT programs using the RT loader
- reserving and releasing peripheral devices on behalf of RT programs, etc.

These commands may only be used by the users RT and SYSTEM .

System Commands

- starting and stopping the system
- creating, entering and releasing file directories
- creating and deleting users
- allocating resources, etc.

These commands may only be used by the user SYSTEM .

All commands are described in Chapters 3, 4, 5, and 6 and a complete list is given in Appendix A.

A *Monitor Call* is a special hardware instruction allowing a user to invoke a large set of service functions, to be executed within the operating system on behalf of the calling program.

Also, some hardware instructions have restricted usage within a multi-programming, multiterminal operating system. Such instructions have a corresponding Monitor Call function.

As Monitor Call instructions can only be activated through programs written in MAC assembly or NORD PL languages, a set of small subroutines is available for programs written in FORTRAN. Sections 3.5 to 3.7, 7.6 and Chapter 9 give a detailed description of all available Monitor Call functions. A list of all Monitor Calls is given in Appendix B.

1.4 *HARDWARE ENVIRONMENTS*

The minimum hardware configuration required to run SINTRAN III is:

- NORD-10 standard CPU, including 16K words of main memory
- Console terminal
- Paper tape reader

The range of standard peripherals includes paper tape reader and punch, card reader and punch, Teletypes, alphanumeric and graphic display systems, line printers which print 60 to 1500 lines per minute, matrix plotters/printers, cassette tape input/output, magnetic tape stations, 7-track and 9-track, fixed head drums, cartridge disks, A/D-D/A equipment, facsimile transmitting interface modem interfaces and CAMAC interface.

Optional hardware:

- main memory up to 256K words, both core and solid state memory in the same system. Multiport access.
- high-speed direct memory access channel 1M word/sec. in interleaved processing.
- up to 4 mass storage controllers (independent of types) used for system and user program storing.
- no limitation in number of mass storage controllers and types for file and data storage.
- moving-head cartridge disks. Up to 4 cartridge disk units per controller and 4.6 or 9.2 Mbytes per unit. Average access time 47.5ms, 156K word/sec. transfer rate.
- moving-head disks. Up to 8 disk units per controller with a capacity of 3M or 66M bytes per unit. 38.5ms average access time, and 600K words/sec. transfer rate.
- fixed head drum. One drum unit per controller with capacity from 64K to 1024K words per unit. 10.5ms average acesss time, and 100K words/sec. transfer rate.
- magnetic tapes up to 4 units per controller, 7-track, 45 ips, 200, 556 or 800 bpi, 9-track, 75 ips, 800 or 1600 bpi.
- magnetic cassette tapes.
- card readers 285 or 600 cards/minute.

- NORDCOM graphic and semigraphic colour display systems.
- line printers. Prints from 200 to 1100 lines per minute.
- terminals. Hard copy: 10 to 120 characters per second. CRT screen: 10 to 960 characters per second.
- graphic plotters and displays.
- data communication interfaces.
- paper tape readers and punches.

For further information, please contact Norsk Data A.S.

1.5 *AVAILABLE SUBSYSTEMS*

SINTRAN III offers many programming languages, subsystems and utility programs which are extremely efficient tools for the users of the system.

1.5.1 *Programming Languages*

To let the user implement his applications as easily and economically as possible, SINTRAN III accepts programs written in the following languages:

STANDARD FORTRAN

follows ANSI STANDARD FORTRAN and has ISA Real-time Extensions. The user may call subroutines written in NORD PL and MAC from his FORTRAN program. FORTRAN programs may be executed in all three modes of operation. FORTRAN also has an interactive debugging facility.

NODAL

is an interpreting, higher-level, interactive language especially suited for process control applications. NODAL may be executed in all three modes of operation and may call subroutines in NORD PL and MAC.

BASIC

is an interpreter following Dartmouth College 71 specification. From his BASIC program, the user may call FORTRAN, NORD PL and MAC subroutines. Programs written in BASIC may be executed in timesharing and batch modes.

NORD PL

is a medium level language especially suited for system programming. By using the NORD PL, the programmer will more quickly write and debug programs, more easily modify them, and make them more reliable and easier to read and understand than using the traditional assembly language. SINTRAN III is written in NORD PL.

MAC

is an assembly language and debugging package for the NORD computers.

Each language translator is accessed by a unique SINTRAN III command.

1.5.2 *Subsystems and Utility Programs*

The following subsystems and utility programs are available to a SINTRAN III user.

FILE MANAGEMENT SYSTEM

SINTRAN III offers the user of mass storage systems a general purpose file management system for use of permanent files, scratch files, and peripheral device files. The system provides a flexible file security mechanism that allows the programmer to specify the degree of security desired. The files may be accessed in sequential or random mode.

SIBAS

is a data base system where efforts are heavily placed on the users' possibilities of representing complex data structures and on the separation of application programs from the data base. SIBAS is an extensive tool for applications in business oriented on-line systems and ADP systems. SIBAS data handling routines follow the specification given in CODASYL DATA BASE TASK GROUP, APRIL 71 REPORT.

RT LOADER

enables the user to load RT programs in binary relocatable format onto segments while real-time processing is running.

QED

is an interactive program for editing text. It has extensive facilities for inserting, deleting, and changing lines of text, a line editing feature. Text may be read from and written onto any file. QED is extremely efficient for on-line program development.

RUNOFF

The RUNOFF program will help the user to write reports under SINTRAN III by processing the raw text information held in a computer file and provide a printed document of a quality acceptable for publication. The control commands are few and easy to learn.

DDC PACKAGE

The Direct Digital Control packages running under SINTRAN III, MEAS, PROCSY and PROSO give the user extensive tools for implementing process control application in his NORD-10 computer. The packages control and process analog signals and perform conventional PID, cascade, ratio and other regulation functions.

NORD IDT

The NORD Intelligent Data Terminal programs allow the user to communicate with Honeywell Bull 6000, IBM 360/370, CYBER 74 and Univac 1108/1110 machines through remote job entry terminal simulators.

In addition, subsystems also include scientific and statistical program libraries such as, the Scientific Subroutine Package (SSP) written in FORTRAN.

2 PREPARING AND EXECUTING TIMESHARING JOBS

This chapter gives a few examples of how to run a timesharing job from your terminal.

The first step of a job is always to log in, the last is to log out. These steps are described in Section 3.3.1.

The steps in between may be very simple, i.e., calling a processor of some kind with the RECOVER command (see Section 3.3.2.1).

2.1 *A RUN USING QED, THE TEXT EDITOR*

1. Turn on the terminal. Turn on the on-line switch.

2. Press "Escape" (the ESC key).

3. The terminal responds with the time of day, the date, and the word ENTER.

4. Type the user name, followed by cr (carriage return). You must be known by the system as a user before you may enter.

5. The terminal responds by printing PASSWORD.

6. Type your password. If you have none type cr. Remember that the password will not be echoed on your terminal.

7. The terminal responds with OK.

8. If the accounting system is active the terminal will print PROJECT-NUMBER. Answer this by typing a project number (a decimal number), followed by cr.

9. The terminal prints the character @. This means that it is expecting a command.

10. Type QED then cr (or RECOVER QED cr).

11. The terminal responds with

    QED 3.4
    *

    This means that QED is ready to accept commands. (Refer to the QED Users Guide for details.)

12. As a very simple QED run, we will type a few lines of text, list them on the terminal, and write them on a file. Type A cr (this is the append command). Type the following lines:

    THIS IS MY FIRST SYMBOLIC FILE cr
    I AM GOING TO PUT IT ON A DISK FILE, cr
    WITH THE NAME I-MADE-IT cr
    (CTRL)L

    When you finish a line with cr, QED responds with lf (line feed). (CTRL)L is a control character which finishes your text. It is typed by pressing the control key and the L key simultaneously. It will not appear on your terminal.

13. QED responds by printing *.

14. Type L1, $ cr and all your text will be printed on the terminal. QED will finish it by printing *.

15. Type W"I-MADE-IT" cr. The quotation marks tell QED that this is a new file. If the file already exists, the quotation marks must not be used.

16. QED creates the file and writes the text on the file. Later, it prints on your terminal:

    48 WORDS WRITTEN
    *

17. Type F cr, and you will leave QED.

18. The terminal responds by typing @. You have now left QED and have returned to the system.

19. Type LOG cr. This is the log-out command.

20. The terminal responds by typing the time of day, the date, and – – EXIT – –.

21. We are now back where we started.

2.2 *A RUN USING BASIC*

1. To log in, follow steps 1 to 9 in Section 2.1.

2. Type BASIC cr.

3. The terminal responds

   BASIC ON LINE
   NEW OR OLD – – –

4. Answer by typing NEW cr.

5. The terminal responds

   NEW PROGRAM NAME –– ––

6. Answer by typing OLA cr or another name of your choice.

7. The terminal responds

   READY

8. Type on the terminal

   10 PRINT "THIS IS MY FIRST BASIC JOB" cr
   20 END cr
   RUN cr

9. The terminal prints

   THIS IS MY FIRST BASIC JOB
   DONE

10. To leave BASIC, type BYE cr

11. The terminal prints

    EXIT BASIC

12. Type LOG cr, as in steps 19 to 21, Section 2.1.

This is a very simple BASIC job. If you wish to do more complicated jobs refer to the BASIC Reference Manual.

2.2.1 *The Execution of a BASIC Program from a QED File*

It is possible to have a BASIC program on a disk file (written by QED – see Section 2.1). The above example should be changed as follows (assume that the disk file is named COMPLIC):

4. Answer by typing OLD cr.

5. The terminal responds

   OLD FILE NAME _ _

6. Answer by typing COMPLIC cr

7. The terminal responds

   WAIT FOR READY _ _

   The READY message is given after the file has been fetched.

8. Type on the terminal

   RUN cr

2.2.2 *The SAVE and GET Commands*

Files may also be manipulated directly from BASIC. If you have typed in a BASIC program, you can save it on a disk file by the SAVE command. SAVE "file-name" cr if it is a new file, or SAVE file name cr if the file already exits.

A file may be fetched from the disk by the command GET:

GET <filename> cr

2.3 *AN FTN (FORTRAN) RUN*

For a more detailed description of the FTN compiler and the binary loader (BRL), refer to the appropriate manuals.

It is assumed that the following symbolic FTN program is already on a disk file called MARY. It can be put there by QED, see Section 2.1:

```
      PROGRAM MARY
      WRITE (1,10)
10    FORMAT (*1 MARY HAS BEEN EXECUTED *)
      STOP
      END
      EOF
```

1. Log in as described in steps 1 to 9 in Section 2.1.

2. Type FTN cr.

3. The terminal responds with

   ```
   NORD FTN
   $
   ```

4. The dollar sign means that the FTN compiler is ready to accept commands. Type:

   COM MARY, TERM, "MARY" cr

5. The compiler will now compile your program, list it on the terminal, and put the resulting BRF code (Binary Relocatable Format) on a new file called MARY (or MARY:BRF).

   If this file already exists, the quotation marks should not be used as they are used only when starting a new file. Note that the file MARY with your symbolic FTN program has the file type SYMB (MARY:SYMB) so the system can distinguish between the two MARY's. See Section 3.2.3 for further explanation.

   In some systems, the terminal is not named TERM, but TELETYPE. In this case, the command in step 4 above should be

   COM MARY, TELE, "MARY" cr.

6. After compilation, the FTN compiler prints:

   ```
   6 STATEMENTS COMPILED
   $
   ```

   on the terminal.

7. Type EX cr to leave the compiler.

8. The terminal responds by printing @.

9. Type LDR cr.

   This is a command to fetch the loader (BRL, expanded by the run-time system).

10. The terminal responds by printing

    BINARY LOADER
    L*

11. L* means that the loader is ready to accept commands. Type A MARY cr

    and your program will be loaded. After loading, the loader prints L*.

12. If you wish you may type:

    W1 cr

    The loader responds by printing a table of entry points (in this case MARY = 060000), the address of the first free location, and the address of blank common.

13. Type S cr to start your program. The terminal will print

    MARY HAS BEEN EXECUTED
    octal address STOP 0
    @

    You are then back in the system.

14. Type LOG cr as in steps 19 to 21 in Section 2.1.

2.3.1 *FORTRAN Compiler Commands*

The set of FTN compiler commands includes, among others, CLC and DEBUG.

CLC cr, or
CLC octal number cr,

means if you list your program during compilation, every FTN statement will be preceded by an octal address, starting with the number in the CLC command. CLC cr is equivalent to CLC 0 cr.

This may be very useful for debugging purposes when the program is loaded. By looking at the entry point list from the loader (W1 command) and your compilation list, you will be able to locate every FTN statement in memory.

Some error messages (run-time errors) print the octal address at the beginning of the FTN statement where the error occurred. By means of the debugging routine DEBUG it is possible to execute the FTN statements, one by one, to introduce breakpoints and trace functions, to display the contents of the different variables and to change their values by referring to their names, etc. These functions are performed dynamically at run-time.

The debugging routine must be introduced as a reference point by the command DEBUG at compile time and loaded by the loader with the command A DEBUG.

This routine is most useful for FTN debugging. See the NORD STANDARD FORTRAN REFERENCE MANUAL (ND-60.011) for details.

2.4 *THE COMPILATION OF A SIMPLE NORD-PL PROGRAM*

Suppose the following simple NORD-PL program has been written onto the file NPL1 with the QED processor:

```
SUBR SORT                              % START OF SORT
INTEGER I1, I2, I3, I4
INTEGER ARRAY II(12)
SORT:A:=0; * MON 4                     % SET BREAK MODE
0=:I1
11=:12
FOR I1 TO I2 DO
  T:=1; *MON 1; MON 65                 % INPUT 1 ELEMENT
  A BZERO 7                            % CLEAR PARITY BIT
  A=: II(I1)
OD
0=:I1
10=: I2
FOR I1 TO I2 DO                        % PERFORM SORTING
  I1 + 1 =:I3
  FOR I3 TO I2 + 1 DO
    IF II(I1)<II(I3) THEN
      A:=II(I1)
      T:=II(I3)=:II(I1)
      A=:II(I3)
    FI
  OD
OD
0=:I1; I1=:I2
A:=15; T:=1; * MON 2; MON 65           % CARRIAGE RETURN
A:=12; *MON 2; MON 65                  % LINE FEED
FOR I1 TO I2 DO
  A:=II(I1)
  T:=1; *MON 2; MON 65                 % WRITE 1 CHARACTER
OD
*MON 0                                 % RETURN TO SINTRAN III
RBUS                                   % END OF SORT
@EOF
```

The program will read 10 characters from the terminal, sort them in descending order by the ASCII code, and write them out on the terminal.

The programmer now wants to compile the NPL program, get a program listing on the terminal and write the object program on the symbolic file MAC2. This is done as follows:

1. Log in. Do as in steps 1 to 9 i Section 2.1. The terminal prints @.

2. Type NORD–PL cr. The terminal answers:

   NORD–PL 74.12.07.

3. Type @DEV NPL1, 1, MAC2 cr

4. The compiler now prints the symbolic NPL program on the terminal and writes the symbolic MAC program (object program) on the file MAC2. Error messages are printed on the terminal.

The MAC assembly program should now be assembled by the MAC assembler and started as a MAC program.

For a complete description of the facilities offered by the NORD-PL compiler, see the manual NORD–PL Users' Guide.

2.5 *A RUN USING THE MAC ASSEMBLER*

In the following, we will use as an example, a program (written in MAC) that

1. outputs a question mark on the terminal
2. reads a file name from the terminal
3. opens the file
4. copies the file to the terminal
5. exits when it encounters the end-of-file character (027).

We assume this program is already on a file written by QED. The file has the name EXAMPLE. This file should be written in an orderly and readable manner.

Before writing the program on the file, one should use the QED-command MTO(0) to eliminate the tab characters, as our program EXAMPLE does not expand tab characters.

Here is the program.

```
TOR,    SAA     ##?     % ASCII CODE OF ?
        SAT     1       % LOGICAL UNIT NUMBER
        MON     2       % OUTPUT ONE BYTE
        MON     065     % ERROR MESSAGE
        SAX     0       % READ FILE-NAME FROM TERMINAL
        SAT     1       % OPEN FOR SEQUENTIAL READ
        LDA     PER     % ADDRESS OF FILE TYPE
        MON     050     % OPEN FILE
        MON     065     % ERROR MESSAGE
        STA     FILIN   % SAVE THE FILE NUMBER
AGAIN,  LDT     FILIN   % READ A CHARACTER FROM
        MON     1       % THE FILE
        MON     065     % ERROR MESSAGE
        AAA     -027    % END-OF-FILE CHARACTER?
        JAF     *+2     % NO
        MON     0       % YES. RETURN TO SINTRAN III
        AAA     027
        SAT     1       % WRITE THE CHARACTER ON
        MON     2       % THE TERMINAL
        MON     065     % ERROR MESSAGE
        JMP     AGAIN   % TAKE NEXT CHARACTER
PER,    FTYPE
FTYPE,  #SY
        #MB
FILIN,  0
)WRITE  TOR FILIN
)LINE
```

If some of the points below are not clear to you, consult your MAC Manual.

1. Log in as in Section 2.1 – steps 1 to 9.

2. Type MAC cr.

3. When MAC is ready to accept input from the terminal, it prints cr lf.

4. Type)CLEAR cr to clear tables, etc.
   Type (WRTM cr to set MAC in write mode.
   Type 040000/ to set current location counter to 040000.

5. MAC responds by printing the contents of location 040000 on the terminal.

6. Type)9ASSM EXAMPLE cr.
   MAC will now assemble the file EXAMPLE.

7. When assembly is finished, MAC prints

   TOR:040000 FILIN:040030

   This means that the instruction labelled TOR is in location 040000, and the constantly labelled FILIN is in location 040030.

8. Type *:
   MAC responds by printing 040031. This means that current location counter is now 040031, i.e., your program occupies locations 040000 < 040030.

9. Type ?
   MAC responds by printing cr lf. This means that there are no undefined symbols, i.e., the assembly seems to be correct.

   If there had been undefined symbols, or labels, say LAB1, you could type LAB1 and get the octal reference addresses. This would show you the address of the instruction or constant that referred LAB1.

10. Your program may be executed if you type

    TOR! or 40000! (40000 is the default start address)

    But if you want to execute the program many times, you should dump it on a disk file as follows:

11. Type)9TSS

    The terminal responds by printing @.

12. Type DUMP "EXAMPLE" 040000 040000 cr

    Now your program will be dumped on a new file called EXAMPLE:PROG, with start address 040000. The start address is used when you apply the RECOVER command, and the restart address is used after the program has been interrupted (or it has terminated) and you type CONTINUE.

13. After the dump is finished, the terminal prints @. Type

    EXAMPLE cr

    and your program is fetched and entered. It will print ? on the terminal.

14. You may now copy a file to the terminal. Type

    EXAMPLE cr

    and you will get EXAMPLE:SYMB listed on the terminal.

    If you should type the file name incorrectly, you will get an error message and return to the system (because of MON 065). To try once more, type CONTINUE cr.

15. After the execution of the program, you might like to see what the contents of FILIN (the file number of the opened file) was. Type:

    LOOK-AT CORE cr.

16. The terminal responds READY:

17. Type 040030/.

18. The terminal responds 101 which is an octal file number. Type @ to get back to the system.

19. If you would like to see the contents of your registers when execution has finished, type

    STATUS cr

    and you will get a list of the register contents (A should be zero!).

20. Finally, log out as in steps 19 to 21 in Section 2.1.

3 TIMESHARING AND BATCH USERS

3.1 *INTRODUCTION*

This chapter is related to the user category which uses the SINTRAN III system for program development, testing and executing programs in the background environment of the system.

The information in this chapter relates also to the users RT and SYSTEM, though their special tasks are described in Chapters 5 and 6.

This chapter describes the various services the users may obtain from the SINTRAN III system, and is divided into sections covering the many Commands, Monitor Calls and Utility functions the users may activate to aquire these services.

3.1.1 *Timesharing Users*

This name has been used to cover all users who activate the SINTRAN III system from an interactive terminal. This activation may consist of: typing commands to start execution of a program (whether it is a private program or a subsystem provided by ND), debugging the program by inspection and change of locations within the user's virtual memory space, or by typing source program lines through the editor QED for later compiling or assembling into a running program unit.

All these tasks may be executed in a multiprocessing, multi-programming environment with many simultaneous users at their terminals, even local or remote batch processing, all requiring an equal share of the main system resource: the Central Processing Unit.

This resource is divided among the requestors so that available CPU time, after high-priority real-time processes have been executed, is given to each user in a short fraction of time, a *time-slice,* where the system is devoted to his task. As most users are not able to fully utilize the CPU for a complete time-slice, due to input or output of data from or to files or peripheral devices, this time-slice may be terminated by the system so that other users may have the CPU while the system performs the tasks of reading or writing data to and from the program.

In this way, the system will share the CPU resource between active terminals and other processes, and give each terminal a priority level which may later be used when selecting the process to be activated from those that are waiting and ready to continue their task.

The process described above is handled by the SCHEDULER part of the SINTRAN III system, where terminals are assigned a priority of 60, 50, 40 or 20, (octal) within a priority scheme ranging from 377 (octal) as highest priority to 0 as lowest priority.

A priority of 60 is assigned to a terminal process when typing a control character such as carriage return. The process is also given a time-slice of 1 second, thus, securing a fast response to the interactive user. If this is not enough, the user will receive 4 seconds more, but now on priority 50. If his program is still running, it will then be considered to be a CPU-bound program and put into the *time-slice queue*. Each program in the time slice queue will, in turn, have a time-slice with priority 40. The other programs waiting for time-slice will have priority 20. The size of the time slice may vary from 4 to 14 seconds, depending on the terminal activity.

The figure below illustrates the priority scheme:

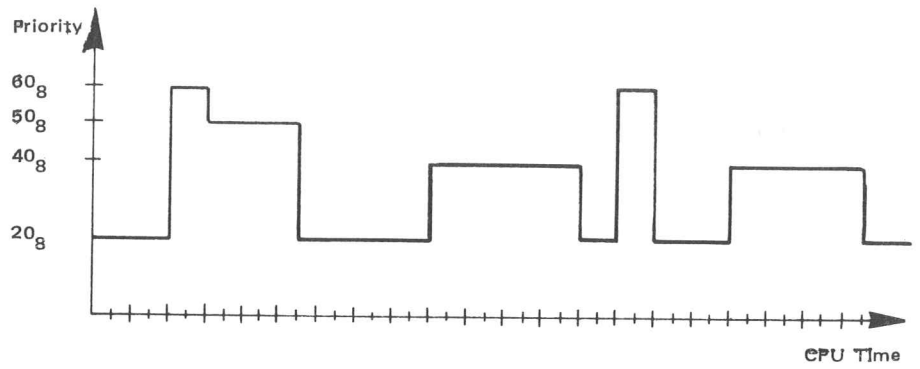


Figure 3.1: *The Change in Priority for Timesharing and Batch Jobs*

3.1.2 *Batch Users*

This name has been used for users executing their tasks in different *Batch Processors*. The SINTRAN III system may have one or several independent Batch Processor tasks, each processor having its own priority.

A user may append *Batch jobs* to these processors by building *Batch-Input-Files* on disk or punched cards.

A Batch-Input-File contains all necessary commands and user data, and may contain one or more tasks. A *task* is, for instance, compiling a FORTRAN program. Another task could be loading and executing the program, etc. Tasks are activated with the same type of commands as for the interactive terminals.

As batch jobs are processed independently of the ongoing terminal activity for the same user, two special commands should be used to identify the user and to terminate the Batch-Input-File.

Batch-Input-Files on disk may be appended by the terminal user to the Batch-Processor-Queue with the command @APPEND-BATCH, or a deck of cards may be given to an operator.

A detailed description of the Batch-Processor and related commands is given in Section 3.7 – Batch Processing.

Almost all commands may be executed within a Batch-Job except those that are interactive. The commands themselves, including necessary parameters, must be contained on a single line or card.

3.2 *THE FILE MANAGEMENT SYSTEM*

This system, although it may be viewed as a separate system, is an integral part of the SINTRAN III, and adds powerful file management functions normally found on large computers.

A complete description is given in the manual NORD FILE SYSTEM, ND-60-052.

A short description of the most common commands is given in Section 3.5, and some special commands for the user SYSTEM is given in Chapter 6.

A short introduction to some of the file concepts is given below.

A *'file'* in this context means a collection of records or blocks, ordered randomly or sequentially. The File Management System manipulates files on disks, drums, magnetic tapes, cassette tapes or standard peripherals. Files on disks, drums and magnetic tapes are treated in a uniform manner. The storage unit is always 1024 words (2K bytes), however, the user may address files with any other block size.

A file is named with a character string, and this name is used in all commands to the file system. When a file is accessed, the file name must be connected to a file number and this number is used in the access routines.

Each file has one *owner* who has to be defined as a user in SINTRAN III. The owner is normally the user who created the file. A file is always allocated in the owner's area on the mass storage device (directory). Each user may declare up to eight other users as *friends* and give them privileged access possibilities to his files. Other users are regarded as *public users.*

The File Management System provides individual protection of files, with separate protection modes for the owner, owner's friends and the public users access of the file.

3.2.1 *File Directories*

Files are collected into *file directories,* containing files for one or more users. Each mass storage device maintained by the system (disks, drums and magnetic tapes) may be used to contain a file-directory, and is completely independent of all other devices. File-directories contained on removable media as disk packs or magnetic tape reels may be moved to other installations and used there.

Each device medium (pack or tape reel) to be used as a file-directory must be created with a name and entered before it can be used by the system. When a directory is not needed anymore, it must be released and dismounted. The next time it is needed, it must be mounted and entered again, etc.

The first file-directory entered (usually where SINTRAN III resides together with related subsystems) is regarded as the *main directory*. The main directory is the last directory that can be released, and it cannot be released if any users are logged in.

3.2.1.1 CREATING USERS AND RESERVING SPACE

Before a user can establish files in a directory, the user name must have been defined in that directory by the @CREATE-USER command, and space must have been reserved by the @GIVE-USER-SPACE command. Such commands, which manipulate file-directories, are restricted so that only the user SYSTEM may activate them.

Users may be created on several file-directories, but must have been created on the residing main-directory. All information about a user's friends and their access privileges is kept in the main-directory, together with the user's declared password. Before logging in on a terminal or starting a batch-job, the user name must have been declared in the main-directory, but need not have space reserved on that directory.

3.2.1.2 DEFAULT DIRECTORIES

Any directory entered may be declared as a *default directory* by the command @SET-DEFAULT-DIRECTORY. This means that the users need not specifiy the directory name when creating or accessing files in this directory. A user should not be given space in more than one default directory. If he has, he must still specify the directory name as a prefix to the file name. Main directory is always default.

3.2.2 *Creating Files (Indexed and Contiguous Files)*

A file may be created for the first time by the @CREATE-FILE command or by, for instance, the @OPEN-FILE or the @DUMP commands, with the file name surrounded by quotation marks. All subsystems and user program files may be created in the same way by simply enclosing the name in a pair of quotation marks.

Files created by the @CREATE-FILE command with the number of pages greater than zero are allocated on a contiguous area on the mass storage device and are defined as *contiguous files*. Such files may only be given more pages by means of the @EXPAND-FILE command.

Files created by the @CREATE-FILE command, (or created in other ways) with the number of pages equal to zero, may have their pages scattered throughout the mass storage device and are defined as *indexed files*. The size of these files may be expanded by new pages, dynamically, as the user writes onto the file. The indexes are kept on a separate page belonging to the file, so that an indexed file always needs one page more than a contiguous file with the same contents. Indexed files cannot be expanded by the @EXPAND-FILE command.

See also Section 3.4.1.

3.2.3 *File Types*

In addition to the file name, an alphabetical *file type* is added to the file name to designate the purpose of the file. This file type may be set by the user as a 4 character extension of the file name, separated by a colon (:). The following file types are used as default when creating/accessing files in various subsystems:

:SYMB	symbolic program file
:BRF binary	binary relocatable file
:PROG	absolute program in executable format
:CORE	core-image file
:BIN	binary absolute program
:DATA	users data files

With these file types, a program may take advantage of the same name for all different files. The various subsystems will access the proper file as its input or output file when the file name is given. (See the example in Section 3.2.7.) Otherwise, the user is free to introduce any other file type when he creates a new file.

3.2.4 *File Versions*

Files may also be created in one or more *versions*, that is, complete copies of the file, so that the last version written is version 1, the previous version is 2, etc. When a file is created in more than one version, the operating system will access the one with the highest number when opened for write (and afterwards change the version no. to 1) and the version 1 when opened for read.

The user may also specify a certain version of the file to be accessed by appending the version number to the file name and file type, separated by a semicolon (;). New versions of an existing file may be established by the command: @CREATE-NEW-VERSION <file name> <number of pages>, or by printing the version number within quotes by, for instance, the command: @OPEN-FILE FILE1:DATA; "4" ,R.

The command @CREATE-FILE FILEX;3,10 will create 3 versions of the file FILEX – each version with 10 pages.

3.2.5 *File Protection and Accessing Modes*

A file may be accessed in many different ways. The access mode of a file is specified as a parameter to the OPEN-FILE command, and may be a combination of the following characters:

R read access
W write access
X random access
A append
C common access (write allowed for more than one user simultaneously). Only allowed on contiguous files.

Only certain combinations are legal. See Sections 3.4.5.1 and 3.5.1.1.

The access mode may be restricted, either due to physical reasons or because the file is protected by some access modes. Legal access is defined for the three groups of users: Owner, Friends and Public users.

The legal access to a file may be changed by the @SET-FILE-ACCESS command for each user group, and may be a combination of the following characters:

R read permitted
W write permitted
A append permitted (the file may be expanded)
C common access permitted
D directory access permitted (the file may be created, deleted, legal access mode changed and new versions created)
N no access permitted

Default file access when a file is created is:

for PUBLIC: read permitted
for FRIENDS: read, write and append permitted
for OWNER: all

3.2.6 *Friend's Access*

To allow a group of users access to his files the owner must have defined these users by the @CREATE-FRIEND command. Each defined friend may be declared with different access restrictions by the @SET-FRIEND-ACCESS command, and may be a combination of the following characters:

R read permitted
W write permitted
A append permitted
C common access permitted
D directory access permitted, or
N no access permitted.

Thus, a friend's access to a file is defined both by his access allowed in general (with the @SET-FRIEND-ACCESS command) and by the friend access permitted to the file in question (defined with the @SET-FILE-ACCESS command).

When accessing a file belonging to another user, the file name must be preceded by the user's name enclosed in parenthesis. If the file in question does not reside on that user's default file-directory, both the file-directory name and user's name must precede the file name, the two first names are separated by a colon and enclosed in parenthesis.

Owners accessing files in other than the default directory must also let the file-directory name precede the file name, and the directory-name must be enclosed in parenthesis.

Default friend access when a friend is created is: read, write and append. See also Section 3.4.2.

3.2.7 *Summary of File-Name*

A complete file name may be a construction like:

(<file-directory-name>:<owner-name>) <filename>:<type>; version

where file-directory-name, owner name, and filename may consist of 1 to 16 characters, filetype may consist of 1 to 4 characters, and version-number range is from 1 to 256.

Example:

(PACK-FOUR:USER-ONE) FILE –3: SYMB;1

The names and file type may be abbreviated as long as it does not become ambiguous, and a special character, the "–", may be used to divide the names into subparts, each of which may also be abbreviated.

3.3 *UTILITY COMMANDS*

When a terminal or a batch job is initiated under SINTRAN III, it is first connected to the Command Interpreting Processor (CIP) which reads each command line from the terminal or batch input device, checks its validity, and performs the requested action. After a particular command is executed, or when a user-program or a subsystem terminates, the control is returned to the CIP.

In this manual, all commands shown are preceded by a @, which is the CIP's leading character.

During a terminal session, this @ is printed automatically to indicate that the previous action is terminated and that SINTRAN III is ready for a new command. Note that this @ character should not be typed by the user.

When submitting batch jobs, or reading commands from other devices than the terminal, all commands must start with the @ character to recognize them as commands.

A command is a string of characters separated into words by a comma or a number of spaces. The first word is the command name, followed by *parameter* words if necessary.

In batch jobs the command name and all parameters must be contained on one line or card. In the terminal session missing parameters will be requested by CIP.

Certain parameters may take a default value which will be used if the parameter word is omitted, by giving the comma separators or by responding with a carriage return when the parameter is asked for.

Command names and parameter words may be abbreviated by giving sufficient characters to distinguish it from other permissible values. A special character, the "–", is used to separate words in two or more distinct parts. Any part may be abbreviated as long as it doesn't become ambiguous.

Consider, as an example, the commands @LOAD-BINARY and LIST-FILE. One way in which the first command may be typed is:

@LOAD
@LOAD-B
@L-BINARY
or
@LO-BI

The second command may be typed as:

@LIST-FI
@L-FIL
or
@LI-FI
etc.

However, if only L is typed, the CIP will respond with the error message:

AMBIGUOUS

This abbreviated look-up function may also be utilized by the user when typing names of files and devices, except when naming a file the first time.

Any typing errors may be corrected before the command line is terminated by the carriage return, as follows:

depressing the CONTROL key together with A deletes one character at a time, responding with → for each, and

depressing the CONTROL key together with Q deletes the whole line, responding with ← and allowing a new line to be typed.

The editing is valid throughout the whole system whenever a type-in is requested.

3.3.1 *Starting and Terminating a Terminal-Session*

3.3.1.1 LOGGING-IN

A terminal is activated by pressing the "escape"-key on the keyboard of the terminal.

The Command Interpreting Processor types out the time and date, and asks for the user's identification (created for him by user SYSTEM) by typing out

ENTER

and waits for the name, terminating with a carriage return. A protecting password, chosen by the user himself, is asked for by typing out

PASSWORD:

and waits for the password to be typed. The password is only read by the CIP, and not typed out clearly at the terminal, to avoid unauthorized access to the system

When the user-name and correct password is checked, the CIP types out

OK.

If the user-name is not defined or an incorrect password is given, the ENTER line is repeated.

A project number is asked for if the accounting system has been initiated by the user SYSTEM. The project number is used to collect the computer time and terminal time a session has lasted.

The CIP types out

PROJECT NUMBER:

and waits for the user to type in the project number this session should be accounted to. When logging out, the duration of this session is typed out.

After this log-in procedure is finished, the CIP types the @ character, and waits for a command to be given.

3.3.1.2 LOGGING-OUT

When the terminal-session is finished, the user should log out from the system by giving the command

@LOGOUT.

The time and date is typed together with the time-used information if the ACCOUNTING system is active. A finished line with

––EXIT––

shows that the terminal is inactive, and power may be turned off.

3.3.1.3 DIALED-UP TERMINALS (DUPLEX, ECHO)

For users that access the system through dialed-up telephone lines, the log-in procedure starts when the high-pitch-tone is heard. The "escape"-key may now be pressed. Terminals are normally connected in full *duplex* mode, which means that the user can type in simultaneously while the system is typing out. Such type-in's are hidden until the system retransmits, *echos,* the received characters in their proper place in the generated type-out. A certain amount of commands or other information may, in this manner, be sent to the system without being lost.

3.3.1.4 PASSWORD

The previously mentioned password may easily be changed by the command

@CHANGE-PASSWORD <old-password>, <new-password>

The old password must be correctly specified to change the password. A password may consist of any characters (including control characters) except carriage return.

3.3.1.5 THE "ESCAPE"–KEY AND THE "BREAK"–KEY

After the log-in is performed, the pressing of the "escape"-key or "break"-key will interrupt a command, a subsystem or a user-program which is running, and control is returned to the CIP.

If a user-program or subsystem was active, a message indicating where the program was interrupted is typed out:

USER-BREAK AT P-$XXXXXX_8$

where XXXXXX is the octal program address to be executed next. All user registers are saved and may be examined by the command @STATUS.

The program may be restarted by supplying the @GOTO-USER command with the address where the program was interrupted. All open files are closed when control returns to the CIP.

3.3.2 *Execution of a User-program or Subsystem*

3.3.2.1 THE RECOVER COMMAND

This command retrieves a program file previously created by the DUMP command, and starts execution of it.

The format is:

@RECOVER <filename>

When starting programs in the user's own file-directory, or subsystems under the user SYSTEM, the word RECOVER can be left out. The file name itself becomes a recovering command.

In other words, instead of typing:

```
@RECOVER MAC
@RECOVER FTN
@RECOVER BASIC
@RECOVER MYPROGRAM
```

one may simply type:

```
@MAC
@FTN
@BASIC
@MYPROG
```

The file type is by default :PROG. The execution begins at the start address specified in the @DUMP command.

The search in the file-directories will be performed as follows:

- the file-directory, where the user giving the command is given space, will be searched first. If not found, the user SYSTEM's file-directory is searched.

- if a user name is specified in the file name, only that user's file-directory will be searched.

Note: PROG files should not be given the same name as a SINTRAN III command, because when the word RECOVER is omitted, the searching in the file directories will be preceded by an investigation of the command list in order to find a command with the given name.

3.3.3 *Creation of an Executable Program*

3.3.3.1 THE DUMP COMMAND

This command saves the contents of the user's virtual memory plus the central registers on the specified file.

The format is:

@DUMP <filename><start address><restart address>

where

<file name>

must be a disk-file, which must be specified in the RECOVER command for later retrieval. The file type is by default :PROG.

<start address>

is the address where the program should begin execution.

<restart address>

is the address where the program should be reentered with the @CONTINUE command.

The amount of virtual memory to be saved may be specified by the @MEMORY command, if not, values set by the previous @LOAD-BINARY, @PLACE-BINARY or @RECOVER commands will be in effect.

3.3.3.2 THE MEMORY COMMAND

This command defines the area to be saved by a DUMP command.

The format is:

@MEMORY <lower-bound><upper-bound>

where the boundaries may be specified by an octal address between 0 and 177777. However, the total size of the area should not exceed 177777.

The command does not affect the virtual memory space for the user.

The current settings of the lower and upper bounds are reset by the last @LOAD-BINARY, @PLACE-BINARY or @RECOVER commands used.

3.3.4 *Restarting Execution of a Program*

A program may be restarted after it has terminated or been interrupted by the "escape"-key, or by the "break"-key.

3.3.4.1 THE CONTINUE COMMAND

This command is used to restart the execution of a program previously started by the @RECOVER command.

The format is:

@CONTINUE

The program is restarted in the <restart address> specified in the @DUMP command.

3.3.4.2 THE GOTO-USER COMMAND

This command is used to start execution in the users virtual memory, at an address specified.

The format is:

@GOTO-USER <address>

where the <address> is an octal value of the first instruction to be executed.

The command is usually used after the program has been interrupted, by means of the "escape"-key, and when the user wants to continue execution at the address where the interrupt occurred.

Remember that all opened files are closed when a program is terminated or aborted, unless the command @SET-PERMANENT-OPEN has been used.

3.3.5 *Loading a Binary Program*

Binary program files on disk or paper tape, previously created by the)BPUN command in the MAC assembler, may be loaded into the user's virtual memory by the following commands.

3.3.5.1 THE LOAD-BINARY PROGRAM

This command reads the specified file into user's virtual memory and starts the execution of the program.

The address used for loading and starting is found in the program file, written there by the)BPUN command in MAC.

The format is:

@LOAD-BINARY <filename>

While loading, a checksum is generated and compared with the checksum-word in the program file itself. Execution will not be started if the checksum differs.

3.3.5.2 THE PLACE-BINARY COMMAND

This command is the same as @LOAD-BINARY, except that the program is not started. This may be done by the @GOTO-USER command.

These two commands "simulate" the action of pressing MASTER-CLEAR and LOAD buttons on a stand-alone NORD-10.

Programs in Binary Relocatable Format (BRF) should not be loaded by the previous commands, but by means of one of the link-loaders:

BRL the binary relocatable loader, or

LDR the binary relocatable loader, including FORTRAN run-time and library system, or

OVLDR The FORTRAN overlay loader.

3.3.6 *Examination of User's Registers and Memory Contents*

Two commands have been implemented by which the user may inspect and change the contents of any memory location within the virtual memory space, and of the user-accessible registers.

3.3.6.1 THE STATUS COMMAND

This command prints on the terminal the contents of the registers.

The format is:

@STATUS

The register contents are given as octal numbers. The type-out is as follows:

P=xxxxxx	program counter
X=xxxxxx	post-index register
T=xxxxxx	temporary register
A=xxxxxx	accumulator
D=xxxxxx	double accumulator
L=xxxxxx	subroutine link address register
S=xxxxxx	status register
B=xxxxxx	pre-index (base) register

The register contents will always reflect the user's virtual memory, and will not be changed when executing system commands.

3.3.6.2 THE LOOK-AT COMMANDS

This command may be used to examine and modify memory locations and registers.

The format is:

@LOOK-AT <space-reference>

where <space-reference> may be

MEMORY — meaning user's virtual memory space. This is allowed for all users.

SEGMENT — A real-time program segment on mass storage may be reached. A segment number must be given as an additional parameter. This is allowed only for the users RT and SYSTEM. A modification causes a permanent change of the specified location on the segment.

RTCOMMON — Locations of the common area for RT programs may be reached. This is allowed only for the users RT and SYSTEM. A modification takes place in the memory and leads only to a temporary change of the specified location. The next time SINTRAN is loaded, the "old" values are retained.

IMAGE — Locations of the memory image of the resident part of SINTRAN III on mass storage can be reached. This is allowed only for the user SYSTEM. A modification causes a permanent change of the specified location.

RESIDENT — Locations of the resident parts of the SINTRAN III operating system can be reached. This is allowed only for the user SYSTEM. A modification takes place in the memory and leads only to a temporary change of the specified location. The next time the corresponding core-image is loaded to the memory, the "old" values are retained.

When the <space-reference> given has been checked for legality, and made available if mass storage segments are involved, the message READY is typed.

To examine a location, the octal address should be typed followed by a slash (/). The octal contents will then be printed. The contents may now be changed by typing an octal value, followed by a carriage return. If only a cr, without a new value, is given, the contents remain unchanged and the contents of the next location are printed.

If an asterisk (*) is typed, the current address will be printed.

The contents of registers can be addressed in the same way, using a single letter to specify the register. The letters are:

P, X, T, A, D, L, S, B.

When a character not mentioned above is typed, the command is terminated and control will return to normal control mode.

If locations on mass storage segments are changed, the pages will be written out so that "patches" will be made permanent. Locations changed in the user's virtual memory or the resident part of the operating system are changed temporarily. They may be altered when loading a user program or reloading the system.

This command is also mentioned in Sections 5.4 and 6.4.3.

3.3.7 *Obtaining System Information*

Certain commands may be used to obtain various information from the system.

3.3.7.1 THE DATCL COMMAND

This command prints the current setting of clock and date.

The format is:

@DATCL

The print-out has the form:

hour.minute.second date month-name year
for instance:
09.00.01 31 JANUARY 1975.

3.3.7.2 THE TIME-USED COMMAND

This command prints the CPU time and the terminal time used since the user ENTERED or a batch-job is started.

The format is:

@TIME-USED

The print-out has the format:

TIME-USED <n> MIN <n> SEC (CPU time)
OUT OF <n> MIN <n> SEC (terminal time)

3.3.7.3 THE WHO-IS-ON COMMAND

This command lists the terminal number and the name of the users entered.

The format is:

@WHO-IS-ON

3.3.7.4 THE WHERE-IS-FILE COMMAND

This command may be used to see if a peripheral device as line printer, tape reader and so on is occupied by another user or free to use. It may also be activated concerning a file in one of the directories entered.

The format is:

@WHERE-IS-FILE <file-name>

Where <file-name> may be any name defined for a peripheral device or for an existing file in an entered directory.

If not in use, the message "FREE TO USE" is printed, if already occupied the message

ALREADY RESERVED BY THE USER <user-name>
AT TERMINAL-NO <n>

is printed.

3.3.8 *The MODE Command*

This command has been implemented to allow a user to execute a set of commands and other input responses, previously stored on a disk file, punched on paper-tape, or comprised of a card-deck.

The format is:

@MODE <input file> <output file>

where

<input file>

> is the file name or device unit from which SINTRAN III will now take input lines, until the end of file or another @MODE command is detected,

<output file>

> is the file name or device unit on which commands are listed together with the command output, if any.

If the user program reads or writes data on device unit number 1 (file name: TERMINAL), such data will be taken from the <input file> and written on the <output file>.

If the end of file or a command @MODE 1,1 is reached on the <input file>, the control will be returned to the user's terminal.

The execution of the command file will continue running under the user currently logged on the terminal from where the initial @MODE command was issued.

The execution may be interrupted by typing the "escape"-key but may not be continued as all opened files will be closed. The last process initiated may be continued by typing the @CONTINUE command.

Should error condition occur within a @MODE job, the offending error message will be written on the <output file>, and the message ***BATCH JOB ABORTED*** will be typed on the terminal and the execution terminated.

Within a @MODE file all commands are legal but certain restrictions exist:

- the first character of a command line must be the "@", which corresponds to the herald character typed by the system in front of commands in direct mode (*).

- all command parameters must be given on the same line as the command itself because the system cannot ask for missing parameters.

- although they are legal, some commands are not very well suited within a @MODE file. Among these are, for example, the @LOOK–AT command.

Note that if a @MODE command has been issued where the <input file> is the terminal, the typed characters are not echoed on the terminal as they are routed to the <output file>.

User's input data may be interspersed with command lines in the same way as if they were typed from the terminal.

Such data files may be terminated by the end-medium character (octal 27) which is treated as a normal character when reading the @MODE file.

(*) This applies only to SINTRAN commands. Commands to other subsystems such as the editor or loader must not be prefixed by that subsystem's leading character(s).

3.3.9 *The CC Command*

This command does absolutely nothing and is, therefore, very useful for making comment lines in big MODE or BATCH jobs.

3.3.10 *The HELP Command*

This command will list the names of all SINTRAN III commands on the terminal.

3.3.11 *The HOLD Command*

The format is:

@HOLD <time> <time unit>

<time unit> can have the values:

1. basic time units (normally 20 milliseconds)
2. seconds
3. minutes
4. hours

This will keep the terminal waiting for the number of time units specified by <time>.

3.3.12 *The TERMINAL-MODE Command*

The user can, to some degree, decide how the system should treat the terminal. When the command:

@TERMINAL-MODE

is given, it will ask three questions.

The first question is:

CAPITAL LETTERS?

If the user answers YES, all small letters will be converted to capital letters. If the answer is NO, there will be no conversion. Other answers will give no change in the mode.

The second question is:

DELAY AFTER CR?

If the answer is positive some dummy characters will be printed after each carriage return. This is necessary for some terminals (for example SILENT 700) if they run at 30 characters per second.

The third question is:

STOP ON FULL PAGE?

This feature is useful for fast displays. If 20 lines have been listed on the display without any intervening input, the output will stop so that the user may take a look at the output. The listing will continue as soon as any break character is typed. The control/shift P can be recommended as a continue character because it will be completely ignored.

3.3.13 *The IOSET Command*

This command can be used to set an I/O-device in a given state.

The format is:

IOSET <logical unit> <read/write> <program> <control>

The <logical unit> identifies a device, which must be reserved (opened) beforehand. <read/write> means input part is 0 and output part is 1. <program> must be 0. <control> has a different meaning for different devices but generally -1 means "reset device". See also Section 3.6.2.

3.4 *THE MOST COMMONLY USED FILE MANAGEMENT SYSTEM COMMANDS*

As previously mentioned, the File Management System is described in a separate manual: NORD FILE SYSTEM, ND-60.052.

In this chapter, the most commonly used file handling commands will be described.

3.4.1 *Creating and Deleting Files*

The most important commands for creating and deleting files are:

@CREATE-FILE <file name><no of pages>

> will create a contiguous file with the given number of 1K word pages (1024 decimal words, or 2048 bytes), or an indexed file with no pages allocated if <no of pages> given is zero.
>
> The default file type is :DATA, and if a version number is included in the file name, a number of copies will be created.

To allocate a file on a specific area on a disk or drum, the command

@ALLOCATE-FILE <file name><page address><no of pages>

> will create a continuous file, starting from the <page number> given. The page size is always 1K words (2K bytes) and the page address is given as a page number from the start of the device, which is always page 0.
>
> If more than one version is created, they are allocated after each other with version 1 from the specified page address.

To delete a file and release the pages used, the following command is given

@DELETE-FILE <file name>

> If a version number is specified, only that version is deleted. Otherwise, all versions of a file are deleted. The file type must always be specified.

Contiguous files are not allowed to grow dynamically larger than the allocated number of pages, except by the command:

@EXPAND-FILE <file name> <number of pages>

> The file must be contiguous.

Indexed files may expand dynamically up to the reserved free space for the owner as the user writes onto the file.

3.4.2 *File Protection and Access Modes*

A file may be protected from unauthorized use by the command

@SET-FILE-ACCESS <file name> <public access> <friend access> <owner access>

Separate access modes may be specified for the owner of a file, for other users declared as friends, or all other users of the installation.

The access modes may be a combination of:

R	read access,
W	write access,
A	append access,
C	common access permitted, and
D	directory access, or
N	no access permitted.

Directory access means that the file may be deleted, or that access mode may be changed.

If an access string is empty, i.e., only the comma separator or a carriage return is given, the access mode for that user group will not be changed.

Up to 8 friends may be declared and given access mode by the commands:

@CREATE-FRIEND <user name>

@SET-FRIEND-ACCESS <user name><access mode>

where <user name> must be one known to the system, and <access mode> is described above.

Friends may be deleted by the command:

@DELETE-FRIEND <user name>

3.4.3 *File Statistics and User Information*

There are several commands used to give the user certain information about created files, reserved space, declared friends, etc.

In all, the following commands the parameter <output unit> denotes the device or file where the information will be written. This may be any file or peripheral unit, default unit being the user's terminal.

The <file name> or < user name> in the following commands may be abbreviated, and only the selected parts that match the name given will be included in the response. If the file name is empty or only carriage return is given, all files for the user will be selected. For <file names> also those files of a specific file type may be selected, giving only the name of the type preceded by a colon.

If the file-directory-name and user name is not included in the <file name>, the default file-directory will be searched.

The commands are:

@LIST-FILES <file name><output unit>

> gives a list of all files that match the <file name>, with the file-entry-number full file name including directory and user name, file type and version number.

@LIST-OPENED-FILES <output unit>

> lists the file names and numbers of all currently opened files for the logged in user.

@FILE-STATISTICS <file name> <output unit>

> gives a list of all files that match the <file name>, and in addition to the name also the type of file (indexed or continuous), access modes for public users friends and the owner date of creation of the file, number of times and last date opened for read and for write, and size of the file both in pages and bytes.

@LIST-FRIENDS <friend name> <output unit>

> lists those users declared as friend, and the access mode, defined.

@LIST-USERS <user name> <output unit>

> lists those users created on the main directory or those on a specific directory that precedes the user name.

@USER-STATISTICS <user name> <output unit>

lists some user information such as date created, number of pages in use and number of pages reserved.

3.4.4 *Copying Data to and from Files and Devices*

Two commands are implemented to copy data between peripheral units and files:

@COPY <destination> <source>

reads bytes from the <source>unit and writes on the <destination>unit. The destination unit may be a new file to be created by enclosing the name in double quotation marks.

@COPY-FILE <destination> <source>

reads pages of 1K words from <source file> and writes on the <destination file>.

For both commands, data is completely code-independent and the copy operation continues until all pages have been copied from a file, or until time-out is received from a peripheral unit.

Default file type for files is :SYMB.

Both commands may also be used to or from peripherals. @COPY–FILE will also correctly copy files with holes.

3.4.5 *Opening and Closing Files*

Before a file or peripheral unit can be accessed for read or write, it must have been opened for use with the intended access mode specified.

A file is opened using its name, but is accessed with a logical unit number.

This number is returned by the opening function, provided that the file is available and the user is allowed to access the file. This number must later be used in all references to the file within the program.

Files may be opened from the outside of a program using Commands, or from the inside of a program using Monitor Call Functions.

3.4.5.1 THE OPEN-FILE COMMAND

This command is used to make a file available for access, and returns the logical unit number to be used for accessing the file.

The format is:

@OPEN-FILE <filename> [:<type>[; version>]] ,<access-type>

where

<file name>

could be the name of a peripheral unit, eg., LINE-PRINTER or a user filename, which may be prefixed with file-directory-name and user-name if necessary.

:<type>

if a file with a specific type is to be opened. The default value is :SYMB.

;<version>

if a specific version number is to be opened. The default value is 1.

<access-type>

specifies the intended access-mode, a combination of

R - read
W - write
X - random access
A - append
C - common access (only contiguous file)

Only the following combinations are legal:

R - sequential read
W - sequential write
RW - sequential read and write
RX - random read
WX - random read and write
WA - sequential write append
RC - random read with read and write access from other users allowed (only contiguous files)
WC - random read and write with read and write access from other users allowed (only contiguous files)

If the user is allowed to access the file, the following message is printed:

FILE-NUMBER IS = n

where n is the logical unit number to be used when accessing the file later.

New file names and/or number of versions may be created together with the opening of the file by enclosing the name or number in quotation marks.

If, for any reason, it is not possible to open the file, an explanatory message is given.

A list of logical unit numbers is given in Appendix C.

3.4.5.2 THE CONNECT-FILE COMMAND

This command may be used to open a mass storage file with a predefined file number. Otherwise it acts like the OPEN command.

The format is:

@CONNECT-FILE <filename> <file number> <access type>

where

<filename> and
<access type>

are like the OPEN command,

<file number>

is the logical unit number the file should be associated with. The number must be within 101_8 to 177_8, and not previously opened by the user.

3.4.5.3 THE CLOSE-FILE COMMAND

The following command may be used for closing previously opened files:

@CLOSE-FILE <file number>

The file with the specified number will be closed. If <file number> is −2, all files for the entered user are closed. If <file number> is −1, all files that are not permanently opened will be closed.

3.4.6 *Reserving and Releasing Files and Peripheral Devices*

To reserve a file or peripheral device for exclusive use from the current terminal, the following command is available:

@RESERVE-FILE <file name>

where

<file name>

is the name of a peripheral device, for instance, LINE-PRINTER, or of a directory file.

A previously reserved file or peripheral device might be released from the terminal by the command:

@RELEASE-FILE <file name>

3.5 *FILE HANDLING FROM USER PROGRAMS*

This section will describe different methods for allocating files, reading and writing both sequentially and randomly, and closing files from a user program.

"Files" in this context means any peripheral device or file kept in the file directories either on disk, drum or magnetic tape. It does not include the use of magnetic tapes or cassette tapes treated as freestanding devices separated from the file management system, such as is explained in Section 9.2. Sequentially accessible files may be treated device-independently. That means that the user, during the execution of the program, may access peripheral units and disk or magnetic tape files similarly.

Randomly accessible files must be kept on disks or drums, due to the physical constraints of other devices.

3.5.1 *Opening Files from a Program*

3.5.1.1 SUBROUTINES FOR OPENING FILES

For programs written in FORTRAN, a subroutine may be used to open a file and connect it with the file code used in the READ and WRITE statements.

The format is:

CALL OPEN (<file name>, <file code>, <access code>)

where

<file name>

may be a character string or an array name containing the name of the file. Default file type is :DATA. The last character should be a blank.

<file code>

is a decimal constant or an integer variable containing the number used in READ/WRITE.
INCH, OUTCH, SETBS, RFILE, WFILE and CLOSE statements.

<access code>

a decimal constant or an integer variable containing an access code with the following value:

0 - write sequential
1 - read sequential
2 - random read or write
3 - random read only
4 - sequential read or write
5 - sequential write append
6 - random read or write common (only contiguous files)
7 - random read common (only contiguous files)

If the user is allowed to access the file, a table is maintained linking the file code with the logical unit number provided by the file management system.

If the file cannot be opened, an error message is given.

The user may also use the subroutine as an integer function. Thus the value will be set to zero if the file is opened, or to the octal error number, as described in Appendix D.

Example:

```
INTEGER  OPEN
ISTAT = OPEN (<filename>, <file code>, <access code>)
IF (ISTAT.NE.Ø) GO TO
```

The <filename> may be a character string of Hollerith type (e.g., 20HMYFIL), enclosed in single apostrophes (e.g., 'MYFIL') or an array containing the name of the file. The character string should be ended with a blank.

Example 1:

```
CALL OPEN ('MYFIL:DATA ', 10,0)
WRITE (10) BINARRAY
```

Example 2:

```
      INTEGER FILNAME (20), OPEN
      WRITE (1,10)
10    FORMAT   (/,'$GIVE  NAME OF INPUT-FILE:')
      READ (2,20)FILENAME
20    FORMAT (20A2)
      ISTAT = OPEN (FILENAME,10,1)
      :
      READ (10,30)DATA
      :
```

3.5.1.2 MONITOR CALLS FOR OPENING FILES

For programs written in MAC assembly or NORD PL languages, a monitor call function may be used to open a file. A description of the monitor call functions is given in Section 3.6.

MON 50 (OPEN)

X register:
pointer to file name string. When X = 0 the file name is read from the terminal!

A register:
pointer to default file type string

T register:
Access code:

0 - sequential write
1 - sequential read
2 - random read or write
3 - random read only
4 - sequential read or write
5 - sequential write append
6 - random read or write common (only contiguous files)
7 - random read common (only contiguous files)

Return:
A- reg: Error code

Skip return:
A-reg: File number

Example:

```
        LDX     (FNAME          % POINTER TO FILE NAME
        LDA     (FTYPE          % POINTER TO FILE TYPE
        SAT     0               % ACCESS CODE WRITE
        MON     50              % OPEN
        JMP     ERROR           % RETURN:ERROR
        STA     FNUM            % SKIPRETURN:FILE NUMBER
        :
        :

FNAME           'REPORT-FILE'   % FILE NAME
FTYPE           'DATA'          % FILE TYPE
FNUM,           0               % FILE NUMBER
)FILL
```

3.5.2 *Accessing Files*

Files may be accessed with sequential or random monitor calls using the file number returned when opening the file.

FORTRAN programs will normally access files through the FIO - Formatted Input and Output system with the READ and WRITE statements, but may use a set of integer functions executing similar monitor calls.

In this section both the MAC assembly and the FORTRAN coding will be given.

3.5.2.1 SEQUENTIAL FILE ACCESS

Reading bytes from a file is done by the monitor call.

MAC coding: INBT (MON 1)

```
LDT     FNUM    % T-REG = FILE NUMBER
MON     1       % INBT
JMP     ERR     % ERROR RETURN:A = ERROR
                  NUMBER
STA     CHAR    % SKIPRETURN:A = BYTE IN
                  BITS 0 - 7
```

FORTRAN function:

ICH = INCH (<file code>)

where

ICH receives an 8-bit character (16-bit if data link) from the device buffer without any modification, except for card reader which is converted to ASCII. If there are no bytes in the buffer, the program will go into waiting state. Negative result means error.

Writing bytes to a file is done by the monitor call.

MAC coding: OUTBT (MON 2)

```
LDT     FNUM    % T-REG = FILE NUMBER
LDA     CHAR    % A-REG = BYTE IN BITS 0 - 7
MON     2       % OUTBT
JMP     ERROR   % ERROR-RETURN! A = ERROR
                  CODE
JMP     NEXT    % SKIP-RETURN:NORMAL
```

FORTRAN function:

ISTAT = OUTCH (<file code>, <char.value>)

The 8 right bits of the <char.value> (16 bits if data link) is outputed to the buffer. If there is no room in the buffer, the program will be in waiting state until more room is available. ISTAT receives a zero-value if OK, otherwise a negative value denotes an error condition.

3.5.2.2 RANDOM FILE ACCESS

Some monitor calls may be used to access files in a random manner, reading or writing blocks of equal size, by specifying the block number. Block size may be set before the first attempt to read or write. If not, the default block size of 256 decimal words (400 octal) will apply.

All FORTRAN functions described here may also be called as sub-routines. If an error occurs a message is displayed and the program terminates.

The monitor calls and FORTRAN functions are:

Set Block Size

MAC coding: SETBS (MON 76)

```
        LDA     FNUM        % T-REG = FILE NUMBER
        LDA     BSIZE       % A-REG = BLOCK SIZE (WORD)
        MON     76          % SETBS
        JMP     ERROR       % ERROR RETURN:A = ERROR
                              NUMBER
        JMP     OK          % SKIP RETURN:OK
FNUM,       0
BSIZE,:     20              % BLOCK SIZE 16 DECIMAL
                              WORDS
```

FORTRAN function:*

```
INTEGER SETBS
ISTAT = SETBS (<file code>,<block size>)
IF (ISTAT.NE.Ø)GO TO
```

or

CALL SETBS (<file code>, <block size>)

where

<block size>

is a decimal constant or an integer variable containing the number of 16 bit words of each block.

ISTAT returns with a zero if OK, otherwise a negative value is returned.

Read Block Random

Reads one block of specified size (default 256 decimal words) from the file, which must have been opened with random read or read/write access mode:

MAC coding: RPAGE (MON 7)

```
        LDT     FNUM        % T-REG = FILE NUMBER
        LDX     (ADR        % X-REG = CORE ADDRESS
        LDA     BLKNO       % A-REG = BLOCK NUMBER
        MON     7           % RPAGE
        MON     65          % ERROR RETURN:A = ERROR
                              CODE
        JMP     OK          % SKIP– RETURN:A = 0

)FILL
FNUM,       0
ADR,        0
*+n/0                       % n = BLOCK SIZE (OCTAL)
BLKNO,      0               % BLOCK NUMBER
```

FORTRAN function:

```
INTEGER RFILE
ISTAT   = RFILE (<file code>, <return flag>, <memory address>,
        <block no>, <no of words>)
IF (ISTAT.NE.Ø) GO TO
```

or

```
CALL RFILE (<file code>, <return flag>, <memory address>, <block
        no>, <no of words>)
```

where

<return flag>

must be an integer variable if = 0: The program will wait until the transfer is finished. If ≠ 0: The process will continue as soon as the transfer is started.
The monitor call WAITF (see following) can be used to check if the transfer is finished.

<memory address>

array where the record will be read into.

<block no>

an integer variable containing the block number where reading begins, counting from 0,

<no of words>

is an integer variable containing the number of 16 bit words to be transferred into the program. This may be less or larger than the specified block size.

ISTAT will be set to zero if the request is OK, or to negative value if error occurs.

The user should be aware that files are allocated in "pages" of 1K words (1024 decimal words) and a block size which is >1024 will force the operating system to read several pages to fullfill the request.

Write Block Random

Writes one block of specified size (default block size is 256 decimal words) to the file, which must have been opened with random write access mode.

MAC coding:

```
        LDT     FNUM        % T-REG = FILE NUMBER
        LDX     (ADR        % X-REG = MEMORY ADDRESS
        LDA     BLKNO       % A-REG = BLOCK NO.
        MON     10          % WPAGE
        MON     65          % ERROR-RETURN:A = ERROR
                              CODE
        JMP     OK          % SKIP-RETURN:A = 0

)FILL
FNUM,       0
ADR,        0
*+n/0                       % n = BLOCK SIZE (OCTAL)
BLKNO,      0               % BLOCK NUMBER
```

FORTRAN function:

INTEGER WFILE
ISTAT = WFILE (<file code>, <return flag>, <memory address>, <block no>, <no of words>)

or

CALL WFILE (<file code>, <return flag>, <memory address>, <block no>, < no of words>)

where

<return flag>

must be an integer variable. If = 0: The program will wait until the transfer is finished. If ≠0: the program will continue as soon as the transfer is started. The monitor call WAITF (see following) can be used to check if the transfer is finished.

<memory address>

array where the record will be written from

<block no>

an integer variable containing the block number where writing begins, counting from 0.

<no of words>

an integer variable containing the number of 16 bit words to be transferred. This may be less than or greater than the specified block size.

ISTAT will be set to zero if the request is OK, or to a negative value if error occurs.

CALL WAITF (<file number>, <return flag>)

This call is used to check whether a transfer is finished or not. If the transfer is completed, there will be immediate return. If the <return flag> is equal to zero and the transfer is not completed, the calling RT program will be set in a waiting state until the transfer is finished. If the <return flag> is set (non-zero) and the transfer is not finished, there will be an immediate error return (non-zero integer function value).

3.5.3 *Closing of Files*

Opened files will automatically be closed when the program terminates, either normally or after an error message.

The user may also close files using the following monitor calls or FORTRAN functions.

MAC coding: CLOSE (MON 43)

```
        LDT     FNUM        % T-REG = FILE NUMBER
        MON     43          % CLOSE
        MON     65          % ERROR RETURN:A = ERROR
                              CODE
        JMP     OK          % SKIP-RETURN:A = 0
FNUM,       0               % FILE NUMBER
```

FORTRAN function:

```
INTEGER CLOSE
ISTAT = CLOSE (<file code>)
```

or

```
CALL CLOSE (<file code>)
```

where

<file code>

is an integer variable containing the file code or –2 to close all opened files. The value –1 will cause a closing of all files that are not permanently opened.

ISTAT will be set to zero if OK, or to a negative value if error occurs.

3.6 *MONITOR CALL FUNCTIONS*

A major part of the service functions provided by the SINTRAN III system are implemented through a special hardware instruction, the MON instruction (octal value 153xxx), and may be activated by the user programs. And, as some hardware instructions are not allowed to execute within a user program, e,g,. the IOX instruction for input and output directly to devices, such functions must be activated through Monitor Call functions.

When executing a MON instruction within a user program, an interrupt is generated, transferring control to a system routine on level 14. This routine activates the required function on a lower interrupt level (5 or 3) on behalf of the calling program.

In this way, a significant amount of subroutines are available to all users, without enlarging the program with the subroutine code.

Programs written in MAC assembly or NORD PL languages can use the MON instructions directly while programs written in other languages may use a set of small subroutines calling the appropriate MON instructions. These subroutines are described Sections 3.5 and 3.6.2.

Monitor calls are activated by an octal number, ranging from -200_8 to 177_8 giving 400_8 (256 decimal) different functions. At the moment, approximately 177_8 functions are implemented, of which an installation might define 7 of their own purpose, to be generated together with their version of the SINTRAN III system.

Passing of Parameters is Done in Two Different Ways:

- through various hardware registers, normally the T, A and X registers, of which the user must have set appropriate values, pointers, etc. before the MON instructions are executed, and

- in the standard call format, where the A register points to a parameter list containing the address to the value for each parameter.

Most monitor call functions use the two following instructions as return points, the first if errors occur, the second when the function is performed normally.

In the case of an error return, the A register will contain an error number. This error number may be used to obtain an explanatory message. A list of all error numbers and messages is given in Appendix D.

In the case of a normal return, usually noted as a skip return, the A register will contain a zero value, or a value defined by each monitor call.

Examples of the calling sequences in MAC assembly:

a) using registers for parameters:

The monitor call INBT inputs a byte from a device:

```
        SAT     1           % DEVICE-NO IN T-REG
                            % 1 IS FOR TERMINAL
        MON     1           % INBT
        JMP     ERR         % ERROR-RETURN:
                            % A-REG = ERROR-NUMBER
        STA     CHR         % SKIP-RETURN:
                            % A-REG = BYTE IN BITS 0 - 7
```

b) using standard call format for parameters:

The monitor call CLOCK to obtain data and time:

```
        LDA     PARAM       % A-REG POINTS TO THE
                            % PARAMETER LIST
        MON     113         % CLOCK

        :

PARAM,  ARRAY               % POINTS TO A SEVEN-WORD
                            % ARRAY
ARRAY,  0                   % BASIC UNITS
        0                   % SECONDS
        0                   % MINUTES
        0                   % HOURS
        0                   % DAY
        0                   % MONTH
        0                   % YEAR
```

Section 3.6.1 describes the monitor calls where the T, A, D and X registers must be set before the MON instruction is executed. The ERROR RETURN and the SKIP RETURN denote the contents of the A register after execution.

Section 3.6.2 lists the monitor calls using the standard call format. In this case, the parameters are described by names in the correct order.

Both sections describe only the monitor calls available from timesharing and batch programs. With a few exceptions, denoted in the text, these monitor calls may also be used from RT programs. (See also Section 3.6.3.)

However, many monitor calls may only be executed from RT programs. Such monitor calls are described in Section 7.7.

3.6.1 *Monitor Calls Available from MAC and NORD-PL Programs*

MON 0 <no parameters>	LEAVE or RTEXT Terminates the executing program and returns control to the operating system. All reserved units are released and opened files are closed. If called from an RT program, no file will be closed.
MON 1 T = device number Error-return:A = error no. Skip-return: A = byte, in right half	INBT Reads one byte from the specified device. The byte pointer is increased by one.
MON 2 T = device number A = byte, in right half Error-return: A = error no. Skip-return: A = 0	OUTBT Writes one byte to the specified device. The byte pointer is increased by one.
MON 3 A = 0 echo all characters A = 1 echo all but control characters A = 2 special MAC echo strategy A < 0 no echo	ECHOM Defines the echo mode for the user terminal. Default echo mode is 1. If called from an RT program, the T register must contain the logical number of the terminal.
MON 4 A = 0 always break A = 1 break only on control characters A = 2 special MAC break strategy A < 0 no break	BRKM Defines break mode for characters used to resume execution of user programs waiting for input from the terminal. Default break mode is 1. If called from an RT program, the T register must contain the logical number of the terminal.
MON 5 T = block no., from 0 and up X = core address to transfer to Error-return:A = error no. Skip-return:A = 0	RDISK Reads random 256 decimal word blocks from the scratch file, logical unit 100 octal. Only to be issued by background (timesharing) programs.

MON 6
T = block no., from 0 and up
X = core address to transfer from
Error-return:A = error no.
Skip-return:A = 0

WDISK
Writes random 256 decimal word blocks to the scratch file, logical unit 100 octal. Only to be issued by background programs.

MON 7
T = file number
A = block number
X = core address to transfer to
Error-return:A = error no.
Skip-return:A = 0

RPAGE
Reads random 256 decimal word blocks from a file. Logical unit 101 - 177 octal.

MON 10
T = file number
A = block number
X = core address to transfer from
Error-return:A = error no.
Skip-return:A = 0

WPAGE
Writes random 256 decimal word blocks to a file. Logical unit 101 - 177 octal.

MON 11
AD = time in basic time units (normally 20 milliseconds)

TIME
Returns current internal time.

MON 13
T = device number
Error-return:A = error no.
Skip-return:A = 0

CIBUF
Clear device input buffer.

MON 14
T = device number
Error-return:A = error no.
Skip-return:A = 0

COBUF
Clear device output buffer.

MON 43
T = file number
Error-return:A = error no.
Skip-return:A = 0

CLOSE
Closes a previously opened file. If T = -1 all opened files for this user will be closed. If T = 1 all opened files that are not set permanently open are closed. (Scratch files are not closed.)

MON 45
MON 46
MON 47

DBRK
GBRK
SBRK
Special monitor call used by MAC assembler for break points. Not to be used by others.

MON 50
X = points to file name string
A = points to file type string
T = 0 sequential read
1 sequential write
2 random read or write
3 random read
4 sequential read or write
5 sequential write append
6 random read or write common
7 random read common
Error-return:A = error no.
Skip-return:A = 0

OPEN (SINTRAN III version)
Opens a file named in a character string pointed to by the X register, with a default file type pointed to by the A register. The type of file access is specified in the T register. If a file is found, the file number to be used in subsequent operation is returned in the A register. Character string for file name and type must be terminated by a single apostrophe ('). If X = 0 the file name string is read from the terminal.

MON 51

DMAC BREAKP
Special monitor call used by the DMAC assembler for setting breakpoints. Not to be used by others.

MON 62
T = file number
Error-return:A = error no.
Skip-return:AD = no. of bytes

RMAX
Reads the maximum number of bytes in a file available for sequential read, starting with one. This is the value of the maximum byte pointer + 1.

MON 64
A = error number

ERMSG
Types an explanatory error message for a corresponding error number contained in the A register. The program continues. The message is typed on the user terminal or batch on mode output file, or if issued from an RT program, on terminal 1.

MON 65
A = error number

QERMS
Same as previous command but the program terminates. May be used directly as the error return instruction in other monitor calls.

Example:

```
SAT  2       % T=TAPE-READER
MON  1       % INBT
MON  65      % ERROR-MESSAGE
STA  CHR     % SAVE BYTE
```

MON 66
T = device number
Error-return:A = error no.
Skip-return:A = no. of bytes

ISIZE
Returns the number of bytes currently in the input buffer of a device.

MON 67
T = device number
Error-return:A = error no.
Skip-return:A = no. of bytes

OSIZE
Returns the free number of bytes currently in the output buffer of a device.

MON 70

COMND, Command Monitor Call
On entry, the A register points to a character string, which will be interpreted as a SINTRAN III command. This monitor call can be used in background programs only.

Example:

```
LDA (STR
MON 70
:
STR,'DELETE-FILE XXX:SYMB'
```

MON 73
T = file number
AD = no. of bytes
Error-return:A = error no.
Skip-return:A = 0

SMAX
Sets the maximum byte pointer to point to the last byte available for sequential read, starting with 0.

MON 74
T = file number
AD = byte pointer
Error-return:A = error no.
Skip-return:A = 0

SETBT
Sets the byte pointer of a file to the byte which will be accessed by INCH/OUTCH the next time, starting with 0.

MON 75
T = file number
Error-return:A = error no.
Skip-return:AD = byte pointer

REABT
Reads the byte pointer of a file. The byte pointer points to the byte to be accessed by the next INCH/OUTCH call , starting with 0.

MON 76
T = file no.
A = block size (in words)
Error-return:A = error no.
Skip-return:A = 0

SETBS
Sets block size of a file opened for random read or write access.

MON 77
T = file number
AD = block pointer
Error-return:A = error no.
Skip-return:A = 0

SETBL
Sets the block pointer of a file.

3.6.2 *Monitor Calls also Available from FORTRAN*

This section describes the monitor calls implemented with the standard call format of SINTRAN III. These routines may, in FORTRAN, be activated using the CALL statement or as INTEGER FUNCTIONS for those routines that denote occurrence of error conditions with a negative value returned through the A register.

MAC assembly programs may use the following code sequence:

```
        LDA     (PARAM          % A-REG POINTS TO A PARAMETER
                                % LIST
        MON     xxx             % xxx IS THE MON-CALL NUMBER
        JAN     ERROR           % A-REG NEGATIVE DENOTES
                                % ERROR
        JMP     OK              % SKIP RETURN (NORMAL)
)FILL
PARAM,      PAR1                % THE PARAMETER LIST POINTS
            PAR2                % TO THE VALUE OF EACH
            PAR 3               % PARAMETER
```

Error messages may be typed at the user terminal, in the case of an RT program error messages are typed at the terminal with logical unit number 1, and the program is terminated.

MON 76 CALL SETBS (<file code>, <block size>)

The block size of a file is set to be accessed randomly. (See also Section 3.5.2.2.)

MON 104 CALL HOLD (<time>, <time unit>)

The calling program will be set in waiting state for the number of <time units> specified by <time>, <Time unit> may be specified as:

1 basic time units (normally 20 milliseconds)
2 seconds
3 minutes
4 hours

MON 113 CALL CLOCK (<array>)

The current clock/calendar is returned to the seven word integer <array> of the calling program. The <array> will contain: basic units, seconds, minutes, hour, day, month and year.

MON 114 TD = TUSED(0)

The parameter is dummy. The double precision integer function value gives the CPU time used up until now, counted from log-in time, in basic time units (usually 20 milliseconds).

This monitor call can be used from background (timesharing) programs only.

MON 117 ISTAT = RFILE (<file code>, <return flag>, <memory address>, <block no>, <no of words>)

This monitor call reads from a random block in a file. (See Section 3.5.2.2.)

MON 120 ISTAT = WFILE (<file code>, <return flag>, <memory address>, <block no>, <no of words>)

The monitor call writes onto a random block in a file. (See Section 3.5.2.2.)

MON 141 <value> = IOSET (<logical unit>, <read/write>, <program>, <control>)

Control information will be set for a logical unit. If <read/write> equals zero, the input part is reserved for a two-way unit, otherwise, if it equals one, it means the write part. If <control> equals -1, the unit will be reset. Otherwise, <control> has a special meaning for each device type. <program> specifies a program which the unit is supposed to be reserved by. If not, IOSET will return a negative function value. This will also occur if an illegal logical unit is specified. If everything is OK, a value greater than or equal to zero will be returned.

Example:

I = IOSET (2, 0, PROG, -1)

This means: Clear and reset the tape reader which is reserved for program PROG.

From background, only devices reserved by the background program itself may be operated on. That means that the parameter <program> must be equal to zero.

For card reader, the following values of the <control> parameter means:

-1: Clear buffer and set ASCII mode - all characters are converted to ASCII code, and trailing blanks are ignored.

0: Set ASCII mode.

1: Set binary mode. Subsequent INBT monitor calls will return a 12 bit column image.

For some devices, such as Process I/O and NORDCOM, the meaning of <control> will be described elsewhere.

MON 145 <error code> = ACM (<logical unit>, <function>, <memory address>, <DMA address>, <word count>)

This is the monitor call to transfer a block of words to/from an external memory.

<logical unit> identifies the external memory. This unit must be reserved beforehand. The available logical numbers are 734_8, 735_8, 736_8, 737_8 and 740_8.

<function> is the function code:

0 - Read
1 - Write
2 - Lock/Write/Unlock
3 - Clear

Example:

```
INTEGER FUNCTION ACM
DIMENSION IARR (100)
:
CALL RESERV (734B, 0, 0)
IX = ACM (734B, 1, IARR, IDMA, 100)
```

This monitor call is an option and can be included at system generation time.

MON 161 I = INSTR (<logical no>, <memory address>, <max no >, <terminator>)

This monitor call reads a string of characters from a peripheral device, identified by <logical no>. The monitor call will read characters from the device and pack them into the user's area (starting at <memory addr>), and go on until at least one of the following conditions is met.

1. The maximum number is reached

2. The terminator is found

3. The device buffer is empty

The function value has the following contents:

Bits 14 and 15:

0: Maximum number of characters reached without finding the terminator.

1: Terminator found

2: Buffer emptied without finding terminator

3: Device error

In case of error, the rest of the word will contain an error number, otherwise it will contain the number of characters read. If -1 is returned, it means bad parameter. If the device buffer is empty when INSTR is called, the calling program will be set into a waiting state.

MON 162 I = OUTST (<logical no>, <memory addr>, <number>)

The monitor call will move the characters from the user's area to the device buffer. If there is not enough room, the program will be put in a waiting state.

The function value has the following contents:

Bits 14 and 15:

0: OK
1: Not used
2: Buffer too small; nothing is done
3: Device error; the rest of the word contains error number

Function value = -1 means bad parameter.

The monitor calls INSTR and OUTST are options. They may be included at system generation time.

3.6.3 *Monitor Calls Not Available in RT Programs*

The following monitor calls cannot be used in an RT program.

Name	Mon.Call No.(octal)	Description
RDISK	5	read 256 words from scratch file (file number 100_8)
WDISK	6	write 256 words on scratch file (file number 100_8)
DBRK	45	used by MAC for debugging RT programs
GBRK	46	used by MAC for debugging RT programs
SBRK	47	used by MAC for debugging RT programs
TUSED	114	get CPU-time
RSIO	143	monitor call for deciding if a program is running in batch or in time-sharing

3.7 *BATCH PROCESSING*

3.7.1 *Introduction*

In addition to the interactive timesharing communication with the SINTRAN III operating system, the user may also execute jobs in batch mode.

Batch jobs are executed independent of any terminals, under control of a *Batch Processor*. A batch process must be initiated by the user SYSTEM before any users may submit jobs to be executed under its control. An installation may have several batch processors running simultaneously, with different execution priority, if desired.

The users may build batch-input-files in a similar way to the @MODE command. A batch-input-file contains all commands and input lines necessary to carry out the requested activities, and may be stored on a disk file, punched on paper tape, or comprised of a card deck.

The user submits this batch-input-file for execution: by *appending* it to a *batch queue* and by specifying the file name in the @APPEND-BATCH command. In addition, the user gives the name of an output file or a device unit where the command response and results should be written. In the case of a file name, this file may later be emptied on a line printer.

In other installations, it may be suitable to give job decks to an operator, who will submit them for execution from the card reader.

3.7.2 *Definitions*

A *batch job* is an entity consisting of commands to the SINTRAN III system and possibly source input to subsystems or user programs called.

The first command in a batch job is always the @ENTER command, identifying the user. A batch job is terminated by two consequtive ESC characters (octal 33). These two characters correspond to the @LOGOUT command in direct mode.

The card code for ESC is multipunched 7-8-9.

A batch-input-file may contain any number of jobs and a job may use any number of batch-input-file/batch-output-file parts. Thus, the boundaries between jobs and batch-input-files are completely independent of each other.

A *batch queue* is a collection of batch-input-file/batch-output-file pairs, held internally in SINTRAN III, to be executed by a batch process. Each batch process has its own batch queue. New input/output pairs are added to the queue by the @APPEND-BATCH command.

A *batch process* is an RT program very much like the one that handles interactive communication with the SINTRAN III system from terminals. The task of a batch process is to execute jobs on a batch-input-file, one at a time.

In principle, there may be an unlimited number of batch processes running parallel to each other and other activities in the system. The maximum number of batch processes is determined at system generation time.

The extra overhead introduced by a batch process is approximately the same as by adding an extra terminal. A batch process is initiated by a @BATCH command, and terminated by an @ABORT-BATCH command. Both commands are issued only by the user SYSTEM. A batch process will be in one of the following states:

PASSIVE - this means that the batch process has not been started.

IDLE - this means that the batch process has entered waiting state because the batch queue is empty.

ACTIVE - this means that the batch process is working on a job.

When appending a batch-input-file to a batch queue in idle state, it will be *activated*. When end-of-file is reached on a batch-input-file, the batch process fetches the next batch-input/output pair of the queue, and continues taking input from the new batch-input-file and giving output to the new batch-output-file. When the batch queue for this process is empty, the process enters idle state again.

3.7.3 *Batch Commands for the User SYSTEM*

The following commands are only to be issued by the user SYSTEM:

3.7.3.1 THE BATCH COMMAND

This command is used to activate an unused (passive) batch process.

The format is:

@BATCH

The response is:

BATCH NUMBER = n

where <n> is a decimal number used in commands to identify a particular batch process.

If there are no unused batch processes in the system, the message

NO BATCH AVAILABLE

is given.

When a batch process is started, it immediately enters idle state because the batch queue is empty. It will be activated by the first @APPEND-BATCH command.

3.7.3.2 THE ABORT-BATCH COMMAND

This command will abort a batch process and release all resources reserved. Any job running is also aborted, the batch queue is cleared and the batch process becomes PASSIVE.

The format is

@ABORT-BATCH <n>

where <n> is the number given by the @BATCH command.

3.7.4 *Batch Commands for Public Users*

The following commands may be used by any user.

3.7.4.1 THE LIST-BATCH-PROCESS COMMAND

This command lists the status of the batch process in the system.

The format is:

@LIST-BATCH-PROCESS

Example of a @LIST-BATCH-PROCESS list for a system with three processes defined:

1. IDLE NO USER LOGGED ON

2. ACTIVE USER <user name> LOGGED ON

3. PASSIVE

The command has no effect on the batch process.

3.7.4.2 THE LIST-BATCH-QUEUE COMMAND

This command lists the contents of a specific batch queue.

The format is

@LIST-BATCH-QUEUE <n>

where <n> is a batch process number.

Example of a @LIST-BATCH-QUEUE list:

@LIST-BATCH-QUEUE 1

1 CARD READER LINE PRINTER
2 (USER NAME) BATIN LINE PRINTER

The command has no effect on the batch process.

3.7.4.3 THE APPEND-BATCH COMMAND

This command adds to the specified batch queue, a pair of batch input/output file names.

The format is

@APPEND-BATCH <n> <input file> <output file>

where

<n>

is the number of a batch process

<input file>

is the name of the file where batch-job-input is to be taken

<output file>

is the name of the file used to collect input.

If the batch process is IDLE, it will be ACTIVATED.

If the batch process is not started, the message BATCH PASSIVE is given.

If the batch queue is full, the message BATCH QUEUE FULL is given, and the user may try later.

NOTES:

- If the batch-input-file is owned by other than user SYSTEM, the user-name must precede the file name.

- The batch-input-file must have read access for all users having jobs on it, and for the user SYSTEM.

- The batch-output-file must have write append access for all users having jobs on the corresponding batch-input-file.

3.7.4.4 THE ABORT-JOB COMMAND

This command may be used to abort an ongoing batch job. The job will be aborted and the remaining activities in the job will be skipped. The next batch job will be initiated.

The format is

@ABORT-JOB <batch-number>, <user name>

where

<batch number>

designates which BATCH-PROCESS this job belongs to.

<user name>

is the name of the owner of the batch job.

If the <user name> is not logged on to the batch process at the moment an error message is given, the ongoing batch job will continue.

This command may be issued, either by user SYSTEM or the user currently logged on the batch process.

3.7.4.5 THE DELETE-BATCH-QUEUE-ENTRY COMMAND

This command may be used to remove an entry from the batch queue, that is, kill a job waiting for execution before it is started. If the job is started, the ABORT-JOB command must be used.

The format is

@DELETE-BATCH-QUEUE-ENTRY <batch number>, <input part of queue entry>, <output part of queue entry>

where

<batch number>

identifies the batch process

<input part of queue entry>

is the batch input file name given in the APPEND-BATCH command

<output part of queue entry>

is the batch output file name given in the APPEND-BATCH command

Exact match is required between the second and third parameter of this command and the batch queue entry. The exact format of the batch queue entry to be removed may be checked by the LIST-BATCH-QUEUE command.

If the batch queue contains two equivalent entries, the first one will be removed.

This command may be issued by the user SYSTEM and the user owning the job to be deleted.

3.7.5 *Batch Commands within the Batch-Job-File*

Certain commands must only be given within a batch-job-file.

3.7.5.1 THE ENTER COMMAND

This command must be the first command in a batch job. It identifies the *owner* of the job.

The format is

@ENTER <user name>, <password>,<project no>, <max time>

where

<user name>

must be a name of a legal user.

<password>

is the correct password for the user. If no password is defined, two commas (,) must be used to separate <user name> and <project no>. The password will not be printed on the output listing.

<project no>

should be given as a decimal integer if the accounting system is in use by the installation. If no project number is required, the entry should be given as two commas separating <password> and <max time>.

<max time>

is the maximum CPU minutes the total batch job requires, given as a decimal integer. If the time is reached, the batch job is aborted. Default time is 1 minute.

If the <user name> or <password> is not valid, the batch job will be aborted.

The ENTER command is ignored when running @MODE. This means that the same file can be used for @MODE as well as batch.

3.7.5.2 THE SCHEDULE COMMAND

This command is used to reserve devices for the current batch job, thus preventing errors from occurring if trying to use the devices already reserved by other users.

The format is

@SCHEDULE <device no>..<device no>

where

<device no>

is the octal value of the logical device numbers given in Appendix C. The most common logical device numbers are:

2 Paper tape reader
3 Paper tape punch
4 Card reader
5 Line printer

If one of the specified devices is reserved by other users, the batch process will enter waiting state until they are released.

To prevent deadlock situations, no device can be used by the batch job before a SCHEDULE command is given. If it is, the error message

DEVICE ALREADY RESERVED

is given, and the batch job will be aborted.

Most commands are allowed during a batch job but certain restrictions exist:

- The first character of a SINTRAN command line must be the @, which corresponds to the leading character typed by the system in interactive mode.

- All command parameters must be given on the same line as the command name itself because the system cannot ask for the missing parameters.

- Although they are legal, some commands are not very well suited within a batch job file. Among these are, for example, the @LOOK-AT command.

- The commands @LOGOUT and @MODE are not allowed within a batch job.

If errors occur within a batch job, the offending error message is written on the <batch-output-file> together with the message

BATCH JOB ABORTED

and the job will be aborted.

If an error occurs in accessing the batch input or output files, an error message will be given on the console terminal.

3.7.6 *Special Monitor Calls for Batch Jobs*

As a user program may be executed both as an interactive terminal program and as a batch program, it might be necessary to know in which mode the actual execution has been initiated from

A special monitor call is implemented:

MON 143 RSIO

Returned values:

A = 0 if interactive mode
= 1 if batch job
= 2 if @MODE job

T = file number of command input file

D = file number of command output file

X = user number in main directory and is achieved by the @LIST-USERS command.

If the program uses logical device number 1 for its input and output, the reading and writing automatically uses the batch input and output file currently assigned.

3.7.7 *Example of a Batch-Job-File*

The following shows the contents of a batch-job-file with the following tasks:

- a FORTRAN compilation
- loading the compiled program and some other subroutines, and loading system subroutines from the system library
- dumping the program for later usage
- executing the program, with data from the batch-input-file

The batch-job-file may look like:

```
@ENTER  USER-ONE <password>, <project no>, 2

@FTN
COM TEL,,100
PROGRAM A
-
-
-
END
EOF
EX
@LDR
A 100
A SUBLIB
A FTNLIBR
W1
H
@DUMP PROGA 60000 60000
@PROGA
-
-
-
@
ESC ESC
```

3.8 *THE SPOOLING SYSTEM*

A peripheral file may be created in more versions than the existing number of corresponding peripherals. All versions of the file not connected to a device number will be treated as spooling files.

Example:

```
@CREATE-FILE LINE-PRINTER;10,0
@SET-PERIPHERAL-FILE LINE-PRINTER, 5
@SET-FILE-ACCESS LINE-PRINTER WA RWA RWAD
```

There are now ten versions of the file LINE-PRINTER. The first version is a peripheral file with device number 5. The remaining files are spooling files.

Spooling files may be utilized for output spooling if the actual SINTRAN III system is generated with an optional spooling program for the peripheral in question.

If the system is generated with a spooling program, output spooling may be initiated with the command @START-SPOOLING with the peripheral file name as parameter.

Example:

The system is generated with a spooling program for device number 5 and the file LINE-PRINTER has ten versions; nine spooling files and one peripheral file with device number 5. Output spooling on the line printer can then be initiated with the command

```
@START-SPOOLING LINE-PRINTER
```

When output spooling is started with the @START-SPOOLING command, the peripheral (line-printer with device number 5 as in the above examples) is reserved by the spooling program and cannot be used directly.

Example:

The file LINE-PRINTER;1 is a peripheral file and spooling is initiated on this device as in the above examples. The command

@OPEN-FILE LINE-PRINTER;1 W will give the error message

FILE ALREADY RESERVED

When spooling is initiated, all output to the peripheral must go to the spooling files. When a user tries to open the peripheral he will not get the peripheral itself but the first free spooling file of that peripheral. When the file is closed, the file is linked to a spooling queue for the peripheral and eventually emptied on the peripheral.

Example:

```
@START-SPOOLING LINE-PRINTER
@COPY-FILE LINE-PRINTER USER-FILE-ONE
```

The file USER-FILE-ONE is copied onto a spooling file version of the file LINE-PRINTER. The spooling file is linked to the spooling queue when the COPY-FILE command is finished. The file is emptied while the user continues with other commands.

If more than one spooling file exists, then more than one user may open the peripheral at the same time or the same user may open the peripheral several times.

The spooling queue may be examined with the command LIST-SPOOLING-QUEUE with parameter peripheral file name.

Example:

```
@COPY-FILE LINE-PRINTER FILE-ONE
@COPY-FILE LINE-PRINTER FILE-TWO
@COPY-FILE LINE-PRINTER FILE-THREE
@LIST-SPOOLING-QUEUE LINE-PRINTER
(PACK-ONE:SYSTEM) LINE-PRINTER:;3, etc.
                        :
                        :
                        :
```

The first spooling file version (version 2 - with a copy of FILE-ONE) is not in the queue because printing of this file has already started.

A user may also insert his own file in the spooling queue and get a desired number of copies of the file. This is accomplished with the command @APPEND-SPOOLING-FILE with parameters, peripheral file name, name of the file to be appended to the spooling queue, and the number of copies desired.

Example:

```
@APPEND-SPOOLING-FILE LINE-PRINTER FILE-ONE 2
@APPEND-SPOOLING-FILE LINE-PRINTER FILE-TWO 1
```

A file may be removed from the queue with the @DELETE-SPOOLING-FILE command. The current print-out may be aborted or restarted with the commands @ABORT-PRINT or @RESTART-PRINT.

The spooling program may be discontinued with the @STOP-SPOOLING command. The spooling output is always discontinued after the end of the current print-file. The peripheral is released and may be accessed directly. The spooling files may still be used. These files and any user file may be inserted in the spooling queue. The spooling program will resume with the first file in the queue when the @START-SPOOLING command is eventually used.

The number of pages on the disk which may be used by the spooling files are limited to 500. The spooling files are usually files in the main directory that belong to user SYSTEM. The above mentioned limit is used to secure that no more than 500 user SYSTEM's pages are used for spooling. This limit may be changed with the following two commands:

@GIVE-SPOOLING-PAGES and
@TAKE-SPOOLING-PAGES.

These commands will increase or decrease the limit with the number of pages specified. The command @SPOOLING-PAGES-LEFT will return the number of pages still available.

If the system runs out of spooling pages (either by reaching the maximum limit of 500 or because user SYSTEM has no pages left), all user programs currently doing output to spooling files will enter a waiting state. The spooling program will then start printing one of the spooling files, return the pages to the pool of free spooling pages, and restart the waiting user programs.

3.8.1 *Spooling Commands for the User SYSTEM*

3.8.1.1 START SPOOLING

@START-SPOOLING <peripheral file name>
(Restricted)

Starts the spooling program for the specified peripheral. The peripheral will be reserved for the spooling program and the spooling program will print every file linked to the spooling queue for that device until the @STOP-SPOOLING command is used.

If more than one version of the file is a peripheral, the spooling programs for all peripheral versions of the file is started. One specific peripheral may be selected by including a version number in the file name.

An error message will appear if the specified file name is not the name of a peripheral or if no spooling program exists for a specified peripheral.

3.8.1.2 STOP SPOOLING

@STOP-SPOOLING <peripheral file name>
(Restricted)

Stops the spooling program for the specified peripheral and release the peripheral from the spooling program. Any file currently being printed by the spooling program will be completed before the spooling program is stopped. The spooling queue is unaffected by the command and files may still be appended to the queue. The spooling program will resume printing the files in the spooling queue when the @START-SPOOLING command is used.

3.8.2 *Spooling Commands for Public Users*

3.8.2.1 APPEND SPOOLING FILE

@APPEND-SPOOLING-FILE <peripheral file name>, <file name>, number of copies

The file specified in the second parameter is appended to the spooling queue for the specified peripheral. The specified number of copies of the file is printed in due time on the peripheral.

3.8.2.2 DELETE SPOOLING FILE

@DELETE-SPOOLING-FILE <peripheral file name>, <file name>

The file specified in the second parameter is removed from the spooling queue for the specified peripheral. Only user SYSTEM and the user which appended the file to the queue can delete the file from the queue.

3.8.2.3 LIST SPOOLING QUEUE

@LIST-SPOOLING-QUEUE <peripheral file name>

List all the entries in the spooling queue for the specified peripheral.

3.8.2.4 ABORT PRINT

@ABORT-PRINT <peripheral file name>

Aborts the current print-out on the specified peripheral and let the spooling program continue with the next file in the queue. The command has no effect if the spooling program for a specified peripheral is not started or if no file is being printed. Only user SYSTEM and the user which appended the file to the queue can abort the printing of the file.

3.8.2.5 RESTART PRINT

@RESTART-PRINT <peripheral file name>

Restart the printing of the file currently being processed by the spooling program. The command has no effect if the spooling program for the specified peripheral is not started or if no file is being printed. Only user SYSTEM and the user which appended the file to the queue can restart the printing of the file.

3.8.2.6 GIVE SPOOLING PAGES

@GIVE-SPOOLING-PAGES <number of pages>

Only a certain number of pages of the disk can be used by the spooling files. This number may be increased with this command. Note that the command does not guarantee that the disk space is available. This command is only intended to limit the number of disk pages used by the spooling system. 500 pages are initially given to the spooling system.

3.8.2.7 TAKE SPOOLING PAGES

@TAKE-SPOOLING-PAGES <number of pages>

This command may be used to decrease the number of pages the spooling files may use. The number of pages to be taken must be unused.

3.8.2.8 SPOOLING PAGES LEFT

@SPOOLING-PAGES-LEFT

Lists the number of pages left that can be used by the spooling files. Note that the number given is an upper limit and that the disk space may be used by other files.

4 MORE ABOUT SINTRAN III

4.1 *THE INTERRUPT SYSTEM*

The structure of SINTRAN III is greatly simplified by use of the different program levels in NORD-10. By running independent tasks on different program levels, all priority decisions are determined by hardware. This is extremely efficient because almost no overhead takes place due to the rapid level switching.

The NORD-10 has 16 program levels. Each of these has a complete register set, including A, D, T, X, B, L, P, Status and Paging Control Registers (PCR). A change of levels needs only 0.9 microseconds.

An interrupt to a special level means that the corresponding bit in the Priority Interrupt Detect (PID) register is set by an internal or external condition. The bits in this register may be set by hardware or software and cleared by software. A change to a special level is only legalized if the corresponding bit in the Priority Interrupt Enable (PIE) register has been set by the SINTRAN III operating system itself, by an RT program or by a direct task (see Section 4.6).

A change to a higher level is performed by hardware when an enabled interrupt occurs. If two or more interrupts appear simultaneously, the one with the highest priority (level) will be executed first. Interrupts to levels lower than the one running will activate that level, when all higher levels have given up priority, by executing the WAIT-instruction, which results in a drop to a lower level.

SINTRAN III uses the levels as follows:

15	Not used
14	Internal interrupt
13	Real time clock interrupt and driver
12	Input interrupt and drivers
11	Mass storage interrupt and drivers
10	Output interrupt and drivers
9	Free to be used for direct tasks (See Section 4.6)
8	
7	
6	
5	Monitor
4	Free to be used
3	RT programs, Time-sharing and Batch programs
2	Free to be used for direct tasks
1	
0	Idle loop.

Level 15 is not used.

Level 14 is activated by monitor calls (the MON instruction) or by internal errors detected by hardware. Possible error conditions are: power fail, memory out of range, memory parity error, IOX time out error, privileged instruction, Z indicator is set, illegal instruction, page fault and memory protect violation.

Level 13 receives an interrupt each 20th millisecond. This leads to an updating of the real-time clock.

Levels 12-10 are activated by external I/O interrupts. The appropriate driver routine will be started. Interrupts concerning byte/word oriented input operations have higher priority than interrupts related to mass storage devices with direct memory access and byte/word oriented output operations. The reason for this is that such input operations are not always initiated and controlled by the SINTRAN III operating system itself, but may also be activated by an external process or by a keyboard operator.

Level 5 is occupied by the SINTRAN III monitor, which performs all administration jobs such as: starting up, aborting and terminating user programs, reserves and releases resources, segment and page handling, etc.

User programs are executed on level 3.

When an interrupt to a higher level occurs, the execution starts from the instruction pointed at by the P register of that level. A short interrupt analysis might take place and the most necessary and time critical functions will be performed.

Normally, levels 10-14 complete their operations by activating the SINTRAN III monitor on level 5, which executes some administration jobs, i.e., loads a new RT program, changes priority of background programs, etc., before control resumes by a user program on level 3.

Level 0 is occupied by the *idle loop*. This is a small program which increments the D register with 1 each time it runs through the loop. By displaying the D register of level 0 on the operator's panel, it is possible to observe the load of the activity in the NORD-10 computer. With no other activity in the computer, the idle loop counts up the D register completely, approximately 40 - 50 times per minute. This may be measured exactly by watching how many times the most significant bit of the D register lights up during one minute. This number may be compared with the corresponding number when there is a certain activity in the system. The quotient may, thus, supply the user with a rough idea of the loading of the computer under certain conditions.

The X register on level 0 contains the least significant part of the internal clock and is incremented each 20th millisecond. The T register contains the contents of the memory location addressed by the switch register on the operator's panel. This might be utilized to observe the dynamic change of certain memory locations.

Levels 9 - 6, 4 and 2 - 1 are not covered by the SINTRAN III operating system but are free to be used by direct tasks (see Section 4.6). Programs implemented on these levels will run completely independant of SINTRAN III and may not use any monitor calls, files or other facilities supplied by the operating system.

4.2 *MEMORY MANAGEMENT SYSTEM*

The Memory Management System includes a *paging system,* a *permit protection system,* and a *ring protection system.*

In SINTRAN III the memory management system is used for the following purposes:

1. Dynamic memory allocation and paging. The page size is 1K words (= 1024 words).

2. Extension of maximum physical address space from 64K words to 256K words.

3. Extension of maximum program size (virtual address area) to 64K word independent of the physical memory size.

4. Memory protection between parts of a program, detecting attempts to modify read-only areas or executing data.

The smallest part to be given permit protection and ring protection is a *segment.* (See Section 4.4.) The protection bits of a segment are set by the RT loader when the segment is built.

4.2.1 *The Users Virtual Memory Space*

The Timesharing and Batch users running under the SINTRAN III system may have a logical (or virtual) address space for a program up to 64K, all on one segment. This virtual address space is normally created by one of the loaders.

The transformation of a logical (virtual) address to a physical address is performed by the paging system by means of the hardware page index tables. (See Section 4.2.2)

The virtual address space is like a contiguous area, but the corresponding physical pages may be scattered throughout the physical memory. When a part of the program which is not in the memory is addressed, the SINTRAN III system will make it available in pages of 1K. If there is no free page in memory, the SINTRAN III system will make space available by writing one of the least recently used pages back on the intermediate store. Thus, the programs in Timesharing/Batch mode execute in a *Demand-page mode,* and may utilize less physical memory than the logical address space should require, however, the turnaround time will be longer due to additional reading and writing of intermediate pages. (See Section 4.4.)

In addition the SINTRAN III system uses a 4K system segment, sharable between all terminals and batch processes, containing the Command Interpreting Processor and some additional routines.

4.2.2 *The Hardware Page Index Tables (PIT)*

NORD-10 has four page index tables. Each of these contain 64 16-bit words, thus, covering the full 64K virtual address space.

Each page index table contains one entry for each logical page, thus it maps the full 64K virtual (logical) address space into the up to 256K physical address space. 16-bits logical addresses are transformed into 18-bits physical addresses which consist of an 8-bits physical page number (block number) and a 10-bits displacement.

Each entry in the page index table has the following format:

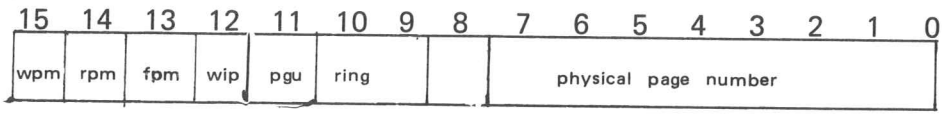


Figure 4.1.

Bits 0-7: Physical page number.
8 bits give a maximum of 256 pages.

Bit 8: Not used.

Bits 9-10: Ring number.
These bits decide which ring the page belongs to.

Bit 11: Page used.
This bit is automatically set by hardware whenever the page is accessed and then remains set. The bit is cleared by program.

Bit 12: Written in page.
This bit is automatically set equal to one the first time a write into the page occurs and then remains set. It is cleared by program (whenever a new page is brought from mass storage). If this bit is set, the page is written back to mass storage before it is replaced.

Bit 13: Fetch permitted.
FPM = 0: Locations in this page may not be executed as instructions.
FPM = 1: Locations in this page may be executed as instructions.

Bit 14:
- RPM = 0: Locations in this page may not be read (they may be executed).
- RPM = 1: Locations in this page may be read if the ring bits allow.

Bit 15: Write permitted.
- WPM = 0: It is impossible to write into locations in this page.
- WPM = 1: Locations in this page may be written into if the ring bits allow.

The four hardware page index tables are used like this:

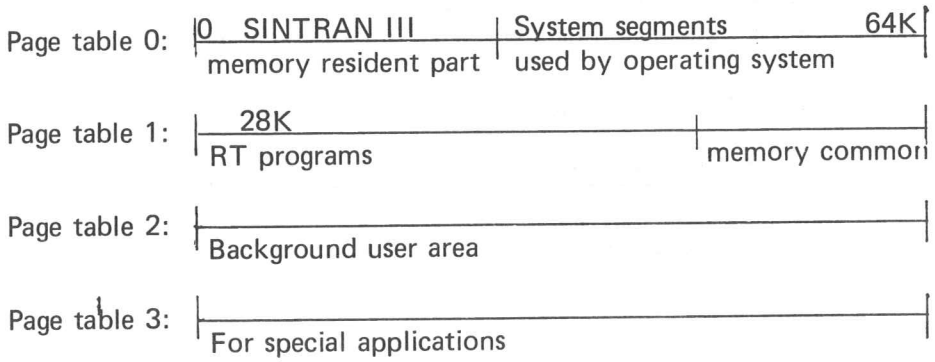


Figure 4.2.

This means that a background program can address a virtual area of 64K, and so can an RT program, including the memory common area.

The lower part of the page index table 0 and the memory common part of page index table 1 will be initialized at system start and will never be changed, i.e., the corresponding pages will never be swapped out of memory (resident memory). The other entries in the page index tables will contain entries for the segments of the currently running program. The unused table entries will contain zero, so that trying to use their corresponding logical address will result in error.

The Page Index Tables are placed in a bipolar memory (high speed memory), also denoted as *shadow memory* because it has addresses from 177400 to 177777 (octal) and, thus, lies in the shadow of the normal memory area with the same addresses. (See also Section 4.2.6.)

4.2.3 *The Paging System*

The transformation of a 16-bits virtual address into an 18-bits physical address is done by hardware when the Memory Management System is switched on. To understand this transformation, for a moment disregard the fact that there are four page index tables, but consider Figure 4.3.

To address any location within a 1K address space (in this case 1 page) 10 bits are required. This part of the physical address represents the displacement within a page and occupies bits 0-9. These bits are taken directly from bits 0-9 of the virtual address.

Now bits 10-15 of the virtual address are used as an index to choose one of the 64 entries in the page index table. From the selected entry, the lower 8 bits that constitute a physical page number (one of 256 pages), are transferred to bits 10-17 of the physical address.

To decide which of the four page index tables to choose, two bits in the paging control register, of the present interrupt level, are utilized by the paging system.

For further explanation of the paging system, see "Course Manual UH10 - Introduction to NORD-10 CPU".

4.2.4 *The Permit Protection System*

The Read, Write and Fetch Permit Protection System is implemented by defining, in bits 13-15 in each PIT entry, how the page may be used. In hardware, this information is compared with the instruction being executed, i.e. if it is load (read), store (write), instruction fetch or indirect address.

Bit 15: WPM - Write Permitted.
If an attempt is made to write into a write protected page, an internal interrupt (memory protect violation) to program level 14 will occur, and no writing will take place.

Bit 14: RPM - Read permitted.
If an attempt is made to read from a read protected page, an internal interrupt (memory protect violation) to program level 14 will occur, and no reading will take place.

Bit 13: FPM - Fetch Permitted.
If an attempt is made to execute instructions from a fetch protected page, an internal interrupt (memory protect violation) to program level 14 will occur, and the execution will not be started.

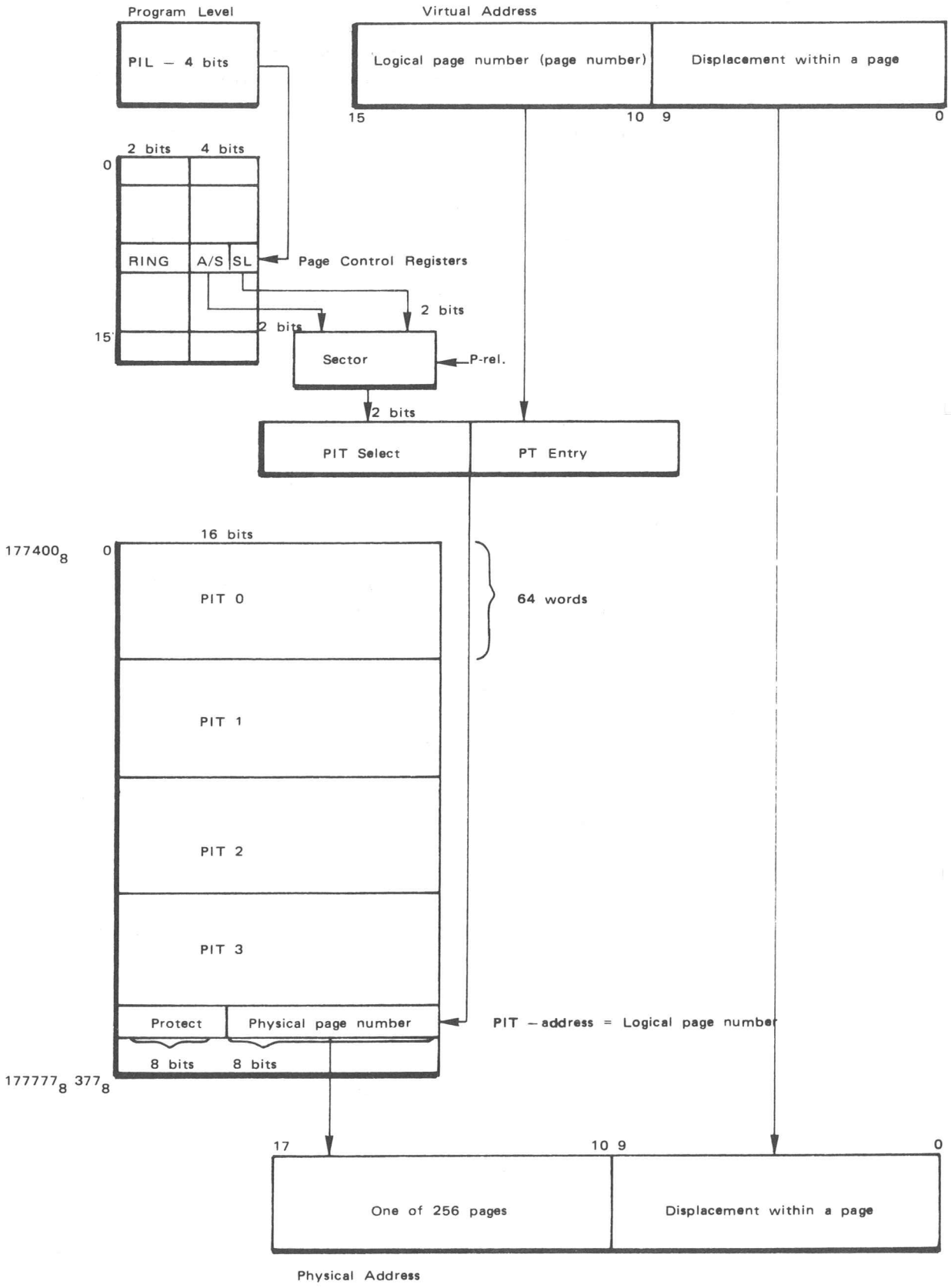


Figure 4.3: *Virtual to Physical Address Translation*

(See also Section 4.2.2 and Figure 4.1.)

Indirect addresses may be taken both from pages which have FPM = 1 and from pages which have RPM = 1.

All combinations of WPM, RPM and FPM are permitted. However, the combination where WPM, RPM and FPM are all zero is interpreted as Page Not In Memory and will generate an internal interrupt with the internal interrupt code equal to Page Fault.

The smallest user area to be given permit protection is one segment (see Section 4.4.).

4.2.5 *The Ring Protection System*

The Memory Management System includes a Ring Protection System, where 64K virtual address space is divided into four different classes of programs, or rings.

Two bits in each page index table entry are used to specify which ring the page belongs to. Refer to Figure 4.1.

The ring bits have the following meaning:

Bit 10, 9

0 0 Ring 0:

The program may only access (read or write) locations with ring zero. This access is controlled by the RPM, WPM, and FPM bits. Locations outside ring 0 are completely inaccessible.

0 1 Ring 1:

The program may access locations in ring 1 and ring 0. Access is controlled by the RPM, WPM, and FPM bits.

1 0 Ring 2:

The program may access locations in ring 2, 1 and 0. The access is controlled by the RPM, WPM and FPM bits.

1 1 Ring 3:

The whole address space is accessible if not protected by the RPM, WPM and FPM bits.

An illegal ring access will cause an internal interrupt (memory protect violation) to program level 14, and the instruction which caused the interrupt will not be executed.

The recommended way of using the ring bits is as follows:

Ring 0: User Programs

Ring 1: Compilers, Assemblers

Ring 2: Operating System (Utilities, Commands)

Ring 3: Kernel of Operating Systems

Associated with the ring bits in a PIT entry are the two ring bits in the current program level's Paging Control Register – PCR.

Before a program can start executing, the PCR on relevant program level is loaded by the Operating System with information as to which PIT, Alternative PIT and Ring is to be used. The program's PIT must also be loaded by the Operating System prior to execution. When execution starts, the ring bits in the PCR will be compared with the ring bits in the PIT entry, by hardware. The two rings should always be equal, otherwise an internal interrupt will be generated. (Except when going from a higher to a lower ring.)

By utilizing the four-mode ring protection system, SINTRAN III offers an efficient protecting system. A program which is placed on a specific ring cannot be accessed by a program that resides on a ring of lower priority.

This system is used to protect RT programs on a higher ring from RT programs on a lower ring, and to protect the memory common area from RT programs on ring 0. Ring 3 and ring 2 are used for the system kernel and its subsystems (Operator Communication, RT Loader, File Management, etc.), while ring 1 and ring 0 are used for user RT programs. Timesharing/batch programs run on ring 0. The user programs are individually protected by the paging system. The smallest user area to be given ring protection is one segment. (See Section 4.4.)

In addition to these protect features, the ring protection system equips SINTRAN III with a set of *privileged instructions* legal only on ring 3 and ring 2, for use by SINTRAN III's kernel and its subsystems, and for RT programs which may execute some of the privileged instructions. These instructions are of the type which could be disastrous if executed by a user's program. For any on-line system with a large number of potentially undebugged programs, this protection system is extremely important. Direct tasks (see Section 4.6) will reside on ring 2.

4.2.6 *Privileged Instructions*

In a multiuser-multitask system, a user is not permitted to use all instructions in the instruction set. Some instructions may only be used by the Operating System, and this category of instructions is called Privileged Instructions.

Privileged Instructions:

- Input/Output instructions.
- All instructions which control the Memory Management and Interrupt System.
- Interprogram level communication instructions.

The only instruction the user has available for user-system communication is the Monitor Call Instruction – MON. The MON instruction may have up to 256 different parameters or calls. When the machine executes the MON instruction it generates an internal interrupt. A list of the monitor calls is given in Appendix B.

The privileged instructions may only be executed on rings 2 and 3, i.e., only by SINTRAN III or by RT programs placed on ring 2. If programs on ring 0 and 1 try to execute any privileged instruction, an internal interrupt to level 14 will be generated, and the instruction will not be executed.

The Page Index Tables may only be accessed (read from or written into) by software, either by programs residing on ring 3 or by any program while the Memory Management System is switched off. When the Memory Management System is switched off there is no hardware checking of the permit protection and ring protection bits and no control of privileged instructions. In this case, the physical address is equal to the virtual address.

4.3 *REAL-TIME PROCESSING*

Real-time processing allows the user to perform time dependent and time critical work that requires very rapid information processing. Real-time processing is used primarily in applications where data gathered during a physical process must be inputed and operated upon so quickly, that the results can be used to influence the process as it develops. Real-time processing is also used in many on-line commercial applications where a guaranteed response time is required.

A real-time program, called *RT program*, generally responds to or controls external events. Under real-time processing, there are four principal methods of activating programs.

- external interrupts
- program requests
- operator's requests and
- time scheduling.

The programs may have a full range of executing times, frequencies and start conditions. The system ensures that the most important RT program will always be run first by providing 256 program priority levels with any number of programs on each level.

The priority range is 0-255. However, an RT program with priority 0 will never be executed, because a system RT program called DUMMY is always present, running with priority 0. Time-sharing and batch jobs run on priority levels 16, 32, 40 and 48 (20_8, 40_8, 50_8 and 60_8).

Each RT program has an *RT description,* which contains information about the RT program needed by the system. An RT description occupies 20 locations and all RT descriptions are set up in a contiguous table in resident core, the *RT description table.* The size of this table is specified at system generation time. The RT loader will create the RT description at load time of an RT program. An RT program is specified by the address of its RT description or by the program name. There will always be some system RT programs included in a SINTRAN III system, one for each terminal which may be used as a timesharing terminal, one for each batch process and some other system RT programs.

Because the RT programs are activated at random points of time,the situation might occur that two or more programs want to start at the same time. The natural way of administrating this problem is to list the RT descriptions of the programs to be executed in an *execution queue,* where some programs are waiting while another is being processed.

The RT descriptions in the execution queue are sorted by priority. A program which must be activated rapidly, should be given a high priority, so that its RT description is placed before the others in the execution queue. An RT program may also interrupt the present running program when this one has a lower priority. But the real-time monitor stores away information, so that the interrupted program may continue afterwards.

As mentioned in this section, an RT program may be scheduled for execution at a certain point in time. The RT descriptions of these programs, that are waiting for activation in this manner, are listed in the *time queue.* In this queue the RT descriptions are sorted by starting time. When the activation time for an RT program occurs, its RT description is moved from the time queue into the execution queue. An RT description may also be entered directly into the execution queue without passing through the time queue.

A detailed description of how to administrate RT programs is given in Chapter 5.

A description of the real-time monitor calls available in MAC, NORD PL and FORTRAN and a description of the commands in the RT loader are given in Chapter 7. The RT Loader is described completely in a separate manual.

4.4 *PROGRAM STRUCTURE – SEGMENTS*

The basic program concept is the *segment.* A segment is a contiguous and limited virtual address area (on mass storage). At execution time, the segment (or part of it) is loaded to memory. A virtual address area is an image of the memory. There may be many of these images (many segments). The image of the memory may be greater than the physical memory. A segment is limited to 64K because the maximal magnitude of a virtual address space is 64K. (64K is the possible space to address with 16 bits.) A program which has a program area of 64K and a data area of 64K cannot be effectuated under SINTRAN III, but only on a NORD-10 computer without SINTRAN III.

In physical core, a segment may be scattered because of the hardware paging system. A segment will always consist of a multiple of entire pages of 1K words.

A segment is specified by its number, the *segment number.* The segment number range is from 0 to 254, but the maximum segment number range available in a SINTRAN III system is specified at system generation time (when the operating system is compiled).

Each segment has a *segment description* containing necessary information about the segment needed by the system. A segment description occupies 5 locations and all segment descriptions are set up in a contiguous table in resident core, *the segment table.* The segment description is set up by the RT loader and put into the table when the segment is built.

- A segment may consist of a set of reentrant subroutines used by different RT programs.
- A segment may consist of common data areas for different RT programs.
- A program may have its code on one segment and its data on the other.
- Several RT programs may be on the same segment.

There are two types of segments:

- Non-demand segments, all of which must be in core before the program can be started.
- Demand-segment, only part of which is needed at a time. If a *page fault interrupt* occurs, the monitor will fetch the missing page, and the program will continue.

Non-demand segments are normally used for real-time programs, because of short and well-defined transfer times and fast monitor call handling.

Demand segments are used when a segment is too big to be in core at a time. The "users area" of all time-sharing/batch jobs are demand segments.

The segment type is determined by the programmer through the RT loader at load time.

An RT program may, concurrently, have a maximum of two segments in the virtual memory. These segments should not overlap. The initial segments used by an RT program are decided when the RT program is loaded. One or both segments may be temporarily exchanged with other segments during the execution of the RT program, using the monitor calls MCALL and MEXIT. Whenever an RT program is started again, it uses its original initial segment numbers. The technique of exchanging segments may be used for program segmenting.

The smallest user area to be given permit protection and ring protection is one segment. This implies that all pages belonging to the same segment have the permit protection and ring protection bits set equally. The protection of a segment may be determined by the programmer at load time through a special command to the RT loader.

A segment can be fixed in memory by means of a monitor call so that it will not be swapped out until it is released again.

In addition to the segments, the RT programs may also have access to a *memory resident common data area.* This area is placed on protection ring 1, so that programs on ring 0 cannot access this area. The size of the memory common area is specified at system generation time.

The SINTRAN III system itself uses the following segments:

Octal Segment Number	
0	Used by the monitor
1	Used by the monitor
2	Core image
3	Operator Communication segment
4	RT Loader segment
5	Error Program segment
6	File Management segment
7	MACD segment
10	System segment, Terminal I
11	User's segment, Terminal I
12	User's segment, Terminal I

There will also be one system segment and one user segment for each additional terminal used for timesharing and for each batch process.

4.5 *RESERVATION OF LOGICAL UNITS (RESOURCES) FROM RT PROGRAMS*

Logical units may be:

- I/O devices
- Semaphores
- Internal devices

All logical units must be reserved before they may be used by an RT program.

They are reserved as follows:

1. by the RT program itself, using Monitor Call RESRV,
2. from another RT program using Monitor Call PRSRV,
3. form a terminal by the commands RESRV and PRSRV.

All resources (I/O devices, semaphores, internal devices, etc.) used by an RT program must be reserved for the RT program by the monitor calls RESRV or PRSRV, before the RT program is allowed to use the resources. If the resource is already reserved by another RT program, the RT program trying to reserve the resource may be set in a waiting state until the resource will be free, or the RT program may continue without using the actual resource. (See monitor calls RESRV, PRSRV, Section 7.7.1.) The resources reserved by an RT program will, normally, be released when the RT program is terminated, or the resources may be released by the monitor calls RELES or PRLS. The resources reserved by an RT program will not be released if the RT program has given up control of its latest execution with the monitor calls HOLD or RTWT.

4.5.1 *Semaphores*

A semaphore is a binary variable which can have one of two values: reserved or unreserved.

A semaphore represents a resource that RT programs may reserve and release. The semaphores are used when logical units or program units are looked upon as common resources for other RT programs. The calling RT programs utilize a semaphore by reserving and releasing it to prevent other RT programs from a concurrent use of the same resource.

For instance, if a reentrant RT program system contains a nonreentrant subroutine, a semaphore may be used to prevent a concurrent execution of this nonreentrant subroutine.

Semaphores may also be utilized in an RT program system to avoid a simultaneous updating of global data (for instance memory common) by two or more RT programs.

There is a one-to-one relationship between a semaphore and a resource given by the designer of an RT program system.

The semaphores are free to be utilized by different RT programs wanting to reserve resources.

A semaphore consists of 4 memory locations (words) in resident memory, the first of which points to the next element in the *reservation queue* of resources reserved by the same RT program. The third location points to the RT description of the first waiting program, if any. In this respect, semaphores are also utilized for creating a *waiting queue* of RT programs wanting to reserve a resource already occupied.

Semaphores have the logical number 300_8 - 377_8, but the actual number of semaphores available in a SINTRAN III system is decided at system generation time.

4.5.2 *Files and RT Programs*

Before any use of files is permitted in RT programs, the command RTENTER must have been executed at least once after the system is started.

An RT program may use the files in the same way as a time-sharing/batch program, i.e., all monitor calls concerning files are available in RT programs, except RDISK and VDISK.

An RT program simulates the user RT to the file system, i.e., if an RT program uses a file name in a monitor call without specifying the file owner's name, the file system will act in the same way as if the user RT was logged in and specified, the same file name from a time-sharing program.

A mass storage file opened by an RT program is open for all RT programs in the system.

Peripheral files (line printers, tape readers, etc.) are usually used from RT programs without using the file management system. Peripheral files are reserved by RT programs and then operated upon, without being opened. Mass storage files should be reserved as a logical unit.

There is a difference between the CALL OPEN statement (open file) used in ordinary FORTRAN, where the calling program assigns a file number to the file, and the use in reentrant FORTRAN, where the program must utilize the file number returned from the file management system.

In ordinary FORTRAN, the file number given by the programmer is not used internally, but there will be a correspondance between the internal file number and the number given by the programmer. RT programs wanting to use a file opened by another RT program *should use the internal numbers!* which are 100_8 for the first opened file, 101_8 for the second, 102_8 for the next, etc. (This correspondence is established by the FORTRAN run time system, which receives the internal file number from the file management system.)

Examples:

1.
```
PROGRAM COPEN, 25
DIMENSION IARRAY (512)
INTEGER OPEN, RFILE
IBLOCK = 0
NWORD = 512
IERR = OPEN ('DATA REGISTER:DATA  ',150,2)
IF (IERR.NE.0) GO TO 10
IERR = RFILE (150, 0, IARRAY, IBLOCK, NWORD)
```

2.
```
PROGRAM COPEN,25
DIMENSION IARRAY (512)
INTEGER OPEN, RFILE
IBLOCK = 0
NWORD = 512
IERR = OPEN ('DATA REGISTER:DATA  ',IFILE,2)
IF (IERR.NE.0) GO TO 10
IERR = RFILE (IFILE, 0, IARRAY, IBLOCK, NWORD)
```

3.
```
PROGRAM COPEN, 25
DIMENSION IARRAY (512)
INTEGER OPEN, RFILE
IBLOCK = 0
NWORD = 512
IFILE = 150
IERR = OPEN ('DATA REGISTER:DATA  ',IFILE,2)
IF (IERR.NE.0) GO TO 10
IERR = RFILE (IFILE, 0, IARRAY, IBLOCK, NWORD)
```

Example 1 is to be compiled in normal mode. The file number 150 is given by the programmer.

Example 2 is compiled in reentrant mode. The variable IFILE will contain the file number after the OPEN call. The file number is given by the file system.

Caution: The command RT to the FORTRAN compiler generates reentrant code. "RT", in this connection, does not mean "Real-Time"!

Example 3 will work in both cases.

4.5.3 *Internal Devices*

An internal device has two *data fields*, with one common *ring buffer*, used for communication between RT programs. The size of the ring buffer and the number of internal devices available in a SINTRAN III system is established at system generation time. The internal devices are byte oriented, i.e., one byte at a time will be transferred to/from the internal devices (ring buffers). Optionally, 16 bits words in each INBT/OUTBT monitor call can be transferred.

To understand the concept of ring buffers, imagine two routines or programs, one is writing into the buffer, the other is reading from it. When the writing routine comes to the end, it starts from the beginning again if the reading routine has read some characters already and, thus, made some space available at the top of the ring buffer. The reading routine acts in a similar way. Having reached the end, it continues reading from the top of the buffer if there are some characters to pick up.

This is administrated by means of two pointers, a write pointer and a read pointer. The ring buffer is empty when the read pointer reaches the write pointer and is full when the write pointer reaches the read pointer. Both of these situations cause the calling RT program to enter a waiting state. Special monitor calls (ISIZE, OSIZE) give the actual number of bytes occupied in the ring buffer, and may then be applied to avoid the waiting state. (See Section 3.6.1.)

The RT programs using an internal device, must reserve the device and then use the monitor calls INBT/OUTBT to access the internal device. From a FORTRAN program, the internal device may be accessed in the same way as an ordinary peripheral device (Teletype, line printer, etc.), using READ/WRITE, INPUT/OUTPUT or CALL INCH/OUTCH statements. The internal devices have the device numbers 200_8 - 277_8, but the actual number of internal devices in a SINTRAN III system is decided at system generation time.

A data field is an internal table containing information about the corresponding device (in this case, the ring buffer). The data field is used by the SINTRAN III operating system for administration of the device.

4.6 *DIRECT TASKS*

A Direct Task is a routine running on one of the free interrupt levels 1, 2, 4, 6, 7, 8, or 9. A Direct Task may be started by an RT program by activating (by the instruction MST PID for instance) the interrupt level where the Direct Task is placed. Otherwise, *a Direct Task will run independent of the SINTRAN III System,* and a Direct Task cannot use monitor calls, files or other facilities in SINTRAN III. If a Direct Task is running on a higher interrupt level than the SINTRAN III monitor (usually level 5), the SINTRAN III monitor will not be restarted before the Direct Task has terminated, by giving up the priority (by the WAIT instruction for instance), while a Direct Task running on a lower interrupt level than the RT programs (usually level 3) will not be active unless there are no other activities in the system.

4.6.1 *The Implementation of a Direct Task into a SINTRAN III System*

First, the direct task routine must be loaded to a segment using the RT loader. Then, the routine should be fixed in core by using the monitor call FIX or FIXC. Finally, the ENTSG monitor call should be used to enter a direct task or a driver routine into the operating system. (See Section 7.7.1.) The given page index table may be different from the page index table set by the RT loader. This means that the segment may be reached through two page index tables, simultaneously, with different protect settings. The ENTSG monitor call will always set read, write and fetch permitted in the page index table.

The start address will be put into the P register of the specified interrupt level.

A new driver routine on interrupt level 10, 11, 12, or 13 may be entered into a SINTRAN III system in the same way as a Direct Task, but datafields and entries in the logical unit number tables must be established at system generation time, or by use of MACM before the SINTRAN III system is started.

4.6.2 *Calling RT Programs from Direct Tasks*

When a Direct Task wants to start an RT-program, a subroutine in SINTRAN III can be called:

```
LDA (ELEM
JPL I (RTDIR          % SINTRAN III SUBROUTINE
      :
      :
ELEM, RTPRG; 0; 0; 0; 0;
```

The A-register points to an element of 5 locations. The first is a pointer to the RT description of the RT-program; the rest is used for working space for RTDIR.

Since RTDIR is on page index table 0, and the parameter element is also on page index table 0, this mechanism can be used directly only from Direct Tasks on page index table 0. However, using level 14 and a few locations on PIT 0, it can be used also from other page index tables, for example number 3.

Example:

```
% Code on page table 3

:
IOF
LDA RTDSC                  % POINTER TO RT-DESCRIPTION
IRW 160 DT                 % SET REGISTER ON LEVEL 14
LDA (ADDR1;IRW 160DA       % ADDRESS TO PARAMETER EL-
                             EMENT
LDA (PRTDR;IRW 160 DP      % ROUTINE ON PAGE TABLE ZERO
LDA (40000; MST PID        % ACTIVATE LEVEL 14
ION
:
:

% SOMEWHERE ON PAGE INDEX TABLE ZERO:

% ROUTINE ON LEVEL 14:

PRTDR,   COPY SA DX
         STT, X            % STORE INTO ELEMENT
         JPL I (RTDIR      % SINTRAN III SUBROUTINE
         WAIT
         JMP I (ENT14      % SINTRAN III ROUTINE
         )FILL

% PARAMETER ELEMENTS ON PAGE TABLE ZERO
% ONE FOR EACH DIRECT TASK

ADDR1, 0; 0; 0; 0
ADDR2, 0; 0; 0; 0
ADDR3, 0; 0; 0; 0
```

4.6.3 *Activation of Direct Tasks from Interrupts*

A general driver to activate direct tasks has been implemented. Each level has a logical number with a corresponding data field. The numbers are:

- 440_8 - level 6
- 441_8 - level 7
- 442_8 - level 8
- 443_8 - level 9

If, for example, level 7 should be activated by a certain CAMAC interrupt, the monitor call ASSIG can set up the connection:

CALL ASSIG (441B, LAMX, !CRATE)
(See Chapter 3.10.)

In this case, all patching is avoided. If the direct task should be started by a non-CAMAC interrupt, the data field pointer must be put into the proper IDENT or EXTEND table.

The driver increments a location in the data field each time an interrupt occurs. This can be used by the direct task to detect whether some interrupts have come too fast to be processed by the direct task. If the direct task wants to check this, the exit sequence can be as follows:

```
DISP 4;!NTEGER LEV, COUNT;PSID        % in data field
*IOF
COUNT –1 = :COUNT
IF = 0 THEN LEV; *MCL PID             % give up level
FI
*ION
GO START                              % next execution
```

Page index table 3 is available for direct tasks. On page index table 0 the available area starts on the first page following the core resident section and goes up to location 67777_8.

5 THE USER RT

5.1 *THE PURPOSE OF THE USER RT*

A number of programs will normally be run as Real Time programs (RT programs). RT programs are discussed in Chapter 7.

Examples of programs which would be implemented as RT programs are:

- programs which should be started and executed within a given time interval
- programs which should be started and executed when an external event (interrupt) occurs
- programs which should be started and executed on demand from another RT routine
- programs that should have higher (or lower) priority than timesharing users
- etc.

These programs will affect the total SINTRAN III system, seen by a timesharing user:

- availability of the system's resources (CPU and peripherals will be reduced)
- the CPU time is not time-sliced between RT programs. They are run solely on a priority basis. The RT program with the highest priority and something to do will always be running. The response time for timesharing users may, therefore, be affected.
- as will be shown in Chapter 7, some RT programs may be allowed to execute all NORD-10 instructions, and thereby even stop the system

It is, therefore, clear that RT programs should be handled with some care, and that a good knowledge of the operating system and installation is required to implement such programs.

A special timesharing user, user RT, is allowed to supervise the RT programs. This user acts like any other timesharing user to the system. It has the same rules for logging in and out, has space for files and may execute programs in timesharing.

The difference, compared to other users, is the availability of an extended command set. The extra commands give the necessary control functions.

It is important to distinguish between user RT, who is a timesharing user with some privileges (commands), and the RT programs which are self-contained programs that can execute independent of timesharing users and terminals.

The function of user RT is to load the RT programs, debug them in real-time, supervise the execution of the programs and be responsible for the files used by the RT programs. In the following paragraphs, all commands are listed which are special for user RT.

5.2 *THE REAL-TIME LOADER*

All RT programs must be loaded onto segments by the RT loader as described in Section 7.6.

The loader is activated by the command

@RT-LOADER

and will answer by typing its version number.

User RT must have Write and Append access to a file named RTFIL:DATA, owned by the user SYSTEM, in order to be able to use the RT LOADER.

5.3 *FILE HANDLING COMMANDS*

To set up RT programs as a user of files, the following command must be given

@RTENTER

This must be done before any file can be used by RT programs, and every time the system is restarted. However, if the command INITIAL-COMMAND has been given before the system was last started, the RTENTER command is not needed.

When an RT program accesses a file, user RT is simulated to the file system. If no owner is specified in the file name, user RT is assumed to be the owner.

The above command has no effect on user RT's access to the file system as a timesharing user.

The command

@RTOPEN-FILE <filename> <open access>

works like the OPEN-FILE command, except that the file is opened for use by RT programs, not for the logged on user.

The returned file number can be used by all RT programs for accessing the file.

Note that files can also be opened by calls from an RT program. These files will be accessible from all RT programs by using the returned file number.

Example:

@RTOPEN-FILE DATAFIL1, RX
FILE NUMBER IS 100

Blocks from DATAFIL1 can now be read by all RT programs using 100_8 as file number.

The command

@RTCONNECT-FILE <file number> <file number> <open access>

acts like the previous command, except that the file number to be used is supplied in the command. If the file number is out of range or already used the file will not be opened.

@RTCLOSE-FILE <file number>

will close the specified file for real-time programs.

@LIST-RTOPENED-FILES

will list all files opened for RT programs.

5.4 *THE LOOK-AT COMMAND*

As seen in Section 3.3.6.2, contents of the memory, or the exact contents of the 64K program area, may be examined and changed by the LOOK-AT command. This command is more powerful for user RT, the exact syntax being :

@LOOK-AT <area>

where <area> may be:

MEMORY: the users 64K address area, exactly as for other users.

SEGMENT <number>: one of the program segments.

RTCOMMON: core common area for RT programs.

By this command locations in the loaded programs may be inspected and changed and core common areas may be addressed. This feature is useful for debugging and checking of state and inspecting variables in running programs.

(This command is also mentioned in Sections 3.3.6.2 and 6.4.3.)

5.5 *MONITOR COMMANDS*

When running programs in timesharing mode, user RT cannot run programs that cannot be run by any other user. As will be shown in Chapter 7, RT programs may use some monitor calls not available for timesharing users. Some of these monitor calls may be given as commands by user RT and are listed here. Only the syntax and one example is given for each. The calls are described in more detail in Section 7.6.

@RT <program name>

@RT KLOKK will set the program KLOKK up for immediate execution, i.e., put the RT description into the execution queue.

@SET <program name> <time> <time unit>

@SET KLOKK, 5.3 will set KLOKK up for execution in 5 minutes. The RT description will be put into the time queue. Five minutes later it will be moved to the execution queue.

@ABSET <prog. name> <second> <minute> <hour>

@ABSET KLOKK, 0,10,13 will set KLOKK up for execution at 1:10 PM (13:10). The RT description will be put into the time queue. At 1:10 PM it will be moved to the execution queue.

@INTV <prog. name> <time> <time unit>

@INTV KLOKK 30,2 will be prepared for execution every 30 seconds. First, the RT description will be put into the time queue. 30 seconds later it will be moved to the execution queue and at the same time put into the time queue again for a new execution another 30 seconds later. This will be repeated periodically. The sequence of executions may be stopped by the command @DSCNT.

@ABORT <prog. name>

@ABORT KLOKK will abort program KLOKK if it is running, or in waiting state, remove it from the time queue and execution queue, release all resources and prevent periodical execution.

@CONCT <prog. name> <logical unit>

@CONCT KLOKK, 410B. The RT description of KLOKK will be put into the execution queue every time an interrupt occurs on device 410_8.

@DSCNT <prog. name>

@DSCNT KLOKK. All connections made by CONCT are disconnected. Interval execution is prevented. The RT description is removed from time queue.

@PRIOR <prog. name> <priority>

@PRIOR KLOKK, 112 will set the priority of KLOKK to 112. This is useful to decide the urgency of the program KLOKK.

@UPDAT <minutes> <hours> <day> <month> <year>

The watch and calendar will be given new values. Otherwise, they are updated by a special clock routine in the monitor which regards, for instance, leap years.
@UPDAT 20,10,27,6,1976 will set current date to June 27, 1976 at 10:20.

@CLADJ <time> <time unit>

The watch and calendar will be set forward if the <time> is positive, backwards when negative.
@CLADJ 5,4 will advance the clock by 5 hours.

@FIX <segment number>

The segment will be fixed in core.
@FIX 35 will fix segment 35 to the memory, i.e., it is not allowed to be swapped out.

@UNFIX <segment number>

The segment will not be fixed in core.
@UNFIX 35. Segment 35 may be swapped again.

@FIXC <segment number> <first physical page>

@FIXC 40,50 will fix segment 40_8 to a continuous area in memory, starting at page 50_8, i.e., address 120000.

@PRSRV <logical unit> <read/write> <prog. name>

@PRSRV 9,1 KLOKK. The output part of device 11_8 (TTY 2) is reserved for KLOKK.

@PRLS <logical unit> < read/write>

@PRLS 9,0. The input part of device 11_8 is released from the RT program having reserved it.

@RTOFF <prog. name>

@RTOFF KLOKK. KLOKK cannot be started until an RTON is given.

@RTON <prog. name>

@RTON KLOKK. The status set by RTOFF is removed, KLOKK may be started again.

@ENTSG <segment no> <page index table> <interrupt level> <start address>

@ENTSG 42,3,9,30000B. The program on segment 42_8 will be entered to run as a direct task on page index table 3, interrupt level 9, location 030000 is the start address of the program.

5.6 *UTILITY COMMANDS*

The command

@LIST-TIME-QUEUE

will list on the terminal all programs that are in the time queue.

In the same way

@LIST-EXECUTION-QUEUE

will list on the terminal all programs in the execution queue.

In the above commands, as well as in some error messages, RT programs may be defined by an octal address instead of a symbolic name. The address is in this case the address of the RT description for this program. By the command

@GET-RT-NAME <octal address>

the supplied address will be translated to a symbolic name, if such a name exists. (In addition to some internal programs in SINTRAN there is one RT program for each terminal and batch processor. These programs have two names.)

The command

@LIST-RT-DESCRIPTION <RT name>

will cause the information contained in the RT description of that program to be listed on the terminal. The information will be:

- ring number
- priority
- when it was last started
- length of interval when periodical
- starting address
- segment numbers
- register contents
- active/passive and
- actual segments.

The command

@LIST-SEGMENT <segment number>

will give a list of the first page number. This will include:

- length
- unit
- mass storage address and
- the status of the permit protection.

6 THE USER SYSTEM

6.1 *THE PURPOSE OF THE USER SYSTEM*

In any installation used by a number of different users, some central supervision over the installation is needed. These supervisory functions are system start and stop, installing a version of the operating system or a subsystem, aborting "wild" programs, establishing new users, controlling the use of mass storage, etc.

In SINTRAN III these functions are monitored by user SYSTEM. This user has his own password and space for files and may, like any other timesharing users, run jobs using the same commands and monitor calls. User SYSTEM may log in from any terminal in the installation and may, like other users, be active from more than one terminal at the same time.

The difference compared to the other users is that the command set for user SYSTEM is a super-set of the commands available to other users. The extra commands are used for supervisory functions. To obtain easy access to the common resources (peripherals and subsystems, QED, FORTRAN) of the installation, the user SYSTEM is normally the owner of peripherals and system programs. The area of the user SYSTEM is automatically searched when another user refers to common resources (file names) not found in his own area.

Because of the powerful commands available, only a few persons using the installation should know how to log on as SYSTEM. Those persons should have a good knowledge of the installation and SINTRAN. In fact, user SYSTEM has more or less the same functions as the console operator in other computer installations.

It should be noted that when referring to the console terminal in SINTRAN III, the terminal with the lowest device number (normally Teletype 1/Display 1) is meant. This terminal is different from the others in three respects:

1. Error messages from RT programs and SINTRAN are always printed on Teletype 1. (Optionally, this may be altered to another terminal.) A list of the error messages is given in Appendix D.

2. When the computer is in STOP mode, the built in microprogram can only be reached from this terminal.

3. When the command

   @SET-UNAVAILABLE

   is given it is impossible to log in from terminals other than terminal 1. This condition may be reset by the command

   @SET-AVAILABLE

   Both commands are restricted to user SYSTEM.

6.2 *DIRECTORIES*

A complete file system may consist of one or more directories. Each mass storage device maintained by the file management system has its own directory, and is completely independent of all other devices. Devices (disk packs, magnetic tapes, etc.) may be moved to other installations and used there.

Each directory is identified by a unique name (up to 16 characters) which is given when the directory is initialized. The name and some other vital information (pointers to users registered on the directory, user files, the number of reserved and used pages, etc.) is saved on the actual device. Before the directory can be used, it must also be "entered". In the ENTER command, the device will be checked whether or not a directory with the given name is mounted on the specified unit. If it is, the directory with its files is made available for use.

The first directory entered is defined as *main directory.* This directory cannot be released if any user is logged on. All users of the file system must be users in main directory. If no directory name is specified in a command the main directory is usually assumed.

Files on an entered directory may be accessed (read, written, or created) by using the directory name as a prefix to the file name. Any number of entered directories may be set as *default directories.* Default directories are used when no name is specified in the commands for creating and accessing files. Main directory is always default directory. If a user has space in more than one default directory, the directory name must be given as part of the file name. Each user is, normally, given space in only one default directory.

6.2.1 *Initializing a Directory*

A directory is created by the command:

@CREATE-DIRECTORY <directory name>, <device name>, [<unit>], [<fixed or removable>]

Example:

To create a directory called TAPE 1 on the magnetic tape mounted on tape station 0, (numbering starts at 0) type

@CREATE-DIRECTORY TAPE1, MAG-TAPE, 0

To create a directory on the fixed cartridge on disk unit 1, type

@CREATE-DIRECTORY PACK4, DISK, 1, F

Fixed or Removable is relevant for the 9.2 M bytes cartridge disks only.

Note: If only one unit (No. 0) is installed, the unit number is neither necessary nor legal in the command.

When the command is executed, the directory name is written on the first page of the device. If the device is a disk or drum, it is tested for bad spots (tracks), and a Bit File is created. The bit file has one bit for each page on the unit. This bit is set to one if the page is occupied. On a cartridge disk, the bit file is allocated in the middle of the disk, the maximum length of a contiguous file is, thereby, half the total length of the disk.

An old directory will be completely destroyed if a new directory is created on the same device and all old files will be lost.

6.2.2 *Enter, Set-Default and Release Directory*

When a device (disk pack or magnetic tape) is mounted, it must be activated. This is done by the command:

@ENTER-DIRECTORY <directory name>, <device name> [,<unit>]
[,<'F' or 'R'>]

Example:

To enter the magnetic tape created in the previous paragraph, one types

@ENTER-DIRECTORY TAPE1, MAG-TAPE, 0

If the specified name matches the name found on the device, the directory is entered, otherwise an error message is given.

To set a directory as default directory one uses the command

@SET-DEFAULT-DIRECTORY <directory name>

If a directory is not a default directory, the directory name must be given as a prefix when accessing files on this directory. This is also true if a user has space in more than one default directory.

A directory is released by the command

@RELEASE-DIRECTORY <directory name>

A directory may only be released if no files are opened on the directory.

After the directory is released, it may be entered again, or another device may be mounted on the unit.

6.2.3 *Statistics Commands*

The command

@LIST-DIRECTORIES-ENTERED <directory name> < output file>

will list the names of entered directories and where they are mounted, e.g.

@LIST-DIRECTORIES-ENTERED P, TERMINAL

will list all directories with names beginning with P on the terminal, while,

@LIST-DIRECTORIES-ENTERED,,LINE-PRINTER

will list all entered directories on the line printer.

The command

@DIRECTORY-STATISTICS <directory name>, <output file>

will give unit number, directory status (default, main), unreserved and unused space on the specified output device.

These commands may, like all statistical commands, be given by any user, but are of special interest to user SYSTEM.

6.2.4 *Directory Back-up*

In all installations, back-ups should be taken regularly to obtain necessary security.

There are, in principle, two reasons for taking back-up:

1. All systems may fail sooner or later either in hardware or in software.

2. User errors: A user may destroy one or more of his files by giving wrong commands, e.g., a DELETE-FILE command with a wrong file name, or written onto a file instead of read from it. User SYSTEM may even give the command CREATE-DIRECTORY instead of ENTER-DIRECTORY when a device is mounted. If no back-up is kept, all files on the directory would then be lost.

There are two different ways of taking backup:

- stand alone programs, or
- commands to the file management system.

6.2.4.1 STAND-ALONE PROGRAMS

Two stand-alone programs are available:

- DIMS (Disk Maintenance System) has a copy command that may be used for copying one complete disk pack to another. The program is delivered with a users guide.

- MCOPY is a program for copying of disks and drums to and from magnetic tapes. (Example: MCOPY can copy a complete disk pack to a magnetic tape and copy it back when necessary.) The program is delivered with a users guide.

6.2.4.2 ON-LINE BACK-UP

The command

@COPY-DIRECTORY <destination directory name> <source directory name>

copies all files in the source directory onto the destination directory. The users and the file names will be the same in the destination directory as in the source directory. The destination directory should be empty when the command is given, i.e., the directory should be created, but no users or files should exist.

@COPY-DEVICE <destination device name> <source device name>

copies all pages on the source device onto the destination device. Applies to devices such as disks, drums and magnetic tapes. Destination device cannot be an entered directory.

6.2.5 *Directory Maintenance Commands*

A number of commands are available for maintenance of the file management system. These are:

@RENAME-DIRECTORY <old directory name > <new direcotry name> <device name> [<unit>] [<'F' or 'R'>]

@RENAME-USER <directory name>:] <old user name> <new user name>

@TEST-DIRECTORY <directory name>

@REGENERATE-DIRECTORY <directory name>

@DUMP-DIRECTORY-ENTRY <device name> [<unit>] [<'F' or 'R'>] <output file>

@CHANGE-DIRECTORY-ENTRY <device name> [<units>] [<'F' or 'R'>]

@DUMP-USER ENTRY <directory name>, <user number>, <output file>

@CHANGE-USER-ENTRY <directory name>, <user number>

@DUMP-OBJECT-ENTRY <user name>, <object number>, <output file>

@CHANGE-OBJECT ENTRY <user name>, <object number>

@DUMP-BIT-FILE <directory name>, <block number>, <output file>

@CHANGE-BIT-FILE <directory name>, <block number>

@DUMP-PAGE <directory name>, <page address>, <output file>

@CHANGE-PAGE <directory name>, <page address>

For further description of the file management system, refer to the manual "NORD File System" - ND-60.052.

6.3 *SUPERVISION OF OTHER USERS*

User SYSTEM is responsible for creating and deleting all users, and for the amount of space each of them may use on mass storage devices. There is no master password in the system, but user SYSTEM may clear the password for a specific user.

6.3.1 *Creating and Deleting other Users*

A new user is introduced to the system by the command

@CREATE-USER [<directory name>:] <user name>

A user must exist in all directories where he is given space. In addition, he must also exist in main directory. If the directory name is omitted in the above command, main directory is assumed.

When a user is created, he has no password.

Example:

Create a new user USER-ONE, assuming he will be allocated space in directory PACK 5, which is not main directory:

@CREATE-USER USER-ONE Now he exists in main directory

@CREATE-USER PACK5:USER-ONE Now USER-ONE exists in PACK5

A user is removed from a directory by the command

@DELETE-USER [<directory name>:] <user name>

It is not allowed to delete a user who has files in the specified directory. In this case, an error message is given but the user is not removed.

Example:

Assume user USER-ONE has created files in directory PACK5 and is moving to another installation, bringing with him the device PACK5. He may then may be deleted (if desired) in the main directory by

@DELETE-USER USER-ONE

To use his files at another installation, PACK5 must be mounted and entered there, and USER-ONE has to be created in that main directory

@ENTER-DIRECTORY PACK5, DISK, 2, R (disk unit 2)
@CREATE-USER USER-ONE

6.3.2 *Giving and Taking User Space*

A user is given space by the command

@GIVE-USER-SPACE [<directory name>:] <user name>, <number of pages>

Example:

Give user PER 100 pages (100K words) on PACK5.

@GIVE-USER-SPACE PACK5:PER, 100

The number of pages supplied is decimal and the user space is increased by that number of pages. An error message is given if there are not that many pages unreserved in the directory.

Unused pages may be taken from a user by

@TAKE-USER-SPACE [<directory name>:] <user name>, <number of pages>

Like CREATE- and DELETE-USER, GIVE- and TAKE-USER assume main directory if no directory is specified.

6.3.3 *Password*

The rules for user SYSTEM's password are the same as for any other user. In addition, SYSTEM may clear any other user password by

@CLEAR-PASSWORD <user name>

This may be used when :

1. a user has forgotten his password
2. a user owning files must be deleted

After this command the person responsible for the system (or anyone!) may log in as the specified user and delete his files.

Note: User SYSTEM must not forget his password!

6.4 *SYSTEM UTILITY COMMANDS*

6.4.1 *Terminals*

The command

@TERMINAL-STATUS <terminal number>

will list some information about the specified terminal:

STATUS:	Active, Passive (logged out), Command Mode, User Mode
USER:	The name of the user currently logged in
LAST COMMAND:	The last command given from the terminal, e.g. FTN, QED or WHO-IS-ON, etc.

The terminal number is the logical decimal device number for the terminal. (Logical device number for teletypes/displays are 1, 9, 34, 35. Refer to the Table of Logical Device numbers - Appendix C.) These numbers are the same as returned by the WHO-IS-ON command. The command may be given by all users but is of special interest to user SYSTEM.

The command

@STOP-TERMINAL <terminal number>

will log out the specified terminal.

The message ***ABORTED BY SYSTEM*** will be printed on the specified terminal.

This command is used if, for some reason, the user SYSTEM has to log out the terminal or user.

When special jobs are to be performed by the user SYSTEM (i.e., to make back-up copies of directories, to take down the operating system, etc.) the command:

@SET-UNAVAILABLE

may be given. Now, no user may log in from other terminals than from the one with device number 1. If they try, the message

SYSTEM UNAVAILABLE

is issued on that terminal. Users who have already logged in may continue their communication with the system until they log out. This situation remains until the command

@SET-AVAILABLE

is given.

6.4.2 *Stopping the Operating System*

Sometimes the system has to be stopped, for maintenance, fatal errors, or closing down for the night.

The command

@STOP-SYSTEM

will simulate a power-fail. All information contained in the CPU's registers will be saved, whereby, the CPU will go to *stop mode*.

In this case, it may be restarted by pressing the MASTER CLEAR and RESTART buttons on the operator's panel. On restart, the logged on users may continue in their programs, a start-up procedure is not necessary.

When the system is to be stopped for a longer period (for the night) the disk(s) should be stopped. The procedure is:

- Log out all users (preferably)
- Press the buttons STOP and MASTER-CLEAR on the operator's panel
- Stop the disks

6.4.3 *Restarting the System from Memory Image*

The system can be restarted from the memory image kept on mass storage by pushing the MASTER CLEAR and LOAD buttons, provided that the setting of the ALD register is correct. The same effect will occur by executing the command:

@RESTART-SYSTEM

All RT programs and segments loaded by the RT loader will remain. The files will be closed, and no directories will be entered. However, the main directory can be entered by the command @INITIAL-COMMAND (following) which has been executed earlier.

When @RESTART-SYSTEM is used, the current physical memory (first 32K) will be saved on a segment file, if such a file has been specified.

This command is an option: it can be included at system generation time.

@INITIAL-COMMAND <command string>

is used to specify an @ENTER-DIRECTORY command to be executed at subsequent restarts from core image. The command string will be saved and executed at restart time. The command RTENTER will also be executed. Further directories can be entered in a user-written subroutine called from the start-up sequence.

Example:

@INITIAL-COMMAND ENT-DIR P-ONE DISK-1 0 R

6.4.4 *The LOOK-AT Command*

As seen in Chapter 3, one can, with the help of the command

@LOOK-AT MEMORY

examine and modify locations in a users 64K address space.

The command is somewhat stronger for user SYSTEM. The exact syntax is

@LOOK-AT <area>

where <area> can be

MEMORY	the current users 64K address space
RESIDENT	the memory resident system (SINTRAN)
IMAGE	the (memory) image kept on mass storage
SEGMENT	one of the program segments
RTCOMMON	memory common area for RT programs

By this command, locations in SINTRAN may be examined or changed (patched). It is obvious that this feature should be used with care!

Remember that the file management system resides on one of the segments; it is not resident. All RT programs loaded by the RT loader are also found on segments and may be examined and modified.

(This command is also mentioned in Sections 3.3.6.2 and 5.4.)

6.4.5 *Error Print-out Device Setting*

Normally, the error messages from RT programs will be written on Terminal 1. However, it is possible to route them to some other terminal by using the command:

@SET-ERROR-DEVICE <logical number>

Example:

@SET-ERROR-DEVICE 9

which will cause the error messages to appear on Terminal 2.

6.5 *THE ACCOUNTING SYSTEM*

Whenever a user logs out, a record is written onto the account file on the disk. This file, named (SYSTEM) ACCOUNTS:DATA, consists of blocks a length of 256 words, each block containing 16 records of 16 words. A record has the following information:

Word 0-6	user name	character
7	user number	binary
8	project number	binary
9-10	log off time and data	binary
11	console seconds	binary
12	CPU seconds	binary
13	terminal number	binary
14-15	unused	binary

Log-off time and date are packed into two words, or a 32 bit field, as follows:

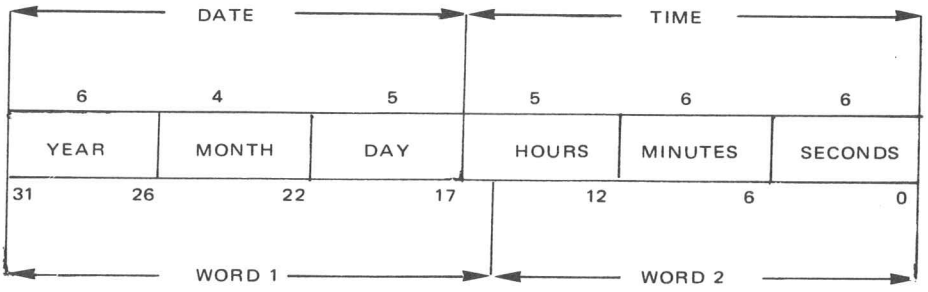


Figure 6.1: *Format of Record with Log Off Date and Time*

The first block of the file contains the following information:

1. Word 1 contains the number of records written in the file. This number is increased by 1 for every log-off.

2. Word 2 contains the desired number of records. When this number is reached, a message will be sent every time a user logs off, APPROACHING END OF ACCT FILE. The accounting file should then be listed and reset by the INIT-ACCOUNTING command.

3. Word 3 contains the maximum number of records. If this number is reached, a message will be sent every time a user logs off. END OF ACCT FILE ENCOUNTERED. No accounting is done after this number is reached.

The account records themselves start in the second block. The file is updated physically for every new record. This involves reading and writing the first block (updating the record count) and writing the block which contains the new record.

6.5.1 *Commands*

@INIT-ACCOUNTING [<desired>] [<max>]

<desired>

desired number of accounts. By use of this parameter, the user specifies the number of accounts he wants to the account file before he gets the warning that the file is running full. The parameter should be specified as a decimal number.

<max>

maximum number of accounts permitted on the file. When this limit is reached, no more accounting will take place before the account file is reset. Users can log off, but the request for accounting will be ignored. <max> should be specified as a decimal number.

This command initializes and starts the accounting system. It may only be executed by the user SYSTEM. It writes the first block of the file (SYSTEM) ACCOUNTS:DATA. Word 1 (record count) contains 0. Words 2 and 3 contain the desired number and maximum number as given in the command. If 0 or empty, default values of 500 and 600 are used.

@START-ACCOUNTING

This command starts the accounting, but does not initialize the accounting file. It may only be executed by user SYSTEM.

@STOP-ACCOUNTING

This command stops the accounting system. The accounting file is not affected. It may only be executed by user SYSTEM.

6.5.2 *List Accounts*

This is a program delivered by ND to sort and list the accounting file. It is not part of SINTRAN III.

The logged records are sorted by project number and user, and the total CPU and terminal time (console time) for each user and project is computed. After the program is loaded and saved (@PLACE-BINARY and @DUMP commands), it may be started by typing the program name. When execution is finished, the @INIT-ACCOUNTING command should be given to reset the accounting file.

The following RECOVER command then activates the program:

@LIST-ACC

The program lists/dumps the current content of the accounting file. Also, resets/initiates accounting if desired.

The program now asks for the following parameters.

OPERATOR INPUT FILE	file name (default TERM)
RESET FLAG	1 if reset accounting file
ACCOUNTING FILE	file name (default ACCOUNTS)
DESIRED & MAX NO OF ACCOUNTS: (used if reset flag, if (0,0) then default 500, 600)	no , no
DUMP-FILE	file name (no default)
LIST-FILE	file name (no default)

The following output is given by the program.

If dump file is given then the accounting file is copied while this program is running.

The listing consists of 3 tables:

1. Log Offs

   One line for each log off entered in the accounting file, with the following information:

   a) User Name and Number

b) Project Numer
c) Data, log on and log off time (YEAR, MONTH, DAY, HOUR, MIN, SEC)
d) used console and CPU times (HOURS, MIN, SEC)

2. Accounted users

Consists of one table for each user sorted by user number (ascending user number).

Each table consists of:

a) User Name and Number
b) Project Number, used console and CPU sec for each of his projects (ascending project number).
c) Total Console and CPU sec used by this user.

3. Accounted Projects

Contains Console and CPU secs used by each project (ascending project number) and total time for all projects.

6.6 *THE BATCH SYSTEM*

The batch system is described in Chapter 3. Two commands are special for user SYSTEM. They are the commands to activate or abort a batch process: @BATCH and @ABORT-BATCH.

@BATCH

This command finds an unused batch process and starts it. It then prints BATCH NUMBER = <batch number> where <batch number> is a decimal integer which may be used in future commands to identify the batch process.

If there are no unused batch processes in the system, the message NO BATCH AVAILABLE is printed.

After the batch process is started, it immediately enters waiting state as the batch queue will initially be empty. It will automatically be restarted when a batch input file – batch output file pair is entered into the batch by an @APPEND-BATCH command.

@ABORT-BATCH <batch number>

This command will abort a batch process and release all resources reserved by the batch process. Any job currently running will be aborted immediately, and the batch queue will be cleared.

6.7 *PERIPHERAL DEVICES*

Peripheral devices are supported by the file management system. User SYSTEM is responsible for introducing them to the file system. This is done by the command

@SET-PERIPHERAL-FILE <file name> <device number>

This command defines the specified file as a peripheral file. The device number is an octal value. A list of device numbers is found in the appendix.

Example:

To introduce the line printer one types

@SET-PERIPHERAL-FILE "LINE-PRINTER" 5

Legal access is specified by the command

@SET-FILE-ACCESS L-P, WA, WA, WAD

The command

@SET-TERMINAL-FILE <file name>

will specify <file name> as the name of all terminals.

Example:

After

@SET-TERMINAL-FILE "TERMINAL"

TERMINAL may be used as an output or input file name, i.e.,

@SET-FILE-ACCESS TERM, RWA, RWA, RWAD
@COPY TERMINAL, <any file>

will copy <any file> to the terminal.

6.8 *REMOTE JOB ENTRY*

6.8.1 *General Remarks*

RJE emulators are delivered for emulating IBM 2780, IBM 3780, IBM HASP WORK STATION, CDC 200 USER, UNIVAC DCT 2000 and Honeywell Bull GERTS 115. An emulator is delivered on paper tape in BRF format. The emulators are run as RT programs, and must be loaded by the RT loader. The start and stop of an RJE emulator is a SYSTEM function, although user RT normally supervises RT programs.

The BRF tape should be copied to a mass storage file. This is done by

@COPY "RJEFILE : BRF" TAPE-READER

Remember that SYMB is default for COPY.

Load procedure of an RJE emulator:

```
@RT-LOADER

RT LOADER 75.07.01

*NEW-SEGMENT XX
NEW SEGMENT NO' 37
*ALOAD RJEFILE
*END-LOAD
*EXIT-LOADER

@
```

To start the emulator one of the following commands must be given:

@RT IBM4	if IBM 2780 or 3780
@RT DCT4	if DCT 2000
@CDC4	if CDC 200 USER
@GRTS4	if GERTS 115
@RT HASP	if HASP WORK STATION

The message xxx NORD xxxx RUNNING xxx is then displayed on the RJE communication terminal (normally terminal 2, logical unit 9' see Appendix C) xxxx is the type of emulator. One may then continue on the RJE communication terminal according to the users guide for that type of emulator.

To stop an RJE terminal, one may give the command :AB on the communication terminal of the RJE emulator, or stop it by the SINTRAN command @ABORT given from another terminal. In this case all RT programs concerning RJE must be aborted.

If IBM 3780:

@ABORT IBM0, @ABORT IBM1, @ABORT IBM2, @ABORT IBM3, @ABORT IBM4.

IF DCT 2000

@ABORT DCT0,. . . . @ABORT DCT4.

A similar procedure is used for other RJE emulators.

6.8.2 *The Remote Batch Queue*

In the same way as for the local batch, it is possible to queue files containing remote batch jobs on a remote batch queue held internally in SINTRAN III.

When the command :RB is given, the emulator will start reading input file names from the batch queue. The :SB command will reset the emulator to interactive mode, where input file names are given from the communication terminal.

6.8.3 *Commands to Maintain the Remote Batch Queue*

6.8.3.1 THE APPEND-REMOTE COMMAND

This command adds a remote batch input file to the specified remote batch queue.

The format is:

@APPEND-REMOTE <host computer> <input file>

where

<host computer>

is one of the following names:

IBM
CDC
UNIVAC
HONEYWELL-BULL

<input file>

is the name of the file where batch-job-input is to be taken.

NOTES:

- If the remote-batch input file is owned by other than users SYSTEM or RT, the user-name must precede the file name.

- The remote-batch input file must have read access for user RT.

6.8.3.2 THE LIST-REMOTE-QUEUE COMMAND

This command lists the contents of a remote batch queue.

The format is:

@LIST-REMOTE-QUEUE <host computer>

where

<host computer>

is one of the following names:

IBM
CDC
UNIVAC
HONEYWELL-BULL

Example of a @LIST-REMOTE-QUEUE list:

```
@LIST-REMOTE-QUEUE IBM
1 CARD-READER
2 (USER-NAME) IBMJOB
@
```

6.8.8.3 THE DELETE-REMOTE-QUEUE-ENTRY COMMAND

This command may be used to remove an entry from the remote batch queue. That is, kill a job for execution before it is sent to the host computer.

The format is:

@DELETE-REMOTE-QUEUE-ENTRY <host computer> <queue entry>

where

<host computer>

is one of the following names:

IBM
CDC
UNIVAC
HONEYWELL-BULL

<queue entry>

is the file name given in the @APPEND-REMOTE command.

An exact match is required between the second parameter of this command and the remote batch queue entry. The exact format of the remote batch queue entry to be removed may be checked by the @LIST-REMOTE-QUEUE command.

If the remote batch queue contains two equivalent entries, the first one will be removed.

This command may be issued by user SYSTEM and the user owning the job file to be deleted.

7 REAL TIME PROGRAMS

For a description of the theory behind the implementation of real-time programs under SINTRAN III, consider Sections 4.3,4.4 and 4.5. This chapter concerns the practical rules for programming and loading of real-time programs. The rules for execution of RT programs are described in Chapter 5.

7.1 *RT PROGRAMS WRITTEN IN FORTRAN*

FORTRAN programs running as RT programs are written and compiled in the same way as in timesharing/batch mode, except that priority must be specified in the PROGRAM statement. The FORTRAN RT programs compiled in the ordinary way, will keep their data values from one execution to another.

Example:

Suppose the following program is written onto a symbolic file named KLOKK with the QED editor as follows:

```
@QED

QED 3,4
*A
            PROGRAM KLOKK, 100
            IBELL = 7*256
            IDEV = 9
            CALL    RESRV (IDEV, 0, 0)
10          IA = INCH (IDEV)
            IF (IA.NE.101B) GO TO 10
            CALL RELES (IDEV, 0)
            CALL INTV (KLOKK, 10, 2)
20          CALL RTWT
            CALL RESRV (IDEV, 1, 0)
            WRITE (IDEV, 100) IBELL
            CALL RELES (IDEV, 1)
100         FORMAT (1X,A1)
            GO TO 20
            END
            EOF
*W "KLOKK"
128 WORDS WRITTEN
*F

@
```

For compiling and loading the following commands would be necessary:

For compiling:

```
@FTN

NORD FTN
$CLC 0
$REFMAP
$COM KLOKK, LINE-PRINTER, "KLOKK"
16 STATEMENTS COMPILED
$EX
@
```

For loading: (The RT Loader commands are described in Section 7.7.3.)

```
@RT-LOADER
RT-LOADER 75.07.01

*NREENTRANT-LOAD  KLOKK,,
NEW SEGMENT  NO 30
*END-LOAD
*EXIT-LOADER

@
```

The RT program KLOKK may be started with the command @RT KLOKK, and then the KLOKK program will reserve the input part of logical unit number 9 (usually terminal 2) and read characters from that unit until an A is read. The input part will then be released. The program will make itself periodical with an interval of 10 seconds and then terminate with a call to RTWT. The KLOKK program may be started once more with the command @RT KLOKK. It will then reserve the output part of the terminal, give the character BELL on the logical unit 9 and release it again every 10th second, until the KLOKK RT program is made non-periodical by the command @ABORT KLOKK or by the command @DSCNT KLOKK.

7.2 *REENTRANT FORTRAN PROGRAMS*

The FORTRAN compiler will produce the reentrant code, so that different RT programs may use the same subroutines when the command $RT is used. In reentrant FORTRAN, all local data is dynamically written into a data stack by the FORTRAN run-time system. FORTRAN programs compiled in $RT mode must use a special version of the FORTRAN run-time system, usually (and hereafter) called FTNRTLIBR. In this version of the FORTRAN run-time system, most of the routines are reentrant. The number of stack locations needed by each subroutine is printed on the FORTRAN program listing when using the command $REFMAP. Additionally, if the program contains READ/WRITE or other I/O statements, *the FORTRAN I/O system needs 264_8 stack locations.*

The stack area needed by one *instance* of a subroutine will be released when that instance is terminated with a RETURN/END statement. The maximum number of stack locations needed by the subprogram is the number of stack locations needed by one instance, multiplied by the number of possible instances to be nested.

The beginning of the data stack will be determined when loading the FORTRAN run-time system FTNRTLIBR and the length of the stack must be *defined later in the load procedure.* Default stack size is 1K words.

In reentrant FORTRAN, *each RT program must have its own data stack* and its *own set of the non-reentrant FORTRAN run-time system routines.*

If, in reentrant FORTRAN, more than one RT program is loaded to the same segment, the entry points of the non-reentrant FORTRAN run-time system routines, including the stack routines, must be deleted after each loading of the FORTRAN run-time system and stack definitions. This must be done because each RT program must have its own version of the non-reentrant run-time routines, and it is done by using the RT loader command DELETE-NREENTRANT. It is done automatically after the command REENTRANT-LOAD.

Example:

Suppose the following three programs, TERM1, TERM2, and TERM3 and the subroutine SUBR are written onto four different files correspondingly named with the QED editor:

```
PROGRAM TERM 1, 50
IDEV = 9
CALL SUBR (IDEV)
END
EOF
```

```
      PROGRAM TERM2, 50
      IDEV = 34
      CALL SUBR (IDEV)
      END
      EOF

      PROGRAM TERM3, 50
      IDEV = 35
      CALL SUBR (IDEV)
      END
      EOF

      SUBROUTINE SUBR (IDEV)
      CALL RESRV (IDEV, 1,0)
      WRITE (IDEV, 10) IDEV
10    FORMAT (1X, 'THIS IS LOGICAL UNIT NO:', I3)
      RETURN
      END
      EOF
```

The programs should be compiled and loaded in the usual manner.

The compiling is done as follows:

```
@FTN

NORD FTN
$CLC0
$REFMAP
$RT
$COM TERM1, LINE-PRINTER, "TERM1"
5 STATEMENTS COMPILED
$CLC 0
$COM TERM2, LINE-PRINTER, "TERM2"
5 STATEMENTS COMPILED
$CLC 0
$COM TERM3, LINE-PRINTER, "TERM3"
5 STATEMENTS COMPILED
$CLC 0
$COM SUBR, LINE-PRINTER, "SUBR"
7 STATEMENTS COMPILED
$EX
@
```

The loading is done as follows: (See Section 7.7.3.)

```
@RT-LOADER

RT-LOADER 76.03.04

*NEW-SEGMENT      ,,,
NEW SEGMENT NO: . . . . 30
*SET-LOAD-ADDRESS 30 20000
*REENTRANT-LOAD SUBR,,,
*END-LOAD
*REENTRANT-LOAD TERM1,30,,
NEW SEGMENT: 31
*END-LOAD
*REENTRANT-LOAD TERM2,30,,
NEW SEGMENT: 32
*END-LOAD
*REENTRANT-LOAD TERM3,30,,
NEW SEGMENT: 33
*END-LOAD
*EXIT-LOADER

@
```

The subroutine SUBR is loaded to segment number 30 together with the necessary FORTRAN run-time system routines, and the program TERM1 is loaded to segment 31 together with the necessary FORTRAN run-time system routines, including the data stack, and segment 31 is linked to segment 30 to get hold of the subroutine SUBR. In the same way, TERM2 is loaded to segment number 32 and TERM3 to segment number 33. The default stack size is 1K.

The RT programs, TERM1, TERM2, and TERM3 may then be started with the @RT command. The RT program TERM1 will write a message on logical unit number 9, the RT program TERM2 will write a corresponding message on logical unit number 31, and the RT program TERM3 will write the message on logical unit number 35.

The three RT programs and the common subroutine SUBR may be loaded to the same segment in the following way:

```
@RT-LOADER

RT-LOADER 75.07.01

*REENTRANT-LOAD SUBR,,,
NEW SEGMENT NO: 34
*REENTRANT-LOAD TERM1,,,
*REENTRANT-LOAD TERM2,,,
*REENTRANT-LOAD TERM3,,,
*END-LOAD
*EXIT-LOADER

@
```

A main program itself is not reentrant, because only one instance of the main program is executed at a time. It can only call reentrant subprograms. However, in a reentrant program structure, both the main program as well as the subprograms must be compiled in $RT mode. The stack could be considered to belong to the main program, i.e., all the reentrant subprograms will have their local variables dynamically allocated in the same stack. Thus, if a main program calls the reentrant subroutine A, and later calls subprogram B, then subroutine B might occasionally get its local data placed in the same area where A formerly had its local variables. For this reason, reentrant subprograms will not retain the values of their local data from one execution to another.

If, on the other hand, the main program calls subprogram A, which calls subprogram B (all reentrant), then their local data will be placed after each other in the stack. The stack length should then be specified accordingly, in the loading procedure.

Because of the stack mechanism, recursive subprograms may be introduced under SINTRAN III. Consider the following example for calculating N ! = N * (N - 1) * --- 2 * 1 where by definition 0 ! = 1.

Example:

```
      PROGRAM NFAC,30
      COMMON IDEV
      DOUBLE INTEGER NF,FACU
      IDEV = 39
      CALL RESRV (IDEV,0,0)
      READ (IDEV,1) N
1     FORMAT (I1)
      CALL RELES (IDEV,0)
      NF = FACU (N)
      CALL RESRV (IDEV,1,0)
      WRITE (IDEV,2) N,NF
2     FORMAT (I2,*! =*,I7)
      CALL RELES (IDEV,1)
      CALL RTEXT
      END
      INTEGER FUNCTION FACU (K)
      COMMON IDEV
      DOUBLE INTEGER FACU,KK
      IF (K .EQ. 0) GO TO 10
      KK = K*FACU (K-1)
      CALL RESRV (IDEV,1,0)
      WRITE (IDEV,3) K,KK
3     FORMAT (I2, I10)
      CALL RELES (IDEV,1)
      FACU = KK
      RETURN
```

```
10    FACU = 1
      RETURN
      END
      EOF
```

Suppose that the symbolic program lies on the file NFAC.

The compiling should be done as follows:

```
@FTN

NORD FTN
$RT
$REFMAP
$COM NFAC, L-P, "NFAC"
30 STATEMENTS COMPILED
$EX
```

The loading should be done as follows:

```
@RT-L

RT-LOADER  76.03.24

*REENTRANT-LOAD NFAC,,,
*END-LOAD
*EXIT-LOAD
```

Now the program may be started by the command @RT NFAC. If the programmer has logged in on terminal 39 he should now log out, so that the communication with the real-time program NFAC may start. If, for instance, 9 is given as input to the read statement in the main program, the effect of the interesting statement KK = K * FACU (K-1) is clearly demonstrated by the table output on the terminal. The first instance of the integer function FACU to reserve the terminal and to give an output is the second to the last one being called.

The local variable KK will occupy a new location in the stack for each level being nested, and will get the different values on each level.

The COMMON variable IDEV is not placed in the stack and need not to get its value initiated each time the integer function is called.

7.2.1 *Summary of Reentrant FORTRAN Programs*

RT programs in reentrant FORTRAN will not remember their local data values from one run to another. Only data in COMMON will be kept from one run to the next execution.

DATA statement for local variables (not COMMON variables) is not allowed.

DATA statement for common variables is legal.

Each RT program in reentrant FORTRAN must have its own data stack, the length of the data stack is decided when loading the RT program to a segment by means of the RT-LOADER.

Each RT program in reentrant FORTRAN must have its own copy of the non-reentrant FORTRAN run-time system routines.

Reentrant FORTRAN programs/routines should only use the special version of the FORTRAN run-time system, FTNRTLIBR.

Ordinary FORTRAN programs/routines which are not compiled in reentrant ($RT) mode must only use the ordinary version of the FORTRAN run-time system FTNLIBR.

All reentrant FORTRAN routines used by FORTRAN RT programs must be compiled in $RT mode by the FORTRAN compiler. Mixed routines, i.e., some routines compiled in the ordinary way (not $RT mode) and some routines compiled in $RT mode, are not allowed to be used by the same RT program.

7.3 *REENTRANT MAC/NORD-PL SUBROUTINES CALLABLE FROM REENTRANT FORTRAN PROGRAMS*

The easiest way to make a MAC/NORD-PL subroutine used by reentrant FORTRAN, is to call the subroutine with a local array as a parameter, and then use this array for all local data in the subroutine. This array will be put into the stack by the FORTRAN run-time system, i.e., each main RT program will have its own array for the MAC/NPL subroutines local data.

Example:

```
      PROGRAM PROGR,20
      DIMENSION ICHAR (80), LOCDAT(25)
      .....

      C CALL TO MAC/NPL SUBROUTINE FOR READING NCHAR
      C NUMBER OF CHARACTERS FROM THE TERMINAL WITH
      C THE LOGICAL UNIT NUMBER IDEV.
      C THE CHARACTERS WILL BE STORED IN THE ARRAY
      C ICHAR
            CALL RCHAR (IDEV, NCHAR, ICHAR, LOCDAT)
            .....
            .....
            END
            EOF
      % MAC SUBROUTINE TO READ CHARACTERS FROM
      % A TERMINAL. COUNTS THE NUMBER OF CHARACTERS,
      % AND STORES THE CHARACTERS IN THE ARRAY ICHAR.
      % THIS SUBROUTINE WILL USE THE ARRAY LOCDAT AS
      % ITS LOCAL VARIABLES.
      )9BEG
      )9ENT RCHAR
      SVB = 0
      SVL = 1
      COUNT = 2

RCHAR, SWAP    SA DB
       LDX     3, B        % POINTS TO THE ARRAY LOCDAT
       STA     SVB, X      % SAVE B REG OF THE FTN PROG
       COPY    SL DA
       STA     SVL, X      % SAVE RETURN ADDRESS
       STZ     COUNT, X    % INIT. CHAR. COUNTER
```

```
LOOP,   LDT    I   0, B        % LOGICAL UNIT NO IN T REG.
        MON        1           % READ ONE CHAR
        MON        65          % ERROR RETURN
        BSET       ZRO 70 DA   % MASK OUT PARITY BIT
        AAA        -15         % CHECK FOR CARRIAGE RETURN
        JAZ        OUT
        AAA        15          % CHAR IS NOT CARRIAGE RETURN
        JPL        STORE       % STORE CHAR IN ICHAR
        JMP        LOOP        % READ NEXT CHAR
STORE,  LDT        2,B         % T POINTS TO ARRAY ICHAR
        COPY       SX DD       % SAVE X IN D
        LDX        COUNT, X    % CHARACTER COUNTER IN X
        SBYT                   % STORE BYTE
        COPY       SD DX       % RESTORE X
        MIN        COUNT, X    % INCREMENT CHAR. COUNTER

        EXIT
OUT,    LDA        COUNT, X    % NO OF CHARACTERS READ
        STA    I   I,B         % STORE INTO NCHAR
        LDA        SVB, X      % LOAD OLD CONTENTS OF B
        COPY       SA DB       % RELOAD B REGISTER
        LDA        SVL, X      % RETURN ADDRESS
        COPY       SA DP       % RETURN TO FORTRAN PROGRAM
)FILL
)9END
)9EOF
)LINE
```

Note that in this example the MAC program has no local data locations.

7.4 *RT PROGRAMS IN MAC AND NORD-PL*

The)9RT command (statement) will declare a MAC/NORD-PL program as an RT program, i.e., the)9RT command in MAC assembly code will give the same information to the RT loader at load time as the PROGRAM statement in FORTRAN. The)9RT command (statement) must have two symbolic arguments, program name and program priority.

Example:

```
)9BEG
PRIOR = 15
)9RT MACPR PRIOR

PROGA, MON 134        % RTEXT, Terminate program

)9END
)9EOF
)LINE
```

In this example, the name of the RT program will be MACPR, the priority will be PRIOR (15_8) and the RT program start address PROGA (first instruction to be executed) will be equal to the load address, when the)9RT information is read by the RT loader.

The)9RT command must be placed directly before the statement in which the program is to be started, and may be preceded by other commands and instructions. The label PROGA itself is uninteresting in this example.

The program consists of one machine instruction and will only execute the monitor call RTEXT to terminate itself.

7.5 *COMMUNICATION BETWEEN RT PROGRAMS*

There are several different ways of communication (exchanging information) between RT programs.

1. Memory common may be used as a common data area.

2. RT programs residing on different segments may have a common area on another segment as their common data area.

3. Several RT programs may reside on the same segment and have their common data area on the same segment.

4. Internal devices may be used for communication between RT program, i.e., the RT programs giving information may reserve the internal device for output, and then "write" on the internal device. The RT program receiving information may reserve the internal device for input, and then "read" the information from the internal device.

5. If there is much data to exchange between RT programs and the exchanging time is not critical, files are useful for exchanging data.

6. Instead of using files, the RT programs may utilize several segments as common data area. This is a faster way of exchanging data than using ordinary files, because it will use the segment administration instead of the file management system.

Using this method for common data area, each RT program accessing this data area will use one segment for program and then use the other data segments (only one at the same time) as its second segment. Generally, all the data segments have the same size, they may be looked upon as if each of them were a record in a file. The monitor call MEXIT may be used to exchange data segments (records).

Example:

```
      PROGRAM PROGR, 50
      COMMON/IREC/INAM (10), !ADR (20), !DATE (6)
      .....
C
C FETCH THE ACTUAL SEGMENT (RECORD) IRECNO
C
      CALL MEXIT (IRECNO)
      .....
      .....
      END
      EOF
```

7.5.1 *Defining a COMMON Area in a MAC or NORD-PL Program*

The)9ASF command (statement) is used to define a labelled COMMON area in a MAC or NPL program. The)9ASF command (statement) must have two symbolic arguments, the first is the COMMON label and the second is the size of the COMMON area.

Example:

```
)9BEG
)9ENT DEFCO
CSIZE = 1000
)9ASF CLABL CSIZE   % DEFINE THE COMMON AREA
                    % CLABL WITH THE LENGTH
)9END               % CSIZE (= 1000_8)
)9EOF
)LINE
```

7.5.2 *Accessing a COMMON Area from a MAC or NORD-PL Program*

The)9ADS command (statement) is used to make the address of a COMMON area known in a MAC/NPL program. The)9ADS command (statement) has two symbolic arguments, the first is the COMMON label name and the second is the displacement relative to the start of the actual common area.

Example:

```
      PROGRAM COMM1,20
      COMMON/LABL1/IADR (100), XY
      COMMON/LABL2/IADR2 (10), I
      COMMON/LABL3/ADR (20), FILNR
      .....
      .....
      END
      EOF
% ASSEMBLY ROUTINE TO MOVE THE CONTENT OF THE
% ARRAY IADR2 IN THE COMMON BLOCK LABL2 TO
% THE LOCAL
 % ARRAY
% ARR IN THIS SUBROUTINE.
      )9BEG
      )9ENT     ACOMM
      DISP = 0
ACOMM,LDX  COADR      % POINTS TO START OF
                      LABL2 (IADR2)
      COPY     SX DT
      AAT      12     % T POINTS TO END OF IARR2
      LDA      (ARR
      COPY     SA DB  % B POINTS TO START OF ARR
```

```
% MOVE ELEMENT FROM IARR2 IN COMMON AREA
% LAB2 TO THE ARRAY ARR
LOOP, LDA       ,X          %  FETCH  ELEMENT  FROM
                            % IADR2
      STA       ,B          % STORE ELEMENT IN ARR
      AAX       1           % INCREMENT ADDRESS
                            % POINTER TO IADR2
      SKP       SX LST DT
      MON       134         %  THE  MOVING  OF  THE
                            % ELEMENTS IS FINISHED.
                            %  TERMINATE PROGRAM
      AAB       1           % INCREMENT ADDRESS
                            % POINTER TO ARR
      JMP       LOOP
COARD, )9ADS    LABL2DISP% ADDRESS OF IADR2(1) IN
                            % COMMON BLOCK LABL2
ARR   ,0                    % THE ARRAY NAMED ARR
*+ 12                       % STARTS HERE
)FILL
)9END
)9EOF
)LINE
```

7.6 *MONITOR CALLS AVAILABLE FROM RT PROGRAMS ONLY*

Several monitor calls may be called only by programs running as real-time programs. Such monitor calls are described in this section.

RT programs may also execute some monitor calls that are available from timesharing and batch programs. Those monitor calls are described in Chapter 3.

The <prog. name> parameter used in the following monitor calls should be an RT program name. If the calling RT program itself is wanted as the <prog. name> parameter, the parameter is set equal to 0 (zero). All RT program names used in FORTRAN programs must be declared as external by the EXTERNAL statement. Otherwise, the RT program names will be established as local variables in the FORTRAN program.

7.6.1 *Subroutines for Executing Monitor Calls from FORTRAN RT Programs*

CALL RT (<prog. name>)

The RT program specified by the parameter will enter the execution queue immediately (independent of any clock interrupt).

Example:

```
CALL RT (PR1)
```

CALL SET (<prog. name>, <time>, <time unit>)

The RT program given by <prog. name> will enter the time queue. The parameter <time unit> may have the values 1, 2, 3, 4:

1. basic time units (normally 20 milliseconds)
2. seconds
3. minutes
4. hours

Other values will give an error message and the calling program is aborted. The parameter <time> gives the number of time units the program has to stay in the time queue before it is transferred to the execution queue. If the number is ≤ 0, the RT program will be transferred from the time queue to the execution queue the first time the basic time unit counter in the monitor is incremented.

These two parameters <time> and <time unit> are also used in the monitor calls INTV, HOLD and CLADJ.

If the RT program is already in the time queue, it will be removed from the queue before it is inserted again with the new specifications.

Example:

CALL SET (RT1, 10, 2)

The RT program RT1 will be scheduled for execution in 10 seconds, reckoned from the moment SET is executed.

CALL ABSET (<prog. name>, <second>, <minute>, <hour>)

The RT program given by <prog. name> will enter the time queue. The last three parameters give the time of day for execution. If the time has expired at the moment ABSET is called, the program will be scheduled the next day at the time specified.

If the RT program is already in the time queue, it will be removed from this queue before being inserted, according to the present ABSET parameters.

If a time parameter has an illegal value, an error message is given and the calling program is aborted.

If the clock is adjusted by means of a call of CLADJ while the program is in the time queue, the execution time is modified to fit the new clock setting.

Example:

CALL ABSET (PROG, 0, 30, 17)

The RT program PROG will be scheduled for execution at 17:30.

CALL INTV (<prog. name>, <time>, <time unit>)

The RT program given by <prog. name> will be prepared for periodic execution. The last two parameters give the time between each execution. However, the first execution must be initiated by other means, for instance, by the CALL RT or CALL SET.

The periodic execution property set by INTV will be reset by a call of DSCNT or ABORT (see CALL DSCNT, CALL ABORT following), thus stopping a series of periodic executions.

The interval may be modified by another call of INTV without an intervening DSCNT or ABORT.

If the starting of periodic RT programs are delayed because of other RT programs with higher priority, the delays will not be accumulated. Thus, synchronism is preserved. However, if the start is delayed until the time for the next start, the program is scheduled for an immediate repetition after the first execution. Only one repetition is allowed, so that if the start is delayed with two intervals or more, at least one execution is lost. Illegal values of <time> and <time unit> will cause an error message and the calling program will be aborted.

Example:

```
      C THE PROGRAM PP IS TO RUN EACH 20 MINUTE
      CALL INTV (PP,20,3)

C FIRST EXECUTION STARTS 6 MINUTES FROM NOW ON:
      CALL SET (PP, 6,3)
```

CALL DSET (<prog. name>, <time>)

The RT program given by <prog. name> will enter the time queue. The parameter <time> is a double precision number of basic time units giving the time the program has to stay in the time queue, reckoned from the moment DSET is called.

Example:

```
CALL DSET (RTP, TM1)
```

CALL DABST (<prog. name>, <time>)

The RT program given by <prog. name> will enter the time queue. The parameter <time> is a double precision number of basic time units giving the absolute point of time when the program is to leave the time queue, and enter the execution queue.

Example:

```
CALL DABST (RTIMP, TM2)
```

CALL DINTV (<prog. name>, <time>)

The RT program given by <prog. name> will be prepared for periodic execution. The parameter <time> is a double precision number of basic time units giving the time between each execution. See the description of the monitor call INTV.

Example:

```
CALL DINTV (RTPR, DTIM1)
```

CALL RTWT

The calling program will be set in a waiting mode. Its resources will not be released. The next time the program is started, for instance, by a call of RT from some other program, it will continue with the statement after the call of RTWT.

Example:

```
      CALL RTWT
C
C THE PROGRAM WILL START AT THIS POINT
C THE NEXT TIME IT WILL BE STARTED
C
```

CALL HOLD (<time>, <time unit>)

The calling program will be in a waiting state for the time given as parameters:

<time units>:

1. basic units
2. seconds
3. minutes
4. hours

Example:

```
CALL HOLD (10, 2)
```

The calling program will wait 10 seconds before the execution will continue in the next instruction. Resources will not be released.

CALL ABORT (<prog. name>)

The specified program will be aborted if it is running. If the RT program is in time queue or execution queue, it will be removed from those queues. All reserved resources will be released and periodical execution set by INTV or DINTV will be reset. Connections established by CONCT calls are not released.

Example:

CALL ABORT (PRX)

CALL CONCT (<prog. name>, <logical units>)

The RT program given by the first parameter will be connected to an interrupt line, i.e., the program will be inserted into the execution queue each time an interrupt signal occurs on that logical unit.

The logical unit numbers are determined at system generation time belonging to the I/O system. Several units may be connected to one program. Illegal numbers cause the calling program to be aborted, and an error message will be given. Only units with a "connect" driver routine are legal.

Example:

CALL CONCT (CPIN, 410B)

CALL DSCNT (<prog. name>)

Any connection established by CONCT will be removed. If the program has been made periodical by INTV or DINTV, this will be reset. If the program is in time queue it will be removed from that queue. Reserved resources are not released. The execution of the calling program continues.

Example:

CALL DSCNT (PRGA)

IP = PRIOR (<prog. name>, <priority>)

The RT program given by <prog.name> will have its priority permanently changed. The parameter <priority> keeps the new priority value and may range from 1 (lowest) to 255 decimal (highest). Programs with priority zero will never be started. The old priority will be returned as in integer function value.

Example:

CALL PRIOR (RTPR, 30)

CALL UPDAT (<minutes>, <hours>, <days>, <months>, <years>)

The clock and calendar will get new values. The internal time representation and time queue will not be affected. If a parameter is specified outside its range (e.g., minute > 60), an error message is given, and the calling program is aborted. For <years> a value < 1976 is illegal.

Example:

CALL UPDAT (24, 11, 24, 2, 1976)

This will set current time to February 24, 1976 at 11:24 o'clock.

CALL CLADJ (<time>, <time units>)
(Clock Adjust)

The parameter <time unit> may have the values 1, 2, 3 or 4:

1. basic time units
2. seconds
3. minutes
4. hours

Other values will give an error message, and the calling program will be aborted. The parameter <time> gives the number of time units the clock/calendar will be decremented or incremented. If the parameter <time> is negative the clock will stand still for the proper number of <time units>.

If there are any programs in the time queue inserted by ABSET, these will have their start time and queue position adjusted to fit the next clock setting. This concerns also periodic executions, if the first start was specified by means of ABSET.

Example:

CALL CLADJ (15, 2)

The clock/calendar will be advanced by 15 seconds.

CALL FIX (<segment number>)

This monitor call is used to make a segment temporarily memory resident. The segment, which must be of non-demand type, will be brought into memory. Then it will be flagged in the segment table, so that it will not be swapped out.

If <segment number> refers to a non-existent or demand segment, an error message will be given, and the calling program will be aborted. Only a limited amount of physical memory may be used for fixed segments at a time. This amount will be specified at system generation time, or it may be changed by the user before the SINTRAN III operating system is started.

Example:

CALL FIX (35)

Segment number 35 will be fixed (not allowed to be swapped to mass storage) in memory and will not be swapped out from memory before it is unfixed again.

CALL UNFIX (<segment number>)

If the segment <segment number> has been fixed in memory by means of the FIX monitor call, this condition will be removed, so that the segment can be swapped back to mass storage.

Example:

CALL UNFIX (35)

Segment number 35 may be swapped onto mass storage again.

<value> = RESRV (<logical unit>, <read/write>, <return flag>)

This routine is used to reserve a logical unit. If <read/write> equals zero, the input part is reserved for a two-way unit, or if it equals one, it means the output part. If the unit is already reserved, the program will be set in a waiting state if <return flag> equals zero. If the unit is reserved and the <return flag> is set non-zero there will be an immediate return with negative function value. If the unit is free, there will be immediate return with zero function value, and the logical unit will be reserved. RESRV must be declared as INTEGER (not INTEGER FUNCTION) when testing on the function value <value>.

Example:

IVALUE = RESRV (5, 1, 0) or
CALL RESERV (5, 1, 0)

This means: reserve the logical unit number 5 (usually the Line Printer) for the calling program. If the logical unit is already reserved, the calling program will be set into a waiting state and will not be started again before the logical unit is free to use for the calling program.

CALL RELES (<logical unit>, <read/write>)

The reserved unit will be released if it is reserved for the calling program. If <read/write> equals zero, the input part is reserved for a two-way unit, or if it equals one, it means the output part.

If RELES is not called, the unit will be released when the RT program is terminated.

Example:

CALL RELES (LUN, 0)

The logical unit number LUN will be released from the calling program if this unit is reserved for this program.

<value> = PRSRV (<logical unit>, <read/write>, <prog. name>)

The logical unit will be reserved for the RT program specified by the parameter <prog. name>. A two-way unit <read/write> equal to zero means that the input part will be reserved, otherwise, that the output part is reserved. If the unit is already reserved, a negative function value is returned. If not, zero is returned and the reservation will be performed. PRSRV must be declared as INTEGER (not INTEGER FUNCTION) when testing the function value <value>.

Example:

IVALUE = PRSRV (LUN, 0, RTPROG)
CALL PRSRV (LUN, 0, RTPROG)

This means: if the logical unit LUN is free to use, it will be reserved for the RT program RTPROG and the function value PRSRV will be set to zero. If the logical unit LUN is already reserved for another RT program, the function value PRSRV will be negative and no reservation will be performed.

CALL PRLS (<logical unit>, <read/write>)

This unit will be released from the program having reserved it.

Example:

CALL PRLS (5, 0)

The logical unit number 5 will be released from the RT program having reserved it.

Note: This call should not be used for taking a terminal from another user!

<value> = WHDEV (<logical unit>, <read/write>)

If the logical unit is reserved for some program, the address of the RT description of the program occupying the unit will be returned as the function value. If the unit is free, zero will be returned. WHDEV must be declared as INTEGER (nor INTEGER FUNCTION) when testing on the function value <value>.

Example:

IVALUE = WHDEV (LUN, 0)

This means: the address of the RT description for the RT program having reserved the logical unit LUN will be returned as the function value WHDEV. If the logical unit LUN is free to use, the function value WHDEV will be set to zero.

Note: The name of this routine was changed in January 1976. The former name was WHERE which is also the name of a plotting function.

CALL RTEXT

The calling RT program will be terminated and all its reserved resources will be restarted.

Example:

CALL RTEXT

Otherwise, the following statements may be used as a return from a FORTRAN RT program to the monitor:

END - Statement

Control will be given to the monitor, which will release the reserved resources of the program.

Example:

```
PROGRAM TCOM, 30
CALL SUBR (3)
END
```

CALL MEXIT (<segment no>)

MEXIT is used from FORTRAN when an RT program wants to load a new segment instead of its second current segment. Only the second current segment may be exchanged with the segment number used as parameter in MEXIT call.

Example:

```
CALL MEXIT (37)
```

This means: the calling RT program will have its second current segment exchanged with segment number 37.

CALL RTOFF (<program name>)

The starting of a program is prohibited. After this call the RT program is put into RTOFF status and will not be allowed to be started before the RTOFF state is removed by the RTON call (following).

Example:

```
CALL RTOFF (RTPRG)
```

This means: the RT program RTPRG cannot be started (by RT, SET, external interrupt, etc.) before the RTOFF state is removed.

CALL RTON (<program name>)

RTON is used to remove the RTOFF status of an RT program, i.e., the RT program may be started (by SET, RT, external interrupt, etc.) and executed after a RTON call.

Example:

```
CALL RTON (RTPR1)
```

The PR program RTPR1 is now allowed to be executed.

CALL ERRMON (<error number>, <suberror number>)

The ERRMON monitor call will write an error message on terminal 1. The error number must be in the interval 50 - 69 while the suberror may be any integer number.

The error message will be written as follows:

aa.bb.cc ERROR nn IN rr AT ll ss: USER ERROR.

The symbols have the following meaning:

aa.bb.cc	time when the error message was printed
aa	hours
bb	minutes
cc	seconds
nn	<error number>.
rr	octal address corresponding to an RT program name. (Address of an RT description).
ll	octal address (virtual address) where the error occurred.
ss	<suberror number>.

The calling RT program will not be aborted when using ERRMON.

Example:

CALL ERRMON (50, 100)

This means that the following error message will be printed on terminal 1:

15.45.02 ERROR 50 IN 20762 AT 572 100: USER ERROR.

<value> = ABSTR (<logical unit no> <function code> <memory address> <block address> <number of blocks>)

This is a monitor call for data channel transfers between physical core and a mass storage device. The monitor call and parameters must be in permanent core or on a fixed segment, residing on protect ring 2 and page index table 0.

The memory address to transfer data to/from, must be a continuous area in physical memory. The area is not allowed to cross a 64K memory bank boundary. <value> will be set negative if an error has been detected in ABSTR else <value> is set positive. ABSTR must be defined as INTEGER (not INTEGER FUNCTION) when testing on the function value <value>.

Parameters in the ABSTR monitor call:

<logical unit no>

the logical unit number for the actual mass storage device. See the list in Appendix C.

<function code>

the function code parameter specifies the operation to be performed on the mass storage device.

Bits 0 -5:
0 - read
1 - write
2 - read test
3 - compare
20 - read status

Bits 6 -7: Drive number

<memory address>

double precision physical memory address (not virtual) to transfer data to/from.

<block address>

mass storage address to transfer data to/from. For magnetic tape, this parameter holds the unit number. For cartridge disks bit 15 = 1 means the fixed pack, bit 15 = 0 means the removable pack.

<number of blocks>

number of "hardware blocks" to transfer in the call. Refer to separate manuals for information about "hardware block" size (on disk: sector, on magnetic tape: record). For magnetic tape, this parameter contains the record size (number of words). For read operations, the value will be changed to actual record size if the record is less than specified.

Example:

IX = ABSTR (502B, 0, MEMO, 20000B, 3)

This means: read 600_8 words from disk unit 0 to the physical memory address MEMO (MEMO must be double precision). The data will be read from sector address 20000_8.

Note: In a previous version, an additional parameter was used to keep the unit number.

CALL ENTSG (<segment no>, <page index table no>, <interrupt level>, <start address>)

This is a monitor call to enter a Direct Task or a driver routine into a SINTRAN III system. Parameters in the ENTSG monitor call are:

<segment no>

the segment number where the Direct Task or driver routine is loaded into.

<page index table no>

specifies which page index table the routine will use.

<interrupt level>

specifies on which interrupt level the routine will run. If it is a driver routine on level 10, 11, 12 or 13, to enter into the system one of the free interrupt levels must be specified instead of the actual level, i.e., interrupt level 10, 11, 12 or 13 are not allowed to be used as the parameter <interrupt level>.

<start address>

logical start address, entry point of the Direct Task routine.

Example:

CALL ENTSG (36, 3, 6, 50000B)

This means: the Direct Task routine loaded into segment number 36 will be entered into the system. This Direct Task will use page index table number 3 and it will run on interrupt level 6. The start address of the routine is 50000_8.

CALL FIXC (<segment number>, <first physical page>)

This monitor call is used to make a segment memory resident, similar to the FIX monitor call. The difference is that when using FIXC, the segment will be placed in a contiguous area of physical memory. The parameter <first physical page> determines where it is to be placed in physical memory.

Example:

CALL FIXC (77, 48)

The segment number 77 will be fixed in memory (not allowed to be swapped to mass storage), and it will be placed from address 140000_8 in physical memory.

7.6.2 *Monitor Calls Available from MAC/NORD-PL Only*

For an extensive list of the correspondance between the monitor call names and number, see Appendix B.

MCALL (MON 132)

This monitor call is used when a subroutine on a different segment is wanted.

The T register contains a pointer to a data element of two locations, holding the address of the subroutine. The first location holds the address, and the second holds the new segment numbers, one in each half-word. The word containing the two bytes ACTSEG1 and ACTSEG2, in the RT description will receive the new segment numbers. If a segment number is zero, only the other segment is wanted. This may be utilized to inform the operating system, explicitly, that the segment which is now replaced by a zero, may be swapped out completely. If that segment is of non-demand type, the corresponding memory space will be released for other programs. If a segment number is 277_8, the corresponding segment will be the same as in the calling program.

A call of MCALL will cause the new segments to be fetched and the subroutine will be started. The L register will then hold the return address and the T register contains the segment numbers of the calling program. Return from the subroutine will be performed by the monitor call MEXIT (see following).

MEXIT (used together with MCALL) (MON 133)

This monitor call will cause a return from the subroutine.

The T and L registers must have the same values as they had after the corresponding MCALL. Then the old segments will be used and the calling program will be resumed.

Example:

```
        LDT     PARLI       % T POINTS TO DATA ELEMENT
        MON     132         % MCALL
        .                   % RETURN HERE AFTER MEXIT
PARLI,  SUBR                % START ADDRESS AFTER MCALL
        10030               % GET IN SEGMENT NO 20₈ & 30₈
SUBR,   STT     SAVT        % ENTRY POINT AFTER MCALL
        COPY    SL    DT    % SAVE T AND L REGISTERS
        STT     SAVL
        .
        .
        LDA     SAVL
        COPY    SA DL       % L AND T REGISTERS HAVE THE
                            % SAVE VALUE
        LDT     SAVT        % AS AFTER MCALL
        MON     133         % MEXIT
SAVT,   0
SAVL,   0
```

7.6.3 *The Difference Between Using Some Monitor Calls From MAC/NORD-PL and From FORTRAN*

ABSTR

The only difference between using the monitor call ABSTR from MAC/NPL and from FORTRAN, is that in MAC/NORD- PL, the logical unit number for the mass storage device shall not be in the parameter list, but in the T register, otherwise, the parameter list is the same.

Example:

```
        LDT     LOGNO       % LOGICAL UNIT
                            % NO IN T
        LDA     (PARAM      % A POINTS TO
                            % PARAMETER LIST
        MON     131         % ABSTR
        JAN     ERROR       % ERROR IF A < 0
        .
        .
        .
```

```
PARAM,    (1                    % FUNCTION CODE WRITE
          MEMO                  % PHYSICAL MEMORY ADDRESS
          (20000                % MASS STORAGE ADDRESS
          (1                    % ONE HARDWARE BLOCK
LOGNO,    502
)FILL
MEMO, * + 200/                  % ONE DISK SECTOR
%
% THE ADDRESS OF MEMO           MUST BE EVEN !
%
```

This means: write 1 block on disk unit 0. The data will be transferred from the buffer IARR to sector address 20000_8 on disk. In this example, it is assumed that the program resides in resident core, otherwise the core address (buffer address) should have been altered from the logical address IARR to the actual (corresponding) physical memory address.

ERRMON

The ERRMON monitor call used in MAC/NORD-PL should have the error number in the A-register and the suberror number in the T-register. The error number (in A-register) must be in ASCII code, maximum 2 digits.

Example:

```
          LDA     ERNO          % ERROR NUMBER
                                % (OCTAL)
          SAT 5                 % SUBERROR NUMBER
          MON     142           % ERRMON
          .
          .
          .
ERNO,     #56                   % 56 IN ASCHII CODE
```

This causes the following error message to be printed on terminal 1:

13.20.30 ERROR 56 IN 17641 AT 1 5: USER ERROR.

MEXIT

The MEXIT monitor call may be used to exchange segments in an RT program, i.e., to substitute one of the two current segments of the RT program.

Example:

```
            LDT     SEGNO       % NEW CURRENT
            JPL     CHSEG       % SEGMENT NUMBER
            .                   % CONTINUE HERE
            .                   % AFTER MEXIT
            .                   % L POINTS TO
            .                   % RETURN ADDRESS
CHSEG,      MON     133         % MEXIT
SEGNO,      177436              % THE NEW CURRENT
                                % SEGMENT NUMBERS,
                                % 1 SEGMENT NUMBER
                                % IN EACH HALF-WORD
```

This means that the calling RT program will keep its first current segment in the memory and load segment number 36 as its second segment number.

The other monitor calls must have the same arguments whether they are used from FORTRAN or from MAC/NORD-PL. In the following the examples, demonstrated in Section 7.6.1, will be placed into a MAC context. The coresponding FORTRAN calls will also be presented.

```
CALL RT (PR1)
            LDA     (PLIST      % A POINTS TO
                                % PARAMETER LIST
            MON     100         % RT
            .
            .
PLIST,      (PR1                % RT PROGRAM PR1

CALL SET (RT 1, 10, 2)
            LDA     (PLIST      % A POINTS TO
                                % PARAMETER LIST
            MON     101         % SET
            .
            .
            .
PLIST       (RT 1               % RT PROGRAM RT1
            (12                 % 10 SECONDS (12₈)
            (2                  % TIME UNIT
                                % SECOND
```

```
CALL ABSET (PROG, 0, 30, 17)
        LDA     (PLIST      % A POINTS TO
                            % PARAMETER LIST
        MON     102         % ABSET
        .
        .
        .
PLIST   (PROG               % RT PROGRAM PROG
% SET ASSEMBLER IN DECIMAL MODE
)DEC
        (0                  % SECOND
        (30                 % MINUTE
        (17                 % HOUR

% RESET ASSEMBLER TO OCTAL MODE

)OCT

CALL INTV (PP, 20, 3)
        LDA     (PAR        % A POINTS TO
                            % PARAMETER LIST
        MON     103         % INTV
        .
        .
        .
PAR,    (PP                 % RT PROGRAM PP
        (24                 % 20 TIME UNITS
                            % (SECOND)
        (3                  % TIME UNIT,
                            % SECOND

CALL DINTV (RTPR, DTEM1)
        LDA     (PARL       % A POINTS TO
                            % PARAMETER LIST
        MON     130         % DINTV
        .
        .
        .
PARL,   (RTPR               % RT PROGRAM RTPR
        DTEM1
DTIM1,  0                   % DOUBLE WORD,
                763         % THE INTERVAL
                            % FOR RT PROGRAM
                            % RTPR WILL BE 763(8)
                            % BASIC TIME UNITS
```

```
CALL RTWT
        .
        .
        MON     135     % RTWT
                        % CONTINUE NEXT
                        % EXECUTION HERE

CALL HOLD (10, 2)
        LDA     (PAR    % A POINTS TO
                        % PARAMETER LIST
        MON     104     % HOLD
        .
        .
PAR,    (12             % NO OF TIME
                        % UNITS, 10
        (2              % TIME UNITS,
                        % SECOND

CALL ABORT (PRX)
        LDA     (PLIST  % A POINTS TO
                        % PARAMETER LIST
        MON     105     % ABORT
        .
        .
PLIST,  (PRX            % ABORT RT
                        % PROGRAM PRX

CALL CONCT (CPIN, 400B)
        LDA     (PLIST  % A POINTS TO
                        % PARAMETER LIST
        MON     106     % CONCT
        .
        .
        .
PLIST,  (CPIN           % RT PROGRAM CPIN
        (400            % INTERRUPT LINE 400 OCTAL

CALL DSCNT (PRGA)
        LDA     (PAR    % A POINTS TO
                        % PARAMETER LIST
        MON     107     / DSCNT
        .
        .
        .
PAR,    (PRGA           % DISCONNECT RT
                        % PROGRAM PRGA FROM
                        % ALL INTERRUPT
                        % LINES; AND RESET
                        % PERIODICALLY
                        % EXECUTIONS
```

```
CALL PRIOR (RTPR, 30)
        LDA     (PLIST      % A POINTS TO
                            % PARAMETER LIST
        MON     100         % PRIOR
        .
        .
        .
PLIST,  (RTPR               % RT PROGRAM RTPR
        (36                 % PRIORITY, 30

CALL UPDAT (24, 11, 24, 2, 1976
        LDA     (PAR        % A POINTS TO
                            % PARAMETER LIST
        MON     111         % UPDAT
        .
        .
        .
% SET ASSEMBLER IN DECIMAL MODE
)DEC
PAR,    (24                 % MINUTE
        (11                 % HOUR
        (24                 % DAY
        (2                  % MONTH
        (1976               % YEAR
% RESET ASSEMBLER TO OCTAL MODE
)OCT

CALL CLADJ (15, 2)
        LDA     (PARL       % A POINTS TO
                            % PARAMETER LIST
        MON     112         % CLADJ
        .
        .
        .
PAR,    (17                 % NO OF TIME
                            % UNITS (15
        (2                  % TIME UNIT,
                            % SECOND

CALL FIX (35)
        LDA     (PARL       % A POINTS TO
                            % PARAMETER LIST
        MON     115         % FIX
        .
        .
        .
PARL,   (43                 % SEGMENT NO 35
```

```
CALL UNFIX (35)
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     116         % UNFIX
        .
        .
        .
PARLI,  (43                 % UNFIX
                            % SEGMENT NO 35

IX = RESERV (5, 0, 0)
        LDA     (PAR        % A POINTS TO
                            % PARAMETER LIST
        MON     122         % RESRV
        .
        .
        .
PAR,    (5                  % LOGICAL UNIT NO
        (0                  % READ/WRITE FLAG
        (0                  % RETURN FLAG

CALL RELES (LUN, 0)
        LDA     (PAR        % A POINTS TO
                            % PARAMETER LIST
        MON     123         % RELES
        .
        .
        .
PAR,    LUN,                % LOGICAL UNIT
                            % NO. LUN (5)
        (0                  % READ/WRITE
                            % FLAG
LUN,    5,

!X = PRSRV (LUN, 0, RTPRO)
        LDA     (PARAM      % A POINTS TO
                            % PARAMETER LIST
        MON     124         / PRSRV
        .
        .
        .
PARAM,  LUN                 % LOGICAL UNIT
                            % NO (LUN = 6)
        (0                  % READ/WRITE FLAG
        (RTPRO              % RT PROGRAM RTPRO
LUN,    6
```

```
CALL PRLS (5, 0)
        LDA     (PARAM      % A POINTS TO
                            % PARAMETER LIST
        MON     125         % PRLS
        .
        .
        .
PARAM,  (5                  % LOGICAL UNIT NO
        (0                  % READ/WRITE FLAG

IX = WHDEV (LUN, 0
        LDA     (PARL       % A POINTS TO
                            % PARAMETER LIST
        MON     140         % WHDEV
        .
        .
        .
PARL,   LUN                 % LOGICAL UNIT
                            % NO (LUN = 6)
        (0                  % READ/WRITE FLAG
LUN,    6

CALL RTEXT
        MON     134         % RTEXT

CALL RTOFF (RTPRG
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     137         % RTOFF
        .
        .
        .
PARLI,  (RTPR1              % RT PROGRAM
                            % RTPR1

CALL ENTSG (36, 3, 6, 50000B)
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     157         % ENTSG
        .
        .
        .
PARLI,  (44                 % SEGMENT NO 36
        (3                  % PAGE INDEX
                            % TABLE 3
        (6                  % INTERRUPT
                            % LEVEL 6
        (50000              % START ADDRESS,
                            % 50000₈
```

```
CALL FIXC (77, 48)
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     160         % FIXC
        .
        .
        .
PARLI,  (115                % SEGMENT NO 77
        (60                 % PHYSICAL
                            % PAGE NO.

CALL DSET (RTP, TMR1)
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     126         % DSET
        .
        .
        .
PARLI,  (RTP                % RT PROGRAM RTP
        TMR1
TMR1,   0                   % DOUBLE WORD,
                            % NUMBER OF
        500                 % BASIC TIME
                            % UNITS

CALL DABST (RTIMP, TMP2)
        LDA     (PARLI      % A POINTS TO
                            % PARAMETER LIST
        MON     127         % DABST
        .
        .
        .
PARLI,  (RTIMP              % RT PROGRAM
                            % RTIMP
        (TMP2
TMP2,   1                   % DOUBLE WORD,
                            % THE TIME IN
                            % BASIC TIME
                            % UNITS' THE RT
        174001              % PROGRAM RTIMP
                            % WILL BE STARTED
```

7.7 *THE REAL-TIME LOADER*

A more complete description of the RT loader is given in the manual "RT Loader, ND-60.051".

The RT loader may only be called by the users RT and SYSTEM. Only one user at a time may communicate with the RT loader.

7.7.1 *General Remarks*

The RT loader's main function is to load program units in binary relocatable format (BRF) while the system is running. The functions of the RT loader may be summarized as follows:

- relocate the program unit so that the code conforms to the specific locations in virtual memory.
- link the program units together by means of a linking table (LTBL) containing symbolic names of entry points.
- maintain a symbolic file (RTFIL) on mass storage containing the names of all real-time programs, entry points and COMMON areas known to the system.
- allocate RT descriptions in the RT description table.
- build segments. Only two new segments may be built at the same time but new segments may be linked to another already existing segment, if there is no virtual overlap between segments "linked" together.
- allocate segment descriptions in the segment table.
- print out information from the linking table and the RTFIL.
- execute "editing" functions on the linking table, the RTFIL and the RT description table.
- allocate data storage in resident memory and on the segments.
- allocate mass storage space for the segments.

The RT loader is called for execution by the command @RT-LOADER.

The first time the RT loader is executed, after having been installed, the linking table and the RTFIL are initialized.

The normal way of operating the RT loader is to specify commands in a conversational mode at a terminal. However, commands may also be read from a file by using the @MODE command. Whenever a program unit is to be loaded, the proper segments must be specified beforehand, except for the NREENTRANT-LOAD and REENTRANT-LOAD commands.

The RT loader maintains two tables containing symbolic informaton. In order to make efficient use of the loader, the operator should be familiar with their purpose and contents.

The LINKING TABLE (LTBL) contains:

- all symbolic information of the segments currently being built and all symbolic information of resident memory (common memory).

The RTFIL TABLE contains:

- all symbolic informaton of resident memory and of all existing segments which have been built by the RT loader.

ON RESET-LOADER and END-LOAD commands, the LTBL is cleared and new contents are fetched from the RTFIL. After such a command, the entry points in resident memory, including the RT program names, are found in the LTBL. Whenever an existing segment is specified in a "load" command, all entry points on this segment are fetched from the RTFIL, thus, making them available during load time.

A summarization of the different types of symbols in the linking table (LTBL) follows:

- entry points in resident memory (always in LTBL).
- entry points and references on the segments currently being built.
- memory resident COMMON area labels (always in LTBL).
- segment resident COMMON area labels.
- RT program names (always in LTBL).

Each time the loading of new segments is finished, the symbolic information in LTBL is transferred to the RTFIL (on END-LOAD command). Therefore, all symbols known to the system are stored on this file.

During load time, a scratch file is used as temporary storage for the code. Information about data areas (COMMON) and their initial values is kept in LTBL until the END-LOAD command is given. Then the COMMON areas are allocated and initialized. Thereafter, the actual segment file is searched for a sufficiently great free area and the segment(s) on the scratch file is copied to the segment file.

The RT loader maintains a bit map of the segment files.

7.7.2 *Segment Files*

The segment files are contiguous areas on mass storage (disk or drum) where all segments in the SINTRAN III system reside. The swapping of pages will be performed between the area of a segment in a segment file and physical memory at run-time immediately before the code on the segments is to be executed.

The segment files may be defined on any disk or drum file directory in a SINTRAN III system.

The segment files are defined by the operator command @ALLOCATE-FILE and the RT loader command *DEFINE-SEGMENT-FILE.

7.7.3 *RT Loader Commands*

7.7.3.1 CLEAR AN EXISTING SEGMENT

*CLEAR-SEGMENT <segment no>

The segment <segment no> will be cleared, i.e., the space on the segment file occupied by the segment <segment no> will be released and the segment number <segment no> will be free again. The segment cannot be one of the segments initially present in the SINTRAN III system. The segment will not be cleared if it is one of the segments of an existing RT program, if the segment is currently being used by an RT program, or if it has been fixed using the FIX or FIXC command.

The parameter <segment no> is given the value of zero, which is equivalent to memory common, and the question "CLEARING MEMORY COMMON?" will be printed. If the answer Y, for yes, if given the memory common pointers will be reset to their initial values, and all memory common labels will be deleted from the linking table and the RTFIL.

When clearing a segment, all symbols defined on this segment will be deleted from RTFIL and the linking table.

7.7.3.2 DECLARE AN RT PROGRAM NAME

*DECLARE-PROGRAM <rt program name>

The symbol <rt program name> will be defined as the name of an RT program and an entry in the RT description table will be allocated. This command is necessary when loading RT programs which have other RT programs as "externals", and these 'external RT programs" are not yet defined or declared.

7.7.3.3 DEFINE THE NAME OF A SEGMENT FILE

*DEFINE-SEGMENT-FILE <segment file name> <segment file no>

Define the segment file number <segment file no>. The parameter <segment file name> will be the name of the segment file. If the segment file number <segment file no> is already defined, then this segment file's name and the question REDEFINE SEGMENT FILE? will be printed. The answer Y, for yes, will result in the segment file's name being changed to <segment file name>.

Before using the DEFINE-SEGMENT-FILE command, the specified segment file must have been defined with the ALLOCATE-FILE command and the mass storage address of the segment file must have been defined with the ALLOCATE-FILE command and the mass storage address of the segment file must have been inserted in the "BLST" array in the SINTRAN III system.

Example:

```
*DEFINE-SEGMENT- FILE
SEGMENT FILE NAME: FIXED-PACK:SYSTEM) –
SEG-FIL1:DATA
SEGMENT FILE NO.: 1
*
```

7.7.3.4 DEFINE A SYMBOL

*DEFINE-SYMBOL <symbol> <value> [<segment no>]

Define the symbol <symbol> on the segment <segment no> and give it the value <value>. The parameter <segment no> must be an existing segment or one of the segments currently being built. The default value of the parameter <segment no> is the current "load segment', the segment last loaded into the current load operation.

Example:

```
*NEW-SEGMENT,,,
NEW SEGMENT NO: 31
*DEFINE-SYMBOL
SYMBOL NAME: SYMB!
VALUE: 0
SEGMENT NO: 31
*DEFINE-SYMBOL SYMB2 1 31
*
```

7.7.3.5 DELETE THE NAME OF THE NON-REENTRANT ROUTINES IN THE "REENTRANT" FORTRAN LIBRARY (FTNRTLIBR)

*DELETE-NON-REENTRANT

The names of the non-reentrant routines in the reentrant FORTRAN library will be deleted. This command is useful when building a reentrant system with more than one RT program on the same segment. After each RT program is loaded: define the end of the stack, delete the names of the non-reentrant routines, set the new load address (equals end of stack plus one), load the next RT program, etc.

The names of the non-reentrant routines in the "reentrant" FORTRAN library are:

8DXI, DEXP, DLOG, DLOG10, DSIN, DCOS, DSORT, DATAN, DTAN2, DMOD, 8DIV, 8STAC, STPNT, STBEG, STEND, 8RTEN, 8ENTR, 8STKI.

7.7.3.6 DELETE AN RT PROGRAM

*DELETE-PROGRAM <rt program name>

The RT program named <rt program name> will be deleted from RTFIL and from the linking table, and the RT programs entry in the RT description table will be free again. If the RT program <rt program name> is active, the DELETE-PROGRAM command is illegal.

7.7.3.7 DELETE A SYMBOL FROM THE LINKING TABLE

*DELETE-SYMBOL <symbol>

The symbol names <symbol> will be deleted from the linking table. The symbol <symbol> must not be a common label or an RT program.

7.7.3.8 END A LOAD OPERATION

*END-LOAD

The END-LOAD command must terminate all load operations. This command will close the segments currently being built. The segments will be moved from the scratch file to the segment file and the RTFIL. The linking table, the segment table and the RT description table will be updated. The RTFIL table will be written to the file RTFIL during the END- LOAD command. If there are undefined symbols in the linking table when an END-LOAD command is typed, the question NEGLECTING REFERENCES? will be printed. If the answer is Y, for yes, then the END-LOAD command will continue. Otherwise, the END- LOAD command is terminated and the load operation may continue.

If the command NREENTRANT-LOAD was the last "load" command, then the file RTNLIBR will automatically be scanned in the END-LOAD command if there are undefined symbols in the linking table.

Example:

```
*NREENTRANT-LOAD 200-USER,,
NEW SEGMENT NO: 33
*END-LOAD
*
```

7.7.3.9 EXIT FROM THE RT LOADER

*EXIT-LOADER

This command will update the file RTFIL and then leave the RT loader and give control to the SINTRAN III command processor.

7.7.3.10 LIST THE AVAILABLE COMMANDS

*HELP [<output file>]

This command will list all the RT loader's commands on the <output file>. The output will be in alphabetical order.

If the terminal is used as <output file>, the output is divided into three parts. For each part, the RT loader will give the question NEXT COMMANDS?. If the answer is Y, for yes, then the next part is listed, otherwise the command is terminated. The terminal is the default value of the <output file> parameter.

7.7.3.11 LOAD A SINTRAN III MEMORY ONLY SYSTEM

*IMAGE-LOAD <image file> <output file> [<bootstrap start addr>]

This command will set the RT loader in "image load" mode, i.e., loading will be to a file instead of to a segment.

The parameter <image file> is the name of the file where the SINTRAN III C system is resident in binary format. <output file> is the name of the file where the completed SINTRAN III C system will be dumped by the END-LOAD command. The parameter <bootstrap start addr> is the address of the bootstrap, i.e., the address where the bootstrap will be placed in memory when the SINTRAN III C system is loaded and started. The default value of <bootstrap start addr> is the value of the load address when the load operation is terminated.

The "image load" mode is reset by the END-LOAD and the RESET-LOADER command.

Example:

```
*IMAGE-LOAD
IMAGE FILE: CORE-SINTRAN:SYMB
OUTPUT FILE: TAPE-PUNCH
BOOTSTRAP START ADDRESS:
*SET-LOAD-ADDRESS 26000
*NREENTRANT-LOAD 200-USER
*END-LOAD
```

7.7.3.12 LIST THE AVAILABLE FREE SEGMENT NUMBERS

*LIST-FREE-SEGMENTS <output file>

The unused segment numbers in the system will be listed on the <output file>. Default value of <output file> is the terminal.

7.7.3.13 LOAD FROM THE SPECIFIED INPUT FILE INTO THE SPECIFIED SEGMENT

*LOAD [<input file>] [<load-segment>] [<link-segment>]

Load BRF code from the file <input file> into the segment <load-segment>. The <load-segment> must have been specified in a NEW-SEGMENT command before it may be used in the LOAD command. The <link-segment> must be an existing segment, or one of the two segments currently being built. Line-segment means that all symbols defined on the link-segment will be available in the load operation. There must be no virtual address overlap between the load-segment and the link-segment. If no <input file> parameter is specified the last input file specified will be used. If no <load-segment> is specified, the last segment used as load-segment or the last segment specified in a NEW-SEGMENT command will be used. Default value of the parameter <link-segment> is the second segment currently being built, or no link-segment if no "second" segment is specified. The parameter <link-segment> may be given the value zero to avoid linking to another segment in a load operation.

Example:

```
*NEW-SEMGENT,,
NEW SEGMENT NO: 34
*LOAD
INPUT FILE: TW2
LOAD-SEGMENT NO.: 34
LINKING-SEGMENT NO.:
*LOAD WAITF,,
*LOAD FTNLIBR,,
*END-LOAD
*
```

7.7.3.14 SPECIFY THE NEW SEGMENT TO BE BUILT

*NEW-SEGMENT [<segment no>] [<ring>] [<demand/non-demand>] [<permit protection bits>]

Allocates a segment number to use the current load operation. The <segment no> must be an available free segment number and the default value is the first free segment number. The parameter <demand/non-demand> specifies whether the segment will be a demand segment or a non-demand segment. The default type is non-demand. Legal values for the parameter <demand/non-demand> are the characters ND for non-demand and DM for demand. The parameter <permit protection bits> specifies whether the segment is to be fetch permitted, read permitted, or write permitted. Legal values for this parameter are F for fetch, R for read, and W for write permitted, or a combination of these three characters. Default value is RFW.

A maximum of two segments may be specified by the NEW-SEGMENT command in the same load operation.

Example:

```
*NEW-SEGMENT
SEGMENT NO: 40
RING: 2
SEGMENT TYPE: DM
PROTECTION BITS: RF
*NEW-SEGMENT
SEGMENT NO:
RING:
SEGMENT TYPE:
PROTECTION BITS:
NEW SEGMENT NO: 35
*
```

In the first NEW-SEGMENT command in the example, the segment number 40 is specified to be a demand segment on protect ring 2 and only read and fetch permitted. In the second NEW-SEGMENT command only default parameters are used and the result is that the first free segment, number 35, is allocated. This segment is non-demand. It resides on protection ring 0 and it is read, write and fetch permitted.

7.7.3.15 LOAD FROM THE SPECIFIED INPUT FILE INTO THE CURRENT LOAD-SEGMENT

*NREENTRANT-LOAD [<input file>] [<link-segment>]

Load BRF code from the file <input file> into the current load segment, which is the last segment loaded into the current load operation or the last segment specified in a NEW-SEGMENT command. If no current load segment exists, the first free segment number will be allocated and used as the current load segment. If a new segment is allocated, it will be a non-demand segment residing on protection ring 0 and it will be read, write and fetch permitted. The link segment <link-segment> must be one of the two segments currently being built or an already existing segment, or <link-segment> can equal zero meaning that no linking is desired. The default value of the parameter <link-segment> is the last segment used as link segment or the "second" segment currently being built. The default value of the parameter <input file> is the last file used as <input file>.

The file FTNLIBR, containing the FORTRAN run-time system, will be scanned (loaded from) in the END-LOAD command if there are undefined symbols and the last load command was the NREENTRANT-LOAD command.

Example:

```
*NREENTRANT-LOAD
INPUT FILE: TW2
LINKING-SEGMENT NO.:
NEW SEGMENT NO: 35
*NREENTRANT-LOAD WAITF,,
*END-LOAD
*
```

7.7.3.16 LOAD REENTRANT PROGRAM SYSTEMS ONTO A SPECIFIED SEGMENT

*REENTRANT-LOAD [<input file>] [<line-segment>] [<stack length>]

Load BRF code into the current load segment from the file <input file>. The current load segment is the last segment loaded into the current load operation, or the last segment specified in a NEW-SEGMENT command. If no current load segment exists, then the first free segment number will be allocated as the current load segment. This segment will be a non-demand segment, residing on protection ring 0, and will be read, write and fetch permitted.

The <link-segment> may refer to one of the segments currently being built, an already existing segment or have the value zero if no linking is desired. The default value of the parameter <link-segment> is the last segment used as link segment in the current load operation. The last file used as <input file> is default value of the parameter <input file>.

After each REENTRANT-LOAD command the file FTNRTLIBR, containing the "reentrant" FORTRAN run-time system, is scanned if the symbol STEND (end of stack) is undefined. Then the symbol STEND is defined and the names of the non-reentrant routines are deleted from the linking table. The symbol STEND will receive a value equal to the load address after the file FTNRTLIBR is scanned plus the value of the parameter <stack length>. The load address of the segment will be set equal to STEND plus one. 1K words is the default value of the parameter <stack length>.

This command is useful when building a system consisting of reentrant FORTRAN programs on the same segment. The BRF code of the various RT programs should be placed on different files and one then uses a single REENTRANT-LOAD command for each RT program.

Example:

```
*REENTRANT- LOAD
INPUT FILE: REENT-TW2
LINKING-SEGMENT NO:
STACK LENGTH: 400
NEW SEGMENT NO: 40
*END-LOAD
*
```

7.7.3.17 RESET THE RT LOADER

*RESET-LOADER

This command will reset the RT loader to its initial state, which is the state after the last EXIT-LOADER, END-LOAD or RESET-LOADER command.

7.7.3.18 ALLOCATE COMMON AREA IN RESIDENT CORE

*SET-CORE-COMMON <common label>

The common area labelled <common label> will be allocated in resident memory. This command must be used before the common area <common label> is loaded.

7.7.3.19 SET THE LOAD ADDRESS OF A SEGMENT

*SET-LOAD-ADDRESS <segment no> <load address>

Set the current load address of the segment <segment no> to the value <load address>. The segment <segment no> must be one of the segments currently being loaded in, or segment number can have the value zero meaning memory common. When memory common is specified, the question CHANGING LOAD ADDRESS OF MEMORY COMMON? is printed, and this must be answered with Y, for yes, before any changing memory common load address can occur.

7.7.3.20 COMMAND TO ALLOCATE COMMON AREA ON THE SECOND SEGMENT CURRENTLY BEING BUILT

*SEG-SEGMENT-COMMON <common label>

The common area labelled <common label> will be allocated on the segment specified in the second NEW-SEGMENT command in a load operation. This command must be used before the common label <common label> is defined.

7.7.3.21 LIST NAMES OF ALL THE COMMON LABELS IN THE LINKING TABLE

*WRITE-COMMON-LABELS <output file>

List the names, addresses and the segment numbers of all the common labels defined or declared in the linking table, on the file <output file>. The terminal is the default value of the parameter <output file>.

7.7.3.22 WRITE THE LOWER AND UPPER ADDRESS LIMITS, AND THE CURRENT LOAD ADDRESS OF A SPECIFIED SEGMENT

*WRITE-LOAD-ADDRESS <segment number>

Write the lowest virtual address, the highest virtual address and the current load address of the specified segment <segment no>. This segment must be one of the segments currently being built. When the value zero is given for the parameter <segment no>, the addresses of memory common are listed.

7.7.3.23 LIST THE NAMES OF THE RT PROGRAMS

*WRITE-PROGRAMS [<output file>]

List the names of all the RT programs defined and declared on the file <output file>.

Each of the RT programs's two segment numbers and the address of each RT program's RT description will also be listed. Declared RT programs will not have segment numbers, so question marks will be written instead of segment numbers.

The default value of the parameter <output file> is the terminal.

7.7.3.24 LIST OUT THE UNDEFINED SYMBOLS

*WRITE-REFERENCES [<output file>]

All undefined symbols in the linking table will be listed on the <output file>. The terminal is the default value of the parameter <output file>.

7.7.3.25 LIST THE SYMBOLS IN RTFIL

*WRITE-RTFIL [<segment no>] [<output file>]

List all the symbols with the segment number <segment no> on the file <output file>. If no <segment no> is specified, all the symbols in RTFIL will be listed. The terminal is the default value of the parameter <output file>.

7.7.3.26 LIST ALL THE INFORMATION ABOUT A SPECIFIED SEGMENT

*WRITE-SEGMENTS [<segment no>] [<output file>]

List all information about the specified segment <segment no>. The information listed is the segment number, the segment's lower and higher virtual addresses, the mass storage address (in pages) relative to the start of the segment file, the segment file number, the page index table number, on which protection ring the segments reside and the memory protection type (demand/non-demand).

If no parameter <segment no> is specified, then all segments are listed out. When the value zero is given for the parameter <segment no> the address limits of the memory common area are listed. The terminal is the default value of the parameter <output file>.

7.7.3.27 LIST THE DEFINED SYMBOLS IN THE LINKING TABLE

*WRITE-SYMBOLS <output file>

List the names, the segments and the values of all defined symbols in the linking table, on the file <output file>. The terminal is the default value of the parameter <output file>.

8 SINTRAN III/SINTRAN III COMMUNICATION

8.1 *INTRODUCTION*

The SINTRAN III/SINTRAN III communication system is an optional part of the SINTRAN III I/O system for communication between two (or more) SINTRAN III systems. The two systems may be SINTRAN III/10 or SINTRAN III/12 systems. The communication serves the following purposes:

a) Data transfer between two user programs, one in each SINTRAN III system. This connection will, from the user program's point of view, look like an internal device connection between two user programs in the same SINTRAN III system.

b) Remote terminal communication. This means that a user on a terminal connected to one of the SINTRAN III systems may run the operator communication system and background system on the remote SINTRAN III system.

c) Remote load. The remote SINTRAN III computer may be loaded from the other SINTRAN III system. Only main memory will be loaded.

d) Watch-dog connection.

The communication is essentially the same, regardless of the communication line used. The line must fulfill the following requirements:

a) Logical full duplex connection. At least half duplex hardware is required.

b) Binary transparent transmission facilities on byte level.

c) Sufficient capacity for the actual load.

8.2 *COMMUNICATION LINE*

The communication line will be divided in up to thirty-two logical lines each way, hereafter called channels. The channels will be numbered from zero to thirty-one. If more than thirty-two channels are needed, two communication lines must be available.

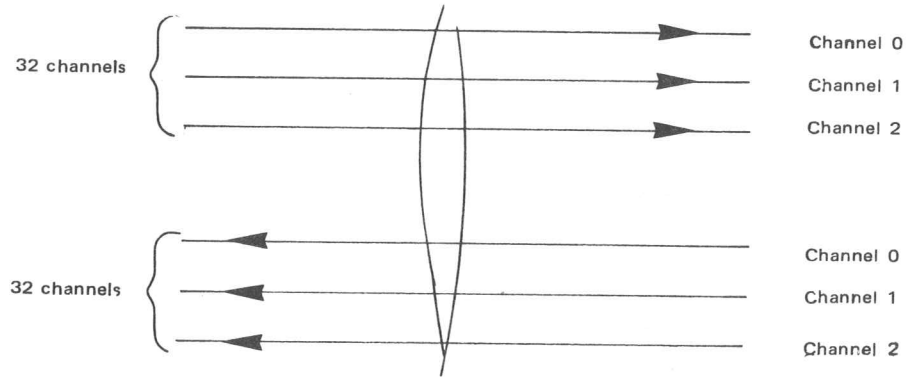


Figure 8.1: *One Communication Line*

Each channel is provided with a buffer on each side. A buffer is scheduled for transmission either when the buffer is full or when a break character is inserted in the buffer. When a buffer is scheduled for sending, the corresponding channel may be supplied with a new buffer from a buffer pool.

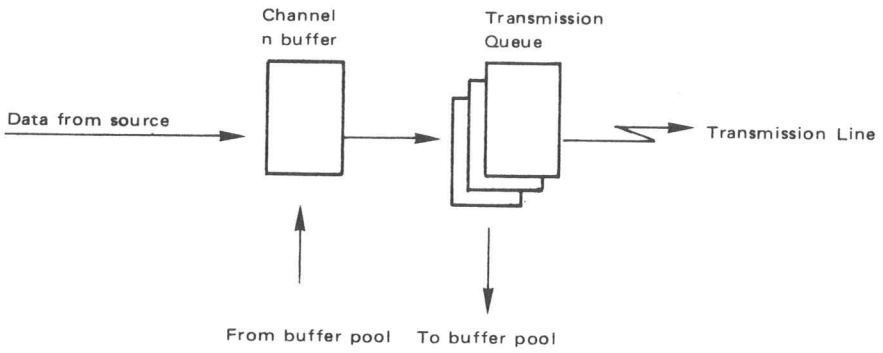


Figure 8.2.

On the receiving side, the opposite action is performed. Buffers are received and queued for the various destinations.

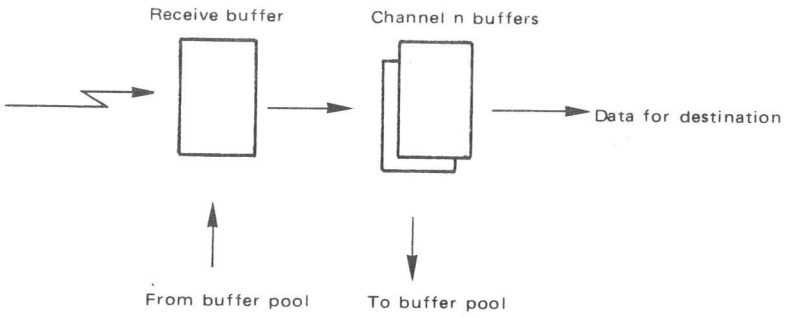


Figure 8.3.

Acknowledgement for correctly received buffers are transmitted together with the data buffers in the opposite direction. Up to four buffers may be transmitted without receiving acknowledgement. This is done by dividing the buffers into four groups. For each group, the buffer is not discarded until acknowledgement for this group is received. The buffers for sending are always directed to the four groups in an eyelie manner to ensure correct sequence. On the receiving side, they are distributed in the same cyclic manner.

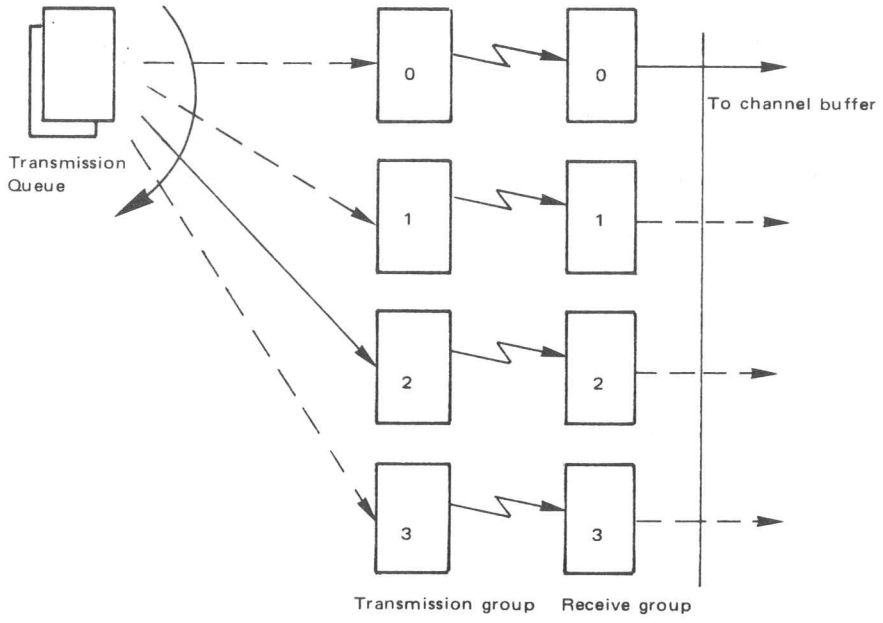


Figure 8.4.

When a buffer is transmitted it is preceded by a buffer header and followed by a cyclic check sum. The transmission buffers have the following format :

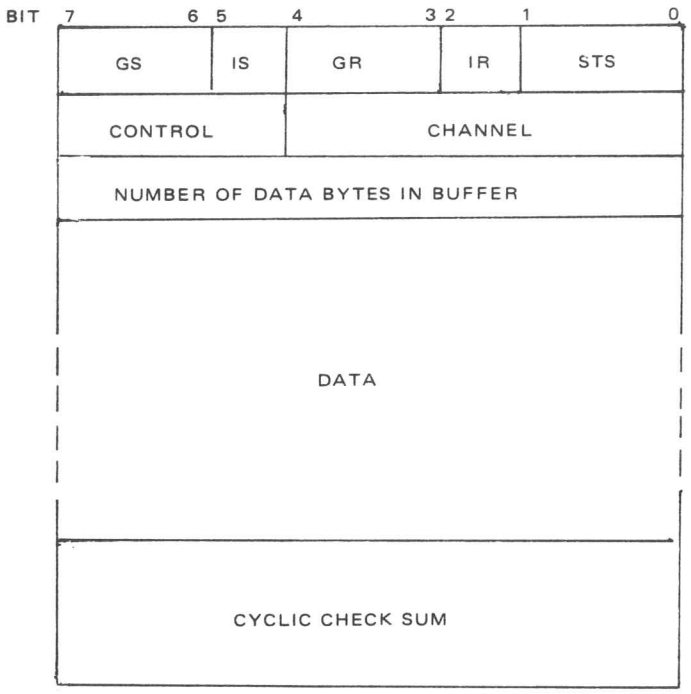


Figure 8.5: *Format of a Transmission Buffer*

The fields in the buffer header have the following meaning:

GS - Group number for this buffer

IS - Sequence number for GS

GR - Group number for the status information in STS

IR - Sequence number for GR

STS - Status information for buffer transmitted in the opposite direction.

1 - not acknowledged
2 - acknowledge
3 - wait acknowledge

The control field may have the following values:

0 - data transfer
1 - define break strategy
2 - define echo strategy
3 - request for input
4 - turn off request for input
5 - system configuration message (used for initialization)

Depending on the type of connection used, this format may be framed by syne, start of text, end of text, etc.

A buffer in a transmission group is not discarded until acknowledgement for this group (with correct sequence number) is received. If not acknowledged is received, or time-out, the buffer is retransmitted.

Each channel is assigned a logical device number on each side and may be reserved, released and accessed exactly like any other device in SINTRAN III. The logical device number on the sending side does not have to be the same as the one on the receiving end. Refer to Figure 8.6.

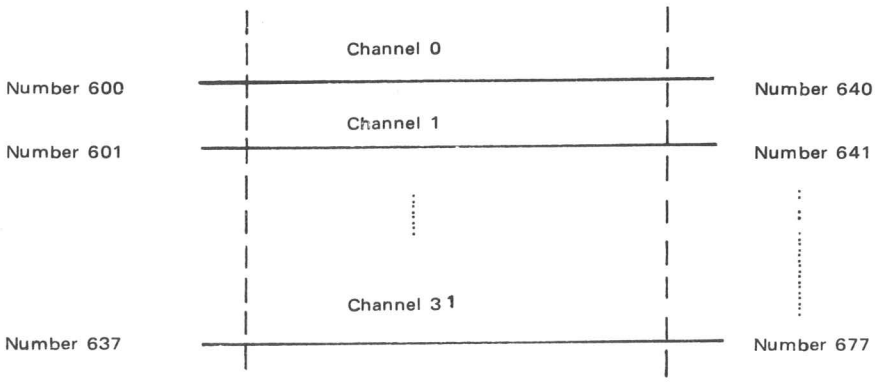


Figure 8.6.

8.2.1 *Commands to Initiate and Terminate Communication*

@START-COMMUNICATION <line number>

Parameters:

<line number> may be omitted if there is only one possible remote connection.

Function:

Initiate communication on a communication line. A configuration table containing the correspondence between channel numbers and logical device numbers is sent to the remote computer. If the remote computer answers with its own configuration table within 12 seconds, the message COMMUNICATION ESTABLISHED is printed. If no answer is received, the message NO REPLY is printed and control is returned to the operators communication. However, the system continues to retransmit the configuration table until a @STOP-COMMUNICATION command is given. Thus, there may be an arbitrary time delay between the START-COMMUNICATION commands on the two computers.

This command may only be executed by user SYSTEM.

@STOP-COMMUNICATION <line number>

Parameters:

<line number> may be omitted if there is only one possible remote connection.

Function:

Terminate communication on a communication line.

This command may only be executed by user SYSTEM.

8.2.2 *The COMMUNICATION-STATUS Command*

The format is:

@COMMUNICATION-STATUS <line printer>

Parameters:

<line number> may be omitted if there is only one possible remote connection.

Function:

The following information on the communication line will be listed on the terminal.

- the logical device numbers of the implemented channels on both sides of the line
- the number of unacknowledged messages sent
- the number of unacknowledged messages received
- the number of messages received out of sequence
- COMMUNICATION RUNNING or COMMUNICATION DEAD. (This command does not affect the communication.)

8.3 *DATA TRANSFER*

Some of the channels may be used for data transfers between two user programs. Such a channel is used in the following way:

1. The channel must be reserved by the user programs. Each program receives the corresponding logical device number on its side of the connection.

2. The receiving program asks for input (by the monitor call INBT) and is set in waiting state until a buffer is received on this channel.

3. The sending program sends output (by the monitor call OUTBT) and is set in waiting state when one of the following conditions occurs:

   - a break character is sent
   - the buffer pool is "almost" full
   - a wait acknowledge is received on this channel.

   The sending program is restarted again when a request for input is received on this channel. A request for input is sent from the receiving side when a break character is encountered on input, and the receiving program asks for more input. A wait acknowledge is sent if the input queue for a channel exceeds a predefined (system generation parameter) number of buffers. This is done to prevent one channel from occupying the whole buffer pool if the receiving program reads data at a lower rate than the sending program sends data. A wait acknowledge simulates a break character at the end of the last transmitted buffer on this channel.

4. The break strategy may be defined by the receiving program. The strategy is transmitted to the sending system as a special buffer with control = 1. If a negative break strategy is defined, no characters will be break characters and all transmission buffers will have the maximum length, except the last one which is sent when the sending program executes a CLOSE-FILE or IOSET on the channel. Note that the break strategy should be set to break as little as possible to utilize the transmission line and reduce system overhead.

The following standard SINTRAN monitor calls may be used on a communication channel:

INBT	Input a byte
OUTBT	Output a byte
CIBUF	Clear input buffer
COBUF	Clear output buffer
IOSET	On input: equivalent to CIBUF On output: equivalent to COBUF
BRKM	Set break strategy
ECHOM	Set echo strategy. This monitor call will only have effect if the program on the other side is a remote terminal processor.

If the channel is defined to the file system as a peripheral file by the @SET-PERIPHERAL-FILE command, the monitor calls OPEN-FILE and CLOSE-FILE may also be used.

It should be seen from the description above that the channels on a communication line may be used as independent byte oriented peripheral devices.

Example of assembly program to receive data from a communication channel with device number 600:

```
        LDA     (REPAR
        MON     122         % Reserve channel
        LDT     (600
        MON     13          % Clear input buffer
        JMP     ERROR       % Error exit
        SAA     -1
        MON     4           % Set break strategy
LOOP,   LDT     (600
        MON     1           % Input a byte
        JMP     ERROR       % Error exit
        .                   % Process the byte read
        .                   % and test if finished
        JMP     LOOP
REPAR'  (600
        (0
        (0
        )FILL
```

It is always recommended to use the Clear input buffer monitor call in the initializing sequence, in case the program last using the channel terminated abnormally.

Example of FORTRAN program to write a record to a communication channel:

```
      :
      :
      I = RESERV (600B, 1, 0)
      I = IOSET (600B, 1, 0, -1)
      :
      WRITE (600B, 10).....
10    FORMAT (.....)
```

In addition to the standard monitor calls mentioned above there is a special I/O monitor call, WRQI, for use on communication channels.

MON 163

Input parameter:

T-reg. = channel number

Exit:

Skip return if okay. No skip return if error, error code in A-reg.

Function:

The monitor call will place the calling program into waiting state until a request for input message is received from the remote computer.

This monitor call is useful in interactive communication programs, when one does not want to start the local echoing of terminal input before the receiving program asks for input.

8.4 *TERMINAL CONNECTION*

Some of the channels may be connected to a remote processor. These channels will be used for communication from a terminal user in one SINTRAN III system to the operator's communication and background system in the other SINTRAN III system.

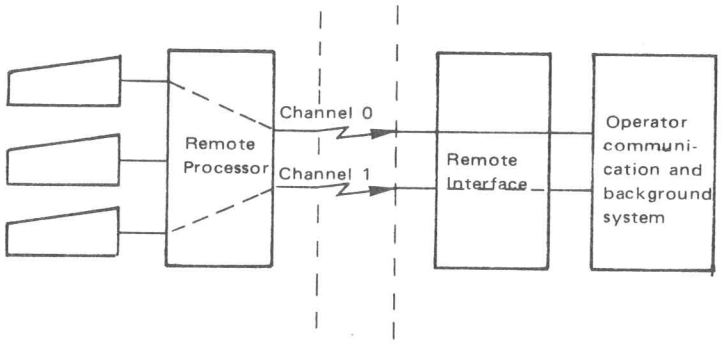


Figure 8.7.

A terminal user may be connected to the Remote processor by typing REMOTE n on this terminal. The number n is only used if he may be connected to more than one remote SINTRAN III system. The Remote processor will select one channel to the remote system and connect the terminal to his channel. If no channels are available, there will be no connection.

After connection, the user may use the remote system exactly as if he was connected to the local system, with one exception. The "Rub-out" character will bring him back to the local system, where he may disconnect the remote connection by typing LOCAL. However, the combination "Shift, control L" "Rub-out" will be mapped to "Rub-out' and transmitted to the remote system.

Example:

```
.
. local processing
.
.
@REMOTE
CHANNEL NUMBERS' LOCAL -1600 REMOTE -600
"FSC"
17.10.15 1 JANUARY 1975
ENTER OLE
PASSWORD'
OK
R@
.
.
. Remote processing
.
.
R@LOGOUT
17.15.10 1 JANUARY 1975
-EXIT-
"Rub-out"
@
.
. Local processing
.
.
```

The R@ (instead of a single @) indicates that one is working on the remote computer.

If the user after a "Rub-out" types REMOTE, he will be connected to the same channel as before. He will then be in the same state (on the remote computer) as before he typed "Rub-out" and may continue giving input to the remote system. If the remote system gives output to the remote terminal while this is in local mode, this output will appear on the terminal when the REMOTE command is retyped.

In this way, it is possible to go back and forth between the remote and the local system, and do processing at both computers at the same time. For example, while a compilation is going on the remote computer one may use the editor on the local system. The channel is reserved for the remote terminal connection until a @LOCAL command is given.

Summary of commands

@REMOTE <line number>

Parameters:

<line number> may be omitted if there is only one possible remote connection.

Function:

If no remote connection exists for this terminal (no REMOTE command has been given since the last LOCAL command) a free channel is found and the terminal is connected to the background processor of the remote computer. If a remote connection already exists the terminal is connected to this channel. In this case, the REMOTE command is the inverse of the "Rub-out" character.

"Rub-out":

This single character returns control to the local command processor, so that terminal input again is considered as local commands. To send a "Rub-out" to the remote computer, press the Shift, Control and L buttons, and then the "Rub-out" character.

@LOCAL

Parameters:

None.

Function:

Disconnect remote connection. After this command the communication channel used by the remote connection is released and may be used for other purposes.

8.5 *REMOTE LOAD*

Remote load must be initialized by a bootstrap in the microprocessor's load format. The bootstrap is transmitted without buffer headers. The bootstrap program is loaded in the remote computer. When the program is started, it will accept buffers with load data.

Loading of core in a remote computer from a local computer is done in the following way:

- The bootstrap program must reside (in the local system) on a system file called (SYSTEM REMOTE-BOOTSTRAP:BPUN in the MAC assemblers) BPUN format. This will usually be done at system generation time.

- Push the master clear button on the remote computer and type <octal number> and on the consol terminal. <octal number> is the hardware device number of the communication line. If the ALD register is set to the right value, only the master clear and load buttons have to be pushed.

- Give the command

  @REMOTE-LOAD <load-file>, <bootstrap address>, <line number> on the local computer.

  The <load-file> must contain the program to be loaded in the MAC assembler's BPUN format. Default file type is BPUN.

  The <bootstrap address> specifies where to place the bootstrap program in the remote computer. The bootstrap will occupy approximately 400_8 words from the bootstrap address given.

  It is the users responsibility to avoid overlapping the bootstrap and the program to be loaded.

  <Line number> may be omitted if there is only one possible remote connection.

  The REMOTE-LOAD command may only be executed whent the communication system is stopped on that line.

Example:

@REMOTE-LOAD TAPE-READER, 37400, 1

A binary tape, placed in the Tape Reader on the local system will be loaded into the remote system connected to line number 1. The bootstrap will occupy the memory area 37400-37777 in the remote computer.

8.6 *WATCH-DOG*

A watch-dog connection may be obtained by running an RT-program in both computers, sending and receiving some dummy message to/from the other computer. If the other computer does not answer within a predefined time interval, some recovery action may be taken.

9 SPECIAL PERIPHERAL DEVICES

In this chapter the application of some special peripheral devices by means of SINTRAN III commands and monitor calls, will be described.

9.1 *THE DEVICE-FUNCTION COMMAND*

The command @DEVICE-FUNCTION may be used to perform operations on magnetic tapes, cassette tapes, versatec printer/plotter and floppy disk. The @DEVICE-FUNCTION command has the following format:

@DEVICE-FUNCTION <file name> <function name> (<optional parameter1>) (<optional parameter 2>)

where

<file name>

is the name of the device unit, specified in the command @SET-PERIPHERAL-FILE

<function name>

is the name of the operation to perform on the device.

<optional parameter1>
<optional parameter2>

These two parameters are used to specify more information needed by some operations.

Explanation of the functions:

The functions READ-RECORD and WRITE-RECORD will transfer a specified amount of data from/to the specified address in the user area (background segment) to/from the specified device unit.

The function UNLOCK-AND-STOP means that the cassette tape will stop and the read/write head will be removed from the tape. The cassette may then be removed from the cassette driver.

The function LOCK-CASSETTE means that the read/write head of the cassette drive will be moved onto the cassette tape, ready for read/write the cassette tape. The cassette tape cannot be removed from the cassette drive when the cassette is in lock position.

The function READ-STATUS will read the status register of the specified device unit by means of an IOX instruction. The function READ-LAST-STATUS will return the status from the last operation on the device without extending any IOX instructions. (The status is saved by the driver.)

Table 9.1.

Function Name:	Function Code (octal):	Optional Parameter 1:	Optional Parameter 2:	Allowed on Mag. Tape:	Allowed on Cassette Tape:	Allowed on Versatec:	Allowed on Floppy Disk:
Read-Record	0	Octal addr. of data buffer in users area	Octal no. of words to read	yes	yes	no	yes
Write-Record	1	Octal addr. of data buffer in users area	Octal no. of words to write	yes	yes	yes	yes
Unlock-and-Stop	5			no	yes	no	no
Lock-Cassette	6			no	yes	no	no
Erase-Tape	7			no	yes	no	no
Advance-to-EOF	10	No of EOF marks to pass over		yes	yes	no	yes
Reverse-to-EOF	11	No. of EOF marks to pass over		yes	yes	no	yes
Write-EOF	12	No. of EOF marks to write		yes	yes	no	yes
Rewind	13	To load point or addr. 0		yes	yes	no	yes
Write-Erase-Gap	14	No. of erase-gaps to write		yes	yes	no	no
Back-Space-Records	15	No. of records to back space		yes	yes	no	yes
Advance-Records	16	No. of records to advance		yes	yes	no	yes
Unload	17			yes	yes	no	no

Function Name:	Function Code (octal):	Optional Parameter 1:	Optional Parameter 2:	Allowed on Mag. Tape:	Allowed on Cassette Tape:	Allowed on Versatec:	Allowed on Floppy Disk:
Read-Status	20			yes	yes	yes	yes
Clear-Device	21			yes	yes	yes	no
Select-Density-and-Parity	23	Density/parity value (see section 9.2.2.1)		yes/no	no	no	no
Read-Last-Status	24			yes	yes	yes	yes
Set-Alphanumeric-Mode	30			no	no	yes	no
Set-Graphic-Mode	31			no	no	yes	no
Give-Form-Feed	32	No. of form feeds to give		no	no	yes	no
Set-Floppy-Format	40	Format no. (see section 9.2)		no	no	no	yes
Format-Floppy	41			no	no	no	yes

Table 9.1, concluded.

9.2 *MAGNETIC TAPES AND CASSETTE TAPES*

This section describes the usage of magnetic tape and cassette tape, as freestanding devices independent of the File Management System.

Magnetic tape and cassette tape units may be declared as peripheral units at the system initialization, similar to other peripheral units as paper tape reader, line printer, etc. This must be done by the user SYSTEM with the @SET-PERIPHERAL-FILE command.

The logical device numbers for magnetic tapes and cassette tapes are:

Device	Unit No.	Logical Device No. (octal)	Logical Device No. (decimal)
Mag. tape controller 1	0	40	32
Mag. tape controller 1	1	41	33
Mag. tape controller 1	2	25	21
Mag. tape controller 1	3	33	27
Mag. tape controller 2	0	32	26
Mag. tape controller 2	1	34	28
Cassette tape cont. 1	0	20	16
Cassette tape cont. 1	1	21	17

9.2.1 *Sequential Read and Write*

Reading or writing data is accomplished using the standard SINTRAN III Input/Output system, with the INBT/OUTBT monitor call functions.

Data is normally divided into records of 1K words (2048 bytes) for magnetic tape, or 256 words (512 bytes) for cassette tape, but shorter records may be read and written too.

When writing on a tape unit with the OUTBT monitor call, the last record will be transferred to the unit when the file is closed, but no EOF mark is written. This record may be shorter than other records.

To write an EOF mark on the tape unit, the command @DEVICE-FUNCTION should be used. See Section 9.1.

The name of a tape unit may be given in an @OPEN-FILE command or other file name requests from user programs or subsystems.

9.2.2 *The Monitor Call MAGTP*

This monitor call may be used to access a magnetic tape or cassette tape from a user program, to read, write or position the tape unit to a file or record.

To prevent other users from accessing the device unit when the MAGTP is used, the user may open the actual device unit (peripheral file) with the @OPEN-FILE command and close it with the @CLOSE-FILE command to allow other users access to the device unit. When the MAGTP monitor call is used from an RT program, the RT program may reserve the device unit, making it unaccessable for other users and RT programs, and releases it later.

The calling sequence in FORTRAN is:

ISTAT = MAGTP (<function code>, <memory address>, <logical device number>, <max words>, <words read>)

where ISTAT receives a function value.

The calling sequence in MAC assembly is:

```
        LDA     (PLIST          % POINTER TO PARAMETER
                                  LIST
        MON     144             % MAGTP
        JAF     ERROR           % ERROR OCCURRED
        .....
PLIST,  FUNC                    % FUNCTION CODE
        MEMAD                   % MEMORY ADDRESS
        UNIT                    % LOGICAL DEVICE NUMBER
        MAXWD                   % MAX WORDS TO READ/WRITE
        READW                   % ACTUAL WORDS READ
```

The A register receives a function value.

If the returned function value is zero, then the MAGTP call is executed without errors. If the function value is unequal to zero, then an error occurred in the MAGTP call. The error message may be written out by the monitor calls MON 64 and MON 64.

If the <function code> is 20_8 or 24_8 (read status or read last status) the hardware status word is returned as the function value.

The various parameters are:

<function code>

specifies function to be performed according to the table on the following page.

<memory address>

designates a record area

<logical device number>

logical device number of the desired device unit.

<max words>

maximum number of words to read or write

<words read>

actual number of words read

The function codes are the same as in Table 9.1.

For function code 5 to 24_8 the parameters <memory address>, <max word> and <read word> are dummy parameters but must be specified in the CALL statement.

9.2.2.1 TANDBERG MAGNETIC TAPE

Using Tandberg 7 tracks magnetic tape, the density and parity may be set by program as follows:

ISTAT = MAGTP (<function code>, <memory address>, <logical device number>, <value>, <words read>)

<function code> = 23_8 set density and parity:

<value> = 0: 800 BPI, odd parity
<value> = 1: 556 BPI, odd parity
<value> = 2: 200 BPI, odd parity
<value> = 4: 800 BPI, even parity
<value> = 5: 556 BPI, even parity
<value> = 6: 200 BPI, even parity

The <memory address> and <words read> parameters are dummy for <function code> = 23_8.

If density/parity is set by the monitor call MAGTP, this density/parity will be used until it is changed again with the monitor call MAGTP. Default value of density/parity is VALUE = 0.

If an error occurs when <function code> is not 20_8 or 24_8 the returned function value will be as follows:

Bit 0: Tape on line
Bit 1: Write enable ring present
Bit 2: Tape standing on load point
Bit 3: CRC error/fatal error
Bit 4: LRC error/soft error
Bit 5: Control or modus word error. Trying to write on protected tape, reversing tape at load point, tape unit not on line, etc. Action inhibited.
Bit 6: Bad data block. An error detected.
Bit 7: End of file detected.
Bit 8: The search character is detected
Bit 9: End of tape detected.
Bit 10: Word counter not zero
Bit 11: DMA error
Bit 12: Overflow (in read)
Bit 13: Tape busy or formatter busy
Bit 14: Formatter busy
Bit 15: Interrupt when formatter ready.

By <function code> 20_8 and 24_8 the returned status code will be:

Bit 0: Ready interrupt enabled (cleared by the interrupt)
Bit 1: Error interrupt enabled (cleared by the interrupt)
Bit 2: Device active
Bit 3: Device ready for transfer
Bit 4: Error inclusive or of bits 5, 6, 7, 8, 9, 11 and 12.
Bits 5-15: As indicated in the table above.

9.2.2.2 STATUS CODE FOR HEWLETT-PACKARD MAGNETIC TAPE

A register - Bit position set to 1:

Bit 0: Ready interrupt enabled (cleared by the interrupt)

Bit 1: Error interrupt enabled (cleared by the interrupt)

Bit 2: Device active

Bit 3: Device ready for transfer

Bit 4: Inclusive or of error bits (6, 9, 10, 11 and 12) or if a reverse command is tried when the unit is at load point.

Bit 5: Write enable ring present

Bit 6: LRC error

Bit 7: EOF detected

Bit 8: Load point (this status remains also after the first forward command after load point is detected)

Bit 9: EOT detected

Bit 10: Parity error

Bit 11: DMA error

Bit 12: Overflow in read

Bit 13: Density select 1 = 800 BPI, 0 = 556 or 200 BPI

Bit 14: Magnetic tape unit ready (selected, on line and not rewinding)

Bit 15: Bit 15 loaded by previous control word

9.2.2.3 STATUS WORD FOR CASSETTE TAPE PHILIPS

A register - Bit position set to 1:

Bit 0: Ready for transfer, interrupt enabled

Bit 1: Error interrupt enabled

Bit 2: Device active

Bit 3: Device ready for transfer

Bit 4: Inclusive OR of errors, subflags 0, 1, 4, 5

Bit 5: Write enable

Bit 6: Cassette side (A = 1, B = 0)

Bit 7: Bit clock

Bit 8: Read fail

Bit 9: Sync fail

Bit 10: Not used

Bit 11: Not used

Bit 12: Drive Fail

Bit 13: Write Protect Violation

Bit 14: Beginning or end of tape

Bit 15: Not used

9.3 *FLOPPY DISK*

The SINTRAN III system can handle a maximum of two floppy disk controllers with three drives (units) on each controller.

Before a diskette can be used, it must be formatted using the @DEVICE-FUNCTION command.

9.3.1 *Floppy Disk as File Directory*

A diskette may be used as a file directory. Due to the relatively small storage capacity, 154K words, it is recommended that there is only one user on each diskette. The sequence of commands to make a file directory on a diskette is:

1. Insert a formatted diskette in a free floppy disk drive.

2. Log in as user SYSTEM on a terminal and give the following commands:

   @CREATE-DIRECTORY
   @ENTER-DIRECTORY
   @CREATE-USER
   @GIVE-USER-SPACE (maximum 148 pages)

3. Log out and log in again using your own user name. The diskette may now be used as a file directory.

4. When the user has finished his work with the diskette, type the command:

   @RELEASE-DIRECTORY

   and remove the diskette from the Floppy disk before logging out.

   The next time the diskette is to be used, the command sequence will be:

   1. Insert the diskette into a free floppy disk drive and log in using your own user name.

   2. Type the command @ENTER-DIRECTORY and the diskette may be used as a file directory.

3. When the user has completed the work with the diskette, type the command

   @RELEASE-DIRECTORY

   and remove the diskette from the floppy disk unit.

9.3.2 *Floppy Disk Used as a Sequential Peripheral File*

The floppy disk units may be used as sequential peripheral files such as paper tape reader, paper tape punch, magnetic tapes, etc. The different floppy disk units (drives) must then be given a name using the command:

@SET-PERIPHERAL-FILE

The logical device numbers of the floppy disks are as follows:

Device:	Unit No.:	Log. dev.(octal):	Log. dev.(decimal):
Floppy disk contr. 1	0	1000	512
Floppy disk contr. 1	1	1001	513
Floppy disk contr. 1	2	1002	514
Floppy disk contr. 2	0	1003	515
Floppy disk contr. 2	1	1004	516
Floppy disk contr. 2	2	1005	517

Reading or writing data is accomplished using the monitor calls INBT/OUTBT from MAC/NORD-PL, and the READ/WRITE, INPUT/OUTPUT or INCH/OUTCH statements from FORTRAN.

When using the monitor call OUTBT to write onto the floppy disk, the last record (block) will be transferred to the unit when the file is closed, but no EOF mark will be written. The last record (block) will be filled with zeros. To write an EOF mark on the diskette, the command @DEVICE-FUNCTION should be used.

9.3.3 *Accessing the Floppy Disk Using the Monitor Call MAGTP (MON 144)*

The monitor call MAGTP may be used to position (set the disk address) the diskette and to transfer data to/from the diskette.

The available functions of the monitor call MAGTP are the same as listed in Table 9.1.

A description of some of the available functions:

Function Code: (octal)	Name:
12	Write EOF mark. Write a block in a special format to make the block as an EOF mark. The disk address is incremented by one.
13	Rewind. The disk address is set to zero.
15	Backspace one record. The disk address is decremented by one.
16	Advance one record. The disk address is incremented by one.
40	Set disk format. The available formats are: Format 0: 256 words per sector. 8 sectors per track. Format 1: 128 words per sector. 15 sectors per track. Format 2: 64 words per sector. 26 sectors per track. (The standard format used by Norsk Data A.S. is format 0.)
41	Format the diskette. All data on the diskette is overwritten and the diskette is formatted (new addresses are written).

The status word of the floppy disk has the following format:

Bit No.:	Bit Position Set to 1:
0	Interrupt enabled
1	Not used
2	Device busy
3	Device ready for transfer
4	Inclusive OR of errors
5	Deleted record detected

Bit No.:	Bit Position Set to 1:
6	Read/write completed
7	Seek completed
8	Drive not ready
9	Write protect
10	Not used
11	Address mismatch
12	CRC error
13	Not used
14	Data overrun
15	Not used

9.4 *VERSATEC PLOTTER/PRINTER (DMA INTERFACE)*

The Versatec may be used in the same way as other line printers, using the monitor call OUTBT for printing from MAC/NORD-PL and the statements WRITE, OUTPUT and OUTCH from FORTRAN. The last characters outputted will be transferred to the Versatec when the file is closed.

The logical device number for Versatec plotter/printer is:

Device Name:	Log. dev. (octal):	Log. dev. (decimal):
Versatec controller 1	22	18
Versatec controller 2	23	19

The Versatec has two modes, print mode and graphic mode. When the Versatec file is closed, it is always set in print mode.

9.4.1 *Monitor Calls*

The monitor call MAGTP (MON 144) may be used to access the Versatec.

To prevent other users from accessing the Versatec unit when the call MAGTP is used, the background user may open the Versatec unit for sequential write with the @OPEN-FILE command, or the OPEN monitor call and later close it to allow other users to access the Versatec unit again.

From a real-time program, the monitor calls RESRV and RELES on the Versatec output I/O datafield, will establish the same effect as OPEN-FILE/CLOSE-FILE in timesharing/batch.

Calling Sequence:

FORTRAN: ISTAT = MAGTP (<function code> , <memory address>, <logical device no>, <max words>, <words read>)

Assembly:

```
LDA (PARLIST
MON 144
JAF ERR
,,,
PARLIST,
        FUNC
        MEMAD
        UNIT
        MAXWORDS
        WORDSREAD
```

Return: A register (IERR) = 0, no error
A register (IERR) ≠ 0, means error. Error code in A register (IERR). The error message may be written out by using the monitor calls MON 64 or MON 65.

Parameters:

<function code>: Legal values:

1	write one record
20_8	read status
21_8	clear versatec
24_8	read last status
30_8	set print mode
31_8	set graphic mode
32_8	give form feed

<memory address>Users buffer area

<log.device no> Logical device number of the specific versatec.

<max words> Number of words to transfer

<words read> Dummy

Versatec Status Word:

Bit No.:

0	Ready for transfer, interrupt enabled
1	Error interrupt enabled
2	Device active
3	Device ready for transfer
4	Inclusive or of errors (bits 6 and 7)
5	Not used
6	No paper
7	Plotter not on line
13	Plotter ready
8-12, 14-15	Not used, some bits may be set to one when the status register is read

9.5 *SINTRAN III/CAMAC COMMUNICATION*

The CAMAC system is a general purpose modular electronic instrumentation, standard for data handling, intended for applications requiring numerous and fast transfers of information (data, control) between various instruments and the computer. A CAMAC system contains one or more *crates* (CAMAC chassis) and each crate is separated into 24 stations where various kinds of CAMAC modules may be placed. For further documentation refer to the CAMAC-CC/NORD-10 Manual (ND-12.007).

SINTRAN III is able to handle 16 crates. Each crate may handle 16 grated LAM interrupts (1-16), plus one RT interrupt on level 13 which is identified and assigned as LAM Number 0.

By means of the monitor call (CONCT), different RT programs may be connected to different LAM interrupts on levels 10, 11, 12 and 13.

9.5.1 *Monitor Calls*

The following seven monitor calls have been implemented in SINTRAN III in connection with CAMAC (MON 146-154).

MON 146 INIT

CALL INIT (<flag>, <crate no>, <level>)

Initialized a crate for the appropriate level, i.e., clears data away and masks and writes COST (enables RT, ERROR and L for actual level).

Return: <flag> = 0, OK
<flag> = 24, NOT OK

MON 147 CAMAC

CALL CAMAC (<value>, <flag>, <crate no>, <station>, <sub address>, <function>)

Operates the CAMAC (executes NAF). If <function> is clear, the contents of COST are returned in <value>.

If <function> is read, data is returned in <value>.

For clear and read functions <flag> must be $\geqslant 0$ on entry.

If <function> is write, data must be in <value> and <flag> must be > 0 before the call.

Return: <flag> = 0, OK
<flag> = 24, NOT OK

If bit 15 in <crate no> is set equal to one, <value> is treated as integer instead of floating.

MON 150 GL

CALL GL (<value>, flag>, <crate no>)

Reads graded LAM status. The status is returned in <value>.

If <crate no> = 1 on the entry, the last IDENT code is returned as an integer in <value>.

Return: <flag> = 0, OK
<flag> = 24, NOT OK

MON 151 LMASK

CALL LMASK <value>, <flag>, <crate no>)

Reads and writes MASK.

If <flag> ≥ 0, the mask is read and returned in <value>.
If <flag> < 0, the contents of <value> are written to the mask.

Return: <flag> = 0, OK
<flag> = 24, NOT OK

MON 152 CONTR

CALL CONTR(<value>, <flag>, crate no>)

Reads and writes COST (control/status register)

If <flag> ≥ 0, the COST is read and returned in <value>.
If <flag> < 0, the contents of <value> are written to COST.

Return: <flag> = 0, OK
<flag> = 24, NOT OK

MON 153 IOXN CALL IOXN (<value>, <flag>, <device no>)

Issues direct IOX commands.

If <flag> ⩾0, an output transfer is executed and the information is returned in <value>.
If <flag> < 0, an output transfer is executed and the output information must be in <value> before the call.

Return: <flag> = 0, OK
<flag> = 24, NOT OK

MON 154 ASSIG

CALL ASSIG (<logical no>, <grated LAM>, <crate no>)

Assigns a graded LAM in CAMAC ident table to logic number in logic number table.

Note that LAM 0 (<grated LAM> = 0) is used for high priority interrupts on level 13.

The calls CAMAC, 6L, LMASK and CONTR handle their <value> parameters as an integer if bit 15 in the <crate no> parameter is set, otherwise, <value> is treated as a real number.

Note, for use with MAC, Standard FORTRAN parameter communication is applied.

9.6 *SINTRAN III/GRAPHICAL OUTPUT*

The monitor call MON 155 has been implemented in SINTRAN III in connection with graphic output. At the moment, the following devices are using this monitor call.

1. Graphic NORDCOM systems
2. Pen plotters
3. Tektronix display

The parameters are as follows:

CALL GRAPHIC (<x>, <y>, <n>, <dvn>, <func>)

where

<x> and <y>

are the floating point coordinates to the new point, relative to the current reference point.

<n>

is one of five integer codes described in the manual NORD PLOT PACKAGE for the graphic NORDCOM and pen plotter systems.

<dvn>

is the logical device number.

<func>

is a code to select one of three routines:

<func> = 1 GO PLOT

<func> = 2 GO PLOTS
(routine to establish reference point and/or clear a NORDCOM screen)

<func> = 3 GO NEWP
(routine to select one pen or one screen)

For further documentation see the manual NORD PLOT PACKAGE (ND-60.058).

9.6.1 *Tektronix Display*

For Tektronix display, the same monitor call can be used, but with different values for <func>.

<value> = PLOTT (<x>, <y>, <z>, <dvn>, <func>)

There are 14 functions, most of them giving <value> = 0 if device ready else device status, error value or function value.

<value> = 100 if function number of of range
= 20 if device not present
= 21 if device not reserved

<func> = 0: Normal <x>, <y>, <z>
<x> = horizontal deflection (0 - 4095)
<y> = vertical deflection (0 - 4095)
<z> = intensity deflection (0 - 7)
Origin in lower left corner
Negative <x>, <y>, or <z> means, use old value.

Error values:

101 = <x> > 4095
102 = <y> > 4095
103 = <z> > 7

<func> = 1: Read <z> intensity register
<value> = <z> register (old value)

<func> = 2: Read status register
<value> = STATUS register

<func> = 3: Reset device
DEVICE CONTROL register = 1

<func> = 4: Erase
The display is cleared

<func> = 5: Make hard copy

<func> = 6, 7: Set/reset non-store mode

<func> = 8, 9: Set/reset D mode (not used)

<func> = 10, 11: Set/reset cursor mode

<func> = 12, 13: Set/reset view mode

<dvn> is logical device number of Tektronix display (19 at present).

Note: The user himself is responsible for taking care of returned status error and eventual re-try.

After normal <x>, <y>, <z> output, the device is always ready. However, the functions "erase" and "make hard copy" take time.

9.6.2 *NORDCOM Monitor Calls*

9.6.2.1 THE TRACKER BALL MONITOR CALL (MON 156)

The tracker ball monitor call is used with four different function codes:

1. CALL TRACB (<value>, <function> [1]<, <tractor ball dvn>, <screen no>, <wait flag>)

   <value> = double word where the X and Y values are placed after read from the screen

   <value> = -1 if nothing has been read yet.
   <value> = -2 if anything wrong with the call.

   <function> = 1 means read values from the screen

   <tr.ball dvn> = logical device number of the tracker ball

   <screen no> = determines which screen to be read from (1, 2, 3, or 4)

   <wait flag> = wait flag = 0): if nothing had been read (read button not pushed)
   - 1 is returned to <value>.
   wait flag = 1): the monitor call is in wait status until the read button is pushed. Then the X and Y values are returned to <value>.

2. CALL TRACB (<value>, <function [2], <tractor ball dvn>, <SM 1 dvn>, <SM 2 dvn>)

   This monitor call is used once after restart of the SINTRAN III system to connect the actual selector modules to the actual tracker ball.

   <value> = double word.
   <value> = -2 if anything is wrong with the monitor call

<function>	=	2 means connect SM1 and SM2 to the tracker ball
<tr.ball dvn>	=	same as for function 1.
<tr.ball dvn>	=	logical device number of selector module number 1 to be connected to the tracker ball
<SM 2 dvn>	=	logical device number of selector module number 2 to be connected to the tracker ball.

3. CALL TRACB (<value>, <function [3], <tr.ball dvn>, <SM 3 dvn>, <SM 4 dvn>)

   Same as the previous function except that selector module 3 and 4 are now connected. SM parameter = 0 if no such selector module in the system.

4. CALL TRACB (<value>, <function [4], <tr.ball dvn>, 0, 0)

   This function is used to clear and enable the tracker ball. Interrupts are detected before this call is made.

```
INT,    LDA     (PAR 1
        MON     156
        LDA     (PAR 2
        MON     156
        LDA     (PAR 3
        MON     156
        MON     0
PAR1,   VALU1;  (2; (724; (714; (742
VALU1,  0; 0
PAR2,   VALU2; (3; (724; (770; (0
VALU2,  0; 0
PAR3,   VALU3; (4; (724; (0; (0
VALU3,  0; 0

)FILL
)LINE
```

   Note: The three selector modules in this example are on three different ACM.

For information about the NORDCOM System Software see the manual, A Description of the NORDCOM System.

APPENDICES

APPENDIX A

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
ABORT	Name	RT	Stop RT program	5.5
ABORT-BATCH	Batch Number	RT	Abort batch process	3.7, 3.6.6
ABORT-JOB	Batch No., User	RT	Abort batch job	3.7, 4.4
ABSET	Name, Sec., Min., Hr.	RT	Start RT program at time of day	5.5
ALLOCATE-FILE	File, Name, Page Addr. No. of pages	PUBLIC	Create and allocate file	3.5.1
ALLOCATE-NEW-VERSION	File, Name, No. of Pages, Page Addr.	PUBLIC	Create and allocate new version	
APPEND-BATCH	Batch No., Inp. File, Output File	PUBLIC	Append batch input and output files	3.7.4.3
APPEND-REMOTE	Computer, File Name	PUBLIC	Submit job for RJE	
BATCH		RT	Start batch process	3.7.3, 6.6
CC		PUBLIC	Comment (for use in batch jobs)	
CHANGE-BIT-FILE	Dir. Name, Block No.	SYSTEM	Change specified bit file block	
CHANGE-DIRECTORY-ENTRY	Device Name, Unit, F/R	SYSTEM	Change contents of directory	
CHANGE-OBJECT-ENTRY	User Name, Object No.	SYSTEM	Change specified object entry	
CHANGE-PAGE	Dir. Name	SYSTEM	Change specified page	
CHANGE-PASSWORD	New Password, Old Password	PUBLIC	Change the user's password	3.3.1.4

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
CHANGE-USER-ENTRY	Dir. Name, User No.	SYSTEM	Change specified user entry	
CLADJ	Time, Unit	RT	Adjust internal clock	5.5
CLEAR-PASSWORD	User Name	SYSTEM	Clear the user's password	
CLOSE-FILE	File No.	PUBLIC	Close opened file	
COMMUNICATION-STATUS	Line	PUBLIC	List status for line	
CONCT	Name, Line	RT	Connect RT program to interrupt line	5.5
CONNECT-FILE	File Name, File No., Access Mode	PUBLIC	Open file for access through given file number	3.4.1.2
CONTINUE		PUBLIC	Restart background program	3.3.4.1
COPY	Destin. File, Source File	PUBLIC	Copy file or device	3.5.4
COPY-DEVICE	Dest. Dev., Source Device	SYSTEM	Copy all pages from source device to destination device	6.2.4.2
COPY-DIRECTORY	Dest. Directory Name, Source Directory Name	SYSTEM	Copy all files in the source directory to destination directory	6.2.4.2
COPY-FILE	Destin. File, Source File	PUBLIC	Copy file or device	3.5.4
CREATE-DIRECTORY	Dir. Name, Device	SYSTEM	Create a directory entry	6.2.1

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
CREATE-FILE	File Name, No. of pages	PUBLIC	Create one or more versions of a file	3.5.1
CREATE-FRIEND	Friend Name	PUBLIC	Create one friend	3.5.2
CREATE-NEW-VERSION	File Name, No. of Pages	PUBLIC	Create one or more versions of a file	
CREATE-USER	Dir. Name, No. of Pages	SYSTEM	Create a new user	
DATCL		PUBLIC	Print current time and date on the terminal	3.3.7.1
DELETE-BATCH-QUEUE-ENTRY	Batch No., Input File, Output File	PUBLIC	Delete an entry in the batch queue	
DELETE-FILE	File Name	PUBLIC	Delete a file from directory	3.5.1
DELETE-FRIEND	Friend Name	PUBLIC	Delete a previously created friend.	
DELETE-REMOTE-QUEUE-ENTRY	Computer, File Name	PUBLIC	Delete an entry in remote batch queue.	
DELETE-USER	Dir. Name, User Name	SYSTEM	Delete user from specified directory	
DIRECTORY-STATISTICS	Dir. Name, Output File	PUBLIC	List statistics of entered directory	6.2.3
DMAC		RT	Assembler for system debugging	
DSCNT	Name	RT	Disconnect RT program	5.5
DUMP	File, Start, Restart	PUBLIC	Save background program	3.3.3.1
DUMP-BIT-FILE	Dir. Name, Block No. Output File	SYSTEM	Octal dump of one 16 words block	

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
DUMP-DIRECTORY-ENTRY	Device Name, Unit, F/R, Output File	SYSTEM	Octal dump of directory entry	
DUMP-OBJECT-ENTRY	User Name, Object No., Output File	SYSTEM	Octal dump of object entry	
DUMP-PAGE	Dir. Name, Page Addr. Output File	SYSTEM	Octal dump of one 1K page	
DUMP-USER-ENTRY	Dir. Name, User No. Output File	SYSTEM	Octal dump of user entry	
ENTER		PUBLIC	Dummy command, so that batch files can be run as mode jobs. For batch jobs, the command will log in the user.	3.7.5.1
ENTER-DIRECTORY	Dir. Name, Device	SYSTEM	Enter directory into system	6.2.2
ENTSG	Segment No., Pit, Int. Level, Start Address	RT	Program on segment will run as a direct task.	5.5
EXPAND-FILE	File Name, No. of Pages	PUBLIC	Expand File with specified number of pages	
FILE-STATISTICS	File Name, Output File	PUBLIC	List names and statistics of files with names that match the given one	3.5.3
FIX	Segment Number	RT	Fix segment in memory	5.5
FIXC	Segment Number, First	RT	Fix segment in contiguous memory area	5.5
GET-RT-NAME	Octal Address	RT	Convert address of RT description to name	5.6

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
GIVE-USER-SPACE	Dir. Name, User Name, No. of Pages	SYSTEM	Give user space in directory	
GOTO-USER	Octal Address	PUBLIC	Start background program	3.3.4.2
HELP		PUBLIC	List all existing commands	
HOLD	Time, Unit	PUBLIC	Let the program wait for some time	
INITIAL-COMMAND	Command	SYSTEM	Set a command, for example, ENTER-DIRECTORY, to be executed at system start	
INIT-ACCOUNTING	Desired, Max.	SYSTEM	Initialize and start accounting	6.5.1
INTV	Name, Time, Unit	RT	Make RT program periodical	5.5
IOSET	Unit, R/W, Program Name, Control	PUBLIC	Reset device, clear buffers	
LIST-ACCOUNT	Input, Oper. Inp. File, Flag, Accfi., Desir., Max., Dumpfi, Listfi, Output	PUBLIC	List 3 tables from account file	6.5.2
LIST-BATCH PROCESS		PUBLIC	List batch process	3.7.4.1
LIST-BATCH-QUEUE	Batch Number	PUBLIC	List contents in batch queue	3.7.4.2
LIST-DIRECTORIES-ENTERED	Dir. Name, Output File	PUBLIC	List names of directories entered	6.2.3
LIST-EXECUTION-QUEUE		PUBLIC	List execution queue	5.6

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
LIST-FILES	File Name, Output File	PUBLIC	List names of files	3.5.3
LIST-FRIENDS	Friend Name, Output File	PUBLIC	List names of friends with names that match specified friend name	3.5.3
LIST-OPENED-FILES	Output File	PUBLIC	List numbers and names of opened files	
LIST-REMOTE-QUEUE	Computer	PUBLIC	Dist. remote job queue	
LIST-RT-DESCRIPTION	RT Name	PUBLIC	List information about segment	5.6
LIST-RTOPENED-FILES	Output file	PUBLIC	List number and files opened by RT programs	
LIST-SEGMENT	Segment Number	PUBLIC	List information about segment	5.6
LIST-TIME-QUEUE		PUBLIC	List time queue	5.6
LIST-USERS	Dir. Name, User Name, Output File	PUBLIC	List names of all users with names that match the specified name	3.5.3
LOAD-BINARY	File Name	PUBLIC	Load and start)BPUN format	3.3.5.1
LOCAL		PUBLIC	Disconnect remote connection	3.3.6.2
LOGOUT		PUBLIC	Logout user and release terminal	
LOOK-AT	Space Reference	PUBLIC	Examine (and change) locations	5.4

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
MAG-TAPE-FUNCTION	Function, Unit, No. of Records	PUBLIC	Magnetic tape handling	3.8.3
MEMORY	Lower Bound, Upper Bound	PUBLIC	Define dump area	3.3.3.2
MODE	Input, Output	PUBLIC	Change command I/O device	3.3.8
NORD-50		PUBLIC	NORD-50 communication	
OPEN-FILE	File Name, Access	PUBLIC	Open file for specified access	3.4.1.1
PLACE-BINARY	File Name	PUBLIC	Load)BPUN format	3.3.5.2
PRIOR	RT Name, Priority	RT	Set priority of RT program	5.5
PRLS	Log, Unit, R/W	RT	Release unit from RT program	5.5.
PRSRV	Unit, R/W, Name	RT	Reserve unit for RT Program	5.5
RECOVER	File Name	PUBLIC	Start background program	3.3.2.1
REGENERATE-DIRECTORY	Directory Name	SYSTEM	Regenerate specified directory	
RELEASE-DEVICE-UNIT	Device Name, Unit	PUBLIC	Release device unit reserved by RESERVE-DEVICE-UNIT	
RELEASE-DIRECTORY	Directory Name	SYSTEM	The directory may be removed	6.2.2
RELEASE-FILE	Device Name	PUBLIC	Release specified unit from terminal	
REMOTE	Line Number	PUBLIC	Connect terminal to remote processor	8.4

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
REMOTE-LOAD	Load File, Bootstrap Address, Line No.	PUBLIC	Initialize remote load by bootstrap	8.5
RENAME-DIRECTORY	Old Dir. Name, New Dir. Name, Device Name, Unit, F/R	SYSTEM	Rename directory on specified device	
RENAME-FILE	Old File Name, New File Name, New Type	PUBLIC	Change the name or type of a file	
RENAME-USER	Dir. Name, Old User Name, New Name	SYSTEM	Change name of user in specified	
RESERVE-FILE	Device Name	PUBLIC	Reserve specified unit for terminal	
RESERVE-DEVICE UNIT	Device Name, Unit	PUBLIC	Reserve device unit for special use	
RESTART-SYSTEM		SYSTEM	Simulates "Master-Clear" and "Load"	
RT	RT Name	RT	Start RT program	5.5
RTCLOSE-FILE	File Number	RT	Close file for RT programs	5.3
RTCONNECT-FILE	File Name, File No., Access Mode	RT	Open file with number for access by RT Programs	5.3
RTENTER		RT	Enter user RT	5.3
RT-LOADER		RT	Load and start RT loader	
RTOFF	RT Name	RT	Inhibit RT program	5.5
RTON	RT Name	RT	Allow RT program	5.5

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
RTOPEN-FILE	File Name, Access	RT	Open file for access by RT programs	5.3
SAVE-DIRECTORY	Dest. Device, Source Device	SYSTEM	Copy all files in the source directory to destination directory	
SCHEDULE	Device Number	PUBLIC	Reserve devices for batch jobs	3.7.5.2
SCRATCH	File Name, Access	PUBLIC	Open specified file as scratch file	
SET	RT Name, Time, Unit	RT	Start RT program in a given time	5.5
SET-AVAILABLE		SYSTEM	Each terminal is available for logging in	6.1
SET-BLOCK-POINTER	File No., Block No.	PUBLIC	Set byte point to first byte in block	
SET-BLOCK-SIZE	File No., Block Size	PUBLIC	Set block size of specified opened file	
SET-BYTE-POINTER	File No., Byte No.	PUBLIC	Set byte pointer to specified byte number	
SET-DEFAULT-DIRECTORY	Directory Name	SYSTEM	Set specified directory as default	6.2.2
SET-ERROR-DEVICE	Device No.	SYSTEM	Set logical number of the devices where error messages should come	
SET-FILE-ACCESS	File Name, Public Access, Friend Access, Owner Access	PUBLIC	Set file access modes	3.5.2
SET-FRIEND-ACCESS	Friend Name, Access	PUBLIC	Set access mode of friend	3.5.2

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
SET-PERIPHERAL-FILE	File Name, Device No.	SYSTEM	Define name of peripheral device	
SET-PERMANENT-OPENED	File No.	PUBLIC	File is permanently opened. If not closed by @Close-File-1	
SET-TERMINAL-FILE	File Name	SYSTEM	Define name of terminal	
SET-UNAVAILABLE		SYSTEM	Only terminal 1 is available	6.1
START-ACCOUNTING		SYSTEM	Start the accounting system	6.5.1
START-COMMUNICATION	Line No.	RT	Initiate communication on a line number	8.2.1
STATUS		PUBLIC	List register values	3.3.6.1
STOP-ACCOUNTING		SYSTEM	Stop accounting system	6.5.1
STOP-COMMUNICATION	Line No.	SYSTEM	Terminate communication	8.2.1
STOP– SYSTEM		SYSTEM	Simulate power fail	6.4.2
STOP-TERMINAL	Terminal No.	SYSTEM	Logout user and release terminal	6.4.1
TAKE-USER-SPACE	Directory, Name, User	SYSTEM	Take unused pages in specified directory	
TERMINAL-MODE	Yes/No	PUBLIC	Define the working mode of the terminal	
TERMINAL-STATUS	Term No.	PUBLIC	Print information on this terminal	6.4.1

Command:	Parameters:	Used By:	Short Description:	Refer to Chapter:
TEST-DIRECTORY	Directory Name	SYSTEM	Test directory for page conflicts	
TIME-USED		PUBLIC	Print time used	3.3.3.2
UNFIX	Segment	PUBLIC	Allow a segment to be swapped	5.5
UPDAT	Min.,Hour, Day, Month, Year	RT	Update clock and calendar	5.3
USER-STATISTICS	Div. Name, User Name, Output File	PUBLIC	List names and statistics of all users with names that match the specified name	3.5.3
WHERE-IS-FILE	Peripheral Name	PUBLIC	Is peripheral free or reserved?	3.3.7.4
WHO-IS-ON		PUBLIC	Who has logged into the system?	3.3.7.3

APPENDIX B

APPENDIX B

Monitor Calls

Monitor Call:	Command:	Refer to Chapter:
0	RTEXT	3.6.1
1	INBT	3.4.2.1, 3.6.1
2	OUTBT	3.4.2.1, 3.6.1
3	ECHOM	3.6.1
4	BRKM	3.6.1
5	RDISK	3.6.1
6	WDISK	3.6.1
7	RPAGE	3.4.2.2, 3.6.1
10	WPAGE	3.4.2.2, 3.6.1
11	TIME	3.6.1
12	NOT USED	
13	CIBUF	3.6.1
14	COBUF	3.6.1
15–42	NOT USED	
43	CLOSE	3.4.3, 3.6.1
44	NOT USED	
45	DBRK	3.6.1
46	DBRK	3.6.1
47	SBRK	3.6.1
50	OPEN	3.4.1.3, 4.3.6.1
51	DMAC BREAKPOINT	3.6.1
52–61	NOT USED	
62	RMAX	3.6.1
63	NOT USED	

Monitor Call:	Command:	Refer to Chapter:
64	ERMSG	3.6.1
65	QERMS	3.6.1
66	ISIZE	3.6.1
67	OSIZE	3.6.1
70	CMND	3.6.2
71–72	NOT USED	
73	SMAX	3.6.1
74	SETBY	3.6.1
75	REABT	3.6.1
76	SBSIZ	3.4.2.2, 3.6.1
77	SETBC	3.6.1
100	RT	3.6.2, 7.7.1
101	SET	3.6.2, 7.7.1
102	ABSET	3.6.2, 7.7.1
103	INTV	3.6.2, 7.7.1
104	HOLD	3.6.2, 7.7.1
105	ABORT	3.6.2, 7.7.1
106	CONCT	3.6.2., 7.7.1
107	DSCNT	3.6.2, 7.7.1
110	PRIOR	3.6.2, 7.7.1
111	UPDAT	3.6.2, 7.7.1
112	CLADJ	3.6.2, 7.7.1
113	CLOCK	3.6.2
114	TUSED	3.6.2
115	FIX	3.6.2, 7.7.1
116	UNFIX	3.6.2, 7.7.1
117	RFILE	3.4.2.2, 3.6.2

Monitor Call:	Command:	Refer to Chapter:
120	WFILE	3.4.2.2.
121	WAITF	3.4.2.2
122	RESRV	7.7.1
123	RELES	7.7.1
124	PRSRV	7.7.1
125	PRLS	7.7.1
126	DSET	7.7.1
127	DABST	7.7.1
130	DINTV	7.7.1
131	ABSTR	7.7.1, 7.7.2
132	MCALL	7.7.2
133	MEXIT	7.7.1, 7.7.2
134	RTEXT	7.7.1
135	RTWT	7.7.1
136	RTON	7.7.1
137	RTOFF	7.7.1
140	WHDEV	7.7.1
141	IOSET	3.6.2
142	ERRMON	7.7.1, 7.7.2
143	RSIO	3.7.6
144	MAGTP	3.8.2
145	ACM	3.6.2
146	INIT	3.10.1
147	CAMAC	3.10.1
150	GL	3.10.1
151	LMASK	3.10.1

Monitor Call:	Command:	Refer to Chapter:
152	CONTR	3.10.1
153	IOXN	3.10.1
154	ASSIG	3.10.1
155	PLOTT	3.11, 3.11.1
156	TRACB	9.6.2.1
157	ENTSG	7.7.1
160	FIXC	7.7.1
161	INSTR	3.6.2
162	OUTST	3.6.2
163	WRQI	8.3
164	WSEG	3.6.2
165	DIW	
166	D0LW	
167	NOT USED	
170	US0	
171	US1	
172	US2	
173	US3	
174	US4	
175	US5	
176	US6	
177	US7	

APPENDIX C

APPENDIX C

LOGICAL DEVICE NUMBERS USED IN SINTRAN III

Octal Logical Device No.:	Decimal Logical Device No.:	Device Name:
0-77	56-63	Character devices
100-177	64-127	Mass Storage Files
200-277	128-191	Internal devices
300-377	192-255	Semaphores
400-477	256-319	Process Control Devices/Connect Devices
500-577	320-383	System Devices
600-677	384-447	SINTRAN III/SINTRAN III communication devices
700-777		NORDCOM devices
1000-1077		Extension of character devices
1100-1177		System devices
1200-1277		Not used
1300-1377		Not used

Octal Logical Device No.:	Decimal Logical Device No.:	Device Name:
0	0	Dummy Device (not used)
1	1	Teletype/Display 1
2	2	Tape reader 1
3	3	Tape punch 1
4	4	Card reader 1
5	5	Line printer 1
6	6	Synchronous Modem 1
7	7	Asynchronous Modem 1
10	8	Plotter 1
11	9	Teletype/Display 2
12	10	Tape Reader 2
13	11	Tape punch 2
14	12	Card reader 2
15	13	Line Printer 2
16	14	Synchronous Modem 2
17	15	Asynchronous Modem 2
20	16	Cassette drive 1
21	17	Cassette drive 2
22	18	Versatec Printer/Plotter 1
23	19	Versatec Printer/Plotter 2
24	20	Tektronix Display
25	21	Magnetic Tape 2 unit 2
26	22	Synchronous Modem 5
27	23	Synchronous Modem 6
30	24	Synchronous Modem 3

Octal Logical Device No.:	Decimal Logical Device No.:	Device Name:
31	25	Synchronous Modem 4
32	26	Magnetic Tape 2 unit 0
33	27	Magnetic Tape 2 unit 3
34	28	Magnetic Tape 2 unit 1
35	29	Line Printer 3
36	30	CDC Link
37	31	Teletype link
40	32	Magnetic Tape 1 unit 0
41	33	Magnetic Tape 1 unit 1
42-47	34-39	Teletype/Display 3-8
50	40	Card punch 1
51	41	Card punch 2
52-57	42-47	Asynchronous Modem 3-8
60-67	48-55	Teletype/Display 9-16

Logical Device Numbers 400 - 477

400-437	CAMAC interrupts or special process interface
440	Direct task level 6
441	Direct task level 7
442	Direct task level 8
443	Direct task level 9
450-461	CONNECT device

Logical Device Numbers 500-577

500	Internal Device for error-message RT-program
501	Semaphore for segment transfer
502	Disk 1 Datafield
503	RT-loader semaphore
504	General lock for file system
505	User-file-buffer lock
506	Object-file-buffer lock
507	RT-open-file-table lock
510	Device buffer 1 lock (disk 1)
511	Disk 1, unit 0, R-bit-file-buffer lock
512	Disk 1, unit 0, F-bit-file-buffer lock
513	Disk 1, unit 0, R-directory lock
514	Disk 1, unit 0, F-directory lock
515	DF1, file-transfer lock for disk 1, drum 1, drum 2, disk 2, big disk
516	DF2, open-file monitor call from RT-program datafield
517	RTFIL semaphore
520	Device buffer 2 lock (big disk)
521	Get device buffer lock
522	Disk 1, unit 2, R-directory lock
523	Disk 1, unit 1, F-directory lock
524	Disk 1, unit 1, R-bit-file-buffer lock
525	Disk 1, unit 1, F-bit-file-buffer lock
526	DF3, file-transfer lock for magnetic tape 1 unit 0, Magnetic tape 1 unit 1
527	Schedule semaphore
530	Accounting semaphore

531	DF 6 CDC link monitor call datafield
532	Batch process 1
533	Batch process 1
534	Batch process 2
535	Batch process 2
536	Batch process 3
537	Batch process 3
540	Internal Device Remote Batch IBM
541	Internal Device Remote Batch UNIVAC
542	Internal Device Remote Batch Honeywell Bull
543	Internal Device Remote Batch CDC
544	Drum 2, Datafield
545	Drum 2, bit-file-buffer lock
546	Drum 2, directory lock
547	Drum 2, device buffer lock
550	Drum 1, Datafield
551	Drum 1, bit-file-buffer lock
552	Drum 1, directory lock
553	Drum 1, device buffer lock
554	Disk 1, unit 3, R-bit-file-buffer lock
555	Disk 1, unit 3, F-bit-file-buffer lock
556	Disk 1, unit 3, R-directory lock
557	Disk 1, unit 3, F-directory lock
560	Magnetic tape 1, Datafield
561	Magnetic tape 1, unit 1, directory lock
562	Magnetic tape 2, unit 0, directory lock
563	Magnetic tape 2, unit 0, device buffer

564	Magnetic tape 1, unit 1, device buffer lock
565	Magnetic tape 1, unit 2, directory lock
566	Magnetic tape 1, unit 2, device buffer lock
567	CDFIE - CD link datafield
570	Disk 1, unit 2, R-directory lock
571	Disk 1, unit 2, F-directory lock
572	Disk 1, unit 2, R-bit-file-buffer lock
573	Disk 1, unit 2, F-bit-file-buffer lock
574	DF4, monitor call datafield for cassette
575	Cassette datafield
576	DF5, monitor call datafield for Versatec
577	Versatec datafield

Logical Device Numbers 1100-1127

1100	Big-disk datafield
1101	Big-disk, unit 0, directory lock
1102	Big-disk, unit 0, bit-file-buffer lock
1103	NORD-50 datafield
1104	Disk 2 datafield
1105	Disk 2, unit 0, R-directory lock
1106	Disk 2, unit 0, F-directory lock
1107	Disk 2, unit 0, R-bit-file-buffer lock
1110	Disk 2, unit 0, F-bit-file-buffer lock
1111	Magnetic tape 2 datafield
1112	Magnetic tape 2, unit 0, Directory lock
1113	Magnetic tape 2, unit 0, device buffer lock
1114	Magnetic tape 2, unit 1, directory lock

1115	
1116	DR7, monitor call datafield for magnetic tape 2
1117	Big disk, unit 1, directory lock
1120	Big disk, unit 1, bit-file-buffer lock
1121	Big disk, unit 2, directory lock
1122	Big disk, unit 2, bit-file-buffer lock
1123	Big disk, unit 3, directory lock
1124	Big disk, unit 3, bit-file-buffer lock
1125	VE2FI - Versatec controller 2
1126	DF8 - monitor controller 2
1127	Magnetic tape 2, unit 3, directory lock
1130	Disk 2, unit 1, R directory lock
1131	Disk 2, unit 1, F directory lock
1132	Disk 2, unit 1, R bit-file lock
1133	Disk 2, unit 1, F bit-file lock
1134	Disk 2 device buffer lock

APPENDIX D

APPENDIX D – ERROR MESSAGES

Error messages issued by system programs are not always self-explanatory. As they do not follow any standard format, at the moment, it is not always easy to find an explanation of their meaning. We have, therefore, collected all error messages from the most commonly used systems. These include SINTRAN III monitor, SINTRAN III file management system, SINTRAN III RT loader, BRL (binary relocating loader), standard FORTRAN, BASIC, MAC, NORD PL, and QED. The file management system list is especially important since all I/O ERROR codes from systems under SINTRAN III refer to this list.

ND is presently engaged in the task of defining standard user interfaces, including error messages, and we plan to implement these in the not too distant future. In the meantime, we hope these pages may be helpful.

D.1 *SINTRAN III MONITOR*

D.1.1 *Run Time Errors*

At run time, errors may be detected by the system. Most of the errors will cause the current RT program to be aborted and the error message:

aa.bb.cc ERROR nn IN rr AT ll ; tttt
xx yy

will be printed.

If the error occurs in a background program, the error message will be written on the corresponding terminal. For RT-programs, the error message will come to the error message terminal (usually Terminal 1).

The parameters have the following meaning:

aa.bb.cc	-	Time when the error message was printed. aa - hours bb - minutes cc - seconds
nn	-	Error number. For further explanation, refer to the list on the following page.
rr	-	Octal address corresponding to an RT program name.
ll	-	Octal address where the error occurred.
tttt	-	Explaining text.
xx,yy	-	Numbers carrying additional information about the error. One or both numbers can be omitted. For further explanation refer to the list on the following page.

Example: @01.43.32 ERROR 14 IN 16105 AT 114721;
OUTSIDE SEGMENT BOUNDS

In case of a segment transfer error, an additional message TRANSF! will be given.

D.1.2 *Run Time Error Codes*

Error Code	Meaning	xx	yy	Program Aborted
00	Illegal monitor call			yes
01	Bad RT program address			yes
02	Wrong priority in PRIOR			yes
03	Bad memory page	page no.		
04	Ring protect			yes
05	Memory protect			yes
06	Batch input error	error no.		yes
07	Batch output error	error no.		yes
08	Batch system error	error no	L-register	yes
09	Illegal parameter in CLOCK			yes
10	Illegal parameter in ABSET			yes
11	Illegal parameter in UPDAT			yes
12	Illegal time parameters			yes
13	Page fault for non-demand			yes
14	Outside segment bounds			yes
15	Bad segments in MCALL/MEXIT			yes
16	Bad segment in FIX/UNFIX			yes
19	Too byte segment			yes
20	Disk/drum transfer error	Hardware device no.	unit	no (aborted if segment transfer)
21	Disk/drum transfer error	disk address	hardware status	no
22	False interrupt	level		no

Error Code	Meaning	xx	yy	Program Aborted
23	Device error	hardware device no.	hardware status	no
24	Internal interrupt		bit no.	yes
26	Mass storage time-out			no
27	Error in CONCT			yes
28	FORTRAN I/O error	File error no.		yes
29	MON 64 and MON 65	Error no.	(See NORD File System)	yes
30	Divide by zero			yes
31	Permit violation			yes
32	Ring violation			yes
33	Illegal instruction			yes
35	RT-FORTRAN stack error			yes
36	Privileged instruction			yes
37	IOX error	Address	Level	no
38	Memory Parity	PEA-reg.	PES-reg	yes
39	Memory out of range	PEA-reg.	PES-reg.	yes
40	Power fail			no

D.2 *SINTRAN III FILE SYSTEM*

D.2.1 *Error Codes Returned from Monitor Calls*

Error Code	Meaning
000	Not used
001	Not used
002	Bad File Number
003	End of File
004	Card Reader Error (Card Read)
005	Device Not Reserved
006	Not Used
007	Card Reader Error (Card not read)
010	Not used
011	Not used
012	End of Device (Time-Out)
013	Not used
014	Not used
015	Not used
016	Not used
017	Not used
020	Not used
021	Illegal character in parameter
022	No such page
023	Not decimal number
024	Not octal number
025	You are not authorized to do this
026	Directory not entered
027	Ambiguous directory name
030	No such device name

Error Code	Meaning
031	Ambiguous device name
032	Directory entered
033	No such logical unit
034	Unit occupied
035	Master block transfer error
036	Bit file transfer error
037	No more tracks available
040	Directory not on specified unit
041	Files opened on this directory
042	Main directory not last one released
043	No main directory
044	Too long parameter
045	Ambiguous user name
046	No such user name
047	No such user name in main directory
050	Attempt to create too many users
051	User already exists
052	User has files
053	User is entered
054	Not so much space unreserved in directory
055	Reserved space already used
056	No such file name
057	Ambiguous file name
060	Wrong password
061	User already entered
062	No user entered
063	Friend already exists

Error Code	Meaning
064	No such friend
065	Attempt to create too many friends
066	Attempt to create yourself as friend
067	Continuous space not available
070	Not directory access
071	Space not available to expand file
072	Space already allocated
073	No space in default directories
074	No such file version
075	No more pages available for this user
076	File already exists
077	Attempt to create too many files
100	outside device limits
101	No previous version
102	File not continuous
103	File type already defined
104	No such access code
105	File already opened
106	Not write access
107	Attempt to open too many files
110	Not write and append access
111	Not read access
112	Not read, write and common access
113	Not read and write access
114	Not read and common access
115	File reserved by another user
116	File already opened for write by you

Error Code	Meaning
117	No such user index
120	Not append Access
121	Attempt to open too many mass storage files
122	Attempt to open too many files
123	Not opened for sequential write
124	Not opened for sequential read
125	Not opened for random write
126	Not opened for random read
127	File number out of range
130	File number already used
131	No more buffer space
132	No file opened with this number
133	Not mass storage file
134	File used for write
135	File used for read
136	File only opened for sequential read or write
137	No scratch file opened
140	File not reserved by you
141	Transfer error
142	File already reserved
143	No such block
144	Source and destination equal
145	Illegal on Tape device
146	End of tape
147	Device unit reserved for special use

Error Code	Meaning
150	Not random access on tape files
151	Not last file on tape
152	Not tape device
153	Illegal address reference in monitor call
154	Source empty
155	File already open by another user
156	File already open for write by another user
157	Missing parameter
160	Two pages must be left unreserved
161	No answer from remote computer
162	Device cannot be reserved
163	Overflow in read
164	DMA error
165	Bad Datablock
166	CONTROL/MODUS word error
167	Parity error
170	LRC error
171	Device error (read-last-status to get status)
172	No device buffer available
173	Illegal mass storage unit number
174	Illegal parameter
175	Write-protect violation
176	Error detected by read after write
177	No EOF mark found
200	Cassette not in position

Error Code	Meaning
201	Illegal function code
202	Time out (no datablock found)
203	Paper fault
204	Device not ready

D.3 *SINTRAN III – REAL TIME LOADER*

D.3.1 *Error Diagnostics*

The RT loader divides the errors into the following three groups:

1. Syntax errors. Example: ILLEGAL PARAMETER TYPE. The given parameter is not the correct type (parameter types: octal number, file name, symbols). No restrictions in use of the RT loader after these errors.

2. Serious errors. Example: SE-ILLEGAL BRF CONTROL BYTE which means that a wrong BRF control byte is found in the BRF input stream. The loading operation may be started before the RESET–LOADER–command is used. All serious error-messages will be preceded by the characters "SE–".

3. Fatal errors. If there is something seriously wrong in the RT loader's tables or in one of the system tables (which the RT loaders uses) or some serious errors occur in a critical sequence (for example, updating the segment link), the error message FATAL RT–LOADER ERROR and the content of the registers will be printed. These errors should be reported to Norsk Data-Elektronikk.

D.3.2 *Error Messages*

Error Message	Explanation
INTERNAL RT LOADER ERROR	Error in the internal loader stack is detected.
TABLE FILLED	No more space available in the linking table.
RTFIL FILLED RT DESCRIPTION TABLE FILLED	
ILLEGAL CONTROL BYTE	The control byte in the BRF input stream is illegal.
ILLEGAL VIRTUAL ADDRESS	The virtual address is outside the prespecified limitations.
ZERO-RELATIVE AD ADDRESS	The program unit is relocated relative to address zero. Too old version of the compiler/ assembler is used.
LOADING AREA FILLED	The virtual address area for one of the segments is full.
PRIORITY ERROR	The program priority is outside the legal area.
CHECKSUM ERROR	Wrong checksum on a BRF program unit.
SOURCE LANGUAGE ERROR	The compiler has detected source language error and produced INHB byte to prevent execution of the program.
DOUBLE DEFINITION	
DATA ERROR	Attempted initialization of unknown COMMON area.
COMMON ERROR	COMMON area expanded.
UNDEFINED DATA	Reference to unknown COMMON area.
SEGMENT NOT AVAIL.	The referenced segment not available.
RS COMMAND MUST BE USED	Use the RS command before further loading is allowed.
SEGMENT OVERLAP	Virtual address overlap between segments has just been loaded.
NO ROOM ON SEGMENT FILE	

Error Message	Explanation
NO PRIORITY SPECIFIED IN < PROGRAM NAME>	PRIOR-byte after MAIN-byte missing. Default priority inserted.
NEGLECTING REFERENCES?	Undefined references exist.
ILLEGAL COMMAND	

D.3.3 *Description of the "Illegal Command" Message*

Command	Description
	Command unknown.
	Command illegal because of earlier error conditions (utilize the RS command).
AS	Illegal segment number.
CL/CS	No correspondance with earlier definition of the label.
CR	Illegal segment number.
DD	Segment area number illegal ($0 \leqslant DD \leqslant 3$).
DE	Specified entry point unknown. Specified entry point is an RT program name.
DP	Specified RT program unknown. Specified name is not classified as RT program.
DV	Illegal device number.
EP	Symbol specified is not an RT program name. Segment specified not existing. Overlap between the two segments specified.
ER	Illegal Address. Double definition of symbol.
ES	Illegal address.
EQ	Illegal segment number. Double definition of symbol. No address specified. Address specified equal to unknown symbol.
LA	Illegal segment number. Illegal address.
MD	Illegal device specification.
NS	Illegal segment number. Illegal protection ring ($0 \leqslant RING \leqslant 2$)
RF	Symbol already defined.
RT	Symbol already defined.

D.4 *BINARY RELOCATING LOADER*

D.4.1 *Error Messages*

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

Error Code	Meaning
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	Defined label (system error)
12	Illegal character in octal number

D.5 *STANDARD FORTRAN*

D.5.1 *Compiler Error Messages*

The error message will be written on either the line printer or the Teletype, depending on which is specified as the listing device. If the user has requested a listing of the program, error messages will be printed on the line following the erroneous line and, in certain cases, on the next line thereafter. The error messages are self-explanatory and are printed in the following format:

***ERROR IN <subprogram unit name> <label> + <displacement>
<error text>

where label denotes last statement number or 0 if none are encountered yet. Displacement denotes the number of statements beyond the labeled one in which the error occurred.

D.5.2 *FORTRAN Formatting Error Messages*

Error Code	Meaning	Type of Error F/I
71	Illegal character in format	F
72	Parantheses nested deeper than 5	I
73	Attempt to fetch character beyond format	F
74	Attempt to store character beyond format	F
75		
76	Argument error unidentified type specification (system error)	F
77		
78		
79		
80	Output record exceeds 136 characters	I
81	Format requires a greater input record	I
82	Input record exceeds 136 characters	I
83	Wrong parity in input field	I
84	Bad character in input field	I

Error Code	Meaning	Type of Error F/I
85	Integer overflow	I
86	Real overflow on input	I
87	Real underflow on input	I
88	Real overflow on output	I
89	System error	F
90	Too bit input record	F

F = Fatal, I = Informative

D.5.3 *Arithmetical Library Error Messages*

The error message is written as a combination of letters, for instance:

ddddd RUN ERR CH

This means that COSH erred in the neighbourhood of core address ddddd.

IN RT FORTRAN the error looks like

RUN ERR CH rrrrr

Here, rrrrr means the name of the RT program.

Error Code	Meaning
AA	Error in 8AXA 8AXA was called with negative base. Result set to zero.
AI	Error in 8AXI Base equal to zero and exponent negative. Result set to 1.0E99.
AT	Error in ATAN2 Both arguments equal to 0.0. Result set to 0.0.
CH	Error in COSH Argument greater than 2^{14}. Result set to 1.0E99.

Error Code	Meaning
CO	Error in COS Argument greater than 2^{14}. Result set to 0.0.
DI	Error in 8DIV Second argument equal to 0. Result set to ± 32767, depending on sign of first argument.
EX	Error in EXP. Argument greater than 2^{14}, ln2. Result set to 1.0E99.
GO	Argument error in a computed or assigned GO TO statement. Program returns to first label in the list.
IX	Error in 8IXI. Overflow in result. Result set to 32767.
LN	Error in ALOG, ALOG1, ALOG2. Argument less or equal to 0. Result set to -1.0E99.
SH	Error in SINH Argument greater than 2^{14}. Result set to SIGN (X) · 1.0E99.
SI	Error in SIN Argument greater than 2^{14}. Result set to zero.
SQ	Error in SQRT Argument less than zero. Result set to zero.

D.6 *BASIC*

D.6.1 *Basic Error Messages*

The message you may encounter when writing or running a BASIC program are listed here. These error messages may originate in different parts of the system. In system without mass memory, error messages are normally given as error codes.

D.6.2 *Compiling*

When you are writing statements the compiler will check for syntax errors. These error messages are denoted with error codes CE followed by an error number to be looked up in Compiler Error Messages. The system will now check the input character. If this is a question mark, the erroneous line will be printed with the errors marked.

D.6.3 *Run Time*

When executing programs the BASIC system may print error messages like: RE 3 IN LINE s. The error number may be looked up in Run Time Error Messages.

The arithmetic functions SIN, COS, LOG, etc. have a special error format documented in Mathematical Library Error Messages.

D.6.4 *Compiler Error Messages*

Error Code	Meaning
CE1	Illegal character in this context.
CE2	Minor arithmetic error, progably operator missing.
CE3	Something wrong with parentheses.
CE4	No line number or illegal line number.
CE7	Expected variable not found.
CE8	Expected number not found.
CE9	Illegal word in this context.
CE10	Only string variables legal in this context.
CE11	= used illegally or omitted.

Error Code	Meaning
CE12	"omitted or used illegally.
CE13	Illegal string format.
CE14	Mixed mode string – arithmetic.
CE15	No relational operator found in IF statement.
CE16	Word (GO TO, GOSUB, TO, THEN) expected, not found
CE17	No value for variable with FOR, LET
CE18	Illegal FOR looping variable.
CE19	Illegal use of CHR$ or SEG$.
CE20	Array must have an index in this context.
CE21	Array index not legal in this context.
CE22	No end of array subscript found.
CE23	Illegal array index format.
CE26	Program name too long.
CE30	Illegal use of files.
CE31	Illegal file terminator
CE32	FN-function used recursively.
CE33	Illegal format for FN-function.
CE39	CON legal only in execution code.
CE40	Array previously defined two-dimensional.
CE41	Array previously defined one-dimensional.
CE42	Only numeric arrays legal in this context.
CE43	Only two-dimensional arrays legal in this context.
CE44	Array dimensions not matching.
CE45	Illegal operator in MAT statement.
CE46	MAT multiply does not allow same array on both sides of assignment operator.
CE55	Print Using must start with string.

D.6.5 *Run Time Error Messages*

Error Code	Meaning
RE1	Out of numeric data for READ statement.
RE2	Out of string data for READ statement.
RE3	END not last statement.
RE5	Dividion by zero tried, overflow.
RE6	Undefined string variable used.
RE8	GOSUB nested too deeply, table full.
RE9	No return address with RETURN statement.
RE10	FOR – NEXT nested too deeply.
RE11	More NEXT than FOR.
RE12	More FOR than NEXT.
RE13	FOR–NEXT illegally nested, variables do not match.
RE14	FOR–NEXT loop illegally nested.
RE15	Increment 0 with FOR statement.
RE16	DEF FN and FNEND illegally nested.
RE17	Number of input data incorrect.
RE18	Input data terminated illegally.
RE20	Negative index illegal.
RE21	Index too big, overflow.
RE22	Array dimensions not matching.
RE23	Only square matrics with MAT IDN and MAT INV.
RE25	Illegal file number.
RE26	Illegal file name, file used illegally.
RE27	Program tried to read EOF.
RE28	ETB (end-of-file mark) read.
RE29	Tried to read/write sequentially in random file.

Error Code	Meaning
RE30	System out of core.
RE31	Too many core tables used, table full, use the command TABLE.
RE32	Input buffer overflow.
RE33	Core table full, probably too many errors in the program.
RE34	Illegal FN-name.
RE35	Statement reached illegally, only legal within multiple line DEF FN.
RE36	Illegal number of arguments in FN-function.
RE37	Undefined FN-function called.
RE39	Command CON used illegally.
RE40	Zero or negative margin illegal.
RE41	No margin with random file.
RE44	Illegal string size.
RE46	String too long.
RE47	MAT is used with disk arrays.
RE60	Print using not allowed with disk strings.
RE61	Print using format error.

D.6.6 *Mathematical Library Error Messages*

Error Messages	Function	Error condition:
RUN ERR AA	↑	For A ↑ B if A = 0 and B < 0. Result: A ↑ B = 0. For A ↑ B if B $\log_2$A ⩾2 ↑ 14. Result: A ↑ B = 1·E + 99
EXPONENT ROUNDED	↑	For A ↑ B if A < 0 and B not integer. Result: A ↑ B = 1/A ↑ ABS(INT(B)).
RUN ERR AI	↑	For A ↑ B if B $\log_2$A ⩾ 2 ↑ 14. Result: A ↑ B = 1·E + 99.
RUN ERR CO	COS	If argument ⩾ 2^{16} radians.
RUN ERR SI	SIN	Result set equal to 0.
RUN ERR EX	EXP	For EXP(x) if x/ln2 ⩾2^{16}. Result: set equal to 1·E99.
RUN ERR LN	LOG	Argument (x) less or equal to zero. Result is set equal to 1.E99.
RUN ER SQ	SQR	Argument < 0. Result is set to 0.

D.7 *MAC*

D.7.1 *Error Messages, Their Meaning and Action to Take*

(ERROR

Literal dumped too far from referencing instruction. Insert a)FILL command in the source program within the relative addressing range of the instruction. Reassemble.

)FILL MISSING

Insert a)FILL command before the)9END of the source program. Reassemble.

DOUBLE DEF

The symbol indicated has already been defined. This is not necessarily an error. The value of the symbols latest definition is used.

ENT DEFINED BEFORE

A symbol given in a)9ENT command was previously defined. Reassemble deleting either the symbol's previous definition (probably before the)9BEG command) or its inclusion in the)9ENT command.

EXT DEFINED

A symbol given in a)9EXT command was previously defined. Reassemble deleting either the symbol's previous definition or its inclusion in the)9EXT command.

EXT IN ADDRESS ARITHMETIC

An arithmetic expression included an external symbol. For example:

```
)9BEG
)9EXT     PER
          LDA I (PER +1          % ILLEGAL
```

Correct in the source program and reassemble.

EXT KILLED

An external symbol was included in a)KILL command. For example:

```
)9BEG
)9EXT     PER
)KILL     PER                    % ILLEGAL
```

Delete symbol from)KILL command in source program and reassemble.

FABS NOT FOUND

A symbol included in a)9FABS command was not previously defined. Define the symbol and continue assembly from the)9FABS command. (Symbols included in the)9FABS, 9ASF,)9AGL,)9RT, and)9LC commands must also be previously defined.)

IC

An illegal character was found in the object stream. The character is ignored.

IL ADR

An attempt was made to assemble or jump (via the ! command) into MAC itself. If the latter, try again with a legal address. If the former, correct the program and reassemble.

IL EXPRESSION

MAC has encountered an expression having double relocation or direct access to an external symbol. For example:

```
)9BEG
)9EXT     PER
A,        Ø
B,        Ø
B-A                              % LEGAL
B+A                              % ILLEGAL
          LDA     PER            % ILLEGAL
          LDA I   (PER           % LEGAL
```

Correct the source program and reassemble.

IL INST

Something is illegal about an instruction. Correct source program and reassemble.

IL MNE

An attempt was made to redefine one of MAC's built-in symbols. The attempted definition is ignored.

IL USE OF)ENT OR)EXT

A)9ENT or)9EXT was used somewhere other than before the first instruction or constant after a)0BEG. Fix the source program and reassemble.

IL USE OF COMMAND

Illegal use of a command. For example:

)KILL 5

Fix the source program and reassemble.

MACRO ERR NO ØØØØØ1

An attempt was made to redefine a macro. The macro definition is ignored.

MACRO ERR NO ØØØØØ2

The macro tables overflowed. The macro definition is ignored.

MACRO ERR NO ØØØØØ3
MACRO ERR NO ØØØØØ4

Both these error messages indicate the use of a symbol preceded by a $ within a macro body but not declared in the macro's formal parameter list. The macro definition is ignored.

MISSING PARAMETER

A macro call has insufficient parameters. The call is ignored.

OPTION IS MISSING

An option was "called" which was not included in MAC. Either, do not attempt to use the option or construct a version of MAC including the option. Reassemble.

POSS. FLT

There was possibly, a fault in assembly into the indicated location. Examine the location and correct if necessary.

RANGE EX.

An attempt was made to reference a location outside the addressing range of the referencing instruction. Fix the source program and reassemble.

TABL FULL

One of MAC's tables overflowed.

UDEF ENTRY

Add the appropriate)9ENT command to the source program and reassemble.

WHAT?

Illegal use of the)command. Give the correct command on-line and continue assembly.

<SYMBOL>=

A space preceded the = in the attempted definition of the symbol. Give the correct definition on-line and continue assembly.

D.8 *NORD-PL*

D.8.1 *Diagnostic Messages from the Compiler*

If the compiler detects an error, it prints a diagnostic message on the list device, preceded by some asterisks. If the list device is equal to zero, it prints instead on the communication device, the name of the last label and the number of lines after the label, followed by the diagnostic message. Usually the compilation will continue: however, in a few cases the compilation has to stop, returning control to the operator.

Message	Meaning
Error, ill. base	Error in a BASE statement
Error, buffer full	Too long statement or object instruction
Error in command	
Error in compiler	The compiler is destroyed, or maybe there is a bug in the compiler
Error, ill. condition	Error in the conditional compiling commands (LIB, SLIB, STLIB or NSLIB)
Error in data expression	Illegal operand or operator in a data expression
Error in decl.	Error in a declaration statement
Error, ill. disp.	Error in a DISP statement
Error, ill. elem.	A basic element is found in a place where it should not be
Error in elem.	An ill-formed basic element
Error, in else/fi	Bad nesting of THEN-ELSE-FI
Error, ill. else/fi/od	Bad nesting of THEN-ELSE-FI or DO-OD
Error in expr.	Error in an executable expression
Error in for	Error in a FOR statement
Error in if	Error in an IF statement
Error in I/O	I/O error signalled by the surrounding system
Error, no FI/OD	Unmatched THEN/ELSE or DO at the end of a subroutine

Message	Meaning
Error, no (	Missing left parenthesis in a data list
Error, ill. operation	This operation is not implemented in hardware, or non-corresponding operands.
Error in output	Error message from the surrounding system.
Error in relation	Ill-formed relation in an IF or FOR statement
Ill. statement	The statement is illegal in this context, or illegal in an expression
Error in subr.	Error in a SUBR statement
Error, table destroyed	Probably overlapping of compiled/assembled program and the compiler's symbol table.
Error, table full	Too many symbols in the program
Error, too complex	Too complex construction in an executable expression; the backtracking stack is filled.
Error, undefined	Undefined local symbols at the end of a subroutine.

D.8.2 *Diagnostic Messages from the Assembler*

Some errors can be detected at assembly time only, because the compiler does not keep track of memory address values. Below is a list of the most usual errors. For more information, see the manual "MAC User's Guide".

Message	Meaning
RANGE EX.	A label or variable is used too far away from where it was defined. It can for example, occur for GO to a label defined earlier, or at an OD statement.
POS FLT	Perhaps a label has been defined too far after the place where it was used. It can occur for a forward GO or in an ELSE, FI or OD statement. However, this message can occur if underfined symbol is part of a data expression. Then it can normally be ignored.
(ERROR	Too far between the filling in of literals. The compiler outputs a)FILL command at each RBUS statement. However, the programmer can put *)FILL commands inbetween.

D.9 *QUICK EDITOR*

D.9.1 *Error Messages*

While using QED, the user may encounter certain warning messages which indicate conditions that the user may not be aware of.

Following is a list of these messages and a description of what they mean.

Error Message	Meaning
STACK OVERFLOW STACK UNDERFLOW	These should never happen. If they do, then something is wrong with QED.
BUFFER 3/4 FULL	Indicates that the text buffer is nearly full. No more READ's accepted. This happens only when using a non-mass storage QED.
PARITY ERROR AT LINE n	Indicates that, during a READ operation, a character with wrong parity was read at line n. The bad character is replaced with a ?.
NO MORE BUFFER SPACE	No more space for the text buffer. Current operation aborted.
TRANSFER ERROR n	Transfer of a page to or from mass storage failed. The error word is n. Current operation is aborted.
FATA ERROR AT n	A situation exists which QED thinks is impossible or which it does not know how to handle. This error happens when the user tries to use lines longer than 128 characters. The location where the error occurred is n.
I/O ERROR n	A sequential I/O error has occurred. The error code is n. The current operation is aborted.

INDEX

****

A L P H A B E T H I C A L I N D E X OF THE MANUAL

SINTRAN III - USER'S GUIDE ND-60.050.06 .

****

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

COMMENT AND EVALUATION SHEET

SINTRAN III Users Guide
June 1976

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: _____

_____

_____

– we want bits of the future

A/S NORSK DATA-ELEKTRONIKK LØRENVEIEN 57 OSLO 5 NORWAY PHONE: 21 73 71 TELEX: 18284