THE SINTRAN III NODAL SYSTEM

Oslo University Version

T.B. Skaali

Institute of Physics
University of Oslo

POB 1048, Blindern, Oslo 3, Norway

Report 80-17

October 1980

The SINTRAN III NODAL System

Oslo University Version

T.B. Skaali
Institute of Physics, University of Oslo
POB 1048, Blindern, Oslo 3, Norway

October 1980

-ooOoo-

Throughout the report, replace the characters Æ and Å with [and], respectively.

1. Introduction

This report describes the Oslo University implementation of the NODAL system under the operating system SINTRAN III. At present, the NODAL system has been installed on two NORD-10.S computers at the Physics Department and on a NORD-100 at the Chemistry Department, University of Oslo. These computers are running with 77D, 70A and 60A versions of SINTRAN III, respectively.

The basic NODAL system is documented in a CERN Report, cf. Ref. [1] (The NODAL System for the SPS). NORSK DATA A.S does not support NODAL, hence the source programs have been supplied by CERN. At the CERN SPS Control System the NODAL programs are running under an operating system called SYNTRON, which originates from the SINTRAN II monitor developed for the NORD-1 computer. However, CERN also supports a basic SINTRAN III version of NODAL, and most of the facilities and functions described in Ref. [1] are also available under this operating system.

Apart from a few, and minor modifications, the basic part of the Oslo University NODAL system does not differ from the CERN version. However, the Oslo University implementation has been expanded with new functions which enable the user to execute many of the SINTRAN III monitor calls from the NODAL level. In particular the most important RT monitor calls have been implemented in this way, a property which renders possible the use of NODAL as a RT program administrator.

A short summary of the main features of NODAL is given in chapter 2. A NODAL subsystem which can preinterpret program statements, and thereby increase the execution speed, is described in chapter 3. The implementation under SINTRAN III is described in chapter 4, whereas chapters 5, 6 and 7 contain documentation on functions and facilities available in the Oslo University version.

2. Basic Features of NODAL

NODAL is a high level programming language based on FOCAL and SNOBOL4, with some influence from BASIC. The language was developed to operate on the computer network controlling the SPS accelerator at CERN.

NODAL is an interpretive language designed for interactive use. This is the most important aspect of the language, and is reflected in its structure. The interactive facilities makes it possible to write, debug and modify programs much faster than with compiler based language like FORTRAN and ALGOL.

The language has three special features. The first is a syntax which supports a multi-computer network operation. (The multi-computer facilities are not included on the NODAL installations at Oslo University). The second is extensive string handling features which makes it easy to write good programs for operator interaction. The third feature is the ability to add machine code functions for specific applications. A special type of such functions is the data module, Ref. 2. By means of data modules, process control equipment cornected to the computer can be handled as program variables.

Some of the examples given in Ref. [1] are specific to the operation of the SPS computer network. However, NODAL is well suited for most process control applications. NODAL also supports the international instrumentation interface CAMAC. An important feature of the interpreter is that all commands can be used in the "immediate" mode, i.e. they are executed immediately after being typed in. This is very useful for investigation of error conditions in external equipment.

3. The NODILER Preinterpreter

The inherent slow speed of the interpretation process of NODAL makes the language less suited for applications that involve time critical sections or lengthy calculations. The mean execution time for a single NODAL command is a few milliseconds. One way to gain speed is to write the most critical parts as assembly code routines that can be called as NODAL functions or subroutines. Another more general approach is compilation. The NODILER compiler is a special subprogram that can be included in a NODAL system. By means of this compiler it is possible to compile individual statements of a NODAL program. This new option is not described in Ref. [1], the use of it is however very simple, and will be illustrated below.

The NODILER compiler has been constructed such that the structure of a compiled and an uncompiled line is basically the same, cf. Ref. 3. (This compatibility is essential for the operation of the computer control network of the SPS accelerator). In fact, the NODILER generates what is termed as "threaded code", which can be described as a sequence of machine code subroutine calls to routines that perform each operation.

In average, the improvement in execution time of a compiled line versus interpretation is about a factor of 4, but the actual number for different types of statements deviates strongly from this mean value. The loop statements FOR and WHILE are obvious candidates for compilation, in particular when one of the task globals ARG(1) ... ARG(16) is used as loop counter, cf. example below. An increase in code size of a factor 2 - 3 is the penalty, plus that the NODILER subprogram occupies an additional 5K of memory space.

By typing an asterix in front of a line one signals to the NODILER that this line is to be compiled (or nodiled). The compilation operator * can be inserted by means of the EDIT command. When the line has been nodiled an eventual LIST of it will signal this that by the asterix * has been replaced by @.

Case 1

```
1.10 DIM A(2000)
1.20 FOR I=1,ARSIZE(A); SET A(I)=I

29 WORDS OF TEXT
Execution time 6.5 sec.
```

The insertion of * before FOR in line 1.20 will nodile this line. This compilation will also increase the size of the program..

Case 2

```
1.10 DIM A(2000)
1.20 @FOR I=1,ARSIZE(A); SET A(I)=I

74 WORDS OF TEXT
Execution time 2.4 sec.
```

The execution time can be reduced further by replacing the loop counter I with one of the ARG's.

Case 3

```
1.10 DIM A(2000)
1.20 &FOR ARG(1)=1,ARSIZE(A); SET A(ARG(I))=ARG(I)

53 WORDS OF TEXT
Execution time 1 sec.
```

Compilation of line 1.10 has no significant effect. Note however that Case 3 in non-compiled form executes slowly due to the use of ARG(1), around 14 sec!!

A nodiled line can be edited in the normal way, since the text of the line is retained. Error(s) in a compiled line are signalled by the replacement of the compilation operator with an exclamation mark ! . A nodiled line can at any time be reduced to interpretative code again by removal of the compilation operator.

NODAL programs that contain nodiled lines can be saved onto and loaded from files in the usual way.

4. Implementation under SINTRAN III

SINTRAN III is a multiprogramming, multi-lingual, real-time operating system which allows 3 modes of operation; real-time, time-sharing (background), and batch. The various operation modes and facilities are described in Ref. 4. NODAL can be implemented both as a RT and background system.

A NODAL system is logically structured in two parts :

1) Interpreter plus system functions

2) Working area

The working area contains the user program text and program variables. The interpreter is written as re-entrant code so only one copy is required irrespective of the number of concurrent users. This feature is utilised for generation of the real-time system, cf. chapter 4.2. However, each active program requires its own working area.

The NODAL system offers a wide range of options and support functions. The options to be included and the size of working area etc. are specified at generation time by means of assembly symbols. These assembly symbols directs the assembler to select the corresponding program units. The assembly symbols are explained in the source programs.

A list of the system variables in any specific NODAL system can be obtained by typing LISR. The resulting output gives the names, parameter types, sizes, etc. for each variable or function. Correspondingly, the HELP command lists the available NODAL commands with a minimum permissible abbreviation. (Note that a NODAL command can be abbreviated, whereas the name of functions must be typed out in full. LISR is implemented as a function, or more precisely, a call function. Another useful function is LISV, which gives a list of program variables and the content of the working area).

Under SINTRAN III two file types are used as default identifiers. The NODAL commands

SAVE, OLD, LOAD, RUN, OVERLAY, LDEF, SDEF

take :NOD as default type. Files accessed by the OPEN and string input/output functions use :SYMB as default type.

4.1 Background NODAL

The background version of NODAL can be entered from any user by means of the SINTRAN command

@RECOVER NODAL

or simply

@NODAL

Note that if "ESCAPE" is pressed while running NODAL the control is given to SINTRAN III. Return to NODAL without losing any information is obtained by

@CONTINUE

A useful function in background NODAL is the MON("COMMAND"), where COMMAND stands for any SINTRAN III command allowed from background. (The SINTRAN monitor call MON 70 is used). As an example,

MON("LIST-FILE,,,")

will produce a list of all files belonging to the user.

The QUIT command stops NODAL and puts the terminal back into control of the operating system.

No CAMAC operations are allowed from a background NODAL. This restriction is imposed by SINTRAN III. (It is, however, possible to remove this restriction by means of a patch in SINTRAN).

4.2 Real Time NODAL Programs

A Real Time program, called RT program, is a program which has its own RT description and which has been loaded into the SINTRAN III system by a special program, the RT Loader. A description of the RT Loader and the concept of RT program segments is given in Ref. 4. RT programs are known to the system by symbolic names; the RT program names.

RT programs are in general used to control external equipment and events. Under real time processing, there are four principal methods of activating programs:

operator requests
- program requests
- time scheduling
- external interrupts

The SINTRAN III system provides commands and monitor calls for RT program control. These facilities are documented in Ref. 4.

Unlike background programs, which are terminal independent (the terminal logical device number is always 1), a RT program must explicitly reserve a terminal before it can be used. The logical device number of the terminal is defined in the program code.

The SINTRAN III RT NODAL system consists of one interactive RT program assigned to each terminal, plus a number of socalled file-driven RT NODALs. The NODAL function LISP gives the name and statii for all RT NODALs of a specific system.

The system is generated with the interpreter and the functions on a segment that is common to all RT NODALs, and with their working areas placed on separate segments. This construction offers several advantages.

- only one copy of the interpreter is required in memory
- the working area segments can be assigned different attributes, such as demand/non-demand, etc.

4.2.1 Interactive RT NODAL Programs

An interactive RT NODAL is operated in very much the same way as a background NODAL, but the user has the full access to control of external equipment and to the various RT-functions of NODAL, cf. chapter 5.

Each interactive RT NODAL has been given a name

NOD<dev>

where <dev> stands for the device number of the terminal. In standard SINTRAN III systems the terminal device numbers are 1, 9, 34-39.

A interactive RT NODAL is activated from the user RT by the SINTRAN III command

@RT <progname>

However, if <progname> is trying to reserve the same terminal as the one from where the command is given, the startup of the program is suspended until the terminal is released. The @LOGOUT command releases the terminal.

The activation of an interactive RT NODAL, say from terminal 34, is carried out in the following way.

- Login under user RT
- @RT NOD34
- @LOGOUT
- NOD34 should now start by printing out a message on the terminal. If not, check terminal device number.

4.2.2 File-driven RT NODAL Programs

A file-driven RT NODAL has no interactive facilities. It can however print to a pre-assigned terminal. cf. description of the assignment function later in this chapter.

The file-driven RT NODALs are named

RTN¹ RTN2 ..... RTNX

where X stands for the total number of NODALs of this type.

A file-driven RT NODAL is always associated with a NODAL program stored on file. This program file must belong to user RT and have the same name as the RT NODAL itself. Hence the program of RTN¹ is stored on the file (RT)RTN¹:NOD.

File-driven RT NODALs can be activated by program requests, external interrupts or run on time scheduling. When activated, the RT program loads the program on the associated file into the working area and executes it.

Initially, no output terminal is assigned to a file-driven RT NODAL, i.e. the terminal device number is given the dummy value 0. Terminal assignment can be done by means of the function RTASSG(RTNx,<devno>).

RTASSG(RTN¹,34) : assign output of terminal 34 to RTN¹.

RTASSG(RTN¹,0) : detach terminal from RTN¹

Note however that the execution of the program will be suspended if the assigned terminal is already in use. This is true even if the program in question is not going to use the terminal, because the reservation of the device is done at the startup of NODAL.

The terminal assignment and the control of file-driven RT NODALs is normally done from an interactive RT NODAL.

4.3 Oslo University Revisions to the NODAL System

Some revisions and corrections to the CERN NODAL programs have been done. In addition many new functions have been implemented. Those which are of general interest are documented in chapter 5, 6 and 7, others which are (and will be) written for special applications are described elsewhere. All changes are documented in the source code.

The most important revisions to the system are

- The working area layout has been restructured in order to allow different sizes of the stack area (program and variables area) for programs within a system.
  The working area is structured as follows (in ascending address space):
  1. RT-description extension + file buffer area
  2. TEKPLOT data field (if included)
  3. Defined function area (if included)
  4. Stack area (for program text and variables)
  5. NODAL global block

- Any SINTRAN III RT program can now be controlled from a RT NODAL

- Output terminals can be assigned to file-driven RT NODALs

- The "ESCAPE" action can be disabled and enabled by RT NODALs

- The QUIT command can no longer be abbreviated

- The output formatting routine has been modified. For integer output the maximum number of significant digits has been increased from 5 to 6. Furthermore the decimal point will be suppressed for output of integer values only if the default format is used.

- Space can be allocated in the working area for a data field for TEKPLOT; a graphic program package for Tektronix terminals, Ref. 5.

The related NODAL functions are described in chapter 5. The various options are selected at generation time by means of conditional assembly symbols. The symbols and their implication are documented in the source code.

In the basic version supplied by CERN, RT operations could only be done on file-driven RT NODALs within a specific system, that is RTN1, RTN2, ..., RTNX. In the Oslo University version any SINTRAN III RT program can be controlled from a RT NODAL. This feature can be implemented for SINTRAN III systems both with and without the monitor call

MON GRTDA (= get RT-description address).

For SINTRAN III versions without this monitor call the NODAL functions RTDESC and RTDLIS must be included, cf. chapter 5.1. For SINTRAN III versions with MON GRTDA this option can be included directly in the RT-NODAL system.

5. NODAL - SINTRAN III Functions

This chapter gives a description of the NODAL SINTRAN III functions. None of these functions are documented in Ref. 1. Note that chapter 12.2 of Ref. 1 does not apply for SINTRAN III NODAL systems.

Of the RT-functions described in this chapter, the following ones has been supplied by CERN:

RT, RTSET, ABSET, INTV, KILL, CONCT, DSCNT

Most of the CAMAC functions written for Oslo University version, cf. chapter 5.4, are similiar to those implemented at CERN.

5.1 RT Program Control

The functions RTDESC and RTDLIS are designed for SINTRAN III systems without the monitor call MON GRTDA.

RTASSG Assign output terminal to file-driven RT NODAL
CALL RTASSG(<prog>,<device no>)
Example : CALL RTASSG(RTN1,34)
assign terminal 34 to RTN1
Example : CALL RTASSG(RTN1,0)
detach current terminal from RTN1
Terminal assignment must only be done on a passive program. The function LISP gives the statii for the file-driven RT NODALs.

RTDESC Declare non-NODAL RT program
CALL RTDESC(<entry no>,<prog>,<RT-desc. address>)
The program name and RT-description address are stored in a table which is common to all RT NODALs. The parameter <entry no> defines the entry in this table. Up to 6 non-NODAL RT programs can be defined. The octal value of <RT-desc. address> is displayed by the SINTRAN command LIST-RT-PROGRAMS.
Example : CALL RTDESC(2,SCAN,%253'13)
Define the RT program SCAN with RT-description address 253'13 (in octal) as the second entry of the table.

RTDLIS List declared non-NODAL RT programs
CALL RTDLIS

5.2 "ESCAPE" Control Functions

A NODAL program can be stopped by pressing the "ESCAPE" key on the terminal. A program will also be stopped by (CTRL)B if it is in a ASK or $ASK command. Hence inadvertently use of these keys can stop the execution of a critical program. For RT NODALs the "ESCAPE" action can be controlled by the functions ESCON and ESCOF. Care should be taken not to start an endless loop if the "ESCAPE" action has been disabled!!!

The "ESCAPE" action is automatically re-enabled when program execution is terminated, either in a normal way or by an error.

ESCOF Disable "ESCAPE" action

ESCON Enable "ESCAPE" action

5.3 General RT Functions

The parameter <prog> can be the name of a file-driven RT NODAL or any other RT program. If SINTRAN does not contain MON GRTDA, the name and RT-description address of an "other" program must have been declared by means of the function RTDESC, cf. 5.1.

RT Start RT program
CALL RT(<prog>)
Example : CALL RT(RTN1)

RTSET Start RT program at relative time
CALL RTSET(<prog>,<no of time units>,<time unit>)
Example : CALL RTSET(RTN2,10,2)
start RTN2 in 10 seconds

ABSET Start RT program at given time of day
CALL ABSET(<prog>,<seconds>,<minutes>,<hours>)
Example : CALL ABSET(RTN3,0,12,17)

INTV Prepare RT program for periodic execution
CALL INTV(<prog>,<no of time units>,<time unit>)
Example : INTV(RTN1,60,2)
prepare for execution every 60 seconds

KILL Disconnect and abort RT program
CALL KILL(<prog>)
Example : KILL(RTN1)
KILL is the NODAL equivalent of MON 107 (DSCNT) plus MON 105 (ABORT)

CONCT Connect RT program to interrupt line
CALL CONCT(<prog>,<logical unit>)
Example : CALL CONCT RTN4,£401)
connect to line 401 (octal)

DSCNT Disconnect RT program
CALL DSCNT(<prog>)
Example : CALL DSCNT(RTN4)
NODAL equivalent of MON 107 (DSCNT)

PRIOR Change priority of RT program
SET IP=PRIOR(<prog>,<priority>)
The return value of the function is the old priority.
Example : SET IP=PRIOR(NOD34,100)
Change the priority of NOD34 to 100 (decimal)

FIX Fix segment in memory
CALL FIX(<segment number>)
Example : CALL FIX(£45)
If <segment number> refers to a non-existent or demand segment, the calling program will be aborted

UNFIX Unfix segment from memory
CALL UNFIX(<segment number>)
Example : CALL UNFIX(£45)

FIXC Fix segment in physical memory
CALL FIXC(<segment number>,<first physical page>)
Example : CALL FIXC(£45,48)
Fix segment number 45 (octal) in contigous physical memory from page 48 (decimal), i.e. address 140000 (octal). If <segment number> refers to a non-existent or demand segment, the calling program will be aborted

RTWT Set calling program in waiting state
Example : RTWT
The calling program will be set in a waiting state for an unspecified amount of time. The next time the program is started, for instance by an

interrupt, it will continue with the statement following the RTWT call

RTOFF The execution of a program is inhibited
CALL RTOFF(<prog>)
After this call the RT program is set in RTOFF state and will not be allowed to be started before the state is removed by the RTON call.

RTON Reset the RTOFF state of a program
CALL RTON(<prog>)

RESRV Reserve a logical unit to calling program
SET RS=RESRV(<logical unit>,<read/write>,
<return flag>)
This function is used to reserve a device identified by the device number <logical unit>, cf. Ref. 4. If <read/write> equals zero, the input part is reserved for a two-way unit, if it equals one, the output part is reserved, and if it equals two, both the input and the output parts are reserved simultaneously. If the unit is already reserved, the program will be set in a waiting state if <return flag> equals zero, if <return flag> is non-zero there will be an immediate return with negative function value. If the unit is free, there will be immediate return with zero function value, and the logical unit will be reserved.

Examples.
SET RS=RESRV(32,2,1)
Reserve Magnetic tape controller 1, unit 0, for input and output. If the return value is -1, the controller is already reserved by another program.

SET RS=RESRV(&200,1,0)
Reserved internal device 200 (octal) for output. Set calling program in waiting state if device is already reserved by another program. The waiting program will be reactivated when the device becomes ready for reactivation.

RELES Release a logical unit from calling program
SET RS=RELES(<logical unit>,<read/write>)
The reserved unit will be released if it is reserved for the calling program. If <read/write> equals zero, the input part is released for a two-way unit, if it equals one, the output part is released, and if it equals two, both the input and the output parts are released. If the unit is

reserved by another program, there will be an immediate return with negative function value, otherwise the return value will be zero.
Example.
SET RS=RELES(32,2)
Release Magnetic tape controller 1, unit 0, from the calling program.

PRSRV Reserve a logical unit for a RT program
SET RS=PRSRV(<logical unit>,<read/write>,<prog>)
The logical unit will be reserved for the RT program specified by the parameter <prog>. For a two-way unit the parameter <read/write> has the same meaning as for RESRV. If the unit is already reserved a negative function value is returned. If not, zero is returned and the reservation will be performed.
Example.
SET RS=PRSRV(32,2,SHIVA)
Reserve Magnetic tape controller 1, unit 0, input and output, for the RT program SHIVA.

PRLS Release a logical unit from the program having reserved it
CALL PRLS(<logical unit>,<read/write>)
The parameters have the same meaning as for RESRV and RELES.
Example.
CALL PRLS(512,2)
Release Floppy disc from the program having reserved it.
Note.
This function should not be used for grabbing peripheral devices from another users.

WHDEV Where is logical unit
SET WH=WHDEV(<logical unit>,<read/write>)
If the logical unit is reserved, the address of the RT description of the program occupying the unit will be returned as the function value. If the unit is free, zero will be returned.
<read/write> = 0 for input part,
1 for output part

GRTNA Get name of RT program
$SET S=GRTNA(<RT description address>)
The name of the RT program will be returned as a string. The return string will be empty if no name is found, i.e. illegal value of the <RT description address>.
Example.

TYPE GRTNA(WHDEV(32,0))
Type name of RT program having reserved the magnetic tape station

RTDSC Copy RT description to NODAL array
SET IC=RTDSC(<prog>,<array>)
The RT description of <prog> will be copied to a NODAL array specified by <array>. The array must be of type integer, and the minimum size must be 26 elements. The function return value gives the number of devices connected to the RT program.
Example.
DIM-INT A(26); SET IC=RTDSC(SCANX,A)

5.4 CAMAC Functions

CAMAC Execute CAMAC cycle
SET Z=CAMAC(C,N,A,F) for read/control F
SET CAMAC(C,N,A,F)=Z for write F
CAMAC status register returned for control F

CAMAQ Execute CAMAC cycle with Q-response
SET Z=CAMAQ(C,N,A,F,Q) for read/control F
SET CAMAQ(C,N,A,F,Q)=Z for write F
Return value of Q = Q-response 0 or 1

INIT Initialise a CAMAC crate
CALL INIT(<crate no>,<interrupt level>)
Execute CAMAC Z-cycle
Clear MASK register
Set interrupt level to 10, 11 or 12,
normally 12.
Enable ERROR, RT and LAM demand

CONTRO Read/write CAMAC status register
SET CONTRO(<crate no>)=Z
SET Z=CONTRO(<crate no>)

MASK Read/write CAMAC MASK register
SET MASK(<crate no>)=Z
SET Z=MASK(<crate no>)

UMASK Masked set MASK register
SET UMASK(<crate no>)=<GL no>
Masked set bit in MASK register corresponding to <GL no> = 1-16

GL Read CAMAC GL register
SET Z=GL(<crate no>)

IDENT Read last CAMAC IDENT
SET Z=IDENT(<crate no>)

IOX Issue direct IOX command
SET IOX(<hardware device no>)=Z
SET Z=IOX(<hardware device no>)

ASSIGN Assign CAMAC GL to logical device no
CALL ASSIGN(<logical dev. no>,<crate no>,<GL no>)
<GL no> = 0 for high priority interrupt on level 13
<GL no> = 1-16 for normal LAMs from the crate to interrupt level defined by the INIT call, normally 12.

5.5 Magnetic Tape and Floppy Disc Handling

The monitor call MAGTP is used to access a magnetic tape or cassette tape from a user program, to read, write or position the tape unit to a file or record. MAGTP can also be used to position a floppy disc and to transfer data to/from the diskette. The monitor call is described in more detail in Ref. 4.

This monitor call has been implemented as a NODAL function of the same name. The function can be called both from background and RT NODAL

MAGTP Execute SINTRAN III monitor call MAGTP
SET RW=MAGTP(<logical device no>,<function code>,
<NODAL data element>,<max. words>,<repeat>)

The parameters have the following meaning :

<logical device no> = device number of the desired unit, cf. Ref. 4 (SINTRAN III Users Guide).

<function code> = specific function to be performed, cf. Ref 4.

<NODAL data element> = NODAL integer array or simple variable. For read/write operations, the data element must be an integer array to/from which data will be transferred. For function codes 5 to 24 (octal) no data is transferred and the data element can be a simple variable.

<max. words> = maximum number of words (16 bit integers) to read or write. If <max. words> = 0, the word count is defined to be the length of the integer array. If the value of <max. words> exceeds the array length a NODAL "file error" return will take place.

<repeat> = repetition count, defines number of operations of the specified type. Only meaningful for record and file skip operations.

Function return value = number of words read.

For function codes 5 to 24 (octal) the parameters <NODAL data element> and <max. words>, and the function return value, are ignored.

Examples :
Z is a simple variable and A an integer array
SET RW=MAGTP(32,&13,Z,0,1) rewind magnetic tape
SET RW=MAGTP(512,&10,Z,0,4) advance over 4 EOF marks on floppy disc
SET RW=MAGTP(32,0,A,128,1) read 128 words from record on magnetic tape to array A
SET RW=MAGTP(512,0,A,0,1) read record from floppy disc to array A
SET RW=MAGTP(512,1,A,0,1) write binary content of array A to floppy disc

5.6 Input/Output and I/O Buffer Functions

The input/output functions INBT, OUTBT, INPUT, INPC, OUTPUT and OUTC are desribed in Ref. 1. These functions have been supplemented with the buffer functions CIBUF, COBUF, ISIZE and OSIZE. The functions can be called both from background and RT NODAL.

Note that the NODAL functions INBT and OUTBT transfer 8 bit bytes. If 16 bits transfer is required, use DINOUT.

The function LASTC can only be called from RT programs.

LASTC Get last character typed on a terminal
SET Z=LASTC(<logical number>)
This function is used to get the ASCII value (with parity bit removed) of the last character typed on a terminal.
Example.
SET Z=LASTC(1DEV)
Get ASCII value of last character typed on the attached terminal.

CIBUF Clear device input buffer
CALL CIBUF(<device no>)

COBUF Clear device output buffer
CALL COBUF(<device no>)

ISIZE Return number of bytes in input buffer
SET NB=ISIZE(<device no>)

OSIZE Return number of bytes in output buffer
SET NB=OSIZE(<device no>)

DINOUT Execute S-III monitor calls MON INBT and OUTBT
SET Z=DINOUT(<device no>) : INBT
SET DINOUT(<device no>)=Z : OUTBT
DINOUT may be used to transfer 16 bit words (SINTRAN option)

The buffer functions are useful for instance in communications between RT programs by means of internal devices.

One program can write onto an internal device.

```
1.10 SET DN=%200
1.20 SET Z=RESRV(DN,1,0)
1.30 $SET OUTC(DN)="HALLO BLOCKHEAD"
1.40 SET Z=RELES(DN,1)
```

Another program can read this message.

```
1.10 SET DN=&200
1.20 SET Z=RESRV(DN,0,0)
1.30 IF ISIZE(DN)>0; TYPE INPUT(DN)
1.40 SET Z=RELES(DN,0)
```

5.7 Functions for Fast Input/Output of Integers Arrays

String input/output under NODAL is slow. Hence the reading or writing of formatted data is very time consuming. As experimental data is often stored as elements of an integer array, two functions have been implemented for reading and writing integer arrays from or onto file.

The referenced data arrays must be single dimensional arrays of single or double precision. The concept of double precision integer arrays is discussed in chapter 6. Two formatting types are implemented; the normal integer format and a socalled free format. Each number written in integer format is terminated by CR and Linefeed. In the free format only the sign, if negative, and significant digits are written, terminated by comma (,). Zeros are marked by comma only. CR and Linefeed are written to limit each line to maximum 72 characters.

The functions can be called both from background and RT NODAL.

DINWR Write content of integer array onto file
SET WS=DINWR(<file number>,<array>,<first element>,
<last element>,<field length>)

<file number> = file logical number as returned by OPEN FILE operation

<array> = NODAL single or double precision integer array

<first element> = index of first array element to be written

<last element> = index of last array element to be written
If <first element> = <last element> = 0, the whole array is written onto file.

<field length> = field length of integer format, range 1 - 16. If <field length> = 0 the data is written in free format.

Function return value = status, 0 if OK, -1 if the field length is too short. In the latter case the surplus digits will be dropped.

DINRD Read integer numbers from file to array
SET RS=DINRD(<file number>,<array>,<first element>,
<last element>,<format>)

<file number> = file logical number as returned by OPEN FILE operation

<array> = NODAL single or double precision integer array

<first element> = index of first array element to be read into

<last element> = index of last array element to be read into
If <first element> = <last element> = 0, the data will be stored from the first array element.

<format> = format flag, 0 for free format, 1 for fixed field length.

Function return value = number of data elements read. The sign bit is set if number too big for single precision representation or bad format.

Examples.
In the following examples, A denotes an integer array and FI a file number that has been returned from an OPEN call, for instance
SET FI=OPEN("W","(RT)DATA-FILE-1")
SET WS=DINWR(FI,A,1,20,0) Write the first 20 elements of A onto file in free format

$SET OUTPUT(FI)=DATE
$SET OUTPUT(FI)="EXPERIMENT NO 2"
SET WS=DINWR(FI,A,0,0,12) Write date and text followed by all elements of the array with field length 12 onto file

5.8 SINTRAN III Monitor Calls - NODAL Functions

The following table gives the relation between SINTRAN III Monitor Calls and NODAL functions.

Monitor Call No	Monitor Call Name	NODAL Function Call Name	Type	Callable Backgr.	RT
1	INBT	INBT	10	Yes	Yes
2	OUTBT	OUTBT	9	Yes	Yes
11	TIME	TIME	10	Yes	Yes
13	CIBUF	CIBUF	11	Yes	Yes
14	COBUF	COBUF	11	Yes	Yes
26	LASTC	LASTC	10	No	Yes
27	RTDSC	RTDSC	10	No	Yes
43	CLOSE	CLOSE	11	Yes	Yes
50	OPEN	OPEN	10	Yes	Yes
66	ISIZE	ISIZE	10	Yes	Yes
67	OSIZE	OSIZE	10	Yes	Yes
100	RT	RT	11	No	Yes
101	SET	RTSET	11	No	Yes
102	ABSET	ABSET	11	No	Yes
103	INTV	INTV	11	No	Yes
104	HOLD	WAIT-TIME		Yes	Yes
105	ABORT	KILL (*)	11	No	Yes
106	CONCT	CONCT	11	No	Yes
107	DSCNT	DSCNT	11	No	Yes
110	PRIOR	PRIOR	10	No	Yes
113	CLOCK	DATE	24	Yes	Yes
115	FIX	FIX	12	No	Yes
116	UNFIX	UNFIX	12	No	Yes
122	RESRV	RESRV	10	No	Yes
123	RELES	RELES	10	No	Yes
124	PRSRV	PRSRV	10	No	Yes
125	PRLS	PRLS	11	No	Yes
135	RTWT	RTWT	18	No	Yes
136	RTON	RTON	10	No	Yes
137	RTOFF	RTOFF	10	No	Yes
140	WHDEV	WHDEV	10	No	Yes
144	MAGTP	MAGTP	10	Yes	Yes
147	CAMAC	CAMAC	8	No	Yes
		CAMAQ	8	No	Yes
150	GL	GL	10	No	Yes
152	GRTNA	GRTNA	24	No	Yes
153	IOXN	IOX	8	No	Yes
154	ASSIG	ASSIGN	11	No	Yes
160	FIXC	FIXC	12	No	Yes

(*) KILL = MON 107 + MON 105

6. Functions for Double Precision Integers

Double precision integer is not implemented as data type in NODAL. In many applications, in particular data accumulation into integer arrays, double precision may be needed. The functions described below handle the columns of integer arrays of dimension (2,x) as double integers. A double precision integer array of length = 100 is declared by DIM-INT A(2,100). The elements A(1,i) and A(2,i) together constitutes a signed 32 bits integer.

In order to freely use single and double precision integer arrays as parameters, the functions accept the following array types :

```
DIM-INT A(2,x)
DIM-INT B(1,x)
DIM-INT C(x)   A = C(x,1) A
```

Note that the maximum number of rows in an array is 4095. However , for one-dimensional arrays this problem can be circumvented by defining the number of rows = 1, i.e

```
DIM-INT EA(1,6000)
```

DINT Double precision integer/floating conversion
SET Z=DINT(<array>,<column>)
SET DINT(<array>,<column>)=Z
The array must be declared as shown above.
Example : SET Z=DINT(A,3)
SET DINT(A,5)=1234567
TYPE DINT(A,5)

ZERA Store zero in integer array
CALL ZERA(<array>,<col1>,<col2>)
Store zero in <array> from column <col1> to <col2>, both inclusive.
If <col1>=<col2>=0, zero is stored in all elements.
Example : CALL ZERA(A,0,0)

INTA Integrate content of array
SET Z=INTA(<array>,<col1>,<col2>)
The parameters have the same meaning as for ZERA.
Example : TYPE INTA(B,0,0)

ADDA Add the contents of two arrays
CALL ADDA(<array1>,<array2>,<array3>,
<col1>,<col2>,<col3>)
Add arrays <array1> and <array2> and place result in <array3>. Start with columns <col1>, <col2> and <col3> in the respective arrays. The operation is

terminated when either array is exhausted. The destination array may be one of the source arrays.
Example : CALL ADDA(A,B,C,1,1,20)
CALL ADDA(A,B,A,1,1,1)

MAXA Find index for maximum value
NODAL read only function
SET Z=MAXA(<array>,<col1>,<col2>)
Find index for maximum value in array <array> between columns <col1> to <col2>, both inclusive. If <col1> = <col2> = 0, scan whole array.
Example : SET Z=MAXA(A,0,0)
TYPE A(MAXA(A,1,10))
Type maximum value of first 10 elements of single precision array A
TYPE DINT(A,MAXA(A,1,10))
Type maximum value of first 10 elements of single or double precision array A

7. Display Functions

Two functions have been written for a CAMAC based graphic display system; a Tektronix 603 storage scope controlled by three CAMAC modules.

SEN SDD-2015 Storage Display Driver
SEN FDD-2012 Display Driver
SEN CH-2018 Character Generator

DSDEF Define CAMAC crate no and station numbers
CALL DSDEF(C,N2015,N2012,N2018)

DSSPEC Display content of integer array as a spectrum
CALL DSSPEC(<array>,<ybase>,<chn1>,<chn2>,
<scale factor>,<marks>)
The single or double precision integer array <array> constitutes a data spectrum where the channels are numbered from 0 to <no of columns>-1. The spectrum is displayed from channel <chn1> to channel <chn2>, with y-base = <ybase>. The value of <ybase> can be from 0 (zero) to 1023 (points can be displayed on the screen in a 1024x1024 matrix). The horizontal layout of the displayed section is adjusted to fill the screen. The total height of the screen corresponds to 2**<scale factor> counts. Markers are displayed as vertical bars in the specified channels. The parameter <marks> stands for an integer array with four elements, for instance DIM-INT MA(4). In this

array MA(1-2) define the main markers, whereas MA(3-4) define shorter auxiliary markers. A negative value of MA(i) will suppress the corresponding marker.

```
Example
DIM-INT A(1024); % Data array
DIM-INT MA(4); SE MA(1)=200; SE MA(2)=500
SE MA(3)=335; SE MA(4)=-1
DSSPEC(A,0,0,1023,12,MA)
```

Display array A as a spectrum with ybase = 0 and vertical full scale = 2**12 (= 4096). Main markers are in channels 200 and 500, one auxiliary marker in channel 335.

8. TEKPLOT Program Package

TEKPLOT is a program package for generation of graphics on Tektronix terminals of type 4006 and 4013. The package is written as a set of NODAL functions. TEKPLOT is documented in Ref. 5.

Acknowledgements

The author wants to express his thanks to the SPS Division at CERN for the NODAL source programs and for many inspiring discussions, and to Dr. Ola Sveen, NORSK DATA, for help during the implementation.

References

1. M.C. Crowley-Milling and G.C. Shering, THE NODAL SYSTEM FOR THE SPS, CERN 78-07

2. M.C. Crowley-Milling, The Data Module, The Missing Link in High Level Control Languages, Third International Conference on Trends in on-line computer systems, Sheffield 27-29 March 1979

3. J. Altaber and P.D. Van der Stok, The NODILER: a preinterpreter for NODAL, CERN/SPS/ACC/PVdS/Rep. 79-4

4. SINTRAN III - User's Guide, Publ. no ND-60.050.08

5. O. Liestol, TEKPLOT, a graphics program package for NODAL, Thesis for the cand.real degree, Oslo University 1980.

Appendix A

EXAMPLES ON THE RT USE OF NODAL

The interactive facilities of NODAL makes it a powerful system for setting up and handling RT programs. In many ways NODAL can be used as a RT operating system.

The administration of a set of RT programs, written in any language, can be performed by an intercative RT NODAL. Within a RT NODAL system, the file driven NODALs can be activated by all four principal methods; operator requests, program requests, time scheduling and external interrupts. An interactive RT NODAL can be reactivated by an external interrupt, an example is given below.

Example 1

Assume that the file (RT)RTN3:NOD contains a NODAL program that is to be executed every 60 seconds. The program must terminate itself by the command QUIT. From an interactive NODAL this task can be set up by the following commands.

```
INTV(RTN3,60,2)
RT(RTN3)
```

The same commands can be used for any RT program, say SCANX, provided that RTN3 is replaced by SCANX in the commands above. The command

```
KILL(RTN3)
```

will disconnect and abort the program.

Example 2

If RTN3 is to be started by an interrupt from CAMAC crate no 2 with Graded Lam 12, the program can be connected via an assigned logical device number. In SINTRAN III the octal device numbers 400-437 are used for CAMAC connect interrupts.

```
ASSIGN(Æ400,2,12)
CONCT(RTN3,Æ400)
```

The command

DSCNT(RTN3)

will disconnect the program.

Example 3

When connecting a RT program to a CAMAC interrupt, the following three conditions must be met.

i) The LAM Demand bit of the Crate Controller Status (COST) register must be set. This bit is set during the INIT sequence and not cleared during the interrupt handling, hence one usually don't have to worry about this bit.

ii) The bit in the Crate Controller MASK register which corresponds to the specified Graded Lam number must be set. There is a one-to-one correspondence between the Graded Lam number (1-16) and the bit number of the 16 bits MASK register. The MASK bit is automatically cleared by the interrupt identfy instructions of SINTRAN III. Therefore this bit must be set up again by the user program. In NODAL this can be done by the function UMASK.

iii) The CAMAC module itself must be enabled for interrupt (Lam). Further information are given in the manuals.

The following program, which can be run by an interactive as well as a file driven RT NODAL, connects itself to an interrupt from crate no 1 with Graded Lam no 15. After setting up the module for some action that will generate a Lam in due time, the program suspends itself until the interrupt arrives. In this example the name of the RT NODAL is denoted by NOD34.

```
2.10 SET C=1; SET LN=15; SET LD=£400
2.20 ASSIGN(LD,C,LN); CONCT(NOD34,LD)
2.30 Enable CAMAC module
     Set up some action
2.40 DO 3; % Wait for Lam
2.45 GOTO 2.30  if not finished
2.50 DSCNT(NOD34)
2.55 Continue ...

3.10 SET UMASK(C)=LN
3.20 RTWT
3.30 Clear module Lam, for instance by A(0)F(10)
```

Of course, if the interrupt never arrives, the program will remain dormant.