J. MICHELSON, H. HALLING

Kernforschungsanlage Jülich GmbH.
Zentrallabor für Elektronik/
Abt. Nukleare Elektronik und
on-line Datenverarbeitung

The CAMAC Intermediate Language

Kernforschungsanlage Jülich GmbH., Zentrallabor für Elektronik/
Abteilung Nukleare Elektronik und On-Line Datenverarbeitung
517 Jülich, Postfach 365
- J. MICHELSON, H. HALLING -

The CAMAC Intermediate Language

Author's Note: This presentation reflects the state of the CAMAC Intermediate Language at the time it was originally presented. Readers are cautioned to consult the ESONE Software Working Group (SWG) for information about possible changes and the current state of development.

Since its publication in 1969, the CAMAC standard has become a well-established system for construction of peripheral systems, especially data-acquisition and control electronics. Its major advantages to users include modularity, which allows easy expansion of hardware; complete mechanical and electrical compatibility between equipment from different manufacturers; and a standardized hardware interface to the controlling system. The corresponding standardized software interface has not yet been fully developed. There is an active effort underway in the ESONE SWG to develop standardized software for CAMAC, in cooperation with the US AEC NIM CAMAC SWG.

The work of the ESONE SWG has been in two complementary areas. The CAMAC Language, which was described by Mr. Degenhardt, is a high-level language. The Intermediate Level Language, or IML, is an assembly-level language containing the parameters required to control CAMAC hardware. Thus it is a simpler, more pragmatic approach that can be implemented more easily using existing macro assemblers. (Fig. 1) An important similarity between both languages is that they both use the host-parasite concept; both contain only information concerning control of CAMAC hardware and interfacing to the host language and operating system, while the actual data-processing programs are written in the host language. (Fig. 2)

This presentation will summarize the general problems of attempting to standardize software for CAMAC, describe the current state of IML, and introduce an implementation of IML compatible with a FORTRAN host.

The general problems of standardizing software for CAMAC have been summarized very well in a paper published in the IEEE Transactions on Nuclear Science, CAMAC tutorial issue. Excerpts of the paper, written by Richard F. Thomas, Jr. of the Los Alamos Scientific Laboratory, and entitled "Some Aspects of CAMAC Software", are quoted below:

"... NIM and ESONE have given some effort to trying to identify common software problems and suggest general solutions to them, but in the final anylysis the generality of CAMAC hardware limits the extent to which standard software can be defined... It may be helpful, however, to discuss those aspects of CAMAC systems where software problems may exist, to indicate which of these may be alleviated to some extent by standard software, and to suggest general approaches to some of the problems.

"There are three major areas of variability, more or less independent of each other, in most CAMAC systems. [(Fig. 3)] The computer and CAMAC interface comprise one part, and one which can often be covered with a veneer of uniformity using established techniques. The second part is the system description, i.e., how many branches, crates, and modules are there, and where can a given module or collection of modules be found in the system?... [The] locations of modules are normally embodied in a program or its input parameters. The third and last part of the problem is the description of the programs, or procedures, which a particular module or set of modules is to perform. These programs typically depend only on the modules themselves, and not upon their locations or the computer systems to which they are interfaced.

"... [The] separation [of these three aspects of the problem]can be accomplished for the computer and its CAMAC interface by writing programs in languages such as FORTRAN, in which a considerable degree of machine independence can be achieved, and by providing a standardized interface to the CAMAC hardware from such languages.... The remaining two parts of the system description each depend totally on the specific application, and the most one can hope for from general software is that it will make the process of embodying this information in a program as easy as possible. Separating the configuration information from the procedural part of the CAMAC application program generally is not difficult, but may require a conscious effort on the part of the programmer ....

"Perhaps the most difficult problem to deal with in specifying general software for CAMAC is the generality of the hardware ... It is true that CAMAC imposes some rules and constraints on the device interfaces which are to be plugged into the crates, but very much more is unknown than is known about a system until the

modules are specified. Neither the function codes implemented, the meaning, if any, of Q responses, nor the number, meaning, or mode of handling of LOOK-AT-ME signals can be known in advance.

"This problem of generality... leads naturally to a two-level handler concept. (Fig. 4) A low-level handler provides the basic capability to communicate with a CAMAC branch or crate; a high-level handler, much less general, is written to deal with each module or set of modules which may form a functional unit in the system. ...each of these high-level handlers would call upon a single, low-level CAMAC handler for issuing the various CAMAC commands to the devices, for recognizing and sorting the LOOK-AT-ME signals, and for controlling access to the dataway.

"One last point which a designer should consider is the problem of efficiency... It is a well-known characteristic of computer programs that general programs written in high-level languages tend to be slower than special-purpose programs written in assembly languages.This ... is an argument for classification and specialization of procedures which are executed in CAMAC hardware so that they can be ... written to take full advantage of the hardware capabilities available in a given interface while at the same time being accessible to programs in higher-level languages... One classification which can be applied to CAMAC procedures is a functional one: ...The address scan, repeat, and stop modes of using the Q response, defined in [EUR-41ooe] , are examples of such procedures, and the higher-level handlers suggested in the previous paragraph could also be examples... + An alternative classification which is not suitable for standardization over the totality of CAMAC systems, but which is nevertheless valuable to the users of a particular computer and CAMAC interface combination is based on ... [classes] of procedures tailored to a particular type of hardware [;these] probably offer the maximum performance available without individually writing each procedure in machine language."

Now let's consider the present state of the IML as an attempt to deal with some of the problems described by Mr. Thomas. In the ESONE SWG it became clear that a single syntax for IML for all possible hosts was not an optimal approach, so the SWG has concentrated its efforts on a semantic description, leaving open the possibility of several different syntaxes for particular host languages and operating systems.

+ Incidentally, it seems likely that some of these modes would operate much differently if the software people had become interested in CAMAC earlier!

Before starting with semantic details, let's take a short look at CAMAC hardware. One can see several basic structural features about CAMAC from Fig. 5: First, it is a multiplexer system. Therefore it cannot execute truely simultaneous operations in different modules or subaddresses. (This would require using the same bus - the DATAWAY - for more than one operation at the same time.) CAMAC can only perform interleaved operations. Second, the system can be divided roughly into three levels, the first two of which are defined by the ESONE standards EUR-41ooe, and EUR-46ooe. These define the specifications for connecting modules to the DATAWAY and for interconnecting crates to the BRANCH HIGHWAY, respectively. The third level of the system, as yet not standardized by ESONE, is the interconnection of one or more BRANCHES to the host computer-system. It is probable that this highest level can never be standardized. Third, the CAMAC system is a modular standard, so it is possible to use only those levels which are suitable for a given system. For example, a small system can utilize a single crate with a special crate controller acting as an interface to the computer (a 41oo-only system), but a very large system can utilize an "augmented branch" structure (41oo, 46oo, and extensions which are not standard). A serial transmission system for interconnecting crates is under development to complement the parallel BRANCH HIGHWAY. Also, a few points about terminology - usually, the commands to hardware beyond the DATAWAY level are called system commands, while the function directed at a particular module and the parts of the module address taken together are usually called the CNAF. (Fig. 6)

Now let's look at some possible system-structures for CAMAC software. Two general approaches for software control of peripherals are to use an I/O control-language dedicated to the appropriate peripheral (e.g. a channel control program in the 4oo4) or to use a set of I/O requests completely under host-language control (e.g. macro calls to the supervisor in the 4oo4). (Fig. 7) An I/O language is a means of describing a sequence of actions, and includes some sequential control or branching capabilities. In contrast, I/O requests are self-contained individual operations which do not interact except through the host language and operating system. All control capabilities reside in the host.

A good hardware analogy to the I/O control-language is the use of an autonomous I/O processor to control peripherals (e.g. a graphic-display processor). The hardware analogy to the use of I/O requests is a single-instruction processor which performs one action and then waits for further requests or asks the central processor for service (e.g. hardware teletype controller). The basic structure of CAMAC hardware is that of a single-instruction processor or slave system.

The advantages of a CAMAC system using I/O requests include easy implementation and easy interfacing with an existing operating system and host language. For requests which perform complex actions such as a block-transfer, the overhead is not excessive, but for sequences of simple CAMAC operations (one CNAF each), a language approach is more efficient because it minimizes overhead in the host language, especially for conditional sequences of simple operations. The IML attempts to combine the best features of both approaches by defining the parameters to describe either approach and letting the individual implementation choose which approach to follow.

For both approaches, IML still uses the host-parasite concept (Fig. 2) because the cost in both time and money for ESONE or other individual research laboratories to develop and maintain entire software-support systems for many different types of computers is too great - the necessary assemblers, compilers, and operating-system executives usually already exist, and the most cost-effective approach is to use existing software support as far as possible, and extend it with some form of support which deals only with control of the CAMAC hardware and interfacing this support to the host language and operating system as a parasite. This also makes use of the CAMAC extensions much easier. Users are not forced to learn entire new systems. They can program their applications in widely-used languages such as FORTRAN, and run them using the existing operating systems that they already understand. They only have to learn about the parameters holding the CAMAC control information and the information about the interface to the host language and operating-system. This is why ESONE has worked on the IML as well as the CAMAC language. These parameters are the essential part of the semantic description which we'll try to examine next. First, let's define a few structural elements of the IML semantics as it is currently being discussed by the ESONE SWG.

Since CAMAC hardware can not be directly described in most host languages, the IML must provide the means of describing CAMAC hardware. IML provides 5 different types of CAMAC addresses.(Fig. 8) These data types extend the capabilities of the host language. The first two types describe single CAMAC addresses, and the last 3 define arrays of CAMAC addresses which are accessed sequentially.

Because the IML is a parasite language which extends the host's capabilities, the IML uses the software data-types of the host. (Fig. 9) In this way, the data read from or written to the CAMAC hardware can be directly manipulated by host-language data-processing statements.

The IML includes a variety of language elements to describe the operations to be performed on CAMAC hardware. (Fig. 1o) Most of these are concerned with exactly how a particular operation will be performed, or the algorithm to be used. Some language-elements from the host are used (such as numbers or variables for the repeat-count, and statement identifiers for TEST statements), while others are defined directly in IML (for example, the MODE SPECIFIER which describes how the Q-response should be used). An important area which is currently under discussion by the SWG is the definition of language elements for status information. In addition to the actual data transferred, the status which results from a CAMAC operation must be made available to the host-language program.

In an attempt to classify procedures functionally, the IML defines the actual algorithms used to execute certain types of data transfers. These definitions go beyond the hardware specifications of EUR-41ooe to define the software interactions. An example of reading 1o words from a module in REPEAT MODE (waiting for each word to be ready) shows the software aspects which are not defined in EUR-41ooe, such as the repeat count and the test for too many attempts. (Fig. 11)

The various elements of the IML are the set of parameters used to define CAMAC operations. (Fig. 12) The hardware and software data types and the special language elements define the access to a CAMAC module, and the status-information elements tell the program what happened. The special language elements are grouped together in the condition field, and the status-information elements are grouped together in the status field (or they will be when they have been defined better)!

Lastly, the IML separates the types of CAMAC operations into several different classes, on the basis of the function of the operation and the types of parameters which each class requires. (Fig. 13A) Examples are: READ, which transfers data from CAMAC to the host-language program; WRITE, which transfers data the other way; MOVE, which transfers data directly between two CAMAC modules (for example a group of scalers and a hardware display module); and TEST, which tests the Q-response from a particular module and causes a conditional branch depending upon the result.

In addition to the ACTION statements of IML, there are non-executable statements which provide linkage information from the IML program to the host language and operating system. (Fig. 13B) These statements include declarations of hardware data-types and interrupt-linkage information, etc.

The largest unresolved problem in IML is the definition of standard handling of interrupts. This is clearly a difficult problem owing to the diversity of computer interrupt-systems, the interactions with executive functions of operating systems, and the different possible uses of LOOK-AT-ME signals from a CAMAC crate. These request signals may be associated with program interrupts for one module, or be associated only with data-transfer requests (DMA) for synchronizing hardware block-transfers for a different module. The software cannot easily distinguish between these two classes of LOOK-AT-ME signals without additional information. In many systems, the service routines must be imbedded in the operating system, but they cannot be standard since each type of module requires its own special routine!

Lack of space prevents a full description of the implementation which has been under development at KFA, but Fig. 14 gives an impression of the form of IML when used with a FORTRAN host. The IML parameters have been expressed in FORTRAN language directly, and the special language - elements have been coded into numerical values (either by a pre-compiler or the programmer). All that is required to implement the system is to create the necessary subroutines in assembly-language. The linkage statements are also made using CALL statements to avoid modifying the compiler.

At KFA we have also been studying other host languages, such as macro-assemblers and the implementation-language POLYP (being developed by SCS) as possible long-range solutions for CAMAC software using IML.

COMPARISON

CAMAC-LANGUAGE	IML
HIGH-LEVEL	LOW-LEVEL
REQUIRES SPECIALIZED COMPILER	USES EXISTING COMPILERS OR MACRO ASSEMBLERS
USER-ORIENTED	PARAMETER-ORIENTED
EASIER TO USE	EASIER TO IMPLEMENT

FIG. 1

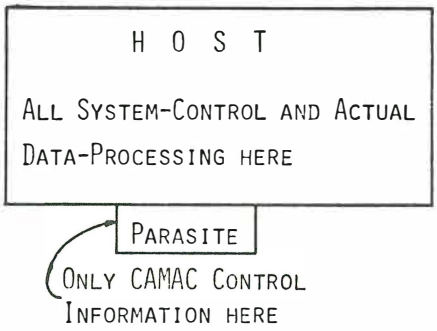
H O S T
ALL SYSTEM-CONTROL AND ACTUAL DATA-PROCESSING HERE
PARASITE
ONLY CAMAC CONTROL INFORMATION HERE

FIG. 2

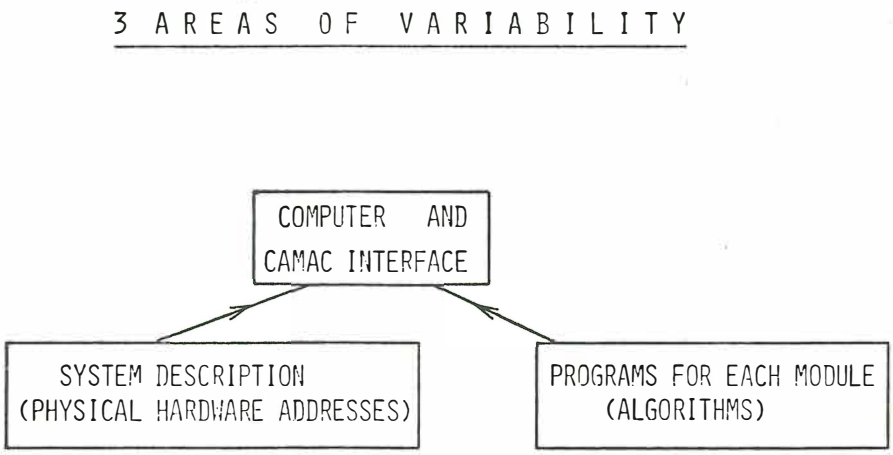
3 AREAS OF VARIABILITY
COMPUTER AND CAMAC INTERFACE
SYSTEM DESCRIPTION (PHYSICAL HARDWARE ADDRESSES)
PROGRAMS FOR EACH MODULE (ALGORITHMS)

FIG. 3

TWO-LEVEL-HANDLER CONCEPT

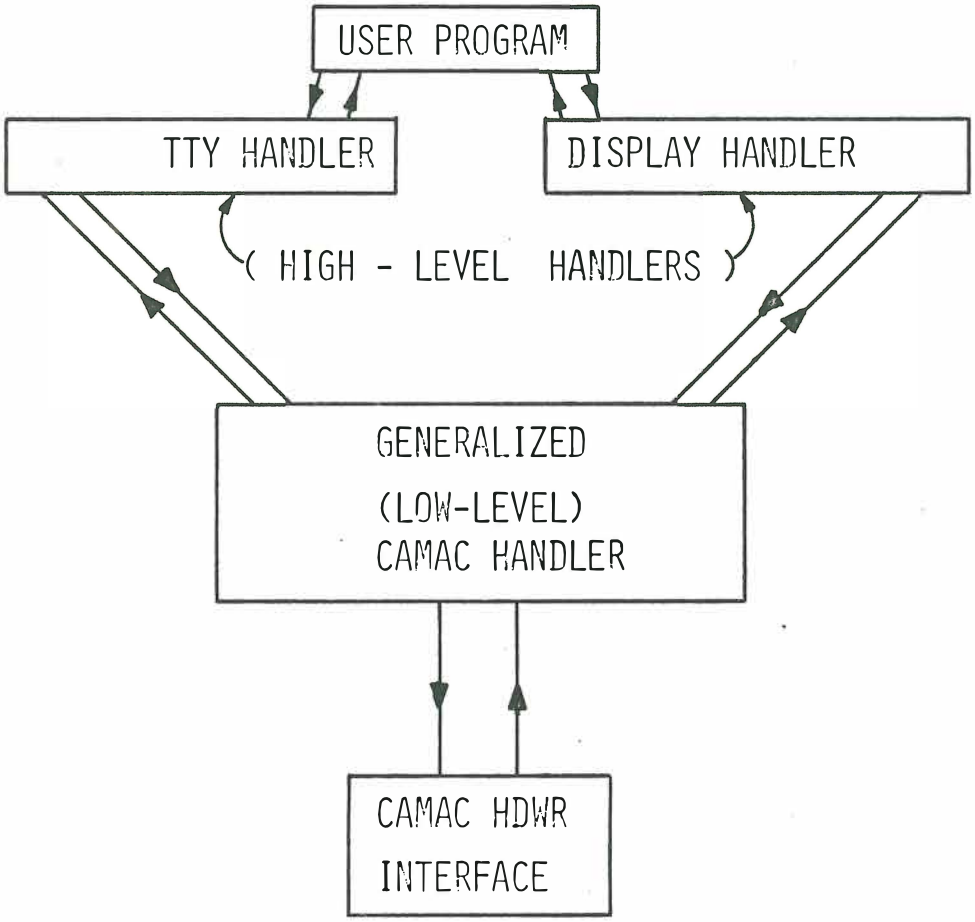

FIG. 4

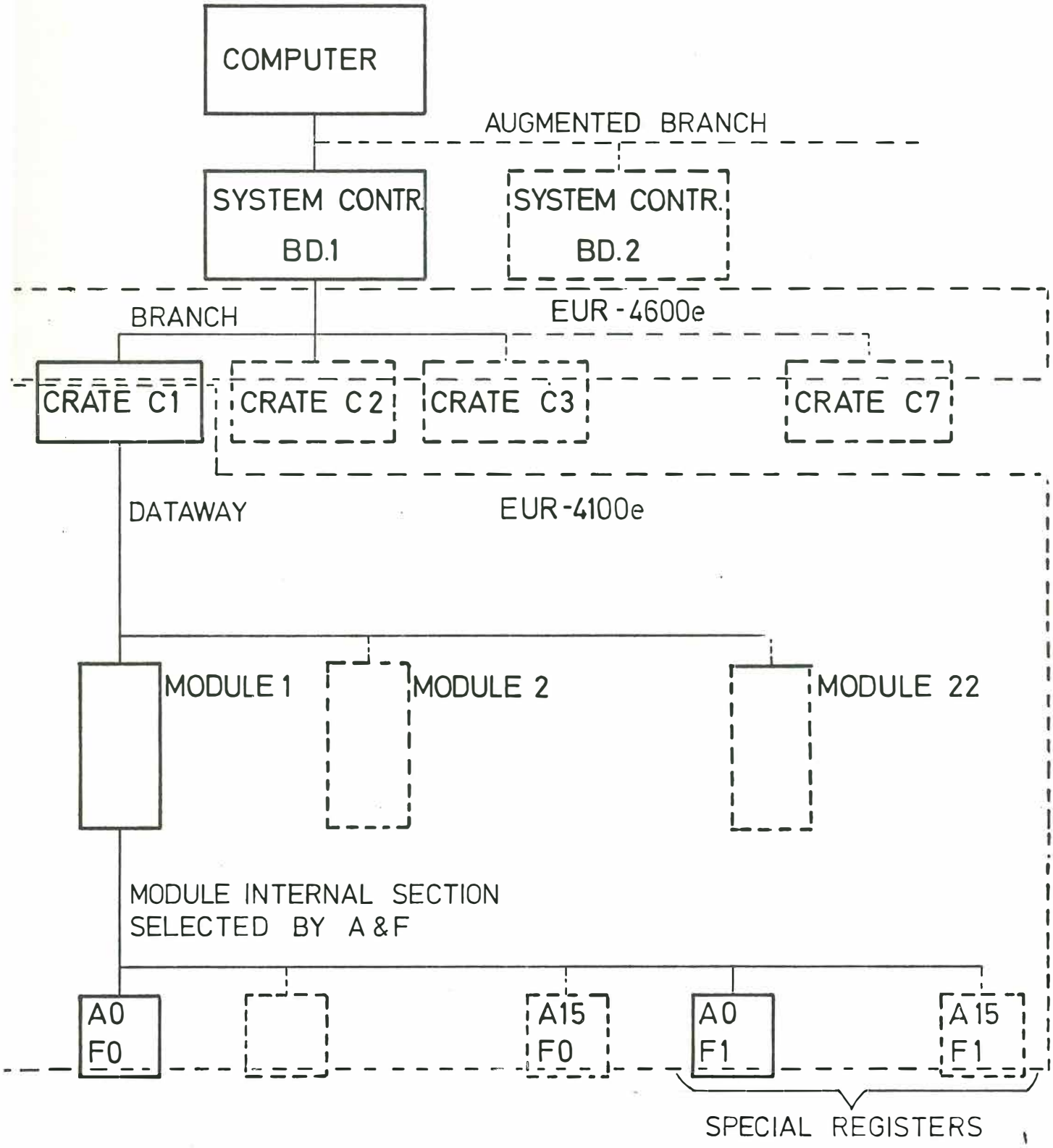

FIG.5 CAMAC MULTIPLEXER SYSTEM

C N A F (ETC.)

C =	CRATE ADDRESS	(3 BITS)
N =	STATION NUMBER	(5 BITS)
A =	SUB-ADDRESS	(4 BITS)
F =	FUNCTION CODE	(5 BITS)
	0 - 7 = READ	
	8 -15 = CONTROL (PART ONE)	
	16 -23 = WRITE	
	24 -31 = CONTROL (PART TWO)	
B =	BRANCH NUMBER	(IF USED)

FIG. 6

COMPARISON

I/O LANGUAGE	I/O REQUESTS
DESCRIBES SEQUENCE OF ACTIONS	DESCRIBES INDEPENDENT OPERATIONS (SELF - CONTAINED)
PROVIDES SOME SEQUENTIAL CONTROL INTERNALLY	ALL SEQUENTIAL CONTROL IN THE HOST
SIMILAR TO AN AUTONOMOUS DISPLAY-PROCESSOR	SIMILAR TO A HARDWARE TTY CONTROLLER
CONTAINS SOME MASTER CAPABILITIES	COMPLETELY SLAVE SYSTEM
MORE EFFICIENT FOR SINGLE COMMANDS	EASIER TO INTEGRATE WITH EXISTING SYSTEMS

FIG. 7

DATA TYPES

HARDWARE

H (HARDWARE) - NUMERICAL B, C, N, A

P (POINTER) - NAMED B, C, N, A

G (GIVEN ARRAY) - NAMED GROUP OF B, C, N, A'S (RANDOM)

C (CALCULATED ARRAY) - NAMED GROUP OF B, C, N, A'S WHICH CAN BE COMPUTED FROM THE FIRST AND LAST USING INCREMENTS

Q (Q-SCAN ARRAY) - NAMED GROUP OF PARAMETERS FOR A "Q ADDRESS - SCAN". FIRST B, C, N, A AND LAST B, C, N, A VALUES.

FIG. 8

DATA TYPES

SOFTWARE

USE THE DATA-TYPES OF THE HOST.

E.G. FORTRAN:

SIMPLE VARIABLE - I

ARRAY VARIABLE - IX (1o)

FIG. 9

LANGUAGE ELEMENTS

STATEMENT IDENTIFIERS

REPEAT COUNTS (DESIRED NUMBER OF OPERATIONS)

MODE SPECIFIERS

RQ REPEAT MODE
SQ STOP MODE } SEE EUR-4100E

.
.
.

ACCESS MODES

SINGLE-ADDRESS (VARIABLES/ARRAY ELEMENTS)

MULTIPLE-ADDRESS (ARRAYS)

DEFINITION OF ALGORITHMS

STATUS INFORMATION

Q - RESPONSE

X - RESPONSE

COUNT OF OPERATIONS ACTUALLY ACHIEVED

HARDWARE ERRORS

FIG. 10

A L G O R I T H M S

EXAMPLE

REPEAT MODE, 1o WORDS, READ:

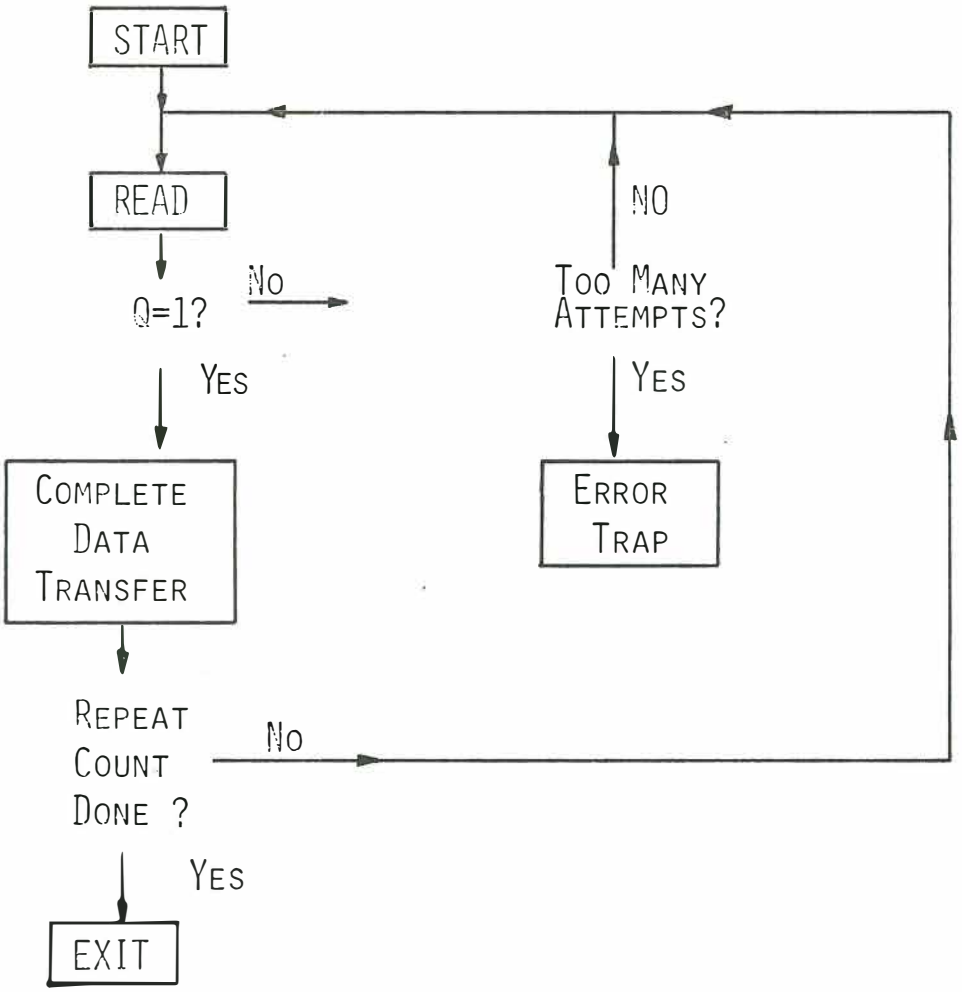

FIG. 11

PARAMETERS FOR ACTION STATEMENTS

F - CODE

EXTERNAL REFERENCE (BCNA)

INTERNAL REFERENCE (SOFTWARE ADDRESS)

MODE SPECIFIER
REPEAT COUNT
STATEMENT IDENTIFIER

REQUIRED ONLY FOR CERTAIN OPERATIONS

CONDITION FIELD
(DEFINES METHOD OF ACCESS TO CAMAC MODULE, OR THE ALGORITHM)

STATUS - INFORMATION ELEMENTS

FIG. 12

TYPE OF ACTION STATEMENT	TYPE OF DATA-TRANSFER OR OPERATION
READ	CAMAC → PROGRAM
WRITE	PROGRAM → CAMAC
OPERATE	CONTROL FUNCTION → CAMAC
MOVE	CAMAC → CAMAC
STATUS	Q-RESPONSE → PROGRAM
TEST	BRANCH ON Q-RESPONSE
SYSTEM	PROGRAM → INTERFACE HDWR.
ERROR	PROGRAM → INTERFACE SFTWR.
EXEC	PROGRAM → OPERATING SYSTEM

FIG. 13A

LINKAGE STATEMENTS

INFORMATION FROM IML TO THE HOST

FIG. 13B

CAMAC CALL IN FORTRAN (FORTRAN IV PDP-11)

GENERAL FORM

CALL READ (F, HRDW, SOFTW, MODE [,STATUS])

F: F CODE

HRDW: CAMAC ADDRESS INFORMATION (NAMED)

A) SINGLE ADDRESS:

1) ADDRESS VALUE

2) POINTER

B) GIVEN ARRAY

C) CALCULATED ARRAY

D) Q SCAN ARRAY

SOFTW: NAME OF A MEMBER OF A CLASS OF HOST DATA TYPE OR STRUCTURE

MODE ARRAY: WAY OF ACCESS AND ADDITIONAL INFORMATION (USE OF Q, X, ERROR ETC.)

STATUS ARRAY:

A) MAY BE IN A COMMON AREA HOLDING CURRENT STATUS (WC, LAST Q, ETC.)

B) STATUS VARIABLE (OPTIONAL) FOR REALTIME SYSTEMS (REENTRANCY)

EXAMPLE

CALL READ (Ø, SCALER, HOLD, EASY, ERR)

ARRAY EASY:

REPEAT COUNT

WAIT FOR EACH INPUT WORD (REPEAT MODE)

TRY ONLY 1ØØØ TIMES

FIG. 14