SPS/ACC/PvdS/MANUAL 81-18

Compiled NODAL Data Module Systems Manual

P.D.V. van der Stok

GENEVA
1981

CERN LIBRARIES, GENEVA

CM-P00070598

The work described here was done in close collaboration with
J. Altaber, V. Frammery, C. Gareyte and T. Stokka.

E. d'Amico, W. Kalbreier, P. Nathan, H. Verhagen and M. Tyrrell
have suggested improvements.

P. Brummer, M. Collins, F. Ghinet and L. Jirden
assured the integration into the SPS computer network.

Technical support for the ACC came from C. Guillaume and R. Rausch.

CONTENTS

1. INTRODUCTION 1

2. IMPLEMENTATION PRINCIPLES 2

3. THE NODAL COMPILER 5
 3.1 Compiler operating procedure 5
 3.2 Compiler list file format 7
 3.3 Symbol table file 8
 3.4 System variable file 8

4. THE DATA MODULE CONSTRUCTION ELEMENTS 10
 4.1 Property table 10
 4.2 The Data Module body 12
 4.3 Protection scheme 13
 4.4 Data table 14

5. MODE OF OPERATION 16
 5.1 Installation in the NORD computer 16
 5.2 Interactive testing in the NORD computer 16
 5.3 Installation in ACC 17
  5.3.1 Functional description of an ACC-based Data Module 17
  5.3.2 Software packages for Data Module support in ACC 17
  5.3.3 Interrupt level programming 19
  5.3.4 Hardware access from the ACC 20
   5.3.4.1 CAMAC access 21
   5.3.4.2 MPX access 21
   5.3.4.3 CIMBUS serial access 23
 5.4 Interactive testing of Data Modules in ACC 23

References 24

APPENDIX I 25

APPENDIX II 29

APPENDIX III 41

1. INTRODUCTION

The Super Proton Synchrotron (SPS) control system assures proper access to the accelerator equipment through a unique software interface: the Data Module[1–3].

One module accesses all equipment belonging to one family. In its simplest form the equipment is accessed through a function call with two standard parameters as shown in the example below:

```
SET    A = VACPMP(EQ,#PTY)
SET    VACPMP(EQ,#PTY) = B
```

The family of the equipment is defined by the name of the function (VACPMP ⇆ vacuum pump). The first parameter 'EQ' defines the equipment number within the family. The second parameter '#PTY' defines the action which has to be performed. The relative position of the function call and the '=' sign defines the nature of the action (Read in case 1, Write in case 2 of the example).

Until recently, Data Modules were written in the manufacturer's assembly language (MAC) or in the intermediate language (NPL). At system generation time, they were embedded in the operating system and loaded into the NORD computers where they can be called from NODAL interpretive programs.

This mode of operation has been successfully applied for the construction of the SPS, and more than 90 Data Modules[4] have been manufactured in this way, requiring a lot of effort for debugging and improving them, and for their continual maintenance. Indeed, a reduction in manpower availability, together with the demand for new Data Modules for the $p\bar{p}$ project, made it desirable to improve the programming environment for Data Module production and maintenance.

In addition, the recent advent of microprocessor based CAMAC modules (ACC) has made it possible to relieve the NORD computer CPU of Data Module execution, by exporting most of the Data Module code into the ACC[5]. In this context it is desirable

- to make the writing and the subsequent modification of data modules easier;
- to provide good debugging tools;
- to ensure maximum transportability.

The obvious solution to these requirements is to write the Data Module in NODAL language; the high-level code can then be interpreted in the debugging phase, and compiled when it is put into real-time operation.

This scheme requires the following elements:

- a NODAL interpreter
- a NODAL compiler
- a set of Data-Module-oriented NODAL functions and commands.

These elements are available on the NORD computers and on the ACC-TMS 9900; the first two elements have been described in previous publications[6–9]. This manual will describe all three elements, and refer to the other publications when the subject is outside the Data Module scope.

2. IMPLEMENTATION PRINCIPLES

Three phases can be discerned during the development of a Data Module:

- the writing and testing of the NODAL Data Module in an interactive way;
- the compilation and linking-loading of the NODAL Data Module;
- the execution of the compiled Data Module.

The three phases will be described with the aid of three separate diagrams.

In Fig. 1, the Data Module source code is written in NODAL and loaded into the text buffer of the NODAL interpreter. The latter can be situated in the NORD as well as in the ACC.

The execution of the Data Module code can be started by calling the dedicated function DMSCAL through an immediate command such as

```
   > TYPE   DMSCAL(EQ,#PTY)
or > SET    DMSCAL(EQ,#PTY) = 1234
```

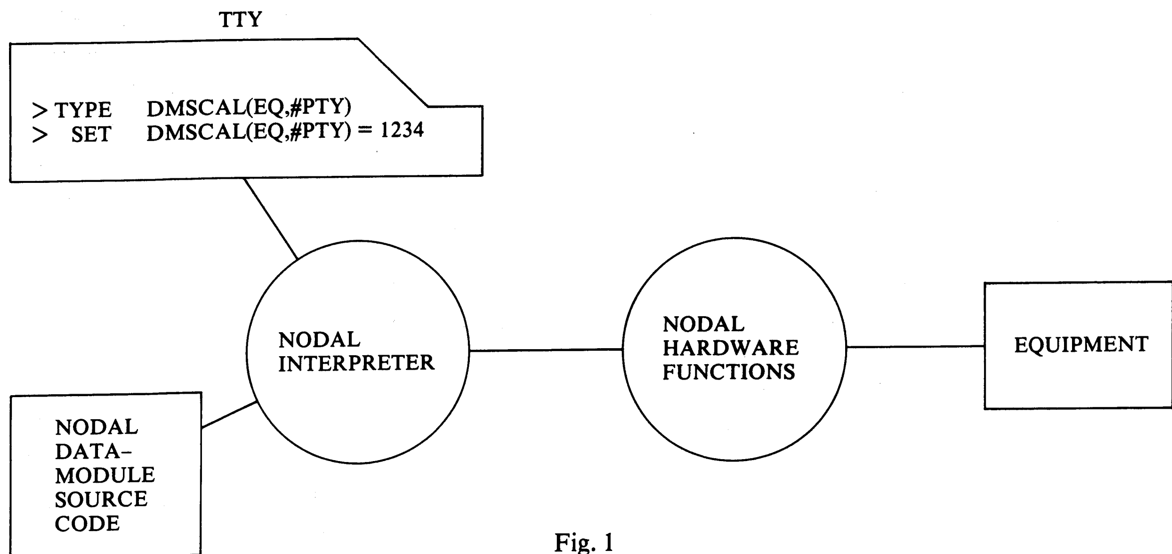


Fig. 1

Figure 2 shows how the source code can be compiled for the two target machines (NORD and ACC). For the NORD computer the object code is later linked and loaded into the list of Data Modules.

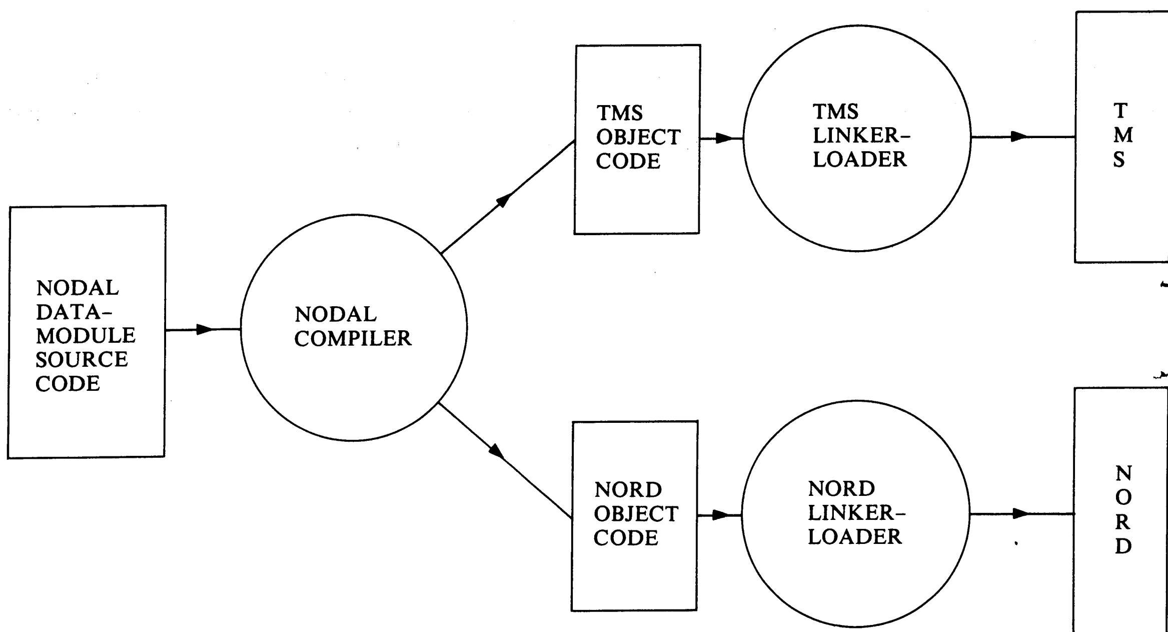


Fig. 2

For the ACC the object code is down-line loaded into the ACC memory, or better still, burned into EPROM. For the ACC Data Module an entry is defined in the Data Module list of the NORD computer.

Figure 3 shows how the Data Module calls are routed to the Data Module code.

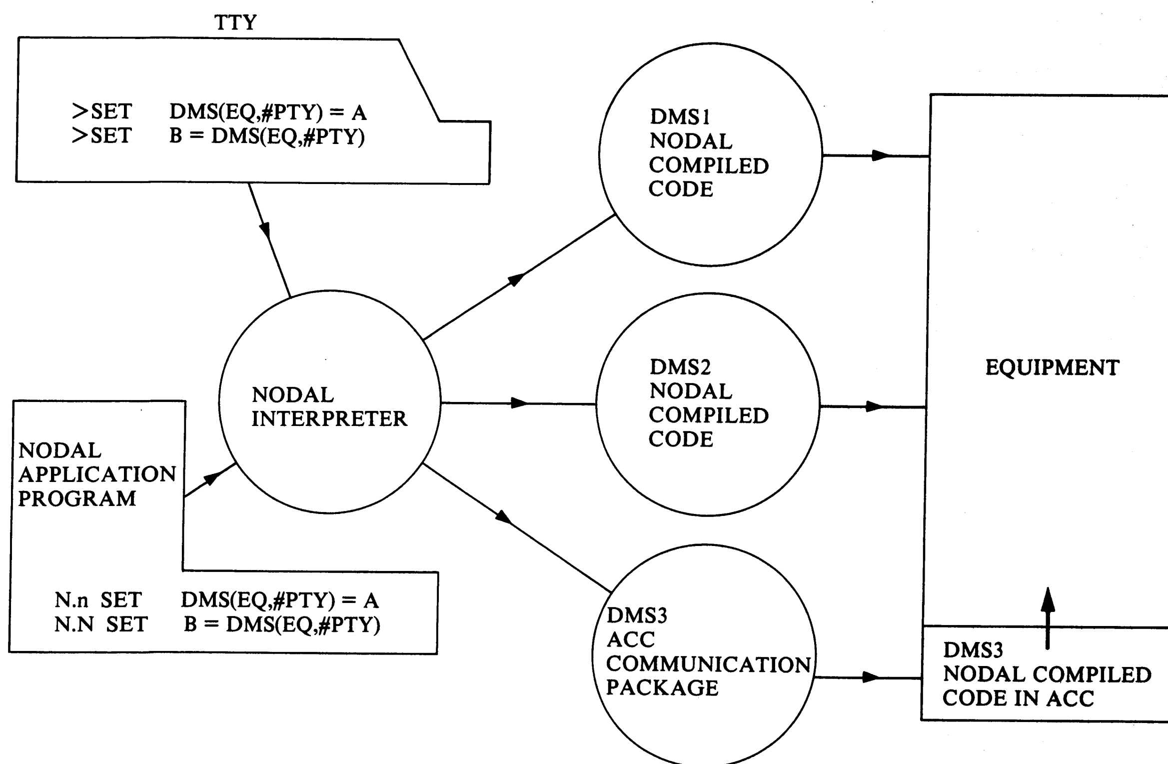


Fig.3

For a NODAL program no difference exists between a compiled Data Module residing in the NORD computer or one residing in the ACC-TMS. In both cases the compiled Data Module (e.g. DMS) can be accessed by a NODAL interpretive program in the usual way:

```
SET  DMS(EQ,#PTY) = A
SET  B = DMS(EQ,#PTY)
```

The implementation of the Data Module in an ACC raises a slight problem there.

In the SPS context, access to the accelerator hardware is performed through the MPX control unit, which is housed in the same CAMAC crate as the ACC. This MPX control unit can be accessed both by the ACC and the NORD host. To avoid access conflict, the ACC-TMS or the NORD is master of this MPX control unit. In the first case, if the NORD host has to access the UC, it has to ask the ACC to access the MPX link. For this purpose, the description of the MPX control unit in the NORD host is modified and the MPX request is routed transparently to the ACC, which executes the MPX functions and sends back eventual results.

In the second case, when the NORD is master of the UC, the UC is reserved in the NORD before the Data Module call is routed to the ACC.

Independently of the language in which the Data Module has been written, the following three elements are discerned (see Fig. 4):

- property table
- Data Module code
- data table

The property table specifies the entry point of the property paragraph to be executed for each property.

The Data Module code is structured into:

- an introduction, (initialization phase);
- a property paragraph, where the appropriate action is performed on the equipment;
- a conclusion, where errors can be returned.

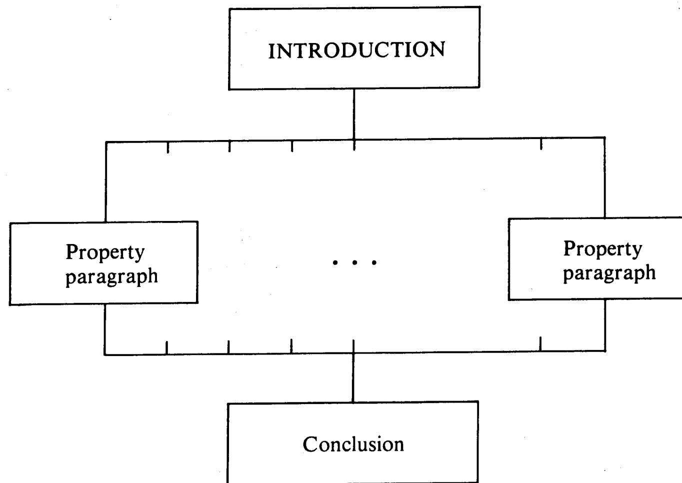


Fig.4

The Data Module code together with its property table is called the Data Module body.

The data table contains the living parameters of the Data Module; it can be represented as a two-dimensional array, each row representing a piece of equipment.

When the NODAL language is used for writing a Data Module, these elements will take the following shapes:

- The property table is a two-dimensional NODAL array, where the paragraph entry points are line numbers of the NODAL Data Module code.
- The Data Module code is a normal NODAL program. Its execution is started at the NODAL line with the lowest line number. The property paragraph starts at the line number specified in the property table and execution finishes at an END statement.
- The data table is in the form of a NODAL array for the ACC, and of a NODAL element for the NORD.

The Data Module code uses functions which are dedicated to Data Module information flow. The three most important ones are:

- PROBR, which performs the property branching (standard properties included) and handles the access protection;
- PUT, which sends data to the calling program;
- GET, which receives data from the calling program.

3. THE NODAL COMPILER

The NODAL compiler is implemented under the NORD operating system SINTRAN-III; it is built up of two parts:
- the compiler;
- symbol and system variables predefinition processor.

The compiler handles a subset of the NODAL language. This subset consists of the DATA handling facilities and mathematical operations. It does not support interactive programming facilities (a feature of an interpreter) and does not support string handling. The compiler operates in two passes. In a first pass the NODAL commands are converted to an intermediate code of a virtual machine. The instruction set of this virtual machine acts on a floating-point accumulator which uses one word of 16 bits for the exponents and two words of 16 bits for the mantissa. The first pass is common to all target machines.

The second pass of the compiler is target-machine dependent: it converts the intermediate code into machine code for the target machine.

To guarantee identical program calculation results for all target machine, a full NORD floating-point package is developed with each code generator.

The symbol and system variable predefinitions processor allows four operations:
- The definition of system variables with predefined values. They will be located in a area common to all routines compiled in one pass.
- The predefinition of symbol values. In contrast to the variables, their values cannot be changed during the execution of a program. They represent the ASCII representation of certain numbers.
- The predefinition of routines which can be called from the compiled code and which are user dependent.
- The definition of NODAL variables and arrays with predefined values. They can later be loaded into the NODAL buffer of the NODAL interpreter through the commands OLD and LOAD.

3.1 Compiler operating procedure

The NODAL compiler is activated by typing:

@NODCOMP–NORD

for the NORD target machine, and

@NODCOMP–9900

for the TMS–9900 target machine.

Whenever the command processor expects the operator to enter a command it outputs a dollar sign ($). A command consists of a command name followed by parameters if necessary. Several commands, along with all required parameters, may be written on the same line.

The command name consists of one or more parts separated by hyphens (–). Each part of the command name may be abbreviated as long as the command can be distinguished from all other command names.

The standard editing characters are available while typing commands.

The collection of parameters is done in a standardized way as follows:
- Parameters are separated either by a comma or by any number of spaces, or by a combination of comma and spaces.
- Parameters may be null, in which case a default value is assigned.
- When a parameter is missing (as opposed to null) it is asked for, and the command processor expects the operator to supply the required parameter plus more parameters if he so wishes.
- When a parameter syntax error is detected, an error message is printed and the parameter is asked for.
- Excess parameters are ignored.

HELP (command name)

The HELP command lists available commands on the terminal. Only those commands that have (command name) as a subset are listed. If (command name) is null then all available commands are listed.

EXIT

The EXIT command returns control to the SINTRAN-III command processor.

LINES (lines per page)

The LINES command enables the operator to specify the number of lines per page on the listing device.

LIST (list directive)

The LIST command is used to select various listing options. If a list directive has not been given, then only the source program will be listed. A LIST command with an empty parameter will cause all available list directives to be listed on the terminal.

The following are legal list directives:

- SYMBOLS enables the listing of the symbol table;
- GENERATED–CODE enables the listing of the intermediate code in symbolic form;
- DEFAULT sets the listing mode to the default list mode. In other words, listing of symbols and generated code is disabled.

COMPILE (main program) (functions) (listfile) (objectfile)

The COMPILE command is used to compile a NODAL program from the main program file along with a set of defined functions from the function file. A listing will be generated on the listing file. If a listing file is not supplied, then no listing will be generated, but error messages will be printed on the terminal. Object output will be generated on the object file, if supplied. The default file type for the two input files is :NOD. The default file type for the listing file is :LIST. The default file type for the object file depends upon the individual code generator connected to the compiler.

DATAMOD <name> <main program> <propertyfile> <listfile> <objectfile>

The DATAMOD command is used to compile a NODAL Data Module with name <name> from the main program file along with its property table from the property file.

A listing will be generated on the listing file. If a listing file is not supplied, then no listing will be generated but error messages will be printed on the terminal. Object output will be generated on the object file if supplied. It is possible to continue the compilation of more Data Modules to the same object file, by specifying only <name> <main program> and <property file>. When <name> is not specified, the end of the Data Module list is assumed. The default file types for the input files are :NOD and :SYMB respectively. The default file types for the output files are similar to those used for the COMPILE command.

The property file may be left out if it is desired to compile only a main program.

SYTBL <inputfile> <listfile>

The SYTBL command is used to process a symbol table file, as will be described in the *Symbol Table File* section.

SYSVR <inputfile> <listfile> <objectfile>

The SYSVR command is used to process a system variable file, as will be described in the *System Variable File* section.

A list of input files can be written onto the same object and list files. Striking the cr (carriage return) key when the input file is asked for will terminate the list.

SYSND <inputfile> <listfile> <variablefile>

The SYSND command is used to process a NODAL variable file, as is described in the *system variablefile* section. The type of object file is: NOD.

3.2 Compiler list file format

The compiler listing always contains the NODAL program lines. If the GENERATED-CODE option has been specified, then the intermediate, or virtual, code is listed in symbolic form after each program line. At the end of each NODAL program or defined function, a symbol table is output if the SYMBOLS option has been specified.

PAGE HEADING

The first three lines of a page constitute the page heading. Before the heading lines are printed, the listing device is advanced to a new page. If the listing device is the terminal, a blank line is printed instead of advancing to the next page. The heading consists of the following fields:

- Compiler name and version number.
- Current date and time.
- Page number.
- The name of the module currently being compiled. This is MAIN for the main program and the function name along with formal parameters for a defined function.
- One blank line.

VIRTUAL CODE LISTING

The listing of the intermediate code, if enabled, consists of several fields for each line:

- The octal address of the intermediate code. This address is relevant only to the implementer of a code generator.
- The label field for a LABEL instruction. An 'L' character is output in front of the label index value.
- The name of the virtual instruction (see Ref. 7).
- The operands of the virtual instruction in symbolic form.

ERROR MESSAGES

If an error is detected in a line, the error message is output following the line in error. The error message is preceded by five asterisks (*****). If no listing device has been specified, then the error message also includes the line number and the name of the defined function or MAIN for the main program in which the error occurred.

INTERNAL SYMBOL TABLE

If the SYMBOLS option has been supplied, then a symbol table will be listed after each main program or defined function. This symbol table includes information about each parameter, local variable, and local array used in the program.

3.3 Symbol table file

The NODAL compiler provides a facility for adding function and constant definitions to the internal symbol table SYTBL.

The SYTBL file may contain statements as described below. Statements are terminated either by end of line or semicolon.

% comment

The remainder of the line is taken to be a comment.

CONSTANT name=value, ...

This statement defines the specified name to be synonymous with the specified value. The value may be any constant expression. Whenever the compiler encounters this name, the associated constant value will be substituted for it.

DEF-MATH fname(nops)=rname, ...

This statement defines the name 'fname' to be a mathematical function. The number of operands 'nops' must be one or two. The runtime subroutine which implements the function is specified by 'rname'.

DEF-RW fname(parameter list)=rname, ...
DEF-WO fname(parameter list)=rname, ...
DEF-RO fname(parameter list)=rname, ...
DEF-CALL fname(parameter list)=rname, ...

These statements are used to define read/write (type 8), write only (type 9), read only (type 10), or call (type 11) assembly language functions. The parameter list may contain from zero to eight of the following parameter type names:

RVAL real value corresponds to 6VAL
IVAL integer value corresponds to 6VAL
SVAL string value not available in compiler
NREF NODAL reference not available in compiler
RREF real reference corresponds to 6VRF
IREF integer reference corresponds to 6VRF
RARR real array corresponds to 6ARF
IARR integer array corresponds to 6ARF
NAME NODAL name corresponds to 6NAM

The runtime subroutine which implements the function is specified by 'rname'.

END

This statement specifies the end of the symbol table file.

3.4 System variable file

The NODAL compiler provides a facility whereby the user may predefine and preset variables and arrays which are to be used by the compiled main program and defined functions. These 'system variables' are written, by the code generator, to a specified object file as a standard NODAL list. This means that other programs may easily share these system variables by simply searching the NODAL list. The SYSVR command writes the variables onto an object file in the object code specified by the code generator for a particular target machine. The SYSND command writes the variables in the standard NODAL format independent of the type of target machine, such that they can be loaded into the NODAL buffer with the aid of a LOAD or OLD command.

The system variable file may contain statements as described below. Statements are terminated either by end of line or semicolon.

% comment

The remainder of the line is taken to be a comment.

CONSTANT name=value, ...

This statement defines the specified name to be synonymous with the specified value. The value may be any constant expression. Whenever the compiler encounters this name, the associated constant value will be substituted for it.

REAL name, ...

The specified name is defined to be a system variable. Its value is initialized to zero.

DIM name (number of elements), ...

The specified name is defined to be a floating-point one-dimensional system array. Each element of the array is initialized to zero.

DIM name(index1,index2), ...

The specified name is defined to be a floating-point two-dimensional system array. Each element of the array is initialized to zero.

DIM-I name(number of elements), ...

The specified name is defined to be an integer one-dimensional system array. Each element of the array is initialized to zero.

DIM-I name (index1,index2), ...

The specified name is defined to be an integer two-dimensional system array. Each element of the array is initialized to zero.

SET name=value, ...

The system variable is initialized to the specified value.

SET name (index)=value, ...

The specified one-dimensional system array element is set to the specified value.

SET name (index1,index2)=value, ...

The specified two-dimensional system array element is set to the specified value.

SET name=(value1, ..., value n), ...

The specified array is initialized, starting from the first element, to the set of specified values. For two-dimensional arrays the first index increases most rapidly.

END

This statement specifies the end of the system variable file.

If the symbol table option is enabled while the system variable file is being processed, then a symbol table of the defined constants, variables, and arrays will be output to the listing file.

4. THE DATA MODULE CONSTRUCTION ELEMENTS

4.1 Property table

The purpose of the property table is to drive the branching to the appropriate NODAL line according to the property which has been invoked in the Data Module call. The property table is specified as a two-dimensional integer NODAL array declared through the DIMENSION-INTEGER NODAL statement.

For single property, the property table is a 4 × N array, N being the number of properties:

DIM-I PTY(4,N)

NAME 1 (RAD 36)	WRITE ENTRY PROP 1	READ ENTRY PROP 1	PROPERTY 1 STATUS
NAME 2 (RAD 36)	WRITE ENTRY PROP 2	READ ENTRY PROP 2	PROPERTY 2 STATUS
NAME N RAD 36	WRITE ENTRY PROP N	READ ENTRY PROP N	PROPERTY N STATUS

For multiproperty[3], the property table is a 5 × N array:

DIM-I PTY(5,N)

FIRST NAME 1	LAST NAME 1	WRITE ENTRY PROP 1	READ ENTRY PROP 1	PROPERTY 1 STATUS
FIRST NAME 2	LAST NAME 2	WRITE ENTRY PROP 2	READ ENTRY PROP 2	PROPERTY 2 STATUS
FIRST NAME N	LAST NAME N	WRITE ENTRY PROP N	READ ENTRY PROP N	PROPERTY N STATUS

The contents of the property table are set up through the SET command as shown below:

– *Single property*

```
DIM-I   PTY(4,2)

SET   PTY = (#CON, @10.10, @10.20,  0,
             #REF, @20.10, @20.20,  0)
```

defines the single properties CON and REF with the Write entry points, the NODAL lines 10.10 and 20.10 respectively, and with the Read entry points 10.20 and 20.20.

– *Multiproperty*

```
DIM-I   PTY(5,2)
SET     PTY = (#CON, #CON,  @10.10,  @10.20,  0,
               #RF1, #RF9,  @20.10,  @20.20,  0)
```

The multiproperty RF1, RF2, ..., RF9 replaces the single-property REF while keeping the same entry points.

The property branch is performed in the entry section of the Data-Module body by calling the function PROBR. When the routine PROBR is called in the Data Module, the first entry is branched in case of a Write access and the second entry in the case of a Read access. When the entry is zero, the error message: 'NO SUCH PROPERTY' is returned when one tries to access it.

The last column of the property table is a status word. Its layout is shown below:

15 ... 9	8	7 6 5 4 3	2 1 0
S	A	D	P

P = protection code
D = displacement
A = array handling bit
S = property specific section.

The meaning of each separate item is explained below:

– D is the number which will be returned by the function DISP in the case of single property. In the multiproperty case, the function DISP returns the difference between the first entry name and the wanted property plus the number D.

For example,

```
#CON, #CON, entry W, entry R, [20
#AD0, #AD9,  entry W, entry R, [30
```

(the sign [means: octal number).

When the property #CON is called, the function DISP will return the value two.

When the property AD0 is required, DISP will return the value three.

When the property AD6 is wanted, DISP will return the value AD6−AD0+3=9.

–A is bit 8; it should be set when calls to the Data Module with this property allow array transfer. When bit 8 (A) is zero, only

```
     SET   DMS(EQ, #PTY) = X
or   SET   X = DMS(EQ, #PTY)
```

is allowed. However, when bit 8 is set to one, this specific property coding allows the transfer of arrays through

```
      DMS(ARR, 'W', EQ, #PTY)
and   DMS(ARR, 'R', EQ, #PTY)
```

– The protection code P and the meaning of property specific section S will be discussed in Section 3.3.

The global section number and the capability word are set by the command PROT.

PROT(N,k)

means that the global section is N, global capability is k. The default value for both is zero.

For the NORD computers the data table width is specified in the property table file with the symbol DTW. For example:

DTW. = 9

will set the data table width to 9.

The documentation of the Data Module should be written into the property table, such that the user documentation can be extracted with the aid of the SCAN program.

The user documentation consists of two parts:

- The system information, where the name of the author and all consecutive modifications of the Data Module are described. This information is preceded by three percent signs: %%%.
- The functional description of the Data Module, which describes the installation locations, the Data Module names and the generic name, plus the number of entries into each data table. The information is preceded by two percent signs: %%.

4.2 The Data Module body

The Data Module code is a normal NODAL program which has two variables implicitly defined: the equipment number and the property. These variables are set at the calling stage. Their value can be obtained in the Data Module program through the functions EQNO and PTY.

All arguments to and from the Data Module are passed through the commands PUT and GET.
PUT writes an array or variable to the calling program.

Example 1

Calling program:

```
> DIM-I    A(10)
> CALL    DMSCAL(A,'R', EQ, #PTY)
```

Data Module:

```
DIM-I    CC(10)
10.20    PUT    CC;    END
```

writes the contents of the integer array CC into the array A of the calling program.

Example 2

Calling program:

```
>TYPE    DMSCAL(EQ, #PTY)
```

Data Module:

```
10.20    SET    C = 10;    PUT    C;    END
```

will result in the typing of the number 10 by the calling program.

GET reads an array or variable from the calling program into the Data Module.

Example 1
Calling program:

```
> SET     DMSCAL(EQ,#PTY) = 10
```

Data Module:

```
10.20     GET     A;   END
```

will store the value 10 into the variable A.

Example 2
Calling program:

```
> DIM-I     A(22)
> CALL      DMSCAL(A,'W',EQ,#PTY)
```

Data Module:

```
DIM-I     BB(22)
```

This array can be defined in the SYSVR file

```
10.20     GET     BB;   END
```

will store the 22 values of the array A of the calling program into the array BB of the Data Module.

All errors which occur during the execution of the Data Module are returned as normal NODAL error messages to the calling program.

Typical errors are:
- divisions by zero,
- array dimension error,
- working area full,
- argument list error,
- resources exhausted,

the last error being returned when a wrong call to the PUT and GET commands is made.

User-defined errors can be returned with the aid of the function FLAG. For example,

```
SET     FLAG = 38;   END
```

will result in the error message: NOT IMPLEMENTED

As mentioned previously, the branching to the correct property paragraph will be done through the routine PROBR. When this routine is executed it will look through the property table to see if the property specified in the Data Module call exists. In that case it will resume execution at the line specified in the property table.

The routine PROBR supports 7 default properties: #STA, #CON, #PWS, #SBT, #LCK, #CAP, and # SEC. Their meaning is explained in Section 4.4.

4.3 Protection scheme

The same protection scheme as the one already in use for the assembly language Data Modules is used for the Data Modules written in NODAL. The ingredients are:

- the protection code in the property table;
- data table password contained in the data table;
- the global section and capability stored in the property table;
- the routine PROBR, which performs all the checks required.

The working of the protection coding is as follows:

PROBR looks at the word DAT.(1, EQNO), which should contain the data table status word (see Section 4.4). Checks are then made on the bits PWS and SBT:

- when PWS is zero, no protection is wanted;
- when PWS is one and SBT is one, the section of the calling program should be equal to the global section defined in the property table;
- when PWS is one and SBT is zero, the protection is dependent on the protection code P defined in the property table status word (see Section 4.1).
- P = 0: no check is done;
- P = 1: the capability of the calling program should be equal to the global capability;
- P = 2: the section of the calling program should be equal to the global section;
- P = 3: not used;
- P = 4: the capability bit which is defined by the data table password should be one;
- P = 5: the section of the calling program should be equal to the section defined in the data table password;
- P = 6: 1 or 5;
- P = 7: the section of the calling program should be equal to the section defined in the property status word.

4.4 Data table

The data table is a two-dimensional array, which is not directly accessible from the Data Module residing in the NORD-10 or NORD-100. However, when the Data Module resides in a microprocessor, the data table should be explicitly defined in the SYSVR file.

The layout is as shown below and is defined by

DIM-I DAT.(N,M)

resulting in a data table of N entries for M pieces of equipment.

For the NORD, the number of entries N is specified in the property table file (see Section 4.1).

ENTRY	1	2	//	N
EQ, 1	DATA TABLE STATUS	DATA TABLE PASSWORD	//	
EQ, 2			//	
⋮			//	
EQ, M			//	

Of the N entries, the first two are used for back-up and protection coding. Their layouts are:

- word 1 contains
  - bit 15 : CON software connect,
  - bit 14 : SBT out-of-service switch,
  - bit 13 : PWS password switch,
  - bit 12, 11: LCK lock-up status;

the rest of word 1 will be used for back-up purposes;

- word 2 contains the data password, and is only used when unit protection is wanted.

15 9					4 0
SECTION	0	0	0	0	CAPABILITY BIT No.

Bits 9-15 contain the unit specific section number;
Bits 0-4 contain the unit specific capability bit.

The contents of the data table status and password are manipulated by the standard properties:
#STA R will return the data table status word
#CON R/W will Read/Write the CON bit in the data table status word
#PWS R/W will Read/Write the PWS bit in the data table status word
#SBT R/W will Read/Write the SBT bit in the data table status word
#LCK R/W will Read/Write the lock-up status in the data table status word
#CAP R/W will Read/Write the capability bit number in the data table password
#SEC R/W will Read/Write the section in the data table password.

For example, the statement

```
SET     DMS(2,#PWS) = 1
```

will set bit 13 in Entry 1 of EQ,2 to one.

The protection of the standard properties will be such that when PWS = 1 and SBT = 0 and no unit protection is used:

#CON is executed only when the global capability defined in the property file is the same as the global capability of the calling program.

The other default properties will be executed only when the global section is equal to the global section of the calling program.

However, when PWS = 1 and SBT = 0 and unit protection is used for any one of the non-standard properties, then

#CON is executed only when the section of the calling program is equal to the section defined in the data table password.

The other default properties will be executed only when the capability bit defined in the data table password is set in the capability word of the calling program.

5. MODE OF OPERATION

5.1 Installation in the NORD computer

The compiled Data Module consists of two parts.

i) The Data-Module body, which contains the Data-Module code together with the property table. The body is created from two separate files by the NODAL compiler, as explained in Section 3. The body is loaded into the NORD computer as a NODAL element by the function LDBRF. This NODAL element has the type number 28: 'BRF CODE'.

ii) The data tables are created with the required number of entries/equipment using the functions CREDT or CREDTS. CREDTS allows the creation of a generic name and the equipment names, while CREDT allows the creation of only one Data-Module header per Data-Module body.

The Data Modules thus installed support the default properties (#STA, #CON, #PWS, #SBT, #LCK, #CAP, and #SEC, described in Sections 4.3 and 4.4). The data table width (i.e. the number of data table entries/equipment) is defined in the property file through the command DTW.

It should be noted that when the Data-Module programmer wants to specify the code for the standard properties himself, the protection scheme used in the PROBR function will no longer work. In that case the Data-Module programmer should do the branching to the desired property himself with the aid of the function PTY.

Since the dynamic creation of data tables leaves the data tables empty (i.e. data tables are filled with zeros), the data table contents should be initialized. For this purpose the function DTWRT and DZWRT are available in the NORD.

The arrays needed for DZWRT or DTWRT can be created with the aid of the SYSND facility available under the NODAL compiler. The users are strongly advised to prepare NODAL programs to set up their data tables and initialize the Data Modules.

When the Data Module includes a driver part, which is activated by an interrupt, variables should be accessible from both the compiled Data Module and the driver code, which is usually written in MAC or NPL. For this purpose the SYSVR command available from the NODAL compiler can be used. The command processes a QED file, in which the shared arrays are specified. The created object file contains BRF code which can be loaded into the computer with the aid of the LDBRF function. By first loading the system variables into the computer, these are automatically linked to all Data Modules which reference these system variables.

5.2 Interactive testing in the NORD computer

- Create the property file under QED as described in Section 4.1. Do not use the PROT and DTW. commands.
- Enter the NODAL compiler and use the SYSND command to transform the property file into a NODAL array.
- Enter (DMDEV)DM–NODAL; use the LOAD command to load the NODAL file which contains the property table.
- Create a data table by typing

```
DIM-I    DATA(DTW,N)
```

N = number of units

DTW = data table width.

Note: the name DATA is compulsory.

- Now load or write the Data Module NODAL code.
- The NODAL program is activated with the function CALDMS by typing

```
> SET    CALDMS(N#PTY) = X
> SET    X = CALDMS(N,#PTY)
```

Several things should be kept in mind when using this facility:

i) The ERASE command should not be used.

ii) The first time, CALDMS should be called with equipment No. 1.

iii) The support functions LISDM, DTWRT, DTREA, DTSIZE, DXREA, DXWRT, DZREA, DZWRT will not work.

iv) The function LISPTY will list the property file to ODEV. The format is the same as that needed for the compiler.

v) LINEC allows the modification of a line entry in the property table.

Example: To set the Write entry for the third entry in the property table to 11.22, the following command must be typed:

```
>SET    PTY(2,3) = LINEC(11.22)
```

For multiple properties this is:

```
> SET    PTY(3,3) = LINEC(11.22)
```

5.3 Installation in ACC

5.3.1 Functional description of an ACC-based Data Module

The Data Module in an ACC consists of the Data-Module body created by the NODAL compiler, and a data table located in the system-variable area (see Section 3.4). The name of the NODAL array serving the data table purpose must be declared as a parameter in the call to PROBR.

Several facilities have been built around a Data Module in an ACC. It is possible to connect NODAL code to the four external interrupts as described in Section 5.3.3. An ACC can control an MPX-UC, as described in Section 5.3.4.

To survey the correct functioning of the ACC, a link between the Data Module and the NORD host is made through the functions STATUS in the ACC and the function ACFASP in the NORD computer.

The 16-bit word written by the STATUS function is read by the ACFASP function. The most significant bit of this word is the status bit for the FASP surveillance.

When RUN ERRORS occur in the ACC, an RT program is activated in the NORD, which reads the contents of the RUN ERROR buffer in the ACC and stores them into the RUN ERROR buffer of the NORD.

The Data Module together with its system variables are linked to the ACC run-time software with the aid of a MODE file running on a NORD SINTRAN–III computer. The object file is later loaded into the ACC with the function TMLOAD. The latter function permits the linking of several files, such that one ACC object file with a hardware definition file can serve different ACCs with a different hardware environment.

EPROMs are created by copying the contents of the ACC memory. Before copying, the ACC should have been reset and started once, such that the interrupt pointers in the interrupt table can be updated to decrease the interrupt switching time.

5.3.2 Software packages for Data Module support in ACC

The ACC has a 32K byte memory for program and data storage (hexadecimal address 0–7FFF). Below we give a table of all separate packages which are available for the Data-Module support, followed by an explanation of the purpose of each package:

- Interrupt table
- Communication package
- Hardware access package
- Compiled code executor
- Mathematical functions
- Floating-point arithmetic package
- Functions
- Compiled code containing interrupt routines, Data Module, and Background
- SYSVR variables
- Data Module stack from SWORK to EWORK
- Interrupt + Background stacks
- Error and communication buffers
- Stack space for service modules

- *Interrupt-table layout*

The table is analogous to the one defined in the TMS-9900 Manual. In the ACC, four external interrupts are available, and three internal interrupts for error recovery. The latter are only used by the system software, while the action to be performed after one of the four external ones can be defined in a NODAL program.

- *Communication package*

The communication package can be seen as an extension of the SYNTRON package residing in the NORD host computer. It passes the Data Module calls to the wanted Data Module, performs LISDM, RUN ERROR handling, DATA-TABLE HANDLING, and SYSVR variable retrieval.

- *Hardware access package*

The hardware access package assures uniform access to the hardware controlled by the ACC. Modules are available to access an MPX-UC, a CIMBUS module, or a CAMAC module. The software performs error checking and signals illegal access through the RUN ERROR mechanism.

- *Compiled code executor*

The package executes the threaded code generated by the NODAL compiler. The compiled code consists of references to the modules residing in this package.

- *Mathematical functions*

The mathematical functions — SQR, AT2, SIN, COS, EXP, LOG — are contained in this package.

- *Floating-point arithmetic package*

The NODAL compiled code uses a three-word floating-point format which is exactly identical to the one residing in the NORD machines. Thus the results produced by the NODAL compiled code are exactly identical to the ones produced by the NODAL interpreter running on the NORD machines.

- *Functions package*

This package contains all the functions required for the Data Module execution, and several functions also frequently used in the NODAL interpreter, e.g. BIT, IOR, etc.

- *Compiled code*

Here the user-defined NODAL compiled programs are loaded. They may contain interrupt and background routines, as well as several Data Modules together with their property table.

– *SYSVR variables*

In this package all user-defined variables created by the SYSVR command are located. These variables permit the sharing of data between the interrupt routines and the Data Modules. Also all variables which should not be reset to zero before the execution of the Data Module (data tables) should be located here.

N.B. Up to (but not including) the SYSVR variables, all code can be loaded into EPROM. The rest of the memory should be RAM.

– *Data Module stack*

This is the working area for the Data Modules. All Data Module code is re-entrant, and recursive programming is possible. For this purpose the Data Module stack contains the data local to the Data Module packages mentioned above.

– *Interrupt and background stacks*

These stacks serve the same purpose for the interrupt and background programs as the Data Module stack. They permit the concurrent execution of the four interrupt routines, the background routine, and the Data Module.

– *Error and communication buffers*

These buffers contain the errors created by the RUN ERROR mechanism, and the information which is passed between the microprocessor and the host computer.

– *Stack space for service routines*

This stack space is used by the internal hardware routines, the communication routines, and the interrupt routines. They permit the mutual exclusion of shared resources through the BLWP and LIMI instructions.

5.3.3 Interrupt level programming

The additional facilities of interrupt coding and background processors exist on the microprocessor-based software. The interrupt coding is activated by the four front-panel interrupts. The code which should be executed immediately after an interrupt should be preceded by the command

```
WAIT–I    N
```

with $1 \leq N \leq 4$.

The NODAL lines following these commands are executed every time interrupt N is passed to the microprocessor. The execution is stopped when an END or other WAIT–I command is met.

The background program is preceded by the command BACKG.

All lines following this command are permanently executed, but can be interrupted by the Data Module or interrupt handler. When an END or WAIT–I command is met, the execution is stopped; it will be resumed within 16 ms.

All communication between the interrupt-driven code, the Data Module code, or the background code, passes between the variables defined in the SYSVR file:

Example

```
SYSVR   file

    DIM–I    A(4)
    REAL     BC
```

NODAL program

```
1.1  WAIT-I 1;   SET  A(1) = A(1)+1
1.2  WAIT-I 2;   SET  A(2) = A(2)+1
1.3  WAIT-I 3;   SET  A(3) = A(3)+1
1.4  WAIT-I 4;   SET  A(4) = A(4)+1
1.5  BACKG;      SET  BC = BC+1;  END
```

In the above example, four counters are incremented every time one of the four front-panel interrupts has arrived. In the meantime the counter BC is updated every 16 ms by the background processor. It is completely feasible to return the array A(4) or the counter BC to the NORD computers with the aid of the Data Module.

To create a link with the FASP program running in the host NORD computer, the function STATUS has been made. When bit 15 of STATUS is set to one, FASP will activate a scheduled program to further investigate the error.

Example

```
1.5          BACKG
1.55   IF    CAMAC(C,N,A,F) = 1;   SET   STATUS = -1
1.60   IF    CAMAC(C,N,A,F) = 0;   SET   STATUS = 0
```

Shown above is the very simple case where the background processor looks at a module N. Depending on the status of this module, STATUS is set to zero or minus one. In this latter case, FASP will initiate the error procedure that is asked for.

The priorities of the programs are shown below in descending order:

Interrupt 1
Interrupt 2
Interrupt 3
Interrupt 4
Data Module
Background

The interrupt layout in the ACC should be as shown below (see drawing CERN-SPS 7-2420-01-007-3, ACC repérage des points):

IT0	-	INTR25	LEVEL 15
IT1	-	4	14
IT2	-	3	13
IT3	-	2	12
IT4	-	1	11
IT5	-	QXERI	10
IT6	-	INTTY	9
IT7	-	5 (TO CAMAC)	8

5.3.4 Hardware access from the ACC

Three modes of hardware access are possible from a Data Module: CAMAC, MPX, and CIMBUS serial.

When the Data Module resides in an ACC, access is only possible to modules linked to the same CAMAC crate as the ACC.

5.3.4.1 CAMAC access

Access to these modules is done through the function CAMAC or SCAM.

In the ACC the additional commands CAMAC and CADRIV are available (cf. Appendix III–2).

5.3.4.2 MPX access

The MPX modules in the MPX stations that are linked through an MPX control unit accessible from the Data Module can be accessed with the functions:

MPXI, CMPX, GPMPX, MPX and the command MPDRIV:

(cf. Appendix III–2).

- MPXI, to initialize the MPX link;
- CMPX, GPMPX, MPX, and MPDRIV to access an MPX module.

The coding, being re-entrant, may be accessed from different program levels. However, the analog acquisition should only be accessed from one of the interrupt, Data Module, or background levels. (If interference happens, the analog acquisition continues at the Data Module level and an error message is returned at the driver or background level.)

There are two different cases of MPX link control, defined by DEFACC and controlled by the function GIVACC depending on the presence of real–time access from the ACC to MPX.

1) No real–time access to the MPX link

   In this case the MPX link can be accessed from the Data Module resident in the ACC and also directly from the NORD. So any ACC Data Module request from the NORD must be preceded by a reservation of the MPX link in the NORD.

2) Real–time access to the MPX link

   In this case the MPX link is controlled by the ACC, any access to this MPX link should be done from this ACC. The MPX multifunction access is forbidden.

   An ACC can control only one MPX link. When this particular MPX link is accessed from the NORD host computer, the latter passes a request to the ACC to access the MPX link. In this case there is no longer a reservation of the MPX link in the host computer but a reservation of the ACC.

   The time-out, alarm, and EOAA (End of Analog Acquisition) interrupts are no longer handled by the NORD coding but by the ACC.

   The connection from the MPX UC to the ACC front panel should be done as follows:

TIME-OUT	→	interrupt 1
EOAA	→	interrupt 2
Alarm	→	interrupt 3 (not used for the moment).

Consequently, only NODAL coding attached to interrupt 4 will be activated.

To disconnect the MPX link from the ACC, the function TAKACC must be used. Then the NORD will take over the control of this MPX link.

The function ACCSTA tells the user which MPX links are allocated to which ACCs.

The operating systems in the ACCs controlling different MPX links differ from each other by the definitions of N9MPX, ADCD, and ALDCD.

a) N9MPX defines the position of the MPX control unit in the CAMAC crate:

15	14 13 … 9	8 … 0
0	0 N	0

Example

N9MPX EQU > 2400 for the 4th UC in the crate.

b) ADCD is the address of the ADC flag:

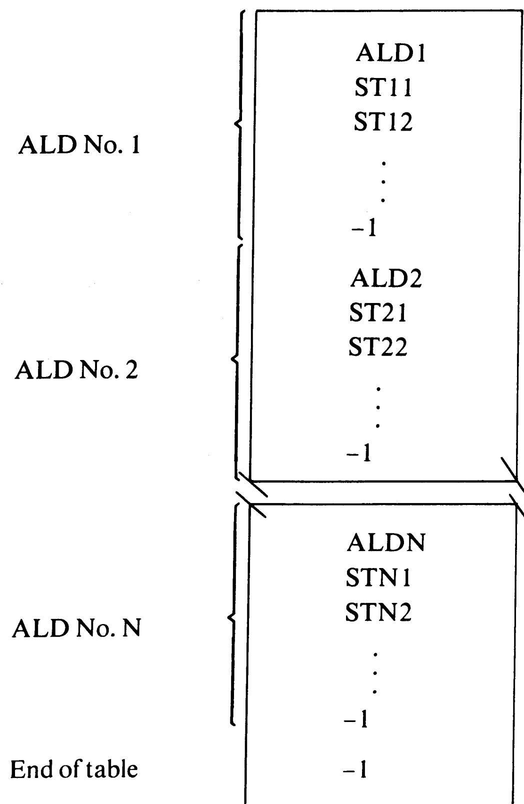
ADC flag = 0 when no ADC is present,
1 when an ADC is present.

Example

ADCD DATA 0 means: no ADC present.

c) ALDCD is the address of a pointer to a table (BMLD) containing the necessary information for access to ALD MPX modules.

When this pointer is zero, it means that no ALD modules are present. The table has the following structure:



STNM = M^{th} station concerned with ALD No. N.
ALDN has the structure

15 — 12	11 — 7	6 — 3	2 — 0
0	ALD No. N Station number	ALD No. N Peripheral number	0

Example

```
ALDCD   DATA   BMLD
BMLD    DATA   ALD1,  ST11  ST12,  -1
        DATA   -1
ALD1    EQU    > 470                     ST = 8;  PN = 14
ST11    EQU      8
ST12    EQU      7
```

5.3.4.3 CIMBUS serial access

The functions CBSI and GPCBS are made to access CIMBUS modules situated in a CIMBUS serial crate. The link between the CAMAC crate containing the ACC and the CIMBUS serial crate should be made through the two CAMAC modules '3151 (serial link)' and the CIMBUS module 'CAMAC serial I/O interface'.

The purposes of the functions are:

CBSI – initialize the CIMBUS serial link
GPCBS – access CIMBUS module in CIMBUS serial crate

(cf. Appendix III–2).

The code being re-entrant, it can be called from all levels.

These functions do not exist in the NORD SYNTRON system; thus no routines for the reservation of the serial link have been implemented.

5.4 Interactive testing of Data Modules in ACC

- Create the property table under QED as described in Section 4.1. Do not use the PROT and DTW. commands.
- Enter in NODAL compiler and use the SYSVR command to transform the property file to a Texas object file.
- Link the property file to the NODAL interpreter with the mode file LOAD–NODAL as shown in Appendix II. The object file is written to the file DUMP:TEX. This latter file should be loaded into the ACC with the function TMLOAD.
- When the ACC is activated with the CAMAC function A0 F28 (reset) and A0 F25 (start), the NODAL interpreter is ready. Strike the cr (carriage return) key on the TTY connected to the ACC; then the message NODAL, etc., will appear.
- The data table is then created by the command DIM–I DAT.(DTW,N):
  N = number of units
  DTW = data table width.

Note: the data table name should be the same as the name specified as a parameter in the PROBR call.
- Load or write the Data Module code.
- The NODAL program is activated as a Data Module with the aid of the function CALDMS:

```
SET   CALDMS(N,#PTY) = X
SET   CALDMS(N,#PTY)
```

In short, the Data Module is written in an interactive way with the aid of the NODAL interpreter. The layout of the property table should be known beforehand. With the SYSVR command, the table is compiled for a TMS-9900 object file. With the aid of the mode file LOAD–NODAL (available from the SPS/ACC group section SOFT) the property file together with the wanted NODAL interpreter modules is written onto the file DUMP:TEX. The latter is copied to a SYNTRON-compatible floppy disk for late loading into the ACC.

Example

```
@NODCOMP-9900
TMS-9900 NODAL COMPILER
*SYSVR PROPER, L-P, PROPER
SYSVR FILE: ↓
NO ERRORS DETECTED
*EXIT

@MODE LOAD-NODAL L-P

@NODAL
NODAL
> RESFLO(O)
> FLCOPY('DUMP:TEX','<510> OBJEC.')
>QUIT
```

References

1) M.C. Crowley-Milling, The data-module, the missing link in high-level control languages, presented at the 3rd Int. Conf. on Trends in on-Line Computer Systems, Sheffield, 1979.

2) J. Altaber and F. Beck, The distributed data-base for the CERN SPS control system, presented at the European Workshop on Industrial Computing Systems (PURDUE, Europe), Vienna, 1980.

3) G.J. Jennings, Data module Systems Handbook, CERN SPS/ACC/GJ/Comp. Note 79-34 (1979).

4) G.J. Jennings, SPS NODAL Functions and Data Modules, CERN SPS/ACC/GJ/79-12.

5) P.D.V. van der Stok, Transparent access to microprocessors, Presented at the 1980 NOCUS Meeting, Helsingør, 1980.

6) M.C. Crowley-Milling and G. Shering, The NODAL system for the SPS, CERN 78-07 (1978).

7) NITTEDATA, NODAL cross-compiler. Reference manual.

8) NITTEDATA, NODAL code generator for TMS-9900.

9) NITTEDATA, TMS-9900 NODAL interpreter. Reference manual.

APPENDIX I

SUPPORT FUNCTIONS FOR DATA MODULES IN THE NORD

This section describes the functions available for installing Data Modules in the NORD computer.

FUNCTION NAME:	LDBRF – load BRF object file
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	LDBRF (AREA,N,"FILE")

Purpose

This function loads the BRF file FILE into the SYNTRON area AREA with number N.
AREA can be:

- RES – for loading in the resident list,
- PROG – for loading in the program area,
- DATA – for loading in the DATA area.

N is the area number; it should be zero for loading in the resident list. Under the SINTRAN run-time system, only loading into RES is authorized.

FUNCTION NAME:	CREDT – create data table with Data Module header
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	CREDT(N,NAME,DMS,NU)

Purpose

This function creates a data table for Data Module body NAME with the Data Module name DMS. The data table will be located in the SYNTRON run-time system area: DATA with number N.

When N = 0 the data table will be located in the same area as the Data Module body. N should be zero when used under the SINTRAN run-time system.

The number of equipment numbers available to DMS is defined by NU.

When NU = −1, this is an equipment function.

FUNCTION NAME:	CREDTS – create data tables with a generic name header and several Data Module name headers
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	CREDTS(N,NAME,DMSAR,UAR,FLAR)

Purpose

This function creates a set of data tables on data area N for the Data Module body NAME. When N is zero, the data table will be located in the same area as the Data Module body NAME. Under the SINTRAN-III run-time system, N should be equal to zero. DMSAR is a string array which contains the names of the Data Modules. The integer array UAR contains the number of equipment numbers available to this Data Module.

The integer array FLAR specifies the first equipment number. When the use of a generic name is needed, the first name of the string array DMSAR contains the generic name, and the first entry of FLAR should be zero. The dimensions of FLAR, DMSAR, and UAR should be equal.

For example, consider the creation of a generic name GNAM1 with four equipment numbers for the Data Module body. A further four Data Module names (DMS1, DMS2, DMS3, and DMS4) should be defined which access separate pieces of equipment. The arrays should then be of dimension five, and their contents is shown below:

ARRAY NAME:	DMSAR	UAR	FLAR
	GNAM1	24	0
	DMS1	1	4
	DMS2	5	8
	DMS3	13	10
	DMS4	23	2

With the aid of the generic name, one can access equipment numbers 1–24:

DMS1 can access equipments 1–4
DMS2 can access equipments 5–12
DMS3 can access equipments 13–22
DMS4 can access equipments 23 and 24.

SUPPORT FUNCTIONS FOR DATA MODULES IN AN ACC

This section describes the functions available in the NORD-10 for bebugging compiled Data Modules and regulating the control of the MPX control units.

FUNCTION NAME: DEFACC – define ACC data field
FUNCTION TYPE: 11 – call assembly language function
CALLING SEQUENCE: DEFACC(#L.N.,C,N,GL, option)

Purpose

This function creates a data field for an ACC in crate C, module N, with graded LAM GL. It allocates the logical number #L.N. to this particular ACC. The interrupt table DVTAB is patched, such that the LAM GL coming from the ACC is recognized by the SYNTRON run-time system.

Error messages are generated when double definitions of #L.N. or clashes with the GLs of other modules occur.

The function ACCSTA will list the results of the DEFACC command.

The option parameter is meaningful only in connection with the function GIVACC:

if option = RTUC, then there is real-time access to the MPX link from the ACC. Consequently, any access to the MPX link from the NORD will be preceded by a reservation of the ACC, and then executed in the ACC;

if option ≠ RTUC, then there is no real-time access to the MPX link from the ACC and any ACC Data Module request from the NORD will be preceded by a reservation of the MPX link.

FUNCTION NAME:	DFACDM – declare a Data Module resident in the ACC
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	DFACDM(#L.N.,DMS)

Purpose

This function creates a NODAL Data Module header with name DMS. The parameters are set up such that all calls to this Data Module plus the service calls LISDM, DTREA, DTSIZE, and DTWRT are transmitted to the ACC with logical number #L.N.

The latter should have been declared by the DEFACC function.

FUNCTION NAME:	GIVACC – MPX UC is accessed by ACC
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	GIVACC(UC, #L.N.)

Purpose

The function GIVACC signals to the SYNTRON system of the host computer that the ACC with logical number #L.N. accesses the MPX control unit UC. This function acts in two ways depending on the preceding DEFACC call.

Only after an assignment by this function will the functions ACNOD or TACNOD perform action on the MPX assignment.

FUNCTION NAME:	TAKACC – stops ACC
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	TAKACC(#L.N.)

Purpose

The function TAKACC stops the ACC CAMAC module and disconnects the MPX control unit (UC) from the ACC to which it might have been allocated by the function GIVACC. Consequently the MPX control unit (UC) is accessed directly from the host's SYNTRON. The functions TACNOD and ACNOD no longer have any meaning.

FUNCTION NAME:	ACNOD – start-up interactive NODAL in ACC
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	ACNOD(#L.N.)

Purpose

The function ACNOD starts the interactive NODAL in an ACC with logical number #L.N., which contains the NODAL interpreter. When an MPX control unit has been allocated to this ACC, the MPX control unit will no longer be known by the NORD-10 SYNTRON run-time system.

FUNCTION NAME:	TACNOD – release UC from the ACC interactive NODAL
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	TACNOD(#L.N.)

Purpose

The function TACNOD permits the access from the host NORD-10 to the UC controlled by the ACC with logical number#L.N.

FUNCTION NAME:	ACLOAD – load system variables from ACC
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	ACLOAD(#L.N., NAME)

Purpose

The function ACLOAD loads the NODAL system variable NAME into the NODAL buffer of the calling program from the ACC with logical number #L.N. When ALL is specified, all system variables resident in the ACC will be loaded into the NODAL buffer.

FUNCTION NAME:	ACFASP – read FASP status of ACC
FUNCTION TYPE:	8 – read only assembly language function
CALLING SEQUENCE:	SET X = ACFASP(#L.N.)

Purpose

The function ACFASP returns into X the 16-bit word last written by the function STATUS in the ACC with logical number #L.N.

FUNCTION NAME: ACINI – initialize ACC containing a Data Module
FUNCTION TYPE: 11 – call assembly language function
CALLING SEQUENCE: ACINI(#L.N.)

Purpose

The function ACINI tests the ACC, which contains a Data Module, on the proper functioning of the communication hardware.

When an error is detected, a RUN ERROR 45 is generated and a NODAL error is returned.

- ERR20: Argument list error ⇒ L.N. is not found
- ERR34: Hardware error ⇒ LAM cannot be cleared
- ERR32: Device not connected ⇒ LAM is not generated
- ERR62: Bad pattern ⇒ unexpected bit pattern in transfer register
- ERR63: Bad pattern assignment ⇒ memory test in ACC failed
- ERR6 : Bad pattern match ⇒ CAMAC function F17A0, F0A0, and F16A0 failed

The function execution time is about 0.8 seconds. After correct execution of the function, the ACC is ready to accept commands (MASK is set).

FUNCTION NAME:	TMLOAD – linker-loader of TMS-9900 object code into ACC
FUNCTION TYPE:	18 – free assembly language function
CALLING SEQUENCE:	TMLOAD(C,N,:L1,FILE1,:L2,FILE2...)

Purpose

The function TMLOAD loads and links several TMS-9900 object files into the ACC residing in the crate at position N. The possibility exists for specifying a load address, preceded by the sign : for each separate file. When no load addresses are specified, the files are loaded contiguously in core starting from address zero.

COMMANDS AND FUNCTIONS AVAILABLE IN THE DATA MODULE

III.1 Data Module support

The functions which support the Data Module structure are described in the following pages.

FUNCTION NAME:	PROBR – make property branch
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	PROBR(DAT.) for microprocessors
	PROBR for NORD–10,100
RESTRICTION:	None

Purpose

The PROBR function executes the branching to the appropriate NODAL line, depending on the property table and the property with which the Data Module is called. The layout and creation of the wanted property file is explained in Section 4.1.

The argument needed in the case of a microprocessor-based Data Module is the reference to the data table, which is defined in the SYSVR file (see Section 4.4).

PROBR performs all the protection checking, to verify if the access is allowed (see Section 4.3).

PROBR supports the standard properties and will execute the wanted coding (see Sections 4.4 and 4.3).

FUNCTION NAME: FLAG – specify call type,
– return error

FUNCTION TYPE: 8 – read/write assembly language function

CALLING SEQUENCE: SET FLAG = X
SET X = FLAG

RESTRICTION: None

Purpose

The FLAG function serves two purposes, depending on the type of call:

- As a Read function, FLAG returns the four integer values: -2, -1, 1, and 2, from which the type of Data-Module call is known:
  - -2: Read Array
  - -1: Read Value
  - +1: Write Value
  - +2: Write Array
- As a Write function FLAG returns an error to the program which called the Data Module.

Example

SET FLAG = 38

will result in the error message: NOT IMPLEMENTED

FUNCTION NAME:	ERROR – read/write error number
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET ERROR = X SET X = ERROR
RESTRICTION:	None

Purpose

After an access to the hardware, this function returns the error number 34 when a hardware error has been detected. In Write mode it is equivalent to the function FLAG.

FUNCTION NAME:	CAPABI – return capability
FUNCTION TYPE:	10 – read only assembly language function
CALLING SEQUENCE:	SET X = CAPABI
RESTRICTION:	None

Purpose

The CAPABI function returns the 16-bit capability word of the calling program.

FUNCTION NAME:	SECTN – return section number
FUNCTION TYPE:	10 – read only assembly language function
CALLING SEQUENCE:	SET X = SECTN
RESTRICTION:	None

Purpose

The SECTN function returns the section number of the calling program.

FUNCTION NAME:	DISP – return displacement
FUNCTION TYPE:	10 – read only assembly language function
CALLING SEQUENCE:	SET X = DISP
RESTRICTION:	None

Purpose

The DISP function returns the displacement defined in the property status word. In the case of a multiproperty, the difference with the lowest property is added. In this way it is possible to specify a displacement in the data table column, depending on the type of property (see Section 4.1).

FUNCTION NAME:	EQNO – return equipment number
FUNCTION TYPE:	10 – read only assembly language function
CALLING SEQUENCE:	SET X = EQNO
RESTRICTION:	None

Purpose

The EQNO function returns the equipment number specified in the Data Module call by the calling program.

FUNCTION NAME:	PTY – return property
FUNCTION TYPE:	10 – read only assembly language function
CALLING SEQUENCE:	SET X = PTY
RESTRICTION:	None

Purpose

The PTY function returns the property specified in the Data Module call by the calling program.

FUNCTION NAME:	DTI – read/write integer to data table
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET DTI (N) = X
	SET X = DTI (N)
RESTRICTION:	None

Purpose

The DTI function makes it possible to write to or read from a data table column specified by the equipment number in the Data Module call. This function is compulsory in the NORD-10, as the data table is hidden from the user. In the microprocessor-based Data Module its use is recommended for compatibility reasons.

The argument N should lie within $1 \leq N \leq N_{max}$, where N_{max} is the length of the data table column.

Example

The data table contains M pieces of equipment and N_{max} entries per equipment:

SET DTI(N) = X will be equivalent to
SET DAT.(N,EQNO) = X
$1 \leq N \leq N_{max}$

FUNCTION NAME: DTF – read/write three-word floating point to data table
FUNCTION TYPE: 8 – read/write assembly language function
CALLING SEQUENCE: SET DTF(N) = X
SET X = DTF(N)
RESTRICTION: None

Purpose

The DTF function makes it possible to write to or read from three consecutive data table column entries, specified by the equipment number in the Data Module call. This function is compulsory in the NORD-10, as the data table is hidden from the user. In the microprocessor-based Data Module, its use is recommended for compatibility reasons.

The argument N should lie within $1 \leq N \leq N_{max} - 2$, where N_{max} is the length of the data table column.

Example

The data table contains M pieces of equipment and N_{max} entries per equipment number:

SET DTF(N) = X

will put the floating contents of X into DAT.(N,EQNO), DAT.(N+1,EQNO) and DAT.(N+2,EQNO).

$1 \leq N \leq N_{max} - 2.$

COMMAND NAME	PUT – return value of variable or values of array to calling program
CALLING SEQUENCE:	PUT X; X = variable or array.
RESTRICTION:	None

Purpose

The PUT command stores the contents of the variable X (three words) or the array X (n words) in the variable or array specified by the Data Module call.

COMMAND NAME:	GET	– fetch values of array or value of variable from calling program
CALLING SEQUENCE:	GET X; X = variable or array	
RESTRICTION:	None	

Purpose

The GET command stores the contents of the variable (three words) or the array (n words) specified by the Data Module call into the variable X or the array X.

FUNCTION NAME: USAR – read/write user array
FUNCTION TYPE: 8 – read/write assembly language function
CALLING SEQUENCE: SET A = USAR(I)
SET USAR(I) = A
RESTRICTION: Only available in NORD computer

Purpose

This function transmits one integer value at a time to the array specified in the Data Module call. For example, the following piece of code in the Data Module

SET USAR(4) = 9: END

will put the value 9 into the fourth element of the array specified in the Data Module call

```
> DIM–I B(10)
> DMS(B,"R",EQ,#PTY)
> TYPE B(4)
```

will result in the typing of 9, which is the number stored into B(4) by the Data Module code shown above.

FUNCTION NAME:	STATUS – set or read FASP status
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET X = STATUS
	SET STATUS = X
RESTRICTIONS:	Only available in ACC

Purpose

The STATUS function is available only in the microprocessor-based Data Module. In principle its value should only be set by the surveillance program defined after the BACKG command.

The status function communicates the status of the hardware controlled by the ACC to the FASP program. The setting to one of bit 15 in the status word will result in the scheduling of an appropriate program by FASP.

The other 14 available bits can be used by the Data Module to communicate more information to the NORD-10 about the type of error. STATUS can be read from the NORD-10 by another function: ACFASP.

Example

```
SET   STATUS = -1      status error occurred
SET   STATUS =  0      all is O.K.
```

III.2 Hardware support

The functions and commands described in this section provide the means by which the Data Module can communicate with the physical world (CAMAC, CIMBUS, or MPX).

FUNCTION NAME:	CBSI – CIMBUS serial initialization
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	CBSI(N)
RESTRICTION:	Available only in ACC

Purpose

The CBSI function performs a general initialization of one serial link (modules CAMAC 3151 in positions N and N+1 in the CAMAC crate). Its action comprises:
- initialization and enabling of the receiver and transmitter;
- setting of the transmitter as master.

FUNCTION NAME:	GPCBS – executes a CIMBUS function in a CIMBUS serial module
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET GPCBS(N,MO,SA,FC) = X
	SET X = GPCBS(N,MO,SA,FC)
RESTRICTION:	Available only in ACC.

Purpose

The GPCBS function executes a CIMBUS function (FC) in a CIMBUS module (MO) at a subaddress (SA) through a serial link attached to serial I/O CAMAC modules in positions (N) and (N+1) in the CAMAC crate.

X is returned when FC is a Read function. When FC is a Write or single-pulse function then GPCBS is used in Write mode.

FUNCTION NAME:	MPXI – initialize MPX control unit
FUNCTION TYPE:	11 – call assembly language function
CALLING SEQUENCE:	MPXI(UC)
RESTRICTION:	None

Purpose

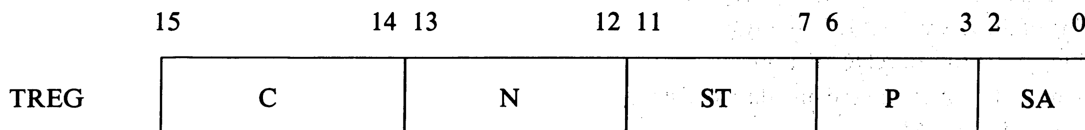
The MPXI function performs a general initialization of an MPX control unit (UC) and its associated MPX link.

Its action comprises:
- initialization of command and data registers;
- clearing of the LAM on ADC (if any);
- checking the status of ADC (if any);
- checking status of command and data registers;
- enabling of UC LAM;
- initialization of ALDs (if any).

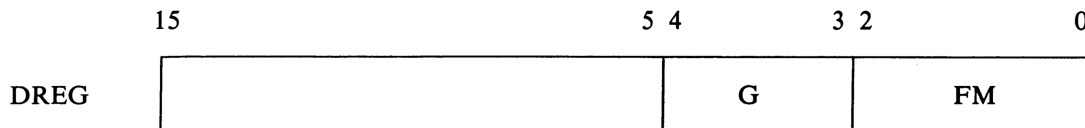
FUNCTION NAME: CMPX – MPX access
FUNCTION TYPE: 8 – read/write assembly language function
CALLING SEQUENCE: SET CMPX(TREG,DREG) = X
SET X = CMPX(TREG,DREG)
RESTRICTION: None

Purpose

The CMPX function executes an MPX function:



where
C = crate number (1 to 3)
N = UC in the crate (0 to 3)
ST = station number of the MPX module
P = peripheral number of the MPX module
SA = subaddress



FM = MPX function
= 0 or 4 for single pulse
= 1 or 5 for status test
= 3 or 7 for analog acquisition
= 2 for digital control
= 6 for digital acquisition
G = scale for analog acquisition
= 0 for 5 V scale
= 1 for 1 V scale
= 2 for 10 V scale

FUNCTION NAME:	GPMPX – MPX access
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET GPMPX(UC,ST,P,SA,FM) = X
	SET X = GPMPX(UC,ST,P,SA,FM)
RESTRICTION:	None

Purpose

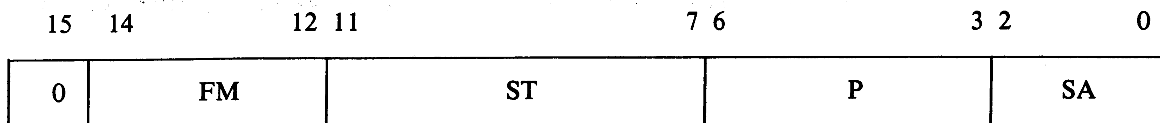
The GPMPX function executes an MPX function FM in an MPX module (P) in an MPX station (ST) at a subaddress (SA) through an MPX control unit (UC).

X is returned when FM is a Read function. When FM is a Write or Control function, GPMPX is used in Write mode.

FUNCTION NAME:	MPX – MPX access
FUNCTION TYPE:	8 – read/write assembly language function
CALLING SEQUENCE:	SET MPX(UC,FSPA) = X
	SET X = MPX(UC,FSPA)
RESTRICTION:	None

Purpose

The MPX function executes an MPX function through an MPX control unit UC.
The MPX module and function are completely defined by FSPA:



FM = MPX function
ST = MPX station
P = MPX module
SA = subaddress

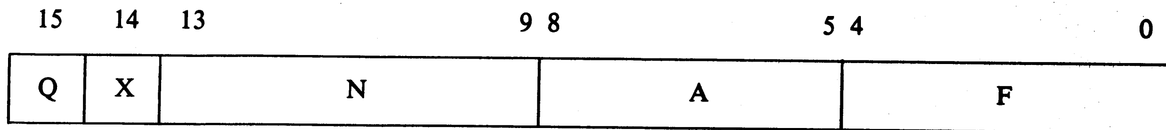
X is returned when FM is a Read function. When FM is a Write or Control function, MPX is used in Write mode.

COMMAND NAME: CADRIV – execute a row of CAMAC functions
CALLING SEQUENCE: CADRIV A B expression. A and B are integer arrays; expression is optional.
RESTRICTION: Available only in ACC

Purpose

The CADRIV command rapidly executes a whole row of CAMAC functions defined in the integer array A.

Layout of one word of A:



Q = Q response wanted
X = X response wanted
N = CAMAC module
A = CAMAC subaddress
F = CAMAC function.

When F is a Read function the result is written into array B. When F is a Write function the contents of B are written into the CAMAC module.

The function A(N) takes as parameter the value of B(N + expression).

When the expression is not specified, the value 0 is assumed as offset, and the setting-up time of the command is smallest, $\simeq$ 410 μs. When the expression is specified, the setting-up time will be increased by at least 1 ms depending on the type of expression.

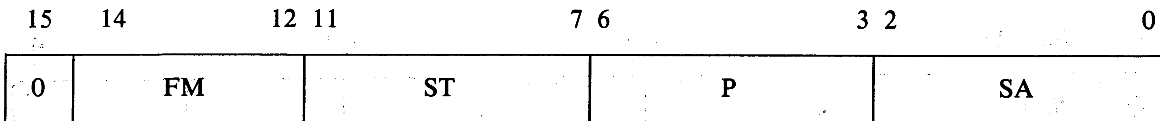
Each CAMAC access takes about $\sim$80 μs in the compiled mode.

FUNCTION NAME: MPDRIV – executes a row of MPX functions
CALLING SEQUENCE: MPDRIV A B expression, A and B are integer arrays, expression is optional
RESTRICTION: Available only in ACC

Purpose

The MPDRIV command rapidly executes a whole row of MPX functions defined in the integer array A.

Layout of one word of A:



FM = MPX function
ST = MPX station
P = MPX module
SA = subaddress

When FM is a Read function the result is written into array B. When FM is a Write or command function, the contents of B are written into the MPX module.

The function A(N) takes as parameter the value of B(N + expression). When the expression is not specified, the value 0 is assumed as offset, and the setting-up time of the command is smallest, ≈ 500 μs. When the expression is specified, the setting-up time will be increased by at least 1 ms, depending on the type of expression. Each MPX access takes about 200 μs.

FUNCTION NAME: CAMAC – execute CAMAC function

CALLING SEQUENCE:

CAMAC(N = ℓexp; A = ℓexp;
F = ℓexp) FROM C

CAMAC(N = ℓexp; A = ℓexp;
F = ℓexp) TO C.

C is variable or integer array
ℓexp = first value, step value, end value.
Default step value = 1.
Default end value = first value.

RESTRICTION: Available only in ACC

Purpose

The series of CAMAC functions specified by the three- loop expressions within the brackets are executed. When dummy values need to be specified, FROM C is replaced by FROM (V = ℓexp).

Example

CAMAC(N = 2; A = 0; F = 16) F (V = 1,200)

will write the numbers 1, 2, 3,..., 200 into N = 2; A = 0 with function 16.

When C is a variable, the value of C is used as an argument for the CAMAC functions specified within the brackets.

When C is an integer array of dimension k, k loops are executed when the number of CAMAC loops is 1. Otherwise the number of CAMAC loops should also be equal to k.