Behavior Research Methods & Instrumentation
1974, Vol. 6 (5), 493-496

Use of an algebraic language in laboratory control programming with small (4K) computers

NICHOLAS VITULLI and JAMES H. REYNOLDS*
Colgate University, Hamilton, New York 13346

Most lab-control programs for small (4K) computers must be written in assembly or machine language, making it necessary for the user to expend considerable time and effort both in learning assembly-programming skills and in writing programs. One of the most frequently used small laboratory computers (the PDP-8 series) can, however, be programmed to provide sophisticated control using an algebraic language (FOCAL) in as little as 4K memory locations. The FOCAL language includes a special program sequence, FNEW, which provides capacity for a single user-defined function written in machine language. This paper describes an implementation of the FNEW sequence that permits execution of multiple user-written machine language subprograms within a FOCAL program. Use of this capability in the context of process-control problems in the behavioral laboratory is discussed and illustrated.

Most commercially available computers require special hardware interfaces which allow them to turn experimental circuits on and off, or to sense inputs, on command. Programming instructions for controlling this additional hardware are normally in machine language. Some laboratories have computers of sufficient size to incorporate these machine instructions into subroutines which permit process-control programming in a higher-level language such as FORTRAN (e.g., Restle & Brown, 1969). Many labs, however, are equipped with small (4K) machines that do not have sufficient memory to support an effective higher-level language. Thus, psychologists with programming skills in a high-level language are often discouraged from using laboratory computers because their use requires development of additional assembly or machine language skills.

For laboratories equipped with any of the Digital Equipment Corporation's (DEC) PDP-8 series of computers, this difficulty can be surmounted in large part by writing lab control programs in FOCAL[1] and, when it becomes necessary to provide interface instructions in machine language, to do so by using the FOCAL function subprogram FNEW in the manner described below. FOCAL is an easy-to-learn algebraic language that is efficient enough to provide advanced programming capabilities in PDP-8 machines with 4K memory. The FNEW calling and exiting sequence, described in the 1972 edition of DEC's handbook *Programming Languages* (Vol. 2, pp. 11-45 through 11-54), allows the FOCAL user to write a function subprogram in assembly language. To use FNEW, the user first inserts his own routine within the calling-exiting sequence, and then adds the entire FNEW subprogram (calling sequence, user's routine, and exiting sequence) as an overlay to FOCAL-8 or FOCAL-69. As a result, the user can, in effect, create a machine-language function unique for his purposes and can then call the function from any FOCAL program in the same way that any standard FOCAL function can be called.

While this usage of FNEW helps to expand the versatility of the FOCAL language, it is still limited to some extent because only one such new routine can be added easily.[2] However, the present paper illustrates a way that overcomes this limitation by constructing a FNEW subprogram that permits execution of any sequence (and even differing sequences) of machine language instructions, at any point or points in a FOCAL program, simply by calling FNEW and giving (as the arguments to FNEW) the machine instructions to be executed. Figure 1 presents a flow chart and listing of the modified routine that includes the original calling sequence listed in the handbook (p. 11-46); it also includes the instructions that permit FNEW to execute any sequence of machine instructions passed to it as arguments.

The listing in Fig. 1 contains several important modifications of the FNEW sequence illustrated in the handbook (p. 11-46): the listing includes appropriate locations of START to be used when the user's answers to the two FOCAL questions about loading additional functions vary (see p. 11-47 in handbook); two instructions (see Locations 5311, 5312) have been added which store the contents of Location 16 temporarily whenever the routine is called, thus preserving any information previously stored there; and most important, seven instructions (see Locations 5325-5333), which initialize the accumulator and the Location ANS, return the original information to Location 16, and then begin the instruction sequence which permits execution of up to 10 machine-language arguments any time FNEW is called, have been added.

The logic of this latter sequence is not difficult to follow. The original FNEW instructions listed at Loca-

*Requests for reprints should be sent to James H. Reynolds, Psychology Dept., Colgate University, Hamilton, New York 13346.

```
                / ***** F N E W *****************************************
          0035  BOTTOM=0035
          0374  FNTABF=0374
          0053  INTEGER=0053
          4540  PUSHJ=4540
          0044  FLAC=0044
          0136  EFUN31=0136
          0066  CHAR=0066
          1613  EVAL=1613
          5541  POPJ=5541
                / ***** PRESENT LENGTH IS 66 OCTAL LOCATIONS *****
                /START=4617+66                  /YES=YES
                /START=5177+66                  /NO=YES
          5311  START=5377+66                   /NO=NO

          0035  *BOTTOM
0035      5310  FNEW=.
          0410  *FNTABF+14
0410      5311  FNEW
          5311  *START
5311      1016  FNEW,   TAD     16      /       *********************************
5312      3346          DCA     S16     /.......*   PREPARE LOCATION 16 FOR    *
5313      1352          TAD     LIST-1  /       *PRESENT USE AND FUTURE RETURN*
5314      3016          DCA     16      /       *********************************
5315      3350          DCA     KOUNT                   /
5316      2350          ISZ     KOUNT                   /
5317      4453          JMS I   INTEGER/        *********************************
5320      3416          DCA I   16      /.......*   LOAD UP TO 10 MACHINE      *
5321      4540          PUSHJ           /       *LANGUAGE ARGUMENTS INTO FNEW,*
5322      5366          ARG             /       *       STARTING AT LIST       *
5323      7410          SKP             /       *********************************
5324      5316          JMP     FNEW+5                  /
5325      7300          CLA CLL         /       *********************************
5326      3351          DCA     ANS     /.......*  ZERO ANS AND ACCUMULATOR,   *
5327      1334          TAD     RET     /       *AND ADD JMP RET+1 INSTRUCTION*
5330      3416          DCA I   16      /       *  TO END OF LIST ARGUMENTS    *
5331      1346          TAD     S16     /       *********************************
5332      3016          DCA     16                      /
5333      5353          JMP     LIST    /       *********************************
5334      5335  RET,    JMP     RET+1   /.......*          EXECUTE LIST           *
5335      7300          CLA CLL         /       *********************************
5336      1351          TAD     ANS                     /
5337      7110          CLL RAR         /       *********************************
5340      3045          DCA     FLAC+1  /.......*  RETURN TO FOCAL MAINLINE    *
5341      7010          RAR             /       *PROGRAM WITH CONTENTS OF ANS *
5342      3046          DCA     FLAC+2  /       *  AS VALUE OF FUNCTION FNEW  *
5343      1347          TAD     C14     /       *********************************
5344      3044          DCA     FLAC
5345      5536          JMP I   EFUN31
5346      0000  S16,     0
5347      0014  C14,     14
5350      0000  KOUNT,   0
5351      0000  ANS,     0
5352      5352          LIST-1
                / ***** ALLOW UP TO 10 ARGUMENTS *****
5353      0000  LIST,   0
5354      0000          0
5355      0000          0
5356      0000          0                                       /PATCHES TO RUN FOCAL WITHOUT INTERRUPTS
5357      0000          0
5360      0000          0                               0063    *63
5361      0000          0                                       /       1354    /WAS 2676
5362      0000          0                                       /       2414    /WAS 2666
5363      0000          0                               2732    *2732
5364      0000          0                                       /       5336    /WAS 6001
5365      0000          0                               2762    *2762
5366      1066  ARG,    TAD     CHAR                            /       7000    /WAS 6064
5367      1376          TAD     MCOMMA
5370      7640          SZA CLA                                 /PATCHES TO ELIMINATE ";" AND "="
5371      5375          JMP     .+4
5372      4540          PUSHJ                           1217    *1217
5373      1612          EVAL-1                  1217    7600            7600    /WAS 4551
5374      7001          IAC                             6002    *6002
5375      5541          POPJ                    6002    7600            7600    /WAS 4551
5376      7524  MCOMMA, 7524                                    $
```

(Figure continuation)

Fig. 1. Listing and flowchart of a subprogram designed to process machine language instructions. Machine language instructions.

tions 5315-5324 serve to load all the FNEW arguments successively into the available-space locations, starting at LIST. When all arguments have been thus loaded as machine-language instructions, the sequence in 5325-5333 initializes the location ANS, then deposits the instruction JMP RET + 1, at the end of that list of instructions, and branches to LIST; execution proceeds from there. Since the last of the instructions now stored in the previously empty locations following LIST is JMP RET + 1, all of the machine language arguments will be executed in order and then control will be directed to RET + 1, from which point FNEW completes its execution and returns to the mainline FOCAL program.

The first nine lines of code in Fig. 1 identify routines that are part of FOCAL, and are explained in the handbook. The FNEW routine itself performs the series of tasks illustrated by the flowchart accompanying the listing. The instructions between ARG and MCOMMA are used to determine whether more arguments from the FOCAL mainline program must be read into LIST. An important location to note is ANS. ANS is set to zero when FNEW is entered, and its value is returned to FOCAL as the value of FNEW when the subprogram is finished. As illustrated below, the user can vary the value of ANS by providing appropriate instructions in the FOCAL mainline programs, thus making ANS valuable as a signal, a switch, or a storage location for information to be used by the mainline program. The patches listed at the bottom of Fig. 1, which allow FOCAL to run without interrupts and eliminate the colon and equal sign, are described on pp. 11-59-11-60 in *Programming Languages* (1972, Vol. 2).

The call to the FNEW subroutine from the FOCAL mainline program is

$$v = FNEW (arg\ 1, arg\ 2, \ldots, arg\ N)$$

where v is any variable name in FOCAL and the args 1-N are 4-digit decimal translations of the PDP-8 octal machine-language instructions the user wants to execute. The largest value for N depends upon the number of free locations set aside for the LIST in FNEW. Although the upper limit is 10 in the sample listing in Fig. 1, more locations may be added at the user's discretion.

As an example of the modified FNEW's use, assume that we have a computer equipped with an interface that can turn an electrical device on and off (e.g., a light stimulus, tape recorder, slide projector, etc.) by machine-language instructions, and can also sense the closing of a circuit by a toggle switch, keypress, or similar means. Assume also the following PDP-8 machine code for operating the stimulus circuit and sensing the flag which indicates that the response circuit has been activated (see Table 1). The first three octal instructions, which operate the interface, are determined arbitrarily for any given computer when the interface is installed. The code for the PDP-8 instruction ISZ ANS depends, of course, upon the location of ANS in the FNEW routine. The listing in Fig. 1 shows the address of ANS to be 5351, so the code 2351_8 is the valid ISZ ANS instruction for this listing. If FNEW is stored at other starting addresses, the octal code for ISZ ANS must be changed correspondingly.

Table 1
PDP-8 Machine Code

Octal Code	Decimal Equivalent[3]	Function
6311	3273	Turn on stimulus
6312	3274	Turn off stimulus
6314	3276	Skip next instruction if response flag is raised
2351	1257	ISZ ANS (increment ANS by 1)

Using these instructions with FNEW, the following simple FOCAL program will turn on a stimulus and repeatedly print the message "NOT YET" on the Teletype until the response key is pressed, at which time it will turn the stimulus off, print the message "NOW," and stop.

```
1.1 SET Q = FNEW (3273)
1.2 IF (FNEW (3276, 1257)) 1.5, 1.5, 1.3
1.3 TYPE "NOT YET",!
1.4 GO TO 1.2
1.5 SET Q = FNEW (3274)
1.6 TYPE "NOW",!
1.7 QUIT
```

In analyzing the program, note that all arguments in each FNEW call are decimal equivalents of the desired octal code. Also, FNEW will always return a value which, even though it may not be used, must either be assigned to a variable (e.g., Q in Lines 1.1 and 1.5) or accounted for as a value in a FOCAL instruction (e.g., IF in Line 1.2). Specifically, in Line 1.1, the FNEW routine turns on the stimulus ($3273_{10} = 6311_8$ = Turn on stimulus), and then returns a value of $\emptyset$ that is assigned to Q. In Line 1.2, the first argument in FNEW asks if the flag on the response circuit is raised ($3276_{10} = 6314_8$). If it is, the next instruction is skipped; so FNEW ends by returning to the IF statement with a value of $\emptyset$. If the response key has not yet been pressed, the flag is not raised, so the next instruction in the FNEW argument is executed. This instruction (1257_{10}) is ISZ ANS, which increments the value of ANS from $\emptyset$ to 1. Consequently, FNEW returns the integer 1 to the IF statement in Line 1.2. Thus, the argument for IF will be a positive integer 1 until the key is pressed, at which time it becomes ϕ. The remainder of the program is straightforward; Lines 1.2-1.4 constitute a testing loop until the argument for IF becomes $\emptyset$, after which the stimulus is turned off (Line 1.5) and the program finishes.

Another basic operation required in laboratory control is accurate timing of a stimulus display. The

following FOCAL program is designed to turn on a stimulus circuit for T sec and then to turn it off:

```
1.1 TYPE "ENTER T",!; ASK T
1.2 SET T = -T
1.3 SET R = .0103
1.4 SET Q = FNEW (3273)
1.5 SET T = T + R
1.6 IF (T) 1.5, 1.7, 1.7
1.7 SET Q = FNEW (3274)
1.8 QUIT
```

The user may enter any value for T to the nearest .01 sec. The stimulus circuit is activated in Line 1.4, and timing takes place via the conditional loop at Lines 1.5-1.6. The accuracy of timing will vary depending upon the value assigned to the variable R. Using a 4K PDP-8/L, the writers obtained accuracy within ±.02 sec when R = .0103 and T ranged between .25 and 10 sec. Other machines and other time ranges may require slightly different calibrations, but the program can easily be "tuned" for accuracy by altering the value of R in Line 1.3.

This level of timing and accuracy, while appropriate for some types of research, is of course inadequate for experiments requiring timing in milliseconds. For milliseconds timing, more sophisticated software clocks that employ delay loops can be written in machine language and then embedded in a FOCAL program by using the FNEW routine. The writers are currently experimenting with a software clock consisting of 16 machine-language instructions that can be used in a slightly larger FNEW routine than that listed in Fig. 1. When properly calibrated, it provides accuracy within ±.002 sec for times ranging between .010 and 4.000 sec.[4] Also, the FNEW routine can be used to provide program control for real-time crystal clocks with millisecond accuracy. (These can be added as peripheral equipment to the PDP-8.)

With a combination of the techniques shown in these examples, it is not difficult to write more sophisticated programs that control multiple S-R channels, monitor response times, collect and print out trial-by-trial performance, and so on. The only real limitation on this usage of FOCAL is memory size—i.e., since FOCAL itself takes up some memory, the allowable size of the program written in FOCAL on any given machine will not be as large as that which could be written in assembly language. Even so, we have been able to put into a 4K machine (equipped with a real-time crystal clock) a FOCAL program that controls a discrimination-learning experiment in which four Ss (fish) can be run simultaneously at four different experimental stations. The computer times and controls presentation of both light and shock stimuli, and records response times (correct to ±.1 sec), independently for each S. In another context, the FNEW routine has been used to control a digital-analog converter in an experiment conducted by the Colgate University Physics Department. With larger memory, the potential is limited only by the programming skills of the user. The required skills need no longer be confined to machine or assembly language, but can be developed more easily in a higher-level language that students and researchers can quickly master.

REFERENCES

Restle, F., & Brown, T. V. A computer running several psychological laboratories. Behavior Research Methods & Instrumentation, 1969, 1, 312-317.

NOTES

1. FOCAL is a copyrighted trademark of the Digital Equipment Corporation.
2. Actually, the handbook offers procedures by which up to three such routines may be overlayed. The procedures are complex, however, and require a listing of the entire FOCAL program in order to place the routines accurately in available core (p. 11-47 of handbook).
3. An octal-decimal conversion table is available in DEC's *Small Computer Handbook* (1972, Appendix H).
4. Listings and documentation for the current version of the microsecond software clock are available on request.

(Received for publication June 28, 1974;
revision received July 30, 1974.)