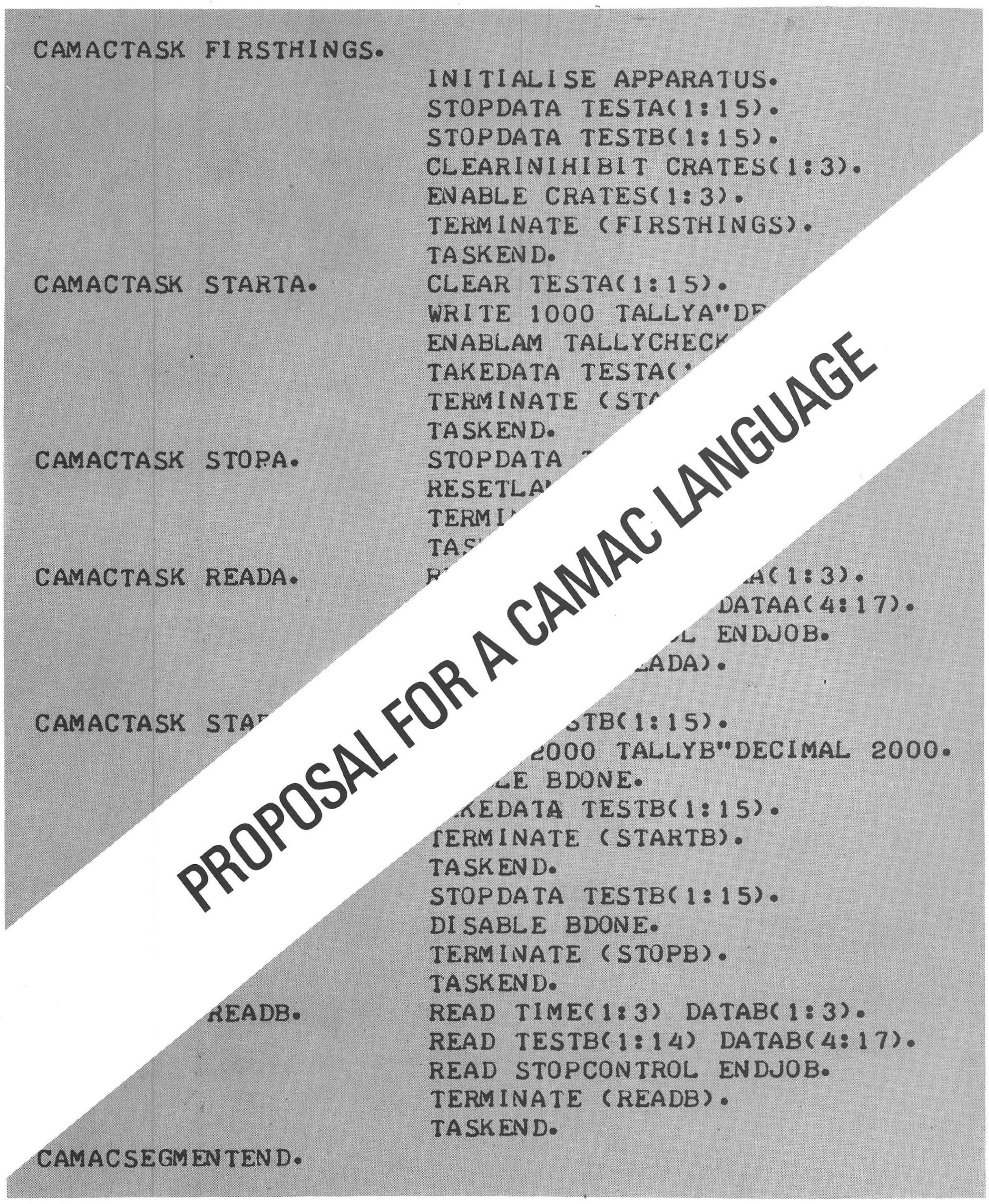
CAMAC bulletin

A publication
of the
ESONE Committee

ISSUE No.5

November 1972
Supplement



```
CAMACTASK FIRSTHINGS.
                          INITIALISE APPARATUS.
                          STOPDATA TESTA(1:15).
                          STOPDATA TESTB(1:15).
                          CLEARINIHIBIT CRATES(1:3).
                          ENABLE CRATES(1:3).
                          TERMINATE (FIRSTHINGS).
                          TASKEND.
CAMACTASK STARTA.         CLEAR TESTA(1:15).
                          WRITE 1000 TALLYA"DE
                          ENABLAM TALLYCHECK
                          TAKEDATA TESTA(
                          TERMINATE (ST
                          TASKEND.
CAMACTASK STOPA.          STOPDATA T
                          RESETLA
                          TERMI
                          TAS
CAMACTASK READA.          R                 A(1:3).
                                        DATAA(4:17).
                                      L ENDJOB.
                                       EADA).

CAMACTASK STA                  STB(1:15).
                             2000 TALLYB"DECIMAL 2000.
                             LE BDONE.
                           KEDATA TESTB(1:15).
                          TERMINATE (STARTB).
                          TASKEND.
                          STOPDATA TESTB(1:15).
                          DISABLE BDONE.
                          TERMINATE (STOPB).
                          TASKEND.
          READB.          READ TIME(1:3) DATAB(1:3).
                          READ TESTB(1:14) DATAB(4:17).
                          READ STOPCONTROL ENDJOB.
                          TERMINATE (READB).
                          TASKEND.
CAMACSEGMENTEND.
```

PROPOSAL FOR A CAMAC LANGUAGE

PREFACE

The widespread adoption of the CAMAC system of modular instrumentation in computer controlled real-time applications has generated a need for appropriate software. The ESONE Software Working Group has therefore generated this proposal for suitable software statements in the form of a language definition. These statements extend beyond those required to interact with CAMAC hardware because of the real-time environment and the need to supplement existing languages.

In this preliminary form the language is presented to potential users for comment on its usefulness in satisfying their needs and its suitability for implementation. This feedback is essential.

A wide range of facilities has been proposed so that appropriate subsets are available for different implementations. It is hoped that this approach, together with the inclusion of controversial items, will stimulate comment on the relative usefulness of the facilities.

During the development of the proposal it has been critically reviewed several times by the CAMAC Software Working Group of the USAEC NIM Committee and incorporates its many contributions and criticisms.

Interested individuals and organisations are invited to contact members of the software Working Groups (see, p 42) or to send their comments promptly to either.

H. Halling, Secretary ESONE SWG,
Kernforschungsanlage Jülich GmbH,
Zentrallabor für Elektronik/NE,
517 JÜLICH, Postfach 365, Germany.

or

S. Dhawan, Chairman NIM-CAMAC SWG,
Yale University, Sloane Laboratory,
NEW HAVEN, Connecticut 06520, U.S.A.

Distributed by :
Commission des Communautés Européennes
29, rue Aldringen
Luxembourg

Standard Software for CAMAC - Computer Systems

by

I. N. Hooton,

Chairman of the ESONE Software Working Group,

A.E.R.E., Harwell, England.

CAMAC

PROPOSAL FOR A CAMAC LANGUAGE

by the

ESONE COMMITTEE SOFTWARE WORKING GROUP

SUMMARY

This paper defines a language suitable for writing program statements referring to CAMAC hardware. These statements fall into two main classes, one being declaration statements associating names with various CAMAC entities, the other specifying the CAMAC activities which take place at run-time. The problem of implementation is briefly discussed.
Appendices give a proposed formal syntax and examples of how the statements may be used.

ESONE
SWG 14/72

Sept. 1972

Contents

1. INTRODUCTION

This paper defines a CAMAC language, that is a language reflecting the characteristics defined in EUR 4100e and EUR 4600e which are the detailed specifications of CAMAC hardware (Ref 1 and Ref 2). Although the definition cannot be mandatory, its general adoption (or the adoption of subsets derived from it) will have many advantages. Programs will be easier to write and, equally important, easier to read; the transfer of program descriptions will be easier as the peculiarities of the interface between the computer and CAMAC are eliminated; programs may be transferred between compatible installations, and, finally, co-operation in the writing of translators will be possible.

The CAMAC language defined in this report deals principally with those statements in a program that are concerned with communication between the computer and the hardware associated with CAMAC. Thus to write a complete program which also processes the information that comes from (or is sent to) the CAMAC hardware, the CAMAC language must be associated with a more conventional programming language. The latter may be an assembly language, a higher level language such as Fortran, or an advanced language containing real-time features. The association of these two languages, the CAMAC language and the host language, simplifies the use of CAMAC for data acquisition and control.

The particular features of the CAMAC language which simplify the writing of programs are the following:

(a) Descriptive names can be defined and used to refer to functional entities in a CAMAC system. For example a register implementing a counting function at a particular sub-address of a module may be given the name SCALER. Another register in the same module, even at the same sub-address but accessed through group 2 functions, may be given the name STATUS. At a higher level a collection of crates may be given the name EXPERIMENT, and a branch may be given the name LAB2.

(b) The particular details of I/O formats and conventions of specific computers and system controllers are not apparent to the application programmer.

(c) The book-keeping associated with block transfers can be organised and optimised by the system software.

It must be understood that it is only possible to consider the use of this language in an environment (computer and programming system) having the ability to communicate with CAMAC. This may be through assembly code, through sub-routine calls or through an input/output package in a higher level language. The CAMAC language simply provides a standard format which, through a translator, generates the code necessary to invoke these abilities. The language provides for all the hardware facilities defined in EUR 4100e and EUR 4600e. A particular system may not have implemented all these facilities and hence will not be able to make use of all the features of the language.

This definition has to serve three distinct groups of readers. The needs of the application programmer are served by the informal descriptions of the properties of the language and by the examples. Second, the translator writer's needs are served by the more formal syntactic definitions, which are necessary to ensure that all the possible constructions will be recognised by, or explicitly rejected by, a particular translator. Third, the designer of system controller hardware will find the definition useful in designing a controller to match a particular computer to CAMAC.

Section 2 defines the notation used to express the formal syntax of the CAMAC language. Section 3 consists of the CAMAC language definition using both informal descriptions and the syntax notation. This section is divided into several parts; the first of these (3.1) defines certain basic elements which are used throughout the following definitions.

Translator directives are defined in 3.2. These allow CAMAC language statements to be recognised and also delimit sections of the CAMAC program.

A series of declaration formats are defined in 3.3. They allow the programmer to rename some of the translator directives and the CAMAC function codes. They can be used in some cases to achieve compatibility with the host language, in others to prevent conflict. They can be used to introduce names which have a direct meaning in the application area involved. They allow the programmer to replace the set of defined English language mnemonics by others, possibly in his native language and to introduce mnemonics for any of the undefined CAMAC function codes he may use. Other declarations allow the naming of CAMAC hardware, either as individual items or as arrays, the naming of demands from CAMAC and the naming of computer locations as variables, arrays and lists.

The statements which are used to specify the operations normally associated with CAMAC are defined in 3.4. These are statements concerned with moving data, either single words or blocks between CAMAC and the computer or concerned with control activity in CAMAC modules. It also defines a conditional statement specifying branching on flag conditions. In the CAMAC definitions the Q line is the primary physical flag but it has different meanings depending on the function code and sub-address. Statements required for interaction with the operating-system, particularly in a multi-task environment, are also proposed in this section.

It was mentioned earlier that this language is seen as being used in conjunction with another language. Section 4 discusses the relationship between these two languages. It may be very close when using embedded code or tenuous when using an autonomous CAMAC processor. These two extremes are apparent in the examples.

Section 5 lists some of the possibilities for further development.

Appendix 1 brings the definitions together into a formal syntax for the language.

Appendix 2 contains examples showing how the various types of statement can be combined to produce a program. It has been made more complex than is absolutely necessary in order to introduce most of the defined statements and in particular to demonstrate the flexibility of the CAMAC naming section.

2. SYNTAX NOTATION

The syntax notation is used to describe the structure of the language elements, not their meaning. It indicates the order in which elements may, or must, appear, the punctuation that is required and the options that are allowed.

(1) A notation variable is the name of a general class of elements in the programming language. A notation variable is a string of characters chosen from the lower case letters, decimal digits and hyphen. It must begin with a letter. The composite symbol ::= is used in this context with the meaning "is defined as".

(2) A notation constant denotes the literal occurrence of the characters represented.

(3) Braces { }are used to denote the grouping of more than one element into a syntactic unit. Individual notation variables or constants are themselves syntactic units.

(4) A vertical stroke | indicates that a choice is to be made. It may be interpreted as the equivalent of "or".

(5) Square brackets [ ] denote options. Anything enclosed in such brackets may appear once or it may not appear at all.

(6) Three dots ... denote the repetition of the immediately preceding syntactic unit none, one or more times in succession.

(7) Spaces are optional immediately preceding or succeeding ::= or | or [or] or { or } or ... and between adjacent units. A space is mandatory between adjacent notation variables and adjacent notation constants.

3. LANGUAGE ELEMENTS

3.1 Basic Symbols

3.1.1 Characters

letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
alphanumeric-character ::= letter | digit
operator ::= adop | mulop
adop ::= + | −   where + means add
mulop ::= * | /   where − means subtract
where * means multiply
where / means divide
as integer operators.

It has been proposed that the definition of operator be extended to include logical operators such that
operator ::= adop | mulop | logop
logop ::= # | & | ! | @   where # means not
where & means and
where ! means inclusive or
where @ means exclusive or
The logical operators have lower precedence than adop and are written in the order of descending precedence.

delimiter ::= : | , | (|) | | = | ' | operator
Note the delimiter | | representing a space.

CAMAC is a free-format language. Multiple spaces or formatting characters (e.g. end-of-line, tabulation) are syntactically equivalent to a single space. Spaces, or their equivalents, adjacent to delimiters (including the delimiter 'space') are ignored by the syntax and may be used freely to aid the physical layout of the program. All symbols must be delimited by a space or its equivalent unless some other delimiter is specified.

syntax-character ::= alphanumeric-character | delimiter
non-syntax-character ::= ! | " | # | $ | % | & | . | ; | < | > | ? | @ | [| \ |] | ↑ | _
character ::= syntax-character | non-syntax-character
The characters are the ASCII characters having binary form represented by the decimal numbers from 32 to 95 inclusive (octal 40 to 137) Reference 3.

3.1.2 Syntax Elements

The basic syntax elements of the language are formed from the characters listed above with the following constraints. A number of specific character strings are used as System Symbols, and these are introduced at appropriate points in the text. Elementary character strings which the user constructs are:
integer ::= digit ...
denoting the positive decimal integral number represented by the string of digits, and
identifier ::= letter [alphanumeric-character ...]
used as a name for an entity within the language.

It has been proposed that integer be defined to permit the presentation of integers in binary, octal, decimal and hexadecimal form (see the definition of number in 3.4.3).

Identifiers identical to system symbols may not be used. The standard system symbols are listed in Appendix 1.

3.1.3 CAMAC Hardware Elements

References to CAMAC hardware make use of B, C, N, A, I to represent Branch, Crate, Station-number, Sub-address and Bit-position respectively. The symbols F, G1 and G2 refer to the CAMAC Function and to the Group 1 and Group 2 subsets of these functions. As an aid to programming G1 and G2 are assigned as access modes of the hardware register. Other access modes are provided by the qualifiers P, Q, R and S (see section 3.3.2). The symbol GL refers to the Graded-L register within the system controller. The state of the Response line during a CAMAC operation is stored in the system controller and is accessed directly as Q. Flags within modules are accessed as LAM or STATUS depending on the functions providing access.

The Command Accepted signal (X) is not directly specified in the CAMAC language but may be accessed by a user defined flag S(integer).

3.2 Translator Directives

The main function of these directives is to simplify the design of the CAMAC translator by defining the limits of various types of information. The directives are in the form of English words. Those which are used frequently may be equivalenced to a single symbol. Alternatively this re-naming may be done as a preamble by the translator.

CAMACSEGMENT	defines the start of a CAMAC segment within a total program. It precedes the segment-name in the first statement in the CAMAC language. It includes the attributes of BEGINCAMAC. It may not be equivalenced.
ENDSTATEMENT	is used to terminate each statement in the CAMAC language. It will usually be equivalenced to a single symbol, for example a semi-colon (;).
BEGINCAMAC	introduces a statement or a sequence of statements which are all in the CAMAC language. It is not a separate statement but a prequalifier preceding the statement. It may be equivalenced.
ENDCAMAC	terminates the scope of BEGINCAMAC and the BEGINCAMAC attributes of CAMACSEGMENT. It includes the attributes of ENDSTATEMENT. It may be equivalenced.
CAMACTASK	defines the start of a task for separate execution. It precedes the task name within the first statement of the task. It may be equivalenced.
TASKEND	terminates the scope of the preceding CAMACTASK. It is a complete statement and may be equivalenced.
CAMACSEGMENTEND	terminates the scope of CAMACSEGMENT. It includes the attributes of ENDCAMAC. It is a complete statement and may be equivalenced.
NOTE	introduces a comment. It follows the last element of the effective statement and is closed by any directive which has the attribute of ENDSTATEMENT. It cannot include any such directive. It may be used as the first element of a statement, i.e. a statement of comment only. It may be equivalenced.
CEQV CNAME CDMD CDCL CREF	are section headers introducing the various declarative parts of the program. They form part of a statement which may include a comment but which must be terminated by ENDSTATEMENT or its equivalent. The section headers may not be equivalenced. Each section is terminated by a new section header.
CACT	is a section header for the action part of the program and may not be equivalenced. It also ends the declarative parts of the program. The action section is terminated by CAMACSEGMENTEND.

3.2.1 Structure of Program

The CAMAC statements of a combined-language program are contained within CAMAC segments. A CAMAC segment extends from CAMACSEGMENT to CAMACSEGMENTEND directives. It is defined as capable of independent compilation by the CAMAC translator. A segment may be written exclusively in the CAMAC language or it may, by use of the ENDCAMAC-BEGINCAMAC bracketing directives, overlap with host language statements.

A segment may contain both declaration statements and action statements. It may also contain one or more CAMAC tasks. A task cannot contain declaration-statements. It may contain statements which cause another task to be executed and so on. The lexical structure of a single CAMAC segment may be illustrated informally as follows.

```
non-camac-text
    CAMACSEGMENT segment-name terminator
        CEQV   ⎫
        CNAME  ⎪
        CDMD   ⎬  declaration sections
        CDCL   ⎪
        CREF   ⎭
    CACT
    actions
        CAMACTASK task-name-1 terminator
        actions
        TASKEND
        CAMACTASK task-name-2 terminator
        actions
        TASKEND
    CAMACSEGMENTEND
non-camac-text
```

3.3 Declarations

There are five classes of declaration statement, each in a defined section of the program. The first is the class of equivalence statements which are of two different types, one allowing alternative names for certain system words held in the translator symbol tables, the other allowing identifiers to be associated with constant values known at compile time. The second class consists of those statements used to allocate names to parts of the CAMAC hardware. The third class of statements is that which associates names with demands from the CAMAC hardware. The fourth class consists of those statements providing information about software buffers and other storage locations. The fifth class lists those names which may be referenced by other program segments.

3.3.1 Equivalence Statements

There are two kinds of equivalence statement. The first is an EQV statement which allows the programmer to give alternative names to certain special symbols. These symbols are certain of the translator directives defined in para. 3.2 and the CAMAC function code forms F(n). The status flags may also be equivalenced. The result of the EQV statement is effective after the recognition of the terminator of the statement. The old symbol is not erased so there may be multiple forms of the special system symbols. The new symbol introduced is restricted to be either a single non-syntax-character or an identifier. System symbols may not be used on the left hand side of an EQV statement.

The second form of statement is an equals statement (=). This may be used to simplify much of the elementary arithmetic used to arrive at the numerical values used particularly in the CAMAC naming section. The statement allows integer values to be given to identifiers (symbolic constants) and the combination of these integers in arithmetic expressions. A single pass structure is imposed on the naming section by the rule that all symbolic constants must be given a value by appearing on the left hand side of an equals statement before being used on the right hand side.

The equivalence statements are contained in a section defined by the following syntax.

    ceqv-section ::= ceqv-section-header ceqv-statement ...
    ceqv-section-header ::= CEQV terminator
    terminator ::= [comment] ENDSTATEMENT
    comment ::= NOTE [non-camac-text]

Non-camac-text is any string of characters not containing terminating directives or any substrings to which they have been equivalenced.

    ceqv-statement ::= ceqv-statement-body terminator
    ceqv-statement-body ::= ceqv-s-b1 | ceqv-s-b2
    ceqv-s-b1 ::= {non-syntax-character | identifier}EQV special-system-symbol

Special system symbols are defined in Appendix 1.

    ceqv-s-b2 ::= symbolic-constant = {integer | expression}
    symbolic-constant ::= identifier
    expression ::= term [{adop term}...]
    term ::= primary [{mulop primary}...]
    primary ::= constant | (constant {adop constant}...)
    constant ::= integer | symbolic-constant
    adop ::= + | -
    mulop ::= * | /

To minimise the use of parentheses the usual conventions are followed. Multiplication and division have equal precedence and take precedence over addition and subtraction which each have the same precedence. For operators with equal precedence the order of association is from left to right.

It has been proposed that expression be defined so that a term may itself be an expression. This would allow more elaborate forms and save the programmer from generating intermediate symbolic-constants.

3.3.2 CAMAC Naming Statements

The main items that need naming are registers which are usually thought of as residing at a sub-address, of a module, in a crate, on a branch. The statements of the naming section allow this long reference to be stated explicitly or to be developed, for example, in an hierarchical manner by first giving names to larger entities, e.g. branches and crates and using these in the subsequent naming of smaller entities, e.g. modules and sub-addresses.

Certain qualifying information may be given in addition to the B, C, N, A, I values. One class is an address extension indicating Group 1 or Group 2 registers (G1, G2). If neither is specified G1 is assumed if such a qualifier is required. The second class defines the access mode of a data array or sequence. Briefly these access qualifiers have the following meanings.

The qualifier P specifies that the named array of CAMAC registers is to be addressed in parallel.

The qualifier Q specifies that the named array is to be stepped through sequentially with the address changes controlled by the Response (Q) from the module. This is the Address Scan mode defined in EUR 4100e section 5.4.3.1.

The qualifier R specifies hardware which is to be accessed in the Repeat mode defined in EUR 4100e section 5.4.3.2. The hardware generates Q=1 during the Read or Write operation if it is ready to participate in a data transfer. Otherwise it generates Q=0.

The qualifier S specifies hardware which is to be accessed in the Stop mode defined in EUR 4100e section 5.4.3.3. The hardware generates Q=1 during each Read or Write operation while the block of data is being transferred and generates Q=0 for the operation after the end of the block.

Only one access qualifier can be given to a particular hardware name and must be explicitly declared. When different modes are used at different points in a program then different names must be declared.

The statements relating to CAMAC hardware name allocation are contained in a CAMAC naming section defined in the following way.

```
    c-name-section ::= c-name-header c-name-statement ...
    c-name-header ::= CNAME terminator
    c-name-statement ::= c-name-statement-body terminator
    c-name-statement-body ::= c-name-s-b1 | c-name-s-b2
    c-name-s-b1 ::= c-name [(size)] = address-set [G1 | G2] [mode]
    c-name ::= identifier
    size ::= 1: constant              If no size is specified c-name is assumed to be a
    mode ::= P|Q|R|S                  single element.
```

It has been proposed that size be defined as

```
    size ::= constant : constant
```

in order to permit values significant to the programmer to be used or to enable arrays to be developed in a series of definitions.

```
    address-set ::= address-value [{ , address value}...]
```

Address values are sufficiently specified CAMAC addresses, i.e. they contain all the necessary Branch, Crate, Module or Sub-address references relative to their level.

```
    address-value ::= address-component [address-component ...]
    address-component ::= {c-name [(address-list)] } | {c-type (address-list)}
    c-type ::= B|C|N|A|I
    address-list ::= address-element [{ , address-element}...]
    address-element ::= constant [: constant [: constant] ]
```

Address-element permits an array to be specified. The interpretation when three constants are used is "from the first, to the second incrementing by the third". With two constants the increment is taken as one and with one constant reference is made to one member.

The address values must not have more than one component of a given c-type. The order of writing the components has meaning when sequential access is used, the rightmost component is cycled through first, then the second from the right and so on.

```
    c-name-s-b2 ::= c-name [(size)] = EXT (c-type) [G1 | G2] [mode]
```

This permits CAMAC hardware to be referred to in one segment even though the absolute BCNAI values are not known. The size, G1 or G2 and mode provide information required for the translation of the segment. The c-type specifies the lowest element in the hierarchy B, C, N, A, I (usually A or I) so that appropriate functions may be generated. The companion segment which specifies the hardware address must contain the c-name in its CREF section (see 3.3.5). The naming section is given a single pass structure by the following rules.

a The CEQV section which makes equivalences must precede the CNAME section in which they are used so that all symbolic constants have values.

b All hardware names (c-name) must be associated with numerical values by having appeared on the left-hand side of an address allocation statement before being used on the right-hand side.

3.3.3 Demand Naming Statements

The source of a particular demand must be known if it is to be controlled. Within a module the source may be identified either by a subaddress or by a bit-position in the Lam Status Register (see

EUR 4100e (1972) section 5.4). The demand from a crate is controlled through a virtual module, N30, at sub-address 10 (see EUR 4600e section A1.6.1).

All these can be given CAMAC names. The CAMAC naming section also allows branches to be given names, and so a branch demand can be associated with a name though the detailed control signals will depend on the design of branch driver or system controller.

Thus at all levels the demand signals can be associated with a CAMAC name which has to be defined in the CNAME section of the program. These demand signals are then given demand names in a demand naming section, CDMD (CAMAC demand) with the following syntax.

c-dmd-section ::= c-dmd-header c-dmd-statement ...
c-dmd-header ::= CDMD terminator
c-dmd-statement ::= c-dmd-statement-body terminator
c-dmd-statement-body ::= c-dmd-s-b1 | c-dmd-s-b2
c-dmd-s-b1 ::= demand-name = c-name [GL integer]
demand-name ::= identifier

The GL integer indicates that the demand source is connected to the particular GL bit.

c-dmd-s-b2 ::= demand-name = EXT (c-type)

This permits a demand to be specified for use within a segment while the link to a c-name is defined in another segment.

3.3.4 Software Naming Statements

There are two types of software name to be considered, one refers to data buffer areas which are involved in transfers, the other refers to store locations holding CAMAC hardware addresses. The section also includes a statement which can be used to load these locations with the values associated with defined CAMAC hardware names.

The data buffer areas need further definition to indicate whether each named word has to hold the whole 24 CAMAC Read Write bits or whether it can be limited to the computer word length. When the transferred wordlength is 24 bits and the computer wordlength is less, then two (or more) computer words must be allocated for each transferred word. The CAMAC word is right aligned in two (or more) successive computer words with leading zeros filled.

The naming is done in a section introduced by a header CDCL (CAMAC declaration) with the following syntax.

cdcl-section ::= cdcl-section-header cdcl-statement ...
cdcl-section-header ::= CDCL terminator
cdcl-statement ::= cdcl-statement-body terminator
cdcl-statement-body ::= cdcl-s-b1 | cdcl-s-b2 | cdcl-s-b3 | cdcl-s-b4
cdcl-s-b1 ::= [d-type] d-item [{ , d-item}...]
d-type ::= CAMACLENGTH | COMPUTERLENGTH

If no d-type is specified then COMPUTERLENGTH is assumed.

An alternative proposal defines

d-type ::= DATALENGTH (d)

where d specifies the minimum number of bits required to contain the transferred data.

d-item ::= v-item | a-item | l-item
v-item ::= variable-name
variable-name ::= variable
a-item ::= array-name (size)
array-name ::= variable
l-item ::= list-name (size) L
list-name ::= variable

Access to an a-item is always by the array-name and a subscript which determines the element or elements of the array to be accessed.

Access to an l-item can be either by the list-name subscripted as for an a-name, or by the list-name without a subscript. In this latter case successive accesses are directed to consecutive elements of the list.

```
cdcl-s-b2 ::= p-type p-item [{ , p-item}...]
p-type ::= CAMACADDRESS
p-item ::= p-name [(size)]
p-name ::= variable
variable ::= identifier
cdcl-s-b3 ::= p-name [(range)] = c-name [(range)]
range ::= constant [ : constant]
cdcl-s-b4 ::= EXT e-item [{ , e-item}...]
e-item ::= v-item | a-item | l-item | p-item P
```

The identifiers used in this section should conform to the conventions of the host programming system.

The problem of establishing linkages to the data processing part of the program and the protection of the data buffers is the responsibility of the program writer using his knowledge of the total programming system being used.

The translator may need a number of variables which are private in the sense that they are of no interest to the application programmer but they would be passed to any host language programming system. A number of identifiers must be reserved. The programmer is forbidden to use identifiers starting with letters OO.

3.3.5 Reference Statements

Within a program containing separate segments it is necessary to be able to define the scope of declarations. Segment names are assumed to be valid outside the segment in which they are declared. CAMAC hardware names, demand names and task names are local to the segment in which they are declared unless they are specifically listed as available to be referenced. This is done in a section introduced by a header CREF (CAMAC references) with the following syntax.

```
c-ref-section ::= c-ref-header c-ref-statement ...
c-ref-header ::= CREF terminator
c-ref-statement ::= c-ref-statement-body terminator
c-ref-statement body ::= c-ref-s-b1 | c-ref-s-b2 | c-ref-s-b3 | c-ref-s-b4 | c-ref-s-b5 | c-ref-s-b6 |
                         c-ref-s-b7
c-ref-s-b1 ::= NAME c-name [ { , c-name}...]
c-ref-s-b2 ::= DEMAND demand-name [ { , demand-name}...]
c-ref-s-b3 ::= TASK task-name [ { , task-name}...]
c-ref-s-b4 ::= VARIABLE variable-name [{ , variable-name }...]
c-ref-s-b5 ::= ARRAY array-name [{ , array-name }...]
c-ref-s-b6 ::= LIST list-name [{ , list-name }...]
c-ref-s-b7 ::= ref-item [ { , ref-item}...]
ref-item ::= {c-name | demand-name | task-name | variable-name | array-name | list-name}
```

These declarations state that the names, which are fully defined within the segment, have validity outside the segment and may be referenced by other segments using EXT declarations.

3.3.6 Order of Declaration Sections

The definitions do not specify the order of the various declaration sections and there may be more than one of each in a segment. Certain rules are necessary to simplify any translator design. These are as follows.

(1) All equivalences must be defined before being used.

(2) All symbolic constants must be assigned values before being used.

(3) All c-names must be defined before being used.

3.4 Action Statements

The statements which cause activity in the CAMAC system at run time are simple in concept and few in number. These action statements involve information transfer between the computer and CAMAC or within CAMAC. The information may be data, status or control. The CAMAC hardware may be user modules or system hardware, crate controllers or system controllers.

The action statements are in a section introduced by a header CACT (CAMAC action) with the following syntax.

```
action-section ::= action-section-header action-statement ...
action-section-header ::= CACT terminator
action-statement ::= [label] {[repeat-spec] {transfer-statement | control-statement}} |
                     {branch-statement | executive-statement}
```

3.4.1 Labels

All statements defined in this section (3.4) may be labelled for reference.

```
label ::= label-name :
label-name ::= {letter | digit}...
```

The label-name must conform to the conventions of the host programming system.

3.4.2 Repeat Qualifier

Transfer statements (3.4.3) and control statements (3.4.4) may be qualified by a repeat specification which directly precedes the statement body and occurs after any label.

```
repeat-spec ::= REPEAT (integer-denotation)
integer-denotation ::= variable | constant
```

The repeat qualifier causes the operation to be repeated the specified number of times precisely as if the unqualified statement had been written that number of times in sequence.

3.4.3 Transfer Statements

The characteristic of this group of statements is that they involve two data references, at least one of which is in the CAMAC hardware, the external reference. In addition they will usually, but not universally, involve a store location or an array of store locations, the internal reference.

The language does not allow reference to named registers such as "accumulator". Certain computers and programming systems may have a uniform addressing structure in which registers and store are treated in a uniform way. In such cases registers can be referenced.

Informally the syntax of the transfer statements is of the form

```
transfer-statement ::= transfer-action source destination
```

The transfer-action part of the statement has to allow the expression of all the CAMAC function codes involving data transfer. They may be in the form F(n) where n is an appropriate integer between 0 and 31, or they may be in the form of standard mnemonics as defined in table 1.

Alternatively the transfer-action may be a mnemonic defined by an equivalence statement. Possible forms of the latter would be

```
LIRE EQV F(0)
SCHREIBE EQV F(16)
NEWFUNCTION EQV F(6)
```

One of the standard mnemonics is TRANSFER. This is unique in that it involves two external references. It implies reading data from the first reference, the source and then writing this same data to the second reference, the destination.

Table 1 lists the interpretation of the standard mnemonics as CAMAC function codes.

The formal definitions of the transfer statements follow.

```
transfer-statement ::= transfer-statement-body terminator
transfer-statement-body ::= t-s-b1 | t-s-b2 | t-s-b3 | t-s-b4 | t-s-b5
t-s-b1 ::= transfer-action-1 external-reference internal-reference
transfer-action-1 ::= F(n1) | READ | READCLR | READCOMP | READLAM | READSTAT |
                      MOVE
n1 ::= 0|1|2|3|4|5|6|7|8|27

t-s-b2 ::= transfer-action-2 {number | internal-reference}external-reference
transfer-action-2 ::= F(n2) | WRITE | SETSEL | CLEARSEL | MOVE
n2 ::= 16|17|18|19|20|21|22|23
number ::= constant | binary | octal | hexadecimal
binary ::= BIN'{0|1}...'
octal ::= OCT'{0|1|2|3|4|5|6|7}...'
hexadecimal ::= HEX'{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...'
```

The maximum value of all numbers is 24 bits equivalent. It is proposed that the bracketing single quotes (') be omitted.

```
internal-reference ::= variable-name |{array-name (array-spec)} | {list-name (array-spec)}
array-spec ::= integer-denotation [ : integer-denotation]

t-s-b3 ::= {TRANSFER | MOVE}source destination
source ::= external-reference
destination ::= external-reference

t-s-b4 ::= {READ | MOVE}external-reference list-name [EXIT] label-name

t-s-b5 ::= { WRITE | MOVE}list-name external-reference [EXIT] label-name
```

These two transfer statements allow data to be accessed sequentially from arrays that have been defined as lists (see section 3.3.4). When the last word in the list is accessed the program branches to label-name.

```
external-reference ::= {c-name [(range)] } | {p-name [(array-spec)] } | direct-reference |
                       demand-name
direct-reference ::= {[B(integer-denotation)] C(integer-denotation) N(integer-denotation)
                     A(integer-denotation) [I(integer-denotation)] [G1 | G2] } | GL
```

GL refers to the information obtained by a GL Request as the Graded-L word.

```
integer-denotation ::= constant | variable
```

The direct reference only allows the specification of one CAMAC address. The integer-denotation components refer to crate, station and sub-address values with optional branch and bit values. The variable part of integer-denotation allows run time modifications of address.

TABLE 1—THE INTERPRETATION OF MNEMONICS INTO FUNCTION CODES

MNEMONIC	STATEMENT-BODY	EXTERNAL REF c-type	EXTERNAL REF G1/G2	FUNCTION CODE	REMARKS

READ or MOVE	t-s-b1, t-s-b4	A
		A
			G1
			G2
				F(0)
				F(1)	

READCLR	t-s-b1	A	G1	F(2)	*Note 1*
READCOMP	t-s-b1	A	G1	F(3)	*Note 1*
READLAM	t-s-b1	A	NIL	F(8)	*Note 2*
READSTAT	t-s-b1	A	NIL	F(27)	*Note 2*

WRITE or MOVE	t-s-b2, t-s-b5	A
		A
			G1
			G2
				F(16)
				F(17)	

SETSEL	t-s-b2	A
		A
			G1
			G2
				F(18)
				F(19)	

CLEARSEL	t-s-b2	A
		A
			G1
			G2
				F(21)
				F(23)	

TRANSFER
or
MOVE
	t-s-b3	A
		A
			G1
			G2
				F(0)
				F(1)
					from source

		A
		A
			G1
			G2
				F(16)
				F(17)
					to destination

CLEAR	c-s-b1	A
		A
			G1
			G2
				F(9)
				F(11)	

		I
		I
			G1
			G2
				F(21)
				F(23)
					Note 3

CLEARLAM	c-s-b1	A	NIL	F(10)	*Note 2*
		I	G2	F(23)	at A(12) *Note 3*

ENABLE
or
SET
	c-s-b1	A
		I
		I
			NIL
			G1
			G2
				F(26)
				F(18)
				F(19)
					Note 2
					Note 3
					Note 3

DISABLE	c-s-b1	A
		I
		I
			NIL
			G1
			G2
				F(24)
				F(21)
				F(23)
					Note 2
					Note 3
					Note 3

EXECUTE	c-s-b1	A	NIL	F(25)	*Note 2*
INITIALISE	c-s-b2	B		—	Generate BZ
		C		F(26)	at N(28) A(8)
SETINHIBIT	c-s-b2	C		F(26)	at N(30) A(9)
CLEARINHIBIT	c-s-b2	C		F(24)	at N(30) A(9)
ENABLEINT	c-s-b2	B			Enable BD input
		C		F(26)	at N(30) A(10)
DISABLEINT	c-s-b2	B			Disable BD input
		C		F(24)	at N(30) A(10)
CLEARSYS	c-s-b2	C		F(26)	at N(28) A(9)
LAM *Note 4*	branch	A	NIL	F(8)	*Note 2*
		I	G2	F(1)	at A(14)
STATUS *Note 4*	branch	A	NIL	F(27)	*Note 2*
		I	G2	F(1)	at A(14)
		I	G1	F(0)	

Note 1: If no group is specified Group 1 is assumed. The specification of Group 2 is an error condition.

Note 2: The specification of Group 1 or Group 2 is an error condition.

Note 3: Bit(s) 1 must be set in the internal reference

Note 4: Or any name equivalenced.

3.4.4 Control Statements

These have a slightly simpler form as they do not involve an internal reference.

control-statement ::= control-statement-body terminator
control-statement-body ::= c-s-b1 | c-s-b2
c-s-b1 ::= control-action-1 external-reference
control-action-1 ::= F(n3) | CLEAR | CLEARLAM | ENABLE | DISABLE | EXECUTE | SET
n3 ::= 9 | 10 | 11 | 12 | 13 | 14 | 15 | 18 | 19 | 21 | 23 | 24 | 25 | 26 | 28 | 29 | 30 | 31

c-s-b2 ::= control-action-2 external-reference
control-action-2 ::= INITIALISE | SETINHIBIT | CLEARINHIBIT | ENABLEINT | DISABLEINT | CLEARSYS

The control statements are divided into two classes due to the different address domain of the external reference. For class 1, in common with all other action statements, this is at c-type A (sub-address) or I(bit-position). For class 2 it is at either c-type C (crate) or B (branch).

3.4.5 Branching Statements

These statements are used to specify unconditional branching and branching depending on the condition of the external hardware. There is one response defined in CAMAC, the Q line. It will be referred to as LAM when it gives the response to F(8), as STATUS when it gives the response to F(27) and Q when it is used to refer simply to the Response received from the last CAMAC operation. There may also be flags S(integer) in the system controller which are inherently implementation dependent. Names can be assigned to them in an equivalence section.

The syntax of the branch statement is as follows.

branch-statement ::= branch-statement-body terminator
branch-statement-body ::= [condition flag] GOTO label-name
condition ::= IF | IFNOT
flag ::= {camac-flag external-reference} | system-flag
camac-flag ::= LAM | STATUS
system-flag ::= Q | {S(integer)}

3.4.6 Executive Statements

The applications of CAMAC are often in fields which depend critically on timing and the synchronisation of tasks. In some implementations of the combined language it is desirable or even essential to separate the CAMAC language and host language statements into distinct segments. It is not then possible to rely exclusively on the host language for the executive calls required. The following statements are therefore under active consideration for inclusion in the CAMAC language.

executive-statement ::= executive-statement-body terminator
executive-statement-body ::= e-s-b1 | e-s-b2 | e-s-b3 | e-s-b4 | e-s-b5 | e-s-b6

e-s-b1 ::= TERMINATE | QUIT (task-name)

TERMINATE	is used within a task to indicate that it has completed.
QUIT	is used within a task to suspend it. A further stimulus is expected which will cause the task to be resumed.

These two statements are used in CAMAC tasks which are executed in response to an external stimulus.

e-s-b2 ::= {INITIATE | AWAIT | TERMINATE | DONOW} (task-name)

INITIATE	schedules the task (task-name) to run concurrently.
AWAIT	suspends the current activity unless or until the task (task-name) has completed.
TERMINATE	causes the task (task-name) to terminate.
DONOW	combines INITIATE and AWAIT, that is it schedules the task (task-name) to run and suspends the current activity until the task has completed. It is the analogue of "CALL" in non-real-time systems.

These statements are used by the program controlling a task.

e-s-b3 ::= ACTIVATE (task-name, demand-name)

ACTIVATE	schedules the task (task-name) to run concurrently on the demand (demand-name).

e-s-b4 ::= ALLOCATE demand-name PRIORITY (integer-denotation)

ALLOCATE	gives a software controlled priority in the demand-structure as compared with the hardwired priority available in the GL pattern.

All the above statements provide for interaction with the operating system and are therefore only appropriate in a suitable environment. Such statements are however essential in a real-time-multi-tasking situation.

e-s-b5 ::= LINK (demand-name, label-name)

LINK	connects a demand-name to a location in order to make use of suitable host language statements.

e-s-b6 ::= INITIALISE (list-name)

INITIALISE	causes the next transfer to access the first item in the list.

4. IMPLEMENTATION

As there are none of the usual data processing statements in the CAMAC language it must be used in conjunction with a conventional programming language. Thus two translation processes may be necessary before a program can be effective. One is the translation of the CAMAC language, the other is the translation of the data processing language. The relationship between the two processes may be very close or very tenuous.

The close relationship is typified by an application or programming style in which CAMAC language statements and data processing language statements are intermixed and there is no clear functional separation between the two parts of the program. In this case the development of the CAMAC language translator will be simplified if use can be made of the data processing language translator which would then act as a host for the CAMAC language. From this follows the idea of a preprocessing translator which accepts a program written in the mixed language, the CAMAC language and the host language, and produces as output a program entirely in the host language. The host language is therefore the target language of the translator.

The tenuous relationship exists when the overall program is written in separately compiled sections which are later consolidated by a loader program. Here the CAMAC language statements do not occur in the same sections as the processing language statements. The target language is then the instruction code required by the hardware or software controlling CAMAC in a form acceptable by the loader.

A single translation process is sufficient if an existing data processing language is extended to include CAMAC statements, or if a data processing language is developed which includes CAMAC.

These three approaches are respectively for a mixture of language statements, two independent languages and a single language. The problems of implementation vary greatly with the approach adopted.

Within the CAMAC language there are definitions which allow the easy expression of statements which can evoke all the complex possibilities defined in EUR 4100e and EUR 4600e. Particular implementations need not provide all the facilities defined. In these cases an appropriate sub-set of the complete language may be selected and used in the form and spirit of this definition.

Facilities within the translator can augment those of the language definition. The depth of error diagnosis should be considered for example. Helpful documentation aids could be provided, various forms of listing and symbol tables could be provided as options. This would be particularly important for debugging with an implementation using a host language.

5. FUTURE EXTENSIONS

Certain other facilities have been briefly considered by the Software Working Group and these may result in additional definitions at a later stage.

The possibility of using the contents of external addresses directly as operands has been discussed. Statement types may be added to allow arithmetic and logic operations between variables at least one of which is in CAMAC hardware.

Statement types may be added to allow the specification of privilege and the implementation of protection facilities.

6. REFERENCES

1. CAMAC: A modular instrumentation system for data handling. Description and Specification. EURATOM EUR 4100e 1972.

2. CAMAC: Organisation of multi-crate systems. Specification of the Branch Highway and CAMAC Crate Controller Type A. EURATOM EUR 4600e 1972.

3. Standard Code for Information Interchange. USA Standard X34 1968. Available from: USA Standards Institute, 1430 Broadway, New York, N.Y.10016.

Appendix 1

THE SYNTAX OF THE LANGUAGE

This appendix brings together all the definitions from section 3 and uses the notation specified in section 2.

It is an analytical approach starting from the definition of a program and continuing until either a basic-element or a system-symbol is reached. For convenience the basic-elements are grouped together and the statements in which they are defined are listed below:

i-d (integer-denotation)	173	text	182
constant	175	non-camac-text	184
symbolic-constant	176	terminator	183
integer	177	adop (operator)	192
variable	178	mulop (operator)	193
identifier	179	non-syntax-character	194

The system-symbols each consist of a string of characters and are listed in statements 200 to 233. System-symbols may be divided into two classes, special-system-symbols (see 201–209) which may be equivalenced and fixed symbols (see 210–233) which may not be equivalenced.

A definition of non-camac-text is as follows.

Non-camac-text is any string of characters excluding the symbol CAMACSEGMENT and any special-directive (see 202) or any symbol equivalenced to a special-directive.

SYNTAX

```
 1. program ::= [non-camac-text] {camac-segment [non-camac-text] }...
 2.  camac-segment ::= segment-header [text] segment-body [text] segment-terminator
 3.   segment-header ::= CAMACSEGMENT segment-name terminator
 4.    segment-name ::= identifier
 5.   segment-body  ::= main-part [text] [{task-part [text] }...]
 6.    main-part ::= {[text] definition-section}... [text] [action-section]
 7.     definition-section ::= c-eqv-section | c-name-section | c-dmd-section | c-dcl-section |
                                c-ref-section
 8.      c-eqv-section ::= c-eqv-section-header c-eqv-statement ...
 9.       c-eqv-section-header ::= CEQV terminator
10.       c-eqv-statement ::= c-eqv-statement-body terminator
11.        c-eqv-statement-body ::= c-eqv-s-b1 | c-eqv-s-b2
12.         c-eqv-s-b1 ::= {non-syntax-character | identifier}EQV special-system-symbol
13.          non-syntax-character see 194
14.          special-system-symbol see 201
15.         c-eqv-s-b2 ::= symbolic-constant = {expression | integer}
16.          expression ::= term [{adop term}... ]
17.           term ::= primary [{mulop primary}... ]
18.            primary ::= constant | (constant {adop constant}...)
19.      c-name-section ::= c-name-section-header c-name-statement ...
```

```
20.     c-name-section-header ::= CNAME terminator
21.     c-name-statement ::= c-name-statement-body terminator
22.       c-name-statement-body ::= c-name-s-b1 | c-name-s-b2
23.         c-name-s-b1 ::= c-name [(size)] = address-set [G1 | G2] [mode]
24.           c-name ::= identifier
25.           size ::= 1 : constant
26.           address-set ::= address-value [ { , address-value}...]
27.             address-value ::= address-component ...
28.               address-component ::= {c-name [(address-list)] } | {c-type (address-list)}
29.                 c-name see 24
30.                 c-type ::= B|C|N|A|I
31.                 address-list ::= address-element [ { , address-element}...]
32.                   address-element ::= constant [ :constant [ :constant] ]
33.           mode ::= P|Q|R|S
34.         c-name-s-b2 ::= c-name [(size)] = EXT (c-type) [G1 | G2] [mode]
35.           c-name see 24
36.           size    see 25
37.           c-type  see 30
38.           mode    see 33
39.   c-dmd-section ::= c-dmd-section-header c-dmd-statement ...
40.     c-dmd-section-header ::= CDMD terminator
41.     c-dmd-statement ::= c-dmd-statement-body terminator
42.       c-dmd-statement-body ::= c-dmd-s-b1 | c-dmd-s-b2
43.         c-dmd-s-b1 ::= demand-name = c-name [GL integer]
44.           demand-name ::= identifier
45.           c-name see 24
46.         c-dmd-s-b2 ::= demand-name = EXT (c-type)
47.           demand-name see 44
48.           c-type see 30
49.   c-dcl-section ::= c-dcl-section-header c-dcl-statement ...
50.     c-dcl-section-header ::= CDCL terminator
51.     c-dcl-statement ::= c-dcl-statement-body terminator
52.       c-dcl-statement-body ::= c-dcl-s-b1 | c-dcl-s-b2 | c-dcl-s-b3 | c-dcl-s-b4
53.         c-dcl-s-b1 ::= [d-type] d-item [ { , d-item}...]
54.           d-type ::= COMPUTERLENGTH | CAMACLENGTH
55.           d-item ::= v-item | a-item | l-item
56.             v-item ::= variable-name
57.               variable-name ::= variable
58.             a-item ::= array-name (size)
59.               array-name ::= variable
60.               size see 25
61.             l-item ::= list-name (size) L
62.               list-name ::= variable
63.               size see 25
64.         c-dcl-s-b2 ::= p-type p-item [ { , p-item}...]
65.           p-type ::= CAMACADDRESS
66.           p-item ::= p-name [(size)]
67.             p-name ::= variable
```

```
 68.            size see 25
 69.          c-dcl-s-b3 ::= p-name [(range)] = c-name [(range)]
 70.           p-name see 67
 71.           c-name see 24
 72.           range ::= constant [ :constant]
 73.          c-dcl-s-b4 ::= EXT e-item [{ , e-item}...]
 74.           e-item ::= v-item | a-item | l-item | p-item P
 75.            v-item see 56
 76.            a-item see 58
 77.            l-item see 61
 78.            p-item see 66
 79.      c-ref-section ::= c-ref-section-header c-ref-statement ...
 80.       c-ref-section-header ::= CREF terminator
 81.       c-ref-statement ::= c-ref-statement-body terminator
 82.        c-ref-statement-body ::= c-ref-s-b1 | c-ref-s-b2 | c-ref-s-b3 | c-ref-s-b4 | c-ref-s-b5 |
                                     c-ref-s-b6 | c-ref-s-b7
 83.          c-ref-s-b1 ::= NAME c-name [{ , c-name}...]
 84.          c-ref-s-b2 ::= DEMAND demand-name [{ , demand-name}...]
 85.          c-ref-s-b3 ::= TASK task-name [{ , task-name}...]
 86.          c-ref-s-b4 ::= VARIABLE variable-name [{ , variable-name}...]
 87.          c-ref-s-b5 ::= ARRAY array-name [{ , array-name}...]
 88.          c-ref-s-b6 ::= LIST list-name [{ , list-name}...]
 89.          c-ref-s-b7 ::= ref-item [{ , ref-item }...]
 90.           ref-item ::= c-name | demand-name | task-name | variable-name | array-name |
                            list-name
 91.            c-name          see 24
 92.            demand-name see 44
 93.            task-name ::= identifier
 94.            variable-name  see 57
 95.            array-name      see 59
 96.            list name        see 62
 97.    action-section ::= action-section-header [text] {action-statement [text] }...
 98.     action-section-header ::= CACT terminator
 99.     action-statement ::= [label] {executive-statement | {[repeat-spec] {transfer-statement |
                                 control-statement}} | branch-statement}
100.      label ::= label-name :
101.       label-name ::= {letter : digit}...
102.      executive-statement ::= executive-statement-body terminator
103.       executive-statement-body ::= e-s-b1 | e-s-b2 | e-s-b3 | e-s-b4 | e-s-b5
104.        e-s-b1 ::= TERMINATE | QUIT
105.        e-s-b2 ::= {INITIATE | AWAIT | TERMINATE | DONOW}(task-name)
106.         task-name        see 93
107.        e-s-b3 ::= ACTIVATE (task-name, demand-name)
108.         task-name        see 93
109.         demand-name   see 44
110.        e-s-b4 ::= ALLOCATE demand-name PRIORITY i-d
111.         demand-name   see 44
112.        e-s-b5 ::= LINK (demand-name, label-name)
```

```
113.          demand-name   see 44
114.          label-name    see 101
115.      repeat-spec ::= REPEAT (i-d)
116.      transfer-statement ::= transfer-statement-body terminator
117.       transfer-statement-body ::= t-s-b1 | t-s-b2 | t-s-b3 | t-s-b4 | t-s-b5
118.        t-s-b1 ::= transfer-action-1 external-reference internal-reference
119.          transfer-action-1 ::= {F(n1)} | READ | READCLR | READCOMP | READLAM |
                                   READSTAT | MOVE
120.           n1 ::= 0|1|2|3|4|5|6|7|8|27
121.          external-reference ::= { c-name [(range)] } | {p-name [(array-spec)] } |
                                      direct-reference | demand-name
122.           c-name    see 24
123.           range     see 72
124.           p-name    see 67
125.           array-spec ::= i-d [ :i-d]
126.           direct-reference ::= {[B(i-d)] C(i-d) N(i-d) A(i-d) [I(i-d)] [G1 | G2] } | GL
127.          internal-reference ::= variable-name | {array-name (array-spec)} |
                                     {list-name (array-spec)}
128.           variable-name  see 57
129.           array-name     see 59
130.           list-name      see 62
131.           array-spec     see 125
132.           demand-name  see 44
133.        t-s-b2 ::= transfer-action-2 {number | internal-reference}external-reference
134.          transfer-action-2 ::= {F(n2)} | WRITE | SETSEL | CLEARSEL | MOVE
135.           n2 ::= 16|17|18|19|20|21|22|23
136.          number ::= binary | octal | hexadecimal | constant
137.           binary ::= BIN'{0 | 1}...'
138.           octal ::= OCT'{0|1|2|3|4|5|6|7}...'
139.           hexadecimal ::= HEX'{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}...'
140.          internal-reference  see 127
141.          external-reference see 121
142.        t-s-b3 ::= {TRANSFER | MOVE}source destination
143.          source ::= external-reference  see 121
144.          destination ::= external-reference  see 121
145.        t-s-b4 ::= {READ | MOVE}external-reference list-name [EXIT] label-name
146.        t-s-b5 ::= {WRITE | MOVE}list-name external-reference [EXIT] label-name
147.          external-reference see 121
148.          list-name          see 62
149.          label-name         see 101
150.      control-statement ::= control-statement-body terminator
151.       control-statement-body ::= c-s-b1 | c-s-b2
152.        c-s-b1 ::= control-action-1 external-reference
153.          control-action-1 ::= {F(n3)} | CLEAR | CLEARLAM | SET | ENABLE |
                                  DISABLE | EXECUTE
154.           n3 ::= 9|10|11|12|13|14|15|18|19|20|21|22|23|24|25|26|28|29|30|31
155.          external-reference  see 121
156.        c-s-b2 ::= control-action-2 external-reference
```

```
157.          control-action-2 ::= INITIALISE | SETINHIBIT | CLEARINHIBIT |
                                   ENABLEINT | DISABLEINT | CLEARSYS
158.          external-reference see 121
159.       branch-statement ::= branch-statement-body terminator
160.        branch-statement-body ::= [condition flag] GOTO label-name
161.         condition ::= IF | IFNOT
162.         flag ::= {camac-flag external-reference} | system-flag
163.          camac-flag ::= LAM | STATUS
164.          external-reference see 121
165.          system-flag ::= Q | {S(integer)}
166.         label-name  see 101
167.    task-part ::= task-header [text] action-section [text] task-terminator
168.     task-header ::= CAMACTASK task-name terminator
169.      task-name     see 93
170.     action-section see 97
171.     task-terminator ::= TASKEND
172.   segment-terminator ::= CAMACSEGMENTEND
173. i-d ::= integer-denotation
174.  integer-denotation ::= constant | variable
175.   constant ::= symbolic-constant | integer
176.    symbolic-constant ::= identifier  see 179
177.    integer ::= digit ...  see 189
178.   variable ::= identifier
179.    identifier ::= letter [alphanumeric-character ...]
180.     letter  see 188
181.     alphanumeric-character  see 187
182. text ::= ENDCAMAC non-camac-text BEGINCAMAC
183.  terminator ::= [NOTE non-camac-text] ENDSTATEMENT
184.   non-camac-text ::= character ...
185.    character ::= syntax-character | non-syntax-character
186.     syntax-character ::= alphanumeric-character | delimiter
187.      alphanumeric-character ::= letter | digit
188.       letter ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
189.       digit ::= 0|1|2|3|4|5|6|7|8|9
190.      delimiter ::= ,|:|(|)|   |=|'|operator
191.       operator ::= adop | mulop
192.        adop ::= + | -
193.        mulop ::= * | /
194.     non-syntax-character ::= !|"|#|$|%|&|.|;|<|>|?|@|[|\|]|↑|—
```

SYSTEM SYMBOLS

200. system-symbols ::= special-system-symbols | fixed-symbols
201. special-system-symbols ::= special-directive | function | camac-flag | system-flag | radix
202. special-directive ::= BEGINCAMAC | ENDCAMAC | NOTE | ENDSTATEMENT | CAMACTASK | TASKEND | CAMACSEGMENTEND
203. function ::= F{n1 | n2 | n3}
204. n1 ::= 0|1|2|3|4|5|6|7|8|27
205. n2 ::= 16|17|18|19|20|21|22|23
206. n3 ::= 9|10|11|12|13|14|15|18|19|20|21|22|23|24|25|26|28|29|30|31
207. camac-flag ::= LAM | STATUS
208. system-flag ::= Q | {S(integer)}
209. radix ::= BIN | OCT | HEX
210. fixed-symbols ::= directive | software-symbol | camac-symbol | mnemonic
211. directive ::= CAMACSEGMENT | CEQV | CNAME | CDMD | CDCL | CREF | CACT | EXT
212. software-symbol ::= EQV | L | P | REPEAT | EXIT | GOTO | NAME | DEMAND | TASK | VARIABLE | ARRAY | LIST | d-type | p-type | condition
213. d-type ::= COMPUTERLENGTH | CAMACLENGTH
214. p-type ::= CAMACADDRESS
215. condition ::= IF | IFNOT
216. camac-symbol ::= G1 | G2 | GL | c-type | mode
217. c-type ::= B|C|N|A|I
218. mode ::= P|Q|R|S
219. mnemonic ::= executive-action | transfer-action | control-action
220. executive-action ::= e-a1 | e-a2 | e-a3 | e-a4
221. e-a1 ::= TERMINATE | QUIT
222. e-a2 ::= INITIATE | AWAIT | TERMINATE | DONOW
223. e-a3 ::= ACTIVATE
224. e-a4 ::= ALLOCATE | PRIORITY | LINK | INITIALISE
225. transfer-action ::= t-a1 | t-a2 | t-a3 | t-a4 | t-a5
226. t-a1 ::= READ | READCLR | READCOMP | READLAM | READSTAT | MOVE
227. t-a2 ::= WRITE | SETSEL | CLEARSEL | MOVE
228. t-a3 ::= TRANSFER | MOVE
229. t-a4 ::= READ | MOVE
230. t-a5 ::= WRITE | MOVE
231. control-action ::= c-a1 | c-a2
232. c-a1 ::= CLEAR | CLEARLAM | SET | ENABLE | DISABLE | EXECUTE
233. c-a2 ::= INITIALISE | SETINHIBIT | CLEARINHIBIT | ENABLEINT | DISABLEINT | CLEARSYS

Appendix 2

SAMPLE PROGRAMS

Two examples are given of the use of the CAMAC language. These have been chosen solely to illustrate the form of statements used. The applications are approximations to real jobs but the host languages used are entirely imaginary. The hardware* is common to both examples and permits two tests (TESTA and TESTB) to be performed. The duration of the tests is controlled by counters fed with timing pulses or other event signals. In the first example a purely sequential procedure is followed. TESTA is performed and the results are printed out, followed by TESTB and its results. The cycle then recommences and continues until halted by a manual STOPCONTROL. In this example CAMAC statements are embedded in the host language program and the sequence is controlled by interrupts.

In the second example the two tests proceed in parallel and make use of a common print routine whenever the results are available. The host language or operating system is assumed to have specified multi-task scheduling calls. In this example the CAMAC language statements are separate from host language statements.

Comments that naturally belong to the program are in upper case, while those which describe or explain the language appear in normal typescript.

The program is headed with its name in the host language and this is followed by the CAMAC language segment header. The segment opens with the equivalence section of the declarations.

```
CAMAC-EXAMPLE-ONE;
CAMACSEGMENT             EXAMPLEONE        ENDSTATEMENT
NOTE EQUIVALENCE         DECLARATIONS      ENDSTATEMENT
     CEQV          ENDSTATEMENT
                   .   EQV    ENDSTATEMENT ENDSTATEMENT
                   ?   EQV    BEGINCAMAC.
                   "   EQV    ENDCAMAC.
                   \   EQV    NOTE.
```

The period (.) cannot be used as equivalent to the terminator ENDSTATEMENT in the statement that makes the equivalence. These equivalences reduce the effort required in program writing and may be chosen in relation to the host language or to improve readability of the program. In these examples the period (.) is used as a statement terminator in comparison with semi-colon (;) assumed for the host language. This enables the reader to distinguish between statements in the defined CAMAC language and the fictitious host language.

```
NOTE  THE CAMAC ENABLE AND DISABLE FUNCTIONS MAY BE USED TO CONTROL
      DATA-TAKING OR TO CONTROL THE GENERATION OF DEMANDS; THE
      FOLLOWING EQUIVALENCES ARE MADE SO THAT THE PROGRAM IS EASIER TO
      UNDERSTAND.

      TAKEDATA   EQV    F(26).
      STOPDATA   EQV    F(24).
      ENABLAM    EQV    F(26).
      DISABLAM   EQV    F(24).
      RESETLAM   EQV    F(10).
```

This completes the equivalencing and may be immediately followed by the hardware declarations without having to use BEGINCAMAC or its equivalent.

* See Figure 1, inside the back cover.

\ CAMAC NAMING SECTION.

```
CNAME.    APPARATUS              =   B(1).
          CONTROLCRATE           =   APPARATUS C(1).
          Z                      =   CONTROLCRATE.
```

These declarations assign the names APPARATUS to Branch 1 and CONTROLCRATE to Crate 1 on that Branch. This method of naming permits a simple modification to the program if either the Branch or the Crate are reassigned. The name Z is used subsequently as a shortened form for CONTROLCRATE.

```
          CLOCK                  =   Z N(1).
          SECONDS                =   CLOCK A(0).
          MINUTES                =   CLOCK A(1).
          HOURS                  =   CLOCK A(2).
          TIME (1:3)             =   HOURS, MINUTES, SECONDS.
```

This continuation of the naming technique permits modifications of the hardware address of CLOCK without disturbing the definition of the hardware array TIME.

```
          COUNTER                =   Z N(2).
          TALLY                  =   COUNTER A(0).
          TALLYCHECK             =   COUNTER A(1).
```

NOTE TALLY IS THE ADDRESS ASSOCIATED WITH DATATAKING AND TALLYCHECK CONTROLS THE DEMAND SIGNAL.

NOTE IN THE SECOND EXAMPLE THERE ARE TWO COUNTERS SO THE FOLLOWING DECLARATIONS ARE MADE.

```
          TALLYA                 =   TALLY.
          TALLYCHECKA            =   TALLYCHECK.

          COUNTERB               =   Z N(3).
          TALLYB                 =   COUNTERB A(0).
          TALLYCHECKB            =   COUNTER I(1) G2.
```

It will be noted that 'TALLYCHECK' is accessed via subaddress A(1) and that 'TALLYCHECKB' is accessed by the bit position I(1) of a Group Two Register. Having made these declarations the programmer may treat the two demand sources identically in the subsequent program.

NOTE AS AN AID TO PROCESSING, THE RESULTS FROM EACH TEST ARE FED AS BINARY NUMBERS (24 BIT MAXIMUM) TO A HARDWARE CONVERTER; A STRING OF 8 ASCII DECIMAL CHARACTERS (WITH LEADING ZEROS REPLACED BY SPACES) MAY THEN BE READ FROM THE CONVERTER; INTERNAL STATUS (Q=1) IS GENERATED WHEN THE CONVERSION IS COMPLETE.

The period could not be used within the above as it would have terminated the note.

```
          CONVERTER              =   Z N(4) A(0).
          BINARYINPUT            =   CONVERTER.
          ASCIIOUTPUT            =   CONVERTER S.
```

The access qualifier (S) indicates that the output of the converter operates in the Stop Mode of Q; that is Q=1 indicates that a valid word has been read, while Q=0 indicates that end-of-block has been passed.

```
                    PRINTER                    =    Z N(6) A(0) R.
```

NOTE THIS MODULE INTERFACES A TELETYPEWRITER TO CAMAC: AFTER TYPING A CHARACTER IT RAISES A LAM DEMAND TO ASK FOR THE NEXT CHARACTER (IF LAM IS ENABLED).

The access qualifier (R) indicates that the module operates in Repeat Mode of Q, giving Q=0 if it is unable to accept data.

```
                    STOPCONTROL                =    Z N(7) A(0).
```

NOTE THIS IS A MANUAL CONTROL TO STOP BOTH TESTS.

```
                    CHARACTERISTICS(1:4) =    Z N(1:4) A(15) G2.
```

The first four modules have bit patterns describing their characteristics at the preferred subaddress A15 and are accessed by Group 2 functions as indicated by G2.

```
                    DATACRATEX                 =    APPARATUS C(2).
                    DX                         =    DATACRATEX.
                    DATACRATEY                 =    APPARATUS C(3).
                    DY                         =    DATACRATEY.
```

The method of hierarchic naming is continued for two further crates. Again this eases the problems of relocation of the hardware.

The following is a method of generating arrays of uniformly distributed hardware which avoids errors in calculation and eases the problem of subsequent modification.

```
    CEQV.           M                          =    10 \ NUMBER OF MODULES.
                    W                          =     2 \ MODULE WIDTH.
                    F                          =     W \ FIRST ADDRESS.
                    L                          =    (M-1) * W + F. \ LAST ADDRESS.

    CNAME.          MEANTEMPS(1:M)             =    DX N(F:L:W) A(0) P.
```

The array MEANTEMPS has 10 elements in Crate 2 starting at N=2 and progressing to N=20 by increments of 2. The required registers are located at subaddress A(0) in each module. Because the elements all require the same subaddress this array may be accessed in parallel for appropriate functions (e.g. CLEAR). This is indicated by the access function (P). The same array could also be accessed in the Address Scan mode of Q for other functions (e.g. READ). In order to be unambiguous in the access mode to be used, a distinct name is given as follows:

```
                    SEQUTEMPS (1:M)            =    MEANTEMPS(1:M) Q.
```

It will be noted that when MEANTEMPS is used on the right-hand side it does not imply its previously defined access qualifier.

```
PEAKPRESSA(1:4)        =   DY  N(1:2) A(0:1).
```

This declaration expands to give the following sequence of addresses.

```
NOTE                       B(1)   C(3)   N(1)   A(0)
                           B(1)   C(3)   N(1)   A(1)
                           B(1)   C(3)   N(2)   A(0)
                           B(1)   C(3)   N(2)   A(1).

PEAKPRESSB(1:4)        =   DY  A(0:1)  N(3:4).
```

This declaration expands to give a different sequence of addresses.

```
NOTE                       B(1)   C(3)   N(3)   A(0)
                           B(1)   C(3)   N(4)   A(0)
                           B(1)   C(3)   N(3)   A(1)
                           B(1)   C(3)   N(4)   A(1).
```

This organisation of the CAMAC address fields permits an array to be built up to give the most suitable sequence for subsequent processing.

```
FLOWA(1:5)             =   DY  N(5,19,7,9,11)   A(0).

FLOWB(1:5)             =   DY  N(13:17:2)    A(0),
                           Z    N(10)     A(0),
                           B(1)    C(3)    N(21)  A(0).
```

In practice the form B(1) C(3) N(21) A(0) would not be used in conjunction with the technique of hierarchical naming since it does not give automatic address modification when, for example, the physical address of the crate is changed.

```
CRATES(1:3)            =   Z, DX, DY.
BRANCHDEMAND(1:3)      =   CRATES(1:3) N(30) A(10) P.
```

This declaration makes use of the virtual address in Crate Controller A which controls the BD signal. The parallel access (P) permits the BD signals to be enabled and disabled at source in the three crates at the same time.

This completes the naming of the units but these may now be compounded into further arrays if required.

```
INSTRUMENTSA(1:14)     =   MEANTEMPS(1:5),
                           PEAKPRESSA(1:4),
                           FLOWA(1:5).

INSTRUMENTSB(1:14)     =   MEANTEMPS(6:10),
                           PEAKPRESSB(1:4),
                           FLOWB(1:5).

TESTA(1:15)            =   INSTRUMENTSA(1:14), TALLYA.

TESTB(1:15)            =   INSTRUMENTSB(1:14), TALLYB.
```

The arrays TESTA and TESTB could have been built up initially from the individual units or via a mapping technique, as shown in the second example.

The method shown here is chosen for ease of reconfiguration and is virtually self-documenting.

```
\ DEMAND DECLARATIONS.
```

The Demand Declarations permit names to be given to the demands for subsequent use, and make a permanent connection between these demands and the hardware addresses at which they are controlled. They also allow the specific bit (in the Graded-L pattern) associated with the demand to be defined.

```
    CDMD.       DONE            =   TALLYCHECK GL20.
                ADONE           =   TALLYCHECKA GL20.
                BDONE           =   TALLYCHECKB GL19.
                NEXT            =   PRINTER GL16.

\ SOFTWARE DECLARATIONS.
    CDCL.
                CAMACLENGTH
                    DATA(1:17) L.
```

The access qualifier L identifies ‘DATA’ as an array which may be accessed as a list.

```
                COMPUTERLENGTH
                    LETTER,
                    TEMPORARY,
                    LIMIT,
                    ENDJOB.
```

NOTE COMPUTER LENGTH IN THIS APPLICATION IS 16 BITS.

These declarations define all variables used in CAMAC language statements. Because TEMPORARY is only used by CAMAC it need not conform to the host language conventions.

```
                CAMACADDRESS
                    UNITS(1:15),
                    UNITSA(1:15),
                    UNITSB(1:15).
```

In the CAMAC statements indirect reference is made to hardware addresses via core locations. This permits run-time modification to these addresses—at least to the extent of interchanging addresses. The above declarations reserve the locations required to hold the CAMAC addresses in some appropriate form.

```
                UNITSA(1:14)    =   INSTRUMENTSA(1:14).
                UNITSA(15)      =   TALLY.
                UNITSB(1:14)    =   INSTRUMENTSB(1:14).
                UNITSB(15)      =   TALLY.
```

These declarations specify the previously defined BCNA addresses which are to be listed in sequence in the previously defined address space.

NOTE THIS COMPLETES THE CAMAC DECLARATIONS SECTION.

The program continues with the host language declarations including any required to establish links with CAMAC statements; for example it may be necessary to repeat the CAMAC software declarations in the form of GLOBAL declarations.

This is followed by the action statements of the program. The order of presentation here is not necessarily that which would be followed normally.

CACT.

NOTE IN THE FIRST EXAMPLE TESTA IS STARTED; WHEN THE TALLY RUNS OUT A DEMAND 'DONE' CALLS AN INTERRUPT ROUTINE WHICH STOPS TESTA, READS THE RESULTS AND PRINTS THEM; TESTB IS THEN STARTED AND SO ON."

```
EXAMPLE1:  ENDJOB    =    FALSE;
           ?INITIALISE APPARATUS"
```

This is an isolated CAMAC statement embedded in host language. It is bracketed by the equivalents of BEGINCAMAC(?) and ENDCAMAC("). It disables all demand sources, clears data registers and applies an overall inhibit to the equipment.

```
LF(1)                     =    138;
```

The above is assumed to be a host language statement which loads the character code value for Line Feed into the location LF(1).

```
?STOPDATA     UNITSA(1:15).
 STOPDATA     UNITSB(1:15).
```

These statements make indirect references to hardware addresses via the core locations assigned in the CAMACADDRESS declarations.

```
CLEARINHIBIT    CRATES(1:3).
ENABLE    BRANCHDEMAND(1:3)"
CALL    SETUPA;
CALL    START;
```

The program now continues with other activities, irrelevant to this example, which will be interrupted.

The subroutines called are as follows:

```
SUBROUTINE   SETUPA;
    FLAGA              =    FALSE;
    UNITS(1:15)        =    UNITSA(1:15);
    LIMIT              =    1000;
    LETTER             =    'A';
    RETURN;
    END;
```

In the above host language subroutine the statements have the following meanings. The Boolean variable FLAGA is set to false, the array UNITS is loaded with the values held in UNITSA, LIMIT is loaded with the decimal value 1000 and LETTER is loaded with the ASCII value for letter A. RETURN is the run time exit from the subroutine and END defines the lexical termination of this section.

```
          SUBROUTINE    START;
               EXTERNAL    READANDPRINT;
              ?LINK (DONE, READANDPRINT).
               CLEAR UNITS(1:15).
               WRITE LIMIT TALLY.
               ENABLAM TALLYCHECK.
               TAKEDATA UNITS(1:15)."
               RETURN;
               END;
```

The subroutine START uses CAMAC language action statements to operate on the hardware in the manner defined by the previous SETUPA subroutine. TAKEDATA, previously equivalenced to ENABLE, starts the hardware operations. When the Tally runs out it generates the demand 'DONE' which causes 'READANDPRINT' to run.

```
? NOTE    INTERRUPT ROUTINE ACCESSED BY DONE."

     READANDPRINT:         EXTERNAL CHARACTER;
                          ?STOPDATA UNITS(1:15).
                           RESETLAM TALLYCHECK.
                           READ TIME(1:3) DATA(1:3).
                           READ UNITS(1:14) DATA(4:17).
                           LINK (NEXT, CHARACTER).
                           WRITE DATA BINARYINPUT EXIT ERROR.
```

This statement accesses DATA as a list structure. Since this is the first access it will transfer the first word from the list and should not perform the branching part of the statement.

```
                           DISABLAM PRINTER.
                           WRITE OCT '215' PRINTER.          \CARRIAGE RETURN.
                           REPEAT(3) WRITE OCT '212' PRINTER.
```

This statement causes the octal value 212 = Line Feed to be written to the module controlling the printer three times in succession. However the printer cannot accept a new write command until it has dealt with the previous one and will return Q=0 if the operation is unsuccessful (R mode of Q). The operation will therefore be repeated (possibly several thousand times) until three line feeds are achieved.

```
                           ENABLAM PRINTER.
                           WRITE LETTER PRINTER.

NOTE  AFTER PRINTING 'LETTER' THE PRINTER RAISES DEMAND 'NEXT'."

                           RESTORE;
     ERROR:                error recovery action;
                           END;

? NOTE    INTERRUPT ROUTINE ACCESSED BY NEXT.

     CHARACTER:            RESETLAM PRINTER.
     READY:                READ ASCIIOUTPUT TEMPORARY \S MODE OF Q.
                           IF Q GOTO TT.
```

The 'S mode of Q' gives Q=1 if a valid transfer was performed by 'ASCIIOUTPUT' and Q=0 if the string of digits in the converter have all been read.

```
                        WRITE DATA BINARYINPUT     EXIT     BUFFERMT.
   WAIT:                IFNOT STATUS CONVERTER GOTO WAIT.
                        GOTO READY.
   TT:                  WRITE TEMPORARY PRINTER."
                        RESTORE;
  ?BUFFERMT:            LINK (NEXT, LASTNUMBER).
   WAIT2:               IFNOT STATUS CONVERTER GOTO WAIT2.
   LASTNUMBER:          RESTLAM PRINTER.
                        READ ASCIIOUTPUT TEMPORARY.
                        IF Q GOTO TT.
                        READ STOPCONTROL ENDJOB"
                        IF ENDJOB GOTO JOBDONE;
                        IF FLAGA GOTO ANEXT;
                        CALL SETUPB;
                        GOTO SEQUENCE;
   ANEXT:               CALL SETUPA;
   SEQUENCE:            CALL START;
   JOBDONE:             RESTORE;

   SUBROUTINE           SETUPB;
                        FLAGA         =    TRUE;
                        UNITS(1:15)   =    UNITSB(1:15);
                        LIMIT         =    2000;
                        LETTER        =    'B';
                        RETURN;
                        END;
? NOTE   THIS COMPLETES THE FIRST EXAMPLE.

CAMACSEGMENTEND
```

In the second example the hardware of TESTA and TESTB run in parallel and may call for 'PRINTOUT' in any order. The host language or operating system is assumed to have the real-time–multi-task scheduling calls proposed in section 3.4.6. In this example the CAMAC action statements are kept separate from host language statements and are contained within CAMACTASKS.

```
CAMAC-EXAMPLE-TWO;

CAMACSEGMENT    EXAMPLETWO    ENDSTATEMENT

   CEQV         ENDSTATEMENT
                .   EQV     ENDSTATEMENT ENDSTATEMENT
                ?   EQV     BEGINCAMAC.
                "   EQV     ENDCAMAC.
                \   EQV     NOTE.

   TAKEDATA         EQV     F(26).
   STOPDATA         EQV     F(24).
   ENABLAM          EQV     F(26).
   DISABLAM         EQV     F(24).
   RESETLAM         EQV     F(10).
```

NOTE BY WAY OF EXAMPLE THE CAMACTASKS ASSOCIATED WITH THE PRINTOUT ARE INCORPORATED IN A SEPARATE SEGMENT; THE RELEVANT DECLARATIONS WILL THEREFORE BE MADE IN THAT SEGMENT.

The CAMAC apparatus used in this example is the same as that for example one. An alternative method of naming the hardware is adopted here, solely as a demonstration of the techniques available to the programmer. The apparatus is first mapped onto an arbitrary array of hardware entities; names are then assigned, as required, to elements of the array. Either method of naming could be used in both examples.

```
\ CAMAC NAMING SECTION.

    CNAME.

              CRATES(1:3)             =   B(1) C(1:3).

              MODULES(1:15)           =   N(1) A(0:2),
                                          N(2) A(0:1),
                                          N(3) A(2),
                                          N(3,4,6,7,10) A(0),
                                          N(1:4) A(15).

              K(1:15)                 =   CRATES(1) MODULES(1:15).

              TIME(1:3)               =   K(3,2,1).
              TALLYA                  =   K(4).
              TALLYCHECKA             =   K(5).
              TALLYCHECKB             =   C(1) N(3) I(1) G2.
              TALLYB                  =   K(7).
              BINARYINPUT             =   K(8).
              ASCIIOUTPUT             =   K(8) S.
              PRINTER                 =   K(9) R.
              STOPCONTROL             =   K(10).
              CHARACTERISTICS(1:4)    =   K(12:15) G2.
              CONVERTER               =   K(8).

              L(1:10)                 =   CRATES(2) N(2:20:2) A(0) P.

              M(1:17)                 =   CRATES(3) N(1:4) A(0:1),
                                          CRATES(3) N(5:21:2) A(0).

              TESTA(1:15)             =   L(1:5), M(1:4,9,16,10:12), K(4).

              TESTB(1:15)             =   L(6:10), M(5,7,6,8,13:15),
                                          K(11), M(17), K(7).

\ DEMAND DECLARATIONS.

    CDMD.
              ADONE                   =   TALLYCHECKA GL20.
              BDONE                   =   TALLYCHECKB GL19.
              NEXT                    =   PRINTER GL16.
```

```
\ SOFTWARE DECLARATIONS.

    CDCL.           CAMACLENGTH
                        DATA(1:17) L,
                        DATAA(1:17),
                        DATAB(1:17).

                    COMPUTERLENGTH
                        LETTER,
                        LIMIT,
                        LF(1:3) L,
                        ENDJOB,
                        TEMPORARY.

\ GLOBAL DECLARATIONS.

    CREF.           NAME PRINTER, BINARYINPUT, ASCIIOUTPUT, CONVERTER.

                    TASK  FIRSTHINGS,
                          STARTA, STOPA, READA,
                          STARTB, STOPB, READB.

                    VARIABLE    LETTER, LIMIT, ENDJOB, TEMPORARY.

                    ARRAY       DATAA, DATAB.

                    LIST        DATA, LF.

NOTE  THIS COMPLETES THE CAMAC DECLARATIONS SECTION.

ENDCAMAC.
```

The program continues with the host language declarations including any required to establish links with CAMAC statements; for example it may be necessary to repeat the CAMAC software declarations in the form of GLOBAL declarations.

This is followed by the action statements of the program. The order of presentation here is not necessarily that which would be followed normally.

```
            EXAMPLE2:       CAMACTASK FIRSTHINGS;
                            DONOW (FIRSTHINGS);
                            ENDJOB    =    FALSE;
                            LF(1) = 138; LF(2) = 138; LF(3) = 138;
                            INITIATE (SEQUENCEA);
                            INITIATE (SEQUENCEB);
                            AWAIT (SEQUENCEA);
                            AWAIT (SEQUENCEB);
                            STOP;
```

```
COMMENT    THIS GIVES THE OVERALL SCHEDULING: THE TASK FIRSTHINGS
           INITIALISES THE HARDWARE, THE LOGICAL VARIABLE ENDJOB (WHICH
           STOPS THE TESTS WHEN TRUE) IS SET TO FALSE, THE ARRAY LF(1:3) IS
           LOADED WITH LINEFEED CODES AND THEN SEQUENCEA AND SEQUENCEB
           ARE SCHEDULED: WHEN BOTH SEQUENCEA AND SEQUENCEB HAVE
           COMPLETED THE JOB IS DONE;

COMMENT    THE DETAILED SCHEDULING OF SEQUENCEA AND SEQUENCEB MAY NOW
           BE SET OUT INDEPENDENTLY;
```

The above COMMENTS are assumed to be in host language because this section of the program contains no CAMAC language statements.

```
           TASK SEQUENCEA;
                          CAMACTASK STOPA, STARTA, READA;
                          ACTIVATE (STOPA, ADONE);
                          DONOW (STARTA);
                          AWAIT (STOPA);
           LOOPA:         DONOW (READA);
                          IF ENDJOB GOTO ENDA;
                          ACTIVATE (STOPA, ADONE);
                          DONOW (STARTA);
                          FLAGA    =    TRUE;
                          DONOW (PRINTOUT);
                          AWAIT (STOPA);
                          GOTO LOOPA;
           ENDA:          FLAGA    =    TRUE;
                          DONOW (PRINTOUT);
                          TASKDONE;
                          END;

COMMENT    HAVING SCHEDULED STOPA TO RUN ON DEMAND ADONE, THE EQUIP-
           MENT OF TESTA IS STARTED BY STARTA. WHEN THE TALLY RUNS OUT
           ADONE IS GENERATED AND STOPA EXECUTED. WHEN STOPA COMPLETES
           READA COLLECTS THE DATA AND READS THE STOPCONTROL INTO
           ENDJOB. IF ENDJOB IS NOT TRUE TESTA IS RESTARTED AND PRINTOUT
           IS CALLED. SEQUENCEA WILL NOT CONTINUE UNTIL PRINTOUT HAS
           COMPLETED AND STOPA HAS AGAIN BEEN COMPLETED. IF ENDJOB IS
           TRUE THE CURRENT RESULTS ARE PRINTED AND THE TASK TERMINATES;

COMMENT    IN THIS EXAMPLE TESTA AND TESTB ARE SO SIMILAR THAT SEQUENCEA
           AND SEQUENCEB ARE VIRTUALLY IDENTICAL;

           TASK SEQUENCEB;
                          CAMACTASK STOPB, STARTB, READB;
                          ACTIVATE (STOPB, BDONE);
                          DONOW (STARTB);
                          AWAIT (STOPB);
           LOOPB:         DONOW (READB);
                          IF ENDJOB GOTO ENDB;
                          ACTIVATE (STOPB, BDONE);
                          DONOW (STARTB);
                          FLAGB    =    TRUE;
                          DONOW (PRINTOUT);
                          AWAIT (STOPB);
                          GOTO LOOPB;
```

```
                    ENDB:               FLAGB     =     TRUE;
                                        DONOW (PRINTOUT);
                                        TASKDONE;
                                        END;

                    TASK PRINTOUT;
                                        CAMACTASK LINEFEED, NEWLINE, CHARACTER,
                                        MARKER, LASTNUMBER, STARTLAST;
                                        IF FLAGA GOTO PRINTA;
                                        IF FLAGB GOTO PRINTB;
                                        GOTO ERROR;
                    PRINTA:             DATA(1:17)    =    DATAA(1:17);
                                        FLAGA         =    FALSE;
                                        LETTER        =    'A';
                                        GOTO PRINT;
                    PRINTB:             DATA(1:17)    =    DATAB(1:17);
                                        FLAGB         =    FALSE;
                                        LETTER        =    'B';
                    PRINT:              ACTIVATE (LINEFEED, NEXT);
                                        DONOW (NEWLINE);
                                        AWAIT (LINEFEED);
                                        ACTIVATE (CHARACTER, NEXT);
                                        DONOW (MARKER);
                                        AWAIT (CHARACTER);
                                        ACTIVATE (LASTNUMBER, NEXT);
                                        DONOW (STARTLAST);
                                        AWAIT (LASTNUMBER);
                                        TASKDONE;
                                        END;

COMMENT             THE PRINTOUT TASK IS DELIBERATELY NON RE-ENTRABLE SINCE THE
                    RECORD OF ONE TEST MUST BE SEPARATED FROM ANY OTHER. WHEN
                    ENTERED THE ROUTINE DETERMINES WHETHER SEQUENCEA OR
                    SEQUENCEB REQUIRED IT AND SETS UP THE ARRAY 'DATA(1:17)' AND
                    THE IDENTIFYING LETTER APPROPRIATELY. IT THEN GENERATES A
                    CARRIAGE RETURN ('NEWLINE') FOLLOWED BY LINEFEEDS IN RESPONSE
                    TO THE DEMAND 'NEXT' FROM THE PRINTER WHEN IT IS READY. AFTER
                    THREE LINEFEEDS THE IDENTIFIER IS PRINTED ('MARKER') AND THEN
                    THE RESULTS ('CHARACTER'). THE 'LASTNUMBER' REQUIRES SPECIAL
                    TREATMENT;
```

BEGINCAMAC
The host language section of the program is now complete and the following is written exclusively in the CAMAC language.

```
CACT.

CAMACTASK FIRSTHINGS.
                                        INITIALISE APPARATUS.
                                        STOPDATA TESTA(1:15).
                                        STOPDATA TESTB(1:15).
                                        CLEARINHIBIT CRATES(1:3).
                                        ENABLE CRATES(1:3).
                                        TERMINATE (FIRSTHINGS).
                                        TASKEND.
```

```
NOTE  'INITIALISE APPARATUS' DISABLES ALL DEMANDS, CLEARS REGISTERS AND
      APPLIES AN OVERALL INHIBIT; WHEN THE INDIVIDUAL UNITS HAVE BEEN
      BROUGHT UNDER CONTROL THE OVERALL INHIBITIONS MAY BE REMOVED.

CAMACTASK STARTA.           CLEAR TESTA(1:15).
                            WRITE 1000 TALLYA \ DECIMAL 1000.
                            ENABLAM TALLYCHECKA.
                            TAKEDATA TESTA(1:15)
                            TERMINATE (STARTA).
                            TASKEND.

NOTE  HAVING CLEARED THE DATA REGISTERS AND LOADED THE HARDWARE TALLY,
      THE DEMAND SOURCE IS ENABLED AND DATATAKING STARTED; WHEN THE
      TALLY RUNS OUT DEMAND 'ADONE' CALLS FOR TASK 'STOPA'.

CAMACTASK STOPA.            STOPDATA TESTA(1:15).
                            RESETLAM TALLYCHECKA.
                            TERMINATE (STOPA).
                            TASKEND.

NOTE  THE SCHEDULING ENSURES THAT TASK READA IS NOT ENTERED UNTIL ARRAY
      'DATAA' IS FREE.

CAMACTASK READA.            READ TIME(1:3) DATAA(1:3).
                            READ TESTA(1:14) DATAA(4:17).
                            READ STOPCONTROL ENDJOB.
                            TERMINATE (READA).
                            TASKEND.

NOTE  THE TASKS ASSOCIATED WITH SEQUENCEB ARE OF THE SAME FORM.

CAMACTASK STARTB.           CLEAR TESTB(1:15).
                            WRITE 2000 TALLYB \ DECIMAL 2000.
                            ENABLE BDONE.
                            TAKEDATA TESTB(1:15).
                            TERMINATE (STARTB).
                            TASKEND.

CAMACTASK STOPB.            STOPDATA TESTB(1:15).
                            DISABLE BDONE.
                            TERMINATE (STOPB).
                            TASKEND.

CAMACTASK READB.            READ TIME(1:3) DATAB(1:3).
                            READ TESTB(1:14) DATAB(4:17).
                            READ STOPCONTROL ENDJOB.
                            TERMINATE (READB).
                            TASKEND.

NOTE  FURTHER TASKS ARE CONTAINED IN THE NEXT SEGMENT.

CAMACSEGMENTEND
```

```
CAMACSEGMENT     CAMACPRINTOUT    ENDSTATEMENT

        CEQV ENDSTATEMENT
                       EQV ENDSTATEMENT ENDSTATEMENT

     CNAME.      PRINTER              =    EXT (A) R.
                 BINARYINPUT          =    EXT (A).
                 ASCIIOUTPUT          =    EXT (A) S.
                 CONVERTER            =    EXT (A).

     CDCL.       EXT LF(1:3) L, DATA(1:64) L, LETTER, TEMPORARY.

     CREF.       TASK  NEWLINE, LINEFEED, MARKER, CHARACTER, STARTLAST,
                       LASTNUMBER.

     CACT.

NOTE  THE FOLLOWING ARE THE TASKS ASSOCIATED WITH PRINTOUT.

CAMACTASK NEWLINE.         ENABLAM PRINTER.
                           WRITE OCT '215' PRINTER \ CARRIAGE RETURN.
                           TERMINATE (NEWLINE).
                           TASKEND.

CAMACTASK LINEFEED.        RESETLAM PRINTER.
                           WRITE LF PRINTER     EXIT    SHIFTED.
                           QUIT (LINEFEED).

SHIFTED:                   DISABLAM PRINTER.
                           TERMINATE (LINEFEED).
                           TASKEND.

NOTE  NEWLINE IS INITIATED FROM THE COMPUTER AND PRINTS CARRIAGE RETURN;
      THE RESULTING DEMAND 'NEXT' CALLS 'LINEFEED' AND PRINTS LINEFEED
      (CONTENTS OF LF(1)); SUBSEQUENT DEMANDS REPEAT THIS TASK AT LF(2)
      AND LF(3).

CAMACTASK MARKER.          RESETLAM PRINTER.
                           WRITE DATA BINARYINPUT      EXIT    ERROR.
                           WRITE LETTER PRINTER.
                           ENABLE PRINTER.
ERROR:                     TERMINATE (MARKER).
                           TASKEND.

NOTE  MARKER PRINTS THE IDENTIFIER A OR B; THE RESULTING DEMAND 'NEXT'
      CALLS 'CHARACTER' WHICH PRINTS THE CONVERTED BINARY NUMBERS UNTIL
      THE LAST ONE IS BROUGHT FROM CORE.

CAMACTASK CHARACTER.       RESETLAM PRINTER.
READY:                     READ ASCIIOUTPUT TEMPORARY.
                           IF Q GOTO TT.
                           WRITE DATA BINARYINPUT      EXIT    BUFFERMT.
                           REPEAT (50) IF STATUS CONVERTER GOTO READY.
                           GOTO ERROR 2.
```

```
TT:                          WRITE TEMPORARY PRINTER.
                             QUIT (CHARACTER).
BUFFERMT:                    IFNOT STATUS CONVERTER GOTO BUFFERMT.
ERROR 2:                     TERMINATE (CHARACTER).
                             TASKEND.

CAMACTASK STARTLAST.         MOVE ASCIIOUTPUT PRINTER.
```

The above statement has two external addresses, the source followed by the destination.

```
                             TERMINATE (STARTLAST).
                             TASKEND.

CAMACTASK LASTNUMBER.        RESETLAM PRINTER.
                             MOVE ASCIIOUTPUT TEMPORARY.
                             IFNOT Q GOTO ENDPRINT.
                             MOVE TEMPORARY PRINTER.
                             QUIT (LASTNUMBER).

ENDPRINT:                    DISABLAM PRINTER.
                             TERMINATE (LASTNUMBER).
                             TASKEND.

NOTE  'MOVE' HAS MEANING 'TRANSFER', 'READ' AND 'WRITE' IN THE ABOVE.

NOTE  'STARTLAST' TRANSFERS THE FIRST ASCII CHARACTER OF THE LAST BINARY
      NUMBER TO THE PRINTER; THE RESULTING DEMAND 'NEXT' CALLS
      'LASTNUMBER' AND PRINTS THE NEXT CHARACTER: SUBSEQUENT DEMANDS
      REPEAT THIS TASK UNTIL THE LAST CHARACTER HAS BEEN PRINTED.

NOTE  THIS COMPLETES THE SECOND EXAMPLE.

CAMACSEGMENTEND
```

Membership of the Software Working Groups

ESONE Software Working Group.

E. De Agostino, CNEN-CSN Cassaccia, Italy

W. Attwenger, SGAE Wien, Austria

Palle Christensen, AEC Risö, Denmark

W. K. Dawson, Dep. Phys. Univ. Alberta, Canada

P. Elzer, Phys. Inst. Uni Erlangen, Germany

H. Halling, KFA Jülich, Germany (Secretary)

J. Harneit, GfK Karlsruhe, Germany

I. N. Hooton, AERE Harwell, England (Chairman)

A. Katz, CEN Saclay, France

H. Klessmann, HMI Berlin, Germany

A. Lewis, AERE Harwell, England

J. Lukacs, MTA-KFKI Budapest, Hungary

B. E. F. Macefield, Nucl. Phys. Univ. Oxford, England

H. J. Metzdorf, Euratom Ispra, Italy

H. Meyer, BCMN Geel, Belgium

R. Patzelt, T.H. Inst. f. Elek.Messtchnik Wien, Austria

A. C. Peatfield, DNPL Daresbury, England

P. Quivy, CENG Grenoble, France

H.-J. Trebst, Phys. Inst. Uni Erlangen, Germany

I. P. Vanuxem, CERN Geneva, Switzerland

P. Wilde, RHEL,Chilton, England

W. Woletz, HMI Berlin, Germany

NIM-CAMAC Software Working Group.

L. Costrell, NBS Washington DC 20234

R. Cottrell, SLAC, Stanford CA 94305

W. K. Dawson, TRIUMF Vancouver, Canada

Satish Dhawan, Yale, New Haven, Connecticut 06520 (Chairman)

D. Gustafson, SLAC, Stanford CA 94305

C. P. Hohberger, BNL, Upton, Long Island, N.Y. 11973

F. A. Kirsten, LBL Univ. of California, CA 94720

R. A. LaSalle, Flastate Univ., Tallahasse, Florida 32306

F. R. Lenkszus, ANL, Argonne, Illinois 60439

L. Robinson, Lick Observatory, Santa Cruz, CA 95060

D. Rosenberg, NAL, Batavia, Illinois 60510

R. F. Thomas Jnr. LASL Los Alamos, New Mexico 87544 (deputy chairman)

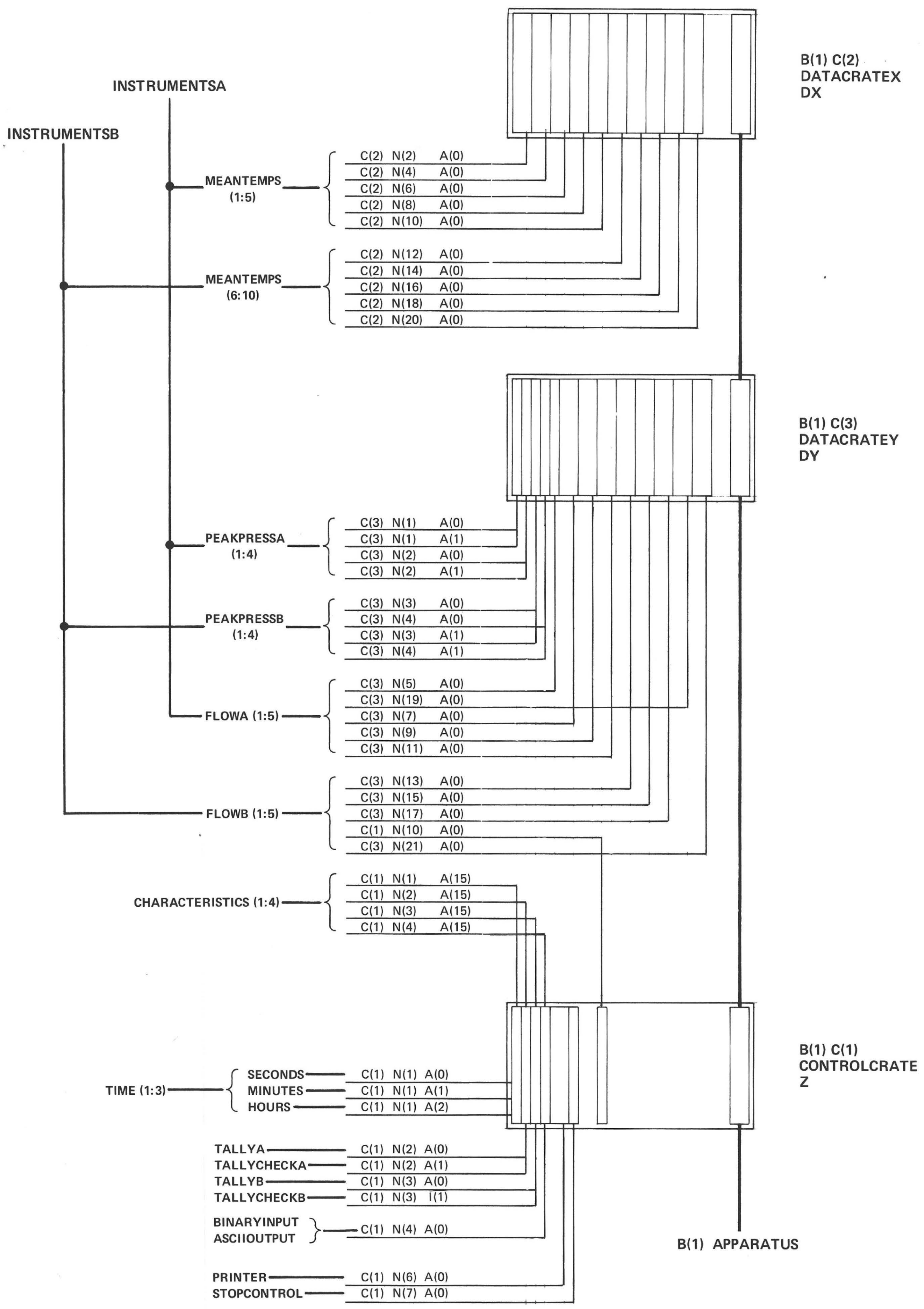


Fig. 1 Hardware addresses used in sample programs