K/CSD/TM-20

Contract No. W-7405 eng 26

COMPUTER SCIENCES DIVISION

A COLLECTION OF FORTRAN SUPPORT ROUTINES

Steven B. Cliff
Computing Applications Department

JANUARY 1978

UNION CARBIDE CORPORATION, NUCLEAR DIVISION
operating the
Oak Ridge Gaseous Diffusion Plant . Oak Ridge National Laboratory
Oak Ridge Y-12 Plant . Paducah Gaseous Diffusion Plant
for the
DEPARTMENT OF ENERGY

TABLE OF CONTENTS

ABSTRACT

Descriptions of several routines designed to support and extend FORTRAN programs used on the IBM 360/370 series computers are included in this document. These routines, which have been used in a variety of programs, run the gamut from input processors to timers, to absolute address accessors, to mathematical analysis routines. Most of the routines are written in IBM 360/370 Assembler Language.

ACKNOWLEDGMENTS

The work of B. D. Dingus in preparing some of the graphs in this report is acknowledged.

Fiscal support for the development came from a variety of sources, including the Computing Applications Department of the Computer Sciences Division, the Engineering Technology Division, and the Operations Analysis and Planning Division of Union Carbide Corporation and by National Science Foundation Grant GK-37434 and the National Institutes of Health Grant HL-15564 administered by Dr. J. H. Forrester of the University of Tennessee. This support was essential to the development of these routines.

Discussions with E. D. Drennen, J. E. Park, L. I. Schlemper, and B. D. Dingus contributed to the concepts for some of these routines.

PART I

TRIDIG - A TRIDIAGONAL EQUATION SOLVER

PART I

TRIDIG - A TRIDIAGONAL EQUATION SOLVER

An Assembler Language equivalent to the tridiagonal equation solver, TRIDIG, as developed by J. E. Park [1], has been developed for use on IBM 360/370 computers. Particular attention was given to fully optimizing the use of the pipeline/parallel processing capabilities on the Model 195 at ORGDP, as described in IBM documentation [2].

TRIDIG is invoked by

CALL TRIDIG (A,B,NA,NB,MIL,NDIM)

where

NA	is the number of the node with which the first equation is associated,
A	is the tridiagonal matrix A(NDIM,3),
A(N,J)	with J=1,2,3, are the -, 0, + elements for equation for the Nth node,
B	is the constant vector in the matrix equation on entry and solution vector on exit,
MIL	is ignored, but must be present to maintain compatibility with FORTRAN TRIDIG,
A & B	are double precision.

CONSTRAINTS:

A must be "well conditioned." (See Ref. [1].)

$1 \leq NA, NB, NDIM \leq 100$

$NA < NB < NDIM$.

The assembler version is faster than the FORTRAN-H, OPT=2, by 52% for a system of three equations, falling asymptotically to 30% for 93 equations. Savings of 10% or more in the total CPU time of some applications programs have already been observed, making the investment of writing this version justified with a reasonable date of cost recovery expected.

In testing TRIDIG, it became desirable to compare it with not only FORTRAN-H, OPT=2, but also the other available compilers and at other optimization levels. Thus the local ORNL FORTRAN (FTN63), IBM FORTRAN-H, OPT=0 (H0), IBM FORTRAN-H, OPT=1 (H1), IBM FORTRAN-H, OPT=2 (H2), IBM FORTRAN-G (FORT-G), and the IBM FORTRAN-H, extended plus, OPT=3 (FORT X), compilers were all tested along with the assembler version of TRIDIG. In all cases, the overhead was constant with only the compiler of the test routine varying. All runs were made on the IBM 360/195 at ORGDP, with the same system of equations for the same number of samples. The results of these tests are given in the following graphs.

Figure 1 shows the time required to perform 5000 solutions as a function of the number of equations. In each case, the relationship is linear, as expected, since the time of the solution technique is known to be proportional to the number of equations. The overhead is a constant 0.61 seconds for all cases and is included in the time for all plots. Figure 2 is of the same data, but with different scaling to ease discriminations of H2, FORT X, and Assembler.

Figures 3 and 4 have the data of Figures 1 and 2 normalized relative to HOPT2. Thus HOPT2 is displayed as a straight line with faster compilers below it and slower ones above. Figure 3 especially accents the poor performance of the G, FTN63, HOPT0, and HOPT1 which are steadily diverging from the HOPT2 efficiency level. The XOPT3 and Assembler routines are continually getting better as the number of equations increase as is clearly visible in Figure 4.

Thus, TRIDIG is a good example of a CPU intensive routine which can be coded in Assembly Language with considerable CPU savings. The appendix is a complete source listing of TRIDIG.

REFERENCES

1. James E. Park, Utility Routines for Tridiagonal Matrices, UCCND/CSD/INF-74, November, 1975, pp. 13-14.

2. IBM System/360 Model 195 Functional Characteristics, GA22-6943-4, October, 1975.

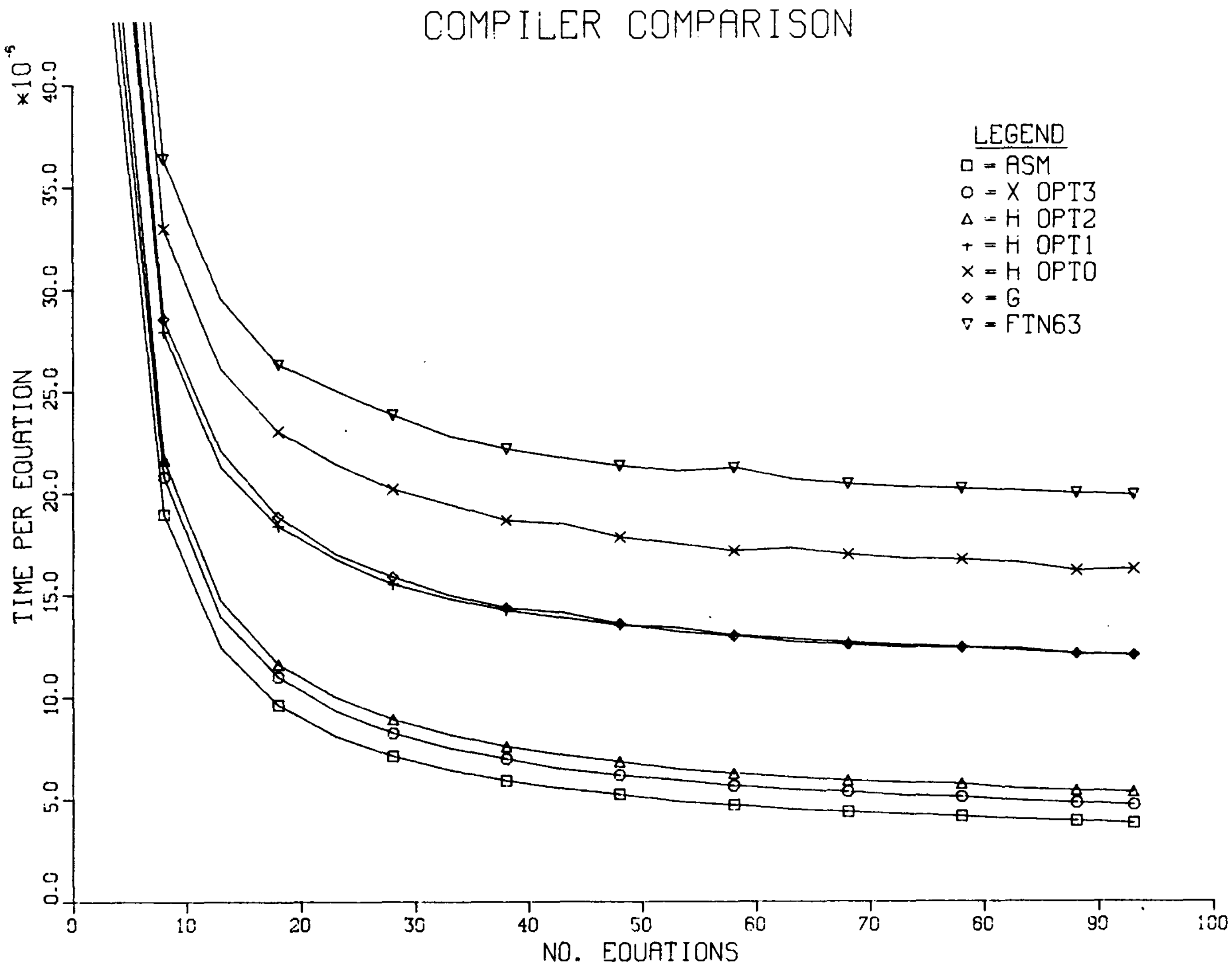
COMPILER COMPARISON
LEGEND
□ = ASM
○ = X OPT3
△ = H OPT2
+ = H OPT1
× = H OPT0
◇ = G
▽ = FTN63
TIME PER EQUATION
×10⁻⁶
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
NO. EQUATIONS
0
10
20
30
40
50
60
70
80
90
100

Figure 1

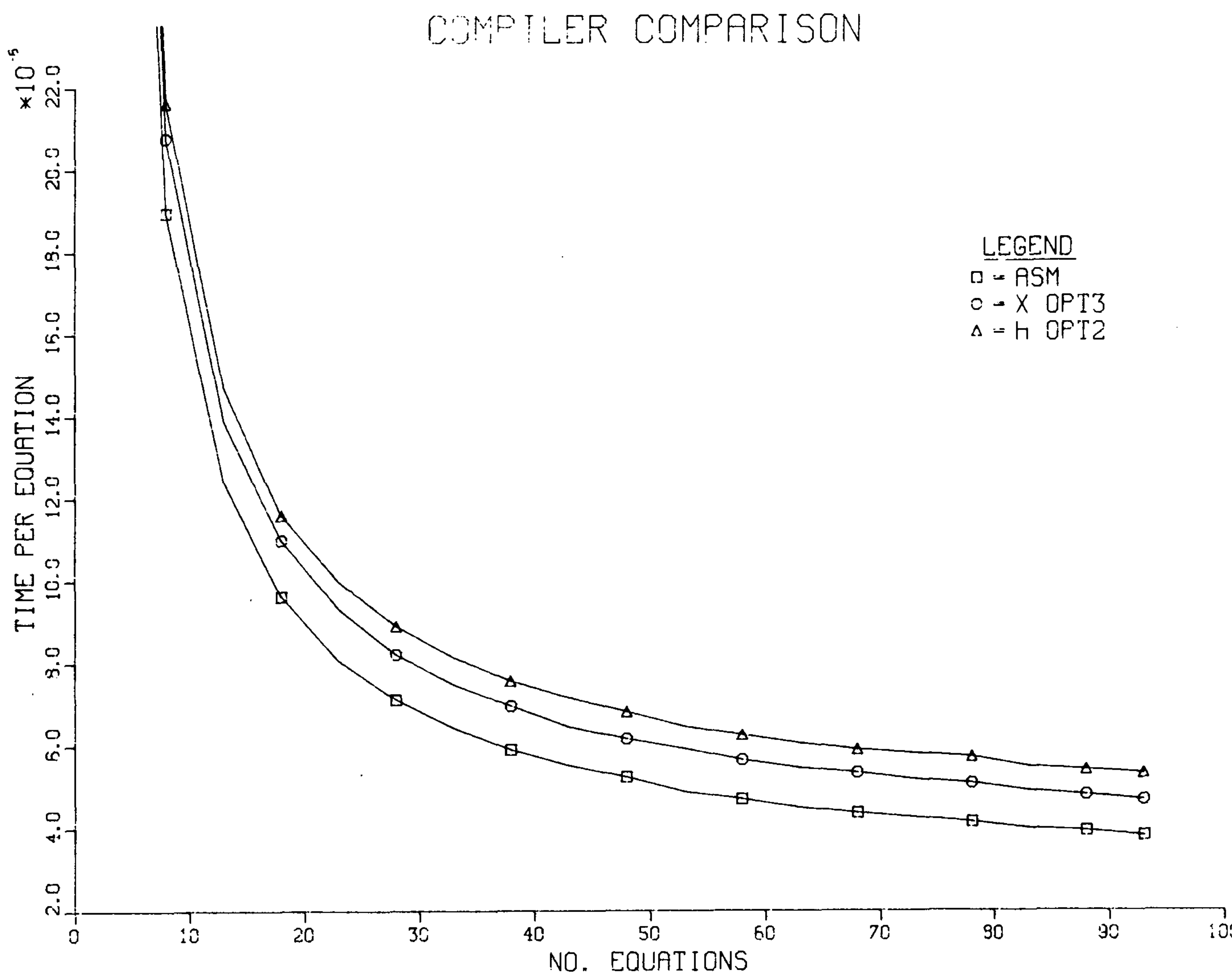
COMPILER COMPARISON
TIME PER EQUATION
*10^-5
NO. EQUATIONS
LEGEND
□ = ASM
○ = X OPT3
△ = H OPT2

Figure 2

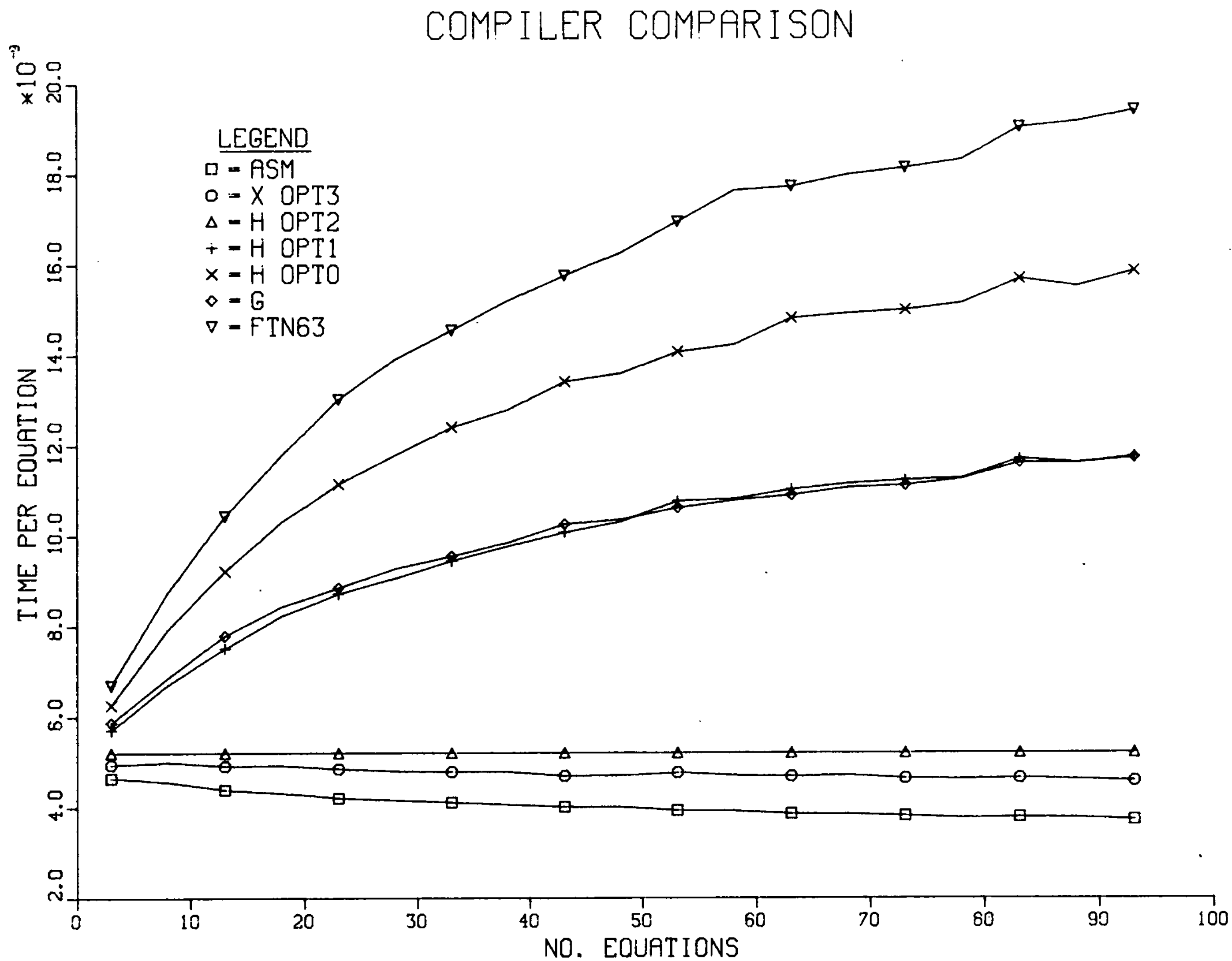
COMPILER COMPARISON
LEGEND
□ - ASM
○ - X OPT3
△ - H OPT2
+ - H OPT1
× - H OPT0
◇ - G
▽ - FTN63
TIME PER EQUATION
×10^-3
2.0
4.0
6.0
8.0
10.0
12.0
14.0
16.0
18.0
20.0
0
10
20
30
40
50
60
70
80
90
100
NO. EQUATIONS

Figure 3

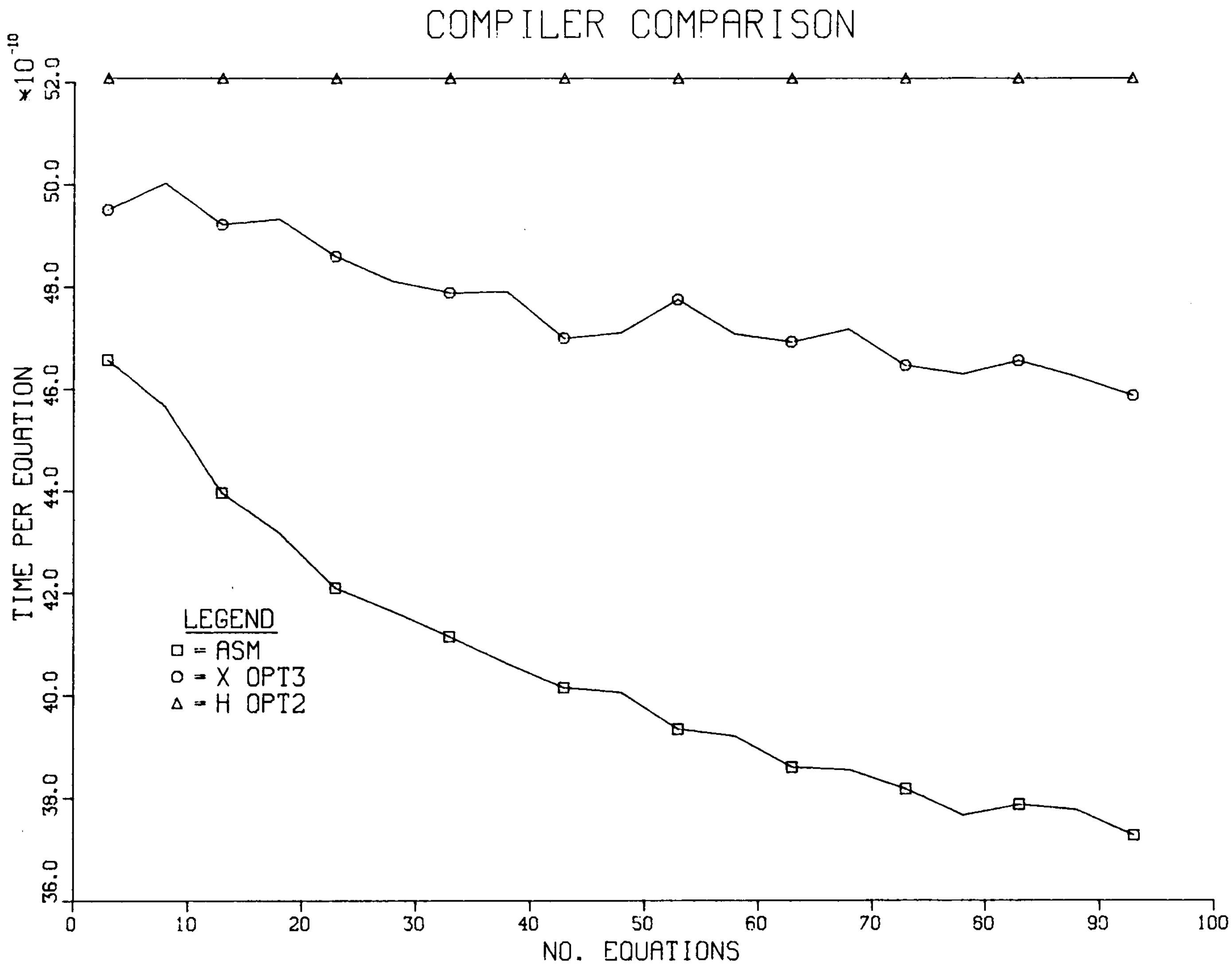
COMPILER COMPARISON
$\times 10^{-10}$
52.0
50.0
48.0
46.0
44.0
42.0
40.0
38.0
36.0
TIME PER EQUATION
0
10
20
30
40
50
60
70
80
90
100
NO. EQUATIONS
LEGEND
□ = ASM
○ = X OPT3
△ = H OPT2

Figure 4

PART I, Appendix 1

Source Listing of TRIDIG

```
TRIDIANG CSECT 0
*
* TRI-DIAGONAL EQUATION SOLVER
*
* BASED ON FORTRAN SUBROUTINE TRIDIG BY J.E. PARK
*     AND IS OPERATIONALLY EQUIVALENT EXCEPT NO DEBUG TYPE WRITES ARE
*     AVAILABLE.  HOWEVER, THE SAME ARGUMENT LIST IS USED.
* DEVELOPED  BY STEVEN B. CLIFF JUNE 8,1977
* USAGE:
*
*    CALL TRIDIG (A, B, NA, NB, MIL, NDIM)
*
*  NA IS NUMBER OF NODE WITH WHICH THE FIRST EQUATION IS ASSOCIATED.
*  NB IS NUMBER OF NODE WITH WHICH THE LAST  EQUATION IS ASSOCIATED.
*   A IS TRIDIAGONAL MATRIX A(NDIM,3).   A(N,J) WITH J=1,2,3 ARE -,0,+
*             ELEMENTS FOR EQUATION FOR NTH  NODE.
*   B IS CONSTANT VECTOR IN MATRIX EQUATION ON ENTRY AND SOLUTION
*             VECTOR ON EXIT
* MIL IS IGNORED
*NDIM IS FIRST DIMENSION OF A, ONLY DIMENSION OF B
*
* CONSTRAINTS:
*
*   A MUST BE WELL CONDITIONED.
*   NA,NB,NDIM MUST BE BETWEEN 0 AND 101
*
* THIS ROUTINE IS OPTIMIZED PER SUGGESTIONS IN
*      IBM SYSTEM/360 MODEL 195 FUNCTIONAL CHARACTERISTICS
*      GA22-6943-3
*
* 56 EXECUTABLE INSTRUCTIONS ARE HERE VS. ALMOST 200 BY FORTH,OPT=2
* FORWARD  LOOP HAS 11 VS. 23
* BACKWARD LOOP HAS  6 VS. 16
*
*
*
*  REGISTER USAGE AND ASSIGNMENTS:
*
*  GENERAL:
*
ARG   EQU  1   POINTER TO ARGUMENT LIST
A1    EQU  2   BASE FOR A(X,1) THUS A(22,1) = A1(22)
A2    EQU 11   BASE FOR A(X,2) THUS A(22,2) = A2(22)
A3    EQU 12   BASE FOR A(X,3) THUS A(22,3) = A3(22)
B     EQU  3   BASE FOR B(X)
Z     EQU 14   BASE FOR Z(X)
U     EQU 10   BASE FOR U(X)
NA    EQU  4   ADDRESS OF NA, THEN NA
NB    EQU  5   ADDRESS OF NB, THEN NB
NDIM  EQU  7   ADDRESS OF NDIM, THEN DIM
BASE  EQU 13   BASE REGISTER FOR LOCAL ADDRESSING
SAVE  EQU 13   SAVE REGISTER FOR SYSTEM LINKAGE
EP    EQU 15   HAS ADDR OF ENTRY POINT
LINK  EQU 14   RETURN ADDRESS IS HERE
X     EQU  6   INDEX REGISTER
CM8   EQU  8   CONSTANT -8 - MUST BE EVEN; USES REG+1 (BXLE,BXH)
```

```
C8     EQU  8    CONSTANT 8 - MUST BE EVEN: USES REG+1 (BXLE,BXH)           TRID0570
*                                                                           TRID0580
*                                                                           TRID0590
*                                                                           TRID0600
*   DISPLACEMENTS:                                                          TRID0610
*                                                                           TRID0620
K      EQU  8    B(K)=K(X,B)                                                TRID0630
KP1    EQU 16    B(K+1)=KP1(X,B)                                            TRID0640
KM1    EQU  0    B(K-1)=KM1(X,B)                                            TRID0650
*                                                                           TRID0660
*   FLOATING POINT:                                                         TRID0670
*                                                                           TRID0680
RZ     EQU  6    REGISTER STORAGE FOR Z                                     TRID0690
EL     EQU  2    INTERMEDIATE QUANTITY EL                                   TRID0700
RU     EQU  4    REGISTER STORAGE FOR U                                     TRID0710
C1     EQU  0    CONSTANT 1.0D0                                             TRID0720
RB     EQU  6    REGISTER STORAGE FOR B                                     TRID0730
         ENTRY TRIDIG                                                       TRID0740
         USING TRIDIG,EP                                                    TRID0750
TRIDIG   B     GO             SKIP OVER ID AND STORAGE AREA                 TRID0760
         DC    XL1'7'         SEVEN CHARACTERS IN NAME                      TRID0770
         DC    CL7'TRIDIG '   NAME=TRIDIG WITH PAD                          TRID0780
REGS     DS    9D             SAVE AREA FOR LINKAGE                         TRID0790
DBLE1    DC    1D'1.0'        F.P. CONSTANT 1                               TRID0800
CM1      DC    1F'-1'         INTEGER CONSTANT -1                           TRID0810
AUVECT   DC    A(UVECT-16)                                                  TRID0820
*                                                                           TRID0830
*                                                                           TRID0840
*                                                                           TRID0850
*                                                                           TRID0860
*                                                                           TRID0870
GO       STM   LINK,LINK-2,12(SAVE)    SAVE CALLER'S REGS                   TRID0880
         LR    X,SAVE         HOLD USER SAVE AREA ADDRESS                   TRID0890
         LA    BASE,REGS      SET MY SAVE AREA (ALSO BASE REG)              TRID0900
         DROP  EP             DO NOT NEED EP AS BASE                        TRID0910
         USING REGS,BASE      BECAUSE SAVE IS READY                         TRID0920
         ST    X,4(,SAVE)     PUT CALLER'S ADDRESS IN MINE                  TRID0930
         ST    SAVE,8(,X)     PUT MY SAVE AREA IN CALLER'S                  TRID0940
         LM    A1,NDIM,0(ARG) GET ADDRESES FOR ALL ARGUMENTS                TRID0950
         LA    C8,16           SET CONSTANT 8(16  FIRST) (IGNORE MIL)       TRID0960
         L     NA,0(,NA)      GET NA                                        TRID0970
         SR    A1,C8          SET A1 (MUST ALLOW FOR K)                     TRID0980
         L     NB,0(,NB)      GET NB                                        TRID0990
         SR    B,C8                                                         TRID1000
         L     NDIM,0(,NDIM)  GET NDIM                                      TRID1010
         SRL   C8,1              NOW ITS 8                                  TRID1020
         LR    A2,A1          SET A2                                        TRID1030
         SLL   NDIM,3         MULTIPLY NDIM BY 8                            TRID1040
         L     U,AUVECT       SET U: (ADCON INSURES ADDRESSABILITY)         TRID1050
         SLL   NA,3           MULTIPLY NA BY 8                              TRID1060
         AR    A2,NDIM               AS REQUIRED                            TRID1070
         SLL   NB,3           MULTIPLY NB BY 8                              TRID1080
         LA    Z,ZVECT-16      SET Z                                        TRID1090
         LR    X,NA           SET X                                         TRID1100
         LR    A3,A2          SET A3                                        TRID1110
         LD    C1,DBLE1       GET & KEEP CONSTANT 1.0D0                     TRID1120
```

```
         AR     A3,NDIM              AS REQUIRED
         LR     C8+1,NB         SET LIMIT FOR X
*
* HAVE ALL PRELIMINARY STUFF, NOW BEGIN CALCULATIONS
* :*
* :*     Z(NA)=B(NA)
* :         LD     RZ,K(X,B)      SAVE Z(NA) FOR LATER
* :         STD    RZ,K(X,Z)
* :*
* :*     U(NA)=1.0D0 / A(NA,2)
* :         LD     C1,DBLE1       GET & KEEP CONSTANT 1.0D0
* :         LDR    RU,C1          DO
* :         DD     RU,K(X,A2)        INVERSION
* :         STD    RU,K(X,U)
* :         AR     X,C8           INCREMENT X FOR LOOP
* :*     EL=A(K,1)*U(K-1)
* :FORWARD  LCDR   EL,RU          GET U(K-1), COMPLEMENTING IN PROCESS
* :         MD     EL,K(X,A1)     MPY BY  A(K,1)
* :*
* :*     Z(K)=B(K)-EL*Z(K-1)
* :         MDR    RZ,EL          -EL*Z(K-1)
* :         AD     RZ,K(X,B)      ADD B(K)
* :         STD    RZ,K(X,Z)      STORE Z(K), KEEP IT IN RZ FOR NEXT LOOP
* :*
* :*     U(K)=1.0D0/(A(K,2)-EL*A(K-1,3))
* :         MD     EL,KM1(X,A3)   -EL*A(K-1,3) DISCARD EL
* :         AD     EL,K(X,A2)     ADD A(K,2)
* :         LDR    RU,C1          GET 1.0D0
* :         DDR    RU,EL          DO INVERSION
* :         STD    RU,K(X,U)      STORE U(K), RZ WILL HAVE Z(K-1)
* :*
* :*    DO 51 K=NAT1,NB
* :* 51 CONTINUE
* :         BXLE   X,C8,FORWARD
* :*
* :*
* :         LR     X,NB           SET X TO LAST ELEMENT
* :         LR     CM8+1,NA          SET LIMIT TO NA
* :         A      CM8+1,CM1         BUMP LIMIT ONE BECAUSE OF BXH
* :         LCR    CM8,C8         SET INCREMENT TO -8
* :*
* :*     B(NB)=Z(NB)*U(NB)
* :*
* :         MDR    RZ,RU          Z(NB)*U(NB)
* :         STD    RB,K(X,B)      SET B(NB)
* :*
* :*
* :         AR     X,CM8          DECREMENT X FOR LOOP
* :*
* :*     B(J)=(Z(J)-A(J,3)*B(J+1))*U(J)
* :BACKWARD MD     RB,K(X,A3)     B(J+1)*A(J,3): B(J+1) IS IN RB
* :         LCDR   RB,RB          COMPLEMENT RB
* :         AD     RB,K(X,Z)      ADD Z(J)
* :         MD     RB,K(X,U)      MPY BY U(J)
* :         STD    RB,K(X,B)      STORE B(J);  B(J+1) IS IN RB
* :*
```

```
* :*      DO 52 K=NC,NB
* :* 52   CONTINUE
* :          BXH    X,CM8,BACKWARD
* :*
* :*
*
*
*       Z(NA)=B(NA)
*       U(NA)=1.0D0 / A(NA,2)
          LD     RZ,K(X,B)       SAVE Z(NA) FOR LATER
          LDR    RU,C1           D0
          STD    RZ,K(X,Z)
          DD     RU,K(X,A2)         INVERSION
          STD    RU,K(X,U)
          AR     X,C8            INCREMENT X FOR LOOP
          CNOP   0,8
* DO STMT GENERATS SOME OF THE INTRODUCTORY CODE AND MORE LATER; BUT
*       NONE HERE
*
* WHILE IN FORWARD LOOP, THE PRECEEDING Z & U ARE IN RU & RZ
* INTERMEDIATE QUANTITY EL, IS NEVER STORED AND THE TWO SUBTRACTS
*       USING EL ARE MADE ADDS BY COMPLEMENTING EL FIRST
*
*       EL=A(K,1)*U(K-1)
*
*       Z(K)=B(K)-EL*Z(K-1)
*
*       U(K)=1.0D0/(A(K,2)-EL*A(K-1,3))
FORWARD   LCDR   EL,RU           GET U(K-1), COMPLEMENTING IN PROCESS
          MD     EL,K(X,A1)      MPY BY  A(K,1)
          MDR    RZ,EL           -EL*Z(K-1)
          MD     EL,KM1(X,A3)    -EL*A(K-1,3) DISCARD EL
          LDR    RU,C1           GET 1.0D0
          AD     RZ,K(X,B)       ADD B(K)
          AD     EL,K(X,A2)      ADD A(K,2)
          STD    RZ,K(X,Z)       STORE Z(K), KEEP IT IN RZ FOR NEXT LOOP
          DDR    RU,EL           DO INVERSION
          STD    RU,K(X,U)       STORE U(K), RZ WILL HAVE Z(K-1) NEXT LOOP
*
*     DO 51 K=NAT1,NB
* 51 CONTINUE
          BXLE   X,C8,FORWARD
*
* FORWARD LOOP IS 9 WORDS LONG AND WILL BE CONTAINED FULLY IN THE
*     16 WORD INSTRUCTION STACK AND WILL BE EXECUTED IN LOOP MODE
*
* THE BACKWARD LOOP IS ESTABLISHED AS
*       DO 52 K=NB-1,NA,-1  (ALL MULTIPLIED BY THE WORD LENGTH-8)
*
*
*       B(NB)=Z(NB)*U(NB)
* RECALL: NB WAS LIMIT OF FORWARD LOOP, THEREFORE RU=U(NB); RZ=Z(NB)
* NOTE  : NO LONGER NEED RZ, SO USE IT FOR RB
*
*
*       J+=NB+NC-1
```

```
* NOT NEEDED
*
         LR    CM8+1,NA          SET LIMIT TO NA
         MDR   R7,RU           Z(NB)*U(NB)
         LR    X,NB            SET X TO LAST ELEMENT
         LCR   CM8,C8          SET INCREMENT TO -8
         A     CM8+1,CM1         BUMP LIMIT ONE BECAUSE OF BXH, NOT BXHE
         STD   RB,K(X,B)       SET B(NB)
         AR    X,CM8           DECREMENT X FOR LOOP
         CNOP  0,8
*     DO 52  K=NC,NB
* NO CODE NOW, BUT HAS EARLIER, AND WILL LATER
*
*     B(J)=(Z(J)-A(J,3)*B(J+1))*U(J)
BACKWARD MD    RB,K(X,A3)      B(J+1)*A(J,3): B(J+1) IS IN RB
         LCDR  RB,RB           COMPLEMENT RB
         AD    RB,K(X,Z)       ADD Z(J)
         MD    RB,K(X,U)       MPY BY U(J)
         STD   RB,K(X,B)       STORE B(J):  B(J+1) IS IN RB
*
*     DO 52 K=NC,NB
* 52  CONTINUE
         BXH   X,CM8,BACKWARD
*
* BACKWARD LOOP IS 5 AND ONE HALF WORDS LONG: IT WILL BE EXECUTED
*   FULLY IN THE STACK IN LOOP MODE
*
*     RETURN
         L     SAVE,4(,SAVE) UNLINK SAVE AREAS
         LM    LINK,LINK-2,12(SAVE) RESET THE REGISTERS
         MVI   12(13),X'FF'
         BR    LINK                 BYE
*
* THE U & Z VECTORS PER HIRASAKI'S FORMULATION ARE HERE. THEIR
*     DIMENSION OF 101 IMPOSES THE LIMITS ON NA,NB,NDIM LISTED ABOVE
ZVECT    DS    101D
UVECT    DS    101D
         END
```

PART II

CADTIMER - TASK TIMING ROUTINES

PART II

CADTIMER - TASK TIMING ROUTINES

One of the tools most needed in analyzing and improving software is accurate, precise measurement of the time spent in the various sections of a program. This information is essential not only to finding the CPU intensive portions but also to evaluating the effectiveness of modifications made in efforts to improve sections under scrutiny.

Prior to the present development, the best tools apparently available for use at ORGDP were the routines ICLOCK, ITIME, JSTIME [1] or similar routines developed at ORNL with precisions of one/one-hundredth (0.01) of a second. This level of precision is unacceptable to a computer as fast as the 360/195 with over 185,000 machine cycles (about 50,000 machine instructions) between clock "ticks". Frequently, programs have sections which are much shorter than one/one-hundredths of a second but which are executed thousands of times per use of the program.

Obviously, a set of better timing routines was needed. It is to fill this need that the CADTIMER routines were written. These routines use (via STIMER and TTIMER supervisor calls [2]) the 26.04166 microsecond clock available on the 360/195, allowing only 482 machine cycles (about 120 machine instructions) within a clock interval. It must be noted that, although the clock intervals are 26.04166 microseconds, the time is updated only every fourth tick. Thus, the CADTIMER routines have a limit of 104.16664 microseconds as the true time between ticks.

The CADTIMER general-purpose timing routines have a total of 21 entry points, 20 of which return an indicator of the time used by the task. The 20 time-evaluating entry points are named in the following manner:

$$\left[\begin{array}{c} \begin{bmatrix} \mathrm{I} \\ \mathrm{R} \\ \mathrm{D} \end{bmatrix} \quad \begin{bmatrix} \mathrm{S} \\ \mathrm{M} \\ \mathrm{H} \end{bmatrix} \\ \mathrm{I26} \end{array}\right] \quad \begin{bmatrix} \mathrm{TOT} \\ \mathrm{INT} \end{bmatrix}$$

The first letter indicates the type and length of the timer value returned with I, R, D referring to four-byte integer, four-byte real and eight-byte real, respectively. The second letter indicates the units of measurement with S, M, H referring to seconds, minutes, and hundredths of seconds, respectively. The single three-letter group, I26, refers to four-byte integers with timer units as units. The last pair of three-letter options select the interval over which measurement is to be made, INT representing the interval since the last call to an "INT" routine, while TOT represents the total time of the interval since the first call to any CADTIMER routine. Thus, to get the time in seconds as a single-precision real number since the last call to an "INT" routine, the entry point RSINT is used. (See Appendix 1 for more examples and details of the calling conventions.)

The 21st routine, NTIMC, returns as a four-byte integer, the number of times any of the CADTIMER routines have been called.

The first call to any of the routines (except NTIMC) is used as the setup call and a zero (of the appropriate type) is returned. Thereafter, each routine responds as its name implies. A maximum of 12 hours CPU time is allowed by any program which uses CADTIMER. Since CADTIMER uses STIMER and TTIMER, no other use of them should be made. Further, CADTIMER is not overlayable--it must be in the root segment of any overlay program.

These timer routines have already been used in situations where the precision afforded by the predecessor timers would have been totally unsatisfactory. Intervals as short as the previous clock's ticks can now be measured accurately allowing much finer study of the characteristics of programs than was previously available.

The complete source listing for CADTIMER is contained in Appendix 2.

REFERENCES

1. CSD Programmer's Notebook, November 10, 1975, pp. 20-7, 20-9, and 20-10.

2. Supervisory Services and Macro Instructions, IBM Manual GC28-6646-7, Sections 77 and 82.

PART II, Appendix 1

ADDITIONAL COMMENTS ON THE USAGE OF CADTIMER

Each of the 21 entry points has one argument and, because of FORTRAN conventions, may be invoked either as a FUNCTION or SUBROUTINE. For example, the sequence

```
 1.  IMPLICIT DOUBLE PRECISION (D)
 2.  CALL IMTOT (ISET)
 3.  ( 1.4E-2 Seconds of Computation)
 4.  R1V = RSTOT (R1A)
 5.  R2V = RSINT (R2A)
 6.  R3V = SNGL ( DSTOT (D3A))
 7.  (2.4E-2 Seconds of Computation)
 8.  R4V = RSTOT (R4A)
 9.  R5V = RSINT (R5A)
10.  R6V = RSINT (R6A)
11.  CALL IHTOT (I7A)
12.  (4.2 Seconds of Computation)
13.  R8V = RSTOT (R8A)
14.  D9V = DHINT (D9A)/100.0
15.  R10V = R8V - SNGL (D9A)
16.  R11V = I26TOT(I11A)*26.01466E-06
17.  CALL ITIMC (I12A)
18.  CALL RMTOT (R13A)
```

would result in the following values for the indicated variables. (The times for all statements except the three times explicitly noted are assumed to be zero.)

Variable	Value	Comment
ISET	0	The first call to CADTIMER always returns 0.
R1V,R1A	1.4E-2	Both the argument and functional value are REAL*4
R2V,R2A	1.4E-2	Since no previous TUINT routine has been called, the total time is returned.
R3V,D3A	1.4E-2, 1.4D-2	Total time is still 1.4E-2. This shows relationship between all RUYYY* and DUYYY routines.
R4V,R4A	3.8E-2	Total time since first call.
R5V,R5A	2.4E-2	Time since last TUINT routine (last call was Line 5).
R6V,R6A	0.0	Time since last TUINT routine (last call was Line 9).
I7A	3	Three-hundredths of a second has elapsed. Note that ISTOT would have returned zero.
R8V,R8A	4.238	Total time since Line 2.
D9A	420.E-2	420.E-2 seconds since Line 10.
D9V	4.2	D9V could have been set to DSINT. This shows relationship between TSYYY and THYYY routines.
R10V	3.8E-2	Total time before beginning of current interval.
R11V	4.238	Total elapsed time (same as RSTOT).
I26TOT	162739	Number of timer increments since Line 1. This line shows relationship between I26YYY and RSYYY routines.

---

*The nomenclature TUYYY is interpreted as follows. The T is the type of routine: I, R, or D. The U is the unit of the routine: S, M, or H. The YYY is the interval of measure next: TOT or INT. Thus RUYYY refers to all the routines that return real values, TUINT to any interval measuring routine, and TSYYY to any second returning routine.

Variable	Value	Comment
I12A	11	Number of calls to CADTIMER routines. This does not include calls to NTIMC.
R13A	7.06E-2	Total number of minutes since first call (Line 2).

Note again that the first call to any CADTIMER routine (except NTIMC) returns zero and initializes both the total and interval timers. Each successive call to TUTOT routine returns total CPU time since this first call. Each call to TUINT routine resets the interval time and returns the length of the interval. The type and length of the argument are the same as the functional value (if used) and the appropriate argument must be present. The length and type of the value returned depend on the first letter of the routine name; I is always four-byte integer, D is always eight-byte real, and R is always four-byted. These routines are related by

IUYYY=IFIX(RUYYY)=IFIX(SNGL(DUYYY))

RUYYY=SNGL(DUYYY).

Note that the IFIX function causes truncation and loss of significance for IUYYY routines. The SNGL function will cause loss of significance after about four seconds.

The second letter of each routine name determines the units of the value returned; M is minutes, S is seconds, and H is hundredths of seconds. These are related by

THYYY=TSYYY/100=TMYYY/6000.

Some loss of significance can occur with the divisions, but except for integer types, loss should be negligible as implemented in CADTIMER.

The exception to this two-letter typing and units rule is the I26YYY routines, which return the actual number of timer increments as a four-byte integer. Thus, these routines are related to RSYYY by

RSYYY=FLOAT(I26YYY)*26.04166E-6

and the earlier relationships can be used to derive correspondences between I26YYY and any TUYYY routine. No roundoff or truncation can occur with the I26YYY routines which afford maximum precision and accuracy, but in a less convenient form. CADTIMER carefully minimizes the errors in these conversions; hence, the user should get the correct type, length and unit by invoking the appropriate routine.

The last three letters of the routine name determine the interval over which the time is measured; TOT being the total CPU time since the first call to a CADTIMER routine, and INT being the interval since the last call to a TUINT routine unless no TUINT has been previously called, then it is equivalent to a call to TUTOT.

The exception to all of these naming conventions is the special routine, NTIMC, which returns the number of times any of the other CADTIMER routines have been called. NTIMC does not set the clocks, either total or interval, nor does it increment the number of calls counter. The value returned is a four-byte integer number of times called.

The 21 entry points and their use is described in Table 1. The type column refers to both the argument and functional value which are always the same. The naming conventions are clearly evident in all applicable routines.

Table 1

CADTIMER

ENTRY	VALUE		
	Type	Unit	Interval
IHTOT	I*4	.01 Sec	Total since first call
RHTOT	R*4	.01 Sec	Total since first call
DHTOT	R*8	.01 Sec	Total since first call
ISTOT	I*4	Seconds	Total since first call
RSTOT	R*4	Seconds	Total since first call
DSTOT	R*8	Seconds	Total since first call
IMTOT	I*4	Minutes	Total since first call
RMTOT	R*4	Minutes	Total since first call
DMTOT	R*8	Minutes	Total since first call
IHINT	I*4	.01 Sec	Interval since last INT call
RHINT	R*4	.01 Sec	Interval since last INT call
DHINT	R*8	.01 Sec	Interval since last INT call
ISINT	I*4	Seconds	Interval since last INT call
RSINT	R*4	Seconds	Interval since last INT call
DSINT	R*8	Seconds	Interval since last INT call
IMINT	I*4	Minutes	Interval since last INT call
RMINT	R*4	Minutes	Interval since last INT call
DMINT	R*8	Minutes	Interval since last INT call
I26TOT	I*4	26.04166E-6 Sec	Total since first call
I26INT	I*4	26.04166E-6 Sec	Interval since last INT call
NTIME	I*4	Number of calls	Since first call

PART II, Appendix 2

SOURCE LISTING OF CADTIMER

```
         MACRO
         BEGIN &EP1,&EP2,&EP3,&OTIME
         DROP  SAVE
         USING &EP1,EP        SET TEMPARY BASE
         ENTRY &EP1           PUT EP1 IN ESD
&EP1     B     &EP1.GO        SKIP ID
         DC    X'05'          LEN  OF ID
         DC    CL5'&EP1'      ID
&EP1.GO  STM   LINK,LINK-2,12(SAVE)  SAVE THE REGS
         LR    OLDS,SAVE      SAVE OLD SAVE AREA POINTER
         L     SAVE,ADREGS    GET  NEW POINTER
         MVI   SETARG,X'50'   SET EP OPCODE = SINGLE INTEGER STORE
         B     GO&SYSNDX      GO JOIN OTHER EP ROUTINES
         USING &EP2,EP
         ENTRY &EP2
&EP2     B     &EP2.GO
         DC    X'06'
         DC    CL6'&EP2'
&EP2.GO  STM   LINK,LINK-2,12(SAVE)
         LR    OLDS,SAVE
         L     SAVE,ADREGS
         MVI   SETARG,X'70'   SINGLE FLOAT STORE
         B     GO&SYSNDX
         USING &EP3,EP
         ENTRY &EP3
&EP3     B     &EP3.GO
         DC    X'06'
         DC    CL6'&EP3'
&EP3.GO  STM   LINK,LINK-2,12(SAVE)
         LR    OLDS,SAVE
         L     SAVE,ADREGS
         MVI   SETARG,X'60'   DOUBLE FLOAT STORE
         DROP  EP
         USING REGS,BASE
GO&SYSNDX LA   TEMP,&OTIME
         BAL   LINK,GETIT
         MEND
         SPACE 3
CADTIMER CSECT 0
         PRINT GEN
*
* VERSION 1.0  7- 1-77 STEVEN B. CLIFF
*
* THESE ROUTINES ALL ACCESS THE TTIMER & STIMER MACROS AND NO
*   OTHER ACCESS TO THEM SHOULD BE MADE.
*
          EJECT
* GENERAL PURPOSE TIMING ROUTINES DEVELOPED IN THE
*   ENGINEERING SUB-DEPARTMENT OF THE COMPUTING APPLICATIONS
*   DEPARTMENT OF THE COMPUTER SCIENCES DIVISION OF THE
*   NUCLEAR DIVISION OF UNION CARBIDE CORPORATION AT THE
*   OAK RIDGE GASEOUS DIFFUSION PLANT ON THE IBM 360/195
*
* 21  ENTRY POINTS ARE PROVIDED. ALL HAVE ONE ARGUMENT WHOSE
*   TYPE AND RETURNED VALUE IS THE SAME AS THE FUNCTIONAL
*   VALUE RETURNED.
```

```
*
* 20 OF THE ROUTINES RETURN THE CPU TIME AS THEIR FUNCTION VALUE
*   AND WILL DESCRIBED TOGETHER. THE NAMES OF THE ROUTINES DETERMINE
*   THE TYPE, LENGTH, UNIT, AND MEASUREMENT INTERVAL BY THE FOLLOWING
*   FORMULA:
*             |I| |S| |TOT|
*             |R| |M| |INT|
*             |D| |H|
*              |I26|
*
*   WHERE: THE FIRST CHARACTER DETERMINES THE TYPE AND LENGTH:
*                I IS INTEGER (NORMAL LENGTH = 4 BYTES)
*                R IS REAL    (SINGLE PRECISION = 4 BYTES)
*                D IS DOUBLE  (DOUBLE PRECISION = 8 BYTES)
*
*          THE SECOND CHARACTER DETERMINES THE UNITS OF RETURNED VALUE:
*                S IS SECONDS
*                M IS MINUTES
*                H IS HUNDRETHS OF SECONDS, I.E., S/100
*
*          THE THREE CHARACTER GROUP RETURNS THE ACTUAL NUMBER
*                OF TIMER UNITS, EACH UNIT=26.01466E-06 SEC
*
*          THE LAST THREE CHARACTERS DETERMINE MEASUREMENT INTERVAL:
*                TOT IS FOR THE INTERVAL CONSISTING OF THE TOTAL TIME
*                       SINCE THE FIRST CALL TO ANY OF THESE CADTIMERS
*                INT IS FOR THE INTERVAL SINCE THE LAST CALL TO AN "INT"
*                       ROUTINE OR SINCE THE FIRST CALL IF NO "INT" HAS
*                       BEEN CALLED.
*
*   THUS ISTOT RETURNS THE TOTAL NUMBER OF SECONDS SINCE THE FIRST
*                CALL TO ANY CADTIMER ROUTINE IN AS AN INTEGER.
*                DUE TO TRUNCATION ALL TIME LESS THAN 1 SECOND IS LOST.
*        RMINT RETURNS THE NUMBER OF MINUTES SINCE THE LAST CALL TO
*                AN INT ROUTINE AS A SINGLE PRECISION REAL NUMBER,
*                NO TIME NECESARILY LOST DUE TO TRUNCATION
*                OR ROUND OFF
*        I26INT RETURNS THE NUMBER OF TIMER UNITS SINCE THE LAST
*                CALL TO AN 'INT' ROUTINE AS AN INTEGER
*
*                FIRST CALL AS AN INTEGER NUMBER. THIS IS EQUILVALENT
*
*                TO THE CSD ROUTINE ICLOCK
*   IDENTITIES:  (ONLY A FEW OF THE MANY THAT EXSIST)
*        IMTOT = ISTOT/60      ISTOT = IHTOT/100 = IFIX(RSTOT)
*        RMINT = SNGLE(DMINT) = RSINT/60.0 = RHINT/60.0/100.0
*        ISTOT = IFIX(FLOAT(I26TOT)*26.04166E-06)
*
         EJECT
*
*
* USAGE: PICK THE DESIRED ROUTINE USING THE FORMULA ABOVE.
*            (FOR EXAMPLE DHTOT)
*        INVOKE IT IN EITHER OF TWO WAYS:
*            AS A FUNCTION - DBLTIM = DHTOT (DT)
*            AS A SUBROUTINE -   CALL DHTOT (DT)
*        RESULT FROM BOTH IS THAT DT IS SET AS THE DOUBLE PRECISION
*            TOTAL TIME AND THE VARIABLE DBLTIM IS THE SAME AS DT
*
*
```

```
* THE TWENTY-FIRST ROUTINE NTIMC RETURNS THE NUMBER OF TIMES ANY OF
*   THE TIME ROUTINES HAVE BEEN CALLED. THIS COUNTER IS INCREMENTED
*   BY ONE WITH EACH CALL TO THE CADTIMER ROUTINES
*
*
* USAGE: ICNT = NTIMC(IC)    OR    CALL NTIMC (IC)
*        RESULTS  IN IC AND ICNT BEING SET TO THIS COUNT
*
*
* THE FIRST CALL TO ANY CADTIMER ROUTINE INITIALIZES THE CLOCK AND
*   RETURNS A ZERO AS A TIME VALUE. GENERALLY THIS WILL ELIMINATE THE
*   NEED FOR A SPECIAL, SET-UP CALL.
*
* REGISTER ASSIGNMENTS:
BASE     EQU   13        NORMAL BASE REG
SAVE     EQU   13        SAVE AREA POINTER
EP       EQU   15        ENTRY POINT ADDRESS
LINK     EQU   14        RETURN ADDRESS
ARGADR   EQU   1         ADDRESS OF ARGUMENT'S ADDRESS
FUNCT    EQU   0         FUNCTIONAL VALUE REG
TEMP     EQU   2         SCRATCH REG
ARG      EQU   12        ADDRESS OF ARGUMENT
TIME     EQU   11        CURRENT TIME
OLDS     EQU   9         OLD SAVEAREA POINTTER
TMP      EQU   8         SCRATCH REGISTER
*
*  MASK FOR TEST UNDER MASK
ON       EQU   X'FF'
*
* STORAGE & CONSTANTS
*
REGS     DS    9D        REG SAVE AREA
LONGTIME DC    1F'1658880042' 12HR MAX CPU TIME ALLOWED
FIRSTIME DC    1F'0'     VALUE OF TIMER AT FIRST CALL
LASTIME  DC    1F'0'     VALUE OF TIMER AT LAST CALL
CALCNT   DC    1F'0'     COUNT OF NUMBER OF TIMES CALLED
* CONVERSION CONSTANTS:
IINTOH   DC    1F'384'            FR INTERVALS TO HUNDRETHS (FIXED)
IINTOS   DC    1F'38400'          FR INTERVALS TO SECONDS   (FIXED)
IINTOM   DC    1F'2304000'        FR INTERVALS TO MINUTES   (FIXED)
RINTOH   DC    1D'26.04166E-04'   FR INTERVALS TO HUNDRETHS (FLOAT)
RINTOS   DC    1D'26.04166E-06'   FR INTERVALS TO SECONDS   (FLOAT)
RINTOM   DC    1D'43.40277E-08'   FR INTERVALS TO MINUTES   (FLOAT)
*
          EJECT
* SUBROUTINE TO LINK THE SAVE AREAS AND GET CURRENT TIME
*   IN REGISTER TIME, AND INTERVAL IN REG FUNCT
*   IN INTERVALS (BOTH FIXED & FLOATING PT)
*   TEMP HAS ADDRESS OF LASTTIME FOR THIS TYPE CALL
*
         USING REGS,SAVE
GETIT    ST    OLDS,4(,SAVE)   LINK
         ST    SAVE,8(,OLDS)          SAVE AREAS
         L     ARG,0(,ARGADR)  GET ADDR. OF ARGS
         TM    CALLED,ON       HAS TIMER BEEN SET?
         BO    SET             YES=BRANCH
         STIMER TASK,BINTVL=LONGTIME
         MVI   CALLED,ON       SET CALLED FLAG
         TTIMER
         ST    FUNCT,FIRSTIME  SET FIRSTIME
         ST    FUNCT,LASTIME   SET TIME OF LAST CALL
         LR    TIME,FUNCT
```

```
         SR    FUNCT,FUNCT       GET ZERO FOR TIME AT FIRST CALL
         B     GOTIME            SKIP TTIMER
SET      TTIMER
         LR    TIME,FUNCT        SAVE CURRENT TIME
         S     FUNCT,0(,TEMP)    GET CORRECT INTERVAL
         LCR     FUNCT,FUNCT
GOTIME   EQU   *                 HAVE INTERVAL IN FUNCT
         ST    FUNCT,FLOAT+4     FLOAT INTERVAL
         L     TMP,CALCNT         INCREMENT CALLED COUNTER
         LD    FUNCT,FLOAT
         LA    TMP,1(,TMP)
         ST    TMP,CALCNT
         BR    14
CALLED   DC    X'00'
         DS    0D
FLOAT    DC    X'4E000000'
         DC    1F'0'
          EJECT
*
* BEGIN EXECUTABLE CODE
*
* THE XHINT ROUTINES
*
         BEGIN IHINT,RHINT,DHINT,LASTIME
         ST    TIME,LASTIME
         MD    FUNCT,RINTOH       CONVERT INTERFVALS TO
         SRDA  FUNCT,32           HUNDRETHS, BOTH FIX & FLO
         D     FUNCT,IINTOH
         LR    FUNCT,FUNCT+1
         B     RETURN
          EJECT
*
* THE XSINT ROUTINES
*
         BEGIN ISINT,RSINT,DSINT,LASTIME
         ST    TIME,LASTIME  SET TIME OF THIS CALL
         SRDA  FUNCT,32      CONVERT HUNDRETHS TO SECONDS
         D     FUNCT,IINTOS      FIXED
         MD    FUNCT,RINTOS      AND FLOATING
         LR    FUNCT,FUNCT+1 DISCARD REMANDER
         B     RETURN
          EJECT
*
* THE XMINT ROUTINES
*
         BEGIN IMINT,RMINT,DMINT,LASTIME
         ST    TIME,LASTIME
         SRDA  FUNCT,32      CONVERT TO MINUTES
         D     FUNCT,IINTOM
         MD    FUNCT,RINTOM
         LR    FUNCT,FUNCT+1
         B     RETURN
          EJECT
*
* THE XHTOT ROUTINES
*
```

```
         BEGIN IHTOT,RHTOT,DHTOT,FIRSTIME
         MD    FUNCT,RINTOH
         SRDA  FUNCT,32
         D     FUNCT,IINTOH
         LR    FUNCT,FUNCT+1
         B     RETURN
          EJECT
*
* THE XSTOT ROUTINES
*
         BEGIN ISTOT,RSTOT,DSTOT,FIRSTIME
         SRDA  FUNCT,32
         D     FUNCT,IINTOS
         MD    FUNCT,RINTOS
         LR    FUNCT,FUNCT+1
         B     RETURN
          EJECT
*
* THE XMTOT ROUTINES
*
         BEGIN IMTOT,RMTOT,DMTOT,FIRSTIME
         SRDA  FUNCT,32
         D     FUNCT,IINTOM
         MD    FUNCT,RINTOM
         LR    FUNCT,FUNCT+1
         B     RETURN
          EJECT
*
* THE NTIMC ROUTINE
*
         USING NTIMC,EP
         ENTRY NTIMC
NTIMC    B     NTIMCGO
         DC    X'06'
         DC    CL6'NTIMC '
NTIMCGO  STM   LINK,LINK-2,12(SAVE)
         LR    OLDS,SAVE
         L     SAVE,ADREGS
         DROP  EP
         USING REGS,SAVE
         MVI   SETARG,X'50'
         ST    OLDS,4(,SAVE)
         ST    SAVE,8(,OLDS)
         L     ARG,0(,ARGADR)
         L     FUNCT,CALCNT
         B     RETURN
         DROP  SAVE
*
          EJECT
*
*
* THE I26XXX ROUTINES:
*
          USING I26INT,EP
          ENTRY I26INT
I26INT    B      I26INTGO
```

```
         DC    X'06'
         DC    CL6'I26INT'
I26INTGO STM   LINK,LINK-2,12(SAVE)
         LR    OLDS,SAVE
         L     SAVE,ADREGS
         USING REGS,SAVE
         DROP  EP
         LA    TEMP,LASTIME
         BAL   LINK,GETIT
         ST    TIME,LASTIME
         MVI      SETARG,X'50'
         B     RETURN
*
*
         DROP  SAVE
         USING I26TOT,EP
         ENTRY I26TOT
I26TOT   B     I26TOTGO
         DC    X'06'
         DC    CL6'I26TOT'
I26TOTGO STM   LINK,LINK-2,12(SAVE)
         LR    OLDS,SAVE
         L     SAVE,ADREGS
         USING REGS,SAVE
         DROP  EP
         LA    TEMP,FIRSTIME
         BAL   LINK,GETIT
         MVI      SETARG,X'50'
         B     RETURN
         DROP  SAVE
*
         EJECT
*
* RETURN REQUIRES BOTH FUNCT REGS TO BE SET
*       AND THAT THE OPCODE FOR THE STORE AT SETARG BE SET FOR THE
*       CORRECT TYPE AND LENGTH FOR THE TYPE OF ARGUMENT FOR
*       THIS CALL.
RETURN   EQU   *
*
*********************WARNING********************
*
* SETARG IS ONLY A SKELETION TO GET THE BASE,
*       INDEX, DISPLACEMENT, AND OPERAND REGISTER
*       CORRECT. THE OPCODE MUST BE SET BEFORE EACH
*       EXECUTION.
*
*************THIS IS SELF MODIFYING CODE**********
*
**************WARNING**********************
*
SETARG   ST    FUNCT,0(ARG)  STORE ARGUMENT-TYPE & LEN FIXED
         ST    FUNCT,20(OLDS) SET INTEGER FUNCTIONAL VALUE
         LR    SAVE,OLDS     RESET SAVE AREA POINTER
         LM    LINK,LINK-2,12(SAVE) RESTORE REGS
         MVI   12(SAVE),ON  SET SUCCESSFUL RETURN FLAG
         BR    LINK
```

```
         SPACE 5                                                CADT0349
ADREGS   DC    A(REGS)                                          CADT0350
         END                                                    CADT0351
```

PART III

CONVERT- FREE-FORM INPUT ROUTINES

PART III

CONVERT - FREE-FORM INPUT ROUTINES

INTRODUCTION

Most FORTRAN programs use fixed format input requiring considerable effort on the part of the user to follow the format requirements. Further, several runs are usually required to resolve input errors. Additionally, several constraints are placed upon both the programmer and the user which can substantially increase the complexity of the input process. The CONVERT free-form input package alleviates the following FORTRAN constraints. A variable number of input items is easily handled with CONVERT, not the user, counting the number of input items. Any number of scalers and arrays of any mix of types is allowed with the arrays loaded in parallel, rather than sequentially. To allow these flexibilities, CONVERT alters the familiar concept of cards by using special characters, rather than card boundaries, to delimit groups of input.

CONVERT is a group of seven entry points which allow a free-form input to IBM FORTRAN programs. Items are separated by commas and may be real, integer, logical, or alphanumeric in nature. CONVERT uses a "logical record" concept whereby a given group of data may span any number of physical input cards. CONVERT was originally used in an earlier form described in [1].

USER CHARACTERISTICS

Each input group, or "logical record," will have as many input values as specified in the documentation for the program of which CONVERT is a part. Each logical record may be composed of any number of input cards, with as many used as needed to input all the required data. (The amount of data within a given block may be variable--check the overall program documentation.) The end of data on each logical record is noted by a semicolon (;) as the last character to be scanned on a card. All characters after the semicolon are treated as comments. A card may have only a semicolon if previous cards in this logical record have defined all the required data, if any. All intermediate cards, that is, all cards in a logical record except the last, have the end of data on each card noted by a colon (:), indicating more data follows on additional cards. All characters after the colon are treated as comments, and a card whose first nonblank character is a colon is a comment only card.

Card boundaries are irrelevant for all processing except as just described, where colons and semicolons are used to demark the end of cards. All the data in one logical record is treated as one continuous stream during conversion. (Indeed the key benefits of the colon-semicolon conventions lie in this variable, unlimited length record format, since all the data does not have to appear on one card.)

This continuous stream is interpreted by either of two basic techniques, alphanumeric and numeric, as specified by the programmer of the program using CONVERT. A logical record may be either alphanumeric or numeric, but never mixed.

Alphanumeric data is any continuous string of characters except colons or semicolons terminated by a semicolon. The logical record may span any number of cards, each ending with a colon. The length used by CONVERT will be that specified by the original programmer and should be specified by the program documentation. If more characters are supplied in the input than the program requested, only the number requested will be transferred; the remainder will be ignored. If too few characters are provided, CONVERT will pad with blank characters as needed.

Numeric data is any mixture of integer, logical, or real variables, separated by commas and ending with a semicolon. Items may be any number of arrays or scalars, but all scalar items must precede the array items on each logical record with the number of items and their types specified in the documentation for the program which uses CONVERT. The valid forms of numeric items are displayed in Figure 1, where they are divided by variable type. CONVERT uses only the syntax of the input, interpreted as in Figure 1, to determine the type of input. The user must follow the documentation for the program which uses CONVERT and ensure that correct types are used in all places. The most common error is the absence of a decimal point in a real whole number specification (i.e., 42 instead of 42.).

The scanning of each field begins at the end of the previous field or the beginning of the logical record and proceeds left to right, ignoring all extraneous characters, including blanks, until a sign,

Type	Valid Forms	Example
Integer	S#,	1, -123, +4562, 421,
Real	S#.,	1., +42., -61.,
	S#.#,	6.41, -1296.82, 69.4221,
	S#ES#,	6E3, 64E-12, 194E+36,
	S.#ES#,	6.E4,-.942E-6, .1E 01
	S#.#ES#,	+84.9420E+10, -89.4E1,80.1E4
Logical	CTC,	T,.True.,All is True Today,
	CFC,	F,False,All is False Today,

Where:

- S is an optional sign, any of "+", "-", or blank. If absent or blank, a "+" is assumed.
- # is any number of digits 0 - 9.
- , is the item separator.
- . is the decimal locator.
- E is the exponent (power of 10) indicator.
- C is an optional string of characters except T or F.
- T is the FORTRAN .TRUE. indicator.
- F is the FORTRAN .FALSE. indicator.

Figure 1

CONVERT Numeric Input Forms

digit, a "T" or an "F" is found. If a "T" or an "F" is found, even after finding a sign, a digit, or an "E", CONVERT stores the appropriate logical value and then skips all characters until a comma or semicolon is found. If a sign or a digit is found, scanning continues until a decimal point, a comma, or an "E" is found. If a comma is found, the sign-digit pair is converted to an integer and stored. If a decimal point is found, a flag indicating that this item is real is set and the scan continues through any digits that follow to the decimal point until a comma or an "E" is found. If an "E" is found, the real flag is set and the exponent is scanned in the following manner. If the character after the "E" is a sign (+, -, blank), it is appropriately noted and the next two characters are interpreted as digits of the exponent, unless the second is the comma ending the field. If the character after the "E" is not a sign, the next two characters are interpreted as the exponent, unless the second is a comma. When a comma is found in the item, it is converted to real, since the real flag was set, and stored. Any characters between the exponent and the comma terminating the item are ignored.

Thus, CONVERT accepts items in standard FORTRAN format except that extraneous characters including blanks are always ignored (except in the exponent field of a real value) and the values are separated by commas rather than fixed columns.

Data in a logical record are always arranged with scalar quantities first, if any, followed by the array items, if any. The number of scalars and arrays and the number of items in the arrays are determined by the program using CONVERT, and its documentation must be consulted to enter the correct number of data items. After all the scalars have been entered, the arrays are entered in parallel (that is, element 1 of array 1, element 1 of array 2, ... , element 1 of array N, element 2 of array 1, element 2 of array 2, ... , element 2 of array N, element 3 of array 1, etc.) until all the data is in. While the number of arrays is constant, the number of items in each array is not necessarily constant. Indeed, this parallel loading of arrays with a variable number of elements is one of the reasons for using CONVERT. Since CONVERT counts the number of items stored and passes the number back to the calling program, the user is generally spared the task of counting the number of items in the input stream.

PROGRAMMING CHARACTERISTICS

The preceding section described the information needed to prepare data for a program which uses CONVERT. This section describes the calling sequences and conventions needed to incorporate CONVERT into a larger program.

CONVERT assumes the existence of a common block named ALFAIN with a length of at least 88 bytes. The first four bytes is a logical variable set to .FALSE. if no end of file in the input data was found or .TRUE. if the end of file was found. The second four bytes are an integer variable with the FORTRAN unit number to be used in reading the input data. The next 80 bytes are the card image currently being converted. A typical statement establishing this common block is

COMMON/ALFAIN/LEOF,IUNIT,RECORD(20)

which must be included in the calling routine.

CONVERT consists of two CSECTs, one Assembler and the other FORTRAN. The FORTRAN CSECT, a subroutine named REREAD, has no arguments and is responsible for actually reading the data into the record field in ALFAIN and calls entry EOFRR upon reaching end of file in the input data. As listed here, it reads from the FORTRAN unit specified in word 2 of ALFAIN and lists each record without change on FORTRAN Unit 6 as it is read. It is a simple routine and may easily be modified to suit particular requirements. The Assembler CSECT is named CONVERT# and has six entry points, one of which must be called only by the REREAD routine.

Entry point CNVRTA has two integer arguments specifying the number of scalars and arrays in subsequent logical record that CONVERT will interpret. Entry point CNVRTT is CNVRTA's complement. It, too, has two integer arguments, but CNVRTT sets them to the current values of the number of scalars and arrays, respectively.

The EOFRR entry sets the end of file flag in ALFAIN to TRUE and returns to the caller of CONVERT, not to the routine which called EOFRR. This nonstandard linkage requires that a REREAD be called only by CONVERT and that only REREAD call EOFRR.

Entry point CNVRTC has two arguments, an array and an integer variable. CNVRTC reads the next logical record from the input and converts it as a character or alphanumeric data, filling the array with the data. The integer specifies the number of characters to be returned with CNVRTC truncating or padding the input as required.

The most-used entry point is CONVERT, which reads the next logical record, converts it as numeric data, and stores it in appropriate scalar and array locations as defined in the last call to CNVRTA. The argument list is of variable length and type also, depending on the last call to CNVRTA. The argument list first has as many scalars as specified by CNVRTA, then the correct number of arrays and, finally, an integer which returns the number of items placed in each array. The types of all the arguments except the last must match the type indicated by the syntax of the input data as CONVERT will return items of the type specified by the syntax, not as specified in the calling sequence. If the user inputs data such that all of the arrays do not get the same number of items, the count will ignore the extra and they will be lost. The USER CHARACTERISTICS segment has substantial information on the way data is input and stored which will not be repeated here.

The entry point LIGNOR is a logical function (all the others are subroutines) which returns a .TRUE. value if the character after the semicolon ending the last logical record was an ampersand (&). LIGNOR must be called only after CONVRT has been called. It can be used to flag the logical record in some way. For instance, if the ampersand is present, the record could be ignored by the calling program or two types of interdispersed data could be distinguished by this ampersand. The use of this routine and warnings about the indiscriminate inclusion of ampersands after semicolons are up to the programmer using this package.

Figure 2 is a list of possible calling sequences for CONVERT and some comments on them. Appendix 1 has a typical input deck.

Thus CONVERT is a usable alternative to the fixed column input of standard FORTRAN,and it provides some options on input that FORTRAN does not easily support such as a variable amount of input. A complete source listing is found in Appendix 2.

REFERENCE

1. Cliff, S. B., "A Method for the Study of Experimental Pulsative Flow Through a Converging-Diverging Tube," unpublished Master's thesis, The University of Tennessee, Knoxville, June, 1976.

CALL CNVRTA (4,7)

Sets CONVRT for four scalars and seven arrays, setting its argument list to 12 variables long.

CALL CNVRTT (NSCAL,NARR)

Sets NSCAL to 4 and NARR to 7 since 4,7 were used in the last call to CNVRTA.

CALL CNVRTC (CHARAC,14)

Reads next logical record and defines the first 14 characters of CHARAC from the input.

CALL EOFRR
CALL REREAD

Both of these calls are illegal by user programs and must not be used.

CALL CONVRT (N1,N2,R1,L1,N3,N4,R2,R3,L2,L3,N5,NEL)

Reads next logical record and stores the first four items in scalars N1, N2, R1, L1 and then loads the remainder in the arrays N3, N4, R2, R3, L2, L3, N5. Finally it sets the integer NEL to the number of elements placed in each of the arrays. The use of four scalars and seven arrays was determined by the last call to CNVRTA. Thus the number of arguments is the sum of the two arguments of CNVRTA plus 1, for NEL. The types of the variables must agree with those on the input data. If a total of 25 items was on the logical record, NEL will have a value of 3 (((25-4)/7)=3;(#Items - #Scalars)/#Arrays)=NEL).

LOGICAL LIGNOR

IF (LIGNOR(0)) GO TO 42

If the character after the semicolon was an ampersand, LIGNOR(0) will be TRUE and execution will pass to statement 42. If the character is not an ampersand, LIGNOR will be FALSE and the branch will not be taken.

Figure 2

Possible Calling Sequences for CONVRT

```
COMMON/ALFAIN/LEOF,IUNIT,RECORD(20)

IF (LEOF) GO TO 94
```

Tests LEOF for end of file on the last call to CNVRTC or CONVRT and goes to 94 only if end of file was found.

NOTES: Both arguments for CNVRTA and the second for CNVRTC are inputs to CONVRT and are not changed. The argument to LIGNOR is ignored. All other arguments are meaningless on entry and are defined by CONVRT.

The input variables must have been previously set.

The dimensions of all arrays in the call to CONVRT must be greater than the value of NEL on return.

The user should check for array overflows.

Figure 2 (Contd.)

PART III, Appendix 1

EXAMPLES OF CONVERT USAGE

Statement Number		Statement
1		IMPLICIT INTEGER (I,N)
2		IMPLICIT REAL ((D,R,T,P)
3		LOGICAL LEOF L1, L2, LNOPRM
4		DIMENSION IA1(30), RA2(30), HEADER(10), TIME (30,20) PRES(30,20), IREFNO(20, ITABLN(20)
5		COMMON /ALFAIN/ LEOF, IUNIT, RECORD(20)
6		EQUIVALENCE (IA1,RA2)
7		IUNIT=5
8		CALL CONVRTC(HEADER,40)
9		CALL CNVRTA(0,1)
10		DEFLT1=42.6
11		DEFLT2=3.1415
12		DEFLT3=32.2
13		IDFLT1=6
14		IDFLT2=12
15		CALL CONVRT(IA1,NINPUT)
16		IF (NINPUT .GT. 30) CALL ABEND(1)
17		IF (LEOF) CALL ABEND(2)
18		IF (NINPUT .LT. 3) CALL ABEND(3)
19		IVAR1=IAI(1)
20		IVAR2=IA1(2)
21		RVAR3=RA2(3)
22		IF (NINPUT .GE. 4) IDFLT1=IA1(4)
23		IF (NINPUT .GE. 5) DEFLT1=RA2(5)
24		IF (NINPUT .GE. 6) DEFLT2=RA2(6)
25		IF (NINPUT .GE. 7) IDFLT2=IA1(7)
26		IF (NINPUT .GE. 8) DEFLT3=RA2(8)
27		IF (NINPUT .GE. 9) WRITE(6,101)
28	101	FORMAT (10X,'TOO MANY ITEMS ON CARD #2, REST IGNORED')

Figure 1-1

Sample Use of CONVERT

Statement Number		Statement
29		IF (IDFLT1 .GT. 20) CALL ABEND (4)
30		CALL CNVRTA (1,2)
31		DO 200 I=1, IDFLT1
32		CALL CONVRT (IREFNO(I), TIME(1,I), PRES(1,I), ITABLN(I))
33		IF (LEOF) CALL ABEND(5)
34		IF (ITABLN(I) .GT. 30) CALL ABEND(6)
35	200	CONTINUE
36		LNOPRM= .FALSE.
37		CALL CNVRTA(4,1)
38		CALL CONVRT (L1,R1,R2,I3,IA1,NINPUT)
39		IF (NINPUT .GT. 30) CALL ABEND (7)
40		IF (LEOF) LNOPRM= .TRUE.
		.
		.
		.

Figure 1-1 (Contd.)

Card Number	Card
1	THIS IS A SAMPLE DECK FOR CONVERT: HEADER
2	14, 16, 62.443,: ALWAYS SPECIFY 3 VALUES
3	3, : THREE TIME VS. PRESSURE TABLES
3	647, 1.4E-03, : IDFLT1, DEFLT1
5	; END OF CONTROL RECORD
6	: BEGIN TIME VS. PRESSURE TABLES
7	1 : TABLE #1
8	0.0, 10.0, 1.0,20.2, 2.0, 31.0, : BEGIN SPIKE
9	3.0, 28.0, 3.5, 25.2, 3.75, 26.8 : NOTE HUMP
10	4.0, 22.0, 5.0, 15.0, 6.0, 12.5 : END OF SPIKE
11	7.0, 10.0, 1E10, 10.0 ; STEADY AT 10
12	3 : TABLE #3
13	0.0, 1E3, 3.0, 7.0E2, : SLOW FALL FOR 3 SEC.
14	3.1, 6.32E2, 3.2, 6.04E2, 3.3, 5.5E2, 3.4, 5E2, 3.5, 4.00 : RAPID DEPRESSURIZATION
15	3.6, 4.00, : SHORT STABILITY
16	3.8, 5E2, 4.0, 6E2, 5.0, 7E2 : SLOW PRESSURIZATION TO STEADY
17	: THE FOLLOWING CARDS DESCRIBE THE LIQUID SLUG AT 6.0 SEC
18	: THEY HAVE BEEN DELETED FOR THIS TEST
19	: 5.5, 8E2, 6.0, 8.62E2 : UP TO PEAK PRESSURE
20	: 6.3, 7.84E2, 6.7, 6.9E2, 7.25, 7.E2 : BACK TO STEADY
21	1E10, 7E2; STEADY STATE AT 700
22	2 : TABLE #2
23	0.0, 100.0, 10.0, 200.0, 20.0, 100.0, 1E10, 1E2 ;

Figure 1-2

Sample Data Deck

Assume that the executable statements of Figure 1-1 are in the program using CONVERT and that Figure 1-2 is the data deck to be read. This example defines a header array with statements 4 and 8 which read card 1. Note the use of comments after the data on this record and throughout the input deck. Then a control record, cards 2 to 5, is read into a scratch array area by statements 9 and 15. Statement 16 terminates the program if the scratch area has been exceeded, while statement 17 terminates upon early end of file. Statement 18 ensures that the three required quantities are present and statements 19, 20, and 21 move them from the scratch area into usage area. Statements 10 to 14 set defaults for the optional variables on the control record.

Then, using the number of items placed in the scratch area as a key, statements 22 to 26 change the default values if the user has input them. In this case, six items were specified on data cards 2 through 5, changing three of the optional values. The spreading over multiple cards allows the data to be changed easily; here data cards 3 and 4 could be removed returning them to their default values without generating invalid syntax. Statements 27 and 28 issue a warning message if too many items were in the control record. Statement 29 then ensures that array boundaries will not be exceeded in the next loop. Data card 6 is strictly a comment which is ignored as the next logical record, cards 6 to 11, is read in the first execution of the loop in statements 31 to 35. Statement 30 sets CONVERT for one scaler (a reference number for each table) and two arrays (a time versus pressure curves). Statement 32 invokes CONVRT with the appropriate arrays. Note that the length of each table (ITABLN) is automatically set by CONVRT. Statements 33 and 34 ensure that sufficient data is supplied, but that the table arrays are not exceeded. Table #1 is spread over cards 7 to 11 and specifies a table with 11 pairs of values. The table stored next has a reference number of 3 and is specified in data cards 12 to 21. The table was originally 17 pairs of values long, but the pressure hump due to the slug of liquid noted in cards 16 to 20 has been commented out and will be ignored. Thus this table is nine pairs of points long. The third table has a reference number of 2 and is cards 22 and 23. Only four pairs of values comprise this table. Next, an optional parameter table is assumed to be present in statement 36. Statement 37 defines the next record to have four scalers and one array, and statement 38 attempts to read it. Statement 39 ensures the integrity of storage areas, and statement 40 checks to see if any data was present. Since all the data had already been read, the flag of no parameter is set to TRUE indicating it was not input.

Thus this example illustrates several of CONVERT's features and some of the coding techniques that can be used with it. Records 16, 17, 18, 27, 28, 29, 33, 34, and 39 are needed only to check the data, providing more user security in input preparation than is normally found in most programs.

PART III, Appendix 2

SOURCE LISTING OF THE CONVERT ROUTINES

```
         MACRO
         LINK
         LA    2,SVE001
         ST    13,4(0,2)            SAVE
         ST    2,8(0,13)              AREA
         LR    13,2                      LINKAGES
         MEND
CONVERT# CSECT 0             BEGIN ASSEMBLE
         ENTRY CNVRTT
*CALL CNVRT (NOSCAL,NOARRAY)
*CONVERT TEST E.P. - TO GET CURRENT VALUES OF CONSTANTS
*NOSCAL IS NUMBER OF SCALERS - A POSITIVE INTEGER
*NOARRY IS NUMBER OF ARRAYS -  A POSITIVE INTEGER
         USING CNVRTT,15
CNVRTT   SAVE  (14,12),,*
         LINK
         LM    2,3,0(1)
         LA    9,NOSCAL
         MVC   0(4,2),0(9)
         MVC   0(4,3),4(9)
         L     13,4(0,13)
         RETURN  (14,12),T
         ENTRY CNVRTA
*CALL CNVRTA(---SAME ARGUMENTS AS CNVRTT---)
*CONVERT ALTER E.P. - USED TO CHANGE THE CONSTANTS OF THIS PROG
         USING CNVRTA,15
CNVRTA   SAVE  (14,12),,*
         LINK
         LM    2,3,0(1)
         LA    9,NOSCAL
         MVC   0(4,9),0(2)
         MVC   4(4,9),0(3)
         L     13,4(0,13)
         RETURN  (14,12),T
         ENTRY EOFRR
*IF END OF FILE IS FOUND IN REREAD, EOFRR IS CALLED AND IT RETURNS
*    CONTROL TO THE  PROGRAM THAT CALLED CONVRT AND SETS EOF FLAG
         USING *,15
EOFRR    L      13,SVE001+4
         L      7,ALFAINAD
         LA     5,1(0,0)
         ST     5,0(0,7)
         WTL  'END OF FILE IN REREAD'
         ABEND   65
         RETURN (14,12),T
         ENTRY CONVRT           DECLARE ENTRY POINT TO BE AT CONVRT
*THIS ROUTINE CONVERTS THE ALPHAMERIC INPUT FROM THE TELETYPE TO
*    APPROPRIATE NUMERIC FORM
*USAGE- CALL CONVRT (RECORD,A,B,C,D,....G,S,T,....Z)
*   WHERE RECORD IS A CHARACTOR STRING TO BE DECODED, OF ANY LENGTH,
*             OF THE FORM$,$,$,$...$,C  WHERE $ IS A REAL OR INTEGER
*             NUMBER IN CHAR FORM, SEPARATED BY COMMAS, AND C IS
*             EITHER A COLON OR SEMICOLON IT IS IN "ALFAIN"
*ANY NUMBER OF SCALORS AND/OR ARRAYS ARE PERMITTED AS SET BY CNVRTA
*A CARD OF FORM : COMMON /ALFAIN/ LEOF,INR,RECORD     MUST APPERA IN
*A CARD OF FORM : COMMON /ALFAIN/ LEOF,INR,RECORD     MUST APPEAR IN
```

```
*BOTH THE CALLING AND REREAD PROGRAMS.  RECORD IS AS DESCRIBED ABOVE     CNV10056
*LEOF IS LOGICAL END OF FILE IN REREAD--SET TO .TRUE. VALUE BY THIS      CNV10057
*PROGRAM IF END OF FILE IS FOUND; IT IS SETTO .FALSE. IF NO EOF FOUND    CNV10058
* INR IS NOT TOUCHED BY THIS PROGRAM BUT CAN BE USED TO TRANSMIT         CNV10059
*  INPUT UNIT NUMBER TO REREAD                                           CNV10060
*IF INPUT IS REAL, REGUALR REAL VALUES ARE RETURNED                      CNV10061
*AN INTEGER IS ANY CONSISTING ONLY OF NUMBERS                            CNV10062
*A REAL INPUT IS ANY THAT HAS EITHER A DECIMAL OF AN E TO DENOTE A       CNV10063
*       POWER OF TEN                                                     CNV10064
*ALL BLANKS AND ILLEGAL CHAR REGUARDLESS OF LOCATION ARE IGNORED         CNV10065
*       EXCEPT AS NOTED                                                  CNV10066
*       A,B,C,....F,G, ARE VARIABLES OF LENGTH FOUR BYTES, EITHER        CNV10067
*              REAL OR INTEGER, THAT CORRESPOND TO EACH $ IN RECORD      CNV10068
*       S,T...., ARE LINEAR ARRAYS OF ANY LENGTH , WITH EACH ELEMENT     CNV10069
*              OF LENGTH FOUR BYTES TO RECIEVE DATA VALUES IN  PARRELL   CNV10070
*NOTE LOGICAL VALUES CAN BE RETURNED TO ANY ARGUMENT IF THAT IS TYPE     CNV10071
***  NOTE  *** THE USER MUST KEEP TYPES CORRECT; THIS ROUTINE            CNV10072
*       CHECKS THE SYNTAX IF THE INPUT RECORD TO DETERMINE TYPE          CNV10073
*       Z IS THE NUMBER OF VALUES PLACED IN EACH ARRAY ON RETURN TO      CNV10074
*              CALLING PROGRAM                                           CNV10075
*IF INPUT IS INTEGER, REGULAR INTEGER VALUES ARE RETURNED                CNV10076
*A COMMA "," DELIMITS EACH VALUE OR WORD                                 CNV10077
*A PERIOD "." DENOTES DECIMAL LOCATION, IF NEEDED                        CNV10078
*ALL BLANKS, REGUARDLESS OF LOCATION, ARE IGNORED                        CNV10079
*A COLON ":" DENOTES END OF A RECORD THAT IS CONTINUED - IT MUST BE      CNV10080
*    PRECEEDED BY A COMMA                                                CNV10081
*A SEMICOLON ";" DENOTES THE END OF A RECORD THAT IS NOT CONTINUED, IT   CNV10082
*    MUST BE PRECEEDED BY A COMMA                                        CNV10083
*IF INPUT IS CHARACTOR "T", A FORTRAN LOGICAL .TRUE. VALUE IS RETURNED   CNV10084
*IF INPUT IS CHARACTOR "F", A FORTRAN LOGIACL .FALSE. IS RETURNED        CNV10085
*AN E DENOTES A REAL VALUE WITH THE POWER OF TEN FOLLOWING TH E          CNV10086
*ONCE A T OR F IS ENCOUNTERED, ALL CHAR ARE SKIPPED UNTIL A COMMA        CNV10087
*ONCE AN E IS FOUND, THE OPTIONAL SIGN CHAR IS CHECKED FOR, THEN AT      CNV10088
*   MOST TWO DIGITS OF EXPONENT, THEN ALL CHAR ARE SKIPPED UNTIL A       CNV10089
*   COMMA IS FOUND.THIS IS THE ONLY REGION THAT EXTRANOUS CHAR           CNV10090
*   (EXCEPT BLANKS) ARE NOT IGNORED                                      CNV10091
*INPUT FORMS:                                                            CNV10092
*       INTEGER: S#,        1,-123,+4521,1245,                           CNV10093
*       REAL: S#.,          2.,-541.,+5874.,                             CNV10094
*              S#.#,    .   3.2,12.456,+257.14,-12.006,                  CNV10095
*              S#ES#,       7E5,+1452E-12,-4E 5,                         CNV10096
*              S.#ES#,      +.20345E28-.00234E-42,                       CNV10097
*              S#.#ES#,     +12.542E-16,-142.563E3,8.452E-458            CNV10098
*              EACH S IS AN OPTIOMAL SIGN CHAR, +OR-, + IF OMMITED       CNV10099
*              # IS ONE OR MORE DIGITS OF SET 0-9                        CNV10100
*              . IS THE DECIMAL LOCATOR                                  CNV10101
*              E IS THE EXPONENT INDICARATOR                             CNV10102
*              , IS THE REQUIRED SEPARATOR BETWEEN VALUES                CNV10103
*REGISTER ASSIGNMENTS:                                                   CNV10104
*   GENERAL    0,1       USED AS WORK REGISTER PAIR                      CNV10105
*              2         COUNTS THE ARGUMENTS                            CNV10106
*              3         COUNTS THE ARRAYS                               CNV10107
*              4-5       PAIR, THE NUMBER IS ASSEMBLED HERE              CNV10108
*              6         COUNTS NO OF DIGITS AFTER A . OR E; IF          CNV10109
*                             NEGATIVE, THE NUMBER IS AN INTEGER         CNV10110
*              7         FOLLOWS DOWN THE INPUT RECORD POINTING TO THE   CNV10111
```

```
*                             CHAR TO BE DECODED
*              8         THE EXPONENT IS ASSEMBLED HERE, IF ZERO-
*                        THERE IS NO EXPONENT
*              9         INCREMENTS DOWN THE LIST OF ADDRESSES TO
*                             INDICATE WHICH ARGUMENT IS NEXT
*              10        FIXED POINT CONSTANT 10
*              11        INDEXES THROUGH ARRAYS
*              12        FIXED POINT CONSTANT 4 KEPT HERE
*              13        BASE REGISTER,  SAVE AREA ADDRESS
*              14-15     SUBROUTINE LINKAGE
*    FLOATING POINT:
*              0         FLOATING POINT (REAL) NUMBER ASSEMBLED AND
*                             CONVERTED HERE
*              2              FLOATING POINT CONSTANT 10.0
*              4         FLOATING POINT ZERO
*              6         NOT USED
*
         USING CONVRT,15
CONVRT   SAVE  (14,12),,*     SAVE THE REGISTERS
         LINK
         B     SVE001+72
         USING   SVE001,13
         DROP  15             DROP 15 AS BASE REG
SVE001   DS    18F
         SR    11,11          SET INDEX REGISTER TO ZERO
         LR    2,11           SET ARGUMENT COUNTER TO INITIAL VALUE
          LR   3,11           SET ARRAY COUNTER TO ZERO
         LA    12,4           GET CONSTANT "FOUR"
         LA    10,10          GET CONSTANT "TEN"
         SDR   4,4            GET FLOATING POINT ZERO CONST
         LD    2,D10          GET FLOATING POINT 10.0
         LR    9,1            PUT ADR OF ARG IN REG 9
         L     1,NOARRY
         A     1,NOSCAL
         MR    0,12
         L     1,0(1,9)
         ST    1,ADTNPT                   OF LAST ARGUMENT
         L     7,ALFAINAD     SET REG7 TO BEGINNING OF INPUT RECORD
         LA    7,8(0,7)
*BEGIN LOOP TO DECODE A WORD
WORDLOOP SR    5,5            CLEAR REGS 5 & 6
         SR    8,8
         STH   5,FLAG         SET FLAGS TO ZERO
         L     6,CM1000       MAKE REG6 VERY NEGATIVE AS A FLAG
         B     CHARLOOP+4     SKIP THE INCREMENTING OF REG1
*BEGIN LOOP TO DECODE A CHARACTOR
CHARLOOP LA    7,1(0,7)       ADD 1 TO REG7-IT POINTS TO THE CHAR TO
*                                         BE DECODED
         CLI   0(7),X'40'     IS THIS CHAR A BLANK?
         BE    CHARLOOP       BRANCH BACK IF IT IS  - I.E. SKIP IT
         CLI   0(7),X'6B'     IS IT A COMMA?
         BE    COMMA          BR TO DECODE A COMMA IF IT IS
         CLI   0(7),X'4E'     IS THIS CHAR A '+'?
         BE    CHARLOOP       BR IF IT IS - I.E. SKIP IT
         CLI   0(7),X'60'     IS THIS CHAR A '-'?
         BNE   *+12           SKIP FLAG SETTING IF IT IS
```

```
         MVI   FLAG,X'FF'      SET FLAG BITS TO ONE                          CNV10168
         B     CHARLOOP        RETURN FOR NEXT CHAR                          CNV10169
         CLI   0(7),X'4B'      IS IT A DECIMAL?                              CNV10170
         BNE   *+10            BR IF NOT A DECIMAL                           CNV10171
         SR    6,6             SET 6 TO ZERO AS FLAG                         CNV10172
         B     CHARLOOP        RETURN FOR NEST CHAR                          CNV10173
         SR    1,1             CLEAR REG1 TO RECIEVE THIS CHAR               CNV10174
         IC    1,0(0,7)        GET THIS CHARACTOR                            CNV10175
         S     1,F0            TRY TO CONVERT IT TO A NUMBER                 CNV10176
         BM    LOGICAL         BR TO LOGICAL VARIABLE IF CHAR IS NOT A #     CNV10177
         A     6,C1            INCREMENT DIGIT COUNTER BY OMR                CNV10178
         MR    4,10            SHIFT PREVIOUS DIGITS IN THIS WORD BY 10      CNV10179
         AR    5,1             ADD LOW ORDER DIGIT TO HIGH ORDER ONES        CNV10180
         B     CHARLOOP        GO TO NEXT CHAR                               CNV10181
LOGICAL  CLI   0(7),X'E3'      IS IT A "T"?                                  CNV10182
         BE    TRUE            BR IF IT IS                                   CNV10183
         CLI   0(7),X'C5'      IS THIS CHAR AN "E"?                          CNV10184
         BE    REAL#           BR TO TAKE CARE OF EXP                        CNV10185
         CLI   0(7),X'C6'      IS THIS CHAR A F?                             CNV10186
         BE    FALSE           BR IF IT IS                                   CNV10187
         B     CHARLOOP        SKIP UNKNOWN CHAR                             CNV10188
TRUE     LA    5,1(0,0)        GET TRUE VALUE                                CNV10189
         B     *+6             SKIP NEXT INSTRUCTION                         CNV10190
FALSE    SR    5,5             GET FORTRAN .FALSE. VALUE                     CNV10191
         LA    7,1(0,7)        SKIP TO NEXT CHAR                             CNV10192
         CLI   0(7),X'6B'      IS IT  A COMMA?                               CNV10193
         BNE   FALSE+2         SKIP CHAR                                     CNV10194
         B     READY           BR TO STORE WORD IF CHAR IS COMMA             CNV10195
REAL#    LA    7,1(0,7)        SKIP THE "E"                                  CNV10196
         CLI   0(7),X'40'      IS THIS CHAR A BLANK?                         CNV10197
         BE    REAL#           BR IF IT IS A BLANK                           CNV10198
         LTR   6,6             CHECK- HAS A DECIMAL BEEN FOUND?              CNV10199
         BNM   *+6             BR IF ONE HAS                                 CNV10200
         SR    6,6             FORCE DECIMAL RECORDING IF ONE HAS NOT        CNV10201
         CLI   0(7),X'4E'      IS THE EXP POS?                               CNV10202
         BE    *+16            SKIP NGS SIGN PROCESSING  IF IT IS            CNV10203
         CLI   0(7),X'60'      IS THE EXP NEGATIVE?                          CNV10204
         BNE   *+20            SKIP SIGN BIT SETTING IF NO SIGN GIVEN        CNV10205
         MVI   FLAG+1,X'FF'    SET NEGATIVE EXP BITS TO ONES                 CNV10206
         LA    7,1(0,7)        SKIP SIGN CHAR                                CNV10207
         CLI   0(7),X'40'      IS THIS CHAR A BLANK                          CNV10208
         BE    *-8             SKIP IF IT IS                                 CNV10209
         IC    8,0(0,7)        GET THIS CHAR                                 CNV10210
         S     8,F0            CHANGE TO NUMERIC FORM                        CNV10211
         BM    COMMALOK        GO LOOK FOR COMMA                             CNV10212
         LA    7,1(0,7)        SKIP FIRST CHAR                               CNV10213
         CLI   0(7),X'40'      IS THIS CHAR A BLANK?                         CNV10214
         BE    *-8             SKIP IT IF IT IS                              CNV10215
         SR    1,1             CLEAR REG1 TO RECIEVE THIS CHAR               CNV10216
         IC    1,0(0,7)        GET NEXT CHAR                                 CNV10217
         S     1,F0            CONVERT TO NUMERIC FORM                       CNV10218
         BM    COMMALOK        GO LOOK FOR COMMA                             CNV10219
         ST    1,WORK          SAVE SECOND DIGIT                             CNV10220
         LR    1,8             PUT FIRST DIGIT IN REG1                       CNV10221
         MR    0,10            SHIFT FIRST DIGIT OVER                        CNV10222
         A     1,WORK          ADD SECOND DIGIT                              CNV10223
```

```
         LR    8,1           PUT TOTAL EXP IN 8                         CNV10224
         B     COMMALOK      GO LOOK FOR COMMA                          CNV10225
         LA    7,1(0,7)      SKIP THIS CHAR                             CNV10226
COMMALOK CLI   0(7),X'6B'    IS THIS CHAR A COMMA?                      CNV10227
         BNE   COMMALOK-4    GO LOOK FOR COMMA                          CNV10228
COMMA    LTR   6,6           IS THIS WORD AN INTEGER?                   CNV10229
         BM    INTEGER       BR IF IT IS                                CNV10230
         MVC   WORK,FLOATC   GET CONSTANT FOR FLOATING REAL NO.         CNV10231
         ST    5,WORK+4      ST THE INTEGER                             CNV10232
         LD    0,WORK        LOAD FLOATED, UNNORMALIZED NO TO FPRO      CNV10233
         ADR   0,4           ADD '0.0' TO NORMALIZE NO.                 CNV10234
         TM    FLAG,X'FF'    CHECK FOR SIGN OF INTEGER                  CNV10235
         BNO   *+6           BR IF POSITIVE                             CNV10236
         LNDR  0,0           COMPLEMENT FPRO - MAKE IT NEGATIVE         CNV10237
         LTR   6,6           CHECK NO OF DIGITS IN FRACTIONAL PART      CNV10238
         BNP   *+10          BR IF NO DIGI&O  F&5I-4E&&D-&              CNV10239
         DDR   0,2           DIVIDE BY '10.0'                           CNV10240
         BCT   6,*-2         BR UNTIL EXP IS EXHAUSTED                  CNV10241
         LTR   8,8           CHECK FOR EXPONENT                         CNV10242
         BNP   FLOATED       BR IF EXP IS NOT POS-NO OR ZERO EXP        CNV10243
         TM    FLAG+1,X'FF'  CHECK EXP SIGN                             CNV10244
         BO    *+14          BR IF EXP SIGN IS NEG                      CNV10245
         MDR   0,2           MULTIPLY BY 10.0                           CNV10246
         BCT   8,*-2         BR UNTIL EXP IS EXHAUSTED                  CNV10247
         B     FLOATED       SKIP NEG EXP CALC                          CNV10248
         DDR   0,2           DIVIDE BY 10.0                             CNV10249
         BCT   8,*-2         BR UNTIL EXPONENT IS EXHAUSTED             CNV10250
FLOATED  STD   0,WORK        STORE FLOATED WORD                         CNV10251
         L     5,WORK        GET WORD FOR STORAGE IN CALLING PROG       CNV10252
         B     READY         SKIP SIGN CHECK                            CNV10253
INTEGER  TM    FLAG,X'FF'    CHECK SIGN OF INTEGER                      CNV10254
         BNO   READY         SKIP IF POS                                CNV10255
         LNR   5,5           MAKE NEG IF NEEDED                         CNV10256
READY    LA    2,1(0,2)      INCREMENT ARG COUNTER                      CNV10257
         C     2,NOSCAL      CHECK FOR THE ARRAYS AS                    CNV10258
         BP    ARRAYS                  ARGUMENTS AND BR IF NECESSARY    CNV10259
         L     1,0(0,9)      PUT ADR OF ARGUMENT IN 1                   CNV10260
         ST    5,0(0,1)      STORE THIS WORD IN CORRECT ARGUMENT        CNV10261
         AR    9,12          SKIP TO NEXT ARGUMENT                      CNV10262
         LA    7,1(0,7)      SKIP COMMA                                 CNV10263
LITLELOP CLI   0(7),X'7A'    IS THE NEXT CHAR A COLON?                  CNV10264
         BE    COLON         BR IF IT IS                                CNV10265
         CLI   0(7),X'5E'    IS IT A SEMICOLON?                         CNV10266
         BE    SEMICOLN      BR TO DECODE A SEMICOLON IF IT IS          CNV10267
         CLI   0(7),X'40'    IS IT A BLANK?                             CNV10268
         BNE   WORDLOOP      IF NOT BLANK, RETURN TO DO NEXT WORD       CNV10269
         B     LITLELOP-4    CHECK NEXT CHAR                            CNV10270
*THERE ARE 4 ARRAYS THAT RECIEVE VALUES AND THEY RECIVE VALUES IN       CNV10271
*     PARALLEL. HENCE COUNTERS MUST BE SET AND CHECKED TO PUT VALUES    CNV10272
*     IN CORRECTLY                                                      CNV10273
ARRAYS   L     1,0(0,9)      PUT ADR OF ARGUMENT IN 1                   CNV10274
         ST    5,0(11,1)     STORE WORD IN CORRECT ARRAY WITH PROPER    CNV10275
*                                      INDEX VALUES                     CNV10276
         AR    9,12          SKIP TO NEXT ARGUMENT                      CNV10277
         LA    3,1(0,3)      INCREMENT ARRAY COUNTER                    CNV10278
         C     3,NOARRY      CHECK-IF LAST   ARRAY- DO NOT BRANCH       CNV10279
```

```
         BNE   LITLELOP-4            OTHERWISE BR TO CHECK FOR COLONS  CNV10280
*IF AT FOURTH ARRAY, MUST RESET COUNTERS                                CNV10281
         AR    11,12          INCREMENT ARRAY INDEX TO NEXT WORD        CNV10282
         LR    1,3            GET NO OF ARRAYS                          CNV10283
         MR    0,12                                                     CNV10284
         SR    9,1            BACK ARGUMENT POINTER TO FIRST ARRAY      CNV10285
         SR    3,3            RESET ARRAY COUNTER                       CNV10286
         B     LITLELOP-4     CHECK FOR COLONS                          CNV10287
*DECODE COLON BY READING IN THE NEXT RECORD AND RESETING REG7 TO        CNV10288
*     THE BEGINNING OF THE NEW RECORD                                   CNV10289
*REREAD HAS NO ARGUMENTS AND IS CALLED FROM THIS POINT                  CNV10290
COLON    L     15,RERADD      REG15 IS ADR OF REREAD ROUTINE            CNV10291
         BALR  14,15          CALL REREAD ROUTINE                       CNV10292
         L     7,ALFAINAD     SET REG7 TO BEGINNING OF INPUT RECORD     CNV10293
         LA    7,8(0,7)                                                 CNV10294
         SDR   4,4            GET FLOATING POINT ZERO CONST             CNV10295
         LD    2,D10          GET FLOATING POINT 10.0                   CNV10296
         B     LITLELOP       GO TO NEXT WORD LOOP VIA COLON CHECK      CNV10297
*DECODE SEMICOLON BY DETERIMINING  THE TOTAL NO. OF POINTS AND          CNV10298
*     STOREING ITAND RETURING TO THE CALLING PROGRAM                    CNV10299
SEMICOLN LTR   11,11          HAS THE ARRAYS BEEN REACHED?              CNV10300
         BNP   RETURNS        BR IF NO ARRAYS USED                      CNV10301
         M     10,C1          PREPARE ARRAY INDEX REG FOR DIVISION      CNV10302
         DR    10,12          REG11 HAS TOTAL NO. OF POINTS             CNV10303
         L     1,ADTNPT       REG 1 HAS ADR OF LAST ARGUMENT            CNV10304
         ST    11,0(0,1)      STORE TOTAL NO. OF POINTS                 CNV10305
RETURNS  SR    5,5            SET EOF VALUE TO .FALSE.-                 CNV10306
         ST    7,SEMIADD     SAVE ADDRESS OF END OF RECORD              CNV10307
         L     7,ALFAINAD                                               CNV10308
         ST    5,0(0,7)                                                 CNV10309
         L     13,4(0,13)     UNLINK SAVE AREAS                         CNV10310
         RETURN (14,12),T     RETURN TO CALLING PROGRAM                 CNV10311
         ENTRY  LIGNOR                                                  CNV10312
*LOGICAL FUNCTION TO DETERMINE IF THIS RECORD SHOULD BE IGNORED         CNV10313
*LRSULT = LIGNOR (RECOED)                                               CNV10314
*LRSULT = .RRUE. IF & FOLLOWS ;                                         CNV10315
*LRSULT = .FALSE. IF NO & FOLLOWS ;                                     CNV10316
         USING  LIGNOR,15                                               CNV10317
LIGNOR   SR    0,0                                                      CNV10318
         L     1,SEMIADD                                                CNV10319
         CLI    1(1),X'50'                                              CNV10320
         BE    AMPER                                                    CNV10321
         BR    14                                                       CNV10322
AMPER    L     0,C1                                                     CNV10323
         BR    14                                                       CNV10324
FLAG     DS    1H             FLAGS FOR SIGN BITS AS NEEDED             CNV10325
CM1000   DC    1F'-1000'      LARGE NEGATIVE NUMBER FOR FLAG IN REG 6   CNV10326
D10      DC    1D'10.0'       FLOATING POINT TEN                        CNV10327
WORK     DS    1D             A DOUBLE WORD OF WORKING SPACE IN CORE    CNV10328
FLOATC   DC    X'4E00000000000000'     FLOATING CONV CONSTANT           CNV10329
C1       DC    1F'1'          CONSTANT  1                               CNV10330
FO       DC    X'000000F0'    CONVERSION CONSTANT-ALPHAMERIC TO NUMERIC CNV10331
ADTNPT   DS    1F             ADR OF LAST ARGUMENT                      CNV10332
ALFAINAD DC    V(ALFAIN)                                                CNV10333
RERADD   DC    V(REREAD)      ADR OF REREAD ROUTINE                     CNV10334
NOSCAL   DC    1F'7'                                                    CNV10335
```

```
NOARRY   DC     1F'4'
SEMIADD  DS     1F
         END
      SUBROUTINE REREAD
      LOGICAL LEOF
      COMMON /ALFAIN/ LEOF,INR,RECORD(20)
      DATA IC/0/
      READ(INR,1,END=10) RECORD
 1    FORMAT(20A4)
      IC=IC+1
      WRITE (6,2) IC,RECORD
    2 FORMAT(15X,I6,'=',T25,20A4)
      LEOF = .FALSE.
      RETURN
 10   LEOF=.TRUE.
      CALL EOFERR
      RETURN
      END
```

PART IV

PARMETER - PARAMETER FIELD ACCESSING ROUTINES

PART IV

PARMETER - PARAMETER FIELD ACCESSING ROUTINES

Most FORTRAN programs are controlled from data read through the FORTRAN library from various unit numbers. This control suffices for most programs, but there are occasions where control from another source is desired. The parameter field of the EXEC Job Control Language Statement [1]

```
//STEPNAME EXEC FORTHCLG,PARM.GO='PARAMETER FIELD'
```

provides such an extra input. A good example of the use of this field is the FORTRAN-H compiler itself. The compiler uses the SYSIN file as source input, which has no control specifications built in. All compiler options are provided through the parameter field of the FORT step, e.g., (PARM.FORT='XREF,MAP'). Although the compiler is written primarily in standard FORTRAN, it has assembler language code similar to the two routines discussed below to access the parameter field.

The first routine, ALPARM, is an INTEGER FUNCTION which has one argument:

```
INTEGER ALPARM
ILEN = ALPARM (PARM)
```

The functional value (ILEN) is the number of characters in the PARM field, with zero returned if no field was specified. If ILEN is positive, the characters from the PARM field are copied (A4 FORMAT) into the argument, PARM, which must be dimensioned to accept the entire PARM field which may be 100 characters (25 words) long. Only the first ILEN characters are transferred; the rest remain as before the invocation of ALPARM.

The second routine, PARM, is also an INTEGER FUNCTION with one argument:

```
INTEGER*2 PARM, I2
I = PARM (I2).
```

This routine provides for a typical use of the PARM field since it scans the total field looking for the value of the keyword "MODEL" which is assigned a value by a field such as "MODEL=2D". Thus, the functional value PARM and the argument I2 are both returned with the value of MODEL, here "2D", in an INTEGER*2 format. The value returned consists of the two characters (not binary numbers) of MODEL. Thus, MODEL can have the value of any two characters available for use in the PARM field, whether numeric or not. If no PARM field was specified or the string "MODEL=" was not found, or both characters were not specified, a binary zero is returned. Because of FORTRAN conventions, PARM may also be invoked by

```
CALL PARM (I2)
```

where the functional value is ignored. The phrase "MODEL=" may occur anywhere in the PARM field, intermixed with other characters as desired.

Both of these routines may be invoked repeatedly in the same program without harm, and they may be invoked in an intermixed fashion if desired.

Appendix 1 is a complete source listing.

REFERENCE

1. Job Control Language Reference, IBM Manual GC28-6704-3, pp. 89-90.

PART IV, Appendix 1

SOURCE LISTING OF PARMETER

```
PARMAMTR CSECT  0
* VERSION 1.0 JUNE 20,1977 STEVEN B. CLIFF
*
* THESE ROUTINES ACCESS THE PARM FIELD OF THE EXEC STATMENT
*
* USAGE FOR PARM:
*        I=PARM(I2)
*
* WHERE I2 RECIVES THE FIRST TWO CHARACTERS AFTER THE STRING
*                "MODEL=".  IF NO PARM FIELD OR NO "MODEL="
*                A NUMERIC ZERO IS RETURNED. I2 IS TYPED
*                INTEGER *2
*        I  RECIEVES THE SAME AS I2 EXCEPT, BECAUSE OF
*                STANDARD FORTRAN LINKAGES, IT MAY BE TYPED AS
*                I*2,I*4,L*4 OR L*1 WITH APPROPRIATE RESULTS.
*                IF TYPED AS L*1 ONLY ONE CHARACTER (THE SECOND)
*                WILL BE AVAILABLE, OTHERWISE THE TYPES ARE THE SAME
*
*
*
* USAGE FOR ALPARM:
*        J=ALPARM(IA)
* WHERE IA  IS AN ARRAY WHICH RECIVES ALL OF THE PARM FIELD
*                THAT IS PRESENT. SINCE THE FIELD MAY BE
*                UPTO 100 CHARACTERS LONG, IT SHOULD BE DIMENSIONED
*                TO AT LEAST 100 CHARCTERS, 25 WORDS.
*        J RECIVES THE NUMBER OF CHARACTERS PLACED IN IA,
*                AND IS TYPED EITHER INTEGER*4 OR INTEGER*2.
*                IF NO PARM FIELD IS PRESENT, J WILL BE ZERO.
*        NOTE: ONLY THE FIRST J CHARACTERS OF IA WILL BE INTIALIZED.
SAVE     EQU     13
BASE     EQU     13
X        EQU     2
TEMP     EQU     1
FUNT     EQU     0
LINK     EQU     14
EP       EQU     15
         USING   PARM,EP
         ENTRY   PARM
PARM     B       GO
         DC      XL1'06'
         DC      CL6'PARM  '
REGS     DS      9D
C7       DC      1F'7'
C8       DC      1F'8'
C100     DC      1F'100'
MODEL    DC      CL6'MODEL='
GO       STM     14,12,12(SAVE)
         LR      X,SAVE
         LA      BASE,REGS
         DROP    EP
         USING   REGS,BASE
         ST      X,4(,SAVE)
         ST      SAVE,8(,X)
         L       11,0(,1)
KEEP     LR      3,2
```

```
         L     2,4(,3)          GET FIRST SAVE AREA
         LTR   2,2        FIRST IF NO BACK CHAIN
         BNZ   KEEP
         L     3,24(,3)
         L     3,0(,3)
         LTR   3,3
         BNM   RETURN
         LH    5,0(,3)
         C     5,C8
         BL    RETURN
         C     5,C100
         BH    RETURN
         LA    10,1
         LA    3,2(,3)          SKIP OVER COUNT
LOOPM    CLI   0(3),C'M'        LOOK FOR M
         BE    GOTM
LOOPMGO  AR    3,10
         BCT   5,LOOPM
RETURN   SR    0,0
         B     BYEBYE
GOTM     CLC   MODEL,0(3)
         BNE   LOOPMGO
         C     5,C7
         BL    RETURN
         IC    0,6(3)
         SLL   0,8
         IC    0,7(3)
         SLL   0,16
         SRA   0,16
BYEBYE   STH   0,0(,11)
BYEBYEGO L     13,4(,13)
         ST    0,20(,13)
         LM    14,12,12(13)
         MVI   12(13),X'FF'
         BR    14
         ENTRY ALPARM
         USING ALPARM,EP
ALPARM   B     GO2
         DC    XL1'06'
         DC    CL6'ALPARM  '
ADREG    DC    A(REGS)
MOVIT    MVC   0(0,11),2(3)
GO2      STM   14,12,12(SAVE)
         LR    X,SAVE
         L     BASE,ADREG
         DROP  EP
         USING REGS,BASE
         ST    X,4(,SAVE)
         ST    SAVE,8(,X)
         L     11,0(,1)
KEEP2    LR    3,2
         L     2,4(,3)          GET FIRST SAVE AREA
         LTR   2,2        FIRST IF NO BACK CHAIN
         BNZ   KEEP2
         L     3,24(,3)
         L     3,0(,3)
```

```
         LTR      3,3                                                  PRM10112
         BNM      RETURN                                               PRM10113
         LH       5,0(,3)                                              PRM10114
         LTR      5,5                                                  PRM10115
         BNP      RETURN                                               PRM10116
         C        5,C100                                               PRM10117
         BH       RETURN                                               PRM10118
         EX       5,MOVIT                                              PRM10119
         LR       0,5                                                  PRM10120
         B        BYEBYEGO                                             PRM10121
         END                                                           PRM10122
```

PART V

ABSADRES - ABSOLUTE ADDRESSING AND OTHER GOODIES

PART V

ABSADRES - ABSOLUTE ADDRESSING AND OTHER GOODIES

Access to absolute memory addresses is not available in FORTRAN, yet occasions do arise where complex coding can be dramatically simplified if variables can be accessed not by name, but by absolute address. It was to fill this need that the ABSADRES routines were written. Several additional routines were added to ease other situations.

There are eight classes of routines in the ABSADRES group. The first class returns the absolute address of its argument as its functional value. The second class returns as its functional value the value of the variable whose absolute address is given as an argument. The third class returns the address of its argument list as a functional value. The fourth calls the routine given as the second argument with the argument list address given as the first argument. The fifth clears an array to zeroes, and the sixth sets an array to blanks. The seventh class provides the complement of the second class by storing a value in a location referenced by its absolute address. The eighth class provides a null subroutine.

The first class of routines consists of seven entry points which return the address of their first argument as a functional value. The entry names are DADRES, RADRES, IADRES, LADRES, ADDRES, LOCFN, and LOCATN; and, due to FORTRAN conventions, they are all alike in that their functional values may be any type desired. However, the intention is that they would be types REAL*8, REAL*4, INTEGER*4, LOGICAL*4, REAL*4, INTEGER*4, and INTEGER*4, respectively. The multiple names were given to ease interfacing with FORTRAN coding conventions. Figure 1 is a table of Class 1 routines.

The second class of routines consists of nine entry points which return the value of a location in memory referenced by its absolute address given as an argument. All nine have this one argument, but the alignment and length of the value returned vary depending upon the routine called. One byte length with any alignment is assumed by entry BVALUE, while entry HVALUE assumes a length of two bytes and halfword alignment. LOGICAL*1 and INTEGER*2 are the suggested types for BVALUE and HVALUE, respectively, but INTEGER*4 may also be successfully used. Entries LVALUE, IVALUE, RVALUE, and VALUE are alike since all fetch four bytes aligned on full-word boundaries. Their suggested types are LOGICAL*4, INTEGER*4, REAL*4, and REAL*4, respectively. The DVALUE entry returns eight bytes with doubleword alignment as a double precision value. The two complex entries, CVALUE and CDVALU, set both real and imaginary parts from consecutive words and doublewords, respectively, and their suggested types are COMPLEX*8 and COMPLEX*16. Figure 2 is a table of the second class of routines.

The third class of routines consists of the single entry ARGADR which is an integer function returning the address of the argument list itself. The fourth class has two entries, ARGCAL and CONFUS, which are the same. They expect two arguments, both addresses. The first is the address of a routine name, which must appear in an EXTERNAL statement and to which control is transferred with an argument list whose address is the second argument. The latter address may be established by ARGADR.

The fifth and sixth classes have entries ZEROUT and BLANKS, respectively. They expect two arguments--the first is an array and the second is a word count of the length of the array. ZEROUT will then set to numeric zero (floating point and fixed point are the same) the array for as many four-byte words as specified. (Note: Word alignment is assumed and the user must adjust the word count to reflect element lengths other than four bytes.) BLANKS is the same as ZEROUT except four blank characters fill the four-byte words instead of numeric zeroes. These blanks are compatible with any standard FORTRAN A-type format.

The seventh class consists of nine subroutine entries which provide the complement of the second class of routines, since they store values into locations referenced by absolute address. All entries have two arguments--the absolute address of the location to be changed and the value it is to receive. The length of the value and the length of storage to be changed are determined by the entry point used. In all cases, the value is placed in the location specified one byte at a time without inspection, with no assumed alignment. Entry BSTORE moves one byte, while entry HSTORE moves two bytes. LOGICAL*1 for characters and INTEGER*2 would be typical data types for these routines. Entries RSTORE, LSTORE, ISTORE, and STORE all move four bytes (one word) with possible types of REAL*4, LOGICAL*4, INTEGER*4, and INTEGER, respectively. Each of CSTORE and DSTORE moves eight bytes with typical types of complex and double precision. The last entry, CDSTOR, moves 16 bytes and is used for the COMPLEX*16 data type. Figure 3 is a table of this class of routines.

The last class has one entry, ABSADR, which is a null routine consisting of only a RETURN statement. It may be invoked as a function or subroutine with or without an argument list of any sort.

These eight classes of routines can be used in some programming situations for simpler and faster programs. For example, many programs consist logically of two phases, an input and set-up phase and the transient or calculation phase. Two such programs are RELAP and PINSIM, both used by the ORNL-BDHT program [1]. They both allow the user to specify up to nine specific quantities to be printed in a "minor edit" with a very high frequency, with each minor edit producing one line with all nine variables and the transient time listed. In both cases the specific variable can be any of several dozen quantities (temperatures, pressures, densities, flow rates, etc.) which are defined for several positions (volumes, slabs, junctions, etc.). Both programs have input routines which decode the user input (which may appear as "AP 32" or "PHIW(1;6)") into an internal code which will allow the minor editing routine in the calculation phase to select the specific quantity to be printed. However, a dramatic difference in the two programs arises from PINSIM's use of ABSADRES and RELAP's use of normal FORTRAN techniques. PINSIM stores the absolute memory address of the specific quantity to be printed, while RELAP stores flags and pointers. Then, in the calculation phase, PINSIM, with an extremely simple, short loop (3 FORTRAN statements), obtains and prints the nine variables. RELAP, on the other hand, must decode the flags and pointers in a long, more complex loop (230 statements) to fetch the same nine variables. Indeed PINSIM's entire minor edit routine is only 39 statements, while RELAP's corresponding routine is 514 statements! The same technique could be used in the handling of trips, even to the point of resetting them when required.

A null subroutine is occasionally useful when an external routine can be specified. For example, ERRSET in the FORTRAN library allows the specification of user error-handling exit. The most used example of a null program segment is IEFBR14, which is a null routine.

Routine	Suggested Type
DADRES	DOUBLE PRECISION, REAL*8
RADRES	REAL, REAL*4
LADRES	LOGICAL, LOGICAL*4
IADRES	INTEGER, INTEGER*4
LOCFN	INTEGER, LOGICAL, OR REAL
LOCATN	INTEGER, LOGICAL, OR REAL

Figure 1

Class 1 Routines

Routine	Length	Alignment	Suggested Type
DVALUE	8 Bytes	Double Word	DOUBLE PRECISION, REAL*8
RVALUE	4 Bytes	Full Word	REAL, REAL*4
LVALUE	4 Bytes	Full Word	LOGICAL, LOGICAL*4
IVALUE	4 Bytes	Full Word	INTEGER, INTEGER*4
VALUE	4 Bytes	Full Word	REAL, INTEGER, OR LOGICAL
BVALUE	1 Byte	Byte	LOGICAL*1
HVALUE	2 Bytes	Halfword	INTEGER*2
CVALUE	8 Bytes	Full Word	COMPLEX, COMPLEX*8
CDVALU	16 Bytes	Double Word	COMPLEX*16

Figure 2

Class 2 Routines

Routine	Length	Suggested Type
BSTORE	1 Byte	LOGICAL*1
HSTORE	2 Bytes	INTEGER*2
RSTORE	4 Bytes	REAL*4
LSTORE	4 Bytes	LOGICAL*4
ISTORE	4 Bytes	INTEGER*4
STORE	4 Bytes	INTEGER, REAL OR LOGICAL
CSTORE	8 Bytes	COMPLEX*8
DSTORE	8 Bytes	REAL*8
CDSTOR	16 Bytes	COMPLEX*16

Figure 3

Class 7 Routines

The argument list addressing routines have yet to be applied in an actual applications program, but an array of addresses of routines and a corresponding array of addresses of arguments lists could be passed to a routine which invoked the various routines in the array "blindly," without knowing which routine was being invoked, Possible uses include the dynamic specification of the execution path of a program. Also, since an argument list is simply an array of addresses, the user can build an argument list dynamically.

Appendix 1 is a complete source listing.

PART V, Appendix 1

SOURCE LISTING OF ABSADRES

```
ABSADRES CSECT 0
*
* THESE ROUTINES ARE DESIGNED TO GIVE FORTRAN PROGRAMS ACCESS
*   TO ABSOLUTE ADDRESSING SCHEMES AND OTHER GOODIES
*
* THE FIRST ROUTINE RETURNS THE ABSOLUTE ADDRESS OF ITS ARGUMENT AS ITS
*   FUNCTIONAL VALUE
*
* THE SECOND ROUTINE RETURNS THE VALUE STORED IN THE ABSOLUTE ADDRESS
*   GIVEN BY ITS ARGUMEN AS ITS FUNCTIONAL VALUE.
*
* BOTH ROUTINES RETURN INTEGER, SINGLE PRECISION FLOATING POINT,
*   DOUBLE PRESCISION FLOATING POINT, AND LOGICAL VALUES, HENCE THE
*   SEVERAL ENTRY POINTS. I*2 AND L*1 ARE NOT SUPPORTED FOR THE FIRST
*   FUNCTION. EXTENDED PRECISION IS NOT SUPPORTED.
*
* STANDARD FORTRAN FUNCTION LINKAGE IS ASSUMED, BUT THESE ROUTINES DO
*   NOT APPEAR IN TRACEBACKS IF ADDRESSING, PROTECTION, OR ALIGNMENT
*   ERRORS OCCUR.
*
* GENERAL PURPOSE REGISTER ZERO IS USED FOR VALUE RETURN AS IS FP REGO
*
* THE THIRD ROUTINE RETURNS THE ADDRESS OF THE ARGUMENT LIST ITSELF
*
* THE FOURTH CALLS THE ROUTINE GIVEN AS THE SECOND ARGUMENT WITH THE
*   ARGLIST GIVEN AS THE FIRST ARGUMENT
*
* THE FIFTH CLEARS AN ARRAY TO ZEROS
* THE SIXTH CLEARS AN ARRAY TO BLANKS
*
* VERSION 1.0, APRIL 10,1977 STEVEN B. CLIFF
*
*
*
*
*
* ALL OF THE ENTRIES IN THE FIRST, SECOND, AND THIRD ROUTINES ARE
*   FUNCTIONS AND ALL USE THE FIRST ARGUMENT. IF MORE ARGS ARE PRESENT,
*   THEY ARE IGNORED.
*
* THE FUNCTION TYPE FOR ROUTINE IS ENTIRELY UP TO THE CALLING PROGRAM
*   AS INTEGER*4,REAL*4, AND REAL*8 VALUES ARE RETURNED. THE ARGUMENT
*   MAY BE OF ANY DESIRED.
*
* THE FUNCTION TYPE FOR ROUTINE 2 IS SOMEWHAT ENTRY DEPENDANT:
*   ENTRY     FUNCTION TYPE          ARGUMENT TYPES (AS USED TO ROUTINE1)
*   BVALUE    I4,L4,L1,I2            ANY  (ANY ALIGMENT)
*   HVALUE    I4,L4,L1,I2            ANY EXCEPT L1 (HALF WORD ALIGMENT)
*   RVALUE    I4,L4,R4,R8,L1,I2      ANY EXCEPT I2,L1 (FULL WORD ALIGMENT)
*   LVALUE    F4,L4,R4,R8,L1,I2      ANY EXCEPT I2,L1 (FULL WORD ALIGMENT)
*   IVALUE    I4,L4,R4,R8,L1,I2      ANY EXCEPT I2,L1 (FULL WORD ALIGMENT)
*   VALUE     I4,L4,R4,R8,L1,I2      ANY EXCEPT I2,L1 (FULL WORD ALIGMENT)
*   DVALUE    I4,L4,R4,R8,L1,I2      R8,C8,C16 (DBLE WORD ALIGMENT)
*   CVALUE    I4,L4,R4,R8,C8,I2,L1,I2    C8  (TWO FULL WORD ALIGNMENTS)
*   CDVALU    I4,L4 R4,R8,C8,C16,L1,I2   C16 (TWO DBLE WORD ALIGNMENTS)
*
```

```
* WHILE ALL THE ABOVE TYPE MIXES ARE LEAGAL, THE FOLLOWING IS DESIGNED:
*
*   BVALUE    I4                        CHARACTORS,L1
*   NVALUE    I4                        CHARACTORS,I2
*   RVALUE    R4                        R4
*   LVALUE    L4                        L4
*   IVALUE    I4                        I4
*   VALUE     R4,I4                     R4,I4
*   DVALUE    R8                        R8
*   CVALUE    C8                        C8
*   CDVALU    C16                       C16
*
* OF COURSE LOTS OF GAMES CAN BE PLAY WITH UNUSUAL RESULTS WITH
*   ALL THESE ENTRIES
*
*
* ROUTINE 3 COULD NOT CARELESS ABOUT THE TYPE AND OR LENGTH OF THE
*   ARGUMENT IT IS LANDED
*
* ROUTINE 4 EXPECTS ADDRESSES AS BOTH ARGUMENTS (AS DOES ALL OF
*   ROUTINE 2, FOR THAT MATER), THE FIRST EITHER FROM ROUTINE 3
*   OR SOME SIMILAR TRICK, THE SECOND CAN BE FROM ROUTINE 1 OR
*   AN EXTERNAL ROUTINE IN FORTRAN
*
* ROUTINES 5 AND 6 EXPECT ARRAYS AS THE FIRST ARG AND INTEGERS AS THE
*   SECOND. FULL WORD ALIGNMENT IS ASSUMED
*
* SOME EQUILVALENT ACTIONS BASED IN FORTRAN
*  IMPLICT I4(I-K),L*4(L),R4(R-Z),R8(D),C8(C),C16(CD),I4(B,LOCFN)
*  I=IADRES(J)                          I=ADDRES(J)     (ADDRES AS I4)
*  R=RADRES(S)                          R=ADDRES(S)     (ADDRES AS R4)
*              ALL THE ADDRESS ROUTINES ARE EQULLVALENT
*
*
*  I=IVALUE(IADRES(J))                  I=J
*  R=RVALUE(RADRES(S))                  R=S
*  L=LVALUE(LADRES(L1))                 L=L1
*  K=IADRES(J);I=IVALUE(K)              I=J
*  I=BVALUE(LOCFN(J)) SET BOTTOM BYTE OF I TO TOP BYTE OF J
*  I=BVALUE(LOCFN(J)+2) SET BOTTOM BYTE OF I TO THIRD BYTE OF J
*  R=RVALUE(RADRES(C)+4)                R=AIMAG(C)
*     AND MUCH MORE STRANGE POSSIBLITIES
*
*  V=ARGCAL(ARGADR(X),SIN)              V=SIN(X)
*  CALL CONFUS(ARGADR(I,J,0),PDUMP)     CALL PDUMP(I,J,0)
*  V=ARGCAL(ARGADR(X),ADDRES(SIN) V=SIN(X)
*
* THE POSSIBLITIES BY SETTING VARIABLES AND LATER USEING THEM ARE
*   QUITE NUMEROUS
*
*
*
* ROUTINE 1: RETURN THE ADDRESS
*
         ENTRY DADRES,RADRES,LADRES,IADRES,ADDRES,ABSADR,LOCFN,LOCATN
* ALL OF THESE ENTRY POINTS ARE IDENTICAL, DIFFERENT NAMES ARE
```

```
*   GIVEN TO EASE FOOLING THE COMPILER
*
DADRES   EQU   *
RADRES   EQU   *
LADRES   EQU   *
IADRES   EQU   *
ADDRES   EQU   *
LOCFN    EQU   *
LOCATN   EQU   *
         MVI   0(1),X'00'          CLEAR HIGH ORDER BIT
         SDR   0,0      CLEAR FP REG 0 TO RECIEVE ADDRESS IN TOP
         LE    0,0(,1) GET ADDRESS FOR FLOATING RETURNS
         L     0,0(,1) GET ADDRESS FOR FIXED POINT RETURNS
         BR    14       RETURN
*
*
* ROUTINE 2: RETURN THE VALUE
         ENTRY DVALUE,RVALUE,LVALUE,IVALUE,VALUE,BVALUE,CVALUE,HVALUE
*
* DIFFERENCES IN ROUTINES ARE DUE TO ALIGNMENT ASSUMTIONS
*
* BYTE ALIGNMENT (FP NOT ALLOWED)
BVALUE   EQU   *
         SR    0,0      CLEAR REG FOR FIXED POINT RETURN
         L     1,0(,1)
         L     1,0(,1)        GET ARG.
         IC    0,0(,1) GET BYTE
         BR    14       RETURN
*
* HALFWORD ALIGNMENT (FP NOT ALLOWED)
HVALUE   EQU   *
         L     1,0(,1) GET ADDRESS
         L     1,0(,1)        GET ARG.
         LH    0,0(,1) GET HALFWORD
         BR    14       RETURN
* FULLWORD ALIGNMENT
RVALUE   EQU   *
LVALUE   EQU   *
IVALUE   EQU   *
VALUE    EQU   *
         SDR   0,0      CLEAR FP REG 0
         L     1,0(,1)
         L     1,0(,1) GET ADDRESS OF FULL WORD
         LE    0,0(,1) GET FP VALUE
         L     0,0(,1) GET FIXED POINT VALUES
         BR    14       RETURN
*
*
* DOUBLE WORD ALIGNMENT
DVALUE   EQU   *
         L     1,0(,1)
         L     1,0(,1) GET ADDRESS
         LD    0,0(,1) GET DOUBLE WORD
         L     0,0(,1) ALSO GET BINARY VALUE(TOP HALF ONLY)
         BR    14       RETURN
*
```

```
* SINGLE PRECISION COMPLEX
CVALUE   EQU   *
         SDR   0,0      CLEAR FP REG 0&2 SO VALUE IS ALSO PRECISION
         SDR   2,2       INCREASED FROM SINGLE TO DOUBLE IF DESIRED
         L     1,0(,1) GET ADDRESS
         L     1,0(,1)
         LE    0,0(,1) GET REAL PART
         LE    2,4(,1) GET IMAGINARY PART
         L     0,0(,1) GET BINARY PART?
         BR    14       RETURN
*
* DOUBLE PRECISION COMPLEX
         ENTRY CDVALU
CDVALU   EQU   *
         L     1,0(,1) GETADDRESS
         L     1,0(,1)
         LD    0,0(,1) GET REAL PART
         LD    2,8(,1)  GET IMAGINARY PART
         L     0,0(,1)  GET BINARY PART?
         BR    14        RETURN
*
* ROUTINE  3: RETURN ADDRESS OF  ARGUMENT LIST
         ENTRY ARGADR
ARGADR   EQU   *
         LR    0,1    GET ADDRES OF ARG LIST
ABSADR   EQU   *
         BR    14     RETURN
*
* ROUTINE  4: CALL SECOND ARG WITH FIRST AS ARGLIST
         ENTRY CONFUS,ARGCAL
ARGCAL   EQU   *
CONFUS   EQU   *
         L     15,4(,1) GET ADDRESS OF SECOND ARGUMENT
         L     1,0(,1)  SET NEW ARG LIST
         L     1,0(,1)
         BR    15        CONTINUE ONWARD
*
* ROUTINE  5: ZERO THE ARRAY THAT IS THE FIRST ARG, WHICH SEOND
*    ARGUMENT WORDS LONG
         ENTRY ZEROUT,BLANKS
         USING BLANKS,15
ZEROUT   SR    0,0       GET ZERO
         LA    15,10(,15) FIX BASE REG
         B     ZANDB
*
* ROUTINE  6: BLANKS THE ARRAY LIKE ZEROUT
BLANKS   L     0,BLANK  GET BLLANKS
ZANDB    ST    14,SAVE14 SAVE 14
         L     14,4(,1)
         L     14,0(,14) GET ARG # 2-THE COUNT
         L     1,0(,1)   GET ARG # 1-THE ARRAY
LOOPIT   ST    0,0(,1)   SET NEXT WORD
         LA    1,4(,1)   BUMP POINTER
         BCT   14,LOOPIT SKIP BACK COUNT TIMES
         L     14,SAVE14 RESTORE 14
         BR    14        RETURN
```

```
*
* ROUTINE 7 STORES IN ABSOLUTE LOCATIONS -THE  EXACT OPPOSITE
*  OF ROUTINE 2 EXCEPT 7 IS SUBROUTINE WHILE 2 IS FUNCTION
*
* USAGE:  (ALL E.P. SIMILAR)
*   CALL STORE (LOCA,VALUE)
*
*      WHERE LOCA IS ABSOLUTE ADDRESS TO RECIEVE VALUE VALUE.
*      VALUE HAS LENGTH IMPLIED BY CHOICE OF E.P.
*
         USING *,15
         ENTRY DSTORE,RSTORE,LSTORE,ISTORE,STORE,BSTORE,CSTORE,HSTORE
         ENTRY CDSTOR
BSTORE   EQU   *
         LA    0,0      MOVE 1 BYTE     (ZERO SINCE CNT IS 1 LOW)
         LA    15,DOIT
         BR    15
*
         USING *,15
HSTORE   EQU   *
         LA    0,1      MOVE 2 BYTES
         LA    15,DOIT
         BR    15
*
         USING *,15
RSTORE   EQU   *
LSTORE   EQU   *
ISTORE   EQU   *
STORE    EQU   *
         LA    0,3      MOVE 4 BYTES
         LA    15,DOIT
         BR    15
CSTORE   EQU   *
         USING *,15
DSTORE   EQU   *
         LA    0,7      MOVE 8 BYTES
         LA    15,DOIT
         BR    15
*
         USING *,15
CDSTOR   LA    0,15     MOVE 16 BYTES
         LA    15,DOIT
         USING DOIT,15
DOIT     EQU   *
         ST    14,SAVE14 SAVE 14
         L     14,4(,1) GET ADDRS OF VALUE
         L     1,0(,1)  GET ADDRS OF ARG
         L     1,0(,1)  GET ARG= ADDRS TO RECIEVE VALUE
         STC   0,MVIT+1 SET NUMBER OF CHARACTOR TO USE
MVIT     MVC   0(0,1),0(14) MOVE IT
         L     14,SAVE14 RESET  14
         BR    14          RETURN, NOTE: REG 0 HAS # OF BYTES MOVED
SAVE14   DS    1F
BLANK    DC    X'40404040'
         END
```

PART VI

VARIN - VARIABLE LENGTH RECORD INPUT ROUTINE

PART VI

VARIN - VARIABLE LENGTH RECORD INPUT ROUTINE

Normal IBM FORTRAN cannot read variable length records with format control. These variable length records include not only those from run time FORTRAN but also the compiler SYSPRINT and other common systems programs. The subroutine VARIN allows these records to be successfully read by run time FORTRAN under A1 format.

The Queued Sequential Access Method (QSAM) is used to read records of length not greater than 137 bytes and in the variable blocked format with ASA carriage control characters. The DDNAME is SYSIN and the file must be physical sequential.

VARIN internally fixes the DCB as follows:

```
//SYSIN DD DCB=(DSORG=PS,LRECL=137,RECFM=VBA,OPTCD=C).
```

Normally only the blocksize (BLKSIZE) subparameter needs to be specified in the DCB parameter. Of course, unit, dataset name, volume, etc., information must be supplied as needed.

The calling sequence to VARIN is

```
CALL VARIN (LEN,REC)
```

where LEN is the number of characters in the input record (-1 on the end of file) and REC is the storage area (133 words) which receives the input record in an A1 format (A1 format has one character per word in the high-order byte with blank characters in the low-order three bytes).

While all 133 words of REC are initialized to blanks, only the first LEN words will have data from the record. LEN is exactly the

LRECL for the current record. LEN is minus 1 (-1) and REC is all blanks upon end of file. VARIN opens DDNAME SYSIN on the first call and leaves it open until end of file when it is closed.

VARIN may also be invoked as an integer function with value the same as LEN. For example, after

```
INTEGER VARIN
LEN1=VARIN(LEN2,REC)
```

LEN1 and LEN2 will have identical values.

VARIN has been used to process FORTRAN compiler SYSPRINT output for the microfiche indexing routine FFIN [1].

Appendix 1 is a complete source listing.

REFERENCE

1. Steven B. Cliff and Brenda D. Dingus, *FFIN - FORTRAN Microfiche Indexer*, K/CSD/INF-78/10, March 1978.

PART VI, Appendix 1

SOURCE LISTING OF VARIN

```
VARINOUT CSECT 0
         ENTRY VARIN
*
* CALL VARIN(LEN,REC)
*    TO RECEIVE LEN WORDS (A1 FORMAT) IN REC
*    LEN=0 ON EOF.  ALSO, VARIN MAY BE USED AS INTEGER FUNCTION = LEN
         USING IHADCB,10
*
*
         USING SAVEA,13
SAVEA    DS    18F
VARIN    SAVE  (14,12),,*
         USING VARIN,15
         LR    3,13                LINK
         L     13,ADSAV            SAVE
         DROP  15                  AREA
         ST    13,8(3)             AS
         ST    3,4(13)             REQUIRED
         LM    11,12,0(1)          GET ARGS 11=LEN,12=REC
         LA    10,DCBIN            SET BASE FOR DCB
         TM    DCBOFLGS,DCBOFOPN   IS SYSIN OPEN?
         BNZ   INOPEN              YES
*        WTL     'OPEN SYSIN'
         OPEN  (DCBIN,INPUT)
INOPEN   EQU   *                   SYSIN IS OPEN,CLEAR REC
         MVI   0(12),X'40'
         MVC   1(200,12),0(12)     CLEAR 50 WORDS
         MVC   201(200,12),200(12)         CLEAR 51 TO 100 WORDS
         MVC   401(131,12),400(12)         CLEAR 101 TO 133 WORDS
         GET   DCBIN,RDW           GET NEXT RECORD
         LH    9,DCBLRECL                  GET   CHAR IN BUF
         S     9,C4
         ST    9,0(,11)            SET LEN
         ST    9,20(,13)           SET FUNCTIONAL VALUE
         LA    8,BUF
LOP      MVC   0(1,12),0(8)        MOVE 1 CHARACTOR
         LA    12,4(,12)           SKIP THROUGH REC (A1)
         LA    8,1(,8)             SKIP THROUGH BUF (A4)
         BCT   9,LOP               MOVE RDW CHARACTORS
         L     13,4(,13)           UNLINK
         RETURN (14,12),T          RETURN
*
*END OF FILE
ENDSYSIN EQU     *
*        WTL   'END OF SYSIN'
         CLOSE (DCBIN)
         L       0,M1
         ST      0,0(,11)
         ST      0,20(,13)
         L       13,4(,13)
         RETURN (14,12),T
M1       DC      1F'-1'
C4       DC      1F'4'
RDW      DS      1F                RECORD DESCRIPTOR WORD
```

```
BUF      DS    40F                 THE INPUT RECORD                     VARN0640
ADSAV    DC    A(SAVEA)                                                 VARN0650
DCBIN    DCB   DDNAME=SYSIN,DSORG=PS,EODAD=ENDSYSIN,LRECL=137,          XVARN0660
               RECFM=VBA,OPTCD=C,MACRF=GM                               VARN0670
         DCBD  DSORG=QS,DEVD=RD                                         VARN0680
         END                                                            VARN0690
```

PART VII

ABEND - USER-REQUESTED ABNORMAL PROGRAM END ROUTINE

PART VII

ABEND - USER-REQUESTED ABNORMAL PROGRAM END ROUTINE

Nearly all programs encounter abnormal conditions which are best handled by program terminations. The FORTRAN STOP n statement could be used in such situations, but, since the traceback capabilities of the FORTRAN extended error handling feature are not invoked, no record of the routine with control when the error was recognized or of its calling history is made available to the user. While the error code (assignable with the number n on the STOP n statement) is very useful, often it alone does not provide sufficient information. To meet this need, this ABEND routine was written. It not only allows a numeric code, but it also invokes the traceback feature.

The calling sequence for ABEND is

```
CALL ABEND (ERRCOD)
```

where ERRCOD is a four-byte integer with value between 0 and 4095, inclusive.

ABEND saves the caller's registers in a standard system SAVE area at entry point ABENDREG, making their location in any dump easier to find. Then ABEND calls ERRTRA, a standard entry in the FORTRAN library to produce a traceback. From this traceback, the full calling sequence up to the call to ABEND can be determined, including the statement number of the call to ABEND itself. Further, Register 0 in the traceback is the hexadecimal representation of the ERRCOD specified in the argument. Three WRITE TO LOG (WTL) macros are then issued, putting the lines

```
USER ABEND

      PROGRAM ABNORMAL TERMINATION REQUEST

USER ABEND
```

in the system log listing of the program. Finally, a supervisor request for an abend is issued. This abend will have the user-completion code specified by the ERRCOD argument. The supervisor then terminates the program writing a dump to any SYSUDUMP or SYSABEND datasets present. ABEND does not return to the calling routine.

Appendix 1 is a complete source listing.

PART VII, Appendix 1

SOURCE LISTING OF ABEND

```
ABEND    CSECT  0
         ENTRY  ABENDREG
         EXTRN  ERRTRA
         PRINT  GEN
         USING  *,15
         SAVE   (14,12),,*         SAVE ALLTHE REGS AND SET LINKAGES
         LA     2,ABENDREG
         ST     13,4(0,2)
         ST     2,8(0,13)
         LR     13,2
         B      BEGIN
ABENDREG DS     18F                SAVE AREA
         USING  ABENDREG,13
         DROP   15
BEGIN    L      3,0(0,1)           GET ADDR OF ARG
         L      0,0(0,3)           GET ARGUMENT INTO REG 0 FOR TRACEBACK
         L      2,192(0,0)         GET DUMP, STEP CODES
         LR     3,0
         SLL    3,4
         SRDL   2,4
         L      15,ERRT            GET THE TRACE BACK
         BALR   14,15
         WTL    'USER ABEND'
  WTL '           PROGRAM ABNORMAL TERMINATION REQUEST'
         WTL    'USER ABEND'
         LA     15,0(0,3)          GET ABEND CODE
         LR     1,3       GET ARGS FOR ABEND CALL
         SVC    13                 DO ABEND
ERRT     DC     A(ERRTRA)          ADRES OF TRACE BACK ROUTINE
         END
```

PART VIII

SET - ARRAY-SETTING ROUTINES

PART VIII

SET - ARRAY-SETTING ROUTINES

One of the most common actions a FORTRAN program takes is the setting of an array to constant values, typically requiring a complete DO loop. These three routines were written to simplify this task.

The primary routine is the general array setting routine SET with calling sequence:

CALL SET (LEN, SKIP, ARRAY, WORD)

where

LEN is a four-byte INTEGER specifying the number of full words to be assigned a value,

SKIP is a four-byte INTEGER specifying the number of full words to be skipped between each assigned word,

ARRAY is a four-byte array to be assigned with total length at least LEN times SKIP,

WORD is the four-byte value to be used in setting ARRAY.

WORD (which may contain an INTEGER, LOGICAL, or CHARACTER value) is placed in the first word of ARRAY, then in FIRST + SKIP, then FIRST + 2 * SKIP,..., then FIRST + LEN * SKIP. If SKIP is 1, consecutive memory locations will be set, as would be desired in a single-dimensional array or when setting all of a multidimensional array. If SKIP is not 1, parts of a multidimensioned array may be set, allowing columns or planes of two- or three-dimensional arrays to be defined without altering the remainder.

The two most common uses of SET would be to define all of an array to either a numeric zero or to blank characters. To ease this operation, two additional entries are defined with calling sequences:

CALL SETBLK(LEN,ARRAY)

CALL SETZER(LEN,ARRAY)

where the arguments are the same as for SET. The missing arguments, SKIP and WORD, are fixed. SKIP is set to 1, for consecutive location assignment. WORD is set to blanks (Hex '40404040') for SETBLK and to numeric zero (Hex '00000000') for SETZER. Note that integer and floating-point zero are identical and that even DOUBLE PRECISION variables can be initialized by SETZER or SETBLK if the LEN variable is adjusted to account for the extra length of the variable ARRAY.

Appendix 1 is a complete source listing.

PART VIII, Appendix 1

SOURCE LISTING OF SET

```
SET      CSECT 0                                                           SET10000
         USING *,15                                                        SET10001
*     CALL SET (LEN,SKIP,ARRAY,WORD)                                       SET10002
*     TAKES "WORD" AND PUTS IT IN LEN LOCATIONS OF ARRAY,STARTING          SET10003
*        WITH THE FIRST THEN FIRST+SKIP,FIRST+2*SKIP.....                  SET10004
*             FIRST+LEN*SKIP                                               SET10005
*LEN = A*B*C WHERE DIMENSION ARRAY(A,B,C)                                  SET10006
*EXAMPLE CALL SETZER(10*20,NUM) WHERE DIMENSION NUM (10*20)                SET10007
* IF THE LENGTH OF THE VARIABLE IS OTHER THAN 4 BYTES, LEN MUST            SET10008
*   BE ADJUSTED ACCORDINGLY                                                SET10009
*  IN ALL CASES ARRARY SHOULD BE ALIGNED ON A FULL WORD BOUNDARY           SET10010
*    (THIS IS AUTOMATIC FOR ALL BUT INTEGER*2 & LOGICAL*1)                 SET10011
*THIS IS A TRUE SUBROUTINE-WILL APPEAR IN TRACE BACK                       SET10012
*                                                                          SET10013
*     CALL SETBLK (LEN,ARRAY)                                              SET10014
*        SAME AS SET EXCEPT WORD IS 4 BLANK CHARACTORS, SKIPS = 1          SET10015
*     CALL SETZER (LEN,ARRAY) SAME AS SET EXCEPT WORD IS NUMERIC           SET10016
*        ZERO,SKIP = 1                                                     SET10017
*SETZER WORKS FOR BOTH REAL AND INTEGERS AS  DESCRIBED ABOVE               SET10018
         ENTRY SETZER                                                      SET10019
         ENTRY SETBLK                                                      SET10020
         SAVE  (14,12),,*                SAVE REGS & LINK SAVE AREA        SET10021
         LM    2,5,0(1)                  GET ADDRESS OF ARGUMENTS          SET10022
*  REG 2 HAS THE LENGTH OF THE ARRAY                                       SET10023
*  REG 3 HAS THE NUMBER OF WORDS TO B SKIPPED BETWEEN INSERTIONS           SET10024
*  REG 4 HAS FIRST ELEMENT TO RECIECE A CHAR                               SET10025
*  REG 5 HAS THE WORD TO BE STORED                                         SET10026
         L     9,0(3)                    GET SKIPS                         SET10027
         LA    6,4                       GET CONSTANT FOUR                 SET10028
         MR    8,6                       MAKE SKIPS SUOTABLE FOR BYTEADD   SET10029
         LR    3,9                       3 NOW HAS THE BYTE FORM OFSKIP    SET10030
         L        10,0(5)                                                  SET10031
         LR    12,15                                                       SET10032
         B     ALLTOGTH                                                    SET10033
         DS    0D                                                          SET10034
         USING    *,15                                                     SET10035
SETBLK   SAVE     (14,12),,*                                               SET10036
         L     10,BLANKS                                                   SET10037
         L     12,ASF                                                      SET10038
         B     TOGETHER                                                    SET10039
         DS    0D                                                          SET10040
         USING *,15                                                        SET10041
SETZER   SAVE  (14,12),,*                                                  SET10042
         L     10,ZERO                                                     SET10043
         L     12,ASF                                                      SET10044
TOGETHER L        2,0(1)                                                   SET10045
         L        4,4(1)                                                   SET10046
         LA       3,4                                                      SET10047
         USING    SET,12                                                   SET10048
         DROP 15                                                           SET10049
ALLTOGTH LA    8,SAREA                                                     SET10050
         ST       13,4(8)                                                  SET10051
         ST       8,8(13)                                                  SET10052
         LR       13,8                                                     SET10053
         L     11,0(2)                   GET LENGTH                        SET10054
         LPR   11,11                     INSURE THAT IT IS POSITIVE        SET10055
```

```
         BZ     *+16                                                        SET10056
STORE    ST     10,0(0,4)              STORE WORD                           SET10057
         LA     4,0(3,4)              INCREMENT 4 BY SKIPS                  SET10058
         BCT    11,STORE              RETURN TO DO NEXT WORD                SET10059
         L      13,4(13)                                                    SET10060
         RETURN      (14,12),T                                              SET10061
ASF      DC     A(SET)                                                      SET10062
SAREA    DS     9D                                                          SET10063
BLANKS   DC     X'40404040'                                                 SET10064
ZERO     DC     1F'0'                                                       SET10065
         END                                                                SET10066
```

K/CSD/TM-20

INTERNAL DISTRIBUTION

Computer Sciences

1. L. L. Anthony
2. D. E. Arnurius
3. B. Beard
4. A. A. Brooks
5. H. P. Carter/ ORNL CSD Library

6-25. S. B. Cliff

26. R. L. Cox
27. K. E. Cross
28. J. S. Crowell
29. B. D. Dingus
30. E. D. Drennen
31. R. H. Fowler
32. R. E. Funderlic
33. P. Gaffney
34. G. E. Giles
35. R. W. Henderson
36. H. R. Hicks
37. J. T. Holdeman
38. S. K. Iskander
39. R. D. McCulloch
40. J. E. Park
41. C. E. Price
42. L. I. Schlemper
43. J. G. Sullivan

44-47. R. E. Textor

48. J. A. Tindall
49. J. N. Tunstall
50. W. D. Turner
51. G. W. Westley
52. G. E. Whitesides
53. J. W. Wooten
54. J. H. Zeigler
55. ORGDP CSD Library

Gaseous Diffusion

56. G. J. Kidd
57. G. F. Malling

Operations Analysis and Planning

58. E. Von Halle
59. H. G. Wood

Separation Systems

60. A. J. Szady

Engineering

61. W. C. Stoddart

Engineering Technology

62. R. C. Hager
63. R. A. Hedrick
64. D. G. Thomas

ORGDP Information Services

65-67. D. S. Napolitan

ORGDP Library

Copies 68-71

ORGDP Records - RC

Copy 72

EXTERNAL DISTRIBUTION

73. J. H. Forrester, The University of Tennessee, Knoxville, TN 37916

74-100. Technical Information Center, Department of Energy, Post Office Box 62, Oak Ridge, TN 37830