Extending Forth in a Camac Controlled Muon Channel

Robbie Spruit
TRIUMF
Vancouver, B.C.
Canada, V6T 2A3

Abstract

Control and diagnostic software was developed for a recently commissioned muon channel at TRIUMF. Logistics gave rise to separate efforts in several programming languages. This paper describes the Forth diagnostic package. The choice of programming language is discussed briefly. Several extensions to Forth, and their usage, are shown in the framework of a detailed account of the software implementation. Emphasis is placed on the production of readable code and on the design of constructs that closely model the structure of the application.

Preface

When the M15 channel at TRIUMF delivered its first muons, the very first channel settings had been established with the aid of a small Basic program. Subsequent beam tuning was done with a set of Forth routines, until the adaptation of an existing control program, written in C for another channel, was completed.

The Basic program was coded in six hours. It was less than two pages and permitted the checking of the cabling and of the computer access to the power supply interfaces.

The Forth application routines were developed in three weeks, took seven pages and allowed full control and diagnostics.

Adaptation of the C program took a month. This involved adding a few pages to the existing fifty and some restructuring necessitated by porting to a different operating system and compiler. This package was not meant for diagnostics. It features a channel definition language and a level of user interface not addressed by the other programs.

The efforts in the different languages varied widely in scope. They were not undertaken to study the comparative merits of programming languages but rather to do a specific job with the tools at hand.

This paper describes the Forth program used for channel tuning and hardware diagnostics. On several occasions modifications or extensions of the Forth language were introduced to make the code more readable.

Choice of Language

Clearly, if the task is simple enough, the choice of programming language is less important and a simple language like Basic can be very effective.

Forth excels in flexibility and speed of implementation but it lacks the wide acceptance and standardization of C. The choice between Fortran and Forth on technical grounds can be quite clear: Fortran for math, Forth for control. Include the programming language C and the arguments are less evident. C is well suited to either type of task.

The advantages of Forth, interactiveness and structural extensibility, were not sufficient reason to rewrite existing application software. However, when setbacks in the acquisition and installation of the system software began to jeopardize the timely completion of the C approach, a parallel effort was started in Forth.

A version of CalTech Forth [2] had already been used on site to run FASTBUS test software [4] under RSX11-M on a PDP 11/34. Since the control system for M15 was to employ a Micro-11 with RSX11-M, there were no problems with the installation.

Beam Line Overview

TRIUMF is the name for Canada's meson facility in Vancouver, British Columbia. It is used for pure research in nuclear and particle physics as well as for applied research programs such as: a) the treatment of cancerous tumours with pion beams, b) the production of medical radioisotopes and c) the use of neutron beams for geological analyses. TRIUMF is operated by the universities of Alberta, British Columbia, Victoria and Simon Fraser under a contribution from the National Research Council of Canada.

Figure 1 shows the layout of the 147 m long main building. The six segment cyclotron, 18 m in diameter, allows the simultaneous extraction of multiple proton beams at different energies of up to 520 MeV and 140 uA. Two targets, placed in the path of proton beam line 1, are the sources for a total of six secondary beams of pions and muons.

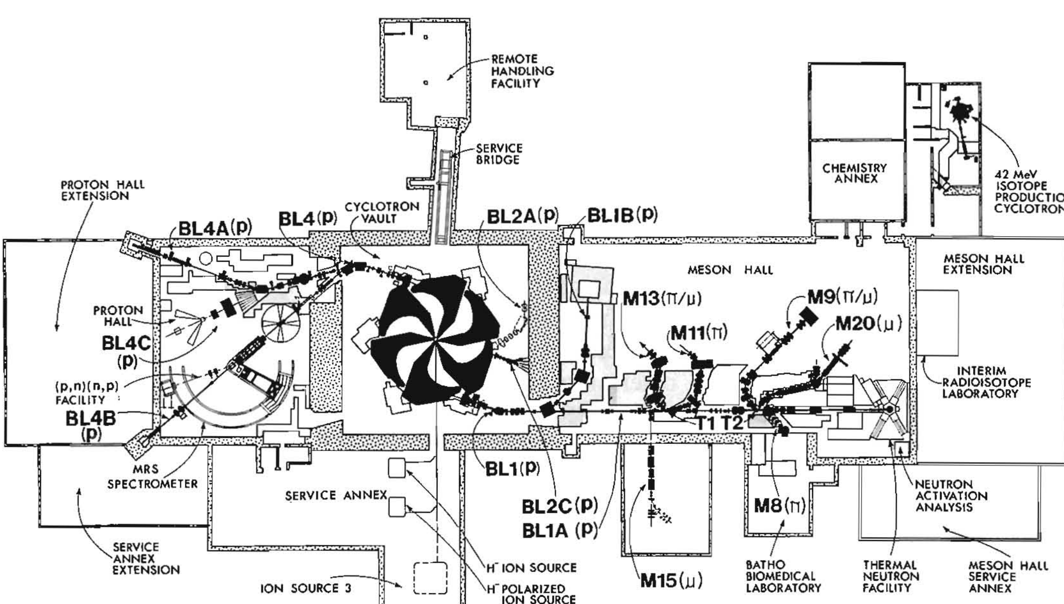

Figure 1. Floor plan of the TRIUMF cyclotron building

M15 is a dedicated "surface" muon channel. It collects positively charged muons from pions decaying at rest within a few microns of the meson production target's surface. Surface muons are longitudinally spin polarized and may be collected into beams of high optical quality. These properties are exploited in two categories of experiments: measurements of muon decay to test modern theories of particles, and muon spin rotation experiments to test physical and chemical phenomena quite unrelated to nuclear or particle physics. For example, the muon is a sensitive probe of magnetism in solid state crystals.

A more detailed mechanical layout of the channel is given in Figure 2. The controllable elements shown are dipole magnets (benders), B1 to B4, which steer the beam and quadrupole and sextupole magnets, Q1 to Q17 and SX1 and SX2, which are arranged in pairs or triplets to focus the beam. The two DC separators were not included in the initial installation. Movable slit plates (not shown) select spectral qualities such as divergence, momemtum range, intensity and spot size.

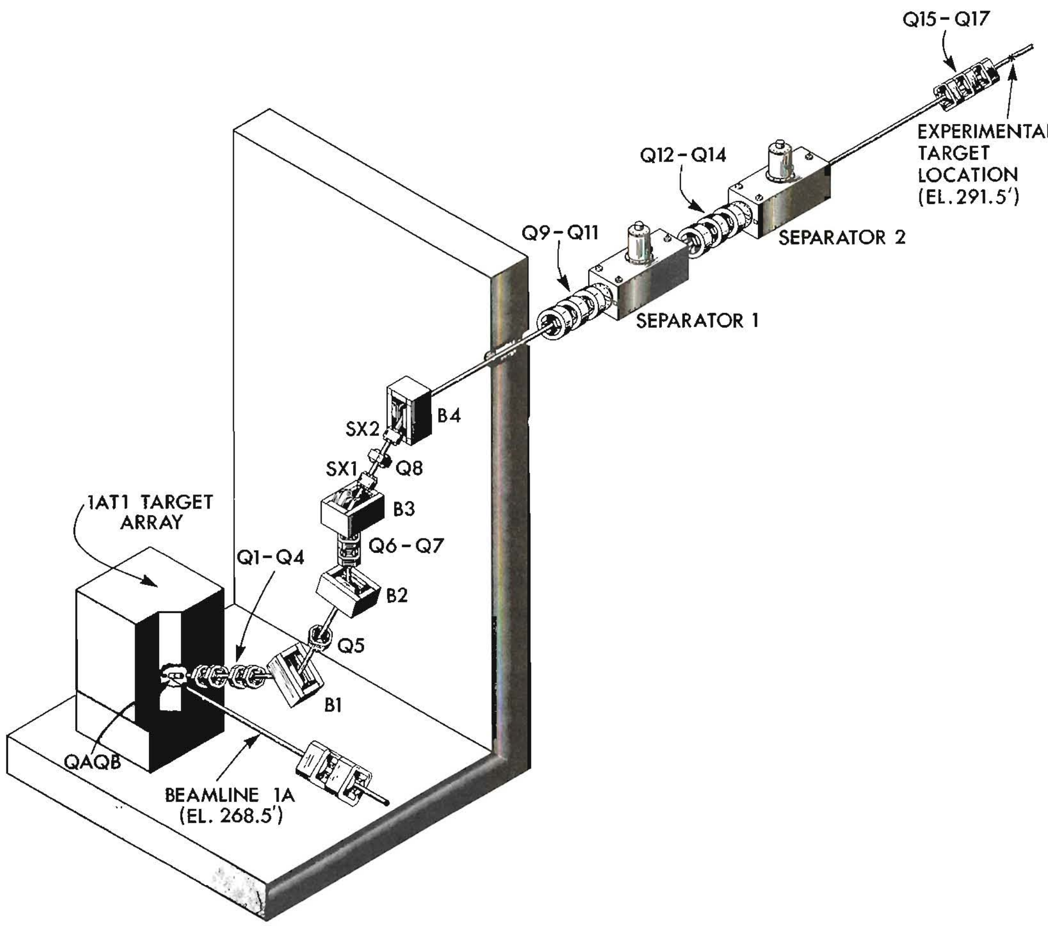

Figure 2. Muon channel M15

Channel Control

Remotely controllable power supplies provide direct currents of up to 750 Amperes to the magnet coils. Local interlock circuitry monitors such conditions as magnet temperature and coolant flow and automatically switches off the corresponding power supply when necessary. Each power supply has an analogue input to set the current regulator, an analogue output to monitor the actual output current, digital outputs indicating on, off and interlock status and digital inputs to switch power on and off and to allow a reset of interlock faults. D/A converters, A/D converters and digital I/O modules housed in a Camac [1] crate provide for computer control as indicated in Figure 3.

The slits are positioned with AC motors, driven by a microprocessor in the same Camac crate, and interfaced through similar A/D and digital I/O modules.

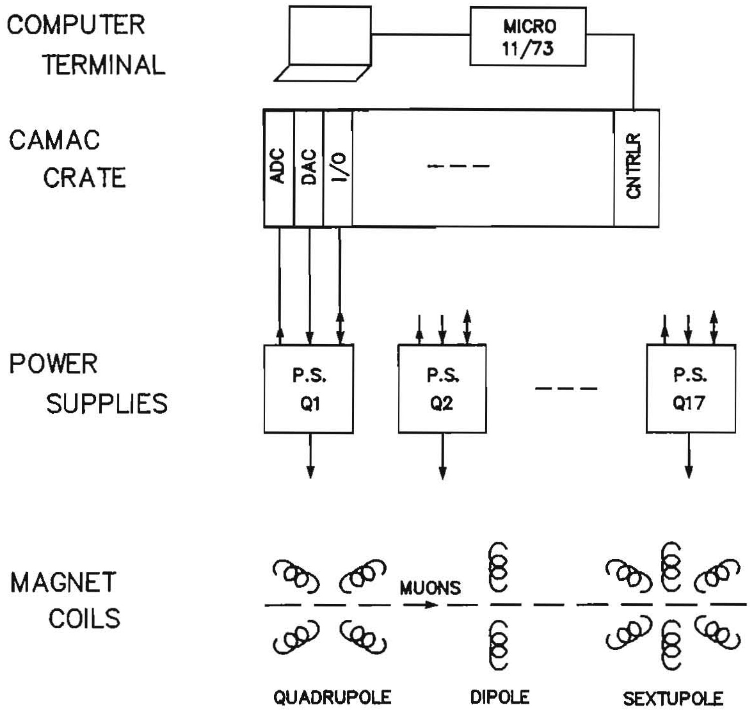

Figure 3. Camac interface connections

Camac, – the phrase 'Computer Automated Measurement And Control' has been adopted to make it an acronym –, is the name for an instrumentation bus standard which originated in the nuclear science community.

Extensive experimental calibration (tuning) is required to obtain a beam with particular characteristics. A given beam line tune is best represented by a set of magnetic field strengths. Monitoring the actual fields with probes may be costly, difficult or impossible. The value given to the D/A converter that sets the magnet current may provide a suitable measure. Such is the case for M15, so that tunes are normally described by a set of DAC settings.

Automatic tuning has, for economic and technical reasons, only recently been given serious consideration. Integration of the beam line control system with the detector and data acquisition systems is still in the planning stage.

The type of control considered here is the provision of convenient facilities for setting and monitoring the state of a number of beam line elements. The interpretation of the detector readout and the choice of parameters for the elements are operations performed by a beam line physicist.

Forth Implementation

The source code of the application programs is listed in the appendix. The file named **LOAD** loads the following files (**VT100**, **CAMAC**, **CAMEM**, **DIGI**, **POWER**, **SLITS**, **CONFIG** and **USER**) on top of the modified CalTech RSX Forth.

The file **CAMEM** deals with access and diagnostics for a special Triumf Camac memory module, while in **SLITS** motor control is passed to a microprocessor via a protocol through such a memory module. These two files have been left out in order to avoid an encumbrance of site dependent trivia. The type of features they offer and the programming principles they depend on are equally well, or better, explained with the files **DIGI** and **POWER**.

The conceptual design began with a tabular representation of the elements of the muon channel. This became the file **CONFIG**. Then the notions of what one wanted to do with the elements were put in the file POWER, for magnet power supplies, and in the file SLITS for the moveable slit plates. The files VT100 and CAMAC are general utilities. The user interface in the file USER was held to a minimum since eventually this would be handled by the existing C software.

The following narrative of the principal design modules is presented in the order in which the files are loaded.

Terminal

Initially, basic control was developed for a hardcopy terminal mode in which standard Forth terminal I/O is adequate. Later, software was added to support a 24×80 character video display.

A set of cursor commands is grouped in a file named after the terminal type, in this case a VT100. The word ESC sends an escape sequence made up of the next word and preceded by an escape character. A generic compilation construct was devised to allow the creation of words such as ESC, which differ only slightly from regular string output but would normally be awkward to implement as they are to be used inside as well as outside of definitions. ESC and ." are defined as follows:

```
: ESC   BL ($) 33 EMIT WRITE ;

: ."    & " ($) WRITE
```

The definition for ($) was a bit tricky, since it has to cause the word in which it is used to be state sensitive, but the result is satisfying: it makes it easy to define some very useful words. ESC makes it possible to code terminal dependent escape sequences in a format that is identical to the specification in the user's manual. The effect of the sequences can be checked interactively, with no need for additional definitions.

Camac

A subaddress (A) in a slot (N) in a Camac crate is declared as an addressable entity to which a maximum of 32 I/O function codes (F) may be applied.

The brevity of the routines presented in the listing derives from a number of simplifications. All status is polled so interrupt handling is not necessary. Direct access to the memory mapped I/O page, and the resulting compromise in operating system security, is acceptable. There is only one crate. Multi-branch, multi-crate addressing was not needed.

The minimum functionality required for this application, a 16-bit read and a 16-bit write, could be coded in a few lines. The facilities provided here are used in general non-interrupt Camac applications. Camac error messages can optionally be directed to specific fields on the screen, using the message facility defined in the terminal file.

The Channel

The description of the channel configuration in terms that suit computer control can be done with lists or tables showing the hardware (Camac) layout. Software to read and interpret such lists creates a program data base. Control routines, to be invoked by operator commands, can then be written to act on these.

The option of 'intelligent constructs' in Forth allows for a particularly elegant presentation. The central notion is to treat a beam line element as an active entity, characterised by configuration parameters and by the functions that are expected of it.

Allocation and initialization of configuration and working parameters was done in the CREATE part of a class define construct (named PS:) and the functionality of the various command options in the DOES> part.

Functions

Standard Forths define a variable as a routine that pushes an address on the stack. The value located at this address may then be read (by @) or written (by !). A suggestion by Charles Moore led to the 'smart variable' which would return its value rather than its address unless preceded by the word TO in which case it would take on a new value from the stack [3]. The word VAR: is used to define such variables in this version of Forth. Its implementation relies on a state variable, set by TO and reset by the variable.

Extending this concept to our channel elements leads to software designations of elements that return a value unless told to take on a new value by a 'prefix operator' [5]. The selection of different functions, such as +TO for incrementing, is implemented using different values of the same state variable. For power supplies, the words ON, OFF and RESET are treated in the same fashion as TO and +TO.

For diagnostic purposes it is useful to change the meaning of an element's value. The words DAC and ADC indicate that the power supply values are to refer to the setting of the D/A converter or to the measure of the actual output current obtained through A/D conversion. The words AMPS and COUNTS indicate whether the values are measured in Amperes or in DAC or ADC counts.

When the interpretation of a value is in doubt, one can always type it explicitly. For example:

```
25 AMPS TO Q1   35 TO Q2   ADC Q1 . Q2 .
```

would check the setting of currents to the first two quadrupole magnets.

The words TO, +TO, ON, OFF and RESET refer to a single power supply. The state variable they affect is implicitly and immediately reset. The words DAC, ADC, COUNTS and AMPS explicitly set or reset a state and apply to any number of power supplies.

There are instances where one may want to issue a command for a group of power supplies. For example, once a tune has been established for a momentum of 30 MeV/c, the tune for 27 MeV/c may be obtained by scaling all fields down by 10 percent. Status display is another example of a command that could apply to all supplies. To arrive at a tune it is necessary to sweep selected groups of magnets through certain momentum ranges.

For these cases command names were chosen with a left parenthesis as the last character. These assign a certain value to a state variable which keeps its value until reset explicitly by a right parenthesis. Thus

`?( B1 B2 B3 B4 )` shows the status of the first four benders

and

`-10 S( B1 B2 )` scales the first two down ten percent.

The implementation of the TO concept originally used simple values for the state variable, 0 as the default value, 1 for TO, and 2 for +TO. With the proliferation of command options, this method began to stand out as an example of poor software practice: defining the same association in more than one place and hoping that the definitions agree.

A 'switch' class define construct (named SW:) was implemented as a remedy. It allows the prefix operators to be defined such that they switch a state or fuction variable to some unique value, in this case the parameter field address. As can be seen in the definition of PS: in the file POWER, comments are no longer required to identify the commands.

Parameters

In a CREATE DOES> construct the parameter field address is available on the stack when the part following DOES> is executed. The individual parameters can then be retrieved by applying offsets to this address.

For power supplies, the large number of parameters required to define their state dictated the creation of a naming convention. The initial approach was to store the parameter field address on entry after DOES> in a variable, named PAR.

For each parameter then a word was written to access it, e.g

```
VAR: PAR
: DAF   PAR 4 + ;
: ADF   PAR 8 + ;
```

were used to get the address of the full scale values of the DAC and the ADC. These definitions evolved into variables of the form:

```
' PAR 4 PAR.OF: DAF
' PAR 8 PAR.OF: ADF
```

where PAR.OF: was such that the parameters now worked with prefix operators, which looked well since read access was much more common than overwriting.

In the course of development, the number and order of parameters was changed a few times. Each change required an edit of the offset literals. This was not difficult but there was an awareness of something not being quite right. PAR.OF: was replaced by 'underbar colon', which needs no arguments. It assumes it is being used in the context of a parameter list accessed via a pointer, called PAR_. The name change of PAR to PAR_ reflects the change in type, from VAR: to PNT:.

Not only was the readability of the parameter declarations greatly improved hereby, the way was opened up for a similar improvement in parameter allocation and initialization. The pointer declaration, PNT:, was extended to mark the beginning of a data structure, with subsequent parameter declarations increasing the size. The word ALLOCATE, used in the CREATE part of a CREATE DOES> construct, allots space for the parameters and makes them accessible by name. Thus, allocation and initialization no longer require knowledge of offsets or order of declaration.

User interface

An on-line help facility was added and the software was installed to come up automatically with a continuous status display after logging in to the operating system.

Help consists of a list of the most common command options, the last of which is the command to reinvoke the display. Thus there is always an indication of what options are available. A minimum of typing skill is required to switch between display and command mode. When in command mode, the user has access to the entire Forth command set. This is where this implementation stops being user friendly. Logically, CalTech Forth's entire dictionary, the assembler included, is just one long list. Fatal results, even if unlikely, are possible by mistyping.

Early Forth systems have always been criticized for such surprises. There are a number of possible preventive programming measures. None were pursued, since power prevailed over protection, and time was of the essence.

Notwithstanding its known drawbacks and pitfalls, this simple user interface is highly effective, requires a minimal development time, and is immediately accessible to unfamiliar users without hindering the more experienced.

Conclusions

The description of the Forth application program made it possible to present some language constructs, data and function structures or pseudostructures 'in real life' as extensions to the Forth language.

A concise style of implementing functional descriptions is achieved when individual references to parameters or structure members may be made without having to refer explicitly to the structure itself.

Coding prefix format commands by name permits a more readable implementation of constructs that make use of such operations.

References

[1] Modular Instrumentation and Digital Interface System (CAMAC) *ANSI/IEEE Std 583-1975.*

[2] M.S. Ewing, *The CalTech Forth Manual,* June 1978. A Technical Report of the Owens Valley Radio Observatory, California Institute of Technology, Pasadena, Ca 91125.

[3] P. Bartholdi, The TO solution, and 'TO' continued, *FORTH Dimensions,* Vol. 1, No. 4/5, 1979.

[4] C. Logg, Fastbus Diagnostic Operating System (FBDOS), Aug 1982. Informal paper, Stanford Linear Accelerator Center, Ca 94305.

[5] K. Schleisiek, Multiple Code Field Data Types and Prefix Operators, *Jrnl. of Forth Appl. & Res.* Vol. 1, No. 2, Dec 1983.

Manuscript received June 1985.

Robbie Spruit, P.Eng., M.Sc.(Eng) Delft, learned about Forth in 1976, when working on data acquisition and instrument control systems for an international telescope construction project. As a member of the Forth Standards Team, he took part in the definition of the Forth-79 standard. Mr. Spruit is an independent consultant, based in Vancouver, B.C., Canada, with particular experience in control, communication and interface systems in engineering and specific environments. Among his current interests is the application of computer systems to natural language services.

Appendix

CalTech Forth

CalTech RSX Forth's direct threaded code, its sixteen thread dictionary, compressed name fields and sequential source files make for extremely fast compilation. This compensates for the lack of a Forth editor. To make a change in a source file one has to exit Forth, invoke the system editor and rerun Forth. It is possible to run a system program from within Forth but the entire process of reloading took only a few seconds so there was no pressing need to implement this feature.

A useful feature is the validity of program flow constructs outside of definitions. It is more convenient to type `20 0 DO READ . LOOP` to show the result of twenty read actions than to have to go through the sequence of encasing this phrase in a new definition, executing it once and `FORGET`ting it.

A disadvantage of CalTech Forth is the divergence from the more widely used versions of Forth. We did change it, but rather than making a rigorous conversion to a Forth 79 or 83 standard we made modifications as required to be able to execute code that looked like standard Forth. A few of the nonstandard words listed below are CalTech's. Any inconsistencies are ours.

Non-standard words

`FLIST <filespec>`	List the specified file.
`FLOAD <filespec>`	Load, i.e. start interpreting, the specified file.
`LIST ( a n -- )`	List the file whose n char. filespec starts at a.
`LOAD ( a n -- )`	Load a file, e.g.: `CAMAC LOAD`.
`>FILE <filespec>`	Create a file and direct standard output to it.
`>TER`	Redirect standard output to the terminal.
`?TER ( -- c )`	Get a keyboard input character, zero if none.

`*/R ( a b c -- r )`	Return rounded result of a*b/c.
`BIT ( n -- v )`	Raise 2 to the power n.
`CON: ( n CON: <name> -- )`	Define a (direct code) constant (can't be changed).
`ESC <string>`	Output an escape sequence or compile what's needed to do it.
`RANGE ( n a b -- n f )`	True if n is in the range a,b (inclusive).
`SHIFT ( n1 n -- n2 )`	Shift n1 n bits left or, if n < 0, −n bits right.
`\`	Treat rest of line as a comment.
`O. ( n -- )`	Show integer value in octal.
`T. ( n -- )`	Show a number in base 10.
`.R ( n w -- )`	Show in current base, right adjusted in a field of width w.
`O.R ( n w -- )`	Octal output, right adjusted in a field of w characters.
`T.R ( n w -- )`	Decimal output, right adjusted.
`TIME.`	Show time of day.
`DATE.`	Show date.

Extensions

`@VAR ( -- n )`	Get state for `TO`-variables and clear it.
`VAR: <name>`	Define a `TO`-variable.
`DVAR: <name>`	Define a double `TO`-variable.
`SW: ( a SW: <name> -- )`	Define `<name>` such that it stores its pfa in a.
`PNT: <name>`	Name and start a parameter list.
`_: <name>`	Define a single integer parameter variable.
`_D: <name>`	Define a double integer parameter variable.
`ALLOCATE ( a -- )`	Allot space for and redirect the list identified at a.
`WHILE`	As in Forth-83, but `REPEAT` allows any number of `WHILE`s.
`CASE`	Equivalent to `OVER = IF DROP`.
`C.ERR ( n -- )`	Abort in a bad case, show n and error message.
`ENDS`	End nested `ELSE`s. Replaces any number of `THEN`s, but not those that bracket an `IF` or `CASE` clause without `ELSE`.

Listing 1

An implementation of the `VAR:`, `SW:` and `PNT:` extensions follows. Address and assembler conventions are specific to this version of PDP-11 Forth.

```
CODE @VAR   0 ( no-op) TST,   S-) 0 # MOV,   ' @VAR @# CLR,   NEXT,

1 ' @VAR SET TO

2 ' @VAR SET +TO

: VAR:          \ <name> ; define an integer 'TO-variable'.

          CREATE 0 , DOES> @VAR
          0 CASE @  ELSE
          1 CASE !  ELSE
          2 CASE +! ELSE C.ERR ENDS ;
```

```
: DVAR:        \ <name> ; define a double integer 'TO-variable'.

         CREATE Ø , Ø , DOES> @VAR
         Ø CASE D@  ELSE
         1 CASE D!  ELSE
         2 CASE D+! ELSE C.ERR ENDS ;

: SW:     \ a SW: <name> -- ; define a function switch that applies to a.

         HERE 8 + ( pfa of word to be defined) SWAP SET ;

VAR: @PNT     \ @PNT holds the address of the pointer defined with PNT:.
              \ This implementation does not allow nested structures.

: PNT:        \ <name> ; Define a pointer to a parameter list.
              \ The pointer itself, when invoked by name, leaves the
              \ address at which can be found 1) the address, 2) the size of
              \ the list.

         CREATE   HERE TO @PNT   Ø ( address),   Ø ( size),   DOES> ;

: @PNT,       \ n -- ; in the declaration of a parameter of size n, set
              \ up the address and offset, and update the size of the list.

              @PNT DUP ( address),  2+ DUP @ ( offset),   +! ( update size) ;

: _:          \ <name> ; define an integer parameter variable.

         CREATE 2 @PNT, DOES> D@ @ + @VAR
         Ø CASE @  ELSE
         1 CASE !  ELSE
         2 CASE +! ELSE C.ERR ENDS ;

: _D:          \ <name> ; define a double integer parameter variable.

         CREATE 4 @PNT, DOES> D@ @ + @VAR
         Ø CASE D@  ELSE
         1 CASE D!  ELSE
         2 CASE D+! ELSE C.ERR ENDS ;

: ALLOCATE    \ a -- ; assume 'a' to be a pointer to a list.
              \ Initialize the pointer and allot space for the list.

         HERE OVER !   2+ @ 1+ 2/ Ø DO Ø , LOOP ;
```

Listing 2

Source files

Each of the source files listed below starts with a comment line (in parentheses), in which the first word is the filename.

```
( LOAD  files for diagnostics on M15)

     fload VT100
     fload CAMAC
     fload DIGI
     fload CAMEM
     fload POWER
     fload SLITS
     fload CONFIG
     fload USER

( VT100 terminal dependent cursor addressing)

: HOME  ESC [H ;
: XY    ESC [ T. ." ;"T. ." H"; \ x y -- ; move cursor to col x,
                                \ row y
: CLRS  ESC [J ;                \ Clear screen from cursor
: CLRL  ESC [K ;                \ Clear from cursor to end of line
: UP    ESC [A ;                \ Move cursor up
: BKSP  ESC [D ;                \ Move cursor left

DVAR: XYMES  0 -1 TO XYMES      \ x,y for messages, ignore if y<0

: M"    \ Use instead of ." to direct messages to XYMES
        & " ($) XYMES DUP 0> IF XY ELSE DDROP THEN WRITE ;

( DIGI  bit assignments of TRIUMF's I/O module)

-1 CON: 'NODIGI'    \ used to fake status when a digi module is not
                    \ provided

: BS?:  CREATE BIT , DOES> @ OVER AND ;  \ Define a 'bit set?' test
: BIT:  BIT CON: ;

\ input bits
                               when reset:
        0 BS?: 'OK'               \ Interlock fault
        1 BS?: 'ON'               \ Power Off
        3 BS?: 'REMOTE'           \ Local

\ output bits

        0 BIT: PWR-ON
        1 BIT: PWR-OFF
        2 BIT: INTLK-RESET
```

```
( CAMAC for PDP-11 with a Kinetics 3912 crate controller)

OCTAL 166000 CON: CAMAC \ Crate's address
DECIMAL

: NA   \ n a -- na ; encode slot & subaddress into Unibus address
        0 15 RANGE 0= ABORT" Subaddress must be 0-15 " SWAP
        0 30 RANGE 0= ABORT" Slot nr. must be 0-30 "
        4 SHIFT + 1 SHIFT CAMAC + ;
: N:    CON: ;      \  n -- ; define a slot (module) no.
: RD:   CODE S-) SWAP @# MOV, NEXT, ;  \ na -- ; define an input action
: WR:   CODE @# S)+ MOV, NEXT, ;    \ na -- ; define an output action

0 0 NA RD: RD        0 0 NA WR: WR     \ Data bits 15 -  0
0 1 NA RD: RD-HI     0 1 NA WR: WR-HI   \ Data bits 23 - 16
0 4 NA RD: ST1       0 5 NA RD: ST2     \ Status register 1 & 2

VAR: #NA     \ provide global access to last used N, A and
VAR: #F                    \                           function code

: =F    TO #F TO #NA  #F #NA B! ;   \ na f --  ; execute a single action
: X      ST2 2 AND 0= ;              \ -- x ; True if 'X' was generated
: Q      ST2 1 AND 0= ;              \ -- q ; True if 'Q' was generated

OCTAL
: NA>    2/ DUP -4 SHIFT 37 AND SWAP 17 AND \ na -- n a
: NA.    NA> SWAP  2 T.R  3 T.R ;     \ decode and show n and a
: NAF.   ." NAF: "  #NA NA.  #F 3 T.R SPACE ;  \ show last n, a and f
: X?   2 AND IF M" No X, " NAF. THEN ;
: Q?  ST2 IF       ST2 4 AND IF M" Time out,"ELSE
                   ST2 2 AND IF M" No X,"    ELSE
                   ST2 1 AND IF M" No Q,"    ELSE ENDS NAF. CR THEN ;
DECIMAL
: F:    CREATE  0  7 RANGE IF   , DOES> @ =F RD Q? ELSE
      16 23 RANGE IF  , DOES> @ ROT WR =F Q? ELSE
       0 31 RANGE IF  , DOES> @ =F X?       ELSE
          1            ABORT" F-code must be 0 to 31 " ENDS ;

\    ( na -- data)      ( na -- ) ( data na -- ) ( na -- )
\       'read'          'operate'   'write'  'operate'
        0 F: RD1         8 F: TLM        16 F: WT1  24 F: DIS
        1 F: RD2         9 F: CL1  17 F: WT2 25 F: XEQ
        2 F: RC1        10 F: CLM  18 F: SS1 26 F: ENB
        3 F: RCM        11 F: CL2  19 F: SS2 27 F: TST
```

```
\ 16-bit data diagnostic for a single subaddress:

: W=R?\ n1 n2 -- n1 ; complain if different
       DDUP - IF OVER M" Wrote:"6 O.R ."  read:"6 O.R NAF. CR
       ELSE DROP THEN ;

: CHKD\ n1 -- n1  ;check write read on current N and A
       DUP #NA WT1   0 WR   #NA RD1  W=R? ;

: CHK-DATA \ na -- ; Check single one's and single zeroes
       TO #NA 1 BEGIN CHKD COM CHKD COM ?DUP WHILE 2* REPEAT ;

( POWER          Control magnet power supplies)

DVAR: PXY     0 3 TO PXY     \ Starting point of first display column
DVAR: PXY2   46 3 TO PXY2    \     "                 second    "
 VAR: PCT    \ Percentage increase for scale command
 VAR: ADC?   \ FLag to get ADC rather than DAC setting
 VAR: AMP?   \ Flag to work in AMPS rather than COUNTS
 VAR: #AD    \ Temporary store for   a/d
 VAR: #DA    \   "                   d/a
 VAR: FUN    \ Function to be performed

' @VAR SW: ON     \ Extend use of TO variables
' @VAR SW: OFF
' @VAR SW: RESET
 ' FUN SW: SCA(   \ Scale by PCT percent
 ' FUN SW: R(     \ Initialize setpoints by reading actual DAC setting
 ' FUN SW: ?(     \ Show DAC, ADC and Status
 ' FUN SW: P(     \ Show parameters
 ' FUN SW: RD(    \ Load setpoint value from next word in input stream
 ' FUN SW: WR(    \ Write ID and DAC value to output stream.
 ' FUN SW: )      \ Revert to standard mode
                  \ for TO, +TO, ON, OFF and RESET
: S(    TO PCT SCA( ;

: ADC    1 TO ADC? ;       : DAC    0 TO ADC? ;
: AMPS   1 TO AMP? ;       : COUNTS 0 TO AMP? ;

PNT: PAR_         \ pointer to the parameter list of a power supply
       _D: _XY    \ col and row for display
       _: _SP     \ Value of set point
       _: _FSA    \ Full scale amps x 10
       _: _DAF    \ DAC's full scale (4095 for 12-, 65535 for 16-bits)
       _: _DAC    \ DAC's camac address
       _: _DG     \ DIGI's camac address (ignored if not there)
       _: _ADF    \ ADC's full scale
       _: _ADC    \ ADC's camac address, set bit 0 for ch 16-31.

: _ID.  PAR_ @ ( pfa>nfa) 8 - ID. ; \ Show name of beam line element

: _AD        \ -- n ; read 1 of 32 subbaddresses
       _ADC DUP 1 AND IF 1- RD2 ELSE RD1 THEN ;
```

```
: _C?A        \ n1 -- n2 ; convert counts to amps (if required)
        AMP? IF  0 _FSA M*  ADC? IF _ADF ELSE _DAF THEN
             -1 CASE 32767, D+ SWAP
                ELSE DUP >R 2/ 0 D+ R> M/ THEN DROP    THEN ;

: _A?C        \ n1 -- n2 ; convert amps to counts
        AMP? IF _DAF -1 CASE 0 SWAP _FSA M/ DROP
        ELSE _FSA */R ENDS THEN ;

: _SDAC _DAF DDUP U> IF SWAP THEN DROP             _DAC WT1 ;
: _SET  _DAF DDUP U> IF SWAP THEN DROP DUP TO _SP _DAC WT1 ;

: FS.         \ n -- ; show full scale in number of bits
   -1 CASE 16 ELSE 4095 CASE 12 ELSE 1023 CASE 10 ELSE ENDS 2 .R ;

: %+     100 + 100 */R ;        \ n1 p -- n2 ; add p percent

: %DIFF  OVER IF OVER - 100 ROT */R
              ELSE SWAP DROP THEN ;  \ n1 n2 --- p

: PSHD   AMP? IF   ."      Ampsx10"      \ header for ps status table
              ELSE ."             "THEN ."   DAC   ADC  " CR
                   .'             '     .' (percent off)' CR ;
: _STATUS     _AD TO #AD    _DAC RD1 TO #DA
        _DG ?DUP IF RD1 ELSE 'NODIGI' THEN  \ pretend 'ok' if no digi
        _ID. _SP _C?A 6 U.R
        'REMOTE' IF _SP #DA %DIFF ?DUP
                    IF 6 .R  ELSE ."        "THEN
                 ELSE            ."  Local"THEN
        'OK' IF
        'ON'     IF #DA 0 _DAF 0 _ADF M/ DROP M/ DROP   #AD %DIFF ?DUP
                    IF 6 .R ELSE ."        "THEN
                 ELSE            ."   off "THEN
             ELSE                ."  intlk"THEN CR DROP ;

: _CTRL          \ controlbit --
        _DG IF _DG WT1
           ELSE DROP ." No remote ON/OFF/RESET for " _ID. THEN ;

: PSHL           \ header for configuration table
        ."       ( a/d  bits     digi        d/a  bits    amps     )"CR ;

: _PS.           \ write an entry of the configuration table
        TAB _ADC NA.   TAB _ADF FS.   TAB _DG NA.   TAB  _DAC NA.
        TAB _DAF FS.   _FSA 10 / 8 .R    ." PS:"   _ID. CR ;
```

```
: PS:           \ <config'n par's> PS: <name> -- ; define a power supply
      CREATE  PAR_ ALLOCATE                    \  and initialize param's
              10 * TO _FSA  BIT 1- TO _DAF  NA TO _DAC
              OVER IF NA ELSE DROP THEN TO _DG          \ zero if no digi
              BIT 1- TO _ADF
              16 /MOD >R NA R> + TO _ADC   \ modify naf for 32 ADC chnls
              PXY TO _XY   PXY DUP 18 > IF DDROP PXY2
                                        ELSE 1+ THEN TO PXY
      DOES>
              PAR_ ! @VAR  FUN
      "    ) CASE
                     0 CASE ADC? IF _AD ELSE _SP THEN _C?A  ELSE
              ( to)  1 CASE _A?C _SET                       ELSE
              ( +to) 2 CASE _A?C _SP + _SET                 ELSE
              '       ON CASE PWR-ON _CTRL                  ELSE
              '      OFF CASE PWR-OFF _CTRL                 ELSE
              '    RESET CASE INTLK-RESET _CTRL  ELSE C.ERR ENDS
                                                 ELSE SWAP ( @VAR) DROP
              ' SCA( CASE _SP PCT %+ _SDAC      ELSE
              '   R( CASE _DAC RD1 TO _SP       ELSE
              '   ?( CASE _XY XY _STATUS        ELSE
              '   P( CASE _PS.                  ELSE
              '  RD( CASE ASKN TO _SP           ELSE
              '  WR( CASE _ID. _SP 6 U.R CR     ELSE C.ERR ENDS  ;
```

```
( CONFIG        Camac modules, power supplies and slits in M15)

  1 N: N1   \ GEC model ADC-32, 32-chnl 12-bit analog to digital conv.
  2 N: N2   \       "
  4 N: N4   \ Joerger model D/A-16, dual 16-bit digital to analog conv.
  5 N: N5   \       "
  6 N: N6   \ Joerger model DAC-8L, 8-ch 12-bit digital to analog conv.
  7 N: N7   \       "
  8 N: N8   \       "
 11 N: N11  \ Triumf model 0550 8-chnl 4-bit digital I/O module.
 12 N: N12  \       "     (read  bit 0 intlck, 1 on, 3 remote)
 13 N: N13  \       "     (write bit 0 on, 1 off, 2 reset)
 15 N: N15  \ Triumf model 0576/1, octal 4-bit Input Gate Output Reg.
 16 N: N16  \       "
 17 N: MM   \ Triumf model 2401, 128 24-bit word memory module.
 22 N: DW   \ Kinetics model 3291, dataway display.
\
\a/d  bits    digi       d/a  bits      amps
\
 N1 0  12     N11 0      N6 0  12       750   ps: Q1
 N1 1  12     N11 1      N6 1  12       750   ps: Q2
 N1 2  12     N11 2      N6 2  12        80   ps: Q3
 N1 3  12     N11 3      N6 3  12        80   ps: Q4
 N1 4  12     N11 4      N4 0  16       250   ps: B1
 N1 5  12     N11 5      N6 4  12        80   ps: Q5
 N1 6  12     N11 6      N4 1  16       250   ps: B2
 N1 7  12     N11 7      N6 5  12        80   ps: Q6
 N1 8  12     N12 0      N6 6  12        80   ps: Q7
 N1 9  12     N12 1      N5 0  16       250   ps: B3
 N1 10 12     N12 2      N6 7  12        80   ps: SX1
 N1 11 12     N12 3      N7 0  12        80   ps: Q8
 N1 12 12     N12 4      N7 1  12        80   ps: SX2
 N1 13 12     N12 5      N5 1  16       250   ps: B4
 N1 14 12     N12 6      N7 2  12       200   ps: Q9
 N1 15 12     N12 7      N7 3  12       200   ps: Q10
 N1 16 12     N13 0      N7 4  12       200   ps: Q11
 N1 17 12     N13 1      N7 5  12       750   ps: SEP1
 N1 18 12     N13 2      N7 6  12       200   ps: Q12
 N1 19 12     N13 3      N7 7  12       200   ps: Q13
 N1 20 12     N13 4      N8 0  12       200   ps: Q14
 N1 21 12       0 0      N8 1  12       750   ps: SEP2    \ no digi
 N1 22 12     N13 6      N8 2  12       200   ps: Q15
 N1 23 12     N13 7      N8 3  12       200   ps: Q16
 N1 24 12     N13 5      N8 4  12       200   ps: Q17
\
\    a/d             igor            memory words
\left    right   left    right     mem  pos   width stat   no.
\
 N2  0   N2  1   N15 0   N15 1     MM  816   832   848    1 slit: SL1
 N2  3   N2  2   N15 3   N15 2     MM  817   833   849    2 slit: SL2
 N2  4   N2  5   N15 4   N15 5     MM  818   834   850    3 slit: SL3
 N2  7   N2  6   N15 7   N15 6     MM  819   835   851    4 slit: SL4
 N2  8   N2  9   N16 0   N16 1     MM  820   836   852    5 slit: SL5
 N2 11   N2 10   N16 3   N16 2     MM  821   837   853    6 slit: SL6
```

```
( USER  routines for M15)

\ Examples of use of the diagnostics in CAMAC, POWER and SLITS,
\ serving as a makeshift user interface.

 3 22 to XYMES  \ col, row for camac error reports

: help  " INFO" list ;
: ??    help ;

: mags  q1 q2 q3 q4 b1 q5 b2 q6 q7 b3 sx1 q8 sx2 b4  \ group all
        q9 q10 q11 sep1 q12 q13 q14 sep2 q15 q16 q17 ; \magnets

: slits sl1 sl2 sl3 sl4 sl5 sl6 ;             \ group all slits

: all   mags slits ;                          \ all elements

: scale s( mags ) ;

: show  pshl p( mags cr slhl slits ) cr ;     \ config'n. parameters

: time.hd   73 0 xy time. cr ;                \ show time in header

: nokey  ?ter 13 - ;    \ -- c ; leave 0 only if return was hit

: //    home clrs pshd
        40 23 xy ." Hit return for attention. (rev. Mar 85)" time.hd
        begin ?( mags )  nokey while
              cr mm mtst nokey while
              ?( slits ) nokey while time.hd repeat
        0 21 xy clrs ." Type ?? for help."cr ;

: save  >file ." ( TUNE.M15 " date. space time. ." )"cr   \ file header
              wr( mags cr slits )                         \ el't settings
        >ter ;

: restore rd( fload ) ; \ restore setpoints from data in file

: svcal >file ." ( CALIB.M15 " date. space time. ."  )"cr
              wm( slits )
        >ter ;

: ldcal rm( " CALIB.M15" load ) ;

r( all )

//
```

```
( INFO  M15 diagnostics.                                    RS/850330 )

Words must be separated by spaces.
Say DAC or ADC, AMPS or COUNTS to qualify subsequent reads and writes.

<value> TO <name>        Set DAC in integer amps x 10 or counts.
<name>                   Print value of DAC or ADC in AMPS or COUNTS.
ON or OFF  <name>        Remotely control power supply for <name>.
RESET <name>             Reset power supply interlock.

MAGS                     Is a name for all power supplies.
<value> S( <names> )     Scale power supplies by integer percentage.
<value> SCALE            Scale all (0 SCALE sets dacs to setpoints).
R( <names> )             Read dac's into setpoints.
?( <names> )             Show setpoint, dac (if different) and adc.
FLIST <filespec>         List a file on the terminal.
SAVE <filespec>          Save settings in a file
RESTORE <filespec>       Restore a tune into the setpoints.
SHOW                     Show configuration parameters.

<pos'n> <width> TO SLx   Set slit x (x=1-6) in .1 mm integers.

BYE                      End the program.

//                       Do display.
```

Sample display shows benders scaled down by 10%. Some DAC-ADC's need adjusting.

```
                DAC    ADC                                   14:56:23
              (percent off)
  Q1      150            5                 SEP1    1010
  Q2      215            1                 Q12     1450       1
  Q3     1475                              Q13     2290       1
  Q4     1400            1                 Q14     1425       1
  B1    34200    -10  intlk                SEP2       0
  Q5     1925            1                 Q15     1330     off
  B2    34700    -10     1                 Q16     2285     off
  Q6     1000                              Q17     1355     off
  Q7     1280            1
  B3    29925    -10     1
  SX1    1080            1                 slit   pos'n   width
  Q8     1350                               SL1       1     700
  SX2    1080                               SL2       0    1001
  B4    30150    -10                        SL3       1     251
  Q9     1225            1                  SL4       0    1001
  Q10    2235            1                  SL5       0     600
  Q11    1250            1                  SL6       1     599

                         Hit return for attention. (rev. Mar 85)
```