CERN - DATA HANDLING DIVISION
DD/73/30
R.D. Russell
October, 1973

CAMAC FACILITIES IN THE PROGRAMMING LANGUAGE PL-11

Abstract

This paper describes the features for programming CAMAC easily and efficiently in the programming language PL-11.

(To be presented at the First International Symposium on CAMAC in Real-Time Computer Applications, December 4-6, 1973, Luxembourg.)

Introduction

PL-11 is an intermediate-level, machine-oriented programming language for the PDP-11 computer (1). It was designed and implemented as the programming tool for the on-line mini-computers used in data acquisition at the CERN OMEGA project (2). Because all the experiment electronics are connected to the computer via CAMAC, facilities have been incorporated into PL-11 that enable the physicists to program both CAMAC and the PDP-11 in a single language. This paper, which is a condensation of an earlier paper (3), discusses just the features of the language that are related to CAMAC.

1. Design Philosophy

There are three basic requirements that must be met by any CAMAC programming system if this system is to be an easily used and understood tool for effective use of CAMAC.

1. The programmer must have the ability to give meaningful symbolic names of his choice to any CAMAC C-N-A triple. The association of a name with a triple appears exactly once in any program (in PL-11, as a declaration at the beginning of the program), and all operations with this CAMAC module are done by referring to this symbolic name. Not only does this make the program more readable, it also makes modification of a CAMAC configuration extremely simple. One has only to change a single declaration and recompile the program. The compiler will then ensure that the addresses in all operations to this module are changed properly.

2. The programmer must have the ability to give meaningful symbolic names of his choice to any CAMAC function.

3. The programmer should be able to use the same syntactic notation for CAMAC variables and functions as he uses for ordinary program variables and functions. This includes the ability to test Q responses in the same manner as normal Boolean tests are made, the ability to index through arrays of CAMAC variables in the same manner as normal arrays are indexed, etc. Such a consistent use of a high-level language syntax makes it easy for a non-professional programmer to utilize CAMAC effectively, since he is able to manipulate CAMAC data in exactly the same way as he deals with normal data.

PL-11 is designed to be syntactically as high-level a language as possible so that readable, structured programs can be written easily. At the same time, PL-11 is semantically as close to the machine architecture as necessary in order to guarantee efficiency and the ability to utilize fully all features of the hardware. Therefore, in addition to the three requirements listed above, there is a fourth requirement satisfied by the CAMAC facilities in PL-11:

4. The compiler should produce efficient, in-line object code for every CAMAC operation. There should be no subroutine-calling overhead and no required run-time system to support the CAMAC features of the language. This implies that an entire real-time operating system can be written in PL-11 without the use of any other language and without any necessary run-time environment.

2. Declaring CAMAC Variables

The purpose of CAMAC variable declarations in PL-11 is to give programmer-defined symbolic names to CAMAC registers so that all operations to CAMAC can be done by reference to these names. The syntactic form of such declarations is analogous to the PL-11 construct that enables the programmer to give a symbolic name to any machine register, memory cell, or I/O device register.

```
INTEGER PRINTBUFFER SYN MEMORY($177566),
        PRINTSTATUS SYN MEMORY($177564);
```

gives the symbolic name PRINTBUFFER to the memory cell with absolute address 177566 octal, and the name PRINTSTATUS to the cell with absolute address 177564 octal.

```
INTEGER CRTBUFFER SYN CRATE 1 STATION 2
                      SUBADDRESS 14,
        CRTSTATUS SYN CRATE 1 STATION 5
                      SUBADDRESS 0 GROUP 2;
```

gives the symbolic name CRTBUFFER to the CAMAC register in crate 1 station 2 subaddress 14 group 1 (by default), and the name CRTSTATUS to the group 2 register at subaddress 0 station 5 of crate 1. In such declarations, the compiler will accept only crate numbers in the range (1,7), station numbers in (0,31), subaddress numbers in (0,15), and group numbers in (1,2).

It is important to note that both memory and CAMAC integers are 16-bit quantities. In keeping with the philosophy of designing a fully integrated system it was decided very early that since a PDP-11 word is only 16 bits all CAMAC modules used in OMEGA would utilize only the low-order 16 bits of the 24 data bits provided in CAMAC. This decision has proven to be correct: all programming errors due to a mismatch of precision are eliminated, and no time is lost packing and unpacking PDP-11 words. Since many 16-bit CAMAC modules are commercially available, no penalty was paid in higher costs or lost data. Clearly this decision also made it easier to integrate CAMAC

variables into PL-11.

3. Using CAMAC Variables

Once a CAMAC variable has been declared, it is used just like any other PL-11 variable. Assuming the declarations written above,

```
X => PRINTBUFFER;
```

moves the contents of location X into location PRINTBUFFER, which causes a character to be typed on the teletype due to the definition of PRINTBUFFER as absolute address 177566 octal (the teletype data register).

```
X => CRTBUFFER;
```

writes the contents of location X into the CAMAC register CRTBUFFER, which presumably causes the module at that CAMAC location to display this character on a CRT.

```
PRINTSTATUS => Y;
```

moves the contents of PRINTSTATUS (the teletype status register) into location Y.

```
CRTSTATUS => Y;
```

reads the contents of CAMAC register CRTSTATUS into memory location Y.

Of course it is not necessary to store these status values in memory as they can be tested directly as follows:

```
WHILE PRINTSTATUS = BUSY DO ;
```

is an empty wait loop for the teletype to become "not busy".

```
WHILE CRTSTATUS = BUSY DO;
```

is a wait loop for the CAMAC display modules to indicate "not busy" in the same manner. To test for an ordinary variable X in some range we can write:

```
IF X >= LOWLIMIT & X <= HIGHLIMIT THEN
     BEGIN   COMMENT  WITHIN RANGE; ... END;
```

If SCALER is declared as a CAMAC register, we can write a similar test:

```
IF SCALER >= LOWLIMIT & SCALER <= HILIMIT
THEN BEGIN COMMENT WITHIN RANGE;...END;
```

If we wish to wait until SCALER reaches some value THRESHHOLD, we write:

```
WHILE SCALER < THRESHHOLD DO ;
```

In all of these examples, the PL-11 compiler generates for each CAMAC reference the two in-line instructions that are required by the manufacturer's CAll interface for a CAMAC operation. The first instruction sets up the crate and function codes, the second sets up the station and subaddress codes and actually performs the transfer. Hence this method is "as efficient as possible" as well as being easy to use and understand.

4. CAMAC Arrays and Block Transfers

A block of ten CAMAC registers starting at station 1 subaddress 0 of crate 1 can be declared in PL-11 as:

```
ARRAY 10 INTEGER SCALER SYN
        CRATE 1 STATION 1 SUBADDRESS 0;
```

If we also define a compile-time constant SPACING to be the address increment between consecutive SCALER modules:

```
EQUATE SPACING SYN 2;
```

then the statement:

```
FOR R1 FROM 0 STEP SPACING UPTO  9*SPACING DO
          0 => SCALER(R1);
```

will generate code to reset to zero ten scalers in consecutive subaddresses at the same station. The same statement will reset to zero ten scalers in consecutive stations if we instead declare:

```
EQUATE SPACING SYN 32;
```

In order to accumulate the sum of these scalers in a variable SUM with the average in variable AVERAGE, we can write:

```
0 => SUM;
FOR R1 FROM 0 STEP SPACING UPTO 9*SPACING DO
            SUM + SCALER(R1);
SUM => AVERAGE/10;
```

If we wish to store the ten scaler values in a ten element array X, we can write:

```
0 => R2;
FOR R1 FROM 0 STEP SPACING UPTO  9*SPACING DO
     BEGIN
     SCALER(R1) => X(R2);   R2 + 2;
     END;
```

where we have to keep an extra index register R2 to account for the constant spacing of 2 between consecutive integers in memory.

This last example demonstrates a transfer of ten words in which each word transferred requires a CAMAC read operation. If this scaler block included a control module that allowed the entire block to be transferred with a single command to that module, this could also be done in PL-11 as follows:

```
INTEGER BLOCKSTART SYN CRATE 1 STATION 3
                       SUBADDRESS 0;

REF(X) => BLOCKSTART;
```

where REF(X) is the memory address of array X and we assume that initializing the BLOCKSTART

control module with a memory address will cause the block of scalers to be transferred into memory starting at that address. A more common situation is to include in the CAMAC control module a block size register, to define how many words to transfer. Then the block transfer is written in PL-11 as:

```
REF(X) => BLOCKSTART;   10 => BLOCKSIZE;
```

5. Other CAMAC Functions

The preceding sections demonstrated the CAMAC read, write, and reset functions in PL-11 and showed that they could be written with the same syntactic form as normal variable operations in PL-11. There are, however, 32 possible CAMAC functions, and any or all of these can be given symbolic names by PL-11 declarations such as the following;

```
CAMAC FUNCTION ENABLE(26), DISABLE(24),
               TESTLAM(8), CLEARLAM(10);
```

where the symbolic name is chosen by the programmer, and the function number must be in the range (0,31). These functions can be applied to any CAMAC variable using the normal function notation:

```
ENABLE(SCALER);
DISABLE(CRTSTATUS);
CLEARLAM(SCALER(R1+8));
```

For example, if we wish to read a block of scalers into an array X such that each scaler is read only when it responds to a TESTLAM function, we can write in PL-11:

```
    0 => R2;
    FOR R1 FROM 0 STEP SPACING UPTO 9*SPACING DO
           BEGIN
WAIT:      TESTLAM( SCALER(R1));
           IF NOT Q THEN GOTO WAIT;
           SCALER(R1) => X(R2);
           R2 + 2;
           END;
```

where we have used the name Q which is a pre-declared PL-11 name for a bit in memory that is set or reset by the CAMAC CAll interface according to the Q-response of a CAMAC operation.

Interrupts caused by CAMAC modules can be handled in PL-11 by declarations such as:

```
CAMAC INTERRUPT 20 PROCEDURE SCALEROVERFLOW:,
    BEGIN . . .  END;
```

This declaration is just an extension to CAMAC of the interrupt handling mechanism present in the PL-11 language. It causes the compiler to set up transfer vectors by which the operating system can call this procedure whenever an interrupt is caused by LAM bit 20. The interrupt number must be in the range (1,24), and all interrupt enabling, disabling, clearing, and resolution of multiple LAM sources can and must be programmed explicitly with PL-11 functions.

6. Conclusion

The CAMAC features integrated into PL-11 are efficient, easy to use, and easy to understand. They make the full range of CAMAC functions available to the programmer in a simple symbolic notation. This has been accomplished by a slight extension of an existing language to include CAMAC variables, functions, and interrupt procedures. Clearly this technique can be applied to other languages, such as FORTRAN, ALGOL, and especially PL/I since its structure allows an easy extension by the addition of a CAMAC attribute. The PL-11 compiler is able to generate highly efficient in-line code for each CAMAC operation due to the simple design of the CAll interface between the PDP-11 and CAMAC. For a more complex interface a compiler would probably have to generate calls to system subroutines, but the notation used by the programmer would remain unchanged (this is the technique used for FORTRAN floating-point operations on many small computers that lack floating-point hardware). It is suggested that such an extension of a widely used language, such as ALGOL, FORTRAN, or PL/I, along these lines would be a highly desirable way to provide a universal tool for programming CAMAC.

References

1. RUSSELL, Robert D. 'Preliminary Specifications for PL-11: A Programming Language for the DEC PDP-11 Computer', OMEGA Project Development Note SW-29 (Nov. 1971).

2. RUSSELL, Robert D. 'The OMEGA Project: A Case History in the Design and Implementation of an On-line Data Acquisition System', Proceedings of the 1972 CERN Computing and Data Processing School CERN 72-21 (Dec. 1972) PP 275-340.

3. RUSSELL, Robert D. 'Easy and Efficient CAMAC Programming in PL-11', Paper presented at the Third European Seminar on Real-time Programming (May 1973).