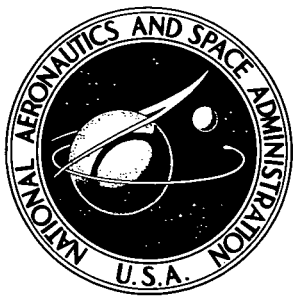
NASA TECHNICAL NOTE



NASA TN D-7148

NASA TN D-7148

A PROGRAMMABLE COMPUTER INTERFACE FOR CAMAC

by Robert W. Bercaw, Theodore E. Fessler, and Jeffrey M. Arnold

Lewis Research Center
Cleveland, Ohio 44135

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • MARCH 1973

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TN D-7148		

4. Title and Subtitle	5. Report Date
A PROGRAMMABLE COMPUTER INTERFACE FOR CAMAC	March 1973
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Robert W. Bercaw, Theodore E. Fessler, and Jeffrey M. Arnold	E-6957
	10. Work Unit No. 503-10

9. Performing Organization Name and Address	11. Contract or Grant No.
Lewis Research Center National Aeronautics and Space Administration Cleveland, Ohio 44135	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Note
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

An interface has been developed for CAMAC instrumentation systems that implements data transfers controlled either by the computer CPU or by an autonomous (data-channel) processor in the interface unit. The data channel processor executes programs stored in the computer memory. These programs consist of standard CAMAC module commands plus special control characters and commands for the processor itself. The interface was built for the PDP-15 computer, which has an 18-bit word structure, but both 18- and 24-bit data transfers can be made. A software system has been written that exploits the many features of the processor.

17. Key Words (Suggested by Author(s))

Computer interface
CAMAC
Instrumentation
Data acquisition
Data processor
Direct memory access

18. Distribution Statement

Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	56	$3.00

* For sale by the National Technical Information Service, Springfield, Virginia 22151

A PROGRAMMABLE COMPUTER INTERFACE FOR CAMAC

by Robert W. Bercaw, Theodore E. Fessler, and Jeffrey M. Arnold

Lewis Research Center

SUMMARY

An interface has been developed for CAMAC instrumentation systems that implements data transfers controlled either by the computer central processing unit (CPU) or by an autonomous (data-channel) processor in the interface unit. The data channel processor executes programs stored in the computer memory. These programs consist of standard CAMAC module commands plus special control characters and commands for the processor itself. The interface was built for the PDP-15 computer, which has an 18-bit word structure, but both 18- and 24-bit data transfers can be made. A software system has been written that exploits the many features of the processor.

INTRODUCTION

The fields of data acquisition and control are undergoing revolutionary change under the impact of small computers. The benefits to be obtained are now so apparent that they need not be discussed. There are, however, numerous pitfalls and difficulties which must be overcome by anyone whose requirements are not satisfied by some off-the-shelf system. Foremost amongst these are the hardware and software problems involved in interfacing specialized research instrumentation to these computers. The makers of such instruments usually do not have the resources to supply the service for the vast variety of computers on the market. The computer manufacturers, on the other hand, are not very concerned with supplying it since the number of systems using any particular instrument is small. Clearly what is called for is some degree of standardization. A number of laboratories in the United States and Europe have formally established a standard interfacing system known as CAMAC (ref. 1), an acronym for computer aided measurement and control. CAMAC replaces the great variety of input-output busses found

on various models of computers with a single nonproprietary design, standardized both mechanically and electrically. It consists of a number of bins or crates each of which will accept up to 24 modules containing instrumentation to be interfaced to the computer. The CAMAC specifications restrict the instrumentation contained in a module only to the extent necessary to insure its electrical and mechanical compatability.

At the rear of the crate, there is a bus structure called a dataway which provides power to the modules and also links them to the computer. It is designed to minimize the logic needed in a module and also to provide a large repertoire of operations for the module designer. The architecture of the dataway is not patterned on the input-output structure of any make of computer, and therefore it is necessary to interface it to the chosen computer, that is, to resolve the time and logic differences between the dataway and the host input-output system. Although this may seem to be just a case of shifting the interfacing problem to a different part of the system, there are significant advantages. The interfacing task is performed just once for all the modules. The instruments and the computer are made independent and either may be changed or replaced with no effect on the other (Even the use of a computer is optional; some systems have been built using only a data recorder.). The price which must be paid is the addition of another unit variously called the interface, the CAMAC controller or the CAMAC processor. Its design is singularly important and will to a large extent determine the performance of the total system.

The CAMAC processor described in this report was designed to fulfill demanding requirements. It was to take data at high rates and also to control a wide variety of instrumentation. (It is currently being used for the automatic operation of a cyclotron facility and for recording the data produced by the experiments performed on it.) The guiding principles used in its design were to allow rapid reconfiguration of experiments, to impose a minimum of constraints on the generality of the CAMAC system, and to permit computer "throughputs" of several tens of thousand words per second. A distinction is drawn here between throughput, which includes computing, and mere data transfer rate.

The CAMAC processor has been designed and built for the Digital Equipment Corporation's (DEC) PDP-15, an 18-bit computer having an input-output structure similar to many other machines. The CAMAC processor consists of two separate

sections. One implements the computer's programmable input-output bus, a facility that places peripheral devices under the control of the program running in the computer's central processing unit (CPU) and that provides for transfers of data betweeen the devices and the computer's accumulator. The second implements the computer's data channel, a facility that provides direct high-speed transfer of data between the devices and memory. We will refer to these two sections as the CIOP, the CAMAC input-output processor, and the SPCC, the stored program CAMAC channel, respectively. A rough block diagram of the system is shown in figure 1. Data may flow between the CAMAC dataway and the computer through either the CIOP or the SPCC. The CIOP involves the accumulator, and the SPCC the memory. Both sections are under the control of the instructions executed by the CPU and both are able to interrupt its operation or supply it with status information. The difference between them is that the CIOP requires one or more input-output instructions for each word transferred, while the SPCC uses input-output instructions only to initialize and stop the data transfer. The "stored program" in SPCC refers to the fact that it operates independently of the CPU under the control of programs composed of commands stored in the computer's memory.

The SPCC is quite unusual and avoids many limitations found in most other designs. Some of its features are

Full repertoire of standard CAMAC operations
Arbitrary sequences of CAMAC operations
Module-initiated program execution
Module-controlled program branching
Dual independent subchannels
Direct memory increment
Independence of the CIOP
Intermixing of 24- and 18-bit data transfers.

This processor was designed as a two-crate controller for the DEC PDP-15; however, the principles embodied in it are more general and can be applied to other computers having comparable input-output structures. There is nothing unusual in its design that would prevent its adaptation to a branch driver.

This report is divided into three major sections. The first two describe hardware aspects of the CIOP and the SPCC. The third section describes the PDP-15 software necessary to use these CAMAC processors. The remainder of this intro-

duction briefly describes the CAMAC system and may be omitted by those familiar with it.

The CAMAC crate fits into a standard relay rack and contains 24 or 25 slots, called stations, into which instrumentation modules may be inserted. There is an 86 contact connector at the rear of each station through which the module communicates with the system. The array of connectors and their interconnecting wiring is called a dataway. Most of the connector contacts are wired to form a bus structure; that is, each contact on a connector is wired to the same contact on all the other connectors. The end connector (number 24 or 25), however, is wired differently than the rest and is known as the control station. It and the one adjacent to it are used to connect the dataway to the computer and to hold the crate controller, the name given to the electronics used to control the operation of the dataway. The dataway is organized into a number of functional substructures. Separate 24 bit wide read and write busses are used to transfer data in the form of 24 bit words (This does not imply that the registers contained in the modules must be 24 bits long.). Control of the modules is primarily though another bus containing four subaddress lines, five function lines, and several master control lines. The subaddress lines may be used to divide a module into as many as 16 submodules; the function lines allow up to 32 different operations to be performed by any submodule. Individual stations are addressed by N or station lines, there being one line between each station and the control station. A second set of lines, called look at me or LAM lines, parallels these and are used by the modules to request service.

The format of the commands used to control the CAMAC dataway and modules is shown in figure 2(a). The five bit field N specifies the station number of the module of interest, A is a four-bit subaddress within the module and C is the crate or bin containing it. Any one of the 32 operations listed in table I may be performed on a module as specified by the five-bit function code field, F. The individual bits are denoted by F1, F2, F4, F8, and F16 where F1 is the least significant. Specific fuction codes, which are composites of the entire field, are distinguished from the individual bits through the use of parentheses. For example, the "overwrite group 1 register" operation, invoked by function code 16, is denoted by F(16). The same convention is used for the A and N codes. It should be noted that most of the bits in the function code have standard useage.

All standarized operations have F4=false and all codes having F8=false involve data transfer. The F16 bit indicates the direction of data transfer.

The status of a given feature of a module is given by the signal it imposes on the "Q-response" bus line in the dataway when it is addressed. Different F-A combinations can be used to test different features. There are conventions for the Q-responses to most of the standard function codes. The Q, H, and E bits in the command word shown in figure 2(a) are specific to this processor and not part of the general CAMAC system. They will be discussed later.

The timing of signals on the CAMAC dataway follows conventional methods for bus structures. The data source is first gated onto the bus and then, after allowing some time for the bus to settle, the data are strobed from the bus into its destination. Figure 2(b) shows the timing of a CAMAC dataway transfer or cycle. Commands and the data to be transferred are placed on the dataway for 1000 nanoseconds accompanied by a signal on the busy control line (B). Two strobe pulses are generated on the S1 and S2 control lines during this period. The first may be used to strobe data off the dataway into a register of either a module or the interface, while the second may be used to clear a register, disable a module, etc.

GLOSSARY

An	any specific CAMAC subaddress code bit; allowed values are A1, A2, A4, and A8
API	automatic priority interrupt
A(n)	any CAMAC subaddress code produced by sum of subaddress code bits; allowed values are A(0) to A(15), for example, A(7)=A1+A2+A4; A(8)=A8
CA	current address
CAMAC	computer aided measurement and control; refers specifically to set of nonproprietary mechanical and electrical standards
CIOP	CAMAC input-output processor; that part of CAMAC interface which communicates with computer via IOT's
CPU	central processing unit; main control and arithmetic unit of a computer
CR	CAMAC command register

Data Channel	hardware in computer that allows an external device to communicate with computer memory without use of the CPU; also known as a direct memory access (DMA) channel
Dataway	The bus structure at the rear of a CAMAC crate
Fn	any specific CAMAC function code bit; allowed values are F1, F2, F4, F8, and F16
F(n)	any CAMAC function code produced by sum of function code bits; allowed values are F(0) to F(31), for example, F(7)= F1+F2+F4; F(8)=F8
HDR	CAMAC high data register
IOP	input-output pulse; pulses sent to external device by CPU when it executes IOT instruction
IOT	input-output transfer; instructions executed by computer that allow CPU to communicate with external device
LAM	look-at-me; signal produced by CAMAC module to request service
LDR	CAMAC low data register
PC	program counter
PI	program interrupt
Q-Response	line in dataway that allows module to return one bit of information to computer
R Lines	read lines; lines in dataway used to transfer data from a module to the computer
SPCC	stored program CAMAC channel; that part of CAMAC interface that communicates with computer via data channel
WC	word count
W Lines	write lines; lines in dataway used to transfer data to a module from the computer

CIOP-CAMAC INPUT-OUTPUT PROCESSOR

The input-output processor of the PDP-15 provides for program controlled

transfers of data between its accumulator and peripheral devices. The input-output transfer instructions (IOT) are used for this purpose, and their format is shown in figure 3(a). They have a six-bit operation code identifing their input-output nature, an eight-bit device select field (organized into a six-bit address and a two-bit subaddress), and a field of three bits to control the issuance of three input-output pulses (IOP). One additional bit allows clearing the accumulator before the execution of the instruction. The computer places the contents of the device select and IOP fields on 11 separate lines of its input-output bus in the time sequence shown in figure 3(b). These 11 signals may then be used by a peripheral to control a variety of operations such as reading, writing, testing flags, etc. The IOT instructions used by the CIOP (and SPCC) are listed in table II.

Comparison of figures 2 and 3 shows that the CAMAC command contains considerably more information than can be programmed into the IOT instruction of the PDP-15 and that the timing requirements are quite different. CAMAC systems cannot be interfaced merely by adapting the signals available on the PDP-15 input-output bus. There must be an active translation between the two languages in which the CAMAC command itself is treated as data and transferred to a holding register (the command register) for use when the actual data transfer takes place. A block diagram of the CIOP is shown in figure 4. In addition to the command register (CR), there is also a data register to resolve the timing differences between the two systems. It is split into two parts because of the difference in the word lengths of the two systems. In order to transfer 24 bits to a CAMAC module, three computer input-output transfers are made: one to the CR, one to the high data register (HDR) and one to the low data register (LDR). The CIOP then executes a dataway cycle to transfer the information from the data registers to a module. The sequence of operations is exactly the same for reading a module into the computer; however, an intermediate holding register is not needed. CAMAC only specifies the minimum times in a cycle (fig. 2(b)), they can be lengthened to accommodate the input-output specifications of the PDP-15 and thus permit the CAMAC read lines to be merely gated onto the input-output bus. Many modules have registers of 18 bits or less. These can be read or loaded using only two transfers by simply omitting the transfer of the high data. Similarly operations that do not involve data transfer (i.e., when F8 = true) are performed by loading only the CR. For all types of transfers, the dataway cycle always takes place

immediately after the last input-output transfer. The high-order bit of the low read lines is also read in with the high read lines for reasons arising in the SPCC. It occupies the least significant bit and is normally trimmed off by a right rotation of the accumulator. This redundant use of a bit does not occur in the high-order write transfers.

There are several other major parts to the CIOP. The IOT decoder interprets the device select and IOP lines of the computer input-output bus and supervises the operations listed in table II. The clock issues the busy signal (B) and the two strobe pulses (S1 and S2) in the sequence shown in figure 2(b), and it controls the gating of data onto or off of the dataway. It initiates a dataway cycle whenever a low-data-transfer IOT is sent or whenever a dataless command (one having F8 = true) is loaded into the CR.

The station decoder produces a signal on the module station line specified by the five-bit N field. It also provides for operations that address all modules by means of virtual modules at some of the nonphysical station numbers. These operations are listed in table III.

Modules may request service through individual look at me (LAM) lines. These may request either the SPCC or the program interrupt facilities (fig. 5). The facilities for handling interrupt requests is rather elementary since it is assumed that most high-rate requests will be handled by the SPCC. All LAM lines of interest are connected in common to the computer interrupt facilities, which may be either program interrupt (PI) or automatic priority interrupt (API). Determining which module has raised its flag is accomplished by polling, using the Q-response line. The result of each test is stored in the Q bit of the CR where it can either be tested with the skip facility of the PDP-15 or be read along with the rest of the CR.

In CAMAC initiated interrupt routines, care must be taken to save and restore the CR and HDR. There is no need to save the LDR since the dataway cycle is performed as soon as it is loaded. The Q flag, being a part of the CR, is automatically saved, however the F8 bit must be cleared prior to restoring the CR, or a new dataway cycle may occur.

DATA CHANNEL PROCESSOR

The purpose of a computer's data channel (also referred to as direct memory

access (DMA)) is to provide for direct transfers of data between memory and a peripheral without interfering with the operation of the CPU. To do so, it must be provided with the address in memory used to store or fetch the data, the address on the input-output bus of the peripheral device, and the direction of transfer. In the PDP-15 single-cycle data channel, this information is provided by the electronics of each peripheral device using the channel. A device requests to transfer data, and the computer subsequently recognizes it by granting it sole control of the input-output bus and data channel. Next, the device sends the memory address to the computer over the input-output address lines of the bus. The data word is then transferred in the direction specified by the device.

In the usual organization of a data channel, a block of memory called a buffer is reserved for the array of data to be transferred. The device is provided with two counting registers known as the current address (CA) and word count (WC) registers. The CA specifies the memory location, and the WC keeps track of the number of words transferred. Both are incremented each time a word is transferred. Normally, the CA register is loaded with the address of the first location in the buffer, and the WC register is loaded with the negated length of the buffer. Transfers occur until the WC register reaches zero or overflows. Then the data channel is turned off, and the CPU is interrupted so that it may service the channel.

The generality of CAMAC complicates this straightforward scheme because the numerous modules with their multiple subaddresses and function codes constitute many devices in one. The elementary approach of having separate electronics for each device becomes either prohibitively expensive or greatly limits the generality of CAMAC. Fortunately, these limitations can be avoided by using an approach that gives the data channel the attributes of a processor. In order to see what this means, consider the CIOP as a prototype data-channel processor and the computer as a black box. The CIOP is then seen as performing operations on modules according to the commands placed in its command register, and the sequence of commands can be considered to be its program. The computer's main duty is to serve as a source (and sink) for data and commands. Of course, the CIOP depends entirely on the CPU for its operation, but it will be seen that the CPU can be replaced by a data channel facility such as is found in the PDP-15.

The principal requirement for an autonomously operating processor is that its command word fully specify its execution phase. CAMAC's function codes are well

suited for this since a single bit, F8, identifies dataless operations while another, F16, defines the direction of data flow. They have no provisions for describing the word size (18 or 24 bits) or to control the processor itself. These must be added, but fortunately the word size of the PDP-15 provides enough room to include the necessary parameters. A processor must also have some method of addressing data and commands in memory. The usual method used for data channels, that of sequential addressing, is generally adequate for both data buffers and program commands. But it is desirable and convenient to provide commands for program jumps and conditional skips. There is little need for more flexible data addressing because of the parallel capabilities of the CIOP.

A simple data-channel processor based on these principles will have a program counter to address commands and WC and CA registers to control the data buffers. It will operate according to the flow chart shown in figure 6. On receiving a request, the processor enters a program by loading its program counter with the starting address of the program to be executed. Commands are then fetched and executed sequentially until an exit code is found in a command. The program counter is incremented after each fetch, while the WC and CA registers are incremented after every data transfer. At the termination of the execution of a program, the processor will, depending on the WC overflow flag, either pause and wait for a new request or disable itself and interrupt the computer CPU.

SPCC Architecture

This section contains a brief description of the main elements of the data-channel processor, SPCC, which we have developed using the preceeding ideas. Details of its construction are given in appendix A. A block diagram of the principal elements and data paths of the SPCC is shown in figure 7. The single-cycle data channel of the PDP-15 is not shown, but it is used for transfers of both data and commands between memory and the SPCC. Data paths are identical to those of the CIOP except for the addition of the program counter.

Four different SPCC programs may be executed on a time shared basis. A program is entered on the request of an "event" or flag from an external device, which may be either a LAM or a BNC coaxial input connector located on the front panel. The LAM's enter through a patch panel so that they may act either as an event input or as a program interrupt (see fig. 5). Event inputs pass through a monitor that

holds requests until they can be serviced and also schedules their servicing according to a fixed priority scheme.

The control of the processor is performed by a module on the dataway that can be addressed like any other CAMAC module. The control of the SPCC is thus accessable to both the CIOP and the SPCC, providing a communication link between SPCC and CPU programs and allowing the SPCC some control over itself. The control module contains WC and CA registers, overflow and error flags, and event enables, and it uses the commands listed in table IV. There are two separate sets of registers to provide two independent subchannels, a requirement if input and output are to be carried on simultaneously. The WC overflow flags are passed to the computer interrupt and skip facility in the same manner as all other module Q's and LAM's except that the SPCC LAM has a dedicated interrupt channel (API).

Other sections of the SPCC are the channel control, which supplies the sequencing and command interpretation logic, and the address multiplexer, which generates memory addresses for the data channel. It obtains address information from the PC, the two CA registers, the read lines of the dataway, and the event monitor. The dataway read line input allows the SPCC to perform the direct memory increment using data from a module as the address. The event monitor generates a unique address for each, known as the event address, from which the SPCC initially loads the PC. The PC and CA registers specify the addresses for commands and data buffers.

The command word used by the SPCC is shown in figure 2(a). It has three bits in addition to the usual CAMAC command information. The most significant bit is used to control the conditional skip. The SPCC will skip the next command if the Q-response from a module is different from the value of this bit. The high data bit is set when it is desired to transfer the full 24-bit word by making two transfers; normally only a single 18-bit transfer is made. Since SPCC programs are usually short, devoting a command to a stop code would significantly lengthen a program. Therefore, the SPCC is stopped by setting an exit bit (E) in the last command to be executed.

Several nonstandard function codes have been employed to provide special handling by the SPCC. F(12) has been employed to provide for unconditional program jumps within a page. The SPCC loads the 12 least significant bits of the

command into its program counter when it encounters this function code. No dataway cycle is executed. Two codes, F(4) and F(6) are used to invoke the direct memory increment mode.

SPCC Operation

Figure 8 illustrates the main steps in processing an event. In the request phase, events are stored in the event latches. When the SPCC becomes free, it activates the highest priority event pending. The event table, composed of the four event addresses (memory locations 24 to 27 (octal)), is read to obtain the address of one of the four relocatable SPCC programs. Normally this address is used to initialize the program counter. The address can be interpreted as a command, however, and placed into the CR instead. The use of this option will be discussed later. In the next or fetch phase, a command is read from memory at the address given by the program counter and, unless it is a jump, it is loaded into the command register and the PC is incremented.

The execute phase is then entered starting with tests of the F8 and H bits of the command. Depending on the results, zero, one or two data transfers are made with the direction of data transfer depending on the F16 bit and then a dataway cycle is executed. The WC and CA registers are incremented after each data transfer. If the Q-response from the module is different from the value programmed into the most significant bit of the command, the program counter is again incremented to create a skip. This provides for device controlled program branching. Finally, the SPCC branches on the exit bit, either returning to the fetch phase or going on to the exit phase. In this last phase the latch of the event serviced is cleared, and the SPCC halts and waits for another event, or, if there is a WC overflow, it disables the subchannel and interrupts the computer for buffer processing. Note that the interrupt occurs when the SPCC program is finished, not when the WC overflows. The actual buffer length must exceed the word count by the size of the largest event.

In pulse height analysis and in a number of other types of experiments, data acquisition consists of building a histogram of the frequency of the values of some variable. Each time an event occurs, the digitized value is used to specify a bin, and the contents of that bin are incremented by one. The SPCC is capable of directly incrementing a word in memory without involving either CPU or its own subchannel core buffers. Commands having F(4) or F(6) invoke this mode. The SPCC strips off

the F4 bit placing a read (F(0)) or read and clear (F(2)) command on the dataway and gates the R1-R15 lines onto the input-output address lines. The data channel is then instructed to perform the add-one operation on the location specified by the module. There is no indication if the word overflows.

Because there are important classes of programs that use only a single data transfer or command, provisions have been made to eliminate the overhead involved in loading the program counter. This is achieved by storing a command in the event table rather than the address of a SPCC program. Since addresses are limited to 15-bit numbers (32 767), a command can be specified by setting the most significant or Q bit to one. The command register is then directly loaded, but additional commands can not be executed because the PC has never been initialized. The error flag is raised if the E bit has not been set equal to one in the command.

Design Considerations

The final design of the SPCC embodies a number of choices which affect its cost and performance. These choices were made to optimize it for our particular use and other solutions might be preferable in other circumstances. The principal decision was to store the channel programs in the computer's main memory. This is the least expensive method, and it is also simple and flexible. The programs can be assembled and loaded using the standard software supplied with the computer and there are no limits to the sizes of the programs. On the other hand, it requires the transfer of a command for every dataway operation and thus limits the SPCC to speeds one half or less of that which could be realized in a unit storing the commands in its own memory.

The number of events was chosen to be four, a number that may appear small when compared with patch wired designs. In the SPCC, however, an event merely defines the starting location of a channel program, and so the number of events only limits the types of transfers that can be active at one time. Additional transfer modes can be implemented by one of three methods. First, the skip facility increases the effective number of channel programs by allowing conditional branching. Second, if the timing of the transfers is not critical, they can share an event in a manner similar to the time sharing of a large computer. All that is required to change the identity of an event is to reload its entry in the event table and to reassign the LAM

that activates it (Several LAM's may be mixed using the wired-OR). Finally, any low rate transfers may be handled by the CIOP.

The SPCC contains dual WC and CA registers and an event can be associated with either set. It could have been designed with a single set or with a set per event. The former was rejected because it would not permit simultaneous input and output and the latter because we did not feel that the advantages justified the expense. The use of a set of WC and CA registers per event allows storing each event in its own core buffer and thus eliminates the need to identify them when packing and unpacking the buffers. The programs handling the buffers are therefore made simpler and faster. Most of these advantages are realized in the present design because most experiments either use two or less events or they have only one event that is critical. One set of registers can be dedicated to the critical event, while others share the second set. A disadvantage in using a set of registers for each event is that the sequence of occurrence of the different events is lost. A disadvantage of the present scheme of buffers is that events cannot interrupt the execution of lower priority events. This capability is possible, but it requires a program counter as well as a WC and CA register per event.

The cost of components and wiring, using the DEC M-series logic and automatic wire wrapping was approximately $5000. This does not include engineering or documentation costs.

PROGRAMMING CONSIDERATIONS

Data transfers between CAMAC modules and the computer memory can be thought of as taking place at three levels. First are data transfers carried out by the central processing unit (CPU) using a series of input-output instructions and module commands in a mainstream program. Next are interrupt-level transfers. These are also carried out by the CPU, but their execution begins when an interrupt signal suspends execution of one program in favor of another (interrupt-level) program. And third are data-channel transfers which, once initiated, proceed without CPU involvement.

With interrupt programming, a device-ready signal interrupts the program in progress so that a series of input-output transfers (IOTs) can be executed by the CPU. When the IOTs have been completed, execution of the interrupted program is resumed. This way, the CPU is free to execute a mainstream program until a flag

occurs. With data-channel programming, many flags can be handled between interrupts. Data-channel transfers do not involve the CPU, which is left to proceed with execution of the current program in a normal way. Only when a whole series of data transmissions has been completed is an interrupt generated. Thus, data-channel transfer programming permits very high data rates.

In order to fully exploit the capabilities of our CAMAC processor, a systems approach has been taken. The aim of this programming system is to simplify writing new programs that make use of a variety of CAMAC modules at mainstream, interrupt, and data-channel levels.

Mainstream Programming

Data transmissions to or from a module in a CAMAC system require making two data transfers. One transfer is between the CAMAC processor and the computer and is made by a series of instructions stored in the computer. The other transfer is between a module and the CAMAC processor and is performed by the processor.

IOT instructions. - The PDP-15 is provided with a set of instructions to control the transfer of data between the CPU and external devices. These input-output transfer (IOT) instructions perform a variety of tasks including the enabling and disabling of devices, the testing of flags and the transfer of 18-bit words of data. Table II summarizes the IOT-instructions (and their mnemonic names) that address the CAMAC input-output processor (CIOP).

The first group (LCR to SQF) of IOT-instructions in table II are those normally used in mainstream programs. The SQF instruction causes the computer to skip the next program instruction if the Q-bit (high-order bit) of the CAMAC command register is set. The other IOT-instructions in this group are for data transfers to and from the computer's accumulator. The last eight IOT-instructions in table II are those used for control and handling of interrupts.

Module commands. - Commands to CAMAC modules are the sum of four parts (see also fig. (2a)):

crate address	x40	(octal)
+ module address		
+ subaddress	x400	(octal)
+ function code	x10000	(octal)
= module command		

In our CAMAC system, the crate address is either 0 or 1, and module addresses are 1 to 24 (decimal) inclusive. Subaddresses and function codes are used according to CAMAC standards (see table I).

Module commands are stored in the computer program. When one is to be executed, it is transmitted to the CIOP command register by the program sequence (PDP-15 assembler language):

```
.
.
LAC  COMND  /load accumulator with module command
LCR         /load CIOP command register
.
.
```

If the function code part of the module command specifies data transfer, the data are subsequently transferred through the accumulator using one or more of the data transfer IOT instructions. For example, if data were being read from a 24-bit module, the program sequence would be

```
.
.
LAC   COMND    /get command to read 24 bits
LCR
RHD            /read high data from CAMAC dataway
CLL!RAR        /clear link and rotate AC right
               /(gets rid of bit 7)
DAC   HIBITS   /deposit accumulator (at HIBITS)
RLD            /read low data from dataway
DAC   LOBITS   /store low data bits too
.
.
```

On execution of the RHD IOT instruction, the seven high-order bits of the 24-bit CAMAC dataway enter the seven low-order bits of the accumulator and the remaining accumulator bits are set to 0. Upon execution of the RLD IOT-instruction, the 18 low-order bits of the dataway enter the accumulator. Thus, the seventh bit of the dataway is read twice. The CLL!RAR instruction in the preceding example is used to get rid of the redundent bit and close the gap so that the two memory lo-

cations, HIBITS and LOBITS, together represent a 24-bit image of the module data. When data are transmitted from the computer to a module, the six low-order bits of the accumulator enter the six high-order dataway lines on execution of the LHD IOT instruction. No shifting is needed.

If the function code specifies testing a module flag, the flag status is first transferred to the high-bit position of the command register. Its presence there can then be tested by the SQF IOT-instruction:

```
      .
      .
LAC  COMND  /get command to test module's flag
LCR
SQF         /skip if Q-flag set
XX          /instruction skipped if flag is set
      .
      .
```

Special all-module commands. - Four special module commands, which address all modules at once, have been included in our CAMAC command repertoire. They are listed in table III. The ZGEN command generates a CAMAC initialize signal (Z), which resets all module registers and disables all module LAM flags. The CGEN command generates a clear-modules signal (C), which does not disable the LAM flags. The SETINH and CLRINH module commands are used when it is desired to control a group of modules such as scalers. (Not all CAMAC modules respond to the inhibit line.)

Interrupt Programming

In this section we consider those parts of a program that arrange for and process interrupts. For convenience, these parts are usually grouped together as a subprogram that can then be thought of as an extension of the hardware with which it associates. In a CAMAC system, the hardware device is a module. Since more than one interrupting module may be needed in a program, it is desirable to have subprograms written so that they do not interact with one another. That way, a variety of subprograms can easily be brought together to do a particular job. This independence is made possible by a system program called CAMAC.. (A listing of CAMAC. is given in appendix B.)

LAM interrupt processing. - At the time an interrupt is honored by the computer, the interrupting device sends a trap address to the CPU - an address which is unique to that device. The memory cell corresponding to the trap address contains the first instruction of the program that services the interrupt.

The CIOP sends the same trap address for all look-at-me (LAM) interrupts. Before any modules are enabled to cause interrupts, a general purpose interrupt routine must exist and its first instruction must be put into the trap location. The system program CAMAC. satisfies these requirements.

The interrupt-level programming in CAMAC. takes care of those program steps needed by all interrupt-processing programs:

(1) Saves computer registers

(2) Saves CAMAC command and high-data registers

(3) Enters Q-test skip chain

(4) Restores CAMAC and computer registers

(5) Returns to interrupted program.

The saving and restoring of all registers allows interrupt-servicing programs to use any of the system hardware (computer or CAMAC) without losing data needed to resume an interrupted program. On the other hand, interrupt-servicing programs cannot use any subprograms that might also be in use by an interrupted program.

The Q-test skip chain is the sequence of module tests that is needed to find which module has raised its flag (Q). For each module that can cause an LAM interrupt, there must be a link in the Q-test skip chain of the form:

```
ENTER LAC  QTEST     /get command to test module's flag
      LCR
      SQF            /skips to BEGIN if flag is up
EXIT  JMP* NEXT      /-- or jumps to next link in chain
BEGIN .              /begin servicing interrupt
      .              /from this module
      .
      .
      JMP* RETURN    /Return to CAMAC.
```

These links test the modules, one at a time. When the module is found, the corresponding interrupt-servicing program is begun.

LAM initialization. - When a module subroutine is first used, its Q-test link must be put into the chain. An initializing entry has been provided in the CAMAC. system program to do this. Typically, a module subroutine will itself have an initializing section entered at mainstream level for the purpose of setting up addresses that will later be used at interrupt level. This initializing section should be of the form:

```
SUB0    0                   /initializing entry to subprogram
        JMS*  .DA           /jump to subroutine for arguments
        JMP   INIT1
ARG1    0                   /argument addresses
ARG2    0                   /(as many as needed)
INIT1   JMS*  CAMAC.        /call to CAMAC.
NEXT    ENTER               /address of Q-test link
RETURN  0                   /address of CAMAC. return
        LAC   (JMP INIT2)                          <
        DAC   INIT1                                <
INIT2   .
        .
```

The first five lines in this example are the standard sequence for subroutines called from FORTRAN-IV programs. When CAMAC. is entered, it gets the 15-bit address ENTER of the Q-test link from the location labeled NEXT and overwrites it with the 15-bit address of the next link in the chain. CAMAC. also fills in the return address to be used by all of the LAM interrupt-servicing routines. This return address points to that section in CAMAC. which restores the computer and CAMAC registers and returns to the interrupted program. The two lines tagged "<" destroy the call to CAMAC. (Each subprogram may call CAMAC. only one time.)

The Q-test chain is formed as the modules are initialized. If a priority schedule is desired, modules should be initialized in order of highest priority. Since only one module is processed at each interrupt, a module whose Q-test link is at the head of the chain can prevent the processing of other modules if its LAM flag keeps coming up. (No provisions have been made for re-ordering the Q-test chain or removing links. However, a link can be effectively disabled by replacing the ENTER instruction with the instruction labeled EXIT.)

CAMAC. initializes itself the first time it is called by a module subroutine. This includes setting up the interrupt trap address, executing the ZGEN module-clearing command, and enabling LAM interrupts. Control of module interrupts after this should be through the enable and disable commands to individual modules.

Mainstream control. - Module subprograms that service interrupts should also have entry points for starting up and shutting down data transfers. Starting up may consist simply of enabling a module or it may involve setting up a data buffer and transferring addresses. Shutting down is done by disabling the module LAM. An example of a simple module subprogram having all of the parts described is given in appendix C.

Data-Channel Programming

The stored-program CAMAC channel processor (SPCC) makes it possible to transfer data between CAMAC modules and the computer memory without using the CPU for each transfer. Module commands are still stored in the computer memory but in this case they are obtained directly by data-channel requests from the SPCC. IOT-instructions are not required; all of the control needed to transfer data to or from a data buffer (a block of computer memory) is provided by the SPCC.

Command decoding by SPCC. - Module commands processed by the SPCC have the same format as those used by the CIOP (refer again to fig. 2(a)). Crate address, module address, subaddress, and function code bits are arranged the same. However, certain function codes are given nonstandard meanings. Also, three bits not used in module commands processed by the CIOP are used specifically to control the SPCC. The additional information contained in module commands for the SPCC is needed because no IOT instructions are used.

Module commands that are to be processed by the SPCC are stored as a channel program in the computer memory in the order that they are to be executed. Execution begins when an external event signals the SPCC. Four event inputs are provided, each one with its own wired-in event address. The event address points to a cell in the computer memory that contains the starting address of a channel program. Execution of the module commands in the channel program proceeds in

order until a special code (EXIT) is encountered, at which time data-channel transfers stop.

Every module command in a channel program is treated by the SPCC as a skip command, depending on the Q-response from the module addressed. The Q-bit code (fig. 2(a)) in the module command is compared with the Q-response from the module: if they differ, the next command in the channel program is skipped. Whenever a skip occurs, bit 15 of the status register (described in a following section) is set and remains set until the next event is processed.

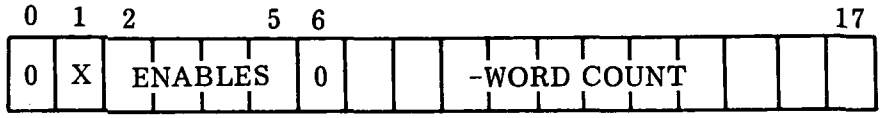
Ordinarily, when a read command is executed by the SPCC, the 17 low-order bits from a module are transferred to memory. (18 bits can be read if bit 1 of the word-count/enable register has been set. This is explained in the following sections.) If the H-bit is set in the module command (fig. 2(a)), the SPCC will transfer two data words to memory. The first holds the seven high-order bits of 24 and the next holds the 17 low-order bits. On write commands, 18 bits are transferred from memory to module if the H-bit is not set. With the H-bit set, the contents of two memory cells are transmitted. The six low-order bits from the first memory cell enter the six high-order bits of the module register; the 18 low-order bits of the module are filled from the second memory cell.

The ADD-1 function codes, F(4) and F(6), are used as modified module-read commands. They signal the SPCC to use the data from a module as a memory address in an increment-memory operation on a cell in the computer memory. (Overflows are ignored.)

Function code F(12) is used by the SPCC to get the address of its next module command when that command is out of sequence. The 12 low-order bits of this command replace the corresponding bits of the SPCC program counter. It allows jumps to any place in the same page of the computer's memory.

SPCC control module. - As was mentioned previously, each event (maximum of four) is associated with an event address and hence with a channel program. Each event is also associated with one of two sets of SPCC control registers (subchannels). The following is a description of the registers found in each of these two subchannels. These registers respond to module commands just like standard CAMAC modules. Table V lists the command repertoire of the SPCC control module. The first group of these module commands (LDCA0 through FLAG1) are the ones that normally appear in user programs. They provide the means of controlling the SPCC.

The current-address register of the SPCC contains the 15-bit address of the next memory cell in the user's data buffer. This register is initially loaded (using the CIOP) with the address of the first word in the buffer. When the user program is ready to let data begin flowing through one of the SPCC channels, the *word-count/enable register for that channel is loaded with a word having the* format:



Loading this word into the word-count/enable register enables events 0 to 3 corresponding to the enable bits 2 to 5 that are set. These events can then start the SPCC when they occur. During the processing of an event, data are transferred to or from the data buffer. At each transfer, the word-count register and the current-address register are incremented by one. If the word count overflows into bit position 6 during any of these transfers, the enable bits are cleared and a flag (end of record) is set to cause an interrupt at the end of event processing. The initial value loaded into the word count should therefore be the 11-bit twos-complement of the number of data words in the data buffer. In the case where events can give rise to input data strings of variable length, the word count should contain an allowance for the maximum number of data words that might be put into the data buffer by one event.

Bit 1 of the word-count/enable register (filled with X in the example) is used to enable transfer of all 18 low-order bits from the CAMAC dataway to the computer memory.

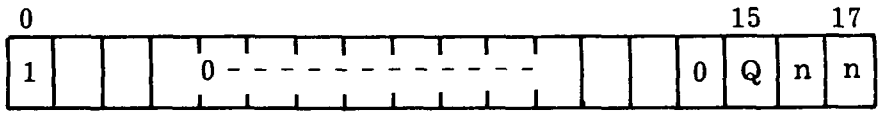
Either of the SPCC subchannels can be stopped from mainstream level by the SPCC-module commands FLAG0 and FLAG1. These commands have the same effect as a word-count overflow; the enable bits are cleared and the end of record (EOR) flag is set. The SPCC then causes an end-of-record interrupt to occur, transferring program control to the interrupt-servicing program.

EOR interrupt processing. - When an EOR interrupt occurs, control transfers to the CAMAC. system program just as with LAM interrupts. CAMAC. saves all of the volatile hardware registers and then transfers control to the user's interrupt-servicing program for the interrupting subchannel. The simplest interrupt servicing program could be no more than a status word to signal the mainstream program that a data buffer has been filled or emptied. It would then

be up to the mainstream program to unpack the data buffer or to load it up as the case may be.

If high input-data rates are expected, the use of two data buffers in a "ping-pong" arrangement is desirable. When an EOR interrupt occurs, the SPCC can be reset to resume running with the current address of an empty data buffer. (The operation of the SPCC is not affected by the current CPU interrupt level. Once a subchannel is enabled, data flows without interruption until it's EOR flag goes up and the subchannel is stopped.) Then, still at interrupt level, the just-filled data buffer can be unloaded. When all of the data have been processed from that buffer, control returns to the interrupted program until the new buffer has been filled.

If more than one kind of event can send data to a buffer, some kind of tag may be necessary to identify the beginning of each data group in the buffer. This can be accomplished by using the SPCC-module command STAT (table IV) as the first command in the channel program for each event. When the SPCC processes the STAT command, a word of the form



is read as an event descriptor into the next cell of the data buffer. Bits 16 and 17 of the status register specify the event. Bit 15 is set whenever a channel-program skip occurs as a result of a difference between the QBIT code of a module command and the Q-response received from the module. The high-order bit of the status register (bit 0) is always read, even when the channel is operating in the 17-bit mode. These status words can be easily identified, therefore, if all of the other data in the buffer had been read in without the high bit.

SPCC initialization. - When an SPCC subchannel is first used, it is necessary to perform an initializing procedure. For this purpose, an entry in CAMAC. has been provided which should be called from the SPCC-using subprogram by the sequence:

```
        .
        .
        JMS*  CAMAK. /SPCC initialization entry
        JMP   NEXT
        ENTRY        /address of handler's entry
        CHANL        /SPCC subchannel (0 or 1)
        EVENTS       /first event address
NEXT    .
        .
```

This sequence tells CAMAC. which subchannel is involved, the entry point(s) to the interrupt-servicing routine for this subchannel, and which of the four channel-program pointers are to be written into the event addresses. Calls to CAMAK. can be made any time it is necessary to change the channel-program pointers or interrupt-servicing routine in either of the subchannels.

EVENTS must be the address of the first of a block of four consecutive pointers. If a particular event is to be associated with the subchannel being initialized, the address of its first module command should be in the corresponding element of the pointer block. An address of zero is used to mark events not associated with the channel being initialized. At the start of an event, the SPCC loads one of these channel-program pointers (corresponding to the event number) into a program counter. Subsequent module commands are then obtained from computer memory addressed by this program counter. If only one module command is needed to process an event, it is more efficient if the SPCC uses the content of the event address as a command instead of the address of its first command. The SPCC has this capability. If the high-order bit of the data in the event address is set, then the SPCC uses the data as a module command instead of a memory address.

Mainstream control of channels. - Each subprogram that uses a CAMAC channel should include provisions for starting and stopping the flow of data through a channel without producing data errors and without interferring with the operation of the other channel.

An example of a subprogram that contains most of these elements is given in appendix D. In this example, data are entered in a single module, one word per event. The LAM line from this module is used to initiate event 0 in the SPCC, using subchannel 0. Direct control of events from mainstream is by the use of enable and disable commands to the module.

Mainstream control of data-buffer processing is achieved by using the SPCC-module command FLAG0 (table III) to cause an EOR interrupt. Buffer processing then proceeds at interrupt level. Partially full (or even empty) buffers are processed in the same way as a full buffer that results from a normal word-count overflow.

In the example, a status word (STATUS) is used to keep track of whether the subchannel was enabled or not when the interrupt occurred. Another status word (START), set by the mainstream section, tells the interrupt-servicing program whether or not the subchannel is to be enabled.

One last point: the input module is first cleared before enabling it so that an event that may have occurred beforehand is not included as the first datum. The module is disabled before final buffer processing so that no "suspended events" remain in the event latch (fig. 10). If the module were allowed to flag an event, the event latch would be set even though the channel was disabled. Later, when the channel was enabled (by loading the word-count/enable register) processing of the event would begin even though by then there might no longer be valid data in the module register.

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, January 15, 1973,
503-10.

APPENDIX A

SPCC HARDWARE

This appendix describes the details of the SPCC which has been built for the PDP-15.

Data Buffer and Command Registers

These registers serve the same roles as in the CIOP. The two processors share the same low data register (LDR), but have separate command (CR) and high (HD) data registers. The LDR can be shared because it never holds information for more than one input-output cycle.

Dataway Clock

The clock is shared with the CIOP and has the same start conditions; that is, upon loading the command register for dataless transfers or upon transferring the low order data to or from the computer.

Program Counter

This is a 15 bit register that specifies the memory address of the next command to be executed by the SPCC. It consists of a 12-bit incrementing register and a three-bit page register. The program counter is normally loaded from a memory event address each time an event is processed and is incremented each time a command is loaded into the command register. The conditional skip feature is provided by making an increment at S1 time in the dataway cycle if the Q-response does not agree with the Q-bit of the command causing the cycle. The lower 12 bits of the PC can be loaded under SPCC program control to provide a program jump. A command containing the function code F(12) loads its low-order 12 bits into the PC instead of the CR. There is no dataway cycle.

Control Module

The access to most of the flags and registers used to control the SPCC is through the dataway as though they are contained in a module at N(23), C(0). This allows the SPCC a control of itself which would not be possible if the module

was directly on the input-output bus. The module is accessible to both the SPCC and the CPU and thus serves a communication link between different levels of programming.

Contained in the control module are the word-count/enable and current-address registers, the status register, and the error and word count overflow flags. Two independent subchannels are provided, each having its own word-count/enable and current address registers. A summary of its command set is given in table III.

Channel LAM's. - The command module's LAM directly drives the PI and API interrupt facilities of the PDP-15 and is provided with its own API trap address (70) at priority 2. It is controlled by the SPCC enable using the ESPCC and DSPCC IOT's. There are three sources of LAM requests: the word count overflow (or end-of-record) flags from the two subchannels and an error flag. These flags may be tested through their Q-responses. The WC overflow flags are of course set by the overflow of their respective word count registers but they may also be set through the dataway (using F25)) to force the processing of an incomplete buffer. Since the overflow flags also disable their respective subchannels, they are actually raised at the completion of the processing of an event to prevent any loss of data.

The error flag acts differently. It is raised if the PC overflows or if there is no exit bit in a single command program, which takes its command directly from one of the four event addresses. The error flag immediately freezes the status of the processor to allow a diagnosis of the error.

Current address registers (CA). - Each subchannel contains a 15-bit register that is used to specify the address to or from which data are transferred. Each one consists of a 12-bit incrementing register and a three-bit page register. They can be loaded using function code F(1), but they cannot be read or cleared. Subchannels 0 and 1 have subaddresses 0 and 1, respectively.

Word-count/enable registers (WC). - Each subchannel contains a two-part register composed of a 12-bit WC register and a five-bit enabling register. The two are packed into one word to shorten interrupt handling. The WC register is automatically initialized when the subchannel is enabled. Bits 02-05 enable event inputs 0 to 3, respectively, and bit 01 enables the most significant read line, R18. In addition to turning on the event input, setting an enable bit also serves to associate the event with the subchannel of the enable. Therefore, a given event

should be enabled by one subchannel at a time. The R18 enable applies to all events associated with its subchannel. Its use will be explained in the next section. The WC register is an incrementing register zhat must be loaded with the 2's complement (12 bit) of the buffer size. When it overflows, it will inactivate its subchannel by clearing its enable register, but only after the current event has been processed. Therefore, the actual dimension of a buffer must exceed the word count by the size of the largest event minus one. The subaddresses of the WC/enable registers for the two subchannels are 2 and 3. They are loaded using F(17) and read using F(1). The enable-register portion can not be read (it yields zeros), but it can be cleared using F(11).

Status register. - When two or more events share a buffer on input, it is usually necessary to label each event in order to guide the program processing the buffer. The control module provides a status register for this purpose that can be read using F(1),A(4). The module places a "1" on the R18 line and the event code (0-3) on R1 and R2. R3 is controlled by a flip flop, which is reset at the start of each event and which is set whenever there is a Q skip (i.e., the Q-response differs from the value in the MSB of the command). Thus the status register shows that a program branch has occurred and allows the definition of subevents. The "1" placed on the R18 line is used to identify the status word, the word created in memory when the status register is read. It becomes a unique tag if the R18 line is inhibited on all other read operations by the enable register.

LAM Patch Panel (LPP)

All of the LAM lines of crate zero are brought into the LAM patch panel (LPP) where they may be connected either to the automatic interrupt facility or to one of the four event lines. All of the API inputs are equivalent and may be used independently. The LPP is located on the front panel of the control station in crate 0. The layout of the LPP, a 50-pin AMP connector, is shown in figure 9.

External Event Lines

Four BNC connectors are provided on the front panel of the control station of crate 0. The inputs accept 3.5- to 8-volt positive pulses 0.5 to 6.0 microsecond long.

Event Monitor

All events or requests for the SPCC first pass through an event monitor where they are held until they can be processed. Event 0 is accorded the highest priority and event 3 the lowest. The highest priority event pending always takes precedence over all others. The event monitor consists of the following elements shown in figure 10.

Event active lights. - Four lights are provided on the front of the crate zero control station to indicate when events are being processed.

Event latches. - These four S-R flip flops serve to hold event requests until they can be processed. They may be cleared by issuing the command F(10)A(0) C(0)N(23) or an initialize (Z). An event latch is reset after its request has been serviced.

Event gates. - The outputs of the event latches pass through the event gates. A gate is open only if the event is enabled and the SPCC is not busy.

Event register. - The outputs of the event gates are stored by this four-bit register. The entire register is cleared after an event is processed.

Priority ladder. - This netword insures that only one event is active at a time. Only the highest priority request in the event register passes through it.

Event address generator. - While an event is active, this generates the address in memory (24 to 27 octal, respectively, for events 0 - 3) from which the program counter will initially be loaded. The four locations are referred to as the event table.

Address Multiplex

The address multiplex provides 15 bits of data for the input-output address lines of the PDP-15 and thus specifies the location in core to or from which data are transferred. Inputs to the multiplexer are the event address generator, the program counter, the two current address registers, and the R1 to R15 read lines of the dataway.

Data Channel Control

This unit provides the logic necessary to operate the PDP-15 data channel. It interprets the command, selects the mode of operation, (input, output, or memory

increment), gates data to the proper register, and controls the input-output address multiplexer.

There are two separate ways of turning event processing on and off: the event enables controlled through the dataway and the channel enable controlled by the IOT instructions ESPCC and DSPCC and the PDP-15 input-output power clear. While these appear to have the same function, there are significant differences that must be recognized to assure proper operation. The event enables are the primary control since they have been designed to prevent any loss of data that might be caused by a WC overflow or programmed stop occurring during a SPCC transfer. They turn off their subchannel only between the processing of events. The action of the channel enable, on the other hand, is immediate and may cause the loss of data.

Timing and Latency

The operational speed of the SPCC is entirely determined by the speed of the single cycle data channel. The channel operates in two modes: normal (or asynchronous) and burst (or synchronous). In the normal mode, data transfers require three memory cycles per word: in the burst mode the first requires three cycles but subsequent transfers take only one cycle per word. The SPCC uses burst mode only when a 24-bit word is transferred. Therefore, fetching a command by the processor takes three microseconds while its execution takes one microsecond per six bits of data transferred. This means that the SPCC operates at approximately one fourth the speed of the CPU. In addition to these times, programs having more than one command require an additional three microseconds to load the PC.

The data channel stands at the top of the PDP-15's priority hierarchy, and care must be taken that the SPCC does not shut out other vital operations. Short latency devices such as disk memories should be placed closer to the computer on the input-output bus to give them higher priority. Since the processing of an event can be quite lengthy when many modules must be read, breathing space has been provided for lower priority operations by restricting SPCC transfers to one every eight microseconds. This means that the processing of a complicated event can be quite lengthy. Because one event shuts out others until its completion, care must be taken if any event has short latency requirements. The rate restriction on

SPCC transfers can be eliminated if there is no other solution, but it has been found that the clock synchronization in the PDP-15 is severely strained by the SPCC operating at full speed and the computer's input-output clock adjustments must be made carefully to assure reliable operation.

APPENDIX B

ASSEMBLY LANGUAGE LISTING OF CAMAC. SYSTEM PROGRAM

```
        .TITLE  CAMAC.
/
/CAMAC SYSTEM PROGRAM
/
        .GLOBL  CAMAC.   /LAM INITIALIZATION
        .GLOBL  CAMAK.   /SPCC INITIALIZATION
        .GLOBL  ECAMCK   /RE-ENABLE FOR RESTART
        .GLOBL  .DA      /EXTERNAL SUBROUTINE

/CAMAC IOT-INSTRUCTIONS
/
LCR=706004    /LOAD COMMAND REGISTER
RLD=706132    /READ LOW DATA BITS
SQF=706001    /SKIP ON Q-FLAG
RDCR=706112   /READ COMMAND REGISTER
RHR=706172    /READ HIGH DATA REGISTER
LHD=706044    /LOAD HIGH DATA REGISTER
LON=706101    /ENABLE LOOK-AT-ME
LOF=706104    /DISABLE LOOK-AT-ME
ESPCC=706161 /ENABLE SPCC
SLF=706021    /SKIP ON LOOK-AT-ME FLAG
SEORF=706061 /SKIP ON END-OF-RECORD FLAG

/MODULE COMMAND CODES
/
F=010000      /VALUE OF F1 (FUNCTION CODE)
A1=000400     /VALUE OF A1 (SUBADDRESS)
A3=3*A1       /VALUE OF A3 (SUBADDRESS)
A4=4*A1       /VALUE OF A4 (SUBADDRESS)
ZGEN=100034   /CLEAR ALL MODULES
```

```
/SPCC MODULE COMMANDS
/
N=000027      /SPCC MODULE ADDRESS
STOP=400000   /FORCES AN ERROR FLAG
         .DEC
STAT=F*1+A4+N          /READ STATUS REGISTER
SENSE0=F*10+A1+N       /SENSE AND CLEAR SUBCHANNEL 0 FLAG
SENSE1=F*10+A3+N       /SENSE AND CLEAR SUBCHANNEL 1 FLAG
SERROR=F*8+N           /SENSE ERROR FLAG
         .OCT

/MISCELLANEOUS CONSTANTS
/
ERR=4          /ERROR REPORT ROUTINE ENTRY ADDRESS
TRAP1   24     /FIRST OF 4 EVENT TRAP ADDRESSES


/ENTRY FOR INITIALIZING LOOK-AT-ME INTERRUPTS
/
CAMAC.  0
        LOF              /DISABLE L-FLAG INTERRUPTS
        JMS  INIT        /INITIALIZE CAMAC SYSTEM
        LAC* CAMAC.      /ENTER MODULE INTO SKIP CHAIN:
        DAC* LAST        /SAVE ADDRESS OF MODULE LINK
        LAC  CAMAC.      /THIS MODULE'S ADDRESS
        DAC  LAST        / SAVED FOR NEXT TIME
        LAC  (NOFLAG)    /ADDRESS OF END OF SKIP CHAIN
        DAC* CAMAC.      / REPLACES MODULE'S LINK ADDRESS
        ISZ  CAMAC.      /INCREMENT ARGUMENT ADDRESS POINTER
        LAC  (CMCNT1)    /GIVE ADDRESS OF RETURN
        DAC* CAMAC.      /  TO INTERRUPT HANDLER
        ISZ  CAMAC.      /INCREMENT ARGUMENT ADDRESS POINTER
        LON              /RE-ENABLE L-FLAG INTERRUPTS
        JMP* CAMAC.      /RETURN TO CALLING PROGRAM
```

```
FIRST   NOFLAG          /ADDRESS OF START OF SKIP CHAIN
LAST    FIRST           /ADDRESS OF LAST LINK IN SKIP CHAIN


/ENTRY FOR SPCC SUBCHANNEL INITIALIZATION
/
CAMAK.  0
        JMS* .DA        /GET ADDRESSES OF ARGUMENTS
        JMP  .+4
PROG    0               /ADDRESS OF SERVICING PROGRAM
CHANL   0               /SUBCHANNEL NUMBER (0 OR 1)
EVENTS  0               /BLOCK OF 4 CHANNEL-PROGRAM POINTERS
        JMS  INIT       /INITIALIZE CAMAC SYSTEM
        LAC* CHANL      /GET SUBCHANNEL NUMBER
        SZA             / IF NOT 0,
        CLA!IAC         / ASSUME 1
        PAX
        LAC  PROG       /GET ADDRESS OF SERVICING PROGRAM
        DAC  USERS,X    /ENTER INTO BUFFER INTERRUPT TABLE
        CLX             /INITIALIZE INDEX AND
        LAC  (4)        /  LIMIT REGISTERS
        PAL
CAMAK1  LAC* EVENTS,X   /SET UP CHANNEL-PROGRAM POINTERS
        SZA
        DAC* TRAP1,X
        AXS  1          /INCREMENT INDEX REGISTER
        JMP  CAMAK1     /LOOP AGAIN
        JMP* CAMAK.     /RETURN TO CALLING PROGRAM
/ENTRY TO RE-ENABLE CAMAC FOR RESTART FROM DUMP
/
ECAMCK  0
        JMS  INIT       /MAKE SURE IT'S INITIALIZED
        JMS  XZGEN      /CLEAR ALL MODULES
        LON             /ENABLE L-FLAG INTERRUPTS
```

```
        ESPCC           /ENABLE SPCC TRANSFERS
        JMP* ECAMCK     /RETURN
/ONCE-ONLY SUBROUTINE TO INITIALIZE CAMAC SYSTEM
/
INIT    0
        JMS  XZGEN      /CLEAR ALL MODULES
ACSAVE  CAL  70         /INITIALIZE API CHANNELS
CRSAVE  16
HRSAVE  SLF
XRSAVE  CMCINT
LRSAVE  CAL  71         /(THESE ALSO USED AS TEMP STORAGE)
MQSAVE  16
SCSAVE  SEORF
SGSAVE  CMKINT
USER    DBK             /DEBREAK FROM CAL LEVEL
TEMP    CLX             /INITIALIZE INDEX AND
        LAC  (4)        /  LIMIT REGISTERS
        PAL
        LAC  (STOP)     /INITIALIZE CHANNEL-PROGRAM POINTERS
INIT1   DAC* TRAP1,X    /  TO TRAP UN-INITIALIZED EVENTS
        AXS  1          /INCREMENT INDEX REGISTER
        JMP  INIT1      /GO BACK
        ESPCC           /ENABLE SPCC
        LAC  INIT2      /OVERLAY ONCE-ONLY INITIALIZATION
        DAC  INIT+1
INIT2   JMP* INIT       /RETURN


/SUBROUTINE TO EXECUTE ZGEN MODULE COMMAND
/
XZGEN   0
        LAC  (ZGEN)     /INITIALIZE ALL MODULES
        LCR
        RDCR            /CHECK OPERATION OF CIOP
```

```
        SAD   (ZGEN)
        JMP*  XZGEN       /COMMAND REGISTER OK
        LAC   (4)         /COMMAND REGISTER NOT OK
        JMS*  (ERR-1)     /.IOPS4 ERROR MESSAGE
        JMP   XZGEN+1     /TRY AGAIN WHEN READY


/HANDLER FOR LAM INTERRUPTS
/
CMCINT  0
        JMS   SAVE        /SAVE REGISTERS
        JMP*  FIRST       /ENTER Q-TEST CHAIN
CMCNT1  JMS   UNSAVE      /RESTORE REGISTERS
        DBR               /DEBREAK AND RESTORE
        JMP*  CMCINT      /RETURN TO INTERRUPTED PROGRAM


/HANDLER FOR EOR INTERRUPTS
/
CMKINT  0
        JMS   SAVE        /SAVE REGISTERS
        CLX               /INITIALIZE INDEX AND
        LAC   (3)         /  LIMIT REGISTERS
        PAL
CMKNT1  LAC   TEST,X      /GET FLAG-TEST COMMANDZ
        LCR
        SQF               /TEST THE FLAG
        JMP   CMKNT2
        LAC   USERS,X     /GET HANDLER ENTRY ADDRESS
        DAC   USER
        JMS*  USER        /ENTER USER'S PROGRAM
        JMS   UNSAVE      /RESTORE REGISTERS
        DBR               /DEBREAK AND RESTORE FROM INTERRUPT
        JMP*  CMKINT      /RETURN TO INTERRUPTED PROGRAM
```

```
CMKNT2  AXS  1
        JMP  CMKNT1     /GO BACK


/ERROR-REPORTING SECTION
/
NOFLAG  LAW  51         /ERROR:  NO FLAG FOUND
        JMP* (ERR)
ERRFLG  0
        LAC  (STAT)     /READ STATUS REGISTER
        LCR
        RLD
        DAC* (ERR-1)    /AND SAVE IT FOR REPORTING
        AND  (3)
        TAD  TRAP1      /CHECK EVENT TRAP
        DAC  TEMP       /FOR STOP CODE
        LAC* TEMP
        SAD  (STOP)
        JMP  NOINIT
        LAW  52         /ERROR: HARDWARE ERROR FLAG
        JMP* (ERR)
NOINIT  LAW  53         /ERROR: UN-INITIALIZED EVENT
        JMP* (ERR)
ADDERR  0
        LAW  54         /ERROR: NO HANDLER FOR CHANNEL
        JMP* (ERR)


/TABLE OF EOR-INTERRUPT FLAG TESTS
/
TEST    SENSE0          /SUBCHANNEL 0
        SENSE1          /SUBCHANNEL 1
        SERROR          /ERROR
```

```
/TABLE OF USER-SUPPLIED, EOR-SERVICING PROGRAMS
/
USERS   ADDERR          /SUBCHANNEL 0
        ADDERR          /SUBCHANNEL 1
        ADDERR          /ERROR
/SUBROUTINE TO SAVE REGISTERS
/
SAVE    0
        DAC  ACSAVE     /ACCUMULATOR
        LACQ
        DAC  MQSAVE     /MQ REGISTER
        LACS
        DAC  SCSAVE     /EAE STEP COUNTER
        CLA!CLL!IAC
        MUL
        1
        DAC  SGSAVE     /EAE SIGN BIT
        PXA
        DAC  XRSAVE     /INDEX REGISTER
        PLA
        DAC  LRSAVE     /LIMIT REGISTER
        RDCR
        DAC  CRSAVE     /CIOP COMMAND REGISTER
        RHR
        DAC  HRSAVE     /CIOP HIGH DATA REGISTER
        JMP* SAVE       /RETURN


/SUBROUTINE TO RESTORE REGISTERS
/
UNSAVE  0
        LAC  HRSAVE
        LHD             /CIOP HIGH DATA REGISTER
        LAC  CRSAVE
```

```
AND   (677777)    /REMOVE F8 BIT
LCR               /CIOP COMMAND REGISTER
LAC   LRSAVE
PAL               /LIMIT REGISTER
LAC   XRSAVE
PAX               /INDEX REGISTER
LAC   SGSAVE
ABS!ECLA          /EAE SIGN BIT
AAC   77
XOR   SCSAVE
TAD   (640402)
AND   (640477)
DAC   .+1
XX                /EAE STEP COUNTER
LAC   MQSAVE
LMQ               /MQ REGISTER
LAC   ACSAVE      /ACCUMULATOR
JMP*  UNSAVE      /RETURN
.END
```

APPENDIX C

AN EXAMPLE OF AN INTERRUPT-LEVEL MODULE SUBPROGRAM

```
        .TITLE  ADC1

/ADC IN ADD-1 MODE THROUGH A CAMAC MODULE WITH
/LOOK-AT-ME INTERRUPTS AT EACH EVENT.

        .GLOBL  ADC0    /INITIALIZE
        .GLOBL  ADCON   /START ACCUMULATING
        .GLOBL  ADCOFF  /STOP ACCUMULATING
        .GLOBL  CAMAC.,.DA /EXTERNAL SUBPROGRAMS

/CAMAC IOT-INSTRUCTIONS
LCR=706004      /LOAD COMMAND REGISTER
RLD=706132      /READ LOW DATA BITS
SQF=706001      /SKIP ON O-FLAG

/ADC MODULE COMMANDS
F=10000         /VALUE OF F1 (FUNCTION CODE)
        .DEC
MOD=11          /ADC MODULE ADDRESS
SADC=F*8+MOD    /SKIP ON ADC FLAG
EADC=F*26+MOD   /ENABLE ADC LOOK-AT-ME
DADC=F*24+MOD   /DISABLE ADC LOOK-AT-ME
RADC=F*0+MOD    /READ AND CLEAR ADC
        .OCT
ADC0    0               /INITIALIZATION ENTRY
        JMS*    .DA     /F4 GET ARGS SUBROUTINE
        JMP     INIT
ARG1    0               /ADDRESS OF 1-ST CHANNEL
INIT    JMS*    CAMAC.  /LAM INITIALIZING CALL TO CAMAC.
NEXT    ENTER           /WILL POINT TO NEXT IN SKIP CHAIN
```

```
RETURN  0                   /WILL BE RETURN FROM INTERRUPT
        LAC     EXIT0
        DAC     INIT        /CAMAC. CAN BE CALLED ONLY ONCE
EXIT0   JMP*    ADC0

ADCOM   0                   /START DATA ACCUMULATION
        LAC     (RADC)      /READ MODULE TO CLEAR IT
        LCR
        RLD
        LAC     (EADC)      /AND THEN ENABLE IT
        LCR
        JMP*    ADCON

ADCOFF  0                   /STOP DATA ACCUMULATION
        LAC     (DADC)      /DISABLE THE MODULE
        LCR
        JMP*    ADCOFF

ENTER   LAC     (SADC)      /TEST MODULE'S FLAG
        LCR
        SQF                 /IF FLAG IS UP, BEGIN PROCESSING
        JMP*    NEXT        /--OTHERWISE GO TO NEXT LINK

BEGIN   LAC     (RADC)      /READ THE CHANNEL NUMBER FROM MODULE
        LCR
        RLD
        TAD*    ARG1        /ADD ON ADDRESS OF CHANNEL 1
        DAC     ADDR        /AND SET THE MEMORY ADDRESS
        ISZ*    ADDR        /ADD-1 TO MEMORY ADDRESS
        JMP*    RETURN      /AND RETURN TO INTERRUPTED PROGRAM
        JMP*    RETURN      /(DISREGARDING OVERFLOWS)
ADDR    0
        .END
```

APPENDIX D

EXAMPLE OF SUBPROGRAM THAT USES SPCC AND "PING-PONG" BUFFERS

```
        .TITLE  ADC2

/ADC IN ADD-1 MODE THRU A CAMAC MODULE USING
/SPCC SUBCHANNEL 0.

        .GLOBL  ADC0    /INITIALIZE
        .GLOBL  ADCON   /START ACCUMULATING
        .GLOBL  ADCOFF  /STOP ACCUMULATING
        .GLOBL  CAMAK.,.DA /EXTERNAL SUBPROGRAMS

/CAMAC IOT-INSTRUCTIONS
LCR=706004      /LOAD COMMAND REGISTER
RLD=706132      /READ LOW DATA BITS
LLD=706024      /LOAD LOW DATA BITS

/MODULE COMMAND CODES
F=010000        /VALUE OF F1 (FUNCTION CODE)
A1=000400       /VALUE OF A1 (SUBADDRESS)
EXIT=000200     /FLAGS END OF CHANNEL PROGRAM

/SPCC MODULE COMMANDS
N=000027        /SPCC MODULE ADDRESS
        .DEC
LDCA0=F*17+N     /LOAD CURRENT ADDRESS
LDWC0=F*17+A1+N  /LOAD WORD-COUNT, ENABLE REGISTER
RDWC0=F*17+A1+N  /READ WORD-COUNT REGISTER
FLAG0=F*25+A1+N  /SET END-OF-RECORD FLAG

/ADC MODULE COMMANDS
MOD=11          /ADC MODULE ADDRESS
```

```
EADC=F*26+MOD    /ENABLE ADC TO MAKE TYPE 1 EVENTS
DADC=F*24+MOD    /DISABLE ADC
RADC=F*0+MOD     /READ AND CLEAR ADC
         .OCT

ADCO     0                /INITIALIZATION ENTRY
         JMS*     .DA     /F4 GET ARGS SUBROUTINE
         JMP      .+2
ARG1     0                /ADDRESS OF DATA ARRAY
         JMS*     CAMAK.  /SPCC-INITIALIZING CALL
         JMP*     ADC0
         ENTRY            /ADDRESS OF INTERRUPT SERVICING
         (0)              /SUBCHANNEL 0 INDICATOR
         EVENTS           /FIRST OF 4 CHANNEL-PROGRAM POINTERS
ADCON    0                /START DATA ACCUMULATION
         LAC      (RADC)  /READ MODULE TO CLEAR IT
         LCR
         RLD
         LAC      (FLAG0) /SET END-OF-RECORD FLAG
         LCR              /TO CAUSE INTERRUPT --
         DAC      START   /(SPCC WILL BE STARTED AT INTERRUPT)
         LAC      (EADC)  /NOW ADC CAN BE ENABLED
         LCR
         JMP*     ADCON

ADCOFF   0               /STOP DATA ACCUMULATION
         LAC      (DADC)  /DISABLE THE MODULE
         LCR
         LAC      (FLAG0) /SET END-OF-RECORD FLAG
         LCR              /TO CAUSE INTERRUPT --
         DZM      START   /PROCESSOR WILL STOP IT
         JMP*     ADCOFF

/EVENT BITS, POINTERS, AND ENTRY TO EOR-SERVICING
```

```
X.=400000        /TAGS TRAP DATA AS MODULE COMMAND
ENABLE=100000            /EVENT ENABLE BIT (EVENT 0 ONLY)
EVENTS  X.+RADC+EXIT     /ONLY ONE COMMAND NEEDED
        0
        0                /0 INDICATES EVENTS NOT USED
        0

ENTRY   0                /EOR-INTERRUPT SERVICING ENTRY
        LAC      STATUS
        SNA              /IF THE CHANNEL WAS OFF,
        JMP      E1      /SKIP OVER BUFFER SWAPPING
        LAC      .A
        LMQ
        LAC      .B      /SWAP BUFFER POINTERS
        DAC      .A
        LACQ
        DAC      .B
        LAC      (RDWC0) /READ THE WORD-COUNT REGISTER
        LCR
        RLD
        ALS      7       /SET TOP 7 BITS OF WORD
        LRSS     7       /COUNT ALL TO THAT OF BIT 7
        AAC      WC      /THEN CALCULATE NO. IN BUFFER
E1      PAL              /AND SET LIMIT REGISTER
        LAC      START
        DAC      STATUS
        SNA              /IF NOT TO BE STARTED UP,
        JMP      UNPACK  /JUMP ON TO UNPACKING SECTION
        LAC      (LDCA0) /LOAD CURRENT ADDRESS REGISTER
        LCR
        LAC      .A      /WITH POINTER TO NEW BUFFER
        LLD
        LAC      (LDWC0) /LOAD WORD-COUNT, ENABLE REGISTER
        LCR
```

```
        LAC     (WC0)   /WITH INITIAL VALUE
        LLD             /NOW CHANNEL IS GOING
UNPACK  LAW     -1      /BEGIN UNPACKING DATA
        PAX
        JMP     INDEX
E2      LAC*    .B,X    /GET THE ADC-CONVERSION NUMBER
        TAD*    ARG1    /ADD ON ADDRESS OF FIRST CHANNEL
        DAC     ADDR    /AND SET THE MEMORY ADDRESS
        ISZ*    ADDR    /ADD-1 TO MEMORY ADDRESS
        NOP             /(DISREGARDING OVERFLOWS)
INDEX   AXS     1       /AND LOOP AGAIN
        JMP     E2      /UNTIL DONE
        JMP*    ENTRY   /THEN RETURN TO INTERRUPTED PROGRAM

WC=40                   /BUFFER LENGTH
WC0=-WC&3777+ENABLE     /INITIAL WORD-COUNT, ENABLE VALUE
.A      A               /BUFFER POINTER
.B      B               /   "       "
A       .BLOCK  WC      /DATA BUFFER
B       .BLOCK  WC      /  "     "
START   0               /START-STOP FLAG
STATUS  0               /ON-OFF STATUS FLAG
ADDR    0               /ADDRESS OF INCREMENTED CHANNEL
        .END
```

REFERENCES

1. Costrell, Louis, ed.: CAMAC-A Modular Instrumentation System for Data Handling. Rep. TID-25875, USAEC, July 1972.
2. Anon.: CAMAC Tutorial Issue. IEEE Trans. on Nucl. Sci., vol. NS-18, no. 2, Apr. 1971 and references There in.
3. Anon.: PDP-15 Systems Reference Manual. DEC-15-BRZB-D, Digital Equipment Corp., Maynard, Mass.

TABLE I. - CAMAC FUNCTION CODES

Code F( )	Function	Use of R and W lines	Function signals					Code F( )
			F16	F8	F4	F2	F1	
0	Read group 1 register	Functions using the R lines	0	0	0	0	0	0
1	Read group 2 register		0	0	0	0	1	1
2	Read and clear group 1 register		0	0	0	1	0	2
3	Read complement of group 1 register		0	0	0	1	1	3
4	Nonstandard[a]		0	0	1	0	0	4
5	Reserved		0	0	1	0	1	5
6	Nonstandard[a]		0	0	1	1	0	6
7	Reserved		0	0	1	1	1	7
8	Test look-at-me	Functions not using the R or W lines	0	1	0	0	0	8
9	Clear group 1 register		0	1	0	0	1	9
10	Clear look-at-me		0	1	0	1	0	10
11	Clear group 2 register		0	1	0	1	1	11
12	Nonstandard[b]		0	1	1	0	0	12
13	Reserved		0	1	1	0	1	13
14	Nonstandard		0	1	1	1	0	14
15	Reserved		0	1	1	1	1	15
16	Overwrite group 1 register	Functions using the W lines	1	0	0	0	0	16
17	Overwrite group 2 register		1	0	0	0	1	17
18	Selective set group 1 register		1	0	0	1	0	18
19	Selective set group 2 register		1	0	0	1	1	19
20	Nonstandard		1	0	1	0	0	20
21	Selective clear group 1 register		1	0	1	0	1	21
22	Nonstandard		1	0	1	1	0	22
23	Selective clear group 2 register		1	0	1	1	1	23
24	Disable	Functions not using the R or W lines	1	1	0	0	0	24
25	Execute		1	1	0	0	1	25
26	Enable		1	1	0	1	0	26
27	Test status		1	1	0	1	1	27
28	Nonstandard		1	1	1	0	0	28
29	Reserved		1	1	1	0	1	29
30	Nonstandard		1	1	1	1	0	30
31	Reserved		1	1	1	1	1	31

[a]Used as add-one type read by channel processor.

[b]Used as program jump by channel processor.

TABLE II. - CAMAC IOT-INSTRUCTIONS

Name = value (octal)	Action
LCR = 706004	Load command register
LHD = 706044	Load high data bits
RHD = 706152	Read high data bits
LLD = 706024	Load low data bits
RLD = 706132	Read low data bits
SQF = 706001	Skip on Q-Flag
RDCR = 706112	Read command register
RHR = 706172	Read high data register
LON = 706101	Enable look-at-me (turn ON)
LOF = 706104	Disable look-at-me (turn OFF)
ESPCC = 706161	Enable SPCC
DSPCC = 706164	Disable SPCC
SLF = 706021	Skip on look-at-me flag
SEORF = 706061	Skip on end-of-record flag

TABLE III. - SPECIAL ALL-MODULE

COMMANDS

Name = value (octal)	Action
ZGEN = 100034	Initialize all modules
CGEN = 100035	Clear all modules
SETINH = 300037	Set module inhibit line
CLRINH = 320037	Clear module inhibit line

TABLE IV. - CAMAC SPCC-MODULE COMMANDS

F=010000 (octal)	Function code 1
N=000027 (octal)	Module address
A1=000400 (octal)	Subaddress 1
A2=2*A1	Subaddress 2
A3=3*A1	Subaddress 3
A4=4*A1	Subaddress 4

Name=value	Action
LDCA0=F*17+N	Load current address, subchannel 0
LDCA1=F*17+A2+N	Load current address, subchannel 1
LDWC0=F*17+A1+N	Load word-count/enable register 0
LDWC1=F*17+A3+N	Load word-count/enable register 1
FLAG0=F*25+A1+N	Set end-of-record flag, subchannel 0
FLAG1=F*25+A3+N	Set end-of-record flag, subchannel 1
RDWC0=F*1+A1+N	Read word count from subchannel 0
RDWC1=F*1+A3+N	Read word count from subchannel 1
STAT=F*1+A4+N	Read status register
CLEAR0=F*11+A1+N	Clear enable bits in subchannel 0
CLEAR1=F*11+A3+N	Clear enable bits in subchannel 1
SENSE0=F*10+A1+N	Sense and clear subchannel 0 flag
SENSE1=F*10+A3+N	Sense and clear subchannel 1 flag
SERROR=F*8+N	Sense error flag
CLRALL=F*10+N	Clear all events and flags

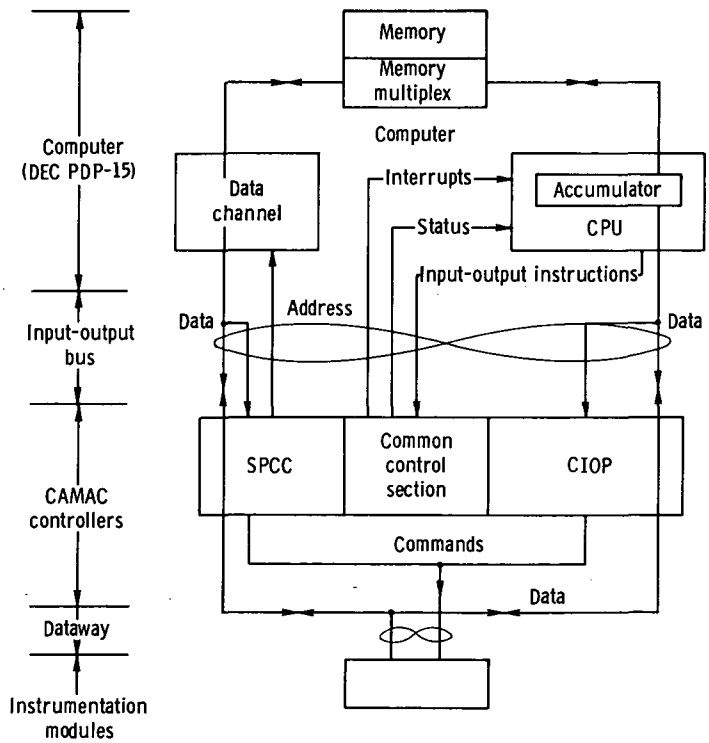
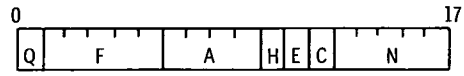
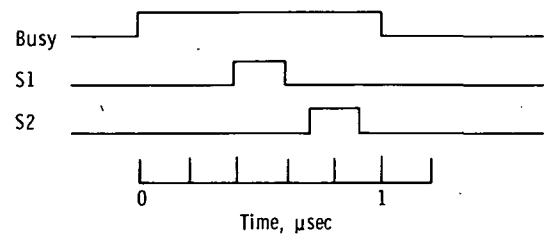


Figure 1. - Block diagram of CAMAC-computer system.



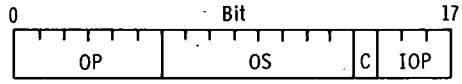
F Function code
A Subaddress
C Crate
N Station number
Q Q-response
H 24-bit transfer (two words)
E Exit after execution

(a) CAMAC command format.



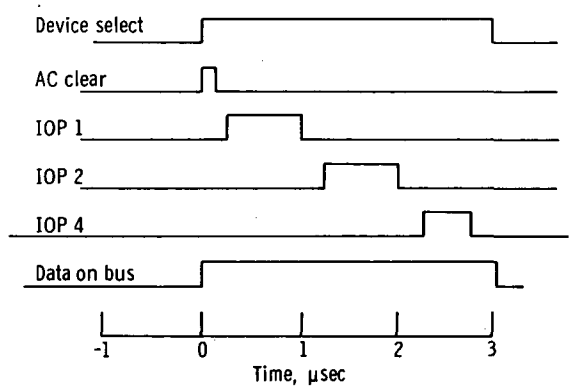
(b) CAMAC dataway timing.

Figure 2. - CAMAC command format and dataway signals.



OP Operation code (70)
DS Device select
C Clear accumulator
IOP Input-output strobe pulses

(a) PDP 15 input-output instruction format.



(b) PDP 15, input-output bus timing.

Figure 3. - PDP 15 instruction format and bus signals.

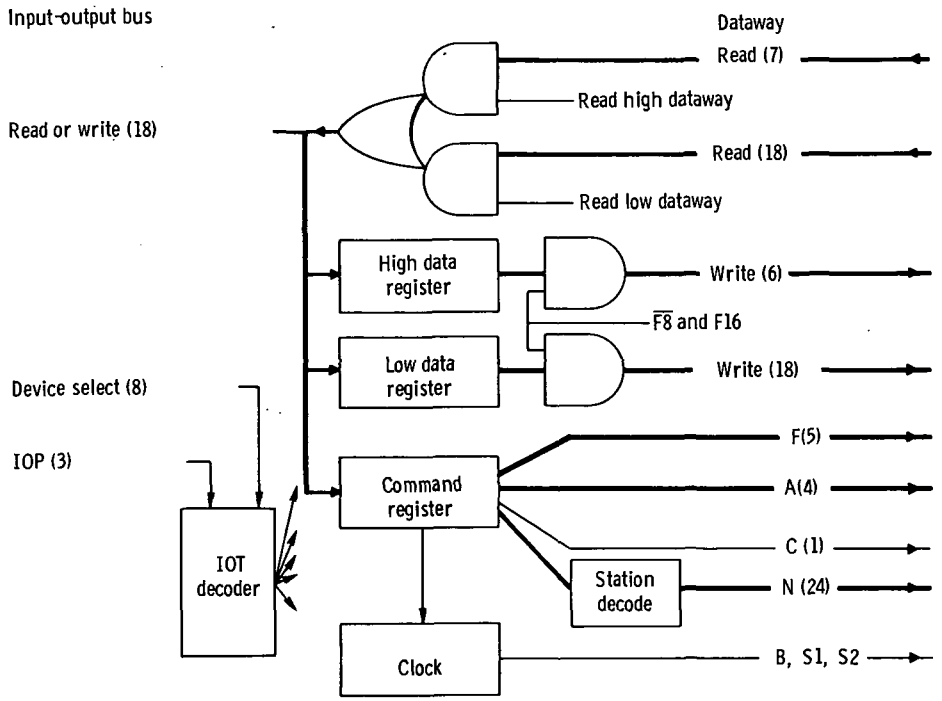


Figure 4. - CAMAC processor operated by programmed input-output bus. The parentheses contain the numbers of signal lines.

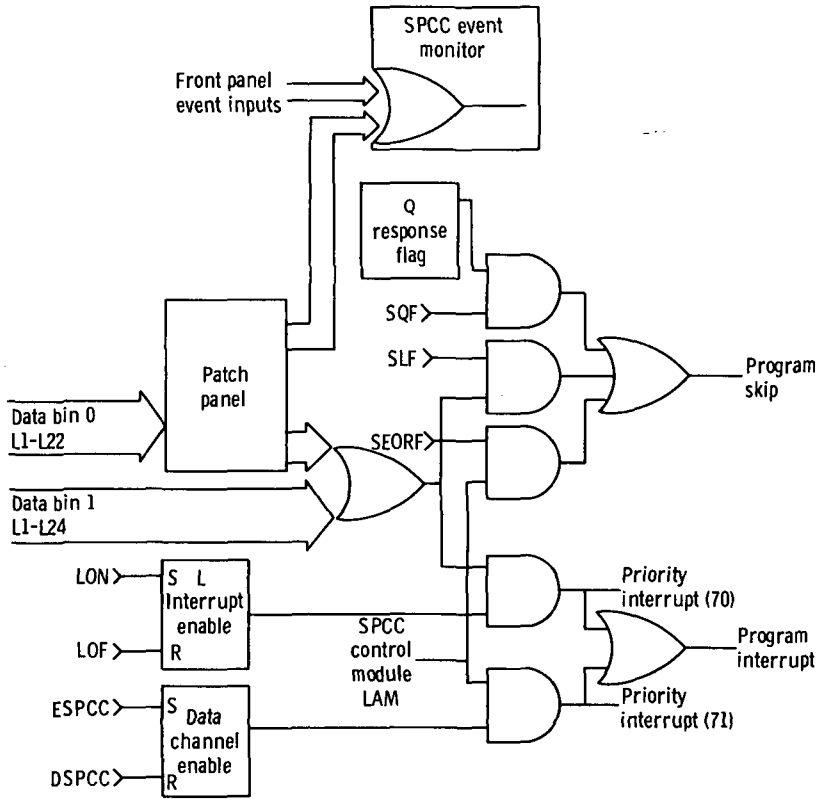


Figure 5. - Service request handling. The capitalized inputs are raised by the PDP 15 instructions listed in table II.

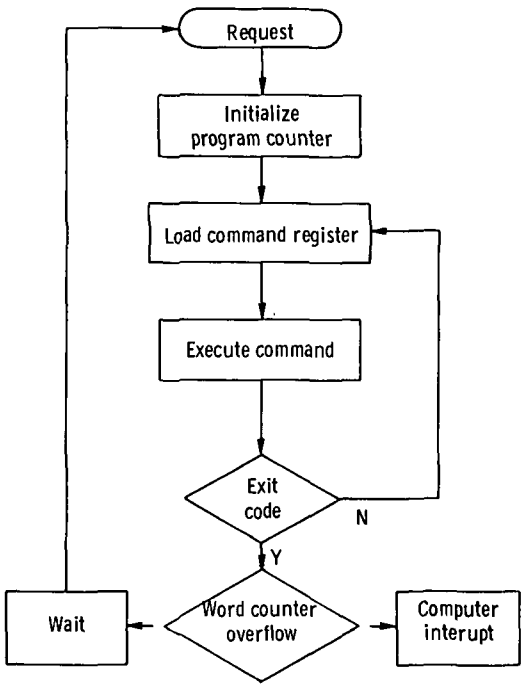


Figure 6. - Flow chart for simple CAMAC data-channel processor. The program counter is incremented each time a command is executed and the word counter is incremented each time a data word is transferred.

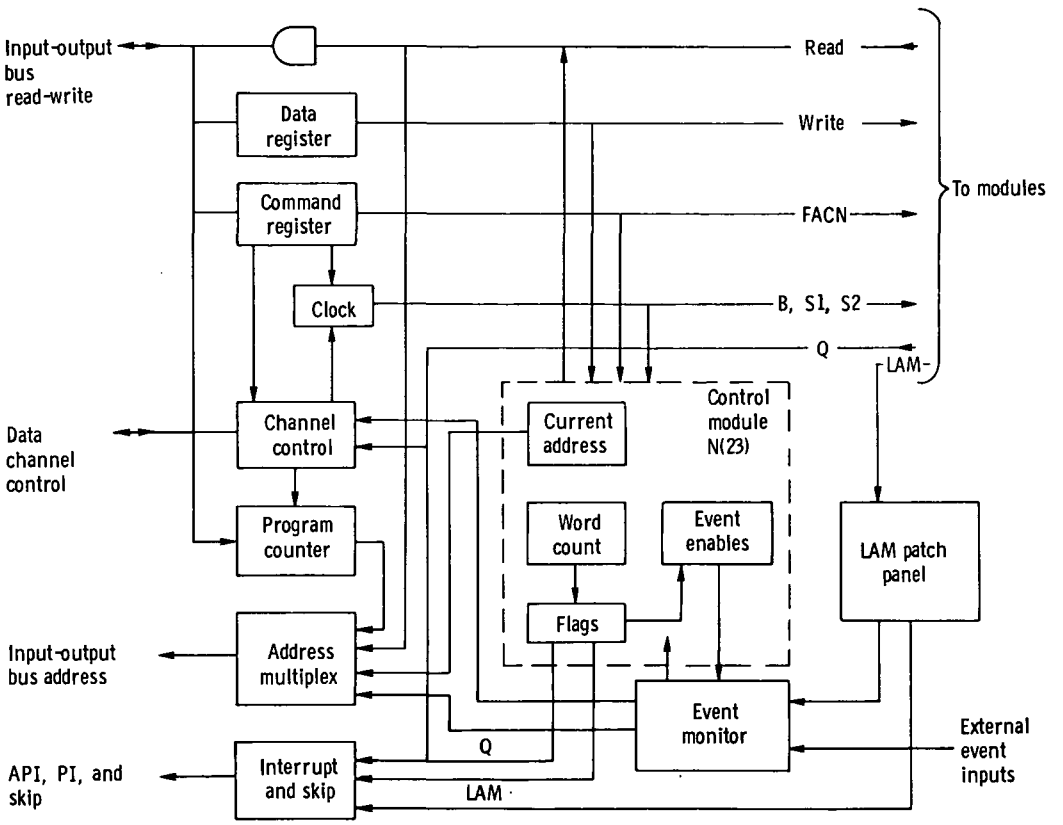


Figure 7. - Block diagram of CAMAC channel processor.

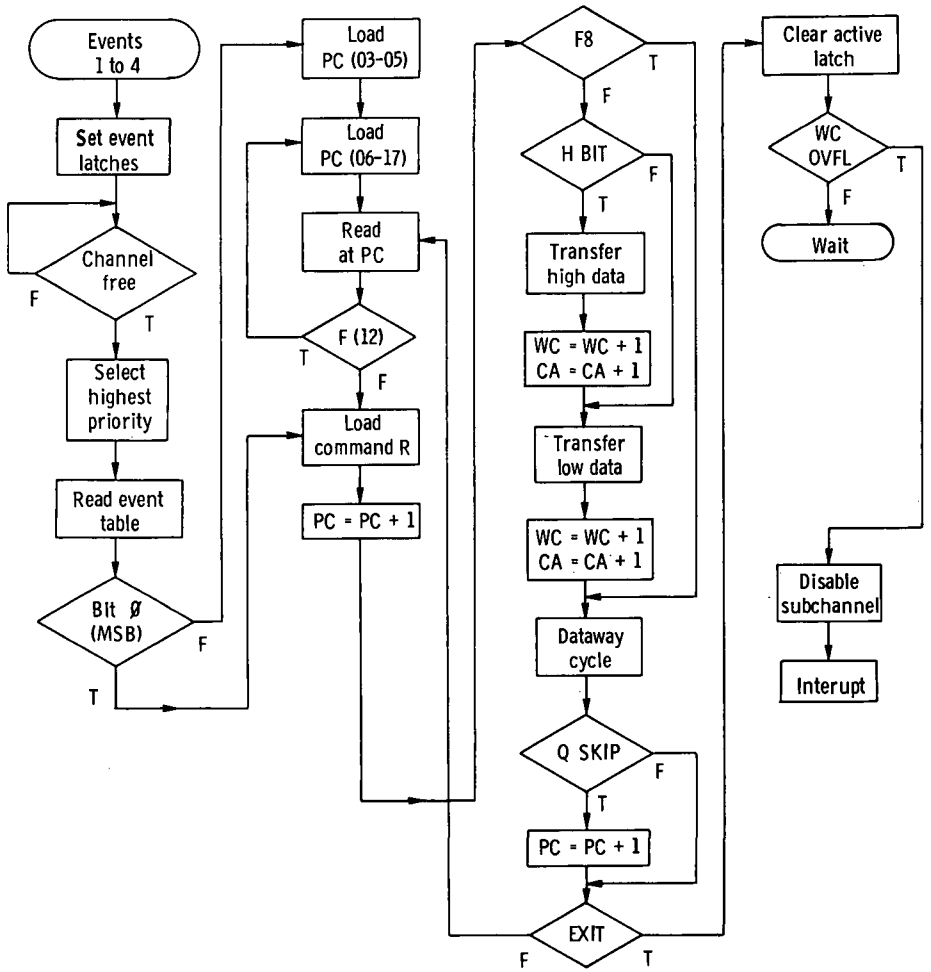
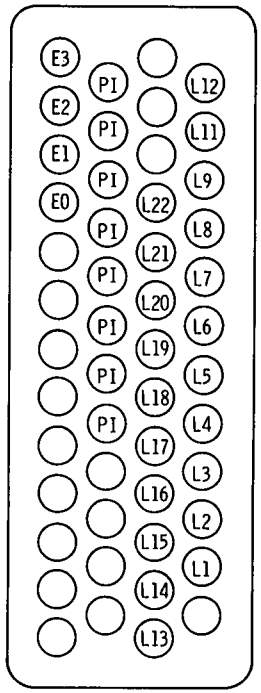


Figure 8. - SPCC flow charts.



E0-E3 Event inputs
PI Program interupt (PI or API) inputs
L1-1224 LAM outputs, crate 0

Figure 9. - LAM patch panel.

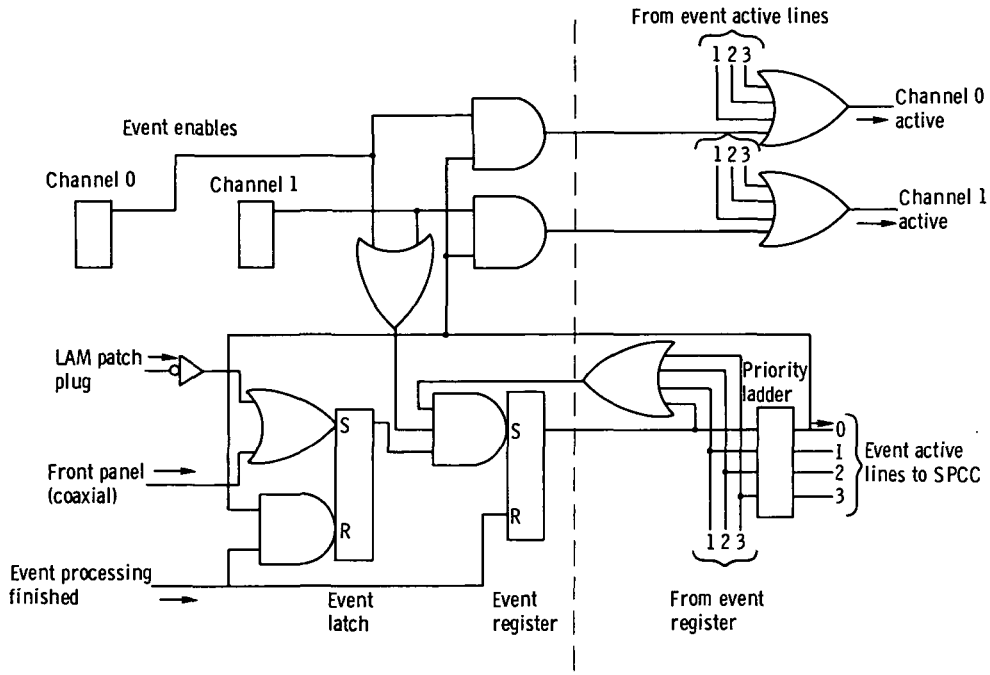


Figure 10. - Event monitor. The left half of the figure shows the circuitry associated with each event input. The right half shows the circuitry common to all four inputs.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE $300

SPECIAL FOURTH-CLASS RATE BOOK

POSTAGE AND FEES PAID
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
451

U.S.MAIL

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons. Also includes conference proceedings with either limited or unlimited distribution.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include final reports of major projects, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546