DECUS 12 BIT SPECIAL INTEREST GROUP
NEWSLETTER

May Number 22 1977

Contributions and correspondence should be sent to:

Robert Hassinger, Coordinator
12 Bit SIG
c/o DECUS
146 Main Street
Maynard, MA 01754

..or..

Liberty Mutual Research Center
71 Frankland Road
Hopkinton, MA 01748

DECUS/Europe contributions are solicited through:

Lars Palmer
DECUS/Europe 12 Bit SIG Newsletter Liaison
Hassle
Fack
S-431 20 MOLNDAL 1
SWEDEN

(Please include reference to Newsletter number and page when inquiring about material published.)

NEWSLETTER DEADLINE

The deadlines for ready-to-use material for the next two Newsletters are 24-June-1977 and 26-August-1977. Material requiring editing/re-typing must be in earlier. Ready-to-use material should use an area 6 1/2 inches (16.5 cm) wide by no more than 9 inches (23 cm) long on each page. It should be on white bond paper whenever possible and must be reasonably clean, lesible and sufficiently dark for good photographic reproduction.

DECUS/US SPRING SYMPOSIUM

The Spring Symposium should be going on or just done about the time this Newsletter reaches you. Unfortunately, the people who wrote up the advanced program did not realize this and said that this issue would document the open session we have planned to handle current developments. As this issue goes to press it is still too early to be sure of the content of that sesson (I expect that some of it will not be developed until the Symposium starts) and it would be too late if it was published here in any case. The open session time is an effort to deal with the desire for current information exchange at a meeting that must be cast in concrete months in advance. The problem is particularly bad this time due to the uncertain state of progress in several software field test situations and the fact that DEC is planning announcements that they are not willing to coordinate with us in advance.

Ideas on better ways to deal with this issue as well as all inputs on Symposia subjects such as how to broaden participation and upgrade usefulness are always welcome.

NOTE FROM LARS PALMER

Lars sent a response to the paper I mentioned in the last Newsletter by Jim van Zee that compares OS/8 FORTRAN IV, OS/8 FORTRAN II, OS/8 BASIC, and his U/W FOCAL. In order to have the response be meaningful, I will try to reproduce Jim's paper as well as Lars' response. Many of the points Lars makes reflect my own reasons for frequently choosing FORTRAN IV for new programs. I do think, however, that an important point for a great many of our readers is software and hardware cost. FORTRAN II may not be the most advanced of the choices but it comes in the basic software kit at no extra cost and it does a great deal. It can do some things that none of the other languages can do (i.e. efficent 12 bit data manipulations at near assembly language speed that can be expressed in FORTRAN and imbedded assembly language instructions for doing special manipulations and operations). It has many of the same advantages as FORTRAN IV in the area of self contained subroutines written in FORTRAN or assembly language that can be loaded or included in a library. U/W FOCAL is the next cheapest in software cost and it is optimized to run on minimum cost hardware configurations. It is the most interactive and responsive choice and it is (to me at least) almost an order of magnitude better tuned to the user's needs than OS/8 BASIC. My personal bias is that OS/8 BASIC is, and always has been, a failure. Its design is faulty, it is not well maintained by DEC, it costs more than the preceding choices and to make it anything close to interactive or powerful you need a relatively expensive system configuration. If you must have the BASIC language for compatibility or training reasons or if you must have fairly effective string manipulations you have to choose it however. Finally, given enough hardware for BASIC effective, I find the better choice for me is usually FORTRAN IV. It is (almost) compatible with ANSI standard FORTRAN IV which gives access to a vast array of programs and trained programmers. The software is portable to a much higher degree than BASIC due to the ANSI standardization of a version that is sufficent to do real work. This is as opposed to BASIC where the "standard" currently being developed is still only for the minimal version which is useful for games and teaching, but not real, high quality production programming that must use all of the system's facilities effectively. For example, the DECUS program libraries contain a rich collection of programs written in "BASIC". On closer inspection it turns out that the larger, more useful packages are usually written in DEC's "BASIC PLUS". I have translated or tried to translate a number of these programs to OS/8 BASIC with poor results. Even though I know something about BASIC PLUS I still find the conversion job as hard as a converson from FOCAL or FORTRAN. On the other hand, I have converted a number of packages in FORTRAN from the PDP-10 and PDP-11 libraries as well as other sources with very little trouble (i.e. SSP, CALCOMP packages, and many others). I have also been able to take OS/8 FORTRAN IV programs directly to the PDP-11 with no changes at all except for I/O unit numbers that I always assign to an integer variable at the beginning of the program anyway and a little caution with the use of Hollerith data and A format. The only caution I

have for prospective users of FORTRAN IV is this: DEC is not actively developing badly needed enhancements (Lars notes that you need a number of user written extensions for really good programs), indeed DEC is not really doing any serious maintenence on FORTRAN IV. The SPR answers are coming out saying things like - yes, this is a bug, it will be considered a permanent restriction - rather than fixing the problems. In view of its advanced features, compatibility and the fact that this is the only DEC software that can use the new FPP-8A floating point option, I think it is sad and strange that DEC seems to be approaching the point where they will be forced to withdraw FORTRAN IV for lack of resources and determination to support it.

Lars also sent some advanced information on the forthcoming European DECUS Symposium. The 12 Bit program is shapeing up very nicely. There are several interesting papers on advanced work with 8's and a workshop. I am pleased to see that RTS/8 seems well represented. DEC will have time to talk about what they are doing and there are hopes of having some 12 bit hardware to demonstrate software, exchange programs and serve as a focal point for user activity the way we try to do in the US. They are also planning user run training seminars for the afternoon and evening of the day before the Symposium begins. By sceduling in this way, members can attend one of the sessions without having to stay in London an extra night.

UNSIGNED COMPARES ON THE PDP-8/12

In talking to Ed Steinbeger recently, the subject of doing unsigned comparisons came up. We agreed that, while not hard to code, they are still one of the most frequent sources of coding errors on the 8 and 12. It is rare to see anything in print that spells out how to do them. Ed offered a copy of some notes he had made on the subject. I have edited them and added some explanation as follows.

Ed was interested in two cases. First is the common case were you define 0000 thru 7777 octal as representing 0 thru 7777 octal (0 thru +4095 decimal). The second case is the less common assumption that 0001 thru 7777 octal represent 1 thru 7777 octal (+1 thru +4095 decimal), and 0000 represents 10000 octal (i.e. +4096 decimal). In other words, in the second case there is no representation for a zero value but there is one for 4096. Ed looked at two ways of coding the comparisons in each of these cases. They are to start with the link bit cleared or with the link bit set. Although the more natural way to use the 8 is to start with the link bit clear, Ed feels that the code comes out better starting with it set. Both versions for each case are shown.

Set up code for unsigned comparison of locations "A" and "B" in the 0-7777 case:

Link starts = 0	Link starts = 1
CLA	CLA
TAD B	TAD B
CLL CIA	CLL CML CIA
TAD A	TAD A

Set up code for unsigned comparison of locations "A" and "B" in the 1-10000 case:

```
Link starts = 0          Link starts = 1
---------------          ---------------
CLA CLL                  CLA CLL CML
TAD     B                TAD     B
SZA                      SZA
CIA                      CIA
DCA     TEMP             DCA     TEMP
TAD     A                TAD     A
SNA                      SNA
CML                      CML
TAD     TEMP             TAD     TEMP
```

The above set up is then followed with a skip sequence selected from below depending on which link start was used and which skip is desired.

Skip next location if:	L starts = 0	L starts = 1
B>A	SZL	SNL
B=A	SZA	SZA
B<A	SZL SNA	SZL SNA
B>=A	SNL JMP .+3 SZA	SNL SZA
B<=A	SNL	SZL
B not equal A	SNA	SZA

If you have any inputs on subjects such as this it would be useful to many of our readers for you to submit them for publication in the Newsletter.

NOTE FROM BENJAMIN A. FAIRBANK

Benjamin wrote wondering how to bridge the gap between the information DEC supplies with OS/8 and the tips in the Newsletter. He feels that many of the contributors are much more advanced than beginners like himself. He would like to know were they get their background information on such things as OS/8 conventions, storage, etc.

(I think the first answer is the OS/8 Software Support Manual which I am always amazed to find how few users know about. Maybe DEC does something odd about the way they distribute it so new users are not made aware of it. The second answer is that our contributors include the authors and current maintainers of the software at DEC as well as a number of people who have spent as much as six or seven years working

with the system. Many of them either have sources (now available for a price) or have "dis-assembled" considerable parts of the system as I did in the early days before sources were available. Just careful study of the source of CCL (which is included in the basic software kit) will uncover many chances for modifying the way the system works to suit your desires. Many of the items available from the DECUS program library also contain valuable lessons for those interested in being advanced OS/8 system programmers. Finally, developing contacts among the active user community through the newsletter, direct contact at the Symposia, and by personal contact through the mail and phone will help broaden your perspective and sources of information. By the way, many of our contributors are not so advanced and all contrabutions are always welcome and solicited. For example see the next paragraph. RH)

Benjamin discibes how he cut the cost of a bootstrap when he started using OS/8. It seems that a ROM bootstrap option for his PDP-8e from DEC would have cost $900. Instead he bought an 8k semiconductor memory board from "brand X" for $650. By rearranging the address jumpers, he was able to put his existing 8k of core memory in fields zero and three and the volatile semiconductor memory in fields one and two. He then wrote a permanently resident bootstrap program for the top page of field three. The program initializes itself, copies the standard DEC bootstrap to Field 0, then jumps to it. Thus for less than the cost of a ROM he was able to upgrade to 16k and have the advantage of a bootstrap saved in non-volatile memory. In his case the area where the permanent bootstrap is saved is rarely altered by the programs he runs. If that got to be a problem he could use the CORE command to protect the top field of memory (at the cost of losing 4k of course).

NOTE FROM JOHN YOUNGQUIST

John sent copies of a couple of TECO editing macros called CLEAN and ALIGN. They clean up tabs in assembly language sources and align the comment fields. They can both be used with the MUNG command. The first line demonstrates how to do this for other macros. I will attach the copies of the typeouts to the Newsletter. John is interested in communicating with anyone else who is writing TECO macros. His address is Verus Instruments Inc., Box 122, Fort Erie, Ontario. He also included a plea for DEC to improve the handling of lower case characters and rubout sequences in the monitor in the next release of OS/8. The problem is that the monitor has always been extremely tight for space and it gets tighter with each release so the desired features take several loctions that are almost impossible to find. I have heard that there is an effort to implement something to help this situation however. One way to minimize the space problem is to use a SET command to automatically patch the monitor for things like a scope style rubout sequence rather than the printing type. This way you only need space for one or the other rather than both. John notes that a good example of how the terminal should be treated is in Dewar Information System's "ICE" editor. He feels that this editor is very good by the way and recommends it highly. He also likes their "ACID" document generator and thinks it is superior to RUNOFF, MEMO and so on.

NOTE FROM LYMAN BYRD

The following information was received a while back but did not get into the last newsletter because I had set it aside to try to find out what program it refers to. I still am not sure but I think it is the one that is in the LAB-8 (LAB-8e?) software. Mr. Byrd writes that while he was trying to figure out a way for the program CONVER.SV to be used under BATCH control he came to the conclusion that the following changes would accomplish the purpose that is to convert data formats from averaging programs to that of FORTRAN format. The change works very well except that you must start the CONVER.SV program over again for the next file. He changed the name of CONVER.SV to F4XAV.SV to distinguish it. The conversion can be any legal input to the conversion program by changing the four locations to the desired ASCII representations.

```
15600   0000    0000
15601   6031    2211    /ISZ    15611
15602   5201    1611    /TAD I  15611
15603   6034    3220    /DCA    15620
15604   3220    1220    /TAD    15620
15605   1220    4221    /JMS    15621
15606   1377    1220    /TAD    15620
15607   7450    5600    /JMP I  15600
15610   5600    xxxx    /null (not used)
15611   1376    5611    /Pointer
15612   7450    0306    /"F
15613   5245    0264    /"4
15614   1375    0274    /"(Backarrow)
15615   4221    0301    /"A
15616   1220    0326    /"V
15617   5600    0215    /"(Carrage return)
```

NOTE FROM ERIC OLSON

Eric wrote with responses to some items in the Newsletter. First he says that the item on static (p. 2, NL 21) sounded like a problem he is having. It seems that at Bromfield they are having a bad time with their one and only DECtape drive. Whenever anything goes wrong with the system, it dies. Since they only have one drive, he says the only way to copy tapes is with a program called CORE from EDUCOMP which reads a hunk of a tape into memory, lets you change tapes, writes it out on the new tape, then starts over again for the next hunk.

In response to Jerome Vuoso's problem (p. 8 NL 21) he notes that when he talked to his DEC salesman about the CLASSIC when it first came out he was told that it was not intended as an expandable system but if you understood a little about the bus structure (i.e. OMNIBUS) you could probably get an interface and install it yourself. (Note: I would suggest that any CLASSIC owners thinking about this approch who are not familiar with the ins and outs of confiburing OMNIBUS systems get some help. The main issues that come to mind are: are there any slots left in the processor?, can the bus drive any more loads?, what do you have to do to extend the bus into an expander box and how will you mount, power, and cool it. One of the cost savings that DEC realizes in the

package systems like the CLASSIC and 310 is to select configurations that just fit in a partcular processor with no room to spare in the box or on the power supplies or in the cabinets. It is true that the processor inside has always been a regular PDP-8 though that has the same expansion potential as if it had been purchased in the normal way. You save money on the special packages because DEC can "mass produce" fixed configurations that make the best use of the configuration possibilities of a given processor model. RH)

He thinks that Jerome can solve his problem of students zeroing or otherwise distroying diskettes by just removing PIP from the system disk. COPY, DIRECT, and DELETE will still work but ZERO and SQUISH will not. This will mean that from time to time someone will have to do file maintenence for everyone using a system disk that does have PIP. You better not have too much on each disk. When I use a floppy disk system I am always filling up the disks and running out of space even with all the functions of PIP available to help maintain the file storage. Note that "DELETE *.*" will still remove everything from the disk but at least the system blocks will not be wiped out.

NOTE FROM JEFF WYATT ON FORTRAN IV BUG

Jeff found a bug in OS/8 FORTRAN IV which may not be known. He did not say if he had submitted an SPR. Jeff reports that the following code compiles incorrectly:

```
DIMENSION A(100),B(100)
EQUIVALENCE (A(100),B(100))
```

The trouble is that the code the compiler generates forgets the dimension for B and starts the executable code that follows these two statements just after A(100). Therefore any references to B(2) thru B(100) will be incorrect and if any values are stored in those locations part of the program will be distroyed! A way around the problem is to dimension A large enough to include all of B (i.e. A(199) or more).

Eye Research Institute
20 Staniford St.
Boston, Mass. 02114 617 742-3140

1) QUERY TO OS/8 EXPERTS: You are running OS/8 and have just returned to the keyboard monitor. Now you replace the medium in the OS/8 system device with another OS/8-system-bearing medium. What must you do? Are there any circumstances (systems have same device configuration; systems are strictly identical; etc.) under which it is safe to just continue? If not, does a restart at 7600 do everything necessary? Or are there subtle dangers in anything short of a full reboot?

2) FORTRAN IV BUG-OF-THE-MONTH: It is unsafe to use any multidimensional array with more than 2047 elements; furthermore, it is unsafe to use any multidimensional array with more than 1023 elements if individual elements or subarrays are to be passed to a subprogram (i.e. if the argument passed to the subprogram is subscripted). Exact details follow.

Suppose we DIMENSION $A(d_1,d_2,d_3)$ and reference $A(I_1,I_2,I_3)$. Let

$$D = I_1 + I_2d_1 + I_3d_1d_2$$

$$MN = 1 + d_1 + d_1d_2 \qquad \text{(i.e. MN = D for A(1,1,1))}$$

$$D' = D - MN = (I_1-1) + (I_2-1)d_1 + (I_3-1)d_1d_2$$

D is easy to calculated, but D' is the correct displacement from the array base. (The problem is that the designers of FORTRAN apparently were used to Roman numerals and didn't know that you should start counting with zero.) Anyway, in a normal subscript reference, what FORTRAN IV does, apparently in order to save cycles, is to calculate D, place it in an index register (say 7), and reference, e.g. FLDA A-MN*3,7 The problem is that in a multidimensional array, D may exceed 4095 even if the array size, and the quantity D - MN, does not. In this case the XR overflows and the wrong address is referenced.

If the subscripted quantity is being passed to a subprogram, as before D is computed and placed in an XR, say 7. Then D is retrieved via an XTA, multiplied by 3, and added in STARTD mode to a JA A-MN*3 to form the argument. The problem here is that XTA retrieves as a signed number, so in this case the wrong address is computed if D exceeds 2047.

For example, if DIMENSION A(2000,2), all elements from (48,1) on up will be transmitted incorrectly, and elements A(96,2) through (2000,2) will be referenced incorrectly even within the subprogram.

The rule is that $d_1 + d_1d_2 + d_1d_2d_3$ must not exceed 2047. In the worst case, DIMENSION A(17,2,2,2,2,2,2) will have problems because D = 2159, although there are only 1088 elements. If the total array size is less than 1024 elements, the problem cannot arise; larger arrays may be permissible depending on the exact dimensions, but it is necessary to do the above calculation to find out.

I've written a nasty SPR about this and hope the reply will be more helpful than DSN April 77, p. 14.

3) A plotter hardware problem with PDP-12's, XY12 interfaces, and Calcomp 565 plotters. The sequence PLPR;PLSF;JMP .-1;PLCF;JMP .-4 which ought to run the carriage to the right, doesn't. It iwill, however, if the program is auto single-stepped at slightly less than maximum speed, or if about five NOP's are inserted between the PLPR and PLSF. If you have this problem, ask your rep about Tech Tip XY12-TT-1 8/20/76. PLPR sets a flip-flop that's supposed to stay set for 2.5 msec; however, IOP1 from the PLSF clears it, and the 565 won't respond to very short pulses.

DEAR BOB:

A problem can turn up with the E.A.E. in newer machines when you want to find out which mode -- Mode B or Mode A -- you are actually in. The standard method (do a DPSZ; if it skips, you're in Mode B) can cause trouble if, in fact, you are in Mode A. The accumulator and the step counter are both altered.

The following is used in the current ETOS system head. I have tried to mark the specifically ETOS stuff; but the idea should be of general use.

```
/ PATCH FOR ETOS HEAD  -- E.A.E.          PAL8-V9H  04/11/77  PAGE 1

            / PATCH FOR ETOS HEAD  -- E.A.E.

            / 'ETOS.SV' BLOCK 65, LOC. 71 (FOR EPIC)

      0000  FIELD 0

      2071  *2071

02071 7441          SCA             /GET STEP COUNT
02072 7104          CLL RAL         /SHIFT UP 1 BIT
02073 3332          DCA EAESAV      /SAVE.
02074 7501          MQA             /(ETOS: GET MQ)
02075 6001          ION             /(ETOS: INTERRUPT ON)
02076 3024          DCA LT2MQ       /(ETOS: SAVE MQ)
02077 7403          ASC SCL         /WHICHEVER
02100 2332          ISZ EAESAV      /SET BIT 11 IF MODE 'B'
02101 7431          SWAB

      2115  *2115

02115 1332          TAD EAESAV      /GET STEP COUNTER THAT WE SAVED
02116 7110          CLL RAR         /SHIFT BACK TO OLD VALUE
02117 7403          ASC             /(ETOS IS IN MODE 'B' AT THE MOMENT)
02120 7420          SNL             /LINK CARRIES MODE INFO
02121 7447          SWBA
02122 7300          CLA CLL

      7441  SCA=    7441    / WORKS THE SAME IN BOTH MODES.
      2132  EAESAV= 2132
      0024  LT2MQ=  24
      7431  SWAB=   7431      / MQL + set Mode B
      7447  SWBA=   7447      /       set Mode A

      7403  ASC=    7403    /MODE 'B':
                            / AC INTO STEP COUNTER
      7403  SCL=    7403    /MODE 'A':
                            / LOAD COMPLEMENT OF BITS 7-11
                            / OF NEXT WORD INTO STEP COUNTER
                            / & SKIP NEXT WORD

            $$$ after J. Dempsey $$$
```

G. Chase OSB

MAR. 29, 1977

ROBERT HASSINGER, COORDINATOR
12 BIT SIG
C/O DECUS
146 MAIN ST.
MAYNARD, MA 01754

#22 PAGE 10

DEAR BOB,

I AM WRITTING TO YOU FOR THE FIRST TIME, AS A MEMBER OF A RELATIVELY LARGE BUT UNORGANIZED GROUP OF OS/8 USERS - THOSE WHO HAVE A GENERAL RADIO 1792 OR 1795 TEST SYSTEM. THE SYSTEM CONSISTS OF A PDP-8/E WITH DUAL RK-05 DRIVES. WE HAVE 28 K OF CORE (SOON TO BE EXPANDED TO 32K), A TTY, A TALLY 2200 - 200LPM LINE PRINTER, HI-SPEED PAPER TAPE READER AND A TEC-415 C.R.T. WITH A DIRECT PARALLEL INTERFACE TO THE 8/E. THIS IS A FAIRLY POWERFUL HARDWARE PACKAGE. IN ADDITION TO THE OS/8 MONITOR, G.R. RDOS ITS OWN MONITOR WHICH IT CALLS CAPS. THE CAPS MONITOR HANDLES TEST PROGRAM DEVELOPMENT AND TEST EXECUTION. CAPS ALSO CONTAINS A POWERFUL TEXT EDITOR UTILIZING THE TEC-415. THE C.R.T. HOLDS A PAGE OF TEXT, AND ALLOWS THE USER TO MODIFY IT IN ANY WAY, ADDING OR DELETING CHARACTERS ON THE SCREEN. THE PAGE IS THEN READ BACK INTO CORE, WHICH IS USED TO BUFFER AS MUCH OF THE FILE AS CAN BE FIT. ANY PAGE IN CORE CAN BE CALLED UP AND MODIFIED AT WILL WHEN THE USER IS SATISFIED, THE CORE BUFFER CAN BE WRITTEN TO DISK. THE EDITOR CAN HANDLE ANY SIZE FILES. SEARCH AND OVERWRITING FACILITIES ARE INCLUDED.

I AM CURRENTLY USING BASIC TO FORM A DATA BASE MANAGEMENT SYSTEM FOR KEEPING TRACK OF PRODUCTION TEST DATA. THIS LEADS TO A COUPLE OF QUESTIONS. DOING STRING MANIPULATION ON LARGE ASCII FILES IS QUITE SLOW, EVEN WITH DISK. I BELIEVE THAT THIS IS CAUSED BY BRTS CONTINUALLY CALLING TO THE .SF AND .FF OVERLAYS. I WAS WONDERING IF ANYONE HAD DONE ANY WORK ON MODIFYING BRTS SO THAT TWO OR MORE OF THE OVERLAYS COULD BE CORE-RESIDENT. WE HAVE MORE THAN ENOUGH CORE, SO THAT EXTENDING BASIC TO REQUIRE 12K OR MORE WOULD BE NO PROBLEM, IF WE COULD GET SOME EXPANDED FEATURES. I FOUND STEVE LIGETT'S BASIC MARGIN FUNCTION VERY USEFUL. OUR LINE PRINTER CAN GO TO 132 COLUMNS. HAS ANYONE EXPANDED THE $TRING ACCUMULATOR TO MORE THAN 72 CHARACTERS ? I WOULD ALSO LIKE TO KNOW IF ANYONE HAS BEEN ABLE TO INCREASE THE NUMBER OF ACTIVE FILES THAT BRTS CAN SUPPORT. (PUTTING TO ANY DEVICE OTHER THAN THE TTY REQUIRES A FILE NUMBER (SUCH AS LPT:). I WOULD BE HAPPY TO SEE THE NUMBER RAISED TO 6 OR 8 AND THE FILE FUNCTIONS MADE CORE-RESIDENT).

ANOTHER AREA THAT I WANT TO EXPLORE IS MULTI-TASKING. ONE DRAW BACK TO THE G.R. SYSTEM IS THAT WE CAN'T DO PROGRAM PREPARATION, LOGIC SIMULATION AND TESTING AT THE SAME TIME. WHEN TESTING, THE TESTER SPENDS A GREAT DEAL OF TIME DOING NOTHING, WAITING FOR THE OPERATOR TO SETUP AND TAKE DOWN THE P.C. CARDS. IT WOULD BE NICE TO BE ABLE TO HAVE A BATCH JOB RUNNING IN THE BACKGROUND. IT WOULD ALSO BE NICE TO BE ABLE TO ATTACH ANOTHER TERMINAL, AND DO MULTIPLE SIMULTANEOUS EDITING OR FILE MANIPULATION. I HAVE CONTACTED THE EDUCOMP PEOPLE ABOUT ETOS. I AM PLANNING TO HAVE THEM IN FOR SOME EXPERIMENTS. IT WILL NOT BE EASY TO ACCOMPLISH WHAT I WANT, SINCE THE ENTIRE CAPS SYSTEM IS NON-INTERRUPT. IF YOU WANT, I'LL LET YOU KNOW HOW I MAKE OUT.

THE SEVERAL HUNDRED G.R. TESTERS IN THE FIELD REPRESENTS A LARGE COMPUTATIONAL RESOVOIR WHICH REMAINS LARGELY UNTAPPED, SINCE G.R. SEEMS RELUCTANT TO FORM A USERS GROUP LIKE DECUS. IF SOME SORT OF MULTI TASKING WAS AVAILABLE, I FEEL THAT MORE OF THE USERS WOULD AVAIL THEMSELVES OF THE POWERS OF THE 8/E. IT IS DOUBTFUL THAT G.R. WILL HELP IN THIS LINE, AS IT SEEMS THAT THEY ARE MORE INTERESTED IN MOVING ON TO THE PDP-11 FOR FUTURE DEVELOPMENT WORK. I WILL BE INTERESTED TO HEAR OF DEC'S (AND OTHERS') DEVELOPMENTS IN THE LINE OF NEW SOFTWARE FOR THE 8/E. KEEP UP THE GOOD WORK WITH THE SIG IN KEEPING US INFORMED. THANK YOU.

Jim Scharf

JIM SCHARF

TEST ENGINEER

CONRAC CORP
32 FAIRFIELD PL
WEST CALDWELL, N J
07006
201-575-8000 X264

Bill Haygood has developed a new computer-assisted instructional (CAI) system for training postal clerks to use the Multi-Position Letter Sorter Machines (MP/LSM/s) at the Salt Lake City, Utah and six other Western post offices. The actual MP/LSM requires 12 operators who must each key one letter per second (via 2 or 3 key hits on a special keyboard) which translates to 3,600 letters per hour per clerk. As recently demonstrated at the Van Nuys, CA post office, the software can support 12 training VT-52 CRT terminals. Since there are severe timing constraints in the simulation, the 12 CRT/s must run in real time. A single PDP-8/e with 32K core memory and a single RK05 disk drive drove 12 VT-52 CRT/s each displaying one item per second and each CRT requiring one disk access per second (12 disk accesses per second). The 2nd test consisted of 11 CRT/s doing the one item per second simulation while the 12th CRT was running OS/8 doing data base generations from the RX8-E - Floppy disk and printing on the LA180 line printer. Additionally, when the CPU had nothing else to do (!), it ran a NULL job in the AC lights. During all the tests, none of the 12 experienced MP/LSM operators could detect any change in the rhythm of the items at any terminal. The CAI trainer program occupies 8K for its re-entrant code and another 8K for user stacks. A special real time executive was used in fields 0 and 1 (all I/O buffers were located in field 1). Any ONE terminal on the system can call and run OS/8 concurrently with CAI on the remaining CRT/s. The software, by the way, was written in its entirety in PAGE8 consisting of approximately 14,000 statements and about a dozen or so macros. The project consisted of three parts: the executive, the CAI trainer, and the data base generator (which runs under OS/8 in either stand-alone or under the time-sharing executive. This test clearly demonstrates that the humble PDP-8 is capable of a lot more than many think. By the way, even during the 12-terminal tests, the NULL job ran at least 50% of the time!

Bill also has available a multi-user OS/8 time-sharing executive which allows 2 or 3 terminals each running OS/8 on a minimum hardware configuration of a PDP-8/e, f, m, or A with 20K memory (for 2 terminals) or 28K (for 3 terms), one RK05 disk drive and a DK8-EA real time clock. System response is said to be very good because all users are always core resident. Bill also has in the works a 2 terminal OS/8 time-sharing executive which will require only 12K memory (with other hardware same as above). Interested parties may write to: Bill Haygood, 3704 Ridgecrest Drive, Salt Lake City, Utah 84118.

```
!THIS IS A TECO MACRO TO PROCESS PAL8 SOURCE FILES
IT REMOVES TABS BETWEEN MNEMONICS AND LABELS WITHOUT
MISALIGNING THE COMMENT FIELD.
STORE IT AS "CLEAN.TE"
IT REQUIRES STUART DOLES ^B PATCH FOR END
OF FILE FLAG    1332/1035  1037
                5002/0011  1332
^B=4095 FILE OPEN,=0 IF EOF SEEN


CALL IT WITH: .MUNG CLEAN,DEV:FILE.PA    !


J HXZ$HK$@I'EB$'S-CGZ$HXZ HK MZ$

!START! OUA ^AMACRO CLEANUP IN PROGRESS
^A
<P 1%A$
J< S    AND     $ -D I $ 4<0A-13"E L OTERM1$'
0A-9"E 9I$ L OTERM1$' C> !TERM1!>$
J< S    AND I   $ -D I $ 2<0A-13"E L OTERM2$'
0A-9"E 9I$ L OTERM2$' C> !TERM2!>$
J< S    TAD     $ -D I $ 4<0A-13"E L OTERM3$'
0A-9"E 9I$ L OTERM3$' C> !TERM3!>$
J< S    TAD I   $ -D I $ 2<0A-13"E L OTERM4$'
0A-9"E 9I$ L OTERM4$' C> !TERM4!>$
J< S    ISZ     $ -D I $ 4<0A-13"E L OTERM5$'
0A-9"E 9I$ L OTERM5$' C> !TERM5!>$
J< S    ISZ I   $ -D I $ 2<0A-13"E L OTERM6$'
0A-9"E 9I$ L OTERM6$' C> !TERM6!>$
J< S    DCA     $ -D I $ 4<0A-13"E L OTERM7$'
0A-9"E 9I$ L OTERM7$' C> !TERM7!>$
J< S    DCA I   $ -D I $ 2<0A-13"E L OTERM8$'
0A-9"E 9I$ L OTERM8$' C> !TERM8!>$
J< S    JMP     $ -D I $ 4<0A-13"E L OTERM9$'
0A-9"E 9I$ L OTERM9$' C> !TERM9!>$
J< S    JMP I   $ -D I $ 2<0A-13"E L OTERM10$'
0A-9"E 9I$ L OTERM10$' C> !TERM10!>$
J< S    JMS     $ -D I $ 4<0A-13"E L OTERM11$'
0A-9"E 9I$ L OTERM11$' C> !TERM11!>$
J< S    JMS I   $ -D I $ 2<0A-13"E L OTERM12$'
0A-9"E 9I$ L OTERM12$' C> !TERM12!>$
^B;> ^APAGES EDITED  ^A QA=$
^AJOB COMPLETE^A$EX$
```

```
!THIS IS A TECO MACRO TO ALIGN THE COMMENT
FIELDS IN A PAL8 SOURCE FILE.
STORE IT AS "ALIGN.TE"
IT REQUIRES STUART DOLES ^B PATCH FOR END
OF FILE FLAG     1332/1035  1037
                 5002/0011  1332
^B=4095 FILE OPEN,=0 IF EOF SEEN

CALL IT WITH  .MUNG ALIGN,DEV:FILE.PA    !

^AMACRO ALIGNMENT IN PROGRESS
^A$J HXZ$HK$@I'EB$'$-CGZ$HXZ HK MZ$Y$
<J<!START! 0UA 0UB S/$ 0L
< 8<1%A$ 0A-47"E OSLASH$' 0A-9"E OTAB$' C>
-C !TAB! 8%B$  C 1UA$>
!SLASH! QA%B$ QB-12"L OEND$'
$QB-25"L  3<9I$ 8%B>$'
<$QB-34"L OEND$' -D -8%B>
!END! L> P ^B;> ^ACOMMENT ALIGNMENT COMPLETE
^A EX$
```

Abstract of GT.PA, OS/8 Handler for Tektronix 4006-1 Graphic Terminal as Console Device

GT.PA is an OS/8 handler for the Tektronix 4006-1 Graphic Display Terminal in alphanumeric mode. It allows the terminal to input and output as the console device (device codes 3 and 4), possibly replacing a teletype in this capacity. The standard OS/8 features are available, plus the added feature of stopping at the bottom of the screen during output, allowing the operator to hit any key in order to erase and refill the screen with the next section of the text.

The handler was created by modifying the Digital KL8E.PA handler. It occupies two pages.

Submitted to DECUS.

Ronald P. Larkin

The Rockefeller University
1230 York Avenue
New York, N.Y. 10021

Jim van Zee
Dept. of Chem. - Univ. of Wash.

In the beginning there was the PDP1, followed soon after by the PDP5 and the PDP8. And DEC looked upon them and realised they needed SOFTWARE. So DEC said to Richard Merrill, 'Let there be a language which all may speak with only 5 minutes instruction' and behold: FOCAL came into being.

From its inception, FOCAL (short for either FOrmula CALculator or Formulating On-line Calculations in an Algebraic Language) was enthusiastically received by PDP8 programmers. This language offered such ease of use, yet had such power and flexibility, that thousands of people adopted and adapted it to solve their problems. FOCAL provided a convenient starting point for applications programs covering the range from machine tool control to hospital analytical instruments. Yet within a few years of its introduction (roughly 1968), DEC apparently lost all interest in 'their' language.

The original FOCAL was written primarily for a 4K papertape installation, but was designed to allow expansion to 8K and adaptation to a time-sharing environment. One of the significant design features was the use of the interrupt system which not only improved the apparent execution speed (since calculations proceeded in parallel with the output), but opened the door for many real-time applications. Another blessing was an optional overlay which increased the arithmetic precision to '10-digits' in place of the more usual '6-digit' floating-point software. Thus FOCAL appealed to a wide range of users: laboratory personel (prominent among the early users of DEC computers) who could adapt FOCAL to automate tedious measurements, and educational groups who found the language easy to teach, very forgiving in terms of syntax, and at once both powerful and precise.

With the advent of a full-blown operating system for the PDP8, however, FOCAL more or less officially disappeared. The new emphasis from Maynard was on FORTRAN (supplied with PS/8, OS/8) and BASIC (available for an additional charge). These languages, of course, had been developed elsewhere and were in widespread use on big machines, and, I am sure, were a necessary part of a successful marketing strategy. We ourselves included the availability of FORTRAN as part of the bid specifications for our initial purchase even though we were not at all certain of its usefulness for laboratory applications. In the mid-seventies DEC continued developments on FORTRAN and began marketing a sophisticated FORTRAN-IV compiler, leaving little old 4K papertape FOCAL far behind.

But the users had not forgotten FOCAL. Many, many people continued to submit improvements to DECUS and several expanded versions appeared which worked with both user-developed monitor systems (such as the RT monitor) and also with OS/8. One of the most successful of these was written by Barry Smith and Dave Schneider at OMSI. Their 'PS/8 FOCAL' was designed simply as an overlay to the last DEC-released version (FOCAL '69) and provided 'device independent I/O' as well as automatic program loading and library maintainence. When we purchased our computer system we got FOCAL '69 on a paper tape and only some months later discovered OMSI's version quite by accident. The point of this story is that if it had not been for an intense interest within the user community, FOCAL would surely by now have become a 'dead language' since the original paper-tape version is certainly not on a par with the advanced foreground/background, field-independent, multi-overlay structure of FORTRAN IV. The purpose of this paper is to examine some of the strengths of a user-developed FOCAL, an outgrowth of PS/8 FOCAL, in comparison to the DEC supported languages currently available to the OS/8 programmer.

The version of FOCAL under discussion has been labeled 'U/W-FOCAL' or 'UWF' for short. In its current form it represents the 'end' of approximately 5 years of evolution, and while it began simply as a massive overlay to FOCAL '69, it was eventually completely rewritten (several times as a matter of fact!) and now bears only a formal resemblance to the original design. There are two chief characteristics of this version: the first is to retain some of the 'simplemindedness' of the original language. By this I mean that no attempt has been made to transcend Field boundaries or to implement fancy data structures or create run-time overlays. What you see is what you've got and you can patch it to do your 'thing' rather easily.

The second characteristic is to pack as many features as is humanly possible into the smallest amount of core space. Both of these characteristics are rather at variance with current software trends. First, of course, the price of memory is now so low that a 'do-it-yourselfer' can have 32K for approximately $1K, while secondly the cost of developing 'core-tight' programs tends to rise exponentially. Economics thus dictates the inefficient use of core and the efficient use of human resources. The FOCAL programmer, on the other hand, 'likes to get his money's worth' and often savors a clever trick (which might take hours to invent) which saves a single line in a program. I will not attempt to justify this attitude, but merely observe it, both in myself and among many other FOCAL programmers throughout the world. Perhaps it is the innate elegance of the language (a quality shared by APL) which makes us want to 'write the whole program on one line'.

This desire for conciseness begins our comparison: in FOCAL all the directives can be abbreviated to a single letter, and expressions not required can simply be omitted. To consider, for example the coding required for a simple loop, we have the following:

FOCAL	BASIC	FORTRAN
F I=1,2,100;	FOR I=1 TO 100 STEP 2\ \NEXT I	DO 999 I=1,100,2

Continuing this example, let us suppose that this loop calls a subroutine in which case the FOCAL programmer would append a phrase such as 'D 2' after the semicolon, the BASIC programmer would insert 'GOSUB 2000' between the slashes and FORTRAN would require the statement '999 CALL SUBROUTINE TWO'. Thus for this simple, yet common example, FOCAL has an appealing simplicity.

Now this sort of thing can be viewed in different lights. First of all, it is undoubtedly true that the 'wordiness' of BASIC and FORTRAN improves the 'readibility' of the program. Indeed, FOCAL programs often appear completely 'dehydrated' and need to be 'soaked for several hours' in order to make them intelligible. This, however, is discretionary on the part of the programmer, who may spell out all the commands if he wishes: FOR I=1,2,100; DO 2. This improves the program legibility immensely, yet is still more compact than the other two examples. But once accustomed to the abbreviations, conciseness is quite a practical advantage since all programs must be typed-in a character at a time, and many of us are '1-finger pokers'. The fact that only the first letter of the command matters is also often convenient. At the ground level it minimises the consequences of spelling errors: FOR and FIR do the same thing! It also makes foreign language versions easy to construct, for example, FÜR works as well in German as FOR does in English.

This leads to the problem of program preparation in general. Certainly one of the most appealing features of FOCAL is the MODIFY command. This allows the programmer to rapidly make changes in an existing program with most of the power of the OS/8 EDITOR, but without having to load in an overlay (as in BASIC), or worse still, return to the monitor and run an entirely separate program (as you must when developing FORTRAN programs). FOCAL is thus more

highly integrated; it looks at the problem of program development for what it is: the constant alternation of 'typing and trying'. Any programmer accomplishes more when he (she) can take advantage of the computer in the development phase as well as in executing the final result. If there is only a single feature which makes FOCAL appealing, it is surely that the programmer can fix a mistake and -immediately- rerun the program. Indeed, he can often just selectively execute a few steps to see if the result is correct.

Since all the defined variables are available for inspection at any time it is a simple matter to change some values, fix up the program and continue. How much grief is caused by not being able to 'look back' when 'version 1' of a FORTRAN or BASIC program goes astray! The program must then be altered to include additional PRINT or WRITE commands and the entire process begun over. This usually means creating a whole series of slightly different source files with all the implications this has for rapidly using up even the largest of mass-storage devices, and of course, those of us with DECtapes or cassette systems absolutely cringe everytime we see the tape start to wind out toward the end!

So FOCAL is simply more productive. This has been verified by at least two large data-processing installations: the Boeing company in Seattle, and the U.S. Coast Guard in Connecticut. Both installations have compared U/W-FOCAL and FORTRAN IV and found that the former has some distinct advantages. True, FOCAL programs run slower since most implementations of FOCAL have been as interpreters.* But! Who cares? As long as the CPU is faster than the lineprinter, or more generally, as long as any output device, even a CRT terminal, is faster than human comprehension of the output, the program running time becomes less important than the program development time.

The key here is the nature of small computer systems, even those which are primarily used simply as 'number crunchers'. The typical PDP8 programmer views his computer as just another 'tool-of-the-trade' and needs to be able to program it quickly to run a slightly different experiment, or analyze the results, omitting the first and third runs, etc. Halfway through the program he may suddenly want to see a few more significant digits, or need to change the calibration routine to include more points... Rather than trying to design an 'all-purpose' program which is compiled once and run forever, FOCAL users prefer a 'hands-on' approach in which some operations are executed as 'direct commands' while others can be developed into convenient subroutines which can then be called directly or saved for use later on. As a simple illustration, if an experimenter would like to examine a few of the data points near a suspected 'glitch' he doesn't need to sit down and write out a special program for this (FORTRAN), or save the program he was running so he can write a new one (BASIC), he will simply type in something like: 'FOR I=100,105;TYPE FRA(I)' where the 'FRA' function (File Random Access) is his means of reading any data point in the file. If an anomoly is found he can then replace it with the average value, or take some other course of action - again all with direct commands or combinations of direct commands and short subroutines which he develops as he needs them.

What I am expounding here, of course, is the old argument in favor of 'interactive' programing and the message is: FOCAL has it and CS/8-BASIC and FORTRAN do not!

- - - - - - - - - - - - - - - - -

* Two notable exceptions are close to home here in Banff: first Jim Gosling (formerly with the University of Calgary - where is he now??) developed a compiler for a 4K version of FOCAL about 5 years ago. And secondly, Prof. A. S. French up at Edmonton has also completed a compiler and an interpreter for his BOSS operating system on a PDP11. I will let him tell that story when he returns from his sabbatical leave next year.

But there are other operational differences, little things perhaps, but ones which tend to become annoying after a while. For instance, the BASIC editor offers no way to correct a program line short of re-typing the whole thing, and likewise there is no simple way to list just a few parts of your program or to print the listing on anything but the terminal. Another annoying feature about BASIC is the restriction on variable names, the fact that only 1 or 2 character names are allowed, with the requirement that the second character MUST be a number. Both FORTRAN and FOCAL allow names of any length, although in FOCAL it is true that the first two characters must be unique and names beginning the the letter 'F' are illegal. FORTRAN has its own naming restrictions, however, since variables beginning with the letters 'I-N' are treated differently from others. This peculiarity can be removed in FORTRAN IV, but nevertheless the restrictions on the use of various data types in expressions has always struck me as unfortunate. Why, for example, if the compiler is smart enough to know that 'I*X' is a 'mixed mode' expression, doesn't it just automatically float the quantity 'I' and quietly go about its business? A similar comment applies to the necessity of writing small integers such as '2' in the form '2.0', or even worse, as '2.0D:00' in order to avoid 'mode errors'. Fortunately both FOCAL and BASIC (and FORTRAN IV) use floating-point numbers throughout and thus avoid this nuisance.

Then there is the question of formatting. Both FORTRAN and FOCAL offer integer, decimal and scientific formats, the choice and the number of digits printed being up to the programmer, while BASIC more or less makes up its own mind, using a format which depends upon the magnitude of the data! It is impossible, for instance, to print the number 1.2345E-6 in BASIC to more than 3 significant figures: 0.00000123 - a situation many will find intolerable. On the other hand, I find just as objectionable FORTRANs idea that if the number is too large for the space allowed the best thing to do is to fill the field with stars! Compare these actions with FOCAL's concept: within a given field specification a number will be output so as to preserve as many significant digits as possible. Thus if you specify a format of %5.03 (the same as F6.3 since FOCAL does not include the decimal point or the sign or the separating space in the field count) you will get the values '12.345', '1234.5' or '1.2345E+06' depending upon the magnitude of the data. In FORTRAN only the first value will be printed and you will have no idea at all about the other values! Another minor annoyance is that both BASIC and FORTRAN insist upon putting the first significant figure to the right of the decimal point in 'E-type' output. Thus you get '0.1000000E+09' rather than the standard form of '1.000...E+08'. One of the small improvements in DWF in this regard is the ability to specify a variable precision scientific format with the first digit printed ahead of the decimal point.

Well, what about input? Inputting numbers from the keyboard in FORTRAN II is frustrating since the only way to correct a typing mistake is to type an illegal character and then re-enter all the values associated with that particular READ command (generally this is unknown). The necessity of adhering to specific column positions when typing in numbers is also very awkward. FOCAL, BASIC and FORTRAN IV all allow free-format input with the ability to 'wipe-out' an incorrect number; the latter two even allow individual digits to be corrected whereas FOCAL requires the entire number to be re-entered. A related question involves the convenience of input prompting. In FOCAL one may identify the quantity being input in one of two ways: the command 'ASK "DATA" VALUE' will print the word 'DATA' and then input the value of 'VALUE', while the command 'ASK ?DATA?' will again print the word 'DATA' and also read in a value for that quantity. In both cases the prompting information is included naturally as part of the input command. This is also true in FORTRAN II, but in BASIC and FORTRAN IV, on the other hand, one must use a separate PRINT (or WRITE) statement to supply the identification. BASIC also has no provision for vertical spacing aside from using a null PRINT command.

Well, you may be thinking, when will he find something bad to say about FOCAL? Let us consider the ability to handle strings. BASIC clearly wins this comparison as it has a very powerful set of string functions while FORTRAN sort of limps along with its 'A' format and UWF finishes last with its FIN/FOUT character-at-a-time functions. This is rather unfortunate since, in the 'real world', string manipulation is probably more important to the user than is arithmetic ability. FOCAL (and FORTRAN) however, were originally developed as algebraic languages and so far only one version has appeared (for the PDP11) with extensive string functions.* So if your problem is to scan a parts inventory or to alphabetize a list of 500 student names, BASIC (or perhaps SNOBOL!) is the language to use.

Let us consider next subroutining facilities (in general) and the ability to link to other program segments. BASIC provides a GOSUB command for calling 'internal' subroutines (those which are written as part of the same program which calls them) and the 'FNx' facility for declaring user defined functions. The GOSUB command permits multiple entry points (which is often convenient) while the CHAIN command allows one program to link to another.

UWF, of course, has the powerful 'DO' command for executing any line or group of lines as an internal subroutine. 'DO's may be combined to achieve extremely compact coding, i.e. 'DO 4,5,6' will call 3 subroutines, and conditional calls via the 'ON' command are also possible. A 'RETURN' command is usually optional since the range of the subroutine is automatically defined by the call (either a group or a single line). This allows the 'subroutine' to actually be a part of the 'main' program, but it does preclude the use of multiple entry points. UWF also has 'FOCAL Statement Function' (FSF) calls which provide for passing arguments to the subroutine and returning with a numerical result. On a larger scale, the LIBRARY GOSUB command permits calling all (or part of) an existing program as a subroutine, while the LIBRARY RUN command permits chaining from one program to another.

FORTRAN however, appears to have the most extensive subroutine features. Internally one is limited to arithmetic statement functions (not available in FORTRAN II), but externally (i.e. coded as a separate program) one can define any sort of complex function or subprogram with its own set of local symbols. Thus while FOCAL programs can be called as subroutines, they must be written to operate with the same variables defined in the 'main' program. FORTRAN subroutines, on the other hand, usually employ 'dummy variables' which are replaced by the those specified in the actual function or subroutine call. Since arguments may consist of array names, this is an extremely powerful facility which is simply not available in FOCAL or BASIC. FORTRAN II can also link from one program segment to another via the CALL CHAIN command whereas FORTRAN IV accomplishes the same thing through its multilevel overlay structure. In general, if the problem at hand requires the manipulation of large data arrays or appears to need 32K when you only have 16, FORTRAN IV is probably the most suitable language.

Another important aspect of a language is its extensibility: how easy is it to add new functions or implement a special subroutine which must be coded in assembly language? BASIC provides for up to 16 user functions which are loaded as needed into a fixed core area; 5 pages are available for such functions. These functions are not permanently resident, however, hence they can not be used to implement any real-time operations. Another inconvenience is that the every program which wishes to use them must contain an appropriatly ordered set of 'DEF' statements specifying the function name and the number of arguments associated with it. There is no way to add new commands: all new features must be implemented as function calls (not always desirable).

- - - - - - - - - - - - - - - - -

* Developed by Darrel J. Duffy at West Virginia University, Morgantown, W.Va.

U/W-FOCAL has room for more than a dozen new functions and perhaps four or five new commands. The routines to implement these would be loaded at fixed locations and would be permanent additions to the language - available to any program which wished to use them. Approximately 4 pages have been reserved for such additions. Since such routines are always resident they may be used for real-time operations such as keeping track of elapsed time, monitoring a Schmitt trigger, etc. FOCAL in fact, has a unique facility for performing a 'DO' call based upon the occurance of a hardware interrupt. User routines can be implemented either as function calls -or- as commands. The latter is particularly convenient for 'control' operations such as setting relays, initiating the clock, etc.

FORTRAN II offers by far the easiest method of implementing assembly language functions. Routines written in SABR can be freely intermixed with FORTRAN statements and can reference FORTRAN variables or call internal subroutines whenever necessary. Such routines can be compiled independently and then linked together to build a highly modular program. Both 'functions' and 'subprograms' can be implemented in this way. The one disadvantage in this method is the code is all relocatable and hence somewhat difficult to debug.

In principle it is possible to construct assembly-language routines for FORTRAN IV by writing them in RALF code and then loading the resultant binary along with all the other subroutines required by the main program. The main difficuly with this venture seems to be understanding the intricacies of the Floating-point Processor when all you have is a description of the instruction set. Since the addressing modes of the FPP are so completely different from those in the familiar world of 'TAD...DCA' and since in all liklihood your function will need to execute in PDP-8 mode (but may be relocated almost anywhere (including right on top of an index register!) part of the battle is simply learning about something new. At the moment it appears best to use FORTRAN IV 'as is' for number crunching rather than trying to seriously modify it for other purposes.

What sort of data types are available in these languages? Both BASIC and FORTRAN IV use a 3-word floating point format with 23 bits of mantissa and 11 bits of exponent (plus signs). FORTRAN II uses a 27-bit mantissa with a 7-bit exponent which provides an extra digit of accuracy at the expense of a smaller range of magnitudes. U/W-FOCAL uses a 4-word floating point format with 11 bits of exponent and 35 bits of mantissa. This translates to '10-digit' accuracy with a magnitude range of 10^614. Single and double precision integer array storage is also available via the FCOM/FBUF functions. FORTRAN II also supports signed 12-bit integers while FORTRAN IV offers both complex and double precision variables. The latter have 59 bits of mantissa - but are only available if you have the appropriate floating point processor. Thus for most installations, U/W-FOCAL offers the most accurate math package available for a PDP8 processor. For those acquainted with earlier versions of FOCAL, let me add that the internal functions are now just as accurate as the basic arithmetic operations.

Branch commands: BASIC has a 'logical IF' which provides a single branch based upon the truth of a postulated arithmetic relationship. This 'looks' convenient, but gets 'messy' in practice since one must frequently test several different possible relationships. Thus to decide if A is <, =, or > B requires the following logic: IF A<B THEN 100\IF A=B THEN 200\. In contrast, both FOCAL and FORTRAN have arithmetic IF tests which can dispose of this question abruptly: IF (A-B)100,200,300. In FOCAL the IF command is even more flexible: any statement number may be omitted to indicate that the program is to continue with the next command. Thus one has both the simplicity of a 'logical IF' with the power of a complete 'arithmetic IF' command. To illustrate: IF (N-1),1.1;IF (N-2),2.1;IF (N-3),3.1. This is a sieve with only 'equality' causing a branch. This particular ex-

ample would be better programed as a 'Computed GOTO'. This element is not available in BASIC but is available in both FOCAL and FORTRAN. In FOCAL one would write: JUMP (N) 1.1,2.1,3.1 which in FORTRAN is: GOTO (11,21,31),N. Anot er form of program transfer is the 'Assigned GOTO'. Again this is not available in BASIC or in FORTRAN II, but in FORTRAN IV consists of a statement such as: ASSIGN 101 TO J...GOTO J. In FOCAL this is easy accomplished y using an arithmetic expression as a statement number. Thus we could say: SET J=10.1 . . .GOTO J. Obviously however, the ability to use computed line numbers in branch commands is much more powerful than the analogy with FORTRAN branches might indicate. The following little loop, for instance, will call three different subroutines: FOR I=13,15;DO I. This is equivalent to: DO 13,14,15. FOCAL also has a conditional subroutine call, the 'ON' command mentioned earlier. This command functions like the arithmetic IF command, but performs a subroutine call to one of three possible routines depending upon the sign of the expression: ON (A-B)1,2,3. When you consider the statements necessary to implement the same control function in FORTRAN or BASIC, you can begin to appreciate the elegance of this language.

A very important consideration for any language running under the OS/8 operating system is what sort of file manipulation routines are available. BASIC provices for up to 4 concurrent files which can be either input or output files. These files may be opened by the program using any sort of user supplied name, and can be declared to be either ASCII or binary files. There is a provision for 'rewinding' a file, i.e. restarting from the beginning, but no provision for 'random access'. Any program I/O except to or from the Teletype must occur through one of these files. There is no provision for specifying the anticipated size of an output file.

U/W-FOCAL can have one output file and two input files. One of the input files may be used for reading -or- writing binary data. The other one (and the output file) are can only be used with ASCII data. To alleviate the lack of OS/8 ports, UWF has an optional internal Lineprinter handler so that the output file may be used for something else. There is a 'rewind' command for the input file, and an 'abort' command for the output file which avoids having to 'close' a temporary file in case an error occurs. All file names may be specified at run time using a computed file name and the 'square bracket' option is available to assist in optimizing file location. A novel approach to file names consists of specifying part of the name in the program and enclosing a variable numeric part in parentheses. Thus the command 'OPEN INPUT FILE(I)' may be used (in a loop perhaps) to read any file from 'FILE0' to 'FILE99'. It is also possible to specify an OS/8 -block number- in place of the file name. This only works for INPUT files, but may be used to speed program loading and file searches on DECtapes. It has also proven to be a valuable system tool. A specialty available in UWF which is not offered elsewhere is the ability to list the directory of any device either in toto, or selectively (including empties). It is also possible to delete files or to create 'dummy' files of any length. These file management features are very useful, for example in recovering 'lost' files or repairing bac directories. Finally, the binary file provides true rancom access to any word in any block on an entire OS/8 mass-storage device. Four data modes are available: signed or unsigned 12-bit integers, signed 24 bit integers and 4-word floating point numbers.

FORTRAN II provides a single OS/8 input and output file, but also has its own internal handlers for the papertape reader/punch and the card reader, lineprinter. A minor nuisance is the need to actively specify the '2-page' handler option to the loader if you expect to use such a handler. All file and device names may be specified at run time although there is a minor difficulty with names which are not six characters long. There is no 'rewind' operation. The files contain only ASCII data unless you have a TC08 DECtape controller, in which case you can read and write binary files with complete random access.

In FORTRAN IV there are 9 logical units which may be assigned in any number of ways. Normally 4 of these logical units are assigned to internal handlers leaving 5 available for general-purpose OS/8 I/O. Up to 8 could be used for this purpose however. File names must be specified via the Command Decoder when loading the program. They can -not- be specified by the program itself. This is distinct disadvantage since it means that it is not possible to automate file operations. However, there are many different sorts of data structures possible: formatted files (ASCII), unformatted files (sequential binary), and direct access files (random access binary files). Both REWIND and BACKSPACE operations are available.

Finally we will consider how these languages support various other devices. BASIC has a support package available for the LAB8/e which provides routines to operate the clock, drive the display and acquire data from various A/D channels. The 'new' EAE is also supported by the BASIC Run-time system. FORTRAN IV also offers support for all the LAB8/e peripherals as well as the PDP12. It also supports the 8/e EAE, but again, not the earlier one. One other device which is supported quite well is the incremental plotter, both the old and the new models. Special subroutine calls are used to access the plotter which is not considered as one of the 'regular' I/O devices. FORTRAN II offers no support for devices other than those controlled by standard OS/8 handlers.

U/W-FOCAL offers support for an extremely wide range of optional hardware. The LAB8/e is fully supported as is the PDP12. Of special note for both of these machines is the availability of a 'scope overlay which permits using the CRT for program development as well as displaying graphical output. The Tektronix graphics terminals are also supported, as are all styles of incremental plotters. The plotter routines allow both plotting and annotation, treating the plotter as just another output device. Thus it is possible to list programs on the plotter in the same way that you would get a listing on a lineprinter, or punch a paper tape. Both the old and the new EAE hardware is supported and although not standard, routines for powerfail/auto-restart have also been implemented.

Miscellaneous: All 4 languages offer double subscripting; BASIC uses subscripts beginning with '0' while FOCAL and FORTRAN start from '1'. FORTRAN IV has implied DO loops for transmitting array elements, FORTRAN II, BASIC and FOCAL do not. FOCAL has a slightly different set of arithmetic priorities in that multiplication takes precedence over division rather than being of equal stature. This only affects expressions such as A/B*C which is A/(B*C) in FOCAL and (A/B)*C in all other languages. UWF, BASIC, and FORTRAN IV all provide a way to print the current system date; in addition, UWF can change it! The program name and the current date are saved in the program 'header' line making it very easy to identify the version of program when it is loaded.

Implementation considerations: BASIC and FORTRAN IV make good use of memory since they are coded to ignore field boundaries. FOCAL and FORTRAN II do not attempt to cross fields and are thus restricted to one field (maximum) for the main program, using an additional field for storing variables. BASIC uses overlays in order to fit all of the run-time system into 8K and still have some room for the program. This has serious implications for operation on a tape system, especially if you are not thoughtful about mixing I/O and string functions, for example, since these reside in different overlays. The 8K version of UWF can only use 1-page handlers and has, obviously, some restrictions on the size of the program and the number of variables. Not all of the features mentioned above are available in the 8K version. Unlike FORTRAN IV and BASIC, a separate version of UWF is required for different memory sizes. This is a result of optimizing the assembly for each configuration. All four languages will operate in the background under RTS8.

In conclusion I hope I have presented a quick overview of both U/W-FOCAL and its contenders - OS/8-BASIC and FORTRAN II/IV - such that it appears that U/W-F is, indeed, a serious language for the PDP8 programmer, and one with very pronounced advantages for some applications. Many people have participated in the development throughout the years. I would like to thank especially Mr. Marquis Woods for his many fruitful suggestions, Paul Diegenbach for his invaluable assistance with the manual, Jim Crapuchettes for his deep insight into the 'whys and wherefors' of FOCAL, Tom McIntyre for supplying me with a decent editor (SCROLL) and the latest copy of RUNOFF. The current version would never have happened without the support of C. R. Tamasi and the Hydrographic Service (Canada). Closer to home, Dr. A. L. Khiram has been more than patient through endless revisions, and last, but foremost, a special note of thanks to my family.

Jim van Zee has written a well composed article comparing the various OS/8 high level languages. (Well not all. COBOL, ALGOL and LISP are also in OS/8 - Did you know that?)

I have also at various times tested most of the available OS/8 languages, but the evolution at our installation has been slightly different to that described by Jim. I shall try to give you my views on some of the points he raises.

Let me start with two general remarks.

1. I do not think it is quite fair to compare the end line of the evolution of FOCAL as produced by the extremely good FOCAL users, like Jim himself, with DEC standard products in the form of BASIC and FORTRAN. We must in the comparison include user produced enhancements of these languages.

2. The main part of my discussion will focus on comparisons between FORTRAN IV and FOCAL, these representing more or less the two extremes of the different lines of program development.

Let me just give some short remarks on the other two relevant languages.

a) BASIC. I agree with most of the remarks Jim makes on BASIC. However, for those who feel that the big drawback of OS/8 BASIC is its editor, let me just point out that there exists another BASIC usable under OS/8 produced by EDUCOMP Corporation, Hartford, USA. In this version the editor is an integral part of the program and is much more powerful than the OS/8 BASIC editor. There also exists an highly interactive BASIC compiler for lab usage (Lab BASIC), an extension of 8K BASIC. I have a small patch making it usable under OS/8 in 12K of core.

b) As to FORTRAN II, I have almost forgotten it as an useful language. However, two small remarks. There exists a subroutine in the DECUS Library which makes it possible to do unformatted input to FORTRAN II with the same kind of power that the FORTRAN IV input has and let us not forget that the FORTRAN II is a very powerful language if you have a lot of integer manipulation to do in your program. Any program which does floating and integer calculation with a heavy slant towards the integer part, runs much faster in the FORTRAN II than in any other language, due to the fact that the FORTRAN II is the only language which uses real true PDP-8 integers to handle the integers with only PDP-8 code to do integer arithmetic.

Now let us look at the specific points that Jim has brought up, specially then with reference to FOCAL and FORTRAN IV.

As Jim starts to point out, FORTRAN IV is really a language optimized to utilize all the possible power of the PDP-8 system rather than utilizing the small system as well as possible. I think a good deal of the differences between mine and Jim's approach to the languages lies here. Jim works in an university environment, where people do a lot of the programming on a half hobby basis. It is not considered "funny" if you spend a lot of time

on getting a program in on a single line. I work on an industrial installation where we want to get the maximum power out of our machine and where it is cheaper to pay for machine hardware than for human resources. Normally, it is better to put in another field of core in the machine than to spend a lot of time reducing the core requirements of the program. Jim also mentions the CPU running times and his attitude is that in most PDP-8 installations the CPU time is simply always so short that it is not worth talking about in relation to the human times. This attitude is not quite true as you well know. We have a fast PDP-8 with a FPP. Even in this machine we sometimes get running times countable in half hours and hours. Maybe you then would say, so why run them on the PDP-8? Well, the thing is that the PDP-8 is there, the time is available so I run them on it. With the FPP and FORTRAN the program may take half an hour to run in the PDP-8. If I went along to a large machine, at a computing center, it would take at least a day before I had my answers, so I prefer to run it on the PDP-8. And even in the true PDP-8 laboratory environment running time can become critical. If you do laboratory work and have calculations including fourier transforms, it is very easy to get CPU times running into the minutes even with a FPP. And as reasonable speed factors are setting FOCAL to 1, FORTRAN without FPP about 10 and FORTRAN with FPP about 2^{00} it is quite evident that these programs are very difficult to run in FOCAL.

The conciseness of FOCAL is a very neat point in many cases. If I want to write a small program to tabulate the logarithms of a function between two given values, I would almost certainly write it in FOCAL. It will only require a few lines of code maybe one single statement to achieve it. In FORTRAN my amount of coding would be much larger. However, if I write a program this is two or three pages long in FOCAL with the conciseness that FOCAL can give me, put it away for a month and then try to use it again, I have to, as Jim so well puts it, soak the program for several hours before I can use it. In FORTRAN this is not quite so true. A reasonably well written FORTRAN program is much easier both for me and for somebody else to modify. It is not quite true that this can be solved by writing the FOCAL program wordy because, as Jim knows as well as anyone of us, the content of the FOCAL program is saved as it is, while the FORTRAN program is compiled. Therefore, the FORTRAN program takes no more space to save if its variables are given 5 letter names instead of one letter names or if it is heavily commented, while a FOCAL program quickly runs out of space if I do not try to conserve space by keeping down the variable names and the amount of comments.

I certainly agree with Jim that the interactive programming feature of FOCAL is much stronger than that of any other language. However, comparing to large installations, I would say that OS/8 BASIC and FORTRAN are not interactive. The difficulty of running back and forth between say TECO and FORTRAN in a disc based OS/8 system is not great. Changing a line in a FORTRAN program and rerunning it is done in a matter of a few seconds. This situation is of course quite different if the media is DEC tape in which case running FORTRAN or BASIC becomes a very frustrating non-interactive process. Again this shows what I said in the beginning. We in our case can find it easier to pay for a disc than to pay for the time necessary to develope a required program in another language.

Let us look at some of the nuisance points that Jim brings up.

Firstly, FORTRAN has a format which is equivalent to FOCAL's format, the G format, in which I just define the number of digits I want and FORTRAN changes the decimal point to fit my requirement. Secondly, there is a definite usage of the asterisc field (oh, I certainly agree that it is often irritating when a program has run for quite some time, suddenly a field of asteriscs is produced as the only answer). If you write a program which is to be run many times, it is very neat to have the program put its output in nice vertical columns. In this case, it can be annoying when a column suddenly is destroyed by the language deciding it needs a wider field for a digit than I thought it would. Which of these two points is more difficult, I do not know. My approach is to use the G format in FORTRAN, whenever I am unsure what the result will be. The asteriscs in FORTRAN have another usage by the way. If I want to mark a field with asteriscs, as it is extremely difficult in FORTRAN in this case to put an error text in the field, something which FOCAL will allow me to do quite nicely.

As Jim also has remarked, the unformatted input reader routines in FORTRAN IV as they are now, are very powerful, more powerful than those of FOCAL. With the DECUS published patches, it is possible for the executing program to check if the input was correct and reask for the input if there was a numerical error in it. This I do not think is possible to do in any of the other languages.

Jim seems to think that it is very difficult to add new routines to FORTRAN. When talking about the subroutines, Jim forgets the most powerful feature of FORTRAN, that of its standardization. It is possible to use many hundred subroutines available from various sources in a FORTRAN program without any kind of reediting of the subroutine. We have e.g. the SSP package of subroutines, many of which are very useful. It is also possible to use many programs as such as available from various sources. We have implemented some of the BMD programs in a PDP-8 computer. The approach to the usage of the assembler subroutines shows how dependent this is on a thorough knowledge of the system you are working with. I would say that one of the reasons that I prefer FORTRAN IV over the FOCAL and BASIC languages is because of the easy calling up of assembler subroutines. In the first place there are now ways of calling PDP-8 routines without ever looking at the RALF assembler code. Such routines are available in the DECUS Library written by Robert Phelps. However, I do not think it is very difficult to learn and understand the FPP code. The difficulty of the same order is that of learning the floating point package used by the interpreters and the code one then has learnt is very much more powerful. Also to write to a subroutine to do a specific job in FPP code or in PDP-8 code means that I only have to write and compile this piece of code, nothing else, and then load it up with the FORTRAN system. This is of course the basis of the true subroutines used in FORTRAN and I find that it gives a very high power to the combination of FORTRAN and assembler code. It is even quite easy to link in a large PDP-8 code. We have an example of one such code which fills the whole of field one and which is then linked up to FORTRAN routines for the number crunching. The PDP-8 code existed long before the FORTRAN IV code and was simply relocated from field zero and linked up with FORTRAN, a trick which would be very difficult to do in the FOCAL atmosphere.

As to the question of data formats, FOCAL comes out definitely first. One of the cases where I turn to FOCAL is when I need an high precision calculation of such format that FORTRAN IV cannot do it. However, I will soon be receiving a PDP-8A with a FPP and with this hardware the power of the complex and double precision variables in FORTRAN IV becomes available. This is an extremely powerful double precision format, 59 bits of mantissa, and with the PDP-8A and FPP it will become more and more used in the PDP-8 installations. I am not going to question Jim's point of the different features of the programming languages, branch, statements etc. Jim's points are well brought out and I quite agree with him that when looking at that kind of detail, FOCAL quite often comes out ahead.

The file manipulations of FORTRAN IV of course are the strongest of any of the high level languages. In contrary to what Jim says, it is possible to call USR from the program. It is possible to construct program names. We have programs running which use the current date as set in the date word as a file name. Again we rely on the routines constructed by Robert Phelps. But again I think it is unfair not to include this kind of features in such a comparison when we are comparing against U/W-FOCAL. When it comes to support of various devices, again I think FORTRAN has one of the easiest ways of implementing such. In the first place, FORTRAN supports most of Lab 8 devices as standard (in some cases the standard support is that for PDP-12 device, however, the DECUS routines are available for the PDP-8 devices). It also supports the FPP12 which makes for an extremely fast number crunching, a system which is unbeatable even by many PDP-11 installations. Adding new devices, either as standard asynchronous calls or as interrupt driven or driven through the clock mechanism, is not difficult and is supported in the original program package.

In conclusion let me summarize my views. We are very grateful for people like Jim for constructing extremely powerful variants of FOCAL. Certainly carry on with this work! Give us FOCAL variants that will handle as much as possible of the sophisticated hardware appearing on the scene, that will utilize FOCAL's ability for the highly driven interactive programming, where as you say Jim, I can sit down and type a few commands to read the value from an AD converter and calculate the sine of that value. But let us also be grateful to DEC for the construction of FORTRAN IV. A program which really comes to its rights in large PDP-8 installations like ours where we find that the majority of our programs require 16K of core and even then will run into 4,5 maybe 10 overlays. Programs even with a FPP can run for several minutes, sometimes up in the half-hour range. Using the fast disc devices compiling and loading of the programs, specially with the support of the batch monitor, is not the slow and tedious process that it is on a DEC tape system. Finally, just so that nobody will misunderstand me, let me say that I have worked myself up from a 4K PDP-8 system with paper tape reader and punch up to the system which I am now running, a disc based system with dual processors, one running timesharing and one running a FPP. However, I shall come back later with a separate story to give an idea of the installation.