What the DI3000 EXEC Does

The DI3000 EXEC is written in EXEC2 and depends on features of VM/SP. It uses the following CP and CMS commands:

CONWAIT	GENMOD	LOAD	SET
DROPBUF	GLOBAL	MAKEBUF	START
EXECIO	HELP	QUERY (STACK	
FILEDEF	INCLUDE	SENTRIES	

The DI3000 EXEC works like this:

1. Checks for ? among arguments and prints help if ? is found. Uses Cornell's HELP processor, but it should not be too difficult to convert the help file to IBM HELP format.

2. Checks for R/W A disk.

3. Sets default options. You can change the defaults in this section.

4. Parses command line

   a. Checks that at least the minimum abbreviations were requested.

   b. Compares each option against list of legal options. You can change the list of acceptable options.

   c. Parses all options, even if a bad one is found. Does not load or run routines if one or more bad options found.

   d. Save current CMS settings and turns off any settings that could cause an asynchronous message to appear in the graphics area if certain message-sensitive devices were requested. You can modify this section to turn other settings off or include other devices in the list of sensitive ones.

5. Sends a message to USE to keep track of the number of times this exec was successfully invoked. Delete this line.

6. CLEARs and sets FILEDEFs for TERM and the error message file. Cornell calls the file DI3000 MESSAGES; PVI calls it ERROR MESSAGES.

7. GLOBALs TXTLIBs corresponding to the sections of DI3000 requested.

8. LOADs application program using the CLEAR option.

9. INCLUDEs VS FORTRAN and CMS runtime libraries (we had too many TXTLIBs to GLOBAL them all).

10. GENMODs and/or RUNs loaded module depending on options requested.

11. Cleans up (restores any saved settings, CLEARs FILEDEFs).

Throughout, the exec checks for error conditions and uses EXECIO to print CMS-style messages.

SHARE

SHARE SESSION REPORT

SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
61	A733	Using the Fortran Extended Error	

PROJECT	SESSION CHAIRMAN	INST. CODE
Fortran	W. Horowitz	CSS

D & B Computing Services 187 Danbury Rd. Wilton, Ct. 06897

SESSION CHAIRMAN'S COMPANY, ADDRESS, AND PHONE NUMBER

USING THE FORTRAN EXTENDED ERROR HANDLER

May 1983

Pat Hennessy

Hughes Aircraft Co.
2000 E. El Segundo Blvd. E1/F128
El Segundo CA 90245
(HUG)

4/B/RC/1

ABSTRACT

The extended error handler is a set of routines which permit the Fortran programmer to take control when exceptional events occur during execution of the Fortran library subroutines. This paper gives several examples illustrating the use of these error handling routines.

The routines may be used to extend processing capability by circumventing the standard error handling associated with the language, or they may be used to provide informative messages, useful in debugging during execution.

Five subroutines will be discussed: ERRSAV, ERRSTR, ERRTRA, ERRSET, and ERRMON. In addition, examples of user exit subroutines will be presented.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

THE OPTION TABLE

DESCRIPTION OF OPTION TABLE ENTRIES

In the Fortran library there is a member, IFYUOPT, which contains no executable code. It is a table of constants called the "option table". Associated with each error detected by routines in the Fortran library there is an entry in the option table which determines the action which will take place when that error is encountered. A description of each type of error can be found in *VS FORTRAN Application Programming: Library Reference SC26-3989.*

EXAMINING THE OPTION TABLE ENTRIES

Each table entry occupies 8 bytes (a double word). Figure 1 shows a program I used to examine the option table entry for error 218 (a permanent I/O error occurred).

```
      REAL *8 OTE218
      CALL ERRSAV(218,OTE218)
      WRITE(6,1) OTE218
1     FORMAT(' OPTION TABLE ENTRY FOR 218: ',Z16)
      STOP
      END
```

Figure 1. Examining an Option Table Entry.

The output is: OPTION TABLE ENTRY FOR 218: 0A05005200000001

We interpret this result as explained in Appendix D of *VS FORTRAN Application Programming: Language Reference GC26-3986.*

The first byte '0A' means that we will tolerate a 218 error ten (the decimal equivalent of 0A) times. If we have ten errors of this type, execution should be terminated. If this byte had had the value 00, it would mean to never terminate execution because of a 218 error, but to continue regardless of how many times it occurs.

The second byte '05' determines the number of times the error message associated with a 218 error should be printed. (see also the description of bit 5 below)

The next byte '00' shows how many times the error has occurred during this execution. Note that 255 is the maximum value one byte can represent. One of the flag bits (see below) is used as the 256's place, and counting may continue up to 511. (see description of bit 2 below).

The next byte '52', the flag byte, must be interpreted in binary: 01010010. Each bit is an internal flag.

Bit 0 — indicates whether or not a carriage control character should be supplied when an output line is generated by the error handler (see the discussion of 212 below).

Bit 1 — indicates whether or not a Fortran user may modify this entry in the option table.

Bit 2 — this bit is the 256's place of the error count. Counting stops at 511.

Bit 3 — indicates whether or not the contents of the I/O buffer should be printed.

Bit 4 — give an informative message only.

Bit 5 — can override the number of times an error message prints:

If 0, it means that the value given in the second byte should rule.

If 1, it means that every occurrence of the error should produce a message.

Bit 6 — determines whether or not a traceback map should be printed.

Bit 7 — is reserved.

The default Option Table uses the following values for the flag byte:

00 — No options selected.

Error 205

02 — A traceback should be printed.

Errors 153, 156-158, 162-165, 167, 168, 230, 240

42 — User may modify, traceback should be printed.

Errors 140-150, 152, 154, 155, 159-161, 166, 169-204, 206-211, 213, 214, 216, 217, 219, 220, 226, 228, 231-237, 239, 241-301

4C — User may modify, always print informative message.

Error 151

52 — User may modify, I/O buffer and traceback should be printed.

Errors 212, 215, 218, 221-225, 227, 229, 238

The remaining four bytes are used to specify the address of a user written error handler. If the value is different from 00000001, when the error occurs control will transfer to the specified address. If the value is 00000001, a "standard fixup" will be applied in order to continue processing.

MODIFYING THE OPTION TABLE ENTRIES

ERRSAV

The ERRSAV subroutine illustrated above has two arguments:

CALL ERRSAV(*msgno,dpvar*)

where *msgno* is the message number of interest, and *dpvar* is a double precision variable. The option table entry for *msgno* is copied to the eight byte variable *dpvar*.

ERRSTR

The ERRSTR subroutine is the complement of ERRSAV:

CALL ERRSTR(*msgno,dpvar*)

where *msgno* is the message number of interest, and *dpvar* is a double precision variable. The eight byte variable *dpvar* is copied to the option table entry for *msgno* if permitted by bit 1 of the flag byte presently in the option table entry.

One interesting use of ERRSAV and ERRSTR is to first use ERRSAV to capture the option table entry for a particular error, turn off bit 2 of the flag byte, set the third byte of the double word to zero, and then use ERRSTR to save it back into the table. If this is done immediately before execution terminates, the error will not show up in the summary usually printed after program execution.

ERRSET

You can modify the option table entry for a particular error by using ERRSAV to get a copy of the entry, modify your copy, and use ERRSTR to store it back into the table. However, if you wish to supply a user subroutine, you would have to know its entry point. In addition, much of the manipulation requires single byte arithmetic, something that is not easily done in Fortran. A simpler way to modify an entry in the option table is to use the ERRSET routine. This routine can be used in conjunction with ERRSAV, and ERRSTR to temporarily modify the table, or it can be used alone to modify an entry for the entire execution.

In an attempt to make ERRSET easy to use, the convention was adopted that if a particular call argument was zero, or omitted from the end of the parameter list, the option table entry corresponding to that argument should remain unchanged. While nice in principle, this convention makes it difficult to set a value to zero. This problem was overcome by adopting another convention -- giving a parameter an impossible value will (usually) cause it to be set to zero.

The calling sequence to ERRSET is:

CALL ERRSET(*ierno,inoal,inomes,itrace,iusadr,irange*)

ierno — the number of the error message whose option table entry is to be modified.

inoal — number of times to tolerate the error. This value is used to set the first byte of the option table entry. If the value of the first byte of the option table entry is zero, it means to tolerate the error forever. If you specify a zero or a negative integer for this entry, it means to leave the option table entry unchanged. If you specify a value greater than 255, it means to set the first byte of the option table entry to zero, thus permitting the error to occur an unlimited number of times. (256 => infinite)

inomes — the number of times the error message is to be printed. If you specify a value of zero, the table entry is unchanged. If you specify a negative number, the table entry is set to zero and no messages are printed. If you specify a number greater than 255, bit 5 of the flag byte is set and an unlimited number of messages are printed. (256 => infinite, −1 => never print a message)

itrace — modifies bit 6 of the flag byte. If you specify a value of 0, the bit is unchanged. If you specify a value of 1, the flag is cleared, and no traceback is printed. If you specify a value of 2, the flag is set, and a traceback will be printed.

iusadr — modifies the last four bytes of the option table entry. If the value is zero, the table entry is unchanged. If the value specified is a subprogram name (which must appear on an EXTERNAL statement in the source program), the last word of the table entry is set to the address of the entry point of the subprogram. If the value is 1, the last word of the table entry is set to 00000001, indicating that there is no user supplied error handler.

irange — if this value is numerically higher than the one specified in the first argument, it means that the option table modifications are to be applied to all table entries from ierno through irange. If ierno is 212, then this value (0 or 1) specifies the value to give to bit 0 of the flag byte.

Error Number 212. A 212 error is a FORMATTED I/O, END OF RECORD. If the error occurs during a WRITE, a new output record is started. If the standard option is used, this new record will not have a carriage control character. By setting bit 0 of the option table entry for 212 to a 1, you cause the new record to have the carriage control character corresponding to single spacing.

An example of ERRSET is given in Figure 4 page 10.

ADDING YOUR OWN OPTION TABLE ENTRIES

Appendix A shows the source program I use to generate a custom version of IFYUOPT. I.B.M. supplies a macro for this purpose, however I prefer this simpler and more straightforward version.

In CMS, if the text file IFYUOPT TEXT exists on an accessed disk, the local copy will be chosen in preference to the one in the Fortran library. Note that you can define a V type constant as the address of a fixup routine, but then this routine will be loaded whenever the local version of IFYUOPT is used.

USER GENERATED MESSAGES

ERRTRA

The ERRTRA subroutine is the simplest of the five routines supplied by I.B.M. It requires no arguments, and its function is to provide a traceback telling where you are now, and how you got there. If you have included user written assembler routines in the executable module, it is important to have followed the linkage conventions as described in *VS FORTRAN Application Programming: Guide SC26-3985.* You may insert a CALL ERRTRA within any source subprogram, provided that the subprogram does not have ERRTRA as an antecedent, since Fortran does not provide for recursion.

I found ERRTRA useful in the following situation. Our installation supports a very large interactive optical design and evaluation program. Originally developed using TSO in a small region, it seemed a good idea at the time to have a single subroutine whose function was to print error messages to the user. These messages were collected into the routine PRERR which accepted an integer CALL argument, K, which determined which message should be output. As the other routines in the package were executed, a value for K was calculated, and eventually passed to PRERR. If K was zero, the PRERR routine returned without producing any message.

Eventually the program grew to where overlays were required. At this point it seemed exceptionally silly to load in an overlay which frequently did nothing. So one by one, the routines were modified to include the statements:

```
IF (K .NE. 0) CALL PRERR(K)
```

so that PRERR would only be called if needed.

Since the subroutines numbered in the hundreds, we were never sure that all of them were fixed. So we inserted the following statements into PRERR:

```
      SUBROUTINE PRERR(K)
      IF (K .NE. 0) GO TO 10
      CALL ERRTRA
      RETURN
10    GO TO (20,30,.......),K
C     TO PRINT THE APPROPRIATE ERROR MESSAGE
```

This code causes a traceback whenever PRERR is entered unnecessarily; the message is frightening enough to make the users quickly report it.

In release 3.0 of VS Fortran, all messages, including the traceback, have been significantly improved over those given in previous releases or products. The traceback now includes the load address of each subprogram in the chain, the offset of the calling instruction, the location of the start of each parameter list, and the value of the first word of each argument passed; interpreted in hexadecimal, integer, and character. Internal statement numbers are displayed if the source program was compiled using the GOSTMT option.

ERRMON

ERRMON is probably the least understood of the five routines supplied by I.B.M. I shall attempt to explain it through the following example.

Suppose I am requested to provide a subroutine, MATH, which accepts four parameters, A, B, C, and N. My task is to calculate C from A and B. N controls the operation to be performed. If N is 1,2,3, or 4, I should add, subtract, multiply, or divide accordingly. What shall I do if N has an value other than 1,2,3, or 4? I will define and document the "standard corrective action". In this example the standard fixup will be to act as if N were 1, and add. I will also alert the user by printing message 302.

The first step is to add an entry to IFYUOPT for error 302. The sample source program for IFYUOPT provided in Appendix A includes this entry.

I now code the program shown in Figure 2, page 6.

Now suppose I have a user who wishes a different fixup. Say, for example, that he would like the corrective operation to be a little more complex. If the illegal value for N is 7, he wishes to multiply, but for any other illegal value he will accept my corrective action. In that case he may call ERRSET to supply a user routine to apply his action. The code he would write is given in Figure 3, page 7.

If you choose to use ERRMON, then error toleration, message printing, counting number of occurrences, and the address of the user processing routine will all be controlled by the option table entry. You, of course, must supply ERRMON with the error message text to be used, as well as the parameters to be passed to the user error handler.


```
      SUBROUTINE MATH(A,B,C,N)
      CHARACTER * 28 MSG
      EQUIVALENCE (MSG,MGL)
      DATA MSG/'    WPH302I ILLEGAL OP CODE' /
      MGL = 24
      K = N
1     IF (K .LT. 1 .OR. K .GT. 4) THEN
         CALL ERRMON(MSG,IFIX,302,K)
         IF (IFIX .EQ. 0) K = 1
         GO TO 1
      ENDIF
      GO TO (2,3,4,5),K
2     C = A + B
      RETURN
3     C = A - B
      RETURN
4     C = A * B
      RETURN
5     C = A / B
      END
```

Figure 2. Example of ERRMON

The calling sequence to ERRMON is:

CALL ERRMON(*imes,iretcd,ierno* [*,data1,data2,...*])

imes a count of the number of characters in the message to be printed (in integer *4), followed by the character string, see Figure 2.

iretcd a return code from the error handler. If no user exit has been supplied to the option table entry, a zero will be returned, and standard correction should be applied. If the user has modified the option table entry to indicate that he has supplied a correction routine, he will be passed all of the parameters which were originally passed to ERRMON, except *imes.* The user indicates that he wishes no further corrective action by setting the first parameter (*iretcd*) to 1, or that he wishes standard corrective action applied also, by setting the first parameter (*iretcd*) to 0. (See *VS Fortran Compiler and Library: Diagnosis SC26-3990* Errors detected by the Library or a User Program).

ierno the error message number

data1... these parameters are optional. The user correction routine will be passed all of the parameters which were originally passed to ERRMON, except *imes.* Therefore he will receive *iretcd,ierno,data1...* The usual caveats apply to the mode of parameters, and the dangers which accompany passing constants which may be altered.

EXAMPLES

This section exhibits several examples using the routines described above, as well as demonstrating user corrective routines.

SUPPLYING A USER CORRECTIVE ROUTINE

In the ERRMON example given in Figure 2 on page 6 we suppose that the user wants to multiply rather than add should an invalid value of 7 be given for N. This is accomplished as follows:

```
      EXTERNAL FIXIT
      CALL ERRSET(302,0,0,0,FIXIT)
      CALL MATH (2.,3.,C,7)
      WRITE(6,*)C
      END
      SUBROUTINE FIXIT(IRT,MSGNO,K)
C
C     PREPARE FOR STANDARD FIXUP
C
      IRT = 0
      IF (K .NE. 7) RETURN
C
C     IT'S 7 I'LL FIX IT
C
      IRT = 1
      K = 3
      END
```

Figure 3. A Main Program and User Exit

In the main program, the user calls ERRSET to supply the name of the user fixup routine for error 302. Notice that this name must be specified on an EXTERNAL statement, otherwise Fortran will assume FIXIT is the name of a variable, and will send its (undefined) value (probably 0) to ERRMON, which will assume it is receiving an address, or that the entry table should remain unchanged. This is a tough bug to find no matter what symptoms you have. Since the value of N is out of range, MATH will call ERRMON with four parameters, which will in turn call FIXIT with three parameters, *iretcd*, *ierno*, and K. MATH has protected the original value of N by copying it into K so that the user's constant will not be altered.

The program sets the return code to zero so that if N is other than 7, MATH will apply the standard corrective action. However, if N is 7, FIXIT makes the return code 1 to avoid the standard corrective action, and then sets K to 3 in order to specify multiplication. MATH, as written, will again check the value to see if user correction has made it legal; if not, ERRMON

is called again. Without a tolerance threshold, this could result in an endless loop. I.B.M. routines behave in a similar manner.

THE WAITER PROGRAM

This sample program is called WAITER for two reasons. First, the purpose of the program is to perform service as files appear in the CMS virtual card reader. Second, in our environment, files don't appear in the reader very frequently, so most of the time the virtual machine is in the wait state.

The READ statement at statement 1 attempts to read a record from the card reader. When the card reader is empty, a 218 error will occur causing control to pass to statement 3. ERRSET has been invoked to tolerate a 218 error for an unlimited number of times, and to suppress all messages.

The REWIND statements prevent Fortran from trying to read from FILE FT01F002. The "processing" in this sample is simply to write a copy of the input record to file 6, but could be replaced with any code desired.

...There's a fly in my soup

Control passes to statement 3 when the reader is empty. In this sample I print a message, and call the assembler routine HOTRDR, which executes a WAITD macro, waiting for an interrupt. When the interrupt happens, we return to the Fortran program in order to process the new file.

```
      CHARACTER * 80 A
C
C                    ERRNO    TOLERATE      NO         NO
C                             INFINITE   MESSAGES   TRACEBACK
      CALL ERRSET(  218,       256,         -1,         1        )
C     SINCE THE LAST TWO ARGUMENTS HAVE BEEN OMITTED
C     THEIR CURRENT VALUES ARE UNCHANGED.
C
1     READ(1,5,END=2,ERR=3)A
      WRITE(6,6)A
      GO TO 1
C
C     HERE IF END OF FILE
C
2     REWIND 1
      GO TO 1
C
C     HERE WHEN READER IS EMPTY
C
3     WRITE(6,4)
4     FORMAT(' WAITING FOR READER' )
      REWIND 1
      CALL HOTRDR
      GO TO 1
C
5     FORMAT(A)
6     FORMAT(1X,A)
      END
**********************************************************************
*
*
*                         GIVE USER A HOT READER
*
*
**********************************************************************
HOTRDR   CSECT
         SAVE      (14,12),,*              SAVE REGISTERS
         LR        R10,R15                 EST ADDRESSABILITY
         USING     HOTRDR,R10              TELL THE ASSEMBLER
*
         WAITD     RDR1                    WAIT UNTIL INTERRUPT
*
*        CONTINUE WHEN AN INTERRUPT HAPPENS
*
         RETURN    (14,12),T,RC=0          RESTORE REGS AND EXIT
*
         REGEQU
         end
```

Figure 4. A program to process files as they appear in the reader.

SOLUTION OF A FORMAT CONVERSION PROBLEM

This final example illustrates how we are able to submit jobs which produce plots to a remote computer. We then wish to plot the output from those jobs locally. In this example, the host computer has a peculiar convention which we are forced to accommodate; all of the output returned to us is in lower case.

The programs which we send have standard calls to plot subroutines. We also send to the remote machine fake "plot" subroutines which do not plot, but rather punch card images containing the parameters given at the time of the plot calls. When we receive the cards back, they are read and interpreted locally. At this time we call the real plot subroutines, and produce the plots at our site.

The "plot" decks appear in the submitting user's virtual card reader, and are usually very large. We simply FILEDEF a unit to the reader, and process the input records as they are read. In this way we do not have to give the users huge disks, since the files never reside on their disks.

The program which reads and processes the plot deck is written in Fortran. Reading cards with alphabetic characters poses no problem, but reading floating point numbers in 'E' format when the 'E' is in lower case causes a 215 format conversion error. The solution was to use the extended error handling routines to provide a user correction.

The main program calls ERRSET to set the tolerance of 215 type errors to an unlimited number, and to give the entry point of the user corrective routine ESUB. A call to ERRSAV is done so that error occurrence counting can later be modified to give a count of "true" errors, i.e. those errors caused by other than encountering a lower case 'e' in a floating point number.

We now follow in detail what happens when the main program attempts to read such a card from the virtual reader. VLDIO# is called, which in turn calls IFYVCVTH who gets mad when he finds a lower case 'e'.

So IFYVCVTH calls ERRMON with the 215 error message, a variable for the return code, the message number (215), and the address of the place in the buffer which contains the illegal character.

ERRMON checks the option table entry for 215 and finds that the job can continue regardless of the number of times the error occurs, and that the user has provided a correction routine, ESUB. ERRMON then calls ESUB with all the parameters it received from IFYVCVTH except the first.

ESUB checks the offending character, and if it is not a lower case 'e', sets the return code to zero, increments its local error counter, and returns to IFYVCVTH which applies standard corrective action.

If, however, ESUB finds that the buffer contains a lower case 'e' the buffer contents are replaced with an upper case 'E', the return code is set to one, the local error counter is not incremented, and control is returned to IFYVCVTH. IFYVCVTH does not apply standard corrective service, but simply continues the format conversion using the substituted character.

When the main program senses an end of file from the reader it calls ECNT, an entry point in ESUB. The true error count maintained in ESUB is inserted into the copy of the option table entry for 215 that the main program has been saving. This copy, via ERRSTR, is now stored into the option table so that when the summary of errors is printed, only "true" errors will be reported.

```
        IMPLICIT REAL*8 (A-H,O-Z)
        EXTERNAL ESUB
C       REAL * 8 OPT215
C       THIS PROGRAM PLOTS METAFILES
C
        CALL ERRSAV(215,OPT215)
        CALL ERRSET(215,256,-1,1,ESUB)
C
3       READ (5,1,END=101) ISUB,INT,F
1       FORMAT(I1,....)
C
        GO TO (10,20,30,40,50),ISUB
        WRITE(6,100)
100     FORMAT(' ILLEGAL METAPLOT CARD')
C
C       NOW TO GET THE TRUE ERROR COUNT
C
101     CALL ECNT(OPT215)
        CALL ERRSTR(215,OPT215)
C
        STOP
C
10      CALL ....
        GO TO 3
20      ...
        ...
        GO TO 3
30
        ...
        END
```

Figure 5. The PLOTMETA Main Program


```
*********************************************************************
*
*         THE ESUB PROGRAM WILL CONVERT A LOWER CASE E
*         TO UPPER, AND MAINTAIN A LOCAL ERROR COUNT
*
*         P1 = ADD OF RETURN CODE FIELD (*4)
*               IF I MAKE IT 0, => STAND FIXUP
*               IF I MAKE IT 1, => NO STAND FIXUP
*         P2 = ADD OF ERROR NUMBER (*4)
*         P3 = ADD OF INVALID CHAR (*1)
*
*         THE ENTRY POINT ECNT RETURNS THE LOCAL VALUE OF
*         ERROR COUNT INTO BYTE 2 OF THE DOUBLE WORD ARGUMENT.
*
*********************************************************************
ESUB     CSECT
         ENTRY    ECNT
         USING    *,R15
         SAVE     (14,12),,*
         LM       R2,R4,0(R1)         R2 = ADD RET CD, R4= ADD BAD CHR
         CLI      0(R4),C'E'-X'40'    IS P3 A LOWER CASE 'E'?
         BE       FIXIT               YES - SUBSTITUTE A CAP E
STFIX    L        R5,ERRCNT           LOCAL ERROR COUNT
         CH       R5,=H'255'          IS ERRCNT 255?
         BNE      STFIX1              NO - INCREMENT LOCAL COUNT
         TM       ERRFLG,X'20'
         BO       STFIX3              COUNT = 511
         MVI      ERRFLG,X'20'        SET BIT 2--256'S PLACE
         LA       R5,0                RESET LOW ORDER BYTE
         B        STFIX2
STFIX1   LA       R5,1(,R5)           INCREMENT ERROR COUNT
STFIX2   ST       R5,ERRCNT           SAVE IT
STFIX3   LA       R5,0                CLEAR R3
         ST       R5,0(R2)            SET P1 = 0 ( => TAKE STAND FIXUP)
         B        DONE                RETURN
FIXIT    MVI      0(R4),C'E'          REPLACE WITH UPPER E
         LA       R5,1                SET R3 = 1
         ST       R5,0(R2)            SET P1 = 1 ( => AVOID STAND FIXUP)
DONE     RETURN   (14,12),T,RC=0
         USING    *,R15
ECNT     SAVE     (14,12),,ECNT
         L        R1,0(,R1)           R1 = ADD OF P1 (A DOUBLE WORD)
         L        R2,ERRCNT           PUT MY ERRCNT INTO BYTE 2 OF
         STC      R2,2(R1)            HIS DOUBLE WORD
         OC       3(1,R1),ERRFLG      HIS FLAG BYTE
         RETURN   (14,12),T,RC=0
ERRCNT   DC       1F'0'
ERRFLG   DC       XL1'00'
         REGEQU
         END
```

Figure 6. The ESUB Subroutine

APPENDIX A
SOURCE PROGRAM FOR IFYUOPT

```
IFYUOPT  CSECT
NUMENT   DC    AL4((EOT-BOT)/8)          NUM OF TABLE ENTRIES
FIRSTERR DC    AL4(140)                  ERR MSG NUM 1ST ERR
*
BOT      EQU   *                         BEGINNING OF TABLE
*
*                                         #     #    DMP   USR   STD   TRC
*                                        TOL   MSG   BUF   MOD   FIX   BAK
*
ERR140   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR141   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR142   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR143   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR144   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR145   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR146   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR147   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR148   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR149   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR150   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR151   DC    XL8’0000004C00000001’     --    --     N     Y     N     N
ERR152   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR153   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR154   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR155   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR156   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR157   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR158   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR159   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR160   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR161   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR162   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR163   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR164   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR165   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR166   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR167   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR168   DC    XL8’0101000200000001’      1     1     N     N     -     Y
ERR169   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR170   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR171   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR172   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR173   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
```



```
ERR174   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR175   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR176   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR177   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR178   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR179   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR180   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR181   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR182   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR183   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR184   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR185   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR186   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR187   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR188   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR189   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR190   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR191   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR192   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR193   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR194   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR195   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR196   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR197   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR198   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR199   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR200   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR201   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR202   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR203   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR204   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR205   DC    XL8’0101000000000001’      1     1     N     N     -     N
ERR206   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR207   DC    XL8’0005004200000001’     --     5     N     Y     Y     Y
ERR208   DC    XL8’0005004200000001’     --     5     N     Y     Y     Y
ERR209   DC    XL8’0005004200000001’     --     5     N     Y     Y     Y
ERR210   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR211   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR212   DC    XL8’0A05005200000001’     10     5     Y     Y     Y     Y
ERR213   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR214   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR215   DC    XL8’0005005200000001’     --     5     Y     Y     Y     Y
ERR216   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR217   DC    XL8’0101004200000001’      1     1     N     Y     Y     Y
ERR218   DC    XL8’0A05005200000001’     10     5     Y     Y     Y     Y
ERR219   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
ERR220   DC    XL8’0A05004200000001’     10     5     N     Y     Y     Y
```

```
ERR221   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR222   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR223   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR224   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR225   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR226   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR227   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR228   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR229   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR230   DC    XL8'0101000200000001'    1   1   N   N   -   Y
ERR231   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR232   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR233   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR234   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR235   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR236   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR237   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR238   DC    XL8'0A05005200000001'   10   5   Y   Y   Y   Y
ERR239   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR240   DC    XL8'0101000200000001'    1   1   N   N   -   Y
ERR241   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR242   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR243   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR244   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR245   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR246   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR247   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR248   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR249   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR250   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR251   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR252   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR253   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR254   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR255   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR256   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR257   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR258   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR259   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR260   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR261   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR262   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR263   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR264   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR265   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR266   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR267   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
```

132


```
ERR268   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR269   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR270   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR271   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR272   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR273   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR274   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR275   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR276   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR277   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR278   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR279   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR280   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR281   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR282   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR283   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR284   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR285   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR286   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR287   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR288   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR289   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR290   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR291   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR292   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR293   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR294   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR295   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR296   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR297   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR298   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR299   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR300   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
ERR301   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
*
*        PUT LOCAL ENTRIES HERE--302 - 899
*
ERR302   DC    XL8'0A05004200000001'   10   5   N   Y   Y   Y
EOT      EQU   *                            END OF TABLE
         END
```