34

VSE update

June 1999

In this issue

VSE Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: xephon@compuserve.com

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Subscriptions and back-issues

A year's subscription to *VSE Update*, comprising four quarterly issues, costs £100.00 in the UK, $150.00 in the USA and Canada, £106.00 in Europe, £112.00 in Australasia and Japan, and £110.50 elsewhere. In all cases the price includes postage. Individual issues starting with the March 1991 issue, are available separately to subscribers for £25.00 ($37.50) each including postage.

Editor

Fiona Hewitt

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, EXECs, and other contents of this journal before making any use of it.

Contributions

Articles published in *VSE Update* are paid for at the rate of £170 ($250) per 1000 words for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*.

VSE Update on-line

Code from *VSE Update* can be downloaded from our Web site at http://www.xephon.com; you will need the user-id shown on your address label.

Printed in England.

Submitting a batch job from a batch application

The two programs presented in this article were developed and tested under VSE/ESA Version 1.3 and are now running under Version 2.2.2.

SUBROUTINE B232RDR

The first program, B232RDR, allows a batch job to be submitted to the VSE/POWER reader queue by a simple subroutine call. One parameter must be passed to the subroutine; a second parameter is optional.

The first parameter is an input parameter, and contains the job that you want to submit to the VSE/POWER reader queue. Every job card must be prefixed by two fullwords. The length of the corresponding card must be stored in the first fullword, and the second fullword must be initialized with binary zeros. The last job card is indicated by at least one following X'FF' (high value).

The optional second parameter is an output parameter. It is a one-byte character string, and contains the return code of the spool operation. Possible values are:

- ‘0’ – Success
- ‘1’ – Error in first parameter detected by B232RDR
- ‘J’ – Spooling the job to VSE/POWER failed
- ‘P’ – Put service failed
- ‘S’ – XPCC SENDR failed
- ‘X’ – XPCC connect failed
- ‘V’ – XPCC identify (Logon) failed

B232RDR is called from a COBOL program as follows:

```
...

   Ø1 RC-B232RDR PIC X(1).

   Ø1 INFO-JOB.
      Ø2 FILLER  PIC S9(8) COMP VALUE +32.
      Ø2 FILLER  PIC S9(8) COMP VALUE ZERO.
      Ø2 FILLER  PIC X(32) VALUE '* $$ JOB JNM=OPERINFO,CLASS=P'.
      Ø2 FILLER  PIC S9(8) COMP VALUE +16.
      Ø2 FILLER  PIC S9(8) COMP VALUE ZERO.
      Ø2 FILLER  PIC X(16) VALUE '// JOB OPERINFO'.
      Ø2 FILLER  PIC S9(8) COMP VALUE +72.
      Ø2 FILLER  PIC S9(8) COMP VALUE ZERO.
      Ø2 FILLER  PIC X(Ø9) VALUE '// PAUSE'.
      Ø2 REASON  PIC X(63).
      Ø2 FILLER  PIC S9(8) COMP VALUE +4.
      Ø2 FILLER  PIC S9(8) COMP VALUE ZERO.
      Ø2 FILLER  PIC X(Ø4) VALUE '/&'.
      Ø2 FILLER  PIC S9(8) COMP VALUE +8.
      Ø2 FILLER  PIC S9(8) COMP VALUE ZERO.
      Ø2 FILLER  PIC X(Ø8) VALUE '* $$ EOJ'.
      Ø2 FILLER  PIC S9(8) COMP VALUE -1.
   ...

   PROCEDURE DIVISION.
   ...
   MOVE 'PROGRAM TEST : WRONG DATA FROM SYSIN' TO REASON.
   CALL 'B232RDR' USING INFO-JOB RC-B232RDR.
   IF RC-B232RDR NOT = 'Ø' ...
   ...
```

In this example, the following job is spooled to the VSE/POWER reader queue :

```
* $$ JOB JNM=OPERINFO,CLASS=P
   // JOB OPERINFO
   // PAUSE PROGRAM TEST : WRONG DATA FROM SYSIN
   /&
   * $$ EOJ
```

It may seem unusual to have two fullwords preceding every job card even though one halfword would have been enough. But the VSE/POWER spool interface needs these fullwords, and I decided to keep the subroutine simple instead of using macros like GETVIS and FREEVIS to create and free a new parameter structure to satisfy the VSE/POWER requirements.

To avoid alignment problems, the length stored in the first fullword is always a multiple of four.

B232RDR

```
TITLE 'B232RDR - SUBMIT JOB TO READER QUEUE'
B232RDR  CSECT
```

```
B232RDR  AMODE ANY
B232RDR  RMODE 24
**********************************************************************
*        CROSS-PARTITION COMMUNICATION CONTROL BLOCK (DSECT)
**********************************************************************
         MAPXPCCB
         EJECT
**********************************************************************
*        SPOOL PARAMETER LIST (SPL, DSECT)
**********************************************************************
SPLDSECT PWRSPL TYPE=MAP
         TITLE 'B232RDR - SUBMIT JOB TO READER QUEUE'
**********************************************************************
*        REGISTER EQUATES
**********************************************************************
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
**********************************************************************
* REGISTER USAGE:
*     R15 PROGRAM ENTRY POINT
*     R14 RETURN ADDRESS
*     R13 SAVE AREA ADDRESS
*     R12 WORK REGISTER, END OF BUFFER
*     R11 WORK REGISTER, ADDRESS OF USER DATA
*     R1Ø WORK REGISTER, LENGTH OF SPOOLED DATA
*     R9  BASE REGISTER
*     R8  WORK REGISTER, RETURN ADDRESS FROM SENDR
*     R7  ADDRESS OF SPOOL PARAMETER LIST (SPL)
*     R6  WORK REGISTER, RETURN ADDRESS USED BY INTERNAL SUBROUTINES
*     R5  ADDRESS OF SPOOLED DATA
*     R4  ADDRESS OF CROSS-PARTITION CONTROL BLOCK (XPCCB)
*     R3  ADDRESS OF RETURN CODE (OPTIONAL SECOND PARAMETER)
*     R2  WORK REGISTER, XPCC FUNCTION BYTE
*     R1  ADDRESS OF PARAMETER LIST, USED BY IBM MACROS
*     RØ
**********************************************************************
```

```
         EJECT
**********************************************************************
*        PROLOGUE USING CALLER'S ADDRESSING MODE (24 OR 31)
*        CAPPING (SEE VSE/ESA EXTENDED ADDRESSABILITY)
**********************************************************************
B232RDR  CSECT
         USING *,R15                   ESTABLISH ADDRESSABILITY
         STM   R14,R12,12(R13)         SAVE CALLER'S REGISTERS
         LA    R1Ø,SAVEAREA            ADDRESS OWN SAVE AREA
         ST    R1Ø,8(,R13)             SAVE ADDRESS OWN SAVE AREA
         LA    R9,START24              LOAD NEW BASE REGISTER
         LA    R11,STARTANY            EPILOGUE IN CALLER'S AMODE
         BSM   R11,R9                  SAVE OLD AMODE, START AMODE 24
START24  DS    ØH
         DROP  R15
         USING *,R9                    ESTABLISH ADDRESSABILITY
         ST    R11,SAVEAREA+72         SAVE ADDRESS EPILOGUE, OLD AMODE
         ST    R13,SAVEAREA+4          SAVE ADDRESS CALLER'S SAVE AREA
         LR    R13,R1Ø                 ESTABLISH OWN SAVE AREA
         EJECT
**********************************************************************
*        INITIALIZE CONTROL BLOCKS (XPPCB AND SPL)
**********************************************************************
         MVC   OWNXPCCB(IJBXEND-IJBXSTRT),KSTXPCCB
         LA    R6,SPLSEND              SPOOL PARAMETER LIST (SEND)
         LA    R1Ø,SPLSENDK            SPOOL PARAMETER LIST (CONSTANT)
         LA    R7,SPLGLEN              LENGTH OF SPOOL PARAMETER LIST
         LR    R11,R7                  LENGTH OF SPOOL PARAMETER LIST
         MVCL  R6,R1Ø                  INITIALIZE SPOOL PARAMETER LIST
         EJECT
**********************************************************************
*        LOAD ADDRESS OF PARAMETERS
**********************************************************************
         SR    R3,R3                   CLEAR REGISTER 3
         L     R5,Ø(,R1)               ADDRESS OF FIRST PARAMETER
         TM    Ø(R1),X'8Ø'             TEST OPTIONAL SECOND PARAMETER
         BO    ONEPARM                 ONLY ONE PARAMETER
         TM    4(R1),X'8Ø'             TEST OPTIONAL SECOND PARAMETER
         BZ    RETURN                  MORE THAN ONE OR OTHER ERROR
         L     R3,4(,R1)               ADDRESS OF SECOND PARAMETER
         MVI   POWRC,C'Ø'              DEFAULT RETURN CODE (SUCCESS)
ONEPARM  DS    ØH
         EJECT
**********************************************************************
*        TEST SUPPLIED FIRST PARAMETER
**********************************************************************
         LR    R1Ø,R5                  ADDRESS OF FIRST RECORD
         USING RECPRFIX,R1Ø            PREFIX, ESTABLISH ADDRESSABILITY
NXTLINE  DS    ØH
         CLI   RECCCODE,X'ØØ'          TEST COMMAND CODE
         BNE   DATAERR                 ERROR
         CLI   RECTYPE,RECTNORM         TEST RECORD TYPE
         BNE   DATAERR                  ERROR
         ICM   R1,B'1111',RECLOGNO      TEST LOGICAL NUMBER OF RECORD
         BNZ   DATAERR                  ERROR
         ICM   R1,B'ØØ11',RECLNGTH      LENGTH OF RECORD POSITIVE
         BNP   DATAERR                  NO, ERROR
         CH    R1,=H'8Ø'                MORE THAN 8Ø BYTES
         BH    DATAERR                  YES, ERROR
         LA    R1Ø,RECPRFXL(R1,R1Ø)     ADDRESS OF NEXT RECORD
         CLI   RECCCODE,X'FF'           END OF DATA
         BNE   NXTLINE                  NO, TEST NEXT RECORD
         DROP  R1Ø
         EJECT
**********************************************************************
*        LOAD ADDRESS OF CONTROL BLOCKS
**********************************************************************
         LA    R4,OWNXPCCB              ADDRESS XPCCB
         USING IJBXPCCB,R4              ESTABLISH ADDRESSABILITY
         LA    R11,IJBXSUSR             USER DATA, PART OF XPPCB
         USING PXUUSER,R11              ESTABLISH ADDRESSABILITY
         LA    R7,SPLSEND               ADDRESS SPOOL PARAMETER LIST
         USING SPLDSECT,R7              ESTABLISH ADDRESSABILITY
         EJECT
**********************************************************************
*        BUILD OWN APPLICATION ID
*        PREFIX JCL, LAST NUMBER OF CPU ID AND PARTITION ID
**********************************************************************
         LA    R2,L'CPUSTOR
         EXTRACT ID=CPUID,              CPU ID
               AREA=CPUSTOR,
               LEN=(R2)
         MVC   IJBXAPPL+3(1),CPUSTOR+3 STORE CPU ID
         OI    IJBXAPPL+3,X'FØ'         MAKE LAST NUMBER READABLE
         MVC   IJBXAPPL+4(2),CPUSTOR+8 STORE PARTITION ID
         EJECT
**********************************************************************
*        SETTING UP A COMMUNICATION PATH
*        PROGRAM IDENTIFICATION (LOGON)
**********************************************************************
         LA    R2,IJBXID                FUNCTION IDENTIFY
         BAL   R6,XPCCSERV              EXECUTE XPCC FUNCTION
         LTR   R15,R15                  TEST RETURN CODE
         BZ    CONNECT                  NO ERROR, CONNECT
         MVI   POWRC,C'V'               LOGON ERROR, RETURN CODE 'V'
         B     TERMIN                   TERMINATE COMMUNICATION
         EJECT
**********************************************************************
*        BUILDING A COMMUNICATION PATH (CONNECT)
**********************************************************************
CONNECT  DS    ØH
         LA    R2,IJBXCON               FUNCTION CONNECT
         BAL   R6,XPCCSERV              EXECUTE XPCC FUNCTION
         LTR   R15,R15                  TEST RETURN CODE
         BZ    SENDR                    NO ERROR, SEND JOB TO POWER RDR
         MVI   POWRC,C'X'               CONNECT ERROR, RETURN CODE 'X'
         B     DISCONN                  DISCONNECT COMMUNICATION
         EJECT
*********************************************************************
*        STARTING THE PUT SERVICE
*********************************************************************
SENDR    DS    ØH
         MVI   PXUBTYP,PXUBTSPL         TYPE OF PROGRAM'S SEND BUFFER
         BAL   R8,SENDRR                SENDING THE BUFFER
         EJECT
*********************************************************************
*        CHECK THE VSE/POWER RETURN CODE
*********************************************************************
         CLI   IJBXRUSR+PXPRETCD-PXPUSER,PXPRCOK
         BE    PUTOPEOK                 REQUEST SUCCESSFULLY OPENED
         MVI   POWRC,C'P'               SENDR ERROR, RETURN CODE 'P'
         B     DISCONN                  DISCONNECT COMMUNICATION
         EJECT
*********************************************************************
*        SEND THE BUFFER CONTAINING THE RECORDS
*********************************************************************
PUTOPEOK DS    ØH
         MVI   PXUBTYP,PXUBTNDB         SENDR BUFFER IS A DATA BUFFER
         L     R12,=F'65536'            ALLOWED LENGTH OF BUFFER (64K)
         AR    R12,R5                   ADDRESS END OF BUFFER
         CR    R1Ø,R12                  SUFFICIENT SPACE
         BNL   SEGMENT                  NO, SEND MULTIPLE BUFFERS
SENDLAST DS    ØH
         MVI   PXUACT1,PXUATEOD         INDICATE LAST BUFFER
         LA    R8,TSTPREPL              RETURN ADDRESS AFTER SENDR
SENDOTH  DS    ØH
         ST    R5,IJBXAD31              ADDRESS OF BUFFER
         OI    IJBXAD31,IJBXM8Ø         ONE BUFFER WITH SUPPLID LENGTH
         SLR   R1Ø,R5                   LENGTH OF BUFFER
         ST    R1Ø,IJBXBLN              STORE LENGTH OF BUFFER
         B     SENDRR                   PERFORM FUNCTION SENDR
         EJECT
*********************************************************************
*        SPLIT LARGE JOBS INTO MULTIPLE BUFFERS
*********************************************************************
NEXTSEGM DS    ØH
         AR    R5,R1Ø                   ADDRESS OF NEXT RECORD
         L     R12,=F'65536'            ALLOWED LENTH OF BUFFER (64K)
         AR    R12,R5                   ADDRESS END OF BUFFER
SEGMENT  DS    ØH
         LA    R8,NEXTSEGM              RETURN ADDRESS AFTER SENDRR
SEGON    DS    ØH
         LR    R1Ø,R5                   ADDRESS OF FIRST RECORD
         USING RECPRFIX,R10            PREFIX, ESTABLISH ADDRESSABILITY
NXTLINES DS    0H
         SR    R1,R1                   CLEAR REGISTER 1
         ICM   R1,B'0011',RECLNGTH     LOAD LENGTH OF RECORD
         LA    R1,RECPRFXL(R1,R10)     ADDRESS OF NEXT RECORD
         CR    R1,R12                  SUFFICIENT SPACE
         BNL   SENDOTH                 NO, SEND BUFFER
         LR    R10,R1                  ADDRESS OF NEXT RECORD
         CLI   RECCCODE,X'FF'          END OF DATA
         BNE   NXTLINES                NO, CHECK NEXT RECORD
         B     SENDLAST                SEND LAST BUFFER
         DROP  R10
         EJECT
*********************************************************************
*        CHECK THE VSE/POWER RETURN CODE
*********************************************************************
TSTPREPL DS    0H
         CLI   IJBXRUSR+PXPRETCD-PXPUSER,PXPRCOK
         BE    DISCONN                 REQUEST SUCCESSFULLY COMPLETED
         MVI   POWRC,C'J'              SENDR ERROR, RETURN CODE 'J'
         EJECT
*********************************************************************
*        DISCONNECT COMMUNICATION PATH
*********************************************************************
DISCONN  DS    0H
         LA    R2,IJBXDSC              FUNCTION DISCONNECT
         BAL   R6,XPCCSERV             EXECUTE XPCC FUNCTION
         EJECT
*********************************************************************
*        TERMINATE SPOOL ACCESS (LOGOFF)
*********************************************************************
TERMIN   DS    0H
         LA    R2,IJBXTRM              FUNCTION TERMINATE
         BAL   R6,XPCCSERV             EXECUTE XPCC FUNCTION
         EJECT
*********************************************************************
*        PASS RETURN CODE TO OPTIONAL SECOND PARAMETER
*********************************************************************
SENDRC   DS    0H
         LTR   R3,R3                   TEST OPTIONAL SECOND PARAMETER
         BZ    RETURN                  NO SECOND PARAMETER SUPPLIED
         MVC   0(L'POWRC,R3),POWRC     PASS RETURN CODE TO PARAMETER
         EJECT
*********************************************************************
*        RETURN CONTROL TO CALLING PROGRAM
*********************************************************************
RETURN   DS    0H
         L     R13,SAVEAREA+4          ADDRESS OF CALLER'S SAVE AREA
         L     R11,SAVEAREA+72         ADDRESS EPILOGUE, CALLER'S AMODE
         BSM   0,R11                   RESTORE CALLER'S AMODE
STARTANY DS    0H
```

```
         LM    R14,R12,12(R13)         RESTORE CALLER'S REGISTERS
         SR    R15,R15                 CLEAR REGISTER 15 (RETURN CODE)
         BR    R14                     RETURN TO CALLER
         EJECT
**********************************************************************
*        EXECUTE FUNCTION SENDR
*
*        HOW TO CALL IT:
*        BAL   R8,SENDRR
**********************************************************************
SENDRR   DS    ØH
         LA    R2,IJBXSNDR             FUNCTION SENDR
         BAL   R6,XPCCSERV             EXECUTE XPCC FUNCTION
         LTR   R15,R15                 TEST RETURN CODE
         BZ    WAITPOW                 NO ERROR, WAIT
SENDERR  DS    ØH
         MVI   POWRC,C'S'              RETURN CODE
         B     DISCONN                 DISCONNECT
         EJECT
**********************************************************************
*        WAIT FOR THE EVENT CONTROL BLOCK TO BE POSTED
**********************************************************************
WAITPOW  DS    ØH
         LA    R2,IJBXSECB             ADDRESS OF EVENT CONTROL BLOCK
         WAIT  (R2)                    WAIT FOR POWER ECB TO BE POSTED
         CLI   IJBXREAS,X'ØØ'          TEST REASON CODE
         BNE   SENDERR                 ERROR, BRANCH TO SENDERR
         BR    R8                      NO ERROR, RETURN
         EJECT
**********************************************************************
*        PERFORM XPCC FUNCTION
*
*        HOW TO CALL IT:
*        LA    R2,XPCC FUNCTION BYTE
*        BAL   R6,XPCCSERV
**********************************************************************
XPCCSERV DS    ØH
         XPCC  XPCCB=OWNXPCCB,                                         *
               FUNC=(R2)
         BR    R6                      RETURN
         EJECT
**********************************************************************
*        SUPPLIED DATA IN ERROR
**********************************************************************
DATAERR  DS    ØH
         MVI   POWRC,C'D'              RETURN CODE 'D' (DATA IN ERROR)
         B     SENDRC                  RETURN TO CALLER
         EJECT
**********************************************************************
*        CROSS-PARTITION COMMUNICATION CONTROL BLOCK (MODIFIED)
**********************************************************************
```

```
OWNXPCCB XPCCB APPL=JCL,                                            *
               BUFFER=(SPLSEND,SPLGLEN),                            *
               REPAREA=(SPLRECV,SPLGLEN),                           *
               TOAPPL=SYSPWR
         EJECT
*********************************************************************
*        CROSS-PARTITION COMMUNICATION CONTROL BLOCK (CONSTANT)
*********************************************************************
KSTXPCCB XPCCB APPL=JCL,                                            *
               BUFFER=(SPLSEND,SPLGLEN),                            *
               REPAREA=(SPLRECV,SPLGLEN),                           *
               TOAPPL=SYSPWR
         EJECT
*********************************************************************
*        SPOOL PARAMETER LIST (SPL) (CONSTANT)
*********************************************************************
SPLSENDK PWRSPL TYPE=GEN,                                           *
               PRFX=VPM,                                            *
               REQ=PUT,                                             *
               USERID=B232RDR,                                      *
               QUEUE=RDR
         TITLE 'B232RDR - SUBMIT JOB TO READER QUEUE'
*********************************************************************
*        SPOOL PARAMETER LIST (SPL), SEND AND RECEIVE
*********************************************************************
SPLSEND  DS    ØF                      SPOOL PARAMETER LIST, SEND
         ORG   *+SPLGLEN               LENGTH OF SPOOL PARAMETER LIST
SPLRECV  DS    ØF                      SPOOL PARAMETER LIST, RECEIVE
         ORG   *+SPLGLEN               LENGTH OF SPOOL PARAMETER LIST
         EJECT
*********************************************************************
*        OWN SAVE AREA
*********************************************************************
SAVEAREA DS    19F                     OWN SAVE AREA
         EJECT
*********************************************************************
*        WORKING STORAGE
*********************************************************************
POWRC    DS    C                       RETURN CODE
         DS    ØD
CPUSTOR  DC    X'FFFFFFFFFFFFFFFFFFFF' CPU NUMBER (EXTRACT MACRO)
         EJECT
         LTORG
         END
```

PROGRAM B234RDD

Because there are a lot of messages on the VSE/ESA console, we were finding that error messages were sometimes ignored if they didn’t force an operator to reply. But, at the same time, we didn’t want the corresponding partition to pause, especially at night. So we decided to assign a partition (CLASS=T) to display messages needing operator notification. In order to do this, I developed program B234RDD, which calls subroutine B232RDR from an Assembler program.

Program B234RDD is called at job control level by coding

```
// EXEC B234RDD,PARM='--- MESSAGE-STRING ---'
```

It submits the following job to the VSE/POWER reader queue:

```
* $$ JOB JNM=INFO,CLASS=T,PRI=9
   // JOB INFO ACCOUNTING-STRING
   // PAUSE --- MESSAGE-STRING ---
   /&
   * $$ EOJ
```

Only the first 60 characters of the PARM string are used, and the string can be omitted.

The following job shows how B234RDD can be used:

```
* $$ JOB JNM=NOWAIT
   // JOB NOWAIT
   // ON $RC > 4 GOTO ERRORRC
   // ON $CANCEL GOTO CANCELLED
   // ON $ABEND  GOTO ABENDED
   // SETPARM STEP=Ø1Ø
   // DLBL ...
   // EXEC ...
   ...
   // SETPARM STEP=99Ø
   ...
   // GOTO EXIT
   /. ERRORRC
   // SETPARM RETC=$RC
   // EXEC B234RDD,PARM='JOB NOWAIT ENDED WITH RETURN CODE &RETC IN STEP
&STEP'
   // GOTO EXIT
   /. CANCELLED
   // EXEC B234RDD,PARM='JOB NOWAIT CANCELLED IN STEP &STEP'
   // GOTO EXIT
   /. ABENDED
   // EXEC B234RDD,PARM='JOB NOWAIT : ABEND IN STEP &STEP'
   /. EXIT
   /&
   * $$ EOJ
```

Note that the special partition is not dedicated purely to console messages from B234RDD, but is also used by other jobs that require manual operator intervention – for example, file data transfer on external devices like cartridges or diskettes.

Note also that:

- The CLASS=T parameter can be altered by changing the literal T in the statement &RDRCLASS SETC 'T' to a value of your choice.
- You need to change the dummy string ACCOUNTING-STRING, which is part of the // JOB statement, to the string you require.
- If the B232RDR subroutine signals no errors, return code 0 is issued to job control; otherwise, B234RDD issues return code 8.
- Because of an IBM restriction in the EOJ macro with the RC keyword, the B234RDD program can only be executed below the 16 MB line (RMODE 24).

B234RDD

```
LCLC  &RDRCLAS
&RDRCLAS SETC  'T'                     CLASS
         TITLE 'B234RDD - SUBMIT PAUSE JOB (CLASS &RDRCLAS)'
B234RDD  CSECT
**********************************************************************
*        REGISTER EQUATES
**********************************************************************
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
**********************************************************************
```

```
* REGISTER USAGE:
*    R15 PROGRAM ENTRY POINT
*    R14 RETURN ADDRESS
*    R13 SAVE AREA ADDRESS
*    R12
*    R11
*    R1Ø
*    R9  BASE REGISTER
*    R8
*    R7
*    R6
*    R5
*    R4
*    R3  ALLOWED LENGTH OF PARM STRING
*    R2  LENGTH OF PARM STRING
*    R1  ADDRESS OF PARM STRING (INPUT PARAMETER)
*    RØ
*********************************************************************
         EJECT
*********************************************************************
*        LOAD BASE REGISTER, CHECK PARM STRING
*********************************************************************
         BALR  R9,Ø                     LOAD BASE REGISTER
         USING *,R9                     ESTABLISH ADDRESSABILITY
         LA    R13,SAVEAREA             ADDRESS OF OWN SAVE AREA
         CR    R1,R15                   TEST PARAMETER ADDRESS
         BE    SPOOL                    PARM STRING MISSING
         TM    Ø(R1),X'8Ø'              TEST HIGH ORDER BIT
         BNO   SPOOL                    MORE THAN ONE PARAMETER, IGNORE
         EJECT
*********************************************************************
*        MOVE PARM STRING TO STORAGE
*********************************************************************
         L     R1,Ø(,R1)                ADDRESS OF PARM STRING
         LH    R2,Ø(,R1)                LENGTH OF PARM STRING
         LA    R3,L'PARAM               ALLOWED LENGTH OF PARM STRING
         CR    R2,R3                    COMPARE LENGTH
         BNH   PARMLGOK                 LENGTH IS OK
         LR    R2,R3                    SET LENGTH TO ALLOWED LENGTH
PARMLGOK DS    ØH
         BCTR  R2,Ø                     LENGTH USING EXECUTE
         EX    R2,MVCPARM               MOVE PARM STRING TO STORAGE
         EJECT
*********************************************************************
*        SPOOL JOB TO POWER READER QUEUE
*********************************************************************
SPOOL    DS    ØH
         CALL  B232RDR,(VSEJOB,B232RTC)
         SR    R15,R15                  CLEAR RETURN CODE (JOB CONTROL)
         CLI   B232RTC,C'Ø'             TEST RETURN CODE (B232RDR)
         BE    RETURN                   NO ERROR
         LA    R15,8                    RETURN CODE 8 (JOB CONTROL)
RETURN   EOJ   RC=(R15)
         EJECT
*********************************************************************
*        OWN SAVE AREA
*********************************************************************
SAVEAREA DS    18F
         EJECT
*********************************************************************
*        WORKING STORAGE
*********************************************************************
MVCPARM  MVC   PARAM(Ø),2(R1)           MOVE PARAMETER
VSEJOB   DC    F'32'
         DC    F'Ø'
         DC    CL24'* $$ JOB JNM=INFO,CLASS='
         DC    CL1'&RDRCLAS'            CLASS
         DC    CL7',PRI=9'
         DC    F'32'
         DC    F'Ø'
         DC    CL32'// JOB INFO ACCOUNTING-STRING'
         DC    F'72'
         DC    F'Ø'
         DC    CL9'// PAUSE'
PARAM    DC    CL6Ø' '
         DC    CL3' '
         DC    F'4'
         DC    F'Ø'
         DC    CL4'/&&'
         DC    F'8'
         DC    F'Ø'
         DC    CL8'* $$ EOJ'
         DC    X'FF'
B232RTC  DS    C
         END   B234RDD
```

Walter Richters
(Germany)

Doing something more than sorting with DFSORT/VSE – part 1

Of all the courses I teach and the articles I write, *Don't Program, Use Utilities* is the most popular, because both managers and technical professionals instantly see the benefit of the concept. They may not agree on anything else, but they know that choosing between writing a program and using a utility is a 'no brainer' decision.

Since I developed the course in 1987, I have concentrated on two utilities – IDCAMS and DFSORT. Over the years since, I have been able to do a lot of observation and even a little benchmarking. Plus, DFSORT performance has improved significantly in that period.

Fifteen years ago, when I had a lot more energy than brains, I would write a few key VSE utility functions in Assembler, coding my own Channel Command Words (CCWs) to avoid the overhead of standard sequential file access methods. I took some perverse pleasure in knowing that no one could write anything that would perform better.

Today, DFSORT/VSE 3.4 – in fact, all DFSORT releases – would outperform my home-grown utilities, significantly so for large files. One way that DFSORT gets its speed is by reading and writing multiple tracks at a time, not just a single track like my Assembler program did. In those days, reading a track instead of a record or block at a time was considered the height of performance perfection.

A WAY OF THINKING

The best way to reduce the amount of programming you do is to think of DFSORT, IDCAMS, and other utilities as general-purpose utilities, not specific-purpose packages. IDCAMS is not just a VSAM define, load, and delete utility; it can do a great deal more. DFSORT is not just a sort utility; it can do an amazing number of different things.

It is also important to think of DFSORT as extremely fast. If you can get DFSORT to do something, it will almost certainly perform better than any other method of accomplishing the same task. The obvious exception to this rule of thumb is when the solution does not really fit the problem very well – for example, using sequential processing when direct (keyed) processing is called for.

THE KNOWLEDGE PREREQUISITE

If you don’t know all of DFSORT’s features, it’s very unlikely that you’ll think to use it as frequently as you should. After all, if you need to ensure that certain fields in a file are unique and records with the same values are to be purged, you won’t think of using DFSORT if you don’t know it can perform this function.

So, a review of DFSORT statements and their capabilities is an obvious and important place to start. Next, and not to be overlooked, is an understanding of the order in which DFSORT statements are processed, so you will get the results you expect!

SORT

The most obvious use of the SORT statement is to sort a file based on the contents of a field in specific columns of each record of the file.

```
SORT FIELDS=(15,7,A),FORMAT=CH
```

sorts a file in ascending order by a seven-byte character field beginning in column 15 and ending in column 21.

This assumes that the file has fixed-length records. If the records are variable length, DFSORT considers column 1 as the start of the four-byte Record Descriptor Word (RDW) that indicates the length of the record. The data begins in column 5. So, if the records in the file are variable length, the above statement sorts the file based on a seven-character field beginning in column 11 and ending in column 17. This can be very confusing when switching between DFSORT and other tools and programming languages where the data begins in column 1.

The file is input as SORTIN1 and output as SORTOUT. These file names can be overridden by the OPTION FILNM= statement.

```
OPTION FILNM=(MASTOUT,MASTIN)
SORT FIELDS=(15,7,A),FORMAT=CH
```

In this case, the input is from MASTIN

```
//MASTIN DLBL ...
```

or

```
//MASTIN TLBL ...
```

and the output is to MASTOUT.

DFSORT STATEMENT FORMAT

All DFSORT statements follow the same format, similar to Assembler and JCL. One or more consecutive blanks are delimiters between the label, operation, operand fields, and comments.

```
SORT FIELDS=(15,7,A),FORMAT=CH
```

There is no label – labels serve no practical purpose in DFSORT – so the statement cannot start in column 1. SORT is the operation, and FIELDS and FORMAT are the keywords used in the operand fields. There are no comments.

To summarize the important points:

- Do not start in column 1 (unless you use a label).
- Never use lower case, except within quotes or comments.
- Do not go past column 71 (70 if the line is continued).
- Never code blanks around equals signs, commas, parentheses, or anywhere else in operand fields, except within quotes or comments.
- Use single quotes, not double.
- Continue a statement by ending the operand fields with a comma followed by a blank; the continuation line can begin anywhere in columns 2-71.

Comments can be coded at the end of a statement

```
SORT FIELDS=(15,7,A),FORMAT=CH  SORT BY EMPLOYEE NUMBER
```

or on a continued statement.

```
SORT FIELDS=(15,         EMPLOYEE NUMBER START COLUMN
              7,         EMPLOYEE NUMBER LENGTH
              A),        ASCENDING ORDER
            FORMAT=CH    CHARACTER FORMAT
```

Blank lines are difficult to use, outweighing any benefit they may provide in improving readability. Because blank lines are allowed only as continued remarks, they require that the previous statement have a non-blank character in column 72 – the continuation column. This only works at the end of a DFSORT/VSE statement

```
SORT FIELDS=(15,7,A),FORMAT=CH  SORT BY EMPLOYEE NUMBER          X
         (blank line)
```

and cannot be used in the middle of a continued statement.

```
SORT FIELDS=(15,         EMPLOYEE NUMBER START COLUMN
              7,         EMPLOYEE NUMBER LENGTH
              A),        ASCENDING ORDER                         X
```

```
          (blank line not allowed here)                         X
            FORMAT=CH    CHARACTER FORMAT
```

Not discussed in this initial article, ICETOOL is a part of DFSORT, but uses a somewhat different approach, including a different statement format.

OTHER SORT PARAMETERS

As well as sorting in ascending order based on the EBCDIC values of each character, DFSORT can correctly sort a large number of different data formats, including both signed and unsigned numeric formats. For example, FORMAT=ZD handles signed zoned decimal, FORMAT=PD handles signed packed decimal, FORMAT=FI handles signed fixed-point binary, and FORMAT=BI handles unsigned binary. At first glance, BI appears to be the same as CH until you realize that field position and length can be specified down to the bit level with BI!

A column.bit representation is used for specifying individual bits and byte.bit for lengths. To be consistent with existing notation, the first bit (left-most or high order bit) of each byte is indicated by zero. When specifying a field position,

```
17.Ø
17.
17
```

all refer to the beginning of the byte that begins in column 17. To sort by a one-and-a-half byte (12 bits) unsigned binary field beginning in the lower half of the byte that begins in column 17, you would code:

```
SORT FIELDS=(17.4,1.4,A),FORMAT=BI
```

The most common use of sub-byte or bit-level notation is for sorting based on the values of flag bits.

As well as normal numeric data, there are also a number of formats that cover ASCII data. Six formats address the need to correctly sort two-digit years for Year 2000-related issues; the Y2PAST operand of the OPTION statement can even be used to override the installation default century windowing used. And FORMAT=AQ allows the standard EBCDIC sort (FORMAT=CH) order to be modified with the ALTSEQ statement.

Specifying

```
SORT FIELDS=(15,7,D),FORMAT=CH
```

reverses the order of the records by sorting in descending order. All of the formats can be specified in either ascending or descending order.

If multiple fields are required to determine the sort order – when the first one matches, a second field must be checked, such as sorting employees by division, then by employee number within each division – it can be as easy as increasing the length, if the fields are adjacent, of the same format, and in the same order, ascending or descending.

```
SORT FIELDS=(15,11,A),FORMAT=CH
```

Otherwise, multiple sort fields can be specified:

```
SORT FIELDS=(15,7,D,36,4,A,43,9,A),FORMAT=CH
```

If the fields have different data formats, the FORMAT= operand can be removed and the format is then specified for each field in the FIELDS= operand as the third parameter:

```
SORT FIELDS=(15,7,CH,D,36,4,PD,A,43,9,CH,A)
```

If the data is in more than one file, the files do not have to be concatenated before running DFSORT. Up to nine files can be input for sorting as SORTIN1 to SORTIN9. They are sorted just as if all the data came from one big file made up of all the records in SORTIN1, followed by all the records in SORTIN2, etc.

If more than one file is input, the number of files must be indicated on the FILES= operand:

```
SORT FIELDS=(15,7,D),FORMAT=CH,FILES=5
```

As indicated earlier, the SORTINn and SORTOUT file names can be overridden:

```
OPTION FILNM=(MASTOUT,MASTIN,TRANS1,TRANS2,TRANS3,TRANS4)
SORT FIELDS=(15,7,D),FORMAT=CH,FILES=5
```

EQUALS

But what happens when a file is sorted but several records have the same value for the sort field(s)? If the EQUALS operand is specified in SORT, MERGE, OPTION or the installation default, records with equal sort field values will be kept in the same order as that in which they were input. Since this is rarely necessary and slows down large sorts, NOEQUALS is normally set as the installation default.

However, EQUALS does make some processes possible. For example, the SUM statement can be used to eliminate duplicate records, where a duplicate is defined as any record for which specified field(s) match. SUM deletes the second and subsequent duplicates, retaining the first record. Only with EQUALS can you guarantee that the first record in the file for each set of duplicates is the record retained. This can be critical in a process that presorts the file on other fields. For example, if you want to create a list of programs, ignoring duplicates and old versions and showing the date of the latest version, you could pre-sort the list in descending order by date. Assuming that the date is input in character format as mm/dd/yyyy:

```
SORT FIELDS=(15,4,D,  SORT ON YYYY
              9,2,D,  SORT ON MM
             12,2,D), SORT ON DD
     FORMAT=CH
```

Then, in a second DFSORT job step, sort and sum by program name with EQUALS specified to retain the original sort order:

```
SORT FIELDS=(1,8,A),FORMAT=CH,EQUALS
SUM FIELDS=NONE
```

Unfortunately, however, this will not work, as DFSORT ignores the EQUALS specification when the SUM statement is used. One way that does work will be shown with the MERGE statement (below). There is one consolation, however: even if this two-SORT method did work, the MERGE method would still be more efficient.

It should be noted that ICETOOL can also do this, and a lot more, with duplicate records (see future articles in this series).

The default NOEQUALS can cause some unusual problems. On the same day that I was writing this article, I ran a Year 2000 parallel run that involved both unit testing and system testing – end-to-end testing of the complete application. In one case, the current system date was used; in the other, 29 February 2000.

Although the final results were identical, one intermediate data file was different. Of the nearly one million records in the file, three were identified as different in a hard-to-read report by an extremely slow compare program. In each case, the report displayed two seemingly identical records. It was not until the two files were displayed side by side at the point of one of the differences that it was clear what was going on. The two records were reversed in one file when compared to the other. Further investigation indicated that, in each case, the two records that were reversed had identical sort keys.

So, not only does NOEQUALS not define the sort order of records with identical sort keys, but a NOEQUALS sort is not repeatable – the next time that the same file is sorted, there is no assurance that records with the same sort key will be in the same order.

NOT SORTING

It is not necessary to sort records. Obviously, copying without sorting is a lot faster.

```
SORT FIELDS=COPY
```

will copy SORTIN1 to SORTOUT, probably faster than any other available software product, and measurably faster than writing your own channel commands to copy a track at a time, as I described at the beginning of the article.

Coding

```
SORT FIELDS=COPY,FILES=2
```

will concatenate SORTIN1 and SORTIN2 (ie the first record of SORTIN2 will follow immediately after the last record of SORTIN1) into SORTOUT.

SKIPREC and STOPAFT are the only operands that are permitted with FIELDS=COPY.

```
SORT FIELDS=COPY,SKIPREC=10000,STOPAFT=1000
```

will copy record numbers 10,001 through 11,000 of SORTIN1 to SORTOUT. SKIPREC and STOPAFT can also be specified on the OPTION statement:

```
OPTION SKIPREC=10000,STOPAFT=1000
SORT FIELDS=COPY
```

MERGE

The concept of a merge is to take several already-sorted files and create one big sorted file with all the records in it. For most of us, merging was one of those fundamental processing methods we learnt at college. Simplified, it was a two-step process:

- Look at the key value of the current record in each of the two or more input files.
- The next record output is the one with the lowest key value.

Other than this rather specialized process, the DFSORT MERGE statement can also be used to define a sort field for a file that is already in the correct sorted order, without going through the considerable overhead of trying to sort an already sorted file. This is especially useful with the SUM statement to summarize or otherwise eliminate duplicate records.

This method can be used to overcome DFSORT/VSE's habit of ignoring an EQUALS specification when SUM is specified. Recoding the example that did not work in our discussion of EQUALS above, the problem of creating a program list can be rewritten as follows, though still in two steps:

```
SORT FIELDS=(1,8,A,15,4,D,9,2,D,12,2,D),FORMAT=CH
```

then:

```
MERGE FIELDS=(1,8,A),FORMAT=CH,FILES=1
SUM FIELDS=NONE
```

The complete sort is done in the first step. The file is sorted alphabetically by program name. For any identical program names, the records are sorted first by year, then month, then day of month, in descending order – reverse chronological order.

The MERGE statement of the second step does not really do anything other than define the program name as the summation field for the SUM statement that follows. The SUM statement will delete all but the first record for each program name. The FIELDS=NONE specification indicates that there are no totals to be calculated. SUM will be described in detail in future articles.

As well as being the only method that works (EQUALS is ignored when SUM is being used), this would have run significantly faster for large files than the original example, since it eliminates both the second sort and the use of the performance-sapping EQUALS operand.

PROCESSING ORDER

As this article continues into the next issue, it will become clear that it is essential to know the order in which DFSORT statements and exits are processed. A classic question in this area is: if you reformat a record with the INREC statement, should an INCLUDE or OMIT statement (which selects or rejects records for processing) refer to the format of the record before or after INREC has reformatted it?

DFSORT statements, options, and exits are processed in the following order:

```
SKIPREC=
E15/E32 exits
INCLUDE or OMIT
STOPAFT=
INREC
SORT or MERGE
SUM
OUTREC
E35 exit
```

Although it's good practice to code the statements in the order in which they're processed, it is not essential. Nor will changing the order in which they're coded change the processing order.

To answer the question posed at the beginning of this section: INCLUDE or OMIT must refer to the position of fields before INREC has reformatted the record.

NEXT ISSUE

The next installment of this article begins with a detailed look at INREC and OUTREC, not to be confused with INCLUDE and OMIT, which will also be covered. And there's lots more to come. In the meantime, check out IBM's DFSORT/VSE home page at http://www.ibm.com/storage/dfsortvse/

Jon E Pearkins
(Canada)

Reading the IESCNTL file and creating a DFHSNT

Here, we continue the article begun in the December issue, which reads the IESCNTL file and creates a DFHSNT including all the JCL necessary to assemble, LNKEDT, and new it.

```
 AØ3Ø-NEXT-DFHCSD.
D    DISPLAY 'DF-STATUSØ=' DF-STATUS ', DF-SWØ=' DF-SW
D        ', DF-EOFØ=' DF-EOF ', SV-KEYØ=' SAVE-DF-KEY.
D    DISPLAY 'DF-KEYØ=' DF-KEY.
D    DISPLAY 'DF-RECØ=' DF-RECORD.
     IF DF-EOF = 'Ø' AND GRPLIST-TABLE > SPACE AND DF-SW = '1'
         MOVE DF-KEY TO SAVE-DF-KEY
         MOVE LOW-VALUE TO DF-KEY
         MOVE SAVE-DF-KEY(15:8) TO DF-NAME
         MOVE HIGH-VALUE TO DF-KEY(22:1)
D        DISPLAY 'DF-STATUS1=' DF-STATUS ', DF-SW1=' DF-SW
D            ', DF-EOF1=' DF-EOF ', SV-KEY1=' SAVE-DF-KEY
D        DISPLAY 'DF-KEY1=' DF-KEY
D        DISPLAY 'DF-REC1=' DF-RECORD
         START DFHCSD KEY > DF-KEY
         MOVE '2' TO DF-SW.
     IF DF-EOF = 'Ø' AND GRPLIST-TABLE > SPACE AND DF-SW = '2'
         READ DFHCSD NEXT RECORD
         IF DF-STATUS = 'ØØ' AND
             DF-NAME NOT = SAVE-DF-KEY(15:8)
D                DISPLAY 'DF-STATUS2=' DF-STATUS ', DF-SW2=' DF-SW
D                    ', DF-EOF2=' DF-EOF ', SV-KEY2=' SAVE-DF-KEY
D                DISPLAY 'DF-KEY2=' DF-KEY
D                DISPLAY 'DF-REC2=' DF-RECORD
                 MOVE SAVE-DF-KEY TO DF-KEY
                 START DFHCSD KEY > DF-KEY
                 READ DFHCSD NEXT RECORD
D                DISPLAY 'DF-STATUS3=' DF-STATUS ', DF-SW3=' DF-SW
D                    ', DF-EOF3=' DF-EOF ', SV-KEY3=' SAVE-DF-KEY
D                DISPLAY 'DF-KEY3=' DF-KEY
D                DISPLAY 'DF-REC3=' DF-RECORD
                 IF DF-STATUS NOT = 'ØØ' OR DF-NAME NOT =
                     GRPLIST-TABLE-ENTRY(INDXG3-COUNT)
                         ADD 1 TO INDXG3-COUNT
                         IF INDXG3-COUNT = INDXG2-COUNT
                             MOVE '1' TO DF-EOF
                             GO AØ3Ø-READ-IESCNTL
                         ELSE
                             MOVE 'Ø' TO DF-SW
                             GO AØ3Ø-READ-DFHCSD
                 ELSE
                     MOVE '1' TO DF-SW
```

```
D                    DISPLAY 'DF-STATUS4=' DF-STATUS ', DF-SW4='
D                        DF-SW ', DF-EOF4=' DF-EOF ', SV-KEY4='
D                            SAVE-DF-KEY
D                    DISPLAY 'DF-KEY4=' DF-KEY
D                    DISPLAY 'DF-REC4=' DF-RECORD
                     GO AØ3Ø-NEXT-DFHCSD
         ELSE
         IF DF-STATUS NOT = 'ØØ'
             AND DF-NAME NOT = SAVE-DF-KEY(15:8)
D                DISPLAY 'DF-STATUS5=' DF-STATUS ', DF-SW5='
D                    DF-SW 'DF-EOF5=' DF-EOF ', SV-KEY5='
D                        SAVE-DF-KEY
D                DISPLAY 'DF-KEY5=' DF-KEY
D                DISPLAY 'DF-REC5=' DF-RECORD
                 ADD 1 TO INDXG3-COUNT
                 IF INDXG3-COUNT = INDXG2-COUNT
                     MOVE '1' TO DF-EOF
                     GO AØ3Ø-READ-IESCNTL
                 ELSE
                     MOVE 'Ø' TO DF-SW
                     GO AØ3Ø-READ-DFHCSD.
     IF GROUP-NAME-TABLE > SPACE
         MOVE 1 TO INDXY3-COUNT
         PERFORM VARYING INDXY3-COUNT FROM 1 BY 1 UNTIL
             INDXY3-COUNT > 91 OR
                 GROUP-NAME-TABLE-ENTRY(INDXY3-COUNT) = SPACE OR
                     HIGH-VALUE
         MOVE SPACE TO GROUP-NAME SRCH-PARM2-SRC
         MOVE QUOTE TO SRCH-PARM2-SRC-RQUOTE
         MOVE GROUP-NAME-TABLE-ENTRY(INDXY3-COUNT) TO
             SRCH-PARM2-SRC
         MOVE Ø TO TALLY
         INSPECT SRCH-PARM2-SRC TALLYING TALLY FOR ALL '*'
         IF TALLY > 1
             DISPLAY 'JOB ' VSE-NAME '-MORE THAN 1 ASTERISK IN GRO
-                'UP NAME ' UPON CONSOLE
             DISPLAY '            -GROUP NAME=' SRCH-PARM2-SRC
                 UPON CONSOLE
             GO A999-CALL-BOMBER
         END-IF
         IF SRCH-PARM2-SRC(1:1) = '*'
             MOVE SRCH-PARM2-SRC(2:7) TO GROUP-NAME
             MOVE SPACE TO SRCH-PARM2-SRC
             MOVE GROUP-NAME TO SRCH-PARM2-SRC(1:7)
             MOVE QUOTE TO SRCH-PARM2-SRC(8:1)
             MOVE SPACE TO SRCH-PARM2-SRC-RQUOTE
         END-IF
         INSPECT SRCH-PARM2-SRC REPLACING ALL ' ' BY '+'
         INSPECT SRCH-PARM2-SRC REPLACING ALL '*' BY '+'
         MOVE 'F' TO SRCH-PARM1-DIR
         MOVE 'E' TO SRCH-PARM1-REL
         MOVE DF-NAME TO SRCH-PARM3-TAR
         MOVE +1 TO SRCH-PARM5-STR
         MOVE +8 TO SRCH-PARM5-END
         MOVE HIGH-VALUE TO SRCH-PARM4-RTC
         MOVE 'DPSRCH' TO SUB-NAME
         CALL SUB-NAME USING SRCH-PARM1 SRCH-PARM2 SRCH-PARM3
             SRCH-PARM4 SRCH-PARM5
                 ON EXCEPTION
                     DISPLAY 'JOB ' VSE-NAME '-DPSRCH NOT CATALOGE
-                        'D AS .PHASE OR NOT IN LIBDEF CHAIN'
                             UPON CONSOLE
                     GO A999-CALL-BOMBER
         END-CALL
D        DISPLAY 'IN-SW=' IN-SW ', EX-SW=' EX-SW
D        DISPLAY 'SRCH-PARM1=' SRCH-PARM1
D        DISPLAY 'SRCH-PARM2=' SRCH-PARM2
D        DISPLAY 'SRCH-PARM3=' SRCH-PARM3
D        DISPLAY 'SRCH-PARM4=' SRCH-PARM4
D        DISPLAY 'SRCH-PARM5=' SRCH-PARM5-STR SRCH-PARM5-END
D            SRCH-PARM5-NUM
         IF SRCH-PARM4-RTC > '1'
             DISPLAY 'JOB ' VSE-NAME '-DPSRCH CALL ERROR, R/C='
                 SRCH-PARM4-RTC UPON CONSOLE
             GO A999-CALL-BOMBER
         END-IF
         IF IN-SW = '1' AND SRCH-PARM4-RTC = 'Ø'
             MOVE 1ØØ TO INDXY3-COUNT
             GO AØ3Ø-READ-DFHCSD
         END-IF
         IF EX-SW = '1' AND SRCH-PARM4-RTC = '1'
             MOVE 1ØØ TO INDXY3-COUNT
             GO AØ3Ø-READ-DFHCSD
         END-IF
         END-PERFORM
     END-IF.
D    DISPLAY 'DF-STATUS6=' DF-STATUS ', DF-SW6=' DF-SW
D        ', DF-EOF6=' DF-EOF ', SV-KEY6=' SAVE-DF-KEY.
D    DISPLAY 'DF-KEY6=' DF-KEY.
D    DISPLAY 'DF-REC6=' DF-RECORD.
     IF DF-EOF NOT = '1' AND DF-STATUS NOT = 'ØØ'
         IF DF-STATUS NOT = '1Ø'
             DISPLAY 'JOB ' VSE-NAME '-DFHCSD READ ERROR, FILE STA
-                'TUS=' DF-STATUS UPON CONSOLE
             MOVE DF-VSAM-STATUS-R15 TO VSAM-R15
             MOVE DF-VSAM-STATUS-FUN TO VSAM-FUN
             MOVE DF-VSAM-STATUS-FBK TO VSAM-FBK
             DISPLAY '            -VSAM STATUS='
                 VSAM-STATUS UPON CONSOLE
             GO A999-CALL-BOMBER
         ELSE
             MOVE '1' TO DF-EOF
```

```
        GO AØ3Ø-READ-IESCNTL.
ADD 1 TO DFRD-COUNT.
IF DF-REC-TYPE = MAP-RECORD
    ADD 1 TO DFMP-COUNT
    MOVE SPACE TO SORT-RECORD
    MOVE 'M' TO SORT-ID
    MOVE SAVE-DF-NAME TO SORT-GRPLIST
    MOVE DF-RECORD TO SORT-IE-RECORD
    MOVE VSE-NAME TO SORT-VSE-NAME
    MOVE DF-NAME TO SORT-DF-NAME
    MOVE DF-REC-TYPE-NAME TO SORT-PROGID
    RELEASE SORT-RECORD
    GO AØ3Ø-READ-DFHCSD.
IF DF-REC-TYPE = PROG-RECORD
    ADD 1 TO DFPG-COUNT
    MOVE SPACE TO SORT-RECORD
    MOVE 'P' TO SORT-ID
    MOVE SAVE-DF-NAME TO SORT-GRPLIST
    MOVE DF-RECORD TO SORT-IE-RECORD
    MOVE VSE-NAME TO SORT-VSE-NAME
    MOVE DF-NAME TO SORT-DF-NAME
    MOVE DF-REC-TYPE-NAME TO SORT-PROGID
    RELEASE SORT-RECORD
    GO AØ3Ø-READ-DFHCSD.
IF DF-REC-TYPE NOT = TRAN-RECORD OR DF-TRANID-TYPE NOT =
    HEX-Ø1-R
        GO AØ3Ø-READ-DFHCSD.
ADD 1 TO DFTR-COUNT.
COMPUTE WK-BYTE = DF-RECORD-LENG - HALF-WORD-46.
IF WK-BYTE NOT > Ø
    MOVE WK-BYTE TO SAVE-WK-BYTE
    DISPLAY 'JOB ' VSE-NAME '-WK-BYTE NOT > THAN Ø, WK-BYTE='
        SAVE-WK-BYTE UPON CONSOLE
    DISPLAY 'JOB ' VSE-NAME '-FOR ' DF-NAME ' ' DF-REC-TYPE
        UPON CONSOLE.
MOVE Ø TO INDX-COUNT.
MOVE SPACE TO SORT-RECORD.
MOVE 'V' TO SORT-ID.
MOVE SAVE-DF-NAME TO SORT-GRPLIST.
MOVE DF-RECORD TO SORT-IE-RECORD.
MOVE VSE-NAME TO SORT-VSE-NAME.
MOVE DF-NAME TO SORT-DF-NAME.
MOVE DF-REC-TYPE-NAME TO SORT-PROGID.
MOVE Ø1 TO SORT-TRANSEC-VALUE-N.
MOVE LOW-VALUE TO SORT-RSL-BYTE.
PERFORM AØ33-MOVE-BYTES THRU AØ33-MOVE-BYTES-EXIT VARYING
    INDX-COUNT FROM 1 BY 1 UNTIL INDX-COUNT > WK-BYTE.
MOVE Ø TO INDX-COUNT.
PERFORM VARYING INDX-COUNT FROM 1 BY 1 UNTIL
    INDX-COUNT > WK-BYTE
        IF DF-TRANID-DATA(INDX-COUNT) = X'ØE'
```

```
                 ADD 1 TO INDX-COUNT
                 IF DF-TRANID-DATA(INDX-COUNT) = X'Ø3' OR X'Ø4'
                     MOVE DF-TRANID-DATA(INDX-COUNT) TO SUB-2-R
                     ADD 1 TO INDX-COUNT
                     MOVE 1 TO SUB-1
                     PERFORM VARYING SUB-1 FROM 1 BY 1 UNTIL
                         SUB-1 > SUB-2 OR SUB-1 > 4
                             MOVE DF-TRANID-DATA(INDX-COUNT) TO
                                 SORT-TRAN-PFKEY(SUB-1:1)
                             ADD 1 TO INDX-COUNT
                     END-PERFORM
                     IF SUB-2-R = X'Ø3'
                         MOVE SORT-TRAN-PFKEY(3:1) TO
                             SORT-TRAN-PFKEY(4:1)
                         MOVE 'Ø' TO SORT-TRAN-PFKEY(3:1)
                     END-IF
D                    DISPLAY 'DF-RECORD=' DF-RECORD
                 END-IF
             END-IF
     END-PERFORM.
     MOVE Ø TO INDX-COUNT.
     PERFORM VARYING INDX-COUNT FROM 1 BY 1 UNTIL
         INDX-COUNT > WK-BYTE
             IF DF-TRANID-DATA(INDX-COUNT) = HEX-Ø6-R
                 ADD 1 TO INDX-COUNT
                 IF DF-TRANID-DATA(INDX-COUNT) < HEX-Ø9-R
                     MOVE DF-TRANID-DATA(INDX-COUNT) TO SUB-2-R
                     ADD 1 TO INDX-COUNT
                     MOVE 1 TO SUB-1
                     PERFORM VARYING SUB-1 FROM 1 BY 1 UNTIL
                         SUB-1 > SUB-2 OR SUB-1 > 8
                             MOVE DF-TRANID-DATA(INDX-COUNT) TO
                                 SORT-PROG-NAME(SUB-1:1)
                             ADD 1 TO INDX-COUNT
                     END-PERFORM
                 END-IF
             END-IF
             IF DF-TRANID-DATA(INDX-COUNT) = X'ØA'
                 ADD 1 TO INDX-COUNT
                 IF DF-TRANID-DATA(INDX-COUNT) < X'Ø5'
                     MOVE DF-TRANID-DATA(INDX-COUNT) TO SUB-2-R
                     ADD 1 TO INDX-COUNT
                     MOVE 1 TO SUB-1
                     PERFORM VARYING SUB-1 FROM 1 BY 1 UNTIL
                         SUB-1 > SUB-2 OR SUB-1 > 4
                             MOVE DF-TRANID-DATA(INDX-COUNT) TO
                                 SORT-REMOTE-SYSID(SUB-1:1)
                             ADD 1 TO INDX-COUNT
                     END-PERFORM
                 END-IF
             END-IF
     END-PERFORM.
     IF SORT-REMOTE-SYSID > SPACE
         ADD 1 TO DFRM-COUNT
         IF SAVE-REMT-Y-OR-N = 'N'
             ADD 1 TO DFDE-COUNT.
     RELEASE SORT-RECORD.
     GO AØ3Ø-READ-DFHCSD.
 AØ3Ø-READ-IESCNTL.
     MOVE LENGTH OF IE-RECORD TO IE-REC-LEN.
     READ IESCNTL NEXT RECORD AT END
         MOVE '1' TO IE-EOF
         GO AØ3Ø-READ-IESCNTL-EXIT.
     IF IE-STATUS NOT = 'ØØ'
         IF IE-STATUS NOT = '1Ø'
             DISPLAY 'JOB ' VSE-NAME '-IESCNTL READ ERROR, FILE ST
-                'ATUS=' IE-STATUS UPON CONSOLE
             MOVE IE-VSAM-STATUS-R15 TO VSAM-R15
             MOVE IE-VSAM-STATUS-FUN TO VSAM-FUN
             MOVE IE-VSAM-STATUS-FBK TO VSAM-FBK
             DISPLAY '            -VSAM STATUS='
                 VSAM-STATUS UPON CONSOLE
             GO A999-CALL-BOMBER
         ELSE
             MOVE '1' TO IE-EOF
             GO AØ3Ø-READ-IESCNTL-EXIT.
     IF IE-TYPE > 'US'
         MOVE '1' TO IE-EOF
         GO AØ3Ø-READ-IESCNTL-EXIT.
     IF IE-REC-LEN NOT = LENGTH OF IE-RECORD
         DISPLAY 'JOB ' VSE-NAME '-IESCNTL "US" RECORD NOT 298 BYT
-            'ES' UPON CONSOLE
         DISPLAY '            -' IE-RECORD UPON CONSOLE
         GO A999-CALL-BOMBER.
     ADD 1 TO CNTL-COUNT.
*    IF CNTL-COUNT > 1Ø
*        MOVE '1' TO IE-EOF
*        GO AØ3Ø-READ-IESCNTL-EXIT.
     INSPECT IE-USERID REPLACING ALL LOW-VALUE BY SPACE.
     PERFORM AØ4Ø-BUILD-SNT THRU AØ4Ø-BUILD-SNT-EXIT.
     MOVE SPACE TO SORT-RECORD.
     MOVE 'U' TO SORT-ID.
     MOVE IE-RECORD TO SORT-IE-RECORD.
     MOVE VSE-NAME TO SORT-VSE-NAME.
     MOVE Ø TO INDX-COUNT.
     PERFORM AØ5Ø-BUILD-SORT-RECORD THRU
         AØ5Ø-BUILD-SORT-RECORD-EXIT VARYING INDX-COUNT FROM 1
             BY 1 UNTIL INDX-COUNT > 64.
     GO AØ3Ø-READ-IESCNTL.
 AØ3Ø-READ-IESCNTL-EXIT. EXIT.
 AØ33-MOVE-BYTES.
     IF DF-TRANID-DATA(INDX-COUNT) = HEX-Ø5-R
```

```
         MOVE INDX-COUNT TO SAVE-INDX-COUNT
         ADD 1 TO SAVE-INDX-COUNT
         IF DF-TRANID-DATA(SAVE-INDX-COUNT) = HEX-Ø1-R
             ADD 1 TO SAVE-INDX-COUNT
             MOVE DF-TRANID-DATA(SAVE-INDX-COUNT) TO
                 SORT-RSL-BYTE.
*        ELSE
*            DISPLAY 'JOB ' VSE-NAME '-RSL LENGTH BYTE NOT WHAT WA
*-               'S EXPCTED FOR ' DF-NAME ' ' DF-REC-TYPE-NAME
*                    UPON CONSOLE
*            DISPLAY 'JOB ' VSE-NAME '-' DF-RECORD UPON CONSOLE.
     IF DF-TRANID-DATA(INDX-COUNT) = HEX-11-R
         MOVE INDX-COUNT TO SAVE-INDX-COUNT
         ADD 1 TO SAVE-INDX-COUNT
         IF DF-TRANID-DATA(SAVE-INDX-COUNT) = HEX-Ø1-R
             ADD 1 TO SAVE-INDX-COUNT
             MOVE DF-TRANID-DATA(SAVE-INDX-COUNT) TO WK-BYTE-2-R
             MOVE WK-BYTE-2 TO SORT-TRANSEC-VALUE-N.
*        ELSE
*            DISPLAY 'JOB ' VSE-NAME '-TRANSEC LENGTH BYTE NOT WHA
*-               'T WAS EXPCTED FOR ' DF-NAME ' ' DF-REC-TYPE-NAME
*                    UPON CONSOLE
*            DISPLAY 'JOB ' VSE-NAME '-' DF-RECORD UPON CONSOLE.
 AØ33-MOVE-BYTES-EXIT. EXIT.
 AØ4Ø-BUILD-SNT.
     MOVE SPACE TO POWP-PARM-SNT-TYPE-TIMEOUT.
     IF IE-SIGN-OFF-TIME > LOW-VALUE
         MOVE 'DPHTOC' TO SUB-NAME
         CALL SUB-NAME USING IE-SIGN-OFF-TIME SNT-TIMEOUT-VALUE
             ON EXCEPTION
                 DISPLAY 'JOB ' VSE-NAME '-DPHTOC NOT CATALOGED AS
-                    ' .PHASE OR NOT IN LIBDEF CHAIN'
                        UPON CONSOLE
                 GO A999-CALL-BOMBER
         END-CALL
         MOVE SNT-TIMEOUT TO POWP-PARM-SNT-TYPE-TIMEOUT.
     MOVE POWP-PARM-SNT-TYPE TO POWP-PARM-1.
     PERFORM B12Ø-SUBMIT.
     MOVE SPACE TO POWP-PARM-SNT-USERID-COMMA.
     MOVE IE-USERID TO POWP-PARM-SNT-USERID-USERID.
     IF POWP-PARM-SNT-USERID-USERID-2 = SPACE
         MOVE ',' TO POWP-PARM-SNT-USERID-USERID-2
     ELSE
     IF POWP-PARM-SNT-USERID-USERID-3 = SPACE
         MOVE ',' TO POWP-PARM-SNT-USERID-USERID-3
     ELSE
     IF POWP-PARM-SNT-USERID-USERID-4 = SPACE
         MOVE ',' TO POWP-PARM-SNT-USERID-USERID-4
     ELSE
     IF POWP-PARM-SNT-USERID-USERID-5 = SPACE
         MOVE ',' TO POWP-PARM-SNT-USERID-USERID-5
```

```
    ELSE
    IF POWP-PARM-SNT-USERID-USERID-6 = SPACE
        MOVE ',' TO POWP-PARM-SNT-USERID-USERID-6
    ELSE
    IF POWP-PARM-SNT-USERID-USERID-7 = SPACE
        MOVE ',' TO POWP-PARM-SNT-USERID-USERID-7
    ELSE
    IF POWP-PARM-SNT-USERID-USERID-8 = SPACE
        MOVE ',' TO POWP-PARM-SNT-USERID-USERID-8
    ELSE
        MOVE ',' TO POWP-PARM-SNT-USERID-COMMA.
    MOVE POWP-PARM-SNT-USERID TO POWP-PARM-1.
    PERFORM B12Ø-SUBMIT.
    MOVE SPACE TO POWP-PARM-SNT-OPERID-COMMA.
    IF SAVE-OPNM-Y-OR-N = 'Y'
        MOVE IE-USERID TO POWP-PARM-SNT-OPNAME-OPNAME
        MOVE POWP-PARM-SNT-OPNAME TO POWP-PARM-1
        PERFORM B12Ø-SUBMIT.
    MOVE IE-OPERID TO POWP-PARM-SNT-OPERID-OPERID.
    IF POWP-PARM-SNT-OPERID-OPERID-2 = SPACE
        MOVE ',' TO POWP-PARM-SNT-OPERID-OPERID-2
    ELSE
    IF POWP-PARM-SNT-OPERID-OPERID-3 = SPACE
        MOVE ',' TO POWP-PARM-SNT-OPERID-OPERID-3
    ELSE
        MOVE ',' TO POWP-PARM-SNT-OPERID-COMMA.
    MOVE POWP-PARM-SNT-OPERID TO POWP-PARM-1.
    PERFORM B12Ø-SUBMIT.
    IF IE-SCTY-64-Ø1 = LOW-VALUE
        DISPLAY 'JOB ' VSE-NAME '-USER RECORD ' IE-USERID ' HAS N
-           'O SECURITY KEYS'
        GO A999-CALL-BOMBER.
    MOVE SPACE TO USER-SECURITY-TABLE
        POWP-PARM-SNT-SCTY-ENTRIES.
    MOVE 'DPBITX' TO SUB-NAME.
    CALL SUB-NAME USING IE-SCTY-Ø8-Ø1 BITS-1-8
        ON EXCEPTION
            DISPLAY 'JOB ' VSE-NAME '-DPBITX NOT CATALOGED AS .PH
-               'ASE OR NOT IN LIBDEF CHAIN' UPON CONSOLE
            GO A999-CALL-BOMBER
    END-CALL
    MOVE 'SCT' TO POWP-PARM-SNT-SCTY-ENTRY(Ø1).
    MOVE 'YKE' TO POWP-PARM-SNT-SCTY-ENTRY(Ø2).
    MOVE 'Y=(' TO POWP-PARM-SNT-SCTY-ENTRY(Ø3).
    MOVE 3 TO INDX-COUNT.
    IF BITS-1 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø1) INDX-COUNT
        MOVE 'Ø1,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø1).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-2 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø2) INDX-COUNT
        MOVE 'Ø2,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø2).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-3 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø3) INDX-COUNT
        MOVE 'Ø3,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø3).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-4 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø4) INDX-COUNT
        MOVE 'Ø4,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø4).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-5 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø5) INDX-COUNT
        MOVE 'Ø5,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø5).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-6 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø6) INDX-COUNT
        MOVE 'Ø6,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø6).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-7 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø7) INDX-COUNT
        MOVE 'Ø7,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø7).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-8 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø8) INDX-COUNT
        MOVE 'Ø8,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø8).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    MOVE 'DPBITX' TO SUB-NAME.
    CALL SUB-NAME USING IE-SCTY-16-Ø9 BITS-1-8.
    IF BITS-1 = '1'
        ADD 1 TO TOTL-SEC-TABLE(Ø9) INDX-COUNT
        MOVE 'Ø9,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(Ø9).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-2 = '1'
        ADD 1 TO TOTL-SEC-TABLE(1Ø) INDX-COUNT
        MOVE '1Ø,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(1Ø).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
    IF BITS-3 = '1'
        ADD 1 TO TOTL-SEC-TABLE(11) INDX-COUNT
        MOVE '11,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
            USER-SEC-TABLE(11).
    PERFORM B1Ø5-CHECK-INDX-COUNT.
```

```
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(12) INDX-COUNT
    MOVE '12,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(12).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(13) INDX-COUNT
    MOVE '13,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(13).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(14) INDX-COUNT
    MOVE '14,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(14).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(15) INDX-COUNT
    MOVE '15,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(15).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(16) INDX-COUNT
    MOVE '16,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(16).
PERFORM B1Ø5-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-24-17 BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(17) INDX-COUNT
    MOVE '17,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(17).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(18) INDX-COUNT
    MOVE '18,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(18).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(19) INDX-COUNT
    MOVE '19,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(19).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(2Ø) INDX-COUNT
    MOVE '2Ø,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(2Ø).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(21) INDX-COUNT
    MOVE '21,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(21).
```

```
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(22) INDX-COUNT
    MOVE '22,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(22).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(23) INDX-COUNT
    MOVE '23,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(23).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(24) INDX-COUNT
    MOVE '24,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(24).
PERFORM B1Ø5-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-32-25 BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(25) INDX-COUNT
    MOVE '25,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(25).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(26) INDX-COUNT
    MOVE '26,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(26).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(27) INDX-COUNT
    MOVE '27,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(27).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(28) INDX-COUNT
    MOVE '28,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(28).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(29) INDX-COUNT
    MOVE '29,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(29).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(3Ø) INDX-COUNT
    MOVE '3Ø,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(3Ø).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(31) INDX-COUNT
    MOVE '31,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
```

```
        USER-SEC-TABLE(31).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(32) INDX-COUNT
    MOVE '32,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(32).
PERFORM B105-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-40-33 BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(33) INDX-COUNT
    MOVE '33,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(33).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(34) INDX-COUNT
    MOVE '34,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(34).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(35) INDX-COUNT
    MOVE '35,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(35).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(36) INDX-COUNT
    MOVE '36,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(36).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(37) INDX-COUNT
    MOVE '37,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(37).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(38) INDX-COUNT
    MOVE '38,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(38).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(39) INDX-COUNT
    MOVE '39,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(39).
PERFORM B105-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(40) INDX-COUNT
    MOVE '40,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(40).
PERFORM B105-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-48-41 BITS-1-8.
```

```
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(41) INDX-COUNT
    MOVE '41,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(41).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(42) INDX-COUNT
    MOVE '42,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(42).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(43) INDX-COUNT
    MOVE '43,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(43).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(44) INDX-COUNT
    MOVE '44,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(44).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(45) INDX-COUNT
    MOVE '45,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(45).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(46) INDX-COUNT
    MOVE '46,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(46).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(47) INDX-COUNT
    MOVE '47,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(47).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(48) INDX-COUNT
    MOVE '48,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(48).
PERFORM B1Ø5-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-56-49 BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(49) INDX-COUNT
    MOVE '49,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(49).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(5Ø) INDX-COUNT
    MOVE '5Ø,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(5Ø).
```

```
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(51) INDX-COUNT
    MOVE '51,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(51).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(52) INDX-COUNT
    MOVE '52,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(52).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(53) INDX-COUNT
    MOVE '53,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(53).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(54) INDX-COUNT
    MOVE '54,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(54).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(55) INDX-COUNT
    MOVE '55,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(55).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(56) INDX-COUNT
    MOVE '56,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(56).
PERFORM B1Ø5-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-SCTY-64-57 BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO TOTL-SEC-TABLE(57) INDX-COUNT
    MOVE '57,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(57).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO TOTL-SEC-TABLE(58) INDX-COUNT
    MOVE '58,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(58).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO TOTL-SEC-TABLE(59) INDX-COUNT
    MOVE '59,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(59).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO TOTL-SEC-TABLE(6Ø) INDX-COUNT
    MOVE '6Ø,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
```

```
        USER-SEC-TABLE(6Ø).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO TOTL-SEC-TABLE(61) INDX-COUNT
    MOVE '61,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(61).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO TOTL-SEC-TABLE(62) INDX-COUNT
    MOVE '62,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(62).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO TOTL-SEC-TABLE(63) INDX-COUNT
    MOVE '63,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(63).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO TOTL-SEC-TABLE(64) INDX-COUNT
    MOVE '64,' TO POWP-PARM-SNT-SCTY-ENTRY(INDX-COUNT)
        USER-SEC-TABLE(64).
PERFORM B1Ø5-CHECK-INDX-COUNT.
IF POWP-PARM-SNT-SCTY-ENTRIES > SPACE
    ADD 1 TO SUBM-COUNT
    MOVE POWP-PARM-SNT-SCTY-Ø1-64 TO
        POWP-PARM-TABLE-ENTRIES(SUBM-COUNT).
PERFORM B116-CHECK-SUBM-COUNT.
MOVE SPACE TO OP-CLASS-TABLE POWP-PARM-SNT-OPCLASS-ENTRIES.
MOVE LOW-VALUE TO WK-BYTES-3.
MOVE IE-OPR-CLASS TO WK-BYTE-3-R.
IF WK-BYTE-3 = 1
    MOVE 'OPCLASS=(1),' TO POWP-PARM-SNT-SCTY-ØØ-SCTY
    MOVE POWP-PARM-SNT-SCTY-ØØ TO POWP-PARM-1
    PERFORM B12Ø-SUBMIT
    MOVE 'Ø1' TO OP-CLASS-TABLE-ENTRY(Ø1)
    GO AØ4Ø-BUILD-SNT-RSL
ELSE
IF WK-BYTE-3 = Ø
    GO AØ4Ø-BUILD-SNT-RSL.
MOVE 'OPC' TO POWP-PARM-SNT-OPCLASS-ENTRY(Ø1).
MOVE 'LAS' TO POWP-PARM-SNT-OPCLASS-ENTRY(Ø2).
MOVE 'S=(' TO POWP-PARM-SNT-OPCLASS-ENTRY(Ø3).
MOVE 3 TO INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-OPR-CLASS-KEYS-Ø8-Ø1
    BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø1,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø1).
PERFORM B11Ø-CHECK-INDX-COUNT.
```

```
IF BITS-2 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø2,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø2).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø3,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø3).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø4,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø4).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø5,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø5).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø6,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø6).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø7,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø7).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø8,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø8).
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-OPR-CLASS-KEYS-16-Ø9
    BITS-1-8.
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-1 = '1'
    ADD 1 TO INDX-COUNT
    MOVE 'Ø9,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(Ø9).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '1Ø,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(1Ø).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '11,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
```

```
        OP-CLASS-TABLE-ENTRY(11).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '12,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(12).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-5 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '13,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(13).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-6 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '14,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(14).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-7 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '15,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(15).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-8 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '16,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(16).
PERFORM B11Ø-CHECK-INDX-COUNT.
MOVE 'DPBITX' TO SUB-NAME.
CALL SUB-NAME USING IE-OPR-CLASS-KEYS-24-17
    BITS-1-8.
IF BITS-1 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '17,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(17).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-2 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '18,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(18).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-3 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '19,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(19).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-4 = '1'
    ADD 1 TO INDX-COUNT
    MOVE '2Ø,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
        OP-CLASS-TABLE-ENTRY(2Ø).
PERFORM B11Ø-CHECK-INDX-COUNT.
IF BITS-5 = '1'
```

```
        ADD 1 TO INDX-COUNT
        MOVE '21,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
            OP-CLASS-TABLE-ENTRY(21).
    PERFORM B11Ø-CHECK-INDX-COUNT.
    IF BITS-6 = '1'
        ADD 1 TO INDX-COUNT
        MOVE '22,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
            OP-CLASS-TABLE-ENTRY(22).
    PERFORM B11Ø-CHECK-INDX-COUNT.
    IF BITS-7 = '1'
        ADD 1 TO INDX-COUNT
        MOVE '23,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
            OP-CLASS-TABLE-ENTRY(23).
    PERFORM B11Ø-CHECK-INDX-COUNT.
    IF BITS-8 = '1'
        ADD 1 TO INDX-COUNT
        MOVE '24,' TO POWP-PARM-SNT-OPCLASS-ENTRY(INDX-COUNT)
            OP-CLASS-TABLE-ENTRY(24).
    PERFORM B11Ø-CHECK-INDX-COUNT.
    IF POWP-PARM-SNT-OPCLASS-ENTRIES > SPACE
        ADD 1 TO SUBM-COUNT
        MOVE POWP-PARM-SNT-OPCLASS-Ø1-24 TO
            POWP-PARM-TABLE-ENTRIES(SUBM-COUNT).
    PERFORM B116-CHECK-SUBM-COUNT.
```

Editor's note: This article will be continued in the next issue.

Bob Botsis
Senior Systems Programmer (USA)

Dumping the linkage-stack

THE PROBLEM

Have you ever wondered why IBM made the linkage-stack available for programs running in any partition and restricted it for supervisor stuff? Well so did I. There is no feature to show the current content of the linkage-stack (ie in case of a program-check), nor can you use the linkage-stack if you are about to issue STXIT. I can't help with the STXIT (maybe a requirement will one day force the authors to rethink), but here is a routine to dump the linkage-stack without dumping all the memory in the machine.

THE SOLUTION

The program, FIXDUMP, runs at any time once after IPL and inserts code that front-ends all calls to $IJBHDUP (the central-dump-routine). When, at a later time, a DUMP is issued, the linkage-stack is dumped ahead of everything else.

I have not yet found a way to include the info into the dump-files; all it does is print the content straight to SYSLST.

LINKAGE-STACK DUMP ROUTINE

```
* $$ JOB JNM=ASSCOMP,CLASS=A
* $$ LST DISP=H,CLASS=X
// JOB ASSCOMP COMPILE FIXDUMP
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION ERRS,SYM,CATAL,NOXREF
 PHASE FIXDUMP
// EXEC ASMA9Ø,SIZE=(ASMA9Ø,64K),PARM='EXIT(LIBEXIT(EDECKXIT))'
FIXDUMP  CSECT
*********************************************************************
*    THIS PROGRAM MODIFIES $IJBHDUP SUCH THAT IT DUMPS LINKAGE_STACK
*      AS WELL
*     MARTIN TRUEBNER   2Ø.JUL.98
*********************************************************************
         PRINT NOGEN
FIXDUMP  RMODE 24
FIXDUMP  AMODE 24
         USING *,3
         BAKR  14,Ø
         LR    3,15
         CDLOAD $IJBHDUP,RETPNF=YES,SVA=YES
         LTR   15,15    CHECK RESULT
         BZ    ABØ5Ø
         WTO   'FIXDUMP COULD NOT LOCATE $IJBHDUP'
         LA    R15,16
         PR
ABØ5Ø    EQU   *
         LR    R4,R1             SAVE LOAD-POINT OF IJBHDUP
         CLC   =X'47FØFØ',X'Ø'(R4)  CHECK START OF IT
         BE    ABØ8Ø
         CLC   =X'9ØE3DØØC583ØFØØEØ7F3',Ø(R4) AM I ALREADY IN
         BNE   ABØ7Ø
         WTO   'FIXDUMP IS ALREADY ACTIVE'
         XR    R15,R15
         PR
ABØ7Ø    EQU   *
```

```
         WTO   'FIXDUMP DID NOT FIND START OF $IJBHDUP'
         LA    R15,16
         PR
ABØ8Ø    EQU   *
         CLI   4(R4),X'12'  SPACE FOR US?
         BH    AB12Ø
         WTO   'FIXDUMP COULD NOT FIND ENOUGH SPACE IN $IJBHDUP'
         LA    R15,16
         PR
AB12Ø    EQU   *
         MODESET KEY=ZERO
         LA     RØ,LINKAGE_LENGTH
         GETVIS LENGTH=(Ø),SVA=YES,LOC=BELOW
         LR     R6,R1
         LR     R7,RØ
         LR     R9,R7
         L      R8,=A(LINKAGE)
         MVCL   R6,R8   MOVE CODE TO SVA
         ST     R1,X'E'(,R4)   STORE ADDRESS OF MY CODE IN $IJBHDUO
         MVC    X'A'(4,R4),Ø(R4)  MOVE RETURN INSTR
*
*        NOW WE RELOCATE ALL ADDRESS-CONSTANTS
*
         USING  LINKAGE,R1
         LA     R15,CCW1
         ST     R15,CCB+8
         LA     R15,LINE1
         STCM   R15,7,CCW1+1
         LA     R15,LINE2
         STCM   R15,7,CCW2+1
         LA     R15,LINE3
         STCM   R15,7,CCW3+1
         LA     R15,LINE4
         STCM   R15,7,CCW4+1
         LA     R15,LINE5
         STCM   R15,7,CCW5+1
         LA     R15,LINE6
         STCM   R15,7,CCW6+1
         LA     R15,LINE7
         STCM   R15,7,CCW7+1
         DROP   R1
*
*   NOW THAT RELOCATION IS DONE WE PLUG IN START CODE
*
         MVC    Ø(1Ø,R4),=X'9ØE3DØØC583ØFØØEØ7F3' MOVE IN NEW CODE
         MODESET KEY=USER
         WTO    'FIXDUMP HAS SUCCESSFULLY IMPLANTED LINKAGE-DUMP'
         XR     R15,R15
         PR
         LTORG
         DROP  R3
```

```
LINKAGE  CSECT
LINKAGE  AMODE  31
LINKAGE  RMODE  24
         USING  *,R3
         CLI    X'11D'(R12),X'Ø5'  IS IT CALL NUMBER 5
         BNE    EXIT
         EXTRACT ID=CR,AREA=(S,CRØ),LEN=64
         L      R15,CRØ+6Ø        GET CR 15
         CLI    Ø(R15),X'Ø1'      IS IT IN USE
         BE     EXIT              SO WE EXIT
*
*   NOW EVERYTHING IS SUCH THAT WE CAN DUMP THE LINKAGE-STACK
*
         SH     R15,=H'168'       BACK UP ONE ENTRY
         MVI    LINE1_T,C'B'
         CLI    X'88'(R15),X'4Ø'   IS IT BRANCH TYPE
         BNE    TYPE_OKAY
         MVI    LINE1_T,C'P'
TYPE_OKAY EQU   *
         UNPK   LINE1_ADR(9),X'94'(5,R15)
         TR     LINE1_ADR(8),TRTAB
         MVI    LINE1_ADR+8,C' '
*   PREPARE FOR LOOP
         LA     R2,LINE3
         LA     RØ,32
LOOP     UNPK   Ø(9,R2),8(5,R15)
         TR     Ø(8,R2),TRTAB
         MVI    8(R2),C' '
         LA     R2,9(R2)
         LA     R15,4(R15)
         BCT    RØ,LOOP
*     NOW WE PRINT ALL LINES
         LA     R1,CCB
         EXCP   (1)
         WAIT   (1)
EXIT     LM     R14,R3,12(R13)
         B      1Ø(R15)
CRØ      DS     16F
CCB      CCB    SYSLST,CCW1
CCW1     CCW    X'Ø9',LINE1,X'4Ø',L'LINE1
CCW2     CCW    X'Ø9',LINE2,X'4Ø',L'LINE2
CCW3     CCW    X'Ø9',LINE3,X'4Ø',72
CCW4     CCW    X'Ø9',LINE4,X'4Ø',72
CCW5     CCW    X'Ø9',LINE5,X'4Ø',L'LINE5
CCW6     CCW    X'Ø9',LINE6,X'4Ø',72
CCW7     CCW    X'Ø9',LINE7,X'ØØ',72
LINE1  DC    C'LINKAGE STACK ANALYSIS- TYPE: X RETURN TO XXXXXXXX '
LINE1_T      EQU *-21,1
LINE1_ADR    EQU *-9,8
LINE2  DC    CL49'  REGS AT STACK(BAKR) Ø-7 FIRST LINE THEN 8 - 15 '
LINE5  DC    CL49' AREGS AT CALL Ø-7 AND 8 TO 15 IN SECOND LINE'
```

```
LINE3  DC    8CL9' '
LINE4  DC    8CL9' '
LINE6  DC    8CL9' '
LINE7  DC    8CL9' ',C' '
TRTAB  EQU   *-24Ø
       DC    C'Ø123456789ABCDEF'
       LTORG
LINKAGE_LENGTH EQU  *-LINKAGE
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
          END
/*
// EXEC LNKEDT
// OPTION PARTDUMP,NOSYSDMP
// EXEC FIXDUMP
/&
* $$ EOJ
```

Martin Truebner
(Germany)

Assembler – the series (part two)

NAMES OF LARGE NUMBERS

I used the word ‘billion’ in the first article in this series to refer to the number 1,000,000,000, forgetting that this can refer to a number one thousand times larger. I considered clarifying every reference in this and subsequent articles, but it seriously impacted the readability, so I am continuing what I started last issue: whenever you see the word ‘billion’, the number is 1,000,000,000 (one thousand million).

MORE RR

There are actually a large number of Register to Register (RR) Assembler instructions. All operate on two general-purpose registers and have an op code – the hexadecimal representation of the machine language instruction as it is stored in a program (object or executable) – in the range X'10' to X'1F' (see Figure 1).

LOAD POSITIVE REGISTER – LPR

```
LPR R2,R1
```

copies the contents of register one into register two, in essence removing a negative sign if one is present. The contents are obviously assumed to be numeric. LPR produces what is known in mathematics as an absolute value.

Although the leftmost (first) bit in the register can correctly be named the sign bit – it is on (the value one in binary notation) for negative numbers – turning the bit off will not do what LPR does.

Binary integers, which can also be used as fixed-point numbers such as currency values like 143.29, are stored in two’s complement notation. In a register, zero would be as you might expect:

```
X'ØØØØØØØØ'
```

```
1Ø - LPR - Load Positive Register
11 - LNR - Load Negative Register
12 - LTR - Load and Test Register
13 - LCR - Load Complement Register
14 - NR - And Register
15 - CLR - Compare Logical Register
16 - OR - Or Register
17 - XR - Exclusive Or Register
18 - LR - Load Register
19 - CR - Compare Register
1A - AR - Add Register
1B - SR - Subtract Register
1C - MR - Multiply Register
1D - DR - Divide Register
1E - ALR - Add Logical Register
1F - SLR - Subtract Logical Register
```

Figure 1: RR op codes

Likewise, one is:

```
X'00000001'
```

But negative one (-1) is:

```
X'FFFFFFFF'
```

Although this might at first seem strange, if you ignore overflow and subtract one from zero (all zero bits), you would get all one bits, which is all Fs in hexadecimal. It is easy to see the advantages of this approach when designing a computer to do subtraction, or even adding positive numbers to negative numbers.

The problem, of course, is that it is not immediately apparent how you would design a computer to change the sign of a number stored in this two's complement notation. In fact, it is easier than it looks, and the name of this notation hints at the solution.

To change the sign of a number, you take the complement (ie flip all the bits, making every one bit a zero and every zero bit a one) and add one. There is only one exception to this rule: the smallest possible, or most negative, number that can be stored in this notation is -2,147,483,648. When you do this operation to it, you get exactly the same number! The reason is that the largest positive number that can be stored is 2,147,483,647, one less than the complement of the most negative number.

In the 32 bits or 4 bytes of a register, the hexadecimal values are:

```
+2,147,483,648 - not possible to represent
-2,147,483,648 - X'80000000'
+2,147,483,647 - X'7FFFFFFF'
```

Since the most common multiply and divide instructions involve some 64-bit integers, it's worth looking at what these numbers look like in 64-bit notation:

```
+2,147,483,648 - X'0000000080000000'
-2,147,483,648 - X'FFFFFFFF80000000'
+2,147,483,647 - X'000000007FFFFFFF'
```

Of course, all this begs the question: what does LPR do if you give it the most negative number? The answer is, two things. It leaves the number untouched, which is as close to a correct answer as there is. Second, it sets a condition code to indicate an overflow has occurred.

In fact, LPR also sets the condition code to indicate whether the result is zero or not. Remember, it cannot be negative.

CONDITION CODE - PART OF THE PSW

It is decision-making that gives computers their power – a decision is made to execute a particular set of instructions based on the value of data or the occurrence of an event within the computer.

On System/3*x*0 CPUs, many machine instructions set the condition code, typically based on the value of the data they encounter. The condition code always has a value from 0 to 3. If a machine instruction sets the condition code, the *Principles of Operation* manual documents the specific circumstances that determine the value.

For example, *ESA/390 Principles of Operation* (SA22-7201-04, DZ9AR004) lists the ‘Resulting Condition Code’ for LPR:

- 0: result zero; no overflow
- 1: –
- 2: result greater than zero; no overflow
- 3: overflow

This indicates that the condition code is set to 2 if the result of the LPR instruction is a value that is greater than zero, but no overflow condition occurred.

Right below the condition code description, the manual indicates that a Fixed-Point Overflow program exception can occur. The program or the operating system may have set a bit in the processor known as the Fixed Point Overflow Mask. It determines whether this type of program exception causes a program interrupt or is ignored.

Program interrupts, in turn, may cause program abends depending on whether a trap is set for that type of interrupt. For example, the VSE Assembler STXIT macro can be used to define a routine that will handle program interrupts. This is typically done by determining what type of program interrupt occurred and deciding by type which interrupts will be allowed to cause a program abend and which can be handled by branching to an appropriate custom-written routine.

As LPR demonstrates, not all condition code values are possible for all instructions that set the condition code. In the case of LPR, the condition code can never be 1. If an instruction is not listed in the *Principles of Operation* as setting the condition code, the condition code value remains the same as it was before the instruction executed.

Both the condition code and the Fixed Point Overflow Mask are stored in the Program Status Word (PSW) – side by side, in fact. The PSW is a part of the CPU, just as the general registers are. If the CPU is interrupted when running a program, the PSW is saved in memory and restored when the program resumes running.

The condition code requires two bits within the PSW. As you might expect, in binary, the four possible condition code values are stored as:

- 0 – 00
- 1 – 01
- 2 – 10
- 3 – 11

If you reviewed the condition codes set by every machine instruction, you would see some patterns emerge. For example, most arithmetic instructions set the condition code based on the result of the operation:

- 0 – zero
- 1 – negative
- 2 – larger than zero
- 3 – overflow

The Load instructions discussed thus far follow the same pattern. The notable exception is LR, which does not set the condition code.

BRANCH

Branch is the term used to describe the process of moving from one place in the program to another. Whether or not a branch occurs can depend on the value of the condition code. You specify which of the possible condition code values indicate that a branch will occur. For example, to branch when the result of a calculation is non-negative, specify condition codes 0 and 2, and perhaps 3 if the branch should occur when an overflow occurs.

In a branch instruction, you specify all of the condition code values that could occur when you want the branch to occur. There is no direct way to branch when a particular condition code does not occur. Instead, you would specify all three of the other condition codes. For example, if you want to branch whenever overflow has not occurred, instead of specifying the overflow condition code, which has a value of 3, you would specify a branch instruction with condition codes 0, 1, and 2 set.

Branch instructions come in many flavours, and there are some shortcut forms that eliminate the need to remember and specify condition code values (these will be discussed in future articles).

LPR REGISTERS

Now, back to the LPR instruction. The two registers specified can be the same or different. Both

```
LPR R2,R1
LPR R1,R1
```

are valid.

LNR

As you might guess, there is also a Load Negative Register.

```
LNR R2,R1
```

copies the contents of Register 1 to Register 2, and forces the sign to be negative. The one exception is zero, which, in two’s complement notation, has a unique value: X'00000000' = 32 all-zero bits when stored in a register. Unlike some other numeric formats – packed and zoned decimal are two good examples – fixed-point binary does not have different values for positive and negative zero.

The official IBM definition of LNR reads as follows: “The two’s complement of the absolute value of the second operand is placed at the first-operand location.”

Interestingly,

```
LPR R2,R1
LNR R2,R1
```

produces the same results as

```
LNR R2,R1
```

It should therefore not surprise you that

```
LNR R2,R1
LPR R2,R1
```

produces the same results as

```
LPR R2,R1
```

LCR

Yes, there is also Load Complement Register.

```
LCR R2,R1
```

copies the numeric contents of Register 1 into Register 2, reversing the sign in the process. The two exceptions are zero and the most negative number (-2,147,483,648), which remain unchanged.

This is the mathematical equivalent of the unary minus sign:

$-n$

where *n* is a positive, zero, or negative number. For example:

$-(+1) = -1$

$-(0) = 0$

$-(-1) = +1$

LCR sets the condition code. An overflow condition occurs when LCR is performed on the most negative number.

LTR

Load and Test Register is a bit easier – it simply sets the condition code based on the value in the register. Although it is most frequently used to test a register:

```
LTR R1,R1
```

it can also be used to copy a register and test the value all with one instruction:

```
LTR R2,R1
```

CR

Rather than setting the condition code based on the arithmetic value in a single register, Compare Register

```
CR R1,R2
```

sets the condition code based on comparing the arithmetic value of two registers:

- 0 – R1 and R2 are equal
- 1 – R1 is less than R2
- 2 – R1 is greater than R2
- 3 – not defined

It is worth comparing these with the conditions codes set by Subtract Register.

```
SR R1,R2
```

sets

- 0 – result is zero
- 1 – result is negative
- 2 – result is greater than zero
- 3 – overflow

But that is not all that you know from the condition codes following an SR instruction. The meanings just quoted are from the IBM manual and refer to the result. But what of the values of the registers before the subtraction occurred? The CR condition code meanings can be applied to SR as well:

- 0 – R1 and R2 are equal

- 1 – R1 is less than R2
- 2 – R1 is greater than R2.

The one exception is the overflow condition – the result being a number with too many hexadecimal digits to fit in a 32-bit register – referring to both underflow and overflow. Overflow occurs only when a negative number is subtracted from a positive number. Likewise, underflow can result only from a positive number being subtracted from a negative number. In both cases, one or both of the numbers must also be very large positive or very small negative.

That is why some have called the Compare instruction a Subtract instruction without subtraction. This may seem an unimportant and unnecessarily confusing topic, but it illustrates the consistency in design of System/3*x*0 machine instructions, both in terms of condition code values and in terms of how the instructions operate. From a learning perspective, it also points out the value of understanding the basic principles that are used widely and consistently in the System/3*x*0 architecture.

CLR

Very similar is the Compare Logical Register instruction.

```
CLR R1,R2
```

sets the condition code based on comparing the contents of two registers. The condition code values are the same as CR (see above).

The difference between CR and CLR lies in how the comparison itself is done. Up to this point, all calculations and comparisons have assumed that the registers referenced contained signed fixed point binary numbers stored in two's complement. But a general-purpose register could just as easily contain:

- An unsigned fixed-point binary number
- A four-byte character string
- Two unsigned half-word fixed-point binary numbers
- Four unsigned one-byte fixed-point binary numbers

- 32 one-bit flags

or fewer flags, characters, or numbers, with the leftover space padded with binary zeros (or just left as is and ignored).

Why is this even an issue? To understand the problem, it's important to remember that, in two's complement, all negative numbers have one thing in common – the first bit is one. Because of this, a CR instruction cannot just blindly begin comparing bits. Otherwise, a negative number, with its first bit equal to one, would look larger than a positive number, with its first bit equal to zero. For example, in binary form:

-1 = 11111...11111

is less than

0 = 00000...00000

CLR, on the other hand, does the simple-minded comparison, one bit at a time, from the first bit to the last. So, it is important to know something about the data before deciding what Assembler instruction to use with it.

FUTURE ARTICLES

There are still more RR instructions worth reviewing. Some require a knowledge of Boolean logic, so an explanation of the basics will also be included, plus some practical applications, especially with bit flags and masking.

And if you ever get asked "What's new in Assembler?", next time you'll discover a snappier comeback than "Nothing much since 1964." One new RR-like instruction is so new that it cannot even be found in the latest edition of the *ESA/390 Reference Summary* – the Assembler programmer's companion.

And much more!

Jon E Pearkins
(Canada)

Eric Loriaux's System/390 Web site

In this article in our series of VSE Web site reviews, we visit Eric Loriaux's System/390 Web site, which can be found at <http://www.loriaux.com/s390/>

One of the most useful exclusively mainframe Web sites is (Eric) *Loriaux's System/390 home page (OS/390, MVS, VSE, VM)*. It opens with a deceptively simple screen, which notes the European mirror site (http://www.ping.be/~ping1475) as an alternative to the URL listed above. The cheery picture is labelled *Mainframers' meeting point*, and that's the mission of the site. An interactive and two-way communication resource, it provides a simple framework for interested parties to easily post and retrieve System/390-related information.

The first main group of links is headed *It's more than just plenty of useful links!* and is described as follows:

Check for new links, post your ads on-line, request a link, read other users' comments, ... and don't forget to subscribe to the newsletter! It's all FREE just like everything on this site!

Free is the right Internet price, so the first link to explore is *About*, to learn about Eric and his site:

Who am I? My name is Eric Loriaux. I'm 30 and I've worked as an OS/390 system-programmer in a social insurance company in Brussels (Belgium) since 1991. We mostly use IBM (OS/390 R2 on a 9672 R24 CMOS), Computer Associates, Landmark and Innovation products. Add to that my passion for telecoms (Internet, Compuserve, Fidonet) since 1992.

What's the purpose of this System/390 home page? To provide a home page for everybody working on System/390 systems (mostly OS/390, MVS, VM, VSE) and particularly system-programmers. When I started using the Internet I couldn't find anything useful that would have summarized the resources available in that particular field. I mean, a lot of companies already had sites of their own but I had the feeling there was a need for an independent S/390 home page, a place for mainframers to meet. So I made one myself. Please remember that this *page is under permanent construction. Its quality greatly depends on you.*

And he dedicates the site to visitors:

This page is meant to be yours too! Please remember: if you are a System/390 system-programmer or application programmer this page is meant to be yours. Comments and suggestions will be greatly appreciated. I'd like this page to become your home page on the Net, something useful rather than my personal toy. For that reason, tell me what you think about it (about the contents and the way the page is constructed too!). Please don't hesitate to send me mail! I would appreciate your filling out my questionnaire and signing the guestbook. If you like the page you should subscribe to the newsletter so we can stay in touch.

The *What's New* page lists more resources than are contained on most sites! Two sections, *Global evolution of this site* and *Recently added links*, give the macro and micro evolution viewpoint. The former lists functional changes, such as the ability to be linked for e-mail from the site while cloaking your e-mail address, and an improved site search capability; the latter category in fact includes multiple resources such as *IBM sites*, *Hardware*, *Software*, *Networking*, *Mailing lists*, *Newsgroups*, *Jobs*, and *People*, which mirror the site's structure. *Advertisements* links to date-ordered (most recent at top) recruiting ads. They're not categorized and are free-form text, but they're easy enough to search for your personal skills keywords. The other three links in this group – *Forms*, *Guestbook*, and *Newsletter* – are discussed below.

The meat of the site is linked in the next main page group, *But it's also a comprehensive list of S/390 sites*, about which Eric writes as follows:

I won't pretend everything is referenced here but at least, you should be able to find a good starting point when you're looking for a S/390 resource. Of course, you won't miss pages like "IBM", "Software" (ISV), "Consulting" (also includes training services), but "people", "newsgroups" and "mailing lists" will help you meet specialists on-line. Recruiters are waiting for you on the job page.

The first links category here is *IBM sites*. Most System/390 professionals have probably visited at least a few IBM Web sites which correspond to their main specialities. But this doesn't mean that this set of links is boring. First, the IBM Web site is sometimes a chore to navigate through, featuring links which aren't obvious and occasionally unfriendly search tools. So this listing may be the easiest way to find topics of interest. Second, because this list includes many – or all? – IBM Web sites of interest to System/390 workers, it makes interesting browsing. It's divided into categories such as *Main page of IBM Sites and IBM directories*, *Systems and subsystems*, *Hardware*, *Software packages*, and *Development tools*. The first category opens with IBM's main page, not very hard to find. But the next link, *Country-specific IBM information*, gives all country-specific IBM sites at a glance (130+ countries referenced), from Afghanistan to Zambia. If you've ever wondered about the missions of IBM laboratories, links here will satisfy you. For instance:

- *IBM Hursley Labs*: IBM United Kingdom Laboratories Limited came to Hursley in 1958. Over the years, the focus of our work has changed from hardware to systems software, for the IBM Networking Systems (NS) line of business. Development work for two of the NS business segments is done at Hursley: transaction processing systems and computer-aided telephony systems.
- *Almaden WebFarm Home Page*: The Almaden Research Center is one of six IBM Research Division facilities world-wide and one of the premier industrial research laboratories in the world. Its employees focus on a wide variety of basic and applied research in computer science, magnetic and optical storage technology, physical and materials science and technology, and scientific and technical application software.
- *IBM T.J. Watson Research Center*: The IBM Thomas J Watson Research Center is the headquarters for the IBM Research Division. Located in Westchester County, New York (Hawthorne and Yorktown Heights), we do research in physical sciences, computer sciences, systems technology, mathematics and information services, applications and solutions.

The *Systems and subsystems* category includes the VSE/ESA home page reviewed in *VSE Update* issue 32 (December 1998), along with CICS and DB2 Web pages. Hardware links range from *S/390 Integrated*

Server to *IBM Microelectronics Home Page (offering plenty of information about chip technology).*

Back on the main page, the next category is *Hardware*, linking to companies from AFP and Amdahl to ViON and Xerox, many well-known and many niche-focused. It's not a complete hardware vendor roster but can serve to locate useful resources and identify alternatives. The *Networking* page includes stalwarts such as Black Box and Cisco, along with many other sources of connectivity tools and skills. The large *Jobs* page (103KB) is divided into companies and people (recruiter) categories. With both categories ordered alphabetically and somewhat more polished entries than the ads page mentioned earlier, it's another useful resource for career planning.

The next link, *Information*, can be hazardous to your immediate productivity, since it offers a wealth of interesting links, divided into categories: *Commercial Services, Places Where You Can Order Books and Courses, Universities and High Schools, User Groups, Personal Home Pages, Other Sites*, and *Files Repository*. *Commercial Services* begins with a link to the Web site for *Beyond Computing*, an interesting and free (at least in the USA) IBM magazine. That's followed by a link to the *COBOL Foundation* which *distributes information on the COBOL industry to program developer organizations and individuals. The information includes the capabilities of COBOL language, available COBOL compiler systems, supporting development tools and utilities, available applications written in COBOL, development projects underway, and job opportunities available*. This is followed by *DB2 Magazine*, *Gartner Group*, and *Xephon*, whose publication you're reading. The *Books* link includes popular sites such as Amazon.com, along with many mainstream publishers and a few publishers (eg Maximum Press, Mike Murach & Associates) with special affinity for System/390 topics. The *User Groups* category is a mixture of cosmic (such as SHARE), regional (eg Tennessee VSE User Group, which notes *VSE is Alive and Well in Tennessee!*), and topical (Xplor International, which *aims to foster development of professionals working within the electronic document systems industry*). The collection of personal home pages addresses all factions of System/390 computing, DB2, MVS, CICS, REXX, etc. Leo Langevin's VSE/ESA Web Page, for instance, offers and recaps VSE events in the Chicago area. The *Books* page is organized by topic, from Assembler to VTAM. Some niches (eg PL/I, REXX) are sadly empty, but they're available to publishers and authors for posting titles.

The main page link to *Training and Consulting* offers an alphabetical list of vendors, world-wide, offering diverse services. If you're feeling insufficiently burdened by e-mail, visiting the *Mailing Lists* page offers a cure. As with other pages on this site, an alphabetical list is offered, ranging from ADSM and ASM370 to VSE and X400. The *Newsgroups* page follows a similar pattern, with entries from alt.cobol to comp.lang.rexx. The next page, *People*, gives a glimpse into the world-wide nature of our industry, with listings for many folk around the world. Entries vary in format and length, from just an e-mail link to a nearly complete resume. It's easy to look for friends and colleagues, or to search for geographic or functional similarities.

Nearing the end of the link categories, the *Software* page, one of the broadest collections of links, is probably also one of the most valuable. Though organized by company name rather than by software products offered, it serves as an aggregation of useful VSE links – among others – such as Barnard Software, B I Moyle Associates, Cross Access, and more.

The site includes a guestbook facility, enabling visitors to leave short or long messages about themselves, the site, or other matters. Many entries are deservedly complimentary, including this from Scott Sherer, president of NaSPA (a professional/technical organization, publisher of Technical Support magazine): *This is SUPER! More members should follow your lead and produce high quality web sites like yours. I'm very impressed!*.

Another powerful means of interacting with this site is the *Search Form*, linked just under the hand-shakers image on the main page. Areas are provided for entering *List of words to search*, *Type of matching*, *Boolean operator*, *Case*, and *Scope of search*; instructions clarify how these operands interact – including the bright red reminder *Remember that 'Word searched' field should only contain the searched strings/keywords! DON'T try to use Boolean operators there (that's what the Boolean pull-down menu is made for)*. Searching for VSE with no other terms indeed retrieved all pages on the site referencing

VSE, in the following categories: people, IBM's sites, information, jobs, mailing lists, newsgroups, software, training and consulting, cross-links, list of System/390 books.

This sort of search is clearly a powerful tool for creating a customized reference/resource list in specialized System/390 topics.

The last grouping on the main page is short, titled *And some other pages that you might be interested in*, and described as *Who references the site? All kinds of search engines. How you could use the logos*. The *Cross-Links* list identifies all the Web sites linking to Eric's site. While many of these sites are linked elsewhere on this site, this list, categorized by *Companies*, *Search Engines and Directories*, *People*, and *Other Sites*, can be used to research communities of interest, on the assumption that sites linking to Eric's site may also be interesting, and may also link to other interesting sites. I especially like how most entries are twofold, giving both the main page address for a site linking, and the specific page on which the link occurs. So, for example, there are links to companies such as MacKinney Systems and Macro 4, along with the pages of links that their Webmasters feel will interest visitors. This structure exploits the nature of the Web, allowing each site to link and be linked, adding together insights and favourites of diverse contributors. The *Copyright Note* page mostly offers to others the graphics with which Eric has illustrated his site; they're available for the modest price of acknowledging Eric as their source.

Saving the best for last, there are several ways to interact with Eric's Web page. Clicking on the *Newsletter* link from the main page displays a simple form for requesting a subscription to site updates – so instead of visiting and galloping through the *What's New* section, you can allow the e-mailed newsletter to highlight new entries in all sections of the site. This ties in nicely with the *Forms* link, which offers self-service data entry and updating for all areas of the site. The power of this site comes from its building momentum and the fact that it is becoming ever more complete, as more people and organizations discover it, link to it, and add their information for retrieval by others. The first form allows you to request addition of links to any of the site's pages – specifying URL and/or e-mail address, page to include the link, and description of resource available. The next link is a questionnaire and feedback form, on which Eric notes, *The purpose of this questionnaire is to help me know who you are and how to fit your needs the best I can. Remember: it won't be used for commercial purposes nor will you be harassed with mail. It's all up to you to fill it entirely or partially (for example, there's no obligation to fill in your real name).*

Finally, on the *Contact Page*, Eric answers a few *Frequently Asked Questions*. Regarding the rules and obligations to register as an organization to one of the pages, it's simply necessary to provide an adequate description of the product or service being offered. The free listing service merely requires enough information to categorize the entry and direct readers to receive or locate additional information. Eric notes that the updates newsletter has over 800 subscribers. Regarding locating products or services, Eric suggests combining the site's search capabilities with judicious use of other resources such as mailing lists, noting that it's impossible for one person or one site to include 100% of System/390 knowledge, and that subject experts on lists are more likely to be able to answer specialized questions. Targeting recruiters seeking applicants, he offers advice on the etiquette of on-line recruiting, suggests other venues for browsing and posting information, and closes with the following: *As you know, there's little unemployment in the mainframe world so I wish you good luck!*

The last question addressed is handling problems which may occur reading newsgroups, for example by reading them instead as mailing lists or by changing newsgroup (Usenet) servers used.

For people wanting to absorb this entire site, or to make it available as a local reference resource without needing constant Internet connection, the *Files Repository* page provides the entire Web site as a bundle, consisting of two files, which Eric describes as follows:

Download the entire site, archived in a ZIP file! Would you like to browse this site without having to be connected? Now it's all yours! Just download the two archive files and unzip them on your PC; all you have to do is browse the main file (index.htm) locally. You have to download both html component (15/01/99) – European date format, of course – and the images (13/07/98) at first. Next time you'll need the latest version, just download the HTML component and unzip it on *your local workstation. The reason for splitting the archive into two components is that the images won't change, most of the time.*

This site is clearly a labour of love on Eric's part, supplemented by sponsorship by Datatrain. Any System/390 practitioner can acquire useful resources here, and any organization offering System/390 products or services can achieve important visibility. The self-service forms available make it easy for all segments of the System/390 community to easily benefit from this rich and growing Web site.

Gabe Goldberg
Computers and Publishing (USA)

Contributing to *VSE Update*

Although the articles published in Xephon *Updates* are of a very high standard, the vast majority are not written by professional writers, and we rely heavily on our readers themselves taking the time and trouble to share their experiences with others. Many have discovered that writing an article is not the daunting task that it might appear to be at first glance. They have found that the effort needed to pass on valuable information to others is more than offset by our generous terms and conditions and the recognition they gain from their fellow professionals. Often, just a few hundred words are sufficient to describe a problem and the steps taken to solve it.

If you have ever experienced any difficulties with VSE, or made an interesting discovery, you could receive a cash payment, a free subscription to any of our *Updates*, or a credit against any of Xephon's wide range of products and services, simply by telling us all about it. For a copy of our *Notes for Contributors*, which explains the terms and conditions under which we publish articles, please write to the editor, Fiona Hewitt, at any of the addresses shown on page 2, or e-mail her on 100336.1412@compuserve.com.

VSE news

IBM has added VSE versions of its previously-announced DB2 Forms Version 1.0 for building and distributing application front-ends to DB2 workstation databases. Applications can be created by developers, governed by administrators, and run by end users on Windows 95, 98, and NT 3.51 or later.

For further information, contact your local IBM representative or visit the Web site at www.software.ibm.com.

* * *

Sterling Software has announced Version 3.0 of its VM:Webgateway Web-to-host software for using legacy applications from a Web browser while maintaining end-to-end security.

Users can Web-enable and Web-enhance all existing mainframe applications on VSE, OS/390, MVS, and VM, and include full-screen applications. It uses Secure Sockets Layer technology to encrypt data transmitted between Web browsers and the mainframe, and it uses client and server certificates that authenticate Web browser users.

For further information, contact:
Sterling Software, 1800 Alexander Bell Drive, Reston, VA 22091, USA.
Tel: (703) 264 8000.
Sterling Software Ltd, 75 London Road, Reading, Berks, RG1 5BS, UK.
Tel: (01734) 391139.
URL: www.sterling.com.

* * *

SEEC has announced its Re-engineering Workbench, a software renewal and transformation package designed for modernizing mainframe systems via Web-enablement, data warehousing, and package replacement.

It's built around SEEC's proprietary COBOL analysis and program-slicing technologies, and includes an NT-based Application Dictionary and integrated PC-based Application Analyst tools that are used to extract business rules and to develop high-level blueprints of the data structures and processes that make up the current mainframe system.

For further information, contact:
SEEC, Park West One, Suite 200, Cliff Mine Road, Pittsburgh, PA 15275, USA.
Tel: (412) 893 0300.
SEEC Europe, Suite 10, Hanover International, Pingewood, Reading, Berks, RG30 3UN, UK.
Tel: (01189) 505505.
URL: www.seec.com.

* * *