178

MVS

July 2001

update

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

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 ($43.00) each including postage.

Organizing Assembler code

The following edit macro was designed to align Assembler code in specified columns and to uppercase, lowercase, or capitalize all or parts of it. Many people, including myself, still write Assembler code in uppercase, although Assembler can now be written in lowercase. Some people like to write comments fully uppercased, while others prefer it in lowercase. Aligning instructions and comments is also common, and each person has their own way of doing things. This macro allows users to do two things:

- Align instructions, operands, and comments in pre-defined columns. Instructions and operands are set to the classical columns 10 and 16, and comments are pre-set to column 36. However, you can change the comment column at any time simply by passing a numeric parameter with a value between 20 and 70. This way, you can dynamically adjust your comments position to obtain a better alignment. If any comment does not fit in, it may remain misaligned, in order to avoid truncation. Under restricted situations, where no alignments are possible, a line will remain as it was. You can use this feature to right-align comments, by specifying column 70. The contents of columns 72 to 80 are not affected. You can also change instructions and operands columns by changing 'yxbase' variables at the beginning of the macro.
- The second thing the macro does is to uppercase, lowercase or capitalize the following three areas of each line: labels, instructions and its operands, and comments. Each corresponds to a positional parameter in the macro. The allowed values are 'L', 'U', and 'C'. Any other value will revert to the default (see the beginning of the code). Comment lines (or lines that begin with an asterisk) are not modified. The same is true for any quoted string, and also for Assember macro (OPEN, DCB, etc) parameters, because if they were lowercased they would not be recognized in most cases.

Since Assembler code often contains '&', I was forced to use a trick to avoid problems with ISPF confusing things. The trick consists of making a global change of ampersands to a high-value character (it could be any other character that does not occur in the source file) before processing any line, and reversing the change afterwards.

Finally, as I said above, the three parameters that control case settings for labels, instructions, and comments are positional. The fourth parameter, a number that specifies the column where comments should start, need not be positional, since a number is self-evident.

This means that you can omit any or all of the first three. A few examples are shown below:

- PRETTY C 40 – capitalize labels and start comments in column 40. Instructions/operands and comments case setting is default.
- PRETTY 30 L – first and third parameters are default. Second parameter is lowercase. Comments start at column 30.
- PRETTY L C C 33 – all parameters specified.

A last note: this macro was written primarily for Assembler code, not Assembler macros. As far as I can tell, Assembler code will not have any problems with it. I did not test it with macros, because I hardly use them. If you want to use it with macros, do so with caution.

PRETTYA

```
/*= REXX - ISPF EDITOR MACRO =====================================*/
/*                                                                */
/*  PRETTYA                                                       */
/*  This macro performs two tasks in Assembler code source:       */
/*  The first task is to align labels, instruction codes, macros, */
/*  operands, and comments in pre-defined columns. If that is not */
/*  possible because some of these elements do not fit in the     */
/*  pre-defined columns, the macro adjusts things the best way it */
/*  can, in order to avoid truncations. As a last resort, it will */
/*  leave the line untouched, and issue a warning message.        */
/*  The second task is to uppercase, lowercase, or capitalize     */
/*  labels, instructions (code and operands), and comments. This  */
/*  corresponds to the first 3 arguments that the macro accepts.  */
/*  These arguments can have the value ''C'' (capitalize) ''L''   */
/*  (lower) or ''U'' (upper). Anything else reverts to the        */
/*  default, as set below by variables argx_def.                  */
/*                                                                */
/*  There is a fourth parameter that controls the comment column. */
/*  It accepts values between 20 and 70. The default is 36.       */
/*                                                                */
/*  Comment lines and quoted strings are not modified.            */
/*  Macro suboperands (recognized by the presence of an ''='')    */
/*  will not be lowercased, because most Assembler macros accept  */
/*  only uppercase suboperands.                                   */
```



```
/*  If there is a lower or capitalize request, the macro sets the */
/*  editor accordingly, otherwise nothing would happen.           */
/*                                                                */
/*==============================================================*/

/* y1base - label minimum length (padded right with spaces)       */
/* y2base - instruction code (oper1) minimum length, idem         */
/* y3base - instruction operands (oper2) minimum length, idem     */
/*          This means that comments will start at column         */
/* y1base+y2base+y3base+2 spaces, or 8+5+21+2 = 36 by default.    */
/* You can change y3base to control comment default column.       */

y1base = 8
y2base = 5
y3base = 21

/* col_comment: anything that begins after this position is looked*/
/* at as comment. It has no relation at all with ybase settings   */

col_comment = 2Ø

/* Default settings for arg1 (label) agr2 (instr) arg3 (comments) */

arg1_def = ''U''
arg2_def = ''U''
arg3_def = ''C''

address ISPEXEC
'ISREDIT MACRO (ARGS)'
upper args

arg1 = word(args,1)
arg2 = word(args,2)
arg3 = word(args,3)
arg4 = word(args,4)

if datatype(arg1,''W'') then arg4 = arg1
if datatype(arg2,''W'') then arg4 = arg2
if datatype(arg3,''W'') then arg4 = arg3
if datatype(arg4,''W'') then do
   if arg4 > 19 & arg4 < 71 then do
      y3base = arg4 - y1base - y2base - 2
   end
   else do
      say ''Invalid value for comment column entered.''
      say ''It must be between 2Ø and 7Ø''.
      exit
   end
end

if arg1<>''L'' & arg1<>''C'' & arg1<>''U'' then arg1 = arg1_def
```

```
if arg2<>''L'' & arg2<>''C'' & arg2<>''U'' then arg2 = arg2_def
if arg3<>''L'' & arg3<>''C'' & arg3<>''U'' then arg3 = arg3_def
if arg1=''L'' | arg2=''L'' | arg3=''L'' |,
   arg1=''C'' | arg2=''C'' | arg3=''C'' then 'ISREDIT CAPS OFF'

y4base = y1base + y2base + y3base
contin_prev = '' ''
lower_tab = 'abcdefghijklmnopqrstuvwxyz'
upper_tab = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

''ISREDIT (last) = linenum '' .ZLAST
''ISREDIT (w1,w2) = display_lines''
''ISREDIT CHANGE '&' X'FF' ALL''

do k = 1 to last
  'ISREDIT (LIN) = line' k
   firstchar = left(lin,1)
   if firstchar = ''*'' then iterate k
   numseq = substr(lin,73,8)
   contin = substr(lin,72,1)
   lin    = left(lin,71)
   if firstchar = '' '' then do
      label = copies('' '',y1base)
      oper1 = word(lin,1)
      other = subword(lin,2)
   end
   else do
      label = word(lin,1)
      oper1 = word(lin,2)
      other = subword(lin,3)
   end

   if other = '''' then do
      oper2 = ''''
   end
   else do
      col_other = pos(other,lin)
      if col_other > col_comment then do
         oper2 = ''''
      end
      else do
         p1 = pos(''''',other)
         if p1 > Ø then do
            p2 = pos(''''',other,p1+1)
            ps1 = pos('' '',other)
            ps2 = pos('' '',other,p2+1)
            if ps1 < p1 then ps = ps1
            else ps = ps2
         end
         else do
            ps = pos('' '',other)
```

```
      end
      if ps = 0 then do
         oper2 = other
         other = ''''
      end
      else do
         oper2 = substr(other,1,ps-1)
         other = strip(substr(other,ps))
      end
   end
end

if contin_prev <> '' '' then do
   oper2 = oper1
   oper1 = ''''
end

if arg1 = ''U'' then upper label
else label = lower_function(label,arg1)

if arg2 = ''U'' then do
   oper1 = upper_function(oper1)
   oper2 = upper_function(oper2)
end
else do
   oper1 = lower_function(oper1,arg2)
   if pos(''='',oper2) = 0 & contin = '' '' &,
      length(oper1) < 5 & left(oper2,1) <> ''('' then do
      oper2 = lower_function(oper2,arg2)
   end
end

if arg3 = ''U'' then other = upper_function(other)
else other = lower_function(other,arg3)

y1 = y1base
if length(label) > y1 then y1 = length(label)
y2 = y2base
if length(oper1) > y2 then y2 = length(oper1)
y3 = y3base
if length(oper2) > y3 then y3 = length(oper2)

lin = strip(left(label,y1) left(oper1,y2) left(oper2,y3),''T'')
y4 = y4base
if length(lin) > y4 then y4 = length(lin)
linw = left(lin,y4) other
if length(linw) > 71 then do
   extra = 71 - length(lin) - length(other)
   if extra < 1 then do
      extra = 1
      say '' Comment truncation would occur in line'' k
```

```
          say '' For that reason, the line was not changed''
          contin_prev = contin
          iterate k
       end
       linw = lin || copies('' '',extra) || other
    end
    lin = left(linw,71) || contin || numseq
   'ISREDIT  line' k '= ''''lin''''
    contin_prev = contin
end k

''ISREDIT CHANGE  X'FF' '&' ALL''
''ISREDIT LOCATE '' w1
''ISREDIT RESET''
exit

/*=============================================================*/

upper_function:
 parse arg p1 '''' z1 '''' p2 '''' z2 '''' p3
 p1 = translate(p1,upper_tab,lower_tab)
 if p2 <>'''' then p2 = translate(p2,upper_tab,lower_tab)
 if p3 <>'''' then p3 = translate(p3,upper_tab,lower_tab)
 if z1 <>'''' then z1 = '''''z1'''''
 if z2 <>'''' then z2 = '''''z2'''''
return strip(p1 || z1 || p2 || z2 || p3)

lower_function:
 parse arg stri, type
 parse var stri p1 '''' z1 '''' p2 '''' z2 '''' p3
 if type = ''C'' then do
    p1a = translate(left(p1,1),upper_tab,lower_tab)
    p1b = translate(substr(p1,2),lower_tab,upper_tab)
    p1 = p1a || p1b
 end
 else do
    p1 = translate(p1,lower_tab,upper_tab)
 end
 if p2 <>'''' then p2 = translate(p2,lower_tab,upper_tab)
 if p3 <>'''' then p3 = translate(p3,lower_tab,upper_tab)
 if z1 <>'''' then z1 = '''''z1'''''
 if z2 <>'''' then z2 = '''''z2'''''
return strip(p1 || z1 || p2 || z2 || p3)
```

Luis Paulo Figueiredo Sousa Ribeiro
Systems Programmer (Portugal)

Ensuring that a dataset is not in use before backing it up

INTRODUCTION

DFDSS allows users to put TOL (ENQF) in a back-up step. It will tolerate the fact that a dataset is already in use at the time that it is backed up. The problem with this scenario is that the back-up may in fact be worthless in the end – if the file has changed whilst being backed up, it is inconsistent and not usable. One way to make sure that a dataset is not in use is to code DISP=OLD in the step that does the back-up. This has the drawback that the entire job will stop to wait for the dataset to be freed. For the duration of the job (not just the backup step) the back-up will then exclusively own the dataset. This is unnecessary because typically the dataset needs to be freed only during the back-up step, not during the entire job.

Dataset ENQ contention is managed by GRS. The GQSCAN macro is the only documented way to communicate with GRS, and it is this macro that we can use to dynamically check whether a dataset is in use at any point in time, rather than having to rely on the disposition coded in the JCL.

The following programs demonstrate this principle. Program GRSQUERY is a subroutine that does a GQSCAN to look for all ENQs on a dataset, the name of which is passed as a parameter. These fields are received as input/returned as output parameters:

```
PARMDATA DSECT          Input parameters
ACTIONIN DS    C        I<== N=none, W=WTO, R=Resolve
DSNAME   DS    CL44     I<== Dataset to verify usage for
#JOBNAME DS    CL2      I<== Number of jobs to report on
NUMENQ   DS    CL2      O==> Number of JOBS enqueued
EXCLENQ  DS    C        O==> Y/N for current exclusive ENQ
EXCLJNAM DS    CL8      O==> Name of job EXCL ENQ'ed
JOBNAME  DS    CL8      O==> Name of Job ENQ'ed
ENQTYPE  DS    C        O==> Type of ENQ
```

The dataset to be checked for is passed as an input parameter. Different actions will then be taken, based on the value specified in the ACTIONIN parameter:

- ACTIONIN=N – if the dataset is in use, give RC=8
- ACTIONIN=W – if the dataset is in use, give a message and RC=8
- ACTIONIN=R – if the dataset is in use, prompt the operator to respond with an 'R' to retry at a later stage. If the operator responds with an 'A' (Abort) the step terminates with RC=8.

The program will also return a list of job names that have the dataset in use, starting at offset JOBNAME, and also the type of usage, 'E' (exclusive) or 'S' (shared) for each of these jobs (ENQTYPE). It is the caller's responsibility to ensure that there is a target area to move the data into. This is controlled by the value put into NUMENQ: if for instance NUMENQ has a value of 2 then there has to be 2x(8+1) = 18 bytes available at offset JOBNAME. If not, an 0C4 is the most likely outcome. The value of NUMENQ can of course be set to 0, in which case no output area is required.

If there is a job that has exclusive use of a dataset (there can obviously be only one) then EXCLJNAM is filled with the job name and EXCLENQ is set to 'Y'.

The GRSQUERY program obviously has other uses as well: it will return information on dataset usage for any dataset, and this information can be used for different purposes. The other program included here, INUSECHK, is a specific application of GRSQUERY to give us the ability to ensure that a dataset is not in use at the time that we need to take a back-up. It does some parameter verification and then calls GRSQUERY. The following are examples of how INUSECHK can be run:

```
//STEP2    EXEC  PGM=INUSECHK,PARM='N,USER.LOADLIB'
```

See if USER.LOADLIB is in use and give RC=8 if it is.

```
//STEP3    EXEC  PGM=INUSECHK,PARM='W,USER.LOADLIB'
```

See if USER.LOADLIB is in use and give a message and RC=8 if it is.

```
//STEP4    EXEC  PGM=INUSECHK,PARM='R,USER.LOADLIB'
```

See if USER.LOADLIB is in use and resolve it with the operator. The operator has to enter 'R' for retry or 'A' for abort, in which case RC=8 is returned.

GRSQUERY

```
GRSQUERY CSECT
GRSQUERY AMODE 31
GRSQUERY RMODE ANY
         BAKR  R14,Ø                   Save Caller's Status
         BALR  R12,Ø
         USING *,12
**********************************************************************
*        Main driver routine
**********************************************************************
LOAD     LR    R4,R1                   Pointer to parameters
         USING PARMDATA,R4             Addressability to parm area
         LA    R3,GETMSIZE             Our requirement
         A     R3,=F'3ØØØØ'            Required for GQSCAN
         GETMAIN R,LV=(3)              Getmain our workarea
         USING GETMAREA,R1
         ST    R13,SAVEAREA+4          Backchain
         DROP  R1
         LR    R13,R1
         USING GETMAREA,R13            Addressability to getmained area
         ST    R4,PARMSTRT             Preserve start of passed parms
         BAS   R14,SCANPARM            Go scan input parameters
         BAS   R14,GETJNAME            Go find our own jobname
CALLGRS  BAS   R14,DOGQSCAN            Set up and do GQSCAN
         LTR   R15,R15                 Do we have any info to process?
         BNZ   RETURN                  No
GODOANAL BAS   R14,ANALSCAN            Analyse output from GQSCAN
         TM    ONLYUSFL,YES            Are we the only user of the ds?
         BO    RETURN                  Yes, get out
         BAS   R14,RESOLVE             Take action depending on WTO ind
         CLI   ACTIONIN,C'R'           Must it be resolved before return?
         BNE   RETURN                  Yes
         L     R1,RETCODE              Pick up return code
         CH    R1,=H'12'               Abort?
         BNE   CALLGRS                 No, retry
RETURN   L     R4,RETCODE              Pick up return code
         LA    R3,GETMSIZE
         A     R3,=F'3ØØØØ'
         FREEMAIN R,LV=(3),A=(13)
         LR    R15,R4                  Copy return code
         PR                            To caller
**********************************************************************
*        This routine scans the input parms for validity
**********************************************************************
SCANPARM EQU   *
         BAKR  R14,RØ                  Preserve our return address
* Put input parm scanning/ verification in here if required
SCANPARX PR                            Back to our caller
**********************************************************************
*        This routine gets our own jobname
**********************************************************************
```

```
GETJNAME BAKR  R14,RØ                Preserve our return address
         XR    R5,R5                 PSA starts at zero
         USING PSA,R5
         L     R6,PSATOLD            Pointer to current TCB
         USING TCB,R6                Addressability to TCB
         L     R7,TCBTIO             Pointer to TIOT
         USING TIOT1,R7              Addressability to TIOT
         MVC   EXTRAREA(EXTRLENG),EXTrMac
         LA    R2,ADDRSPC
         LA    R1,EXTRAREA           Point to extract macro
         EXTRACT ((2)),MF=(E,(1))
         L     R2,ADDRSPC
         LTR   R2,R2                 In use?
         BNZ   BATCH
         L     R5,PSAANEW            Pointer to ASCB
         DROP  R5
         USING ASCB,R5
         L     R5,ASCBJBNS           Pointer to JOBNAME
         MVC   OURJNAM,Ø(R5)         Move jobname in
         B     GETJNAMX              Get out
BATCH    MVC   OURJNAM,TIOCNJOB      Pick up our jobname
GETJNAMX PR                          Back to our caller
*******************************************************************
*        This routine sets up and does the GQSCAN
*********************************************************************
DOGQSCAN EQU   *
         BAKR  R14,RØ                Preserve our return address
         XC    NUMENQ,NUMENQ         Set #RIB to zero
         XC    RETCODE,RETCODE       Reset program's return code
         LA    R2,WORKAREA           GQSCAN output area
         LA    R5,SYSDSN             Point to Qname
         LA    R6,DSNAME             Point to Rname
         OC    DSNAME(44),=44X'4Ø'   Make sure name has spaces at back
         XR    R7,R7                 Length of Rname
         LA    R1,DSNAME             Point to start of dataset name
         LA    R1Ø,44                Maximum dsname length
BLANKSRC EQU   *                     Look for first blank in dsname
         CLI   Ø(R1),C' '            Is is a blank?
         BE    BLANKFND              Yes
         LA    R7,1(R7)              Bump up counter
         LA    R1,1(R1)              Bump up pointer
         BCT   R1Ø,BLANKSRC          Keep on searching
BLANKFND LA    R1,GQMACRO1           List form of macro
         GQSCAN AREA=((2),3ØØØØ),SCOPE=ALL,RESNAME=((5),(6),(7)),      X
               REQLIM=MAX,MF=(E,(1))
CHECKØ   LTR   R15,R15               Successful & dataset in use?
         BZ    DOGQSCAX              Yes
CHECK4   CH    R15,=H'4'             Succesful & dataset not in use?
         BE    DOGQSCAX              Yes
CHECK8   CH    R15,=H'8'             Unsuccessful & area too small?
         BNE   GQERROR               No, must be other error
         LR    R2,R15                Preserve return code
```



```
         WTO   'GRSQUERY(E): -OUTPUT area too small',ROUTCDE=11
         LR    R15,R2                Reload return code
         ABEND ØØ1,DUMP
GQERROR  LR    R2,R15                Preserve return code
         WTO  'GRSQUERY(E): -UNEXPECTed error during GQSCAN',ROUTCDE=11
         LR    R15,R2                Reload return code
         ABEND ØØ2,DUMP
DOGQSCAX PR                          Back to our caller
*********************************************************************
*        This routine analyses GQSCAN output
*********************************************************************
ANALSCAN EQU   *
         BAKR  R14,RØ                Preserve our return address
         ICM   R1,15,#JOBNAME        # of jobs we have space for
         LTR   R1,R1                 Any space?
         BZ    ANALSCAX              No, get out
READRIB  LA    R2,WORKAREA           Point to GQSCAN output
         USING RIB,R2                Addressability to GRS RIB
         LH    R3,RIBVLEN            Length of variable ...
         L     R5,RIBNRIBE           Number of RIBEs returned
         LTR   R5,R5                 Any?
         BNZ   STORNUM               Yes
         ABEND ØØØ3,DUMP             Should never happen
STORNUM  STCM  R5,3,NUMENQ           Store into parameter list
         LA    R2,RIBEND-RIB(R2)     Point to end of RIB,
         AR    R2,R3                 And add length of variable part
         DROP  R2
         USING RIBE,R2               Addressability to RIBE
         XR    R1,R1                 Count number of RIBEs
* The user might ask for fewer jobs than there are actually enqueued
* on the resource. Once we have supplied all the names the user has
* given space for, we still have to keep on scanning all the RIBEs to
* see if one of them has an exclusive ENQ, in use or pending.
         MVI   EXCLENQ,C'N'          Default no exclusive ENQ
         MVC   EXCLJNAM,=CL8'NONE'   No jobname exclusive by default
RIBELOOP EQU   *
         CLC   OURJNAM,RIBEJBNM      Is this us?
         BNE   CHKEXCL               No
         CLC   NUMENQ,=H'1'          Are we the only one?
         BNE   REDUCNUM              No, there are other jobs as well
ONLYUS   OI    ONLYUSFL,YES          Set the flag on
         B     ANALSCAX              Yes, we are the only entry
REDUCNUM LR    R1,R4                 Preserve the value of R4
         L     R4,PARMSTRT           Reload to start of parms
         ICM   R3,3,NUMENQ           Number of jobs ENQ'ed
         BCTR  R3,Ø                  Reduce by 1
         STCM  R3,3,NUMENQ           Store it back
         LR    R4,R1                 Restore pointer into parm fields
CHKEXCL  TM    RIBETYPE,X'ØØ'        Exclusive ENQ?
         BNO   SETSHARE              No, must be shared ENQ
SETEXCL  MVI   EXCLENQ,C'Y'          Yes, there is an exclusive ENQ
         MVC   EXCLJNAM,RIBEJBNM     Remember name of exclusive ENQer
```



```
         CLM   R1,3,#JOBNAME        As many as caller allows us?
         BH    BUMPUP               Yes, don't move more
         MVI   ENQTYPE,C'E'
         B     MOVEJNAM             Also move jobname in
SETSHARE CLM   R1,3,#JOBNAME        As many as caller allows us?
         BH    BUMPUP               Yes, don't move more
         MVI   ENQTYPE,C'S'
MOVEJNAM MVC   JOBNAME,RIBEJBNM     Move Jobname into passed area
         LA    R4,ENTSIZE(R4)       Bump up pointer into parm area
         LA    R1,1(R1)             Bump up counter
BUMPUP   LA    R2,RIBEEND-RIBE(R2)  Bump up pointer into RIBE
         BCT   R5,RIBELOOP          Do for each entry
ANALSCAX PR                         Back to our caller
*********************************************************************
*        This routine decides what to do with the GRS result
*********************************************************************
RESOLVE  BAKR  R14,RØ
         L     R4,PARMSTRT          Reload pointer to passed parms
         LA    R15,8                "Dataset-is-in-use" return code
         ST    R15,RETCODE          Plug into program's return code
RSLTIGN  CLI   ACTIONIN,C'N'        Must we do nothing?
         BE    RESOLVEX             Yes
RSLTWTO  CLI   ACTIONIN,C'W'        Must we WTO only?
         BNE   RSLTWTOR             No, we must do a WTOR
         XR    R1,R1
         ICM   R1,3,NUMENQ          Total # of jobs enqueued
         BCTR  R1,Ø                 Subtract 1
         CVD   R1,DOUBLE            Convert and...
         UNPK  DOUBLE(8),DOUBLE+5(3)
         OI    DOUBLE+7,X'FØ'           make printable
         MVC   WTOAREA(WTOLENG),GENWTO
         MVC   WTOAREA+26(44),DSNAME
         MVC   WTOAREA+81(8),JOBNAME
         MVC   WTOAREA+94(3),DOUBLE+5
         LA    R1,WTOAREA+4         Start of message text in WTO
         LA    R2,1Ø4               Length of the text
         NI    BLANKFLG,X'ØØ'       Turn the found-blank flag off
         MVC   NEWMSG,=13ØC' '      Clear the message area
         LA    R3,NEWMSG            Point to start of new message
DEBLANK1 CLC   Ø(3,R1),=C'ØØØ'      "ØØØ other jobs" text?
         BNE   CKBLANK1             No
         LA    R1,14(R1)            Skip the next 13 characters
         SH    R3,=H'5'             Move the "to" pointer 4 bytes back
         MVC   Ø(5,R3),=C'     '    Replace "and" with blanks
         B     MOVEBACK             Get out of the loop
         SH    R2,=H'14'            Reduce loop counter by length of it
         B     DEBLANK1             Go to top of loop
CKBLANK1 CLI   Ø(R1),C' '           Is it a blank?
         BE    BLANK1               Yes
         NI    BLANKFLG,X'ØØ'       Turn the found-blank flag off
         B     MOVECHR1             Go move the character
BLANK1   TM    BLANKFLG,X'Ø1'       Flag already on?
```

```
         BO    BUMPSRC1             Yes, don't move the character
         OI    BLANKFLG,X'Ø1'       Turn the flag on
MOVECHR1 MVC   Ø(1,R3),Ø(R1)        Move the 1 character
         LA    R3,1(R3)             Bump up the "to" pointer
BUMPSRC1 LA    R1,1(R1)             Bump up the "from" pointer
         BCT   R2,DEBLANK1
MOVEBACK MVC   WTOAREA+4(1Ø4),NEWMSG
         LA    R1,WTOAREA
         WTO   MF=(E,(1))
         B     RESOLVEX
RSLTWTOR MVC   WTORAREA(WTORLENG),GENWTOR
         XR    R1,R1
         ICM   R1,3,NUMENQ          Total # of jobs enqueued
         BCTR  R1,Ø                 Subtract 1
         CVD   R1,DOUBLE            Convert and...
         UNPK  DOUBLE(8),DOUBLE+5(3)
         OI    DOUBLE+7,X'FØ'           make printable
         MVC   WTORAREA(WTORLENG),GENWTOR
         MVC   WTORAREA+34(44),DSNAME
         MVC   WTORAREA+89(8),JOBNAME
         MVC   WTORAREA+1Ø2(3),DOUBLE+5
         LA    R1,WTORAREA+12       Start of message text in WTOR
         LA    R2,122               Length of the text
         NI    BLANKFLG,X'ØØ'       Turn the found-blank flag off
         MVC   NEWMSG,=13ØC' '      Clear the message area
         LA    R3,NEWMSG            Point to start of new message
DEBLANK2 CLC   Ø(3,R1),=C'ØØØ'      "ØØØ other jobs" text?
         BNE   CKBLANK2             No
         LA    R1,14(R1)            Skip the next 13 characters
         SH    R3,=H'5'             Move the "to" pointer 4 bytes back
         SH    R2,=H'14'            Reduce loop counter by length of it
         B     DEBLANK2             Go to top of loop
CKBLANK2 CLI   Ø(R1),C' '           Is it a blank?
         BE    BLANK2               Yes
         NI    BLANKFLG,X'ØØ'       Turn the found-blank flag off
         B     MOVECHR2             Go move the character
BLANK2   TM    BLANKFLG,X'Ø1'       Flag already on?
         BO    BUMPSRC2             Yes, don't move the character
         OI    BLANKFLG,X'Ø1'       Turn the flag on
MOVECHR2 MVC   Ø(1,R3),Ø(R1)        Move the 1 character
         LA    R3,1(R3)             Bump up the "to" pointer
BUMPSRC2 LA    R1,1(R1)             Bump up the "from" pointer
         BCT   R2,DEBLANK2
         MVC   WTORAREA+12(122),NEWMSg
         LR    RØ,RØ                Clear console id
         XC    ECBAD,ECBAD          Clear ECB
         LA    R2,REPLY             Operator reply area
         LA    R3,ECBAD             ECB address
         LA    R1,WTORAREA
         WTOR  ,(2),15,(3),MF=(E,(1))
         WAIT  ECB=(3)              Wait for operator's reply
         OI    REPLY,X'4Ø'          Make uppercase
```



```
         CLI   REPLY,C'A'           Abort?
         BNE   RESOLVEX             NO
         LA    R15,12               Set return code to 12
         ST    R15,RETCODE          Store
RESOLVEX PR                         Back to our caller
*********************************************************************
*        Constants follow
*********************************************************************
SYSDSN   DC    CL8'SYSDSN  '
GENWTO   WTO  'GRSQUERY(I): -Dataset xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxX
               xxxxxxxxxx in use by yyyyyyyy and xxx other jobs',      x
               ROUTCDE=11,MF=L
WTOLENG  EQU   *-GENWTO
GENWTOR  WTOR 'GRSQUERY(I): -Dataset xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxX
               xxxxxxxxxx in use by yyyyyyyy and xxx other jobs, R(etryx
               ) /A(bort)',,,ROUTCDE=13,MF=L
WTORLENG EQU   *-GENWTOR
EXTRMAC  EXTRACT ,'S',FIELDS=(TSO),MF=L
EXTRLENG EQU   *-EXTRMAC
         LTORG
*********************************************************************
*        DSECTs follow
*********************************************************************
GETMAREA DSECT
SAVEAREA DS    18F                  General savearea
EXTRAREA DS    CL(EXTRLENG)         Area to contain EXTRACT macro
PARMSTRT DS    F                    Start address of passed parms
DOUBLE   DS    D                    General work double word
RETCODE  DS    F
REPLY    DS    CL15                 Operator reply area
ADDRSPC  DS    F                    Full word to store ASCB
ECBAD    DS    F                    WTOR ECB
         DS    ØF
WTOAREA  DS    CL(WTOLENG)          Workarea for WTO
         DS    ØF
WTORAREA DS    CL(WTORLENG)         Workarea for WTOR
OURJNAM  DS    CL8                  Our own jobname
BLANKFLG DS    C                    Flag used to deblank messages
ONLYUSFL DS    C                    Flag used to show we're only user
NEWMSG   DS    CL13Ø                Area used to build deblanked msg
GQMACRO1 GQSCAN ,MF=L
WORKAREA EQU   *                    GQSCAN workarea
GETMSIZE EQU   *-GETMAREA
         DS    ØF
PARMDATA DSECT                      Input parameters
ACTIONIN DS    C                    I<== N=none, W=WTO, R=Resolve
DSNAME   DS    CL44                 I<== Dataset to scan for
#JOBNAME DS    CL2                  I<== Number of jobs to report on
NUMENQ   DS    CL2                  O==> Number of JOBS enqueued
EXCLENQ  DS    C                    O==> Y/N for current exclusive ENQ
EXCLJNAM DS    CL8                  O==> Name of job EXCL ENQ'ed
JOBNAME  DS    CL8                  O==> Name of Job ENQ'ed
```



```
ENQTYPE  DS    C                     O==> Type of ENQ
ENTSIZE  EQU   *-JOBNAME             Size of 1 entry
RØ       EQU   Ø
```

Editor's note: register equates go here.

```
R15      EQU   15
YES      EQU   X'8Ø'
NO       EQU   X'ØØ'
         ISGRIB                      DSECT for GRS request info block
         IHAPSA
         IKJTCB
         IEFTIOT1
         IHAASCB
         END
```

INUSECHK CSECT

```
INUSECHK CSECT
INUSECHK AMODE 31
INUSECHK RMODE ANY
         BAKR  R14,Ø                 Save Caller's Status
         BALR  R12,Ø
         USING *,12
**********************************************************************
*        Main driver routine
**********************************************************************
         L     R4,Ø(R1)              Parm pointer
STORAGE  LA    R3,GETMSIZE           Size of storage to get and clear
         STORAGE OBTAIN,LENGTH=(3),LOC=BELOW,SP=Ø
         LR    R2,R1                 Point to getmained area
         LA    R3,GETMSIZE           Length of storage to clear
         XR    R9,R9                 Fill with binary zeroes
         MVCL  R2,R8                 Propagate binary zeroes
         USING GETMAREA,R1
         ST    R13,SAVEAREA+4        Backchain
         DROP  R1
         LR    R13,R1
         USING GETMAREA,R13          Addressability to getmained area
CHKPARM1 CLC   Ø(2,R4),=H'3'         Parms passed?
         BNL   CHKPARM2              Yes
         WTO   'INUSECHK(E): -ACTION option and dataset name required aX
               S INPUT PARAMETERS',ROUTCDE=11
         LA    R15,12
         ST    R15,RETCODE
         B     GETOUT                Get out
CHKPARM2 MVC   ACTIONIN,2(R4)        Action to perform on dataset-in-use
         CLC   2(2,R4),=C'N,'        "No action"?
         BE    GETDSNAM              Yes
CHKPARM3 CLC   2(2,R4),=C'W,'        "WTO action"?
         BE    GETDSNAM              Yes
```

```
CHKPARM4 CLC   2(2,R4),=C'R,'          "RESOLVE action"?
         BE    GETDSNAM                Yes
         WTO   'INUSECHK(E): -ACTION option must be "N(o)", "W(TO)" or X
               "R(ESOLVE)"',ROUTCDE=11
         LA    R15,12
         ST    R15,RETCODE
         B     GETOUT                  Get out
GETDSNAM EQU   *
         LH    R1,Ø(R4)                Get the length of the input parm
         SH    R1,=H'3'                Correct the length
         EX    R1,MOVENAME             Move the dataset name from the parm
         B     CALLROUT                Go call the routine
MOVENAME MVC   DSNAME(Ø),4(R4)
CALLROUT LA    R1,PARMS
         MVC   #JOBNAME,=H'1'
         LINK  EP=GRSQUERY
         ST    R15,RETCODE
GETOUT   L     R4,RETCODE              Pick up retrun code
         LR    R2,R13                  Pointer to storage area
         LA    R3,GETMSIZE             Size of storage to free
         STORAGE RELEASE,LENGTH=(3),ADDR=(2),SP=Ø
         LR    R15,R4                  Reload return code
         PR                            Back to our caller
**********************************************************************
*        Constants follow
**********************************************************************
         LTORG
GETMAREA DSECT
SAVEAREA DS    18F
RETCODE  DS    F
PARMS    DS    ØF
ACTIONIN DS    C                       <==Indicate if we want a WTO(R)
DSNAME   DS    CL44                    <==Dataset to scan for
#JOBNAME DS    CL2                     <==Number of jobs to report on
NUMENQ   DS    CL2                     ==>Number of JOBS enqueued
EXCLENQ  DS    C                       ==>Y/N for current exclusive ENQ
EXCLJNAM DS    CL8                     ==>Name of job EXCL ENQ'ed
JOBNAME  DS    CL8                     ==>Name of Job ENQ'ed
ENQTYPE  DS    C                       ==>Type of ENQ
ENTSIZE  EQU   *-JOBNAME               Size of 1 entry
GETMSIZE EQU   *-GETMAREA
RØ       EQU   Ø
```

Editor’s note: register equates go here.

```
R15      EQU   15
         END
```


A utility to reconstruct lost source code

During the process of migration from VSE/SP to MVS we realized we had a considerable amount of missing source code. Some Assembler subroutines needed for production had to be assembled, but their source code was missing, and because our documentation was not up to date we could not rewrite these programs.

This was why we decided to write the ADISASEM program to restore disassembled source code. ADISASEM reads the input from PDS datasets specified under the DD statement STEPLIB. The load module name, and the optional name of the control section, and the start and end offsets are specified under the DD statement SYSIN. The program processes input data and generates output in the source file. The beginning of the source content, shown below, illustrates the type of the program output that will be obtained:

```
*PROGRAM=ASCEBC  ,SIZE=X000019F0
*  CSECT=ASCEBC  ,SIZE=X000019EC,OFFSETS=(X00000000-X000019EB)
*OFFSET  SOURCE STATEMENT            OBJECT           OBJECT     OFFSET
* DEC                                CODE HEX         CODE CHAR   HEX
ASCEBC    CSECT
L00000    STM    14,12,0012(13)      *90EC D00C       * o(tm)}   00000000
L00004    LR     12,15               *18CF            *  -       00000004
L00006    ST     13,0240(00,12)      *50D0 C0F0       * &}{0     00000006
L00010    LR     02,13               *182D            *          0000000A
L00012    LA     13,0236(00,12)      *41D0 C0EC       * y}{(tm)  0000000C
```

Because the result of disassembling is not deterministic, we have to disassemble the code in multiple iterations. A method to distinguish statements from data inside the load module does not exist. This way ADISASEM offers columns with hex and char representations of code and opcode of recognized statements. This information can help users to choose starting points for partial disassembling.

SOURCE

```
*********************************************************************
* ADISASEM - PROGRAM TO DISASSEMBLE LOAD MODULES
* DATASETS:
*     STEPLIB  - LOAD DATASET THAT CONTAINS PROGRAM TO DISASSEMBLE
*     SYSIN    - INPUT PARAMETER DATASET
*     SOURCE   - OUTPUT DATASET FOR DISASSEMBLED SOURCE CODE
```



```
*     SYSPRINT - MESSAGE DATASET
* PARAMETERS FROM SYSIN DATASET
*——+——1——+——2——+——3——+——4——+——5
*PGMNAME  CSECTNAME START_OFF END_OFF
*
* POZITION     PARAMETER            TYPE
* 1-8          LOAD MODULE NAME     MANDATORY
* 1Ø-17        CSECTNAME            OPTIONAL
* - IF YOU OMIT CSECT NAME ALL CONTROL SECTIONS WILL BE DISASSEMBLED,
*   OR JUST A PART BORDERED BY THE SPECIFIED OFFSETS.
*   2Ø-27        START_OFF          OPTIONAL
* - IF YOU DON'T SPECIFY DISASSEMBLY WILL START FROM THE BEGINNING
*   OF THE CONTROL SECTION
* - OFFSET MUST BE 8 BYTES LONG WITH LEADING ZEROS
*   3Ø-37        END_OFF            OPTIONAL
* - IF END_OFF IS OMITED DISASSEMBLY WILL BE FINISHED AT THE END OF
*   CURRENT CSECT
* - OFFSET MUST BE 8 BYTES LONG WITH LEADING ZEROS
          MACRO
&NAME     SETAMODE  &MODE
          LCLC  &GLAB
&GLAB     SETC  'AM'.'&SYSNDX'
          AIF ('&MODE' EQ '31').MODE31
          AIF ('&MODE' EQ '24').MODE24
.ERROR    IHBERMAC PARAMATAR CAN BE 31 or 24 ONLY
.MODE31   L   1,&GLAB.3A
          BSM  Ø,1
&GLAB.3A  DC A(&GLAB.31+X'8ØØØØØØØ')
          CNOP Ø,4
&GLAB.31  EQU  *
          MEXIT
.MODE24   LA   1,&GLAB.24
          BSM  Ø,1
          CNOP Ø,4
&GLAB.24  EQU  *
          MEND
***********************************************************************
ADISASEM CSECT
ADISASEM AMODE  31
ADISASEM RMODE  24
*——— Prolog ———————————————————————————
         SAVE   (14,12)              Save regs
         BALR   12,Ø
         USING  *,12,11              R12,R11 are basic registers
         ST     14,SAV14
         SR     11,11
         LA     11,4Ø95
         LA     11,1(11)
         AR     11,12                R11=R12+4Ø96
         GETMAIN RU,LV=WRKSIZE       Allocation of the working area
         ST     13,4(1)
         ST     1,8(13)
```

```
         LR     13,1
         USING  WORK,13
         GETMAIN RU,LV=3276Ø
         LR     7,1
*——— Opening of datasets —————————————————
         OPEN   (SYSPRINT,(OUTPUT)),MODE=31
         OPEN   (SOURCE,(OUTPUT),SYSIN,(INPUT)),MODE=31
         OPEN   (STEPLIB,(INPUT)),MODE=31
*——— Set working area to blanks —————————————————
         MVI    PPGMNAME,C' '
         MVC    PPGMNAME+1(PEND-PPGMNAME-1),PPGMNAME
*——— Load of subroutine AINSTR with the list of assembler instr——
         MVC    WPGMNAME(8),=CL8'AINSTR'
         BAL    14,LOADPGM
         ST     Ø,W@INST             ADDRESS OF ADINSTR
         AR     1,Ø
         SH     1,=H'3'
         ST     1,WENDINST           ADDRESS OF THE END
*——— Load of subroutine ASVC with the list of SVC —————————————————
         MVC    WPGMNAME(8),=CL8'ASVC'
         BAL    14,LOADPGM
         ST     Ø,W@SVC              ADDRESS OF ASVC
         AR     1,Ø
         ST     1,WENDSVC            ADDRESS OF THE END
*——— Reading of parameters —————————————————
READ_FROM_PARM  EQU *
         BAL    14,GET_PARM
         BAL    14,DISASEM_PROGRAM
         B      READ_FROM_PARM
*——— End —————————————————————————————-
END      EQU     *
         DELETE EP=AINSTR
         DELETE EP=ASVC
         CLOSE  (SYSIN),MODE=31
         CLOSE  (SOURCE),MODE=31
*——— Epilog ———————————————————————————
         LR     1,13
         L      13,4(13)
         FREEMAIN RU,A=(1),LV=WRKSIZE
         L      15,RET_CODE
         RETURN  (14,12),,RC=(15)
RET_CODE DC  F'Ø'
*======================================================================
*        Prepare load module for disassembling
*======================================================================
DISASEM_PROGRAM DS ØF
         STM    Ø,15,SAVDISPGM
*——— Load of program to disassemble ———————————————
         MVC    WPGMNAME(8),PPGMNAME
         BAL    14,LOADPGM
         ST     Ø,W@PGM
         ST     Ø,WPGMMODE
```



```
         NC     WPGMMODE(4),=X'80000000'
         ST     1,WPGMSIZE           SIZE OF THE PROGRAM TO DISASSEM.
         AR     1,0
         BCTR   1,0
         ST     1,W@ENDPGM           ADDRESS OF THE END
*—— Printing of the header ———————————————————
         BAL    14,PRINT_HEADER
*—— Reading of CESD  name and offsets —————————————
         LA     1,W@CESDG
         ST     1,W@CESDP
         BAL    14,READ_CESDS
*—— Sort CESD information by  offsets —————————————
         BAL    14,SORT_CESD
*—— Print message ————————————————————————
         LA     0,L'MSG00B
         LA     1,MSG00B
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         L      5,W@CESDG
         USING  CESD_LIST,5
NEXT_LIST_CESD EQU *
         LTR    5,5
         BZ     END_DISASEM          IF IT IS NOT, END OF DISASSEMBLING
         CLI    CESD_TYPEH,X'07'     IF NULL Section Definition
         BE     LIST_NEXT_CESD
*—— Set recovery routine ———————————————————
         ESPIE  TEST,WPARM
         CL     15,=F'8'
         BL     NEXT_CESD_NAME
         ESPIE  SET,ESPIEEXT,((1,15))
NEXT_CESD_NAME EQU *
         BAL    14,COPY_CESD_INFO
*—— Check if csect name is specified in parameters ————————
         CLI    PARM_CESD_NAME,C' '
         BE     DISASEMB_NEXT_CESD
         CLC    PARM_CESD_NAME(8),W_CESD_NAME
         BNE    LIST_NEXT_CESD
*—— Checking of specified offsets ———————————————
DISASEMB_NEXT_CESD EQU *
         BAL    14,CHECK_OFFSET
         CLI    WSKIP,C'Y'
         BE     NEXT_CESD
         BAL    14,DISASEMBLING
*—— Move to next CSECT name ———————————————————
LIST_NEXT_CESD EQU *
         L      5,0(5)
         B      NEXT_LIST_CESD
*—— Free CESD list area ————————————————————-
END_DISASEM EQU *
         DELETE EPLOC=PPGMNAME
         L      3,W@CESDG
FREE_CESD EQU *
         LTR    3,3
```

```
         BZ     END_DISASEM_PROG
         FREEMAIN RU,A=(3),LV=CESDSIZE
         L      3,Ø(3)
         B      FREE_CESD
END_DISASEM_PROG EQU *
         LM     Ø,14,SAVDISPGM
         B      Ø(14)
*********************************************************************
*     Reading CESD info from load module and make CESD list structure
*********************************************************************
READ_CESDS DS ØF
         STM    Ø,15,SAVRCESD
         FIND   STEPLIB,PPGMNAME,D
*——— Print message ———————————————————————
         LA     Ø,L'MSGØØA
         LA     1,MSGØØA
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
READ_NEXT_CESD EQU *
         READ   DECB,SF,STEPLIB,(7),'S'
         CHECK  DECB
         CLC    Ø(2,7),CESD          Is record a CSECT list or not?
         BNE    END_READ_CESD        If it is not, end of disassembling
         LH     9,6(7)               R9 - length of CSECT list
         SRL    9,4                  R9=R9/16
         LA     8,8(7)               @R8 - beginning of the first CSECT
*——— Get next CESD information —————————————————
CHECK_NEXT_CESD EQU *
         GETMAIN RU,LV=CESDSIZE
         LR     5,1
         L      1,W@CESDP
         ST     5,Ø(1)
         ST     5,W@CESDP
         SR     15,15
         ST     15,CESD_NEXT
         MVC    CESD_NAME(16),Ø(8)
*——— Checking whether it is a CSECT or not ——————————————
         MVC    CESD_TYPE(5),=C'   LD'
         TM     8(8),X'ØF'
         BNZ    PRINT_CESD
         MVC    CESD_TYPE(5),=C'CSECT'
*——— Print CSECT or LD segment in SYSPRINT ———————————
PRINT_CESD  EQU *
         BAL    14,COPY_CESD_INFO
         LA     1,SYSPRINT
         BAL    14,PRINT_CESD_INFO
*——— Move to next CSECT name ——————————————————-
NEXT_CESD EQU *
         AH     8,=H'16'
         BCT    9,CHECK_NEXT_CESD
         B      READ_NEXT_CESD
END_READ_CESD EQU *
         LM     Ø,15,SAVRCESD
```

```
         B      Ø(14)
*********************************************************************
*        SORT CESD information BY offset
*********************************************************************
SORT_CESD  DS ØF
         STM    Ø,15,SAVWORK
SORT_CESDØ EQU *
         LA     1,W@CESDG
         L      2,W@CESDG
         MVI    WSKIP,C'N'
SORT_CESD1 EQU *
         LTR    2,2
         BZ     END_SORT_CESD
         L      3,Ø(2)
         LTR    3,3
         BZ     END_SORT_CESD
         CLC    18(3,2),18(3)
         BNH    CNECK_NEXT_OFFSET
         MVI    WSKIP,C'Y'
         L      4,Ø(3)
         ST     3,Ø(1)
         ST     2,Ø(3)
         ST     4,Ø(2)
CNECK_NEXT_OFFSET EQU *
         L      1,Ø(1)
         L      2,Ø(1)
         B      SORT_CESD1
END_SORT_CESD EQU *
         CLI    WSKIP,C'Y'
         BE     SORT_CESDØ
         LM     Ø,15,SAVWORK
         B      Ø(14)
*********************************************************************
*        Copy CESD information into work
*********************************************************************
COPY_CESD_INFO EQU *
         MVC    W_CESD_TYPE(5),CESD_TYPE
         MVC    W_CESD_NAME(8),CESD_NAME
         NC     W_CESD_OFFS(1),=X'ØØ'
         MVC    W_CESD_OFFS+1(3),CESD_OFFS
         NC     W_CESD_SIZE(1),=X'ØØ'
         MVC    W_CESD_SIZE+1(3),CESD_SIZE
         L      2,W_CESD_OFFS
         A      2,W_CESD_SIZE
         SH     2,=H'1'
         ST     2,W_CESD_OFFE
         MVC    WOFFSET(4),W_CESD_OFFS
         MVC    WOFFSETEND(4),W_CESD_OFFE
         B      Ø(14)
*********************************************************************
*        Disassembling
*********************************************************************
DISASEMBLING DS  ØF
```

```
         STM    Ø,15,SAVDIS
*——— Starting address for disassembly ————————————-
         L      2,WOFFSET
         A      2,W@PGM
         ST     2,W@IT
         ST     2,W@IP
*——— Printing information about CSECT ————————————
         LA     1,SOURCE
         BAL    14,PRINT_CESD_INFO
         LA     1,SYSPRINT
         BAL    14,PRINT_CESD_INFO
*——— Printing of the header —————————————————-
         LA     Ø,79
         LA     1,MSGØ3
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
         LA     Ø,77
         LA     1,MSGØ4
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
         MVC    MSGØ5(8),W_CESD_NAME
         LA     Ø,L'MSGØ5
         LA     1,MSGØ5
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
*——— Start of disassembly ————————————————————-
         SR     15,15
         STH    15,WMISIZE
         SPLEVEL SET=4
*——— Loop of disassembly —————————————————————-
CIKLUS   EQU    *
         L      1,W@IT           ADDRESS OF CURRENT INSTRUCTION TO R1
         O      1,WPGMMODE
         L      2,W@PGM
         O      2,WPGMMODE
         SR     1,2
         ST     1,WOFFSET
*——— Checking the end of the load module ————————————-
         CLC    WOFFSET,WPGMSIZE
         BNL    END_DIS1
         CLC    WOFFSET,WOFFSETEND
         BH     END_DIS1
NOT_END EQU     *
*——— Extract inf. of the machine instruction ———————————
         BAL    14,EXTRACT_INSTR
*——— Disassembly of the machine instruction ———————————
         BAL    14,DISASSEMBLY_INSTR
         B      CIKLUS
*——— End of disassembly ——————————————————————
END_DIS1 EQU    *
         CLI    WNOT_INSTR,C'Y'
         BNE    END_DIS2
         L      1,W@PGM
```

```
         A      1,WOFFSETEND
         AH     1,=H'1'
         ST     1,W@IT
         BAL    14,PRINTZON
*—— Printing of END instruction —————————————————
END_DIS2 EQU *
         LA     Ø,L'MSGØ6
         LA     1,MSGØ6
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
         LM     Ø,15,SAVDIS
         B      Ø(14)
********************** Load program ***********************************
* Parameters:
* wpgmname program name to load
* Return:
* @Ø  address of the load module
* @1  length of the load module
***********************************************************************
LOADPGM  DS ØF
         ST     14,SAVWORK
         LOAD   EPLOC=WPGMNAME,ERRET=ERRORLOAD
         SLL    1,8
         SRL    1,5
         L      14,SAVWORK
         B      Ø(14)
*—— Error in load ——————————————————————
ERRORLOAD  EQU     *
         CVD    1,WPARM                   CONVERT ABEND CODE IN CHAR
         UNPK   MSGØ8+42(5),WPARM
         OI     MSGØ8+46,X'FØ'
         CVD    15,WPARM                  CONVERT REASON CODE IN CHAR
         UNPK   MSGØ8+55(5),WPARM
         OI     MSGØ8+59,X'FØ'
         MVC    MSGØ8+27(8),WPGMNAME
*—— Printing of error code during load statement ————————
         LA     Ø,59
         LA     1,MSGØ8
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         MVC    RET_CODE(4),=F'16'
         B      END
********************************************************************
*        Get parameters from SYSIN dataset
********************************************************************
GET_PARM DS     ØF
         STM    Ø,15,SAVWORK
         SETAMODE 24
         GET    SYSIN
         LR     4,1
         SETAMODE 31
         SR     1,1
         ST     1,PARM_CESD_OFFS
```



```
         ST     1,PARM_CESD_OFFE
GET_PROGRAM_NAME EQU *
         CLC    Ø(8,4),=C'        '
         BE     PUT_ERROR_MSG
         MVC    PPGMNAME(8),Ø(4)
         LA     3,22
         MVC    MSGØØ+14(8),Ø(4)
GET_CESD_NAME   EQU *
         LA     4,9(4)
         CLC    Ø(8,4),=C'        '
         BE     GET_OFFSET1
         CLI    Ø(4),C' '
         BE     PUT_ERROR_MSG
         MVC    PARM_CESD_NAME(8),Ø(4)
         LA     3,38
         MVC    MSGØØ+3Ø(8),Ø(4)
GET_OFFSET1     EQU *
         LA     4,1Ø(4)
         CLC    Ø(8,4),=C'        '
         BE     GET_OFFSET2
         CLI    Ø(4),C' '
         BE     PUT_ERROR_MSG
         LA     3,59
         MVC    MSGØØ+49(8),Ø(4)
         LA     Ø,8
         LA     1,MSGØØ+5Ø
         BAL    14,CONVHB
         ST     Ø,PARM_CESD_OFFS
GET_OFFSET2     EQU *
         LA     4,1Ø(4)
         CLC    Ø(8,4),=C'        '
         BE     GET_END
         CLI    Ø(4),C' '
         BE     PUT_ERROR_MSG
         LA     3,69
         MVC    MSGØØ+61(8),Ø(4)
         LA     Ø,8
         LA     1,MSGØØ+62
         BAL    14,CONVHB
         ST     Ø,PARM_CESD_OFFE
GET_END         EQU *
         LR     Ø,3
         LA     1,MSGØØ
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         LM     Ø,15,SAVWORK
         B      Ø(14)
*—— Print message about error in parameters ——————————
PUT_ERROR_MSG    EQU *
         LA     Ø,L'MSGØ7
         LA     1,MSGØ7
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
```

```
         LA     Ø,67
         LA     1,MSGØ7A
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         LA     Ø,67
         LA     1,MSGØ7B
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         MVC    RET_CODE(4),=F'12'
         B      END
*********************************************************************
*        Printing of header
*********************************************************************
PRINT_HEADER DS ØF
         ST     14,SAVWORK
         LA     Ø,4
         LA     1,WPGMSIZE
         LA     2,PPGMSIZEH
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
*——— Preparing information about the program ————————
         MVC    MSGØ1+9(8),PPGMNAME
         MVC    MSGØ1+24(8),PPGMSIZEH
*——— Print information about program in SYSPRINT dataset ——
         LA     Ø,L'MSGØ1
         LA     1,MSGØ1
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
*——— Print information about program in SOURCE dataset ————
         LA     Ø,L'MSGØ1
         LA     1,MSGØ1
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
         L      14,SAVWORK
         B      Ø(14)
*********************************************************************
*        Print information about CSECT
*********************************************************************
PRINT_CESD_INFO DS ØF
         STM    Ø,15,SAVWORK
         LR     3,1
*——— Print of header in SOURCE dataset ——————————————
         MVC    MSGØ2+3(5),W_CESD_TYPE
         MVC    MSGØ2+9(8),W_CESD_NAME
         LA     Ø,4
         LA     1,W_CESD_SIZE
         LA     2,MSGØ2+24
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
         LA     Ø,4
         LA     1,WOFFSET
```

```
         LA     2,MSGØ2+43
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
         LA     Ø,4
         LA     1,WOFFSETEND
         LA     2,MSGØ2+53
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
         LA     Ø,62
         LA     1,MSGØ2
         LR     2,3
         BAL    14,PUT_TO_FILE
         LM     Ø,15,SAVWORK
         B      Ø(14)
*********************************************************************
*        Checking if the offset in specified boundaries
*********************************************************************
CHECK_OFFSET DS ØF
         STM    Ø,15,SAVWORK
         CLC    PARM_CESD_OFFS,=F'Ø'
         BE     CHECK_OFF_E1
         CLC    PARM_CESD_OFFS,W_CESD_OFFE
         BH     CHECK_OFF_E3
         MVC    WOFFSET,W_CESD_OFFS
         CLC    PARM_CESD_OFFS,W_CESD_OFFS
         BNH    CHECK_OFF_E1
         MVC    WOFFSET,PARM_CESD_OFFS
CHECK_OFF_E1  EQU *
         CLC    PARM_CESD_OFFE,=F'Ø'
         BE     CHECK_OFF_E2
         CLC    PARM_CESD_OFFE,W_CESD_OFFS
         BL     CHECK_OFF_E3
         MVC    WOFFSETEND,W_CESD_OFFE
         CLC    PARM_CESD_OFFE,W_CESD_OFFE
         BNL    CHECK_OFF_E2
         MVC    WOFFSETEND,PARM_CESD_OFFE
CHECK_OFF_E2  EQU *
         MVI    WSKIP,C'N'
CHECK_OFF_E3  EQU *
         LM     Ø,15,SAVWORK
         B      Ø(14)
******************** Put to SOURCE or SYSPRINT file *******************
*    Parameters:
*    @Ø  message length
*    @1  address of the message area
*    @2  address of DCB parameter of output dataset (SOURCE or SYSPRINT)
***********************************************************************
PUT_TO_FILE DS  ØF
         STM    Ø,15,SAVLOG
         MVI    LOGREC,C' '             SET LOGREC AREA TO BLANKS
         MVC    LOGREC+1(79),LOGREC
```

```
         LR     14,Ø                   STORING THE MESSAGE FROM ADDRESS
         BCTR   14,Ø                   FROM REG 1 IN LOG AREA
         EX     14,MVCLOG
*        STH    Ø,LOGBL
         SETAMODE 24
         PUT    (2),LOG
         SETAMODE 31
         LM     Ø,15,SAVLOG
         BSM    Ø,14
MVCLOG   MVC    LOGREC(Ø),Ø(1)
**********************************************************************
*        Separation of machine code
**********************************************************************
EXTRACT_INSTR DS ØF
         STM    Ø,15,SAVEXTRI             KEEP USED REGISTERS
         L      1,W@IT                    R1 - CURRENT INSTRUCTION
         LA     Ø,2                       RØ - LENGTH OF INSTRUCTIN
         LA     2,WMI                     R2 - ADDRESS OF WORKING AREA
         BAL    14,MOVE_MACHINE_INSTR
         TM     Ø(2),X'CØ'                CHECKING FIRST TWO BITS
         BZ     IZDV1                     IF THEY ARE Ø LENGTH IS 1
         BO     TRI                       IF THEY ARE 1 LRNGTH IS 3
         LA     Ø,4                       OTHERS LENGTH IS 2 HALFWORDS
         B      IZDV1
TRI      EQU    *
         LA     Ø,6                       LENGTH IS 3 HALFWORDS
IZDV1    EQU    *
         STH    Ø,WMISIZE                 SAVE LENGTH
         BAL    14,MOVE_MACHINE_INSTR
         LM     Ø,15,SAVEXTRI             RETURN OF REGISTER CONTENTS
         B      Ø(14)                     RETURN
**********************************************************************
*        Subroutine puts on memory parts form program to working area
*        parameters: rØ - length of area
*                    r1 - address of source area
*                    r2 - length of target area
**********************************************************************
MOVE_MACHINE_INSTR DS ØF
         STM    Ø,15,SAVWORK              SAVE REGISTERS THAT USED
         LA     15,MOVE_MACHINE_INSTRØ
         N      15,=X'7FFFFFFF'
         O      15,WPGMMODE
         BSM    Ø,15
MOVE_MACHINE_INSTRØ EQU *                 Separate machine code
         LR     15,Ø
         AR     15,1
         CL     15,W@ENDPGM
         BNL    MOVE_MACHINE_INSTR3
MOVE_MACHINE_INSTR1 EQU *
         N      1,=X'7FFFFFFF'
         O      1,WPGMMODE
         LR     15,Ø
         LTR    15,15
```



```
         BZ      MOVE_MACHINE_INSTR2
         BCTR    15,Ø               R2 - LENGTH OF INSTR. 1 HALFW
MOVE_MACHINE_INSTR2 EQU *
         EX      15,MVCPREB
MOVE_MACHINE_INSTR3 EQU *
         SETAMODE 31
         LM      Ø,15,SAVWORK
         BSM     Ø,14
MVCPREB  MVC     Ø(Ø,2),Ø(1)
***********************************************************************
*        Disassembling of machine instruction
***********************************************************************
DISASSEMBLY_INSTR DS ØF
         STM     Ø,15,SAVDISI       SAVE REGISTERS TO BE USED
         SR      5,5
         IC      5,WMI              GET FIRST FOUR BITS OF MACHINE CODE
         SRL     5,4
         SLL     5,2                MULTIPLY WITH 4 TO FIND THE OFFSET
         L       1,W@INST           R1 - ADDRESS OF ADINSTR AREA
         L       5,Ø(5,1)           R4 - START OF THE INSTRUCTION AREA
         SR      2,2
DISNOVA  EQU     *
         CL      5,WENDINST         IF END OF THE TABLE REACHED,
         BH      DIS_NO_INSTRØ      STOP SEARCHING
         CLI     Ø(5),X'FF'
         BE      DIS_NO_INSTRØ
         IC      2,2(5)             GET LENGTH OF THE CODE
         LTR     2,2
         BZ      DIS_NO_INSTRØ
         BCTR    2,Ø                DECREASE BY 1 BECAUSE OF EX
         EX      2,CLCKOD           COMPARE WITH MACHINE CODE
         BH      DIS_NO_INSTRØ      IF IT IS GRATER STOP SEARCHING,
         BE      DISOK              BECAUSE CODES ARE SORTED IN ASC ORD.
         LA      5,12(5)            IF IT IS NOT JUMP TO NEXT CODE
         B       DISNOVA
CLCKOD   CLC     Ø(Ø,5),WMI
DISOK    EQU     *                  IF IT IS FOUND
         L       2,8(5)             R2 - ADDRESS OF THE INSTRUCTION FORM
         ST      2,W@FORM
*---- If not instruction ---------------------
         CLI     WNOT_INSTR,C'Y'
         BNE     KONVMI
*---- Printing of the area that is not code in hex format -----
         BAL     14,PRINTZON
         MVC     W@IP,W@IT          CURRENT ADDRESS BECOMES PREVIOUS
         MVI     WNOT_INSTR,C'N'
*---- Conversion of machine instruction to hex format --------
KONVMI   EQU     *                  IF IT IS FOUND
         MVI     WNOT_INSTR,C'N'
         LH      Ø,WMISIZE
         LA      1,WMI
         LA      2,WMIHEX
         STM     Ø,2,WPARM
```

```
         LA     1,WPARM
         BAL    14,CONVBH
*——— If instruction ———————————————————————
COPY_INSTR EQU *
         LR     1,Ø
         BCTR   1,Ø
         EX     1,MVCWMI
         MVC    PPAI(5),3(5)
         CLI    WMI,X'ØA'           IF IT IS SVC
         BNE    DIS_NO_SVC
*——— Disassembly of SVC ————————————————
         SR     Ø,Ø
         IC     Ø,WMI+1
         LR     1,Ø
*——— Converting from binary to decimal format ——————————
         CVD    Ø,WPARM
         UNPK   WNUMDEC(8),WPARM
         OI     WNUMDEC+7,X'FØ'
         MVC    PPAI+7(3),WNUMDEC+5
         L      2,W@SVC
         SLL    1,3
         LA     2,4(1,2)
         CL     2,WENDSVC
         BH     DIS_SVC1
         MVC    PPAIC+1(8),Ø(2)
DIS_SVC1 EQU    *
         SR     Ø,Ø
         SR     3,3
         B      DISASE_PUT_KOD
*——— Disassembly of instructions based on parameters ———————
*———— for disassembly from AINSTR ————————————————
DIS_NO_SVC EQU  *
         L      2,W@FORM
         LA     5,4(2)
         L      2,Ø(2)
         SR     3,3
         IC     3,Ø(2)
         BCTR   3,Ø
         EX     3,MVCKOD
DIS_ALL_TYPE EQU *
         CLI    Ø(5),X'FF'            END OF THE INSTR. FOR DISASSEMBLY?
         BE     DISASE_PUT_KOD        IF YES, THE RESULT IS PRINTED
*——— Preparing parameters for converting form hex to decimal ——
         SR     15,15
         IC     15,Ø(5)
         BCTR   15,Ø
         LA     1,WMIHEX
         AR     1,15
         SR     2,2
         IC     2,1(5)
         SR     15,15
         IC     15,2(5)
         LA     3,PPAI+6
```



```
         AR     3,15
         SR     4,4
         IC     4,3(5)
         CLI    4(5),C'D'
         BE     CONV_HEX_DEC
         BCTR   2,Ø
         EX     2,MVCHEX
         B      NEXT_CONV
MVCHEX   MVC    Ø(Ø,3),Ø(1)
CONV_HEX_DEC    EQU *
         LA     15,CONVHD
         CALL   (15),((2),(1),(4),(3))
NEXT_CONV       EQU *
         LA     5,5(5)
         B      DIS_ALL_TYPE
*——— Call the subroutine for printing disassembled code —————
DISASE_PUT_KOD EQU *
         BAL    14,PUT_ASSEMBLER_KOD
         L      1,W@IT
         AH     1,WMISIZE
         ST     1,W@IT
         MVC    W@IP,W@IT            CURRENT ADDRESS BECOME PREVIOUS
         B      DIS_END
*——— If it is not instruction, it is data ——————————————
DIS_NO_INSTRØ EQU *
         MVI    WNOT_INSTR,C'Y'
         L      1,W@IT
         A      1,=F'2'
         ST     1,W@IT
DIS_END EQU     *
         LM     Ø,15,SAVDISI
         B      Ø(14)                RETURN
MVCKOD   MVC    PPAI+7(Ø),1(2)
MVCWMI   MVC    PPAIC+1(Ø),WMI
****************************************************
* Subroutine convert area from hex to binary code
* parameters:
*   @r1    - Length of input area
*   @r1+4  - Address of input area
*   @r1+8  - Address of output area
****************************************************
CONVBH   DS     ØF
         STM    Ø,15,SAVCONVH
******************************************************************
         L      3,Ø(1)               DU|INA U R3
         L      2,4(1)               A(ULAZ) U R2
         L      1,8(1)               A(IZLAZ) U R1
         SR     Ø,Ø
CONVBH1  EQU *
         O      2,WPGMMODE
         IC     Ø,Ø(2)               PRVA 4 BAJTA U RØ
         STC    Ø,1(1)
         SRL    Ø,4
```

```
         STC    Ø,Ø(1)
         NI     1(1),X'ØF'
         NI     Ø(1),X'ØF'
         TR     Ø(2,1),TABELA
         AH     1,=H'2'
         LA     2,1(2)                  R2=R2+1
         SH     3,=H'1'                 R3=R3-1
         BP     CONVBH1
ENDCONVBH EQU    *
         LM     Ø,15,SAVCONVH
         B      Ø(14)
TABELA   DC     CL16'Ø123456789ABCDEF'
***************************************************
* Subroutine convert area from hex to binary code
* parameters:
*   @rØ  - length of input area
*   @r1  - address of input area
* output:
*    rØ  - result
***************************************************
CONVHB   DS     ØF
         STM    Ø,15,SAVCONVH
         LR     2,Ø
         SR     Ø,Ø
         SR     15,15
CONVHØ   EQU    *
         IC     15,Ø(1)
         CLI    Ø(1),X'FØ'
         BL     CONVHCHA
         SH     15,=H'24Ø'
         B      CONVHNEX
CONVHCHA EQU    *
         SH     15,=H'183'
CONVHNEX EQU    *
         BM     ENDCONVH
         CH     15,=H'16'
         BH     ENDCONVH
         SLL    Ø,4
         AR     Ø,15
         LA     1,1(1)
         BCT    2,CONVHØ
ENDCONVH EQU *
         LM     1,15,SAVCONVH+4
         B      Ø(14)
***********************************************************************
*        SUBROUTINE CONVERTS HEX AREA TO DECIMAL
*        PARAMETERS:
*          @R1    - LENGTH OF INPUT AREA
*          @R1+4  - ADDRESS OF INPUT AREA
*          @R1+8  - LENGTH OF OUTPUT AREA
*          @R1+12 - ADDRESS OF OUTPUT AREA
***********************************************************************
CONVHD   DS     ØF
```



```
         STM    Ø,15,SAVCONVH
*------------------------------
         L      3,Ø(1)            R3  - LENGTH OF INPUT AREA
         L      2,4(1)            R2  - ADDRESS OF INPUT AREA
         L      1Ø,8(1)           R1Ø - LENGTH OF OUTPUT AREA
         L      4,12(1)           R4  - ADDRESS OF OUTPUT AREA
         LA     15,12             RC=12
         CH     1Ø,=H'16'         IF LENGTH(OUTPUT) > 16
         BH     ENDCONVHD         THEN GOTO END
         SR     15,15             RC=Ø
         SR     5,5
         AR     2,3
         BCTR   2,Ø
         LA     7,1
CONVHDØ  EQU    *
         LTR    3,3
         BZ     CONVHNE
         SR     9,9
         IC     9,Ø(2)
         SH     9,=H'24Ø'
         BNM    CONVHNM
         AH     9,=H'57'
CONVHNM  EQU    *
         MR     8,7
         SLL    7,4
         AR     5,9
         BCTR   2,Ø
         BCTR   3,Ø
         B      CONVHDØ
CONVHNE  EQU    *
         CVD    5,WPARM
         UNPK   WNUMDEC(16),WPARM
         OI     WNUMDEC+15,X'FØ'
*--- COPYING DECIMAL NUMBER FROM WORKING AREA TO OUTPUT ----
         LA     6,WNUMDEC
         LA     8,16
         SR     8,1Ø
         AR     6,8
         BCTR   1Ø,Ø
         EX     1Ø,MVC1
*--- EPILOQUE ---------------------------
ENDCONVHD EQU *
         LM     Ø,15,SAVCONVH
         B      Ø(14)
MVC1     MVC    Ø(Ø,4),Ø(6)
**********************************************************************
*      Subroutine print area that does not contain Assembler instruction
**********************************************************************
PRINTZON DS     ØF
         STM    Ø,15,SAVPRINT
         SR     4,4
         L      5,W@IT
         S      5,W@IP
```

```
         L      6,W@IP
         LA     9,8
NOVAZONA EQU    *
         CR     5,9
         BNL    PRINTZ1
         LR     9,5
*—— Convert area from binary to hex format ———————————
PRINTZ1  EQU    *
         LR     Ø,9
         LR     1,6
         LA     2,WMIHEX
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
*—— Forms DC statement with hex data —————————————
         LR     7,9
         BCTR   7,Ø
         EX     7,MVCZON
         MVC    PPAI(7),=C'  DC X'''
         MVC    PPAI+7(4),WMIHEX
         MVC    PPAI+11(4),WMIHEX+4
         MVC    PPAI+15(4),WMIHEX+8
         MVC    PPAI+19(4),WMIHEX+12
         LA     8,PPAI+7
         AR     8,9
         AR     8,9
         MVI    Ø(8),C''''
*—— Print of DC statement ————————————————————
         BAL    14,PUT_ASSEMBLER_KOD
         AR     6,9
         ST     6,W@IP
         SR     5,9
         LTR    5,5
         BNZ    NOVAZONA
         LM     Ø,15,SAVPRINT        RESTORE REGISTER CONTENTS
         B      Ø(14)                RETURN
MVCZON   MVC    PPAIC+1(Ø),Ø(6)
***********************************************************************
PUT_ASSEMBLER_KOD DS ØF
         STM    Ø,15,SAVWRITE
*—— Set output area to blanks ———————————————————
         MVI    WORKCH,C' '
         MVC    WORKCH+1(L'WORKCH-1),WORKCH
         LA     4,WORKCH
         USING  AD#INST,4
*—— Convert offset from binary to hex format ——————————
         L      Ø,W@IP
         L      1,W@PGM
         SR     Ø,1
         ST     Ø,WOFFSET
         LA     Ø,4
         LA     1,WOFFSET
         LA     2,POFFCH
```



```
         STM    Ø,2,WPARM
         LA     1,WPARM
         BAL    14,CONVBH
*—— Convert offset from hex to decimal format ——————————
         LA     1,POFFCH+4
         LA     2,AD#OFFDEC
         LA     15,CONVHD
         CALL   (15),(4,(1),5,(2))
*——— Fill of output record  ———————————————————-
         MVI    AD#LABEL,C'L'
         MVC    AD#OFFHEX(8),POFFCH
         MVC    AD#A(3Ø),PPAI
         MVI    AD#AST1,C'*'
         MVC    AD#HEX1(4),WMIHEX
         MVC    AD#HEX2(4),WMIHEX+4
         MVC    AD#HEX3(4),WMIHEX+8
         MVC    AD#HEX4(4),WMIHEX+12
         MVI    AD#AST2,C'*'
         MVC    AD#ASSEMC(9),PPAIC
*—— Print of output record ———————————————————
         LA     Ø,8Ø
         LR     1,4
         LA     2,SOURCE
         BAL    14,PUT_TO_FILE
*—— Set work area to blanks ———————————————————
         MVI    WORKCH,C' '
         MVC    WORKCH+1(PEND-WORKCH-1),WORKCH
         LM     Ø,15,SAVWRITE
         B      Ø(14)
********************** ESPIE exit *************************************
ESPIEEXT DS ØF
         L   1,4(13)
         L   14,SAV14
         ST  14,12(1)
         LA     Ø,L'MSGØ9
         LA     1,MSGØ9
         LA     2,SYSPRINT
         BAL    14,PUT_TO_FILE
         B      END
SAV14    DS     F
********************** Declaration of datasets ************************
STEPLIB    DCB DDNAME=STEPLIB,DSORG=PO,MACRF=R,EODAD=END
SYSPRINT   DCB DDNAME=SYSPRINT,DSORG=PS,MACRF=PM,RECFM=FB,             X
               LRECL=8Ø,BLKSIZE=616Ø
SOURCE     DCB DDNAME=SOURCE,DSORG=PS,MACRF=PM,RECFM=FB,               X
               LRECL=8Ø,BLKSIZE=616Ø
SYSIN      DCB DDNAME=SYSIN,DSORG=PS,MACRF=GL,EODAD=END
************************** Constants *********************************
CESD     DC    XL2'2Ø8Ø'
CSECD    DC    CL2'SD'
CLABD    DC    CL2'LD'
*************************** Messages *********************************
MSGØØ    DC C' PARM PROGRAM=XXXXXXXX  CSECT=         '
```

```
         DC C' OFFSETS=X         - X         '
MSGØØA   DC C'******* LIST CESD ENTRY ********'
MSGØØB   DC C'******** DISASSEMBLING *********'
MSGØ1    DC C'*PROGRAM=XXXXXXXX,SIZE=XØØØØØØØ'
MSGØ2    DC C'*       =XXXXXXXX,SIZE=XØØØØØØØ'
         DC C',OFFSETS=(XØØØØØØØ-XØØØØØØØ)'
MSGØ3    DC C'*OFFSET  SOURCE STATEMENT                   OBJECT     '
         DC C'      OBJECT    OFFSET'
MSGØ4    DC C'* DEC                                      CODE HEX    '
         DC C'     CODE CHAR   HEX'
MSGØ5    DC C'XXXXXXXX  CSECT'
MSGØ6    DC C'          END'
MSGØ7    DC C'*** WRONG INPUT PARM'
MSGØ7A   DC C'*** ENTER  1- 8 PROGRM NAME (REQUIRED),'
         DC C' 1Ø-18 CSECT NAME (OPTION),'
MSGØ7B   DC C'          2Ø-28 START OFFSET  (OPTION),'
         DC C' 3Ø-38 END  OFFSET (OPTION)'
MSGØ8    DC C' *** ERROR IN LOAD PROGRAM=XXXXXXXX'
         DC C' ABEND=XXXXX REASON=XXXXX'
MSGØ9    DC C' *** END OF DISASSEMBLING'
           LTORG
WORK       DSECT
SAV        DS     18F
SAVPRINT   DS     16F
SAVWRITE   DS     16F
SAVRCESD   DS     16F
SAVWORK    DS     16F
SAVCONVH   DS     16F
SAVEXTRI   DS     16F
SAVDISPGM  DS     16F
SAVDIS     DS     16F
SAVDISI    DS     16F
SAVLOG     DS     16F
W@FORM     DS     F
W@SVC      DS     F
WENDSVC    DS     F
W@INST     DS     F
W@IT       DS     F
W@IP       DS     F
WENDINST   DS     F
WPARM      DS     2D
W@CESDG    DS     F
W@CESDP    DS     F
W_CESD_TYPE DS    CL5
W_CESD_NAME DS    CL8
W_CESD_SIZE DS    F
W_CESD_OFFS DS    F
W_CESD_OFFE DS    F
WMISIZE    DS     H
WMI        DS     CL6
WNOT_INSTR DS     CL1
WSKIP      DS     CL1
WNUMDEC    DS     CL16
```



```
WPGMMODE   DS     F
W@PGM      DS     F
WPGMSIZE   DS     F
W@ENDPGM   DS     F
WOFFSET    DS     F
WOFFSETEND DS     F
WPGMNAME   DS     CL8
PARM_CESD_OFFS DS F
PARM_CESD_OFFE DS F
PPGMNAME   DS     CL8
PPGMSIZEH  DS     CL8
PARM_CESD_NAME DS CL8
WORKCH     DS     CL84
           DS     ØF
POFFCH     DS     CL8
WMIHEX     DS     CL32
PPAI       DS     CL3Ø
PPAIC      DS     CL17
PEND       EQU    *
*———————————— Structure of log ———————————*
LOG        DS     ØF
LOGREC     DS     CL8Ø
LOGSIZE    EQU    *-LOG
WRKSIZE    EQU    *-WORK
AD#INST    DSECT
AD#LABEL   DS     CL1
AD#OFFDEC  DS     CL7
AD#A       DS     CL32
AD#AST1    DS     CL1
AD#HEX1    DS     CL5
AD#HEX2    DS     CL5
AD#HEX3    DS     CL5
AD#HEX4    DS     CL5
AD#AST2    DS     CL1
AD#ASSEMC  DS     CL1Ø
AD#OFFHEX  DS     CL8
AD#SIZE    EQU    *-AD#INST
*          INTERNAL CESD LIST
CESD_LIST  DSECT
CESD_NEXT  DS    F                  PTR TO NEXT CESD
CESD_TYPE  DS    CL5
CESD_NAME  DS    CL8
CESD_TYPEH DS    CL1
CESD_OFFS  DS    CL3
CESD_MODE  DS    CL1
CESD_SIZE  DS    CL3
CESDSIZE   EQU    *-CESD_LIST
           END
```

Module AINSTR contains instructions from current IBM publications. The format enables you to add new instructions and new formats. The syntax rules for adding are described in the comment lines.


```
AINSTR   CSECT
AINSTR   AMODE 31
AINSTR   RMODE ANY
***********************************************************************
*  LIST OF ASSEMBLER INSTRUCTIONS IN FORMAT:
*  DC  XL2'CODE',AL1(CODE LENGTH),A(FORMAT),CL5'MNEMONIC ABBREVIATION'
***********************************************************************
*  INDEX   OF ASSEMBLER INSTRUCTION
***********************************************************************
AHEXØ    DC  AL4(HEXØ)
AHEX1    DC  AL4(HEX1)
AHEX2    DC  AL4(HEX2)
AHEX3    DC  AL4(HEX3)
AHEX4    DC  AL4(HEX4)
AHEX5    DC  AL4(HEX5)
AHEX6    DC  AL4(HEX6)
AHEX7    DC  AL4(HEX7)
AHEX8    DC  AL4(HEX8)
AHEX9    DC  AL4(HEX9)
AHEXA    DC  AL4(HEXA)
AHEXB    DC  AL4(HEXB)
AHEXC    DC  AL4(HEXC)
AHEXD    DC  AL4(HEXD)
AHEXE    DC  AL4(HEXE)
AHEXF    DC  AL4(HEXF)
AFORM    DC  AL4(FORMATI)
***********************************************************************
* F_XXXX - FORMATS OF ASSEMBLER INSTRUCTIONS
***********************************************************************
* $XXXX - FORMATS OF MACHINE CODES
*  1   ADDRESS OF ASSEMBLER INSTRUCTION FORMAT
*  2   DATA CONVERSION FROM MACHINE FORMAT IN HEX TO ASSEMBLER FORMAT
*      IN DECIMAL
*  =>  FP,FL,TP,TL,ONF
*      FP  - from position
*      FL  - in length
*      TP  - to position
*      TL  - in length
*      ONF - output number format (D - DEC, H - HEX )
*  3   X'FF' - END OF INSTRUCTION FOR CONVERSION
***********************************************************************
FORMATI  DS  ØF
*---------- Ø--+--1--+--2--+----------
F_E      DC  YL1(Ø1),C' '                    Assembler   Format dec
*e    format          cccc                   Instruction Format hex
$E       DC  A(F_E)
         DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_R#     DC  YL1(Ø2),C'RR'                   Assembler   Format dec
*rr   format          ccR/                   Instruction Format hex
$R#      DC  A(F_R#)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D' Conversion instructions
         DC  X'FF'
```

```
*rre# format           cccc//R/                 Instruction Format hex
$RRE#    DC  A(F_R#)
         DC  AL1(Ø7),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RR     DC  YL1(Ø5),C'RR,RR'                   Assembler   Format dec
*rr   format           ccRR                     Instruction Format hex
$RR      DC  A(F_RR)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø4),AL1(1),AL1(Ø4),AL1(2),C'D'
         DC  X'FF'
*rre  format           cccc//RR                 Instruction Format hex
$RRE     DC  A(F_RR)
         DC  AL1(Ø7),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø8),AL1(1),AL1(Ø4),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RRF    DC  YL1(Ø8),C'RR,RR,RR'                Assembler   Format dec
*rrf  format           ccccR/RR                 Instruction Format hex
$RRF     DC  A(F_RR)
         DC  AL1(Ø7),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø5),AL1(1),AL1(Ø4),AL1(2),C'D'
         DC  AL1(Ø8),AL1(1),AL1(Ø7),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RX     DC  YL1(14),C'RR,DDDD(XX,BB)'
*rx   format           ccRXBDDD                 Instruction Format hex
$RX      DC  A(F_RX)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø6),AL1(3),AL1(Ø4),AL1(4),C'D'
         DC  AL1(Ø4),AL1(1),AL1(Ø9),AL1(2),C'D'
         DC  AL1(Ø5),AL1(1),AL1(12),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RS     DC  YL1(14),C'RR,RR,DDDD(BB)'
*rs   format           ccRRBDDD                 Instruction Format hex
$RS      DC  A(F_RS)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø4),AL1(1),AL1(Ø4),AL1(2),C'D'
         DC  AL1(Ø6),AL1(3),AL1(Ø7),AL1(4),C'D'
         DC  AL1(Ø5),AL1(1),AL1(12),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RS#    DC  YL1(11),C'RR,DDDD(BB)'
*rs#  format           ccR/BDDD                 Instruction Format hex
$RS#     DC  A(F_RS#)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø6),AL1(3),AL1(Ø4),AL1(4),C'D'
         DC  AL1(Ø5),AL1(1),AL1(Ø9),AL1(2),C'D'
         DC  X'FF'
*———————————— Ø——+——1——+——2——+ ————————————-
F_RI     DC  YL1(11),C'RR,IIIII'
*ri   format           ccRcIIII                 Instruction Format hex
```

```
$RI       DC  A(F_RI)
          DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
          DC  AL1(Ø5),AL1(4),AL1(Ø4),AL1(5),C'D'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_RSI     DC  YL1(11),C'RR,RR,IIIII'
*rsi  format           ccRRIIII                 Instruction Format hex
$RSI      DC  A(F_RSI)
          DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
          DC  AL1(Ø4),AL1(1),AL1(Ø4),AL1(2),C'D'
          DC  AL1(Ø5),AL1(4),AL1(Ø7),AL1(5),C'D'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_S       DC  YL1(8),C'DDDD(BB)'
*s    format           ccccBDDD                 Instruction Format hex
$S        DC  A(F_S)
          DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
          DC  AL1(Ø5),AL1(1),AL1(Ø6),AL1(2),C'D'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_SI      DC  YL1(14),C'DDDD(BB),X''II'''
*si   format           ccIIBDDD                 Instruction Format hex
$SI       DC  A(F_SI)
          DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
          DC  AL1(Ø5),AL1(1),AL1(Ø6),AL1(2),C'D'
          DC  AL1(Ø3),AL1(2),AL1(12),AL1(2),C'H'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_SS1     DC  YL1(21),C'DDDD(LLL,BB),DDDD(BB)'
*ss1  format           ccLLBDDDBDDD             Instruction Format hex
$SS1      DC  A(F_SS1)
          DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
          DC  AL1(Ø3),AL1(2),AL1(Ø6),AL1(3),C'D'
          DC  AL1(Ø5),AL1(1),AL1(1Ø),AL1(2),C'D'
          DC  AL1(1Ø),AL1(3),AL1(14),AL1(4),C'D'
          DC  AL1(Ø9),AL1(1),AL1(19),AL1(2),C'D'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_SS2     DC  YL1(23),C'DDDD(RR,BB),DDDD(BB),RR'
*ss2  format           ccRRBDDDBDDD             Instruction Format hex
$SS2      DC  A(F_SS2)
          DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
          DC  AL1(Ø3),AL1(1),AL1(Ø6),AL1(2),C'D'
          DC  AL1(Ø5),AL1(1),AL1(Ø9),AL1(2),C'D'
          DC  AL1(1Ø),AL1(3),AL1(13),AL1(4),C'D'
          DC  AL1(Ø9),AL1(1),AL1(18),AL1(2),C'D'
          DC  AL1(Ø4),AL1(1),AL1(22),AL1(2),C'D'
          DC  X'FF'
*---------- Ø--+--1--+--2--+ -----------
F_SS3     DC  YL1(23),C'DDDD(LL,BB),DDDD(LL,BB)'
*ss3  format           ccLLBDDDBDDD             Instruction Format hex
$SS3      DC  A(F_SS3)
          DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
```

```
         DC  AL1(Ø3),AL1(1),AL1(Ø6),AL1(2),C'D'
         DC  AL1(Ø5),AL1(1),AL1(Ø9),AL1(2),C'D'
         DC  AL1(1Ø),AL1(3),AL1(13),AL1(4),C'D'
         DC  AL1(Ø4),AL1(1),AL1(18),AL1(2),C'D'
         DC  AL1(Ø9),AL1(1),AL1(21),AL1(2),C'D'
         DC  X'FF'
*——————————— Ø——+——1——+——2——+ ———————————-
F_SS4    DC  YL1(23),C'RR,DDDD(BB),RR,DDDD(BB)'
*ss4  format           ccRRBDDDBDDD             Instruction Format hex
$SS4     DC  A(F_SS4)
         DC  AL1(Ø3),AL1(1),AL1(Ø1),AL1(2),C'D'
         DC  AL1(Ø6),AL1(3),AL1(Ø4),AL1(4),C'D'
         DC  AL1(Ø5),AL1(1),AL1(Ø9),AL1(2),C'D'
         DC  AL1(Ø4),AL1(1),AL1(13),AL1(2),C'D'
         DC  AL1(1Ø),AL1(3),AL1(16),AL1(4),C'D'
         DC  AL1(Ø9),AL1(1),AL1(21),AL1(2),C'D'
         DC  X'FF'
*——————————— Ø——+——1——+——2——+ ———————————-
F_SSE    DC  YL1(17),C'DDDD(BB),DDDD(BB)'
*sse  format           ccccBDDDBDDD             Instruction Format hex
$SSE     DC  A(F_SSE)
         DC  AL1(Ø6),AL1(3),AL1(Ø1),AL1(4),C'D'
         DC  AL1(Ø5),AL1(1),AL1(Ø6),AL1(2),C'D'
         DC  AL1(1Ø),AL1(3),AL1(1Ø),AL1(4),C'D'
         DC  AL1(Ø9),AL1(1),AL1(15),AL1(2),C'D'
         DC  X'FF'
**********************************************************************
*        ASSEMBLER INSTRUCTION
**********************************************************************
HEXØ     DS  ØF  *****************************************************
PR       DC  XL2'Ø1Ø1',AL1(2),CL5'PR',A($E) E PROGRAM RERURN
UPT      DC  XL2'Ø1Ø2',AL1(1),CL5'UPT',A($E) E UPDATE TREE
SCKPF    DC  XL2'Ø1Ø7',AL1(2),CL5'SCKPF',A($E) E SET CLOCK PROG. FIE
TRAP2    DC  XL2'Ø1FF',AL1(2),CL5'TRAP2',A($E) E TRAP
SPM      DC  XL2'Ø4ØØ',AL1(1),CL5'SPM',A($R#) RR SET PROGRAM MASK
BALR     DC  XL2'Ø5ØØ',AL1(1),CL5'BALR',A($RR) RR BRANCH AND LINK
BCTR     DC  XL2'Ø6ØØ',AL1(1),CL5'BCTR',A($RR) RR BRANCH ON COUNT
BCR      DC  XL2'Ø7ØØ',AL1(1),CL5'BCR',A($RR) RR BRANCH ON CONDITION
SSK      DC  XL2'Ø8ØØ',AL1(1),CL5'SSK',A($RR) RR SET STORAGE KEY
ISK      DC  XL2'Ø9ØØ',AL1(1),CL5'ISK',A($RR) RR INSERT STOREGE KEY
SVC      DC  XL2'ØAØØ',AL1(1),CL5'SVC',A($R#) RR SUPERVISOR CALL
BSM      DC  XL2'ØBØØ',AL1(1),CL5'BSM',A($RR) RR BRANCH AND SET
BASR     DC  XL2'ØDØØ',AL1(1),CL5'BASR',A($RR) RR BRANCH AND SAVE
MVCL     DC  XL2'ØEØØ',AL1(1),CL5'MVCL',A($RR) RR MOVE LONG
CLCL     DC  XL2'ØFØØ',AL1(1),CL5'CLCL',A($RR) RR COMPARE LOGICAL
HEX1     DS  ØF *****************************************************
LPR      DC  XL2'1ØØØ',AL1(1),CL5'LPR',A($RR) RR LOAD POSITIVE
LNR      DC  XL2'11ØØ',AL1(1),CL5'LNR',A($RR) RR LOAD NEGATIVE
LTR      DC  XL2'12ØØ',AL1(1),CL5'LTR',A($RR) RR LOAD AND TEST
LCR      DC  XL2'13ØØ',AL1(1),CL5'LCR',A($RR) RR LOAD COMPLE$ENT
NR       DC  XL2'14ØØ',AL1(1),CL5'NR',A($RR) RR AND
CLR      DC  XL2'15ØØ',AL1(1),CL5'CLR',A($RR) RR COMPARE LOGICAL
OR       DC  XL2'16ØØ',AL1(1),CL5'OR',A($RR) RR OR
```

```
XR       DC  XL2'17ØØ',AL1(1),CL5'XR',A($RR) RR EXCLUSIVE OR
LR       DC  XL2'18ØØ',AL1(1),CL5'LR',A($RR) RR LOAD
CR       DC  XL2'19ØØ',AL1(1),CL5'CR',A($RR) RR COMPARE
AR       DC  XL2'1AØØ',AL1(1),CL5'AR',A($RR) RR ADD
SR       DC  XL2'1BØØ',AL1(1),CL5'SR',A($RR) RR SUBTRACT
MR       DC  XL2'1CØØ',AL1(1),CL5'$R',A($RR) RR MULTIPLAY
DR       DC  XL2'1DØØ',AL1(1),CL5'DR',A($RR) RR DIVIDE
ALR      DC  XL2'1EØØ',AL1(1),CL5'ALR',A($RR) RR ADD LOGICAL
SLR      DC  XL2'1FØØ',AL1(1),CL5'SLR',A($RR) RR SUBTRACT LOGICAL
HEX2     DS  ØF **************************************************
LPDR     DC  XL2'2ØØØ',AL1(1),CL5'LPDR',A($RR) RR LOAD POSITIVE
LNDR     DC  XL2'21ØØ',AL1(1),CL5'LNDR',A($RR) RR LOAD NEGATIVE
LTDR     DC  XL2'22ØØ',AL1(1),CL5'LTDR',A($RR) RR LOAD AND TEST
LCDR     DC  XL2'23ØØ',AL1(1),CL5'LCDR',A($RR) RR LOAD COMPLE$ENT
HDR      DC  XL2'24ØØ',AL1(1),CL5'HDR',A($RR) RR HALVE
LRDR     DC  XL2'25ØØ',AL1(1),CL5'LRDR',A($RR) RR LOAD ROUNDED (EXT.
MXR      DC  XL2'26ØØ',AL1(1),CL5'MXR',A($RR) RR MULTIPLAY
MXDR     DC  XL2'27ØØ',AL1(1),CL5'MXDR',A($RR) RR MULTIPLAY
LDR      DC  XL2'28ØØ',AL1(1),CL5'LDR',A($RR) RR LOAD
CDR      DC  XL2'29ØØ',AL1(1),CL5'CDR',A($RR) RR COMPARE
ADR      DC  XL2'2AØØ',AL1(1),CL5'ADR',A($RR) RR ADD NORMALIZED
SDR      DC  XL2'2BØØ',AL1(1),CL5'SDR',A($RR) RR SUBTRACT NORMALIZED
MDR      DC  XL2'2CØØ',AL1(1),CL5'MDR',A($RR) RR MULTIPLAY
DDR      DC  XL2'2DØØ',AL1(1),CL5'DDR',A($RR) RR DIVIDE
AWR      DC  XL2'2EØØ',AL1(1),CL5'AWR',A($RR) RR ADD UNNORMALIZED
SWR      DC  XL2'2FØØ',AL1(1),CL5'SWR',A($RR) RR SUBTRACT   UNNORMAL.
HEX3     DS  ØF **************************************************
LPER     DC  XL2'3ØØØ',AL1(1),CL5'LPER',A($RR) RR LOAD POSITIVE
LNER     DC  XL2'31ØØ',AL1(1),CL5'LNER',A($RR) RR LOAD NEGATIVE
LTER     DC  XL2'32ØØ',AL1(1),CL5'LTER',A($RR) RR LOAD AND TEST
LCER     DC  XL2'33ØØ',AL1(1),CL5'LCER',A($RR) RR LOAD COMPLE$ENT
HER      DC  XL2'34ØØ',AL1(1),CL5'HER',A($RR) RR HALVE
LRER     DC  XL2'35ØØ',AL1(1),CL5'LRER',A($RR) RR LOAD ROUNDED
AXR      DC  XL2'36ØØ',AL1(1),CL5'AXR',A($RR) RR ADD NORMALIZED
SXR      DC  XL2'37ØØ',AL1(1),CL5'SXR',A($RR) RR SUBTRACT NORMALIZED
LER      DC  XL2'38ØØ',AL1(1),CL5'LER',A($RR) RR LOAD
CER      DC  XL2'39ØØ',AL1(1),CL5'CER',A($RR) RR COMPARE
AER      DC  XL2'3AØØ',AL1(1),CL5'AER',A($RR) RR ADD NORMALIZED
SER      DC  XL2'3BØØ',AL1(1),CL5'SER',A($RR) RR SUBTRACT NORMALIZED
MER      DC  XL2'3CØØ',AL1(1),CL5'$ER',A($RR) RR MULTIPLAY
DER      DC  XL2'3DØØ',AL1(1),CL5'DER',A($RR) RR DIVIDE
AUR      DC  XL2'3EØØ',AL1(1),CL5'AUR',A($RR) RR ADD UNNORMALIZED
SUR      DC  XL2'3FØØ',AL1(1),CL5'SUR',A($RR) RR SUBTRACT
HEX4     DS  ØF **************************************************
STH      DC  XL2'4ØØØ',AL1(1),CL5'STH',A($RX) RX STORE HALFWORD
LA       DC  XL2'41ØØ',AL1(1),CL5'LA',A($RX) RX LOAD ADDRESS
STC      DC  XL2'42ØØ',AL1(1),CL5'STC',A($RX) RX STORE CHARACTER
IC       DC  XL2'43ØØ',AL1(1),CL5'IC',A($RX) RX INSERT CHARACTER
EX       DC  XL2'44ØØ',AL1(1),CL5'EX',A($RX) RX EXECUTE
BAL      DC  XL2'45ØØ',AL1(1),CL5'BAL',A($RX) RX BRANCH AND LINK
BCT      DC  XL2'46ØØ',AL1(1),CL5'BCT',A($RX) RX BRANCH ON COUNT
BC       DC  XL2'47ØØ',AL1(1),CL5'BC',A($RX) RX BRANCH ON CONDITION
LH       DC  XL2'48ØØ',AL1(1),CL5'LH',A($RX) RX LOAD HALFWORD
```

```
CH       DC  XL2'49ØØ',AL1(1),CL5'CH',A($RX) RX COMPARE HALFWORD
AH       DC  XL2'4AØØ',AL1(1),CL5'AH',A($RX) RX ADD HALFWORD
SH       DC  XL2'4BØØ',AL1(1),CL5'SH',A($RX) RX SUBTRACT HALFWORD
MH       DC  XL2'4CØØ',AL1(1),CL5'MH',A($RX) RX MULTIPLAY HALFWORD
BAS      DC  XL2'4DØØ',AL1(1),CL5'BAS',A($RX) RX BRANCH AND SAVE
CVD      DC  XL2'4EØØ',AL1(1),CL5'CVD',A($RX) RX CONVERT TO DECIMAL
CVB      DC  XL2'4FØØ',AL1(1),CL5'CVB',A($RX) RX CONVERT TO BINARY
HEX5     DS  ØF ***********************************************
ST       DC  XL2'5ØØØ',AL1(1),CL5'ST',A($RX) RX STORE
LAE      DC  XL2'51ØØ',AL1(1),CL5'LAE',A($RX) RX LOAD ADDRESS EXTEND.
N        DC  XL2'54ØØ',AL1(1),CL5'N',A($RX) RX AND
CL       DC  XL2'55ØØ',AL1(1),CL5'CL',A($RX) RX COMPARE LOGICAL
O        DC  XL2'56ØØ',AL1(1),CL5'O',A($RX) RX OR
X        DC  XL2'57ØØ',AL1(1),CL5'X',A($RX) RX EXCLUSIVE OR
L        DC  XL2'58ØØ',AL1(1),CL5'L',A($RX) RX LOAD
C        DC  XL2'59ØØ',AL1(1),CL5'C',A($RX) RX COMPARE
A        DC  XL2'5AØØ',AL1(1),CL5'A',A($RX) RX ADD
S        DC  XL2'5BØØ',AL1(1),CL5'S',A($RX) RX SUBTRACT
M        DC  XL2'5CØØ',AL1(1),CL5'M',A($RX) RX MULTIPLAY
D        DC  XL2'5DØØ',AL1(1),CL5'D',A($RX) RX DIVIDE
AL       DC  XL2'5EØØ',AL1(1),CL5'AL',A($RX) RX ADD LOGICAL
SL       DC  XL2'5FØØ',AL1(1),CL5'SL',A($RX) RX SUBTRACT LOGICAL
HEX6     DS  ØF ***********************************************
STD      DC  XL2'6ØØØ',AL1(1),CL5'STD',A($RX) RX STORE
MXD      DC  XL2'67ØØ',AL1(1),CL5'MXD',A($RX) RX MULTIPLAY
LD       DC  XL2'68ØØ',AL1(1),CL5'LD',A($RX) RX LOAD
CD       DC  XL2'69ØØ',AL1(1),CL5'CD',A($RX) RX COMPARE
AD       DC  XL2'6AØØ',AL1(1),CL5'AD',A($RX) RX ADD NORMALIZED
SD       DC  XL2'6BØØ',AL1(1),CL5'SD',A($RX) RX SUBTRACT NORMALIZED
MD       DC  XL2'6CØØ',AL1(1),CL5'MD',A($RX) RX MULTIPLAY
DD       DC  XL2'6DØØ',AL1(1),CL5'DD',A($RX) RX DIVIDE
AW       DC  XL2'6EØØ',AL1(1),CL5'AW',A($RX) RX ADD UNNORMALIZED
SW       DC  XL2'6FØØ',AL1(1),CL5'SW',A($RX) RX SUBTRACT
HEX7     DS  ØF ***********************************************
STE      DC  XL2'7ØØØ',AL1(1),CL5'STE',A($RX) RX STORE
MS       DC  XL2'71ØØ',AL1(1),CL5'$S',A($RX) RX MULTIPLY SINGLE
LE       DC  XL2'78ØØ',AL1(1),CL5'LE',A($RX) RX LOAD
CE       DC  XL2'79ØØ',AL1(1),CL5'CE',A($RX) RX COMPARE
AE       DC  XL2'7AØØ',AL1(1),CL5'AE',A($RX) RX ADD NORMALIZED
SE       DC  XL2'7BØØ',AL1(1),CL5'SE',A($RX) RX SUBTRACT NORM
MDE      DC  XL2'7CØØ',AL1(1),CL5'MDE',A($RX) RX MULTIPLY SXS=L
ME       DC  XL2'7CØØ',AL1(1),CL5'$E',A($RX) RX MULTIPLAY
DE       DC  XL2'7DØØ',AL1(1),CL5'DE',A($RX) RX DIVIDE
AU       DC  XL2'7EØØ',AL1(1),CL5'AU',A($RX) RX ADD UNNORMALIZED
SU       DC  XL2'7FØØ',AL1(1),CL5'SU',A($RX) RX SUBTR. UNNORM
HEX8     DS  ØF ***********************************************
SSM      DC  XL2'8ØØØ',AL1(1),CL5'SSM',A($S) S SET SYSTEM MASK
LPSW     DC  XL2'82ØØ',AL1(1),CL5'LPSW',A($S) S LOAD PSW
DIAG     DC  XL2'83ØØ',AL1(1),CL5'DIAG',A(Ø) DIAGNOSE - NEMA SKRA?ENI
* Obsolete from OS37Ø/XA ->
*WRD      DC  XL2'84ØØ',AL1(1),CL5'WRD',A($SI) SI WRITE DIRECT
*RDD      DC  XL2'85ØØ',AL1(1),CL5'ROD',A($SI) SI READ DIRECT
BRXH     DC  XL2'84ØØ',AL1(1),CL5'BRXH',A($RSI) RSI BR REL ON INDEX
```

```
BRXLE    DC  XL2'8500',AL1(1),CL5'BRXLE',A($RSI) RSI BR REL ON INDEX
BXH      DC  XL2'8600',AL1(1),CL5'BXH',A($RS) RS BRANCH ON INDEX
BXLE     DC  XL2'8700',AL1(1),CL5'BXLE',A($RS) RS BRANCH ON INDEX
SRL      DC  XL2'8800',AL1(1),CL5'SRL',A($RS#) RS SHIFT RIGHT SINGLE
SLL      DC  XL2'8900',AL1(1),CL5'SLL',A($RS#) RS SHIFT LEFT SINGLE
SRA      DC  XL2'8A00',AL1(1),CL5'SRA',A($RS#) RS SHIFT RIGHT SINGLE
SLA      DC  XL2'8B00',AL1(1),CL5'SLA',A($RS) RS SHIFT LEFT SINGLE
SRDL     DC  XL2'8C00',AL1(1),CL5'SRDL',A($RS#) RS SHIFT RIGHT DOUBLE
SLDL     DC  XL2'8D00',AL1(1),CL5'SLDL',A($RS#) RS SHIFT LEFT DOUBLE
SRDA     DC  XL2'8E00',AL1(1),CL5'SRDA',A($RS#) RS SHIFT RIGHT DOUBLE
SLDA     DC  XL2'8F00',AL1(1),CL5'SLDA',A($RS#) RS SHIFT LEFT DOUBLE
HEX9     DS  0F ****************************************************
STM      DC  XL2'9000',AL1(1),CL5'STM',A($RS) RS STORE MULTIPLE
TM       DC  XL2'9100',AL1(1),CL5'TM',A($SI) SI TEST UNDER MASK
MVI      DC  XL2'9200',AL1(1),CL5'MVI',A($SI) SI MOVE (I$EDIATE)
TS       DC  XL2'9300',AL1(1),CL5'TS',A($S) S TEST AND SET
NI       DC  XL2'9400',AL1(1),CL5'NI',A($SI) SI AND (I$EDIATE)
CLI      DC  XL2'9500',AL1(1),CL5'CLI',A($SI) SI COMPARE LOGICAL
OI       DC  XL2'9600',AL1(1),CL5'OI',A($SI) SI OR (I$EDIATE)
XI       DC  XL2'9700',AL1(1),CL5'XI',A($SI) SI EXCLUSIVE OR
LM       DC  XL2'9800',AL1(1),CL5'LM',A($RS) RS LOAD MULTIPLE
TRACE    DC  XL2'9900',AL1(2),CL5'TRACE',A($RS) RS TRACE
LAM      DC  XL2'9A00',AL1(1),CL5'LAM',A($RS) RS LOAD ACCESS MULTIPLE
STAM     DC  XL2'9B00',AL1(1),CL5'STAM',A($RS) RS STORE ACCESS MULT.
SIO      DC  XL2'9C00',AL1(2),CL5'SIO',A($S) S START I/O
SIOF     DC  XL2'9C01',AL1(2),CL5'SIOF',A($S) S START I/O FAST
RIO      DC  XL2'9C02',AL1(2),CL5'RIO',A($S) S RESU$E I/O
TIO      DC  XL2'9D00',AL1(2),CL5'TIO',A($S) S TEST I/O
CLRIO    DC  XL2'9D01',AL1(2),CL5'CLRIO',A($S) S CLEAR I/O
HIO      DC  XL2'9E00',AL1(2),CL5'HIO',A($S) S HALT I/O
HDV      DC  XL2'9E01',AL1(2),CL5'HDV',A($S) S HALT DEVICE
TCH      DC  XL2'9F00',AL1(2),CL5'TCH',A($S) S TEST CHANNEL
CLRCH    DC  XL2'9F01',AL1(2),CL5'CLRCH',A($S) S CLEAR CHANNEL
HEXA     DS  0F ****************************************************
AHI      DC  XL2'A700',AL1(1),CL5'AHI',A($RI) RI ADD HALFWORD
TMH      DC  XL2'A700',AL1(1),CL5'TMH',A($RI) RI TEST UNDER MASK
TML      DC  XL2'A701',AL1(1),CL5'TML',A($RI) RI TEST UNDER MASK
BRAS     DC  XL2'A704',AL1(1),CL5'BRAS',A($RI) RI BRANCH REL AND
BRC      DC  XL2'A705',AL1(1),CL5'BRC',A($RI) RI BRANCH REL ON COND
BRCT     DC  XL2'A706',AL1(1),CL5'BRCT',A($RI) RI BRANCH REL ON
MHI      DC  XL2'A70C',AL1(1),CL5'MHI',A($RI) RI MULTIP. HALFW
CHI      DC  XL2'A70E',AL1(1),CL5'CHI',A($RI) RI COMPARE HALFWORD
MVCLE    DC  XL2'A800',AL1(1),CL5'MVCLE',A($RS) RS MOVE LONG EXTENDED
CLCLE    DC  XL2'A900',AL1(1),CL5'CLCLE',A($RS) RS COMPARE LOG LONG
STNSM    DC  XL2'AC00',AL1(1),CL5'STNSM',A($SI) SI STORE THEN AND
STOSM    DC  XL2'AD00',AL1(1),CL5'STOSM',A($SI) SI STORE THEN OR
SIGP     DC  XL2'AE00',AL1(1),CL5'SIGP',A($RS) RS SIGNAL PROCESSOR
MC       DC  XL2'AF00',AL1(1),CL5'MC',A($SI) SI MONITOR CALL
HEXB     DS  0F ****************************************************
LRA      DC  XL2'B100',AL1(1),CL5'LRA',A($RX) RX LOAD REAL ADDRESS
CONCS    DC  XL2'B200',AL1(2),CL5'CONCS',A($S) S CONNECT CHANNEL
DISCS    DC  XL2'B201',AL1(2),CL5'DISCS',A($S) S DISCONNECT CHANNEL
STIDP    DC  XL2'B202',AL1(2),CL5'STIDP',A($S) S STORE CPU ID
```

```
STIDC    DC  XL2'B2Ø3',AL1(2),CL5'STIDC',A($S) S STORE CHANNEL ID
SCK      DC  XL2'B2Ø4',AL1(2),CL5'SCK',A($S) S SET CLOCK
STCK     DC  XL2'B2Ø5',AL1(2),CL5'STCK',A($S) S STORE CLOCK
SCKC     DC  XL2'B2Ø6',AL1(2),CL5'SCKC',A($S) S SET CLOCK
STCKC    DC  XL2'B2Ø7',AL1(2),CL5'STCKC',A($S) S STORE CLOCK
SPT      DC  XL2'B2Ø8',AL1(2),CL5'SPT',A($S) S SET CPU TI$ER
STPT     DC  XL2'B2Ø9',AL1(2),CL5'STPT',A($S) S STORE CPU TI$ER
SPKA     DC  XL2'B2ØA',AL1(2),CL5'SPKA',A($S) S SET PSW KEY FROM
IPK      DC  XL2'B2ØB',AL1(2),CL5'IPK',A(Ø) INSERT PSW KEY
PTLB     DC  XL2'B2ØD',AL1(2),CL5'PTLB',A(Ø) PURGE TLB
SPX      DC  XL2'B21Ø',AL1(2),CL5'SPX',A($S) S SET PREFIX
STPX     DC  XL2'B211',AL1(2),CL5'STPX',A($S) S STORE PREFIX
STAP     DC  XL2'B212',AL1(2),CL5'STAP',A($S) S STORE CPU ADDRESS
RRB      DC  XL2'B213',AL1(2),CL5'RRB',A($S) S RESET REFERENCE BIT
PC       DC  XL2'B218',AL1(2),CL5'PC',A($S) S PROGRAM CALL
PCF      DC  XL2'B218',AL1(2),CL5'PCF',A($S) S PROGRAM CALL
SAC      DC  XL2'B219',AL1(2),CL5'SAC',A($S) S SET ADDRESS SPACE
CFC      DC  XL2'B21A',AL1(1),CL5'CFC',A($S) S COMPARE AND FORM
IPTE     DC  XL2'B221',AL1(2),CL5'IPTE',A($RR) RRE INVALIDATE PAGE
IPM      DC  XL2'B222',AL1(1),CL5'IPM',A($RRE) RRE INSERT PROG MASK
IVSK     DC  XL2'B223',AL1(2),CL5'IVSK',A($RR) RRE INSERT VIRTUAL
IAC      DC  XL2'B224',AL1(2),CL5'IAC',A($RRE#) RRE INSERT ADDRESS
SSAR     DC  XL2'B225',AL1(2),CL5'SSAR',A($RRE#) RRE SET SECONDARY
EPAR     DC  XL2'B226',AL1(2),CL5'EPAP',A($RRE#) RRE EXTRACT
ESAR     DC  XL2'B227',AL1(2),CL5'ESAR',A($RRE#) RRE EXTRACT
PT       DC  XL2'B228',AL1(2),CL5'PT',A($RR) RRE PROGRAM TRANSFER
ISKE     DC  XL2'B229',AL1(2),CL5'ISKE',A($RR) RRE INSERT STORAGE
PRBE     DC  XL2'B22A',AL1(2),CL5'PRBE',A($RRE) RRE RESET REF BIT
RRBE     DC  XL2'B22A',AL1(2),CL5'RRBE',A($RR) RRE RESET REFERENCE
SSKE     DC  XL2'B22B',AL1(2),CL5'SSKE',A($RR) RRE SET STORAGE KEY
TB       DC  XL2'B22C',AL1(2),CL5'TB',A($RR) RRE TEST BLOK
CSCH     DC  XL2'B23Ø',AL1(2),CL5'CSCH',A($S) S CLEAR SUBCHANNEL
HSCH     DC  XL2'B231',AL1(2),CL5'HSCH',A($S) S HALT SUBCHANNEL
MSCH     DC  XL2'B232',AL1(2),CL5'$SCH',A($S) S MODIFY SUBCHANNEL
SSCH     DC  XL2'B233',AL1(2),CL5'SSCH',A($S) S START SUBCHANNEL
STSCH    DC  XL2'B234',AL1(2),CL5'STSCH',A($S) S STORE SUBCHANNEL
TSCH     DC  XL2'B235',AL1(2),CL5'TSCH',A($S) S TEST SUBCHANNEL
TPI      DC  XL2'B236',AL1(2),CL5'TPI',A($S) S TEST PENDING
SAL      DC  XL2'B237',AL1(2),CL5'SAL',A($S) S SET ADDRESS LIMIT
RSCH     DC  XL2'B238',AL1(2),CL5'RSCH',A($S) S RESU$E SUBCHANNEL
STCRW    DC  XL2'B239',AL1(2),CL5'STCRW',A($S) S STORE CHAN REPORT
STCSP    DC  XL2'B23A',AL1(2),CL5'STCSP',A($S) S STORE CHAN PATH STAT
RCHP     DC  XL2'B23B',AL1(2),CL5'RCHP',A($S) S RESET CHANNEL PATH
SCHM     DC  XL2'B23C',AL1(2),CL5'SCHM',A($S) S SET CHANNEL MONITOR
BAKR     DC  XL2'B24Ø',AL1(1),CL5'BAKR',A($RRE) RRE BRANCH AND STACK
CKSM     DC  XL2'B241',AL1(1),CL5'CKSM',A($RRE) CHECKSUM
SQDR     DC  XL2'B244',AL1(1),CL5'SQDR',A($RRE) RRE SQUARE ROOT
SQER     DC  XL2'B245',AL1(1),CL5'SQER',A($RRE) RRE SQUARE ROOT
STURA    DC  XL2'B246',AL1(1),CL5'STURA',A($RRE) RRE STORE US REAL
MSTA     DC  XL2'B247',AL1(1),CL5'$STA',A($RRE) RRE MODIFY STACKED
PALB     DC  XL2'B248',AL1(2),CL5'PALB',A($RRE) RRE PURGE ALB
EREG     DC  XL2'B249',AL1(1),CL5'EREG',A($RRE) RRE EXTRACT STAC REG
ESTA     DC  XL2'B24A',AL1(1),CL5'ESTA',A($RRE) RRE EXTRACT STAC
```

```
LURA     DC  XL2'B24B',AL1(1),CL5'LURA',A($RRE) RRE LOAD USING REAL
TAR      DC  XL2'B24C',AL1(1),CL5'TAR',A($RRE) RRE TEST ACCESS
CPYA     DC  XL2'B24D',AL1(1),CL5'CPYA',A($RRE) RRE COPY ACCESS
SAR      DC  XL2'B24E',AL1(1),CL5'SAR',A($RRE) RRE SET ACCESS
MSR      DC  XL2'B252',AL1(1),CL5'$SR',A($RRE) RRE MULTIPLY SINGLE
MVPG     DC  XL2'B254',AL1(1),CL5'MVPG',A($RRE) RRE MOVE PAGE
MVST     DC  XL2'B255',AL1(1),CL5'MVST',A($RRE) RRE MOVE STRING
CUSE     DC  XL2'B257',AL1(1),CL5'CUSE',A($RRE) RRE COMPARE UNT SUB
BSG      DC  XL2'B258',AL1(1),CL5'BSG',A($RRE) RRE BRANCH IN SUBS
BSA      DC  XL2'B25A',AL1(1),CL5'BSA',A($RRE) RRE BRANCH AND SET
CLST     DC  XL2'B25D',AL1(1),CL5'CLST',A($RRE) RRE COMPARE LOG LONG
SRST     DC  XL2'B25E',AL1(1),CL5'SRST',A($RRE) RRE SEARCH STRING
RP       DC  XL2'B277',AL1(2),CL5'RP',A($S) S REUSE PROGRAM
STCKE    DC  XL2'B278',AL1(2),CL5'STCKE',A($S) S STORE CLOCK EXTENDED
SACF     DC  XL2'B279',AL1(2),CL5'SACF',A($S) S SET ADDRESS SPACE CON
STSI     DC  XL2'B27D',AL1(2),CL5'STSI',A($S) S STORE SYSTEM INFORM
STFPC    DC  XL2'B29C',AL1(1),CL5'STFPC',A($S) $S STORE FPC
CUUTF    DC  XL2'B2A6',AL1(1),CL5'CUUTF',A($RRE) RRE CONVERT UNIC TO UT
CUTFU    DC  XL2'B2A7',AL1(1),CL5'CUTFU',A($RRE) RRE CONVERT UNF8 TO UN
TRAP4    DC  XL2'B2FF',AL1(2),CL5'TRAP4',A($S) S TRAP
LPEBR    DC  XL2'B3ØØ',AL1(1),CL5'LPEBR',A($RR) RR LOAD POSITIVE
LNEBR    DC  XL2'B3Ø1',AL1(1),CL5'LNEBR',A($RRE) RRE LOAD NEGATIVE
LTEBR    DC  XL2'B3Ø2',AL1(1),CL5'LTEBR',A($RRE) RRE LOAD AND TEST
LCEBR    DC  XL2'B3Ø3',AL1(1),CL5'LCEBR',A($RRE) RRE LOAD COMPLEMENT
LDEBR    DC  XL2'B3Ø4',AL1(1),CL5'LDEBR',A($RRE) RRE LOAD LENGTHENED
LXDBR    DC  XL2'B3Ø5',AL1(1),CL5'LXDBR',A($RRE) RRE LOAD LENGTHENED
LXEBR    DC  XL2'B3Ø6',AL1(1),CL5'LXEBR',A($RRE) RRE LOAD LENGTHENED
MXEBR    DC  XL2'B3Ø7',AL1(1),CL5'MXEBR',A($RRE) RRE MULTIPLY LXL=E
KEBR     DC  XL2'B3Ø8',AL1(1),CL5'KEBR',A($RRE) RRE COMP AND SIGNAL
CEBR     DC  XL2'B3Ø9',AL1(1),CL5'CEBR',A($RRE) RRE COMPARE SHORT
AEBR     DC  XL2'B3ØA',AL1(1),CL5'AEBR',A($RRE) RRE ADD SHORT
SEBR     DC  XL2'B3ØB',AL1(1),CL5'SEBR',A($RRE) RRE SUBTRACT NORM
MDEBR    DC  XL2'B3ØC',AL1(1),CL5'MDEBR',A($RRE) RRE MULTIPLY SXS=L
DEBR     DC  XL2'B3ØD',AL1(1),CL5'DEBR',A($RRE) RRE DIVIDE (SHORT)
MAEBR    DC  XL2'B3ØE',AL1(1),CL5'MAEBR',A($RRF) RRF MULTIPLY AND ADD
MSEBR    DC  XL2'B3ØF',AL1(1),CL5'$SEBR',A($RRF) RRF MULTIPLY AND SUB
LPDBR    DC  XL2'B31Ø',AL1(1),CL5'LPDBR',A($RR) RR LOAD POSITIVE LONG
LNDBR    DC  XL2'B311',AL1(1),CL5'LNDBR',A($RRE) RRE LOAD NEGATIVE
LTDBR    DC  XL2'B312',AL1(1),CL5'LTDBR',A($RRE) RRE LOAD AND TEST
LCDBR    DC  XL2'B313',AL1(1),CL5'LCDBR',A($RRE) RRE LOAD COMPLE$ENT
SQEBR    DC  XL2'B314',AL1(1),CL5'SQEBR',A($RRE) RRE SQUARE ROOT
SQDBR    DC  XL2'B315',AL1(1),CL5'SQDBR',A($RRE) RRE SQUARE ROOT
SQXBR    DC  XL2'B316',AL1(1),CL5'SQXBR',A($RRE) RRE SQUARE ROOT
MEEBR    DC  XL2'B317',AL1(1),CL5'$EEBR',A($RRE) RRE MULTIPLY
KDBR     DC  XL2'B318',AL1(1),CL5'KDBR',A($RRE) RRE COMP AND SIGNAL
CDBR     DC  XL2'B319',AL1(1),CL5'CDBR',A($RRE) RRE COMPARE
ADBR     DC  XL2'B31A',AL1(1),CL5'ADBR',A($RRE) RRE ADD
SDBR     DC  XL2'B31B',AL1(1),CL5'SDBR',A($RRE) RRE SUBTRACT NORM
MDBR     DC  XL2'B31C',AL1(1),CL5'MDBR',A($RRE) RRE MULTIPLY
DDBR     DC  XL2'B31D',AL1(1),CL5'DDBR',A($RRE) RRE DIVIDE
MADBR    DC  XL2'B31E',AL1(1),CL5'MADBR',A($RRF) RRF MULTIPLY AND ADD
MSDBR    DC  XL2'B31F',AL1(1),CL5'$SDBR',A($RRF) RRF MULTIPLY AND SUB
LDER     DC  XL2'B324',AL1(1),CL5'LDER',A($RRE) RRE LOAD LENGTHENED
```

```
LXDR     DC  XL2'B325',AL1(1),CL5'LXDR',A($RRE) RRE LOAD LENGTHENED
LXER     DC  XL2'B326',AL1(1),CL5'LXER',A($RRE) RRE LOAD LENGTHENED
SQXR     DC  XL2'B336',AL1(1),CL5'SQXR',A($RRE) RRE SQUARE ROOT EXTENDA
MEER     DC  XL2'B337',AL1(1),CL5'$EER',A($RR) RR MULTIPLY SHORT
LPXBR    DC  XL2'B34Ø',AL1(1),CL5'LPXBR',A($RRE) RRE LOAD POSITIVE
LNXBR    DC  XL2'B341',AL1(1),CL5'LNXBR',A($RRE) RRE LOAD NEGATIVE
LTXBR    DC  XL2'B342',AL1(1),CL5'LTXBR',A($RRE) RRE LOAD AND TEST
LCXBR    DC  XL2'B343',AL1(1),CL5'LCXBR',A($RRE) RRE LOAD  COMPLE$ENTEX
LEDBR    DC  XL2'B344',AL1(1),CL5'LEDBR',A($RRE) RR LOAD ROUNDED L/S
LDXBR    DC  XL2'B345',AL1(1),CL5'LDXBR',A($RRE) RR LOAD ROUNDED E/L
LEXBR    DC  XL2'B346',AL1(1),CL5'LEXBR',A($RRE) RRE LOAD ROUNDED E/S
FIXBR    DC  XL2'B347',AL1(1),CL5'FIXBR',A($RRE) RRE LOAD FP INT EX
KXBR     DC  XL2'B348',AL1(1),CL5'KXBR',A($RRE) RRE COMP AND SIGNAL  EX
CXBR     DC  XL2'B349',AL1(1),CL5'CXBR',A($RRE) RRE COMPARE EXTENDED
AXBR     DC  XL2'B34A',AL1(1),CL5'AXBR',A($RRE) RRE ADD EXTENDED
SXBR     DC  XL2'B34B',AL1(1),CL5'SXBR',A($RRE) RRE SUBTRACT NORM  EXTE
MXBR     DC  XL2'B34C',AL1(1),CL5'MXBR',A($RRE) RRE MULTIPLY EXENDED
DXBR     DC  XL2'B34D',AL1(1),CL5'DXBR',A($RRE) RRE DIVIDE   (EXTENDED)
TBEDR    DC  XL2'B35Ø',AL1(1),CL5'TBEDR',A($RRF) RRF CONV HFP TO BFP
TBDR     DC  XL2'B351',AL1(1),CL5'TBDR',A($RRF) RRF CONV HFP TO BFP
FIEBR    DC  XL2'B357',AL1(1),CL5'FIEBR',A($RRE) RRE LOAD FP INT SH
THDER    DC  XL2'B358',AL1(1),CL5'THDER',A($RRE) RRE CONV BFP TO HFP
THDR     DC  XL2'B359',AL1(1),CL5'THDR',A($RRE) RRE CONV BFP TO HFP
FIDBR    DC  XL2'B35F',AL1(1),CL5'FIDBR',A($RRE) RRE LOAD FP INT LO
LPXR     DC  XL2'B36Ø',AL1(1),CL5'LPXR',A($RRE) RRE LOAD   POSITIVE(EXT
LTXR     DC  XL2'B362',AL1(1),CL5'LTXR',A($RRE) RRE LOAD AND TEST  (EXT
LCXR     DC  XL2'B363',AL1(1),CL5'LCXR',A($RRE) RRE LOAD COMPLEMENTEXT
LXR      DC  XL2'B365',AL1(1),CL5'LXR',A($RX) RX LOAD (EXTENDED)
LEXR     DC  XL2'B366',AL1(1),CL5'LEXR',A($RRE) RRE LOAD ROUNDED E/S
FIXR     DC  XL2'B367',AL1(1),CL5'FIXR',A($RRE) RRE LOAD FP INTEGER  EX
LZER     DC  XL2'B374',AL1(1),CL5'LZER',A($RRE) RRE LOAD ZERO   (SHORT)
LZDR     DC  XL2'B375',AL1(1),CL5'LZDR',A($RRE) RRE LOAD ZERO (LONG)
LZXR     DC  XL2'B376',AL1(1),CL5'LZXR',A($RRE) RRE LOAD ZERO  (EXTENDE
FIER     DC  XL2'B377',AL1(1),CL5'FIER',A($RRE) RRE LOAD FP INTEGER  SH
FIDR     DC  XL2'B37F',AL1(1),CL5'FIDR',A($RRE) RRE LOAD FP INTEGER  LO
EFPC     DC  XL2'B38C',AL1(1),CL5'EFPC',A($RRE) RRE EXTRACT FPC
CEFBR    DC  XL2'B394',AL1(1),CL5'CEFBR',A($RRE) RRE CONV FROM FIXED  S
CDFBR    DC  XL2'B395',AL1(1),CL5'CDFBR',A($RRE) RRE CONV FROM FIXED  L
CXFBR    DC  XL2'B396',AL1(1),CL5'CXFBR',A($RRE) RRE CONV FROM FIXED  E
CFEBR    DC  XL2'B398',AL1(2),CL5'CFEBR',A($RRF) RRF CONV FROM FIX SH
CFDBR    DC  XL2'B399',AL1(2),CL5'CFDBR',A($RRF) RRF CONV FROM FIX LO
CFXBR    DC  XL2'B39A',AL1(2),CL5'CFXBR',A($RRF) RRF CONV FROM FIX EX
CEFR     DC  XL2'B3B4',AL1(2),CL5'CEFR',A($RR) RRE CONV FROM FIX SH
CDFR     DC  XL2'B3B5',AL1(2),CL5'CDFR',A($RR) RRE CONV FROM FIX LO
CXFR     DC  XL2'B3B6',AL1(2),CL5'CXFR',A($RR) RRE CONV FROM FIX EX
CFER     DC  XL2'B3B8',AL1(2),CL5'CFER',A($RRF) RRF CONV FROM FIX LO
CFDR     DC  XL2'B3B9',AL1(2),CL5'CFDR',A($RRF) RRF CONV FROM FIX LO
CFXR     DC  XL2'B3BA',AL1(2),CL5'CFXR',A($RRF) RRF CONV FROM FIX LO
STCTL    DC  XL2'B6ØØ',AL1(1),CL5'STCTL',A($RS) RS STORAGE CONTROL
LCTL     DC  XL2'B7ØØ',AL1(1),CL5'LCTL',A($RS) RS LOAD CONTROL
CS       DC  XL2'BAØØ',AL1(1),CL5'SC',A($RS) RS COMPARE AND SWAP
CDS      DC  XL2'BBØØ',AL1(1),CL5'CDS',A($RS) RS COMPARE DOUBLE AND
CLM      DC  XL2'BDØØ',AL1(1),CL5'CLM',A($RS) RS COMPARE LOGICAL C.
```

```
STCM     DC  XL2'BEØØ',AL1(1),CL5'STCM',A($RS) RS STORE CHARACTERS
ICM      DC  XL2'BFØØ',AL1(1),CL5'ICM',A($RS) RS INSERT CHARACTERS
HEXC     DS  ØF ****************************************************
HEXD     DS  ØF ****************************************************
MVN      DC  XL2'D1ØØ',AL1(1),CL5'MVN',A($SS1) SS MOUVE NU$ERICS
MVC      DC  XL2'D2ØØ',AL1(1),CL5'MVC',A($SS1) SS MOUVE (CHARACTER)
MVZ      DC  XL2'D3ØØ',AL1(1),CL5'MVZ',A($SS1) SS MOUVE ZONES
NC       DC  XL2'D4ØØ',AL1(1),CL5'NC',A($SS1) SS AND (CHARACTER)
CLC      DC  XL2'D5ØØ',AL1(1),CL5'CLC',A($SS1) SS COMPARE LOGICAL
OC       DC  XL2'D6ØØ',AL1(1),CL5'OC',A($SS1) SS OR (CHARACTER)
XC       DC  XL2'D7ØØ',AL1(1),CL5'XC',A($SS1) SS EXCLUSIVE OR
MVCK     DC  XL2'D9ØØ',AL1(1),CL5'MVCK',A($SS2) SS MOUVE WITH KEY
MVCP     DC  XL2'DAØØ',AL1(1),CL5'MVCP',A($SS2) SS MOUVE TO PRIMARY
MVCS     DC  XL2'DBØØ',AL1(1),CL5'MVCS',A($SS2) SS MOUVE TO SECONDARY
TR       DC  XL2'DCØØ',AL1(1),CL5'TR',A($SS1) SS TRANSLATE
TRT      DC  XL2'DDØØ',AL1(1),CL5'TRT',A($SS1) SS TRANSLATE AND TEST
ED       DC  XL2'DEØØ',AL1(1),CL5'ED',A($SS1) SS EDIT
EDMK     DC  XL2'DFØØ',AL1(1),CL5'EDMK',A($SS1) SS EDIT AND MARK
HEXE     DS  ØF ****************************************************
LASP     DC  XL2'E5ØØ',AL1(2),CL5'LASP',A($SSE) SS LOAD ADDRESS SPACE
TPROT    DC  XL2'E5Ø1',AL1(2),CL5'TPROT',A($SSE) SS TEST PROTECTION
MVCSK    DC  XL2'E5ØE',AL1(1),CL5'MVCSK',A($SSE) SSE MOVE WITH SOUR
MVCDK    DC  XL2'E5ØF',AL1(1),CL5'MVCDK',A($SSE) SSE MOVE WITH DEST
MVCIN    DC  XL2'E8ØØ',AL1(1),CL5'MVCIN',A($SS1) SS MOVE INVERSE
MEE      DC  XL2'ED37',AL1(1),CL5'$EE',A($RX) RX MULTIPLY SHORT
PLO      DC  XL2'EEØØ',AL1(1),CL5'PLO',A($SS4) SS PERFORM LOCKED
HEXF     DS  ØF ****************************************************
SRP      DC  XL2'FØØØ',AL1(1),CL5'SRP',A($SS2) SS SHIFT AND ROUND
MVO      DC  XL2'F1ØØ',AL1(1),CL5'MVO',A($SS3) SS MOVE WITH OFFSET
PACK     DC  XL2'F2ØØ',AL1(1),CL5'PACK',A($SS3) SS PACK
UNPK     DC  XL2'F3ØØ',AL1(1),CL5'UNPK',A($SS3) SS UNPACK
ZAP      DC  XL2'F8ØØ',AL1(1),CL5'ZAP',A($SS3) SS ZERO AND ADD
CP       DC  XL2'F9ØØ',AL1(1),CL5'CP',A($SS3) SS COMPARE DECIMAL
AP       DC  XL2'FAØØ',AL1(1),CL5'AP',A($SS3) SS ADD DECIMAL
SP       DC  XL2'FBØØ',AL1(1),CL5'SP',A($SS3) SS SUBTRACT DECIMAL
MP       DC  XL2'FCØØ',AL1(1),CL5'MP',A($SS3) SS MULTIPLAY DECIMAL
DP       DC  XL2'FDØØ',AL1(1),CL5'DP',A($SS2) SS DIVIDE DECIMAL
         DC  X'FFFF',AL1(Ø)
**********************************************************************
         END
```

Module ASVC contains a list of available SVCs. You can add new ones if you need them.

```
ASVC   CSECT
*      SVC LIST
SVCCOUNT  DC A((SVCEND-*-4)/8)
SVCØØ  DC  CL8'EXCP'          SVC    Ø
SVCØ1  DC  CL8'WAIT'          SVC    1
SVCØ2  DC  CL8'POST'          SVC    2
SVCØ3  DC  CL8'EXIT'          SVC    3
SVCØ4  DC  CL8'GETMAIN'       SVC    4
SVCØ5  DC  CL8'FREEMAIN'      SVC    5
```

```
SVCØ6   DC   CL8'LINK'          SVC    6
SVCØ7   DC   CL8'XCTL'          SVC    7
SVCØ8   DC   CL8'LOAD'          SVC    8
SVCØ9   DC   CL8'DELETE'        SVC    9
SVCØA   DC   CL8'GETM R'        SVC   1Ø
SVCØB   DC   CL8'TIME'          SVC   11
SVCØC   DC   CL8'SYNCH'         SVC   12
SVCØD   DC   CL8'ABEND'         SVC   13
SVCØE   DC   CL8'SPIE'          SVC   14
SVCØF   DC   CL8'ERREXCP'       SVC   15
SVC1Ø   DC   CL8'PURGE'         SVC   16
SVC11   DC   CL8'RESTORE'       SVC   17
SVC12   DC   CL8'BLDL D'        SVC   18
SVC13   DC   CL8'OPEN'          SVC   19
SVC14   DC   CL8'CLOSE'         SVC   2Ø
SVC15   DC   CL8'STOW'          SVC   21
SVC16   DC   CL8'OPEN J'        SVC   22
SVC17   DC   CL8'CLOSE T'       SVC   23
SVC18   DC   CL8'DEVTYPE'       SVC   24
SVC19   DC   CL8'TRKBAL'        SVC   25
SVC1A   DC   CL8'CAT.LOC.'      SVC   26
SVC1B   DC   CL8'OBTAIN'        SVC   27
SVC1C   DC   CL8'IBM RES.'      SVC   28
SVC1D   DC   CL8'SCRATCH'       SVC   29
SVC1E   DC   CL8'RENAME'        SVC   3Ø
SVC1F   DC   CL8'FEOV'          SVC   31
SVC2Ø   DC   CL8'ALLOC'         SVC   32
SVC21   DC   CL8'IOHALT'        SVC   33
SVC22   DC   CL8'MGCR'          SVC   34
SVC23   DC   CL8'WTO '          SVC   35
SVC24   DC   CL8'WTL '          SVC   36
SVC25   DC   CL8'SEGLD'         SVC   37
SVC26   DC   CL8'IBM RES.'      SVC   38
SVC27   DC   CL8'LABEL'         SVC   39
SVC28   DC   CL8'EXTRACT'       SVC   4Ø
SVC29   DC   CL8'IDENTIFY'      SVC   41
SVC2A   DC   CL8'ATTACH'        SVC   42
SVC2B   DC   CL8'CIRB'          SVC   43
SVC2C   DC   CL8'CHAP'          SVC   44
SVC2D   DC   CL8'OVLYBRCH'      SVC   45
SVC2E   DC   CL8'TTIMER'        SVC   46
SVC2F   DC   CL8'STIMER'        SVC   47
SVC3Ø   DC   CL8'DEQ '          SVC   48
SVC31   DC   CL8'IBM RES.'      SVC   49
SVC32   DC   CL8'IBM RES.'      SVC   5Ø
SVC33   DC   CL8'SNAP'          SVC   51
SVC34   DC   CL8'RESTART'       SVC   52
SVC35   DC   CL8'RELEX'         SVC   53
SVC36   DC   CL8'DISABLE'       SVC   54
SVC37   DC   CL8'EOV'           SVC   55
SVC38   DC   CL8'ENQ/RES.'      SVC   56
SVC39   DC   CL8'FREEDBUF'      SVC   57
SVC3A   DC   CL8'REL/Q BU'      SVC   58
```

```
SVC3B   DC   CL8'OLTEP'          SVC   59
SVC3C   DC   CL8'STAE'           SVC   6Ø
SVC3D   DC   CL8'IKJEGS6A'       SVC   61
SVC3E   DC   CL8'DETACH'         SVC   62
SVC3F   DC   CL8'CHKPT'          SVC   63
SVC4Ø   DC   CL8'RDJFCB'         SVC   64
SVC41   DC   CL8'IBM RES.'       SVC   65
SVC42   DC   CL8'BTAMTEST'       SVC   66
SVC43   DC   CL8'IBM RES.'       SVC   67
SVC44   DC   CL8'SYNADA/R'       SVC   68
SVC45   DC   CL8'BSP'            SVC   69
SVC46   DC   CL8'GSERV'          SVC   7Ø
SVC47   DC   CL8'ASG/RL.BU'      SVC   71
SVC48   DC   CL8'NO MACRO'       SVC   72
SVC49   DC   CL8'SPAR'           SVC   73
SVC4A   DC   CL8'DAR'            SVC   74
SVC4B   DC   CL8'DQUEUE'         SVC   75
SVC4C   DC   CL8'IFBSTAT'        SVC   76
SVC4D   DC   CL8'IBM RES.'       SVC   77
SVC4E   DC   CL8'LSPACE'         SVC   78
SVC4F   DC   CL8'STATUS'         SVC   79
SVC5Ø   DC   CL8'IBM RES.'       SVC   8Ø
SVC51   DC   CL8'SETPRT'         SVC   81
SVC52   DC   CL8'IBM RES.'       SVC   82
SVC53   DC   CL8'SMFWTM'         SVC   83
SVC54   DC   CL8'GRAPHICS'       SVC   84
SVC55   DC   CL8'DDRSWAP'        SVC   85
SVC56   DC   CL8'ATLAS'          SVC   86
SVC57   DC   CL8'DOM'            SVC   87
SVC58   DC   CL8'IBM RES.'       SVC   88
SVC59   DC   CL8'IBM RES.'       SVC   89
SVC5A   DC   CL8'IBM RES.'       SVC   9Ø
SVC5B   DC   CL8'VOLSTAT'        SVC   91
SVC5C   DC   CL8'TCPEXCP'        SVC   92
SVC5D   DC   CL8'TGETTPUT'       SVC   93
SVC5E   DC   CL8'STCC MAC'       SVC   94
SVC5F   DC   CL8'SYSEVENT'       SVC   95
SVC6Ø   DC   CL8'STAX'           SVC   96
SVC61   DC   CL8'IKJEGS9G'       SVC   97
SVC62   DC   CL8'PROTECT'        SVC   98
SVC63   DC   CL8'DYNALLOC'       SVC   99
SVC64   DC   CL8'IKJEFFIB'       SVC  1ØØ
SVC65   DC   CL8'QTIP'           SVC  1Ø1
SVC66   DC   CL8'AQCTL'          SVC  1Ø2
SVC67   DC   CL8'XLATE'          SVC  1Ø3
SVC68   DC   CL8'TOPCTL'         SVC  1Ø4
SVC69   DC   CL8'IMGLIB'         SVC  1Ø5
SVC6A   DC   CL8'IBM RES.'       SVC  1Ø6
SVC6B   DC   CL8'MODESET'        SVC  1Ø7
SVC6C   DC   CL8'IBM RES.'       SVC  1Ø8
SVC6D   DC   CL8'ESR T4'         SVC  1Ø9
SVC6E   DC   CL8'IBM RES.'       SVC  11Ø
SVC6F   DC   CL8'NO MACRO'       SVC  111
```

```
SVC70  DC  CL8'PGRLSE'         SVC  112
SVC71  DC  CL8'PGFIX...'       SVC  113
SVC72  DC  CL8'EXCPVR'         SVC  114
SVC73  DC  CL8'IBM RES.'       SVC  115
SVC74  DC  CL8'ESR T1'         SVC  116
SVC75  DC  CL8'DEBCHK'         SVC  117
SVC76  DC  CL8'IBM RES.'       SVC  118
SVC77  DC  CL8'TESTAUTH'       SVC  119
SVC78  DC  CL8'G/F MAIN'       SVC  120
SVC79  DC  CL8'VSAM'           SVC  121
SVC7A  DC  CL8'ESR T2'         SVC  122
SVC7B  DC  CL8'PURGEDQ'        SVC  123
SVC7C  DC  CL8'TPIO'           SVC  124
SVC7D  DC  CL8'EVENTS'         SVC  125
SVC7E  DC  CL8'MSS_ICB2'       SVC  126
SVC7F  DC  CL8'IBM RES.'       SVC  127
SVC80  DC  CL8'IBM RES.'       SVC  128
SVC81  DC  CL8'IBM RES.'       SVC  129
SVC82  DC  CL8'RACHECK'        SVC  130
SVC83  DC  CL8'RACINIT'        SVC  131
SVC84  DC  CL8'RACLIST'        SVC  132
SVC85  DC  CL8'RACDEF'         SVC  133
SVC86  DC  CL8'IBM RES.'       SVC  134
SVC87  DC  CL8'IBM RES.'       SVC  135
SVC88  DC  CL8'IBM RES.'       SVC  136
SVC89  DC  CL8'ESR'            SVC  137
SVC8A  DC  CL8'PGSER'          SVC  138
SVC8B  DC  CL8'CVA... '        SVC  139
SVCD7  DC  CL8'CICS   '        SVC  215
SVCD8  DC  CL8'CICS   '        SVC  216
SVCEND EQU *
         END
```

JOB FORSUBMITTING ADISASEM

This is an example of disassembling. In the last step we assemble the result of disassembling and get a load module that is exactly the same as the input to ADISASEM.

```
//useridL JOB (ACCT#),
//           USER=,GROUP=,PASSWORD=,                        /*RACF*/
//           NOTIFY=userid,
//           CLASS=C,MSGCLASS=X,MSGLEVEL=(1,1)
//IDCAMS     EXEC PGM=IDCAMS
//SYSPRINT   DD SYSOUT=X
//SYSIN      DD *
 DELETE userid.#ADLOG.LIST
 SET MAXCC=0
/*
//S2  EXEC PGM=ADISASEM
//STEPLIB    DD DSN=userid.USER.LOAD,DISP=SHR
```

```
//SOURCE    DD DSN=userid.#ADLOG.LIST,DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,DCB=(RECFM=FB,LRECL=8Ø,BLKSIZE=Ø),
//             SPACE=(TRK,(1Ø,5),RLSE)
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
COPYPARM COPYARM2
/*
//S3 EXEC ASMACL,PARM.L='CALL'
//C.SYSLIB  DD DSN=SYS1.MACLIB,DISP=SHR
//          DD DSN=SYS1.AMODGEN,DISP=SHR
//          DD DSN=SYS1.SAMPLIB,DISP=SHR
//C.SYSIN   DD DSN=userid.#ADLOG.LIST,DISP=SHR
//L.SYSLIB  DD DSN=userid.USER.LOAD,DISP=SHR
//*         DD DSN=SYS1.LINKLIB,DISP=SHR
//L.SYSLMOD DD DSN=userid.USER.LOAD(#A),DISP=SHR
```

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

WAIT, POST and the EVENTS macro

When developing applications that need Assembler because of the availability of its interfaces, one of the most common features to be implemented is some form of WAIT and POST mechanism. This is used to enable communication between two separate units of work within a given Address Space, when some degree of synchronization is required. In some cases there are just two such units, one of which has a dependency on the other.

Often, however, there are several, and it may not be an easy task to predict how many will be active at any one time. To this end, the EVENTS macro can be usefully employed to provide the necessary regulation and control over such a fluid environment.

First let us look at the general architecture in which WAIT, EVENTS, and POST operate. OS/390 executes multiple Address Spaces, each of which will contain a job, a TSO user, or a started task. Within each Address Space, there will be at least one potentially ‘dispatchable unit of work’, which is generically known as a ‘task’. A task will be in one of two states – ‘dispatchable’, or ‘non-dispatchable’. This determines whether or not it is eligible to receive service from the host. There may be more than one task in an address space, and there will be a distinct hierarchy between them. The highest level task is often called the ‘mother task’ and lower level tasks are usually called ‘subtasks’. Subtasks may themselves have lower level subtasks, if the program design so demands. Each task is regarded by OS/390 as a separate entity that will require and compete for services from the operating system, given that it has a suitable status ie ‘dispatchable’. In support of this structure is a series of control blocks, which provide the anchor on which the task is managed and also define the hierarchy between the tasks in an address space. These are known as TCBs, or sometimes STCBs.

When a program is executing under a given TCB, there will be a further control block, known as a Request Block (RB), which reflects the status of the program during its execution. There are several types of RB, the most common of which are the Program Request Block (PRB) and the Supervisor Request Block (SVRB). A PRB is created when a program is initiated and an SVRB is created when some SVCs are invoked as part of the code. All RBs are aligned on a doubleword boundary, hence the last three bits of their address are B'000'. This will be shown to have some significance later on. Each RB will disappear when the program or SVC service is complete. RBs are chained to each other in creation sequence with the first RB being chained from the TCB (or STCB). The last RB in the chain represents the active process. Once the function represented by an RB is complete, it is dechained.

Thus the RB that is directly chained from the TCB represents the currently executing program in each task. As a result of instructions executed by this program, the task may or may not be ‘dispatchable’. For instance the program may have initiated an I/O operation and cannot proceed until it has completed. In essence it has to WAIT until something has occurred (I/O completion), which will result in the task being restored to a ‘dispatchable’ state. This is most frequently achieved by the use of a POST mechanism. Whilst the task is waiting, the RB contains a field identifying the instruction at which execution should resume once the WAIT has been honoured (the ‘resume point’). When the circumstances are such that POSTing is appropriate, a mechanism is needed to relate the completion to the appropriate task. This is achieved by the use of yet another control block, the Event Control Block (ECB).

When the program issues the WAIT macro, it specifies the ECB. The ECB must be cleared to X'00000000' before the WAIT is issued. The WAIT service places the address of the RB in the ECB and sets the Task non-dispatchable. It also sets the high bit of the ECB to B'1'. At some future time, when the POST is appropriate, the POST macro is issued, again specifying the ECB. The POST service derives the RB from the ECB and uses this to alter the task's status to dispatchable, whereupon it resumes competition for OS/390 services, using the 'resume point' as stored in the RB.

A task in an application may have a requirement to WAIT for a single event, such as I/O, to complete. A more elaborate task may have more than one I/O operation in action at any instant. In addition, a multi-tasking application may have a requirement to synchronize events between multiple tasks. In some cases, the designer may not be able to predict in advance how many activities require the synchronization services simultaneously. Three mechanisms are available to address these issues. They are:

- WAIT with a single ECB
- WAIT with an ECB list
- The EVENTS macro.

WAIT WITH A SINGLE ECB

The WAIT macro specifies an ECB, suitably initialized. The task that issues the WAIT will, not surprisingly, go into the WAIT state until some other task issues the POST macro specifying the ECB, which can be set with a code to indicate the status of the (now) completed unit of work. A good example of this is provided by the EXCP service, which is used to initiate I/O to a device under program control. The program usually issues the EXCP macro, duly followed by a WAIT macro. The EXCP macro specifies a control block (the IOB) which contains an ECB address. The WAIT macro specifies this ECB and, as a result, the high order bit (bit 0) is set on. The EXCP task then goes into the WAIT state until the I/O supervisor has dealt with the request and POSTs the ECB indicating completion. As a result the high order bit is set OFF and the adjacent bit (bit 1) is set ON. Depending on the success, or otherwise, of the I/O operation, the remaining bits (2 – 31) may be set to indicate various completion states. For instance X'7F' in byte 0 – the leftmost byte – means 'successful completion' and X'41' means some kind of error. Typical of code that exploits this is the following:

```
*   SET UP IOB
         XC    ECB,ECB
         LA    R1,IOB
         EXCP  IOB
*   DO ANYTHING ELSE THAT IS NOT DEPENDENT ON THE I/O
         WAIT  ECB=ECB1
         CLI   ECB1,X'7F'       * I/O COMPLETION OK?
         BNE   BADIO            * NO --- ERROR DIAGNOSIS/RECOVERY
*   PROCESS SUCCESSFUL I/O COMPLETION
         . . .
         . . .

IOB      DC    F'Ø'                 * AT LEAST 28 BYTES LONG
         DC    AL1(Ø),AL3(ECB1)     * LINK FROM EXCP->IOB->ECB
         DC    5F'Ø'                * BALANCE OF IOB - MUST BE
*                                   * INITIALIZED BY PROGRAM
ECB1     DC    F'Ø'
```

The above is fine when there is only one activity that will necessitate a WAIT. But what if there are several events, all of which require a wait and any one of which could complete ahead of the others?

WAIT WITH AN ECB LIST

This is applicable when there are a predictable number of activities, all of which have some synchronization requirements with the task in question. For instance an application can issue EXCP requesting WRITE and READ to two separate devices (as well as a WTOR) and then WAIT for any of them to complete. There is no guarantee as to the sequence of completion, so the application has to be able to handle any combination that might arise. A unique ECB has to be associated with each activity and the WAIT macro is set up with a list of the ECB addresses as follows:

```
         XC    WTORECB,WTORECB      * ENSURE WTOR ECB 'CLEAN'
         WTOR  'ENTER ANY CHAR TO TERMINATE PROCESSING',ANS,1,WTORECB
*              SET UP READ PROCESS
         XC    READECB,READECB      * ENSURE READ ECB 'CLEAN'
         EXCP  READIOB
*              SET UP WRITE PROCESS
         XC    WRITEECB,WRITEECB    * ENSURE WRITE ECB 'CLEAN'
         EXCP  WRITEIOB
*        ANY OTHER CODE THAT CAN BE EXECUTED BEFORE COMPELLED TO WAIT
```



```
*
*
         WAIT  1,ECBLIST=WAITLIST
*
         TM    WRITEECB,X'4Ø'       * DID WRITE COMPLETE?
         BO    PROCWRIT             * YES -- BRANCH
         TM    READECB,X'4Ø'        * NO --- DID READ COMPLETE?
         BO    PROCREAD             * YES -- BRANCH
         B     PROCWTOR             * NO --- ASSUME WTOR POPPED

READIOB  DC    F'Ø'                 * AT LEAST 28 BYTES LONG
         DC    AL1(Ø),AL3(READECB)  * LINK FROM EXCP->IOB->ECB
         DC    5F'Ø'                * BALANCE OF IOB - MUST BE
*                                   * INITIALISED BY PROGRAM
WRITEIOB DC    F'Ø'                 * AT LEAST 28 BYTES LONG
         DC    AL1(Ø),AL3(WRITEECB) * LINK FORM EXCP->IOB->ECB
         DC    5F'Ø'                * BALANCE OF IOB - MUST BE
*                                   * INITIALIZED BY PROGRAM
WAITLIST DS    ØF                   * ALIGN
         DC    A(WRITEECB)          * ADDRESS OF ECB OF WRITE
         DC    A(READECB)           * ADDRESS OF ECB OF READ
         DC    A(WTORECB+X'8ØØØØØØØ') * ADDRESS OF ECB OF WTOR
*                                   * NOTE: HI ORDER BIT SET ON TO
*                                   * SHOW END OF LIST
WRITEECB DC    A(Ø)                 * ALWAYS SET (AND RESET) TO
READECB  DC    A(Ø)                 * ZERO BEFORE WAIT IS ISSUED
WTORECB  DC    A(Ø)                 *
ANS      DC    X'ØØ'                * WTOR RESPONSE PLACED HERE
```

The first parameter of the WAIT macro is a numeric value (the default is 1), which specifies the number of ECBs that must be POSTed before the task becomes ‘dispatchable’, ie leaves the wait state. Only on very rare occasions, where particular conditions arise, will this value be other than 1. The act of ‘collecting up all of the ECBs and waiting for something to happen’ has a number of consequences.

The ECBLIST effectively mandates that you identify, at the time of design, precisely which ECBs (and how many) are going to be used. The nature of ECBLIST processing also implies that you construct it either at assembly time, or as part of program initialization – rebuilding it prior to each WAIT is not a productive use of processor time.

Each ECB has to be modified with the RB address. This could involve some additional storage access if the ECBs are scattered through the program. The processing that occurs after the WAIT has completed implies a definite priority of the activities. The program has to determine which ECB has completed, and if others have completed during the processing of the first.

None of these consequences is difficult to accommodate, but must be factored into the design. If you can accurately predict the number of ECBs in the application the ECBLIST option may prove suitable. You will, no doubt, form your own views on the impact on performance of multiple ECBs, possibly scattered across memory, all of which have to be updated with the PRB address each time the WAIT macro is issued.

THE EVENTS MACRO

The EVENTS Macro enables the user to address all of the issues covered by the two variants of the WAIT macro and also enables the user to cater for an unpredictable number of tasks seeking to inter-communicate and synchronize. It will not give you *carte blanche* (you must have an idea on the number that can concurrently be active), but it does provide far greater flexibility. In addition, POSTed ECBs are 'returned' in their POST sequence. This is to be compared with the ECBLIST option, which has an implied priority – a further design consideration. Programming with the EVENTS macro is divided into four distinct processes:

- Creating an EVENTS table
- Deleting an EVENTS table
- Initializing ECBs and linking them to the EVENTS table
- Processing POSTed ECBs.

All of these are achieved with variants of the EVENTS macro. At the end of this article there is a small program which demonstrates the usage of the various EVENTS services. It contains a number of SNAP macros to show how the ECBs and EVENTS table are affected by the varieties of service calls. Extracts of the code and SNAP output will be used to illustrate the notes describing each of the four processes. Included in the program are 10 ECBs, (ECB1 – ECB10) whose contents will also be examined as the various EVENTS services execute. In a number of cases, the notation (RX) will be employed. This indicates that the EVENTS macro may specify the parameter in question with a value contained in a register. The permitted registers are 2–12 inclusive.

CREATING AN EVENTS TABLE

A EVENTS table is created by specifying the EVENTS macro and supplying the number of table entries to be created in either of the following forms:

```
EVENTS    ENTRIES=20      OR . . .
EVENTS    ENTRIES=(RX)    FOLLOWED BY . . .

   ST     R1,EVENTAD   SAVE ADDRESS OF EVENTS TABLE
```

The number of entries can be specified 'as is' or can be contained in a register (RX). The EVENTS service will create an EVENTS table (in protected storage below 16MB) and return its address in register 1. This address must be specified in all of the other forms of the EVENTS macro. The EVENTS table is of the format: Header (currently 40 bytes) + N * 4-byte entries (N is the value supplied to the macro). You may use the IHAEVNT macro to provide mapping for the fields. In the above case, the EVENTS table created will be (40 + 20*4) = 100 bytes long and has the following format:

```
 EVENTS TABLE AFTER SETUP
008E7E40                                                     00000000 008E73C8
008E7E60 00000000 008E7E80 008E7ECC 008E7E7C    00000078 00000000 00000000 00000000
008E7E80 00000000 008FFD08 00005E28 00005D50    00005E04 000000B4 00005D50 008D57A4
008E7EA0 008D574C 008D57A8 008D5774 00005D50    008D5410 00CD58FA 00005E28 008D5580
008E7EC0 008D55C8 00CD5F6F 00000000 00000000    00000000
```

The ECBs are initially set as shown below:

```
00006D60                   00000000 00000000    00000000 00000000 00000000 00000000
00006D80 00000000 00000000 00000000 00000000
```

The first entry in the EVENTS table (at address 008E7E80) has a value of zero. This is significant, while the content of the rest of the entries is of no relevance. The important detail is that the first entry is set to a value acceptable to subsequent EVENTS service requests.

If you do not take any specific action, EVENTS tables created by a program will be removed on task termination. If you are of a tidy disposition, you can remove it in-line using one of the following forms:

```
EVENTS   ENTRIES=DEL,TABLE=(RX)
EVENTS   ENTRIES=DEL,TABLE=EVENTAD
```

RX is a register that contains the address of the table created earlier.

EVENTAD is a word which itself contains the address of the table created earlier. It is easy to overlook the fact that the two forms are not identical. In all forms of the EVENTS macro where the TABLE keyword is supported, both of these variants are permitted. The TABLE=EVENTAD form will be used throughout this article, for clarity.

Initializing ECBs and linking them to the EVENTS table

Prior to specifying an ECB for any process, you must set it such that it is unambiguously available for such a purpose. In order to achieve this, the first two bits (bit 0 and bit 1), must be set to B'00' before the ECB is used. In practice, it is just as convenient to clear out the whole word. Examples shown here will employ the XC instruction. Other techniques are available to be used if you choose. The important issue is the setting of the WAIT and POST bits.

In order for the EVENTS services to be invoked when an ECB is POSTed, it is necessary to link the ECB to the EVENTS table. This is achieved by an alternative form of the EVENTS macro:

```
EVENTS   TABLE=EVENTAD,ECB=ECB1   OR . . .
EVENTS   TABLE=EVENTAD,ECB=(RX)   RX CONTAINS THE ECB ADDRESS
```

The result of this is that the address of the EVENTS table is placed into the ECB (not the PRB address as before) and the WAIT bit (bit 0) and the EVENTS bit (bit 31) are also set on. After ECB1 is processed in this manner, the EVENTS table storage is unchanged and the ECB storage has the following format:

```
PRIME ECBS TO LINK WITH EVENTS TABLE

00006D60                  808E7E59 00000000    00000000 00000000 00000000 00000000
00006D80 00000000 00000000 00000000 00000000
```

There are two points to note at this juncture:

- The low order bit is set ON. This enables any POST service to determine that this ECB is linked to an EVENTS table, rather than a PRB/SVRB (where the low order bit would be set to OFF).
- The act of linking the ECB to the EVENTS table should be the LAST ECB-related action prior to the eventual POSTing by the relevant service. This is because the modification of the ECB by

the EVENTS service must be in place for the subsequent POST to operate correctly.

Some functions, for example BSAM READ/WRITE, which employ a DECB (which contains an ECB) modify the ECB during their processing. Thus the following sequence would not work correctly:

```
XC     ECB1,ECB1
       EVENTS   TABLE=XXX,ECB=ECB1
       READ       DECB1
```

In order to ensure correct operation, the following sequence is recommended:

- Initialize the ECB
- Invoke the service that may result in ECB modification
- Issue the EVENTS TABLE=. . .,ECB=. . .   macro

The sample program issues the EVENTS macro for each of the ten ECBs. As before, the EVENTS table is unchanged, but the ECBs now have the following format:

```
00006D60                   808E7E59 808E7E59    808E7E59 808E7E59 808E7E59 808E7E59
00006D80 808E7E59 808E7E59 808E7E59 808E7E59
```

PROCESSING POSTED ECBS

At some subsequent point in time the service on which each ECB is WAITing will be deemed complete and it will be POSTed, using code such as:

```
POST    ECB1                OR . . .
POST    ECB1,X'3FFFFFF'
```

The POST service will examine the ECB and, since its low-order bit is on, will access the EVENTS table, whose address was placed in the ECB by the EVENTS TABLE=. . ., ECB=. . . macro. It will then:

- Add the ECB address to the first available entry in the EVENTS Table.
- Set the ECB WAIT bit OFF.
- Set the ECB POST bit ON.
- Complete the ECB with the completion code, if supplied, or set

the default of zero. The ECB address, as stored in the EVENTS table, will have its high-order bit set on. This is the 'end of list' indicator. If the EVENTS table were already populated with one or more entries, the last of these would have had the corresponding bit set on. The addition of another entry results in the 'end of list' flag being 'moved on'. Thus POSTing ECB1 will result in the following:

```
 AFTER POSTING OF ECB1
ØØ8E7E4Ø                                                        ØØØØØØØØ ØØ8E73C8
ØØ8E7E6Ø ØØØØØØØØ ØØ8E7E8Ø ØØ8E7ECC ØØ8E7E8Ø    ØØØØØØ78 ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ
ØØ8E7E8Ø 8ØØØ6D68 ØØ8FFDØ8 ØØØØ5E28 ØØØØ5D5Ø    ØØØØ5EØ4 ØØØØØØB4 ØØØØ5D5Ø ØØ8D57A4
ØØ8E7EAØ ØØ8D574C ØØ8D57A8 ØØ8D5774 ØØØØ5D5Ø    ØØ8D541Ø ØØCD58FA ØØØØ5E28 ØØ8D558Ø
ØØ8E7ECØ ØØ8D55C8 ØØCD5F6F ØØØØØØØØ ØØØØØØØØ    ØØØØØØØØ

ØØØØ6D6Ø                   4ØØØØØØØ 8Ø8E7E59    8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
ØØØØ6D8Ø 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
```

and POSTing ECB2 will result in:

```
Ø AFTER POSTING OF ECB2
ØØ8E7E4Ø                                                        ØØØØØØØØ ØØ8E73C8
ØØ8E7E6Ø ØØØØØØØØ ØØ8E7E8Ø ØØ8E7ECC ØØ8E7E84    ØØØØØØ78 ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ
ØØ8E7E8Ø ØØØØ6D68 8ØØØ6D6C ØØØØ5E28 ØØØØ5D5Ø    ØØØØ5EØ4 ØØØØØØB4 ØØØØ5D5Ø ØØ8D57A4
ØØ8E7EAØ ØØ8D574C ØØ8D57A8 ØØ8D5774 ØØØØ5D5Ø    ØØ8D541Ø ØØCD58FA ØØØØ5E28 ØØ8D558Ø
ØØ8E7ECØ ØØ8D55C8 ØØCD5F6F ØØØØØØØØ ØØØØØØØØ    ØØØØØØØØ

ØØØØ6D6Ø                   4ØØØØØØØ 4ØØØØØØØ    8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
ØØØØ6D8Ø 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
```

If the application continues to POST different ECBs, the EVENTS table will gradually fill up, and if more ECBS are POSTed than there are available table entries, the system will respond with an ABEND.

If an application has initiated an activity on whose completion it is dependent, it has to determine at some point when that dependency has been satisfied. Careful design can enable a degree of overlap, but there will come a time when nothing else can be done. In the earlier part of this article, the WAIT service is invoked and, once the relevant ECB is POSTed, execution resumes.

When the ECBs are linked to an EVENTS table, in the manner described above, another form of the EVENTS service is used to provide the equivalent processing. This form has two options:

```
EVENTS TABLE=XXX,WAIT=YES
EVENTS TABLE=XXX,WAIT=NO
```


At the time the macro is issued, there are two possible scenarios:

- One or more ECBs have been POSTed
- No ECBs have been POSTed.

If WAIT=YES has been coded and no ECBs have been POSTed, the task will go into a wait state until one of the dependent activities issues a POST to an ECB already linked to the EVENTS table. If WAIT=YES has been coded and ECBs have been POSTed, execution continues and R1 contains the address of the entry in the EVENTS table that points to the ECB that was POSTed first. Using the following code on the structures shown above leads to the status as shown below.

```
    EVENTS    TABLE=EVENTAD,WAIT=YES                        *    *
        LR    R2,R1             *                           *    *

 AFTER EVENTS WAIT=YES.         R2 ==> ENTRY

 GPR VALUES
        0-3  90010000  80006B9C  008E7E80  00000000
        4-7  008E97B0  008E7A70  008C5FF8  FD000000
        8-11 008E7C18  00006D58  00006D8C  008E7E58
       12-15 000077D8  000067D8  80FCE878  00000000

 008E7E40                                                 00000000 008E73C8
 008E7E60 00000000 008E7E80 008E7ECC 008E7E84    80000078 00000000 00000000 00000000
 008E7E80 00006D68 80006D6C 00005E28 00005D50    00005E04 000000B4 00005D50 008D57A4
 008E7EA0 008D574C 008D57A8 008D5774 00005D50    008D5410 00CD58FA 00005E28 008D5580
 008E7EC0 008D55C8 00CD5F6F 00000000 00000000    00000000

 00006D60                   40000000 40000000    808E7E59 808E7E59 808E7E59 808E7E59
 00006D80 808E7E59 808E7E59 808E7E59 808E7E59
```

If WAIT=NO has been coded and no ECBs have been POSTed, the EVENTS service returns control to the program with Register 1 containing 0. If WAIT=NO has been coded and one or more ECBs have been POSTed, the EVENTS service returns control to the program with Register 1 pointing to the entry in the EVENTS table that contains the address of the ECB that was posted first. As covered earlier, there may be one or more ECB addresses in the list, the last of which has its high-order bit set on. Given that one or more ECBs have been processed, it is now up to the designer of the application to determine how each is to be processed. One technique that has been used with some success is to position the address of the corresponding process routine adjacent to the ECB, and to locate any useful parameters adjacent to the address of the processing routine thus:

```
ECB1     DC    F'Ø'               * ECB POSTED WHEN ACTION IS COMPLETE
PROC1    DC    V(PROCESS1)        * ADDRESS OF PROCESSING ROUTINE
PARMØ    DC    A(PARMØ)           * PARAMETERS LOADED INTO RØ
PARM1    DC    A(PARM1)           * PARAMETERS LOADED INTO R1
```

Processing an EVENTS macro, which returns one or more POSTed ECBs, could take the form:

```
      EVENTS   TABLE=EVENTAD,WAIT=YES
         LR    R1Ø,R1             * R1Ø -> FIRST ECB ADDRESS
LOOPØØ1Ø DS    ØH                 *
         L     R9,Ø(Ø,R1Ø)        * R9 -> POSTED ECB
         LM    R15,R1,4(R9)       * R15 -> ROUTINE ADDRESS
*                                 * RØ -> PARMØ
*                                 * R1 -> PARM1
         BALR  R14,R15            * CALL ROUTINE
*                                 * HANDLE RETURN FROM ROUTINE
         LTR   R1Ø,R1Ø            * END OF LIST?
         BM    ALLDONE            * YES - NO MORE COMPLETED EVENTS
         AH    R1Ø,=H'4'          * NO --- STEP TO NEXT ENTRY
         B     LOOPØØ1Ø           * AND REPEAT FOR NEXT POSTED ECB
```

There is one last aspect of the EVENTS macro that must be considered. As mentioned earlier, if the application continues to POST ECBs, the 'high water mark' gradually steps down the table until it reaches the end, at which point it will ABEND. In order to avoid this the 'LAST=' keyword is available. This enables the application to indicate the entries that have been satisfactorily processed so that they can be removed and the remainder, if any, 'shuffled' to the start of the table. This is best illustrated by an example. Given a starting point as shown on page 64 (ECB1 and ECB2 POSTed, with EVENTS TABLE=EVENTAD,WAIT=YES), if ECB3, ECB4, and ECB5 are subsequently posted, the EVENTS table and ECBs have the format:

```
POSTING ECBS 3,4 AND 5 (3 SETS)
ØØ8E7E4Ø                                                   ØØØØØØØØ ØØ8E73C8
ØØ8E7E6Ø ØØØØØØØØ ØØ8E7E8Ø ØØ8E7ECC ØØ8E7E9Ø    8ØØØØØ78 ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ
ØØ8E7E8Ø ØØØØ6D68 ØØØØ6D6C ØØØØ6D7Ø ØØØØ6D74    8ØØØ6D78 ØØØØØØB4 ØØØØ5D5Ø ØØ8D57A4
ØØ8E7EAØ ØØ8D574C ØØ8D57A8 ØØ8D5774 ØØØØ5D5Ø    ØØ8D541Ø ØØCD58FA ØØØØ5E28 ØØ8D558Ø
ØØ8E7ECØ ØØ8D55C8 ØØCD5F6F ØØØØØØØØ ØØØØØØØØ    ØØØØØØØØ

ØØØ6D6Ø                   4ØØØØØØØ 4ØØØØØØØ    4ØØØØØØØ 4ØØØØØØØ 4ØØØØØØØ 8Ø8E7E59
 ØØØØ6D8Ø 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
```

If the program processes the completion of the first two ECBs and then issues the EVENTS macro shown above, which indicates that all entries in the EVENTS table up to and including that referencing ECB2 are 'done', the EVENTS table and the ECBs will have the format shown below. Note that the third, fourth, and fifth entries in the EVENTS table in the previous table have now been copied over the first, second, and third entries, and that Register 1 points to the first of this set. Since the high-order bit is set on the third entry, the contents of the fourth and fifth entries are of no relevance.

```
*      ASSUME R2 -> ENTRY IN EVENTS TABLE WHICH POINTS TO ECB2
       EVENTS   TABLE=EVENTAD,WAIT=YES,LAST=(R2)
          LR    R3,R1             * PRESERVE R1 CONTENTS

GPR VALUES
        Ø-3  9ØØ1ØØØØ  9ØØØ6C6Ø  ØØ8E7E84  ØØ8E7E8Ø
        4-7  ØØ8E97BØ  ØØ8E7A7Ø  ØØ8C5FF8  FDØØØØØØ
        8-11 ØØ8E7C18  ØØØØ6D58  ØØØØ6D8C  ØØ8E7E58
       12-15 ØØØØ77D8  ØØØØ67D8  8ØFCE878  ØØØØØØØØ

 AFTER EVENTS ... LAST=(R2)
ØØ8E7E4Ø                                                         ØØØØØØØØ ØØ8E73C8
ØØ8E7E6Ø ØØØØØØØØ ØØ8E7E8Ø ØØ8E7ECC ØØ8E7E88    8ØØØØØ78 ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ
ØØ8E7E8Ø ØØØØ6D7Ø ØØØØ6D74 8ØØØ6D78 ØØØØ6D74    8ØØØ6D78 ØØØØØØB4 ØØØØ5D5Ø ØØ8D57A4
ØØ8E7EAØ ØØ8D574C ØØ8D57A8 ØØ8D5774 ØØØØ5D5Ø    ØØ8D541Ø ØØCD58FA ØØØØ5E28 ØØ8D558Ø
ØØ8E7ECØ ØØ8D55C8 ØØCD5F6F ØØØØØØØØ ØØØØØØØØ    ØØØØØØØØ

ØØØØ6D6Ø                   4ØØØØØØØ 4ØØØØØØØ    4ØØØØØØØ 4ØØØØØØØ 4ØØØØØØØ 8Ø8E7E59
ØØØØ6D8Ø 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59 8Ø8E7E59
```

The EVENTS table has, to all intents and purposes been 'compressed', much in the manner of a PDS, with all of the entries being migrated to the front. If this feature of the EVENTS service is incorporated in the design, the problems associated with the 'table full' condition can be avoided.

ERROR NOTIFICATION AND HANDLING

The EVENTS service shows its vintage, inasmuch as it follows the 1970s style of 'if anything goes wrong, ABEND'. So it is very unforgiving and the designer should make every attempt to ensure that no provocation is generated. All of the abends are of the form Sx7D. A brief description of the cause and possible remedy follows:

- S17D – EVENTS table address incorrect. The table address must be specified in a register or must be specified in a fullword, the address of which is supplied in the EVENTS macro. The difference in styles can be overlooked quite easily.

- S37D – another task is waiting on the same EVENTS table. The design will be complex if this state of affairs can occur.
- S47D – the address specified in the LAST keyword in not within the range currently active for the specified table. Usually this is easily remedied.
- S57D – either the ECB address is incorrect or the ECB has a protect key differing from that of the program issuing the EVENTS macro. This is usually the former and should easily be remedied.
- S67D – the ECB is already WAITing, indicating that a previous EVENTS or WAIT macro has been issued. This is either very easy or quite challenging to solve!
- S77D – either an incompatible level of the macro (not encountered since the MVS and XA days) or an error in the macro parameters (WAIT=YES and WAIT=NO both specified). R15 will contain the values 4 and 8 respectively – but the ABEND still strikes!
- S87D – the EVENTS table is full and the new entry cannot be added. Clearly EVENTS . . . ., LAST= is required.

CONCLUSIONS

The EVENTS service, as implemented via the EVENTS macro, provides an alternative means to WAIT and POST of coordinating asynchronous activities within an application. This article sets out to demonstrate the options available and to show how they may be implemented. The reader is left with the challenge of determining how best to apply, adapt (and improve) the ideas and techniques outlined above.

```
 TITLE  ' SAMPLE PROGRAM SHOWING THE OPERATION OF THE EVENTS SERVICE'
EVENTEST CSECT
      MKBINIT                  *                             *    *
      OPEN     (EVENTDMP,OUTPUT) *                           *    *
      EVENTS   ENTRIES=20      *                             *    *
         LR    R11,R1          * PRESERVE A(TABLE)           *    *
         ST    R1,EVENTAD      * SAVE A(TABLE)               *    *
         ST    R1,SNAPL1ST     * SAVE IN SNAP LIST           *    *
         A     R1,=A(40+(20*4))  * LENGTH OF EVENTS TABLE    *    *
         ST    R1,SNAPL1EN     * SAVE END OF SNAP PARM LIST  *    *
         LA    R9,SNAPL1ST     * SET UP FOR SNAP TYPE 1      *    *
     SNAP      DCB=EVENTDMP,ID=1,PDATA=(REGS),                     +
               LIST=(R9),STRHDR=LABEL1                       *    *
```

```
EVENTS    TABLE=EVENTAD,WAIT=NO                                 *    *
    LR    R3,R1                                                 *    *
SNAP      DCB=EVENTDMP,ID=2,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL2                               *    *
    LA    R10,ECB1          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB2          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB3          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB4          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB5          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB6          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB7          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB8          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB9          * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
    LA    R10,ECB10         * SET UP ECB ADDRESS                *    *
EVENTS    TABLE=EVENTAD,ECB=(R10)                               *    *
SNAP      DCB=EVENTDMP,ID=3,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL3                               *    *
POST      ECB1              *                                   *    *
SNAP      DCB=EVENTDMP,ID=4,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL4                               *    *
POST      ECB2              *                                   *    *
SNAP      DCB=EVENTDMP,ID=5,PDATA=(REGS),                             +
          LIST=(R9),STRHDR=LABEL5                               *    *
EVENTS    TABLE=EVENTAD,WAIT=YES                                *    *
   LR    R2,R1              *                                   *    *
SNAP      DCB=EVENTDMP,ID=6,PDATA=(REGS),                             +
```



```
               LIST=(R9),STRHDR=LABEL6                      *    *
          POST      ECB3              *                     *    *
          SNAP      DCB=EVENTDMP,ID=7,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL7                 *    *
          POST      ECB4              *                     *    *
          SNAP      DCB=EVENTDMP,ID=7,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL7                 *    *
          POST      ECB5              *                     *    *
          SNAP      DCB=EVENTDMP,ID=7,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL7                 *    *
            LA      R2,4(0,R2)        * STEP TO ECB2 ENTRY IN TABLE  *    *
          EVENTS    TABLE=EVENTAD,WAIT=YES,LAST=(R2)        *    *
            LR      R3,R1             * PRESERVE R1 FOR SNAP   *    *
          SNAP      DCB=EVENTDMP,ID=8,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL8                 *    *
          EVENTS    TABLE=EVENTAD,WAIT=NO                   *    *
            LR      R3,R1             * PRESERVE R1 FOR SNAP   *    *
          SNAP      DCB=EVENTDMP,ID=9,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL9                 *    *
          EVENTS    TABLE=EVENTAD,WAIT=NO                   *    *
            LR      R3,R1             * PRESERVE R1 FOR SNAP   *    *
          SNAP      DCB=EVENTDMP,ID=9,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL9                 *    *
          EVENTS    TABLE=EVENTAD,WAIT=NO                   *    *
            LR      R3,R1             * PRESERVE R1 FOR SNAP   *    *
          SNAP      DCB=EVENTDMP,ID=9,PDATA=(REGS),                +
                    LIST=(R9),STRHDR=LABEL9                 *    *
          EVENTS    ENTRIES=DEL,TABLE=EVENTAD               *    *

          CLOSE     EVENTDMP          *                     *    *
          QUIT      0                 *                     *    *
            LTORG                     *                     *    *
EVENTAD   DC    A(0)                  * A(EVENTS) TABLE     *    *
          DS    0F                    * ALIGN THINGS        *    *
SNAPL1ST  DC    A(0)                  * A(START EVENT TABLE)   *    *
SNAPL1EN  DC    A(0)                  * A(END OF EVENT TABLE)  *    *
          DC    A(ECB1)               * A(START OF ECBS)    *    *
          DC    A(ECB10+X'80000000') * A(END OF ECBS)       *    *
ECB1      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB2      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB3      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB4      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB5      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB6      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB7      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB8      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB9      DC    A(0)                  * SET OF 10 ECBS      *    *
ECB10     DC    A(0)                  * SET OF 10 ECBS      *    *
LABEL1    DC    A(L1),X'80000000'                           *    *
LABEL2    DC    A(L2),X'80000000'
LABEL3    DC    A(L3),X'80000000'
LABEL4    DC    A(L4),X'80000000'
LABEL5    DC    A(L5),X'80000000'
```

```
LABEL6   DC    A(L6),X'80000000'
LABEL7   DC    A(L7),X'80000000'
LABEL8   DC    A(L8),X'80000000'
LABEL9   DC    A(L9),X'80000000'
L1       DC    AL1(50),CL50' EVENTS TABLE AFTER SETUP'          *    *
L2       DC    AL1(50),CL50' AFTER EVENTS WAIT=NO  (R3 HAS RESULT)  '
L3       DC    AL1(50),CL50' PRIME ECBS TO LINK WITH EVENTS TABLE   '
L4       DC    AL1(50),CL50' AFTER POSTING OF ECB1                  '
L5       DC    AL1(50),CL50' AFTER POSTING OF ECB2                  '
L6       DC    AL1(50),CL50' AFTER EVENTS WAIT=YES. R2 ==> ENTRY    '
L7       DC    AL1(50),CL50' POSTING ECBS 3,4 AND 5 (3 SETS)        '
L8       DC    AL1(50),CL50' AFTER EVENTS ... LAST=(R2)             '
L9       DC    AL1(50),CL50' AFTER EVENTS WAIT=NO  (R3 HAS RESULT)  '
EVENTDMP DCB   DDNAME=EVENTDMP,MACRF=(W),RECFM=VBA,LRECL=125,DSORG=PS, +
               BLKSIZE=882
         END
```

Arthur Nargs
Systems Programmer (UK)

Software and hardware dependencies

Did you ever have a problem when a program from the test environment is moved to the production environment and does not work as expected? My problem occurred when I wanted to transfer a simple program and one JCL with REPRO inside, from the test to the production environment. The program simply reads a large VSAM dataset (with several million records) and does some changes the input records and writes them to the output dataset. The JCL was even simpler. It just performed a large insert in the VSAM dataset.

In the test environment everything worked well. The insert processes had been redesigned to make the batch window much shorter, resulting in considerable performance enhancements. So there was a surprise after its first run in a production environment where the new insert time was twice as slow as the test environment.

We needed to find the reasons for this performance degradation. We looked at dataset definitions, with catalog searches, looking at definitions of storage and management classes and dependencies with other jobs that run at the same time. Because the VSAM dataset was also defined in the CICS environment, we also tried to find any reason for the slow down in the CICS definitions, but we could find nothing wrong. Evenually we discovered that the reason for the slow running of our tasks was related to the hardware/software combination. We were alerted after watching for differences between the testing and production run. We found out that there was a considerable difference in the SIO indicator. This revealed that there was a problem reading from input and writing in to the output dataset. With further research we determined that:

- There was a difference between disks used for our test and production datasets. The production disks were older and the test disks were a recent acquistion. The new disks were much faster causing a performance difference between the environments.
- Our production datasets were located on the same disks as the system software. System software is in permanent use so throughput on these disks with system software is much lower. The solution was to isolate the system software on some disks and not to define any other datasets there.

Therefore, when we defined our datasets on new faster disks with no system software on them, there were considerable performance improvements. So, having solved our basic problem, we continued to look for additional time savings. Again we looked at the hardware. Disks are connected in disk arrays, and every disk array has its own I/O channels for transferring data to and from the CPU. With more I/O channels defined on a disk array, more data can be transferred to and from disks faster. We found out that our datasets were defined on the disk array with only one I/O channel, so when we increased the number to the maximum of four, our jobs ran much faster.

Even if you are satisfied with the performance of your production environment maybe you can spend some time to check if you can push programs to run a little bit faster. All you need is to check what kind of hardware you have, its organization, and whether you have unused features. I am sure that with a little effort, you can save time during the batch window. This can be very important in data centres that do not have the latest most powerful hardware and software products, where programmers are trying to extract the last CPU cycle to make a modern and efficient information system.

Predrag Jovanovic
Project Developer
Pinkerton Computer Consultants Inc (USA)

e3Sciences have developed a new set of products called PassGen which generate one-time passwords for secure logon to z/OS, OS/390, Unix, and Firewall systems. Two versions of the PassGen family are availiable now; one designed for Windows users and the other for Palm OS users.

Passgen is a toolkit to generate one-time passwords. Passgen can be used to authenticate users in a secure manner across insecure networks, because once the password is used it can never be used again. This system enables users to generate and manage all the passtickets and S/KEYs generated for applications.

PassGen provides two one-time password systems the IETF One Time Password Standard: S/KEY (RFC 1760). Most Unix and Firewall systems provide support for this standard, and the IBM Security Server (RACF) Passticket algorithm, available in RACF, CA-ACF2 and CA Top-Secret. This provides secure logon to IBM mainframe systems. The Passticket can be used as a direct replacement for static Passwords; no changes are required to existing applications.

For further information contact:

e3Sciences Ltd, Kingston House, Kings Stanley, Gloucestershire, GL10 3JF, UK.

Tel: (0870) 870 6166
Fax: (01280) 706 583

http://www.e3Sciences.com

* * *

Computer Associates has released CA-ACF2 6.4 and CA-Top Secret 5.2 security applications, which integrate application-level control across both mainframe and distributed platforms, including Unix, Windows NT, and Linux.

The new releases include the eTrust LDAP Server, letting users of both products use LDAP to query and update mainframe security information for use on distributed systems. Administrators will be able to use the authentication policies established within ACF2 and Top Secret implementations and apply them to any application using TCP/IP. Also, users can access information stored in DB2 UDB Server, commercial and in-house LDAP directories, popular third-party mainframe security solutions, and other back-end data stores to help create a single source of security information.

For further information contact:

Computer Associates plc, Ditton Park, Riding Court Road, Datchet, Slough, Berkshire, SL3 9LL, UK.

Tel: (01753) 577 733
Fax: (01753) 825 464

Computer Associates International, One Computer Associates Plaza, Islandia, NY 11749, USA.

Tel: (631) 342 6000
Fax: (631) 342 6800

http://www.ca.com

* * *