

B S L LIBRARY

F. M. Bonevento

Systems Development Division

Poughkeepsie, New York

October 27, 1967

IBM Confidential

The BSL/10 Supplement to the BSL Library Manual is bound in the back of this document

TABLE OF CONTENTS

	<u>Page No.</u>
I. <u>INTRODUCTION</u>	4
II. <u>BSL OBJECT TIME INPUT/OUTPUT</u>	5
1. Function	5
2. Entry points and functions	5
A. READ	5
B. PRINT	5
C. PUNCH	6
D. CLOSE	6
3. Error conditions detected by BSL Object Time Input/Output	6
4. Sample use of BSL Object Time Input/Output	8
5. Program listing of BSL Object time Input/Output Routine	9
III. <u>PDUMP</u>	19
1. Function	19
2. How to use PDUMP	19
3. Output format produced by PDUMP	20
4. Error conditions detected by PDUMP	20
5. Sample use of PDUMP	21
6. Program listing of PDUMP	22
IV. <u>SUBSTR</u> (SUBSTRING)	32
1. Function	32
2. How to use SUBSTR	32
3. Error conditions detected by SUBSTR	33
4. Sample use of SUBSTR	35
5. Program listing of SUBSTR	37

V.	ERRINT (ERROR-HANDLER)	44
1.	Function	44
2.	How to use the ERROR-HANDLER	44
3.	Errors detected by the ERROR-HANDLER	50
4.	Sample use of the ERROR-HANDLER	51
5.	Program listing of ERRINT	55
VI.	<u>EDIT</u>	62
1.	Function	62
2.	Entry points and functions	62
3.	A. GET	62
	B. GETS	62
	C. PUT	62
	D. PUTS	63
	E. TABSET	63
3.	FORMAT Description	63
	A. Free form format	63
	B. Control format items	64
	C. Replication format items	66
	D. Data format items	68
4.	Coded format lists	71
5.	Restrictions and limitations	74
6.	Error conditions detected by EDIT	75
7.	Program listing of EDIT	77
VII.	APPENDIX I	99
VIII.	INTERRUPT HANDLER (BSL/10 Supplement to BSL/10 Library Manual)	101
1.	Function	102
2.	Entry points and functions	103
	A. IKETRCII	103
	B. IKERRCID	103
3.	Example	104

I. INTRODUCTION

This document contains detailed descriptions and source listings of the programs which reside in the partitioned data set BSLLIB. These programs provide certain basic functions which the BSL programmer will find useful when testing his programs under OS/360.

The BSL programmer uses these functions by invoking them in a BSL CALL statement. The library programs he requests are automatically included in the object program during the linkage editing process, when the user employs the cataloged procedure BSLALG. (If the user does not employ BSLALG, he should provide the LINKAGE EDITOR with a SYSLIB DD card for the BSLLIB when he linkage edits his program.)

The facilities provided by these library programs should not be regarded as part of the BSL language. The output of the BSL compiler is independent of any particular operating environment, but the library programs will function only under Operating System/360. They are provided as a convenience for BSL users who work under OS/360, but their use is purely optional, since they are invoked only by means of the BSL CALL statement. The user is at liberty to use other means to obtain the same services, such as macros within GENERATE statements.

II. BSL OBJECT TIME INPUT/OUTPUT

1. Function

This program employs the OS/360 QSAM access method to provide basic input/output services. The user requests these services by invoking the module using the entry name associated with the function he desires, i.e. (READ, PRINT, PUNCH, CLOSE).

2. Entry points and functions

There are four entry points to the module which perform the following functions:

A. READ

This entry point will cause a logical record to be read into the area which the user passed as an argument. Data is read from the data set with the DD name BSLIN. The user may specify DCB parameters for RECFM, LRECL, BLKSIZE, and BUFNO on the DD card for BSLIN. Values needed for these parameters are set using the following defaults:

RECFM=F, LRECL=80, BLKSIZE=80, BUFNO=3.

These default values will cause an 80 byte record to be read into the area specified by the user. When using the catalogued procedure BSLALG, the user should use the DDNAME GO.BSLIN to refer to this data set.

B. PRINT

This entry point will cause a logical record to be written out from the area which the user passed as an argument. Data is written out on the data set with the DD name BSLOUT. The user may specify DCB parameters for RECFM, LRECL, BLKSIZE, and BUFNO on the DD card for BSLOUT. Values needed for these parameters are set using the following defaults.

RECFM=FA, LRECL=121, BLKSIZE=121, BUFNO=3.

These default values will cause a 121 character record to be written out with the assumption that

the first character is an ASA carriage control character. When the user employs the catalogued procedure BSLALG, he need not supply this DD statement. Note: the default values for BSLOUT will be used.

C. PUNCH

This entry point will cause a logical record to be written out from the area which the user passed as an argument. Data is written out on the data set with the DD name BSLPUNCH. The user may specify DCB parameters for RECFM, LRECL, BLKSIZE, and BUFNO on the DD card for BSLPUNCH. Values needed for these parameters are set using the following defaults:

RECFM=F, LRECL=80, BLKSIZE=80, BUFNO=3.

These default values will cause an 80 byte record to be written out from the area which the user passed as an argument. When the user employs the catalogued procedure BSLALG, he need not supply this DD statement. Note: the default values for BSLPUNCH will be used.

D. CLOSE

This entry point will close the data set with the DD name BSLOUT.

3. Error conditions detected by BSL Object Time Input/Output

This module detects a number of erroneous conditions. When such conditions are detected, the following actions are taken:

- A. An error code is set in the variable EOF. This code can be interrogated in the users program by accessing EOF, (a variable which he has declared FIXED(31) NONLOCAL).
- B. A diagnostic message is written on BSLOUT. If this data set cannot be opened, the message is displayed on the system console. Control is returned to the caller.

The following table summarizes the actions taken by the module:

Condition	ECF Value	Message
End of file on BSLIN	1	None.
No DD statement for BSLIN	2	Message on BSLOUT. No DD statement for BSLIN. It cannot be opened.
No DD statements for BSLIN and BSLOUT	8	Message on console. No DD statement for BSLOUT and BSLIN. They cannot be opened.
I/O error on BSLIN	16	Message on BSLOUT. Error on BSLIN. Data is not transmitted.
I/O error on BSLIN and no DD for BSLOUT	64	Message on Console. No DD statement for BSLOUT and error on BSLIN.
No DD statement for BSLPUNCH	3	Message on BSLOUT. No DD statement for BSLPUNCH. It cannot be opened.
No DD statements for BSLPUNCH and BSLOUT	12	Message on Console. No DD statement for BSLOUT and BSLPUNCH. They cannot be opened.
I/O error on BSLPUNCH	24	Message on BSLOUT. Error on BSLPUNCH. Data is not transmitted.
I/O error on BSLPUNCH and no DD for BSLOUT	96	Message on Console. No DD statement for BSLOUT and error on BSLPUNCH.
No DD statement for BSLOUT	4	Message on Console. No DD statement for BSLOUT.
I/O error on BSLOUT	20	Message on BSLOUT. Error on BSLOUT. Data is not transmitted.

4. Sample use of BSL I/O

This program will reproduce 10 cards and list them starting at the top of a page, with a blank line between each card.

```

REPRO: PROC;
  DCL OUTLINE CHAR(121); /*OUTPUT LINE*/
  DCL INCARD CHAR(80) BASED(ADDR(OUTLINE)+1);
  /*INPUT CARD IMAGE */
  DCL PUNCHOUT CHAR(80) BASED(ADDR(OUTLINE)+1);
  /* PUNCH IMAGE */
  /* NOTICE THAT INCARD AND PUNCHOUT ARE OVERLAYED */
  /* ON OUTLINE */
  OUTLINE = '1'; /* SET CARRIAGE CONTROL TO PAGE */
                /* AND BLANK BUFFER */

  TIME = 1;
  DO I=1 TO 10 BY 1;
  IF TIME = 1 THEN OUTLINE(1)='0'; /* SET CARRIAGE */
  /* CONTROL TO SKIP A LINE */
  ELSE TIME=2;
    CALL READ(INCARD); /* READ A CARD */
  DCL EOF FIXED(31) NONLOCAL;
    IF EOF=16 THEN RETURN;
    /* TEST FOR I/O ERROR ON BSLIN */
    CALL PRINT(OUTLINE); /*PRINT A LINE */
    IF EOF = 20 THEN RETURN;
  /* TEST FOR I/O ERROR ON BSLOUT */
    CALL PUNCH(PUNCHOUT);
    IF EOF=24 THEN RETURN;
  /* TEST FOR I/O ERROR ON BSLPUNCH */
  END;
END REPRO;

```



```
0001     READ=PROC(PARM1);
0002     /* ENTRY POINT TO READ A CARD*/
          DCL(R3 REG(3),R1 REG(1),R0 REG(0),V REG(4) )PTR(31);
          ** W05 ** POSSIBLE CONFLICT IN USE OF REGISTER
          ** W05 ** POSSIBLE CONFLICT IN USE OF REGISTER
0003     DCL(P1 REG(6),P2 REG(7),M REG(5)) PTR(31);
0004     DCL EDF FIXED(31) LOCAL EXTERNAL;
0005     DCL (CARDIN,LINOUT,CRDOUT)CHAR(92) NONLOCAL INTERNAL;
0006     DCL(O1,O2,O3) LABEL NONLOCAL INTERNAL;
0007     DCL OFLAGS BIT(8) BASED(P1);
0008     RESTRICT(3,4,5,6,7);
0009     M=2;
0010     P1=ADDR(CARDIN)+48;
0011     P2=ADDR(O1); /* POINT TO ARG LIST TO OPEN BSLIN*/
0012     GO TO COMMON;
```

```
0013 PRINT:ENTRY(PARM1);
0014 /* ENTRY TO PRINT A LINE*/
      M=4;
0015 P1=ADDR(LINOUT)+48;
0016 P2=ADDR(03);/*POINT TO ARG LIST TO OPEN BSLOUT*/
0017 GO TO COMMON;
```

```
0018 PUNCH:ENTRY(PARM1);
0019 /* ENTRY TO PUNCH A CARD*/
      P1=ADDR(CRDOUT)+48;
0020 M=3;
0021 P2=ADDR(Q2);/*POINT TO ARG LIST TO OPEN BSLPUNCH*/
0022 GO TO COMMON;
```

```

0023 CLOSE:ENTRY;
0024 /* ENTRY TO CLOSE BSLOUT*/
      GEN(CLOSE (LINDOUT)); /* CLOSE PRINT FILE*/
0025 RETURN;
0026 CCOMMON:EOF=0;
0027 SWITCH=0;
0028 V=M;
0029 IF M=4 THEN PRS=4;
0031 ELSE PRS=0;
0032 /* SET PRINT SWITCH FOR OPEN EXIT*/
      R3=ADDR(PARM1);
0033 L1: IF OFLAGS(4)='1'B THEN GO TO OUT1;
0035 R1=P2; /* SET UP OPEN*/
0036 GEN(SVC 19);/*ISSUE OPEN SVC*/
0037 DCL MESS CHAR(121) INIT('IND OD STATEMENT FOR BSL IT CANNOT BE
OPENED');
0038 TEST:IF OFLAGS(4)='0'B THEN IF V=4 THEN GO TO NODD1;
0041 ELSE GO TO ERR6;
0042 OUT1: IF SWITCH=1 THEN GO TO BACK4;
0044 RO=R3;
0045 ERROUT:R1=P1-48; /* POINT TO DCB*/
0046 GEN(L 15,48(0,1)); /* POINT TO I/O ROUTINE*/
0047 GEN(BALR 14,15);
0048 RETURN; /* GO BACK TO CALLER*/
0049 BACK4: EOF=V; /* SET RETURN CODE*/
0050 IF V=2 THEN DO;
0052 MESS(25:29)='IN ';/*SET UP MESSAGE*/
0053 X1:RO=ADDR(MESS); /*FOR BSLIN MISSING DD*/
0054 GO TO ERROUT;
0055 END;
0056 IF V=3 THEN DO;
0058 MESS(25:29)='PUNCH';
0059 /* SET UP MESSAGE FOR MISSING DD FOR BSL PUNCH*/
      GC TO X1;
0060 END;
0061 DCL SYMESS CHAR(121) INIT('LEKRRR ON BSL DATA IS NOT TRANSMITT
ED');
0062 IF V=20 THEN DO;
0064 SYMESS(14:18)='OUT ';/*SYNCHRONOUS ERROR*/
0065 X2:RO=ADDR(SYMESS); /* ON BSLOUT*/
0066 GO TO ERROUT;
0067 END;
0068 IF V=16 THEN DO; /*SYNCHRONOUS ERROR ON BSLIN*/
0070 SYMESS(14:18)='IN ';
0071 GO TO X2;
0072 END;
0073 IF V=24 THEN DO;/*SYNCHRONOUS ERROR ON BSLPUNCH*/
0075 SYMESS(14:18)='PUNCH';
0076 GO TO X2;
0077 END;

```

```

0078      /* ROUTINE TO WRITE ON CONSOLE*/
          ERR5:   EOF=V*4;
0079      DCL BAD LABEL NONLCCAL INTERNAL;
0080          DCL      BADN CHAR(34) BASED(ADDR(BAD)+36);
0081          IF V=2 THEN DO; /*NO DD FOR BSLOUT AND IN*/
0083              BADN='AND BSLIN. THEY CANNOT BE OPENED';
0084              GO TO BAD;
0085              END;
0086          IF V=3 THEN DO; /* NO DD BSLOUT AND PUNCH*/
0088              BADN='AND BSLPUNCH.THEY CANNOT BE OPENED';
0089              GO TO BAD;
0090              END;
0091          IF V=16 THEN DO;
0093              BADN='AND ERROR ON BSLIN';
0094              GO TO BAD;
0095              END;
0096          IF V=24 THEN DO;
0098              BADN='AND ERROR ON BSLPUNCH';
0099              GO TO BAD;
0100              END;
0101      NODD1:   BADN= ' ';
0102      GEN;
BAD      WTO   'NO DD STATEMENT FOR BSLOUT      X

$ENDGEN
0103      RETURN;
0104      ABEND:  EOF=1;
0105      RETURN;      /* SET END OF FILE ON BSLIN*/
0106      READERR: V=16;
0107      GO TO XXX;
0108      PRNTERR: V=20;
0109      GO TO XXX;
0110      PNCERR:  V=24;
0111      GO TO XXX;
0112      ERR6:   IF SWITCH=0 THEN DO;
0114          XXX:      SWITCH=1;
0115          P1=ADDR(LINOUT)+48;
0116          P2=ADDR(O3);
0117          PRS=4;
0118      /* SET PRINT SWITCH FOR OPEN EXIT*/
          GO TO L1;

0119          END;
0120          ELSE DO;
0121              SWITCH=0 ;
0122              GO TO ERR5;
0123          END;
0124      GEN;
CARDIN   DCB   DSORG=PS,MACRF=(GM),DDNAME=BSLIN,BFTEK=S,BFALN=F,      X
          EDDAD=ABEND,EXLST=OPERR,SYNAD=READERR,EROPT=ACC
LINOUT   DCB   DSORG=PS,MACRF=(PM),DDNAME=BSLOUT,BFTEK=S,BFALN=F,      X
          EXLST=OPERR,SYNAD=PRNTERR,EROPT=ACC
CRDOOUT  DCB   DSORG=PS,MACRF=(PM),DDNAME=BSLPUNCH,BFTEK=S,BFALN=F,      X
          EXLST=OPERR,SYNAD=PNCERR,EROPT=ACC
O1       OPEN  (CARDIN,(INPUT,REREAD)),MF=L
O2       OPEN  (CRDOOUT,(OUTPUT,REREAD)),MF=L

```

```
03      OPEN  (LINCUT,(OUTPUT,REREAD)),MF=L
        DS    OF
OPERR   DC    1X'85'
        DC    AL3(CEXIT)
$ENDGEN
```

```

0125  /* THIS PROCEDURE SUPPLIES DCB PARAMETERS WHEN THEY ARE NOT
        SUPPLIED BY THE CALLER*/
        OEXIT:PROC;
0126      DCL DCBRECFCM BIT(8) BASED(R1+36);
0127      DCL DCBBUFNO PTR(8) BASED(R1+20);
0128      DCL DCBBLKSI FIXED(15) BASED(R1+62);
0129      DCL DCBLRECL FIXED(15) BASED(R1+82);
0130      DCL 1 POOLD FIXED(31),
           2 VBUFNO FIXED(15),
           2 VLRECL FIXED(15);
0131      IF DCBRECFCM(1:2)='00'B THEN DO;
0133          DCBRECFCM='10111111'B&DCBRECFCM;
0134          DCBRECFCM='10000000'B|DCBRECFCM;
0135      /* RECFM IS SET TO FIXED*/
        IF PRS=4 THEN DO;
0137      /* IF PRINT FILE THEN MAKE IT HAVE ASA CNTRL CHAR*/
          DCBRECFCM='11111101'B&DCBRECFCM;
          DCBRECFCM='00000100'B|DCBRECFCM;
0138          END;
0139      END;
0140      END;
0141      IF DCBBLKSI=0 THEN DO; /* IS BLKSIZE SET*/
0143          IF PRS=4 THEN DO;
0145              DCBBLKSI=121;
0146              DCBLRECL=121;
0147              VLRECL=121;
0148              RO=121;
0149          END;
0150          ELSE
0151              DO;
0152                  DCBBLKSI=80;
0153                  DCBLRECL=80;
0154                  VLRECL=80;
0155          END;
0156      IF DCBBUFNO=0 THEN DO;
0157      /* CHECK TO SEE IF BUFNO IS SPECIFIED*/
          DCBBUFNO=3;
          VBUFNO=3;
0158          END;
0159          ELSE
0160              DO;
0161                  VBUFNO=DCBBUFNO;
0162          END;
0163          RO=POOLD;
0164          GEN(GETPOOL (1),(C)); /*OBTAIN SPACE FOR BUFFERS*/
0165          RETURN;
0166          END;
0167      IF DCBLRECL=0 THEN DCBLRECL=DCBBLKSI;
0169      RETURN;
0170      END;
0171      END;

```

OCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
104	ABEND	STATIC, LOCAL, LABEL, INTERNAL 104
43	BACK4	STATIC, LOCAL, LABEL, INTERNAL 43, 49
79	BAD	STATIC, NONLOCAL, LABEL, INTERNAL 80, 84, 89, 94, 99
80	BADN	BASED, CHARACTER(34), INTERNAL, BOUNDARY(BYTE,1) 83, 88, 93, 98, 101
5	CARDIN	STATIC, NONLOCAL, CHARACTER(92), INTERNAL, BOUNDARY(BYTE,1) 10
23	CLOSE	STATIC, NONLOCAL, ENTRY, EXTERNAL 23
12	CCMMON	STATIC, LOCAL, LABEL, INTERNAL 12, 17, 22, 26
5	CRDCUT	STATIC, NONLOCAL, CHARACTER(92), INTERNAL, BOUNDARY(BYTE,1) 19
128	DCBBLKSI	BASED, FIXED(15), INTERNAL, BOUNDARY(HWORD,1) 141, 145, 151, 168
127	DCBBUFNO	BASED, PCINTER(8), INTERNAL, BOUNDARY(BYTE,1) 155, 157, 161
129	DCBLRECL	BASED, FIXED(15), INTERNAL, BOUNDARY(HWORD,1) 146, 152, 167, 168
126	DCBRECFM	BASED, PIT(8), INTERNAL, BOUNDARY(BYTE,1) 131, 133, 133, 133, 134, 134, 134, 137, 137, 137, 138, 138, 138
4	END	STATIC, LOCAL, FIXED(31), EXTERNAL, BOUNDARY(WORD,1) 26, 49, 78, 104
45	ERRCLT	STATIC, LOCAL, LABEL, INTERNAL 45, 54, 66
78	ERR5	STATIC, LOCAL, LABEL, INTERNAL 78, 122
41	FRR6	STATIC, LOCAL, LABEL, INTERNAL 41, 112
5	LINCUT	STATIC, NONLOCAL, CHARACTER(92), INTERNAL, BOUNDARY(BYTE,1) 15, 115
33	L1	STATIC, LOCAL, LABEL, INTERNAL

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
		33, 118
3	M	REGISTER(5), PCINTER(31), INTERNAL, BOUNDARY(WORD,1) 9, 14, 20, 28, 29
37	MESS	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 52, 53, 58
40	NODD1	STATIC, LOCAL, LABEL, INTERNAL 40, 101
125	CEXIT	STATIC, LOCAL, ENTRY, INTERNAL 125
7	OFLAGS	BASED, BIT(8), INTERNAL, BOUNDARY(BYTE,1) 33, 38
38	OTEST	STATIC, LOCAL, LABEL, INTERNAL 38
34	OUT1	STATIC, LOCAL, LABEL, INTERNAL 34, 42
6	O1	STATIC, NONLOCAL, LABEL, INTERNAL 11
6	O2	STATIC, NONLOCAL, LABEL, INTERNAL 21
6	O3	STATIC, NONLOCAL, LABEL, INTERNAL 16, 116
1	PARM1	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 18, 32
110	PNCHERR	STATIC, LOCAL, LABEL, INTERNAL 110
130	POOLD	STRUCTURE, STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 163
13	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 13
108	PRNTERR	STATIC, LOCAL, LABEL, INTERNAL 108
30	PRS	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 30, 31, 117, 135, 143
18	PUNCH	STATIC, NONLOCAL, ENTRY, EXTERNAL 18

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
3	P1	REGISTER(6), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 7, 10, 15, 19, 45, 115
3	P2	REGISTER(7), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 11, 16, 21, 35, 116
1	READ	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
106	READERR	STATIC, LOCAL, LABEL, INTERNAL 106
2	R0	REGISTER(0), POINTER(31), INTERNAL, BCUNDARY(WORD,1) 44, 53, 65, 148, 163
2	R1	REGISTER(1), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 35, 45, 126, 127, 128, 129
2	R3	REGISTER(3), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 32, 44
27	SWITCH	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 27, 42, 112, 114, 121
61	SYMESS	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 64, 65, 70, 75
2	V	REGISTER(4), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 28, 39, 49, 50, 56, 62, 68, 73, 78, 81, 86, 91, 96, 106, 108, 110
130	VRUFNO	IN POOLD, FIXED(15), INTERNAL, BCUNDARY(HWORD,1) 158, 161
130	VLPECL	IN POOLD, FIXED(15), INTERNAL, BOUNDARY(HWORD,1) 147, 153
107	XXX	STATIC, LOCAL, LABEL, INTERNAL 107, 109, 111, 114
53	X1	STATIC, LOCAL, LABEL, INTERNAL 53, 59
65	X2	STATIC, LOCAL, LABEL, INTERNAL 65, 71, 76

*** PROC. READ HAD 002 ERRORS
 *** THE FOLLOWING STATEMENT(S) HAD ERRORS
 0002,0002

III. PDUMP ROUTINE

1. Function

This BSL program allows the user to obtain "snap shot dumps" of his BSL program in any one of three formats. The module employs the PRINT entry point in the BSL object time I/O routine to produce its output.

2. How to use PDUMP

The user invokes the PDUMP program passing it a series of arguments. These arguments are grouped into triples and are used as follows:

- A. The first argument is a one character value which indicates the output format required. The allowable formats are:

TYPE CODES

CODE	TYPE	FORMAT
'A'	hexadecimal	A string of characters made up of digits 0-9 and A-F
'B'	Bit string	A string of character ones and zeros surrounded by quotes and followed by the letter B.
'C'	Character string	A string of EBCDIC characters surrounded by quotes.

- B. The second argument is the name of the BSL variable to be displayed.
- C. The third argument is a full word quantity which contains the length of the item to be displayed:

Length = length in bits for a bit string
length in characters for a character string
length in bytes for a hexadecimal item

- D. The final argument to the PDUMP program must be the character '*'. This character indicates the end of the list of items to be dumped. (The PDUMP program accepts variable length argument lists.)
- E. The maximum number of arguments allowed in a CALL statement is 25. However, the user may pass the PDUMP program a greater number by building his own argument list (see the BSL User's Guide).

3. Output format produced by PDUMP

The PDUMP program will skip to a new page each time it is invoked (called) and produce a new page with the heading 'PDUMP REQUESTED'. As each item is displayed it will:

- A. Have its starting location displayed on a new line as follows:

'LOCATION XXXXXXXX CONTAINS'

- B. Its value will start on a new line and continue over as many lines as is required to display it.

4. ERROR conditions detected by PDUMP

The PDUMP program detects three possible error conditions. When these conditions are encountered, a message is written on BSLOUT and control is returned to the calling program (note that any remaining arguments are ignored).

Error Message and Causes

MESSAGE	CAUSE
'ILLEGAL TYPE SPECIFICATION NO DUMP PRODUCED'	A type code other than 'A', 'B', or 'C', has been encountered (check to see if you have omitted the end of list character '*').
'ILLEGAL ADDRESS OR LENGTH SPECIFICATION'	The length of the item to be dumped was not contained in a full word quantity or an illegal address was passed to PDUMP (the argument list is incorrect).

5. Sample Use of PDUMP

```
X:PROC;  
  DCL A CHAR(5) INIT ('FRANK'), B FIXED(15)  
  INITIAL(3), C FIXED(31) INITIAL(15);  
  DCL B1(8) BIT(8), INIT((8) '00000000'B);  
  L3=64;  
  CALL PDUMP('C',A,5,'A',B,2,'A',C,4,'B',B1,L3,'*');  
  
END X;
```

```

0001 PDUMP:PROCEDURE;
0002   DCL R1 PTR(32) REG(1);
0003   DCL RSAVE PTR(32);
0004   RSAVE=R1;
0005   DCL XXAPICA FIXED(31);
0006   DCL RDUM FIXED(31);
0007   DCL BOOC CHAR(1), BOOB PTR(8);
0008   DCL T2 PTR(8);
0009   GEN;
      ST 2,RDUM
      CNOP 2,4
      LA 1,**+12
      BALR 1,1
      DC B'00001111'
      DC AL3(ERR)
      DC B'01111111'
      DC B'11111111'
      SVC 14
$ENDGEN
0010   RESTRICT(2);
0011   GEN;
      BALR 2,0
      SRL 2,24
      ST 1,XXAPICA
      STC 2,XXAPICA
      L 2,RDUM
$ENDGEN
0012   RELEASE(2);
0013   DCL 1 ARG CTL(RSAVE),
      (2PT, /*TYPE*/
      2 PA, /*ADDRESS*/
      2 PL) PTR(31); /*LENGTH IN BITS OR BYTES*/
0014   DCL BUF CHAR(121);
0015   DCL(Q1,Q2,Q3,P5,P4) PTR(31),TRAN(16)CHAR(1) INITIAL('0','1',
      '2','3','4','5','6','7','8','9','A','B','C','D','E','F');
0016   DCL BYTECON CTL(Q1) PTR(8),TYPE CTL(Q2) CHAR(1),LEN CTL(Q3)
      FIXED(31);
0017   DCL ADUMP BIT(32),L90 LABEL;
0018   BUF=' '; /* CLEAR BUFFER*/
0019   NARG=1; /*SET ARGUMENT COUNT */
0020   LINECT=0; /*LINE COUNT SET TO ZERO*/
0021   S1=1; /*PDUMP INITIALIZE SWITCH */
0022   BUF(1:1)='1'; /* SET UP PUT PAGE*/
0023   BUF(2:16)='PDUMP REQUESTFD';
0024   CALL PRINT(BUF);
0025   Q2=PT; /* PCINT TO TYPE*/
0026   LINECT=LINECT+1; /* BUMP LINE COUNT*/
0027   LSTART:IF TYPE='' THEN GO TO EPI; /* END OF LIST*/
0029   ADUMP=PA; /*POINT TO ADDRESS TO CONVERT */
0030   Q1=ADDR(ADUMP);
0031   BUF=' ';
0032   P5=ADDR(L90);
0033   GO TO L6; /* CHECK PAGE STATUS*/
0034   L90: BUF(2:27)='LOCATION CONTAINS';
0035   K=11;

```

```

0036             M=18;
0037             DCL L91 LABEL;
0038             P4=ADDR(L91);
0039             GO TO CC1;
0040     L91:       Q1=PA; /* POINT TO ITEM */
0041             Q3=PL; /* POINT TO LENGTH*/ TIME=1;
0043     IF LEN<=0 THEN GO TO ERR1;
0044     IF TYPE='B' THEN DO;
0045             TLEN=LEN+3; /*LENGTH OF BIT STRING*/
0046             GO TO RL;
0047             END;
0048     ELSE IF TYPE='C' THEN DO;
0049             TLEN=LEN+2; /*LENGTH OF CHAR STRING*/
0050             GO TO RL;
0051             END;
0052     ELSE IF TYPE='E' THEN DC;
0053     TLEN=LEN;
0054     GO TO RL;
0055     END;
0056     ELSE DO;
0057             TLEN=2*LEN;
0058             GO TO RL;
0059             END;
0060     DCL NEXT LABEL;
0061     RL: IF TYPE='E' THEN DC;
0062     IF TLEN<=120 THEN GO TO TL2;
0063     ELSE GO TO TL3;
0064     END;
0065     IF TLEN<120 THEN DC;
0066     TL2:;
0067             P4=ADDR(NEXT);
0068             FIT=0; /* ITEM FITS ON SINGLE LINE */
0069             GO TO P1;
0070             END;
0071     ELSE DO;
0072     TL3:;
0073             TLEN=TLEN-120;
0074             P4=ADDR(RL); /* ITEM REQUIRES PRINT LOOP */
0075             FIT=1;
0076             GO TO P1;
0077             END;
0078     P1: BUF=' ';
0079     DCL L7 LABEL;
0080     P5=ADDR(L7);
0081     L6: IF S1=1 THEN DO;
0082             S1=0;
0083             BUF(1)='F0'X;
0084             LINECT=LINECT+2;
0085             GO TO P5;
0086             END;
0087     ELSE DO; IF LINECT=55 THEN DO;
0088             LINECT=1;
0089             BUF(1:1)='1';
0090             GO TO P5;
0091             END;
0092     END;
0093     END;
0094     END;
0095     END;
0096     END;
0097     END;
0098     END;
0099     END;
0100     END;
0101     END;

```

```

0102             ELSE DO;
0103             BUF(1:1)='40'X;
0104             LINECT=LINECT+1;
0105             GO TO P5;
0106             END;
0107             END;
0108             L7: IF TYPE='A' THEN GO TO CONVERT; /* BRANCH ARITH*/
0110             ELSE IF TYPE='P' THEN GO TO CONVERT; /* BRANCH PTR */
0112             ELSE IF TYPE='B' THEN GO TO BCNT; /* BRANCH BIT*/
0114             ELSE IF TYPE='C' THEN GO TO LCHAR;
0116             ELSE IF TYPE='E' THEN GO TO TL;
0118 ELSE GO TO ERROR;
0119 LCHAR: IF TIME=1 THEN DO; /* MUST PLACE IN FIRST' */
0121             TIME=0;
0122             BUF(2:2)='';
0123             K=3;
0124             GO TO FF;
0125             END;
0126             ELSE DO;
0127 TL::
0128             K=2;
0129             GO TO FF; END;
0131             FF: IF FIT=0 THEN M=TLN;
0133             ELSE M=121;
0134             DCL J REG(8) FIXED(31);
0135             RESTRICT(8);
0136             DO J=K TO M BY 1;
0137             BUF(J)=BYTECON;
0138             Q1=Q1+1;
0139             END;
0140             IF FIT=0 THEN DO;
0142             IF TYPE='E' THEN GO TO FF1;
0144             DCL CRUD CHAR(1);
0145             CRUD='';
0146             BUF(J)=CRUD;
0147 FF1: CALL PRINT(BUF);
0148             GO TO P4;
0149             END;
0150             ELSE GO TO FF1;
0151             CONVERT: IF FIT=0 THEN DO;
0153             K=2;
0154             M=TLN+1;
0155             GO TO CC1;
0156             END;
0157             ELSE DO;
0158             K=2;
0159             M=121;
0160             GO TO CC1;
0161             END;
0162             CC1: DO J=K TO M BY 1;
0163             BTEMP=0;
0164             BTEMP=BYTECON;
0165             BTEMP=BTEMP & 'F0'X; /* CONVERT HEX 4 BITS AT A TIME */
0166             RTEMP=BTEMP/16;
0167             BTEMP=BTEMP+1;

```



```
0168     BUF(J)=TRAN(BTEMP);
0169     BTEMP=0;
0170     BTEMP=BYTECON;
0171     J=J+1;
0172     BTEMP=BTEMP&'OF'X;
0173     BTEMP=BTEMP+1;
0174     BUF(J)=TRAN(BTEMP);
0175     Q1=Q1+1;
0176     END;
0177     CALL PRINT(BUF);
0178     GO TO P4;
0179     BCONT:IF TIME=1 THEN IF FIT=0 THEN DO;
0182     TIME=0;
0183     BUF(2)='***';
0184     COUNT=TLEN-3;
0185     J=3;
0186     K=TLEN;
0187     GO TO TT1;
0188     END;
0189     ELSE DO;
0190     TIME=0;
0191     HIST=1;
0192     BUF(2)='***';
0193     COUNT=119;
0194     J=3;
0195     K=121;
0196     GO TO TT1;
0197     END;
0198     ELSE IF FIT=0 THEN DC;
0200     BUF(2)=B00C;
0201     J=3;
0202     COUNT=TLEN-3;
0203     HIST=0;
0204     IF COUNT=0 THEN GO TO DONE;
0206     ELSE DO;
0207     K=TLEN;
0208     GO TO TT1;
0209     END;
0210     ELSE DO;
0211     BUF(2)=B00C;
0212     J=3;
0214     COUNT=119;
0215     K=121;
0216     GO TO TT1;
0217     END;
0218     DCL MASK BIT(8);
0219     DCL T1 PTR(8);
0220     TT1: IF J>= K THEN GO TO DONE;
0222     ELSE DO;
0223     IF COUNT<=8 THEN DC;
0225     LOOP1=COUNT;
0226     MASK='80'X;
0227     END;
0228     ELSE DO;
```

```

0229 LOOP1=8;
0230 MASK='80'X;
0231 END;
0232 T2=BYTECON;
0233 DO I=1 TO LOOP1 BY 1;
0234 T1=T2&MASK;
0235 IF T1=0 THEN BUF(J)='F0'X;
0237 ELSE BUF(J)='F1'X;
0238 MASK=MASK/2;
0239 J=J+1;
0240 END;
0241 Q1=Q1+1;
0242 COUNT=COUNT-8;
0243 GO TO TT1;
0244 END;
0245 DONE: IF FIT=0 THEN DO;
0247 BUF(J)='''';
0248 J=J+1;
0249 BUF(J)='B';
0250 TT2:CALL PRINT(BUF);
0251 GO TO P4;
0252 END;
0253 ELSE DO;
0254 IF HIST=1 THEN DO;
0256 RPOB=BYTECON&MASK;
0257 IF BOOB=0 THEN BOOC='F0'X;
0259 ELSE BOOC='F1'X;
0260 GO TO TT2;
0261 END;
0262 ELSE GO TO TT2;
0263 END;
0264 NEXT:RSAVE=RSAVE+12;
0265 NARG=NARG+1;
0266 Q2=PT;
0267 IF NARG=101 THEN GO TO EPI;
0269 ELSE GO TO LSTART;
0270 ERROR: BUF='ILLEGAL TYPE SPECIFICATION NO DUMP PRODUCED';
0271 CALL PRINT(BUF);
0272 BUF=' ';
0273 CALL PRINT(BUF);
0274 GO TO EPI;
0275 GEN(USING *,15);
0276 ERR:GEN(L 11,CAP);
0277 GEN(BC 15,LUM);
0278 GEN(DS 0F);
0279 CAP:GEN(DC A(PDUMP+6));
0280 LUM:GEN(LA 13,@SAV001);
0281 GEN(DROP 15);
0282 EPR1:BUF='ILLEGAL LENGTH OR ADDRESS SPECIFICATION';
0283 CALL PRINT(BUF);
0284 BUF=' ';
0285 CALL PRINT(BUF);
0286 EPI:GEN(ST 2,RDUM);
0287 GEN(IC 2,XXAPICA);
0288 GEN(SLL 2,24);

```

```
0289     GEN(MVI   XXAPICA,X'00');
0290     GEN(L     1,XXAPICA );
0291     GEN(SVC   14);
0292     GEN(SPM   2);
0293     GEN(L     2,RDUM);
0294     RETURN;
0295         END;
```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
17	ADUMP	STATIC, LOCAL, BIT(32), INTERNAL, BOUNDARY(BYTE,1) 29, 30
13	ARG	STRUCTURE, BASED, CHARACTER(12), INTERNAL, BOUNDARY(WORD,1)
113	BCONT	STATIC, LOCAL, LABEL, INTERNAL 113, 179
7	BOCB	STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 256, 257
7	BOOC	STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 200, 212, 258, 259
163	BTEMP	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 163, 164, 165, 165, 166, 166, 167, 167, 168, 169, 170, 172, 172, 173, 173, 174
14	BUF	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 18, 22, 23, 24, 31, 34, 85, 91, 99, 103, 122, 137, 146, 147, 168, 174, 177, 183 192, 200, 212, 236, 237, 247, 249, 250, 270, 271, 272, 273, 282, 283, 284, 285
16	BYTECON	BASED, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 137, 164, 170, 232, 256
27C	CAP	STATIC, LOCAL, LABEL, INTERNAL 279
39	CC1	STATIC, LOCAL, LABEL, INTERNAL 39, 155, 160, 162
109	CONVERT	STATIC, LOCAL, LABEL, INTERNAL 109, 111, 151
184	COUNT	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 184, 193, 202, 204, 214, 223, 225, 242, 242
144	CRUD	STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 145, 146
205	DONE	STATIC, LOCAL, LABEL, INTERNAL 205, 221, 245
28	EPI	STATIC, LOCAL, LABEL, INTERNAL 28, 268, 274, 286
276	ERR	STATIC, LOCAL, LABEL, INTERNAL 276
118	ERROR	STATIC, LOCAL, LABEL, INTERNAL 118, 27C
44	ERR1	STATIC, LOCAL, LABEL, INTERNAL

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
		44, 282
124	FF	STATIC, LOCAL, LABEL, INTERNAL 124, 129, 131
143	FF1	STATIC, LOCAL, LABEL, INTERNAL 143, 147, 150
75	FIT	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 75, 82, 131, 140, 151, 180, 198, 245
191	HIST	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 191, 203, 254
233	I	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 233
134	J	REGISTER(8), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 136, 137, 146, 162, 168, 171, 171, 174, 185, 194, 201, 213, 220, 236, 237, 239 239, 247, 248, 248, 249
35	K	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 35, 123, 128, 136, 153, 158, 162, 186, 195, 207, 215, 220
115	LCHAR	STATIC, LOCAL, LABEL, INTERNAL 115, 119
16	LEN	BASED, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 43, 47, 52, 57, 61
20	LINECT	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 20, 26, 26, 92, 92, 96, 98, 104, 104
225	LCOP1	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 225, 229, 233
27	LSTART	STATIC, LOCAL, LABEL, INTERNAL 27, 269
280	LUM	STATIC, LOCAL, LABEL, INTERNAL 280
33	L6	STATIC, LOCAL, LABEL, INTERNAL 33, 88
86	L7	STATIC, LOCAL, LABEL, INTERNAL 87, 108
17	L90	STATIC, LOCAL, LABEL, INTERNAL 32, 34
37	L91	STATIC, LOCAL, LABEL, INTERNAL 38, 40

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
36	M	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 36, 132, 133, 136, 154, 159, 162
218	MASK	STATIC, LOCAL, BIT(8), INTERNAL, BOUNDARY(BYTE,1) 226, 230, 234, 238, 238, 256
19	NARG	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 19, 265, 265, 267
64	NEXT	STATIC, LOCAL, LABEL, INTERNAL 74, 264
13	PA	IN ARG, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 29, 40
1	PDUMP	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
13	PL	IN ARG, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 41
24	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 24, 147, 177, 250, 271, 273, 283, 285
13	PT	IN ARG, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 25, 266
76	P1	STATIC, LOCAL, LABEL, INTERNAL 76, 83, 85
15	P4	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 38, 74, 81, 148, 178, 251
15	P5	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 32, 87, 93, 100, 105
15	Q1	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 16, 30, 40, 138, 138, 175, 175, 241, 241
15	Q2	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 16, 25, 266
15	Q3	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 16, 41
6	RDUM	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
48	RL	STATIC, LOCAL, LABEL, INTERNAL 48, 53, 58, 62, 65, 81
3	RSAVE	STATIC, LOCAL, POINTER(32), INTERNAL, BOUNDARY(WORD,1) 4, 13, 264, 264

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
2	R1	REGISTER(1), POINTER(32), INTERNAL, BCUNDARY(WORD,1) 4
21	S1	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 21, 88, 90
42	TIME	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 42, 119, 121, 179, 182, 190
117	TL	STATIC, LOCAL, LABEL, INTERNAL 117, 127
47	TLEN	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 47, 52, 57, 61, 67, 71, 80, 80, 132, 154, 184, 186, 202, 207
68	TL2	STATIC, LOCAL, LABEL, INTERNAL 68, 73
69	TL3	STATIC, LOCAL, LABEL, INTERNAL 69, 79
15	TRAN	(16), STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 168, 174
187	TT1	STATIC, LOCAL, LABEL, INTERNAL 187, 196, 208, 216, 220, 243
250	TT2	STATIC, LOCAL, LABEL, INTERNAL 250, 260, 262
16	TYPE	BASED, CHARACTER(1), INTERNAL, BCUNDARY(BYTE,1) 27, 45, 50, 55, 65, 108, 110, 112, 114, 116, 142
219	T1	STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 234, 235
8	T2	STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 232, 234
5	XXAPICA	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1)

*** PROC. PDUMP HAD NO ERRORS

IV. SUBSTR1. Function

SUBSTR is a program which is written in BSL and provides the following facilities for character substrings:

- A. Checks the validity of variable substring assignments and prevents assignments which are illegal, while providing diagnostics.
- B. Provides for truncation or padding in variable substring assignments. The length of the substring to be moved is determined by taking the minimum of the length of the specified target substring and the length of the substring defined by starting from the beginning point of the target substring to the declared string end. If the source substring is longer than the target, the source is truncated on the right to the size of the target before it is assigned. If the source string is shorter than the target, it is padded on the right with blanks to the size of target before it is assigned.

2. How to use SUBSTR

SUBSTR requires nine arguments. They are:

- A. The name of the string to which assignment is to be made. This string is called the TARGET. If the assignment is found to be illegal, then the contents of the TARGET variable will be as they were on entry to SUBSTR.
- B. The declared length of the TARGET string in characters, contained in a full word. This value is used for range checking by SUBSTR.
- C. The beginning point of the TARGET substring. This is a full word quantity which indicates the first character of the TARGET substring. (SUBSTRINGS are counted starting from one.)
- D. The end point of the TARGET substring. This is a full word quantity which is used to indicate the stopping point of the TARGET substring.

Substrings are moved from the beginning character point up to and including the end character point.

- E. The name of the string from which data is to be taken. This is called the SOURCE string.
- F. The declared length of the SOURCE string in characters, contained in a full word. This information is used for range checking by SUBSTR.
- G. The beginning point of the source substring. This is a full word quantity which indicates the first character of the SOURCE substring.
- H. The end point of the source substring. This is a full word quantity which indicates the last character of the SOURCE substring.
- I. The last argument is a full word quantity which is either 1 or 0. This is used to indicate whether errors encountered in the SUBSTR program are to be written on the file BSLOUT. If a one is supplied both an error code and an error message is supplied. If a zero is supplied, the error message is suppressed but the error code is set.

(Note: The messages written by SUBSTR use the PRINT entry point of the BSL OBJECT TIME INPUT/OUTPUT ROUTINE.)

3. ERROR Conditions Detected by SUBSTR

When SUBSTR detects an illegal assignment, it sets a variable EOR (declared EOR FIXED(31) LOCAL EXTERNAL) to a value from 1 to 7 and optionally prints out a diagnostic. The following table lists the error messages put out by SUBSTR and the value set in EOR.

(Note SUBSTR has its own error handling routine which in turn uses ERRINT another member of the BSLLIB.)

SUBSTR ERROR CONDITIONS

MESSAGE (on BSLOUT)	EOR Value	Explanation
LOWER BOUND ON TARGET LESS THAN 1	1	The substring begin point on the target is less than one.
UPPER BOUND ON TARGET LESS THAN LOWER	2	The substring end point on the target is less than the substring begin point.
LOWER BOUND ON SOURCE LESS THAN 1	3	The substring begin point on the source is less than one.
UPPER BOUND ON SOURCE LESS THAN LOWER	4	The substring end point of the source is less than the substring begin point.
LOWER BOUND G.T. DCL'D LENGTH OF TARGET	5	The substring begin point on the target is at a point which is past the end of the target string.
LOWER BOUND G.T. DCL'D LENGTH OF SOURCE	6	The substring begin point on the source is at a point which is past the end of the source string.
ILLEGAL ARGUMENT TO SUBSTR PGM * (This message is not suppressed by having the ninth argument to SUBSTR set to zero. It always appears.)	7	This message indicates that one of the nine arguments to the SUBSTR program is illegal and has caused a program check.

The programmer can test the value of EOR on return from SUBSTR. If the value is 0, then the assignment was carried out. If the value is not zero, then its value is (1-7) and will indicate the error type.

4. Sample Use of SUBSTR

```
DCL A CHAR(10) INIT('ABCDEFGHIJ'), B CHAR(5);
      B=" "; I=1; J=5; K=1; M=5;
CALL SUBSTR(A,10,I,J,B,5,K,M,1);
      Result is: A='-----FGHIJ'
```

Notes

- A. Size of TARGET substring is 5 characters.
- B. Size of SOURCE substring is 5 characters.
- C. The result of the BSL statement $A(I;J) = B(K;M)$ is the same as SUBSTR.

```
DCL C2 CHAR(200), C3 CHAR(3);
      DO I=1 TO 200;
      C2(I) = 'A';
      END;
      C3='RST';
      I=3;
      J=200;
      K=3;
      M=3;
CALL SUBSTR(C2,200,I,J,C3,3,K,M,1);
      Result is: 'AAT          197 blanks'
```

Notes

- A. The TARGET substring is 198 characters long.
- B. The SOURCE substring is 1 character long.
- C. The SOURCE substring should be padded right with 197 BLANKS.
- D. The result of the BSL statement:


```
      C2(I:J) = C3 (K:M);
```

 would be to move 198 characters from the 3rd position of C3.

```
DCL C4 CHAR(5) INITIAL('12345'), C5 CHAR(6) INIT('ABCDEF');
      I=2;
      J=5;
      K=1;
      M=6;
```

```
CALL SUBSTR(C4,5,I,J,C5,6,K,M,1);
```

```
Result is:
      '1ABCD'
```

Notes

- A. TARGET substring is 4 characters long.
- B. SOURCE substring is 6 characters.
- C. SOURCE is greater than the TARGET therefore it is truncated on the right to the size of the TARGET.
- D. The result of the BSL statement C4(I:J) = C5(K:M) is the same as SUBSTR.

```
DCL C7 CHAR(5) INITIAL ('12345')
      I=2;
      J=5;
      K=3;
      M=5;
```

```
CALL SUBSTR (C7,5,I,J,C7,5,K,M,1);
```

```
Result is: '1345 '
```

Notes

- A. TARGET substring is 4 characters.
- B. SOURCE substring is 3 characters.
- C. SOURCE substring is padded right with one blank before it is assigned.
- D. The result of the BSL statement C7(I:J) = C7(K:M); would be '1345?' The value depends on what the byte following C7(5) contains.

```

0001 SURSTR:PROCEDURE(RCVR,DLEN1,ST1,ST2,SOU,DLEN2,ST3,ST4,YFA):
0002   DCL MERR ENTRY ,TAR(30) PTR(31):
0003     DCL FOR FIXED(31) LOCAL EXTERNAL:
0004     FOR=0:
0005     J=1:
0006     DO I=1 BY 1 TO 30:
0007       TAB(I)=J:
0008       I=I+1:
0009       TAB(I)=ADDR(MERP):
0010       J=J+1:
0011     END:
0012   /* SET UP ERROR HANDLING*/
        CALL FRRINT(15,TAR,1):
        ** W06 ** DIMENSIONED ITEM NOT SUBSCRIPTED
0013     DCL(RCVR,SOU)PTR(31),(DLEN1,ST1,ST2,DLEN2,ST3,ST4,YFA) FIXED(31):
0014     DCL(RPTR,SPTR)PTR(31):
0015     RPTR=ADDR(RCVR):
0016     SPTR=ADDR(SOU):
0017     DCL TARGET CHAR(1) CTL(RPTR),SOURCE CHAR(1) CTL(SPTR):
0018     DCL BUFFER CHAR(121):/* CHECK INDIVIDUAL SUBSTRS*/
0019     IF ST1<1 THEN DO:
0020       BUFFER=' LOWER ROUND ON TARGET LESS THAN 1':
0021     FOR=1:
0022     GO TO L1:
0023   END:
0024   IF ST2<ST1 THEN DO:
0025     BUFFER=' UPPER ROUND ON TARGET LESS THAN LOWER':
0026   FOR=2:
0027   GO TO L1:
0028   END:
0029   IF ST3<1 THEN DO:
0030     BUFFER=' LOWER ROUND ON SOURCE LESS THAN 1':
0031   FOR=3:
0032   GO TO L1:
0033   END:
0034   IF ST4<ST3 THEN DO:
0035     BUFFER=' UPPER ROUND ON SOURCE LESS THAN LOWER':
0036   FOR=4:
0037   GO TO L1:
0038   END:
0039   IF ST1>DLEN1 THEN DO:
0040     BUFFER=' LOWER ROUND G.T. DCL'D LENGTH OF TARGET':
0041   FOR=5:
0042   GO TO L1:
0043   END:
0044   IF ST3>DLEN2 THEN DO:
0045     BUFFER=' LOWER ROUND G.T. DCL'D LENGTH OF SOURCE':
0046   FOR=6:
0047   GO TO L1:
0048   END:
0049   LEFT1=DLEN1-ST1+1: /*AMOUNT LEFT IN TARGET*/
0050   MOVE1=ST2-ST1+1: /*SIZE OF TARGET SUBSTR*/
0051   LEFT2=DLEN2-ST3+1: /*AMOUNT LEFT IN SOURCE*/
0052   MOVE2=ST4-ST3+1: /* SIZE OF SOURCE SUBSTR*/
0053   IF MOVE1<=LEFT1 THEN MOVE=MOVE1:

```

```
0061     ELSE TMOVE=LEFT1;
0062     /* TMOVE IS MIN OF SPEC TARGET SUBSTR AND TARGET LENGTH*/
0063     IF MOVE2<=LEFT2 THEN SMOVE=MOVE2;
0064     ELSE SMOVE=LEFT2;
0065     /* SMOVE MIN OF SPEC SUBSTR OF SOURCE AND SOURCE LENGTH*/
0066     PAD=0;
0067     IF TMOVE>SMOVE THEN DO; /*PAD TARGET*/
0068     PAD=TMOVE+ST1-1;
0069     L5:DST1=ST1;
0070     DST3=ST3;
0071     DO COUNT=1 BY 1 TO SMOVE;
0072     TARGET(DST1)=SOURCE(DST3);
0073     DST1=DST1+1;
0074     DST3=DST3+1;
0075     END;
0076     IF PAD<=0 THEN DO;
0077     F000=ST1+SMOVE;
0078     DO L000=F000 BY 1 TO PAD;
0079     /* BLANK TARGET*/
0080     TARGET(L000)=' ';
0081     END;
0082     END;
0083     L6: CALL F00CL;
0084     RETURN;
0085     END;
0086     ELSE IF SMOVE>TMOVE THEN DO;
0087     SMOVE=TMOVE;
0088     GO TO L5;
0089     END;
0090     ELSE GO TO L5;
0091     L1: IF YEA=0 THEN GO TO L6;
0092     CALL PRINT(BUFFER);
0093     GO TO L6;
0094     END;
```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
18	BUFFER	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 21, 27, 33, 39, 45, 51, 94
71	COUNT	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 71
1	DLEN1	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 43, 55
1	DLEN2	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 49, 57
69	DST1	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 69, 72, 73, 73
70	DST3	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 70, 72, 74, 74
3	EOR	STATIC, LOCAL, FIXED(31), EXTERNAL, BOUNDARY(WORD,1) 4, 22, 28, 34, 40, 46, 52
83	ERRCL	STATIC, NONLOCAL, ENTRY, EXTERNAL 83
12	FRRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 12
78	FOOP	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 78, 79
6	I	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 6, 7, 8, 8, 9
5	J	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 5, 7, 10, 10
55	LEFT1	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 55, 59, 61
57	LEFT2	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 57, 62, 64
79	LOOP	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 79, 80
23	L1	STATIC, LOCAL, LABEL, INTERNAL 23, 29, 35, 41, 47, 53, 92
69	L5	STATIC, LOCAL, LABEL, INTERNAL 69, 89, 91
83	L6	STATIC, LOCAL, LABEL, INTERNAL

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE 83, 93, 95
2	MERR	STATIC, NONLOCAL, ENTRY, EXTERNAL 9
56	MOVE1	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 56, 59, 60
58	MOVE2	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 58, 62, 63
65	PAD	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 65, 68, 76, 79
94	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 94
1	RCVR	PARAMETER, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 15
14	RPTR	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 15, 17
63	SMQVE	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 63, 64, 66, 71, 78, 86, 88
1	SOU	PARAMETER, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 16
17	SOURCE	BASED, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 72
14	SPTR	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 16, 17
1	ST1	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 19, 25, 43, 55, 56, 68, 69, 78
1	ST2	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 25, 56
1	ST3	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 31, 37, 49, 57, 58, 70
1	ST4	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 13, 13, 37, 58
1	SUBSTR	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
2	TAB	(30), STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 7, 9, 12

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
17	TARGET	BASED, CHARACTER(1), INTERNAL, BCUNDARY(BYTE,1) 72, 80
60	TMOVE	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 60, 61, 66, 68, 86, 88
1	YEA	PARAMETER, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 13, 13, 92

*** PROC. SUBSTR HAD 001 ERROR
*** THE FOLLOWING STATEMENT(S) HAD ERRORS
0012

```
0001 MERR:PROCEDURE(R,X);
0002   DCL EOR FIXED(31) NONLOCAL;
0003   DCL BUF CHAR(121);
0004   BUF =' ILLEGAL ARGUMENT TO SUBSTR PGM';
0005   CALL PRINT(BUF);
0006   EOR=7;
0007   GEN(L   15,4(0,1));
0008   GEN(L   15,40(0,15));
0009   GEN(L   13,4(0,15));
0010   GEN(LM  14,12,12(13));
0011   GEN(BCR 15,14);
0012   END;
```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
3	BUF	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 4, 5
2	EOR	STATIC, NONLOCAL, FIXED(31), EXTERNAL, BOUNDARY(WORD,1) 6
1	MERR	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
5	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 5
1	R	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
1	X	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1)

*** PROC. MERR HAD NO ERRORS

V. ERRINT1. Function

ERRINT is a program written in BSL which employs the SPIE macro to provide the user with a facility similar to PL/I ON-UNITS. It allows the BSL programmer to handle the 15 possible program interruptions by:

- A. Specifying for all or some of these interruptions, external procedures which are to be entered when the interrupt occurs.
- B. Providing system action for those interrupts which the user does not wish to handle.
- C. Providing the ability to ignore some or all of the program interrupts if the user desires.

2. How to use the ERROR-HANDLER

In order to specify error handling, the error routine must be called using the entry name ERRINT, passing three arguments. The three arguments are:

- A. A full word integer which specifies the number of the 15 possible program interrupts the user wishes to handle.
- B. An array of up to 30 quantities each of which is declared as PTR(31). This table should be built such that a number code for an interruption is followed by either:
 - b1) A pointer to the entry point of an external procedure. This is the routine which will handle the interruption specified by the preceding number.
 - b2) A '-1' which indicates that the interruption specified by the preceding number is to be ignored.
 - b3) A '0' which indicates that system action is required for the interruption specified by the preceding number.

BUILDING AN ARGUMENT LIST FOR ERRINT

```

DCL T(6) PTR(31);
T(1) = 4; /* CODE FOR PROTECTION INTERRUPTION */
DCL (PROT,ZD) ENTRY EXTERNAL;
/* PROT IS THE PROCEDURE TO HANDLE A PROTECTION ERROR */
/* ZD IS THE PROCEDURE TO HANDLE A FIXED POINT
DIVIDE ERROR */
T(2)=ADDR (PROT) ; /* INSERT ADDRESS OF PROT INTO
TABLE */
T(3)=9; /* CODE FOR FIXED POINT DIVIDE ERROR */
T(4)=ADDR (ZD) : /* INSERT ADDRESS OF ZD INTO
TABLE */
T(5)=7; /* CODE FOR DATA ERROR */
T(6)=-1; /* INDICATE THAT DATA ERRORS ARE TO BE
IGNORED */
CALL ERRINT (3,T,1);
/* THE FIRST ARGUMENT INDICATES THAT 3 ERROR CONDITIONS
WILL BE HANDLED. THE LAST ARGUMENT INDICATES THAT
THIS IS THE INITIAL CALL TO ERRINT */

```

- C. A full word integer which is either a 1 or not 1. If ERRINT has never been called, then a one must be specified. This causes initialization of error handling by ERRINT and saves the status of interruption handling initialized by a prior SPIE macro. If a value other than one is specified, it is assumed that the user wishes to override error handling specified in a previous call to ERRINT.
- D. The error handling routine will use the values passed to it to initialize a table called CONTAB. This table is defined as CONTAB(15) PTR(31) LOCAL EXTERNAL and is formatted such that any position in it contains the action for the corresponding interrupt as defined by the following table:

<u>NUMBER</u>	<u>INTERRUPTION TYPE</u>	<u>CONTAB POSITION</u>
1	Operation	CONTAB (1)
2	Privileged Operation	CONTAB (2)
3	Execute	CONTAB (3)
4	Protection	CONTAB (4)
5	Addressing	CONTAB (5)
6	Specification	CONTAB (6)
7	Data	CONTAB (7)
8	Fixed Point Overflow	CONTAB (8)
9	Fixed Point Divide	CONTAB (9)
10	Decimal Overflow	CONTAB (10)
11	Decimal Divide	CONTAB (11)
12	Exponent Overflow	CONTAB (12)
13	Exponent Underflow	CONTAB (13)
14	Significance	CONTAB (14)
15	Floating-Point Divide	CONTAB (15)

If the third argument to ERRINT is a one, the entire CONTAB table is set to zero before the user's list is examined. This allows system action to be initialized for those interruptions which the user does not wish to handle. If a value other than one is specified for the third argument, only those values passed by the user are used to update the CONTAB. Since each call to the error routine causes the value of CONTAB to be modified, it is the users responsibility to save its status should he wish to restore it to its prior value. The table may be stored as follows:

```
DCL CONTAB(15) PTR(31) NONLOCAL, SAVE(15) PTR(31);

DO I=1 to 15;

SAVE (I) = CONTAB (I);

END;
```

E. When the Error-Handler gets Control

When a program interruption occurs after ERRINT has been called, the following action takes place:

- e1) The contents of registers 14-12 are saved assuming that register 13 is pointing at the save area of the BSL program in control at the point of interruption.
- e2) The interruption is cleared by the interrupt handler.
- e3) If the error routine determines that the interruption is to be ignored, it merely restores registers and returns control to the interrupted program.
- e4) If system action is required, the location and type of the interrupt are printed out in the data set BSLOUT, and an ABEND dump is given. A system action message would be of the form Condition at XXXXXX (where XXXXXX is the location of the interruption and Condition indicates the type of error).
- e5) If a user routine is to be called, it is called by the error handler which passes two arguments. These arguments provide the user with the status of the machine at the point of interruption. The format of the arguments passed to the user error routine are as follows:

```
Argument 1.  DCL 1 PIE BOUNDARY (DWORD) ,
              2 PICAA PTR (31) , /*ADDR OF ERROR HANDLER*/
              2 OPSWL PTR (31) , /*LEFT OF PSW AT THE*/
                          /*POINT OF INTERRUPT*/
              2 OPSWR PTR (31) , /*RIGHT OF PSW AT*/
                          /*THE POINT OF INTERRUPT*/
( 2 R14, /*REGS 14-2 */
  2 R15 /* AT THE PT OF INTERRUPT */
  2 R0,
  2 R1,
  2 R2 ) PTR (31) ;
```


Argument 2. DCL R(12) PTR(31);

Argument 2 contains registers 3-13 at the point of interrupt, and the address of the location where the interrupt resume point is stored in the ERROR-HANDLER.

Note: (R(11) will point to the SAVE AREA of the interrupted program. R(12) will contain the address where the resume point is stored in the ERROR-HANDLER.)

The user routine must return to the ERROR-HANDLER by a normal return. When this happens the status of the machine will be restored from the save area of the interrupted program. The interrupted program will resume from the point of interrupt (the address where the resume point is stored is in R(12)). The user routine may also choose not to return to the point of interrupt. When the user does not wish to resume at the point of interruption, he should use R(12) as a pointer to reset the resume point and return to the ERROR-HANDLER.

Example

```
DCL RETPT LABEL EXTERNAL;
/* RESUME POINT */
DCL P1 PTR; GOBACK PTR BASED(P1);
    P1=R(12); /* POINT TO SLOT WHERE
/* RESUME POINT IS STORED */
    GOBACK=ADDR(RETPT);
/* RESET RESUME POINT TO RETPT */
    RETURN; /*RETURN TO ERROR-HANDLER */
```

F. Restoring the interruption handling specified before ERRINT is called

To restore the interrupt action specified before ERRINT was called the user should call ERRCL.

Special Note:

The user should not call ERRINT more than once in a given job with a third parameter of one, unless he wishes to destroy previous definitions of interruption handling.

G. Data Sets Used by the ERROR HANDLER

DD NAME	USE
BSLOUT	SYSTEM ACTION MESSAGE
SYSABEND	ABEND DUMP

3. Errors detected by the ERROR-HANDLER

There are only two error conditions which the error handler detects. When the CONTAB table is being built, if an interruption code is less than 1 or greater than 15 then the message, 'ILLEGAL TYPE IT IS IGNORED' is printed out on BSLOUT. Control is returned to the calling program (note that when the CONTAB table is incomplete previous values are in effect). If the number of interruptions specified by the user is greater than 15 or less than zero, then the message 'INCORRECT ROUTINE COUNT' is written on BSLOUT. Control is returned to the caller.

4. Example of the Use of ERRINT

```
TESTC : PROC;

DCL (PL,P2) ENTRY; /* DEFINE ENTRIES OF ROUTINES*/
                  /* TO HANDLE FIXED-PT-DIVIDE*/
                  /* AND ADDRESSING ERROR */

DCL TAB(4) PTR(31); /* DEFINE USER INTERRUPT */
                  /* TABLE */

TAB(1) = 9;        /* SET FIXED-PT DIVIDE CODE */
TAB(2) = ADDR(P1); /* ADDR OF ROUTINE TO */
                  /* HANDLE FIXED-PT DIVIDE */
TAB(3) = 5;        /* SET ADDRESSING CODE */

TAB(4) = ADDR(P2); /* SET ADDR OF ROUTINE */
                  /* TO HANDLE ADDR ERROR */

CALL ERRINT (2,TAB,1); /*INIT ERROR HANDLING */

/* FOR FIXED-PT DIVIDE AND ADDRESSING, OTHERS DEFAULT
   TO SYSTEM ACTION */

    Y = Y/0; /* GET ZERO DIVIDE */

DCL P5 PTR, BYTCN CHAR(1) CTL(P5);

    P5 = -3; /* AFTER ZERO DIVIDE NORMAL RETURN */

/* FROM ERROR HANDLER COMES TO THIS STATEMENT */
/* THIS STATEMENT CAUSES AN ADDR ERROR */

DCL L3 LABEL LOCAL EXTERNAL;

/* THIS DEFINES A RETURN POINT TO BE USED */
/* BY THE ADDR ROUTINE */

BYTCN= 'A';
```

```
L3:  TAB(2) = -1; /* SET UP TO IGNORE DIVIDE */
      /* ERROR */

CALL ERRINT (2, TAB, 0);
DCL CAT ENTRY CTL (P5);

P5 = 1;
Y = Y/0;

DCL BUF CHAR (121) LOCAL EXTERNAL;
/* DEFINE MESS BUFFER */,
BUF= ' ZERO DIVIDE IGNORED;
CALL PRINT (BUF); /* CALL TO BSLLIB PGM TO */
      /* PRINT MESSAGE */

CALL CAT; /* FORCE SPECIFICATION ABEND */

END;
```

```
P1: PROC (RR,R2) ;
    DCL BUF CHAR (121) NONLOCAL;
    BUF= ' FIXED POINT DIVIDE';
    CALL PRINT (BUF) ;
    RETURN; /* ERROR HANDLER RETURNS TO POINT OF */
    END; /* INTERRUPT */

P2: PROC (RX,R3) ;
    DCL BUF CHAR(121) NONLOCAL;
    DCL L3 LABEL NONLOCAL; /* PLACE I WISH TO BRANCH */
    DCL 1 RX BDY (DWORD), /* PIE AT INTERRUPT */
        (2 PICAA PTR(31),
         2 OPSWL ,
         2 OPSWR ,
         2 R14A ,
         2 R15A ,
         2 R0A ,
         2 R1A ,
         2 R2A ) FIXED(31);
    DCL R3 (12) FIXED(31) ; /* REG 3 - 13 AND */
                                /* ADDRESS WHERE RETURN */
                                /* POINT IS STORED */
                                /* IN ERROR-HANDLER */
    BUF= ' ADDRESSING ERROR';
    CALL PRINT (BUF) ;
```

```
DCL P6 PTR(31), RETP PTR(31) BASED(P6);  
P6=R3(12); /* POINT TO RETURN POINT ADDR */  
RETP= ADDR(L3); /* SET RETURN POINT TO L3 */  
RETURN; /* ERROR HANDLER WILL RETURN to L3 */  
END;
```

```

0001   ERRINT:PROC(I,X,II) OPTIONS(REENTRANT);
0002   DCL BUF CHAR(121);
0003   DCL TRAN(16) CHAR(1) INITIAL('0','1','2','3','4','5',
    '6','7','8','9','A','B','C','D','E','F');
    ** W13 ** DATA ITEM MAY NOT BE ADDRESSABLE
0004   DCL MESS(15) CHAR(26) INITIAL(' OPERATION AT           ',
    ' PRIVILEGED OPERATION AT ', ' EXECUTE AT           ',
    ' PROTECTION AT           ', ' ADDRESSING AT           ',
    ' SPECIFICATION AT           ', ' DATA AT           ',
    ' FIXED-PCINT OVERFLOW AT ', ' FIXED POINT DIVIDE AT ',
    ' DECIMAL OVERFLOW AT           ', ' DECIMAL DIVIDE AT           ',
    ' EXPONENT OVERFLOW AT           ', ' EXPONENT UNDERFLOW AT           ',
    ' SIGNIFICANCE AT           ', ' FLOATING POINT DIVIDE AT');
    ** W13 ** DATA ITEM MAY NOT BE ADDRESSABLE
0005   DCL I FIXED(31), /* NUMBER OF ROUTINES GIVEN BY USER */
    X(30) PTR(31), /* USER INTERRUPT TABLE */
    II FIXED(31), /* INITIALIZE FLAG */
    XXAPICA FIXED(31) INITIAL(0) /* SLOT TO HOLD USERS PICA*/;
0006   DCL R1 REG(1) PTR(31), R2 REG(2) PTR(31);
    ** W05 ** POSSIBLE CONFLICT IN USE OF REGISTER
0007   * DCL ERMES CHAR(50) INITIAL(' ILLEGAL TYPE IT IS IGNORED');
0008   RSAVE=R1;
0009   RSAP= R2; /* SAVE REGISTERS*/
0010   DUMMY=II;
0011   IF DUMMY=1 THEN GO TO LL1; /* BRANCH SO PICA IS NOT SAVED */
0013   GEN;
    CNOP  2,4
    LA    1,**+12
    BALR  1,1
    DC    B'00001111'
    DC    AL3(ERR)
    DC    B'01111111'
    DC    B'11111111'
    SVC   14
$ENDGEN
0014   RESTRICT(2);
0015   GEN;
    BALR  2,0
    SRL   2,24
    ST    1,XXAPICA
    STC   2,XXAPICA
$ENDGEN
0016   RELEASE (2);
0017   LL1:R1=RSAP;
0018   DCL CONTAB(15) PTR(31) LOCAL EXTERNAL;
0019   R2=RSAP;
0020   IF II= 1 THEN DO JJ=1 TO 15; /*CLEAR INTERRUPT TABLE */
0022   CCNTAB(JJ)=0;
0023   END;
0024   TERM=2*I-1; /* SET COUNT OF INTERRUPTS*/
0025   IF TERM<0|TERM>30 THEN DO;
0027   BUF='1INCORRECT ROUTINE CCUNT';
0028   CALL PRINT(BUF);
0029   RETURN;
0030   END;

```

```

0031         DO J=1 TO TERM BY 2;
0032             INDEX=X(J);      /* PUT IN USER ROUTINES */
0033         IF INDEX<0|INDEX>15 THEN DO;
0034             BUF=ERRMES;
0035         CALL PRINT(BUF);
0036             GO TO DUTE;
0037         END;
0038         ELSE;
0039             CONTAB(INDEX)= X(J+1);
0040         DUTE:END;
0041             RETURN;
0042     ERR:ENTRY(XXX);
0043     DCL XXX PTR(31);
0044     DCL BLOOP PTR(31);
0045     BLOOP=R1;
0046     DCL 1 PIE CTL(BLOOP) BOUNDARY(DWORD),
0047             2 PICAA PTR(31),
0048             (2 OPSWL, /* LEFT HALF PGM PSW */
0049              2 OPSWR, /* RIGHT HALF PGM PSW */
0050             2 R14A,
0051              2 R15A,
0052              2 R0,
0053              2 R1A,
0054              2 R2A ) FIXED(31);
0055     DCL 1 PSWTMP BOUNDARY(DWORD) AUTOMATIC,
0056             (2 TPL, 2 TPR ) FIXED(31), BACK LABEL;
0057     TPL= OPSWL; /* SAVE OLD PSW */
0058     TPR= OPSWR;
0059     OPSWR=ADDR(BACK); /* TRICK SUPERVISOR*/
0060     GEN(L 15,4(0,13));
0061     GEN(L 14,12(0,15));
0062     GEN(BCR 15,14);
0063     DCL SINDX FIXED(15);
0064     DCL TPLA FIXED(15) CTL(ADDR(TPL)+2);
0065     BACK: SINDX= TPLA; /* PICK UP INTERRUPTION CODE */
0066     IF CONTAB(SINDX)=0 THEN DO; BUF=MESS(SINDX);
0067     BUFIN=28;
0068     DO CLAP=1 BY 1 TO 3;
0069     /* CONVERT ADDR */ BTEMP=0;
0070     DCL TPRZ (3) PTR(8) BASED(ADDR(TPR)+1);
0071     BTEMP=TPRZ(CLAP);
0072     BTEMP=BTEMP&'F0'X;
0073     BTEMP=BTEMP/16;
0074     BTEMP=BTEMP+1;
0075     BUF(BUFIN)=TRAN(BTEMP);
0076     BTEMP=0;
0077     BUFIN=BUFIN+1;
0078     BTEMP=TPRZ(CLAP);
0079     BTEMP=BTEMP&'0F'X;
0080     BTEMP=BTEMP+1;
0081     BUF(BUFIN)=TRAN(BTEMP);
0082     BUFIN=BUFIN+1;
0083     END;
0084     CALL PRINT(BUF);
0085     BUF=' ';

```



```

0080         CALL PRINT(BUF);
0081         CALL PRINT(BUF);
0082         CALL CLOSE;
0083         GEN;
           CNOP  0,4
           BC   15,**+8
           DC   AL1(192)
           DC   AL3(24)
           L    1,**-4
           SVC  13
$ENDGEN
0084
0085         END;
0086         DCL RET PTR(31),ERRTN ENTRY CTL(RET);
0087         IF CONTAB(SINDX)=-1 THEN DO;
0088     YD: GEN(L  14,TPR);           /* IGNORE INTERRUPT AND RETURN */
0089         GEN(L  15,4(0,13));
0090         GEN(ST 14,12(0,15));
0091         RETURN;
0092         END;
0093         ELSE DO;           /* CALL PGMER DEFINED ERROR ROUTINE */
0094         RET=CONTAB(SINDX);
0095         OPSWL=TPL;
0096         OPSWR=TPR;
0097         DCL REGYS(12) PTR(31),SPTR REG(13) PTR(31),
PTBACK PTR(31) CTL(SPTR+4),REGT(10) PTR(31) CTL(PTBACK+32);
** W05 ** POSSIBLE CONFLICT IN USE OF REGISTER
0098         DO I5= 1 TO 10;
0099             REGYS(I5)=REGT(I5); /* STORE REGS 3-12 */
0100         END;
0101         DCL BLOOB PTR(31),R14 REG(14) FIXED(31) ;
** W05 ** POSSIBLE CONFLICT IN USE OF REGISTER
0102         GEN(L  14,@SAVC01+4);
0103         REGYS(11)=R14;
0104         REGYS(12)=ADDR(TPR);
0105         CALL ERRTN (PIE,REGYS);
** W06 ** DIMENSIONED ITEM NOT SUBSCRIPTED
0106         GC TC YC;           /* IF USER RETURNS */
0107         END;               /* BRANCH TO RESUME */
0108         FRPCL:ENTRY ;      /* ENTRY TO RESTORE SYSTEM SPIE */
0109         GEN(ST  2,RSAP);
0110         GEN(IC  2,XXAPICA);
0111         GEN(SLL 2,24);
0112         GEN(MVI XXAPICA,X'00');
0113         GEN(L   1,XXAPICA );
0114         GEN(SVC 14);
0115         GEN(SPM 2);
0116         GEN(L   2,RSAP);
0117         RETURN;
0118         END;

```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
48	BACK	STATIC, LOCAL, LABEL, INTERNAL 51, 57
101	BLOOB	AUTOMATIC, POINTER(31), INTERNAL, BOUNDARY(WORD,1)
45	BLOOP	AUTOMATIC, PCINTER(31), INTERNAL, BOUNDARY(WORD,1) 46, 47
63	BTEMP	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 63, 65, 66, 66, 67, 67, 68, 68, 69, 70, 72, 73, 73, 74, 74, 75
2	BUF	AUTOMATIC, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 27, 28, 35, 36, 60, 69, 75, 78, 79, 80, 81
61	BUFIN	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 61, 69, 71, 71, 75, 76, 76
62	CLAP	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 62, 65, 72
82	CLOSE	STATIC, NONLOCAL, ENTRY, EXTERNAL 82
18	CONTAB	(15), STATIC, LOCAL, POINTER(31), EXTERNAL, BOUNDARY(WORD,1) 22, 40, 58, 86, 94
10	DUMMY	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 10, 11
37	DUTE	STATIC, LOCAL, LABEL, INTERNAL 37, 41
43	ERR	STATIC, NONLOCAL, ENTRY, EXTERNAL 43
108	ERRCL	STATIC, NONLOCAL, ENTRY, EXTERNAL 108
1	FRRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
7	ERRMES	STATIC, LOCAL, CHARACTER(50), INTERNAL, BOUNDARY(BYTE,1) 35
85	ERRTN	BASED, ENTRY, INTERNAL 105
1	I	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 5, 5, 24
1	II	PARAMETER, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 5, 5, 10, 20

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
32	INDEX	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 32, 33, 33, 40
98	I5	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 98, 99, 99
31	J	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 31, 32, 40
21	JJ	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 21, 22
12	LL1	STATIC, LOCAL, LABEL, INTERNAL 12, 17
4	MESS	(15), STATIC, LOCAL, CHARACTER(26), INTERNAL, BOUNDARY(BYTE,1) 60
47	OPSWL	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 49, 95
47	OPSWR	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 50, 51, 96
47	PICAA	IN PIE, POINTER(31), INTERNAL, BOUNDARY(WORD,1)
47	PIE	STRUCTURE, BASED, CHARACTER(32), INTERNAL, BOUNDARY(DWORD,1) 105
28	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 28, 36, 78, 80, 81
48	PSWTMP	STRUCTURE, AUTOMATIC, CHARACTER(8), INTERNAL, BOUNDARY(DWORD,1)
97	PTBACK	BASED, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 97
97	REGT	(10), BASED, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 99
97	REGYS	(12), AUTOMATIC, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 99, 103, 104, 105
85	RET	AUTOMATIC, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 85, 94
9	RSAP	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 9, 19
8	RSAVE	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 8, 17

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
47	RO	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
6	R1	REGISTER(1), PCINTER(31), INTERNAL, BOUNDARY(WORD,1) 8, 17, 46
47	R1A	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
101	R14	REGISTER(14), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 103
47	R14A	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
47	R15A	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
6	R2	REGISTER(2), POINTER(31), INTERNAL, BOUNDARY(WORD,1) 5, 19
47	R2A	IN PIE, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
55	SINDX	AUTOMATIC, FIXED(15), INTERNAL, BOUNDARY(HWORD,1) 57, 58, 60, 86, 94
97	SPTR	REGISTER(13), POINTER(31), INTERNAL, BOUNDARY(WORD,1) 97
24	TERM	AUTOMATIC, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 24, 25, 25, 31
48	TPL	IN PSWTMP, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 49, 56, 95
56	TPLA	BASED, FIXED(15), INTERNAL, BOUNDARY(HWORD,1) 57
48	TPR	IN PSWTMP, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 50, 64, 96, 104
64	TPRZ	(3), BASED, PCINTER(8), INTERNAL, BOUNDARY(BYTE,1) 65, 72
3	TRAN	(16), STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 69, 75
1	X	(30), PARAMETER, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 5, 5, 32, 40
5	XXAPICA	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
43	XXX	PARAMETER, PCINTER(31), INTERNAL, BOUNDARY(WORD,1) 44, 44
88	YC	STATIC, LOCAL, LABEL, INTERNAL 88, 106

*** PROC. ERRINT HAD 006 ERRORS
*** THE FOLLOWING STATEMENT(S) HAD ERRORS
0003,0004,0006,0097,0101,0105

VI. EDIT1. Function

EDIT is a library routine intended for use with BSL programs which may be called to perform input and output conversions analagous to the PL/I GET/PUT EDIT statements. Options are provided to perform GET/PUT as an I/O function or as with the PL/I STRING option. These options are obtained by the use of different entry points invoked by a CALL statement.

2. Entry points and functions

The entry points and their functions are as follows:

A. GET (FORMAT, DATA₁, DATA₂, ... , DATA_n);

This is analagous to the PL/I GET EDIT statement and reads a card from file with the DD name BSLIN, converts the fields specified in the FORMAT specification to the appropriate internal forms and assigns them to the corresponding data items. The method of specifying formats and the correspondence between format items and data arguments will be discussed later.

B. GETS (FORMAT, STRING, DATA₁, DATA₂, ... , DATA_n);

This has the same function as a PL/I GET statement with the STRING option. It works in the same way as GET, above, except that no read operation takes place. Instead, the STRING, defined by the second argument, is considered to be the source of the data to be converted.

C. PUT (FORMAT, DATA₁, DATA₂, ... , DATA_n);

This corresponds to the PL/I PUT EDIT statement with the PRINT option. The data items are converted according to the FORMAT specification and the resulting character string is written on the file with the DD name BSLOUT. An ASA carriage control character precedes each record, and is initialized to blank, which will result in single spacing. The user has the capability of skipping lines or page ejecting through the use of the S, P, or L format items, which will be described

later.

- D. PUTS (FORMAT, STRING, DATA₁, DATA₂, ... , DATA_n);

This will have an effect similar to the PL/I PUT EDIT with the STRING option. It is the direct reverse of the function performed by GETS, described above. There is no carriage control field as in PUT, above, and the first position of the resulting string will be the first character in the STRING.

- E. TABSET (N, TAB₁, TAB₂, ... , TAB_n);

This entry is used for initializing tab positions which will be used in conjunction with the T format item. The first argument, N, is the number of tabs in the list, and TAB₁, TAB₂, etc. are the tab positions to be set. The maximum number of tabs is 63. If no call has been made to TABSET, default tab positions will be: 1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, and 111. The use of tabs will be described in detail later.

3. FORMAT description

There are two methods of specifying a FORMAT list to be used in EDIT conversions. The choice of one over the other will be based on considerations such as ease of use, facility, and performance.

A. Free form format

The method which is the easiest to use is to provide a character string which contains the list of format items. The syntax of this format list is as follows:

$$\left\{ \begin{array}{l} \text{item} \\ \text{replication item} \\ \text{replication (list)} \end{array} \right\} \left[\begin{array}{l} \text{,item} \\ \text{,replication item} \\ \text{,replication (list)} \end{array} \right] \dots \text{terminator}$$

where: item is any of the format items described below, replication is either an integer constant or a DO format item,

list has the same description as the format list, and terminator is a period. The terminator is essential, because the EDIT program has no other way of determining the number of data items in the data list.

Blanks are allowed between elements of the list. There are two general types of format items: control format items and data format items. Control format items perform the functions of setting the pointer to be used in accessing fields in the input or output stream, skipping to the next input or output record, skipping to the next page (in the case of PUT), or providing replication information, so that a format item or a format list will be used repeatedly. Data format items describe the type of conversion to be performed on the corresponding data item, and in certain cases, attribute information about the data item. There must be a data item in the argument list to correspond to each data format item in the format list (taking replication into consideration).

B. CONTROL format items

X(n) This will cause the pointer to be advanced n positions to the right of its current position. Intervening positions, in the case of PUT or PUTS, will be filled with blanks. If n is greater than 80 when doing input (GET) the next record will be read in and the pointer positioned to n-80. If this is still greater than 80, the process is continued until a position between one and 80 is obtained. Similarly, on output (PUT), if n is greater than 120, the current record is output and the pointer is positioned to n-120. This process is also repeated, as in GET, until a pointer value is obtained which is less than 120. When using GETS or PUTS, however, the pointer is simply advanced n positions to the right. It is the user's responsibility to ensure that the pointer does not advance beyond the end of the STRING.

C(n) This will cause the pointer to be advanced to position n in the input/output stream. If GET or PUT are being used and n is greater than 80 or 120, respectively, records will be skipped in the same manner as when X(n) causes the pointer to be advanced beyond the end of the record.

If n is less than the current position of the pointer, when using GET or PUT, a skip to the next record is implied. That is, n will be treated in the same way as if it were n+80 in the case of GET or n+120 in the case of PUT. For GETS or PUTS the quantity n will be used to advance the pointer n positions in the string. Backward setting of the pointer in this manner does not cause blanking.

S This will cause the current record to be written, in the case of PUT, and the pointer set to position one of a new line. Similarly, in the case of GET, a new record will be read and the pointer set to position one of this new record. When using GETS or PUTS, this format item is ignored.

P This will cause the current record to be written, in the case of PUT, and the next line to begin on the succeeding page. In the case of GET, this format item is treated the same as S, above. When using GETS or PUTS, this format item is ignored.

L(n) This format item is intended for use with PUT. It will have the same effect as the P format item followed by n-1 S format items. That is, the next line will begin on line n of a new page.

When using GET, this format item will be treated in the same way as the S format item and n is ignored. In the case of GETS or PUTS, this format item is ignored.

T This causes the pointer to be advanced to the next tab position which is greater than the current position of the pointer. Tabs may be set through the use of the TABSET entry point, described above, or the default tabs may be used. There is only one tab table in the EDIT program, which is used for both input and output. If the

user requires different tabbing for these, he must call TABSET each time he desires different setting of the tabs.

If the use of a tab would result in the pointer being advanced beyond the end of the current record, when using GET or PUT, the current record is dispensed with and the pointer is set to the value of TAB1 in the next record. Similarly, if the pointer has already been positioned past the value of the last tab in the table, the current record will be dispensed with and the pointer set to TAB1 in the next record.

C. Replication format items

Replication is technically not a type of format item, but it does perform a control function and is therefore, included as a special case of the control format items. There are two types of replication supported.

n This will cause the immediately following format item or format list enclosed in parentheses to be used repeatedly n times. Each time a data format item is encountered, the next data item in the argument list is accessed. In other words, the effect of using this type of replication is exactly the same as if the immediately adjacent format item or format list were repeated n times. For example:

2 F(6,31)

is exactly equivalent to:

F(6,31) , F(6,31)

DO(n,d) This type of replication is intended to provide the capability of inputting or outputting an entire data aggregate, without having to itemize each element. Its effect is similar to the "implied DO" of PL/I, except that the DO specification must appear in the format list, rather than in the data list as in PL/I. The operation of the DO format item is as follows:

The same data item is referenced n times and the immediately adjacent format item is used repeatedly n times in conjunction with this data. Each successive time the data item is referenced, the address of the data being accessed is incremented by d bytes, in order to obtain the next element of the aggregate. The user must be careful to know the mapping of his data aggregate when specifying this parameter.

The DO format item may also be used in conjunction with a format list enclosed in parentheses. However, this is constrained by the fact that the associated format list may contain only one data format item, although it may have any number of control format items. If this rule is violated, the results are unpredictable. Also, nested DO format items may sometimes have unpredictable results and should be avoided. It should also be noted that a DO format item with the d specification equal zero is equivalent to the simple replication described above. That is:

DC (2,0) F (6,31)

is equivalent to:

2F (6,31)

The following examples demonstrate the use of the DO format item:

- 1 DCL V (5) FIXED (31) ;
- 2 CALL PUT ('DO (5,4) (X (2) , F (6,31)) .', V) ;
- 3 CALL PUT ('5 (X (2) , F (6,31)) .', V (1) , V (2) , V (3) , V (4) , V (5)) ;
- 4 CALL PUT ('X (2) , F (6,31) , X (2) , F (6,31) , X (2) , F (6,31) ,
X (2) , F (6,31) , X (2) , F (6,31) .', V (1) , V (2) , V (3) ,
V (4) , V (5)) ;
- 6 CALL PUT ('DO (5,0) (X (2) , F (6,31)) .', V (1) , V (2) ,
V (3) , V (4) , V (5)) ;
- 7 CALL PUT ('DO (5,4) (DO (5,4) F (6,3)) .', V) ;

In the above example, statements numbered 2 through 6 will have the identical effect of putting out the entire array, V, according to the format specification, F(6,31), and with two blanks between each field.

Statement number 7, however, is not correct, since nested DO specifications are used. The effect, in this case, will be that the inner DO specification will be completed, and an attempt will be made to access a new argument for the second iteration of the outer DO specification. That is, for each level of DO specification, a new data item is required. Thus, in the above example, if five arrays had been mentioned in the data list, the result would have been that all five arrays would have been output in the same manner. If the user desires to take advantage of the nested DO facility, he should make sure he understands this relationship.

D. Data format items

A(n) The associated data item is treated as a character string and n characters are moved from it, in the case of output, to the field in the output string starting at the current position of the pointer. If PUT is being used, and there are not enough spaces remaining on the current line to contain this character string, the current line is output and the field will begin in position one of the next line. In the case of input, n characters are moved from the input string to the associated data item. If there are not n characters remaining in the current record in the input stream, in the case of GET, a new record is used and the n characters are accessed beginning with position one of the new record.

The pointer is then advanced n positions to the right in preparation for the next field.

B(n) The associated data item is treated as a bit string of length(n) and, in the case of output, each bit is converted to its character representation ('1' or '0') and placed into the output string. In the case of input, n characters are accessed from the input string; each character

is expected to be either a '1' or a '0' and these will be converted to a bit '1'B or '0'B and placed in the corresponding position in the bit data item. The same rules apply with regard to the field not fitting in the remaining part of the record as in the A format item.

H(n) The associated data item will be converted to hexadecimal, in the case of output, and placed into a field of length (n). The size of the data item is considered to be n/2 bytes. On input, the field in the input string is considered to contain any combination of the characters '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', or 'F'. Each respective character will be converted to a four bit binary value in the range '0000'B to '1111'B and placed the data item. Any characters other than those mentioned above will produce unpredictable results.

F(n,d) The associated data item will be treated as a fixed point arithmetic variable of precision (d). On output, this variable will be converted to decimal (character) and placed right justified in the field of length (n) in the output string. If the number is negative, a minus sign will appear to the left of the most significant digit. If the field is too short to contain the converted number and sign, if any, truncation will be performed on the left.

On input, the field in the input string is considered to contain numeric characters, optionally preceded by a plus or minus sign. If the sign is present, it must immediately precede the most significant digit. The numeric characters are converted to binary and assigned to the associated data item. If illegal characters are present, the EDIT program will write a diagnostic message on BSLOUT.

Note: Trailing blanks within the field will be treated as zeroes.

U(n,d) This format item provides the facility for the user to specify his own conversion algorithm for a given data item and the current position in the input/output string. There are two data items

associated with each U format item. The first is a pointer to the user's conversion routine and the second is the data item which is to be referenced by the conversion. The n and d specifications are passed along to the user program to be used as desired. It is recommended that the n specification be the length of the field in the input/output string, as the EDIT program will use this to determine whether to skip to the next record. The d specification is optional and, if present, may contain any numeric information of use to the user program.

The user program must be an external procedure and be set up to receive four parameters:

P, N, D, Q in that order, where:

P is a pointer to the beginning of the current field in the input/output string. When the user routine has completed its conversion, it should update P to position the pointer past the given field.

N is the contents of the n specification in the U format item.

D is the contents of the d specification in the U format item.

Q is the data item which is to be converted or to receive the converted result.

P and Q are defined as POINTER(31) and

N and D are defined as FIXED(31).

Since there are two data items associated with the U format item, it is not legal to use it in conjunction with the DO format item. Any attempt to do so will obtain unpredictable results. It is possible to obtain this capability, however, if the user program is so designed. The d specification, for instance, could be used to tell the user routine the number of elements in an array.

The following is an example of the use of the U format item:

```

DECLARE P PTR INIT (ADDR (MYCONV) ),
        USER ENTRY CTL (P) ,
        DATA FIXED (31) ;
CALL PUT ('U(6,25) .',USER,DATA) ;

```

In this example, MYCONV is the name of an external procedure which is set up to receive the arguments described previously. The value of the arguments in this case, will be:

```

ADDR (output buffer) , 6,25, DATA

```

If the recommended practice has been observed, the user routine will have changed the first parameter to ADDR (output buffer) + 6 upon completion.

4. Coded format lists

The second method of specifying a format list is to provide a table which has been initialized to contain internal codes representing format types and binary values of the n and d fields. The table must be defined as:

```

DCL 1 FORMAT (n) ,
    2 T PTR (8) ,
    2 N PTR (8) ,
    2 D PTR (8) ;

```

The names, FORMAT, T, N, and D are, of course, arbitrary, but the structuring must be identical to the example. The dimension of the table is up to the user.

When using this method of format description, T(1) must be equal to binary zero. This is the only way the EDIT program can determine whether the first or second method is being used. Succeeding values of T depend upon the format item being used. The internal codes for each of the format items are as follows:

<u>Format Item</u>	<u>Internal Code</u>
A	1
B	2
F	3
H	4
U	5
X	6
(7
S	8
P	9
T	10
L	11

In addition, replication (both kinds) is represented by the character '*' (or internally '5C'X). For simple replication, the corresponding D element will be zero, and for the DO type of replication, this will contain the d specification.

Grouping of format lists, as in replication, are represented by an entry in the table in which T contains the character '(' (or internally '4D'X), and D contains a level number. The level number should be one greater than that of the previous group which has not been closed out. The group is closed by an entry in the table in which T contains the character ')' (internally '5D'X) and D contains the same level number as the entry specifying the beginning of the group. It is not important that these level number conventions be strictly adhered to, but this is suggested for ease of debugging and documentation. It is important, however, that the level number be non-zero, as a zero value will confuse the EDIT program's housekeeping.

The end of the format list is represented as an entry in the table with T containing the character '.' (internally '4B'X).

An example of the two methods of format description.

```

DECLARE FMT1 CHAR(21) INIT('C(3),2(F(6,31),X(2)).'),
1 FMT2(8), /* C * ( F X ) . */
T PTR(8) INIT(0,7,'5C'X,'4D'X,3,6,'5D'X,'4B'X),
N PTR(8) INIT(0,3,2,1,6,2,1,0),
D PTR(8) INIT(0,0,0,31,0,0,0);

```

The variables FMT1 and FMT2 could be used interchangeably to obtain the identical result.

The choice between the first and second methods of defining format lists may be dependent upon the following considerations:

- A. The user will probably find that the first method is easier to use and provides a more graphic documentation for his program.
- B. The second method may be preferable if the user is interested in reducing processing time to a minimum, since it is not necessary to scan a free-form list.
- C. The second method may be required if the user program is very large and there is not enough memory to contain the SCAN module. The SCAN module is a separate program and may be excluded from the load module at line-edit time by specifying the NCAL option and providing INCLUDE cards for the EDIT program and the READ (BSL INPUT/OUTPUT) program, which is called by EDIT.
- D. If it is desired to change any of the format specifications dynamically, this is easier done when the second method is used. It will be noted, however, that when the first method is used, it is very simple to read a format list into a character string and then use this format list in a succeeding call to EDIT. Thus the user can design his program such that his input records are self-defining.

5. Restrictions and limitations

- A. If the first method of format description is used (character string), the maximum length of the string is 256 characters. For the second method there is no limit.
- B. When the STRING option is used, it is the user's responsibility to insure that the pointer does not advance beyond the end of the user's string.
- C. Character string literals passed as arguments, such as format lists, may not exceed 53 characters. Character string variables may contain up to 256 characters.
- D. The n specification for any of the data format items may not exceed 80 for GET or 120 for PUT. When using GETS or PUTS, there is no limit.
- E. Each call to GET or PUT represents at least one I/O operation. That is, each call to GET causes a record to be read before processing takes place, and each call to PUT causes the current contents of the buffer to be written when processing has been completed. If the user is interested in building up a record through several calls to EDIT, he can do so by using the STRING option.
- F. If the user does not know how many records appear on his input data set and wishes to test for end of data, there is an external variable named EOF, which is defined as FIXED(31) which may be tested. EOF will be set equal to 0 until the end of data is reached, when it will be set equal to one.

6. Error conditions detected by EDIT

There are several error conditions which can be diagnosed by the EDIT program. When one of these is encountered, a suitable message will be written on BSLOUT. Following this will be the current contents of the input/output string for as far as the pointer has advanced. After encountering an error condition, no attempt will be made to recover and continue processing; return will be made to the user program.

Diagnostic messages will be of the form:

'*****HAS OCCURRED WHILE ATTEMPTING EDIT CONVERSION'

where '*****' will be replaced by a specific diagnostic such as one of the following:

'AN ILLEGAL FORMAT ITEM TYPE' - This indicates that a character was encountered which was not one of the recognized format items.

'PROGRAM ERROR TYPE*' - This indicates that an interrupt occurred within the EDIT program. The '*' will be replaced by a hexadecimal interruption code obtained from the PSW. The possible interruption codes and the probable cause for each are given below.

1 Illegal operation - the probable cause is that a branch has been made to a module which has not been loaded. Possible candidates are the SCAN or READ modules, or the user's conversion routine when using a U format item.

5 Addressing - the probable cause is that there are more data format items than data items in the argument list. Be sure to check replication in the format list.

6 Specification - the probable cause is the use of the F format item where the d specification does not agree with the precision of the data item with which it is matched.

7 Data - this can only occur when, using the F format item for input, a non-numeric character has been encountered in the field.

8 Fixed-point overflow - This will occur if the field being processed for input with the F format item contains a number greater than 2,147,483,642 or less than - 2,147,483,642.

'ILLEGAL FORMAT LIST SYNTAX' - This indicates that the format list contains a syntactic error, such as a missing right parenthesis, comma, or terminator. Note that the comma delimiter should not appear

between replication and the format item or list
with which it is associated.


```

0003     RESTRICT(2,3,4,5);
0004 TABSET: ENTRY; /* ENTRY PCINT TO INITIALIZE TAB TABLE */
0005     GEN(ST 1,Q );
0006     R=ARG(1);
0007     N=FF&63; /* GET NUMBER OF TABS (MAX. IS 63) */
0008     TAB(1) = N;
0009     DO J=2 TO N+1; /* LOOP THROUGH ARG LIST AND TAB TABLE */
0010     R=ARG(J);
0011     TAB(J)=FF; /* SET TAB ACCORDING TO CORRESPONDING ARGUMENT*/
0012     END;
0013     RETURN;
0014 PUT: ENTRY(FMT,DATA);
0015     FUNC = 2;
0016     OPT=0;
0017     BUFR = '';
0018     GO TO PROCESS;
0019 GET: ENTRY(FMT,DATA);
0020     FUNC = 2;
0021     OPT=12;
0022     CALL READ(BUF);
0023     GO TO PROCESS;
0024 PUTS: ENTRY(FMT,DATA);
0025     FUNC = 1;
0026     OPT=0;
0027     GO TO PROCESS;
0028 GETS: ENTRY(FMT,DATA);
0029     FUNC = 1;
0030     OPT=12;
0031 PROCESS:
        GEN;
            ST 1,Q
            CNOP 2,4
            LA 1,#+12
            BALR 1,1
            DC X'0F'
            DC AL3(ERA)
            DC X'7FFF'
            SVC 14
            BALR 2,0
            SRL 2,24
            ST 1,XXAPICA
            STC 2,XXAPICA
            L 1,0
        $ENDGEN
0032     IF FUNC = 2 THEN DO;
0034         S = ADDR(BUF);
0035         DD=1;
0036         END;
0037     ELSE DO;
0038         S = ARG(2);
0039         DD=2;
0040         END;
0041     N=0;
0042     P = ADDR(FMT);
0043     J=1;

```

```

0044     IF T(1) .EQ. '00'X THEN DO; CALL SCAN(P); J=0;
0048     END;
0049     K = 1; I=1;
0051 ITER: J= J+1; T1 = T(J);
0053     IF T1 = '*' THEN DO;
0055         K = K+1;
0056         REP(K) = W(J);
0057         IMDO(K)=D(J);
0058         FIRST(K)=1;
0059         START(K)= J;
0060         ND(K) = 0;
0061         GO TO ITER;
0062     END;
0063     IF T1 = '(' THEN DO;
0065         ND(K) = W(J);
0066         GO TO ITER;
0067     END;
0068     IF T1 = ')' THEN DO;
0070 TSTK:     IF REP(K)=1 THEN K=K-1;
0072             ELSE DO;
0073                 REP(K)=REP(K)-1;
0074                 J = START(K);
0075                 IF IMDO(K) .EQ. 0 THEN FIRST(K)=0;
0077             END;
0078             GO TO ITER;
0079     END;
0080     IF T1 = '.' THEN DO;
0082         IF OPT+FUNC=2 THEN CALL PRINT(BUFR);
0084         GO TO EPI;
0085     END;
0086     IF T1 = '?' THEN DO;
0088         FRMSG(2:32)='ILLEGAL FORMAT LIST SYNTAX';
0089         GO TO ERROR;
0090     END;
0091     IF T1<6 THEN DO; /* IF IT IS A DATA FORMAT ITEM */
0093         IF FIRST(K)=1 THEN DO; /* FIRST ITERATION OF AN */
0095             /* IMPLIED DO ( OR ELSE NO IMPLIED DO ) */
0096             DD=DD+1; /*STEP TO NEXT ARGUMENT*/
0097             R =ARG(DD);
0098             END;
0099             ELSE R=R+IMDO(K); /* GET NEXT ELEMENT*/
0100             IF FUNC=2 THEN /* GET OR PUT (NOT STRING) */
0101                 IF OPT=0 THEN /* PUT */
0102                     DO; /* TEST FOR LINE SPILL */
0104                         IF I+W(J)>121 THEN DO;
0106                             CALL PRINT(BUFR);
0107                             BUFR='';
0108                             I=1;
0109                             END;
0110                         ELSE; /* NULL */
0111                     ELSE DO; /* TEST FOR CARD SPILL*/
0112                         IF I+W(J)>81 THEN DO;
0113                             CALL READ(BUF);
0114                             I=1;
0115                         END;

```

```

0115             END;             .             END; END;
0118             IF T1>11 THEN GO TO ERR1;
0120             T1 = T1 + OPT;
0121             GO TO ACTION(T1);
0122 LOUT:        IF FUNC=1 THEN GO TO NOITM;
0124             CALL PRINT(BUFR);
0125             L=120*(W(J)-1)+1;
0126             BUFR='1';
0127             GO TO TSTL;
0128 XOUT:        N=I;             /* CHANGE X FORMAT TO C FORMAT */
0129 COUT:        L=W(J)+N; N=0;
0131             LL = L-1;
0132             IF L<I THEN IF FUNC=2 THEN DO; /* TEST FOR BACKWARD POSITION */
0135                 CALL PRINT(BUFR);
0136                 BUFR='';
0137                 END;
0138                 ELSE;
0139                 ELSE IF FUNC=1 THEN IMAGE(I:LL)=BLANKS;
0141 TSTL:        IF L>120 THEN IF FUNC=2 THEN DO;
0144                 CALL PRINT(BUFR);
0145                 BUFR='';
0146                 L=L-120;
0147                 GO TO TSTL;
0148                 END;
0149             I = L;
0150             GO TO TSTI;
0151 AOUT:        L = W(J);
0152             V1 = I+L-1;
0153             IMAGE(I:V1) = CDATA(1:L);
0154             GO TO INCRI;
0155 BOUT:        V = 8;
0156             L = W(J);
0157             DO KK = 1 TO L/8 +1;
0158                 MSK = '80*X;
0159                 NN = (KK-1)*8;
0160                 IF (NN+8) > L THEN V=L-NN;
0162                 DO JJ = 1 TO V;
0163                     T1 = CDATA(KK);
0164                     T1 = T1 & MSK;
0165                     LL = NN+JJ;
0166                     IF T1 = 0 THEN TMPS(LL) = '0';
0168                     ELSE TMPS(LL) = '1';
0169                 MSK = MSK/2;
0170             END;
0171             END;
0172 MVB:        V1 = I+L-1;
0173             IMAGE(I:V1) = TMPS(1:L);
0174             GO TO INCRI;
0175 HOUT:        L=W(J);
0176             V = (L-1)/2+1;
0177             DO KK = 1 TO V;
0178                 JJ = 2*KK-1;
0179                 T1 = CDATA(KK);
0180                 T2 = T1/16 +1;
0181                 TMPS(JJ) = HEX(T2);

```



```

0182          JJ=JJ+1;
0183          T2 =(T1 & '0F'X) +1;
0184 CLEAR:    TMPS(JJ) = HEX(T2);
0185          END;
0186          GO TO MVB;
0187 FOUT: L = D(J);
0188          IF L > 16 THEN R3 = FF;
0190          ELSE IF L > 8 THEN R3 = FH;
0192          ELSE R3 = P1;
0193          GEN(CVD 3,DEC );
0194          TMPS = PATTERN;
0195          GEN(LR 3,1 );
0196          GEN(LA 1, TMPS+15 );
0197          GEN(EDMK TMPS(16),DEC );
0198          GEN(BC 10,SP );
0199          GEN(BCTR 1,0 );
0200          GEN(MVI 0(1),X'60');
0201 SP: GEN(LR 1,3 );
0202          L = W(J);
0203          V1 = I+L-1;
0204          V2 = 17-L;
0205          IMAGE(I:V1) = TMPS(V2:16);
0206 INCRI: I = I+L;
0207 TSTI:
0208          GO TO NDITM;
0208 ISET: I=1;
0209          GO TO NDITM;
0210 POUT: CC = '1';
0211 SOUT: IF FUNC = 2 THEN GO TO NDITM;
0213          CALL PRINT(BUFR);
0214          CCTL = CC;
0215          BUF = '';
0216          CC = ' ';
0217          GO TO ISET;
0218 PIN:
0219 LIN:
0220 SIN: IF FUNC = 1 THEN GO TO NDITM;
0221          CALL READ(BUF);
0222          GO TO ISET;
0222 XIN: N=I; /* CHANGE X FORMAT TO C FGMAT */
0223 CIN: L=W(J)+N; N=0;
0225          IF L<1 THEN IF FUNC=2 THEN CALL READ(BUF);
0228 TSTC: IF L>80 THEN IF FUNC=2 THEN DO;
0231          CALL READ(BUF);
0232          L=L-80;
0233          GO TO TSTC;
0234          END;
0235          I = L;
0236          GO TO TSTI;
0237 AIN: L = W(J);
0238          V1= I+L-1;
0239          CDATA(1:L) = IMAGE(I:V1);
0240          GO TO INCRI;
0241 BIN: L = W(J);

```

```

0242      V = 8;
0243      DO KK = 1 TO L/8 +1;
0244          MSK = '80'X;
0245          NN = (KK-1)*8;
0246          IF (NN+8) > L THEN V = L-NN;
0248          T2 = 0;
0249          DO JJ=1 TO V;
0250              LL = NN+JJ+I-1;
0251              T1 = IMAGE(LL);
0252              IF T1 = '1' THEN T2 = T2 | MSK;
0254              MSK = MSK/2;
0255          END;
0256          CDATA(KK) = T2;
0257          END;
0258          GO TO INCR1;
0259 HIN:   L = W(J);
0260          V = (L-1)/2 +1;
0261          DO KK = 1 TO V;
0262              JJ = 2*KK+I-2;
0263              T1 = IMAGE(JJ);
0264              JJ = JJ+1;
0265              T2 = IMAGE(JJ);
0266              V1=(T1 && 'F0'X)/16;
0267              T1 = T1 & '0F'X;
0268              T1 = (T1+V1*3)*16;
0269              V2=(T2 && 'F0'X)/16;
0270              T2 = T2 & '0F'X;
0271              T2 = T2+V2*3+T1;
0272          CDATA(KK) = T2;
0273          END;
0274          GO TO INCR1;
0275 FIN:   L = W(J);
0276          R3 = L-1;
0277          R4 = ADDR(IMAGE)+I-1;
0278          SIGN = 'FC'X;
0279 SKPB:  IF R3<0 THEN DO;
0281              R3=0;
0282              GO TO SETF;
0283          END;
0284          IF C = ' ' THEN DO;
0286              R4 = R4+1;
0287              R3 = R3-1;
0288              GO TO SKPB;
0289          END;
0290          IF C = '+' THEN GO TO SKPSGN;
0292          IF C = '-' THEN DO;
0294              SIGN = 'FD'X;
0295 SKPSGN: R4 = R4+1;
0296          R3 = R3-1;
0297          END;
0298 PACKER: GFN(PACK DEC(8),0(0,4));
0299          IF R3>15 THEN DO;
0301              R4=R4+R3-15; /* CAN'T PACK MORE THAN 16 DIGITS */
0302              R3=15;
0303          END;

```

/* INITIALIZE SIGN POSITIVE */

/* SKIP LEADING BLANKS */

/* IF MINUS SIGN IS PRESENT */

/* SET MINUS SIGN MASK */

/* POSITION PAST SIGN CHAR */

/* CAN'T PACK MORE THAN 16 DIGITS */

/* MOVE STARTING ADDR TO THE RIGHT */

```

0304     GEN(EX      3,PACKER );
0305     GEN(OI      DEC+7,X'OF' );
0306     GEN(NC      DEC+7,SIGN  INSERT PROPER SIGN );
0307     GEN(CVB     3,DEC      );
0308     SETF:
0310         IF D(J) > 16 THEN FF = R3;
0311         ELSE IF D(J) > 8 THEN FH = R3;
0312         ELSE P1 = R3;
0313     GO TO INCR1;
0314     TOUT:
0315     TIN: DO KK=2 TO TAB(1)+1; /* SEARCH TAB TABLE */
0316         IF I<=TAB(KK) THEN DO;
0317             I=TAB(KK);
0318             GO TO NDITM;
0319         END;
0320     END;
0321     IF OPT=12 THEN GO TO GETAB; /* FALL THRU IMPLIES SKIP */
0322     IF FUNC=2 THEN DO; /* PUT (NOT STRING) */
0323         CALL PRINT(BUFR);
0324         BUFR='';
0325         I=TAB(2);
0326         END;
0327     GO TO NDITM;
0328     GETAB:IF FUNC=2 THEN DO; /* GET (NOT STRING) */
0329         CALL READ(BUF);
0330         I=TAB(2);
0331         END;
0332     GO TO NDITM;
0333     UOUT: /* USER DEFINED EDIT CONVERSION */
0334     UIN: PU=PUP;
0335         DD=DD+1; /* BUMP TO NEXT ARGUMENT */
0336         R=ARG(DC); /* GET ADDRESS OF DATA ITEM */
0337         L=S+I; /* GET ADDRESS OF FIELD IN STRING */
0338         CALL USER(L,W(J),D(J),R); /* PASS INFORMATION TO USER */
0339         I=L-S; /* UPDATE I ( USER SHOULD HAVE UPDATED L ) */
0340     NDITM:IF ND(K) = 0 THEN GO TO TSTK;
0341         ELSE GO TO ITER;
0342     ERR1:
0343     ERR: ERMSG(2:32)='AN ILLEGAL FORMAT ITEM TYPE ';
0344         GO TO ERROR;
0345     ERA:GEN(L      12,@AD01 );
0346     GEN(LA      13,@SAVCO1);
0347     GEN(L      11,=A(EDIT+6) );
0348     GEN(DROP    15 );
0349     ERMSG(2:32)='A PROGRAM ERROR, TYPE * ';
0350     GEN(SR      3,3 );
0351     GEN(IC      3,7(1) );
0352     R3 = R3 + 1;
0353     ERMSG(24)=HEX(R3);
0354     ERROR: CALL PRINT(ERMSG);
0355     IF FUNC=1 THEN BUFR='';
0356     PLOOP:IF I>120 THEN DO;
0357         BUF(1:120)=IMAGE;
0358         CALL PRINT(BUFR);
0359     END;

```

```
0364             BUFR='';
0365             I=I-120;
0366             GO TO PLOOP;
0367             END;
0368             BUF(1:I)=IMAGE;
0369             CALL PRINT(BUFR);
0370             CALL CLOSE;
0371 EPI:GEN(IC   2,XXAPICA );
0372       GEN(SLL 2,24   );
0373       GEN(MVI XXAPICA,X'00');
0374       GEN(L   1,XXAPICA );
0375       GEN(SVC 14     );
0376       GEN(SPM 2      );
0377             END;
```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
2	ACTION	(24), STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 121
2	AIN	STATIC, LOCAL, LABEL, INTERNAL 237, 2
2	AOUT	STATIC, LOCAL, LABEL, INTERNAL 151, 2
2	ARG	(50), BASED, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 6, 10, 38, 96, 338
2	BIN	STATIC, LOCAL, LABEL, INTERNAL 241, 2
2	BLANKS	IN WKS, CHARACTER(120), INTERNAL, BCUNDARY(BYTE,1) 140
2	ROUT	STATIC, LOCAL, LABEL, INTERNAL 155, 2
2	BUF	IN BUFR, CHARACTER(120), INTERNAL, BCUNDARY(BYTE,1) 22, 34, 112, 215, 220, 227, 231, 332, 362, 368
2	BUFR	STRUCTURE, STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(WORD,1) 17, 83, 104, 105, 124, 126, 135, 136, 144, 145, 213, 325, 326, 359, 363, 364, 365
2	C	BASED, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 284, 290, 292
2	CC	STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 210, 214, 216
2	CCTL	IN BUFR, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 214
2	CDATA	STRUCTURE, BASED, CHARACTER(4), INTERNAL, BOUNDARY(WORD,1) 153, 163, 179, 239, 256, 272
2	CIN	STATIC, LOCAL, LABEL, INTERNAL 223, 2
184	CLFAR	STATIC, LOCAL, LABEL, INTERNAL 184
370	CLOSE	STATIC, NONLOCAL, ENTRY, EXTERNAL 370
2	COUT	STATIC, LOCAL, LABEL, INTERNAL 129, 2
2	D	IN FORMAT, PCINTER(8), INTERNAL, BOUNDARY(BYTE,1)

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
		57, 187, 308, 310, 340
1	DATA	PARAMETER, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 2, 2, 14, 19, 24, 28
35	DD	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 35, 39, 95, 95, 96, 337, 337, 338
2	DEC	STATIC, LOCAL, CHARACTER(8), INTERNAL, BCUNDARY(DWORD,1)
1	EDIT	STATIC, NONLCCAL, ENTRY, EXTERNAL 1
84	EPI	STATIC, LOCAL, LABEL, INTERNAL 84, 371
348	ERA	STATIC, LOCAL, LABEL, INTERNAL 348
2	ERMSG	STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1) 88, 345, 352, 356, 357
345	ERR	STATIC, LOCAL, LABEL, INTERNAL 345, 2, 2, 2
89	ERROR	STATIC, LOCAL, LABEL, INTERNAL 89, 346, 357
119	ERR1	STATIC, LOCAL, LABEL, INTERNAL 119, 345
2	FF	IN CDATA, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 7, 11, 189, 309
2	FH	IN FF, FIXED(15), INTERNAL, BCUNDARY(HWORD,1) 191, 311
2	FIN	STATIC, LOCAL, LABEL, INTERNAL 275, 2
2	FIRST	(16), STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 58, 76, 93
1	FMT	PARAMETER, CHARACTER(256), INTERNAL, BOUNDARY(BYTE,1) 2, 2, 14, 19, 24, 28, 42
2	FORMAT	IN TABLE, (120), CHARACTER(3), INTERNAL, BCUNDARY(WORD,1)
2	FOUT	STATIC, LOCAL, LABEL, INTERNAL 187, 2
2	FUNC	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 15, 20, 25, 29, 32, 82, 99, 122, 133, 139, 142, 211, 218, 226, 229, 323, 330, 358

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
19	GET	STATIC, NONLOCAL, ENTRY, EXTERNAL 19
322	GETAB	STATIC, LOCAL, LABEL, INTERNAL 322, 330
28	GETS	STATIC, NONLOCAL, ENTRY, EXTERNAL 28
2	HEX	STATIC, LOCAL, CHARACTER(16), INTERNAL, BOUNDARY(BYTE,1) 181, 184, 356
2	HIN	STATIC, LOCAL, LABEL, INTERNAL 259, 2
2	HCUT	STATIC, LOCAL, LABEL, INTERNAL 175, 2
2	I	REGISTER(2), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 50, 102, 106, 110, 113, 128, 132, 140, 149, 152, 153, 172, 173, 203, 205, 206, 206 208, 222, 225, 235, 238, 239, 250, 262, 277, 315, 317, 327, 333, 339, 341, 360 365, 365, 368
2	IMAGE	BASED, CHARACTER(120), INTERNAL, BOUNDARY(BYTE,1) 140, 153, 173, 205, 239, 251, 263, 265, 277, 362, 368
2	IMDO	(16), STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 57, 75, 98
154	INCRI	STATIC, LOCAL, LABEL, INTERNAL 154, 174, 206, 240, 258, 274, 313
208	ISET	STATIC, LOCAL, LABEL, INTERNAL 208, 217, 221
51	ITFR	STATIC, LOCAL, LABEL, INTERNAL 51, 61, 66, 78, 344
2	J	REGISTER(5), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 9, 10, 11, 43, 47, 51, 51, 52, 56, 57, 59, 65, 74, 102, 110, 125, 129, 151, 156 175, 187, 202, 223, 237, 241, 259, 275, 308, 310, 340, 340
2	JJ	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 162, 165, 178, 181, 182, 182, 184, 249, 250, 262, 263, 264, 264, 265
2	K	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 49, 55, 55, 56, 57, 58, 59, 60, 65, 70, 71, 71, 73, 73, 74, 75, 76, 93, 98, 342
2	KK	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 157, 159, 163, 177, 178, 179, 243, 245, 256, 261, 262, 272, 314, 315, 317
2	L	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
		125, 129, 131, 132, 141, 146, 146, 149, 151, 152, 153, 156, 157, 160, 161, 172 173, 175, 176, 187, 188, 190, 202, 203, 204, 206, 223, 225, 228, 232, 232, 235 237, 238, 239, 241, 243, 246, 247, 259, 260, 275, 276, 339, 340, 341
218	LIN	STATIC, LOCAL, LABEL, INTERNAL 218, 2
131	LL	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 131, 140, 165, 167, 168, 250, 251
122	LOUT	STATIC, LOCAL, LABEL, INTERNAL 122, 2
158	MSK	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 158, 164, 169, 169, 244, 253, 254, 254
172	MVB	STATIC, LOCAL, LABEL, INTERNAL 172, 186
7	N	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 7, 8, 9, 41, 128, 129, 130, 222, 223, 224
2	ND	(16), STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 60, 65, 342
123	NDITM	STATIC, LOCAL, LABEL, INTERNAL 123, 207, 209, 212, 219, 318, 329, 335, 342
159	NN	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 159, 160, 161, 165, 245, 246, 247, 250
2	OPT	STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1) 16, 21, 26, 30, 82, 100, 120, 321
2	P	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1) 2, 42, 46
298	PACKER	STATIC, LOCAL, LABEL, INTERNAL 298
2	PATTERN	STATIC, LOCAL, CHARACTER(16), INTERNAL, BOUNDARY(BYTE,1) 194
2	PERR	STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1)
2	PIN	STATIC, LOCAL, LABEL, INTERNAL 218, 2
360	PLDOP	STATIC, LOCAL, LABEL, INTERNAL 360, 366
2	POUT	STATIC, LOCAL, LABEL, INTERNAL 210, 2

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
83	PRINT	STATIC, NONLOCAL, ENTRY, EXTERNAL 83, 104, 124, 135, 144, 213, 325, 357, 363, 369
18	PROCESS	STATIC, LOCAL, LABEL, INTERNAL 18, 23, 27, 31
2	PU	STATIC, LOCAL, POINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 336
2	PUP	BASED, POINTER(31), INTERNAL, BCUNDARY(WORD,1) 336
14	PUT	STATIC, NONLOCAL, ENTRY, EXTERNAL 14
24	PUTS	STATIC, NONLOCAL, ENTRY, EXTERNAL 24
2	P1	IN FH, POINTER(8), INTERNAL, BCUNDARY(BYTE,1) 192, 312
2	Q	STATIC, LOCAL, PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 2
2	R	STATIC, LOCAL, POINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 2, 6, 10, 56, 98, 98, 338, 340
22	READ	STATIC, NONLOCAL, ENTRY, EXTERNAL 22, 112, 220, 227, 231, 332
2	REP	(16), STATIC, LOCAL, POINTER(15), INTERNAL, BOUNDARY(HWORD,1) 56, 70, 73, 73
2	R3	REGISTER(3), FIXED(31), INTERNAL, BCUNDARY(WCRD,1) 189, 191, 192, 276, 279, 281, 287, 287, 296, 296, 299, 301, 302, 309, 311, 312 355, 355, 356
2	R4	REGISTER(4), PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 277, 286, 286, 295, 295, 301, 301
2	S	STATIC, LOCAL, PCINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 34, 38, 339, 341
46	SCAN	STATIC, NONLCCAL, ENTRY, EXTERNAL 46
282	SETF	STATIC, LOCAL, LABEL, INTERNAL 282, 308
2	SIGN	STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 278, 294

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
2	SIN	STATIC, LOCAL, LABEL, INTERNAL 218, 2
279	SKPB	STATIC, LOCAL, LABEL, INTERNAL 279, 288
291	SKPSGN	STATIC, LOCAL, LABEL, INTERNAL 291, 295
2	SOUT	STATIC, LOCAL, LABEL, INTERNAL 211, 2
201	SP	STATIC, LOCAL, LABEL, INTERNAL 201
2	SPCE	IN WKS, CHARACTER(120), INTERNAL, BOUNDARY(BYTE,1)
2	START	(16), STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 59, 74
2	T	IN FORMAT, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 44, 52
2	TAB	(64), STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 8, 11, 314, 315, 317, 327, 333
2	TABLE	STRUCTURE, BASED, CHARACTER(480), INTERNAL, BOUNDARY(WORD,1)
4	TARGET	STATIC, NONLOCAL, ENTRY, EXTERNAL 4
2	TIN	STATIC, LOCAL, LABEL, INTERNAL 314, 2
2	TMPS	IN WKS, CHARACTER(16), INTERNAL, BOUNDARY(BYTE,1) 167, 168, 173, 181, 184, 194, 205
2	TOUT	STATIC, LOCAL, LABEL, INTERNAL 314, 2
228	TSTC	STATIC, LOCAL, LABEL, INTERNAL 228, 233
150	TSTI	STATIC, LOCAL, LABEL, INTERNAL 150, 207, 236
70	TSTK	STATIC, LOCAL, LABEL, INTERNAL 70, 343
127	TSTL	STATIC, LOCAL, LABEL, INTERNAL 127, 141, 147
2	TI	STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1)

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
		52, 53, 63, 68, 80, 86, 91, 118, 120, 120, 121, 163, 164, 164, 166, 179, 180, 18 251, 252, 263, 266, 267, 267, 268, 268, 271
2	T2	STATIC, LOCAL, PCINTER(8), INTERNAL, BOUNDARY(BYTE,1) 180, 181, 183, 184, 248, 253, 253, 256, 265, 269, 270, 270, 271, 271, 272
2	UIN	STATIC, LOCAL, LABEL, INTERNAL 336, 2
2	UOUT	STATIC, LOCAL, LABEL, INTERNAL 336, 2
2	USER	BASED, ENTRY, INTERNAL 340
2	V	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 155, 161, 162, 176, 177, 242, 247, 249, 260, 261
2	V1	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 152, 153, 172, 173, 203, 205, 238, 239, 266, 268
2	V2	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 204, 205, 269, 271
2	W	IN FORMAT, PCINTER(8), INTERNAL, BOUNDARY(BYTE,1) 56, 65, 102, 110, 125, 129, 151, 156, 175, 202, 223, 237, 241, 259, 275, 340
2	WKS	STRUCTURE, STATIC, LOCAL, CHARACTER(256), INTERNAL, BOUNDARY(WORD,1)
2	XIN	STATIC, LOCAL, LABEL, INTERNAL 222, 2
2	XOUT	STATIC, LOCAL, LABEL, INTERNAL 128, 2
2	XXAPICA	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1)

*** PROC. EDIT HAD NO ERRORS

```

0001 SCAN: PROC(Q);
0002     DCL (P,Q) PTR,
           C CHAR(1) CTL(P),
           1 TABLE(120),
           2 T CHAR(1),
           2 W PTR(8),
           2 D PTR(8),
           (ITEM,REPL,BLNKS,NUM) ENTRY INTERNAL,
           R4 REGISTER(4),
           R3 REGISTER(3),
           (I,J,K,LEVEL,N,TRUE) FIXED(31),
           TC CHAR(1),
           DEC CHAR(8) BDY(DWORD),
           JJ PTR(8),
           TT CHAR(11) INIT('ABFHUXCSPTL');

0003     RESTRICT(3,4);
0004     I = 1;
0005     P=Q;
0006     T(1) = 'CO'X;
0007     TABLE(2:120) = TABLE(1:119);
** W06 ** DIMENSIONED ITEM NOT SUBSCRIPTED
** W06 ** DIMENSIONED ITEM NOT SUBSCRIPTED
0008     LEVEL=0;
0009 LIST: CALL REPL;           /* SCAN REPLICATION FACTOR, IF ANY */
0010     IF C = '(' THEN DO;    /* IF NOT A GROUP */
0011         CALL ITEM;        /* SCAN FORMAT ITEM */
0012         GO TO IFEND;     /* TEST FOR END OF LIST */
0013     END;
0014     T(I)=C;                /* GROUP BEGIN, SET TYPE CODE */
0015     LEVEL = LEVEL+1;      /* PUSH DOWN GROUP LEVEL */
0016     W(I) = LEVEL;        /* PUT LEVEL IN GROUP FORMAT */
0017     I=I+1;
0018     P=P+1;                /* ADVANCE POINTER */
0019     GO TO LIST;         /* SCAN CONTENTS OF GROUP */
0020 IFEND:CALL BLNKS; /* SCAN OFF BLANKS, LOOK FOR COMMA, PERIOD, OR ' */
0021     IF C=')' THEN DO;    /* END OF GROUP */
0022         T(I)=C;
0023         W(I)=LEVEL;
0024         LEVEL=LEVEL-1; /* POP UP GROUP LEVEL */
0025         I=I+1;
0026         P=P+1;
0027         CALL BLNKS;
0028         IF LEVEL<0 THEN GO TO SERR; /* UNMATCHED PARENS */
0029         GO TO IFEND;
0030     END;
0031     IF C=',' THEN DC;     /* FORMAT ITEM SEPARATOR */
0032     P=P+1;
0033     GO TO LIST;
0034     END;
0035     IF C='.' THEN DC;    /* END OF FORMAT LIST */
0036 LISTEND: T(I)='.';
0037     Q=ADDR(TABLE);
** W06 ** DIMENSIONED ITEM NOT SUBSCRIPTED
0038     RETURN;
0039     END;

```

```

0045 SERR: T(I)='?'; /* SET SCAN ERROR CODE */
0046 I=I+1;
0047 GO TO LISTEND; /* TERMINATE SCAN */
0048 BLNKS: PROC; /* SCAN OFF BLANKS */
0049 SKPB: IF C=' ' THEN DO;
0051 P=P+1;
0052 GO TO SKPB;
0053 END;
0054 END;
0055 NUM: PROC; /* SCAN A NUMBER AND CONVERT IT */
0056 TRUE=1; /* ASSUME NUMBER IS PRESENT */
0057 CALL BLNKS;
0058 TC = C&'0';
0059 IF TC ^= '0' THEN GO TO FLSE;
0061 R4=P;
0062 DO K=1 TO 256;
0063 P=P+1;
0064 TC = C&'0';
0065 IF TC = '0' THEN GO TO LOOPEND;
0067 R3=K-1; /* GET LENGTH MINUS ONE */
0068 GEN(EX 3,PACKER );
0069 GEN(CVB 3,DEC );
0070 N=R3;
0071 CALL BLNKS;
0072 RETURN;
0073 PACKER: GEN(PACK DEC(8),0(0,4));
0074 LOOPEND;;END;
0076 FLSE: TRUE=0;
0077 N=0;
0078 END;
0079 REPL: PROC; /* SCAN REPLICATION FACTOR, IF ANY */
0080 CALL NUM;
0081 IF TRUE ^= 1 THEN DO;
0083 IF C='D' THEN DO;
0085 P=P+1;
0086 IF C='0' THEN P=P+1;
0088 P=P-1;
0089 J=I;
0090 CALL ITEM;
0091 T(J)='*';
0092 END;
0093 RETURN;
0094 END;
0095 T(I) = '*';
0096 W(I) = N;
0097 I=I+1;
0098 END;
0099 ITEM: PROC; /* SCAN FORMAT ITEM */
0100 DO JJ=1 TO 11;
0101 IF C=TT(JJ) THEN DO;
0103 T(I)=JJ;
0104 GO TO BMPP;
0105 END;
0106 END;
0107 T(I)=C;

```

```
0108  BMPP: P=P+1;
0109      CALL BLNKS;
0110      IF C = '(' THEN GO TO ENDITM; /* S OR P FORMAT ITEM (NO W /D */
0111      P=P+1;
0112      CALL NUM;
0113      W(I)=N;
0114      CALL BLNKS;
0115      IF C=',' THEN DO;
0116          P = P+1;
0117          CALL NUM;
0118          D(I)=N;
0119          CALL BLNKS;
0120          END;
0121      IF C = ')' THEN P = P+1;
0122      ELSE DO;
0123  ITMERR:      I=I+1;
0124              T(I)='?';
0125              END;
0126      CALL BLNKS;
0127  ENDITM: I=I+1;
0128      END;
0129      END;
0130      END;
0131      END;
0132      END;
```

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE
2	BLNKS	STATIC, LOCAL, ENTRY, INTERNAL 21, 29, 48, 57, 71, 109, 115, 121, 129
104	BMPP	STATIC, LOCAL, LABEL, INTERNAL 104, 108
2	C	BASED, CHARACTER(1), INTERNAL, BCUNDARY(BYTE,1) 10, 15, 22, 24, 34, 39, 49, 58, 64, 83, 86, 101, 107, 110, 116, 123
2	D	IN TABLE, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 120
2	DEC	STATIC, LOCAL, CHARACTER(8), INTERNAL, BOUNDARY(DWORD,1)
111	ENDITM	STATIC, LOCAL, LABEL, INTERNAL 111, 130
60	FLSE	STATIC, LOCAL, LABEL, INTERNAL 60, 76
2	I	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 4, 15, 17, 18, 18, 24, 25, 27, 27, 41, 45, 46, 46, 89, 95, 96, 97, 97, 103, 107 114, 120, 126, 126, 127, 130, 130
13	IFEND	STATIC, LOCAL, LABEL, INTERNAL 13, 21, 32
2	ITEM	STATIC, LOCAL, ENTRY, INTERNAL 12, 90, 99
126	ITMERR	STATIC, LOCAL, LABEL, INTERNAL 126
2	J	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 89, 91
2	JJ	STATIC, LOCAL, POINTER(8), INTERNAL, BCUNDARY(BYTE,1) 100, 101, 103
2	K	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 62, 67
2	LEVEL	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 8, 16, 16, 17, 25, 26, 26, 30
9	LIST	STATIC, LOCAL, LABEL, INTERNAL 9, 20, 37
41	LISTEND	STATIC, LOCAL, LABEL, INTERNAL 41, 47
66	LOOPEND	STATIC, LCCAL, LABEL, INTERNAL

DCL'D IN	NAME	ATTRIBUTE AND CROSS REFERENCE TABLE 66, 74
2	N	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 70, 77, 96, 114, 120
2	NUM	STATIC, LOCAL, ENTRY, INTERNAL 55, 80, 113, 119
2	P	STATIC, LOCAL, POINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 5, 19, 19, 28, 28, 36, 36, 51, 51, 61, 63, 63, 85, 85, 87, 87, 88, 88, 108, 108 112, 112, 118, 118, 124, 124
73	PACKER	STATIC, LOCAL, LABEL, INTERNAL 73
1	Q	PARAMETER, POINTER(31), INTERNAL, BCUNDARY(WORD,1) 2, 2, 5, 42
2	REPL	STATIC, LOCAL, ENTRY, INTERNAL 9, 79
2	R3	REGISTER(3), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 67, 7C
2	R4	REGISTER(4), FIXED(31), INTERNAL, BOUNDARY(WORD,1) 61
1	SCAN	STATIC, NONLOCAL, ENTRY, EXTERNAL 1
31	SERR	STATIC, LOCAL, LABEL, INTERNAL 31, 45
49	SKPB	STATIC, LOCAL, LABEL, INTERNAL 49, 52
2	T	IN TABLE, CHARACTER(1), INTERNAL, BCUNDARY(BYTE,1) 6, 15, 24, 41, 45, 91, 95, 103, 107, 127
2	TABLE	STRUCTURE, (120), STATIC, LOCAL, CHARACTER(3), INTERNAL, BOUNDARY(WORD,1) 7, 7, 42
2	TC	STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1) 58, 59, 64, 65
2	TRUE	STATIC, LOCAL, FIXED(31), INTERNAL, BCUNDARY(WORD,1) 56, 76, 81
2	TT	STATIC, LOCAL, CHARACTER(11), INTERNAL, BOUNDARY(BYTE,1) 101
2	W	IN TABLE, POINTER(8), INTERNAL, BOUNDARY(BYTE,1) 17, 25, 96, 114

*** PROC. SCAN HAD 003 ERRORS
*** THE FOLLOWING STATEMENT(S) HAD ERRORS
0007,0007,0042

VII. APPENDIX I

ERROR MESSAGE TABLE

Error Message Text	Name of library routine issuing message	Page where message is explained
No DD statement for BSLIN. It cannot be opened.	BSL OBJECT TIME I/O	7
No DD statement for BSLOUT and BSLIN. They cannot be opened.	BSL OBJECT TIME I/O	7
Error on BSLIN. Data is not transmitted.	BSL OBJECT TIME I/O	7
No DD statement for BSLOUT and error on BSLIN.	BSL OBJECT TIME I/O	7
No DD statement for BSLPUNCH. It cannot be opened.	BSL OBJECT TIME I/O	7
No DD statement for BSLOUT and BSLPUNCH. They cannot be opened.	BSL OBJECT TIME I/O	7
Error on BSLPUNCH. Data is not transmitted.	BSL OBJECT TIME I/O	7
No DD statement for BSLOUT and error on BSLPUNCH.	BSL OBJECT TIME I/O	7
No DD statement for BSLOUT.	BSL OBJECT TIME I/O	7
Error on BSLOUT. Data is not transmitted.	BSL OBJECT TIME I/O	7
Illegal type specifications. No dump produced.	PDUMP	20
Illegal address or length specification.	PDUMP	20
Lower bound on target less than 1.	SUBSTR	34
Upper bound on target less than lower.	SUBSTR	34
Lower bound on source less than 1.	SUBSTR	34
Upper bound on source less than lower.	SUBSTR	34

Lower bound G.T. DCL'D length of target	SUBSTR	34
Lower bound G.T. DCL'D length of source.	SUBSTR	34
Illegal argument to SUBSTR pgm	SUBSTR	34
Illegal type it is ignored.	ERRINT	50
Incorrect routine count.	ERRINT	50
An illegal format item has occurred while attempting EDIT conversion	EDIT	75
Program error type * has occurred while attempting EDIT conversion.	EDIT	75
Illegal format list syntax has occurred while attempting EDIT conversion.	EDIT	76

INTERRUPT HANDLER

(BSL/10 Supplement to BSL Library Manual)

1. Function

The interrupt handler is part of both the OS/360 and DOS version of the BSL library. It provides the following information on a program check.

- 1). Type of interrupt and machine address.
- 2). A trace of all active save areas and the statement numbers for the call statements in the chain of active procedures.
- 3). An Abend dump.

The user enables the interrupt handler by calling IKETRCII and disables the interrupt handler by calling IKETRCID.

When an interrupt occurs several messages are printed.

- 1). XXX...XXX INTERRUPT AT LOCATION YYYYYY.
Where XXX...XXX is the type of interrupt and YYYYYY is the machine address of the interrupt.
- 2). A heading line:

SAVE AREA TRACE STARTING WITH PROCEDURE
THAT WAS INTERRUPTED.
- 3). The trace of the save area is then printed in the following format:

SAVE AREA ADDRESS XXXXXX. LAST STATEMENT
EXECUTED YYYYYY.

Where XXXXXX is save area address in a hexadecimal YYYYYY is a statement number of the last statement executed in the procedure.
- 4). An ABEND dump is then given. For OS, a users code of 444 is given.

After printing, control returns to the operating system.

Notes for DOS Users:

1. After calling IKETRCII any previously issued STXIT macros will be disabled.
2. No STXIT macro should be issued between the call to IKETRCII and the call to IKETRCID.
3. The BSL Library routines PDUMP, EDIT and SUBSTR issue STXIT macros. Calling one of those routines after calling IKETRCII will disable the interrupt handler.

2. Entry Points and Functions

A. IKETRCII.

This entry point issues the SPIE or STXIT macro to field program check interrupts. It saves the program mask and for OS, the address of PICA.

The entry point has one argument. It is the displacement from the start of the procedures save area of the half word where the statement number will be saved.

```
CALL IKETRCII (DISPLACEMENT);
```

IKETRCII returns to the user's program.

B. IKETRCID

This entry point disables the interrupt handler. It restores system information such as the program mask and for OS the PICA. It returns to the user's program. It has no arguments.

```
CALL IKETRCID;
```

For OS, any SPIE macro issued prior to the call of IKETRCII will be in effect after IKETRCID has been executed.

3. Example

```
$TRACE,ASSEM
A:PROCEDURE;
$TRACE ON      /*Turn Trace ON*/
CALL IKETRCII(2);    /*Enable Interrupt Handler*/
_____
_____
_____
CALL IKETRCID;      /*Disable Interrupt Handler*/
END;
```

TRACE must be given as one of the compiler options. The trace must be turned on prior to enabling the interrupt handler. If the trace option is not used, the interrupt handler gives no information other than a regular Abend dump.