180

MVS update

September 2001

In this issue

MVS Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 33598
From USA: 01144 1635 33598
E-mail: Jaimek@xephon.com

North American office

Xephon/QNA
PO Box 350100,
Westminster, CO 80035-0100
USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Contributions

Articles published in *MVS Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, you can download a copy of our *Notes for Contributors* from www.xephon.com/contnote.html.

MVS Update on-line

Code from *MVS Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/mvsupdate.html; you will need to supply a word from the printed issue.

Editor

Jaime Kaminski

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.

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1992 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

Printed in England.

A REXX program to submit DEFRAGs

Recently I needed to automate a DEFRAG process for a number of volumes, mostly allocated to development, but with different criteria. We had DASD POOLs that needed daily DEFRAGs, and some that needed weekly DEFRAGs; for some the fragmentation index should be targeted to 0, while for others that was not necessary. Also, a few of the POOLs had volumes that should not be DEFRAGged, because some of the files that resided in them – like PSF libraries. At that time, we were still in the middle of an SMS conversion, and not all the volumes were SMS managed, so I needed to select volumes based on their volser, or their storage group if they were SMS managed.

Based on those premises, I wrote a REXX program that will generate DEFRAG JCL, accepting for input four different parameters. Of those four at least one storage group or one volser must be specified, and the volser may be fully qualified or it may be a generic. If it is a generic, than it must end with an '*'. The input parameters are based on keyword recognition, because this is easier to use than positional. The DFRG program can be invoked with up to four parameters:

- STOR (storage-group(s) name(s))
- VOL (VOLSER(s))
- EXCL (VOLSER(s))
- INDEX (fragmentation index).

The fragmentation index, if omitted, will assume a value of 10. The VOLSER(s) and storage group name(s), when more than one, must be blank delimited. There is no provision for any other kind of separator. Storage group names must be fully qualified. Fully qualified VOLSERs can be intermixed with generic VOLSERs for VOL() and EXCL().

When invoked, DFRG will execute an IDCAMS DCOLLECT in order to obtain the VOLSERs and DEVICE TYPEs of the ONLINE volumes for the parameters specified. If at least one volume is found ONLINE, the DFRG program will generate a JOB for the DEFRAG execution. This JOB will have a first STEP, which will create the SYSIN for the ADRDSSU utility used to perform the DEFRAG; then it will have an instream procedure for the execution of the aforementioned utility, and finally it will have a STEP for each volume to DEFRAG. If you were to invoke the DFRG program with something like:

```
     DFRG vol(MVS*) excl(MVS1*) index(0)
```

You could end by submitting JCL like this:

```
//USERIDX JOB (ACCT#),'DEFRAG VOLUMES',
//            MSGLEVEL=(1,1),
//            CLASS=W,MSGCLASS=X
//*
//MKDATA   EXEC PGM=ICEGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT2   DD  DISP=(,PASS),
//             DSN=&WK1,SPACE=(TRK,(1,Ø)),LRECL=8Ø
//SYSIN    DD  DUMMY
//SYSUT1   DD  *
  DEFRAG DDNAME(DASDØ1) -
         ADMINISTRATOR -
         FRAGMENTATIONINDEX(1Ø) -
         WAIT(2,2)
/*
//*
//DEFRAG  PROC VOLID=,UNIT=
//*
//DEFRAG  EXEC PGM=ADRDSSU,REGION=4M
//SYSPRINT DD  SYSOUT=*
//DASDØ1   DD  DISP=OLD,UNIT=&UNIT,VOL=SER=&VOLID
//SYSIN    DD  DISP=(OLD,PASS),DSN=&WK1
//*
//ENDPROC PEND
//*
//DFRGØØØ1 EXEC DEFRAG,VOLID=MVS2ØØ,UNIT=339Ø
//DFRGØØØ2 EXEC DEFRAG,VOLID=MVS2Ø1,UNIT=339Ø
//*
```

DFRG

```
/* REXX
                        ***********************
                        *     DFRG  2.Ø.Ø     *
                        ***********************
-                                                                     */
arg opt_all
parse value opt_all with "STOR("stor_group")"
parse value opt_all with "VOL("volser")"
parse value opt_all with "EXCL("exclude")"
parse value opt_all with "INDEX("indx")"
ok=1
```

```
if  stor_group="" & volser="" then
    do
        ok=0
        say""
        say"You MUST specify :"
        say"    STOR(storage group name(s))"
        say" or/and"
        say"    VOL(volid(s))"
        say"At least one of this parameters is mandatory"
        say""
    end
else
    nop
if  indx¬="" then
    do
        if  ¬datatype(indx,"W") | indx<0 | indx>999 then
            do
                ok=0
                say""
                say"If specified, the defragmentation index must be :"
                say"  ==> numeric"
                say"  ==> greater than or equal to 0"
                say"  ==> less than or equal to 999"
                say""
                say"If not specified, the default is 10"
                say""
            end
        else
            nop
    end
else
    do
        indx=10
    end
if  ok then
    do
        if  exclude¬="" then
            do
                optns="NODATAINFO EXV("exclude")"
            end
        else
            do
                optns="NODATAINFO"
            end
        call alloc_files
    end
else
    do
        say""
        say"Correct the options in error, and reissue the command"
        say""
    end
return
```

```
/* - - - - - - - - - - - - - */
alloc_files:
"alloc f(sysprint) shr reuse dummy"
hlqs=userid(),
    ||".D"date("J"),
    ||".T"space(translate(time(),,":"),0)
out_dsn="'"hlqs".OUTFILE'"
dd#1="A"time("S")
"alloc f("dd#1") new dsorg(PS) recfm(V B) lrecl(454)",
    "da("out_dsn") space (10 5) tracks release"
if  rc=0 then
    do
        in_dsn="'"hlqs".SYSIN'"
        "alloc f(SYSIN) new reuse dsorg(PS) recfm(F B) lrecl(80)",
            "da("in_dsn") space (1 1) tracks release"
        if  rc=0 then
            do
                dd#2="O"time("S")
                "ALLOC F("dd#2") WRITER(INTRDR) SYSOUT(A) LRECL(80)",
                    "RECFM(F)"
                if  rc=0 then
                    do
                        call format_sysin
                    end
                else
                    do
                        say"Error ("rc") on Internal Reader",
                                "Allocation"
                        say"DFRG Interrupted"
                    end
            end
        else
            do
                say"Error ("rc") allocating SYSIN file for DCOLLECT,",
                    in_dsn
                say"DFRG interrupted"
            end
        "alloc f(sysin) shr reuse da(*)"
    end
else
    do
        say"Error ("rc") allocating output file for DCOLLECT, "out_dsn
        say"DFRG interrupted"
    end
"alloc f(sysprint) shr reuse da(*)"
return
/* - - - - - - - - - - - - - - */
format_sysin:
if  stor_group¬="" then
    do  a=1 to words(stor_group)
        queue"  DCOLLECT OFILE("dd#1")",
            "STOG("word(stor_group,a)") "optns
    end
```

```
if  volser¬="" then
    do  a=1 to words(volser)
        queue"  DCOLLECT OFILE("dd#1")",
            "VOL("word(volser,a)") "optns
    end
"execio "queued()" diskw SYSIN (finis)"
if  rc=0 then
    do
        "alloc f(sysin) reuse old da("in_dsn") delete"
        "CALL *(IDCAMS)"
        if  rc=0 then
            do
                "alloc f("dd#1") old da("out_dsn") delete"
                "execio * diskr "dd#1" (finis stem volume_info.)"
                if  rc=0 then
                    do
                        if  volume_info.0>0 then
                            do
                                call process_data
                            end
                        else
                            do
                                say"No volumes were selected for DEFRAG"
                                say"DFRG interrupted"
                            end
                    end
                else
                    do
                        say"Error ("rc") on "out_dsn" READ"
                        say"DFRG interrupted"
                    end
                "free f("dd#1")"
            end
        else
            do
                say"Error ("rc") during DCOLLECT execution"
                say"Process state unknown"
                say"DFRG interrupted"
            end
    end
else
    do
       say"Error ("rc") on the WRITE for "in_dsn
       say"DFRG Interrupted"
       "dropbuf"
    end
return
/* - - - - - - - - - - - - - - */
process_data:
job_name=userid()||jobsuf()
queue"//"job_name" JOB (ACCT#),'DEFRAG VOLUMES',"
queue"//            MSGLEVEL=(1,1),"
queue"//            CLASS=W,MSGCLASS=X"
```

```
queue"//*"
queue"//MKDATA   EXEC PGM=ICEGENER"
queue"//SYSPRINT DD  SYSOUT=*"
queue"//SYSUT2   DD  DISP=(,PASS),"
queue"//            DSN=&WK1,SPACE=(TRK,(1,Ø)),LRECL=8Ø"
queue"//SYSIN    DD  DUMMY"
queue"//SYSUT1   DD  *"
queue"  DEFRAG DDNAME(DASDØ1) -"
queue"         ADMINISTRATOR -"
if  indx>Ø then
    do
        queue"         FRAGMENTATIONINDEX("indx") -"
    end
queue"         WAIT(2,2)"
queue"/*"
queue"//*"
queue"//DEFRAG  PROC VOLID=,UNIT="
queue"//*"
queue"//DEFRAG  EXEC PGM=ADRDSSU,REGION=4M"
queue"//SYSPRINT DD  SYSOUT=*"
queue"//DASDØ1   DD  DISP=OLD,UNIT=&UNIT,VOL=SER=&VOLID"
queue"//SYSIN    DD  DISP=(OLD,PASS),DSN=&WK1"
queue"//*"
queue"//ENDPROC PEND"
queue"//*"
do  a=1 to volume_info.Ø
    parse value volume_info.a with 25 volid 31 . 69 dev_type 77 .
    queue"//DFRG"right(a,4,"Ø")" EXEC DEFRAG,VOLID="volid",",
        ||"UNIT="strip(dev_type)
end
queue"//*"
zx=queued()
"execio "zx" diskw "dd#2" (finis)"
if  rc¬=Ø then
    do
        say"ERROR ("rc") on Internal Reader WRITE"
        say"Program Situation Unknown"
        say queued()" of "zx" records were left unprocessed on the",
           "Internal Reader"
        say"Program Interrupted"
        "dropbuf"
    end
else
    do
        say"JOB "job_name" SUBMITTED"
        say volume_info.Ø" volumes were selected for DEFRAG"
    end
"free f("dd#2")"
return
/* - - - - - - - - - - - - - - */
jobsuf:
address "ISPEXEC" "VGET jobsuf profile"
```

```
if  jobsuf="" then
    jobsuf="A"
else
    do
        tab_suf="ABCDEFGHIJKLMNOPQRSTUVXWYZØ123456789"
        p1=pos(jobsuf,tab_suf)
        if  p1=length(tab_suf) then
            jobsuf=left(tab_suf,1)
        else
            jobsuf=substr(tab_suf,p1+1,1)
    end
address "ISPEXEC" "VPUT jobsuf profile"
return jobsuf
/* - - - - - - - - - - - - - - */
```

Joao Bentes de Jesus
Systems Programmer
Mundial-Confiana SA (Portugal)

Analysing VLF statistics

The Virtual Lookaside Facility (VLF) enables an authorized program to store named objects in virtual storage managed by VLF and to retrieve these objects by name on behalf of users in multiple address spaces. VLF is designed primarily to improve performance by retrieving frequently-used objects from virtual storage rather than performing repetitive I/O operations from DASD. Some IBM products or components such as LLA, TSO/E, CAS, and RACF use VLF as an alternative way to access data. Since VLF uses virtual storage for its data space there are performance considerations each installation must weigh when planning for VLF.

IBM supplies a default VLF PARMLIB member (COFVLF00) that contains CLASS statements for the VLF classes used by IBM-supplied products. You might need to tailor some of these CLASS statements to meet your installation’s needs. You should tune the MAXVIRT parameter, which specifies the maximum amount of virtual storage that your installation wants VLF to use for the objects in the class. When you specify the MAXVIRT value, ensure that it is large enough to hold most or all of the frequently-used objects in a VLF class. An excessively small value tends to cause thrashing of the data in that VLF class, while an excessively large MAXVIRT value tends to increase the consumption of auxiliary storage because rarely-used data is paged out, rather than discarded. SMF record type 41, record subtype 3, allows you to capture SMF data related to the usage of VLF. If you request subtype 3, the system writes this record every 15 minutes.

The following SAS routine analyses SMF type 41 records to produce a report describing VLF memory usage.

SOURCE

```
//SMF41JOB JOB (Ø18Ø8),
//             "SYSTEM",
//             MSGCLASS=R,
//             MSGLEVEL=(1,1),
//             NOTIFY=&SYSUID,
//             CLASS=4
//*
//DELETE  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=R
//*
//SYSIN    DD *
     DELETE SYS2.SXSPØØ3.PDB.PROD
     SET MAXCC=ØØ
/*
//LOAD   EXEC CALLSAS,
//  OPTIONS='ERRORABEND SOURCE SOURCE2 MACROGEN'
//WORK     DD UNIT=SYSDA,SPACE=(CYL,(5Ø,5))
//SORTWKØ1 DD SPACE=(CYL,(&SORT)),UNIT=SYSDA  ALWAYS HAVE REAL SORTWORK
//SORTWKØ2 DD SPACE=(CYL,(&SORT)),UNIT=SYSDA  DD CARDS (DON'T USE DYNAM
//SORTWKØ3 DD SPACE=(CYL,(&SORT)),UNIT=SYSDA  ALLOCATION OF SORTWORK).
//PDB      DD DSN=SYS2.SXSPØØ3.PDB.PROD,
//         DISP=(,CATLG),
//         UNIT=SYSDA,VOL=SER=MNT$Ø2,SPACE=(CYL,(5,3))
//*
//SMF      DD DISP=SHR,DSN=SYS3.SMF.ARCHIVE.PROD
//*
//SOURCE   DD DISP=SHR,DSN=SYS2.SXSPØØ3.SAS.SOURCE
//SYSIN    DD *
MACRO _SMFFILE ;
 INFILE SMF ;
%
MACRO _SMFHDR ;
  LENGTH DEFAULT=4
         SMFTIME    8
  ;
  INFORMAT SYSTEM $CHAR4. ;
  FORMAT
         SMFTIME    DATETIME19.2
  ;
  LABEL  ID   ='SMF*RECORD*ID'
         SMFTIME='SMF*RECORD*TIME STAMP'
         SYSTEM ='SYSTEM*ID'
  ;
  RETAIN ID SMFTIME SYSTEM ;
  INPUT #1  MVSXAFLG           PIB1.
        #2  ID                 PIB1.
        #3  SMFTIME            SMFSTAMP8.
       #11  SYSTEM             $CHAR4.
       #;
%
 /*===================================================================*/
 /* SMF RECORD 41 (X'29') - DIV OBJECTS AND VLF STATISTICS           */
 /*===================================================================*/
MACRO _VARØ41
 TYPEØ41 (
          KEEP=
               SMFTIME  SYSTEM
               SMF41STY
               SMF41CLS
               SMF41MVT
               SMF41USD
               SMF41SRC
               SMF41FND
               SMF41ADD
               SMF41DEL
               SMF41TRM
               SMF41LRG
               MEM_USAGE_PCT
               HIT_RATIO
         )
%
MACRO _CDEØ41
IF ID=Ø41 THEN DO;
  LABEL
        SMF41STY='RECORD*SUBTYPE                           "
        SMF41CLS='VLF*CLASS*NAME                           "
        SMF41MVT='MAXVIRT*VALUE*IN 4K BLOCKS               "
        SMF41USD='VIRTUAL*STORAGE*USED* IN 4K BLOCKS       "
        SMF41_USAGE='VIRTUAL*STORAGE*USAGE                 "
        SMF41SRC='NUMBER OF*CACHE SEARCHS                  "
        SMF41FND='NUMBER OF*OBJECTS FOUND*IN CACHE         "
        SMF41ADD='NUMBER OF*OBJECTS ADDED*IN CACHE         "
        SMF41DEL='NUMBER OF*OBJECTS DELETED*FROM CACHE     "
        SMF41TRM='NUMBER OF*OBJECTS TRIMMED*FROM CACHE     "
        SMF41LRG='LARGEST OBJECT*ATTEMPTED*TO PUT IN CACHE "
        ;
  INPUT #19 SMF41STY PIB2.        /* FAIRE Ø4-3 = Ø1 (OFFSET) */
        #;
                                  /* SUBTYPE 3: VLF STATISTICS */
  IF SMF41STY EQ 3 THEN
      DO ;
         INPUT #57 SMF41OD4  PIB4.
               #61 SMF41LD4  PIB2.
               #63 SMF41ND4  PIB2.
                #;
         SMF41OD4=SMF41OD4-3 ;
         LOOP_ST_3:
           INPUT  #SMF41OD4     SMF41CLS  $CHAR8.
                                SMF41MVT  PIB4.
                                SMF41USD  PIB4.
                                SMF41SRC  PIB4.
                                SMF41FND  PIB4.
                                SMF41ADD  PIB4.
                                SMF41DEL  PIB4.
                                SMF41TRM  PIB4.
                                SMF41LRG  PIB4.
                  #;
           MEM_USAGE_PCT = SMF41USD / SMF41MVT ;
           HIT_RATIO     = SMF41FND / SMF41SRC ;
           OUTPUT TYPEØ41 ;
           SMF41ND4 = SMF41ND4 - 1;
           IF SMF41ND4 GT Ø THEN
             DO;
```

Systems Programmer (France) © Xephon 2001

NetRexx

NetRexx was designed by IBM Research Fellow Mike Cowlishaw as a Web-oriented version of his REXX language. For mainframers with a background in REXX it is a much simpler alternative to using Java as it requires less coding than Java to produce applications, applets and servlets for any Java Virtual Machine (JVM). Also, Java classes and beans are easily accessible. NetRexx can be used as a translator that produces Java source code. It also can be used as a runtime interpreter. There are two principal Redbooks about NetRexx; *Creating Java Applications Using NetRexx* (SG24-2216-00) and *VM/ESA Network Computing for Java and NetRexx* (SG24-5148-00).

You can download the latest version of NetRexx (Version 2.02) for free from IBM's Hursley research centre's Web site at the following URL: *http://www2.hursley.ibm.com/netrexx*. This site is a goldmine of information containing tutorials, slideshows, free documents and executables to download. If you want to subscribe to IBM's NetRexx mailing list visit: *http://ncc.hursley.ibm.com/majordomo/ibm-netrexx*.

© Xephon 2001

Measuring buffering activity for PDSE datasets

PDSE datasets have been around in MVS and OS/390 for several years. We have noticed in many of the installations that we have supported that PDSE datasets are only marginally utilized. The PDSE dataset has some clear advantages over the traditional PDS dataset, so why is it that the PDSE dataset is not used? One area that PDSE has an advantage over PDS is dataset performance. The PDSE is supposed to have a more efficient directory lookup mechanism. There is an excellent IBM Redbook, *Partioned Data Set Extended Usage Guide*, SG24-6101, which describes how the directory structure is architected within the PDSE.

Chapter 5 of the Redbook details which OS/390 facilities are used to help provide the performance enhancements for PDSE datasets. One of the key features is the use of a dataspace to help manage directory information as well as manage individual members. The dataspace is used as a high-speed cache to avoid sending physical I/O to the dataset where possible. We were curious to understand how we could measure how effective the dataspace was. We looked in the SMF manual and found that record type 42, subtype 1 contained some basic information that could be used to report statitics at the SMS storage-class level.

Although not at the dataset level, this information does provide a useful foundation to start to gain some insight on the effectiveness of the dataspace. We used the information from the Redbook and the SMF manual to develop the attached program, which reads the SMF data and produces a simple report showing the number of hits obtained for the member and the directory entry from the high-speed cache. The interval for producing these records is controlled by a parmlib specification. The JCL needed to execute this program is detailed below:

```
//yourjob card goes here
//STEP0001 EXEC PGM=EXTSM421
//STEPLIB  DD DISP=SHR,DSN=your.load.lib
//SYSUT1   DD DISP=SHR,DSN=your.unloaded.sms.data
//SYSUT2   DD optional,include standard dataset parms for this dataset
//SYSOUT   DD SYSOUT=*,RECFM=FBA,LRECL=133
//DETAILS  DD SYSOUT=*,RECFM=FBA,LRECL=133
```

SYSUT2 is an optional dataset that can be allocated to save the type 42 subtype 1 records in if you want to retain them. Three local macros, $ESAPRO, $ESASTG, and $ESAEPI, have been included. You can change to your own macros with minimal code changes. This program was developed and tested under OS/390 Version 2 Release 8 and DFSMS 1.5.

EXTSM421

```
         TITLE 'EXTSM421 - ANALYSE SMF TYPE 42 SUBTYPE 1 BMF RECORDS'
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* CSECT   : EXTSM421                                                  *
* MODULE  : EXTSM421                                                  *
* DESC    : EXTSM421 IS A ROUTINE WHICH READS SMF TYPE 42, SUBTYPE 1  *
*           RECORDS, AND PRODUCES A SIMPLE REPORT.                    *
* MACROS  : $ESAPRO $ESAEPI $ESASTG OPEN CLOSE DCB DCBD DCBE          *
*           GET PUT WTO                                               *
* DSECTS  : IHADCBD                                                   *
* INPUT   : SYSUT1   - SMF DATA                                       *
* OUTPUT  : SYSUT2   - OPTIONAL, IF PRESENT 42(1) RECORDS ARE COPIED  *
*                      TO IT.                                         *
*           SYSOUT   - MESSAGES DATASET                               *
*           DETAILS  - DETAIL REPORT FILE                             *
* PLIST   : NONE                                                      *
* CALLS   : NONE                                                      *
* NOTES   : 31-BIT ADDRESSING USED FOR ALL FILES.                     *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
EXTSM421 $ESAPRO R12,AM=31,RM=24
         SPACE 1
         OPEN  (UT3,(OUTPUT)),MODE=31
         USING IHADCB,R1               DECLARE A BASE
         LA    R1,UT3                  GET @(DCB WE JUST OPENED)
         TM    DCBOFLGS,DCBOFOPN       Q. OPEN CLEAN?
         BO    UT3_OPEN                A. YES, PROCEED
         DROP  R1
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE SYSUT3 DATASET.       *
* ISSUE A WTO TO USER, SET A RETURN CODE AND EXIT BACK TO OP. SYS.    *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
SYN_UT3  DS    ØH
         SPACE 1
         LA    R1,WTO_MSG              POINT TO THE WTO
         WTO   TEXT=ER_MSGØ1,                                          +
               ROUTCDE=(2,1Ø),                                         +
               DESC=(6),                                               +
               MF=(E,(1))
         SPACE 1
         MVC   RET_CODE,RCØØ1Ø         SET THE RETURN CODE
         B     EXIT_RTN                EXIT PROGRAM
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* WE WERE ABLE TO OPEN UP THE SYSUT3 DATASET.  NOW SEE IF WE CAN OPEN *
* UP THE SYSUT1 DATASET, WHICH IS THE SMF DATA WE WANT TO READ.       *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT3_OPEN DS    ØH
         SPACE 1
         OI    DCB_FLAG,UT3_O          INDICATE UT1 IS OPEN
         OPEN  (UT1,(INPUT)),MODE=31
         USING IHADCB,R1               DECLARE A BASE
         LA    R1,UT1                  GET @(DCB WE JUST OPENED)
         TM    DCBOFLGS,DCBOFOPN       Q. OPEN CLEAN?
         BO    UT1_OPEN                A. YES, PROCEED
         DROP  R1
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE SYSUT1 DATASET.       *
* PLACE A MESSAGE INTO THE SYSOUT DATASET AND EXIT BACK TO THE OP. SYS*
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
SYN_UT1  DS    ØH
         SPACE 1
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,EL_MSGØ2            GET THE LENGTH OF THE MESSAGE
         LA    R15,ER_MSGØ2            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         MVC   RET_CODE,RCØØ1Ø         SET THE RETURN CODE
         B     CLOSE_DS                EXIT PROGRAM
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* ISSUE A MESSAGE INDICATING SYSUT1 DATASET HAS OPENED.               *
* SEE IF WE CAN OPEN UP THE SYSUT2 DATASET.                           *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT1_OPEN DS    ØH
         SPACE 1
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ1            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ1            GET @(MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         OI    DCB_FLAG,UT1_O          INDICATE UT1 IS OPEN
         OPEN  (UT2,(OUTPUT)),MODE=31
         USING IHADCB,R1               DECLARE A BASE
         LA    R1,UT2                  GET @(DCB WE JUST OPENED)
         TM    DCBOFLGS,DCBOFOPN       Q. OPEN CLEAN?
         BO    UT2_OPEM                A. YES, PROCEED
         DROP  R1
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE SYSUT2 DATASET.      *
* PLACE A MESSAGE INTO THE SYSOUT DATASET.                            *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
SYN_UT2  DS    ØH
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,EL_MSGØ3            GET THE LENGTH OF THE MESSAGE
         LA    R15,ER_MSGØ3            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         B     UT2_OPEN                A. YES, PROCEED
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* ISSUE A MESSAGE INDICATING THAT SYSUT2 DATASET HAS BEEN OPENED.     *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT2_OPEM DS    ØH
         SPACE 1
         OI    DCB_FLAG,UT2_O          INDICATE UT2 IS OPEN
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ2            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ2            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* OPEN THE DETAILS DATASET.                                           *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT2_OPEN DS    ØH
         SPACE 1
         OPEN  (UT4,(OUTPUT)),MODE=31
         USING IHADCB,R1               DECLARE A BASE
         LA    R1,UT4                  GET @(DCB WE JUST OPENED)
         TM    DCBOFLGS,DCBOFOPN       Q. OPEN CLEAN?
         BO    UT4_OPEM                A. YES, PROCEED
         DROP  R1
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE DETAILS DATASET.     *
* PLACE A MESSAGE INTO THE SYSOUT DATASET.                            *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
SYN_UT4  DS    ØH
```

```
         SPACE 1
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,EL_MSGØ4            GET THE LENGTH OF THE MESSAGE
         LA    R15,ER_MSGØ4            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         B     UT4_OPEN                A. YES, PROCEED
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* ISSUE A MESSAGE INDICATING THAT DETAILS DATASET HAS BEEN OPENED.    *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT4_OPEM DS    ØH
         SPACE 1
         OI    DCB_FLAG,UT4_O          INDICATE UT2 IS OPEN
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ3            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ3            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         PUT   UT4,DH1
         PUT   UT4,DH2
         PUT   UT4,DH3
         MVC   UT4_CNT,F3              SET LINES PRINTED
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* OPEN PROCESSING IS COMPLETE.  BEGIN PROCESSING THE INPUT DATA.      *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT4_OPEN DS    ØH
         SPACE 1
         ZAP   TSMF_CNT,P_ZERO         INTIALIZE THE COUNTER
         ZAP   T42_CNT,P_ZERO          INITIALIZE THE COUNTER
         ZAP   T421_CNT,P_ZERO         INITIALIZE THE COUNTER
         SPACE 1
LOOP_UT1 DS    ØH
         SPACE 1
         GET   UT1
         CLC   Ø(2,R1),HALF6           Q. CHECK THE RECORD LENGTH
         BL    LOOP_UT1                A. SHORT RECORD, BYPASS
         LR    R2,R1                   GET @(RECORD JUST READ)
         USING SMF42,R2                INFORM THE ASSEMBLER
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* PERFORM SOME BASIC SCREENING ON THE CURRENT RECORD TO SEE IF IT IS  *
* ONE WE ARE LOOKING FOR.                                             *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
         AP    TSMF_CNT,P_ONE          BUMP THE RECORD COUNT
         CLC   SMF42RTY,TYPE42         Q. TYPE 42 RECORD
         BNE   LOOP_UT1                A. NO, GET THE NEXT RECORD
         AP    T42_CNT,P_ONE           BUMP THE RECORD COUNT
         CLC   SMF42STY,STYPE1         Q. SUBTYPE 1
         BNE   LOOP_UT1                A. NO, GET THE NEXT RECORD
         AP    T421_CNT,P_ONE          BUMP THE RECORD COUNT
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* NEED TO LOOK AT THE DATE TO SEE IF IT IS PRE 2ØXX. FROM THIS WE      *
* KNOW HOW TO SET THE DATE UP FOR DISPLAY.                             *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
         MVC   WSMF_DTE,SMF42DTE       GET THE TIME STAMP
         CLI   WSMF_DTE,X'ØØ'          Q. FIRST BYTE SET FOR 19?
         BNE   SMFD_2Ø                 A. NO, SET UP FOR 2ØØØ
         MVI   WSMF_DTE,X'19'          SET FOR 19XX
         B     WSMF_SET                DATE IS SET
         SPACE 1
SMFD_2Ø  DS    ØH
         SPACE 1
         MVI   WSMF_DTE,X'2Ø'          SET FOR 2ØXX
         SPACE 1
WSMF_SET DS    ØH
         SPACE 1
         MVC   HOLDTIME,SMF42TME       SAVE THE TIME
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* NOW WE NEED TO DETERMINE IF THE RECORD IS SPANNED.  WE DO THIS BY    *
* LOOKING AT THE SEGMENT DESCRIPTOR IN THE RDW AT THE BEGINNING OF     *
* THE RECORD. IF THE SEGMENT DESCRIPTOR IS ZERO, THEN WE SIMPLY        *
* ESTABLISH ADDRESSABILITY TO THE RECORD AND PROCEED TO PROCESS IT.    *
* IF THE SEGMENT DESCRIPTOR IS NON-ZERO, THEN WE NEED TO PUT THE THE   *
* PIECES OF THE RECORD TOGETHER IN A WORK AREA, AND THEN WE CAN SET    *
* ADDRESSABILITY TO IT AND PROCESS IT.                                 *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
         CLC   SMF42SGD,=H'Ø'          Q. IS THIS A SPANNED RECORD?
         BE    SGD_ZERO                A. NO
         LA    RØ,WB                   GET @(TARGET LOCATION)
         LH    R1,SMF42RCL             GET THE LENGTH OF THE SOURCE
         LR    R15,R1                  GET THE LENGTH OF THE SOURCE
         LR    R14,R2                  GET @(SOURCE LOCATION)
         MVCL  RØ,R14                  MOVE THE DATA
         LR    R3,RØ                   PRESERVE NEW TARGET LOCATION
         LA    R2,WB                   RESET THE BASE REGISTER
         GET   UT1
         LR    RØ,R3                   RESTORE TARGET LOCATION
         LR    R14,R1                  GET @(RECORD)
         LH    R1,SMF42RCL             GET LENGTH OF THE TARGET
         LH    R1,Ø(R14)               GET LENGTH OF SOURCE DATA
         S     R1,=F'4'                MINUS THE RDW
         LR    R15,R1                  GET THE LENGTH OF THE SOURCE
         LA    R14,4(,R14)             GET THE SOURCE LOCATION
         MVCL  RØ,R14                  MOVE THE REMAINDER OF THE DATA
         SPACE 1
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
* WHEN WE GET HERE WE ARE READY TO SET-UP BASE REGISTERS SO WE CAN    *
* PROCESS THE CURRENT RECORD.                                         *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
         SPACE 1
SGD_ZERO DS    ØH
         SPACE 1
         XR    R3,R3                   CLEAR REGISTER 3
         LA    R3,SMF42_LEN(R3,R2)     POINT PAST THE HEADER
         USING SMF42S1,R3              INFORM THE ASSEMBLER
         ICM   R4,B'1111',SMF42SCO     OFFSET TO STORAGE CLASS SUMMARY
         ICM   R3,B'1111',SMF42BMO     OFFSET TO BMF TOTALS
         LA    R3,Ø(R3,R2)             POINT TO BMF TOTALS
         LA    R4,Ø(R4,R2)             POINT TO STORAGE CLASS SUMMARY
         DROP  R3                      INFORM THE ASSEMBLER
         USING SMF42Ø1A,R3             INFORM THE ASSEMBLER
         USING SMF42Ø1B,R4             INFORM THE ASSEMBLER
         ICM   R5,B'1111',SMF42TNA     GET NUMBER OF STORAGE CLASSES
         SPACE 1
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
* PROCESS ALL OF THE STORAGE CLASSES.                                 *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
         SPACE 1
LOOP_SCS DS    ØH
         SPACE 1
         MVI   UT4_BUFF,C' '           PUT A BLANK IN BYTE 1
         MVC   UT4_BUFF+1(L'UT4_BUFF-1),UT4_BUFF NOW BLANK THE REST
         UNPK  UT4_DTE(L'UT4_DTE),WSMF_DTE UNPACK THE DATE
         SPACE 1
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
* THE TIME IS SAVED IN HUNDREDTHS OF A SECOND SINCE MIDNIGHT.  TO     *
* MAKE THE TIME DISPLAYABLE, WE NEED TO DIVIDE IT BY HOURS, MINUTES   *
* AND SECONDS.  WE DISCARD THE REMAINDER.  FOR THE CALCULATIONS BELOW *
* NOTE THE FOLLOWING:                                                 *
* IN HUNDREDTHS, 1 HOUR   = 36ØØØØ                                    *
*                1 MINUTE = 6ØØØ                                      *
*                1 SECOND = 1ØØ                                       *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
         SPACE 1
         L     R1,HOLDTIME             PICK UP THE TIME
         SLR   RØ,RØ                   CLEAR REGISTER ZERO
         D     RØ,F36ØØØØ              DIVIDE BY HOURS
         CVD   R1,DUBLWORK             MAKE IT DECIMAL
         OI    DUBLWORK+7,X'ØF'        FIX THE SIGN
         UNPK  UT4_TME(2),DUBLWORK+6(2) NOW MAKE IT DISPLAYABLE
         MVI   UT4_TME+2,C':'          FORMAT IT
         SRDL  RØ,32                   SHIFT IT BY 32 BITS
         D     RØ,F6ØØØ                DIVIDE BY MINUTES
         CVD   R1,DUBLWORK             MAKE IT DECIMAL
         OI    DUBLWORK+7,X'ØF'        FIX THE SIGN
         UNPK  UT4_TME+3(2),DUBLWORK+6(2) MAKE IT DISPLAYABLE
         MVI   UT4_TME+5,C':'          FORMAT IT
         SRDL  RØ,32                   SHIFT IT BY 32 BITS
         D     RØ,F1ØØ                 DIVIDE BY SECONDS
         CVD   R1,DUBLWORK             MAKE IT DECIMAL
         OI    DUBLWORK+7,X'ØF'        FIX THE SIGN
         UNPK  UT4_TME+6(2),DUBLWORK+6(2) MAKE IT DISPLAYABLE
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* NOW PICK UP THE DATA FROM THE SMF RECORD.                            *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
         MVC   UT4_PNN(L'SMF42PNN),SMF42PNN GET THE STORAGE CLASS NAME
         ICM   R14,B'1111',SMF42SRT    GET THE DATA PAGE READS
         CVD   R14,DUBLWORK            MAKE IT DECIMAL
         MVC   UT4_SRT(L'PATTERN2),PATTERN2 GET THE EDIT PATTERN
         ED    UT4_SRT(L'PATTERN2),DUBLWORK+4 EDIT THE DATA
         ICM   R14,B'1111',SMF42SRH    GET THE DATA PAGE READ HITS
         CVD   R14,DUBLWORK            MAKE IT DECIMAL
         MVC   UT4_SRH(L'PATTERN2),PATTERN2 GET THE EDIT PATTERN
         ED    UT4_SRH(L'PATTERN2),DUBLWORK+4 EDIT THE DATA
         ICM   R14,B'1111',SMF42SDT    GET THE DIRECTORY PAGE READS
         CVD   R14,DUBLWORK            MAKE IT DECIMAL
         MVC   UT4_SDT(L'PATTERN2),PATTERN2 GET THE EDIT PATTERN
         ED    UT4_SDT(L'PATTERN2),DUBLWORK+4 EDIT THE DATA
         ICM   R14,B'1111',SMF42SDH    GET DIRECTORY PAGE READ HITS
         CVD   R14,DUBLWORK            MAKE IT DECIMAL
         MVC   UT4_SDH(L'PATTERN2),PATTERN2 GET THE EDIT PATTERN
         ED    UT4_SDH(L'PATTERN2),DUBLWORK+4 EDIT THE DATA
         L     R14,UT4_CNT             GET THE LINE COUNTER
         C     R14,F6Ø                 Q. 6Ø LINE ALREADY PRINTED
         BNE   NOT_F6Ø                 A. NO, PRINT THE LINE
         PUT   UT4,DH1
         PUT   UT4,DH2
         PUT   UT4,DH3
         MVC   UT4_CNT,F2              INITIALIZE THE COUNTER
         SPACE 1
NOT_F6Ø  DS    ØH
         SPACE 1
         LA    R14,1(,R14)             INCREMENT THE COUNTER
         ST    R14,UT4_CNT             SAVE THE COUNTER
         PUT   UT4,UT4_BUFF
         LA    R4,SMF42Ø1B_LEN(,R4)    BUMP THE POINTER
         BCT   R5,LOOP_SCS             LOOP FOR ALL STORAGE CLASSES
         B     LOOP_UT1                GO GET ANOTHER SMF RECORD
         DROP  R2,R3,R4                INFORM THE ASSEMBLER
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* END OF FILE ON THE SMF INPUT DATA.  ISSUE A FINAL SET OF MESSAGES   *
* AND THEN GO THROUGH THE FILE CLOSE ROUTINES.                         *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
```

```
EOF_UT1  DS    ØH
         SPACE 1
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ6            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ6            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         MVC   UT3_BUFF+OD_MSGØ6-OK_MSGØ6(L'PATTERN3),PATTERN2
         ED    UT3_BUFF+OD_MSGØ6-OK_MSGØ6(L'PATTERN3),TSMF_CNT
         PUT   UT3,UT3_BUFF
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ4            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ4            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         MVC   UT3_BUFF+OD_MSGØ4-OK_MSGØ4(L'PATTERN1),PATTERN1
         ED    UT3_BUFF+OD_MSGØ4-OK_MSGØ4(L'PATTERN1),T42_CNT+1
         PUT   UT3,UT3_BUFF
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ5            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ5            GET @(ERROR MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         MVC   UT3_BUFF+OD_MSGØ5-OK_MSGØ5(L'PATTERN1),PATTERN1
         ED    UT3_BUFF+OD_MSGØ5-OK_MSGØ5(L'PATTERN1),T421_CNT+1
         PUT   UT3,UT3_BUFF
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* COMMON EXIT POINT.  EACH FILE IS TESTED TO DETERMINE THE STATUS.    *
* IF THE FILE IS OPEN, IT WILL BE CLOSED.                             *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
CLOSE_DS DS    ØH
         SPACE 1
         TM    DCB_FLAG,UT1_O          Q. IS THE FILE OPEN
         BNO   CLO_UT2                 A. NO, CHECK NEXT FILE
         CLOSE (UT1),MODE=31
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ7            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ7            GET @(MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         SPACE 1
CLO_UT2  DS    ØH
         SPACE 1
         TM    DCB_FLAG,UT2_O          Q. IS THE FILE OPEN
         BNO   CLO_UT4                 A. NO, CHECK NEXT FILE
         CLOSE (UT2),MODE=31
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ8            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ8            GET @(MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         SPACE 1
CLO_UT4  DS    ØH
         SPACE 1
         TM    DCB_FLAG,UT4_O          Q. IS THE FILE OPEN
         BNO   CLO_UT3                 A. NO, CHECK NEXT FILE
         CLOSE (UT4),MODE=31
         MVI   UT3_BUFF,C' '           BLANK IN BYTE 1
         MVC   UT3_BUFF+1(L'UT3_BUFF-1),UT3_BUFF COPY TO REMAINDER
         LH    R14,OL_MSGØ9            GET THE LENGTH OF THE MESSAGE
         LA    R15,OK_MSGØ9            GET @(MESSAGE)
         EX    R14,MVCT_UT3            PUT THE MESSAGE IN THE BUFFER
         PUT   UT3,UT3_BUFF
         SPACE 1
CLO_UT3  DS    ØH
         SPACE 1
         TM    DCB_FLAG,UT3_O          Q. IS THE FILE OPEN
         BNO   EXIT_RTN                A. NO
         CLOSE (UT3),MODE=31
         SPACE 1
EXIT_RTN DS    ØH
         SPACE 1
         $ESAEPI
         TITLE 'EXTSM421 - LITERALS'
         SPACE 1
F36ØØØØ  DC    F'36ØØØØ'               USED FOR TIME MANIPULATION
F6ØØØ    DC    F'6ØØØ'                 USED FOR TIME MANIPULATION
F1ØØ     DC    F'1ØØ'                  USED FOR TIME MANIPULATION
F6Ø      DC    F'6Ø'                   USED TO CONTROL PRINTING
F3       DC    F'3'                    USED TO CONTROL PRINTING
F2       DC    F'2'                    USED TO CONTROL PRINTING
HALF6    DC    H'6'                    USED TO TEST RECORD LENGTHS
MVCT_UT3 MVC   UT3_BUFF+1(*-*),Ø(15)   USED TO MOVE MESSAGES TO BUFFER
PATTERN1 DC    XLØ7'4Ø2Ø2Ø6B2Ø212Ø'    EDIT PATTERN FOR DATA
PATTERN2 DC    XLØ9'4Ø6B2Ø2Ø2Ø6B2Ø212Ø' EDIT PATTERN FOR DATA
PATTERN3 DC    CL16' '                  EDIT PATTERN FOR DATA
         ORG   PATTERN3                 ORG BACK
         DC    XLØ4'4Ø2Ø2Ø2Ø'
         DC    XLØ4'6B2Ø2Ø2Ø'
         DC    XLØ4'6B2Ø2Ø2Ø'
         DC    XLØ4'6B2Ø212Ø'
TYPE42   DC    AL1(42)                 USED TO TEST FOR TYPE 42 RECORDS
STYPE1   DC    AL2(1)                  USED TO TEST FOR SUBTYPE 1 RECS.
P_ZERO   DC    PL4'Ø'                  USED FOR PACKED OPERATIONS
P_ONE    DC    PL4'1'                  USED FOR PACKED OPERATIONS
         TITLE 'EXTSM421 - MESSAGES'
         SPACE 1
ER_MSGØ1 DC    C'SMF421-Ø1(E) ERROR OPENING THE SYSOUT DATASET, PROGRAM+
                TERMINATING'
EL_MSGØ1 DC    Y(*-ER_MSGØ1)
```

```
ER_MSGØ2 DC    C'SMF421-Ø2(E) ERROR OPENING THE SYSUT1 SMF INPUT DATASE+
               T. PROGRAM TERMINATING'
EL_MSGØ2 DC    Y(*-ER_MSGØ2)
ER_MSGØ3 DC    C'SMF421-Ø3(W) ERROR OPENING THE SYSUT2 DATASET, COPY OP+
               ERATION WILL N OT BE PERFORMED'
EL_MSGØ3 DC    Y(*-ER_MSGØ3)
ER_MSGØ4 DC    C'SMF421-Ø4(W) ERROR OPENING THE DETAILS REPORT DATASET.+
                PROGRAM OPERATION TERMINATED.'
EL_MSGØ4 DC    Y(*-ER_MSGØ4)
OK_MSGØ1 DC    C'SMF421-Ø1(I) SYSUT1 SMF INPUT DATASET OPENED.'
OL_MSGØ1 DC    Y(*-OK_MSGØ1)
OK_MSGØ2 DC    C'SMF421-Ø2(I) SYSUT2 SMF OUTPUT COPY DATASET OPENED.'
OL_MSGØ2 DC    Y(*-OK_MSGØ2)
OK_MSGØ3 DC    C'SMF421-Ø3(I) DETAILS OUTPUT DATASET OPENED.'
OL_MSGØ3 DC    Y(*-OK_MSGØ3)
OK_MSGØ4 DC    C'SMF421-Ø4(I) NUMBER OF SMF TYPE 42 RECORDS EXAMINED WA+
               S '
OD_MSGØ4 DC    Ø7CL1' '
OL_MSGØ4 DC    Y(*-OK_MSGØ4)
OK_MSGØ5 DC    C'SMF421-Ø5(I) NUMBER OF SMF TYPE 42 SUBTYPE 1 RECORDS E+
               XAMINED WAS '
OD_MSGØ5 DC    Ø7CL1' '
OL_MSGØ5 DC    Y(*-OK_MSGØ5)
OK_MSGØ6 DC    C'SMF421-Ø6(I) NUMBER OF SMF RECORDS READ FROM THE INPUT+
                DATASET WAS '
OD_MSGØ6 DC    CL(L'PATTERN3)' '
OL_MSGØ6 DC    Y(*-OK_MSGØ6)
OK_MSGØ7 DC    C'SMF421-Ø7(I) SYSUT1 SMF INPUT DATASET CLOSED.'
OL_MSGØ7 DC    Y(*-OK_MSGØ7)
OK_MSGØ8 DC    C'SMF421-Ø8(I) SYSUT2 SMF OUTPUT COPY DATASET CLOSED.'
OL_MSGØ8 DC    Y(*-OK_MSGØ8)
OK_MSGØ9 DC    C'SMF421-Ø9(I) DETAILS OUTPUT DATASET CLOSED.'
OL_MSGØ9 DC    Y(*-OK_MSGØ9)
         TITLE 'EXTSM421 - OUTPUT RECORDS'
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* USE THE ASSEMBLER TO HELP SET UP THE SPACING FOR THE OUTPUT LINE    *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
DH1_LFØ1 EQU   L'DH1_1
DH1_LFØ2 EQU   L'DH1_2+DH1_LFØ1
DH1_LFØ3 EQU   L'DH1_3+DH1_LFØ2
DH1_LFØ4 EQU   L'DH1_4+DH1_LFØ3
DH1_LFØ5 EQU   L'DH1_5+DH1_LFØ4
DH1_LFØ6 EQU   L'DH1_6+DH1_LFØ5
DH1_LFØ7 EQU   L'DH1_7+DH1_LFØ6
DH1_SPAC EQU   (L'DH1-DH1_LFØ7)/8
DH1      DC    CL133' '
         ORG   DH1
         DC    C'1'
         ORG   DH1+DH1_SPAC
DH1_1    DC    C' DATE  '
```

```
         ORG   *+DH1_SPAC
DH1_2    DC    C'  TIME  '
         ORG   *+DH1_SPAC
DH1_3    DC    C'STORAGE                       '
         ORG   *+DH1_SPAC
DH1_4    DC    C' MEMBER  '
         ORG   *+DH1_SPAC
DH1_5    DC    C' MEMBER  '
         ORG   *+DH1_SPAC
DH1_6    DC    C'DIRECTORY'
         ORG   *+DH1_SPAC
DH1_7    DC    C'DIRECTORY'
         ORG   DH1+L'DH1
         EJECT
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* USE THE ASSEMBLER TO HELP SET UP THE SPACING FOR THE OUTPUT LINE.   *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
DH2_LFØ1 EQU   L'DH2_1
DH2_LFØ2 EQU   L'DH2_2+DH2_LFØ1
DH2_LFØ3 EQU   L'DH2_3+DH2_LFØ2
DH2_LFØ4 EQU   L'DH2_4+DH2_LFØ3
DH2_LFØ5 EQU   L'DH2_5+DH2_LFØ4
DH2_LFØ6 EQU   L'DH2_6+DH2_LFØ5
DH2_LFØ7 EQU   L'DH2_7+DH2_LFØ6
DH2_SPAC EQU   (L'DH2-DH2_LFØ7)/8
DH2      DC    CL133' '
         ORG   DH2+DH2_SPAC
DH2_1    DC    C'       '
         ORG   *+DH2_SPAC
DH2_2    DC    C'        '
         ORG   *+DH2_SPAC
DH2_3    DC    C'CLASS                         '
         ORG   *+DH2_SPAC
DH2_4    DC    C'DATA PAGE'
         ORG   *+DH2_SPAC
DH2_5    DC    C'DATA PAGE'
         ORG   *+DH2_SPAC
DH2_6    DC    C'DATA PAGE'
         ORG   *+DH2_SPAC
DH2_7    DC    C'DATA PAGE'
         ORG   DH2+L'DH2
         EJECT
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* USE THE ASSEMBLER TO HELP SET UP THE SPACING FOR THE OUTPUT LINE.   *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
DH3_LFØ1 EQU   L'DH3_1
DH3_LFØ2 EQU   L'DH3_2+DH3_LFØ1
DH3_LFØ3 EQU   L'DH3_3+DH3_LFØ2
DH3_LFØ4 EQU   L'DH3_4+DH3_LFØ3
DH3_LFØ5 EQU   L'DH3_5+DH3_LFØ4
```

```
DH3_LFØ6 EQU   L'DH3_6+DH3_LFØ5
DH3_LFØ7 EQU   L'DH3_7+DH3_LFØ6
DH3_SPAC EQU   (L'DH3-DH3_LFØ7)/8
DH3      DC    CL133' '
         ORG   DH3+DH3_SPAC
DH3_1    DC    C'       '
         ORG   *+DH3_SPAC
DH3_2    DC    C'        '
         ORG   *+DH3_SPAC
DH3_3    DC    C'                              '
         ORG   *+DH3_SPAC
DH3_4    DC    C' READS   '
         ORG   *+DH3_SPAC
DH3_5    DC    C'READ HITS'
         ORG   *+DH3_SPAC
DH3_6    DC    C'  READS  '
         ORG   *+DH3_SPAC
DH3_7    DC    C'READ HITS'
         ORG   DH3+L'DH3
         TITLE 'EXTSM421 - DATASET CONTROL BLOCKS'
         SPACE 1
UT1DCBE  DCBE  RMODE31=BUFF,                                           +
               SYNAD=SYN_UT1,                                          +
               EODAD=EOF_UT1
         SPACE 1
UT2DCBE  DCBE  RMODE31=BUFF,                                           +
               SYNAD=SYN_UT2
         SPACE 1
UT3DCBE  DCBE  RMODE31=BUFF,                                           +
               SYNAD=SYN_UT3
         SPACE 1
UT4DCBE  DCBE  RMODE31=BUFF,                                           +
               SYNAD=SYN_UT3
         TITLE 'EXTSM421 - DEFINE THE DCB FOR THE SYSUT1 DATASET'
         SPACE 1
UT1      DCB   DDNAME=SYSUT1,                                          +
               DSORG=PS,                                               +
               MACRF=(GL),                                             +
               DCBE=UT1DCBE
         TITLE 'EXTSM421 - DEFINE THE DCB FOR THE SYSUT2 DATASET'
         SPACE 1
UT2      DCB   DDNAME=SYSUT2,                                          +
               DSORG=PS,                                               +
               MACRF=(PM),                                             +
               DCBE=UT2DCBE
         TITLE 'EXTSM421 - DEFINE THE DCB FOR THE SYSOUT DATASET'
         SPACE 1
UT3      DCB   DDNAME=SYSOUT,                                          +
               DSORG=PS,                                               +
               RECFM=FBA,                                              +
               LRECL=133,                                              +
               MACRF=(PM),                                             +
               DCBE=UT3DCBE
         TITLE 'EXTSM421 - DEFINE THE DCB FOR THE DETAILS DATASET'
         SPACE 1
UT4      DCB   DDNAME=DETAILS,                                         +
               DSORG=PS,                                               +
               RECFM=FBA,                                              +
               LRECL=133,                                              +
               MACRF=(PM),                                             +
               DCBE=UT4DCBE
         TITLE 'EXTSM421 - DYNAMIC AREA'
         SPACE 1
         $ESASTG
DUBLWORK DS    D
HOLDTIME DS    D
RET_CODE DS    F                       RETURN CODE FIELD
DCB_FLAG DS    AL1                     BYTE FOR FILE INDICATORS
UT1_O    EQU   B'1ØØØØØØØ'
UT2_O    EQU   B'Ø1ØØØØØØ'
UT3_O    EQU   B'ØØ1ØØØØØ'
UT4_O    EQU   B'ØØØ1ØØØØ'
TSMF_CNT DS    PL6                     COUNTER FOR TOTAL SMF RECORDS
T42_CNT  DS    PL4                     COUNTER FOR TOTAL TYPE 42 RECS.
T421_CNT DS    PL4                     COUNTER FOR TYPE 42 SUBTYPE 1
UT4_CNT  DS    F                       COUNTER TO CONTROL HEADINGS
WSMF_DTE DS    F
UT3_BUFF DS    XL133
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* USE THE ASSEMBLER TO HELP SET UP THE SPACING FOR THE OUTPUT LINE.   *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT4_LFØ1 EQU   L'UT4_PNN
UT4_LFØ2 EQU   L'UT4_SRT+UT4_LFØ1
UT4_LFØ3 EQU   L'UT4_SRH+UT4_LFØ2
UT4_LFØ4 EQU   L'UT4_SDT+UT4_LFØ3
UT4_LFØ5 EQU   L'UT4_SDH+UT4_LFØ4
UT4_LFØ6 EQU   L'UT4_DTE+UT4_LFØ5
UT4_LFØ7 EQU   L'UT4_TME+UT4_LFØ6
UT4_SPAC EQU   (L'UT4_BUFF-UT4_LFØ7)/8
         SPACE 1
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
* ACTUAL RECORD LAYOUT STARTS HERE.                                   *
* — + — + — + — + — + — + — + — + — + — + — + — + — + — *
         SPACE 1
UT4_BUFF DS    XL133
         ORG   UT4_BUFF+UT4_SPAC
UT4_DTE  DS    XL7
         ORG   *+UT4_SPAC
UT4_TME  DS    XL8
         ORG   *+UT4_SPAC
UT4_PNN  DS    XL3Ø
         ORG   *+UT4_SPAC
```

```
UT4_SRT  DS    XLØ9
         ORG   *+UT4_SPAC
UT4_SRH  DS    XLØ9
         ORG   *+UT4_SPAC
UT4_SDT  DS    XLØ9
         ORG   *+UT4_SPAC
UT4_SDH  DS    XLØ9
         ORG   UT4_BUFF+L'UT4_BUFF
WTO_MSG  WTO   'PLACE HOLDER',MF=L
WB       DS    XL8192
         TITLE 'EXTSM421 - SMF RECORD TYPE 42 MAPPING'
         IGWSMF SMF42_Ø1=YES
         TITLE 'EXTSM421 - MAP OUT THE DCB CONTROL BLOCK'
         DCBD  DSORG=(QS)
         END EXTSM421
```

$ESAPRO MACRO

```
         MACRO
&LABEL   $ESAPRO &AM=31,&RM=ANY,&MODE=P
.**********************************************************************
.*       THIS MACRO WILL PROVIDE ENTRY LINKAGE AND OPTIONALLY
.*       MULTIPLE BASE REGISTERS. TO USE THIS MACRO, YOU NEED TO
.*       ALSO USE THE $ESASTG MACRO. THE $ESASTG DEFINES THE SYMBOL
.*       QLENGTH WHICH OCCURS IN THE CODE THAT &ESAPRO GENERATES.
.*       IF YOU DO NOT CODE ANY OPERANDS, THEN REGISTER 12 WILL BE
.*       USED AS THE BASE. IF YOU CODE MULTIPLE SYMBOLS, THEN THEY
.*       WILL BE USED AS THE BASE REGISTERS.
.*
.*       EXAMPLES:
.*              SECTNAME $ESAPRO          = REG 12 BASE
.*              SECTNAME $ESAPRO 5        = REG 5 BASE
.*              SECTNAME $ESAPRO R1Ø,R11  = REGS 1Ø AND 11 ARE BASES
.**********************************************************************
*
         LCLA  &AA,&AB,&AC
*
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
RA       EQU   1Ø
R11      EQU   11
RB       EQU   11
```

```
R12      EQU   12
RC       EQU   12
R13      EQU   13
RD       EQU   13
R14      EQU   14
RE       EQU   14
R15      EQU   15
RF       EQU   15
*
FPRØ     EQU   Ø
FPR2     EQU   2
FPR4     EQU   4
FPR6     EQU   6
*
&LABEL   CSECT
&LABEL   AMODE &AM
&LABEL   RMODE &RM
*
         SYSSTATE ASCENV=&MODE         SET THE ENVIRONMENT
*
         B     $$$$EYEC-*(R15)         BRANCH AROUND EYECATCHER
         DC    AL1(($$$$EYEC-*)-1)     EYECATCHER LENGTH
         DC    CL8'&LABEL'             MODULE ID
         DC    CL3' - '
         DC    CL8'&SYSDATE'           ASSEMBLY DATE
         DC    CL3' - '
         DC    CL8'&SYSTIME'           ASSEMBLY TIME
         DC    CL3'   '                FILLER
*
$$$$F1SA DC    CL4'F1SA'               USED FOR STACK OPERATIONS
$$$$4Ø96 DC    F'4Ø96'                 USED TO ADJUST BASE REGS
*
$$$$EYEC DS    ØH
*
         BAKR  R14,Ø                   SAVE GPRS AND ARS ON THE STACK
         AIF   (N'&SYSLIST EQ Ø).USER12
         LAE   &SYSLIST(1),Ø(R15,Ø)    LOAD OUR BASE REG
         USING &LABEL,&SYSLIST(1)      LET THE ASSEMBLER KNOW
         AGO   .GNBASE
.USER12  ANOP
         MNOTE *,'NO BASE REG SPECIFIED, REGISTER 12 USED'
         LAE   R12,Ø(R15,Ø)            LOAD OUR BASE REG
         USING &LABEL,R12              LET THE ASSEMBLER KNOW
         AGO   .STGOB
.GNBASE  ANOP
         AIF   (N'&SYSLIST LE 1).STGOB
&AA      SETA  2
&AC      SETA  4Ø96
.GNBASE1 ANOP
*
         AIF   (&AA GT N'&SYSLIST).STGOB
&AB      SETA  &AA-1
```

```
         LR    &SYSLIST(&AA),&SYSLIST(&AB) GET INITIAL BASE
         A     &SYSLIST(&AA),$$$$4Ø96      ADJUST NEXT BASE
         USING &LABEL+&AC,&SYSLIST(&AA)    LET THE ASSEMBLER KNOW
&AA      SETA  &AA+1
&AC      SETA  &AC+4Ø96
         AGO   .GNBASE1
.STGOB   ANOP
*
         L     RØ,QLENGTH              GET THE DSECT LENGTH
*
         STORAGE OBTAIN,LENGTH=(RØ),LOC=(RES,ANY)
*
         LR    R15,R1                  GET @(OBTAINED AREA)
         L     R13,QDSECT              GET DISPLACEMENT INTO AREA
         LA    R13,Ø(R13,R15)          GET @(OBTAINED AREA)
         LR    RØ,R13                  SET REG Ø = REG 13
         L     R1,QLENGTH              GET THE LENGTH OF THE AREA
         XR    R15,R15                 CLEAR REG 5
         MVCL  RØ,R14                  INTIALIZE THE AREA
         MVC   4(4,R13),$$$$F1SA       INDICATE STACK USAGE
         USING DSECT,R13               INFORM ASSEMBLER OF BASE
.MEND    ANOP
*
         EREG  R1,R1                   RESTORE REGISTER 1
         MEND
```

$ESAEPI MACRO

```
         MACRO
         $ESAEPI
.**********************************************************************
.*       THIS MACRO WILL PROVIDE EXIT LINKAGE. IT WILL FREE THE
.*       STORAGE AREA THAT WAS ACQUIRED BY THE $ESAPRO MACRO. YOU
.*       CAN OPTIONALLY PASS IT A RETURN CODE VALUE. THIS VALUE IS
.*       EITHER THE LABEL OF A FULL WORD IN STORAGE, OR IT IS A REG-
.*       ISTER. AS WITH THE $ESAPRO MACRO, YOU NEED TO USE THE $ESASTG
.*       MACRO. THE SYMBOL QLENGTH WHICH OCCURS IN THE CODE THAT IS
.*       GENERATED BY THIS MACRO IS DEFINED BY $ESASTG
.*
.*       EXAMPLES:
.*              $ESAEPI          = NO RETURN CODE SPECIFIED
.*              $ESAEPI (R5)     = RETURN CODE IS IN REG 5
.*              $ESAEPI RETCODE  = RETURN CODE IS IN THE FULLWORD AT
.*                                 RETCODE
.**********************************************************************
         AIF   (N'&SYSLIST EQ Ø).STGFRE
*
         AIF   ('&SYSLIST(1)'(1,1) EQ '(').REGRC
         L     R2,&SYSLIST(1)          GET RETURN CODE VALUE
         AGO   .STGFRE
.REGRC   ANOP
         LR    R2,&SYSLIST(1,1)        GET RETURN CODE VALUE
```

```
.STGFRE  ANOP
*
         L     RØ,QLENGTH              GET THE DSECT LENGTH
*
         STORAGE RELEASE,LENGTH=(RØ),ADDR=(R13)
*
         AIF   (N'&SYSLIST NE Ø).SETRC
         XR    R15,R15                 CLEAR THE RETURN CODE
         AGO   .MEND
.SETRC   ANOP
         LR    R15,R2                  SET THE RETURN CODE
.MEND    ANOP
         PR                            RETURN TO CALLER
* FOR ADDRESSABILITY PURPOSES
         LTORG
         MEND
```

$ESASTG MACRO

```
         MACRO
         $ESASTG
.*********************************************************************
.*       THIS MACRO IS USED IN CONJUNCTION WITH THE $ESAEPI AND $ESAPRO
.*       MACROS. IT PROVIDES A Q TYPE ADDRESS CONSTANT WHICH WILL CON-
.*       THE LENGTH OF THE DSECT. A REGISTER SAVE AREA ID PROVIDED AS
.*       WELL.
.*
.*       EXAMPLES:
.*              $ESASTG
.*      XXX     DC    F          = DEFINE ADDITIONAL STORAGE AREA
.*      YYY     DC    XL255
.*       .       .      .
.*       .       .      .
.*       .       .      .
.*********************************************************************
RCØØØØ   DC    F'Ø'                    USED TO SET RETURN CODES
RCØØØ4   DC    F'4'                    USED TO SET RETURN CODES
RCØØØ8   DC    F'8'                    USED TO SET RETURN CODES
RCØØØC   DC    F'12'                   USED TO SET RETURN CODES
RCØØ1Ø   DC    F'16'                   USED TO SET RETURN CODES
QDSECT   DC    Q(DSECT)                DEFINE A QCON
QLENGTH  CXD                           LET ASM CALCULATE THE LENGTH
DSECT    DSECT
         DS    18F                     SET ASIDE REGISTER SAVE AREA
         MEND
```

Enterprise Data Technologies (USA)

Machine instructions

INTRODUCTION

Over the last few years many machine instructions have been added to the *Principles of Operations* manual (IBM form number SA22-7201). Some of these are eminently useful for the application programmer.

This article discusses four groups of instructions:

- ‘Long’instructions:
  MVCL – Move Characters Long.
  CLCL – Compare Characters Long.
  In the meantime, two further ‘long extended’ instructions have been added: MVCLE and CLCLE. These two newer instructions are similar to the ‘long’ instructions described here, but use a 32-bit length rather than a 24-bit length.

- Program linkage instructions:
  BAKR – branch and stack.
  PR – program return.

- The MVCIN (Move Characters Inverse) instruction.

- ‘String’processing instructions:
  MVST – move string.
  SRST – search string.
  CUSE – compare until substring equal.
  CLST – compare logical string.

Although these are not the only application programmer-oriented instructions that have been introduced since the venerable days of the IBM/360 (for example, the addressing mode instructions, BASR, BASSM, etc), the utility of these other instructions is generally well appreciated. This article does not handle the ‘long’ instructions in detail, but just discusses those aspects not so well known, in particular some useful side-effects.

THE LONG INSTRUCTIONS

The ‘long’ instructions (MVCL, CLCL) process two operands – the lengths and addresses, which are each contained in an even-odd register pair. Each operand can have a length in the range 0...2 24-1 (16 MB). If the two lengths differ, the operand with the shorter length is implicitly padded with the specified pad character to the length of the longer operand.

The contents of the associated ‘address’ register are ignored if the length is zero. Because the processing time for these instructions can be long (even a fast CPU requires a significant amount of time to transfer 16 MB of data), these instructions can be interrupted. Although any such interruption is transparent to the program, it does mean that the specified general registers must be used to store the current instruction parameters, that is, the register contents change during the course of execution. In particular, the specified general registers reflect the state at the end of the execution. This particular side-effect can be useful even when the operands are less than 256 bytes long, that is, MVC (etc) could have been used.

Note: register 0 can be used as a normal addressing register, ie the usual restriction that register 0 is equivalent to address 0 does not apply here.

Format: MVCL *destination,source*

CLCL *comparand1,comparand2*

Before processing:

- *rs* – start address of source field, either 24-bit or 31-bit address depending on the addressing mode; the start address is ignored if the length of the source field is zero.
- *rs'* – length of source the field (low-order 3 bytes), pad byte (high-order byte).
- *rd* – start address of the destination field, either 24-bit or 31-bit address depending on the addressing mode.
- *rd'* – length of the destination field (low-order 3 bytes), high-order byte not used.

After processing:

- *rs* – one byte past the end of the source field.
- *rs'* – number of bytes of the source field that were not moved (non-zero only if the destination field was shorter than the source field).
- *rd* – one byte past the end of the destination field.
- *rd'* – always zero.

The schematic processing is shown in Figure 1:

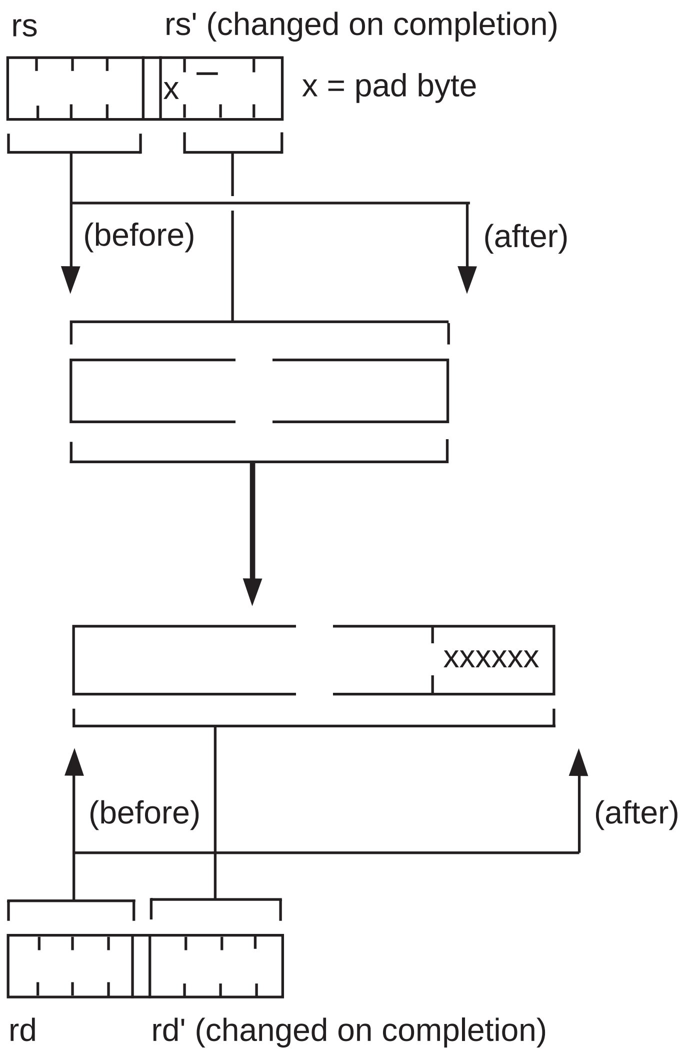


Figure 1: Schematic processing of the long instructions

INITIALIZING A LONG FIELD

One use of the MVCL instruction is to initialize a field. Because the pad character can be used as the initialization, the source field can be omitted (length = 0 and no address is required). For example:

```
         LA    Ø,TARGET         Address of target field
         L     1,=A(L'TARGET)   Length of target field
         L     15,=X'ffØØØØØØ'  ff = initialization byte
         MVCL  0,14
```

Filling a target field

Many applications need to fill a target field with variable length subfields; creating a print line image is a typical application here. Normally the EX instruction on an MVC instruction would be used in such a situation. Such an approach requires quite a lot of housekeeping overhead: reducing the length of the field being moved and maintaining the current address in the target field. Although the MVCL instruction requires more registers, the side-effects (update of the current address) can simplify the processing logic. Example (using 'traditional' code):

```
         LA    1,PA        Start address of target
* move first subfield
         LA    14,FLD1     Address of source field
         LA    15,L'FLD1   Length of field
         SH    15,=H'1'    Length-code (= length -1)
         EX    15,MOVE     Move source field
         LA    1,1(1,15)   Update target address
* move second subfield
         LA    14,FLD2     Address of source field
         LA    15,L'FLD2   Length of field
         SH    15,=H'1'    Length-code
         EX    15,MOVE     Move source field
         LA    1,1(1,15)   Update target address
** etc.
* EX-instruction
MOVE     MVC   Ø(Ø,1),Ø(14)
Example (using MVCL):
         LA    Ø,PA         Start address of target
         L     1,=A(L'PA)   Length of target
* move first subfield
         LA    14,FLD1      Address of source field
         LA    15,L'FLD1    Length of field
         MVCL  Ø,14         Move source field
* move second subfield
         LA    14,FLD2      Address of source field
         LA    15,L'FLD2    Length of field
         MVCL  Ø,14         Move source field
** etc.
```

To summarize, the ‘traditional’ approach is as follows:

- One initialization instruction.
- Five instructions per subfield.
- One housekeeping instruction (ex-target).

Whilst using MVCL is as follows:

- Two initialization instructions.
- Three instructions per subfield.

As can be seen, using MVCL is always shorter, even ignoring the two advantages that it offers:

- The target field bounds are explicitly checked.
- Long source fields (ie fields longer than 256 bytes) can be used without requiring any code changes.

Note: because the fields concerned are fixed, there is no requirement to set the lengths dynamically at execution-time, but the example serves to illustrate the techniques involved.

Comparison

Compared with the CLC instruction, the CLCL instruction has two advantages (in addition to the support of long operands):

- The two fields do not need to have the same length (the shorter field is right-padded with the specified pad character to the length of the longer field).
- At the end of the comparison, the registers indicate where the comparison stopped: both the respective addresses and the remaining number of bytes not compared.

For example:

```
...
         LA    Ø,FLD1    Start address of field 1
         L     1,LFLD1   Length of field 1 + pad-character
         LA    14,FLD2   Start address of field 2
         L     15,LFLD2  Length of field 2 + pad-character
         CLCL  Ø,14      Compare fields
         BL    LOW       Field 1 low
         BH    HIGH      Field 1 high
         BE    EQUAL     Fields identical (possibly padded)
LOW      SL    15,LFLD2
         LCR   15,15     Number of compared bytes
HIGH     SL    1,LFLD1
         LCR   1,1       Number of compared bytes
EQUAL    NOPR  Ø ...     Process identical fields
**
FLD1     DC    C'alphabetagamma'
LFLD1    DC    XL1'ØØ',AL3(L'FLD1)  pad-character + field length
FLD2     DC    C'alphabet'
LFLD2    DC    C'a',AL3(L'FLD2)     pad-character + field length
...
```

PROGRAM LINKAGE INSTRUCTIONS

The program linkage instructions discussed here use the hardware stack to save and restore the current program environment. Although the hardware (linkage) stack can be used for general register (and access register) housekeeping, its main use for the application programmer is to solve some problems that can occur in mixed addressing mode environments rather than standard program save-areas, in particular when an AMODE(31) program invokes an AMODE(24) program. The techniques discussed here apply only to the called program; the calling program is not affected, ie the usual methods of invoking a subprogram are retained (eg CALL macro).

Note: these program linkage instructions have additional functionality of particular interest for use in control programs.

LINKAGE STACK

The linkage stack is a system facility. Hardware instructions are provided for its use. Exceptions are signalled when its bounds, etc, are exceeded. The following two stack exceptions are of primary interest for the application programmer:

- Stack-Empty exception (ie an attempt was made to retrieve a stack entry (eg with the PR instruction) although the stack is empty). This is normally caused by unpaired BAKR-PR instructions.

- Stack-Full exception. The linkage stack has a finite size, but it is normally adequate.

The linkage stack provides the following major advantages compared with traditional techniques:

- Whereas the linkage stack saves both the access registers and the general-purpose registers; the SAVE macro saves only the general-purpose registers.
- The linkage stack is hardware-protected.

BAKR (BRANCH AND STACK)

The Branch and Stack instruction saves the current registers and the access registers (0-15), and the current PSW (including addressing mode) in the (hardware) linkage stack (additional information of limited interest for the application programmer is also saved).

Format: BAKR *retaddr,jumpaddr*

- *retaddr* – return address. The register that contains the address (including the addressing mode) to which a subsequent return is made. It is 14 when standard linkage procedures are used.
- *jumpaddr* – jump address. The register that contains the address (including the addressing mode) to which a control is to be passed. This is normally 0, which means no branch is made and the next sequential instruction is executed.

BAKR when used with PR for application program linkage has the following general form:

```
* save calling environment
         BAKR  14,Ø              Use hardware stack
         LA    13,SA             Program save area
         MVC   4(4,13),=C'F1SA'  Indicate linkage stack used
...
* restore calling environment
         L     15,..             Load program return code
         PR    ,                 Program return
**
SA       DS    18F
```

For simplicity, save-area chaining and unchaining is omitted.

PR (PROGRAM RETURN)

The Program Return instruction restores the general registers and access registers (2 through 14) from the linkage stack and returns in the correct addressing mode to the saved location. Although other information is restored, this information is of limited interest to the application programmer.

Format: PR

Note: The PR instruction does not have any operands.

MOVE INVERSE

The MOVE INVERSE instruction will reverse the contents of a field. Other than the requirement to explicitly reverse the contents of a field, there are some circumstances when it can be simpler to process reversed data.

The MVCIN instruction is unique as an instruction in that it transfers a source field starting from the back.

Format: MVCIN *destination*(*length*),*sourceend*

The schematic processing is shown in Figure 2:

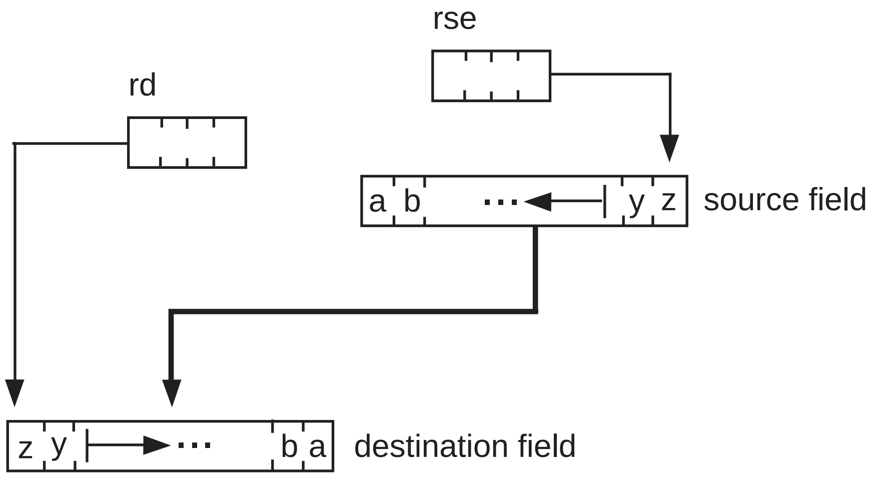


Figure 2: Schematic processing of the MVCIN instruction

Before processing:

rd: Start (lowest numbered byte) address of the destination field.

rse: End (highest numbered byte) address of the source field.

Example: this example finds the length of the significant data in a text field, ie the start of trailing blanks. This is conceptually the same as searching for the first occurrence of a blank in the reversed field contents, for which it is possible to use normal search logic (eg with the SRST (Search String) instruction):

```
...
         MVCIN TEMP,FLDE          Reverse field contents
         LA    4,TEMP             Search field (start)
         LA    3,TEMP+L'FLD-1     Search field (end)
         SR    Ø,Ø                Clear register
         IC    Ø,=C' '            Search character
         SRST  3,4                Search string
         BH    NOTFOUND
* delimiter byte found (Register 3 updated)
         SR    3,4
* Register 3: displacement from right
         LCR   3,3                Negative displacement
         LA    3,FLDE(3)          Address of found character
NOTFOUND NOPR  Ø                  Tag

TEMP     DS    CL256
FLD      DC    CL256'The rain in Spain'
FLDE     EQU   *-1
...
```

STRING PROCESSING INSTRUCTIONS

ESA/390 provided a number a new instructions designed to improve the processing of string-oriented fields typically used in C programs. For example, character fields in C are terminated with a X'00'. However, because native S/370 character-oriented instructions require an explicit length, the field would need to be processed twice without these new instructions – once to find the actual length and once to perform the actual processing.

However, these instructions are not restricted to string processing, they can be used effectively in many other application areas. Because the string instructions are ‘long’ instructions, they are not subject to length restrictions, although special processing logic is required to handle possible instruction interruptions.

- MVST (Move String)
- SRST (Search String)
- CUSE (Compare Until Substring Equal)
- CLST (Compare Logical String).

General note: the MVST, SRST, CUSE, and CLST instructions aresimilar to the ‘long’ instructions (MVCL, CLCL) in that they can process fields longer than 256 characters and the processing can be interrupted. Whereas the interruption of the ‘long’ instructions is transparent to the application programmer (the instruction processing continues automatically), the string instructions must be explicitly reinvoked after an interruption (the condition code is set to indicate whether the processing has completed; BO = instruction interrupted before completion). Because the processing block length is at least 256 bytes, this interruption of the string instructions applies only when the fields involved are longer that 256 bytes; fields lengths not exceeding 256 bytes are guaranteed not to be interrupted.

MVST (MOVE STRING)

The MVST instruction transfers the source field that is terminated with the specified delimiter byte (contained in general register 0) to the destination field; the delimiter byte is also transferred.

Note: because the length of the destination field is not specified, the programmer must take the appropriate precautions to maintain the bounds.

Format: MVST *destination,source*

destination and *source* are each a general register.

Before processing:

- *source* – start address of the source field.
- *destination* – start address of the destination field.

Register 0 – delimiter byte is the low-order byte (the high-order three bytes must be set to X'00').

After processing:

- *BO-condition*: the instruction execution was interrupted before the end of processing. Re-invoke the instruction to continue processing.

The destination register is updated to contain the address of the last byte transferred (the delimiter byte).

The schematic processing is shown in Figure 3:

Example:

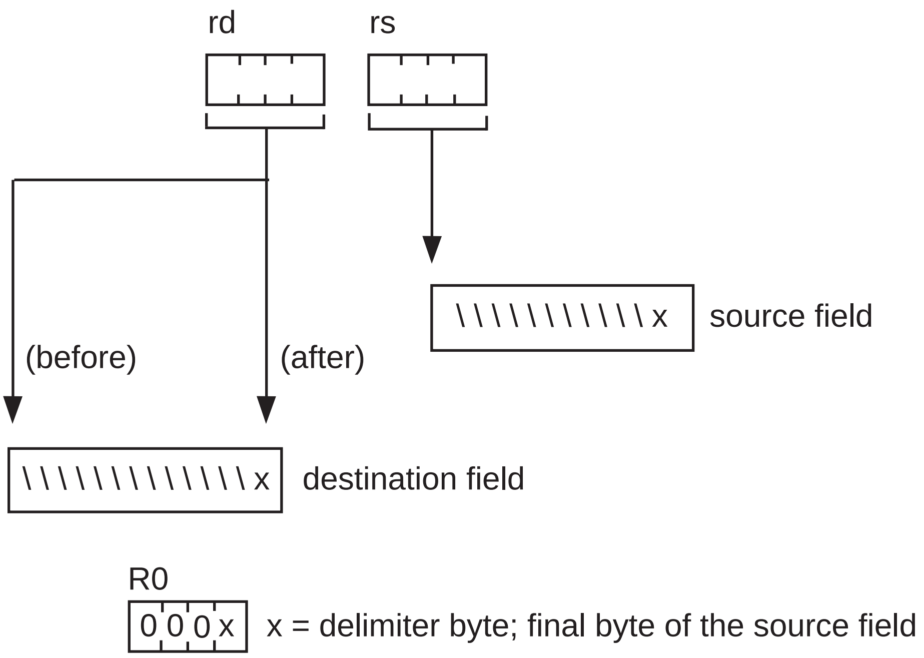


Figure 3: Schematic processing of the MVST instruction

```
         L     Ø,=X'ØØØØØØØ2'     Delimiter byte
         LA    4,INSTR            Source field
         LA    2,OUTSTR           Destination field
         MVST  2,4                Move string
         BO    *-4                Reinvoke, execution interrupted
...
INSTR    DC    C'alpha',X'Ø2',C'gamma',X'Ø2'
OUTSTR   DS    CL512
OUTSTR contains C'alpha',X'02' after execution in this example.
```

Note: the BO instruction is actually superfluous in this case because the length of the source field does not exceed 256 bytes.

SRST (Search String)

The SRST instruction searches for the specified byte (in general register 0) in a string. SRST can sometimes be used as replacement for the TRT instruction (see Example 2).

Format: SRST *stringend,stringstart*

stringstart and *stringend* are each a general register.

The schematic processing is shown in Figure 4:

Before processing:

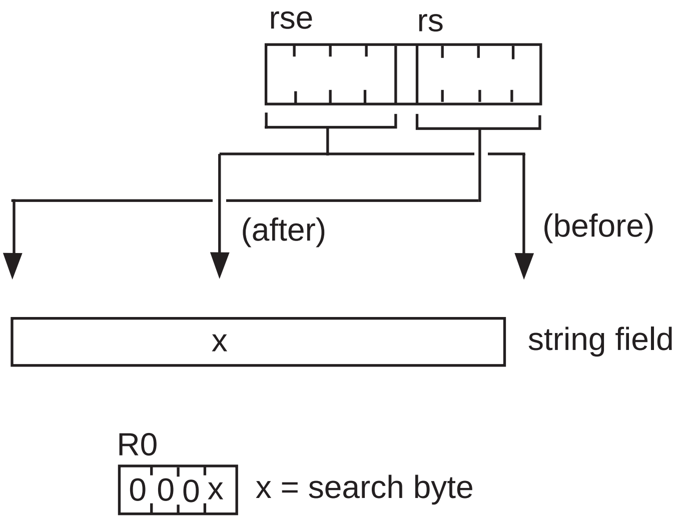


Figure 4: Schematic processing of the SRST instruction

- *stringstart* – address of the first byte of the string.
- *stringend* – address of the next byte after the end of the string.

The search byte is specified in the low-order byte of general register 0 (the high-order 3 bytes must be set to X'00').

After processing:

- *BO-condition* – the instruction execution was interrupted before the end of processing. Re-invoke the instruction to continue processing.
- *BL-condition* – the search character was found. The 'stringend'

register contains the address of the found byte.

- *BE-condition* – the search character was not found. The registers remain unchanged.

Example 1

```
...
         LA    4,INSTR            Start address of input string
         LA    3,INSTRE           End address of input string (+1)
         L     Ø,=X'ØØØØØØØ2'     Search byte
         SRST  3,4                Search string
         BO    *-4                Repeat, execution interrupted
         BL    FOUND              String found (Register 3 updated)
         BH    NOTFOUND           String not found

INSTR    DC    C'alpha',X'Ø2',C'gamma',X'Ø2'
INSTRE   EQU   *
...
```

Example 2

This example shows SRST as a replacement for the TRT instruction. For example, if an EXEC parameter used for switch settings (/A, /B, or /C) is analysed.

```
...
         L     2,Ø(1)
         LA    3,2(2)            A(parameter data)
         LH    4,Ø(2)            L(parameter data)
* Register 3: A(argument); Register 4: L(argument)
* Search for switch (/x (x = A, B, C))
         LR    6,3               Start of parameter
         LA    5,Ø(3,4)          End of parameter
         LR    14,5              Save parameter end address
         L     Ø,=XL4'ØØØØØØ61'  Delimiter = Ø...Ø/
* Processing loop
LOOP     DS    ØH
         LR    5,14              Set parameter end address
         SRST  5,6
         BH    LOOPEND           No (further) delimiter found
* Register 5: A(/switch)
         CLI   1(5),C'A'
         BNE   NOTSWA
         OI    FLAG,X'Ø1'        Set corresponding bit (Ø1) in FLAG
         B     CONTINUE
NOTSWA   CLI   1(5),C'B'
         BNE   NOTSWB
         OI    FLAG,X'Ø2'        Set corresponding bit (Ø2) in FLAG
         B     CONTINUE
NOTSWB   CLI   1(5),C'C'
         BNE   NOTSWC
         OI    FLAG,X'Ø4'          Set corresponding bit (Ø4) in FLAG
         B     CONTINUE
NOTSWC   ABEND 1                   Invalid switch value (=terminate)
CONTINUE LA    6,2(5)              Set next parameter start address (R6!)
         B     LOOP                Continue processing
LOOPEND  DS    ØH
**
FLAG     DC    X'Ø'
...
```

Note: because the length of the source field does not exceed 256 bytes no BO instruction to check for incomplete processing is required in this case.

CUSE (COMPARE UNTIL SUBSTRING EQUAL)

The CUSE instruction compares two strings for the specified number of characters that must agree (general register 0 contains the number of characters). The shorter string is right-padded with the specified pad character (in general register 1). The found string must be located at the same relative position in each of the two fields.

The comparison ends when either a matching string is found (possibly right padded) or the comparison strings are expended. The comparison may be interrupted, although not before at least 256 bytes have been processed. The address and length of the two comparison fields must each be contained in two even/odd register pairs (eg R2,R3 and R14,R15).

Format: CUSE *rega*,*regb*

rega and *regb* are each register pairs (even-odd).

Before processing:

- *rega* – address of the leftmost byte of the first string (the even register of the first even-odd register pair)
- *rega'* – length of the first string (the odd register of the first even-odd register pair)
- *regb* – address of the leftmost byte of the second string (the even register of the second even-odd register pair)

- *regb'* – length of the second string (the odd register of the second even-odd register pair).

Register 0 – length of the string that must match.

Register 1 – padding byte (low-order byte, 3 high-order bytes must be set to X'00').

The schematic processing is shown in Figure 5.

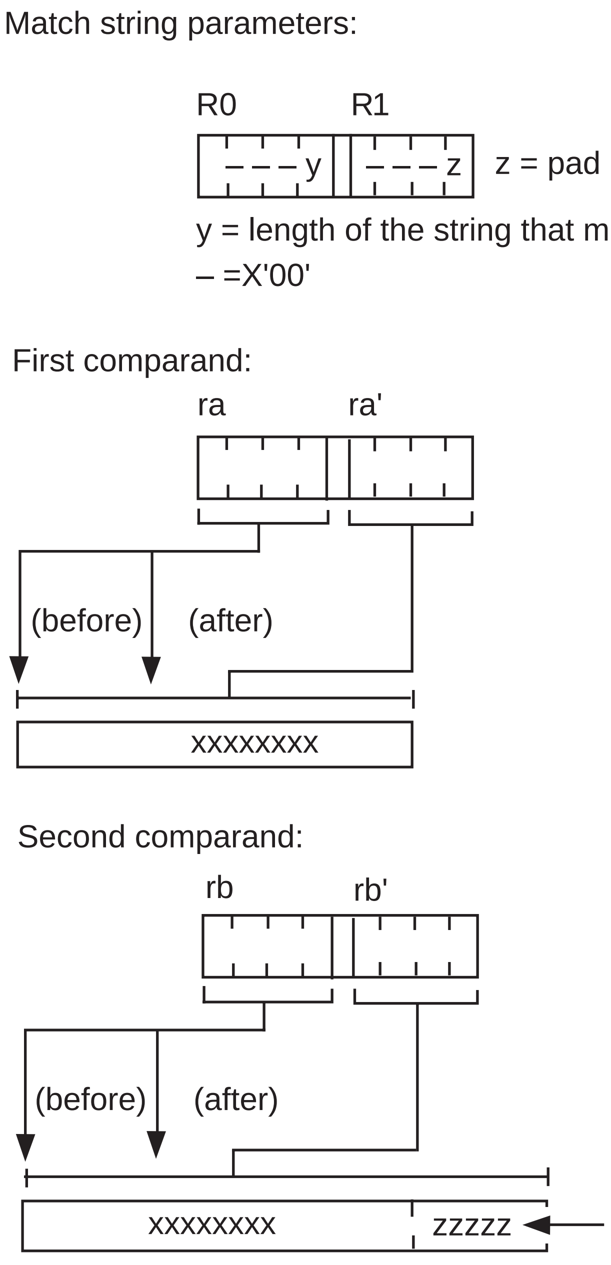


Figure 5: Schematic processing of the CUSE instruction

- *BO-condition* – the instruction execution was interrupted before a matching string was found. Re-invoke the instruction to continue.
- *BE-condition* – an explicit matching string was found.
- *BL-condition* – an implicit matching string was found (the shorter string was padded).
- *BH-condition* – no matching string was found.

The registers are updated to indicate where the processing stopped (address of each matching string, end of the string or address where the instruction was interrupted) and the number of bytes not processed. For example:

```
...
         LA    2,INSTR1  String 1
         LA    3,INSTR1L Length of string 1
         LA    4,INSTR2  String 2
         LA    5,INSTR2L Length of string 2
         LA    Ø,4       Length of the required search string
         LA    1,X'4Ø'   Pad character (here blank)
         CUSE  2,4       Compare string
         BO    *-4       Reinvoke, execution interrupted before
completion
         BL    MATCHPAD  Matching string found (one string was padded)
         BH    NOMATCH   No matching string found
         BE    MATCH     Matching string found (no padding required)
INSTR1   DC    C'123123betagamma'
INSTR1L  EQU   *-INSTR1
INSTR2   DC    C'123412betagammadelta'
INSTR2L  EQU   *-INSTR2
...
```

Note: *beta* was found as the matching string in this example.

CLST (COMPARE LOGICAL STRING)

The CLST instruction compares two strings for equality. General register 0 contains the string delimiter character. The comparison ends when either the delimiter character is found or a difference is detected between the two strings. When the detection of the delimiter character terminates the comparison, the string that contains the first delimiter character found is considered to be the smaller field. If the comparison indicates inequality, the registers are updated with the addresses of the last byte compared in each string. The comparison may be interrupted, although not before at least 256 bytes have been processed.

Format: CLST *string1,string2*

string1 and *string2* are registers, each of which contains the address of a comparison string.

- *string1* – address of the leftmost byte of the first string.
- *string2* – address of the leftmost byte of the second string.

After processing:

- *BO-condition* – the instruction execution was interrupted before processing completed. Re-invoke the instruction to continue.
- *BE-condition* – both strings match.
- *BL-condition* – the first string matches low. Either the first string is logically smaller than the second string or the two strings are identical up to the specified delimiter found in the first string.
- *BH-condition* – the first string matches high. Either the second string is logically smaller than the first string or the two strings are identical up to the specified delimiter is found in the second.

Registers are updated to show where processing stopped. For example:

```
...  .
         L     Ø,DLM     Delimiter
         LA    1,FLD1    Start address of string 1
         LA    2,FLD2    Start address of string 2
         CLST  1,2       Compare strings
         BO    *-4       Reinvoke, execution interrupted before completion
         BL    LOW       String 1 smaller
         BH    HIGH      String 1 larger
         BE    EQUAL     Both strings equal
LOW      NOPR  Ø         Tag for processing first string smaller
HIGH     NOPR  Ø         Tag for processing first string larger
EQUAL    NOPR  Ø         Tag for processing both strings equal
**
DLM      DC    XL3'Ø',C';'
FLD1     DC    C'alphabetagamma',C';'
FLD2     DC    C'alphabet',C';'
...
```

Note 1: this example yields a string 2 high.

Note 2: the labels LOW, HIGH, and EQUAL are supplied here to provide branch addresses for the result of the comparison. Obviously the appropriate application processing must be added.

Systems Programmer (Germany) © Xephon 2001

Synchronization of catalogs and SMS DASD volumes

THE PROBLEM

Catalog entries can become unsynchronized, so that dataset information is different in the BCS, VVDS, and VTOC. These differences may make a dataset inaccessible or otherwise unusable. This can happen after disaster recovery procedures because of DASD failure or after improper administration action. Likely errors include:

- There is a catalog entry for the dataset, but it does not exist on DASD volume.
- Dataset is on a DASD volume, but there is no catalog entry for it.
- There are multiple datasets with the same name on different DASD volumes.

A SOLUTION

The DIAG procedure can help you to find a discrepancy between BCS and VVDS information on SMS disks. Without this procedure you have to compare each catalog with each SMS DASD volume and *vice versa*. It takes a lot of submission and manual work.

The DIAG procedure does all the comparisons in one submission and that is why the duration time is fairly large. However, in our opinion this not a problem because you do not need to use this procedure frequently, only after disaster recovery on DASD volumes and periodically (for example twice a year) for checking purposes. If you rerun the procedure, checking will be performed only on objects that have been found to have a problem (rc>=8) during the first run. This improvement significantly shortens the execution time of the procedure. The procedure gives the following scenario for synchronizing:

- Delete the catalog entry with a DELETE dsname NOSCRATCH statement.

  - Check if the dataset is important and recover it from the back-up copy

- Reconnect the dataset to a catalog for NONVSAM and CLUSTER. We recommend DELETE VVR and rebuilding for alternate indexes.

- Check which dataset is valid and which ones are obsolete and can be deleted. If two or more datasets have the same name only one can be in the catalog. This means that we have to check the data in the first dataset and then rename or uncatalog it. After that we can catalog the second dataset to inspect the data inside it. The action that follows after inspection is as follows:

  - Delete the obsolete dataset with DELETE dsname (NVR or VVR).

  - Catalog the valid dataset with RECONNECT (if it is not already in catalog).

The DIAG procedure will generate statements for the actions previously described in the dataset userid.DIAGNOSE.##REPAIR.LIST. The system programmer or storage administrator must check and edit these statements before implementing them. Statements that pass checking and editing will be implemented by job REPAIRCD. The prerequisites are:

- SMS active.

- RACF permission:

  - Read authority for CONSOLE profile in TSOAUTH class.

  - Read authority for the STGADMIN.IDC.DIAGNOSE. CATALOG profile in FACILITY class.

  - Read authority for STGADMIN.IDC.DIAGNOSE.VVDS profile in FACILITY class.

Note:if you do not activate these profiles in RACF, it is necessary to authorize the DIAGNOSE function. To achieve this, add the AUTHCMD statement in the IKJTSO*xx* member of the parmlib library. With the PARMLIB UPDATE(XX) TSO command you can dynamically activate it.

DIAG

```
/****************************** REXX ********************************/
/* Diagnose all user catalogs and all SMS DASD and generate         */
/* statements for reparation.                                       */
/*                                                                  */
/* %DIAG argument1  [argument2]                                     */
/*                                                                  */
/* Each argument can have one of the following values:              */
/* DIAGCAT   - check all user catalogs and compare them with        */
/*             DASD volumes                                         */
/* DIAGVVDS  - check all VVDS and compare them with all BCS         */
/********************************************************************/
/*  Trace ?R */
 ARG functions
 userid=SYSVAR(SYSUID)
 prefix=SYSVAR(SYSPREF)
 "PROFILE NOPREFIX"
 DSN_ALL_USERCAT  =userid||'.DIAGNOSE.#USERCAT.LIST'
 DSN_ALL_SMS_DASD =userid||'.DIAGNOSE.#SMSDASD.LIST'
 DSN_DIAG_LOG     =userid||'.DIAGNOSE.#DIAGLOG.LIST'
 DSN_DIAG_ERR     =userid||'.DIAGNOSE.#DIAGERR.LIST' /* Only Error */
 DSN_REPAIR       =userid||'.DIAGNOSE.##REPAIR.LIST'
/*---------------------------------------------------------------*/
/* Main procedure                                                */
/*---------------------------------------------------------------*/
 If SYSDSN(DSN_ALL_USERCAT) <> 'OK'
 Then Do
      Call Alloc_DS   USERCAT  DSN_ALL_USERCAT  NEW 123 V
      Call List_Usercat
      End
 If SYSDSN(DSN_ALL_SMS_DASD) <> 'OK'
 Then Do
      Call Alloc_DS   SMSDASD  DSN_ALL_SMS_DASD NEW 123 V
      Call List_Sms_Dasd
      End
 Call Alloc_DS   USERCAT  DSN_ALL_USERCAT  SHR
 Call Alloc_DS   SMSDASD  DSN_ALL_SMS_DASD SHR
 Call Alloc_DS   DIAGLOG  DSN_DIAG_LOG     NEW 133 V
 Call Alloc_DS   DIAGERR  DSN_DIAG_ERR     NEW 133 V
 Call Alloc_DS   REPAIR   DSN_REPAIR       NEW 80  F
 PARSE UPPER VAR functions function.1 function.2
 Do if=1 To 2
    Select
      When function.if = 'DIAGCAT'
        Then Call Diagnose_All_Usercatalog
      When function.if = 'DIAGVVDS'
        Then Call Diagnose_All_VVDS
      Otherwise
      End /* select */
 End /* Do */
 If prefix <> ''
 Then "PROFILE PREFIX("prefix")"
Return
/*-----------------------------------------------------------------*/
/* Procedure LIST ALL USERCATALOG                                  */
/*-----------------------------------------------------------------*/
List_Usercat: Procedure
user_cat.Ø=Ø
t=OUTTRAP('dsnc.',,NOCONCAT)
"LISTCAT USERCATALOG"
rrc=RC
t=OUTTRAP('OFF')
If rrc > Ø
Then Do
     Do i = 1 to dsnc.Ø
        Say '>>>' dsnc.i
     End
     EXIT rrc
     End
Do i = 1 to dsnc.Ø
   Parse var dsnc.i keyword sign user_cat.i
End
user_cat.Ø=dsnc.Ø
Say 'There is 'user_cat.Ø' user catalogs'
"EXECIO * DISKW usercat (STEM user_cat. FINIS)"
Return
/*-----------------------------------------------------------------*/
/* Get names of all user catalogs                                  */
/*-----------------------------------------------------------------*/
Get_User_catalog: Procedure Expose user_cat. user_cat_rc.
user_cat.Ø=Ø
" EXECIO * DISKR usercat (STEM usercat. FINIS) "
Do i=1 to usercat.Ø
   Parse Var usercat.i user_cat.i rrc user_cat_rc.i
End
user_cat.Ø = usercat.Ø
Drop usercat.
Return
/*-----------------------------------------------------------------*/
/* Write names of all user catalogs                                */
/*-----------------------------------------------------------------*/
Put_User_catalog: Procedure Expose user_cat. user_cat_rc.
Do i=1 to user_cat.Ø
   usercat.i=LEFT(user_cat.i,45)||' RC= '||user_cat_rc.i
End
usercat.Ø=user_cat.Ø
" EXECIO * DISKW usercat (STEM usercat. FINIS) "
Return
/*-----------------------------------------------------------------*/
/* Diagnose user catalogs                                          */
/*-----------------------------------------------------------------*/
Diagnose_All_Usercatalog: Procedure
 user_cat.Ø=Ø
 user_cat_rc.0=0
 Call Get_User_Catalog
 sms_unit.0 = k
 sms_dasd.0 = k
 Call Get_SMS_Dasd
 If user_cat.0 > 0
 Then Do
 Say 'We compare 'user_cat.0' user catalogs with 'sms_dasd.0' SMS dasd'
      msg.1=Comment('C',' Repair of User Catalogs ','=')
      " EXECIO 1 DISKW repair  (STEM msg.) "
      diag.0=0
      Do i = 1 to user_cat.0
         If user_cat_rc.i <> 0
         Then Do
              rrc=diagnose_usercat(user_cat.i)
              Say '===> DIAGNOSE icfcatalog ' i user_cat.i    'Rc=' rrc
              If rrc <= 4
              Then Do j = 1 to sms_dasd.0
                      rcc=diagnose_usercat_dasd(user_cat.i, sms_dasd.j)
                      Say '  ---> Compare ' i user_cat.i,
                          ' with ' j sms_dasd.j    'Rc=' rcc
                      If rcc = 4
                      Then rcc=0
                      rrc=MAX(rrc,rcc)
                   End
              Else Do
                   Say '>>> We do not compare with dasd'
                   Say '>>> because of catalog errors'
                   End
              user_cat_rc.i=rrc
              End
         Else Say '>>> Skip diagnose 'i user_cat.i,
                  ' OLD Rc=' user_cat_rc.i
      End
      Call Put_User_Catalog
      End
 Else Say ">>> No usercatalog defined in master catalog"
 return
/*-----------------------------------------------------------------*/
/* Diagnose user catalogs                                          */
/*-----------------------------------------------------------------*/
diagnose_usercat: Procedure Expose sms_dasd.
Arg User_Catalog
t=OUTTRAP('diag.',,NOCONCAT)
"DIAGNOSE ICFCATALOG IDS ("''''User_Catalog''''") NODUMP"
rrc=RC
t=OUTTRAP('OFF')
msg.1= COPIES('-',80)
msg.2= 'DIAGNOSE ICFCATALOG IDS ('user_catalog')  RC='||rrc
msg.0=2
Call Write_Diag_Messages 'diaglog'
```

```
If rrc > 0
Then Do
     Call Write_Diag_Messages 'diagerr'
     Call Check_diagnose_usercat(User_catalog)
     End
If rrc = 4   /* VVDS referenced catalogs were not encountered */
Then Do
     Do j=1 TO diag.0
        If SUBSTR(diag.j,1,9) = 'IDC11374I'
        Then Leave
     End
     Do j=j+1 TO diag.0
        Parse Var diag.j volume rest
        If volume = 'IDC0014I'
        Then Leave
        Else Call Repair_cat   User_Catalog   volume
     End
     End
drop  msg. diag.
Return rrc
/*-----------------------------------------------------------------*/
/* Repair user catalog                                             */
/*-----------------------------------------------------------------*/
Repair_cat: Procedure Expose sms_dasd.
Arg User_Catalog Volume
vvds='SYS1.VVDS.V'||Volume
Do k=1 TO sms_dasd.0
   If sms_dasd.k = volume
   Then Leave
End
If k > sms_dasd.0
Then Do
     t=Check_dasd(Volume)
     If t = 0
     Then recat='YES'
     Else recat='NO'
        End
Else recat='YES'
If recat = 'YES'
Then Do
     "DEFINE CLUSTER(NAME("''''vvds''''")",
     "       VOLUMES("Volume") NONINDEXED RECATALOG) ",
     "       CAT("''''User_Catalog''''")"
     End
Else Do
     msg.1=Comment('L','-','-')
     msg.2=Comment('L',Volume,
     ' does not exist for dataset in cat 'User_Catalog' |||','-')
     " EXECIO 2 DISKW repair  (STEM msg.) "
     Call Del_cat_entry User_Catalog Volume
     End
Drop msg.
```

```
End
Return
/*-----------------------------------------------------------------*/
/* Diagnose user catalogs compared with DASD                       */
/*-----------------------------------------------------------------*/
diagnose_usercat_dasd: Procedure
Arg user_catalog, dasd
vvds='SYS1.VVDS.V'||dasd
t=OUTTRAP('diag.',,NOCONCAT)
"DIAGNOSE ICFCATALOG IDS ("''''user_catalog''''")",
         " COMPAREDS("''''vvds''''") NODUMP"
rrc=RC
t=OUTTRAP('OFF')
msg.1= COPIES('-',8Ø)
msg.2= 'DIAGNOSE ICFCATALOG IDS ('user_catalog') -'
msg.3= '         COMPAREDS('vvds') NODUMP      RC='||rrc
msg.Ø=3
Call Write_Diag_Messages 'diaglog'
If rrc > 4
Then Do
     Call Write_Diag_Messages 'diagerr'
     Call Check_diagnose_usercat(User_catalog)
     End
drop msg. diag.
Return rrc
/*-----------------------------------------------------------------*/
/* Check does DASD exist?                                          */
/*-----------------------------------------------------------------*/
Check_Dasd: Procedure
Arg Volume
cmdresp.Ø=Ø
cmd='D U,VOL='Volume
rrc=mvs_command(cmd)
If rrc = Ø
Then Do
     /* Volume does not exist If response is:                    */
     /* IEE455I UNIT STATUS NO DEVICES WITH REQUESTED ATTRIBUTES  */
     If SUBSTR(cmdresp.1,2,7) = 'IEE455I' | cmdresp.Ø = 1
     Then rrc=16
     End
Return rrc
/*-----------------------------------------------------------------*/
/* Delete dataset entry from catalog, volume does not exist        */
/*-----------------------------------------------------------------*/
Del_cat_entry: Procedure
Arg User_Catalog Volume
t=OUTTRAP('listc.',,NOCONCAT)
"LISTC CAT(" ''''User_Catalog'''' ") VOLUME"
x=RC
t=OUTTRAP('OFF')
If x = Ø
```

```
Then Do k=1 to listc.Ø
        If SUBSTR(listc.k,1,3) <> '   '
        Then Parse var listc.k type cr ds_name
        If INDEX(listc.k,'VOLSER-') > Ø & INDEX(listc.k,Volume) > Ø
        Then Do
             msg.1=Comment('L','Catalog entry 'type ds_name,'-')
             msg.2=Comment('L','points to ' Volume,'-')
             msg.3='   DELETE ' ''''ds_name'''' type ' NOSCRATCH -'
             msg.4='      CAT(' ''''User_catalog'''' ')'
             "EXECIO 4 DISKW repair  (STEM msg.) "
             End
     End
Drop listc.
Return
/*-----------------------------------------------------------------*/
/* Check output from diagnose user catalog                         */
/*-----------------------------------------------------------------*/
 Check_diagnose_usercat: Procedure Expose diag.
 Arg user_catalog
 /* find message */
 /* IDC21363I THE FOLLOWING ENTRIES HAD ERRORS:   */
 Do j=1 to diag.Ø
    If SUBSTR(diag.j,1,9) = 'IDC21363I'
    Then leave
 End
 Do j=j+1 to diag.Ø
    i = INDEX(diag.j,' REASON CODE')
    If i > Ø
    Then Do
         Parse VAR diag.j dsname type sign reason code ncode
         cat_type = Catalog_Entry_type(type)
         msg.1=Comment('C',' Error in Catalog ','-')
         msg.2=Comment('L',dsname cat_type ' reason code='ncode,'-')
         msg.3='   DELETE ' ''''dsname'''' cat_type ' NOSCRATCH -'
         msg.4='      CAT(' ''''User_catalog'''' ')'
         "EXECIO 4 DISKW repair  (STEM msg.) "
         End
 End
Return
/*-----------------------------------------------------------------*/
/* Catalog Entry Type                                              */
/*-----------------------------------------------------------------*/
 Catalog_Entry_type:Procedure
 Arg record_type
 Select
   When record_type = '(A)'
    Then cat_type = 'NONVSAM'
   When record_type = '(B)'
    Then cat_type = 'GDG'
   When record_type = '(C)'
    Then cat_type = 'CLUSTER'
   When record_type = '(G)'
    Then cat_type = 'AIX'
   When record_type = '(L)'
    Then cat_type = 'LIBRARYENTRY'
   When record_type = '(R)'
    Then cat_type = 'PATH'
   When record_type = '(T)'
    Then cat_type = 'TRUENAME'
   When record_type = '(U)'
    Then cat_type = 'USERCATALOG'
   When record_type = '(W)'
    Then cat_type = 'VOLUMEENTRY'
   When record_type = '(X)'
    Then cat_type = 'ALIAS'
   Otherwise cat_type = ''
End /* select */
Return cat_type
/*-----------------------------------------------------------------*/
/* Procedure LIST ALL SMS DASD                                     */
/*-----------------------------------------------------------------*/
List_Sms_Dasd: Procedure
cmdresp.Ø=Ø
cmd='D SMS,SG(ALL),LISTVOL'
rrc=mvs_command(cmd)
If rrc > Ø
Then Exit rrc
Do i=1 to cmdresp.Ø
   If substr(cmdresp.i,2,6) = 'VOLUME'
   Then leave
End
k=Ø
Do i=i+1 to cmdresp.Ø
   If substr(cmdresp.i,27,1) = '+'
   Then Do
        k=k+1       /* Extract volume name + unit */
        Parse var cmdresp.i volume unit rest
        unit=RIGHT(unit,3)
        smsdasd.k = volume||' '||unit
        End
End
smsdasd.Ø = k
Say 'There is 'smsdasd.Ø' SMS dasd'
"EXECIO * DISKW smsdasd (STEM smsdasd. FINIS) "
Return rrc
/*-----------------------------------------------------------------*/
/* Get name of all SMS DASD                                        */
/*-----------------------------------------------------------------*/
Get_Sms_Dasd: Procedure Expose sms_dasd. sms_unit. dasd_rc.
smsdasd.Ø=Ø
" EXECIO * DISKR smsdasd (STEM smsdasd. FINIS) "
Do i=1 to smsdasd.Ø
```

```
   Parse Var smsdasd.i sms_dasd.i sms_unit.i rrc dasd_rc.i
End
sms_dasd.Ø = smsdasd.Ø
sms_unit.Ø = smsdasd.Ø
dasd_rc.Ø  = smsdasd.Ø
Drop smsdasd.
Return
/*------------------------------------------------------------------*/
/* Put name of all SMS DASD                                         */
/*------------------------------------------------------------------*/
Put_Sms_Dasd: Procedure Expose sms_dasd. sms_unit. dasd_rc.
Do i=1 to sms_dasd.Ø
   smsdasd.i=sms_dasd.i||' '||sms_unit.i||' RC= '||dasd_rc.i
End
smsdasd.Ø=sms_dasd.Ø
" EXECIO * DISKW smsdasd (STEM smsdasd. FINIS) "
Return
/*------------------------------------------------------------------*/
/* Issue MVS command                                                */
/*------------------------------------------------------------------*/
Mvs_command: Procedure Expose cmdresp.
 Arg cmd
 sd=SYSVAR("SOLDISP")
 usd=SYSVAR("UNSDISP")
 wait_time = 3Ø
 userid = SYSVAR(SYSUID)
 cart = userid||TIME()          /* create unique CART value */
 "CONSPROF SOLDISPLAY(NO) UNSOLDISPLAY(NO) SOLNUM(9999) UNSOLNUM(Ø)"
 If rc <> Ø
 Then Do
    Say '*** Userid' userid 'needs CONSOLE authority'
    Exit RC
 End
 "CONSOLE ACTIVATE NAME(DIAG)"                /* activate console */
 rrc = RC
 If rrc <> Ø
 Then Do
      Say 'CONSOLE ACTIVATE RC=' rrc
      "CONSOLE DEACTIVATE"
      Exit rrc
      End
 "CONSOLE SYSCMD("cmd") CART("cart")"
 rrc=RC
 grc = GETMSG('cmdresp.','SOL',cart,,wait_time)   * get response */
 rrc=MAX(rrc,grc)
 "CONSOLE DEACTIVATE"                   /* finished with console */
 If sd = 'YES' Then
    "CONSPROF SOLDISP("SD")"
 If usd = 'YES' Then
    "CONSPROF UNSOLDISP("USD")"
 If grc > Ø                 /* GETMSG was NOT OK */
 Then Do
      Say ">>> GETMSG error retrieving message.  RC =" get_rc
      End
 return rrc
/*------------------------------------------------------------------*/
/* Diagnose VVDS on all SMS DASD                                    */
/*------------------------------------------------------------------*/
Diagnose_All_VVDS: Procedure
 sms_dasd.0=0
 sms_unit.0=0
 dasd_rc.0=0
 Call Get_SMS_Dasd
 user_cat.0=0
 Call Get_User_Catalog
 If sms_dasd.0 > 0
 Then Do
 Say 'We compare 'sms_dasd.0' SMS dasd with 'user_cat.0' user catalogs'
      msg.1=Comment('C',' Repair SMS DASD ','=')
      " EXECIO 1 DISKW repair  (STEM msg.) "
      Do i=1 to sms_dasd.0
         If dasd_rc.i <> 0
         Then Do
              rrc=diagnose_vvds(sms_dasd.i,sms_unit.i)
              Say '===> DIAGNOSE VVDS ON ' i sms_dasd.i,
                   sms_unit.i 'RC='rc
              Do j = 1 to user_cat.0
                 rcd=diagnose_vvds_usercat(sms_dasd.i, sms_unit.i, ,
                                           user_cat.j)
                 Say '  ---> Compare ' i sms_dasd.i,
                     ' with ' j user_cat.j 'Rc=' rcd

                 rrc=MAX(rrc,rcd)
              End
              If rrc=4
              Then rrc=0
              dasd_rc.i=rrc
              End
         Else Say '>>> Skip diagnose ' i sms_dasd.i 'OLD Rc=' dasd_rc.i
      End
      Call Put_Sms_Dasd
      End
 Else Say ">>> No SMS DASD"
 Return

/*------------------------------------------------------------------*/
/* Diagnose VVDS                                                    */
/*------------------------------------------------------------------*/
diagnose_vvds: Procedure
Arg Volume, Unit
vvds='SYS1.VVDS.V'||Volume
"ALLOC F(DIAGDD) DS("vvds") SHR VOLUME("Volume") UNIT("Unit")"
```

```
t=OUTTRAP('diag.',,NOCONCAT)
"DIAGNOSE VVDS INFILE(DIAGDD) NODUMP"
rrc=RC
t=OUTTRAP('OFF')
"FREE  F(DIAGDD)"
msg.1= COPIES('-',8Ø)
msg.2= 'DIAGNOSE VVDS IDS('vvds') NODUMP                RC='||rrc
msg.Ø=2
Call Write_Diag_Messages 'diaglog'
If rrc > 4
Then Do
     Call Write_Diag_Messages 'diagerr'
     Call Check_diagnose_vvds Volume Unit
     End
drop diag.   msg.
Return rrc
/*-----------------------------------------------------------------*/
/* Diagnose VVDS                                                   */
/*-----------------------------------------------------------------*/
diagnose_vvds_usercat: Procedure
Arg Volume, Unit, User_Catalog
vvds='SYS1.VVDS.V'||Volume
"ALLOC F(DIAGDD) DS("vvds") SHR VOLUME("Volume") UNIT("Unit")"
t=OUTTRAP('diag.',,NOCONCAT)
"DIAGNOSE VVDS INFILE(DIAGDD) NODUMP COMPAREDS("User_Catalog")"
rrc=RC
t=OUTTRAP('OFF')
"FREE  F(DIAGDD)"
msg.1= COPIES('-',8Ø)
msg.2= 'DIAGNOSE VVDS IDS('vvds') NODUMP '
msg.3= '         COMPAREDS('User_Catalog')              RC='||rrc
msg.Ø=3
Call Write_Diag_Messages 'diaglog'
If rrc > 4
Then Do
     Call Write_Diag_Messages 'diagerr'
     Call Check_diagnose_vvds Volume Unit
     End
drop diag.   msg.
Return rrc
/*-----------------------------------------------------------------*/
/* Check output from diagnose VVDS                                 */
/*-----------------------------------------------------------------*/
 Check_diagnose_vvds: Procedure Expose diag.
 Arg Volume Unit
 vvds='SYS1.VVDS.V'||Volume
 /* find message */
 /* IDC21363I THE FOLLOWING ENTRIES HAD ERRORS:   */
 Do j=1 to diag.Ø
    If SUBSTR(diag.j,1,9) = 'IDC21363I'
    Then leave
```

```
End
Do j=j+1 to diag.Ø
   i = index(diag.j,' REASON CODE')
   If i > Ø
   Then Do
        Parse VAR diag.j entry_name type sign reason code ncode
        Volume_cat=Cet_Vol_From_catalog(entry_name)
        dsname = entry_name
        If type = '(N)'
        Then Do
             vvds_type=' NVR '
             Cat_type ='NONVSAM'
             devt='DEVT(SYSDA)'
             End
        Else Do
             vvds_type=' VVR '
             Cat_type ='CLUSTER'
             devt=''
             i=INDEX(entry_name,'.INDEX') - 1
             If i > Ø
             Then dsname = LEFT(entry_name,i)
             i=INDEX(entry_name,'.DATA') - 1
             If i > Ø
             Then dsname = LEFT(entry_name,i)
             End
        If Volume_cat =''
        Then Do
             If type <> (N) & INDEX(entry_name,'.INDEX') > Ø
             Then return
             msg.1=Comment('C','Dataset not in catalog','-')
             msg.2=' DEFINE 'Cat_type'(NAME(' ''''dsname'''' ') -'
             msg.3='        VOLUMES('Volume') 'devt' RECATALOG)'
             msg.4=Comment('C','or if dataset not necessary','-')
             msg.5=Comment('L','ALLOC F(DASD) DS(' ''''vvds'''',
                   ') UNIT(SYSDA) VOLUME('Volume') SHR ',' ')
             msg.6=Comment('L','DELETE' ''''dsname''''vvds_type,
                           'FILE(DASD)',' ')
             msg.7=Comment('L','FREE F(DASD)',' ')
             "EXECIO 7 DISKW repair  (STEM msg.) "
             End
        Else Do
             msg.1=Comment('C','Duplicate','-')
             msg.2=Comment('L','Catalog entry 'type dsname,'-')
             msg.3=Comment('L','points to ' Volume_cat,'-')
             msg.4=' ALLOC F(DASD) DS(' ''''vvds'''',
                   ') UNIT(SYSDA) VOLUME('Volume') SHR'
             msg.5=' DELETE ' ''''dsname'''' vvds_type' FILE(DASD)'
             msg.6=' FREE F(DASD)'
             msg.7=Comment('C','or if wrong dataset in cat','-')
             msg.8=Comment('L','DELETE ' ''''dsname'''',' ')
             msg.9=Comment('L','DEFINE 'Cat_type'(NAME(',
                            ''''dsname'''' ') -',' ')
              msg.1Ø=Comment('L','    VOLUMES('Volume') 'devt,
                             ' RECATALOG)',' ')
              "EXECIO 1Ø DISKW repair (STEM msg.) "
              End
         Drop msg.
         End
 End
Return
/*------------------------------------------------------------------*/
/* Get dataset volume from catalog                                  */
/*------------------------------------------------------------------*/
Cet_Vol_From_catalog: Procedure
Arg Ds_name
Volume_cat=''
t=OUTTRAP('listc.',,NOCONCAT)
"LISTC ENT("''''Ds_name''''") VOLUME"
x=RC
t=OUTTRAP('OFF')
If x = Ø
Then Do k=1 to listc.Ø
        If index(listc.k,'VOLSER-') > Ø
        Then Do
             Volume_cat=SUBSTR(listc.k,26,6)
             leave
             End
     End
Drop listc.
Return Volume_cat
/*------------------------------------------------------------------*/
/* Enclose message in comment                                       */
/*------------------------------------------------------------------*/
Comment: Procedure
Arg Format,Message,Padc
Select
  When  Format = 'C'
    Then  msg='/* 'CENTER(Message,66,Padc)' */'
  When  Format = 'L'
    Then  msg='/* 'LEFT(Message,66,Padc)' */'
End /* Select */
Return msg
/*------------------------------------------------------------------*/
/* Write diagnose messages to output file                           */
/*------------------------------------------------------------------*/
Write_Diag_Messages: Procedure Expose msg. diag.
Arg DiagFile
" EXECIO * DISKW "DiagFile" (STEM msg.) "
" EXECIO * DISKW "DiagFile" (STEM diag.) "
Return
/*------------------------------------------------------------------*/
/* Alloc dataset                                                    */
/*------------------------------------------------------------------*/
```

```
 Alloc_DS: Procedure
 Arg DD_name DS_name Disp Length_rec Rec_fm
 If Disp='NEW' & SYSDSN(''''Ds_name'''') = 'OK'
 Then Disp='SHR'
 msgstat=MSG("OFF")    /* Inhibit the display of TSO/E information */
                       /* messages */
 "FREE F("DD_name")"
 t=MSG(msgstat)        /* Returns the pervious status of message */.
 If Disp = 'NEW'
 Then "ALLOC F("DD_name") DA("''''DS_name''''") "Disp" CATALOG",
      " SPACE(50,50) LRECL("Length_rec") RECFM("Rec_fm" B)",
      " BLKSIZE(0) RELEASE"
 Else "ALLOC F("DD_name") DA("''''DS_name''''") "Disp" REUSE"
 Return
```

The job for executing DIAG is shown below:

```
//useridD  JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(0,0),NOTIFY=&SYSUID
//DIAGNOSE  EXEC PGM=IKJEFT01,DYNAMNBR=50,
//          REGION=4M
//SYSPROC   DD  DSN=userid.USER.CLIST,DISP=SHR
//SYSTSPRT  DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
//SYSTSIN   DD  *
     %DIAG DIAGCAT DIAGVVDS
/*
```

The job for executing generated statements for reparation is shown below:

```
//REPAIRCD JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(0,0),NOTIFY=&SYSUID
//REPAIRCD  EXEC PGM=IKJEFT01,DYNAMNBR=50,
//          REGION=4M
//SYSTSPRT  DD  SYSOUT=*
//SYSPRINT  DD  SYSOUT=*
//SYSTSIN   DD  DSN=userid.DIAGNOSE.##REPAIR.LIST,DISP=SHR
//
```

EXAMPLES

If errors exist in any catalog, the DIAG procedure puts the following report in dataset userid.DIAGNOSE.#DIAGERR.LIST:

```
------------------------------------------------------------------------
DIAGNOSE ICFCATALOG IDS (CATALOG.MVSICFM.VOS2CAT)  RC=4
IDC11374I THESE ADDITIONAL CATALOG REFERENCED VOLUMES WERE ENCOUNTERED:
  DSN017
  OS3RES
IDC0014I LASTCC=4
------------------------------------------------------------------------
DIAGNOSE ICFCATALOG IDS (CATALOG.USER) -
```

```
         COMPAREDS(SYS1.VVDS.VPSTEST) NODUMP      RC=8
IDC21364I ERROR DETECTED BY DIAGNOSE:
  ICFCAT ENTRY: IBMUSER.TEST (A)
  RECORD: IBMUSER.TEST /ØØ
  OFFSET: X'ØØ5D'
  REASON: 51 - VVDS ENTRY NOT FOUND. SCAN VVDS FAILED.
IDC21363I THE FOLLOWING ENTRIES HAD ERRORS:
  IBMUSER.TEST (A) - REASON CODE: 51
IDCØØ14I LASTCC=8
```

The DIAG procedure generates the following statements for reparation, which are based on the previous report in the dataset userid.DIAGNOSE.##REPAIR.LIST:

```
/* ==================== REPAIR OF USER CATALOGS ==================== */
/* ----------------------------------------------------------------- */
/* DSNØ17  DOES NOT EXIST FOR DATASET IN CAT CATALOG.MVSICFM.VOS2CAT */
/* CATALOG ENTRY CLUSTER CICS41.CSD41Ø.DFHCSD.BKP------------------- */
/* POINTS TO  DSNØ17------------------------------------------------ */
   DELETE CICS41.CSD41Ø.DFHCSD.BKP NONSCRATCH -
      CAT(CATALOG.MVSICFM.VOS2CAT)
    . . .
/* ----------------------------------------------------------------- */
/* OS3RES DOES NOT EXIST FOR DATASET IN CAT CATALOG.MVSICFM.VOS2CAT  */
/* CATALOG ENTRY CLUSTER SYS1.APPCSI-------------------------------- */
/* POINTS TO  OS3RES------------------------------------------------ */
   DELETE SYS1.APPCSI NONSCRATCH -
      CAT(CATALOG.MVSICFM.VOS2CAT)
/* CATALOG ENTRY CLUSTER SYS1.APPCSI-------------------------------- */
/* POINTS TO  OS3RES------------------------------------------------ */
   DELETE SYS1.APPCSI NONSCRATCH -
      CAT(CATALOG.MVSICFM.VOS2CAT)
    . . .
/* ----------------------- ERROR IN CATALOG ------------------------ */
/* IBMUSER.TEST REASON CODE=51-------------------------------------- */
   DELETE IBMUSER.TEST NONSCRATCH -
      CAT(CATALOG.PROBA)
/* ======================== REPAIR SMS DASD ========================= */
/* ----------------------DATASET NOT IN CATALOG---------------------- */
 DEFINE NONVSAM(NAME( 'IBMUSER.TEST1T' ) -
        VOLUMES(PSTEST) RECATALOG)
/* --------------------OR IF DATASET NOT NECESSARY------------------- */
/* ALLOC F(DASD) DS( 'SYS1.VVDS.VPSTEST' ) UNIT(SYSDA) VOLUME(PSTEST) */
/* DELETE 'IBMUSER.TEST1T' NVR FILE(DASD)                             */
/* FREE F(DASD)                                                       */
```

Emina Spasic and Dragan Nikolic
Systems Programmers
Postal Savings Bank (Yugoslavia)

Deployment options for mainframe Linux

INTRODUCTION

By bringing the Linux operating system to the mainframe IBM has given users the chance to exploit the availability, performance and serviceability of the System/390 and z900 architectures with the momentum, speed, and quality of the Open Source development world. Linux brings new applications and potentially vast opportunities. With research suggesting that a third of mainframe sites intend to exploit Linux in the mid-term future, this article provides a detailed overview of the pros and cons on the various deployment methods.

Linux for S/390 and zSeries can operate in four configurations: native, LPAR, using the Virtual Image Facility for Linux, and in guest mode on VM/ESA and z/VM. Each of these modes has its own advantages and disadvantages, although deployment under VM and VIF seems to be the most attractive given the flexibility and the number of systems supported.

NATIVE

Linux can run native on the bare metal of a System/390 or zSeries to provide a single Linux environment. However, because only a single image is supported this is a very limited mode of operation that is not practical or cost effective for most users. It is probable that this configuration would be used only for testing purposes. The other limitation of running Linux for S/390 native is that Linux itself would be responsible for the I/O error recovery on the networked devices losing the benefits of the System/390 and zSeries platforms. Also, because it is not possible to define System/390 storage and I/O configurations from the software level the definitions must be made from the hardware console level.

Linux does not yet support many of the standard system configuration tools such as IOCP or EREP, and without tape support it is difficult to back-up the system. While these limitations may eventually be overcome – the Linux community will almost certainly add the necessary support given time – they will remain for the foreseeable future.

It is possible that this mode could be used by those experimenting with System/390 simulators and looking for a low cost operating environment to support tinkering with the System/390 architecture. Other than this there are therefore few advantages to running a single native Linux image per box.

LPAR

The use of the logical partitioning (LPAR) facilities on the System/ 390 and zSeries can support up to 15 images on a single high-end machine. These images could be used to support Linux development, testing, and production environments, as well as other operating systems. This is a likely initial configuration for sites with only z/OS, OS/390, or VSE/ESA.

This can be useful for introducing Linux-based front-ends to existing z/OS, OS/390, or VSE/ESA-based applications. However, the architectural limitation of 15 LPARs per physical system makes the comparison with non-System/390 hardware unfavourable because of the high initial cost of System/390 hardware.

The I/O hardware devices are dedicated to each partition so data is shared through these devices, not through network transactions as in native mode. The throughput will therefore be high. Active instances can heavily impact the available CPUs for heavily-threaded applications. The use of VM or Virtual Image Facility allows simple distribution and load balancing across CPU engines, providing significantly better throughput than with LPAR deployments.

In most cases, deployment of Linux on LPAR is not suitable for enterprise deployment because of the limitations in management and resource management. However, sites experimenting with enterprise Linux services and making a case for expanding services may have a limited use for this configuration. However, LPAR does have the advantage of being cheap, which is why so many users are exploiting it for Linux.

VIRTUAL IMAGE FACILITY (VIF)

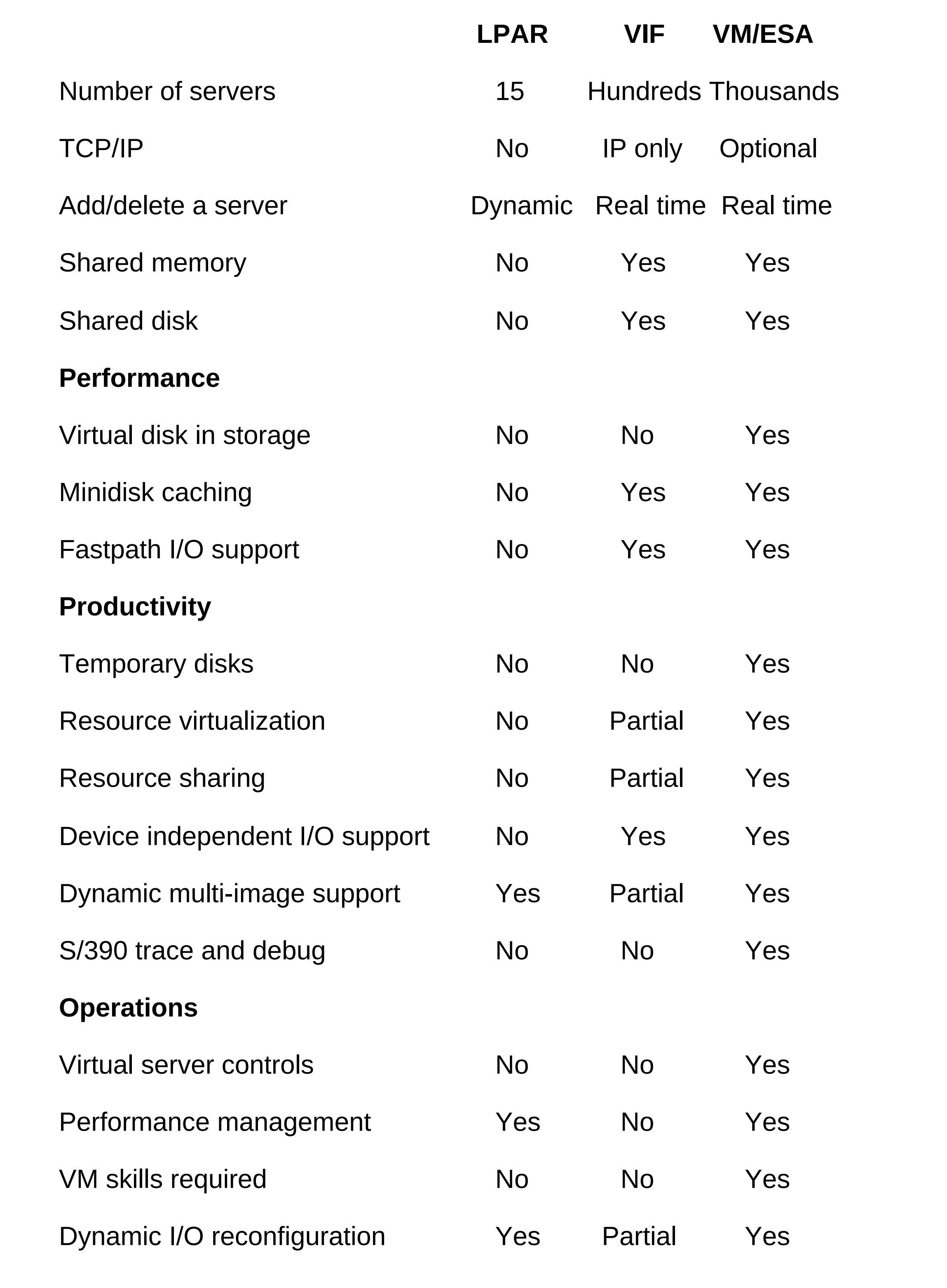
The System/390 Virtual Image Facility for Linux was announced on 26 August, 2000. The Virtual Image Facility is a limited implementation of the VM hypervisor technology enabling users to run hundreds of Linux server images on a single System/390 or zSeries server. This is ideal for users who do not need the tens of thousands of Linux images that VM can support as guests. The Virtual Image Facility offers a low-cost introduction to the virtual environment intended to introduce users unfamiliar with the virtual system environment to system management. There are two principal advantages in Virtual Image Facility over traditional VM/ESA:

- *Cost*: because VIF is a stripped down version of VM with functionality closely linked with Linux it is much cheaper.
- *Skills*: user organizations do not need to invest in VM skills if they do not already have them.

VIF presents a method for sites to deliver quick deployment of Linux for S/390 and z900 systems as part of a planned small- to medium-size deployment that requires more images than an LPAR-based or native System/390 solution can deliver, but that cannot cost-justify a full VM/ESA licence.

It is ideally suited for those who want to move Linux and/or Unix workloads deployed on multiple servers onto a single System/390 server, while maintaining the same number of distinct server images. This provides centralized management and operation of the multiple image environment, reducing complexity, easing administration, and lowering costs.

The Virtual Image Facility allows users to consolidate operations, servers, and networks onto a single physical system for improved manageability. Additionally they can create and manage dynamic Linux images quickly, share system resources among Linux images, provide high-speed communication among Linux images, simplify system resource management, port Unix-like applications more easily to the System/390 platform, and isolate Linux images from one another. Deploying Linux workloads on the Virtual Image Facility is particularly attractive if the workload interacts with System/390 servers, applications, or data located on the same System/390 server.



	LPAR	**VIF**	**VM/ESA**
Number of servers	15	Hundreds	Thousands
TCP/IP	No	IP only	Optional
Add/delete a server	Dynamic	Real time	Real time
Shared memory	No	Yes	Yes
Shared disk	No	Yes	Yes
Performance			
Virtual disk in storage	No	No	Yes
Minidisk caching	No	Yes	Yes
Fastpath I/O support	No	Yes	Yes
Productivity			
Temporary disks	No	No	Yes
Resource virtualization	No	Partial	Yes
Resource sharing	No	Partial	Yes
Device independent I/O support	No	Yes	Yes
Dynamic multi-image support	Yes	Partial	Yes
S/390 trace and debug	No	No	Yes
Operations			
Virtual server controls	No	No	Yes
Performance management	Yes	No	Yes
VM skills required	No	No	Yes
Dynamic I/O reconfiguration	Yes	Partial	Yes

Figure 1: A comparison of running Linux for S/390 using LPAR, VIF, or VM/ESA and z/VM

However, VIF lacks the individual resource management and automation capabilities of VM/ESA and VIF is only available for IFL engines. Therefore, the decision to run VIF or VM on an IFL engine will be based on the availability of VM skills, and the requirements of the applications needed. Figure 1 provides an overview of the comparative functionality of running Linux for S/390 on LPAR, Virtual Image Facility (VIF) or as a guest of VM.

VM/ESA AND Z/VM

Using VM to support Linux could turn out to be the 'killer app' for Linux on the mainframe. While VIF is easy to install and use, and requires little VM skill, VM and z/VM provides heavy-duty system management facilities which allows easy management of hundreds of Linux guests. It also includes system administration tools for performance, accounting, auditing, etc, plus enhanced security features, and a wider variety of supported disk and tape storage devices (see Figure 1). All these capabilities are useful in a large-scale deployment. This is certainly the most flexible and desirable solution and is likely to reverse the decline in VM licences seen in the last decade.

Using a Virtual Machine environment to support Linux allows each end user complete access to the System/390 environment (including CPU, memory, and I/O). VM has over thirty years of maturity to support it. Virtualization is supported through emulation mode on the CPU's and VM's Control Program component.

VM works with Red Hat Linux (*http://www.redhat.com*), SuSE Linux (*http://www.suse.de/en/produkte/susesoft/s390/*), and TurboLinux (*http://www.turbolinux.com*). Support is provided by each distributor, IBM business partners and, obviously, IBM Global Services.

VM enables users to run a large number of Linux server images on a single S/390 or zSeries server. It is ideally suited for those who want to move Linux and/or Unix workloads deployed on multiple servers onto a single S/390 or zSeries server, while maintaining the same number of distinct server images. These Linux images can be deployed on standard processor engines or IFL processor features. Server consolidation often results in cost savings realized by managing large server farms deployed on virtual servers instead of multiple hardware servers.

The new z/VM exploits and supports the z/Architecture, enabling users to run 64-bit capable operating systems (OS/390 Version 2 Release 10, z/OS, and Linux for zSeries) as guests of z/VM Version 4 when z/VM Version 4 is running on a zSeries server in 64-bit mode. A z/VM Version 4 guest operating system running in ESA/390 mode such as VSE/ESA, TPF, OS/390, Linux for S/390, or VM/ESA may realize performance benefits from additional central storage when z/VM is operating in 64-bit mode. In order for z/OS to operate as a guest of z/VM on a zSeries server, both z/VM and z/OS must be operating in 64-bit mode.

There are a number of compelling reasons for running Linux as a guest under VM. This will certainly extend the lifespan of the VM operating system as seen by the release of 64-bit z/VM. It is highly ironic that IBM has been trying to migrate VM users to OS/390 for a long time.

- *Server consolidation* – running Linux on VM will be particularly attractive for users who see the System/390 as the place to centralize and consolidate their growing farms of distributed intranet and Web servers. Resources can be shared among multiple Linux images running on the same VM/ESA system.

- *Virtualization* – the virtual machine environment is highly flexible and adaptable. New Linux guests can be added to a VM/ESA system quickly and easily without requiring dedicated resources. In the rapid pace of the Web arena this could be crucial.

- *System/390 hardware support* – Linux guests can transparently take advantage of VM’s support for System/390 hardware architecture and RAS features.

- *Communications* – VM/ESA provides high-performance communication among virtual machines running Linux and other operating systems on the same processor.

- *Debugging* – VM/ESA offers a functionally rich debug environment that is particularly valuable for diagnosing problems in the Linux kernel and device drivers.

- *Support* – from a support perspective the staff only have to maintain one Linux image. Therefore, all images that end-users have access to will be at the same level of maturity.

- *Growth* – an effective and simple way to grow Linux workload capacity is to add more Linux guests to a VM/ESA system. The time required to set up and deploy a Linux image from VM or VIF is negligible.

WHO IS USING LINUX NOW?

In a recent Xephon survey of large systems users, we reviewed sites that were using Linux on the mainframe. As of June 2001, 8% of repondents were running Linux on their System/390 or zServers. Of these 25% were running Linux under VM or the Virtual Image Facility, while 75% were running Linux in an LPAR.

Two thirds of all sites questioned have no intention of running Linux on a mainframe. In the future, however, a further quarter of all sites plan to run Linux, which will increase its total penetration to one third of all System/390s and zServers. It is estimated that in the future almost a third of mainframes running Linux on S/390 will run Linux under VM or Virtual Image Facility, while two thirds will run Linux in an LPAR.

CONCLUSIONS

The principal driving forces behind this heightened interest in Linux for System/390 and zSeries are the spiralling costs associated with buying and managing the growing number of departmental Unix- and Windows NT-based servers.

This makes the availability of Linux on System/390 of major importance. The ability to achieve the logical extreme of consolidation by putting many environments onto one mainframe will give the System/390 and z900 a boost. The influence of Linux at least encouraged the major Unix vendors to settle on the File System Standard to give something closer to 'Windows-like' transportability of files.

The crucial selling point of a Linux/mainframe solution running VIF or VM is that as the number of virtual Linux servers increases the cost per server decreases. With a conventional set-up running a multitude of departmental servers, the cost increases as the number of physical servers increases. This is the key difference between the System/390 platform and the competition. The only limitation to the number of Linux instances running under VM or Virtual Image Facility is the physical memory and CPU resources of the underlying processors. In most cases, the most significant limitation will be network bandwidth if internal OSA adapters are used (a maximum of 16 adapters are currently supported). With smaller hardware configurations scalability means more boxes, which equates to more maintenance, floorspace, power, and personnel.

Given the overlap between NT and Linux use, it must be assumed that Linux is beginning to eat into specific NT application areas, presumably low down the Web server hierarchy. While it is not discouraging organizations from using NT, it is encroaching on NT's use in specific areas. It is ironic to note that price – of hardware with operating system, of applications software, and of maintaining the currency of software – is cited frequently as the reason for preferring Linux to Microsoft-based solutions.

The introduction of yet another platform type in a world constrained by skill shortages needs genuine justification if it is going to make substantial long-term progress. On the other hand, Linux's position in academic institutions means that there is a certain amount of raw talent coming into the commercial field with some skills in place.

What makes Linux different is the GPL, which removes the licensing considerations and the groundswell of industry and end-user support behind Linux. Linux has also given IBM an opportunity to offer applications that cannot be ported to the z900 quickly, hence making Linux a truly viable open-systems solutions for z900 users.

The coming year will be critical in Linux's progress, defining whether or not the Open Source contender can establish itself as a viable mainframe-based applications platform. The shaky state of the dot-com commercial world in the second quarter of 2001 is more of a concern for Microsoft and Unix than for Linux; the former is competing for sales to the e-business-or-nothing companies while the latter is more likely to be put in place by a company treating e-business, or Web-facing activity, as an addition to its existing core activities.

MVS news

Docucorp has begun shipping Docusave Server 1.5.6, which adds MVS support to its Docusave and Documanage product sets, and enables a connection between mainframe document production and client/server document management, including Internet delivery. The idea is to target companies whose documents are generated on mainframes.

The new system combines mainframe document production with client/server management and Web delivery, including both thick and thin client configurations. The publishing engines that create documents on MVS can now utilize the same document management system that hosts Windows workstations and Web-connected users.

Users can mix and match database and storage systems across platforms. This means MVS DB2 and Windows NT SQL Server tables can work together with VSAM and NT storage area networks, allowing the most appropriate technology to be utilized without integration or reconciliation issues.

Docusave captures documents where they are produced and directly updates the database and storage volumes. The system provides an open common folder in which documents of all types can be filed, managed, and viewed in a single filing scheme.

For further information contact:

Docucorp International, 5910 North Central Expressway, Suite 800, Dallas, Texas 75206-5140, USA.
Tel: (214) 891 6500
Fax: (214) 987 8187

http://www.docucorp.com

* * *

Serena Software has added IMS support for its Serena StarTool application availability products. StarTool FDM (File and Data Manager) is extending its IMS support to include database access for IMS BMP regions, while StarTool DA Batch (Batch Dump Analyser) will now support IMS applications using DLI or BMP regions. BMP support for StarTool FDM is designed to provide a way to fix problems and incorrect data in on-line databases. It removes the need to create and/or run a utility program and then validate the change. Support for IMS in StarTool DA provides a way to locate information in an IMS application dump, which should reduce the time it takes to solve problems in IMS applications. By creating a mini-dump in the sysprint of a failing application, users get access to all the information needed to resolve the cause of an abend.

Serena also announced that its ChangeMan for z/OS and OS/390 now supports IBM’s WebSphere Studio Asset Analyzer, which provides a framework for connecting applications for z/OS, OS/390, and J2EE platforms.

For further information contact:

Serena Software, 500 Airport Boulevard, 2nd Floor, Burlingame, California, 94010-1904, USA.
Tel: (650) 696 1800
Fax: (650) 696 1849

Serena Software UK, Nash House, Repton Place, White Lion Road, Amersham, Buckinghamshire, HP7 9LP, UK.
Tel: 01494 766777
Fax: 01494 766888

http://www.serena.com

* * *