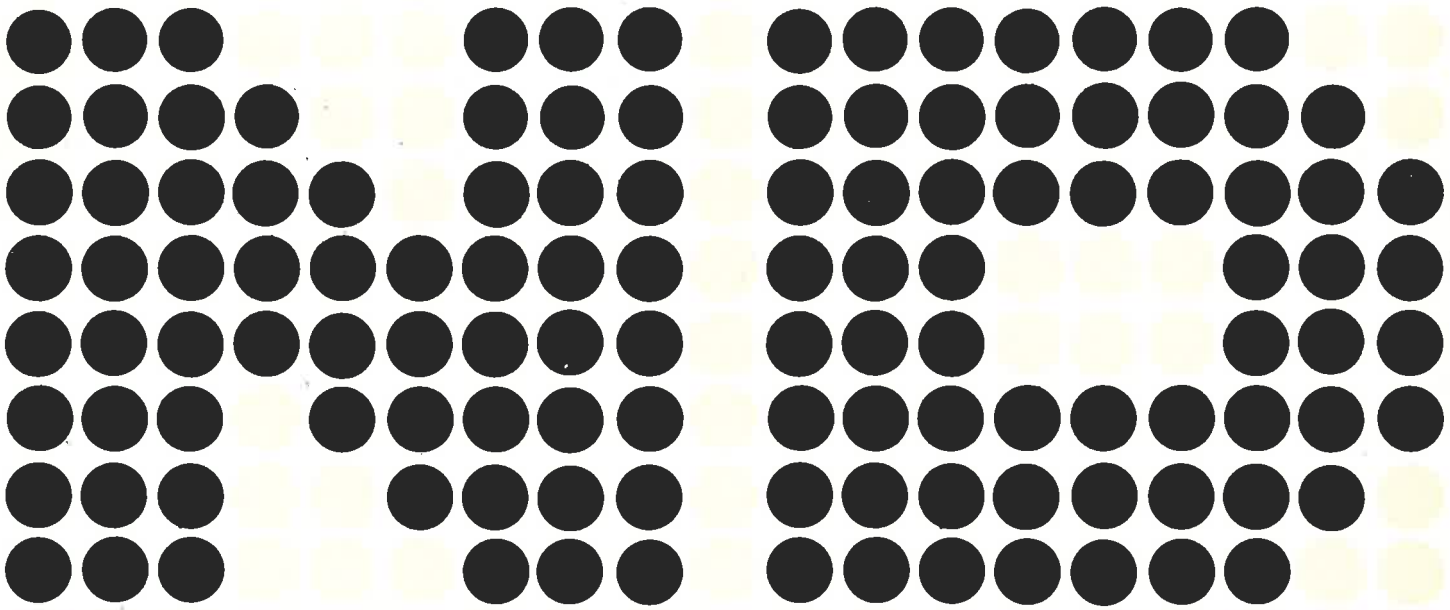
SINTRAN III

Users Guide

A/S NORSK DATA-ELEKTRONIKK



SINTRAN III

Users Guide

REVISION RECORD

Revision	Notes
1/75	Total revision, superceding all previous versions

ND-60.050.03
January 1975

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

TABLE OF CONTENTS

--ooOoo--

Chapters

Appendices

1 INTRODUCTION TO SINTRAN III

SINTRAN III is a multi-programming real-time operating system that supervises the processing of user programs submitted to a NORD-10 computer system. SINTRAN III controls the order in which user programs are executed and allocates the hardware and software resources they require. SINTRAN III also relieves the user from program control, input/output and housekeeping responsibilities by monitoring and controlling the input, loading compilation, run preparation, execution and output of user programs.

The system is highly modular and may be used for a wide range of NORD-10 configurations. Modularity allows memory resident systems of only 8K, expanding to mass storage systems including 256K main memory, discs, drums, etc., and connections to other NORD computers, thus allowing multi-processing systems.

The philosophy behind SINTRAN III makes it especially suited for:

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

and combinations of these processed concurrently. This, not at least because of the subsystems offered under SINTRAN III, helps to ease the users implementation of applications.

1.1 Features

SINTRAN III offers the user many important features, some of these are found elsewhere only in medium to large scale computer systems.

1.1.1 Operating Modes

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

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

1.1.2 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 input and operated upon so rapidly 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 ways of scheduling programs, external requests, program requests, operators requests and time scheduling. The programs may have a full range of execution 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.

1.1.3 Time-sharing Processing

In time-sharing the programmer interacts conversationally with the computer, receiving immediate response to his input. Many users on remote or local terminals can program on-line and make use of SINTRAN III subsystems. This type of interaction where each user receives an equal share of time in a round robin fashion can be used for program development, information retrieval, interactive problem solving, and many more applications where the user will be best serviced by accessing the system directly.

1.1.4 Batch Processing

Batch processing lets the user submit program jobs for computation to the computer. The computations are performed without interactions from the user. Each job contains all control commands, program statements and data required for its execution. Jobs are loaded through on-site devices such as card readers and tape readers.

Batch jobs are divided into two categories, local batch jobs and remote batch jobs. Local batch jobs are those compiled and executed on the local NORD-10 computer. Remote batch jobs are those compiled and executed on a host computer on which the local NORD-10 is a remote terminal.

SINTRAN III can handle both the local batch stream and the re-

mote batch stream concurrently. The local batch stream is scheduled in a first-in-first-out fashion. The local batch jobs share compute time with the time-shared programs. The remote batch processing runs as a low-priority RT program. SINTRAN III outputs the job locally on a device such as a line printer, disc file, or card punch.

1.2 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	following ANSI STANDARD FORTRAN and with ISA Real-time Extensions. The user can call subroutines written in NORD PL and MAC from his FORTRAN program. FORTRAN programs can be executed in all three modes of operation.
NODAL	interpreting higher-level interactive language especially suited for process control applications. NODAL can be executed in all three modes of operation and can call subroutines in NORD PL and MAC.
BASIC	interpreter following Dartmouth College 71 specification. From his BASIC program the user can call FORTRAN, NORD PL and MAC subroutines. Programs written in BASIC can be executed in time-sharing and batch modes.
NORD PL	medium level language especially suited for systems 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	assembly language and debugging package for the NORD computers.

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

1.3 Subsystems and Utility Programs

SINTRAN III is offered with many subsystems and utility programs which will be extremely efficient tools for the users of the system.

FILE SYSTEM	SINTRAN III offers the user of mass storage systems a general purpose file system for use of permanent files, scratch files, and peripheral device files. The system provides a very 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 especially put 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 specifications given in CODASYL DATA BASE TASK GROUP, APRIL 71 REPORT.
RT LOADER	enables the user to load RT programs into mass storage resident systems in binary relocatable format while real-time processing is running.
QED	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 into 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 applications 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 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.

1.4 Processing Efficiency

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

1.4.1 Virtual Memory and Dynamic Program Relocation

The SINTRAN III virtual memory makes it possible to run programs which are larger than the available main memory, or to utilize a main memory of 256K words with a program address space of only 64K words. The virtual memory consists of both main memory and swapping memory on disc.

Due to the overall ability of the memory management system, user programs and SINTRAN III subsystems are dynamically relocated to utilize the main memory most efficiently.

1.4.2 Ring Protection

By use of the four-mode ring protection system SINTRAN III offers the users an extremely efficient protect system. A program that 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 system programs from user programs and the system kernel from its subsystems. Ring 3 and ring 2 are used for the system kernel and subsystems, and ring 1 and ring 0 for user programs. The user programs are individually protected by use of the page protect system.

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 the SINTRAN III's kernel and its subsystems. These instructions are of the type which could be disastrous if executed by a user's program. For any on-line system with a larger number of potentially undebugged programs this is extremely important.

1.4.3 Interrupt System

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

1.4.4 High Data Throughput

High throughput of data is facilitated by the high-speed direct memory access channel. The channel is given an ingenuous solution thereby several high-speed devices can simultaneously share it, but although be given a total throughput equivalent to the maximum speed of the channel. There is no channel overhead time in switching between devices.

1.5 Hardware

The minimum hardware configuration required to run SINTRAN III is:

- NORD-10 standard CPU, including 8K words of main memory
- Console terminal
- Paper tape reader

1.5.1 Optional Hardware

- Main memory up to 256K words, both core and solid state memory in the same system.
- 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 discs. Up to 4 Cartridge disc units per controller and 5 or 10 M bytes per unit. Average access time 47.5 ms, 156K word/sec transfer rate.
- Moving-head disc packs. Up to 8 disc packs per controller and 37 or 74 M bytes capacity per pack. 48 ms average access time, and 600 K word/sec transfer rate.
- Fixed-head drum. One drum unit per controller with capacity from 64K to 104K words per unit. 10.5 ms average access 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. 300, 600 or 1000 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.

1.6 Example of Configurations

With the wide choice of optional equipment, many hardware configurations are possible. For example a small main memory resident system used for a process control application might appear as shown in Figure 1.1. In this system the paper tape reader is used for input and the Teletype for the operators communication with the system, while the process controller is connected directly to the process. Implementation may be eased by use of the DDC package.

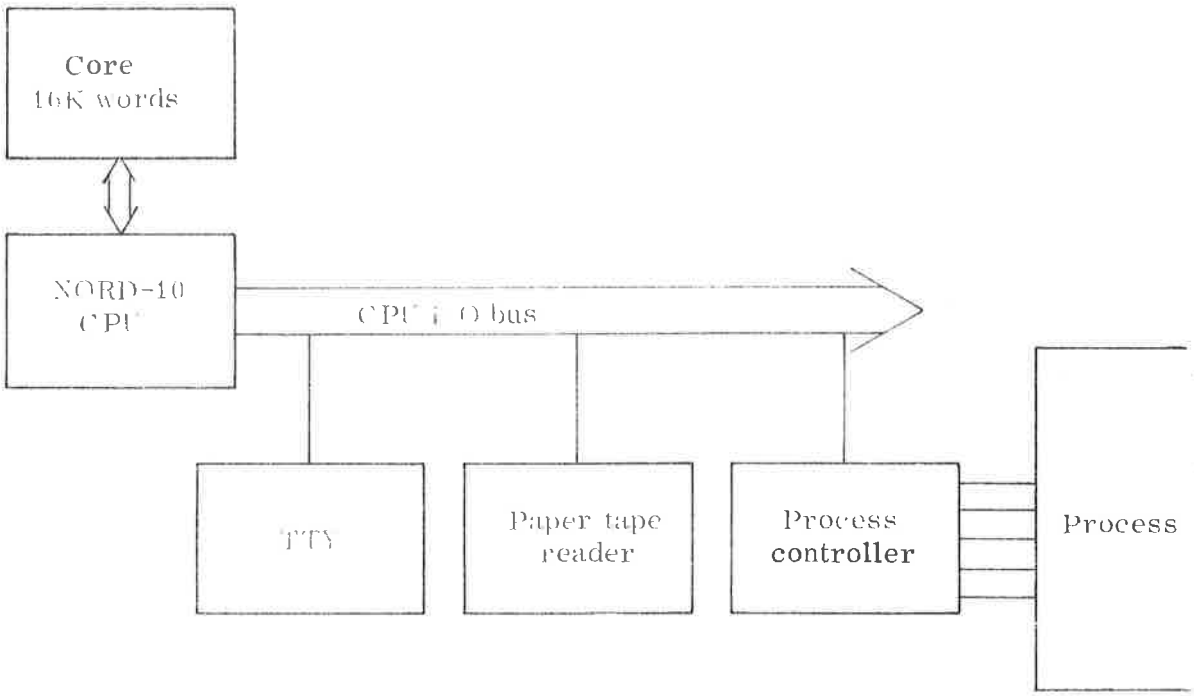


Figure 1.1: Small Process Control System

This system may easily be extended to the disc resident system shown in Figure 1. 2. The user programs developed for the system of Figure 1. 1 can be put directly into the system of Figure 1. 2. The system of Figure 1. 2 opens the possibilities for on-line program development in FORTRAN, NORD PL etc. in timesharing mode, while the process control programs are executed in real-time mode.

Programs can be written and compiled in FORTRAN, NORD PL etc. and by using the QED, the FILE SYSTEM and the other utility packages, the implementation will be greatly eased.

The display terminals are used for on-line communication in timesharing mode, the line printer for output and the disc storage for programs and data files. By putting on a synchronous modem control and a card reader, the system will also have the capabilities of a small batch system.

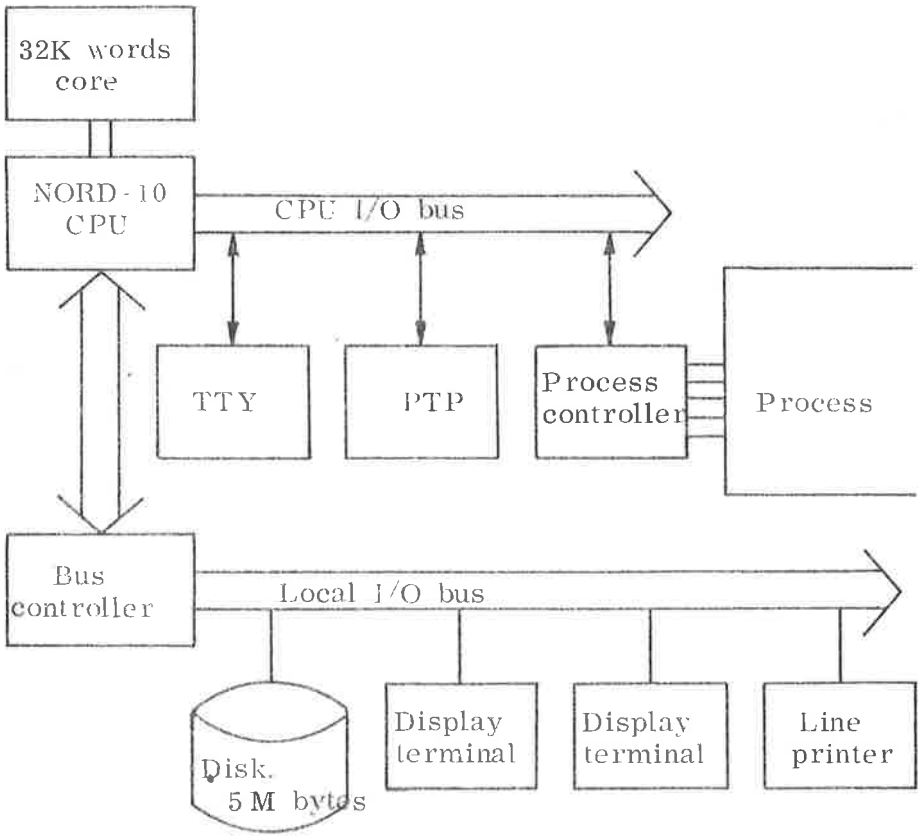


Figure 2. 1: Combined Process Control and Timesharing System

Configurations like Figure 1. 3 will be well suited for many on-line information retrieval and transaction oriented systems by running the data base system SIBAS under SINTRAN III. The 148M bytes disc store will hold the data files while the data entry terminals will be used for data input/output specified by the application.

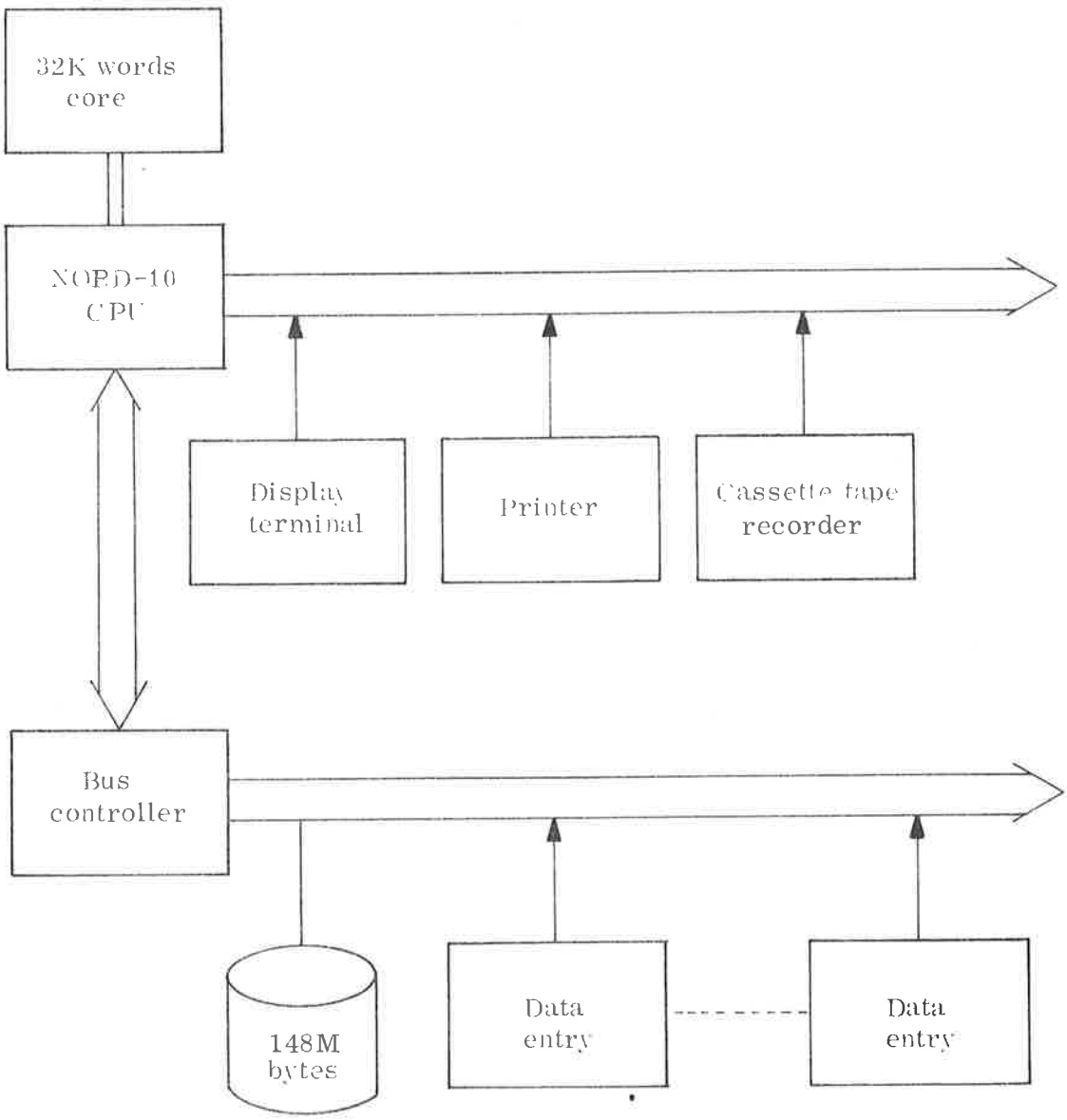


Figure 1. 3: Transaction Oriented System

A larger processing system is shown in Figure 1.4, consisting of two NORD-10 CPU's, standing back-up for each other. The NORD-10 CPU I is intended to be used in a transaction oriented system by using bus controller I and accoringly the 148M bytes disc, magnetic tape station, and the data entry terminals. The magnetic tape system will be used as a data base back-up medium. In addition it will have a modem connection to a host machine acting as a remote terminal.

The NORD-10 CPU II is intended for use in a process control system by using the bus controller II and III. In addition the NORD-10 CPU II will be used for program development and scientific on-line problem solving. By use of the dual bus switch each NORD-10 CPU I has provisions to control all three bus controllers and both the process control and transaction oriented application.

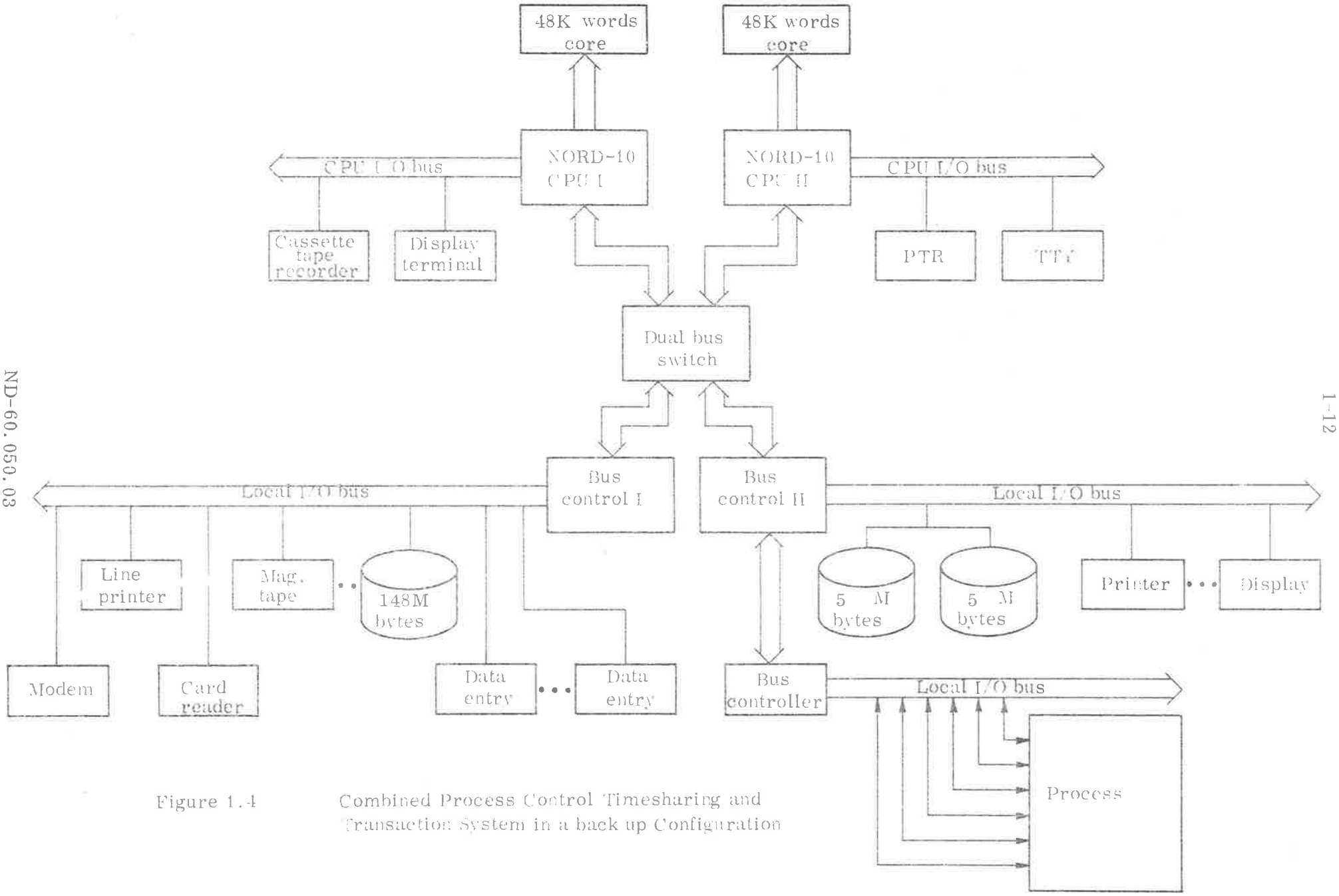


Figure 1.4 Combined Process Control Timesharing and Transaction System in a back up Configuration

2 THE COMPONENTS OF SINTRAN

2.1 Use of Hardware Facilities

2.1.1 The Interrupt System

The NORD-10 has 16 program levels. Each of these has a complete register set, so that change of levels needs only 0.9 μs.

SINTRAN uses the levels like this:

15	-	
14	-	Internal interrupt
13	-	Real-time clock interrupt
12	-	Input interrupt
11	-	Mass storage interrupt
10	-	Output interrupt
9	-	
8	-	
7	-	
6	-	
5	-	Monitor
4	-	
3	-	RT programs
2	-	
1	-	
0	-	

Level 14 is activated by monitor calls or by errors detected by hardware.

External interrupts on levels 10-13 start the proper driver routines.

2.1.2 Memory Management

The memory management system includes a paging system, a memory 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.

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

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

4) Inter-program protection, to prevent one program directly accessing another program on the SINTRAN III system. This is accomplished by means of the ring protection system.

The ring protection system has four levels. On one level it is not possible to access areas on higher levels.

On levels 2 and 3 privileged instructions can be executed, and these levels are used for the SINTRAN III system. User programs are on level 0 and 1.

2.2 Program Structure

The basic program concept is the segment. It is a contiguous area in the logical address space. In physical core it will be scattered because of the hardware paging system.

There are two types of segments:

- Non-demand segment, all of it must be in core before the program can be started.

- Demand segment, only part of it 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 program is too big to be in core at a time. The normal use is background processing.

The segment type is determined at load time.

An RT program can have one or two segments. The RT programs can share segments. This may be used in several ways:

- A segment may consist of a set of re-entrant sub-routines.

- A segment may consist of common data areas.

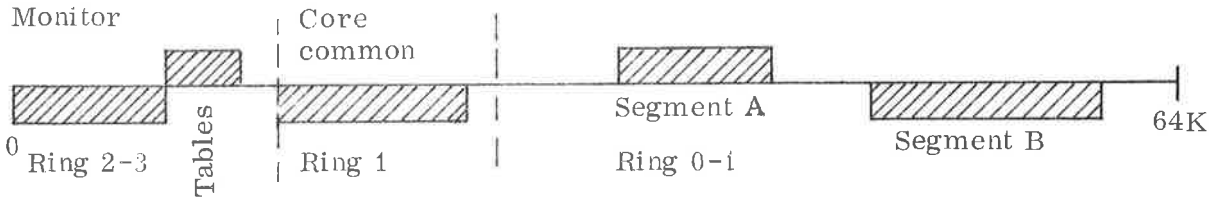
- A program may have its code on one segment and its data on the other.

One or both segments can be changed, using the monitor call MCALL, see Section 3.1.2. This can be used for program segmenting.

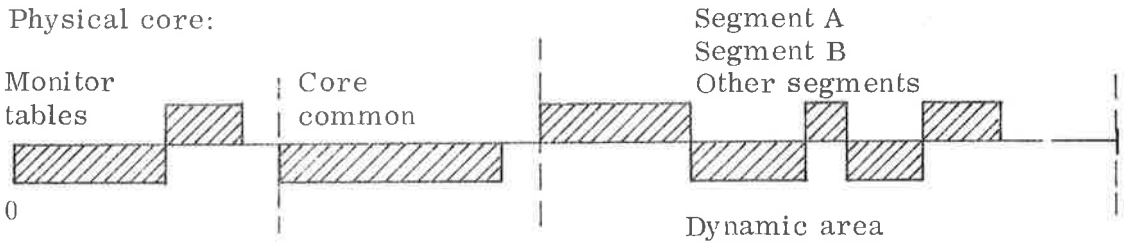
In addition to the segments the RT programs can also have access to a core resident common data area. This area is placed on protection ring 1, so that programs on ring 0 cannot access this area.

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

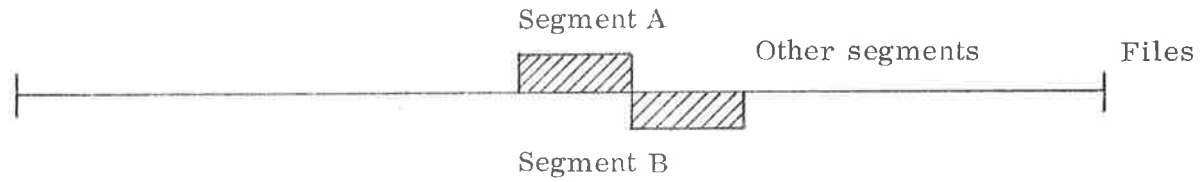
Logical address space:



Physical core:



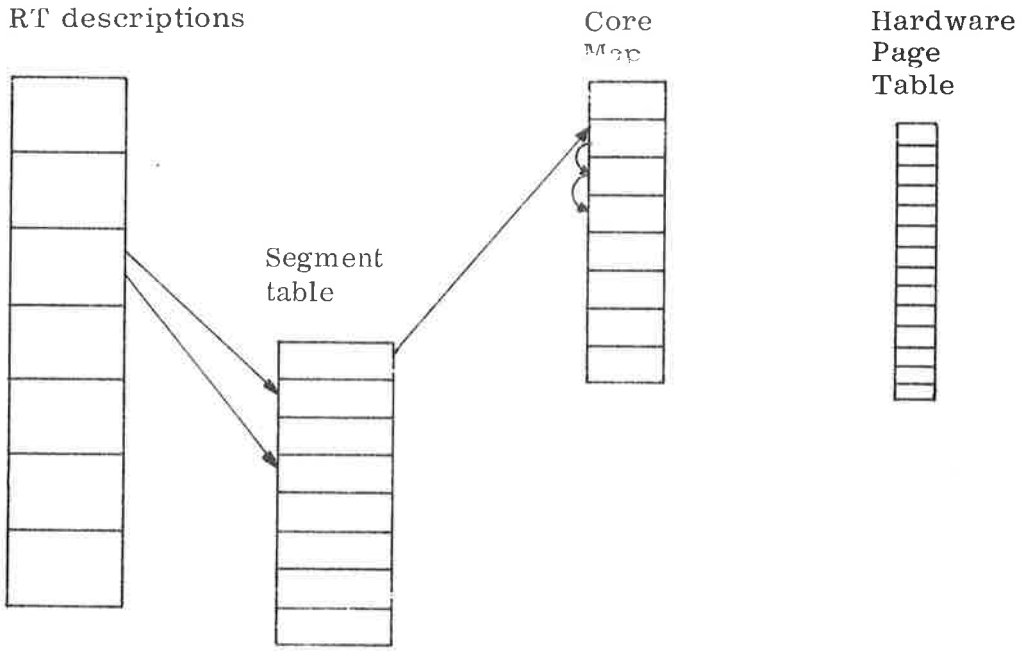
Mass storage:



2.3 System Tables

The system tables are placed in permanent core, not accessible from the user programs.

2.3.1 Program Tables



The RT description table has one element for each RT program existing in the system. An element consists of the following:

Loc. 1 : Link. This location is used for linking RT descriptions together to form the time queues.

Loc. 2 : The right half-word contains the priority of the RT program and the left half-word some status information flags.

Loc. 3 and 4 : This is a double precision number indicating the time when the RT program is to be (or was) scheduled.

Loc. 5 and 6 : This is the time interval if the RT program is to be executed periodically.

Loc. 7 : This contains the core address of the first program instruction.

Loc. 8 : Initial segment numbers. Two numbers, 0-254. If one of them is equal to zero, the program uses only one segment. If both are zero, it is a core resident system program.

Loc. 9 - 16 : Register save area.

Loc. 17 : Wait link - used for linking the programs waiting for resource.

Loc. 18 : Current segment numbers.

Loc. 19 : Current priority.

Loc. 20 : Reservation link - used for linking those resources reserved for this program.

The segment table has one element for each segment in the system. An element consists of five locations.

Loc. 1 : Segment link. All segments being in core at the moment and being allowed to be swapped out are linked together. It is used for the page-removal strategy.

Loc. 2 : Page-link-pointing to the segment's first page in the core map table. If this location contains zero, the segment has no pages in core.

Loc. 3 : Logical address space for the segment. Bits 0-5 contain the first logical page number and bits 6-11 contain the number of pages.

Loc. 4 : Mass storage address of the segment.

Loc. 5 : Flag bits:

Bit 0	=	segment all in core
Bit 1	=	demand segment
Bit 2	=	fixed in core
Bit 3	=	use inhibited
Bits 9-15	=	protect and ring bits to be used by the hardware page table.

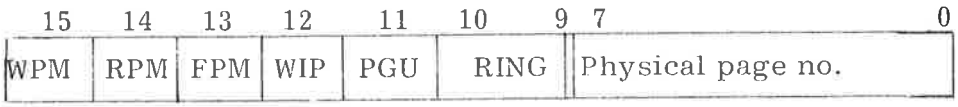
The core map table has one element for each physical page, containing information on what is in core at the moment. An element consists of three locations.

Loc. 1 : Page-link, linking together the pages belonging to a segment.

Loc. 2 : Logical page number (index into the hardware page table).

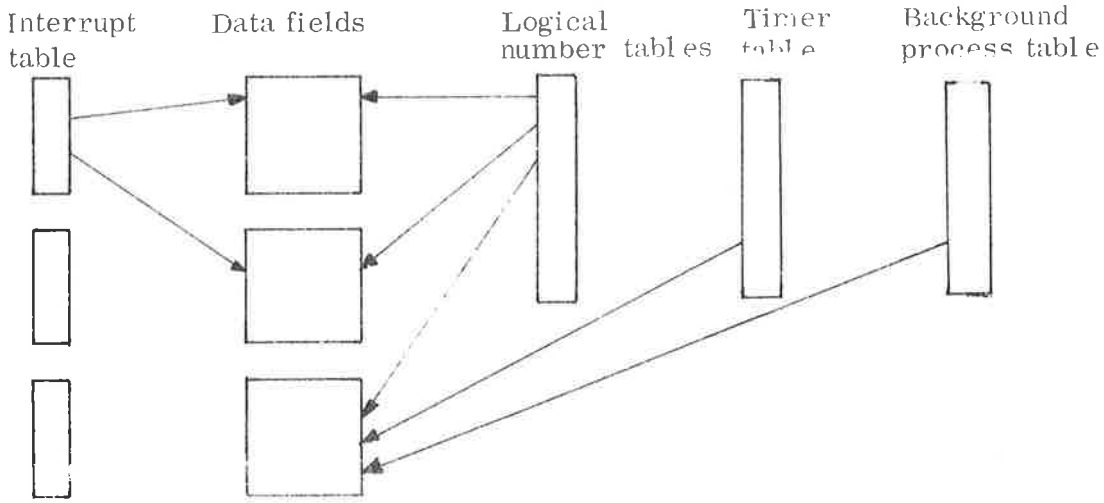
Loc. 3 : Contents to be put into the hardware page table.

The hardware page table consists of 64 high-speed registers, mapping the 64K logical address space. Each entry has the format:



Bits 9-10 : Ring number
Bit 11 : Page used
Bit 10 : Written in page
Bits 13-15 : Memory protect bits

2. 3. 2 Input/Output Tables



The interrupt table contains pointers to the data fields of the physical devices.

The logical number table contains pointers to the data fields of the logical units (internal or external units).

The data field table contains a data field for each logical unit. For some units a data buffer is associated.

The timer table contains pointers to the data fields of the devices needing a time-out check.

The background process table contains pointers to the data fields of the background terminals.

2.4 Parts of the Monitor

The following is a description of the parts of the monitor as shown in Figure 1.5 and 1.6.

2.4.1 Monitor Entry

The monitor works on a separate hardware level. This level may be activated from several other hardware levels. The task of Monitor Entry is to find out why the monitor level was activated and to transfer the program control to the appropriate monitor function.

The monitor may have several calls at the same time. A program calling the monitor may be interrupted by a program on a higher level also calling the monitor, and new calls may occur while the monitor is working.

A monitor call is performed by linking a representation of the call to the other calls which may be waiting. The monitor processes the elements of the chain, deleting each element and executing the corresponding function. When the chain is empty, the monitor level is "given up".

Some of the monitor functions imply the possibility of a different RT program to be activated on return from the monitor. These monitor functions set a flag to indicate that it should be further investigated before leaving the monitor level.

The monitor level may be activated for a number of reasons.

From high levels:

- A process interrupt has occurred, and an RT program should be scheduled for execution.
- An I/O transfer is finished, and the waiting RT program should be restarted.
- A clock interrupt occurs.

From the RT program level:

- The current RT program is finished.
- The current RT program enters waiting mode.
- The user-call RT (<prog.name>) has occured.

The other user-calls (SET, ABSET, ...) do not execute on the monitor level. However, the monitor level is disabled while the corresponding subroutines are being executed, in order to prevent interrupts in critical sections.

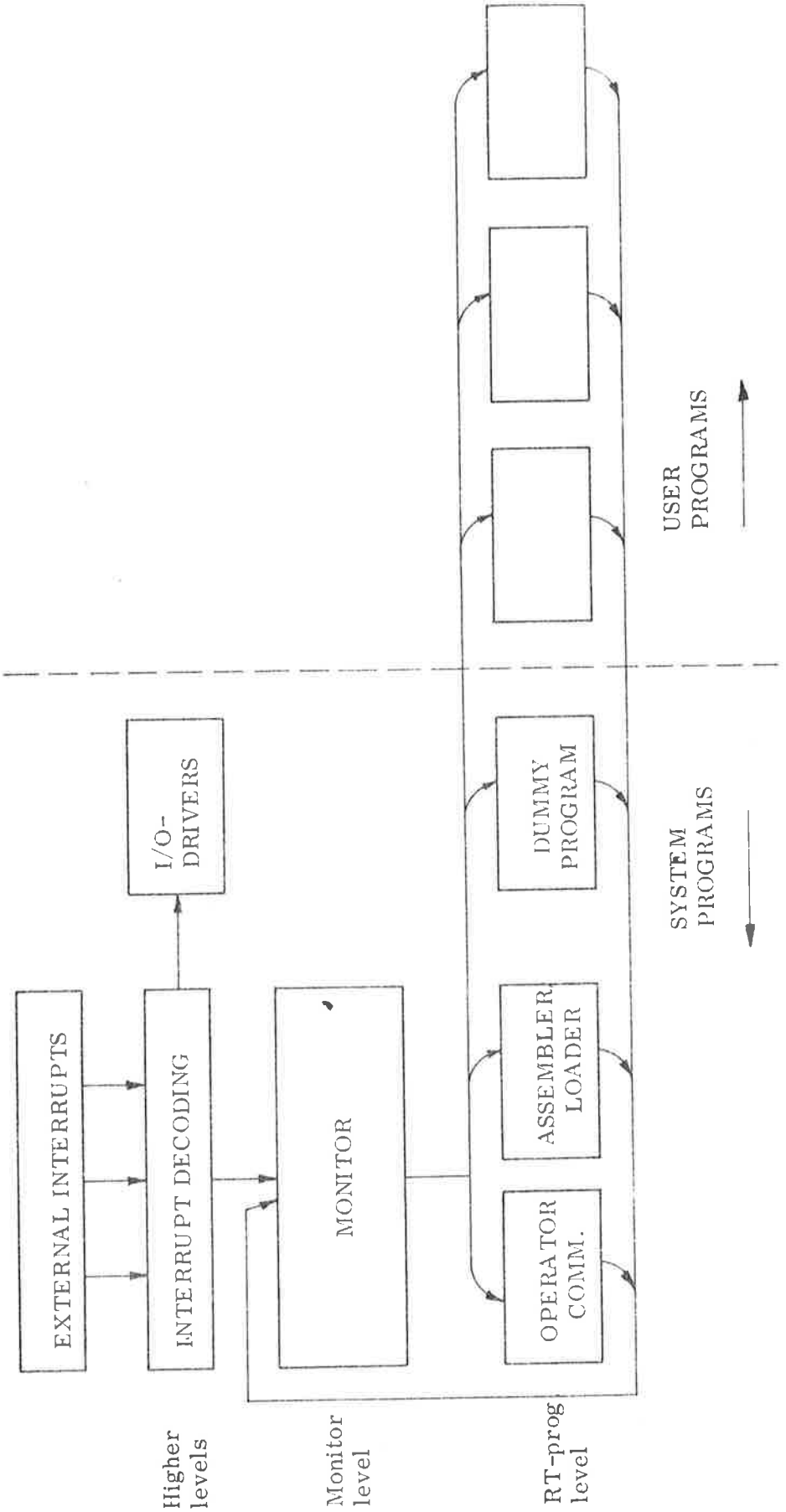

EXTERNAL INTERRUPTS
INTERRUPT DECODING
I/O-
DRIVERS
MONITOR
DUMMY
PROGRAM
ASSEMBLER
LOADER
OPERATOR
COMM.
USER
PROGRAMS
SYSTEM
PROGRAMS
Higher
levels
Monitor
level
RT-prog
level

Figure 1.5

MONITOR

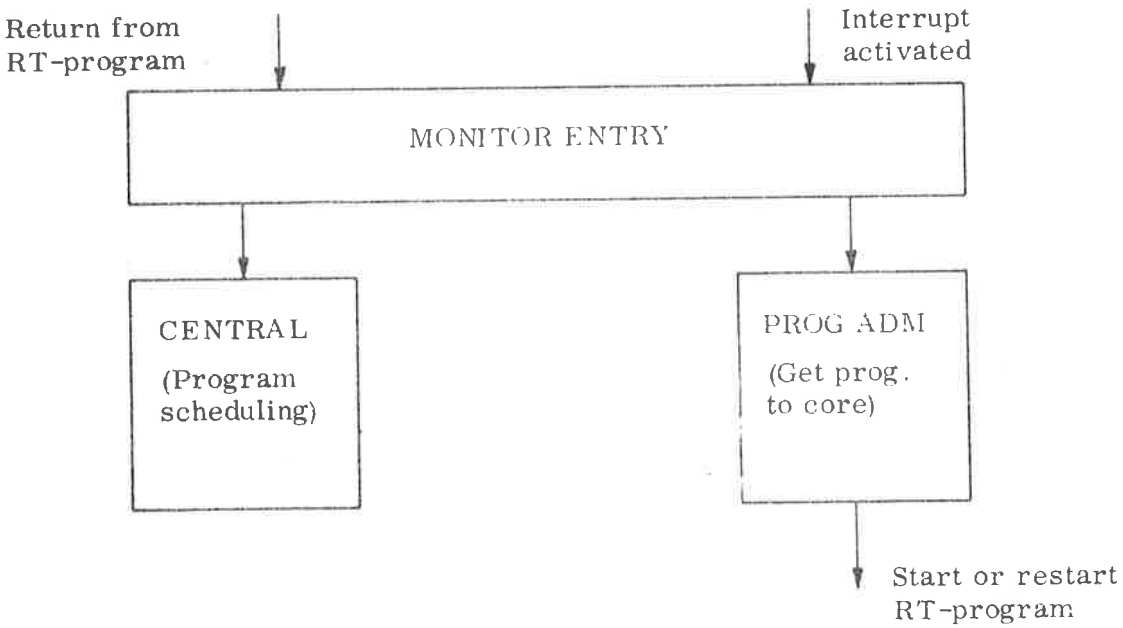


Figure 1.6

2.4.2 Central Monitor

The task of the central part of the monitor is to determine when a new RT program is to be started, giving consideration to priority, time and interrupts.

The central monitor operates on two queues: A time queue and an execution queue. The queues consist of RT descriptions which are linked together.

The time queue consists of RT programs which are to be executed at a future time. The execution queue consists of programs which should be executed as soon as possible, considering their priority. The time queue is ordered with respect to time, and the execution queue with respect to priority.

If an RT program enters the waiting mode, the monitor will take a look further down the execution queue to see if any interrupted programs are ready for execution in the meantime. If not, the monitor will at least find the dummy RT program at the bottom of the execution queue. This program is part of the system: It has lower priority than all other programs, it will always be ready for execution, it will never be finished, and it will do nothing. It has priority 0. This means that a user program with priority 0 will never be started.

2.4.3 Segment Administration

The lower part of the hardware page table (monitor and core common area) is never changed. The dynamic upper part will contain entries for the segments of the currently running RT program. The unused table entries will contain zero, so that trying to use their corresponding logical address space will result in error.

When control is handed from one RT program to another, first the page table entries of the old program is cleared, leaving all dynamic part of the page table equal to zero. Then the entries for the new program will be initialized with values from the core map table.

If a program is going to be started and its segments are not present in physical core, some pages in core must first be removed to give room for the new segments. During the necessary mass storage transfers the program will be in a waiting mode.

2.4.4 Error Reporting

When an error is found by the system, it will be recorded at once, starting a special system RT program which will write an error message on terminal 1. Detailed description will be found in Appendix D, Run Time Errors.

2.5 Input/Output

For detailed description of the I/O calls, see Chapter 3 and the manual NORD File System.

The I/O system gives several facilities:

1. Servicing external units.
2. File access.
3. Internal message transfer between RT programs.
4. General semaphores for sequencing of critical sections.

Each unit is associated with a logical unit number. Some units (terminal, files) will have an input and an output part with the same unit number.

The logical unit numbers are grouped like this:

Octal Numbers:

0-77	Physical devices
100-177	Open files
200-277	Internal devices
300-377	Semaphores
400-477	Process devices
500-577	System devices

The units can be reserved by RT programs, using a monitor call. They can be released either by another monitor call or automatically when the program terminates. If a unit is already reserved, the program will normally be queued for this unit, being set in a waiting state.

The simplest units consist solely of a logical number which can be reserved. Their task is to serve as general semaphores.

The external units can be reached by standard INBT and OUTBT monitor calls. There will normally be a ring buffer associated with each unit.

An internal message transfer unit consists of an input and an output part sharing one ring buffer. One RT pogram can put bytes into the buffer using OUTBT, and a different RT program can fetch them using INBT. Since the access to external and internal units is done in the same way, the communication between two programs in the same computer or in different computers will look the same, except for the unit number.

The files can be accessed sequentially using INBT and OUTBT. In addition, the file blocks can be accessed randomly using the RFILE and WFILE monitor calls. Data transfer and processing can proceed in parallel.

There are two ways of allocating files:

1. Static - the file will consist of a contiguous area on the mass storage.

2. Dynamic - the file will be scattered in the file area of the mass storage.

The access routines will be the same for both types of files. See also the manual NORD File System.

If a program is waiting for input from a unit, it will not be efficient to restart it each time a byte is ready. Therefore the program will remain waiting until a break condition occurs. This break strategy can be set by monitor calls for each unit. Some drivers have special built-in break strategies.

The external devices are treated by the following parts:

Initiating Part

This part is called from the RT program by monitor calls, causing characters to be filled into the buffer if output, and getting characters if input. The task includes also converting from logical unit numbers to physical device numbers.

Driver Routines

The SINTRAN driver routines run on the hardware level of the device interrupt, transferring one character for each activation between device and buffer. There are separate drivers for Teletype input, tape reader input, Teletype/tape punch output, card readers, communication lines, and data links to other computers.

Timer Program

If a requested character has not been transferred in a specified time, INBT or OUTBT will have an error return.

The allowed time is specified separately for each device. If the time is specified equal to zero, no time check will be performed.

2.6 Background System

Each terminal is connected to a background RT program, so that several users can work independently of each other. A supervising RT program time-slices the background programs by periodically changing their priorities.

The background system can be used for

- executing background programs,
- RT program supervising,
- system maintenance.

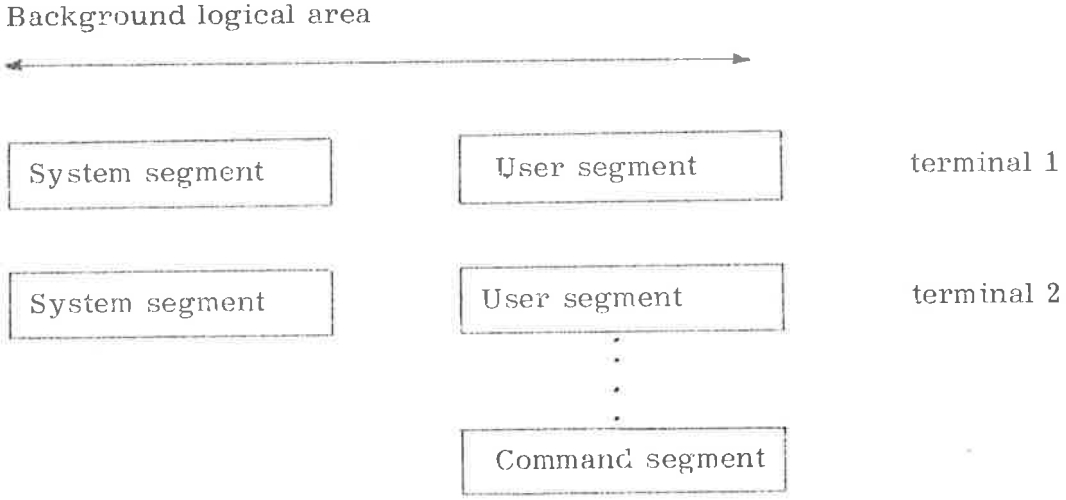
See Section 3.2 for the use of the commands.

A background program consists of two segments:

a) System segment on protect ring 2, containing routines for monitor calls and some commands, and open file tables and buffers.

b) User segment on ring 0. This is the background user's working area, where he can load and execute system programs like the QED editor or the FORTRAN compiler, or his own programs.

In addition there is a segment shared by all background programs. It contains the command decoder and most of the command decoding routines.



3 THE PROGRAMMER'S INTERACTION WITH THE SYSTEM

3.1 Monitor Calls

The monitor call instruction (MON) is used to perform monitor functions. From FORTRAN a set of small subroutines are used, most of them consisting of the MON instruction and an EXIT only (see Appendix E).

If an RT program name equals zero, the calling RT program will be used.

3.1.1 Subroutines callable from FORTRAN

The following subroutines are callable from FORTRAN.

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
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. 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.

If the RT program already is put in the time queue, it will be removed from the queue before being inserted the next time.

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 three last parameters give the time of day for execution. If the time is exceeded at the moment ABSET is called, the program will be scheduled next day at the time specified.

If the RT program already is put in the time queue, it will be removed from this queue before being inserted in the next time.

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 two last parameters give the time between each execution. However, the first execution must be initiated by other means, for instance by a call of SET.

The periodic execution property set by INTV will be reset by a call of DSCNT (see CALL DSCNT below), thus stopping a series of periodic executions.

The interval can be modified by another call of INTV without an intervening DSCNT.

If the starts are delayed because of other RT programs, the delays will not be accumulated. Thus synchronism is preserved. However, if the start is delayed until the time for the next start, an execution is dropped.

The parameter <time unit> may have the values 1, 2, 3, 4:

1: basic time units
2: seconds
3: minutes
4: hours

Other values give an error message, the calling program being aborted.

Example:

```
C       THE PROGRAM PP IS TO RUN EACH 20 MINUTE
        CALL INTV (PP, 20, 3)

C       FIRST EXECUTION STARTS 6 MINUTES FROM
C       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, entering 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 subroutine INTV.

Example:

```
CALL DINTV (<prog.name>, <time>)
```

Return from an RT program.

END - statement in a main program unit.

STOP - statement.

Control will be given to the monitor, which will release the reserved resources of the program. If a STOP statement is used with a number different from zero, the STOP number will be printed on Terminal 1.

Return from assembly-coded RT programs may be done by the monitor call RTEXT (MON 0). The registers may then have arbitrary contents.

Example:

```
PROGRAM TCOM, 30
CALL SUBR (3)
END
```

CALL RTWT

The program will be set in a waiting mode. Its resources will not be released. Next time the program is started, for instance by a call of RT from some other program, it will continue after the call of RTWT.

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

The calling program will be in a waiting state for the time given as parameters.

<time unit>:

1: basic units
2: seconds
3: minutes
4: hours

CALL ABORT (<prog.name>)

The specified program will be aborted if it is running. All reserved resources will be released.

Example:

```
CALL ABORT (PRX)
```

CALL CONCT (<prog. name>,<logical unit>)

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 line.

The logical unit numbers are determined at system generation time, belonging to the I/O system. Several units can be connected to one program. Illegal numbers cause the calling program to be aborted, and an error message will be given.

Example:

```
CALL CONCT (CPIN, 15)
```

CALL DSCNT (<prog. name>)

Any connection established by CONCT will be removed. If the program has been made periodical by INTV, this will be reset. If the program is in time queue or execution queue, it will be removed from the queue.

Example:

```
CALL DSCNT (PRGA)
```

CALL 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.

Example:

```
CALL PRIOR (RTPR, 30)
```

CALL UPDAT (<minute>,<hour>,<day>,<month>, <year>)

The clock and calendar units will get new values. The internal time representation and time queue will not be affected.

If a unit is specified outside its range (e.g., minute >60) an error message is given, and the calling program is aborted. For <year> a value <1974 is illegal.

Example:

```
CALL UPDAT (24, 11, 24, 2, 1974)
```

This will set current time to February, 24, 1974, at 11.24 o'clock.

CALL CLADJ (<time>, <time unit>)
(clock adjust)

The parameter <time unit> may have the values 1, 2, 3, 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 has to be modified. If the time is positive, the clock/calendar will be advanced, otherwise the clock/calendar will stand still for the proper time amount.

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 new clock setting. This concerns also periodic execution, 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 CLOCK (<array>)

The clock/calendar setting at the moment will be recorded in the integer array given as parameter. The seven first elements will contain on return: basic unit, second, minute, hour, day, month and year.

Example:

```
CALL CLOCK (IARR)
WRITE (1, 10) IARR
```

<time> = TIME (0)

This double integer function gives the internal time in basic time units.

Example:

```
DT = TIME (0)
```

CALL FIX (≺segment number≻)

This monitor call is used to make a segment temporarily core resident. The segment, which must be of non-demand type, will be brought into core. 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 core can be used for fixed segments at a time. This amount will be specified at system generation time.

CALL UNFIX (≺segment number≻)

If the segment has been fixed in core by means of the FIX call, this property will be undone, so that the segment can be swapped back to mass storage.

≺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, else 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.

≺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, else if it equals one, it means the write 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 error return with negative function value. If the unit is free, there will be immediate return with zero function value.

CALL RELES (<logical unit>,<read/write>)

The reserved unit will be released if it is reserved from the calling program. If <read/write> equals zero, the input part is reserved for a two-way unit, else if it equals one, it means the write part.

If RELES is not called, the unit will be released when the RT program is terminated.

<value>= PRSRV (<logical unit>,<read/write>,<prog. name>)

The logical unit will be reserved for the RT program specified by the parameter <prog. name>. For a two-way unit <read/write> equal to zero means that the input part is already 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.

CALL PRLS (<logical unit>,<read/write>)

The unit will be released from the program having reserved it.

<value>= WHERE (<logical unit>,<read/write>)

If the logical unit is reserved for some program, the address of the RT description will be returned as the function value. If the unit is free, zero will be returned.

CALL RFILE (<file number>,<return flag>,<core address>,
<block number>,<no. of words>)

This is a subroutine to read a random record from a file. <file number> identifies the file. If <return flag> is zero, the program will be set in a waiting state until the transfer is finished. If <return flag> is set non-zero, there will be return from RFILE as soon as the transfer is started, so that the program and the transfer can proceed in parallel.

The parameter <core address> determines where the record should be placed. In FORTRAN this can be an array name. <block number> gives the file block number where the record starts, while <number of words> defines the record size. There is no inherent restriction on the record size. The first block number is 0.

CALL WFILE (<file number>,<return flag>,<core address>
<block number>,<no. of words>)

This is a subroutine to write a random record onto a file. The parameters have the same meaning as for RFILE.

Note that when RFILE or WFILE is called directly with the MON instruction, also the T register should contain the file number.

RFILE and WFILE can also be called as functions to obtain error information. If the transfer went wrong, a non-zero function value will be returned. Note that RFILE and WFILE then must be declared as INTEGER FUNCTION.

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

This call is used to check whether a transfer is finished or not. If the transfer is finished, there will be immidiate return. If the <return flag> is equal to zero and the transfer is not finished, 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).

<error code> = MAGTP (<function>,<core address>,<unit>,
<maxwords>,<words read>)

This is a monitor call for magnetic tape transfers. <function> is the function code:

0	-	Read one record
1	-	Write one record
10_8	-	Advance to end-of-file
11_8	-	Reverse to end-of-file
12_8	-	Write end-of-file
13_8	-	Rewind
14_8	-	Write skip
15_8	-	Backspace one record
16_8	-	Advance one record
17_8	-	Unload rewind
20_8	-	Read status

<core address> is the logical address where a record should be read or written.

<unit> is the unit number.

<maxwords> is the number of words (16 bits) to be read or written. It cannot be greater than 1024 (2048 bytes).

<words read> is the actual size of the record which is read.

Example:

```
DIMENSION IARR (1024)
I = MAGTP (0, IARR, 0, 1024, N)
I = MAGTP (1, IARR, 1,N, 0)
```

copying a record from unit 0 to unit 1.

<error code> = ACM (<logical unit>, <function>, <coreaddr>, <DMA-addr>, <wordcount>)

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

<logical unit> identifying the external memory. This unit must be reserved on beforehand.

<function> is the function code:

0 - Read
1 - Write
2 - Lock/write/unlock
3 - Clear

Example:

```
INTEGER FUNCTION ACM
DIMENSION IARR (100)
.
.
CALL RESRV (26B, 0, 0)
IX = ACM (26B, 1, IARR, IDMA, 100)
```

<char. value> = INCH (<logical unit>)
(Input character)

This integer function returns an 8-bit character (16 bits if data link) with no modifications, except for card reader, where the card code may be converted to ASCII.

If the input buffer is empty, and the device has no character ready, the program will be in a waiting state until the character has been read from the device. INCH internally uses the monitor call INBT (see Section 3.1.2).

Example:

```
ICH = INCH (1)
```

ICH will get one character from the device with logical unit 1. Negative result means error.

CALL OUTCH (<logical unit>, <char. value>)

The eight least significant bits will be considered to be a character which will be inserted into the output buffer of the unit (16 bits if data link). If the buffer is full, the calling RT program will be set in a waiting state until there is room for the character. Negative A register on return means error. OUTCH internally uses the monitor call OUTBT (see Section 3.1.2).

Example:

```
CALL OUTCH (3, 12B)
```

The character "line feed" (octal 12) will be output on logical unit 3 (usually a paper tape punch).

3.1.2 Monitor Calls from Assembly Programs

INBT

This monitor call reads an eight-bit byte from a unit. The T register contains the unit number. The byte is returned in the A register, with a skip return. In case of error there will be a non-skip return with an error number in the A register:

A = 2 if bad file number
A = 3 if end of file detected
A = 4 if card reader error. The card is read.
A = 5 if illegal device (device not reserved)
A = 7 if card reader error. The card is not read (card crash or feed error).
A = 12_8 if end of device (timeout)

OUTBT

This monitor call writes an eight bit byte to a unit. The T register contains the unit number, and the A register contains the byte. Normally there will be a skip-return. In case of error there will be a non-skip return with an error number in the A register:

A = 2 if bad file number
A = 3 if end of file detected
A = 5 if illegal device (device not reserved)

CIBUF

This is a monitor call to clear the buffer for an input device.

T = logical number.
Return : A = error number
Skipreturn : OK

COBUF

This is a monitor call to clear the buffer for an output device.

T = logical number.
Return : A = error number
Skipreturn : OK

ISIZE

The number of characters in the buffer of an input device is read.

T = logical number.

Return	:	A = error number
Skipreturn	:	A = number of characters

OSIZE

The free room for characters in the buffer of an output device.

T = logical number.

Return	:	A = error number
Skipreturn	:	A = free room

ABSTR

This is a monitor call for data channel transfers between physical core and a mass storage. The monitor call, parameters and core space for transfer must be in permanent core or on a fixed segment.

The T register contains a logical number for the mass storage device. The rest of the parameters are according to the SINTRAN standard call (Appendix A.2). The parameters have different meaning for the different device types.

Discs and drums.

Parameters:

PAR1: Function code

0	- Read
1	- Write
2	- Read test
3	- Compare
20_8	- Read status

PAR2: Core address (double precision)

PAR3: Block address

PAR4: Number of blocks to transport

MCALL

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. If a segment number is zero, only the other segment is wanted. If a segment number is 255, 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 below).

MEXIT

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.

3.1.3 Monitor Calls from Background Programs

The following calls can be used from background programs only.

ECHOM

Define echo mode

A = 0 means always echo
A = 1 means echo everything but control characters
A = 2 means special MAC echo strategy
A < 0 means no echo

BRKM

Define break mode

A = 0 means always break
A = 1 means break only on control characters
A = 2 means special MAC break strategy

For background programs, the monitor calls INBT and OUTBT can be used also for file access. This is not possible from RT programs.

The following monitor calls can be used for background programs as well as for RT programs:

CIBUF
COBUF
ISIZE
OSIZE
CLOCK
TIME
RFILE
WFILE

See also Chapter 6 in the manual NORD File System.

3.2 Commands

A terminal is activated by pressing the "escape" key. The command processor will ask for password, after which the system prints an @ , expecting a command to be typed.

When the user has finished his work, the command LOGOUT should be used to release the terminal.

When typing in commands to the command processor, it is only necessary to type sufficient characters to distinguish the intended command from all other permissible commands. A special character, "-", exists in order to separate a command name into two or more distinct parts. Any and all parts of a command name may be abbreviated. Consider as an example the commands LOAD-BINARY and LIST-FILE. The first command may be typed as LOAD, LOAD-B, L-BINARY, L-B or LO or in quite a few other ways. The second command may be typed as LIST, LIST-F, L-FILE, L-F, LI-F or LI. However, if only L is typed the command processor will indicate that the command is ambiguous.

The abbreviation look-up function just described is a standard function which is almost always available to the user when typing in names. Examples of things which may be abbreviated are: command names, file names and user names.

The collection of parameters for the commands is done in a standardized way as follows. The parameters to a command may be separated by either a comma or any number of spaces. If the user does not know what parameters a command expects or in which order he should type them in, he may simply omit any or all parameters. In this case the command processor will ask for the required parameters.

The commands consist of three groups:

1) Background commands - calling compilers, assembler, and editor. These commands can be used by all timesharing users.

2) Real-time commands - starting and stopping RT programs, loading new RT programs using the RT loader. These commands can be called by the users "RT" and "SYSTEM".

3) System commands - changing locations within the SINTRAN system, and certain file system commands. They can be called by the user "SYSTEM".

See also Chapter 6 in the manual NORD File System.

3.2.1 Background Commands

LOAD-BINARY file

The LOAD-BINARY command simulates the action of pressing MASTER CLEAR and LOAD on the NORD-10. Input is taken from the specified file.

PLACE-BINARY file

Same as LOAD-BINARY except that the loaded program is not started up.

GOTO-USER address

The GOTO-USER command transfers control to the user program at the specified address.

"Escape"

If the "excape" key is pressed while a user program is running, control will return to the utility command processor with a message indicating where the program was interrupted being typed out. All registers are saved. Therefore, the user program may be restarted by supplying the GOTO-USER command with the address at which the program was interrupted. All open files are closed when control returns to the utility command processor.

DUMP file, start address, restart address

The DUMP command saves the contents of the user's virtual memory plus the central registers on the specified file. The start address parameter indicates where the program should be started when it is later retrieved with the RECOVER command. The restart address parameter indicates where the program should be started when restarted with the CONTINUE command.

RECOVER file

The RECOVER command retrieves a program from the specified file and starts it up at its main start address. There is an alternate form of the RECOVER command which is provided for the convenience of the user, whereby one may leave out the name RECOVER completely. In other words, instead of typing RECOVER MAC one may simply type MAC.

CONTINUE

The CONTINUE command is used to restart a program which has previously been started with the RECOVER command. The program is started up at the address specified by the third parameter of the DUMP sommand.

STATUS

The STATUS command lists on the Teletype the contents of the user program's central registers.

LOOK-AT

This is a command to examine and modify locations and registers. As a parameter it can have one of the three symbols:

CORE Locations in the User's address space are affected. The SYSTEM user can also reach locations in fixed core.

SEGMENT A second parameter, the segment number, is required. Then locations on this segment can be reached. This is allowed for the REAL-TIME and SYSTEM users.

IMAGE Locations on the core image on the mass storage can be reached. This is allowed for the SYSTEM user only.

To examine a location, the octal address should be typed, followed by a slash (/). Then the octal contents will be printed. The contents can now be changed by typing an octal number. If a carriage return is given, the contents of the next location will be printed.

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

The contents of the background program registers can be accessed in the same way, using a single letter to address the register. The letters are:

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

If a character not mentioned above is typed, control will be given back to normal control mode.

DATCL

The current time will be printed, from second up to year.

MEMORY <lower bound> <upper bound>

The MEMORY command defines the area to be dumped onto a file by the command DUMP. If the MEMORY command is not used, the bounds set by the last LOAD-BINARY, PLACE-BINARY, or RECOVER will be used.

The MEMORY command does not affect the logical space available for the user.

TIME-USED

The CPU time used will be printed.

3.2.2 Monitor Commands

Most of the monitor calls may be executed as operator commands. The parameters should be specified as (signed) decimal integers, except for RT programs, which are symbolic names or octal numbers.

RT <prog.name>

Example:

```
RT STAX
```

SET <prog.name> <time> <time unit>

Example:

```
SET PP2 18 3
```

ABSET <prog.name> <second> <minute> <hour>

Example:

```
ABSET PXY  0 30 18
```

INTV <prog.name> <time> <time unit>

Example:

```
INTV SAMPL  2 2
```

ABORT < prog. name >

Example:

ABORT OPTI

CONCT < prog. name > < int. line number >

Example:

CONCT RESP 257

DSCNT < prog. name >

Example:

DSCNT RESP

PRIOR < prog. name > < priority >

Example:

PRIOR LPM 19

UPDAT < minute > < hour > < day > < month > < year >

Example:

UPDAT 14 10 3 3 1975

CLADJ < time > < time unit >

Example:

CLADJ 10 2

FIX < segment number >

Example:

FIX 25

UNFIX < segment number >

Example:

UNFIX 25

PRSRV <logical unit> <read/write> <prog. name>

Example:

PRSRV 25 0 PROG

PRLS <logical unit> <read/write>

Example:

PRLS 25 0

RTON <prog. name>

Example:

RTON PROG

RTOFF <prog. name>

Example:

RTOFF PROG

IOSET <logical unit>, <read/write> <prog. name> <control>

Example:

IOSET 2 0 RTX -1

LIST-TIME-QUEUE

The RT programs in the time queue will be listed.

LIST-EXEC-QUEUE

The RT programs in the execution queue will be listed.

LIST-SEGMENT <segment number>

The contents of the element in the segment table will be listed.

LIST-RT-DESCRIPTION <RT-name>

The contents of the RT description will be listed.

RTENTER

The user "RT" will be entered as user for the RT programs. This must be done before any file can be opened for RT programs. The command should be given each time the SINTRAN III system is started.

GET-RT-NAME <octal RT address>

The corresponding name of the RT program will be printed. The octal RT address can occur for instance in error messages.

3.2.3 Console Commands

TERMINAL-STATUS <terminal number>

The following information is listed:

1. Mode: User mode, when a subsystem or user program is active, or Command mode.
2. User name.
3. Last command line.

STOP-TERMINAL <terminal number>

The specified terminal will be logged out.

WHO-IS-ON

The active terminals will be listed with terminal number and user name.

WHERE-IS <file or peripheral name>

The user name and terminal number will be listed.

STOP-SYSTEM

The computer will stop. It can be restarted by pushing the RESTART button. Then the system will continue where it left.

3.2.4 Subsystems

RT-LOADER

The RT loader will load RT programs onto segments.
Manual: SINTRAN III Real-Time Loader.

FTN

The FORTRAN Compiler.
Manual: NORD Standard FORTRAN Reference Manual.

QED

Editor.
Manual: QED Users' Manual.

MAC

Assembler
Manual: MAC Users' Guide.

BRL

Loader for background programs.
Manual: Binary Relocating Loader.

MACF

Assembler version for assembling onto file.
Manual: MAC Users' Guide.

NORD PL

Medium level language compiler.
Manual: NORD PL Users' Guide.

LDR

Loader including FORTRAN run-time system.
Manual: Binary Relocating Loader

KRYSSREF

Cross reference listing program.

BASIC

Interpreter.
Manual: NORD BASIC Reference Manual.

SORT

Sorting routine.
Manual: NORD SORT System.

3.3 Batch System

3.3.1 Introduction

In addition to the usual interactive communication with the SINTRAN background system, the user may also run his background jobs in batch mode. This means that command input is taken from another device or file than the terminal, and that device number 1, which usually identifies the terminal, is interpreted as the batch input file on input and the batch output file on output.

There are two slightly different ways of running batch jobs in SINTRAN:

Type 2:

A batch job may be initiated by the @MODE command. This command just changes the command input and output files to those specified in the @MODE command. The batch job will continue running under the user currently logged on, and control will return to the terminal when end-of-file is reached on the batch input file or the command @MODE TERMINAL TERMINAL is found on the batch input file.

Type 1:

A batch process may be initiated by the @BATCH command. This command will start a background RT program running independently of any terminal. Control will return to the terminal immediately after the @BATCH command and this terminal may be used for other activities running in parallel with the batch process. Control commands to the batch process may be given from any terminal.

All the interactive commands may also be used in batch mode, except the following:

- @LOGOUT and @MODE are illegal in type 1 batch.
- Although they are legal, some commands are not very well suited for use in batch. Among these are for example the LOOK-AT command.

All command parameters have to be written on the same line as the command itself, because the system cannot ask for missing parameters in batch mode.

The first character on a system command line is always an @, which corresponds to the @ written out by the system in front of each command in interactive mode.

3.3.2 Example of a Batch Job

The batch input file has the following contents:

```
@FTN
COM   1,0,OBJ
      PROGRAM TEST
      WRITE (1,1)
1     FORMAT (* FORTRAN TEST OUTPUT *)
      END
      EOF
EX
@ LDR
A  OBJ
S
```

Running of this job will give the following output on the batch output file:

```
@ FTN

NORD FTN
$COM  1,0,OBJ
      PROGRAM TEST
      WRITE (1,1)
1     FORMAT (* FORTRAN TEST OUTPUT *)
      END

      EOF
4     STATEMENTS COMPILED
$EX
@ LDR

BINARY LOADER
L*A OBJ
L*S

FORTRAN TEST OUTPUT
```

If this job is run as type 1 batch, it should be bracketed by an @ENTER command identifying the user and two ESC characters signalling end of job.

3.3.3 Definitions

- The batch input file is the file or input device from where SINTRAN takes its command input when running in batch mode.

  Or more precisely: The batch input file is the input part of device number 1 when running in batch mode.

- The batch output file is the file or output device where SINTRAN lists command input and command output when running in batch mode.

  Or more precisely: The batch output file is the output part of device number 1 when running in batch mode.

3.3.4 Description of Type 2 Batch (MODE type)

The MODE command has the following format:

@MODE <batch input file><batch output file>

The effect of this command is to redefine device number 1 to mean the batch input file on input and the batch output file on output.

When the MODE command is given, SINTRAN will take command input from the batch input file until another MODE command is found (on the batch input file), or end of file is reached on the batch input file. The commands are listed on the batch output file together with the command output (if any).

If the user programs read from device number 1, input will be taken from the batch input device, and all output to device number 1 is routed to the batch output file.

If end of file is reached on the batch input file, control will be returned to the terminal Teletype.

Error conditions:

If an error condition occurs in a MODE job, the usual error message will be printed on the batch output file. Then the message

BATCH JOB ABORTED

will be printed and control returned to the terminal.

3.3.5 Description of Type 1 Batch (BATCH type)

3.3.5.1 Explanation of Terms

- A batch process is a background RT program very much alike the background RT programs handling interactive communication with the SINTRAN system. The task of the batch process is to process the batch jobs on its batch input files, one at a time.

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

  The extra overhead introduced by adding a batch process is approximately the same as by adding a terminal.
  The number of batch processes will usually be limited to one or very few, because one of the reasons for batch processing is to run the jobs one by one, and save overhead introduced by timesharing them.

  A batch process is started by a @BATCH command, and terminated by an @ABORT-BATCH command.

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

  The first command in a batch job is always the @ENTER command identifying the user. The last two characters in a batch job is always ESC (33_8), signalling end of job. These two characters corresponds to the command @LOGOUT in interactive mode. For the convenience of the user who prepares his batch input files from a terminal, the character SHIFT CTRL M (35_8) may also be used as end of job characters. The card code of the ESC character is the multipunch 7-8-9.

- A batch queue is a queue of batch input file - batch output file pairs held internally in SINTRAN. A new pair is added to the queue by the @APPEND-BATCH command. Each batch process has its own batch queue.

  When end of file is reached on a batch input file, the batch process fetches the next batch input file - batch output file pair on the queue, and continues taking input from the new batch input file and giving output to the new batch output file.

If the batch queue is empty, the batch process will enter waiting state.

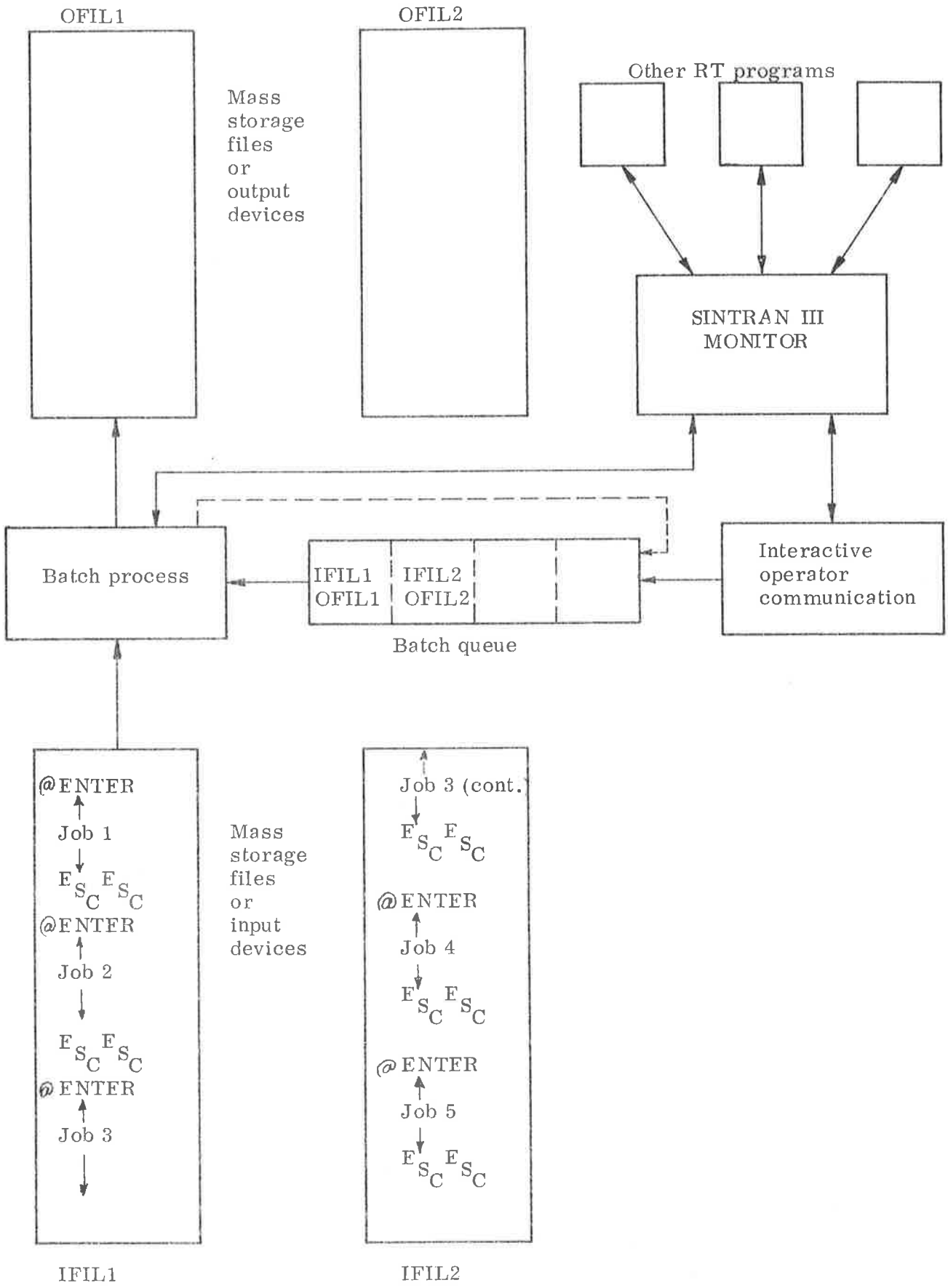
A batch input file may contain any number of jobs, and a job may use any number of batch input file - batch output file pairs. Thus, the boundaries between jobs and the boundaries between batch input files, is completely independent of each other.

A batch process will always be in one of the three following states:

Passive - this means that the batch process is not 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 some job.



This figure shows a system with one terminal and one batch process. The dotted line indicates that a batch job may append jobs to its own batch process.

3.3.5.2 Description of Commands

@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.

When the batch process is started, it immediately enters waiting state, because 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 queue by an @APPEND-BATCH command.

The @BATCH command is usually given from a terminal in interactive mode.

@APPEND-BATCH <batch number><batch input file> <batch output file>

This command appends the <batch input file>-<batch output file> pair specified to the batch queue for the batch process identified by <batch number>. If the batch process is idle, it will be restarted.

If the batch queue is full, the message

BATCH QUEUE FULL

will be printed.

If the specified batch process is passive, the message

BATCH PASSIVE

will be printed.

Notes:

- If the batch input file is owned by another user than SYSTEM, the user name should be specified in brackets in front of the file name.

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

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

Examples of APPEND-BATCH commands:

```
@APPEND-BATCH 1, CARD-READER, LINE-PRINTER
@APPEND-BATCH 2, (NILS) BAFIL, BOFIL
```

The @APPEND-BATCH command is usually given from a terminal in interactive mode.

@LIST-BATCH-PROCESS

This command lists the status of the batch processes in the system. It has no effect on the batch processes.

Example of @LIST-BATCH-PROCESS command for a system with three batch processes:

```
@LIST-BATCH-PROCESS

        1   IDLE, NO USER LOGGED ON
        2   ACTIVE, USER NILS LOGGED ON
        3   PASSIVE
@
```

The @LIST-BATCH-PROCESS command is usually given from a terminal in interactive mode.

@LIST-BATCH-QUEUE <batch number>

This command lists the batch queue of the specified batch process.

Example of @LIST-BATCH-QUEUE command:

```
@LIST-BATCH-QUEUE  1
CARD-READER  LINE-PRINTER
(PER) BINP  LINE-PRINTER
@
```

The @LIST-BATCH-QUEUE command is usually given from a terminal in interactive mode.

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

This command may be used to abort a batch job. The job will be aborted and batch input will be skipped until the next end of job characters. The batch process will then continue processing the next job.

If the specified <user name> is not the user currently logged on the specified <batch number>, nothing will be done.

The @ABORT-JOB command is usually given from a terminal in interactive mode.

@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.

The @ABORT-BATCH command is usually given from a terminal in interactive mode.

@ENTER <user name><password><project number><max. time>

This command is legal only as the first command in a batch job. It identifies the user of the following job, and corresponds to the log on procedure in interactive mode. <user name> may be any name implemented as a user in the system. <password> should be the correct password for the specified user. If the user has no password, just two commas should be present between <user name> and <project number>. <project number> is used for accounting purposes and may be any decimal integer. <max. time> is the maximum CPU time for the job in minutes. If this time limit is reached, the job will be aborted.

The @ENTER command is always given from a batch input file in batch mode.

@SCHEDULE <device number><device number>,.....

This is a general command specially designed for use in batch mode. It is used to reserve the devices mentioned in the parameter list for the current job. If one or more of them are reserved by other programs, the batch process will enter waiting state until they are released. This is very useful in batch mode, because one does not know if a device will be free at run time when one prepares a batch job.

To prevent deadlock situations, no devices can be reserved for the job before the @SCHEDULE command is given. If it is, the error message DEVICE ALREADY RESERVED will be given, and the batch job will be aborted.

The @SCHEDULE command is usually given from a batch input file in batch mode.

3.3.5.3 Example of a Batch Run

The file BINP is owned by user PER and has the following contents.

```
@ ENTER  NILS, PWN, 5
@ FTN
COM  1.0,100
-
-
-       FORTRAN STATEMENTS
-
-
EOF
EX
@ LDR
A  100
S
ESC ESC
@ ENTER  PER, PWP, 4
@ SCHEDULE  3
@ COPY  FAST-PUNCH, PERFILE
ESC ESC
```

It is assumed that the users NILS and PER have the passwords PWN and PWP respectively. Of course, the passwords will not be listed on the batch output file.

To run these two jobs, the following commands could be given from any terminal:

```
@ WHERE-IS  LINE-PRINTER
@ FREE TO USE
@ BATCH
@ BATCH NUMBER = 1
@ APPEND-BATCH 1 (PER)BINP, LINE-PRINTER
@
```

If you have no more batch jobs to run when the batch process has entered idle state, an

```
@ ABORT-BATCH 1
```

should be given to terminate the batch process and release the line printer.

3.3.5.4 Special Monitor Calls concerning Batch Jobs

- If the monitor call RTEXT (MON 134) is executed by a background program running in batch mode, the batch job will be aborted.

- The monitor call RSIO (MON 143) may be executed by a background program if it wants to find out whether it is running in batch or interactive mode.

  The return parameters from RSIO are:

A register	:	0 if interactive mode 1 if type 1 batch 2 if type 2 batch (MODE)
T register	:	command input file number
D register	:	command output file number

3.3.5.5 Error Conditions

If an error condition occurs in a batch job, the job will be aborted and batch input will be skipped until the next end of job characters. The batch process will then continue with the next batch job.

If an I/O error occurs on the batch input or output file, all jobs on current batch input file will be skipped and the batch process will continue with the next batch input file - batch output file pair on the batch queue.

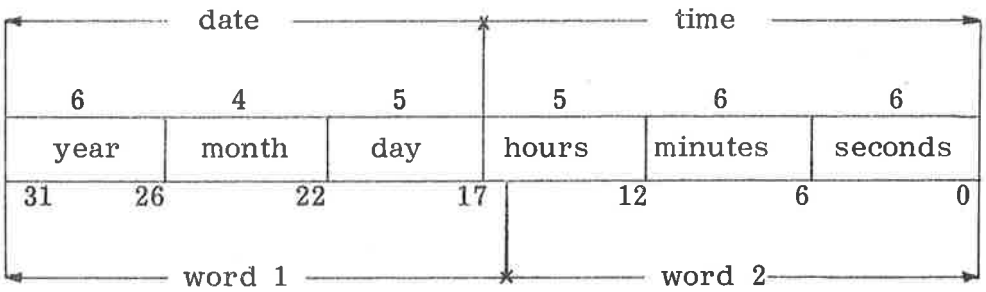
In most cases, an error message will be printed on the batch output file, but if an I/O error has occurred on the output file, this is of course impossible.

4 ACCOUNTING SYSTEM

Whenever a user logs out, a record is written in the account file on the disc. This file, named (SYSTEM) ACCOUNTS: DATA, consists of blocks of length 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 date	binary
	11	console seconds	binary
	12	CPU seconds	binary
	13-15	unused	

Log-off time and date are packed into two words, or a 32 bit field, as follows:



The first block of the file contains only 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 then 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.

4.1 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 no. and max.no. 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.

APPENDIX A

ASSEMBLY CODE IN SINTRAN

A.1 Assembly-coded RT Programs

The name of the RT program, which may be used in other RT programs, is a pointer to the first location of the RT description.

At the moment when the RT program starts, the machine status is like this:

> The P register contains the starting address, as found in location 7 of the RT description.
>
> The other registers have arbitrary values.
>
> In the program itself, all the registers may be used freely, including the B register.

When program execution is finished, the program control should be transferred to the monitor by the monitor call RTEXT (MON 0). The registers may then have arbitrary values.

An example of an assembly-coded RT program follows. The name of the RT program is PER. It has priority 5.

```
                )9BEG
PRI5 = 5
2RTEX = 0
                )9RT PER PRI5
                LDA   (PARAM
                .
                .
                .
                MON   2RTEX
PARAM,          NUMBER
                )DEC
NUMBER,         50
                )FILL
                )9END
```

The RT loader will generate the RT description.

A.2 SINTRAN Standard for Subroutine Calls

The subroutine call is performed by a JPL instruction.

Example:

```
JPL I (SUBR
```

The A register contains the address of the parameter list. The parameter list contains the addresses of the actual parameters. Returns from the subroutine are always to the instruction after the JPL.

On return, the B and X registers will have the same values as before the subroutine jump: the other registers may be changed.

The T-A-D registers may have a function value on return.

Note especially that pointers to the RT descriptions are considered to be variables. Thus, the location in the parameter list will contain the address of such a variable in this case. See Figure A.1 below.

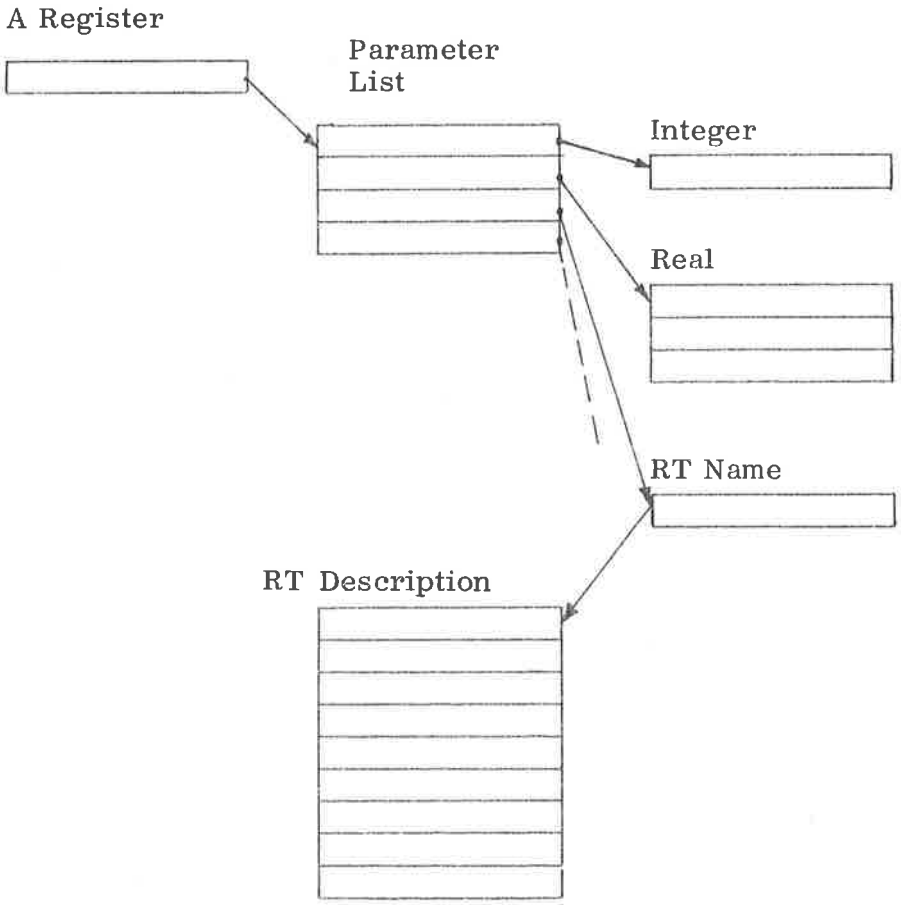


Figure A.1: Standard Call

A.3 Examples of Monitor Calls from Assembly Programs

For subroutines callable from FORTRAN, the corresponding FORTRAN call will also be shown.

CALL RT (PR1)

```
                LDA   (PARAM
                MON   100
                .
                .
                .
PARAM,          (PR1                    % RT PROGRAM
```

CALL SET (RT1, 10, 2)

```
                LDA   (PARAM
                MON   101
                .
                .
                .
PARAM,          (RT1                    % RT PROGRAM
                (12                     % NUMBER OF
                (2                      % SECONDS
```

CALL ABSET (PROG, 0, 30, 17)

```
                LDA   (PARAM
                MON   102
                .
                .
                .
PARAM,          (PROG                   % RT PROGRAM
                (0                      % SECOND
                (36                     % MINUTE
                (21                     % HOUR
```

```
CALL INTV (PP, 20, 3)

                LDA   (PARAM
                MON   103
                .
                .
                .
PARAM,          (PP             % RT PROGRAM
                (24             % NUMBER OF
                (3              % MINUTES
```

Terminating an RT program:

```
                MON   0
```

The registers may then have arbitrary contents. All reserved units of this program will be released.

Set a program waiting:

```
                MON   135
```

The program will stop. It will restart at the same point next time it is started.

```
CALL HOLD (5, 2)

                LDA   (PARAM
                MON   104
                .
                .
                .
PARAM,          (5              % TIME
                (2              % TIME UNIT
```

```
CALL ABORT (PRX)

                LDA   (PARAM
                MON   105
                .
                .
                .
PARAM,          (PRX            % RT PROGRAM TO BE
                                % ABORTED
```

```
CALL CONCT   (CPIN, 5)

             LDA   (PARAM
             MON   106
             .
             .
             .
PARAM,       (CPIN                 % RT PROGRAM
             (5                    % INTERRUPT LINE NUMBER
```

```
CALL DSCNT   (PRGA)

             LDA   (PARAM
             MON   107
             .
             .
             .
PARAM,       (PRGA                 % RT PROGRAM
```

```
CALL PRIOR   (RTPR, 30)

             LDA   (PARAM
             MON   110
             .
             .
             .
PARAM,       (RTPR                 % RT PROGRAM
             (36                   % NEW PRIORITY
```

```
CALL UPDAT   (24, 11, 24, 2, 1974)

             LDA   (PARAM
             MON   111
             .
             .
             .
             )DEC
PARAM,       (24                   % MINUTE
             (11                   % HOUR
             (24                   % DAY
             (2                    % MONTH
             (1974                 % YEAR
```

```
CALL CLADJ  (15, 2)

            LDA  (PARAM
            MON  112
            .
            .
            .
PARAM,      (17                 % NUMBER OF
            (2                  % SECONDS
```

```
CALL CLOCK  (IARR)

            LDA  (PARAM
            MON  113
            .
            .
            .
PARAM,      IARR
IARR,       0; 0; 0; 0; 0; 0; 0;    % CLOCK UNITS WILL BE
                                    % STORED
```

```
DELTA = TIME (0)

            MON  11
            STD  DELTA
            .
            .
            .
DELTA,      0; 0
```

```
INBT

            SAT  2              % TAPE READER
            MON  1              % INBT
            JMP  ERROR
            STA  CHAR
```

```
OUTBT

            LDA  CHAR
            SAT  3              % TAPE PUNCH
            MON  2              % OUTBT
            JMP  ERROR
```

```
CALL DSET   (PROG, DTIME)

                LDA   (PARAM
                MON   126
                .
                .
                .
PARAM,          (RT1                    % RT PROGRAM TO BE
                                        % SCHEDULED
                DTIME
DTIME,          1                       % THIS DOUBLE PRECISION
                32000                   % NUMBER WILL BE ADDED
                                        % TO CURRENT TIME AND
                                        % THE RESULT PLACED IN
                                        % THE RT DESCRIPTION
```

```
CALL DABST   (PROG, DTIME)

                LDA   (PARAM
                MON   127
                .
                .
                .
PARAM,          (PROG
                DTIME
DTIME,          10333                   % THIS DOUBLE PRECISION
                145016                  % NUMBER WILL BE PLACED
                                        % IN THE RT DESCRIPTION
```

```
CALL DINTV   (PROG, DTIME)

                LDA   (PARAM
                MON   130
                .
                .
                .
PARAM,          (PROG
                DTIME
DTIME,          0                       % THIS DOUBLE PRECISION
                6000                    % NUMBER WILL BE PLACED
                                        % IN THE RT DESCRIPTION
```

ABSTR

```
                LDT   LOGNO
                LDA   (PARAM
                MON   131
                .
                .
                .
PARAM,          (0                  % READ
                (0                  % MEANS WAIT
                CORA                % PHYSICAL CORE ADDRESS
                (2700               % ABSOLUTE BLOCK NUMBER
                (1                  % NUMBER OF BLOCKS
CORA,           0
                IARR
IARR = *                            % TRANSFER TO THIS AREA
* + 100/                           % HARDWARE BLOCK SIZE
                                    % FOR DRUM (64)
```

MCALL, MEXIT

```
                LDA   (PARAM        % PARAMETER LIST FOR
                                    % SUBR1
                LDT   (START
                MON   132           % MCALL, OVER TO SUBR1
                .                   % RETURN HERE AFTER
                .                   % MEXIT
                .
START,          SUBR1
                2005                % SUBR1's SEGMENT,
                                    % 4 AND 5
```

SUBR1,

```
                .                   % END OF SUBR1
                .                   % T AND L AS WHEN SUBR1
                                    % WAS ENTERED
                MON   133           % MEXIT, RETURN TO
                                    % CALLING PROG.
```

CALL FIX (33)

```
                LDA   (PARAM
                MON   115
                .
                .
)DEC
PARAM,          (33                 % SEGMENT NUMBER
```

```
CALL  UNFIX  (33

             LDA  (PARAM
             MON  116
             .
             .
             .
             )DEC
PARAM,       (33
```

```
IX = IOSET  (2,  0,  PROG,  -1)

             LDA  (PARAM
             MON  141
             JAN  ERROR
             .
             .
             .
PARAM,       (2              % TAPE READER
             (0              % INPUT
             (PROG           % RT-PROG
             (-1             % CLEAR
```

```
IX = RESRV (4,  0,  0)

             LDA  (PARAM
             MON  122
             JAN  ERROR      % ILLEGAL UNIT
             .
             .
             .
PARAM,       (4              % CARD READER
             (0              % INPUT
             (               % WAIT IF BUSY
```

```
CALL RELES  (4,  0)

             LDA  (PARAM
             MON  123
             JAN  ERROR      % ILLEGAL UNIT
             .
             .
             .
PARAM,       (4
             (0
```

```
IX = PRSRV (3, 1, PROGX)

                LDA   (PARAM
                MON   124
                JAN   ERROR          % UNIT RESERVED OR
                .                    % NON-EXISTING
                .
                .
PARAM,          (3                   % TAPE PUNCH
                (1                   % OUTPUT
                (PROGX

---------------

CALL  PRLS  (3, 1)

                LDA   (PARAM
                MON   125
                JAN   ERROR
                .
                .
                .
PARAM,          (3
                (1

---------------

IVAL = WHERE (3, 1)

                LDA   (PARAM
                MON   140
                JAN   ERROR          % NON-EXISTING DEVICE
                JAZ   NRES           % NOT RESERVED
                STA   RTPRG
                .
                .
                .
PARAM,          (3
                (1

---------------
```

```
CALL RFILE  (101B, 0, IARR, 2, 256)

                LDA   (PARAM
                LDT   I  PARAM
                MON   117
                JAF   ERROR
                .
                .
                .
PARAM,          (101
                (0
                IARR
                (2
                (400
                )FILL
IARR,           * + 400/
```

```
CALL WAITF  (101B, 0)

                LDA   (PARAM
                MON   121
                JAF   ERROR
                .
                .
                .
PARAM,          (101
                (0
```

```
CIBUF

                SAT   2
                MON   13
                JMP   ERROR
```

```
COBUF

                SAT   3
                MON   14
                JMP   ERROR
```

ISIZE

```
                SAT   2
                MON   66
                JMP   ERROR
                STA   NCHAR
```

OSIZE

```
                SAT   3
                MON   67
                JMP   ERROR
                STA   NLEFT
```

IX = MAGTP (0, IARR, 1, 1024, IRD)

```
                LDA   (PARAM
                MON   144
                JAF   ERROR
                .
                .
                .
PARAM,          (0                  % READ A RECORD
                IARR
                (1                  % FROM UNIT 1
                (2000               % MAX WORDS
                IRD
IRD,            0                   % ACTUAL WORDS READ
)FILL
IARR,           * + 2000/           % TO THIS AREA
```

IX = ACM (26B, 1, IARR, IDMAD, 100)

```
                LDA   (PARAM
                MON   145
                JAF   ERROR
                .
                .
PARAM,          (26                 % LOG. NUMBER
                (1                  % WRITE
                IARR
                (IDMAD              % DMA - ADDRESS
                (144                % NO. OF WORDS
)FILL
IARR,           * + 100/
```

APPENDIX B

SYSTEM MONITOR CALLS SUMMARY (File System included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from back-ground	Short Description
Name	Number	Parameters				
ABORT	105	(NAME)	yes	yes	no	Abort an RT program
ABSET	102	(NAME, SEC MIN, HOUR)	yes	yes	no	
ABSTR	131	(READ/WRITE CORE-ADDR, MADDR, BNMBK)	no	yes	no	Transfer data to/from mass storage. This call can be used from system programs only. MADDR=mass storage address BNMBK=number of blocks
ACM	145	(LOG, UNIT, FUNC, CORAD, DMAADDR, WORDS)	yes	yes	yes	Access ACM
CIBUF	13	T = dev. no.	no	no	yes	Clear input buffer
CLADJ	112	(TIME, UNIT)	yes	yes	no	Adjust the internal clock
CLOCK	113	(ARRAY)	yes	yes	yes	Get the clock- and calendar units (7 locations)

SYSTEM MONITOR CALLS SUMMARY (File system included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
CLOSE	43	T = file no.	yes	no	yes	Close file.
COBUF	1	T = dev. no.	no	no	yes	Clear output buffer
CONCT	106	(NAME, LINE)	yes	yes	no	Connect the RT program to the interrupt line.
DABST	127	(NAME, DTIME)	yes	yes	no	Start the RT program at a given time.
DINTV	130	(NAME, DTIME)	yes	yes	no	Connect the RT program to the time interval.
DSCNT	107	(NAME	yes	yes	no	Disconnect the RT program from any interrupt or interval assignment.
DSET	126	(NAME, DTIME)	yes	yes	no	Start the RT program when the given time has expired.
ECHOM	3	A = strategy no.	no	no	yes	Set echo strategy. For background programs only

SYSTEM MONITOR CALLS SUMMARY (File System included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
ERMSG	64	A = Error code	no	no	yes	Write error message (File System)
HOLD	104	(TIME, UNIT)	yes	yes	no	Let the calling program wait for the given time.
INBT	1	T = log. unit A = byte	no	no	yes	Read a byte
INTV	103	(NAME, TIME, UNIT)	yes	yes	no	Connect the RT program to the time interval.
IOSET	141	(NIT, R/W, NAME CONTROL)	yes	yes	no	Set control information to a device.
ISIZE	66	T = dev. no.	no	no	yes	Read number of characters
MAGTP	144	(FUNC, CORAD, UNIT, MAXW, READW)	yes	yes	yes	Access magnetic tape

SYSTEM MONITOR CALLS SUMMARY (File System included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
MCALL	132	(SUBR, ADDR, SEGMENTS)	no	no	no	Call a subroutine on different segments.
MEXIT	133	T = old segments	no	no	no	Return from a sub-routine on different segments.
OPEN	50	X = file, T = access A = type	yes	no	yes	Open file.
OSIZE	66	T = dev. no.	no	no	yes	Read free room in buffer
OUTBT	2	T = log. unit A = byte	no	no	yes	Write a byte.
PRIOR	110	(NAME, PRIOR)	yes	yes	no	Set new priority.
PRLS	125	(UNIT, R/W)	yes	yes	no	Release the unit from any program.
QERM	65	A = error code	no	no	yes	Write error message and quit. (File system)

SYSTEM MONITOR CALLS SUMMARY (File System included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
RDISC	5	T = Block no. X = Core address	no	no	yes	Read 156 words from scratch file 100 (File System)
REABT	75	T = file number AD= byte pointer	no	no	yes	Read byte pointer (File system)
RELES	123	(UNIT, R/W)	yes	yes	no	Release the unit from the current program.
RESRV	122	(UNIT, R/W, RETURN FLAG)	yes	yes	no	Reserve the unit for the current program.
RFILE	117	(UNIT, CONT. FLAG, CORAD, MASSAD, NO. WORDS)	yes	yes	yes	Read a random record from file.
RPAGE	7	T = file no. A = block no. X = core address	no	no	yes	Read 156 words from a file. (File system)
RT	100	(NAME)	yes	yes	no	Start the RT program.

SYSTEM MONITOR CALLS SUMMARY (File system included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
RTWT	135		yes	yes	no	Let the current program wait.
RTEXT	0		no	no	yes	Stop the current program.
SET	101	(NAME, TIME, UNIT)	yes	yes	no	Start RT program in the given time.
SETBL	77	T = file no. AD= block pointer	no	no	yes	Set block pointer (File system)
SETBS	76	T = file no. AD= block size	no	no	yes	Set block size (File system)
SETBT	74	T = file no. AD= byte pointer	no	no	yes	Set byte pointer (File system)
SMAX	73	T = file no. AD= byte pointer	no	no	yes	Set maximum byte pointer. (File system).
TIME	11		yes	yes	yes	Get the current internal time.

SYSTEM MONITOR CALLS SUMMARY (File System included)

SINTRAN III Monitor Calls			Accessible from FORTRAN	Standard sub-routine call	Accessible from background	Short Description
Name	Number	Parameters				
UPDAT	111	(MIN, HOUR, DAY, MONTH, YEAR)	yes	yes	no	Update clock and calendar.
WAITF	121	(UNIT)	yes	yes	no	Wait for a transfer to be finished.
WDISC	6	T = block no. X = core addr.	no	no	yes	Write 256 words onto scratch file 100. (File system)
WFILE	120	(UNIT, CONT. FLAG, CORAD, MASSAD, NO. WORDS)	yes	yes	yes	Write a random record
WHERE	140	(UNIT)	yes	yes	no	Get the RT program having reserved the unit.
WPAGE	10	T = file no. A = block no. X = core address	no	no	yes	Read 256 words from a file. (File system)

APPENDIX C

BACKGROUND SYSTEM COMMAND SUMMARY

Command	Parameters	Used by	Short description
ABORT	NAME	RT	Stop RT program
ABORT-BATCH	BATCH-NO.	RT	Abort batch process
ABORT-JOB	BATCH-NO., USER	RT	
ABSET	NAME, SEC, MIN, HOUR	RT	Start RT program at time of day
BATCH APPEND-BATCH	BATCH-NO. INPUT, OUTPUT	RT USER	Start batch process
COPY	DESTIN. FILE, SOURCE FILE	USER	Copy file or device
CLADJ	TIME, UNIT	RT	Adjust internal clock
CONCT	NAME, LINE	RT	Connect RT program to interrupt line
CONTINUE		USER	Restart background program
DATCL		USER	Print current time and date on the terminal
DSCNT	NAME	RT	Disconnect the RT program
DUMP	FILE, START, RESTART	USER	Save background program
FIX	SEGM. NO.	RT	Fix segment in core
GET-RT-NAME	OCTAL ADDRESS	RT	Convert address to name
GOTO-USER	ADDRESS	USER	Start background program
INIT-ACCOUNTING	NUMBER, MAX	SYSTEM	
INTV	NAME, TIME, UNIT	RT	Connect RT program to time interval

Error number	Meaning	xx	yy	Program aborted
00	Illegal monitor call	RT prog.		yes
01	Bad RT program address	"		"
02	Wrong priority in PRIOR	"		"
04	Ring protect	"		"
05	Memory protect	"		"
09	Illegal parameter in CLOCK	"		"
10	Illegal parameter in ABSET	"		"
11	Illegal parameter in UPDAT	"		"
12	Illegal time parameters	"		"
13	Page fault for non-demand	"		"
14	Outside segment bounds	"		"
15	Bad segments in MCALL/ MEXIT	"		"
16	Bad segment in FIX/UNFIX	"		"
19	Too big segment	"		"
20	Segment transfer error	hardware status	block no.	"
22	False interrupt		level no.	no
23	Device error	hardware status	hardware device no.	"
24	Internal interrupt		bit no.	yes
26	Mass storage time-out		program	no
27	Error in CONCT	RT prog.		yes
28	FTN I/O	Error no.	(See NORD File System)	
35	Stack error			"

APPENDIX E

THE REAL-TIME LIBRARY

The real-time library consists of a set of subroutines in BRF format.

Most of the entries are interfaces between the FORTRAN calls and the monitor calls, consisting of a MON instruction and an EXIT instruction.

In addition a set of stack operations are included, used by FORTRAN to obtain re-entrant program units.

The bit string operations according to ISA-S61.1 are also included.

Bit Operations in FORTRAN

Logical Operations

The logical operations are implemented as integer functions in FORTRAN. In the following m and n are constants, integer variables or array elements. Operations are performed on a full word bit by bit.

Inclusive or

IOR (m, n)

Logical Product

IAND (m, n)

Logical Complement

NOT (m)

Exclusive or

IEOR (m, n)

APPENDIX F

USER EXTENSIONS TO SINTRAN III

F.1 Monitor Calls

The monitor calls 170_8 through 177_8 are set aside for the special needs of a user. The corresponding subroutines can be included at system generation time.

When the monitor call is executed, the subroutine will be entered on RT program level. The user's registers are saved in a data field, to which the X register points. The registers can be reached using the displacements ZPREG, ZXREG,...etc. The B register will be equal to the user's A register.

The normal entry sequence will do:

a) Get hold of any parameter values in core.

b) Copy the X register to the B register.

c) Turn off the monitor level.

If the parameters are conforming to the SINTRAN standard calling sequence (see Appendix A), some existing subroutines can be used for the entry sequence: GET0, GET1, GET2, GET3, GET4 and GET5. These will move the parameters to the data field, where the parameters can be reached using the displacements D0...D5.

On exit, the B register should be the same as the X register on entry. Exit is then a jump to the monitor address RET.

Example:

Monitor call to add two integers.

FORTRAN: I = IPLUS (J,K)

Called subroutine:

```
        )9BEG
        )9ENT IPLUS
IPLUS,  MON   170
        EXIT
        )9END
```

Monitor call routine, US0 corresponding to MON 170:

```
US0,    JPL  I  (GET2          % GET 2 PARAMETER
                               % VALUES, X=:B
        LDA  D0,B
        ADD  D1,B
        STA  ZAREG,B
        JMP  I  (RET
)FILL
```

F.2 User Start Sequence

When the SINTRAN III system is started, an initializing RT program calls a user subroutine USTAR. In this routine the user can do his special initializing, or he can start his own RT program.

Example:

```
        USTAR,  LDA     RTPR
                JAZ     NOPR
                LDA     (PAR
                MON     100             % RT
        NOPR,   EXIT
        PAR,    RTPR
        RTPR,   OWNPR
                )FILL
```

F.3 User Restart Sequence

The restart routine being executed after power fail will call a user routine, where the user can for instance initialize process output values. This subroutine is called before the interrupt system is turned on.

Example:

```
        UREST,  LDA I   (PINI
                IOX     PROCU           % SOME PROCESS
                                        % OUTPUT
                EXIT
                )FILL
```

APPENDIX G

MONITOR CALL NUMBERS

The octal number can be used as an argument in the MON instruction.

0	RTEXT	107	DSCNT
1	INBT	110	PRIOR
2	OUTBT	111	UPDAT
3	ECHOM	112	CLADJ
4	BRKM	113	CLOCK
5	RDISK	114	Not used
6	WDISK	115	FIX
7	RPAGE	116	UNFIX
10	WPAGE	117	RFILE
11	TIME	120	WFILE
12	Not used	121	WAITF
13	CIBUF	122	RESRV
14	COBUF	123	RELES
15-41	Not used	124	PRSRV
42	OPEN (old version)	125	PRLS
43	CLOSE	126	DSET
44	Not used	127	DABST
45	DBRK	130	DINTV
46	GBRK	131	ABSTR
47	SBRK	132	MCALL
50	OPEN	133	MEXIT
51	DMAC BREAKP.	134	RTEXT
52-63	Not used	135	RTWT
64	ERMSG	136	RTON
65	QERMS	137	RTOFF
66	ISIZE	140	WHERE
67	OSIZE	141	IOSET
70-72	Not used	142	ERRMON
73	SMAX	143	RSIO
74	SETBY	144	MAGTP
75	REABT	145	ACM
76	SBSIZ	146-167	Not used
77	SETBC	170	US0
100	RT	171	US1
101	SET	172	US2
102	ABSET	173	US3
103	INTV	174	US4
104	HOLD	175	US5
105	ABORT	176	US6
106	CONCT	177	US7