NASA TECHNICAL MEMORANDUM

NASA TM X- 67985

NASA TM X- 67985

A STORED PROGRAM CHANNEL PROCESSOR FOR CAMAC

by Robert W. Bercaw
Lewis Research Center
Cleveland, Ohio

TECHNICAL PAPER presented at
Digital Equipment Computer Users Society Meeting
San Francisco, California, November 11-13, 1971

A STORED PROGRAM CHANNEL PROCESSOR FOR CAMAC

Robert W. Bercaw
NASA-Lewis Research Center
Cleveland, Ohio

ABSTRACT

A PDP-15 interface has been developed for the CAMAC instrumentation standard which implements the features of both the addressable I/O bus and the single cycle data channel. The data channel section forms an independent I/O processor which executes programs stored in core. Programs consist of CAMAC commands plus special control characters and commands.

E-6716

INTRODUCTION

The CAMAC system of instrumentation[1] has been adopted by numerous laboratories in Europe and the United States as standard method for interfacing research apparatus with the aim of establishing a stable boundary between the instrumentation and the computing. A brief description of CAMAC will clarify the objectives of the work described here. CAMAC replaces the great variety of I/O busses found on various models of computers with a single nonproprietary design standardized both mechanically and electrically. It features a bin or crate which will accept up to 24 modules. A bus or dataway at its rear provides communication to the computer. The CAMAC specifications restrict the instrumentation contained in a module only to the extent necessary to insure compatability with the crate and dataway.

The dataway has a 24 bit read bus, a 24 bit write bus and a control bus. The control bus provides for 16 subaddresses and for performing up to 32 different operations on a module (function codes). Provision is made for polling the stations via a common "Q"-response line. In addition to the busses, each individual module has a pair of private lines for module station selection (N-line) and one for its service requests (LAM, Look-at-me).

The above characteristics are not embodied in the I/O bus of any computer line and thus CAMAC must itself be interfaced to the computer used. The interface must resolve the time and logical differences between the two structures and its design will therefore have a significant influence on the performance of the total system. We have designed and built an interface for the PDP 15 which implements both its programmed I/O bus and its data channel.[2] The section implementing the programmable I/O bus is a conventional design driven by IOT instructions, but the data channel section is quite unusual and avoids many of the limitations found in other designs. It operates independently of the CPU as a true stored-program processor. Some of its features are:

- Full repertoire of standard CAMAC operations
- Arbitrary sequences of CAMAC operations
- Module initiated execution of channel programs
- Module controlled branching of channel programs
- Dual independent data channels
- Direct memory increment
- Full independence of IOT driven processor
- Intermixing of 24 and 18 bit data transfers.

The processor was designed as a two crate controller, however there is nothing fundamental in its design which would prevent its adaptation to a branch driver.

IOT DRIVEN PROCESSOR

Figure 1 illustrates the principal elements of the IOT instruction driven section of the processor. The control section consists of logic to interpret the computer's IOT instructions, a command register to hold the FACN (Function, sub Address, Crate, and station Number), N-decoder and dataway clock. There are also the standard types of interrupt and Q-test facilities. A two part buffer register is used in the write path to resolve word length and timing differences between the I/O bus and the dataway, but there is enough flexi-

bility in the dataway timing specifications to allow read operations to be performed by merely gating the dataway on to the I/O bus, providing some simplification of the hardware and reducing the number of registers which must be saved during interrupts.

Three types of transfers may be performed; dataless, input and output. Only a single IOT instruction, loading the command register, is required for dataless transfers while two or three IOT's are required for data transfers depending on the word size. The IOT's are always made in the same sequence:

1. Send command word
2. Send or receive high order data
3. Send or receive low order data.

The processor will execute a dataway cycle whenever the low-order data is transferred or when the command register is loaded with a dataless transfer command (i.e. one with the F8 bit set).

CHANNEL PROCESSOR

In order to make a transition from the above processor to a channel processor, it is useful to replace the computer which drives it with a black box. It is then clear that the computer's main duty is as a source (and sink) for commands and data. The CAMAC processor then appears to be operating autonomously fetching commands from the box and then executing them, using the box as a pool for the data transferred to or from the modules. A program for the CAMAC channel processor would consist simply of a list of CAMAC commands which are executed in sequence by the processor. Of course, the operation of the processor depends entirely upon 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. There are no provisions for describing the word size (18 or 24 bits) or to control the processor itself, but the 18 bit word size of the PDP 15 provides enough room to include all the necessary parameters. A channel processor must also have some method of addressing data and commands in memory. The usual method of sequential addressing is generally adequate for both data buffers and commands, but it is desirable and convenient to provide for program jumps and conditional skips. There is little need for more general data addressing because of the parallel IOT driven processor.

A simple processor based on these principles will have the flow chart shown in Fig. 2. Upon receiving a request, the processor enters a program by initializing its program counter. 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 word count and current address registers are incremented after every data transfer. At the termination of execution of a program the processor will, depending on the word count overflow flag, either pause and wait for a new request or disable itself and interrupt the computer CPU.

SPCC HARDWARE

A block diagram of the stored program CAMAC channel (SPCC) which we have developed for the PDP 15 is shown in Figure 3. 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 processor. The data paths are identical to those of the IOT driven section, except for the addition of the program counter which must be loaded through the I/O bus.

Four different channel programs may be executed on a time shared basis. A program is entered upon the request of an "event" or flag from an external device which may be either a LAM or a BNC coaxial input 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. The event inputs pass through a monitor which holds requests until they can be serviced and also schedules their servicing according to a fixed priority scheme.

Control of the processor is performed through a unit attached to the dataway which looks like a standard CAMAC module. By giving it module status we have made it accessible to both the IOT driven and channel processors. This provides a communication link between the channel and CPU programs and allows the channel to control itself. The module contains word count and current address registers, overflow and error flags and event enables. There are two sets of registers to provide two independent subchannels, a requirement if input and output are to be carried on simultaneously. Each subchannel contains

a four-bit register to enable the events using it. Setting a bit in the register both enables an event and also associates it with the subchannel. The register is cleared upon word count overflow and must be reset after the interrupt has been serviced. It is made as an extension to the 12 bit word count register to reduce the number of steps needed to reinitialize a subchannel. The word count overflow flags are passed to the computer interrupt and skip facility in the same manner as the other module Q's and LAM's except that the channel LAM has its own API channel.

Other sections of the processor are the channel control, which supplies the sequencing and command interpretation logic, and the address multiplexer which generates the memory addresses for the channel. It obtains addresses from the program counter, the two current address registers, the read lines of the dataway and the event monitor. The dataway read line input allows the processor to perform the direct memory increment using data from a module as an address. The event monitor generates a unique address for each event ($24\text{-}27_8$) from which the processor initially loads the program counter.

SPCC OPERATION

The command word used by the processor is shown in Fig. 4. It has three bits in addition to the usual CAMAC FACN. The most significant bit is used to control the conditional skip. The processor 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 channel programs are usually short, devoting a command to a stop code would significantly lengthen a program. Therefore, the processor is stopped by setting an Exit bit in the last command to be executed.

Several nonstandard function codes have been employed to provide special handling by the processor. F(12) has been employed to provide for unconditional program jumps within a page. The processor 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. The processor trims off the F4 bit creating a read (F(0)) or a read and clear (F(2)) command and then uses the data from the module as an address at which to increment memory.

Figure 5 illustrates the sequence of events in processing an event. In the request phase, requests are stored in the event latches until the channel is free. The latches are then strobed into a register and the highest priority event is activated. The event table, comprised of four fixed addresses in memory (24-27), is read to obtain the address of one of the four relocatable channel programs. Normally this address is placed in the program counter, but because a number of applications employ programs consisting of a single-command, the address can be interpreted as a command. A command is denoted by a one in its most significant bit. In the 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 P.C. is incremented.

The execute phase is then entered starting with tests of the F8 and "H" bits. 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. Incidentally, the burst mode is used if two data transfers are made. The word count and current address registers are incremented after each data transfer. If the Q-response from the module is different from the one programmed into the command, the program counter is again incremented to create a skip. This provides for device controlled program branching. Finally, the processor branches on the exit bit, either returning to the fetch phase or going on to the exit phase. In this last phase the entire event register is cleared, but only the latch of the event serviced is cleared. The processor then halts and waits for another event, or if there is an overflow, it clears the subchannel's enable register and interrupts the computer for buffer processing. Note that the interrupt occurs when the program is finished, not when the word count overflows. The actual buffer length must exceed the word count by the size of the largest event.

When more than one event is used to fill a buffer, it is desirable to tag each event so that the buffer can be unpacked by the main program. An "event active register" is provided in the control module which can be read as part of the event processing. It contains the number of the event and has a one in its most significant bit. If desired, the MSB can be nulled when other modules are read so that these event tags become unique. The MSB need never be lost because it is also read in along with the high order 6 bits

of data. There is a bit in the event active register which is set whenever there is a Q-skip. If the test is made before the register is read, the tag word will indicate its result.

The prime advantage in using the SPCC over the IOT driven processor is in its speed of data transfer and in the reduced interrupt handling,but it can also be used to pre-edit data. One example is in nuclear physics experiments having two or more ADC's run in coincidence, but where it is also important to record the high rate singles spectra. The processor can branch on a flag raised by the coincidence unit, using the direct increment if it is a single or recording all of the data into a buffer for more elaborate processing by the CPU if it is a coincidence. A second example is provided by synchronous data transmission, type II, where there are control bytes transmitted along with the data which are used to indicate when data recording starts and stops. The SPCC can look for these controls and only record valid data.

An overview of the information flow in the processor and computer is shown in Fig. 6. General purpose system routines have been written by Mr. T. Fessler to provide interrupt handling, data buffer management and construction of the table containing the program addresses. An array with the global "EVENTS", containing the channel programs, must be provided. It has been found that these programs are very easily constructed using the MACRO assembler. Details of the processor and software will be presented in a forthcoming NASA Technical Note.

ACKNOWLEDGEMENTS

The author is indebted to T. Fessler and J. Arnold for their suggestions and encouragement. The project was greatly aided by test programs provided by T. Fessler.

REFERENCES

[1] "CAMAC A Modular Instrumentation System for Data Handling". EURATOM Report EUR 4100e, March 1969.
CAMAC Tutorial Issue, IEEE Transactions in Nuclear Science, Vol. NS-18, No. 2, April 1971 and references therein.

[2] PDP 15 Systems Reference Manual, DEC-15-ER2B-D.

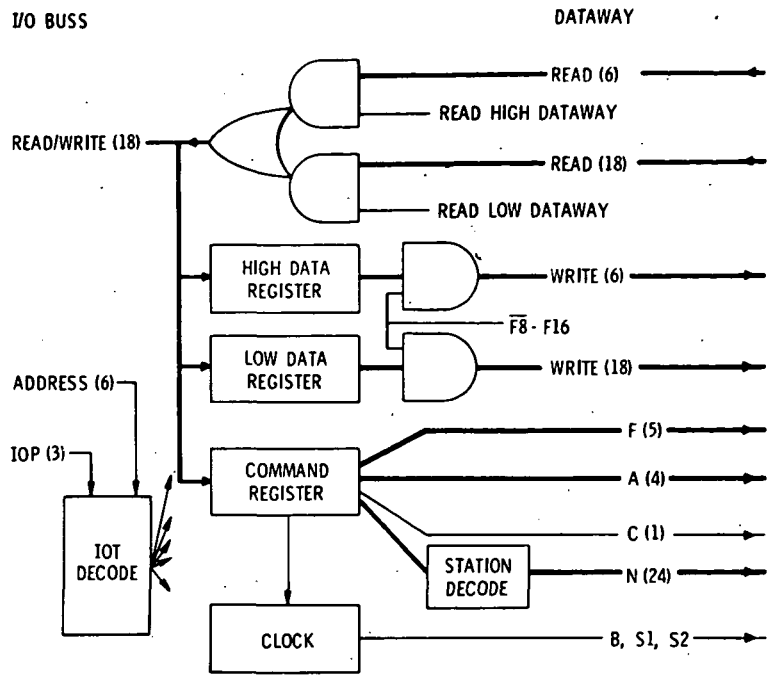


FIG. 1 CAMAC PROCESSOR OPERATED BY THE PROGRAMMED I/O BUS

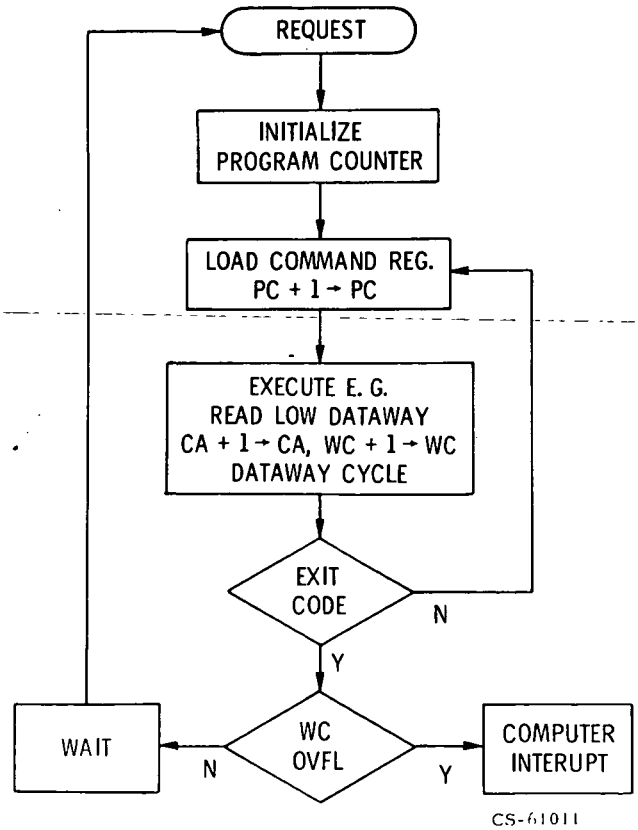


FIG. 2 FLOW CHART FOR A SIMPLE CAMAC DATA-CHANNEL PROCESSOR

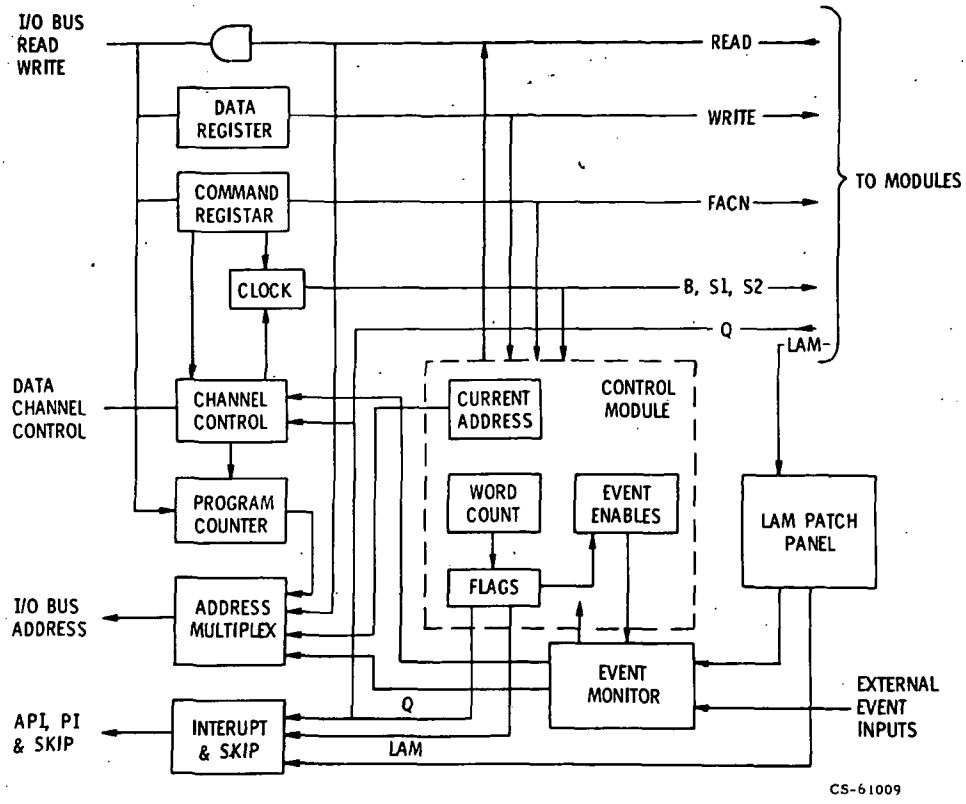
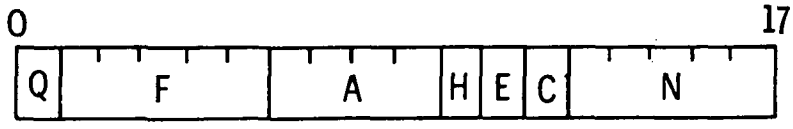


FIG. 3 CAMAC CHANNEL PROCESSOR

COMMAND WORD



F	FUNCTION CODE
A	SUB ADDRESS
C	CRATE
N	STATION NO.
Q	EXPECTED Q-RESPONSE (NO SKIP)
H	24 BIT TRANSFER (2 WORDS)
E	EXIT AFTER EXECUTION

SPECIAL FUNCTION CODES

F (4)	INCREMENT MEMORY; DATAWAY SEES F (0)
F (6)	INCREMENT MEMORY; DATAWAY SEES F (2)
F (12)	UNCONDITIONAL JUMP

CS-61013

FIG. 4 COMMAND STRUCTURE FOR THE SPCC

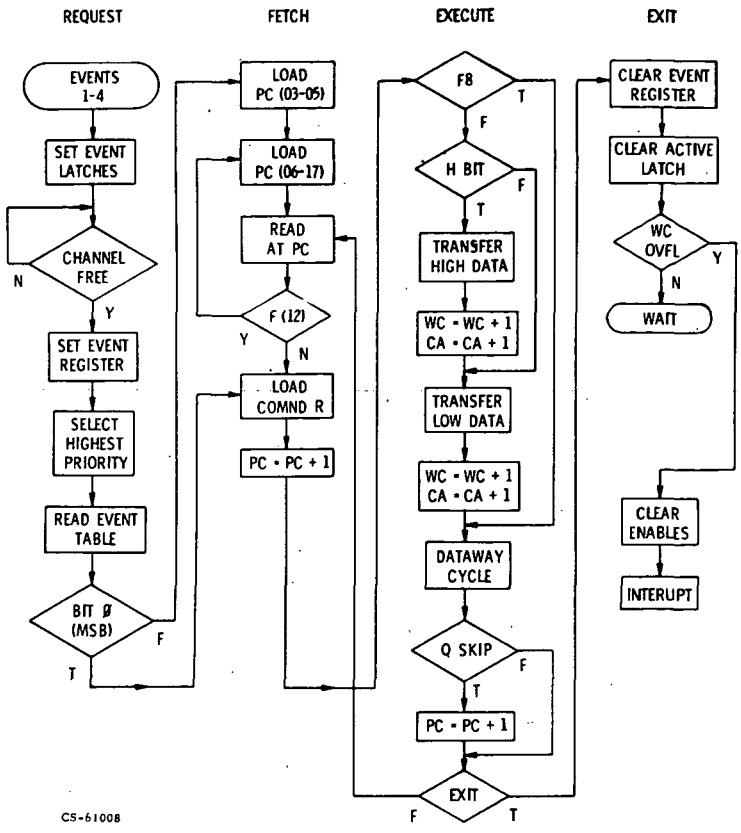


FIG. 5 FLOW CHART FOR SPCC PROGRAM EXECUTION

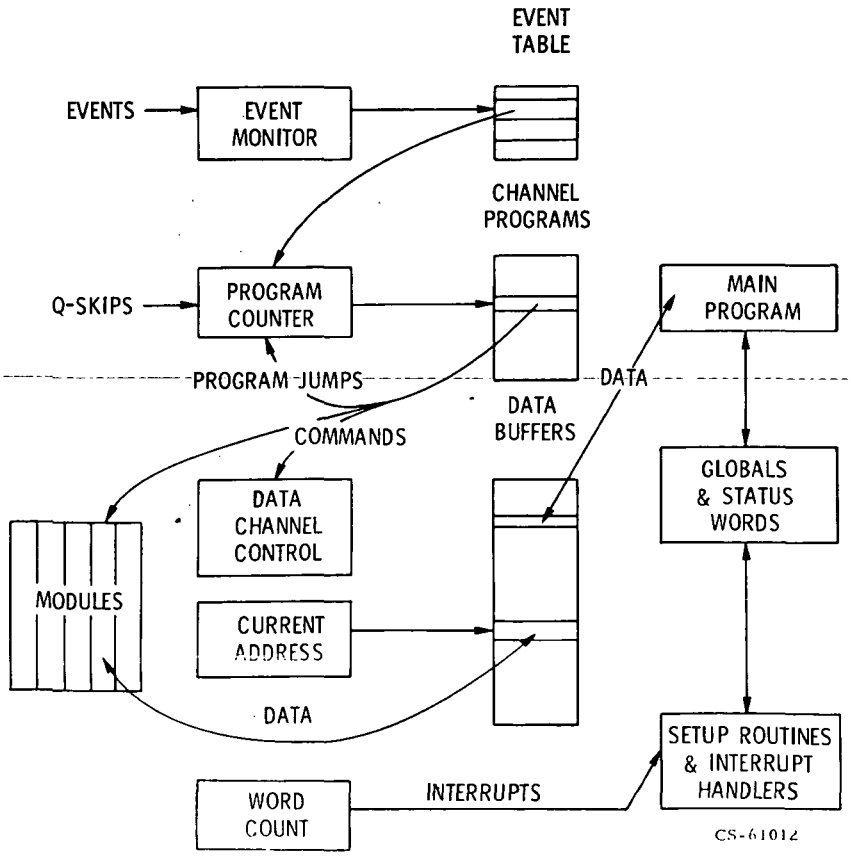


FIG. 6 OVERVIEW OF DATA FLOW IN A SPCC SYSTEM