LOGGRASM Information Logger for Assembler

Version V1R1M17

User Reference

Table of Contents

1.0 Concepts

1.1 Description

LOGGRASM is a utility to perform basic source setup of an Assembler program, and to log basic information about an Assembler program during execution. This utility provides a method to show program execution in Assembler language programs. The log information is written to a data set using QSAM. LOGGRASM is for use in an Assembler batch program. It can run as authorized or as non-authorized, though it uses no authorized services. LOGGRASM is a basic tool to assist students and new programmers learning Assembler on the mainframe.

For a fast-track overview of what LOGGRASM does and the output it produces before you decide if you want to do an install, you can jump ahead and review Section 8 ‘Description of Output Report’.

1.2 Prerequisites

Students need a basic working knowledge of a z/OS environment, and a basic knowledge of TSO/ISPF Editor commands, and know some JCL (e.g., Job Control Language). You should have some introductory programming skills in Assembler. You need access to a z/OS system, and be able to logon under TSO.

1.3 Input

LOGGRASM uses input control cards from the //LGRSYSIN DD for directing the processing of Logger Services during execution of a user program.

1.4 Output

LOGGRASM writes its output to a QSAM DCB with RECFM=FBA and LRECL=133 when a //LGRECOUT DD is present in the JCL. LOGGRASM output is in mixed case.

1.5 Source

The LOGGRASM Assembler source code, Assembler macros, and JCL are fully available to you to make any changes or customizations to LOGGRASM in any manner you wish.

1.6 Assembly/Link-Edit Information

LOGGRASM:

```
OS....................z/OS V1R7 and higher
Environment...........HLASM z/Architecture
Non-IBM Macros Used...LPGMNTRY, LPGMEXIT, LPGMSUBE, LPGMSUBX,
                      #LGPOINT, LCA
AMODE.................64
RMODE.................ANY
LKED Attributes.......RENT
Size of Load-Modules..Approximately 270K
Authorization.........None required
Memory Requirement....Approximately 468K
```

This means that LOGGRASM requires z/Architecture and z/OS V1R7 or higher. Refer to Section 2.4 when you have a level of z/OS less than V1R10.

LOGGRASM services execute in 64-bit addressing mode. User programs may operate in a different addressing mode from that of LOGGRASM without restriction. LOGGRASM will return in the addressing mode of the user program. LOGGRASM is re-entrant.

For the Binder, COMPAT=PM4 (ZOSV1R3, ZOSV1R4) is the minimum level that can be specified in your link-edit.

1.7 Compatibility

LOGGRASM supports z/OS V1R7 and above with z/Architecture (z800, z890, z900, z990, z9, z10, zEnterprise 196, or equivalent). Many new instructions have been made available, but are not used in the LOGGRASM assembler source code which incorporates the use of only zArchitecture base instructions. This was done in order for LOGGASM to maintain operability on z800 machine frames without the engineering upgrades or z/890 cages needed to support some of the newer z/Architecture instructions. Also, certain facilities may not necessarily be available or installed on every CPU model.

Even though it would be more efficient to use the new instructions, this LOGGRASM design to use only the base zAchitecture instruction set will allow the LOGGRASM service to execute on older CPU models or other CPU models where the long-displacement facility, extended immediate facility, general-instructions extension facility, move-with-optional-specifications facility, parsing-enhancement facility, the execute-extensions facility, the high-word facility, or other facilities may not be installed. This also removes any hardware dependencies Logger Services would have if the more advanced instruction sets were used, and Logger Services does not need them to perform its functions anyway.

However, there is no restriction on the user. Students using LOGGRASM may code any instruction in their own Assembler program which is available to them based on the hardware facilities installed on their box.

At the object level, LOGGRASM is fully downward-compatible and up-ward compatible for previously existing features but not new ones. What this means is if LOGGRASM has been assembled on an z/OS system at level x, it will be fully compatible with an older z/OS system. The same object code will execute on a newer system but without the new functionality.

For example, if LOGGRASM is assembled on an z/OS V1R10 system and running on an z/OS V1R7 system, it will produce the same output as if it had been assembled on a V1R7 system. However, if the same object was assembled on V1R10 and executes on an z/OS V1R13 system, it may not produce the same output as if the same LOGGRASM source had been assembled on an V1R13 system.

If you intend to run LOGGRASM on different z/OS levels, you may do so with the objects. It is recommended that you assemble it on the most recent z/OS level.

To ease some of the z/OS level dependencies, there are two JCL members for assembly of the source code. For z/OS V1R10 and above use member ASMLOGGR to assemble the LOGGRASM source code and produce the program objects. For less than z/OS V1R10, then use member ASMLOGG2.

APAR Notes:

1. To correct a S0C4 abend condition in the LOGGRASM ESTAEX recovery routine, you need to ensure the following IBM APAR has been applied to your z/OS V1R9 system:

   OA23937 - RTM fails to provide a 144 byte save area at HBB7740

2. To assemble with ASMLOGG2 the following IBM APAR for z/OS V1R7 is no longer required:

   OA13756 - The macro IHAEXLST was erroneously omitted from being shipped in V1R7, and made available with OA13756.

1.8 Components

The LOGGRASM utility on CBT Tape has the following 51 components:

Version, documentation, and fix information:

```
$$$#DATE.......CBT version file containing a date and time stamp,
               and total number of records in the XMI file.
$$DOC..........The Documentation File in EBCDIC Text format.
$$DOCWRD.......The Documentation File in Microsoft Word format
$$DOCPDF.......The Documentation File in Adobe Acrobat PDF format
$$FIXPAC.......Detail Description of LOGGRASM Changes and Bug Fixes
$$NOTE1........Fix Pack Summary Information for V1R1M05
$$NOTE2........Fix Pack Summary Information for V1R1M06
$$NOTE3........Fix Pack Summary Information for V1R1M07
$$NOTE4........Fix Pack Summary Information for V1R1M08
$$NOTE5........Fix Pack Summary Information for V1R1M09
$$NOTE6........Fix Pack Summary Information for V1R1M10
$$NOTE7........Fix Pack Summary Information for V1R1M11
$$NOTE8........Fix Pack Summary Information for V1R1M11-PTF 1
$$NOTE9........Fix Pack Summary Information for V1R1M12
$$NOTE10.......Fix Pack Summary Information for V1R1M13
$$NOTE11.......Fix Pack Summary Information for V1R1M14
$$NOTE12.......Fix Pack Summary Information for V1R1M15
$$NOTE13.......Fix Pack Summary Information for V1R1M15
$$NOTE14.......Fix Pack Summary Information for V1R1M16
$$NOTE15.......Fix Pack Summary Information for V1R1M17
@FILE757.......CBT file with short description of the CBT757 file
```

Job Control Language (JCL) for Allocation and Assemble:

```
ASMALLOC.......JCL to Allocate Data Sets for Installation of LOGGRASM
ASMLOGGR.......JCL to Assemble LOGGRASM Source Programs (V1R10 or higher)
ASMLOGG2.......JCL to Assemble LOGGRASM Source Programs (V1R9 or less)
```

Assembler Program Source Code for Logger Services:

```
LGMHDCBX.......Logger Services Data Control Block (DCB) Abend Exit
LGMHESTA.......Logger Services Abend Recovery Exit
LGMHESTX.......Logger Services Abend Recovery Exit (v1R10 or less)
LGMHLB64.......Logger Services Above-the-Bar Log Record Buffering
LGMHLRCB.......Logger Services Above-the-Line Log Record Buffering
LGMHLRCE.......Logger Services Log Record Event Publishing
LGMHLRCI.......Logger Services Initialization
LGMHLRCX.......Logger Services Initialization (V1R10 or less)
LGMHLRCT.......Logger Services Termination
LGMHPSTG.......Logger Services Private Area Storage Check
LGMHPSTX.......Logger Services Private Area Storage Check (V1R10 or less)
LGMHRTRY.......Logger Services Diagnostic ESTAEX Retry
LGMHRTRX.......Logger Services Diagnostic ESTAEX Retry (V1R10 or less)
LGMHWTCH.......Logger Services Storage Watch
LGCPLOGR.......Logger Services User Program CopyBook
LGCPLSWA.......Logger Services Work Area Copybook
```

Assembler Source Code for Sample Programs:

```
TESTRISC.......Sample Assembler Tutorial Program
USERPGM1.......Sample Assembler Program 1 (24-bit)
USERPGM2.......Sample Assembler Program 2 (31-bit)
USERPGM3.......Sample Assembler Program 3 (64-bit)
USERPGM4.......Sample Assembler Program 4 (31-bit)
USERPGM5.......Sample Assembler Program 5 (31-bit)
USERPGM6.......Sample Assembler Program 6 (31-bit)
```

Job Control Language (JCL) for Sample Programs:

```
USERJOB1.......Sample JCL to Assemble, Link, and Execute Sample Pgm 1
USERJOB2.......Sample JCL to Assemble, Link, and Execute Sample Pgm 2
USERJOB3.......Sample JCL to Assemble, Link, and Execute Sample Pgm 3
USERJOB4.......Sample JCL to Assemble, Link, and Execute Sample Pgms 4, 5,
               and 6
```

If you are going to upgrade from a fix pack and have already done a basic installation from a previous release, then the minimum number of components you need to run Logger Services in your user program are just the 12 components listed below. Use the ASMLOGGR JCL to create the 10 new object decks from the Assembler source programs and include those objects in the //SYSLIB DD of your Link-Edit, make sure to include the 2 copybooks in your Assembler //SYSLIB DD, and then re-assemble your existing Assembler language programs which use Logger Services in order to pick up the new updates in Logger Services.

```
 LGMHDCBX.......Logger Services Data Control Block (DCB) Abend Exit
 LGMHESTA.......Logger Services Abend Recovery Exit
*LGMHLB64.......Logger Services Above-the-Bar Log Record Buffering
 LGMHLRCB.......Logger Services Above-the-Line Log Record Buffering
 LGMHLRCE.......Logger Services Log Record Event Publishing
 LGMHLRCI.......Logger Services Initialization
 LGMHLRCT.......Logger Services Termination
 LGMHPSTG.......Logger Services Private Area Storage Check
 LGMHRTRY.......Logger Services Diagnostic ESTAEX Retry
 LGMHWTCH.......Logger Services Storage Watch
 LGCPLOGR.......Logger Services User Program CopyBook
 LGCPLSWA.......Logger Services Work Area Copybook
```

* New program for V1R1M17

2.0 Installation

A basic summary of the installation is as follows:

1. Unzip the CBT-Tape file to your PC to extract the FILE757.XMI file.
2. Transfer the FILE757.XMI file in binary to a mainframe sequential data set.
3. Receive the xmit data set to a PDS
4. Run the ASMLOGGR job or ASMLOGG2 job to create the objects.
5. Run sample programs and review doc to become familiar with LOGGRASM.
6. You are ready to code your new Assembler programs using LOGGRASM.

2.1 Step 1 – Unzip File757 and Allocate Data Sets on Mainframe

Use PKZIP, WINZIP, ZTREE (or similar program) to inflate the CBT757.Zip file.

The LOGGRASM copybooks, source, JCL, and program samples are for use on the z/OS platform. The unzip process on your PC will extract a file whose name will be "FILE757.XMI". You need to upload the FILE757.XMI file from the PC to your target mainframe system.

Before you send the FILE757.XMI file to the mainframe as binary, you first need to allocate data sets on the mainframe using ISPF 3.2 or IEFBR14 to process the upload if you have no qualified pre-existing data sets.

Allocate on the mainframe a data set to receive the FILE757.XMI file from the PC. For example:

```
Data Set Name . . . : USER.WORK.RECEIVE    ←(Use name of your choice)
Organization  . . . : PS
Record format . . . : FB
Record length . . . : 80
Block size  . . . . : 3120
1st extent cylinders: 15
Secondary cylinders : 2
Directory blocks. . : 0
Data set name type  : Physical Sequential
```

To receive the contents of data set USER.WORK.RECEIVE, allocate on the mainframe a second data set which will be used to hold the copybooks, source, JCL, and samples For example:

```
Data Set Name . . . : USER.WORK.CNTL      ←(Use name of your choice)
Organization  . . . : PO
Record format . . . : FB
Record length . . . : 80
Block size  . . . . : 27920
1st extent cylinders: 15
Secondary cylinders : 5
Directory blocks. . : 20
Data set name type  : Partitioned
```

Allocate on the mainframe a third data set which will be used to hold the program objects from the assembly. For example:

```
Data Set Name . . . : USER.WORK.OBJECT   ←(Use name of your choice)
Organization  . . . : PO
Record format . . . : FB
Record length . . . : 80
Block size  . . . . : 27920
1st extent cylinders: 5
Secondary cylinders : 2
Directory blocks. . : 10
Data set name type  : Partitioned
```

Allocate on the mainframe a fourth data set which will be used to hold the load module from the Link-Edit. For example:

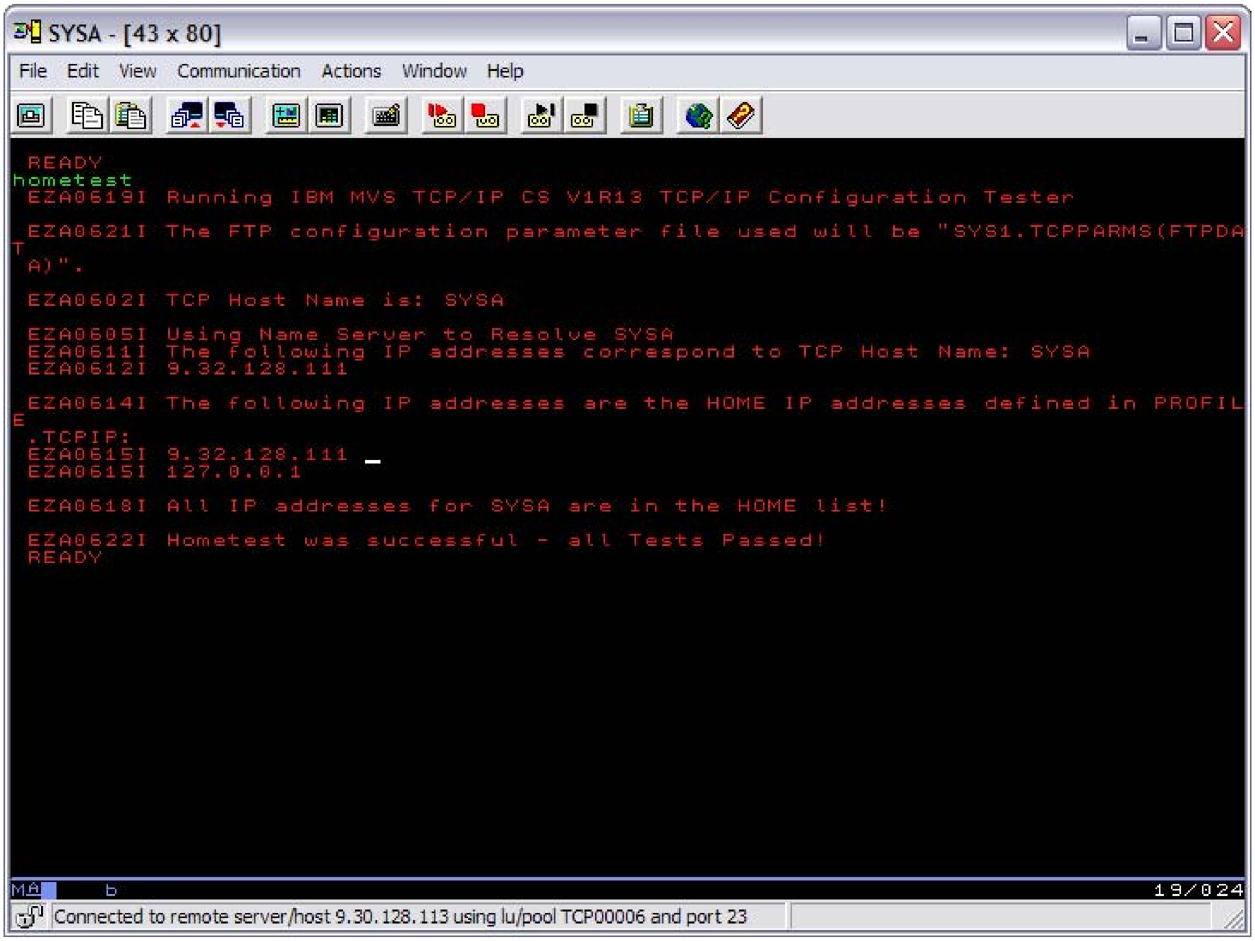
```
Data Set Name . . . : USER.WORK.LINKLIB  ←(Use name of your choice)
Organization  . . . : PO
Record format . . . : U
Record length . . . : 0
Block size  . . . . : 32760
1st extent cylinders: 5
Secondary cylinders : 2
Directory blocks. . : 10
Data set name type  : Partitioned
```

2.2 Step 2 – Upload File757 to Mainframe Data Set

The upload of the FILE757.XMI file goes from your PC to the mainframe into a sequential data set. This sequential data set on the mainframe will be the receive data set you allocated in the previous step (e.g., 'USER.WORK.RECEIVE'). Please remember that the transfer from the PC to the mainframe for this particular file must be in BINARY mode meaning ASCII-to-EBCDIC translation is turned OFF and no CRLF.

Use any file transfer facility in your shop (IND$FILE, FTP, XCom, Extra, etc.) or use the file transfer utilities on your 3270 emulator programs running on the PC (IBM PComm, Attachmate, Rumba, etc.). The DCB attributes for the sequential data set must be RECFM=FB, LRECL=80, BLKSIZE=n*80. You will use the TSO RECEIVE command on the mainframe in a later step to restore the transmit file into a PDS library structure on the mainframe.

Your instructor should have already supplied you with the TCP address for your host mainframe system which would be needed anyway to perform a TSO logon to the mainframe system through your 3270 emulator program. When logged on to TSO you can enter 'hometest' from the TSO command line to display the IP address assigned to your z/OS system. Your TCP/IP value will be different than the example displayed here which shows 9.32.128.111.



```
 READY
hometest
 EZA0619I Running IBM MVS TCP/IP CS V1R13 TCP/IP Configuration Tester

 EZA0621I The FTP configuration parameter file used will be "SYS1.TCPPARMS(FTPDA
T
 A)".

 EZA0602I TCP Host Name is: SYSA

 EZA0605I Using Name Server to Resolve SYSA
 EZA0611I The following IP addresses correspond to TCP Host Name: SYSA
 EZA0612I 9.32.128.111

 EZA0614I The following IP addresses are the HOME IP addresses defined in PROFIL
E
 .TCPIP:
 EZA0615I 9.32.128.111
 EZA0615I 127.0.0.1

 EZA0618I All IP addresses for SYSA are in the HOME list!

 EZA0622I Hometest was successful - all Tests Passed!
 READY
```

```
MA      b                                                              19/024
```

Connected to remote server/host 9.30.128.113 using lu/pool TCP00006 and port 23

The following shows a transmit of the CBT-Tape file named FILE757.XMI from the PC to the receive data set (e.g., USER.WORK.RECEIVE) on the mainframe using FTP. The sequence of these steps may vary depending on your site standards. You will be processing this transfer from a workstation/PC ftp client going to a host mainframe ftp server.

For example:

- Ensure the workstation/PC ftp client software is active.

- If you need to authenticate through a firewall to access the mainframe, then perform your site procedure for authentication with your credentials before attempting to connect to the mainframe.

- Go to the Windows Command Prompt where you will perform text based command line processing.

- Connect to the mainframe using ftp. You will need to enter the TCP address for your specific mainframe system. If you use IPv6 then specify the TCP address for the host server in the colon hexadecimal notation for IPv6. In the following example the IPv4 format for dotted decimal notation is used to specify the TCP address for the host mainframe.

  ```
  C:\Documents and Settings\Administrator>ftp 9.32.128.111
  ```

- Enter your user ID and password. This is your mainframe TSO user-id and password.

  ```
  User (9.32.128.111:(none)): user
  331 Send password please.
  Password:
  ```

- Ensure you have positioned yourself into the correct directory within the host mainframe file hierarchy. The working directory on the host mainframe is your mainframe TSO user-id. In this example the name “USER” is your mainframe TSO user id which automatically becomes the high-level qualifier of the receive data set on the mainframe.

  ```
  230 USER is logged on.  Working directory is "USER.".
  ```

- There are two types of a file in ftp which are binary and text. The .XMI file must be transferred in binary to prevent a corrupt file or lost characters when FILE757.XMI is placed on the mainframe in the receive data set (e.g., “USER.WORK.RECEIVE”). Ensure you set the file type to binary by entering the ‘bin’ command.

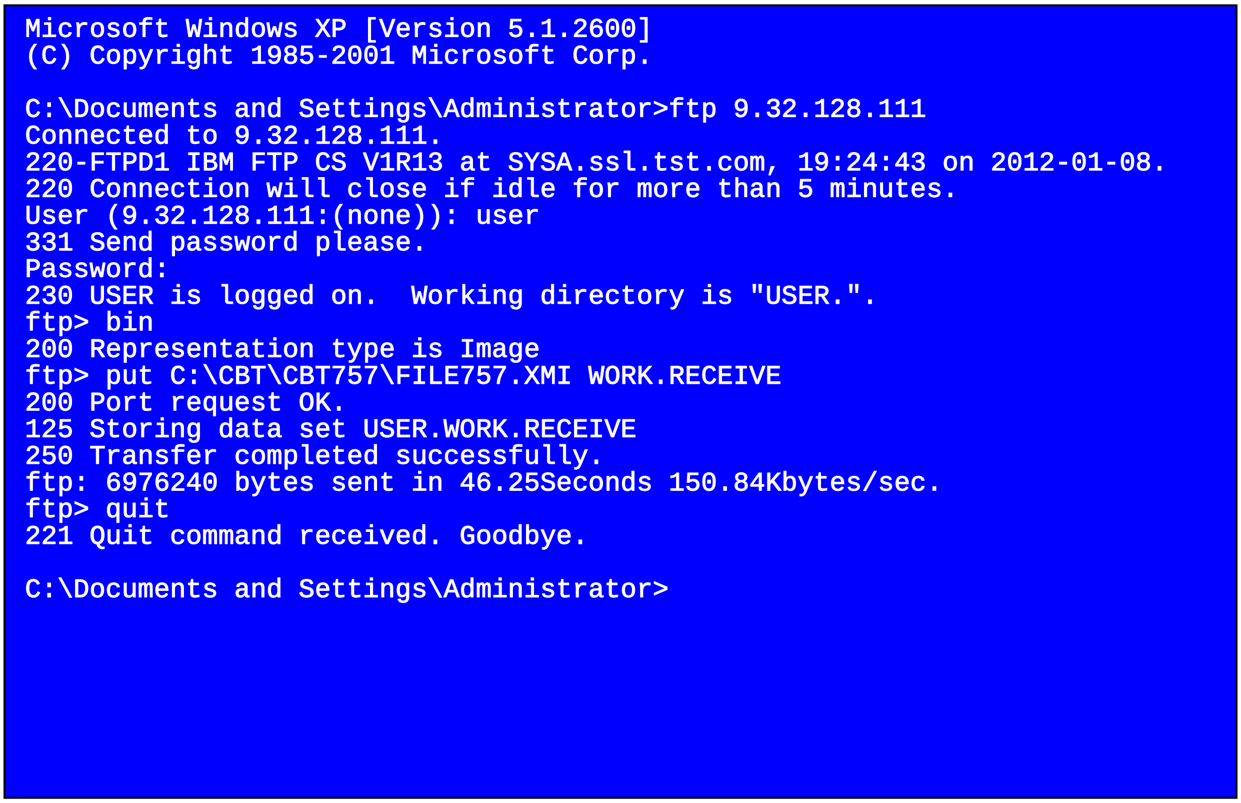
  ```
  ftp> bin
  200 Representation type is Image
  ```

- Determine the PC file to be transferred. This would be the CBT file named "FILE757.XMI". In this example the file on the PC resides in "C:\CBT\CBT757\FILE757.XMI". Determine the name of the mainframe data set to receive the transfer. In this example it is WORK.RECEIVE because the host mainframe working directory was already set by default to the TSO id "USER" which will target the fully qualified data set name of "USER.WORK.RECEIVE".

- Use PUT for the file in binary to indicate the direction of transfer is from your PC to the mainframe host.

  ```
  ftp> put C:\CBT\CBT757\FILE757.XMI  WORK.RECEIVE
  ```

For example:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ftp 9.32.128.111
Connected to 9.32.128.111.
220-FTPD1 IBM FTP CS V1R13 at SYSA.ssl.tst.com, 19:24:43 on 2012-01-08.
220 Connection will close if idle for more than 5 minutes.
User (9.32.128.111:(none)): user
331 Send password please.
Password:
230 USER is logged on.  Working directory is "USER.".
ftp> bin
200 Representation type is Image
ftp> put C:\CBT\CBT757\FILE757.XMI WORK.RECEIVE
200 Port request OK.
125 Storing data set USER.WORK.RECEIVE
250 Transfer completed successfully.
ftp: 6976240 bytes sent in 46.25Seconds 150.84Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.

C:\Documents and Settings\Administrator>
```

2.3 Step 3 – Receive Data Set on the Mainframe

Once the "FILE757.XMI" file has been transferred from the PC to the receive data set on the mainframe, then you will use the TSO LOGON command to perform the sign-on process to the mainframe if you are not already logged on. When you have completed logon to the mainframe the READY mode message is displayed indicating that the initial TSO logon processing has completed, and the system is ready to accept commands.

Once you are logged on to the z/OS system, at the TSO READY command line or ISPF option 6, type the following command:

```
receive indataset('filename')
```

Where filename is the mainframe data set name you used when transferring the FILE757.XMI file in the previous step 2.

For example:

```
receive indataset('user.work.receive')
```

You will receive a message similar to the following:

```
INMR901I Dataset CBT.V484.FILE757.PDS from SBGOLOB on N1
INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

You will type the following and press the Enter key:

```
dataset('pdsname')
```

Where ‘pdsname’ is the data set name of your choosing for the pre-allocated Partitioned Data Set (PDS) where you will receive the LOGGRASM components from ‘USER.WORK.RECEIVE’. In this example the receive is going to data set ‘USER.WORK.CNTL’.

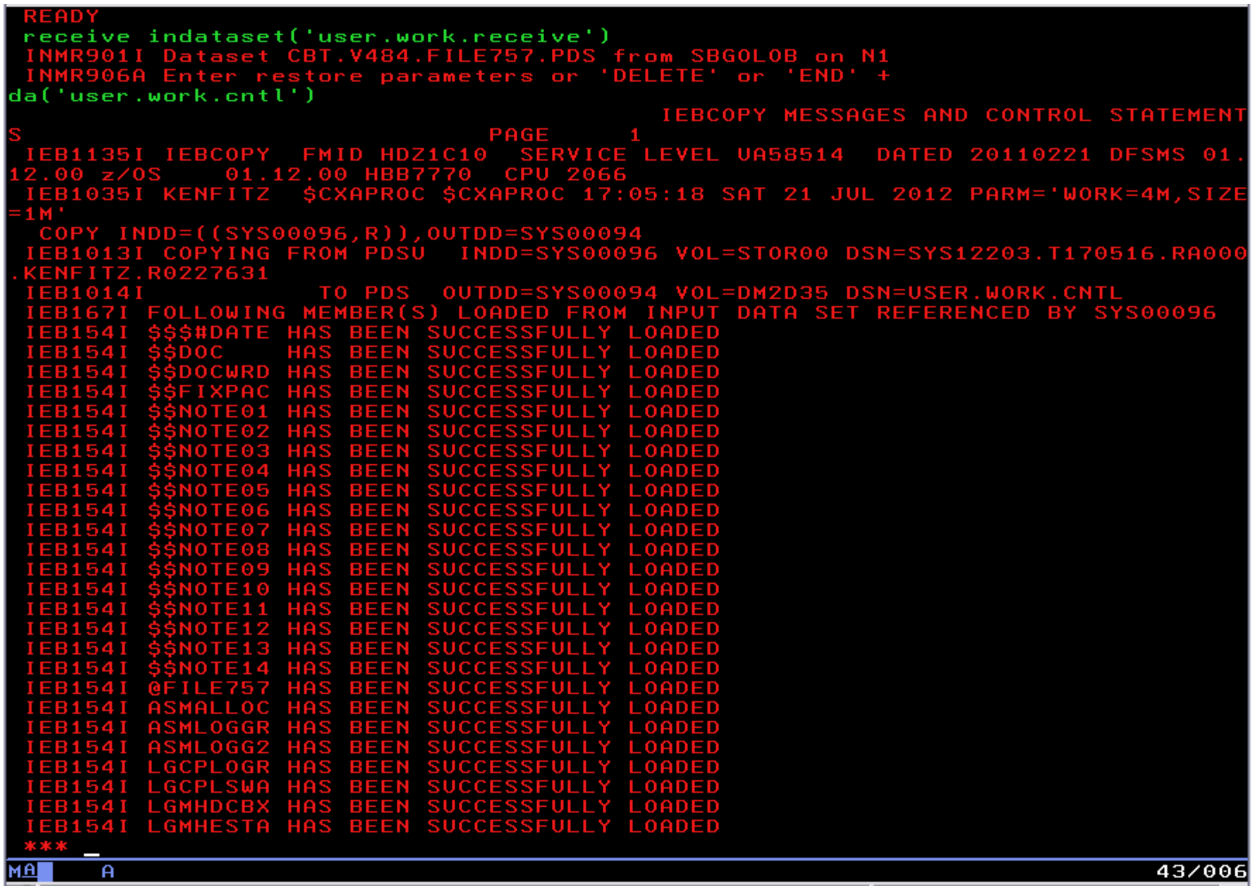
For example:

```
dataset('user.work.cntl')
```

or

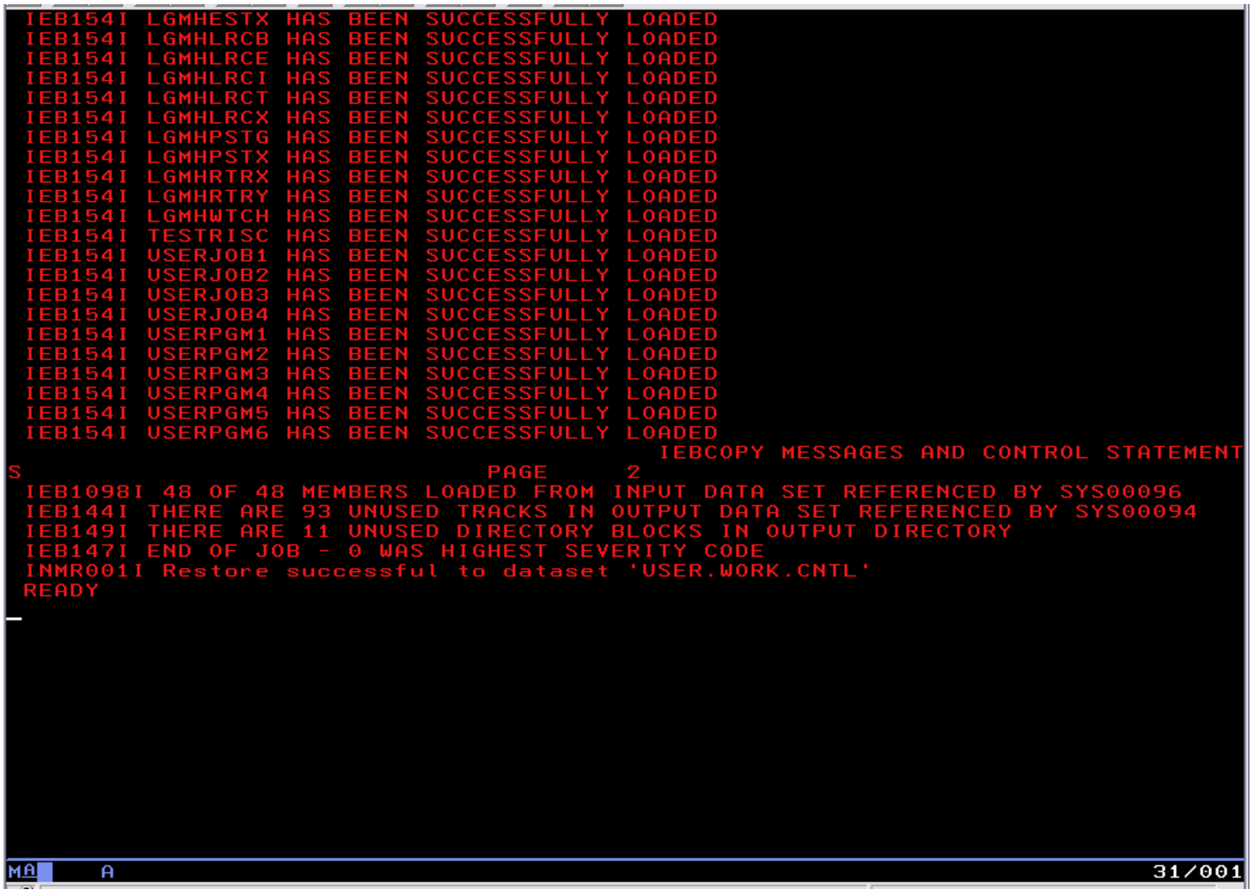
```
da('user.work.cntl')
```

The following is an example from the mainframe of the TSO READY screen where you are entering the RECEIVE command to place the contents of the receive data set ‘USER.WORK.RECEIVE’ from CBT Tape into your own PDS named ‘USER.WORK.CNTL’.



```
 READY
 receive indataset('user.work.receive')
 INMR901I Dataset CBT.V484.FILE757.PDS from SBGOLOB on N1
 INMR906A Enter restore parameters or 'DELETE' or 'END' +
da('user.work.cntl')
                                          IEBCOPY MESSAGES AND CONTROL STATEMENT
S                              PAGE      1
 IEB1135I IEBCOPY  FMID HDZ1C10  SERVICE LEVEL UA58514  DATED 20110221 DFSMS 01.
12.00 z/OS    01.12.00 HBB7770  CPU 2066
 IEB1035I KENFITZ  $CXAPROC $CXAPROC 17:05:18 SAT 21 JUL 2012 PARM='WORK=4M,SIZE
=1M'
  COPY INDD=((SYS00096,R)),OUTDD=SYS00094
 IEB1013I COPYING FROM PDSU  INDD=SYS00096 VOL=STOR00 DSN=SYS12203.T170516.RA000
.KENFITZ.R0227631
 IEB1014I           TO PDS  OUTDD=SYS00094 VOL=DM2D35 DSN=USER.WORK.CNTL
 IEB167I FOLLOWING MEMBER(S) LOADED FROM INPUT DATA SET REFERENCED BY SYS00096
 IEB154I $$$#DATE HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$DOC    HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$DOCWRD HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$FIXPAC HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE01 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE02 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE03 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE04 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE05 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE06 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE07 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE08 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE09 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE10 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE11 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE12 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE13 HAS BEEN SUCCESSFULLY LOADED
 IEB154I $$NOTE14 HAS BEEN SUCCESSFULLY LOADED
 IEB154I @FILE757 HAS BEEN SUCCESSFULLY LOADED
 IEB154I ASMALLOC HAS BEEN SUCCESSFULLY LOADED
 IEB154I ASMLOGGR HAS BEEN SUCCESSFULLY LOADED
 IEB154I ASMLOGG2 HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGCPLOGR HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGCPLSWA HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHDCBX HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHESTA HAS BEEN SUCCESSFULLY LOADED
 *** _
MA     A                                                                  43/006
```

The three asterisks (***) indicate the screen is full and you need to press the ENTER key go to another screen to continue viewing the pending IEB154I message output.



```
 IEB154I LGMHESTX HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHLRCB HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHLRCE HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHLRCI HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHLRCT HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHLRCX HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHPSTG HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHPSTX HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHRTRX HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHRTRY HAS BEEN SUCCESSFULLY LOADED
 IEB154I LGMHWTCH HAS BEEN SUCCESSFULLY LOADED
 IEB154I TESTRISC HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERJOB1 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERJOB2 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERJOB3 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERJOB4 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM1 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM2 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM3 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM4 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM5 HAS BEEN SUCCESSFULLY LOADED
 IEB154I USERPGM6 HAS BEEN SUCCESSFULLY LOADED
                                          IEBCOPY MESSAGES AND CONTROL STATEMENT
S                              PAGE     2
 IEB1098I 48 OF 48 MEMBERS LOADED FROM INPUT DATA SET REFERENCED BY SYS00096
 IEB144I THERE ARE 93 UNUSED TRACKS IN OUTPUT DATA SET REFERENCED BY SYS00094
 IEB149I THERE ARE 11 UNUSED DIRECTORY BLOCKS IN OUTPUT DIRECTORY
 IEB147I END OF JOB - 0 WAS HIGHEST SEVERITY CODE
 INMR001I Restore successful to dataset 'USER.WORK.CNTL'
 READY
_

MA     A                                                                  31/001
```

Once the target PDS “USER.WORK.CNTL” has been loaded, it will contain the documentation, copybooks, JCL, Logger Services assembler source code, and sample Assembler source programs.

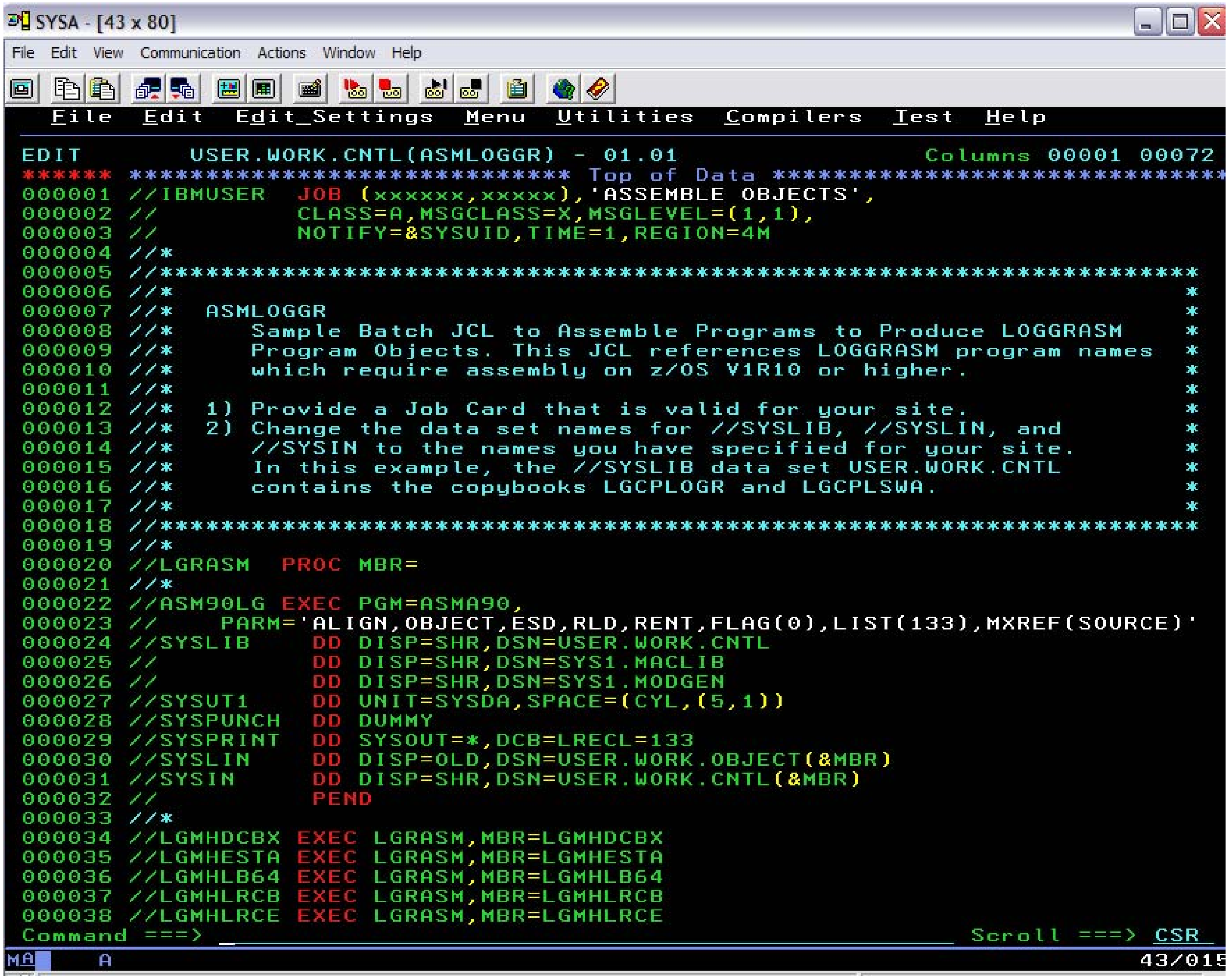
It is recommended that the ‘USER.WORK.CNTL’ (or your own named data set) library be designated as baseline repository library and should be considered sacrosanct, meaning its contents should not be modified except when copying in new components from the CBT-Tape.

This is entirely optional, but you could allocate a new data set and copy (i.e., checkout) the baseline library contents to your own personal staging library (i.e., USER.WORK.STAGING.FIXPAC17.CNTL) where you can make any changes you require to LOGGRASM without disturbing the original contents of ‘USER.WORK.CNTL’. Then you can migrate those changes from staging to other designated baseline libraries of your choosing. Just remember to review the latest '$$NOTEnn' member which summarizes the changes in the current fix pack.

There a member ASMALLOC which contains JCL you can use to allocate libraries using IEFBR14 in a batch job.

2.4 Step 4 – Run Job to Assemble Source and Create Objects

From ISPF navigate to the data set you used to restore the PDS from the receive data set. In this example the data set is “USER.WORK.CNTL”. In data set “USER.WORK.CNTL” edit the JCL in ASMLOGGR to conform to your site requirements. If your mainframe operating system is less than z/OS V1R10, then use JCL member ASMLOGG2.



```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       USER.WORK.CNTL(ASMLOGGR) - 01.01                Columns 00001 00072
 ****** ***************************** Top of Data ******************************
        //IBMUSER  JOB (xxxxxx,xxxxx),'ASSEMBLE OBJECTS',
        //             CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
        //             NOTIFY=&SYSUID,TIME=1,REGION=4M
        //*
        //*********************************************************************
        //*                                                                   *
        //*  ASMLOGGR                                                         *
        //*     Sample Batch JCL to Assemble Programs to Produce LOGGRASM     *
        //*     Program Objects. This JCL references LOGGRASM program names   *
        //*     which require assembly on z/OS V1R10 or higher.               *
        //*                                                                   *
        //*  1) Provide a Job Card that is valid for your site.               *
        //*  2) Change the data set names for //SYSLIB, //SYSLIN, and         *
        //*     //SYSIN to the names you have specified for your site.        *
        //*     In this example, the //SYSLIB data set USER.WORK.CNTL         *
        //*     contains the copybooks LGCPLOGR and LGCPLSWA.                 *
        //*                                                                   *
        //*********************************************************************
        //*
        //LGRASM   PROC MBR=
        //*
        //ASM90LG EXEC PGM=ASMA90,
        //     PARM='ALIGN,OBJECT,ESD,RLD,RENT,FLAG(0),LIST(133),MXREF(SOURCE)'
        //SYSLIB     DD DISP=SHR,DSN=USER.WORK.CNTL
        //           DD DISP=SHR,DSN=SYS1.MACLIB
        //           DD DISP=SHR,DSN=SYS1.MODGEN
        //SYSUT1     DD UNIT=SYSDA,SPACE=(CYL,(5,1))
        //SYSPUNCH   DD DUMMY
        //SYSPRINT   DD SYSOUT=*,DCB=LRECL=133
        //SYSLIN     DD DISP=OLD,DSN=USER.WORK.OBJECT(&MBR)
        //SYSIN      DD DISP=SHR,DSN=USER.WORK.CNTL(&MBR)
        //           PEND
        //*
        //LGMHDCBX EXEC LGRASM,MBR=LGMHDCBX
        //LGMHESTA EXEC LGRASM,MBR=LGMHESTA
        //LGMHLB64 EXEC LGRASM,MBR=LGMHLB64
        //LGMHLRCB EXEC LGRASM,MBR=LGMHLRCB
        //LGMHLRCE EXEC LGRASM,MBR=LGMHLRCE
 Command ===> _                                                Scroll ===> CSR
MA      A                                                                43/015
```

Update the JOB Card in ASMLOGGR to conform to your site requirements or as directed by your instructor. The JOB statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JOB), parameter, and comments. Your jobname should be unique. The jobname must begin in column 3. The jobname is 1 through 8 alphanumeric or national ($, #, @) characters. The first character in the jobname must be alphabetic or national ($, #, @). The jobname must be followed by at least one blank. The account field specifies an account number and other accounting-information which must be formatted as required by your site. Additional information on Job Control Language can be obtained from IBM publication ‘z/OS V1R13 MVS JCL Reference’ as Document Number SA22-7597-15.

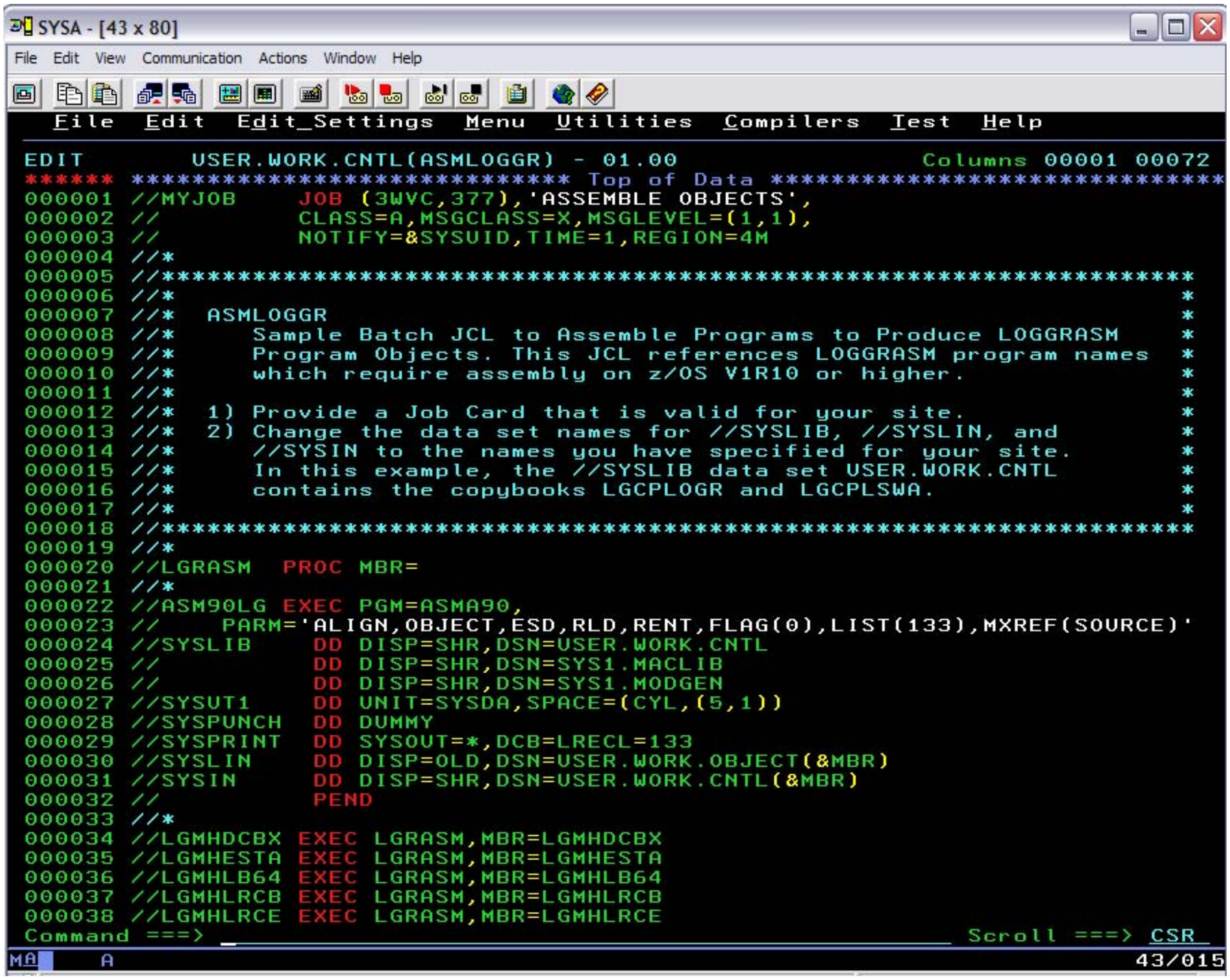
Ensure the proper data set allocations have been specified for the //SYSLIB DD. The data sets SYS1.MACLIB and SYS1.MODGEN are required for the z/OS system macro libraries, and USER.WORK.CNTL for the Logger Services components.

The //SYSLIN DD will be the output data set that will hold the objects produced from the assemble of the Logger Services source programs. Use HLASM V5 or higher for your Assembler (ASMA90).

The input data set is //SYSIN, and contains the Logger services Assembler source programs.

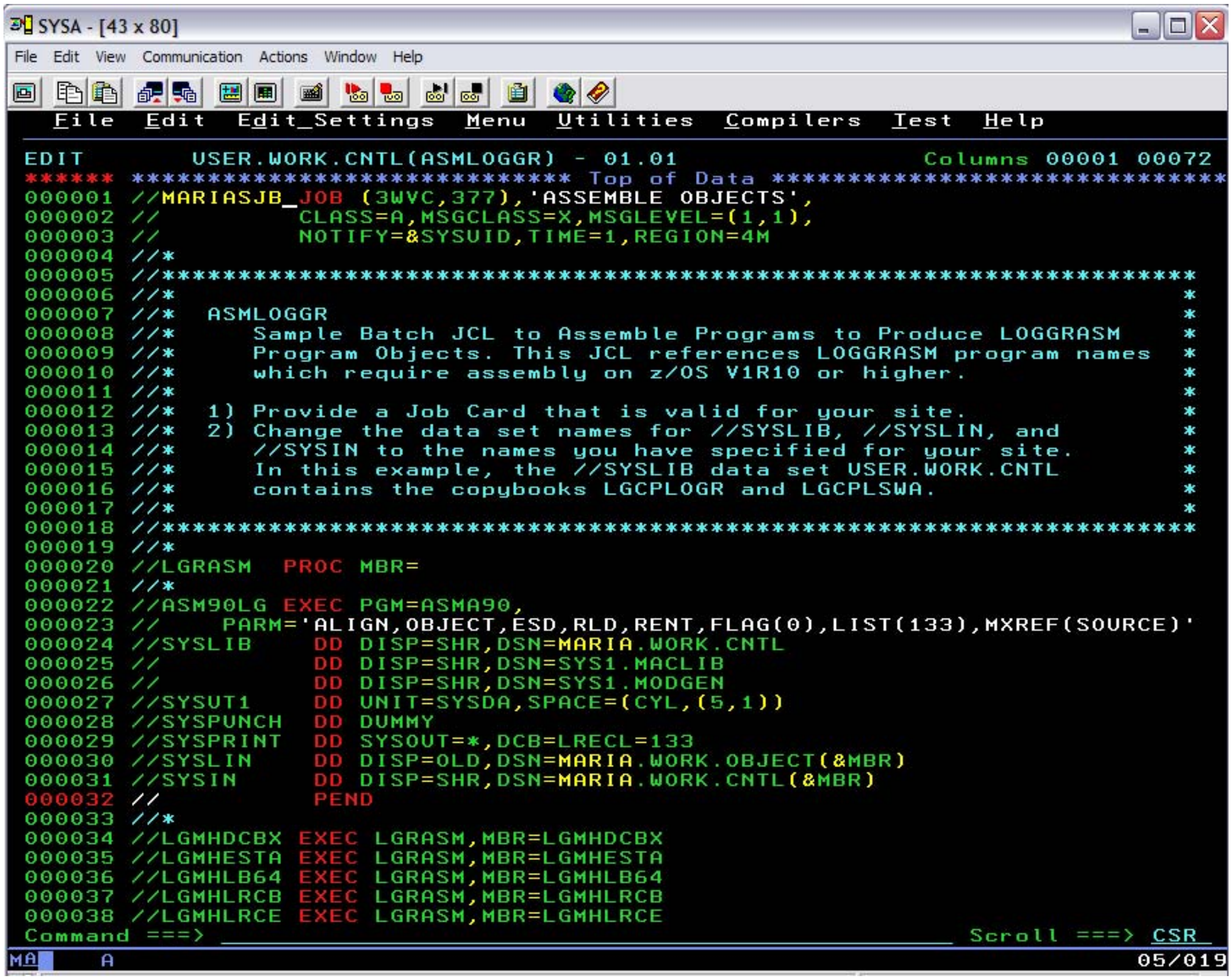
Once all changes are complete, then submit the ASMLOGGR JCL

For example:


SYSA - [43 x 80]
File Edit View Communication Actions Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help
MA A 43/015

```
EDIT       USER.WORK.CNTL(ASMLOGGR) - 01.00               Columns 00001 00072
****** ***************************** Top of Data ******************************
//MYJOB     JOB (3WVC,377),'ASSEMBLE OBJECTS',
//          CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID,TIME=1,REGION=4M
//*
//********************************************************************
//*                                                                  *
//*  ASMLOGGR                                                        *
//*     Sample Batch JCL to Assemble Programs to Produce LOGGRASM    *
//*     Program Objects. This JCL references LOGGRASM program names  *
//*     which require assembly on z/OS V1R10 or higher.              *
//*                                                                  *
//*  1) Provide a Job Card that is valid for your site.              *
//*  2) Change the data set names for //SYSLIB, //SYSLIN, and        *
//*     //SYSIN to the names you have specified for your site.       *
//*     In this example, the //SYSLIB data set USER.WORK.CNTL        *
//*     contains the copybooks LGCPLOGR and LGCPLSWA.                *
//*                                                                  *
//********************************************************************
//*
//LGRASM  PROC MBR=
//*
//ASM90LG EXEC PGM=ASMA90,
//    PARM='ALIGN,OBJECT,ESD,RLD,RENT,FLAG(0),LIST(133),MXREF(SOURCE)'
//SYSLIB     DD DISP=SHR,DSN=USER.WORK.CNTL
//           DD DISP=SHR,DSN=SYS1.MACLIB
//           DD DISP=SHR,DSN=SYS1.MODGEN
//SYSUT1     DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSPUNCH   DD DUMMY
//SYSPRINT   DD SYSOUT=*,DCB=LRECL=133
//SYSLIN     DD DISP=OLD,DSN=USER.WORK.OBJECT(&MBR)
//SYSIN      DD DISP=SHR,DSN=USER.WORK.CNTL(&MBR)
//           PEND
//*
//LGMHDCBX EXEC LGRASM,MBR=LGMHDCBX
//LGMHESTA EXEC LGRASM,MBR=LGMHESTA
//LGMHLB64 EXEC LGRASM,MBR=LGMHLB64
//LGMHLRCB EXEC LGRASM,MBR=LGMHLRCB
//LGMHLRCE EXEC LGRASM,MBR=LGMHLRCE
Command ===>                                                  Scroll ===> CSR
```

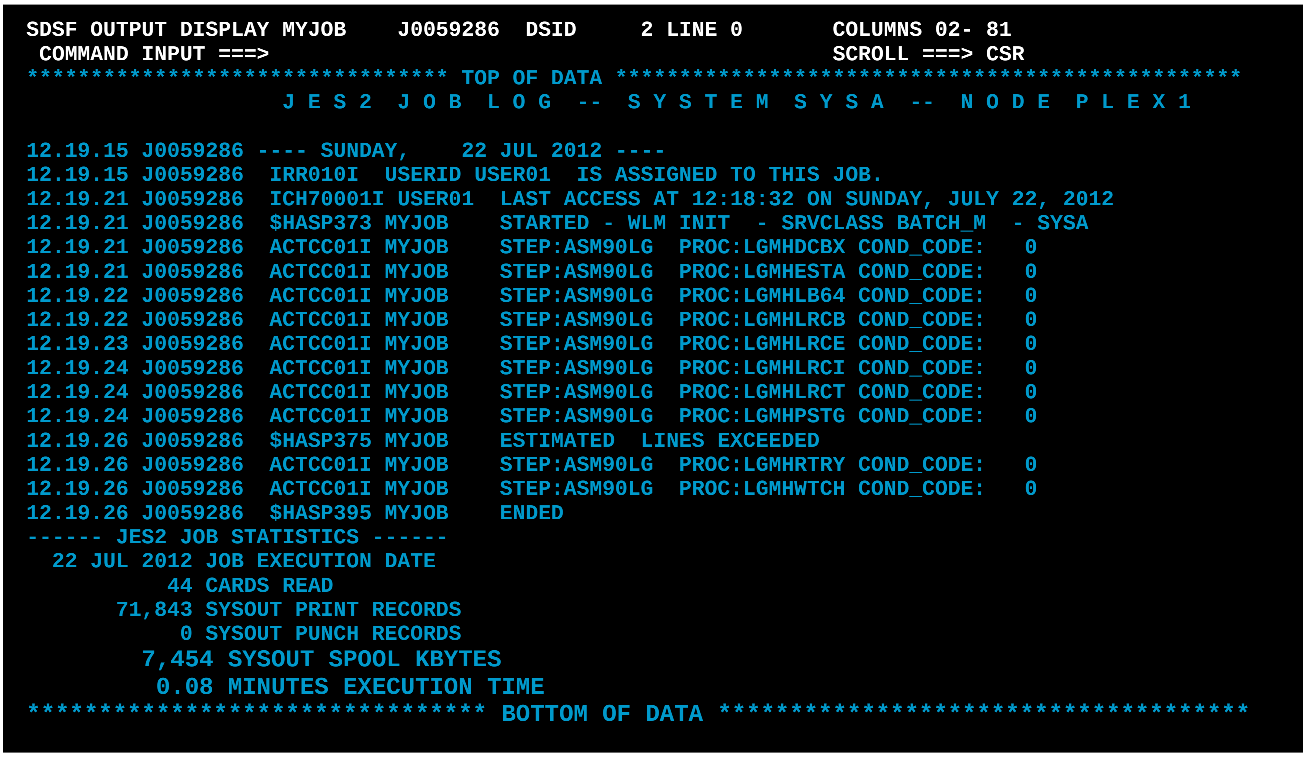
For example:


SYSA - [43 x 80]
File Edit View Communication Actions Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help

```
EDIT       USER.WORK.CNTL(ASMLOGGR) - 01.01            Columns 00001 00072
****** ***************************** Top of Data ******************************
//MARIASJB JOB (3WVC,377),'ASSEMBLE OBJECTS',
//           CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),
//           NOTIFY=&SYSUID,TIME=1,REGION=4M
//*
//*********************************************************************
//*                                                                   *
//*  ASMLOGGR                                                         *
//*     Sample Batch JCL to Assemble Programs to Produce LOGGRASM     *
//*     Program Objects. This JCL references LOGGRASM program names   *
//*     which require assembly on z/OS V1R10 or higher.               *
//*                                                                   *
//*  1) Provide a Job Card that is valid for your site.               *
//*  2) Change the data set names for //SYSLIB, //SYSLIN, and         *
//*     //SYSIN to the names you have specified for your site.        *
//*     In this example, the //SYSLIB data set USER.WORK.CNTL         *
//*     contains the copybooks LGCPLOGR and LGCPLSWA.                 *
//*                                                                   *
//*********************************************************************
//*
//LGRASM  PROC MBR=
//*
//ASM90LG EXEC PGM=ASMA90,
//     PARM='ALIGN,OBJECT,ESD,RLD,RENT,FLAG(0),LIST(133),MXREF(SOURCE)'
//SYSLIB     DD DISP=SHR,DSN=MARIA.WORK.CNTL
//           DD DISP=SHR,DSN=SYS1.MACLIB
//           DD DISP=SHR,DSN=SYS1.MODGEN
//SYSUT1     DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSPUNCH   DD DUMMY
//SYSPRINT   DD SYSOUT=*,DCB=LRECL=133
//SYSLIN     DD DISP=OLD,DSN=MARIA.WORK.OBJECT(&MBR)
//SYSIN      DD DISP=SHR,DSN=MARIA.WORK.CNTL(&MBR)
//           PEND
//*
//LGMHDCBX EXEC LGRASM,MBR=LGMHDCBX
//LGMHESTA EXEC LGRASM,MBR=LGMHESTA
//LGMHLB64 EXEC LGRASM,MBR=LGMHLB64
//LGMHLRCB EXEC LGRASM,MBR=LGMHLRCB
//LGMHLRCE EXEC LGRASM,MBR=LGMHLRCE
Command ===> ________________________________________________ Scroll ===> CSR
MA     A                                                                05/019
```

Verify the job ran to successful completion. The ASMLOGGR job should execute with all steps having a return code of RC=00. If not all steps completed with RC=00 or a JCL error or other error condition, check the JESMSGLG DD for the job, and check the //SYSPRINT DD for any errors messages during the assembly. Make the necessary corrections to clear the error, and resubmit the job.

For example:



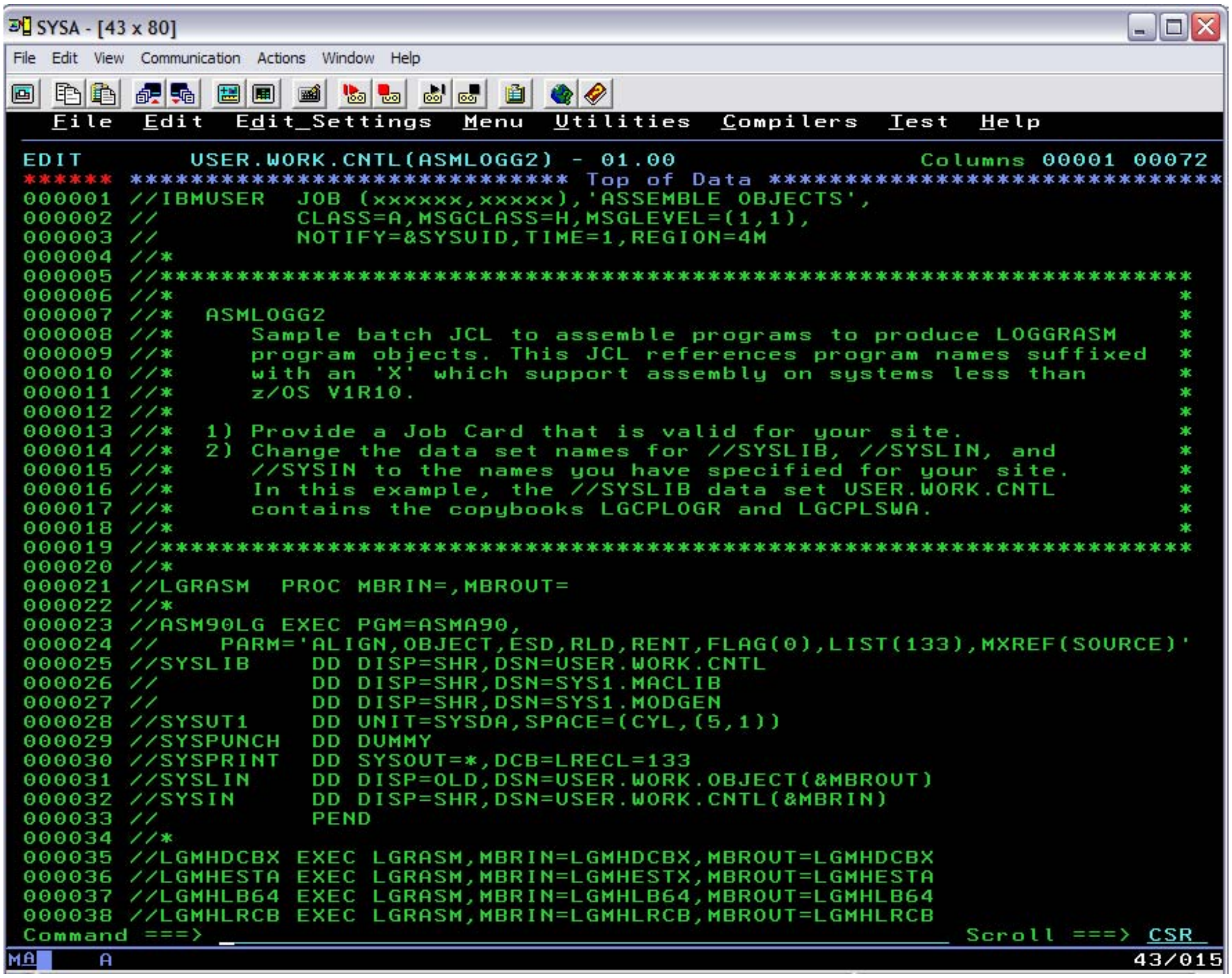
```
SDSF OUTPUT DISPLAY MYJOB    J0059286  DSID     2 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> CSR
********************************* TOP OF DATA ************************************************
                    J E S 2   J O B   L O G  --  S Y S T E M   S Y S A  --  N O D E   P L E X 1

12.19.15 J0059286 ---- SUNDAY,    22 JUL 2012 ----
12.19.15 J0059286  IRR010I  USERID USER01  IS ASSIGNED TO THIS JOB.
12.19.21 J0059286  ICH70001I USER01  LAST ACCESS AT 12:18:32 ON SUNDAY, JULY 22, 2012
12.19.21 J0059286  $HASP373 MYJOB    STARTED - WLM INIT  - SRVCLASS BATCH_M  - SYSA
12.19.21 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHDCBX COND_CODE:   0
12.19.21 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHESTA COND_CODE:   0
12.19.22 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHLB64 COND_CODE:   0
12.19.22 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHLRCB COND_CODE:   0
12.19.23 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHLRCE COND_CODE:   0
12.19.24 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHLRCI COND_CODE:   0
12.19.24 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHLRCT COND_CODE:   0
12.19.24 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHPSTG COND_CODE:   0
12.19.26 J0059286  $HASP375 MYJOB    ESTIMATED  LINES EXCEEDED
12.19.26 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHRTRY COND_CODE:   0
12.19.26 J0059286  ACTCC01I MYJOB    STEP:ASM90LG  PROC:LGMHWTCH COND_CODE:   0
12.19.26 J0059286  $HASP395 MYJOB    ENDED
------ JES2 JOB STATISTICS ------
  22 JUL 2012 JOB EXECUTION DATE
           44 CARDS READ
       71,843 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
        7,454 SYSOUT SPOOL KBYTES
         0.08 MINUTES EXECUTION TIME
*********************************** BOTTOM OF DATA *****************************************
```

Once the target PDS object library //SYSLIN (e.g., USER.WORK.OBJECT) has been loaded from the ASMLOGGR job, the PDS will contain the Logger Services objects. This PDS will need to be included in the //SYSLIB DD concatenation of the link-edit step for any Assembler program you code utilizing Logger Services. By default the binder will automatically search the call library defined on the //SYSLIB DD statement to resolve external references to Logger Services programs.

To assemble with ASMLOGGR, you will need to be at least z/OS V1R10 or above.

If you are less than z/OS V1R10, you can assemble on at least a V1R10 system, and ship the objects to a lower level z/OS.

Otherwise, to assemble at less than V1R10, you can just run the job ASMLOGG2 instead of ASMLOGGR.


SYSA - [43 x 80]
File Edit View Communication Actions Window Help
File Edit Edit_Settings Menu Utilities Compilers Test Help

```
EDIT       USER.WORK.CNTL(ASMLOGG2) - 01.00               Columns 00001 00072
****** ***************************** Top of Data ******************************
//IBMUSER  JOB (xxxxxx,xxxxx),'ASSEMBLE OBJECTS',
//         CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,TIME=1,REGION=4M
//*
//*********************************************************************
//*                                                                   *
//*  ASMLOGG2                                                         *
//*     Sample batch JCL to assemble programs to produce LOGGRASM    *
//*     program objects. This JCL references program names suffixed  *
//*     with an 'X' which support assembly on systems less than      *
//*     z/OS V1R10.                                                   *
//*                                                                   *
//*  1) Provide a Job Card that is valid for your site.               *
//*  2) Change the data set names for //SYSLIB, //SYSLIN, and         *
//*     //SYSIN to the names you have specified for your site.        *
//*     In this example, the //SYSLIB data set USER.WORK.CNTL         *
//*     contains the copybooks LGCPLOGR and LGCPLSWA.                 *
//*                                                                   *
//*********************************************************************
//*
//LGRASM  PROC MBRIN=,MBROUT=
//*
//ASM90LG EXEC PGM=ASMA90,
//    PARM='ALIGN,OBJECT,ESD,RLD,RENT,FLAG(0),LIST(133),MXREF(SOURCE)'
//SYSLIB    DD DISP=SHR,DSN=USER.WORK.CNTL
//          DD DISP=SHR,DSN=SYS1.MACLIB
//          DD DISP=SHR,DSN=SYS1.MODGEN
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSPUNCH  DD DUMMY
//SYSPRINT  DD SYSOUT=*,DCB=LRECL=133
//SYSLIN    DD DISP=OLD,DSN=USER.WORK.OBJECT(&MBROUT)
//SYSIN     DD DISP=SHR,DSN=USER.WORK.CNTL(&MBRIN)
//          PEND
//*
//LGMHDCBX EXEC LGRASM,MBRIN=LGMHDCBX,MBROUT=LGMHDCBX
//LGMHESTA EXEC LGRASM,MBRIN=LGMHESTX,MBROUT=LGMHESTA
//LGMHLB64 EXEC LGRASM,MBRIN=LGMHLB64,MBROUT=LGMHLB64
//LGMHLRCB EXEC LGRASM,MBRIN=LGMHLRCB,MBROUT=LGMHLRCB
Command ===> _                                                 Scroll ===> CSR
MA     A                                                                43/015
```

ASMLOGG2 will assemble different versions of the following programs. These re-named programs suffixed with an X are the same as the standard Logger Services programs except they contain hard coding of the level dependent macro mappings to prevent assembly errors on lower level z/OS systems. These program will not use the new features in the higher level z/OS systems. The output will be member names LGMHESTA, LGMHLRCI, LGMHRTRY, and LGMHPSTG.

a) For LGMHESTA the ASMLOGG2 JCL uses LGMHESTX as its input.
b) For LGMHLRCI the ASMLOGG2 JCL uses LGMHLRCX as its input.
c) For LGMHRTRY the ASMLOGG2 JCL uses LGMHRTRX as its input.
d) For LGMHPSTG the ASMLOGG2 JCL uses LGMHPSTX as its input.

In addition, the LOGGRASM Assembler source code is fully available for you to make any additional changes in the event of any site specific conditions, or to make your own customizations to the LOGGRASM assembler source code in any manner you wish.

With the ASMLOGG2 JCL you should be able to assemble on a lower level z/OS system (at least down to V1R7) to produce compatible objects. This is to accommodate sites that may run an 'EOS' (End of Service) release of z/OS.

z/OS	Announced	General	Withdrawn from	End of
Version		Availability	Marketing	Service
V1R2	2012/04/11	2013/ (1)		
V1R13	2011/07/12	2011/09/30		(2)
V1R12	2010/07/22	2010/09/24	2011/09/25	(3)
V1R11	2009/08/18	2009/09/25	2010/10/26	2012/09/30
V1R10	2008/08/05	2008/09/26	2009/10/26	2011/09/30
V1R09	2007/08/08	2007/09/28	2008/10/27	2010/09/30
V1R08	2006/08/08	2006/09/29	2007/10/22	2009/09/30
V1R07	2005/07/27	2005/09/30	2006/10/23	2008/09/30

(1)There will be a new version of the z/OS operating system where the next release will be known as z/OS Version 2. This new z/OS V2R1 operating system has a planned release some time in the second half of 2013 as part of a new two-year release cycle where previously there was a one-year release cycle .

(2)Also with the introduction of z/OS V2R1, the support for z/OS V1R13 is planned to be expanded from three years to five years. This indicates for Version V1R13 support is likely to be extended from September 30, 2014, to September 30, 2016.

(3)With the introduction of z/OS V1R2, the support for z/OS V1R12 is planned to be expanded from three years to four years. This indicates support for Version V1R12 is likely to be extended from September 30, 2013, to September 30, 2014.

2.5 Step 5 – Run Sample Programs

After the installation is complete, you can verify the Logger Service is available and is working correctly by running the three sample jobs supplied.

Go to data set ‘USER.WORK.CNTL’ or to your own named data set you used for the install. You will need to edit the sample JCL for member USERJOB1 to conform to your site requirements.

Update the Job Card in USERJOB1 to conform to your site requirements. The JOB statement consists of the characters // in columns 1 and 2 and four fields: name, operation (JOB), parameter, and comments. Your jobname should be unique. The jobname must begin in column 3. The jobname is 1 through 8 alphanumeric or national ($, #, @) characters. The first character in the jobname must be alphabetic or national ($, #, @). The jobname must be followed by at least one blank. The account field specifies an account number and other accounting-information formatted as required by your site. More information on JCL can be obtained from IBM publication ‘z/OS V1R13 MVS JCL Reference’ as Document Number SA22-7597-15.

Once the JOB card is complete, ensure the proper data set allocations have been specified for the //SYSLIB (e.g., SYS1.MACLIB, SYS1.MODGEN, USER.WORK.CNTL) in the ASMH step. Check the data set name for the load library (e.g., //SYSLMOD) in the LKED step. Also check the data set name for the //SYSLIB DD in the LKED step as this data set will contain the Logger Services objects created earlier from the ASMLOGGR job in Step 4.

Submit the USERJOB1 JCL and verify successful completion. After the job has run you can view the records created in the //LGRECOUT DD from Job Step RUN executing PGM=USERPGM1. When examining the output records created in the //LGRECOUT DD, at the same time review the source code from the sample program USERPGM1 to see the relationship between the LOGGRASM commands used in the sample program and the resulting display of information in the //LGRECOUT DD records. This review lets you see visually how things work.

For example:

```
//IBMUSER  JOB (xxxxxx,xxxxx),'ASSEM/LINK-EDIT/RUN',
//         CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
//         NOTIFY=&SYSUID,TIME=1,REGION=4M
//*
//********************************************************************
//*                                                                  *
//*  USERJOB1                                                        *
//*     Sample Batch JCL to Assemble, Link-Edit, and Run Sample      *
//*     Program 1 with LOGGRASM.                                     *
//*                                                                  *
//*     The program USERPGM1 in the RUN step opens a data set, prints*
//*     a test message, and closes the data. This JCL will run       *
//*     program USERPGM1 to show an example of log point information *
//*     written to the //LGRECOUT DD.                                *
//*                                                                  *
//*  1) Provide a Job Card that is valid for your site.              *
//*  2) Change the data set names for //SYSLIB and SYSIN in step     *
//*     ASMH to the names you have specified for your site. In this  *
//*     example, the data set ****.WORK.CNTL contains the copybook   *
//*     LGCPLOGR.                                                    *
//*  3) Change the data set names for //SYSLIB and //SYSMOD          *
//*     in step LKED to the names you have specified for your        *
//*     site. In this example data set ****.WORK.OBJECT contains     *
//*     the LOGGRASM objects created by the previous ASMLOGGR job.   *
//*  4) Change the data set name for //STEPLIB in step RUN to        *
//*     the name you have specified for //SYSLMOD in step LKED.      *
//*  5) Submit job, and review //LGRECOUT for output from log        *
//*     points.                                                      *
//*                                                                  *
//********************************************************************
//*
//********************************************************************
//*   ASSEMBLE SAMPLE PROGRAM 1                                      *
//********************************************************************
//*
//ASMH    EXEC PGM=ASMA90,
//     PARM='ALIGN,OBJECT,RENT,ESD,RLD,FLAG(0),LIST(133)'
//SYSLIB    DD DISP=SHR,DSN=USER.WORK.CNTL    (Contains CopyBook LGCPLOGR)
//          DD DISP=SHR,DSN=SYS1.MACLIB
//          DD DISP=SHR,DSN=SYS1.MODGEN
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(1,5))
//SYSPUNCH  DD DUMMY
//SYSPRINT  DD SYSOUT=*,DCB=LRECL=133
//SYSLIN    DD DISP=(,PASS),DSN=&&OBJECT(USERPGM1),
//             UNIT=SYSDA,SPACE=(TRK,(15,15,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN     DD DISP=SHR,
//             DSN=USER.WORK.CNTL(USERPGM1)
//*
//********************************************************************
//*   LINK-EDIT SAMPLE PROGRAM 1                                     *
//********************************************************************
//*
//LKED      EXEC PGM=IEWL,
//             PARM='LIST,XREF,RENT,LET,COMPAT=PM4'
//SYSPRINT  DD SYSOUT=*,HOLD=YES
//SYSTERM   DD SYSOUT=*,HOLD=YES
//SYSLIB    DD DISP=SHR,DSN=USER.WORK.OBJECT      ←───── (Logger Objects)
//SYSLMOD   DD DISP=SHR,DSN=USER.WORK.LINKLIB(USERPGM1)←─────(Load Module)
//SYSLIN    DD DISP=(OLD,DELETE),DSN=&&OBJECT(USERPGM1) ←────(Your Object)
//          DD *
      MODE  AMODE(24),RMODE(24)
      ENTRY USERPGM1
      NAME  USERPGM1(R)
//*
//********************************************************************
//*   RUN SAMPLE PROGRAM 1                                           *
//********************************************************************
//*
//RUN       EXEC PGM=USERPGM1
//STEPLIB   DD DISP=SHR,DSN=USER.WORK.LINKLIB
//SYSPRINT  DD SYSOUT=*
//SYSABEND  DD SYSOUT=*
//LGRECOUT  DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=133)  ←─────(Output Log Records)
//LGRSYSIN  DD *
    LOGEVENT PROGRM,NAME=(*)
    LOGEVENT SUBRTN,NAME=(*)   ─────→ (Input Control Cards)
    LOGEVENT LOGPNT,NAME=(*)
//*
```

Next edit the sample JCL for USERJOB2 to conform to your site requirements. Perform the same activity for this job as the previous USERJOB1 to prepare for job submission. Submit the USERJOB2 JCL This job will intentionally abend with a S0C7 so you can view the Logpoint records and the detailed diagnostic abend report written to the //LGRECOUT DD. Also, an example of an abend report is illustrated in ‘Section 8.3 Sample Abend Report’.

When examining the output records created in //LGRECOUT, at the same time review the source code from the sample program USERPGM2 to see the relationship between the use of LOGGRASM commands in the USERPGM2 program, and the display of information in the //LGRECOUT records that were generated by LOGGRASM commands. This review is a good way to get up to speed quickly on how things work. The USERPGM2 Assembler source is a little larger sample program showing more variations on the use of LOGGRASM commands, and how they can be used to display program values, program entry, program exit, program pathing, register contents, storage areas, and other factors helpful to program debugging. Again, the USERPGM2 program is a sample program designed to intentionally abend to show the LOGGRASM diagnostic abend report

Next edit the sample JCL for USERJOB3 to conform to your site requirements. Perform the same activity for this job as the previous USERJOB2 to prepare for job submission. Submit the USERJOB3 JCL and verify successful completion. After the job has run you can view the records created in the //LGRECOUT DD from Job Step RUN executing PGM=USERPGM3.

USERJOB4 shows an example of how LOGGRASM will support the breaking out of the three RSECTS defined in the one sample program USERPGM2, and placing them in different PDS members as separate programs USERPGM4, USERPGM5, and USERPGM6. Some simple design requirements could be specified by the instructor for each of the three programs based on parameter passing or anything else. In this example three groups of students in a class could work on their separate piece of Assembler code, and then have to learn about team work to bring them together into one workable load module at link-edit, and run it on the mainframe. Edit the sample JCL for USERJOB4 to conform to your site requirements and submit to verify successful completion.

The previous description is for sample programs so you may become familiar with the basics of using LOGGRASM. In addition, the assembler source programs LGMHLRCI, LGMHLRCT, and LGMHRTRY can be reviewed as further examples showing the use of LOGGRASM commands.

The next step is for you to create your own Assembler programs to perform any work of your choosing using LOGGRASM. LOGGRASM is basically designed to support the development of new programs where you are starting from scratch, such as when a student begins an Assembler course in a college or university environment, or as a new programmer interested in learning mainframe Assembler.

Refer to Chapter 3 and Chapter 4 for a description of the Logger Service commands.

3.0 Summary Description of LOGGRASM Commands

3.1 Summary of Commands

Label LPGMNTRY [parameters]

TYPE={MAIN | SUB}
AMOD={24 | 31 | 64}
RMOD={24 | ANY | 31}
BASE=(12[,n]...)
LCAXTRA=(Rn[,Rn]...)
STORAGE=*number*
LCAPFX=*label*
LCADSECT={YES | NO}
LINKRTRN={BR | BSM}
LOG={ON | OFF}
LOGOUT={PRNT | BUFR | BUF64}
LOGTRIM={YES | NO}
COPY@RT={YES | NO}

Label LPGMEXIT [parameters]

RC=[addr]
[(Rn)]
RS=[addr]
[(Rn)]

Label LPGMSUBE [parameters]

RETRN={Rn}
MSG=(*'message text'[,variable*])
PVTAREA={YES | NO}
SHOW=((*'text',address,length*)[,(*'text',address,length*)]...)
WATCH=((*'text',address,length*)[,(*'text',address,length*)]...)
REG={NONE | ALL | (Rn[,Rn]...)}

Label LPGMSUBX [parameters]

> **MSG=('*message text'[,variable*])**
> **PVTAREA={YES | NO}**
> **SHOW=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)**
> **WATCH=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)**
> **REG={NONE | ALL | (Rn[,Rn]...)}**

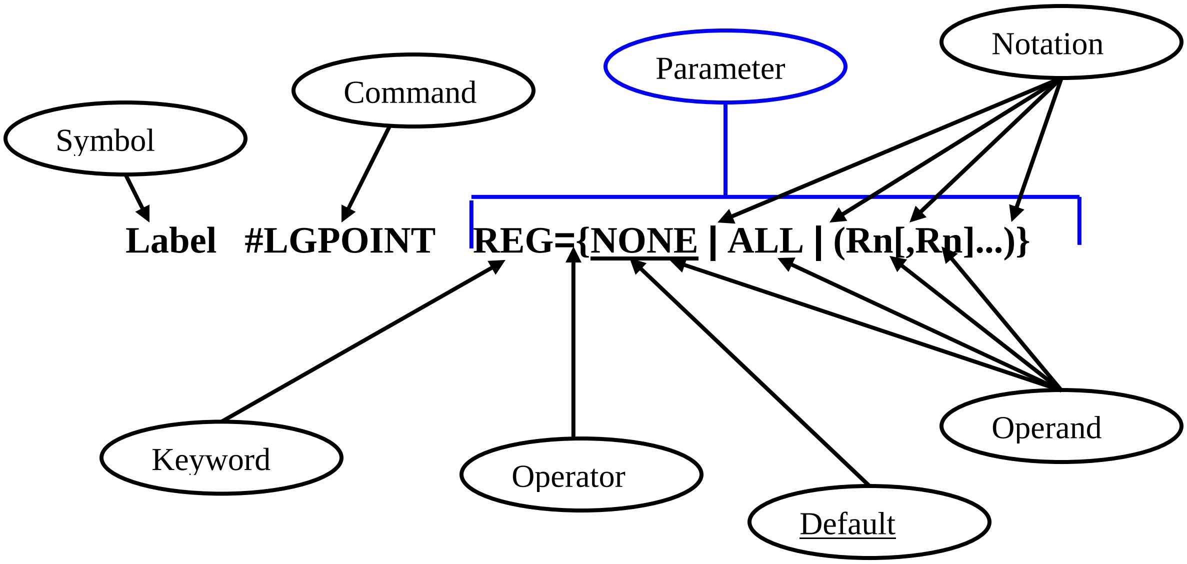
Label LCA {DEFINE | DEFEND}

Label #LGPOINT [parameters]

> **MSG=('*message text'[,variable*])**
> **PVTAREA={YES | NO}**
> **SHOW=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)**
> **WATCH=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)**
> **REG={NONE | ALL | (Rn[,Rn]...)}**
> **COMPRESS={YES | NO}**
> **SUPHEAD={YES | NO}**

3.2 Notations and Syntax

The following is an example of the command structure:



The following are the notational conventions used:

[ ] - Brackets enclose an optional entry. You may code only one or none of the enclosed entries unless followed by an ellipsis which indicates repeating the entry is allowed. This means you are allowed to include the entry, but you are not required to do so. For example:

BASE=(12[,n]...)

BASE=(12,11)

| - An OR sign (a vertical bar) is used to separate alternative entries. It is used to indicate choices between braces and brackets. You must specify one, and only one, of the entries unless there is an indicated default. For example:

AMOD={24 | <u>31</u> | 64}

AMOD=64

{ } - Braces are used to enclose alternative entries. Braces surround required, related entries and indicate that you must code one of the enclosed entries. This means you must use one, and only one, of the entries. For example:

TYPE={MAIN | <u>SUB</u>}

TYPE=MAIN

... - An ellipsis indicates that the entry immediately preceding the ellipsis can be repeated. For example:

REG={<u>NONE</u> | ALL | (Rx[,Rx]...)}

REG=(R3,R4,R5,R8)

UPPERCASE BOLDFACE - Uppercase boldface type indicates entries that you must code exactly as shown. These entries consist of the parameter and any associated punctuation such as commas, parentheses, and equal signs. For example:

LOGOUT={<u>PRNT</u> | BUFR}

LOGOUT=BUFR

UNDERSCORED UPPERCASE BOLDFACE - Underscored uppercase boldface type indicates the default which will be used if you do not specify the parameter. For example:

AMOD={24 | 31 | 64}
RMOD={24 | ANY | 31}

AMOD= and RMOD= not specified so program defaults to AMODE 31 and RMODE 31.

Note: Comma delimited substitution is not supported. Either code the parameter explicitly or do not include the parameter if it is not a required parameter. What this means, for example, is if you want AMODE 31 and desire to take the default, then do not include the parameter. If you do include the parameter, then code it as AMOD=31 to restate the default or AMOD=64 to override the default. Do not code the parameter with a comma immediately after the equal sign operator as in "**AMOD=,**".

AMOD=31,RMOD=31,	Yes
AMOD=,RMOD=31,	No

lowercase bold italic - Lowercase bold italic type indicates a value that you supply, according to the specifications and limits for the Logger parameter. For example:

SHOW=((*'text',address,length*)[,(*'text',address,length*)]...)

SHOW=('My table area #2',(R4),32760)

4.0 Detail Description of LOGGRASM Commands

LOGGRASM commands are defined in the LGCPLOGR copybook. The LGCPLOGR copybook must be included at the beginning of your Assembler program before the first START or RSECT/CSECT Assembler directive. Use the Assembler COPY directive with a source name of 'LGCPLOGR' to include the copybook in your Assembler program.

The minimum setup for an Assembler source program requires the COPY LGCPLOGR, the LPGMNTRY entry, the LPGMEXIT entry, and the LCA DEFINE and LCA DEFEND entries.

Sample programs USERPGM1 through USERPGM6 show examples of using LOGGRASM commands in Assembler source code. In addition, Assembler source programs LGMHLRCI, LGMHLRCT, and LGMHRTRY can be reviewed as further examples showing the use of LOGGRASM commands.

4.1 LPGMNTRY Description

Label LPGMNTRY [parameters]
TYPE={MAIN | SUB}
AMOD={24 | 31 | 64}
RMOD={24 | ANY | 31}
BASE=(12[,n]...)
LCAXTRA=(Rn[,Rn]...)
STORAGE=*number*
LCAPFX=*label*
LCADSECT={YES | NO}
LINKRTRN={BR | BSM}
LOG={ON | OFF}
LOGOUT={PRNT | BUFR | BUF64}
LOGTRIM={YES | NO}
COPY@RT={YES | NO}

Label LPGMNTRY [parameters]

Label (1 to 32 Characters Maximum)

Represents the symbol name provided on the LPGMNTRY entry coded in a program. It is used to provide the RSECT name for a program. It is a required parameter.

TYPE={MAIN | SUB}

Specifies whether this is a user's main program for which the initial Logger Control Area (LCA) must be acquired, or a user subprogram invocation where the called program is to utilize the LCA storage supplied from the caller if sufficient storage is available. The default value is 'SUB'. Ensure you specify TYPE=MAIN for your main program.

AMOD={24 | <u>31</u> | 64}

Specifies the user program addressing mode. Specifying AMOD=64 will allow the program to exploit z/Architecture addressing. Logger Services will follow the necessary linkage conventions, and employ the proper Format-4 and Format-5 save area protocols to properly support 64-bit zArchitecture. This is an optional parameter, and the default value is '31'. In addition, if explicit AMOD= and RMOD= entries are coded, the combination of AMOD= and RMOD= values are checked for conflict with binder conventions.

If you specify AMOD=64, Logger Services will automatically issue SYSSTATE AMODE64=YES,ARCHLVL=2. The SYSSTATE AMODE64=YES is invoked during assembly at the very start of your program when you specify AMOD=64 as you are indicating your program starts and runs in all AMODE 64.

Some macros process differently during assembly according to the SYSSTATE specification. Macros such as ATTACHX, CALL, LINKX, LOAD, XCTLX, and others will expand differently and build their parameter lists consisting of 8-byte entries when SYSSTATE AMODE64=YES if this is what you intend. Also, starting with z/OS V1R13, the DFSMS macros were changed to issue an assembly-time error message and suppress expansion if they are invoked anytime where a SYSSTATE AMODE=64 environment is active.

When coding your own program where you have specified AMOD=64 to Logger Services which causes a SYSTATE AMODE=64 to be issued for your user program, you will need to be aware of your environment if at a later place in your program you invoke a service or program which is AMODE sensitive or AMODE 64 intolerant. For purposes of assembly, you may be required to manually code in your user program SYSSTATE with AMODE64=NO before calling a service or another program even though it does not match the execution environment you established for your own program with AMOD=64. Then you would then need to code SYSSTATE AMODE64=YES immediately after the call to restore your 64-bit environment for the remaining assembly of your user program.

RMOD={24 | ANY | <u>31</u>}

Specifies the user program residence mode. The specification is mapped to the following values: 24 => 24; ANY => 31. This is an optional parameter, and the default value is '31'.

BASE=(<u>12</u>[,n]...)

Specifies the base register(s) to be established for user program addressability. Any number of registers may be specified in the sublist. If multiple base registers are specified, enclose the list in parentheses with the register values separated by a comma. You specify the base registers in the BASE= parameter as only numeric values without a symbolic equate (e.g., BASE=(12,11), not BASE=(R12,R11). The listed registers are loaded and declared as bases in the order specified. The leftmost listed register is loaded with the lowest address.

The first base register declared "must" be register 12. If you do not do specify the BASE= parameter, the default base register value is 'BASE=(12)'.

LCAXTRA=(Rn[,Rn]...)

Specifies any extra base register(s) to address the user program's dynamic storage area (LCA). "R13" is always used as the first base register for the Logger Control Area (LCA) which is your RENT work area. This parameter may be required in order to add extra base registers if your LCA addressability associated to any one RSECT is larger than 4K. Do not specify register 13 in this parameter. R13 has already been declared automatically by Logger Services as the base register for your re-entrant save area (the LCA).

Any number of registers except R13 may be specified in a sublist. If multiple base registers are specified, enclose the list in parentheses with registers values separated by a comma. You may specify the base registers in the LCAXTRA= parameter as either numeric values or as symbol equates (e.g., LCAXTRA=(5) or as LCAXTRA=(R5). The listed registers are loaded and declared as base registers to the LCA in the order specified. This parameter is optional, and there is no default for this parameter.

STORAGE=number

This parameter is used to specify the amount of storage to be obtained for the user program initial Logger Control Area (LCA) stack. The number is specified in increments of 1024 bytes or 1K. However, the size requested will be rounded up to the next 4K boundary before the storage is obtained on a page boundary. The user's main program may specify this parameter if necessary to request a larger dynamic storage area for the LCA stack if the default is deemed insufficient in size for all the anticipated called programs in the chain. This is an optional parameter, and the default value is '32'.

If the storage you specify is too small or you do not specify this parameter and take the default which results in the storage amount being too small, then additional storage is obtained automatically. When a subordinate Assembler program using Logger Services is called from another Assembler program also using Logger Services, the called program's dynamic area in the LCA begins at the end of the caller's dynamic area in the LCA. If Logger Services detects that a called program has insufficient storage available in the previous LCA stack to support its own needs, Logger Services directs the called program to issue its own getmain to support its

LCA requirements, prevent an abend, maintain re-entrant status, and allow program execution to continue. Upon the called program exiting, the storage acquired is released. However, this may result in Logger Services being required to issue constant storage requests on behalf of the called program in order to maintain re-entrant status. Specifying a sufficient size when the STORAGE= parameter is used with TYPE=MAIN simply allows one storage request to service the re-entrant status of all programs, and frees the subordinate lower level programs using Logger Services from the overhead of having Logger Services issue constant dynamic storage requests to maintain the re-entrant status of the user's Assembler program.

LCAPFX=label

Represents the label provided for naming the user's Logger Control Area (LCA) prefix area. This is a required parameter. Recommendation is to use a three character prefix. The LCA prefix label must be unique for each RSECT where it is used in your program.

LCADSECT={YES | <u>NO</u>}

This parameter indicates whether to generate only the Logger Control Area dummy section area. Used when the requirement is to create only a map of the Logger Control Area (LCA). It is an optional parameter and used only by Logger Services. A user program should never have to use this. The default value is 'NO'.

LINKRTRN={<u>BR</u> | BSM}

This parameter indicates the return linkage instruction to generate upon exit from a program or called program. If a main program or called program entry was performed by means of the BRANCH AND SAVE AND SET MODE (BASSM) instruction, then the BRANCH AND SET MODE (BSM) with an R1 field of zeros is intended to be the standard return instruction. If this is the case, then specify BSM for the LINKRTRN= parameter in order to have the proper return linkage generated (i.e., BSM R0,R14). This parameter is optional, and the default value is 'BR'. Basically, make sure your call and return types match as in 'BASSM with BSM' and 'BASR with BR'.

LOG={<u>ON</u> | OFF}

This parameter indicates whether logging is enabled and whether to generate log trace points. Specifying OFF will result in any and all defined log points not being generated in the Assembler program RSECT where LPGMNTRY is defined, and substituted with the just the label name and a "DS 0H" statement. Specifying LOG=OFF with TYPE=MAIN disables logging for the entire program. Specifying LOG=ON with TYPE=MAIN enables logging for the entire program. Specifying LOG=OFF with TYPE=SUB disables logging only for that RSECT. This is an optional parameter, and the default value is 'ON'.

LOGOUT={PRNT | BUFR | BUF64}

When logging is enabled (e.g., LOG=ON), this parameter indicates whether log records are to be printed immediately to the output data set at the time they are generated, or if the log records are to be written to an accumulation buffer area where printing of the log records to the output data set is deferred until user program end.

LOGOUT=BUFR is an optional parameter, and results in the write of log records to a buffer area in storage 'above-the-line' which will avoid log record I/O during user program execution until the end of the user's program. At the end of user program execution the log records accumulated in the buffers are printed immediately to the output data set //LGRECOUT. However, specifying BUFR may result in very large memory use if a high volume of log record output is generated. The amount of available storage will be subject to the REGION size set for the job.

LOGOUT=BUF64 is an optional parameter, and results in the write of log records to a buffer area in storage 'above-the-bar' which will avoid log record I/O during user program execution until the end of the user's program. However, specifying BUF64 may result in very large memory use if a high volume of log record output is generated. The amount of available storage will be subject to the MEMLIMIT set for your address space where the user job is executing. However, even if a higher MEMLIMIT is set for your job, the maximum amount of above-the-bar storage that will be used by Logger Services to hold the log record indices and record data in memory objects will be capped at 10 gigabytes. For purposes of buffering log records, 10 gigabytes is a large enough number to be considered conceptual infinity.

LOGOUT=PRNT is the default parameter, and results in log records being printed immediately to the output data set //LGRECOUT at the time they are generated, and uses the least amount of memory.

LOGTRIM={YES | NO}

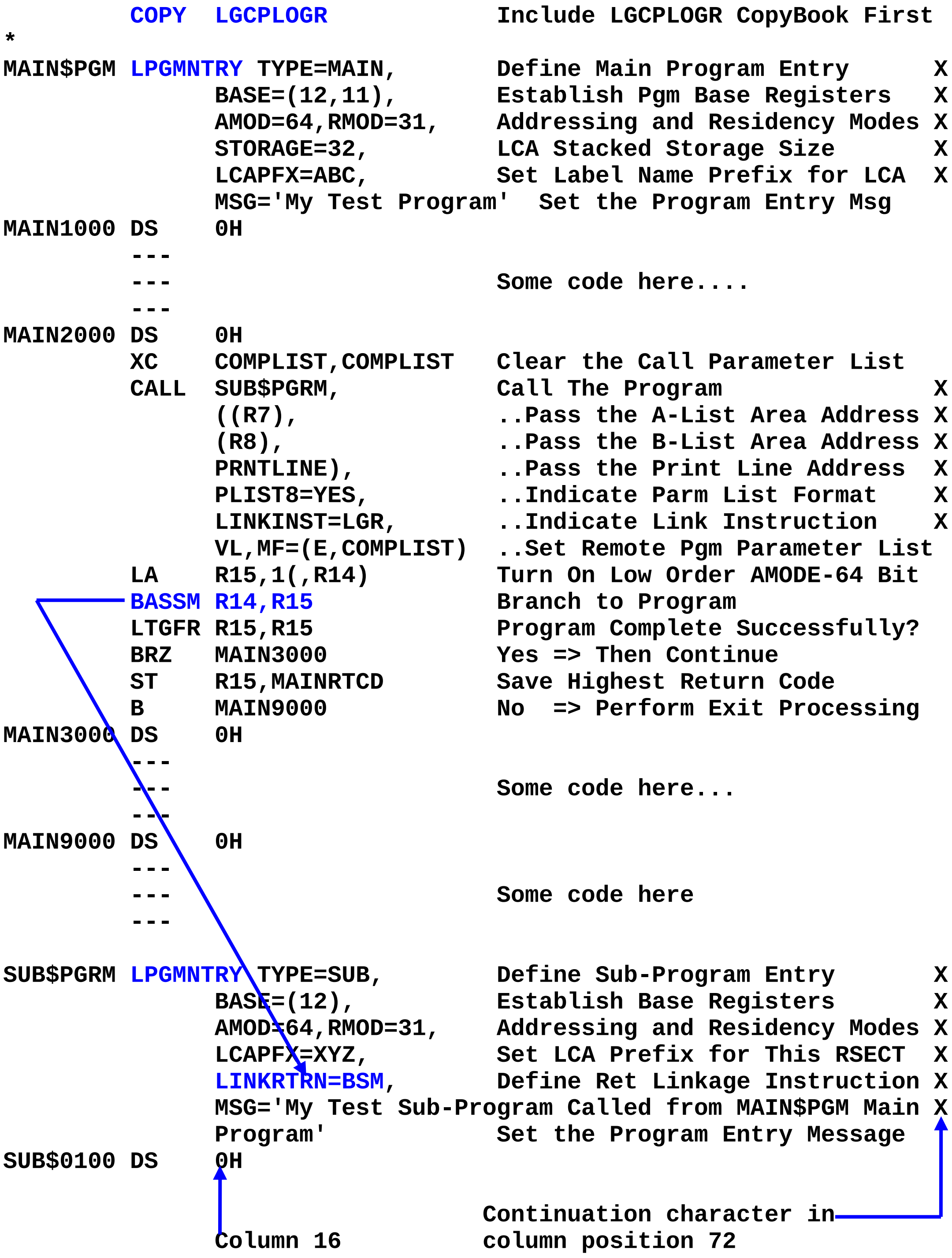
When logging is enabled (e.g., LOG=ON) this parameter when set to Yes indicates that log record output is to be trimmed by suppressing the printing of the card ruler header, the input card images, the environmental report, and the time/stamp suffix area. May be useful in reducing noise caused by timestamp differences in log records when comparing two different log files through a compare utility. May be useful if your Assembler program is being called as an exit and you want a trimmed log report. In that case use DISP=MOD on //LGRECOUT. This is an optional parameter, and the default value is 'NO'.

COPY@RT={YES | NO}

This parameter indicates whether to generate an eye-catcher notice as part of program entry. This is an optional parameter, and is used only by Logger Services. The default value is 'NO'.

4.1.1 LPGMNTRY Example

```
         COPY  LGCPLOGR            Include LGCPLOGR CopyBook First
*
MAIN$PGM LPGMNTRY TYPE=MAIN,       Define Main Program Entry      X
               BASE=(12,11),       Establish Pgm Base Registers   X
               AMOD=31,RMOD=31,    Addressing and Residency Modes X
               STORAGE=32,         LCA Stacked Storage Size       X
               LCAPFX=ABC,         Set Label Name Prefix for LCA  X
               MSG='My Test Program'  Set the Program Entry Msg
MAIN1000 DS    0H
         ---
         ---                       Some code here....
         ---
MAIN2000 DS    0H
         XC    COMPLIST,COMPLIST   Clear the Call Parameter List
         CALL  SUB$PGRM,           Call The Program               X
               ((R7),              ..Pass the A-List Area Address X
               (R8),               ..Pass the B-List Area Address X
               PRNTLINE),          ..Pass the Print Line Address  X
               PLIST4=YES,         ..Indicate Parm List Format    X
               LINKINST=BASR,      ..Indicate Link Instruction    X
               VL,MF=(E,COMPLIST)  ..Set Remote Pgm Parameter List
         LTR   R15,R15             Program Complete Successfully?
         BZ    MAIN3000            Yes => Then Continue
         ST    R15,MAINRTCD        Save Highest Return Code
         B     MAIN9000            No  => Perform Exit Processing
MAIN3000 DS    0H
         ---
         ---                       Some code here...
         ---
MAIN9000 DS    0H
         ---
         ---                       Some code here
         ---


SUB$PGRM LPGMNTRY TYPE=SUB,        Define Sub-Program Entry       X
               BASE=(12),          Establish Base Registers       X
               AMOD=31,RMOD=31,    Addressing and Residency Modes X
               LCAPFX=XYZ,         Set LCA Prefix for This RSECT  X
               MSG='My Test Sub-Program Called from MAIN$PGM Main X
               Program'            Set the Program Entry Message
SUB$0100 DS    0H

                                  Continuation character in
               Column 16          column position 72
```



```
         COPY  LGCPLOGR            Include LGCPLOGR CopyBook First
*
MAIN$PGM LPGMNTRY TYPE=MAIN,       Define Main Program Entry      X
               BASE=(12,11),       Establish Pgm Base Registers   X
               AMOD=64,RMOD=31,    Addressing and Residency Modes X
               STORAGE=32,         LCA Stacked Storage Size       X
               LCAPFX=ABC,         Set Label Name Prefix for LCA  X
               MSG='My Test Program'  Set the Program Entry Msg
MAIN1000 DS    0H
         ---
         ---                       Some code here....
         ---
MAIN2000 DS    0H
         XC    COMPLIST,COMPLIST   Clear the Call Parameter List
         CALL  SUB$PGRM,           Call The Program               X
               ((R7),              ..Pass the A-List Area Address X
               (R8),               ..Pass the B-List Area Address X
               PRNTLINE),          ..Pass the Print Line Address  X
               PLIST8=YES,         ..Indicate Parm List Format    X
               LINKINST=LGR,       ..Indicate Link Instruction    X
               VL,MF=(E,COMPLIST)  ..Set Remote Pgm Parameter List
         LA    R15,1(,R14)         Turn On Low Order AMODE-64 Bit
         BASSM R14,R15             Branch to Program
         LTGFR R15,R15             Program Complete Successfully?
         BRZ   MAIN3000            Yes => Then Continue
         ST    R15,MAINRTCD        Save Highest Return Code
         B     MAIN9000            No  => Perform Exit Processing
MAIN3000 DS    0H
         ---
         ---                       Some code here...
         ---
MAIN9000 DS    0H
         ---
         ---                       Some code here
         ---

SUB$PGRM LPGMNTRY TYPE=SUB,        Define Sub-Program Entry       X
               BASE=(12),          Establish Base Registers       X
               AMOD=64,RMOD=31,    Addressing and Residency Modes X
               LCAPFX=XYZ,         Set LCA Prefix for This RSECT  X
               LINKRTRN=BSM,       Define Ret Linkage Instruction X
               MSG='My Test Sub-Program Called from MAIN$PGM Main X
               Program'            Set the Program Entry Message
SUB$0100 DS    0H
```

This is an example of a main program calling a subprogram. Since program entry to SUB$PGM was performed by means of the BRANCH AND SAVE AND SET MODE (BASSM) instruction from MAIN$PGM, then the BRANCH AND SET MODE (BSM) with an R1 field of zeros is intended to be the standard return instruction. This shows the use of the LINKRTRN= parameter in order to have the proper return linkage generated by Logger Services when SUB$PGM returns back to the calling main program.

4.2 LPGMEXIT Description

Label LPGMEXIT [parameters]
RC=[addr]
RC=[(Rn)]
RS=[addr]
RS=[(Rn)]

Label LPGMEXIT [parameters]

Label (1 to 32 Characters Maximum)

Represents the symbol name provided on the LPGMEXIT entry coded in program. It is optional, and if coded a label will be generated which may be branched to. This entry builds the program exit protocol.

RC=[addr]
[(Rn)]

Represents the program return code value to be passed to the caller in R15. This may be specified as a program label for a "fullword" field (4-bytes) containing the return code, or a register value enclosed in parentheses. If the parameter is omitted, R15 will be set to binary zeros upon program exit.

RS=[addr]
[(Rn)]

Represents the program reason code value to be passed to the caller in R0. This may be specified as a program label for a "fullword" field (4-bytes) containing the reason code, or a register value enclosed in parentheses. If the parameter is omitted, R0 will be set to binary zeros upon program exit.

Note: The LPGMEXIT entry requires a preceding LPGMNTRY entry.

4.2.1 LPGMEXIT Example

```
PGM$EXIT DS    0H
         ST    R0,PGMRSNCD          Set the Reason Code
         ST    R15,PGMRETCD         Set the Return Code
         LPGMEXIT RC=PGMRETCD,      Define Program Exit Protocol   X
               RS=PGMRSNCD
                                      Continuation character in
                                      column position 72

PGM$EXIT DS    0H
         LPGMEXIT RC=(R3),RS=(R5)   Define Program Exit Protocol
```

4.3 LPGMSUBE Description

Label LPGMSUBE [parameters]
RETRN={<u>R14</u> | Rn}
MSG=('*message text*'[,variable])
PVTAREA={YES | <u>NO</u>}
SHOW=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)
WATCH=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)
REG={NONE | ALL | (Rn[,Rn]...)}

Label LPGMSUBE [parameters]

Label (1 to 32 Characters Maximum)
Represents the symbol name provided on the LPGMSUBE entry coded in a program. This parameter is required as it defines the name of the subroutine.

RETRN={<u>R14</u> | Rn}
Specifies the general purpose register to use for the return address. The return address is the address of the instruction following the branch instruction in storage. When the BRANCH AND SAVE instruction (BAS and BASR) is used for linkage to subroutines defined with LPGMSUBE for entry, you may specify a register that is to be used for the return address. This is an optional parameter, and the default is 'R14'.

Note: Additional parameters which may be specified with the LPGMSUBE entry are the same as those available to the #LGPOINT entry. Refer to #LGPOINT for a description.

The LPGMSUBX entry requires a preceding LPGMSUBE entry.

4.3.1 LPGMSUBE Example

```
ABCD3000 DS    0H
         BAS   R14,SUBR0000        Branch to Subroutine
    -
    -
    -
SUBR0000 LPGMSUBE ,                Subroutine Starts Here
    -
    -

ABCD3000 DS    0H
         BAS   R6,SUBR0000         Branch to Subroutine
    -
    -
    -
SUBR0000 LPGMSUBE RETRN=R6         Subroutine Starts Here
    -
```

You can also add log point parameters to a LPGMSUBE entry:

```
SUBR0000 LPGMSUBE SHOW=(('USRSECND',USRSECND,L'USRSECND),          X
             ('USRGMTDF',USRGMTDF,L'USRGMTDF),                     X
             ('USRSTRSC',USRSTRSC,L'USRSTRSC)),REG=ALL,RETRN=6,    X
             MSG=('Bug in Converting Date to Julian',USRSTRDT)
```

4.4 LPGMSUBX Description

Label LPGMSUBX [parameters]

MSG=('*message text*'[,variable])
PVTAREA={YES | NO}
SHOW=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)
WATCH=(('*text*',*address*,*length*)[,('*text*',*address*,*length*)]...)
REG={NONE | ALL | (Rn[,Rn]...)}

Label LPGMSUBX [parameters]

Label (1 to 32 Characters Maximum)

Represents the symbol name provided on the LPGMSUBX entry coded in a program. This parameter is required as it defines the name of the subroutine exit. It requires a preceding LPGMSUBE entry.

Note: Return is through a "BR Rn" instruction where the register number is the register specified in the RETRN= parameter of the LPGMSUBE entry. If a RETRN= parameter was not specified in a previous LPGMSUBE, then the default 'R14' is used as in "BR R14".

Additional parameters which may be specified with the LPGMSUBX entry are the same as those available to the #LGPOINT entry. Refer to #LGPOINT for a description.

4.4.1 LPGMSUBX Example

```
ABCD3000 DS    0H
         BAS   R14,SUBR0000        Branch to Subroutine
         MVC   HERE,THERE          Something
    -
    -
    -
SUBR0000 LPGMSUBE ,                Subroutine Starts Here
  *
  ** Some code goes here
  *
SUBR9000 LPGMSUBX ,                Subroutine Exits and Returns to
                                   Address in Register 14 (R14 is
                                   the Default)
```

```
ABCD3000 DS    0H
         BAS   R6,SUBR0000          Branch to Subroutine
         MVC   HERE,THERE           Something
    -
    -
    -
SUBR0000 LPGMSUBE RETRN=R6          Subroutine Starts Here using R6
  *                                 for Entry
  ** Some code goes here
  *
SUBR9000 LPGMSUBX ,                 Subroutine Exits and Returns to
                                    Address in Register 6 (R6) because
                                    Register 6 Specified in RETRN=
```

4.5 LCA Description

Label LCA {DEFINE | DEFEND}

Label (1 to 32 Characters Maximum)
Represents the symbol name provided on the LCA entry coded in program source. This parameter is optional.

DEFINE
DEFINE indicates this is the beginning of the user's Logger Control Area, and that the LCA prefix area must be generated. This is a dynamic storage area below the line addressable by R13, contains the save area, and is to be used by the user program as a work area to maintain re-entrant status. This is your RENT DSECT area where you may specify individual fields defined via DS instructions with your own labels, and use as a general work area.

DEFEND
DEFEND indicates this is the end of the Logger Control Area for this CSECT.

4.5.1 LCA Example

```
         -
         -
         -
WXYZ$END DS    0H
         SLR   R15,R15              Indicate No Errors in Processing
WXYZEXIT DS    0H
         ST    R15,XYZRETCD         Set the Return Code
         LPGMEXIT RC=XYZRETCD       Define Program Exit Protocol
*
```

```
         LCA   DEFINE              Define Beginning of LCA Storage
*
XYZWORKD DS    10D                 General Work Area
XYZWORKF DS    F                   General Work Area
XYZRETCD DS    F                   Area for Return Code
XYZ$BTME DS    F                   Area for Binary Time Interval
XYZINTCT DS    F                   Area for Time Interval Counter
XYZSTIME DS    0F,XL(WRKTIMRL)     Wait Timer Parameter List Area
XYZ$TIME DS    0D,XL(WRKTIMLN)     Date and Time Parameter List
XYZ$DTTM DS    0CL16               Date & Time Value from TOD Clock
XYZTIME@ DS    PL8                 Time Value as 'HHMMSS'
XYZ$DATE DS    PL4                 Date Value as 'MMDDYYYY'
         DS    XL4                 ..(Reserved)
XYZ$DTWK DS    CL9                 Date & Time General Work Area
XYZPLINE DS    CL133               Work Area for Output Print Line
*
         LCA   DEFEND              Define End of LCA Storage
```

4.6 #LGPOINT Description

Label #LGPOINT [parameters]

MSG=(*'message text'*[,variable])
PVTAREA={YES | NO}
SHOW=((*'text',address,length*)[,(*'text',address,length*)]...)
WATCH=((*'text',address,length*)[,(*'text',address,length*)]...)
REG={NONE | ALL | (Rn[,Rn]...)}
COMPRESS={YES | NO}
SUPHEAD={YES | NO}

Label #LGPOINT [parameters]

Label (1 to 32 Characters Maximum)

Represents the symbol name provided on the #LGPOINT entry coded in program source. This parameter is required as it defines the name of the log point. This name replaces the 'DS 0H' statement commonly used to define labels. The label name for #LGPOINT is limited to a maximum length of 32 characters.

4.6.1 The MSG= Parameter

MSG=('message text'[,variable])

'text' - This parameter specifies text and/or data to be printed when the log point is executed in addition to the standard logger output. It may be specified with just text or with an embedded variable. Text is to be enclosed in single quotes. If any variables are used they must be defined in the program source and be generated prior to the inclusion of the MSG= operand since the data type and length of the variable are used to determine the proper format. Variable data will be formatted according to the data type specified in the DC/DS instruction. (F)ullword, (H)alfword, and (P)acked constants will be printed as decimal numbers, (Z)oned and (C)haracter constants in character, and everything else in hexadecimal representation. Halfwords are assumed to be 2 bytes in length, and fullwords 4 bytes in length. Everything else will be based on the length attribute of the field.

4.6.2 The PVTAREA= Parameter

PVTAREA={YES | NO}

{YES | NO}

Indicates whether to print the current storage allocations for the Private Area. Displays storage allocations for Below-the-Line, Above-the-Line, and Above-the-Bar. While you program is executing this will display a subpool summary report for your job showing how all of the allocated and free space storage amounts in the private area below-the-bar are distributed among the various subpools. CPU and SRB time used are also displayed. Allows you to see the current private area storage allocations and CPU time used at any particular point in your program path where a Logpoint with PVTAREA=YES is defined. The storage area report is printed to the //LGRECOUT DD. The default value is NO.

Storage Analysis Example:

```
XYZ00000 DS   0H
```

changed to

```
XYZ00000  #LGPOINT PVTAREA=YES
```

Results in following being written to output during program execution whenever XYZ00000 is encountered in the execution path of a user program.

```
Log) ==>Log Point**: XYZ00000
Private Area Storage Allocated:
   <16M: In Use=620K       Unused=8571K       Limit=9192K       HighU=620K
   >16M: In Use=347942K    Unused=1352921K    Limit=1700864K    HighU=347942K
   >BAR: Allocated=10,000 Megabytes                 Guard Amount=6,000 Megabytes
   >BAR: Useable  =4,000 Megabytes                  High Water  =4,000 Megabytes
   >BAR: Objects  =1                                Shr Objects =0
   >BAR: Shr Alloc=0 Megabytes                      Shr Hi-Water=0 Megabytes
 Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE
  SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space
          Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line
  003-8   3,968       1           475,136     2,872       352,239,616     0
  004-8   500         500         0           0           2,048,000       536,000
  005-8   500         500         0           0           2,048,000       48,000
  229-5   2           2           0           0           12,288          5,952
  230-0   1           1           4,096       760         0               0
  230-5   3           7           8,192       7,232       4,096           3,424
  230-8   1           1           0           0           4,096           4,072
  251-8   2           2           0           0           200,704         5,832
          ----------  ----------  ----------  ----------  --------------  --------------
  Totals  4,977       1,014       487,424     10,864      356,556,800     603,280
 Job Step CPU: 0HR 2MIN 24.68882SEC       Job Step SRB: 0HR 0MIN 0.18772SEC
```

VSM behavior in z/OS V1R10 was changed for low private subpools. Requests for storage will be biased to allocate from the low end of the DQE. Also, adjacent DQEs will be merged, and storage for a single request may be satisfied in part from the free space at the high end of an existing DQE, and in part from adjacent new DQEs and merged DQEs. This change in z/OS V1R10 is intended to be a performance benefit by reducing the amount of fragmentation resulting in fewer DQEs and FQEs, and reducing the number of control blocks VSM needs to maintain to hold task related storage management information.

This behavior will be most biased in situations when obtaining page-aligned storage for storage lengths of various amounts for the same subpool and key that end on a 4K boundary (no free areas) whether the address of the storage returned is going forward or backward on successive storage requests, and the cumulative result of the several storage requests produces allocated storage areas that are contiguous or adjacent to one another.

This means a group of 4K aligned allocated storage areas under the same subpool and storage protection key which are all adjacent to one another will be counted in V1R10 as one area under one DQE even if, for example, that larger area took 1,000 individual page-aligned Getmains/Storage-Obtains to eventually acquire. In V1R9 or lower each individually allocated storage area from Getmain/Storage was accounted for, and VSM supplied a block descriptor even if the area was 4K aligned, had no free space, and all the separate areas were contiguous to one another. This means when using the PVTAREA=YES in your program on V1R10, under the same exact conditions the area counts reported in the Subpool Summary Report for DQEs in z/OS V1R10 may be significantly different (less) than those reported for your same Assembler program performing the same work running on z/OS V1R9 or lower versions of z/OS. Basically the PVTAREA=YES output may show descriptor queue element (DQE) and free queue element (FQE) counts that are higher since the DQE/FQE chains may be longer for user private area subpools when at z/OS V1R9 or less, or if you are using VSM USEZOSV1R9RULES(NO) in DIAG00 of SYS1.PARMLIB when at z/OS V1R10 or above.

For additional information refer to IBM APAR OA27291.

This APAR is associated with ZOS 1.10. There is an option from a follow-on PTF which came out later to allow you in V1R10+ to revert back to the previous behavior by specifying VSM USEZOSV1R9RULES(YES) in the DIAG*xx* member of SYS1.PARMLIB. You can issue the SET DIAG=*xx* command to dynamically activate the new setting.

4.6.3 The SHOW= Parameter

SHOW=(('text',address,length)[,('text',address,length)]...)

‘text’

This parameter specifies any text message that should be displayed to identify the data area being printed to the log. Specify the text enclosed in single quotes.

address

This specifies the address of the storage area to be printed including 64-bit addresses pointing to storage areas above-the-bar. It may be specified as a program label, or in register format (i.e., (15) or (R15)). The program label references a storage location which needs to be within the addressable range of the designated base register in order for Logger Services to resolve the address. The addressable range is based on Logger supporting a 20-bit displacement. When using a program label the addressable range of the label must be within one megabyte (-512K to +512K) from the address in the base register. This means you need to ensure the program label you specify in the SHOW= is within the one megabyte addressable range of the base register. Otherwise, as another option you can use register notation where you can place the address of any accessible storage area you want to see in a register, and the storage area will be displayed. In register format any register available for use by your program (R0-R15) may be specified. No contents of any registers are changed or altered, and after return from SHOW= the contents of the register used in the SHOW= or any register will be the same as before using the SHOW= service. The storage area to show can be below-the-line, above-the-line, or above–the-bar.

length

This specifies the length of the storage area data to be printed. The minimum length is 1 byte up to a maximum of 32K-1 (32767). It may be specified as an equated value, or as a hard coded numeric expression, or as a register value enclosed in parentheses (e.g., DSECTLEN, L'ANYTHING, 16*4, (R15)) allowing the length of the area to be displayed to be obtained from a register during program execution.

Note: A show point dumps a specified range of storage bytes for the storage area of interest while your program is executing. With the SHOW= parameter, the entire contents of the storage area for the length specified by the user in the SHOW= parameter is printed to the //LGRECOUT DD in a dump format.

When the #LGPOINT SHOW= is coded in your program path, you are not required to be in AMODE(64) to execute the #LGPOINT to display a storage area above-the-bar. However, if you are using register notation in a Logpoint and use a modal instruction to place an address in a register that is a 64-bit address which will be used by a Logpoint, then you will need to be in Amode64 when you execute the modal instruction before entering the Logpoint, but you are not required to be in Amode64 when entering the Logpoint to show a storage area above-the-bar. This is because a modal instruction behaves differently based on the AMODE in effect at the time where the AMODE determines the width of the output register operands. An example of a modal instruction is Load Address (LA).

Some examples of SHOW=:

```
XYZ00000  DS   0H
```

changed to

```
XYZ00000  #LGPOINT SHOW=('My Data Area',MYAREA,L'MYAREA)
```

Results in storage area at label MYAREA being displayed when XYZ00000 is encountered in the execution path of a user program.

```
Log) ==>Log Point**: XYZ00000
+    My Data Area
+       00000000010465E4          7EF87EFA 7EFC7EFE 7F007F02 7F047F06   *=8=.=.=.........*
+       00000000010465F4     +10  7F087F0A 7F0C7F0E 7F107F12 7F147F16   *................*
+       0000000001046604     +20  7F187F1A 7F1C7F1E 7F207F22 7F247F26   *................*
+       0000000001046614     +30  7F287F2A 7F2C7F2E 7F307F32 7F347F36   *................*
+       0000000001046624     +40  7F387F3A 7F3C7F3E 7F407F42 7F447F46   *......... ......*
+       0000000001046634     +50  7F487F4A 7F4C7F4E 7F507F52 7F547F56   *.....<.+.&......*
+       0000000001046644     +60  7F587F5A 7F5C7F5E 7F607F62 7F647F66   *...!.*...-......*
+       0000000001046654     +70  7F687F6A 7F6C7F6E 7F707F72 7F747F76   *.....%.>........*
+       0000000001046664     +80  7F787F7A 7F7C7F7E 7F807F82 7F847F86   *...:.@.=...b.d.f*
+       0000000001046674     +90  7F887F8A 7F8C7F8E 7F907F92 7F947F96   *.h.........k.m.o*
+       0000000001046684     +A0  7F987F9A 7F9C7F9E 7FA07FA2 7FA47FA6   *.q.........s.u.w*
+       0000000001046694     +B0  7FA87FAA 7FAC7FAE 7FB07FB2 7FB47FB6   *.y..............*
+       00000000010466A4     +C0  7FB87FBA 7FBC7FBE 7FC07FC2 7FC47FC6   *...........B.D.F*
+       00000000010466B4     +D0  7FC87FCA 7FCC7FCE 7FD07FD2 7FD47FD6   *.H.........K.M.O*
+       00000000010466C4     +E0  7FD87FDA 7FDC7FDE 7FE07FE2 7FE47FE6   *.Q.........S.U.W*
+       00000000010466D4     +F0  7FE87FEA 7FEC7FEE 7FF07FF2 7FF47FF6   *.Y.......0.2.4.6*
```

```
            LG R2,ORIGIN            Get Above-the-Bar Address
XYZ01000  DS   0H
```

changed to

```
            LG R2,ORIGIN            Get Above-the-Bar Address
XYZ01000  #LGPOINT SHOW=('IARV64 AREA',(R2),32760)
```

Results in a storage area above-the-bar being displayed when XYZ01000 is encountered in the execution path of your program. Your program is not required to be in AMODE(64) when the Logpoint is executed.

```
Log) ==>Log Point**: XYZ01000
+    IARV64 AREA
+       0000000800000000          00000000 00000000 00000000 00000000   *................*
+       0000000800000010     +10  0000000800000010 TO 00000008000007FF SAME AS ABOVE
+       0000000800000800    +800  01F30000 00000000 00000000 00000000   *.3..............*
+       0000000800000810    +810  00000000 00000000 00000000 00000000   *................*
+       0000000800000820    +820  0000000800000820 TO 0000000800007FEF SAME AS ABOVE
+       0000000800007FF0   +7FF0  00000000 00000000                     *........        *
```

The following results in a storage area above-the-bar being displayed when label LVNT5000 is encountered in the execution path of your program. Your program is not required to be in AMODE(64) when the Logpoint is executed, but prior to Logpoint a modal instruction LA was used which required Amode64 to ensure the address of CLDEDATE fills the entire 64-bits of Register 3.

```
         LG R2,ORIGIN               Get starting address
         USING CLNDNTRY,R2          Establish Addressability
         SAM64                      Switch Now to 64-Bit Addressing Mode
         LA    R3,CLDEDATE          Above Bar Address for Date Field
         SAM31                      Switch Back to 31-Bit Address Mode
         DROP  R2                   Remove Listed Register as Base Reg
LVNT5000 #LGPOINT SHOW=('Above the Bar Area',(R3),64),REG=ALL
```

4.6.4 The WATCH= Parameter

WATCH=(('text',address,length)[,('text',address,length)]...)

'text'

This parameter specifies any text message that should be displayed to identify the data being printed to the log when a storage alteration occurs. Specify the text enclosed in single quotes.

address

This specifies the address of the storage area to be monitored including 64-bit addresses pointing to storage areas above-the-bar. It may be specified as a program label, or in register format (i.e., (15) or (R15)). The program label names a storage location which needs to be within the one megabyte addressable range of the designated base register in order for Logger Services to be able to resolve the address into a 20-bit displacement form. Since the addressable range of a base register is one megabyte, ensure the program label you specify in the WATCH= is within the one megabyte (-512K to+512K) addressable range of the base register. Otherwise, with the register notation option you can place the address of any accessible area you want to see into a register and the storage area will be displayed. In register format any register available for use by your program (R0-R15) may be specified. The contents of all registers are not changed, and after return from the WATCH= they will be the same as they were before using the WATCH= service. The storage area to watch can be below-the-line, above-the-line, or above–the-bar.

length

This specifies the length of the data area to be watched up to 32K-8 (32760). It may be specified as an equated value, or as a hard coded numeric expression, or as a register value enclosed in parentheses (e.g., DSECTLEN, L'ANYTHING, 16*4, (R15)) allowing the length of the area to be displayed to be obtained from a register during program execution.

Note: A watch point analyzes a specified range of storage bytes for any changes while your program is executing. A watch point then becomes a lookout for a range of storage which if modified will indicate that a change has occurred to a storage area. The watch does not detect if since the last pass the storage area or some portion thereof may have been updated with the same exact data in the same exact place, nor does it indicate the byte location of where the change may have occurred if there was an alteration. Only when a change is detected will the watch point dump the storage contents indicating a change has occurred. When a change is detected you will need to look at the dump of the storage area produced by the watch point to see where the data in a storage has changed. The watch is maintained through the generation of a unique token value for the storage area of interest when a #LGPOINT WATCH= entry is encountered. Up to 512 watch point tokens can be supported through the use of #LGPOINT WATCH= commands.

When the #LGPOINT WATCH= is coded in your program path, you are not required to be in AMODE(64) to execute the #LGPOINT to watch a storage area above-the-bar. However, if you are using register notation in a Logpoint and use a modal instruction to place an address in a register that is a 64-bit address which will be used by a Logpoint, then you will need to be in Amode64 when you execute the modal instruction before entering the Logpoint, but you are not required to be in Amode64 when entering the Logpoint to watch a storage area above-the-bar. This is because a modal instruction behaves differently based on the AMODE in effect at the time where the AMODE determines the width of the output register operands. An example of a modal instruction is Load Address (LA).

Upon the first time through, the entire storage area for the length specified by the user in the WATCH= parameter is printed to the //LGRECOUT DD in a dump format. On any subsequent passes where a watch point (#LGPOINT WATCH=) is encountered in the program path for the same storage area and the same length, then the storage area under a watch is analyzed for any changes. The storage watch is processed by Logger Services program LGMHWTCH.

If no changes have occurred to the range of bytes under a watch since the last pass, only the program label for the #LGPOINT WATCH= entry is printed to show program flow, but the printing of the storage area is suppressed. Suppressing the output until the watch point detects the storage alteration allows you to avoid generating large amounts of unnecessary output before the storage change is encountered.

Therefore, the printing of the storage area contents anytime after the first time through is the indicator that a change has occurred within a storage area of interest.

If a change is detected, then the range of bytes subject to the storage watch is printed in a dump format where the user can review the dump output from the watch to look for changes to the storage area, whether expected or unexpected. The user can examine the output to verify if a data change is what was intended, or if the storage was updated as intended but the wrong data was stored, or if a storage area was improperly altered due to an unintended overlay.

For example, to set a watch point for a storage area:

```
         LG    R2,ORIGIN               Get high area address
XYZ00000 DS    0H
```

XYZ00000 is changed to add a Logpoint:

```
         LG         R2,ORIGIN          Get high area address
XYZ00000 #LGPOINT WATCH=('IARV64 AREA',(R2),32760)
```

On first time through XYZ00000 in the program path the entire storage area will be displayed.

On the next pass when XYZ00000 is encountered in the execution path of a user program such as when XYZ00000 is inside of a loop, the storage area under a watch will be checked. If no change has occurred to the storage area under a watch, then only the program label is displayed indicating (1)this part of your program path has been encountered, and (2)the storage contents have not changed since the last pass.

```
Log) ==>Watch Point: XYZ00000
```

On the next pass through XYZ00000 in your program path the storage area under a watch will be analyzed, and if a change has occurred since the last pass then the storage area under the watch will be displayed

```
Log) ==>Watch Point: XYZ00000
+    IARV64 AREA
+       0000000100000000          2D2F6263 64656667 6869202C 255F3E3F   *................*
+       0000000100000010     +10  70717273 74757677 78603A23 40273D22   *.........-.. ...*
+       0000000100000020     +20  80616263 64656667 68698A8B 8C8D8E8F   *./..............*
+       0000000100000030     +30  906A6B6C 6D6E6F70 71729A9B 9C9D9E9F   *..,%_>?.........*
+       0000000100000040     +40  A07E7374 75767778 797AAAAB ACADAEAF   *.=......:.......*
+       0000000100000050     +50  5EB1B2B3 B4B5B6B7 B8B95B5D BCBDBEBF   *..........$)....*
+       0000000100000060     +60  7B414243 44454647 4849CACB CCCDCECF   *#...............*
+       0000000100000070     +70  7D4A4B4C 4D4E4F50 5152DADB DCDDDEDF   *'..<(+|&........*
+       0000000100000080     +80  5CE15354 55565758 595AEAEB ECEDEEEF   **........!......*
+       0000000100000090     +90  30313233 34353637 38392020 202020FF   *................*
+       00000001000000A0     +A0  40404040 40404040 40404040 40404040   *                *
+       00000001000000B0     +B0  00000001000000B0 TO 000000010000039F SAME AS ABOVE
+       00000001000003A0    +3A0  01010101 01010101 01010101 01010101   *................*
+       00000001000003B0    +3B0  00000001000003B0 TO 00000001000003FF SAME AS ABOVE
+       0000000100000400    +400  00000000 00000000 00000000 00000000   *................*
+       0000000100000410    +410  0000000100000410 TO 0000000100007FEF SAME AS ABOVE
+       0000000100007FF0   +7FF0  00000000 00000000                     *........        *
```

4.6.5 The REG= Parameter

REG={NONE | ALL | (Rn[,Rn]...)}

NONE

Indicates that no registers are to be printed when the program log point is executed. The default is 'NONE'.

ALL

Indicates that all 16 64-bit general purpose registers and all 16 32-bit access registers are to be printed when the program log point is executed.

Rn

Specifies a 64-bit general purpose register and the corresponding same numbered 32-bit access register are to be printed when the program log point is executed. Specify R0 to R15 as a parameter value to identify the register. A single register, or any combination of registers numbered R0-R15 may be specified in any order. The registers displayed will be in numeric ascending order. If multiple registers are specified, enclose the list of registers in parentheses with the register values separated by a comma.

Display Registers Example:

```
XYZ01000 DS   0H
```

changed to:

```
XYZ01000  #LGPOINT REG=ALL
```

Results in following being written to output during program execution when XYZ01000 is encountered in the execution path of your program.

```
Log) ==>Log Point**: XYZ01000
 +    General Purpose Registers At Log Point: XYZ01000
 +    R0 = 0000000000000014  R1 = 0000000000008928  R2 = 0000000000008929  R3 = 00000000008B6A08
 +    R4 = 0000000000000001  R5 = 000000007FFE37C8  R6 = 00000000008B9028  R7 = 00000000183091D8
 +    R8 = 00000000183081D8  R9 = 0000000000012B38  R10= 0000000000011B38  R11= 0000000000000000
 +    R12= 0000000018302530  R13= 00000000000085D8  R14= 0000000098302CEC  R15= 0000000000D0BB18
 +    Access Registers:
 +    AR0 = 00000000         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000
 +    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000
 +    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000
 +    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000C05
```

4.6.6 The COMPRESS= Parameter

COMPRESS={<u>YES</u> | NO}

YES | NO

Indicates whether multiple spaces should be removed from the message before printing. The default is YES. This is actually compaction and not compression. Compaction is done here as it is a simple process of extracting blanks from a line of text. Compression is a much more complex task, and dictionaries are generally used to obtain the best result where dictionaries contain the most repeated set of characters found in the text, and their respective compressed substitution.

4.6.7 The SUPHEAD= Parameter

SUPHEAD={YES | NO}

YES | NO

A Value of 'NO' suppresses the 'Log) ==>Log Point**: NAME' header and the '+', 'Msg)' line indicators. This is parameter is used internally only by Logger Services when printing selected storage areas during abend processing of a program.

4.6.8 The NAME= Parameter

NAME=name

name

Specifies the name of the log point when generated from within another macro. Used internally only by Logger Services.

4.6.9 #LGPOINT Examples

```
         LA    R1,1(,R1)
         STH   R1,0(,R2)
         L     R3,COUNTR
USER5100 DS    0H
         L     R1,16
         L     R1,0(R1)
         L     R1,4(,R1)
```

Show label USER5100 in output report as an indicator that this path was executed.

```
         LA    R1,1(,R1)
         STH   R1,0(,R2)
         L     R3,COUNTR
USER5100 #LGPOINT
         L     R1,16
         L     R1,0(R1)
         L     R1,4(,R1)
```

Show contents of all registers at label USER5100 when path is executed.

```
         LA    R1,1(,R1)
         STH   R1,0(,R2)
         L     R3,COUNTR
USER5100 #LGPOINT REG=ALL
         L     R1,16
         L     R1,0(R1)
         L     R1,4(,R1)
```

Show contents of only registers 1, 3, and 5 at label USER5100 when path is executed.

```
         LA    R1,1(,R1)
         STH   R1,0(,R2)
         L     R3,COUNTR
USER5100 #LGPOINT REG=(R1,R3,R5)
         L     R1,16
         L     R1,0(R1)
         L     R1,4(,R1)
```

Show contents of all registers and show contents of storage at field USRLABL1 for the length of field USERLABL1 when path is executed at label USER5100 in program.

```
         LA    R1,1(,R1)
         STH   R1,0(,R2)
         L     R3,COUNTR
USER5100 #LGPOINT SHOW=('USRLABL1',USRLABL1,L’USRLABL1),REG=ALL
         L     R1,16
         L     R1,0(R1)
         L     R1,4(,R1)
```

Show message text ‘IARV64 Area’ and show the contents of an above-the-bar storage area pointed to by Register 4 for a length of 32760 bytes when path is executed at label USER5200 in program.

```
         LG    R4,ORIGIN
USER5200 #LGPOINT SHOW=('IARV64 Area',(R4),32760)
```

Show message text ‘USERDATA’ and show contents of the storage area pointed to by Register 1 for a length of 256 bytes when the path is executed at label USER5300 in the program.

```
         LLGT  R1,=A(USERDATA)
USER5300 #LGPOINT WATCH=('USERDATA',(R1),256)
```

Show message text ‘MYDATA’ and show contents of a storage area 384 bytes from the address in Register 0 for the length indicated in Register 15 when the path is executed at label USER5400 in program.

```
         LA    R15,256
         LA    R0,MYDATA
USER5400 #LGPOINT SHOW=('MYDATA',384(R0),(R15))
```

In the following example at program label USER5100 create a concatenated SHOW to display message text ‘USRLABL1’ and display the contents of the storage area at label USRLABL1 for the length of USRLABL1. Then display message text ‘USRLABL2’ and display the contents of a storage area at label USRLABL2 for the length of USRLABL2. Next display message text ‘Show area X’ and display the contents of a storage area at label USRAREAX for the length of USRAREAX, then display message text ‘This is my table’ and display the contents of a storage area at label EBCTOASC for the length of EBCTOASC. Then use MSG= to display a message 'Check Date and Time Stamp' and display the contents of the field labeled USRDATES. The concatenation will perform all of the above actions requested in a single Logpoint.

```
USER5100 #LGPOINT SHOW=(('USRLABL1',USRLABL1,L'USRLABL1),              X
               ('USRLABL2',USRLABL2,L'USRLABL2),                       X
               ('Show area X',USRAREAX,L'USRAREAX),                    X
               ('This is my table',EBCTOASC,EBCTOASL)),REG=ALL,        X
               MSG=('Check Date and Time Stamp',USRDATE$)
```

In the following example at program label USER5100 code the #LGPOINT with REG=ALL to show the contents of all the general purpose registers and access registers, add a concatenated SHOW to display message text ‘NEW-BLK’ and display the contents of a storage area at the address in Register 1 for a length of 28 bytes, then display message text ‘HASHDATA’ and display the contents of a storage area at the address in register 2 for the length of equated value HASHDLEN. Next display your own personal message ‘Checking Area Control Blocks and Storage’ to add a more detailed description to what activity the Log Point you coded is performing.

```
USER5100 #LGPOINT REG=ALL,SHOW=(('NEW-BLK',(R1),28),                    X
               ('HASHDATA',(R2),HASHDLEN)),PVTAREA=YES,                X
               MSG=('Checking Area Control Blocks and Storage')
```

In addition the keywords are not positional, and you may list them in any order.

```
USER5100 #LGPOINT PVTAREA=YES,REG=ALL,SHOW=(('NEW-BLK',(R1),28),        X
               ('HASHDATA',(R2),HASHDLEN)),                            X
               MSG=('Checking Area Control Blocks and Storage')

USER5100 #LGPOINT MSG=('Checking Area Control Blocks and Storage'),    X
               PVTAREA=YES,SHOW=(('NEW-BLK',(R1),28),                  X
               ('HASHDATA',(R2),HASHDLEN)),REG=ALL
```

In the following example at program label USER5100 code the #LGPOINT with MSG= to display an ‘I am here’ message to check if you are reaching this point in your program.

```
USER5100 #LGPOINT  MSG='I am here'
```

In the following example at program label USERLABELNAME_OF_THIRTYTWO_BYTES a #LGPOINT was coded with REG=ALL to show the contents of all the general purpose registers and access registers at the time this execution path is reached in the program.

```
USERLABELNAME_OF_THIRTYTWO_BYTES DS  0H
            -
USERLABELNAME_OF_THIRTYTWO_BYTES #LGPOINT REG=ALL
```

The #LGPOINT is used to generate a log point within a program. This will allow LOGGRASM Logger Services to capture data while your program is executing, and show the requested data in the output data set for DDname //LGRECOUT.

Basically you can use a #LGPOINT to replace the 'DS 0H' statement commonly used to define labels in your program. As long as any program labels specified in the #LGPOINT parameters are currently addressable from the place where #LGPOINT is executed, using this entry in a program will allow you to show any registers, variable data, storage areas, or messages at any location in a program where the #LGPOINT entry is coded allowing you to examine in detail the execution path and logic of your Assembler program.

You can also specify a #LGPOINT with no parameters which will indicate when you have reached a path in your program logic.

USER0200 DS 0H

Change to

USER0200 #LGPOINT

Shows in the output the following:

```
Log) ==>Log Point**: USER0200
```

In addition, instead of using a program label in a Log Point you may use register notation to specify any address, and as long as your program can access the storage area (even above-the-bar), and it is not fetch protected, you will be able to show that storage area with a #LGPOINT. If in error you specify a bad address on a #LGPOINT entry, Logger Services incorporates an ESTAEX recovery exit with retry to allow the user's program to continue. This is intended to prevent Logger Services from terminating the user's program because Logger Services encountered an abend (e.g., S0C4) while attempting to show storage at an invalid address specified by the user.

5.0 Input Control Card Format: //LGRSYSIN

LOGGRASM Logger Services utilizes input control cards to the DDname //LGRSYSIN to implement logging for Assembler programs. There is no default. If you desire logging, then you are required to specify parameters in the //LGRSYSIN DD to indicate the type of processing and output to be directed to the //LGRECOUT DD.

If you have a large program with many log points defined or have log points defined within a program loop, it may result in a large amount of log data being written to the //LGRECOUT DD. You may use input control cards to filter which log events are executed within your program, and thereby control the amount of log data written to the //LGRECOUT output data set. You do this filtering by putting control statements in the //LGRSYSIN DD.

5.1 Operands

An operand field is used to supply the required information for processing the logger in the form of parameters separated by commas. The name of the operand field is LOGEVENT. The operand field has no fixed column requirements, but it must be followed by at least one blank and no continuation cards are allowed.

Comments are optional. Comments may be added by placing an asterisk (*) in column 1. Comments may also be placed after the closing parenthesis of the last operand if separated by at least one blank.

5.2 Keywords

There are three positional keyword parameters. The keywords are “PROGRM,NAME=”, “SUBRTN,NAME=”, and “LOGPNT,NAME=”. Keyword parameters specify the name of the RSECT name or a program label identifying where log processing is to occur.

The "PROGRM" operand indicates that program entry and program exit is to be logged for the program names listed. The program names will actually be the "RSECT" names.

The "SUBRTN" operand indicates subroutine entry and subroutine exit are to be logged for any subroutine names listed.

The "LOGPNT" operand indicates to produce log records for any log points defined in the program.

For example:

```
//LGRSYSIN  DD *
      LOGEVENT PROGRM,NAME=(*)
      LOGEVENT SUBRTN,NAME=(*)
      LOGEVENT LOGPNT,NAME=(*)
//*
```

5.3 Filtering Log Records

Keyword parameters represent the program name or control section name or logpoint name fields. The specification of a name may contain wildcards in the form of the question mark "?" or in the form of the asterisk"*" to allow for the filtering of log output. Patterns can be developed using these wildcard characters "?" or "*" to direct log output.

The special wildcard character "?" is used to represent any value in a single position of the name. The wildcard character "?" can be used multiple times within a designated name to indicate any value in exactly that number of positions.

The wildcard character "*" represents one or more positions in the name instead of the exact character position which the "?" references. Wildcard characters need not be restricted to the end of the name. They can also be specified at the beginning or middle of the name.

The //LGRSYSIN DD is placed in the job step JCL executing your Assembler program.

For example:

```
//STEP      EXEC PGM=USERPGM1
//STEPLIB   DD DISP=SHR,DSN=USER.WORK.LINKLIB
//SYSPRINT  DD SYSOUT=*
//LGRECOUT  DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=133)
//LGRSYSIN  DD *
      LOGEVENT PROGRM,NAME=(*)
      LOGEVENT SUBRTN,NAME=(*)
      LOGEVENT LOGPNT,NAME=(*)
//*
```

Example 1:

```
//LGRSYSIN  DD *
      LOGEVENT LOGPNT,NAME=(USER0?50)
```

The result of the above input statement is to direct LOGGRASM services to process only those log points in the user's program defined at labels USER0250 and USER0350. Only USER0250 and USER0350 will have log information displayed in the //LGRECOUT DD.

```
USER0100 DS    0H
         -
USER0200 #LGPOINT   ==> Logpoint is disabled and not executed
         -
USER0250 #LGPOINT   ==> Logpoint is enabled and executed
         -
USER0300 #LGPOINT   ==> Logpoint is disabled and not executed
         -
USER0350 #LGPOINT   ==> Logpoint is enabled and executed
         -
USER0400 #LGPOINT   ==> Logpoint is disabled and not executed
```

Example 2:

```
//LGRSYSIN  DD *
   LOGEVENT PROGRM,NAME=(PGM?0*)        A comment can also go here
   LOGEVENT SUBRTN,NAME=(SUBS?000)      A comment can also go here
   LOGEVENT LOGPNT,NAME=(LABLX*)        A comment can also go here
   LOGEVENT LOGPNT,NAME=(LABLZ*,READ*)  A comment can also go here
```

The result of the previous input statements is to direct LOGGRASM services to process program entry and exit for any RSECT name of PGMA0100, PGMA0110, PGMB0200, and PGMC0300. Names like PGMA1000 or PGMB2000 or similar would not be processed since the 5th position of "0" would not match according to the pattern. Subroutine entry and subroutine exit is processed for names like SUBS1000, or SUBS2000, or SUBS3000, and so on. Names like SUBT1000 or SUBS1200 are filtered out if defined in the program, and are not processed. Any log points defined with a name prefixed with LABLX, LABLZ, or READ are processed, and any other label names for log points defined in the program are filtered out.

Example 3:

```
//LGRSYSIN  DD *
*      Asterisk in column one is a comment line
   LOGEVENT PROGRM,NAME=(*)
   LOGEVENT SUBRTN,NAME=(*)
   LOGEVENT LOGPNT,NAME=(*)
```

The result of the above input statements is to direct LOGGRASM services to print log output for every program entry, every program exit, every subroutine entry, every subroutine exit, and every log point defined anywhere in the user's Assembler program.

If you do not include a //LGRSYSIN DD or if you comment out the //LGRSYSIN DD in your JCL, by design it will disable logging and no output will be produced even if logging is enabled in your program (e.g., LOG=ON). The log points remain in your program, but upon entry to any log points they will be effectively NO-OP'ed by the execute of an Execute instruction where the target of the Execute is the BCR instruction with R0 specified as the second operand register (e.g., x'0700'), and there will be no overhead from logging.

This is a quick and easy method to run an Assembler program defined with many log points where you do not want the log points to be executed without having to go back, change your code, and re-assemble your programs with LOG=OFF. Also, commenting out the //LGRECOUT DD has the same effect as commenting out //LGRSYSIN. Commenting out either DD disables logging. This means you can run a job with logging, then comment out the DD, and run again with no logging.

6.0 Output Data Set for Log Records: //LGRECOUT

DDname //LGRECOUT represents the report output data set where all informational detail lines are written by LOGGRASM to describe your Assembler program execution.

The //LGRECOUT DD is placed in the job step executing the Assembler program.

For example:

```
      //STEP      EXEC PGM=USERPGM1
      //STEPLIB   DD DISP=SHR,DSN=USER.WORK.LINKLIB
      //SYSPRINT  DD SYSOUT=*
      //LGRECOUT  DD SYSOUT=*,DCB=(LRECL=133,BLKSIZE=133)
      //LGRSYSIN  DD *
            LOGEVENT PROGRM,NAME=(*)
            LOGEVENT SUBRTN,NAME=(*)
            LOGEVENT LOGPNT,NAME=(*)
         //*
```

The output report may be written to a data set or directed to SYSOUT as shown in the following examples:

For example: Sysout

```
//LGRECOUT  DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=133)
```

For example: New Allocation

```
      //LGRECOUT  DD DISP=(NEW,CATLG,CATLG),
      //          DSN=USER.WORK.LOGGER.REPORT,
      //          UNIT=SYSDA,SPACE=(CYL,(10,10)),
      //          DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6118)
```

For example: Existing Allocation:

```
      //LGRECOUT  DD DISP=OLD,DSN=USER.WORK.LOGGER.REPORT
```

After a job run, you may want to transfer the //LGRECOUT output from the mainframe to your workstation for review on the PC since the output has a wide view (e.g., LRECL 133). Otherwise in an ISPF session use horizontal scrolling (PF Key 10, PF Key 11) where needed to view the output beyond column 80.

Ensure to allocate sufficient space for the anticipated volume of log records which may be generated. In the event that all available space for the data set is exhausted, this will lead to a possible x37 abend condition.

```
JOB01234  IEC030I B37-04,IFG0554A,USERPGM2,USERPGM2,LGRECOUT,
          3511,VOL003,USER.WORK.LOGGER.REPORT
```

LOGGRASM has incorporated a DCB Abend Exit for the //LGRECOUT DCB to recover and allow the user program execution to continue for the following conditions.

B37-004	**VTOC, Used 16 Extents, or Volume is Out of Space.**
D37-004	**Primary Allocation Used, No Secondary Allocation Requested**
E37-004	**Used All Available Space**
913-0xx	**Open Error**

However, upon recovery no further output log records are produced even though your user program will continue executing. Before running your job again where there was an x37 abend on the //LGRECOUT DD, you should delete the data set and reallocate it increasing the space allocation (i.e., SPACE=) in the JCL for the //LGRECOUT data set. Also ensure the //LGRECOUT data set resides on a volume with sufficient free space to handle any secondary allocations and prevent further x37 conditions.

For example:

```
//LGRECOUT  DD DISP=(NEW,CATLG,CATLG),
//             DSN=USER.WORK.LOGGER.REPORT,
//             UNIT=SYSDA,SPACE=(CYL,(50,20)), ←——————Increase space
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=24472)
```

If you do not include a //LGRECOUT DD or if you comment out the //LGRECOUT DD in your JCL, by design it will disable logging and no output will be produced even if logging is enabled in your program (e.g., LOG=ON). The log points remain in your program, but upon entry to any log points they will be effectively NO-OP'ed, and there will be no overhead from logging. This is a quick and easy method to run an Assembler program defined with many log points where you do not want the log points to be executed. This removes the need of constantly having to go back, change your code, and re-assemble your programs with variations of LOG=ON/OFF. Also, commenting out the //LGRSYSIN DD has the same effect as commenting out //LGRECOUT. Either one disables logging. This means you can run a job with logging, then comment out the DD, and run again with no logging.

7.0 Description of Sample Assembler Program Using LOGGRASM

```
   ************************************************************************
   *                                                                      *
   * Program Name:    TEST$PGM                                            *
   *                                                                      *
   * Function:        Sample Tutorial Program                             *
   *                                                                      *
   ************************************************************************
   *
   *
#1          COPY  LGCPLOGR            =>Include LOGGRASM CopyBook First<=
   *
#2 TEST$PGM LPGMNTRY TYPE=MAIN,       Define Main Program Entry            X
#3                BASE=(12,11),       Establish Base Registers             X
#4                AMOD=31,RMOD=24,    Addressing and Residency             X
#5                LCAPFX=XYZ          Name the LCA Prefix-RENT Dsect Area
   *
#6          SR    R15,R15             Some code
   *
#7          LPGMEXIT RC=(R15)         Define Program Exit
   *
   ************************************************************************
   * The LCA is your RENT DSECT Area.                                     *
   ************************************************************************
   *
#8          LCA   DEFINE              Define Logger Control Area-(LCA)
   *
#9 WORKAREA DS    20F                 A Work Area
   *
#10         LCA   DEFEND              Define End of Logger Control Area
   *
#11         END
```

Line #1:

This is the LGCPLOGR copybook that defines the Logger services. The COPY statement at assembly time will cause the Assembler to copy the LGCPLOGR code into your source program where the source library for LGCPLOGR will be from the //SYSLIB DD. This means the source library from which the LGCPLOGR copybook is to be copied must be available to the Assembler at assembly time. The LGCPLOGR will be a member in a PDS which is a part of the //SYSLIB DD concatenation in the JCL used to assemble your program.

Line #2:

This is the LPGMNTRY that performs program entry services. Your program knows about the LPGMNTRY macro because it is defined in the LGCPLOGR code that was copied in at Line #1. LPGMNTRY is defined as TYPE=MAIN indicating TEST$PGM is a main program RSECT. LPGMNTRY will automatically setup your program to be re-entrant, assign the base registers, save the registers of the calling program, obtain dynamic storage for a save area, set up R13 to contain the address of this save area, set the forward chain and back chain pointers for the save area addresses, set the RSECT statement, establish the TEST$PGM label for the RSECT, and set up the Logger Services environment by calling the LGMHLRCI program. If previously you have been using an 'ENTRY' type macro to generate your entry code, then LPGMNTRY replaces that macro.

Line #3:

This is the BASE= parameter to LPGMNTRY where you declare your base registers to your program. In the example you have:

BASE=(12,11)

There is a requirement in LOGGRASM that register 12 must be the first base register. After that, you can specify any base register you like (except R13). As a sidebar, the convention is that when there are multiple base registers the assembler chooses the base register that gives the smallest displacement, and if there are two or more base registers that give the same (smallest) displacement the assembler chooses the highest numbered register.

Line #4:

This is an example of specifying the AMOD=31 and RMOD=24 parameters to LPGMNTRY. The default is AMOD=31 and RMODE=31 if you do not specify this parameter. LPGMNTRY will take these values as symbolic variables and automatically generate the AMODE instruction in your program to specify the addressing mode to be associated with the TEST$PGM RSECT, and LPGMNTRY will automatically generate the RMODE instruction in your program to specify the residence mode to be associated with the TEST$PGM RSECT. In addition, LPGMNTRY does not use the CSECT but the RSECT assembler directive in order to further enforce the re-entrant status for TEST$PGM.

LOGGRASM performs certain setup actions based on these parameters. It is recommended that you explicitly specify the MODE statement in your link-edit, and that the AMODE and RMODE attributes in your link-edit be consistent with the AMOD= and RMOD= attributes specified in the LPGMNTRY entry of your main program.

Line #5:

This is the LCAPFX= parameter to LPGMNTRY. This establishes the prefix name to be used by LPGMNTRY to automatically generate the fully qualified label to define your TEST$PGM program save area DSECT, and to define the prefix to other field names defined in the prefix section of the LCA (Logger Control Area).

Line #6:

This is a sample 'SR R15,R15' instruction. All of you own assembler coding would go here between the LPGMNTRY and the LPGMEXIT.

Line #7:

This is the LPGMEXIT that performs program exit services. The LPGMEXIT entry requires a preceding LPGMNTRY entry. It will restore the registers from the save area, process the return code, define a log point, and call the Logger Services termination program LGMHLRCT to print logger messages, release logger resources if required, close Logger data sets, branch back to the calling program, and tour user program will exit. If previously you have been using a 'RETURN' type macro to generate exit code, LPGMEXIT replaces that macro.

Line #8:

This is the "LCA DEFINE" entry. This defines the Logger Control Area (LCA). The LCA is a dynamic storage area automatically acquired by LPGMNTRY, its structure is automatically built by LPGMNTRY, it is automatically maintained by Logger Services, and automatically established with addressability to R13. This LCA is your RENT DSECT area which will be automatically get-main'ed by Logger Services, and contains the TEST$PGM program save area.

The naming convention for this RENT DSECT area in your program TEST$PGM would be as follows:

```
LCAXYZ    DSECT ,        Start of Logger Control Area (LCA)
XYZSAVE   DS   9D        Program Save Area==> R13 Points Here
```

Why is it called 'LCAXYZ' and 'XYZSAVE'? Go back to where the LPGMNTRY was specified in the TEST$PGM program example (Line#5), and you will see the entry LCAPFX=XYZ. LPGMNTRY generated the names automatically, and used the prefix name from LCAPFX to generate the fully qualified label names in the construction of the save area entries. You could have said LCAPFX=ABC in which case the names would have been:

```
LCAABC    DSECT ,        Start of Logger Control Area (LCA)
ABCSAVE   DS   9D        Program Save Area==> R13 Points Here
```

You don't see this because the save area and certain other fields are basically hidden from the user in a prefix area of the LCA. If you want to see the save area label and fields in the LCA prefix area, take the sample TEST$PGM program, remove the #n line numbers and clean it up so it will assemble, and then assemble using 'PRINT GEN'.

For example:

```
COPY  LGCPLOGR          =>Include LOGGRASM CopyBook First<=
PRINT GEN
```

Then on the assembler listing do a find on 'LCAXYZ' to view the structure of the prefix area of the LCA. Then remove the PRINT GEN to assemble later without all the expansions.

However, for all purposes, you would look at everything between the LCA DEFINE and the LCA DEFEND in TEST$PGM as a clean sheet of paper. Basically all you have to do is code the LCA DEFINE macro to establish the LCA, code your variables, and then set the end of the area with the LCA DEFEND.

The LCA is a clean get-main'ed area to maintain complete re-entrant status for your program. It is a below-the-line area where you can place remote parameter lists to support any MF=E generations of macros that are required to be below-the-line. You can use this as a work area to define any fields and areas you need to use in your program up to the 4K addressability of R13 minus the LCA prefix area size. If you need more addressability, then you can manually perform your own Get-Main (or Storage Obtain) and manually bind to a different register, or you could just use the LCAXTRA= parameter in LPGMNTRY where it will do that for you and maintain re-entrant status for your program.

Line #9:

This contains the entry:

```
WORKAREA DS    20F            A Work Area
```

This entry just defines a work area in the LCA which in itself is a RENT DSECT area. All your registers have already been saved in the LCA prefix area upon TEST$PGM program entry. The name WORKAREA was made up. It could have been any name or not defined at all. Everything between LCA DEFINE and LCA END is a 3½K reentrant work area for your user program where you can define any variables of your choosing.

As another program example:

```
*+-------------------------------------------------------------------+*
*|    Program RENT Work Area                                         |*
*+-------------------------------------------------------------------+*
*
         LCA   DEFINE               Define the Program RENT Work Area
*
WORKAREA DS    10D                  A Work Area
DYN$PLST DS    20FD                 CALL Parameter List
DYNAPRMS DS    FD                   Save Area for Dynamic Allocate Parms
DYNAMSG@ DS    FD                   Save Area for Dynamic Alloc Msg Area
DYNAMSGL DS    FD                   Save Area for Len of Msg Buffer Area
DYNCOUNT DS    FD                   Save Counter for Text Unit Entries
PRMCOUNT DS    FD                   Save Count for Output Txt Unit Parms
PRMINCNT DS    FD                   Save Count for Input Text Unit Parms
DYNDSNAM DS    AD                   Pointer to Data Set Name Text Unit
DYNSTATS DS    AD                   Pointer to Data Set Status Text Unit
MSGBUFFR DS    AD                   Save Area for SVC99 Message Buffer
DYNRETCD DS    F                    Save Area for SVC99 Return Code
DYNRSNCD DS    F                    Save Area for SVC99 Reason Code
DYNRTCDE DS    F                    Save Area for SVC99 Return Code
REQBKPNT DS    A                    Save Area for SVC 99 Request Blk Pnt
         DS    0F                   SVC99 Request Blk FullWord Alignment
REQBLOCK DS    XL(S99RBEND-S99RB)   SVC 99 Dynamic Alloc Request Block
         DS    0F                   SVC99 Request Blk Extension FW Align
REQBLKEX DS    XL(S99ERSN+4-S99RBX)  ...SVC 99 Request Block Extension
TEXTPNTR DS    24F                  List of Pointers to SVC99 Text Units
TEXTUNIT DS    CL512                Work Area for the SVC99 Text Units
EMPRLST@ DS    A                    IEFDB476 Parameter List Address
EMPRMLST DS    0F,CL(EMLEN1)        IEFDB476 Parameter List Area
```

```
CATPLIST DS    5F                  Catalog Locate Parameter List Area
LOC$LDSN DS    CL44                Locate Work Area for Data Set Name
LOCLAREA DS    0D                  Locate Requires Doubleword Boundary
         DS    CL265               Locate Work Area for Vol List
*
         LCA   DEFEND              Define End of Program RENT Work Area
```

Line #10:

This is the "LCA DEFEND" entry. This defines the end of the Logger Control Area (LCA). When LPGMNTRY sees the 'LCA DEFEND', it automatically will calculate the address for the next available area in the LCA, determine the area remaining in the LCA, calculate the length for the current area in the LCA, calculate the residual storage amount, calculate an end pointer to be used to determine if an overlay has occurred in the LCA, process the prefix name, build the LCA prefix structure, and build the save area entries.

Line #11:

The END statement which is an Assembler directive which specifies the end of the source program code for the TEST$PGM program.

8.0 Description of Output Report

The following is an example of the output which is printed as a result of using LOGGRASM.

8.1 Output Report Description

The output report begins with the printing of the general header, card ruler header, the input card images from //LGRSYSIN, and the environmental report.

The "**Pgm)==>**" prefix header indicates that the detail lines printed are for an entry point into a main program USERPGM2. The detail line output was due to the coding of the LPGMNTRY entry in the open source with the TYPE=MAIN parameter. The "**Msg)**" prefix indicates that this is a message associated with the LPGMNTRY entry. The plus sign "**+**" prefix indicates that this is the data printed with the preceding LPGMNTRY entry. By default the general purposes registers are also displayed.

For example:

```
         COPY  LGCPLOGR         Include LGCPLOGR CopyBook First
USERPGM2 LPGMNTRY TYPE=MAIN,       Define Main Program Entry (Required)X
               BASE=(12),          Establish Base Register (Default)   X
               AMOD=31,RMOD=31,    Addressing and Residency (Default)  X
               STORAGE=8,          Define LCA Storage Size (Default)   X
               LOG=ON,             Turn On Logging (Default)           X
               LOGOUT=PRNT,        Print Records When Gen'ed (Default) X
               LCAPFX=USR,         Name the LCA Prefix (Required)      X
               MSG='Main User Program 2 Example' Set Any Msg (Optional)
```

```
Pgm)==>Program Entry is To:  USERPGM2.USERPGM2+(000000)
Pgm)==>Program Entered From: z/OS Initiator
Pgm)==>Program Attributes: LMod Start=18100EF8  LMod Entry Point=98100EF8  LMod Size=0000D108
                           LMod Amode=31  LMod Subpool=251  Auth Lib=No    AC=0   Rent/Reus
Msg) Main User Program 2 Example
+    General Purpose Registers On Entry:
+     R0 = 00000000FD000008  R1 = 0000000000006FF8  R2 = 0000000000000040  R3 = 00000000008CBD84
+     R4 = 00000000008CBD60  R5 = 00000000008FF290  R6 = 00000000008B1FE0  R7 = 00000000FD000000
+     R8 = 00000000008FC030  R9 = 00000000008DCCD0  R10= 0000000000000000  R11= 00000000008FF290
+     R12= 0000000083023B12  R13= 0000000000006F60  R14= 0000000080FDE850  R15= 0000000098100EF8
```

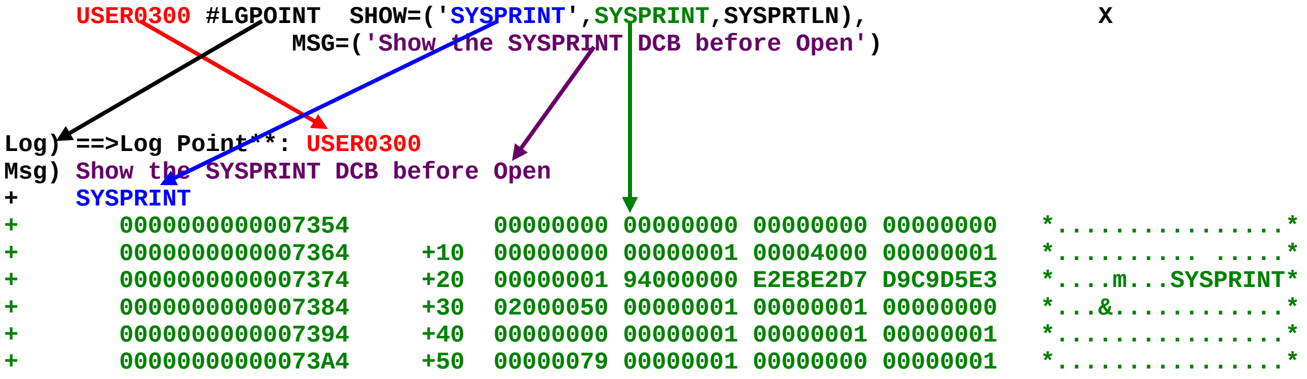
The “**Log) ==>Log Point**:**” prefix header indicates that the detail lines printed are for a log point. The output detail lines shown below are due to the coding of the #LGPOINT entry in the open source of an Assembler program at the label area USER0300 DS 0H. The label where the log point was inserted is displayed.

The SHOW= parameter was specified to request a display of storage. The text SYSPRINT is displayed to identify the data area being printed followed by the data area. The beginning address of each 16 byte data line is display, followed by the offset from the beginning of the data area, the hex representation of the data, and then following by an EBCDIC translation of each byte of storage data to character format with periods "." substituted for those bytes which do not translate to valid printable characters.

For example:

```
USER0300 DS 0H
```

is replaced with:



```
USER0300 #LGPOINT  SHOW=('SYSPRINT',SYSPRINT,SYSPRTLN),                 X
                   MSG=('Show the SYSPRINT DCB before Open')


Log) ==>Log Point**: USER0300
Msg) Show the SYSPRINT DCB before Open
+    SYSPRINT
+       0000000000007354         00000000 00000000 00000000 00000000   *................*
+       0000000000007364     +10 00000000 00000001 00004000 00000001   *.......... .....*
+       0000000000007374     +20 00000001 94000000 E2E8E2D7 D9C9D5E3   *....m...SYSPRINT*
+       0000000000007384     +30 02000050 00000001 00000001 00000000   *...&............*
+       0000000000007394     +40 00000000 00000001 00000001 00000001   *................*
+       00000000000073A4     +50 00000079 00000001 00000000 00000001   *................*
```

In the next example the “**Pgm)==>**” prefix header indicates that the detail lines printed are for an entry point into a called program ABCD0000. The detail line output was due to the coding of the LPGMNTRY entry in the open source with the TYPE=SUB parameter. The “**Msg)**” prefix indicates that this is a message associated with the LPGMNTRY entry. The plus sign “**+**” prefix indicates that this is the data printed with the preceding LPGMNTRY entry. The name of the called program and calling program are displayed for program entry, and the name of the calling program and the offset in the program where the call was made are also displayed. By default the general purposes registers are displayed.

For example:

```
ABCD0000 LPGMNTRY TYPE=SUB,         Define Sub-Program Entry (Required) X
               BASE=(12,11),        Establish Multiple Base Registers   X
               LCAPFX=ABC,          Name the LCA Prefix (Required)      X
               MSG='Called Program 1 Example' ...Message is Optional
```

```
Pgm)==>Program Entry is To:  USERPGM2.ABCD0000+(000000)
Pgm)==>Program Entry Status: CPU State=Problem Pgm  PSW Key=8  Amode=31
Pgm)==>Program Entered From: USERPGM2.USERPGM2+(00042A)
Msg) Called Program 1 Example
+    General Purpose Registers On Entry:
+    R0 = 0000000034800CD8  R1 = 00000000000071F0  R2 = 0000000000000000  R3 = 0000000000000000
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 0000000034801DE0
+    R12= 0000000034800DE0  R13= 00000000000073B8  R14= 00000000B4800A0C  R15= 0000000034800DD0
+    Access Registers:
+    AR0 = B45C548A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000
```

In the following example the “**Sub) ==>**” prefix header indicates that the detail lines printed are for an entry into a subroutine within a Csect of the program. The “**Sub) <==**” prefix header indicates that the detail line printed is for an exit from a subroutine. The detail line output was due to the coding of the LPGMSUBE and LPGSUBX entries in the open source

```
ABCD0400 DS    0H
         BAS   R14,ASUB0000        Branch to Subroutine

*
ASUB0000 DS    0H                  Subroutine Entry
         SLR   R0,R0               * Some Code *
ASUBEXIT BR    R14                 Subroutine Exit
```

```
ABCD0400 #LGPOINT
         BAS   R14,ASUB0000        Branch to Subroutine

*
ASUB0000 LPGMSUBE                  Define Subroutine Entry
         SLR   R0,R0               * Some Code *
ASUBEXIT LPGMSUBX                  Define Subroutine Exit
```

The output will show:

```
Log) ==>Log Point**: ABCD0400
Sub)   ==>Subroutine: USERPGM2.ABCD0000.ASUB0000
          From: USERPGM2.ABCD0000+(05AC)
Sub)   <==Subroutine Exit: USERPGM2.ABCD0000.ASUB0000
```

8.2 Sample Output Report

The following is an example of the output written by Logger Services to the //LGRECOUT DD for the sample program USERPGM3.

```
               */************************************************/
               */      LOGGRASM Service Information Report      */
               */      -----------------------------------      */
               */************************************************/


                             //LGRSYSIN Input Cards
                             ----------------------
....+....10...+....20...+....30...+....40...+....50...+....60...+....70...+....80

      LOGEVENT PROGRM,NAME=(*)
      LOGEVENT SUBRTN,NAME=(*)
      LOGEVENT LOGPNT,NAME=(*)

....+....10...+....20...+....30...+....40...+....50...+....60...+....70...+....80

Program execution started on Saturday 01/21/2012 at 12:04:07.705600

  System Name.................: SYS4A
  Sysplex Name................: PLEX04
  Operating System............: z/OS 01.13.00 FMID(HBB7780) SP7.1.3
  System was last IPL'ed on...: Tuesday 01/10/2012 at 17:35:53.896
  Local Time Offset from UTC..: -8 Hours 00 Minutes (GMT West)
  Data Facility Product.......: z/OS DFSMS V01R13M00
  Job Entry Subsystem.........: JES2 Version(z/OS1.13) FMID(HJE7780) Lvl(42.00)
  CPU Model...................: 2084 Type(C24)  Processors=10  MIPS=3,505.81  MSU=>CEC:1076 IMG:448
  Job Type....................: Batch
  Job Name....................: USERJOB3
  Job Number..................: J0060879
  Address Space ID (ASID).....: 0081
  Executing Program Name......: USERPGM3
  Executing Job Step Name.....: RUN
  Memory Limit (MEMLIMIT).....: 8 Gigabytes
  MEMLIMIT Source.............: MEMLIMIT source was set from JCL
  Region Size.................: 4M
  Below 16M Area Available....: 4,194,304
  Below 16M Maximum Allowed...: 8,364,032
  Below 16M Area Being Used...: 503,808
  Above 16M Area Available....: 134,217,728
  Above 16M Maximum Allowed...: 1,276,116,992
  Above 16M Area Being Used...: 311,296
  User Id.....................: USER001

Pgm)==>Program Entry is To:  USERPGM3.USERPGM3+(000000)                                             007BBAC8 2012021-12:04:07.705600
Pgm)==>Program Entered From: z/OS Initiator                                                         007BBAC8 2012021-12:04:07.705600
Pgm)==>Program Attributes: LMod Start=33F007B0  LMod Entry Point=33F007B1  LMod Size=00042850       007BBAC8 2012021-12:04:07.705600
                           LMod Amode=64  LMod Subpool=251  Auth Lib=No    AC=0   Rent/Reus         007BBAC8 2012021-12:04:07.705600
                           CPU State=Problem Pgm   PSW Key=8                                        007BBAC8 2012021-12:04:07.705600
Msg) Main User Program 3 Example                                                                    007BBAC8 2012021-12:04:07.705600
+    General Purpose Registers On Entry:                                                            007BBAC8 2012021-12:04:07.705600
+    R0 = 00000000FD000008  R1 = 0000000000006FF8  R2 = 0000000000000040  R3 = 00000000007DBD6C     007BBAC8 2012021-12:04:07.705600
+    R4 = 00000000007DBD48  R5 = 00000000007FF358  R6 = 00000000007BEFC8  R7 = 00000000FD000000     007BBAC8 2012021-12:04:07.705600
+    R8 = 00000000007D8918  R9 = 00000000007FF520  R10= 0000000000000000  R11= 00000000007FF358     007BBAC8 2012021-12:04:07.705600
+    R12= 0000000085EEC022  R13= 0000000000006F60  R14= 0000000000FDBC98  R15= 00000000FFFFF002     007BBAC8 2012021-12:04:07.705600
+    Access Registers:                                                                              007BBAC8 2012021-12:04:07.705600
+    AR0 = 00000000         AR1 = 33F04E98         AR2 = 00000000         AR3 = 00000000            007BBAC8 2012021-12:04:07.705600
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000            007BBAC8 2012021-12:04:07.705600
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000            007BBAC8 2012021-12:04:07.705600
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000            007BBAC8 2012021-12:04:07.705600
Log) ==>Log Point**: USER0300                                                                       007BBAC8 2012021-12:04:07.705600
+    Private Area Storage Allocated:                                                                007BBAC8 2012021-12:04:07.705600
+      <16M: In Use=576K       Unused=3583K       Limit=4160K       HighU=576K                      007BBAC8 2012021-12:04:07.705600
+      >16M: In Use=457K       Unused=130614K     Limit=131072K     HighU=457K                      007BBAC8 2012021-12:04:07.705600
+      >BAR: Allocated=258 Megabytes                 Guard Amount=255 Megabytes                     007BBAC8 2012021-12:04:07.705600
+      >BAR: Pvt Used =3 Megabytes                   Pvt Hi-Water=3 Megabytes                       007BBAC8 2012021-12:04:07.705600
+      >BAR: Pvt Obj  =3                             Shr Objects =0                                 007BBAC8 2012021-12:04:07.705600
+      >BAR: Shr Alloc=0 Megabytes                   Shr Hi-Water=0 Megabytes                       007BBAC8 2012021-12:04:07.705600
+      >BAR: Com Alloc=0 Megabytes                   Com Hi-Water=0 Megabytes                       007BBAC8 2012021-12:04:07.705600
+      >BAR: Com Obj  =0                             Large Pages =0                                 007BBAC8 2012021-12:04:07.705600
+    Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE                   007BBAC8 2012021-12:04:07.705600
+     SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space          007BBAC8 2012021-12:04:07.705600
+             Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line        007BBAC8 2012021-12:04:07.705600
+     003-8   3           0           466,944     0           32,768          0                     007BBAC8 2012021-12:04:07.705600
+     229-0   1           0           0           0           4,096           0                     007BBAC8 2012021-12:04:07.705600
+     230-5   2           4           8,192       7,968       0               0                     007BBAC8 2012021-12:04:07.705600
+     251-8   1           1           0           0           274,432         1,968                 007BBAC8 2012021-12:04:07.705600
+             ----------  ----------  ----------  ----------  --------------  --------------        007BBAC8 2012021-12:04:07.705600
+     Totals  7           5           475,136     7,968       311,296         1,968                 007BBAC8 2012021-12:04:07.705600
+    Job Step CPU: 0HR 0MIN 0.00196SEC       Job Step SRB: 0HR 0MIN 0.00015SEC                      007BBAC8 2012021-12:04:07.705600
+    Show IARV64 AREA                                                                               007BBAC8 2012021-12:04:07.705600
+       0000004800000000          00010203 04050607 08090A0B 0C0D0E0F   *................*          007BBAC8 2012021-12:04:07.705600
+       0000004800000010     +10  10111213 14151617 18191A1B 1C1D1E1F   *................*          007BBAC8 2012021-12:04:07.705600
+       0000004800000020     +20  20212223 24252627 28292A2B 2C2D2E2F   *................*          007BBAC8 2012021-12:04:07.705600
+       0000004800000030     +30  30313233 34353637 38393A3B 3C3D3E3F   *................*          007BBAC8 2012021-12:04:07.705600
+       0000004800000040     +40  40414243 44454647 48494A4B 4C4D4E4F   * ...........<(+|*          007BBAC8 2012021-12:04:07.705600
+       0000004800000050     +50  50515253 54555657 58595A5B 5C5D5E5F   *&.........!$*)..*          007BBAC8 2012021-12:04:07.705600
+       0000004800000060     +60  60616263 64656667 68696A6B 6C6D6E6F   *-/.........,%_>?*          007BBAC8 2012021-12:04:07.705600
+       0000004800000070     +70  70717273 74757677 78797A7B 7C7D7E7F   *..........:#@'=.*          007BBAC8 2012021-12:04:07.705600
+       0000004800000080     +80  80818283 84858687 88898A8B 8C8D8E8F   *.abcdefghi......*          007BBAC8 2012021-12:04:07.705600
+       0000004800000090     +90  90919293 94959697 98999A9B 9C9D9E9F   *.jklmnopqr......*          007BBAC8 2012021-12:04:07.705600
+       00000048000000A0     +A0  A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF   *..stuvwxyz......*          007BBAC8 2012021-12:04:07.705600
+       00000048000000B0     +B0  B0B1B2B3 B4B5B6B7 B8B9BABB BCBDBEBF   *................*          007BBAC8 2012021-12:04:07.705600
+       00000048000000C0     +C0  C0C1C2C3 C4C5C6C7 C8C9CACB CCCDCECF   *.ABCDEFGHI......*          007BBAC8 2012021-12:04:07.705600
+       00000048000000D0     +D0  D0D1D2D3 D4D5D6D7 D8D9DADB DCDDDEDF   *.JKLMNOPQR......*          007BBAC8 2012021-12:04:07.705600
+       00000048000000E0     +E0  E0E1E2E3 E4E5E6E7 E8E9EAEB ECEDEEEF   *..STUVWXYZ......*          007BBAC8 2012021-12:04:07.705600
+       00000048000000F0     +F0  F0F1F2F3 F4F5F6F7 F8F9FAFB FCFDFEFF   *0123456789......*          007BBAC8 2012021-12:04:07.705600
+       0000004800000100    +100  00000000 00000000 00000000 00000000   *................*          007BBAC8 2012021-12:04:07.705600
+       0000004800000110    +110  0000004800000110 TO 0000004800007FEF SAME AS ABOVE                007BBAC8 2012021-12:04:07.705600
+       0000004800007FF0   +7FF0  00000000 00000000                     *........        *          007BBAC8 2012021-12:04:07.705600
Pgm)<==Program Exited From: USERPGM3.USERPGM3+(00046C)                                              007BBAC8 2012021-12:04:07.705600
Pgm)<==Program Exit Status: CPU State=Problem Pgm  PSW Key=8  Amode=64                              007BBAC8 2012021-12:04:07.705600
Pgm)<==Program Exited To:   z/OS Initiator                                                          007BBAC8 2012021-12:04:07.705600
+            Return Code =  00000000   Reason Code =  00000000                                      007BBAC8 2012021-12:04:07.705600
+    General Purpose Registers On Exit:                                                             007BBAC8 2012021-12:04:07.705600
+    R0 = 0000000000000000  R1 = 0000000000007200  R2 = 0000000000006F60  R3 = 0000000000000000     007BBAC8 2012021-12:04:07.705600
+    R4 = 0000004800000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000     007BBAC8 2012021-12:04:07.705600
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 0000000000000000     007BBAC8 2012021-12:04:07.705600
+    R12= 0000000033F007BC  R13= 0000000000007000  R14= 000000000000090E  R15= 0000000000000000     007BBAC8 2012021-12:04:07.705600
+    Access Registers:                                                                              007BBAC8 2012021-12:04:07.705600
+    AR0 = 00000000         AR1 = 33F04E98         AR2 = 00000000         AR3 = 00000000            007BBAC8 2012021-12:04:07.705600
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000            007BBAC8 2012021-12:04:07.705600
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000            007BBAC8 2012021-12:04:07.705600
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000            007BBAC8 2012021-12:04:07.705600

LGA01110I Assembler program logging has completed.
  Job Step CPU: 0HR 0MIN 0.00237SEC       Job Step SRB: 0HR 0MIN 0.00020SEC
```

8.3 Sample Abend Output Report

The following is an example of the output written by Logger Services to the //LGRECOUT DD for the sample program USERPGM2. The program USERPGM2 is designed to intentionally abend in order to show an example of the output from the diagnostic abend report created by LOGGRASM.

```
               */***********************************************/
               */      LOGGRASM Service Information Report      */
               */      -----------------------------------      */
               */***********************************************/


                             //LGRSYSIN Input Cards
                             ----------------------
....+....10...+....20...+....30...+....40...+....50...+....60...+....70...+....80

    LOGEVENT PROGRM,NAME=(*)
    LOGEVENT SUBRTN,NAME=(*)
    LOGEVENT LOGPNT,NAME=(*)

....+....10...+....20...+....30...+....40...+....50...+....60...+....70...+....80

Program execution started on Saturday 01/21/2012 at 11:08:17.898496

  System Name.................: SYS06
  Sysplex Name................: PLEX06
  Operating System............: z/OS 01.11.00 FMID(HBB7760) SP7.1.1
  System was last IPL'ed on...: Monday 11/21/2011 at 10:59:40.745
  Local Time Offset from UTC..: -8 Hours 00 Minutes (GMT West)
  Data Facility Product.......: z/OS DFSMS V01R11M00
  Job Entry Subsystem.........: JES2 Version(z/OS1.11) FMID(HJE7760) Lvl(40.05)
  CPU Model...................: 2097 Type(E64)  Processors=4  MIPS=3,436.42  MSU=>CEC:3739 IMG:234
  Job Type....................: Batch
  Job Name....................: USERJOB2
  Job Number..................: JOB02051
  Address Space ID (ASID).....: 0031
  Executing Program Name......: USERPGM2
  Executing Job Step Name.....: RUN
  Memory Limit (MEMLIMIT).....: 2 Gigabytes
  MEMLIMIT Source.............: MEMLIMIT source was set from SMF
  Region Size.................: 4M
  Below 16M Area Available....: 4,194,304
  Below 16M Maximum Allowed...: 6,266,880
  Below 16M Area Being Used...: 503,808
  Above 16M Area Available....: 33,554,432
  Above 16M Maximum Allowed...: 1,301,282,816
  Above 16M Area Being Used...: 315,392
  User Id.....................: USER001

Pgm)==>Program Entry is To:  USERPGM2.USERPGM2+(000000)                                             005E6858 2012021-11:08:17.898496
Pgm)==>Program Entered From: z/OS Initiator                                                         005E6858 2012021-11:08:17.898496
Pgm)==>Program Attributes: LMod Start=32700988  LMod Entry Point=B2700988  LMod Size=00043678       005E6858 2012021-11:08:17.898496
                           LMod Amode=31  LMod Subpool=251  Auth Lib=No    AC=0   Rent/Reus         005E6858 2012021-11:08:17.898496
                           CPU State=Problem Pgm   PSW Key=8                                        005E6858 2012021-11:08:17.898496
Msg) Main User Program 2 Example                                                                    005E6858 2012021-11:08:17.898496
+    General Purpose Registers On Entry:                                                            005E6858 2012021-11:08:17.898496
+    R0 = 00000006FD000008  R1 = 0000000000006FF8  R2 = 0000000000000040  R3 = 00000000005D1D84     005E6858 2012021-11:08:17.898496
+    R4 = 00000000005D1D60  R5 = 00000000005E6E88  R6 = 00000000005B6FE0  R7 = 00000000FD000000     005E6858 2012021-11:08:17.898496
+    R8 = 00000000005FCE28  R9 = 00000000005E6B18  R10= 0000000000000000  R11= 00000000005E6E88     005E6858 2012021-11:08:17.898496
+    R12= 0000000083714D7A  R13= 0000000000006F60  R14= 0000000080FDEBD0  R15= 00000000B2700988     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = 00000000         AR1 = 32705E90         AR2 = 00000000         AR3 = 00000000            005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000            005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000            005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000            005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: USER0300                                                                       005E6858 2012021-11:08:17.898496
Msg) Show the SYSPRINT DCB before Open                                                              005E6858 2012021-11:08:17.898496
+    SYSPRINT                                                                                       005E6858 2012021-11:08:17.898496
+       0000000000007344          00000000 00000000 00000000 00000000   *................*          005E6858 2012021-11:08:17.898496
+       0000000000007354     +10  00000000 00000001 00004000 00000001   *.......... .....*          005E6858 2012021-11:08:17.898496
+       0000000000007364     +20  00000001 94000000 E2E8E2D7 D9C9D5E3   *....m...SYSPRINT*          005E6858 2012021-11:08:17.898496
+       0000000000007374     +30  02000050 00000001 00000001 00000000   *...&............*          005E6858 2012021-11:08:17.898496
+       0000000000007384     +40  00000000 00000001 00000001 00000001   *................*          005E6858 2012021-11:08:17.898496
+       0000000000007394     +50  00000079 00000001 00000000 00000001   *................*          005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: USER0400                                                                       005E6858 2012021-11:08:17.898496
Msg) Show the SYSPRINT DCB after Open                                                               005E6858 2012021-11:08:17.898496
+    SYSPRINT                                                                                       005E6858 2012021-11:08:17.898496
+       0000000000007344          00000000 00000000 00000000 00000000   *................*          005E6858 2012021-11:08:17.898496
+       0000000000007354     +10  00020000 01010F78 00794000 00006DB8   *.......... ..._.*          005E6858 2012021-11:08:17.898496
+       0000000000007364     +20  02000001 94000000 002C0050 005B0F80   *....m......&.$..*          005E6858 2012021-11:08:17.898496
+       0000000000007374     +30  92D7E618 00000001 00000001 08090079   *kPW.............*          005E6858 2012021-11:08:17.898496
+       0000000000007384     +40  00000000 00006C58 00010FF9 00010FF9   *......%....9...9*          005E6858 2012021-11:08:17.898496
+       0000000000007394     +50  00000079 00000001 00000000 00000001   *................*          005E6858 2012021-11:08:17.898496

Pgm)==>Program Entry is To:  USERPGM2.ABCD0000+(000000)                                             005E6858 2012021-11:08:17.898496
Pgm)==>Program Entry Status: CPU State=Problem Pgm  PSW Key=8  Amode=31                             005E6858 2012021-11:08:17.898496
Pgm)==>Program Entered From: USERPGM2.USERPGM2+(000372)                                             005E6858 2012021-11:08:17.898496
Msg) Called Program 1 Example                                                                       005E6858 2012021-11:08:17.898496
+    General Purpose Registers On Entry:                                                            005E6858 2012021-11:08:17.898496
+    R0 = 0000000032700F88  R1 = 00000000000071F0  R2 = 0000000000000000  R3 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 00000000327021A0     005E6858 2012021-11:08:17.898496
+    R12= 00000000327011A0  R13= 00000000000073B8  R14= 00000000B2700CFC  R15= 0000000032701190     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = B1C3857A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000            005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000            005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000            005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000            005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCD0100                                                                       005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCD0200                                                                       005E6858 2012021-11:08:17.898496
+    General Purpose Registers At Log Point: ABCD0200                                               005E6858 2012021-11:08:17.898496
+    R1 = 00000000000A0000  R3 = 0000000000000000  R5 = 0000000000000000                            005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR1 = 00000000         AR3 = 00000000         AR5 = 00000000                                   005E6858 2012021-11:08:17.898496

Pgm)==>Program Entry is To:  USERPGM2.WXYZ0000+(000000)                                             005E6858 2012021-11:08:17.898496
Pgm)==>Program Entry Status: CPU State=Problem Pgm  PSW Key=8  Amode=31                             005E6858 2012021-11:08:17.898496
Pgm)==>Program Entered From: USERPGM2.ABCD0000+(00033C)                                             005E6858 2012021-11:08:17.898496
Msg) Called Program 2 Example                                                                       005E6858 2012021-11:08:17.898496
+    General Purpose Registers On Entry:                                                            005E6858 2012021-11:08:17.898496
+    R0 = 0000000032700F88  R1 = 00000000000075A8  R2 = 0000000000000000  R3 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 0000000000000000     005E6858 2012021-11:08:17.898496
+    R12= 0000000032701958  R13= 0000000000007618  R14= 00000000B27014CE  R15= 0000000032701948     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = B1C3857A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000            005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000            005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000            005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000            005E6858 2012021-11:08:17.898496
```

```
Log) ==>Log Point**: WXYZ0100                                                                       005E6858 2012021-11:08:17.898496
Msg) I am here.                                                                                     005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: WXYZ0200                                                                       005E6858 2012021-11:08:17.898496
+    Private Area Storage Allocated:                                                                005E6858 2012021-11:08:17.898496
+      <16M: In Use=574K       Unused=3585K       Limit=4160K       HighU=574K                      005E6858 2012021-11:08:17.898496
+      >16M: In Use=397K       Unused=32370K      Limit=32768K      HighU=397K                      005E6858 2012021-11:08:17.898496
+      >BAR: Allocated=0 Megabytes                   Guard Amount=0 Megabytes                       005E6858 2012021-11:08:17.898496
+      >BAR: Pvt Used =0 Megabytes                   Pvt Hi-Water=0 Megabytes                       005E6858 2012021-11:08:17.898496
+      >BAR: Pvt Obj  =0                             Shr Objects =0                                 005E6858 2012021-11:08:17.898496
+      >BAR: Shr Alloc=0 Megabytes                   Shr Hi-Water=0 Megabytes                       005E6858 2012021-11:08:17.898496
+      >BAR: Com Alloc=0 Megabytes                   Com Hi-Water=0 Megabytes                       005E6858 2012021-11:08:17.898496
+      >BAR: Com Obj  =0                             Large Pages =0                                 005E6858 2012021-11:08:17.898496
+    Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE                   005E6858 2012021-11:08:17.898496
+     SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space          005E6858 2012021-11:08:17.898496
+             Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line        005E6858 2012021-11:08:17.898496
+     003-8   3           0           466,944     0           32,768          0                     005E6858 2012021-11:08:17.898496
+     230-5   3           5           12,288      11,840      0               0                     005E6858 2012021-11:08:17.898496
+     251-8   1           1           0           0           278,528         2,440                 005E6858 2012021-11:08:17.898496
+             ----------  ----------  ----------  ----------  --------------  --------------        005E6858 2012021-11:08:17.898496
+     Totals  7           6           479,232     11,840      311,296         2,440                 005E6858 2012021-11:08:17.898496
+    Job Step CPU: 0HR 0MIN 0.00116SEC       Job Step SRB: 0HR 0MIN 0.00010SEC                      005E6858 2012021-11:08:17.898496
Pgm)<==Program Exited From: USERPGM2.WXYZ0000+(0003EA)                                              005E6858 2012021-11:08:17.898496
Pgm)<==Program Exit Status: CPU State=Problem Pgm  PSW Key=8  Amode=31                              005E6858 2012021-11:08:17.898496
Pgm)<==Program Exited To:   USERPGM2.ABCD0000+(00033E)                                              005E6858 2012021-11:08:17.898496
+            Return Code =  00000004   Reason Code =  00000005                                      005E6858 2012021-11:08:17.898496
+    General Purpose Registers On Exit:                                                             005E6858 2012021-11:08:17.898496
+    R0 = 0000000000000005  R1 = 00000000000075A8  R2 = 00000000000073B8  R3 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000004  R7 = 0000000000000005     005E6858 2012021-11:08:17.898496
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 0000000000000000     005E6858 2012021-11:08:17.898496
+    R12= 0000000032701958  R13= 0000000000007618  R14= 00000000B27014CE  R15= 0000000000000004     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = B1C3857A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000             005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000             005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000             005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000             005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCD0300                                                                       005E6858 2012021-11:08:17.898496
Msg) Show Contents of Data Table and Regs                                                           005E6858 2012021-11:08:17.898496
+    General Purpose Registers At Log Point: ABCD0300                                               005E6858 2012021-11:08:17.898496
+    R0 = 0000000000000000  R1 = 00000000000075A8  R2 = 00000000800130E1  R3 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 00000000327021A0     005E6858 2012021-11:08:17.898496
+    R12= 00000000327011A0  R13= 00000000000073B8  R14= 00000000B27014CE  R15= 0000000000000100     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = B1C3857A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000             005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000             005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000             005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000             005E6858 2012021-11:08:17.898496
+    DATATABL                                                                                       005E6858 2012021-11:08:17.898496
+       0000000032701722          00010203 04050607 08090A0B 0C0D0E0F   *................*          005E6858 2012021-11:08:17.898496
+       0000000032701732     +10  10111213 14151617 18191A1B 1C1D1E1F   *................*          005E6858 2012021-11:08:17.898496
+       0000000032701742     +20  20212223 24252627 28292A2B 2C2D2E2F   *................*          005E6858 2012021-11:08:17.898496
+       0000000032701752     +30  30313233 34353637 38393A3B 3C3D3E3F   *................*          005E6858 2012021-11:08:17.898496
+       0000000032701762     +40  40414243 44454647 48494A4B 4C4D4E4F   * ...........<(+|*          005E6858 2012021-11:08:17.898496
+       0000000032701772     +50  50515253 54555657 58595A5B 5C5D5E5F   *&.........!$*)..*          005E6858 2012021-11:08:17.898496
+       0000000032701782     +60  60616263 64656667 68696A6B 6C6D6E6F   *-/.........,%_>?*          005E6858 2012021-11:08:17.898496
+       0000000032701792     +70  70717273 74757677 78797A7B 7C7D7E7F   *..........:#@'=.*          005E6858 2012021-11:08:17.898496
+       00000000327017A2     +80  80818283 84858687 88898A8B 8C8D8E8F   *.abcdefghi......*          005E6858 2012021-11:08:17.898496
+       00000000327017B2     +90  90919293 94959697 98999A9B 9C9D9E9F   *.jklmnopqr......*          005E6858 2012021-11:08:17.898496
+       00000000327017C2     +A0  A0A1A2A3 A4A5A6A7 A8A9AAAB ACADAEAF   *..stuvwxyz......*          005E6858 2012021-11:08:17.898496
+       00000000327017D2     +B0  B0B1B2B3 B4B5B6B7 B8B9BABB BCBDBEBF   *................*          005E6858 2012021-11:08:17.898496
+       00000000327017E2     +C0  C0C1C2C3 C4C5C6C7 C8C9CACB CCCDCECF   *.ABCDEFGHI......*          005E6858 2012021-11:08:17.898496
+       00000000327017F2     +D0  D0D1D2D3 D4D5D6D7 D8D9DADB DCDDDEDF   *.JKLMNOPQR......*          005E6858 2012021-11:08:17.898496
+       0000000032701802     +E0  E0E1E2E3 E4E5E6E7 E8E9EAEB ECEDEEEF   *..STUVWXYZ......*          005E6858 2012021-11:08:17.898496
+       0000000032701812     +F0  F0F1F2F3 F4F5F6F7 F8F9FAFB FCFDFEFF   *0123456789......*          005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCD0400                                                                       005E6858 2012021-11:08:17.898496
Sub)   ==>Subroutine: USERPGM2.ABCD0000.ASUB0000                                                    005E6858 2012021-11:08:17.898496
          From: USERPGM2.ABCD0000+(00050C)                                                          005E6858 2012021-11:08:17.898496
Sub)   <==Subroutine Exit: USERPGM2.ABCD0000.ASUB0000                                               005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCD$END                                                                       005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: ABCDEXIT                                                                       005E6858 2012021-11:08:17.898496
Pgm)<==Program Exited From: USERPGM2.ABCD0000+(0004EE)                                              005E6858 2012021-11:08:17.898496
Pgm)<==Program Exit Status: CPU State=Problem Pgm  PSW Key=8  Amode=31                              005E6858 2012021-11:08:17.898496
Pgm)<==Program Exited To:   USERPGM2.USERPGM2+(000374)                                              005E6858 2012021-11:08:17.898496
+            Return Code =  00000000   Reason Code =  00000000                                      005E6858 2012021-11:08:17.898496
+    General Purpose Registers On Exit:                                                             005E6858 2012021-11:08:17.898496
+    R0 = 0000000000000000  R1 = 00000000000075A8  R2 = 0000000000007000  R3 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R4 = 0000000000000000  R5 = 0000000000000000  R6 = 0000000000000000  R7 = 0000000000000000     005E6858 2012021-11:08:17.898496
+    R8 = 0000000000000000  R9 = 0000000000000000  R10= 0000000000000000  R11= 00000000327021A0     005E6858 2012021-11:08:17.898496
+    R12= 00000000327011A0  R13= 00000000000073B8  R14= 00000000B2701538  R15= 0000000000000000     005E6858 2012021-11:08:17.898496
+    Access Registers:                                                                              005E6858 2012021-11:08:17.898496
+    AR0 = B1C3857A         AR1 = 00000000         AR2 = 00000000         AR3 = 00000000             005E6858 2012021-11:08:17.898496
+    AR4 = 00000000         AR5 = 00000000         AR6 = 00000000         AR7 = 00000000             005E6858 2012021-11:08:17.898496
+    AR8 = 00000000         AR9 = 00000000         AR10= 00000000         AR11= 00000000             005E6858 2012021-11:08:17.898496
+    AR12= 00000000         AR13= 00000000         AR14= 00000000         AR15= 00000000             005E6858 2012021-11:08:17.898496
Log) ==>Log Point**: USER0700                                                                       005E6858 2012021-11:08:17.906688
+    SYSPRINT                                                                                       005E6858 2012021-11:08:17.906688
+       0000000000007344          00000000 00000000 00000000 01000000   *................*          005E6858 2012021-11:08:17.906688
+       0000000000007354     +10  00000000 00010F78 00004000 00000001   *.......... .....*          005E6858 2012021-11:08:17.906688
+       0000000000007364     +20  00000001 94000000 E2E8E2D7 D9C9D5E3   *....m...SYSPRINT*          005E6858 2012021-11:08:17.906688
+       0000000000007374     +30  02000050 00000001 00000001 00080000   *...&............*          005E6858 2012021-11:08:17.906688
+       0000000000007384     +40  00000000 00000001 00010FF9 00010FF9   *...........9...9*          005E6858 2012021-11:08:17.906688
+       0000000000007394     +50  00000079 00000001 00000000 00000001   *................*          005E6858 2012021-11:08:17.906688
+    ANY$DATA                                                                                       005E6858 2012021-11:08:17.906688
+       0000000000007244          40404040 40404040 40404040 40404040   *                *          005E6858 2012021-11:08:17.906688
+       0000000000007254     +10  0000000000007254 TO 0000000000007333 SAME AS ABOVE                005E6858 2012021-11:08:17.906688
+       0000000000007334     +F0  40404040 40404040 40404040 40404040   *                *          005E6858 2012021-11:08:17.906688

                         ******************************
                         *      Program Abended       *
                         ******************************
Completion Code      System = 0C7      Reason = 00000000
Abend caused by program interrupt from data exception: (PIC 07)
 The following are some possible reasons for abend:
 -Attempt to execute decimal instruction on data not in signed-packed-decimal format.
 -Attempt to execute decimal instruction on packed data not aligned on rightmost byte in field.
 -Multiplicand in MP did not have at least as many leftmost zeros as the no. of bytes in multiplier.
 -Previous PACK used invalid zoned data or used wrong data area causing invalid packed data field.
 -The sign or digit code of data used in packed decimal, edit, or CVB instruction was invalid.
 -Base register had incorrect address or a storage area with expected packed data was overlaid.
 -Pack field had an invalid digit not 0-9, or last byte had invalid sign (not A, B, C, D, E, or F).

-Abend Summary-

Abend Code.................: System=0C7   Reason=00000000
Abend Location.............: USERPGM2.USERPGM2+(0003DE)  20120121 11.08
Program Status Word........: 078D1000 B2700D6A  (EC)
zArch Program Status Word..: 07851000 80000000 00000000 32700D6A
Data Around PSW............: 32700D64 ==> 00534450 C4EE1FFF 1F00E320
Breaking Event Address.....: 0000000032700D32 at USERPGM2.USERPGM2+(0003AA)
Abending Instruction Length: 4 bytes
CPU State..................: Problem State (Only unprivileged instructions valid)
Program Status Word Key....: 8  (The access key used for storage references by the CPU)
Addressing Mode............: 31 (Program was in 31-bit addressing mode at time of abend)
Condition Code in PSW......: 1  (Result of last executed instruction which set Cond Code in PSW)
Address Space Control Mode.: Primary-space

*** Instruction Which Caused the Abend ***

-Instruction Summary-

Instruction at Address..: 0000000032700D66
Machine Instruction.....: 4450 C4EE
Instruction Op Code.....: 44
Instruction Mnemonic....: EX       R5,1262(R0,R12)
Instruction Description.: Execute
Instruction Format......: R1,D2(X2,B2)
Instruction Type........: RX-a
Instruction Attributes..: Does not set Condition Code
Instruction Facility....: None, zArchitecture base

-Executed Target Instruction Summary-

Instruction at Address..: 0000000032700E82
Target Instruction......: F800 D1A0 2000
Instruction Op Code.....: F8
Instruction Mnemonic....: ZAP      416(0,R13),0(0,R2)
Instruction Description.: Zero and Add Packed Decimal
Instruction Format......: D1(L1,B1),D2(L2,B2)
Instruction Type........: SS-b
Instruction Attributes..: Sets Condition Code
Instruction Facility....: None, zArchitecture base

-Instructions in Vicinity-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement           Attributes
------   ---------------  -----------------------------------  --------------------------------------------------
   -7A   D244 90EF 1000   MVC      239(69,R9),0(R1)
   -74   9680 1004        OI       4(R1),X'80'                 Sets Condition Code
   -70   58F0 C34C        L        R15,844(R0,R12)
   -6C   05EF             BALR     R14,R15                     Modal Instruction
   -6A   D207 D3AC C6E0   MVC      940(8,R13),1760(R12)
   -64   4120 D344        LA       R2,836(R0,R13)              Modal Instruction
   -60   5020 D3B0        ST       R2,944(R0,R13)
   -5C   4120 D3AC        LA       R2,940(R0,R13)              Modal Instruction
   -58   1812             LR       R1,R2
   -56   1801             LR       R0,R1
   -54   1B11             SR       R1,R1                       Sets Condition Code
   -52   0A14             SVC      20    (CLOSE) Type=4        Calls module IGC00020
   -50   EBE1 D150 0024   STMG     R14,R1,336(R13)
   -4A   B98D 0001        EPSW     R0,R1
   -46   9001 D128        STM      R0,R1,296(R13)
   -42   E3F0 C1E0 0017   LLGT     R15,480(R0,R12)
   -3C   56F0 C204        O        R15,516(R0,R12)             Sets Condition Code
   -38   4400 D104        EX       R0,260(R0,R13)
   -34   A7F4 0004        BRC      15,*+8
   -30   3270             LTER     R7,R0                        Sets Condition Code
   -2E   1142             LNR      R4,R2                        Sets Condition Code
   -2C   BF0C D12A        ICM      R0,B'1100',298(R13)          Sets Condition Code
   -28   0400             SPM      R0                           New Condition Code is loaded
   -26   EBE1 D150 0004   LMG      R14,R1,336(R13)
   -20   4120 C66D        LA       R2,1645(R0,R12)              Modal Instruction
   -1C   4130 0007        LA       R3,7(R0,R0)                  Modal Instruction
   -18   B904 0043        LGR      R4,R3
   -14   EB44 0001 000D   SLLG     R4,R4,1
    -E   B904 0054        LGR      R5,R4
    -A   EB55 0004 000D   SLLG     R5,R5,4
    -4   B981 0053        OGR      R5,R3                        Sets Condition Code
  ===>   4450 C4EE        EX       R5,1262(R0,R12)
    +4   1FFF             SLR      R15,R15                      Sets Condition Code
    +6   1F00             SLR      R0,R0                        Sets Condition Code
    +8   E320 D080 0004   LG       R2,128(R0,R13)
    +E   90F0 D0E4        STM      R15,R0,228(R13)
   +12   EBE1 D150 0024   STMG     R14,R1,336(R13)
   +18   B98D 0001        EPSW     R0,R1

General Purpose Registers R0-R15 at Entry to Abend:
  0-3   0000000000000950  00000000005B0628  0000000032701001  0000000000000007
  4-7   000000000000000E  00000000000000E7  0000000000000000  0000000000000000
  8-11  0000000000000000  0000000000000000  0000000000000000  0000000000000000
  12-15 0000000032700994  0000000000007000  00000000B2700CFC  0000000000000000
Access Registers:
  0-3   B1C3857A  00000000  00000000  00000000
  4-7   00000000  00000000  00000000  00000000
  8-11  00000000  00000000  00000000  00000000
  12-15 00000000  00000000  00000000  00000000

-General Purpose Register Summary-

  R0:  0000000000000950 Allocated storage  (PSA+PSAE+PSAX)    Decimal=2,384 Addressable storage is 5,808 bytes
  R1:  00000000005B0628 (Invalid Storage)  0C4 if referenced  Decimal=5,965,352
B-R2:  0000000032701001 Load Module/Csect  USERPGM2+(000679)  Decimal=846,204,929 (31)
  R3:  0000000000000007 Allocated storage  (PSA+PSAE+PSAX)    Decimal=7 Addressable storage is 8,185 bytes
  R4:  000000000000000E Allocated storage  (PSA+PSAE+PSAX)    Decimal=14 Addressable storage is 8,178 bytes
  R5:  00000000000000E7 Allocated storage  (PSA+PSAE+PSAX)    Decimal=231 Addressable storage is 7,961 bytes
  R6:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R7:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R8:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R9:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R10: 0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R11: 0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R12: 0000000032700994 Load Module/Csect  USERPGM2+(00000C)  Decimal=846,203,284 (31)
B-R13: 0000000000007000 Allocated storage  Subpool=003 Key=8  TCB=005E6858 Addressable storage is 36,864 bytes
  R14: 00000000B2700CFC Load Module/Csect  USERPGM2+(000374)  Decimal=846,204,156 (31)
  R15: 0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes

  '?-Rx:' Used as a designated (D), index (X), base (B), odd (O) of an even/odd pair,
          or hardware implied (H) register in abending instruction ZAP
   Note:  Registers flagged are for the target instruction of the Execute instruction.

*** Last Instruction Causing Break in Sequential Execution Before Abend ***

Breaking Event Location: USERPGM2.USERPGM2+(0003AA)
Breaking Event Address.: 0000000032700D32

-Instruction Summary-

Instruction at Address..: 0000000032700D32
Machine Instruction.....: A7F4 0004
Instruction Op Code.....: A74
Instruction Mnemonic....: BRC      15,*+8
Instruction Description.: Branch Relative on Condition
Instruction Format......: M1,RI2
Instruction Type........: RI-c
Instruction Attributes..: Does not set Condition Code
Instruction Facility....: None, zArchitecture base

-Instructions in Vicinity-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement          Attributes
------   ---------------  ----------------------------------  -------------------------------------------------
   -7E   10FC             LPR      R15,R12                      Sets Condition Code
   -7C   BF0C D12A        ICM      R0,B'1100',298(R13)          Sets Condition Code
   -78   0400             SPM      R0                           New Condition Code is loaded
   -76   EBE1 D150 0004   LMG      R14,R1,336(R13)
   -70   4110 D344        LA       R1,836(R0,R13)               Modal Instruction
   -6C   4100 C5F4        LA       R0,1524(R0,R12)              Modal Instruction
   -68   1FFF             SLR      R15,R15                      Sets Condition Code
   -66   BFF7 1031        ICM      R15,B'0111',49(R1)           Sets Condition Code
   -62   05EF             BALR     R14,R15                      Modal Instruction
   -60   D2FF D244 C4F4   MVC      580(256,R13),1268(R12)
   -5A   4820 C4EC        LH       R2,1260(R0,R12)
   -56   47F0 C350        BC       15,848(R0,R12)
   -52   3270             LTER     R7,R0                      Sets Condition Code
   -50   1190             LNR      R9,R0                      Sets Condition Code
   -4E   4110 D1F0        LA       R1,496(R0,R13)             Modal Instruction
   -4A   18E2             LR       R14,R2
   -48   41F0 D244        LA       R15,580(R0,R13)            Modal Instruction
   -44   90EF 1000        STM      R14,R15,0(R1)
   -40   9680 1004        OI       4(R1),X'80'                Sets Condition Code
   -3C   58F0 C34C        L        R15,844(R0,R12)
   -38   05EF             BALR     R14,R15                    Modal Instruction
   -36   D207 D3AC C6E0   MVC      940(8,R13),1760(R12)
   -30   4120 D344        LA       R2,836(R0,R13)             Modal Instruction
   -2C   5020 D3B0        ST       R2,944(R0,R13)
   -28   4120 D3AC        LA       R2,940(R0,R13)             Modal Instruction
   -24   1812             LR       R1,R2
   -22   1801             LR       R0,R1
   -20   1B11             SR       R1,R1                      Sets Condition Code
   -1E   0A14             SVC      20    (CLOSE) Type=4       Calls module IGC00020
   -1C   EBE1 D150 0024   STMG     R14,R1,336(R13)
   -16   B98D 0001        EPSW     R0,R1
   -12   9001 D128        STM      R0,R1,296(R13)
    -E   E3F0 C1E0 0017   LLGT     R15,480(R0,R12)
    -8   56F0 C204        O        R15,516(R0,R12)            Sets Condition Code
    -4   4400 D104        EX       R0,260(R0,R13)
  ===>   A7F4 0004        BRC      15,*+8
    +4   3270             LTER     R7,R0                      Sets Condition Code
    +6   1142             LNR      R4,R2                      Sets Condition Code
    +8   BF0C D12A        ICM      R0,B'1100',298(R13)        Sets Condition Code
    +C   0400             SPM      R0                         New Condition Code is loaded
    +E   EBE1 D150 0004   LMG      R14,R1,336(R13)
   +14   4120 C66D        LA       R2,1645(R0,R12)            Modal Instruction


-Save Area Trace-
Proceeding Backward From Last Save Area To First Save Area:

Called Program Entry Point..: USERPGM2.USERPGM2+(000000)  20120121 11.08
     USERPGM2 Csect Address.: 0000000032700988
     USERPGM2 Csect Length..: 000006F8
Called Program Returns To...: Program was entered via LINK
Calling Program Save Area...: 0000000000006F60
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   00000006FD000008  0000000000006FF8  0000000000000040  000000005D1D84
  4-7   00000000005D1D60  00000000005E6E88  00000000005B6FE0  00000000FD000000
  8-11  00000000005FCE28  00000000005E6B18  0000000000000000  00000000005E6E88
  12-15 0000000083714D7A  0000000000006F60  0000000080FDEBD0  00000000B2700988

-Storage Use at Abend-
 Private Area Storage Allocated:
   <16M: In Use=610K       Unused=3549K       Limit=4160K       HighU=610K
   >16M: In Use=406K       Unused=32361K      Limit=32768K      HighU=406K
   >BAR: Allocated=0 Megabytes                   Guard Amount=0 Megabytes
   >BAR: Pvt Used =0 Megabytes                   Pvt Hi-Water=0 Megabytes
   >BAR: Pvt Obj  =0                             Shr Objects =0
   >BAR: Shr Alloc=0 Megabytes                   Shr Hi-Water=0 Megabytes
   >BAR: Com Alloc=0 Megabytes                   Com Hi-Water=0 Megabytes
   >BAR: Com Obj  =0                             Large Pages =0
 Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE
  SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space
          Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line
  003-8   4           0           503,808     0           32,768          0
  229-5   1           1           0           0           8,192           3,968
  230-5   3           5           8,192       7,968       4,096           1,064
  251-8   1           1           0           0           278,528         2,440
          ----------  ----------  ----------  ----------  --------------  --------------
  Totals  9           7           512,000     7,968       323,584         7,472
 Job Step CPU: 0HR 0MIN 0.25358SEC       Job Step SRB: 0HR 0MIN 0.01594SEC
 Subpool Storage Information:
  Extended Private Storage - Subpool Numbers that may be used for areas Above 16M Line
    ECSA Subpools.... 227      228   231  241
    ELSQA Subpools... 203 thru 225   233  234  235  253  254  255
    ESQA Subpools.... 239      245   246  247  248
    ESWA Subpools.... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252
  Private Storage - Subpool Numbers that may be used for areas Below 16M Line
    CSA Subpools..... 227      228   231  241
    LSQA Subpools.... 203 thru 225   233  234  235  253  254  255
    SQA Subpools..... 226      239   245
    SWA Subpools..... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252

-Data Set Allocations at Abend-
DDName    EXCP Count        Data Set Name                                     Disp  VolSer
STEPLIB   12                USER001.WORK.LINKLIB                              SHR   UT70C1
SYSPRINT                    USER001.USERJOB2.JOB02051.D0000106.?                    JES
SYSABEND                    USER001.USERJOB2.JOB02051.D0000107.?                    JES
LGRECOUT                    USER001.USERJOB2.JOB02051.D0000108.?                    JES
LGRSYSIN                    USER001.USERJOB2.JOB02051.D0000102.?                    JES
```

```
-Printing Storage Areas From Main Program-
Save Area Storage For CSECT: USERPGM2
Begin Dump of LCA Storage Area
   0000000000007000          D3C3C140 C6F5E2C1 00000000 B2700D32   *LCA F5SA........*
+  0000000000007010     +10  00000000 32705E81 00000000 07850000   *.......a.....e..*
+  0000000000007020     +20  00000000 80000000 00000000 000073AC   *................*
+  0000000000007030     +30  00000000 00000000 00000000 00000000   *................*
+  0000000000007040     +40  0000000000007040 TO 000000000000706F SAME AS ABOVE
+  0000000000007070     +70  00000000 00000000 00000000 32700994   *...............m*
+  0000000000007080     +80  00000000 00006F60 00000000 00017000   *......?-........*
+  0000000000007090     +90  00000006 00000000 00000000 00000000   *................*
+  00000000000070A0     +A0  00000000 00000000 00000000 00000000   *................*
+  00000000000070B0     +B0  00000000000070B0 TO 00000000000070CF SAME AS ABOVE
+  00000000000070D0     +D0  00000000 00000000 000073B8 005E6858   *................*
+  00000000000070E0     +E0  C0008C48 00000000 00000000 32700988   *...............h*
+  00000000000070F0     +F0  000006F8 00007000 000003B8 00000000   *...8............*
+  0000000000007100    +100  0CEF0CEF 0CEF0700 07000700 80000000   *................*
+  0000000000007110    +110  00017000 00000000 00000000 00000000   *................*
+  0000000000007120    +120  00000000 0000000A 07850000 80000000   *.........e......*
+  0000000000007130    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000007140    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000007150    +150  00000000 B2700CFC 00000000 00000000   *................*
+  0000000000007160    +160  00000000 00000950 00000000 005B0628   *.......&.....$..*
+  0000000000007170    +170  00000000 00000000 00000000 00000000   *................*
+  0000000000007180    +180  E4E2C5D9 D7C7D4F2 40C4A895 81948983   *USERPGM2 Dynamic*
+  0000000000007190    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  00000000000071A0    +1A0  00000000 00000000 00000000 00000000   *................*
+  00000000000071B0    +1B0  00000000000071B0 TO 00000000000071EF SAME AS ABOVE
+  00000000000071F0    +1F0  00000100 80007244 00000000 00000000   *................*
+  0000000000007200    +200  00000000 00000000 00000000 00000000   *................*
+  0000000000007210    +210  0000000000007210 TO 000000000000723F SAME AS ABOVE
+  0000000000007240    +240  00006FF8 40404040 40404040 40404040   *..?8            *
+  0000000000007250    +250  40404040 40404040 40404040 40404040   *                *
+  0000000000007260    +260  0000000000007260 TO 000000000000733F SAME AS ABOVE
+  0000000000007340    +340  40404040 00000000 00000000 00000000   *    ............*
+  0000000000007350    +350  01000000 00000000 00010F78 00004000   *.............. .*
+  0000000000007360    +360  00000001 00000001 94000000 E2E8E2D7   *........m...SYSP*
+  0000000000007370    +370  D9C9D5E3 02000050 00000001 00000001   *RINT...&........*
+  0000000000007380    +380  00080000 00000000 00000001 00010FF9   *...............9*
+  0000000000007390    +390  00010FF9 00000079 00000001 00000000   *...9............*
+  00000000000073A0    +3A0  00000001 8F000000 00007344 80000000   *................*
+  00000000000073B0    +3B0  00007344 000073B4                     *........        *
LGRWK2II Work Area
   0000000000017000          00000000 C6F4E2C1 00000000 B2708484   *....F4SA......dd*
+  0000000000017010     +10  00D7E618 000194C8 000194C8 00000024   *.PW...mH..mH....*
+  0000000000017020     +20  000194C8 00000001 00019515 000194F8   *..mH......n...m8*
+  0000000000017030     +30  32744000 000812A8 3271E166 32707E90   *.. ....y......=.*
+  0000000000017040     +40  32706E90 32705E90 00000000 0001801F   *..>.............*
+  0000000000017050     +50  00000000 32744000 00000000 00007618   *...... .........*
+  0000000000017060     +60  00000000 32701D98 00000000 32707E90   *.......q......=.*
+  0000000000017070     +70  00000000 32706E90 00000000 32705E90   *......>.........*
+  0000000000017080     +80  00000000 000812A8 00000000 000170D8   *.......y.......Q*
+  0000000000017090     +90  00000000 00000000 00000000 00000000   *................*
+  00000000000170A0     +A0  00000000000170A0 TO 00000000000170CF SAME AS ABOVE
+  00000000000170D0     +D0  00000000 00000000 00000000 C6F4E2C1   *............F4SA*
+  00000000000170E0     +E0  00000000 32709E0B 00000000 3270A189   *...............i*
+  00000000000170F0     +F0  00000000 00000009 00000000 000177F8   *...............8*
+  0000000000017100    +100  00000000 00000008 00000000 000184AD   *..............d.*
+  0000000000017110    +110  00000000 00000009 00000000 00017000   *................*
+  0000000000017120    +120  00000000 0001842E 00000000 7F766400   *......d.........*
+  0000000000017130    +130  00000000 0005B000 00000000 7F766400   *................*
+  0000000000017140    +140  00000000 3270AB78 00000000 00074000   *.............. .*
+  0000000000017150    +150  00000000 327092C8 00000000 00017000   *......kH........*
+  0000000000017160    +160  00000000 000171B0 00000000 00000000   *................*
+  0000000000017170    +170  00000000 00000000 00000000 00000000   *................*
+  0000000000017180    +180  0000000000017180 TO 00000000000171AF SAME AS ABOVE
+  00000000000171B0    +1B0  00000000 C6F4E2C1 00000000 00000000   *....F4SA........*
+  00000000000171C0    +1C0  00000031 000184AD 0005C000 00045000   *......d.......&.*
+  00000000000171D0    +1D0  0005C000 00000009 00017000 0001842E   *..............d.*
+  00000000000171E0    +1E0  7F766400 0005B000 7F766400 3270AB78   *................*
+  00000000000171F0    +1F0  00074280 3270A198 00000000 00000000   *.......q........*
+  0000000000017200    +200  00000000 00000000 00000000 00000000   *................*
+  0000000000017210    +210  0000000000017210 TO 000000000001722F SAME AS ABOVE
+  0000000000017230    +230  00000000 000170D8 00000000 00000000   *.......Q........*
+  0000000000017240    +240  00000000 00000000 00000000 00000000   *................*
+  0000000000017250    +250  0000000000017250 TO 000000000001742F SAME AS ABOVE
+  0000000000017430    +430  00000000 00000000 D3C7D9E6 D2F2C9C9   *........LGRWK2II*
+  0000000000017440    +440  00000000 00000000 00000000 00000000   *................*
+  0000000000017450    +450  00C901EC F7CA00AC 0C000000 010C0003   *.I..7...........*
+  0000000000017460    +460  005E6858 00000000 00000000 00000000   *................*
+  0000000000017470    +470  000174D0 00000000 00000001 FB000000   *................*
+  0000000000017480    +480  00020000 01080FF0 00854000 00006F40   *.......0.e ...? *
+  0000000000017490    +490  86000001 94017510 00540050 005B2048   *f...m......&.$..*
+  00000000000174A0    +4A0  92D7E618 00000001 00000001 08090085   *kPW............e*
+  00000000000174B0    +4B0  00000000 00006DE0 3274CFFD 3274CFFD   *......_.........*
+  00000000000174C0    +4C0  00000085 00000001 00000000 00000001   *...e............*
+  00000000000174D0    +4D0  C4C3C2C5 00380000 00017470 00000000   *DCBE............*
+  00000000000174E0    +4E0  C0800000 00000000 00000000 00000000   *................*
+  00000000000174F0    +4F0  00000000 00000000 00000000 00000000   *................*
+  0000000000017500    +500  00000000 00000000 00850000 00000000   *.........e......*
+  0000000000017510    +510  91017518 00000000 B916002F E3F0F01C   *j...........T00.*
+  0000000000017520    +520  0017A739 05181133 41223000 A5FB0001   *..x.........v...*
+  0000000000017530    +530  0BEF0000 3270B128 00000000 C6F4E2C1   *............F4SA*
+  0000000000017540    +540  00081000 B2700CFC 3270D328 00000950   *..........L....&*
+  0000000000017550    +550  00017000 32701001 00000007 0000000E   *................*
+  0000000000017560    +560  000000E7 00000000 00000000 00000000   *...X............*
+  0000000000017570    +570  00000000 00000000 00000000 32700994   *...............m*
+  0000000000017580    +580  00000000 00000000 00000000 00000000   *................*
+  0000000000017590    +590  0000000000017590 TO 00000000000175AF SAME AS ABOVE
+  00000000000175B0    +5B0  00000000 00000000 00000000 3274DF70   *................*
+  00000000000175C0    +5C0  00000000 00000000 00000000 00000000   *................*
+  00000000000175D0    +5D0  00000000000175D0 TO 000000000001760F SAME AS ABOVE
+  0000000000017610    +610  00000000 3271ED58 00000000 000811F0   *...............0*
+  0000000000017620    +620  00000000 00007000 00000000 00000000   *................*
+  0000000000017630    +630  00000000 00000000 00000000 00000000   *................*
+  0000000000017640    +640  00000000 00000000 00000000 00000000   *................*
+  0000000000017650    +650  00000000 3274D768 00000000 00017000   *......P.........*
+  0000000000017660    +660  00000000 3271ED58 00000000 00000000   *................*
+  0000000000017670    +670  00000000 3271E2E0 00000000 000812A8   *......S........y*
+  0000000000017680    +680  00000000 3270D8BF 00000000 3271E2D1   *......Q.......SJ*
+  0000000000017690    +690  00000000 00017690 00000000 00000970   *................*
+  00000000000176A0    +6A0  00000000 005E6858 00000000 00018000   *................*
+  00000000000176B0    +6B0  00000000 32708A08 00000000 0000000A   *................*
+  00000000000176C0    +6C0  00000000 00000000 00000000 32744000   *.............. .*
+  00000000000176D0    +6D0  00000000 00007000 00000000 32701080   *................*
+  00000000000176E0    +6E0  00000000 32707E90 00000000 32706E90   *......=.......>.*
+  00000000000176F0    +6F0  B1C3857A 00000000 00000000 00000000   *.Ce:............*
+  0000000000017700    +700  00000000 00000000 00000000 00000000   *................*
+  0000000000017710    +710  0000000000017710 TO 000000000001772F SAME AS ABOVE
+  0000000000017730    +730  4B4B4B4B 00000000 00000000 0001A801   *..............y.*
+  0000000000017740    +740  00000000 0000004E 00000000 00017740   *.......+....... *
+  0000000000017750    +750  00000000 00000000 00000000 00000002   *................*
+  0000000000017760    +760  00000000 3271E19F 00000000 00000003   *................*
+  0000000000017770    +770  00000000 32708A88 00000000 00000000   *.......h........*
+  0000000000017780    +780  00000000 00000000 00000000 00000000   *................*
+  0000000000017790    +790  00000000 00000000 00000000 00000000   *................*
+  00000000000177A0    +7A0  00000000 00000000 2012021F 11081790   *................*
+  00000000000177B0    +7B0  66880F00 9066880C 00000000 F1F1F0F8   *.h....h.....1108*
+  00000000000177C0    +7C0  F1F7F9F0 F6F6F8F8 F0000000 00000000   *179066880.......*
+  00000000000177D0    +7D0  00000000 00000000 00000000 00000000   *................*
+  00000000000177E0    +7E0  00000000 00000000 00000000 00000000   *................*
+  00000000000177F0    +7F0  00000000 00000000 00000000 00017000   *................*
+  0000000000017800    +800  00000000 000184AD 00000000 00000009   *......d.........*
+  0000000000017810    +810  00000000 00074158 00000000 00000000   *................*
+  0000000000017820    +820  00000000 00000000 00000000 00000000   *................*
+  0000000000017830    +830  0000000000017830 TO 000000000001788F SAME AS ABOVE
+  0000000000017890    +890  00000000 00000000 00000000 3270809C   *................*
+  00000000000178A0    +8A0  00000000 00000000 00000000 00017000   *................*
+  00000000000178B0    +8B0  3274D768 00007000 00007000 32744000   *..P........... .*
+  00000000000178C0    +8C0  00006F60 00000000 00000000 00006F60   *..?-..........?-*
+  00000000000178D0    +8D0  00000000 3270602C 00000000 32706F60   *......-.......?-*
+  00000000000178E0    +8E0  00000000 32707886 00000000 32708164   *.......f......a.*
+  00000000000178F0    +8F0  00000000 00000000 00000000 00000000   *................*
+  0000000000017900    +900  00000000 00000000 00000000 00000000   *................*
+  0000000000017910    +910  F1F1F0F8 F1F7F9F0 F6F6F8F8 0000F0F0   *110817906688..00*
+  0000000000017920    +920  F5C5F6F8 F5F840F2 F0F1F2F0 F2F160F1   *5E6858 2012021-1*
+  0000000000017930    +930  F17AF0F8 7AF1F74B F9F0F6F6 F8F8C400   *1:08:17.906688D.*
+  0000000000017940    +940  0000E4E2 C5D9D7C7 D4F20000 00000000   *..USERPGM2......*
+  0000000000017950    +950  00000000 00000000 00000000 00000000   *................*
+  0000000000017960    +960  0000E4E2 C5D9D7C7 D4F24040 40404040   *..USERPGM2      *
+  0000000000017970    +970  40404040 40404040 40404040 40404040   *                *
+  0000000000017980    +980  404040F2 F0F1F2F0 F1F2F140 F1F14BF0   *   20120121 11.0*
+  0000000000017990    +990  F84000C0 3AC03CC0 3EC040C0 42C044C0   *8 ........ .....*
+  00000000000179A0    +9A0  46C048C0 4AC04CC0 4EC050C0 52C054C0   *......<.+.&.....*
+  00000000000179B0    +9B0  56C058C0 5AC05CC0 5EC060C0 62C064C0   *....!.*...-.....*
+  00000000000179C0    +9C0  66C04E4D F0F0F0F3 F7F4E4E2 C5D9D7C7   *..+(000374USERPG*
+  00000000000179D0    +9D0  D4F24BE4 E2C5D9D7 C7D4F24E 4DF0F0F0   *M2.USERPGM2+(000*
+  00000000000179E0    +9E0  F3F7F45D 40404040 40404040 40404040   *374)            *
+  00000000000179F0    +9F0  40404040 40404040 40404040 40404040   *                *
+  0000000000017A00    +A00  40404040 40404040 40404040 40404040   *                *
+  0000000000017A10    +A10  4040E4E2 C5D9D7C7 D4F24BE6 E7E8E9F0   *  USERPGM2.WXYZ0*
+  0000000000017A20    +A20  F0F0F04E 4DF0F0F0 F0F0F05D 40404040   *000+(000000)    *
+  0000000000017A30    +A30  40404040 40404040 40404040 40404040   *                *
+  0000000000017A40    +A40  40404040 40404040 40404040 40404040   *                *
+  0000000000017A50    +A50  40404040 40404040 4040F4F0 F4F0C6F4   *          4040F4*
+  0000000000017A60    +A60  F6F000F0 F0F0F0F0 F0F0F0F0 F0F0F1F7   *60.0000000000017*
+  0000000000017A70    +A70  C1F7F000 00000000 00000000 00000000   *A70.............*
+  0000000000017A80    +A80  00000000 00000000 00000000 00000000   *................*
+  0000000000017A90    +A90  0000000000017A90 TO 0000000000017ABF SAME AS ABOVE
+  0000000000017AC0    +AC0  00000000 00000000 00000AD0 00000000   *................*
+  0000000000017AD0    +AD0  32700988 B2700988 00043678 0000FB33   *...h...h........*
+  0000000000017AE0    +AE0  20400000 00000000 00000000 00000000   *. ..............*
+  0000000000017AF0    +AF0  00000000 00000000 00000000 00000000   *................*
+  0000000000017B00    +B00  0000000000017B00 TO 0000000000017B3F SAME AS ABOVE
+  0000000000017B40    +B40  00000000 E281A585 40C19985 8140E2A3   *....Save Area St*
+  0000000000017B50    +B50  96998187 8540C696 9940C3E2 C5C3E37A   *orage For CSECT:*
+  0000000000017B60    +B60  40E4E2C5 D9D7C7D4 F2404040 40404040   * USERPGM2       *
+  0000000000017B70    +B70  40404040 40404040 40404040 40404040   *                *
+  0000000000017B80    +B80  0000000000017B80 TO 0000000000017C3F SAME AS ABOVE
+  0000000000017C40    +C40  40404040 96A495A3 7EF040D4 85878182   *    ount=0 Megab*
+  0000000000017C50    +C50  A8A385A2 40404040 40404040 40404040   *ytes            *
+  0000000000017C60    +C60  40404040 40404040 40404040 40404040   *                *
+  0000000000017C70    +C70  40406EC2 C1D97A40 D7A5A340 E4A28584   *  >BAR: Pvt Used*
+  0000000000017C80    +C80  407EF040 D4858781 82A8A385 A2404040   * =0 Megabytes   *
+  0000000000017C90    +C90  40404040 40404040 40404040 40404040   *                *
+  0000000000017CA0    +CA0  D7A5A340 C88960E6 81A38599 7EF040D4   *Pvt Hi-Water=0 M*
+  0000000000017CB0    +CB0  85878182 A8A385A2 40404040 40404040   *egabytes        *
+  0000000000017CC0    +CC0  40404040 40404040 40404040 40404040   *                *
+  0000000000017CD0    +CD0  40404040 40406EC2 C1D97A40 D7A5A340   *      >BAR: Pvt *
+  0000000000017CE0    +CE0  D6829140 407EF040 40404040 40404040   *Obj  =0         *
+  0000000000017CF0    +CF0  40404040 40404040 40404040 40404040   *                *
+  0000000000017D00    +D00  40404040 E2889940 D6829185 83A3A240   *    Shr Objects *
+  0000000000017D10    +D10  7EF04040 40404040 40404040 40404040   *=0              *
+  0000000000017D20    +D20  40404040 40404040 40404040 40404040   *                *
+  0000000000017D30    +D30  40404040 40404040 40406EC2 C1D97A40   *          >BAR: *
+  0000000000017D40    +D40  E2889940 C1939396 837EF040 D4858781   *Shr Alloc=0 Mega*
+  0000000000017D50    +D50  82A8A385 A2404040 40404040 40404040   *bytes           *
+  0000000000017D60    +D60  40404040 40404040 E2889940 C88960E6   *        Shr Hi-W*
+  0000000000017D70    +D70  81A38599 7EF040D4 85878182 A8A385A2   *ater=0 Megabytes*
+  0000000000017D80    +D80  40404040 40404040 40404040 40404040   *                *
+  0000000000017D90    +D90  40404040 40404040 40404040 40406EC2   *              >B*
+  0000000000017DA0    +DA0  C1D97A40 C3969440 C1939396 837EF040   *AR: Com Alloc=0 *
+  0000000000017DB0    +DB0  D4858781 82A8A385 A2404040 40404040   *Megabytes       *
+  0000000000017DC0    +DC0  40404040 40404040 40404040 C3969440   *            Com *
+  0000000000017DD0    +DD0  C88960E6 81A38599 7EF040D4 85878182   *Hi-Water=0 Megab*
+  0000000000017DE0    +DE0  A8A385A2 40404040 40404040 40404040   *ytes            *
+  0000000000017DF0    +DF0  40404040 40404040 40404040 40404040   *                *
+  0000000000017E00    +E00  40406EC2 C1D97A40 C3969440 D6829140   *  >BAR: Com Obj *
+  0000000000017E10    +E10  407EF040 40404040 40404040 40404040   * =0             *
+  0000000000017E20    +E20  40404040 40404040 40404040 40404040   *                *
+  0000000000017E30    +E30  D3819987 8540D781 8785A240 7EF04040   *Large Pages =0  *
+  0000000000017E40    +E40  40404040 40404040 40404040 40404040   *                *
+  0000000000017E50    +E50  0000000000017E50 TO 0000000000017F3F SAME AS ABOVE
+  0000000000017F40    +F40  40404040 00000000 00000000 000189DF   *    ..........i.*
+  0000000000017F50    +F50  00000000 00000013 00000000 00000000   *................*
+  0000000000017F60    +F60  00000000 00000000 00000000 00000000   *................*
+  0000000000017F70    +F70  0000000000017F70 TO 0000000000017FEF SAME AS ABOVE
+  0000000000017FF0    +FF0  00000000 00000000 00000000 00000000   *................*
PGMLG2II Control Block
   0000000032744000          00000000 00008000 00000097 008B0000   *...........p....*
+  0000000032744010     +10  00000000 00000000 00000000 00000000   *................*
+  0000000032744020     +20  0000000032744020 TO 000000003274407F SAME AS ABOVE
+  0000000032744080     +80  00000000 00000000 00000000 0000E0D7   *...............P*
+  0000000032744090     +90  D9D6C7D9 D4000000                     *ROGRM...        *
End of Logger Service Diagnostic Output.
```

8.4 Sample Abend Report Description

The following is an example of a diagnostic output report written to //LGRECOUT upon a user program abending.

```
                         ******************************
                         *      Program Abended       *
                         ******************************
Completion Code      System = 0C4      Reason = 00000011
Abend caused by program interrupt from page-translation exception: (PIC 11)
 The following are some possible reasons for abend:
 -Note: For this abend the PSW points to the abending instruction and not the next instruction.
 -An invalid address in instruction resulted in a reference to storage not get-mained.
 -A previously valid address in instruction referenced get-main storage area already released.
 -An address in instruction needed Amode-31 to reference 31-bit storage when program was in Amode-24.
 -An invalid length in instruction resulted in a fetch or store to storage not in your address space.
 -BCT count set incorrectly to not be positive results in loop being enforced on a negative count.
 -An LA instead of L or vice versa resulted in branch, base, or index register with invalid pointer.
 -Program loop using bad index to access storage eventually referenced an area not get-mained.
 -Program loop adding to address to access storage eventually referenced an area not in your program.
 -You passed an invalid parameter list address to a called routine and routine encountered S0C4.
 -Attempt was made to close a data set twice without an intervening open.
 -Attempt was made to read or write to a data set where the DCB was not opened.
 -Attempt to close data set where DCB in get-main area and DCB get-main area released before close.

-Abend Summary-

Abend Code.................: System=0C4   Reason=00000011
Abend Location.............: USRTEST7.TVCT0000+(0003D2)  20110920 15.04
Program Status Word........: 078D2001 B2707782  (EC)
zArch Program Status Word..: 07852001 80000000 00000000 32707782
Data Around PSW............: 3270777C ==> E320C5C0 0017C6F0 00001423
Breaking Event Address.....: 0000000000FEEA1A
Abending Instruction Length: 6 bytes
CPU State..................: Problem State (Only unprivileged instructions valid)
Program Status Word Key....: 8  (The access key used for storage references by the CPU)
Addressing Mode............: 64 (Program was in 64-bit addressing mode at time of abend)
Condition Code in PSW......: 2  (Result of last executed instruction which set Cond Code in PSW)
Address Space Control Mode.: Primary-space

-Abending Instruction Summary-

Instruction at Address..: 0000000032707782
Machine Instruction.....: C6F0 0000 1423
Instruction Op Code.....: C60
Instruction Mnemonic....: EXRL     R15,*+10310
Instruction Description.: Execute Relative Long
Instruction Format......: R1,RI2
Instruction Type........: RIL-b
Instruction Attributes..: Does not set Condition Code
Instruction Facility....: Execute-extension facility

-Executed Target Instruction Summary-

Instruction at Address..: 0000000032709FC8
Target Instruction......: EBB0 3030 0081
Instruction Op Code.....: EB81
Instruction Mnemonic....: ICMY     R11,0,48(R3)
Instruction Description.: Insert Characters under Mask Long (Low, 20-bit Long Displacement)
Instruction Format......: R1,M3,D2(B2)
Instruction Type........: RSY-b
Instruction Attributes..: Sets Condition Code
Instruction Facility....: Long-displacement facility
```

```
-Instructions in Vicinity of Abend-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement           Attributes
------   ---------------  -----------------------------------  --------------------------------
   -6E   0A08             SVC      8    (LOAD)
   -6C   12FF             LTR      R15,R15                     Sets Condition Code
   -6A   A774 003A        BRC      7,*+116
   -66   E3F0 C598 0004   LG       R15,1432(R0,R12)
   -60   B904 00E5        LGR      R14,R5
   -5C   E320 E000 0090   LLGC     R2,0(R0,R14)
   -56   B90B 00FF        SLGR     R15,R15                     Sets Condition Code
   -52   1814             LR       R1,R4
   -50   18F0             LR       R15,R0
   -4E   E320 C5B0 0017   LLGT     R2,1456(R0,R12)
   -48   E3F0 C5C8 0090   LLGC     R15,1480(R0,R12)
   -42   1816             LR       R1,R6
   -40   5860 C5B4        L        R6,1460(R0,R12)
   -3C   B904 00F0        LGR      R15,R0
   -38   E310 C5B8 0017   LLGT     R1,1464(R0,R12)
   -32   4160 C3C2        LA       R6,962(R0,R12)              Modal Instruction
   -2E   5830 C5BC        L        R3,1468(R0,R12)
   -2A   E3F0 C5C9 0090   LLGC     R15,1481(R0,R12)
   -24   4140 C3CC        LA       R4,972(R0,R12)              Modal Instruction
   -20   ECBC FF9A 817C   CGIJNH   R11,-127,*-204
   -1A   C25E 0000 0FFE   CLGFI    R5,X'00000FFE'              Sets Condition Code
   -14   A784 000A        BRC      8,*+20
   -10   5850 4000        L        R5,0(R0,R4)
    -C   E330 D1F8 0004   LG       R3,504(R0,R13)
    -6   E320 C5C0 0017   LLGT     R2,1472(R0,R12)
  ===>   C6F0 0000 1423   EXRL     R15,*+10310
    +6   B917 00F0        LLGTR    R15,R0
    +A   E330 0010 0017   LLGT     R3,16(R0,R0)
   +10   E330 3000 0017   LLGT     R3,0(R0,R3)
   +16   EBE1 D150 0024   STMG     R14,R1,336(R13)
   +1C   B98D 0001        EPSW     R0,R1
   +20   9001 D128        STM      R0,R1,296(R13)

General Purpose Registers R0-R15 at Entry to Abend:
  0-3   00000000B2A74AB8  000000000004A100  000000003271A7D4  000000007FFFFFA0
  4-7   000000003270778C  00000000E3300010  0000000032707782  0000000032708FB8
  8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000000000000
  12-15 00000000327073C0  0000000000009728  000000003270784A  0000000000000007
Access Registers:
  0-3   00000000  01010039  00000000  00000000
  4-7   00000000  00000000  00000000  00000000
  8-11  00000000  00000000  00000000  00000000
  12-15 00000000  00000000  00000000  00000C05

-General Purpose Register Summary-

  R0:  00000000B2A74AB8 Load Module/Csect  IGGCSI00+(000000)  Decimal=849,824,440 (31)
  R1:  000000000004A100 (Invalid Storage)  0C4 if referenced  Decimal=303,360
  R2:  000000003271A7D4 Load Module/Csect  USRTEST7+(01A254)  Decimal=846,309,332 (31)
B-R3:  000000007FFFFFA0 (Invalid Storage)  0C4 if referenced  Decimal=2,147,483,552
  R4:  000000003270778C Load Module/Csect  USRTEST7+(00720C)  Decimal=846,231,436 (31)
  R5:  00000000E3300010 (Invalid Storage)  0C4 if referenced  Decimal=-483,393,520
  R6:  0000000032707782 Load Module/Csect  USRTEST7+(007202)  Decimal=846,231,426 (31)
  R7:  0000000032708FB8 Load Module/Csect  USRTEST7+(008A38)  Decimal=846,237,624 (31)
  R8:  0000000032707FB8 Load Module/Csect  USRTEST7+(007A38)  Decimal=846,233,528 (31)
  R9:  000000000004AB38 (Invalid Storage)  0C4 if referenced  Decimal=305,976
  R10: 0000000000049B38 Allocated storage  Subpool=003 Key=8  TCB=005E6968 Addressable storage is 1,224 bytes
D-R11: 0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
  R12: 00000000327073C0 Load Module/Csect  USRTEST7+(006E40)  Decimal=846,230,464 (31)
  R13: 0000000000009728 Allocated storage  Subpool=003 Key=8  TCB=005E6968 Addressable storage is 260,312 bytes
  R14: 000000003270784A Load Module/Csect  USRTEST7+(0072CA)  Decimal=846,231,626 (31)
  R15: 0000000000000007 Allocated storage  (PSA+PSAE+PSAX)    Decimal=7 Addressable storage is 8,185 bytes

  '?-Rx:' Used as a designated (D), index (X), base (B), odd (O) of an even/odd pair,
          or hardware implied (H) register in abending instruction ICMY
   Note:  Registers flagged are for the target instruction of the Execute instruction.
```

```
-Save Area Trace-
Proceeding Backward From Last Save Area To First Save Area:

Called Program Entry Point..: USRTEST7.TVCT0000+(000000)  20110920 15.04
      TVCT0000 Csect Address: 00000000327073B0
      TVCT0000 Csect Length.: 000005E0
Called Program Returns To...: USRTEST7.TCAL0000+(0016B6)  20110920 15.04
Calling Program Save Area...: 0000000000009518
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   000000000004AB38  0000000000049B60  0000000032707292  0000000000000000
  4-7   000000003271A6A8  000000003271A7D4  0000000000000000  0000000032708FB8
  8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000032706BE8
  12-15 0000000032705BE8  0000000000009518  000000003270728E  00000000327073B0

Called Program Entry Point..: USRTEST7.TCAL0000+(000000)  20110920 15.04
      TCAL0000 Csect Address: 0000000032705BD8
      TCAL0000 Csect Length.: 000017D8
Called Program Returns To...: USRTEST7.TGMT0000+(00062E)  20110920 15.04
Calling Program Save Area...: 0000000000008B80
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   000000000004AB38  0000000000049B60  0000000000000003  00000000000088A2
  4-7   00000000000088B6  00000000000003E7  0000000000000000  0000000032708FB8
  8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000000000000
  12-15 0000000032704690  0000000000008B80  0000000032704CAE  0000000032705BD8

Called Program Entry Point..: USRTEST7.TGMT0000+(000000)  20110920 15.04
      TGMT0000 Csect Address: 0000000032704680
      TGMT0000 Csect Length.: 00000930
Called Program Returns To...: USRTEST7.TRDT0000+(00089E)  20110920 15.04
Calling Program Save Area...: 0000000000008608
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   0000000000008880  0000000000049B60  00000000327081E0  00000000000088DE
  4-7   0000000000007808  0000000000007861  00000000B2703A98  0000000032708FB8
  8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000032704308
  12-15 0000000032703308  0000000000008608  00000000B2703B96  0000000032704680

Called Program Entry Point..: USRTEST7.TRDT0000+(000000)  20110920 15.04
      TRDT0000 Csect Address: 00000000327032F8
      TRDT0000 Csect Length.: 00001388
Called Program Returns To...: USRTEST7.USRTEST7+(001812)  20110920 15.04
Calling Program Save Area...: 0000000000007000
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   0000000000007828  0000000000049B60  0000000800000014  00000000005FFE40
  4-7   0000000000007FF4  0000000000000189  FFFFFFFFFFFFFE78  0000000032708FB8
  8-11  0000000032707FB8  000000000004AB38  0000000000049B38  000000003270158C
  12-15 000000003270058C  0000000000007000  00000000B2701D92  00000000327032F8

Called Program Entry Point..: USRTEST7.USRTEST7+(000000)  20110920 15.04
     USRTEST7 Csect Address.: 0000000032700580
     USRTEST7 Csect Length..: 00001FC0
Called Program Returns To...: Program was entered via LINK
Calling Program Save Area...: 0000000000006F60
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   00000006FD000008  0000000000006FF8  0000000000000040  00000000005D1D84
  4-7   00000000005D1D60  00000000005E6E88  00000000005B8FE0  00000000FD000000
  8-11  00000000005FCC30  00000000005E6B30  0000000000000000  00000000005E6E88
  12-15 000000008370AD3A  0000000000006F60  0000000080FDE148  00000000B2700580

-Storage Use at Abend-
 Private Area Storage Allocated:
   <16M: In Use=952K       Unused=5167K       Limit=6120K       HighU=952K
   >16M: In Use=37289K     Unused=224854K     Limit=262144K     HighU=37289K
   >BAR: Allocated=10 Gigabytes                  Guard Amount=6,240 Megabytes
   >BAR: Pvt Used =4,000 Megabytes               Pvt Hi-Water=4,000 Megabytes
   >BAR: Pvt Obj  =1                             Shr Objects =0
   >BAR: Shr Alloc=0 Megabytes                   Shr Hi-Water=0 Megabytes
   >BAR: Com Alloc=0 Megabytes                   Com Hi-Water=0 Megabytes
   >BAR: Com Obj  =0                             Large Pages =0
 Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE
  SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space
```

```
          Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line
  003-8   10          1           741,376     2,872       2,334,720       0
  004-8   5,000       5,000       0           0           20,480,000      5,360,000
  005-8   5,000       5,000       0           0           20,480,000      480,000
  229-5   1           1           0           0           8,192           3,968
  230-0   1           1           4,096       1,824       0               0
  230-5   3           7           8,192       7,600       4,096           368
  230-8   1           1           0           0           4,096           4,080
  251-8   2           2           0           0           385,024         5,440
  252-0   1           1           0           0           4,096           2,744
          ----------  ----------  ----------  ----------  --------------  --------------
  Totals  10,019      10,014      753,664     12,296      43,700,224      5,856,600
 Job Step CPU: 0HR 0MIN 0.27666SEC       Job Step SRB: 0HR 0MIN 0.01498SEC
 Subpool Storage Information:
  Extended Private Storage - Subpool Numbers that may be used for areas Above 16M Line
    ECSA Subpools.... 227      228   231  241
    ELSQA Subpools... 203 thru 225   233  234  235  253  254  255
    ESQA Subpools.... 239      245   246  247  248
    ESWA Subpools.... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252
  Private Storage - Subpool Numbers that may be used for areas Below 16M Line
    CSA Subpools..... 227      228   231  241
    LSQA Subpools.... 203 thru 225   233  234  235  253  254  255
    SQA Subpools..... 226      239   245
    SWA Subpools..... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252

-Data Set Allocations at Abend-
DDName    EXCP Count        Data Set Name                                          Disp  VolSer
STEPLIB   19                USER001.WORK.LINKLIB                                   SHR   UT70C1
SYSPRINT                    USER001.WORK.SYSPRINT                                  SHR   UT70CD
SYSRECNT                    NULLFILE
SYSIN                       NULLFILE
SYSCHECK                    USER001.WORK.SYSCHECK                                  SHR   UT70C7
SYSABEND  4,709             USER001.WORK.DUMP                                      OLD   UT7119
LGRECOUT                    USER001.WORK.TRCEOUT                                   OLD   UT70C1
LGRSYSIN                    USER001.USRTEST7.JOB00359.D0000101.?                         JES

-Printing Storage Areas From Main Program-
Save Area Storage For CSECT: USRTEST7
Begin Dump of LCA Storage Area
   0000000000007000          D3C3C140 C6F5E2C1 00000000 B2701D92   *LCA F5SA.......k*
+  0000000000007010     +10  00000000 327032F8 00000000 00007828   *.......8........*
+  0000000000007020     +20  00000000 00049B60 00000008 00000014   *.......-........*
+  0000000000007030     +30  00000000 005FFE40 00000000 00007FF4   *....... .......4*
+  0000000000007040     +40  00000000 00000189 FFFFFFFF FFFFFE78   *.......i........*
+  0000000000007050     +50  00000000 32708FB8 00000000 32707FB8   *................*
+  0000000000007060     +60  00000000 0004AB38 00000000 00049B38   *................*
+  0000000000007070     +70  00000000 3270158C 00000000 3270058C   *................*
+  0000000000007080     +80  00000000 00006F60 00000000 00008608   *......?-......f.*
+  0000000000007090     +90  00000006 00000000 00000000 00000000   *................*
+  00000000000070A0     +A0  00000000 00000000 00000000 00000000   *................*
+  00000000000070B0     +B0  00000000000070B0 TO 00000000000070CF SAME AS ABOVE
+  00000000000070D0     +D0  00000000 00000000 00008608 005E6968   *..........f.....*
+  00000000000070E0     +E0  C00409F8 00000000 00000000 32700580   *...8............*
+  00000000000070F0     +F0  00001FC0 00007000 00001608 00000000   *................*
+  0000000000007100    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
+  0000000000007110    +110  00050000 00049B38 000004C4 00000000   *...........D....*
+  0000000000007120    +120  00000000 0000000A 07851000 80000000   *.........e......*
+  0000000000007130    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000007140    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000007150    +150  00000000 00000010 00000000 00000000   *................*
+  0000000000007160    +160  00000000 00000001 00000000 00000001   *................*
+  0000000000007170    +170  00000000 00000000 00000000 00000000   *................*
+  0000000000007180    +180  E4E2D9E3 C5E2E3F7 40C4A895 81948983   *USRTEST7 Dynamic*
+  0000000000007190    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  00000000000071A0    +1A0  F0F0C3F8 F6F7F6C5 C5F9F3F9 F2F8C5F0   *00C8676EE93928E0*
+  00000000000071B0    +1B0  C1C5F0F0 F0F0F0F0 F0F1F0C1 F0F0F0F4   *AE000000010A0004*
+  00000000000071C0    +1C0  08000000 00000000 00000000 00000000   *................*
+  00000000000071D0    +1D0  00000000 00000000 00000000 00000000   *................*
+  00000000000071E0    +1E0  00000000 00000000 00000000 00000000   *................*
+  00000000000071F0    +1F0  00C8676E E93928E0 AE000000 010A0004   *.H.>Z...........*
+  0000000000007200    +200  80000000 010A0004 00000000 00000000   *................*
+  0000000000007210    +210  00000000 00000000 00000000 00000000   *................*
+  0000000000007220    +220  0000000000007220 TO 00000000000072AF SAME AS ABOVE
+  00000000000072B0    +2B0  00000008 06400000 00000008 00000000   *..... ..........*
+  00000000000072C0    +2C0  00000000 00000000 00000000 00000000   *................*
+  00000000000072D0    +2D0  00000000 00000000 00000000 00007BBE   *..............#.*
+  00000000000072E0    +2E0  00000000 00000000 00000000 00000000   *................*
+  00000000000072F0    +2F0  00000000 00000000 00000000 329C2000   *................*
+  0000000000007300    +300  00000000 00008000 00000000 329CA000   *................*
+  0000000000007310    +310  00000000 00008000 00000C41 80049CB0   *................*
+  0000000000007320    +320  005E6748 00000000 00000000 00000000   *................*
+  0000000000007330    +330  00000000 00000000 00000000 00000000   *................*
+  0000000000007340    +340  0000000000007340 TO 000000000000735F SAME AS ABOVE
+  0000000000007360    +360  00010000 00900000 00000000 00002800   *................*
+  0000000000007370    +370  00000000 00000000 00000000 00000000   *................*
+  0000000000007380    +380  00000000 00000000 00000008 00000000   *................*
+  0000000000007390    +390  00000000 00000000 00000000 00000000   *................*
+  00000000000073A0    +3A0  00001860 00000000 00000000 00000000   *...-............*
+  00000000000073B0    +3B0  00000000 00000000 00000000 00000000   *................*
+  00000000000073C0    +3C0  00000000000073C0 TO 000000000000742F SAME AS ABOVE
+  0000000000007430    +430  0000749F 0000749F 00000000 005E6968   *................*
+  0000000000007440    +440  00000000 00000000 00000000 00000000   *................*
+  0000000000007450    +450  00000000 00000000 00000000 00000000   *................*
+  0000000000007460    +460  00000000 00000000 00000000 000003E8   *...............Y*
+  0000000000007470    +470  00000001 00000000 00000000 00000000   *................*
+  0000000000007480    +480  FF000000 00000000 00000000 00000000   *................*
+  0000000000007490    +490  00000000 00000000 00000000 00000000   *................*
+  00000000000074A0    +4A0  00000000000074A0 TO 000000000000750F SAME AS ABOVE
+  0000000000007510    +510  00000000 00000000 00000000 000000D2   *...............K*
+  0000000000007520    +520  C8C6F0F0 40404000 00000000 00000000   *HF00   .........*
+  0000000000007530    +530  00000000 00000000 00000000 00000000   *................*
+  0000000000007540    +540  0000000000007540 TO 00000000000077FF SAME AS ABOVE
+  0000000000007800    +800  00000000 00000000 0000789F 00000014   *................*
+  0000000000007810    +810  00007839 00000014 000078C7 00000014   *...........G....*
+  0000000000007820    +820  00007861 00000014 C5C4E3F1 F0F0F1F0   *.../....EDT10010*
+  0000000000007830    +830  F0000000 00000000 00F0F57A F0F04040   *0........05:00  *
+  0000000000007840    +840  40404040 40404040 40404040 40000000   *             ...*
+  0000000000007850    +850  00000000 00000000 00000000 00000000   *................*
+  0000000000007860    +860  00F0F67A F0F04040 40404040 40404040   *.06:00          *
+  0000000000007870    +870  40404040 40000000 00000000 00000000   *     ...........*
+  0000000000007880    +880  00000000 00000000 00000000 00000000   *................*
+  0000000000007890    +890  00000000 00000000 00000000 000000F2   *...............2*
+  00000000000078A0    +8A0  F0F0F361 F0F361F1 F5404040 40404040   *003/03/15       *
+  00000000000078B0    +8B0  40404000 00000000 00000000 00000000   *   .............*
+  00000000000078C0    +8C0  00000000 000000F2 F0F0F361 F0F661F1   *.......2003/06/1*
+  00000000000078D0    +8D0  F5404040 40404040 40404000 00000000   *5          .....*
+  00000000000078E0    +8E0  00000000 00000000 00000000 000000F1   *...............1*
+  00000000000078F0    +8F0  F04FF14F F4F04FF0 4FF04FF0 4FF24040   *0|1|40|0|0|0|2  *
+  0000000000007900    +900  40404040 40404040 40404040 40404040   *                *
+  0000000000007910    +910  0000000000007910 TO 000000000000792F SAME AS ABOVE
+  0000000000007930    +930  40404040 40404040 40404040 404040D9   *               R*
+  0000000000007940    +940  9FBC3C00 004110A1 70410000 00000000   *................*
+  0000000000007950    +950  00000000 00000000 00000000 00000000   *................*
+  0000000000007960    +960  0000000000007960 TO 00000000000079BF SAME AS ABOVE
+  00000000000079C0    +9C0  00000000 00000000 12345678 9012331C   *................*
+  00000000000079D0    +9D0  40404040 404040F0 00000000 00000000   *       0........*
+  00000000000079E0    +9E0  00404040 40404EF1 F0000000 00000000   *.     +10.......*
+  00000000000079F0    +9F0  00000000 00000010 00000000 00000000   *................*
+  0000000000007A00    +A00  00000000 00000000 00000000 00000000   *................*
+  0000000000007A10    +A10  0000000000007A10 TO 0000000000007A2F SAME AS ABOVE
+  0000000000007A30    +A30  00000000 00000000 00004040 40404040   *..........      *
+  0000000000007A40    +A40  40404040 40404040 40404040 40404040   *                *
+  0000000000007A50    +A50  0000000000007A50 TO 0000000000007AAF SAME AS ABOVE
+  0000000000007AB0    +AB0  40404040 40404040 40404040 40404000   *               .*
+  0000000000007AC0    +AC0  00000000 00000000 00000000 00000000   *................*
+  0000000000007AD0    +AD0  0000000000007AD0 TO 0000000000007BAF SAME AS ABOVE
+  0000000000007BB0    +BB0  00000000 00000000 00000000 00002020   *................*
+  0000000000007BC0    +BC0  20202020 20202020 20202020 20202020   *................*
+  0000000000007BD0    +BD0  0000000000007BD0 TO 0000000000007BEF SAME AS ABOVE
+  0000000000007BF0    +BF0  20202020 20202020 20202020 20202041   *................*
+  0000000000007C00    +C00  42434445 46474849 202E3C28 2B7C2651   *.............@..*
+  0000000000007C10    +C10  52535455 56575859 21242A29 3B5E2D2F   *................*
+  0000000000007C20    +C20  62636465 66676869 202C255F 3E3F7071   *................*
+  0000000000007C30    +C30  72737475 76777860 3A234027 3D228061   *.......-.. ..../*
+  0000000000007C40    +C40  62636465 66676869 8A8B8C8D 8E8F906A   *................*
+  0000000000007C50    +C50  6B6C6D6E 6F707172 9A9B9C9D 9E9FA07E   *,%_>?..........=*
+  0000000000007C60    +C60  73747576 7778797A AAABACAD AEAF5EB1   *.......:........*
+  0000000000007C70    +C70  B2B3B4B5 B6B7B8B9 5B5DBCBD BEBF7B41   *........$)....#.*
+  0000000000007C80    +C80  42434445 46474849 CACBCCCD CECF7D4A   *..............'.*
+  0000000000007C90    +C90  4B4C4D4E 4F505152 DADBDCDD DEDF5CE1   *.<(+|&........*.*
+  0000000000007CA0    +CA0  53545556 5758595A EAEBECED EEEF3031   *.......!........*
+  0000000000007CB0    +CB0  32333435 36373839 20202020 20FF4040   *..............  *
+  0000000000007CC0    +CC0  40404040 40404040 40404040 40404040   *                *
+  0000000000007CD0    +CD0  0000000000007CD0 TO 0000000000007FAF SAME AS ABOVE
+  0000000000007FB0    +FB0  40404040 40404040 40404040 40400000   *              ..*
+  0000000000007FC0    +FC0  00000000 00000000 00000000 00000000   *................*
+  0000000000007FD0    +FD0  0000000000007FD0 TO 00000000000085FF SAME AS ABOVE
+  0000000000008600   +1600  00008600 00000000                     *..f.....        *
Save Area Storage For CSECT: TRDT0000
Begin Dump of LCA Storage Area
   0000000000008608          D3C3C140 C6F4E2C1 00000000 B2703B96   *LCA F4SA.......o*
+  0000000000008618     +10  00000000 32704680 00000000 00008880   *..............h.*
+  0000000000008628     +20  00000000 00049B60 00000000 327081E0   *.......-......a.*
+  0000000000008638     +30  00000000 000088DE 00000000 00007808   *......h.........*
+  0000000000008648     +40  00000000 00007861 00000000 B2703A98   *......./.......q*
+  0000000000008658     +50  00000000 32708FB8 00000000 32707FB8   *................*
+  0000000000008668     +60  00000000 0004AB38 00000000 00049B38   *................*
+  0000000000008678     +70  00000000 32704308 00000000 32703308   *................*
+  0000000000008688     +80  00000000 00007000 00000000 00008B80   *................*
+  0000000000008698     +90  00000000 00000000 00000000 00000000   *................*
+  00000000000086A8     +A0  00000000000086A8 TO 00000000000086D7 SAME AS ABOVE
+  00000000000086D8     +D0  00000000 00000000 00008B80 005E6968   *................*
+  00000000000086E8     +E0  00040480 00000000 00000000 327032F8   *...............8*
+  00000000000086F8     +F0  00001388 00008608 00000578 00000000   *...h..f.........*
+  0000000000008708    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
+  0000000000008718    +110  00050000 00049B38 000004C4 00000000   *...........D....*
+  0000000000008728    +120  00000000 0000000A 07852000 80000000   *.........e......*
+  0000000000008738    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000008748    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000008758    +150  000003E8 00000000 00000000 00000000   *...Y............*
+  0000000000008768    +160  00000000 00000000 00000000 0000892E   *..............i.*
+  0000000000008778    +170  00000000 B2703B3E 00000000 32704402   *................*
+  0000000000008788    +180  E3D9C4E3 F0F0F0F0 40C4A895 81948983   *TRDT0000 Dynamic*
+  0000000000008798    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  00000000000087A8    +1A0  00000000 0008566C 66000000 0000103F   *.......%........*
+  00000000000087B8    +1B0  F1F0F0F0 F0F0F0F0 F0F1F0C1 F0F0F0F4   *10000000010A0004*
+  00000000000087C8    +1C0  00000000 00000000 00000000 00000000   *................*
+  00000000000087D8    +1D0  00000000000087D8 TO 00000000000087F7 SAME AS ABOVE
+  00000000000087F8    +1F0  00C8676E E9392E90 10000000 010A0004   *.H.>Z...........*
+  0000000000008808    +200  00000000 00000000 00000000 00000000   *................*
+  0000000000008818    +210  0000000000008818 TO 0000000000008847 SAME AS ABOVE
+  0000000000008848    +240  0000749F 00000000 00007808 000078B3   *................*
+  0000000000008858    +250  0000784D 000078DB 00007875 00007828   *...(............*
+  0000000000008868    +260  00000000 0000000A 00007448 00005460   *...............-*
+  0000000000008878    +270  00000000 00000000 2BA40D50 2C1D6560   *.........u.&...-*
+  0000000000008888    +280  07D30006 000FF2F0 F0F361F0 F361F1F5   *.L....2003/03/15*
+  0000000000008898    +290  40404040 40404040 40404040 40404040   *                *
+  00000000000088A8    +2A0  00000000000088A8 TO 00000000000088C7 SAME AS ABOVE
+  00000000000088C8    +2C0  4040F0F5 7AF0F040 40404040 40404040   *  05:00         *
+  00000000000088D8    +2D0  40404040 4040F2F0 F0F361F0 F661F1F5   *      2003/06/15*
+  00000000000088E8    +2E0  40404040 40404040 4040F2F0 F0F3F0F6   *          200306*
+  00000000000088F8    +2F0  F1F54040 40404040 40404040 40400000   *15            ..*
+  0000000000008908    +300  00000000 00000000 00000000 00000000   *................*
+  0000000000008918    +310  0000F0F6 7AF0F040 40404040 40404040   *..06:00         *
+  0000000000008928    +320  40404040 40400000 000000F2 F0F0F3F0   *      .....20030*
+  0000000000008938    +330  F6F1F5F0 F5F0F0F0 F0F0F6F0 F0F0F0C5   *615050000060000E*
+  0000000000008948    +340  C4E30080 001C0103 166CC1C2 D3D3D4D5   *DT.......%ABLLMN*
+  0000000000008958    +350  C4C5C6C3 C4C5D4D5 D6D7D9E6 E9E8E2E3   *DEFCDEMNOPRWZYST*
+  0000000000008968    +360  E5C1C1C1 C2C2C2C3 C3C3C6C6 C6C5C5C5   *VAAABBBCCCFFFEEE*
+  0000000000008978    +370  D1D2D3C2 C3C4C3C4 C5C4C5C6 C5C6C7C8   *JKLBCDCDEDEFEFGH*
+  0000000000008988    +380  C9D1C9D1 D2D1D2D3 D2D3D4D4 D5D6D7D8   *IJIJKJKLKLMMNOPQ*
+  0000000000008998    +390  D9E2E3E4 E5E6E7C1 C1C4C1C1 C3C1C1C4   *RSTUVWXAADAACAAD*
+  00000000000089A8    +3A0  C1C1C6C1 C1C5C4C4 C1C4C4C3 C4C4C6C3   *AAFAAEDDADDCDDFC*
+  00000000000089B8    +3B0  D3C2C3D3 C1C3D340 206B2020 206B2020   *LBCLACL .,...,..*
+  00000000000089C8    +3C0  206B2020 206B2020 206B2020 206B4040   *.,...,...,...,  *
+  00000000000089D8    +3D0  40404040 40404040 40404040 40404040   *                *
+  00000000000089E8    +3E0  00000000000089E8 TO 0000000000008A57 SAME AS ABOVE
+  0000000000008A58    +450  40404000 00000000 00000000 00000000   *   .............*
+  0000000000008A68    +460  00000000 00000000 00000000 00000000   *................*
+  0000000000008A78    +470  0000000000008A78 TO 0000000000008B57 SAME AS ABOVE
+  0000000000008B58    +550  00000000 00000000 00000000 B2703A98   *...............q*
+  0000000000008B68    +560  00000000 B27042E8 00000000 B2703B3E   *.......Y........*
+  0000000000008B78    +570  00008B78 00000000                     *........        *
Save Area Storage For CSECT: TGMT0000
Begin Dump of LCA Storage Area
   0000000000008B80          D3C3C140 C6F4E2C1 00000000 32704CAE   *LCA F4SA......<.*
+  0000000000008B90     +10  00000000 32705BD8 00000000 0004AB38   *......$Q........*
+  0000000000008BA0     +20  00000000 00049B60 00000000 00000003   *......-.........*
+  0000000000008BB0     +30  00000000 000088A2 00000000 000088B6   *......hs......h.*
+  0000000000008BC0     +40  00000000 000003E7 00000000 00000000   *.......X........*
+  0000000000008BD0     +50  00000000 32708FB8 00000000 32707FB8   *................*
+  0000000000008BE0     +60  00000000 0004AB38 00000000 00049B38   *................*
+  0000000000008BF0     +70  00000000 00000000 00000000 32704690   *................*
+  0000000000008C00     +80  00000000 00008608 00000000 00009518   *......f.......n.*
+  0000000000008C10     +90  00000000 00000000 00000000 00000000   *................*
+  0000000000008C20     +A0  0000000000008C20 TO 0000000000008C4F SAME AS ABOVE
+  0000000000008C50     +D0  00000000 00000000 00009518 005E6968   *..........n.....*
+  0000000000008C60     +E0  0003FAE8 00000000 00000000 32704680   *...Y............*
+  0000000000008C70     +F0  00000930 00008B80 00000998 00000000   *...........q....*
+  0000000000008C80    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
+  0000000000008C90    +110  00050000 00049B38 000004C4 00000000   *...........D....*
+  0000000000008CA0    +120  00000000 0000000A 07851000 80000000   *.........e......*
+  0000000000008CB0    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000008CC0    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000008CD0    +150  00000000 327081D4 00000000 00000010   *......aM........*
+  0000000000008CE0    +160  00000000 0000000F 00000000 00000000   *................*
+  0000000000008CF0    +170  00000000 00000000 00000000 00000000   *................*
+  0000000000008D00    +180  E3C7D4E3 F0F0F0F0 40C4A895 81948983   *TGMT0000 Dynamic*
+  0000000000008D10    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  0000000000008D20    +1A0  03000000 0000074C 00000000 0000003F   *.......<........*
+  0000000000008D30    +1B0  00000000 0000074C 00000000 00000000   *.......<........*
+  0000000000008D40    +1C0  00000000 00000000 00000000 00000000   *................*
+  0000000000008D50    +1D0  0000000000008D50 TO 0000000000008D6F SAME AS ABOVE
+  0000000000008D70    +1F0  0000749F 00008880 000088A2 000088B6   *......h...hs..h.*
+  0000000000008D80    +200  F2F0F0F3 F0F3F1F5 F0F5F0F0 F0F00103   *20030315050000..*
+  0000000000008D90    +210  074C0103 074C0847 4C050000 0FF2F0F0   *.<...<..<....200*
+  0000000000008DA0    +220  F361F0F3 61F1F500 00000000 00000000   *3/03/15.........*
+  0000000000008DB0    +230  00000000 00000000 00000000 00000000   *................*
+  0000000000008DC0    +240  0000000000008DC0 TO 000000000000950F SAME AS ABOVE
+  0000000000009510    +990  00009510 00000000                     *..n.....        *
Save Area Storage For CSECT: TCAL0000
Begin Dump of LCA Storage Area
   0000000000009518          D3C3C140 C6F4E2C1 00000000 3270728E   *LCA F4SA........*
+  0000000000009528     +10  00000000 327073B0 00000000 0004AB38   *................*
+  0000000000009538     +20  00000000 00049B60 00000000 32707292   *......-........k*
+  0000000000009548     +30  00000000 00000000 00000000 3271A6A8   *..............wy*
+  0000000000009558     +40  00000000 3271A7D4 00000000 00000000   *......xM........*
+  0000000000009568     +50  00000000 32708FB8 00000000 32707FB8   *................*
+  0000000000009578     +60  00000000 0004AB38 00000000 00049B38   *................*
+  0000000000009588     +70  00000000 32706BE8 00000000 32705BE8   *......,Y......$Y*
+  0000000000009598     +80  00000000 00008B80 00000000 00009728   *..............p.*
+  00000000000095A8     +90  00000000 00000000 00000000 00000000   *................*
+  00000000000095B8     +A0  00000000000095B8 TO 00000000000095E7 SAME AS ABOVE
+  00000000000095E8     +D0  00000000 00000000 00009728 005E6968   *..........p.....*
+  00000000000095F8     +E0  0003F8D8 00000000 00000000 32705BD8   *..8Q..........$Q*
+  0000000000009608     +F0  000017D8 00009518 00000210 00000000   *...Q..n.........*
+  0000000000009618    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
+  0000000000009628    +110  00050000 00049B38 000004C4 00000000   *...........D....*
+  0000000000009638    +120  00000000 0000000A 07850001 80000000   *.........e......*
+  0000000000009648    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000009658    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000009668    +150  00000000 32704CAE 00000000 32705BD8   *......<.......$Q*
+  0000000000009678    +160  00000000 0004AB38 00000000 00049B60   *...............-*
+  0000000000009688    +170  00000000 00000000 00000000 00000000   *................*
+  0000000000009698    +180  E3C3C1D3 F0F0F0F0 40C4A895 81948983   *TCAL0000 Dynamic*
+  00000000000096A8    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  00000000000096B8    +1A0  00000000 00000000 00000000 00000000   *................*
+  00000000000096C8    +1B0  00000000000096C8 TO 0000000000009717 SAME AS ABOVE
+  0000000000009718    +200  00000000 00000000 00009720 00000000   *..........p.....*
Save Area Storage For CSECT: TVCT0000
Begin Dump of LCA Storage Area
   0000000000009728          D3C3C140 C6F4E2C1 00000000 32707639   *LCA F4SA........*
+  0000000000009738     +10  00000000 32755291 00000000 07850001   *.......j.....e..*
+  0000000000009748     +20  00000000 80000000 00000000 00000000   *................*
+  0000000000009758     +30  00000000 00000000 00000000 00000000   *................*
+  0000000000009768     +40  0000000000009768 TO 0000000000009797 SAME AS ABOVE
+  0000000000009798     +70  00000000 00000000 00000000 327073C0   *................*
+  00000000000097A8     +80  00000000 00009518 00000000 00050000   *......n.........*
+  00000000000097B8     +90  00000000 00000000 00000000 00000000   *................*
+  00000000000097C8     +A0  00000000000097C8 TO 00000000000097F7 SAME AS ABOVE
+  00000000000097F8     +D0  00000000 00000000 00009988 005E6968   *..........rh....*
+  0000000000009808     +E0  0003F678 00000000 00000000 327073B0   *..6.............*
+  0000000000009818     +F0  000005E0 00009728 00000260 00000000   *......p....-....*
+  0000000000009828    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
+  0000000000009838    +110  00050000 00049B38 000004C4 00000000   *...........D....*
+  0000000000009848    +120  00000000 0000000A 07850001 80000000   *.........e......*
+  0000000000009858    +130  00000000 00000000 00000000 00000000   *................*
+  0000000000009868    +140  00000000 00000000 00000000 00000000   *................*
+  0000000000009878    +150  00000000 3270728E 00000000 327073B0   *................*
+  0000000000009888    +160  00000000 0004AB38 00000000 00049B60   *...............-*
+  0000000000009898    +170  00000000 00000000 00000000 00000000   *................*
+  00000000000098A8    +180  E3E5C3E3 F0F0F0F0 40C4A895 81948983   *TVCT0000 Dynamic*
+  00000000000098B8    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
+  00000000000098C8    +1A0  00000000 00000000 00000000 00000000   *................*
+  00000000000098D8    +1B0  00000000000098D8 TO 0000000000009917 SAME AS ABOVE
+  0000000000009918    +1F0  E4E2C5D9 F0F0F140 00000000 7FFFFFA0   *USER001 ........*
+  0000000000009928    +200  00000000 00000000 00000000 00000000   *................*
+  0000000000009938    +210  0000000000009938 TO 0000000000009977 SAME AS ABOVE
+  0000000000009978    +250  00000000 00000000 00009980 00000000   *..........r.....*
```

(Report truncated for this Display)

In the above example of the abend report it shows that the user program abended with a S0C4 due to a page translation exception. In addition, based on the abend code a list of possible causes for the abend is displayed. There could many other reasons with the ones listed being the most common.

```
Completion Code      System = 0C4      Reason = 00000011
Abend caused by program interrupt from page-translation exception: (PIC 11)
 The following are some possible reasons for abend:
 -Note: For this abend the PSW points to the abending instruction and not the next instruction.
 -An invalid address in instruction resulted in a reference to storage not get-mained.
 -A previously valid address in instruction referenced get-main storage area already released.
 -An address in instruction needed Amode-31 to reference 31-bit storage when program was in Amode-24.
 -An invalid length in instruction resulted in a fetch or store to storage not in your address space.
 -BCT count set incorrectly to not be positive results in loop being enforced on a negative count.
 -An LA instead of L or vice versa resulted in branch, base, or index register with invalid pointer.
 -Program loop using bad index to access storage eventually referenced an area not get-mained.
 -You passed an invalid parameter list address to a called routine and routine encountered S0C4.
 -Attempt was made to close a data set twice without an intervening open.
 -Attempt was made to read or write to a data set where the DCB was not opened.
 -Attempt to close data set where DCB in get-main area and DCB get-main area released before close.
```

The LOGGRASM abend report process will analyze the abend to determine which instruction caused the abend. The report shows that program USRTEST7 attempted to process an Execute Relative Long instruction located at address 32707786 which resulted in a S0C4-11 abend. To aid in diagnosis the attributes and characteristics of the abending instruction are displayed including the disassemble of the machine instruction format of the EXRL instruction into its corresponding Assembler language source statement.

```
-Abending Instruction Summary-

 Instruction at Address..: 0000000032707782
 Machine Instruction.....: C6F0 0000 1423
 Instruction Op Code.....: C60
 Instruction Mnemonic....: EXRL     R15,*+10310
 Instruction Description.: Execute Relative Long
 Instruction Format......: R1,RI2
 Instruction Type........: RIL-b
 Instruction Attributes..: Does not set Condition Code
 Instruction Facility....: Execute-extension facility
```

In this example decimal 10,310 = x’2846’, and when divided by 2 = x’1423’ where x’1423’ represents the relative displacement in the number of halfwords from the EXRL instruction that the ICMY instruction resides. .

32707782 + 2846 = 32709FC8 = location of target instruction ICMY.

On an Execute instruction the LOGGRASM abend report process will further analyze the abend, and attempt to determine the instruction to be executed by the execute instruction. The abend report will display the attributes and characteristics of the instruction which was the target of the execute as a further diagnostic aid. The abend report shows the target instruction to be executed by the EXRL instruction was Insert Characters under Mask Long at storage address 32709FC8.

```
-Executed Target Instruction Summary-

 Instruction at Address..: 0000000032709FC8
 Target Instruction......: EBB0 3030 0081
 Instruction Op Code.....: EB81
 Instruction Mnemonic....: ICMY     R11,0,48(R3)
 Instruction Description.: Insert Characters under Mask Long (Low, 20-bit Long Displacement)
 Instruction Format......: R1,M3,D2(B2)
 Instruction Type........: RSY-b
 Instruction Attributes..: Sets Condition Code
 Instruction Facility....: Long-displacement facility
```

As a further diagnostic aid, the LOGGRASM abend report will disassemble the instructions around the area of the abend

```
-Instructions in Vicinity of Abend-

 Hex      Hex Machine
 Offset   Instruction      Assembler Source Statement         Attributes
 ------   ---------------  -----------------------------------  ----------------------
    -6E   0A08             SVC      8    (LOAD)
    -6C   12FF             LTR      R15,R15                     Sets Condition Code
    -6A   A774 003A        BRC      7,*+116
    -66   E3F0 C598 0004   LG       R15,1432(R0,R12)
    -60   B904 00E5        LGR      R14,R5
    -5C   E320 E000 0090   LLGC     R2,0(R0,R14)
    -56   B90B 00FF        SLGR     R15,R15                     Sets Condition Code
    -52   1814             LR       R1,R4
    -50   18F0             LR       R15,R0
    -4E   E320 C5B0 0017   LLGT     R2,1456(R0,R12)
    -48   E3F0 C5C8 0090   LLGC     R15,1480(R0,R12)
    -42   1816             LR       R1,R6
    -40   5860 C5B4        L        R6,1460(R0,R12)
    -3C   B904 00F0        LGR      R15,R0
    -38   E310 C5B8 0017   LLGT     R1,1464(R0,R12)
    -32   4160 C3C2        LA       R6,962(R0,R12)              Modal Instruction
    -2E   5830 C5BC        L        R3,1468(R0,R12)
    -2A   E3F0 C5C9 0090   LLGC     R15,1481(R0,R12)
    -24   4140 C3CC        LA       R4,972(R0,R12)              Modal Instruction
    -20   ECBC FF9A 817C   CGIJNH   R11,-127,*-204
    -1A   C25E 0000 0FFE   CLGFI    R5,X'00000FFE'              Sets Condition Code
    -14   A784 000A        BRC      8,*+20
    -10   5850 4000        L        R5,0(R0,R4)
     -C   E330 D1F8 0004   LG       R3,504(R0,R13)
     -6   E320 C5C0 0017   LLGT     R2,1472(R0,R12)
   ===>   C6F0 0000 1423   EXRL     R15,*+10310
     +6   B917 00F0        LLGTR    R15,R0
     +A   E330 0010 0017   LLGT     R3,16(R0,R0)
    +10   E330 3000 0017   LLGT     R3,0(R0,R3)
    +16   EBE1 D150 0024   STMG     R14,R1,336(R13)
    +1C   B98D 0001        EPSW     R0,R1
    +20   9001 D128        STM      R0,R1,296(R13)
```

Referring to the abend report example produced by LOGGRASM, the abend occurred when the ICMY instruction made an attempt to fetch 3 bytes of data residing in a field at a storage area that was addressed by general register 3, and then load the 3 bytes of data from storage into general register 11. The three byte positions to be used in register 11 would be based on the bit mask in R15 of the execute instruction EXRL. The contents of general register 15 (X’7’ = B’0111’) as the first operand field R_1 in the EXRL instruction would be used to OR the third operand field M_3 in the ICMY instruction. Remember that the OR is done internally by the hardware, and neither the first operand field R_1 in the EXRL nor the executed instruction ICMY are actually altered. Also, by hardware convention the abend which occurred is reported on the EXRL instruction and not on the target instruction ICMY.

Referring to the Executed Instructon Summary and the General Purpose Register Summary portions of the abend report, it shows general register 3 (R3) was used as the second operand base register field (B_2) for the ICMY instruction to be executed by the EXRL, and the first operand register field (R_1) in the ICMY instruction was general register 11 (R11). The issue was the contents of base register 3 plus the displacement D_2 (48 bytes) resulted in an invalid storage address which was used by the ICMY instruction in an attempt to fetch data from a storage area not available to the program resulting the program terminating abnormally in a S0C4 abend with reason code 11.

```
-Executed Target Instruction Summary-

Instruction at Address..: 0000000032709FC8
Target Instruction......: EBB0 3030 0081
Instruction Op Code.....: EB81
Instruction Mnemonic....: ICMY      R11,0,48(R3)
Instruction Description.: Insert Characters under Mask Long (Low, 20-bit Long Displacement)
Instruction Format......: R1,M3,D2(B2)
Instruction Type........: RSY-b
Instruction Attributes..: Sets Condition Code
Instruction Facility....: Long-displacement facility



-General Purpose Register Summary-

   R0:  00000000B2A86AB8 Load Module/Csect  IGGCSI00+(000000)  Decimal=849,898,168 (31)
   R1:  000000000004A100 (Invalid Storage)  0C4 if referenced  Decimal=303,360
   R2:  000000003271A7D4 Load Module/Csect  SVLTEST7+(01A244)  Decimal=846,309,332 (31)
 B-R3:  000000007FFFFFA0 (Invalid Storage)  0C4 if referenced  Decimal=2,147,483,552
   R4:  0000000032707790 Load Module/Csect  SVLTEST7+(007200)  Decimal=846,231,440 (31)
   R5:  00000000E3300010 (Invalid Storage)  0C4 if referenced  Decimal=-483,393,520
   R6:  0000000032707786 Load Module/Csect  SVLTEST7+(0071F6)  Decimal=846,231,430 (31)
   R7:  0000000032708FB8 Load Module/Csect  SVLTEST7+(008A28)  Decimal=846,237,624 (31)
   R8:  0000000032707FB8 Load Module/Csect  SVLTEST7+(007A28)  Decimal=846,233,528 (31)
   R9:  000000000004AB38 (Invalid Storage)  0C4 if referenced  Decimal=305,976
   R10: 0000000000049B38 Allocated storage  Subpool=003 Key=8  TCB=005E6968 Addressable storage is 1,224 bytes
 D-R11: 0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
   R12: 00000000327073D0 Load Module/Csect  SVLTEST7+(006E40)  Decimal=846,230,480 (31)
   R13: 0000000000009728 Allocated storage  Subpool=003 Key=8  TCB=005E6968 Addressable storage is 260,312 bytes
   R14: 000000003270784E Load Module/Csect  SVLTEST7+(0072BE)  Decimal=846,231,630 (31)
   R15: 0000000000000007 Allocated storage  (PSA+PSAE+PSAX)    Decimal=7 Addressable storage is 8,185 bytes

   '?-Rx:' Used as a designated (D), index (X), base (B), odd (O) of an even/odd pair,
           or hardware implied (H) register in abending instruction ICMY
    Note:  Registers flagged are for the target instruction of the Execute instruction.
```

What happened was an address computed by adding a displacement to the contents in register 3 resulted in a reference to a storage area which at the time was not within the active working set in memory for the user program's address space. A page translation exception occurred causing a normal page fault which is a common occurance which usually gets resolved, and you continue normal execution. In this instance the hardware does not reset the PSW to the next sequential instruction because a PIC 11 is recognized by the hardware before the instruction address in the PSW is set to the next sequential instruction. Once the page fault is resolved, the instruction that caused the PIC 11 then has to be repeated now that the missing page is available, and normal execution continues with the next sequential instruction.

However, because the page fault in this instance could not be resolved when the displacement was added to what was basically a bad storage address in register 3 which was then used on the attempted fetch by the ICMY instruction, you received the S0C4 which caused you to abend with the PSW pointing to the abending instruction, not the next instruction as occurs in most abends.

Probable causes would be register 3 was stepped on before it was referenced as a base register, or register 3 had residual data left over from previous use and was never explicity initialized or refreshed with a valid address, or there was an an address field with corrupted data from an overlay, and the contents of which were loaded in register 3, or there was a program logic error where an address field was not initialized with a valid address prior to its contents being loaded into register 3 for later use as a base register by the ICMY instruction. In this instance, the disassemble report showsRegister 3 being loaded prior to execution of the EXRL instruction.

```
    -10   5850 4000         L        R5,0(R0,R4)
     -C   E330 D1F8 0004    LG       R3,504(R0,R13)
     -6   E320 C5C0 0017    LLGT     R2,1472(R0,R12)
   ===>   C6F0 0000 1423    EXRL     R15,*+10310
     +6   B917 00F0         LLGTR    R15,R0
     +A   E330 0010 0017    LLGT     R3,16(R0,R0)
    +10   E330 3000 0017    LLGT     R3,0(R0,R3)
```

As a further diagnostic aid the abend report will display a summary showing the program name and Csect name where the abend occurred. The breaking event address is also displayed showing the address of the last successful 'branch from' instruction. As a further diagnostic aid the abend report will attempt to resolve the Breaking Event Address to display the Load Module/Csect name and the offset within the module or Csect where the last successful 'branch from' instruction occurred.

```
-Abend Summary-

 Abend Code.................: System=0C4   Reason=00000011
 Abend Location.............: USRTEST7.TVCT0000+(0003D2)  20110920 15.04
 Program Status Word........: 078D2001 B2707782  (EC)
 zArch Program Status Word..: 07852001 80000000 00000000 32707782
 Data Around PSW............: 3270777C ==> E320C5C0 0017C6F0 00001423
 Breaking Event Address.....: 0000000000FEEA1A
 Abending Instruction Length: 6 bytes
 CPU State..................: Problem State (Only unprivileged instructions valid)
 Program Status Word Key....: 8  (The access key used for storage references by the CPU)
 Addressing Mode............: 64 (Program was in 64-bit addressing mode at time of abend)
 Condition Code in PSW......: 2 (Result of last executed instruction which set Cond Code in PSW)
 Address Space Control Mode.: Primary-space
```

The abend report will also produce a save area trace which you can reference. Then you should obtain the source listings for the Assembler Csects listed in the calling chain of the save area trace. Ensure the Assembler listing date matches with the load module date and times listed for the Csects displayed in the save area chain if a time-stamp is available.

```
-Save Area Trace-
 Proceeding Backward From Last Save Area To First Save Area:

 Called Program Entry Point..: USRTEST7.TVCT0000+(000000)  20110920 15.04
       TVCT0000 Csect Address: 00000000327073B0
       TVCT0000 Csect Length.: 000005E0
 Called Program Returns To...: USRTEST7.TCAL0000+(0016B6)  20110920 15.04
 Calling Program Save Area...: 0000000000009518
 Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
   0-3   000000000004AB38  0000000000049B60  0000000032707292  0000000000000000
   4-7   000000003271A6A8  000000003271A7D4  0000000000000000  0000000032708FB8
   8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000032706BE8
   12-15 0000000032705BE8  0000000000009518  000000003270728E  00000000327073B0

 Called Program Entry Point..: USRTEST7.TCAL0000+(000000)  20110920 15.04
       TCAL0000 Csect Address: 0000000032705BD8
       TCAL0000 Csect Length.: 000017D8
 Called Program Returns To...: USRTEST7.TGMT0000+(00062E)  20110920 15.04
 Calling Program Save Area...: 0000000000008B80
 Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
   0-3   000000000004AB38  0000000000049B60  0000000000000003  00000000000088A2
   4-7   00000000000088B6  00000000000003E7  0000000000000000  0000000032708FB8
   8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000000000000
   12-15 0000000032704690  0000000000008B80  0000000032704CAE  0000000032705BD8

 Called Program Entry Point..: USRTEST7.TGMT0000+(000000)  20110920 15.04
       TGMT0000 Csect Address: 0000000032704680
       TGMT0000 Csect Length.: 00000930
 Called Program Returns To...: USRTEST7.TRDT0000+(00089E)  20110920 15.04
 Calling Program Save Area...: 0000000000008608
 Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
   0-3   0000000000008880  0000000000049B60  00000000327081E0  00000000000088DE
   4-7   0000000000007808  0000000000007861  00000000B2703A98  0000000032708FB8
   8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000032704308
   12-15 0000000032703308  0000000000008608  00000000B2703B96  0000000032704680

 Called Program Entry Point..: USRTEST7.TRDT0000+(000000)  20110920 15.04
       TRDT0000 Csect Address: 00000000327032F8
       TRDT0000 Csect Length.: 00001388
 Called Program Returns To...: USRTEST7.USRTEST7+(001812)  20110920 15.04
 Calling Program Save Area...: 0000000000007000
 Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
   0-3   0000000000007828  0000000000049B60  0000000800000014  00000000005FFE40
   4-7   0000000000007FF4  0000000000000189  FFFFFFFFFFFFFE78  0000000032708FB8
   8-11  0000000032707FB8  000000000004AB38  0000000000049B38  000000003270158C
   12-15 000000003270058C  0000000000007000  00000000B2701D92  00000000327032F8

 Called Program Entry Point..: USRTEST7.USRTEST7+(000000)  20110920 15.04
      USRTEST7 Csect Address.: 0000000032700580
      USRTEST7 Csect Length..: 00001FC0
 Called Program Returns To...: Program was entered via LINK
 Calling Program Save Area...: 0000000000006F60
 Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
   0-3   00000006FD000008  0000000000006FF8  0000000000000040  00000000005D1D84
   4-7   00000000005D1D60  00000000005E6E88  00000000005B8FE0  00000000FD000000
   8-11  00000000005FCC30  00000000005E6B30  0000000000000000  00000000005E6E88
   12-15 000000008370AD3A  0000000000006F60  0000000080FDE148  00000000B2700580
```

With the LOGGRASM abend report showing the abend code, the location of the error, the instruction causing the error, the disassemble of the instructions in the vicinity of the abend, the register summary, the save area trace, and the dump of the save area storage for each Cesect in your program, you can check your Assembler listing and walk your source code starting from the the most recent called program or Csect listed at the beginning of the save area trace.

Try looking for references to register 3 in your Assembler source code, or the lack of any reference to register 3 to determine the cause of the problem of how did that register end up containing an invalid value when used as a base register for the ICMY instruction. In this instance, the disassemble report shows Register 3 being loaded prior to execution of the EXRL instruction.

R13 (x'D') contains address 9728 + 504 bytes (x'1F8') as a displacement in the LG instruction resolves to address 9920 which points to an 8-byte storage area that contains the data '000000007FFFFFA0' which was loaded into R3. By hardware convention the contents of R0 are not used when R0 is designated as an index register.

```
     -10   5850 4000         L        R5,0(R0,R4)
      -C   E330 D1F8 0004    LG       R3,504(R0,R13)
      -6   E320 C5C0 0017    LLGT     R2,1472(R0,R12)
    ===>   C6F0 0000 1423    EXRL     R15,*+10310
      +6   B917 00F0         LLGTR    R15,R0
      +A   E330 0010 0017    LLGT     R3,16(R0,R0)
     +10   E330 3000 0017    LLGT     R3,0(R0,R3)
```

```
General Purpose Registers R0-R15 at Entry to Abend:
   0-3   00000000B2A74AB8  000000000004A100  000000003271A7D4  000000007FFFFFA0
   4-7   000000003270778C  00000000E3300010  0000000032707782  00000000327088FB8
   8-11  0000000032707FB8  000000000004AB38  0000000000049B38  0000000000000000
   12-15 00000000327073C0  0000000000009728  000000003270784A  0000000000000007

Save Area Storage For CSECT: TVCT0000
 Begin Dump of LCA Storage Area
    0000000000009728          D3C3C140 C6F4E2C1 00000000 32707639   *LCA F4SA........*
 +  0000000000009738     +10  00000000 32755291 00000000 07850001   *.......j.....e..*
 +  0000000000009748     +20  00000000 80000000 00000000 00000000   *................*
 +  0000000000009758     +30  00000000 00000000 00000000 00000000   *................*
 +  0000000000009768     +40  0000000000009768 TO 0000000000009797 SAME AS ABOVE
 +  0000000000009798     +70  00000000 00000000 00000000 327073C0   *................*
 +  00000000000097A8     +80  00000000 00009518 00000000 00050000   *......n.........*
 +  00000000000097B8     +90  00000000 00000000 00000000 00000000   *................*
 +  00000000000097C8     +A0  00000000000097C8 TO 00000000000097F7 SAME AS ABOVE
 +  00000000000097F8     +D0  00000000 00000000 00009988 005E6968   *..........rh....*
 +  0000000000009808     +E0  0003F678 00000000 00000000 327073B0   *..6.............*
 +  0000000000009818     +F0  000005E0 00009728 00000260 00000000   *......p....-....*
 +  0000000000009828    +100  0CEF0CEF 0CEF0700 07000700 40000000   *............ ...*
 +  0000000000009838    +110  00050000 00049B38 000004C4 00000000   *...........D....*
 +  0000000000009848    +120  00000000 0000000A 07850001 80000000   *.........e......*
 +  0000000000009858    +130  00000000 00000000 00000000 00000000   *................*
 +  0000000000009868    +140  00000000 00000000 00000000 00000000   *................*
 +  0000000000009878    +150  00000000 3270728E 00000000 327073B0   *................*
 +  0000000000009888    +160  00000000 0004AB38 00000000 00049B60   *...............-*
 +  0000000000009898    +170  00000000 00000000 00000000 00000000   *................*
 +  00000000000098A8    +180  E3E5C3E3 F0F0F0F0 40C4A895 81948983   *TVCT0000 Dynamic*
 +  00000000000098B8    +190  40C19985 814DD3C3 C15D4040 40404040   * Area(LCA)      *
 +  00000000000098C8    +1A0  00000000 00000000 00000000 00000000   *................*
 +  00000000000098D8    +1B0  00000000000098D8 TO 0000000000009917 SAME AS ABOVE
 +  0000000000009918    +1F0  E4E2C5D9 F0F0F140 00000000 7FFFFFA0   *USER001 ........*
 +  0000000000009928    +200  00000000 00000000 00000000 00000000   *................*
 +  0000000000009938    +210  0000000000009938 TO 0000000000009977 SAME AS ABOVE
 +  0000000000009978    +250  00000000 00000000 00009980 00000000   *..........r.....*
```

You could also try running the job again by placing #LGPOINT REG= commands in the source code path to display register values, or use #LGPOINT SHOW= to display storage areas prior to the abend, or use a #LGPOINT WATCH= in various parts of your program to watch specific storage areas to indicate when or where in your program a storage area was altered prior to the abend. These Logpoints as a debugging aid will help you much more quickly to narrow down the area in your code where the logic error occured that caused the bad address to show up in register R3. In this example you can go back to the assembler source and add #LGPOINT entries before the abend for the Csect TVCT0000 in error .

You can use filtering to limit the LOGGASM output to the programs or Csects in error. With filtering you can invoke only those Logpoints you want to see and ignore anything else. This will avoid being inundated with possibly thousands of lines of log record data that you don’t need. With filtering you can specify the program or Csect by name where Logpoints are coded, and are to be active. Or you can specify which particular Logpoint by name that you want active within an entire program by using filtering.

For example: To filter by Csect name.

```
//LGRSYSIN  DD *
          LOGEVENT PROGRM,NAME=(TVCT*)
```

The following is another example of a diagnostic output report written to //LGRECOUT upon a user program abending with a S0C4 due to improper parms passed to CLOSE. It shows the instruction which abended in the Close, where the Close was coded in the user program, and the breaking event address. The instructions around each event are displayed if Logger Services can disassemble the area in the load module preceding and after the instruction event (e.g., the ‘==>’). However, if Logger Services is unable to successfully disassemble a chuck of storage into a valid sequence of instructions surrounding the event, then the ‘Instruction Summary’ and ‘Instructions in Vicinity’ portions of the report for that event will be bypassed and may not show up at all in the abend report.

```
                         ******************************
                         *      Program Abended       *
                         ******************************
Completion Code      System = 0C4      Reason = 00000011
Abend caused by program interrupt from page-translation exception: (PIC 11)
 The following are some possible reasons for abend:
 -Note: For this abend the PSW points to the abending instruction and not the next instruction.
 -An invalid address in instruction resulted in a reference to storage not get-mained.
 -A previously valid address in instruction referenced get-main storage area already released.
 -An address in instruction needed Amode-31 to reference 31-bit storage when program was in Amode-24.
 -An invalid length in instruction resulted in a fetch or store to storage not in your address space.
 -BCT count set incorrectly to not be positive results in loop being enforced on a negative count.
 -An LA instead of L or vice versa resulted in branch, base, or index register with invalid pointer.
 -Program loop using bad index to access storage eventually referenced an area not get-mained.
 -Program loop adding to address to access storage eventually referenced an area not in your program.
 -You passed an invalid parameter list address to a called routine and routine encountered S0C4.
 -Attempt was made to close a data set twice without an intervening open.
 -Attempt was made to read or write to a data set where the DCB was not opened.
 -Attempt to close data set where DCB in get-main area and DCB get-main area released before close.
 -You took a wild-branch to an invalid address, check for Breaking Event Address if one is available.

-Abend Summary-

Abend Code.................: System=0C4   Reason=00000011
Abend Location.............: IGC00020+(000B3C)
Program Status Word........: 078C0000 00E0AB3C  (EC)
zArch Program Status Word..: 07840000 00000000 00000000 00E0AB3C
Data Around PSW............: 00E0AB36 ==> 1B224111 0000BF27 1001A784
Breaking Event Address.....: 0000000000E0AAF8 at IGC00020+(000AF8)
Abending Instruction Length: 4 bytes
CPU State..................: Supervisor State (All instructions valid)
Program Status Word Key....: 8  (The access key used for storage references by the CPU)
Addressing Mode............: 24 (Program was in 24-bit addressing mode at time of abend)
Condition Code in PSW......: 0  (Result of last executed instruction which set Cond Code in PSW)
Address Space Control Mode.: Primary-space

*** Instruction Which Caused the Abend ***

-Instruction Summary-

Instruction at Address..: 0000000000E0AB3C
Machine Instruction.....: BF27 1001
Instruction Op Code.....: BF
Instruction Mnemonic....: ICM      R2,B'0111',1(R1)
Instruction Description.: Insert Characters under Mask (low)
Instruction Format......: R1,M3,D2(B2)
Instruction Type........: RS-b
Instruction Attributes..: Sets Condition Code
Instruction Facility....: None, zArchitecture base

-Instructions in Vicinity-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement           Attributes
```

```
 ------   ---------------  -----------------------------------  --------------------------------------------
-----
    -7E   1BFF             SR       R15,R15                     Sets Condition Code
    -7C   1BCC             SR       R12,R12                     Sets Condition Code
    -7A   41F0 0008        LA       R15,8(R0,R0)                Modal Instruction
    -76   58C0 3060        L        R12,96(R0,R3)
    -72   181C             LR       R1,R12
    -70   8910 0008        SLL      R1,8
    -6C   8810 0008        SRL      R1,8
    -68   4100 0084        LA       R0,132(R0,R0)               Modal Instruction
    -64   8900 0018        SLL      R0,24
    -60   1610             OR       R1,R0                       Sets Condition Code
    -5E   0A0D             SVC      13    (ABEND) Type=4        Calls module IEAVTRT2, entry point IGC0101C
    -5C   58D0 0010        L        R13,16(R0,R0)
    -58   58D0 D000        L        R13,0(R0,R13)
    -54   58D0 D004        L        R13,4(R0,R13)
    -50   B20A 0000        SPKA     0(R0)                       Semiprivileged instruction
    -4C   900B 5060        STM      R0,R11,96(R5)
    -48   1861             LR       R6,R1
    -46   1211             LTR      R1,R1                       Sets Condition Code
    -44   A774 0017        BRC      7,*+46
    -40   010D             SAM31
    -3E   1870             LR       R7,R0
    -3C   5820 D000        L        R2,0(R0,R13)
    -38   BF27 201D        ICM      R2,B'0111',29(R2)           Sets Condition Code
    -34   4320 2011        IC       R2,17(R0,R2)
    -30   B20A 2000        SPKA     0(R2)                       Semiprivileged instruction
    -2C   1B22             SR       R2,R2                       Sets Condition Code
    -2A   5827 0004        L        R2,4(R7,R0)
    -26   1222             LTR      R2,R2                       Sets Condition Code
    -24   A784 00D3        BRC      8,*+422
    -20   1882             LR       R8,R2
    -1E   1812             LR       R1,R2
    -1C   1B22             SR       R2,R2                       Sets Condition Code
    -1A   A7F4 0029        BRC      15,*+82
    -16   5820 D000        L        R2,0(R0,R13)
    -12   BF27 201D        ICM      R2,B'0111',29(R2)           Sets Condition Code
     -E   4320 2011        IC       R2,17(R0,R2)
     -A   B20A 2000        SPKA     0(R2)                       Semiprivileged instruction
     -6   1B22             SR       R2,R2                       Sets Condition Code
     -4   4111 0000        LA       R1,0(R1,R0)                 Modal Instruction
   ===>   BF27 1001        ICM      R2,B'0111',1(R1)            Sets Condition Code
     +4   A784 0008        BRC      8,*+16
     +8   1871             LR       R7,R1
     +A   1882             LR       R8,R2
     +C   1812             LR       R1,R2
     +E   1B22             SR       R2,R2                       Sets Condition Code
    +10   A7F4 0014        BRC      15,*+40

 General Purpose Registers R0-R15 at Entry to Abend:
   0-3   00000000B2A6DAB8  000000000071AA8C  0000000000000000  0000000000E0AE58
   4-7   00000008005E6858  00000000005FD558  000000003271AA8C  0000000000FB9A80
   8-11  0000000000000000  00000000016BAF20  00000000005FD618  000000007FF880A0
   12-15 0000000001409998  00000000005E6858  0000000000E0A030  0000000000000007
 Access Registers:
   0-3   00000000  01010039  00000000  00000000
   4-7   00000000  00000000  00000000  00000000
   8-11  00000000  00000000  00000000  00000000
   12-15 00000000  00000000  00000000  00000C05

 -General Purpose Register Summary-

   R0:  00000000B2A6DAB8 Load Module/Csect  IGGCSI00+(000000)  Decimal=849,795,768 (31)
 B-R1:  000000000071AA8C (Invalid Storage)  0C4 if referenced  Decimal=7,449,228
 D-R2:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
   R3:  0000000000E0AE58 Load Module/Csect  IGC00020+(000E58)  Decimal=14,724,696 (31)
   R4:  00000008005E6858 Allocated storage  Memory object      Decimal=34,365,925,464
   R5:  00000000005FD558 Allocated storage  Subpool=255 Key=0  Addressable storage is 10,920 bytes
   R6:  000000003271AA8C Load Module/Csect  USRTEST7+(01A88C)  Decimal=846,310,028 (31)
   R7:  0000000000FB9A80 Allocated storage  Subpool=245 Key=0  Addressable storage is 9,600 bytes
   R8:  0000000000000000 Allocated storage  (PSA+PSAE+PSAX)    Decimal=0 Addressable storage is 8,192 bytes
```

```
  R9:  00000000016BAF20
  R10: 00000000005FD618 Allocated storage  Subpool=255 Key=0  Addressable storage is 10,728 bytes
  R11: 000000007FF880A0 Allocated storage  Subpool=255 Key=0  Addressable storage is 3,936 bytes
  R12: 0000000001409998
  R13: 00000000005E6858 Allocated storage  Subpool=255 Key=0  Addressable storage is 1,960 bytes
  R14: 0000000000E0A030 Load Module/Csect  IGC00020+(000030)  Decimal=14,721,072 (31)
  R15: 0000000000000007 Allocated storage  (PSA+PSAE+PSAX)    Decimal=7 Addressable storage is 8,185 bytes

  '?-Rx:' Used as a designated (D), index (X), base (B), odd (O) of an even/odd pair,
          or hardware implied (H) register in abending instruction ICM

*** Last Interrupt Event in Program Before Abend ***

Interrupt Event Location: USRTEST7.TVCT0000+(0003EA)
Interrupt Event Address.: 000000003270799A

-Instruction Summary-

Instruction at Address..: 000000003270799A
Machine Instruction.....: 0A14
Instruction Op Code.....: 0A
Instruction Mnemonic....: SVC      20    (CLOSE) Type=4        Calls module IGC00020
Instruction Description.: Supervisor Call
Instruction Format......: SVC nn
Instruction Type........: I
Instruction Attributes..: Does not set Condition Code
Instruction Facility....: None, zArchitecture base

-Instructions in Vicinity-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement           Attributes
------   ---------------  ----------------------------------   ---------------------------------------------
   -66   0A08             SVC      8    (LOAD) Type=2          Calls module CSVLOAD, entry point IGC008
   -64   12FF             LTR      R15,R15                     Sets Condition Code
   -62   A774 0007        BRC      7,*+14
   -5E   E3F0 C560 0004   LG       R15,1376(R0,R12)
   -58   B904 00E5        LGR      R14,R5
   -54   E320 E000 0090   LLGC     R2,0(R0,R14)
   -4E   B90B 00FF        SLGR     R15,R15                     Sets Condition Code
   -4A   1814             LR       R1,R4
   -48   18F0             LR       R15,R0
   -46   E320 C584 0017   LLGT     R2,1412(R0,R12)
   -40   E3F0 C59C 0090   LLGC     R15,1436(R0,R12)
   -3A   1816             LR       R1,R6
   -38   5860 C588        L        R6,1416(R0,R12)
   -34   B904 00F0        LGR      R15,R0
   -30   E310 C58C 0017   LLGT     R1,1420(R0,R12)
   -2A   4160 C3D8        LA       R6,984(R0,R12)              Modal Instruction
   -26   5830 C590        L        R3,1424(R0,R12)
   -22   E3F0 C59D 0090   LLGC     R15,1437(R0,R12)
   -1C   4140 C3E6        LA       R4,998(R0,R12)              Modal Instruction
   -18   ECBC FF6E 817C   CGIJNH   R11,-127,*-292
   -12   5850 4000        L        R5,0(R0,R4)
    -E   E330 D1F8 0004   LG       R3,504(R0,R13)
    -8   E320 C594 0017   LLGT     R2,1428(R0,R12)
    -2   1812             LR       R1,R2
  ===>   0A14             SVC      20    (CLOSE) Type=4        Calls module IGC00020
    +2   E300 C568 0004   LG       R0,1384(R0,R12)
    +8   B917 00F0        LLGTR    R15,R0
    +C   E330 0010 0017   LLGT     R3,16(R0,R0)
   +12   E330 3000 0017   LLGT     R3,0(R0,R3)
   +18   EBE1 D150 0024   STMG     R14,R1,336(R13)
   +1E   B98D 0001        EPSW     R0,R1

General Purpose Registers at Entry to Event.
Refer to Registers R2-R12 which remain unchanged.
  0-3   000000000B2A6DAB8  0000000000050000  000000003271AA8C  000000007FFFFFA0
  4-7   00000000327079A6  00000000E3300010  0000000032707998  0000000032709240
  8-11  0000000032708240  000000000004AB38  0000000000049B38  0000000000000000
  12-15 00000000327075C0  0000000000050538  0000000032707A28  000000003271AB10
```

```
*** Last Instruction Causing Break in Sequential Execution Before Abend ***

Breaking Event Location: IGC00020+(000AF8)
Breaking Event Address.: 0000000000E0AAF8

-Instruction Summary-

Instruction at Address..: 0000000000E0AAF8
Machine Instruction.....: A774 0017
Instruction Op Code.....: A74
Instruction Mnemonic....: BRC      7,*+46
Instruction Description.: Branch Relative on Condition
Instruction Format......: M1,RI2
Instruction Type........: RI-c
Instruction Attributes..: Does not set Condition Code
Instruction Facility....: None, zArchitecture base

-Instructions in Vicinity-

Hex      Hex Machine
Offset   Instruction      Assembler Source Statement          Attributes
------   ---------------  ----------------------------------  ---------------------------------------------
   -7C   9680 F002        OI       2(R15),X'80'                Sets Condition Code
   -78   D207 6000 3058   MVC      0(8,R6),88(R3)
   -72   9801 41C0        LM       R0,R1,448(R4)
   -6E   989E 41A8        LM       R9,R14,424(R4)
   -6A   5830 3000        L        R3,0(R0,R3)
   -66   47F0 5004        BC       15,4(R0,R5)
   -62   5816 0010        L        R1,16(R6,R0)
   -5E   4110 1020        LA       R1,32(R0,R1)                Modal Instruction
   -5A   4140 0000        LA       R4,0(R0,R0)                 Modal Instruction
   -56   5850 0010        L        R5,16(R0,R0)
   -52   BF57 5111        ICM      R5,B'0111',273(R5)          Sets Condition Code
   -4E   4DE0 5034        BAS      R14,52(R0,R5)               Modal Instruction
   -4A   18F9             LR       R15,R9
   -48   58E0 0010        L        R14,16(R0,R0)
   -44   58E0 E260        L        R14,608(R0,R14)
   -40   B20A 0000        SPKA     0(R0)                       Semiprivileged instruction
   -3C   07FE             BCR      15,R14
   -3A   1BFF             SR       R15,R15                     Sets Condition Code
   -38   1BCC             SR       R12,R12                     Sets Condition Code
   -36   41F0 0008        LA       R15,8(R0,R0)                Modal Instruction
   -32   58C0 3060        L        R12,96(R0,R3)
   -2E   181C             LR       R1,R12
   -2C   8910 0008        SLL      R1,8
   -28   8810 0008        SRL      R1,8
   -24   4100 0084        LA       R0,132(R0,R0)               Modal Instruction
   -20   8900 0018        SLL      R0,24
   -1C   1610             OR       R1,R0                       Sets Condition Code
   -1A   0A0D             SVC      13    (ABEND) Type=4        Calls module IEAVTRT2, entry point IGC0101C
   -18   58D0 0010        L        R13,16(R0,R0)
   -14   58D0 D000        L        R13,0(R0,R13)
   -10   58D0 D004        L        R13,4(R0,R13)
    -C   B20A 0000        SPKA     0(R0)                       Semiprivileged instruction
    -8   900B 5060        STM      R0,R11,96(R5)
    -4   1861             LR       R6,R1
    -2   1211             LTR      R1,R1                       Sets Condition Code
  ===>   A774 0017        BRC      7,*+46
    +4   010D             SAM31
    +6   1870             LR       R7,R0
    +8   5820 D000        L        R2,0(R0,R13)
    +C   BF27 201D        ICM      R2,B'0111',29(R2)           Sets Condition Code
   +10   4320 2011        IC       R2,17(R0,R2)
   +14   B20A 2000        SPKA     0(R2)                       Semiprivileged instruction


-Save Area Trace-
Proceeding Backward From Last Save Area To First Save Area:

Called Program Entry Point..: USRTEST7.TVCT0000+(000000)  20120115 10.06
```

```
      TVCT0000 Csect Address: 00000000327075B0
      TVCT0000 Csect Length.: 000005B0
Called Program Returns To...: USRTEST7.TCAL0000+(00167E)  20120115 10.06
Calling Program Save Area...: 0000000000009518
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   000000000004AB38  0000000000049B60  0000000032707452  0000000000000000
  4-7   000000003271A960  000000003271AA8C  0000000000000000  0000000032709240
  8-11  0000000032708240  000000000004AB38  0000000000049B38  0000000032706DE0
  12-15 0000000032705DE0  0000000000009518  00000000B270744E  00000000327075B0

Called Program Entry Point..: USRTEST7.TCAL0000+(000000)  20120115 10.06
      TCAL0000 Csect Address: 0000000032705DD0
      TCAL0000 Csect Length.: 00001788
Called Program Returns To...: USRTEST7.TGMT0000+(0005E6)  20120115 10.06
Calling Program Save Area...: 0000000000008B80
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   000000000004AB38  0000000000049B60  0000000000000003  00000000000088A2
  4-7   00000000000088B6  00000000000003E7  0000000000000000  0000000032709240
  8-11  0000000032708240  000000000004AB38  0000000000049B38  0000000000000000
  12-15 0000000032704868  0000000000008B80  00000000B2704E3E  0000000032705DD0

Called Program Entry Point..: USRTEST7.TGMT0000+(000000)  20120115 10.06
      TGMT0000 Csect Address: 0000000032704858
      TGMT0000 Csect Length.: 000008C8
Called Program Returns To...: USRTEST7.TRDT0000+(0007EE)  20120115 10.06
Calling Program Save Area...: 0000000000008608
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   0000000000008880  0000000000049B60  0000000032708468  00000000000088DE
  4-7   0000000000007808  0000000000007861  00000000B2703BFC  0000000032709240
  8-11  0000000032708240  000000000004AB38  0000000000049B38  00000000327044B0
  12-15 00000000327034B0  0000000000008608  00000000B2703C8E  0000000032704858

Called Program Entry Point..: USRTEST7.TRDT0000+(000000)  20120115 10.06
      TRDT0000 Csect Address: 00000000327034A0
      TRDT0000 Csect Length.: 000011E0
Called Program Returns To...: USRTEST7.USRTEST7+(0015C2)  20120115 10.06
Calling Program Save Area...: 0000000000007000
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   0000000000007828  0000000000049B60  0000000800000014  00000000005FFE40
  4-7   0000000000007FF4  0000000000000189  FFFFFFFFFFFFFE78  0000000032709240
  8-11  0000000032708240  000000000004AB38  0000000000049B38  000000003270120C
  12-15 000000003270020C  0000000000007000  00000000B27017C2  00000000327034A0

Called Program Entry Point..: USRTEST7.USRTEST7+(000000)  20120115 10.06
     USRTEST7 Csect Address.: 0000000032700200
     USRTEST7 Csect Length..: 00001D20
Called Program Returns To...: Program was entered via LINK
Calling Program Save Area...: 0000000000006F60
Calling Program General Purpose Registers R0-R15 at Entry to Called Program:
  0-3   00000006FD000008  0000000000006FF8  0000000000000040  00000000005D1D84
  4-7   00000000005D1D60  00000000005E6E88  00000000005B8FE0  00000000FD000000
  8-11  00000000005FCE28  00000000005E6B18  0000000000000000  00000000005E6E88
  12-15 0000000083714D7A  0000000000006F60  0000000080FDEBD0  00000000B2700200

-Storage Use at Abend-
 Private Area Storage Allocated:
   <16M: In Use=952K       Unused=5167K       Limit=6120K       HighU=952K
   >16M: In Use=37302K     Unused=224841K     Limit=262144K     HighU=37302K
   >BAR: Allocated=10 Gigabytes                  Guard Amount=6,240 Megabytes
   >BAR: Pvt Used =4,000 Megabytes               Pvt Hi-Water=4,000 Megabytes
   >BAR: Pvt Obj  =1                             Shr Objects =0
   >BAR: Shr Alloc=0 Megabytes                   Shr Hi-Water=0 Megabytes
   >BAR: Com Alloc=0 Megabytes                   Com Hi-Water=0 Megabytes
   >BAR: Com Obj  =0                             Large Pages =0
 Private Area Subpools Allocated: Owned by Task - Acquired by GETMAIN/STORAGE
  SP-Key  Allocated   Free Space  Allocated   Free Space    Allocated       Free Space
          Areas-DQE   Areas-FQE   Below 16M   Below 16M   Above 16M Line  Above 16M Line
  003-8   6           1           704,512     2,872       69,165,056      0
  004-8   100         100         0           0           409,600         107,200
  005-8   100         100         0           0           409,600         9,600
  229-5   2           2           0           0           12,288          5,952
  230-0   1           1           4,096       760         0               0
  230-5   3           8           8,192       7,232       4,096           3,400
  230-8   1           1           0           0           4,096           4,072
  251-8   2           2           0           0           200,704         5,280
          ----------  ----------  ----------  ----------  --------------  -------------
  Totals  215         215         716,800     10,864      70,205,440      135,504
 Job Step CPU: 0HR 0MIN 0.34210SEC       Job Step SRB: 0HR 0MIN 0.01488SEC
 Subpool Storage Information:
  Extended Private Storage - Subpool Numbers that may be used for areas Above 16M Line
    ECSA Subpools.... 227      228   231  241
    ELSQA Subpools... 203 thru 225   233  234  235  253  254  255
    ESQA Subpools.... 239      245   246  247  248
    ESWA Subpools.... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252
  Private Storage - Subpool Numbers that may be used for areas Below 16M Line
    CSA Subpools..... 227      228   231  241
    LSQA Subpools.... 203 thru 225   233  234  235  253  254  255
    SQA Subpools..... 226      239   245
    SWA Subpools..... 229      230   249  236  237
    USER Subpools.... 0  thru  132   240  250  251  252

-Data Set Allocations at Abend-
DDName    EXCP Count        Data Set Name                                          Disp  VolSer
STEPLIB   21                USR001.WORK.LINKLIB                                   SHR   LS80C1
SYSPRINT                    USR001.WORK.SYSPRINT                                  SHR   LS80CD
SYSRECNT                    NULLFILE
SYSIN                       NULLFILE
SYSCHECK                    USR001.WORK.SYSCHECK                                  SHR   LS80C7
SYSABEND  5,066             USR001.WORK.DUMP                                      OLD   LS8119
LGRECOUT  5                 USR001.WORK.TRCEOUT                                   OLD   LS80C1
LGRSYSIN                    USR001.USRTEST7.JOB01473.D0000101.?                         JES

-Printing Storage Areas From Main Program-
Save Area Storage For CSECT: USRTEST7
Begin Dump of LCA Storage Area
```

* Storage area dump was excluded for this display

9.0 Basic Problem Solving

One approach to debugging a problem would be to start with placing a log point in the area of a program where a process is giving you problems. Then use a step through process of adding log points (#LGPOINT) in order to zero in on the exact logic that is causing problems. You can then decide where to establish log points to allow you to step through your program in greater detail, and which areas of your program would be OK to just let execute without stepping through with additional log points. Setting log points will allow you to see the logic flow of your program where you can display register contents, data fields, control block structures, and set storage watches to diagnose the problem.

If there exists an abnormal quantity or content of output in the //LGRECOUT data set, consider a possible logic or coding error for a loop routine within your Assembler program. A Logger Services log point may simply be reporting this error condition through the generation of a high volume of records. The log point will show the label name from your Assembler source code in the //LGRECOUT output report, so look for an unexpectedly high repeating pattern of log records with the same label name to help identify the possible origin of the problem such as an unintended loop. You can remove unnecessary log points and add new log points which can be set for registers, storage areas, or storage watches, and rerun your errant program to diagnose the problem.

Some possible conditions which can induce an unintended loop with too many iterations are:

1. In your program you have a log point within an intended loop routine where your program increments or decrements a counter incorrectly which is causing the exit condition to never be met until many more iterations than intended have passed.

2. In your program you have a log point within an intended loop routine where your program decrements a counter that is initially set incorrectly to not be positive resulting in the loop being continually enforced when the count is negative until the count silently rolls over, and eventually gets set to a value that will cause the exit condition to be met.

3. In your program you have a log point within an intended loop routine where you specified in your program the wrong data or started from the wrong place in your data area resulting in a compare condition which is never met preventing exit from a loop. In addition, within your loop you may be using the wrong index value for a data area or you may be using the same index value for data areas with different structures resulting in an exit condition not being satisfied. This may result in having to cancel your job, and examining the Logger records which have accumulated.

4. In your program you have a log point within an intended loop routine where your program tests for a bit in a bit-string flag byte where that flag is never set, is set incorrectly in relation to your program logic, or is initialized incorrectly.

5. In your program you have a log point within an intended loop routine where your program tests the wrong bit in a bit-string flag byte, or tests the wrong flag byte altogether resulting in an exit condition which may never be met. This may result in having to cancel your job, and examining the Logger records which have accumulated.

6. From any point within your program an incorrect branch (i.e., a wild branch) is taken to a section of code you never intended to be executed at that time, and in the absence of an outright abend that section of code is in a loop routine containing a log point where the incorrect branch results in variables not being set for proper exit from the loop.

7. In your program you have a log point within a loop routine where earlier your program generated an invalid address or loaded an invalid address which caused data to be overlaid with incorrect values, and later in your Assembler program execution this same data area is used within a loop routine where the loop logic is correct but now the data upon which it relies is bad.

Examine the output from //LGRECOUT. A diagnostic summary report is generated upon an abend to assist with determining the cause of an abend in the program.

Examine the output from job including JESMSGLG, JESYSMSG, SYSTERM, SYSPRINT, and dump output from SYSUDUMP, SYSABEND or SYSMDUMP if available.'

For a return or reason code with an abend code, refer to the manual 'z/OS MVS System Codes'.

For the message text or a return or reason code in a message, check the following references:

z/OS MVS System Messages, Vol 1 (ABA-AOM)
z/OS MVS System Messages, Vol 2 (ARC-ASA)
z/OS MVS System Messages, Vol 3 (ASB-BPX)
z/OS MVS System Messages, Vol 4 (CBD-DMO)
z/OS MVS System Messages, Vol 5 (EDG-GFS)
z/OS MVS System Messages, Vol 6 (GOS-IEA)
z/OS MVS System Messages, Vol 7 (IEB-IEE)
z/OS MVS System Messages, Vol 8 (IEF-IGD)
z/OS MVS System Messages, Vol 9 (IGF-IWM)
z/OS MVS System Messages, Vol 10 (IXC-IZP)
z/OS MVS Dump Output Messages'

10.0 LOGGRASM Messages

10.1 LGA01101A

LGA01101A Storage unavailable for Logger service area. Storage Obtain RC= {*return code*}

Indicates an attempt was made to conditionally acquire virtual storage for the Logger Services LG2II work area mapped by the LGCPLSWA copybook, and there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'RAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging.

The response is to increase the value in the REGION= parameter in your JCL, and rerun the job.

If increasing the region size does not correct the problem, consider these general guidelines.

There is a specific range of below-the-line values which may limit what you can specify on your REGION= parameter. These values usually are between 8192K and 16384K and vary from one site to another. These values are impossible values meaning that when they are specified your batch job will never be able to obtain a region required to run the step. If your site does not use an IEFUSI exit to check for these values and dynamically change them to an amount which is less than the maximum allowed for the private area below-the-line, then your job will get an S822 abend.

```
IEF085I REGION UNAVAILABLE, ERROR CODE=20
IEF187I USERPGM  FAILED - SYSTEM ERROR IN INITIATOR
IEF450I USERPGM  STEP1 - ABEND=S822 U0000 REASON=00000014
```

Region values between 16M and up to and including 32M will always set the maximum for below-the-line storage allowed for your site, and above-the-line storage will be set at 32M as the z/OS systems default. This means even if you specify REGION=24M, the z/OS default of 32Meg will always give you a REGION=32M in the absence of an IEFUSE exit specifying otherwise.

Region values between 32M and up to and including 2047M will always set the maximum for below-the-line storage for your site, and above-the-line storage will be set at what you code in the REGION= parameter in the absence of an IEFUSE exit specifying a limiting value. If there is no IEFUSE exit and you code a Region value of 2047M, all of the above-the-line storage allowed for your private area will be made available, though it will be less than 2047.

A Region value of zero (REGION=0M) will always set the maximum for the below-the-line storage for your site, and will make available all of the above-the-line storage for your site in the absence of an IEFUSE exit specifying a limiting value. This will provide for maximum storage availability, but the exact region size that will be made available is site dependent.

10.2 LGA01102A

LGA01102A Storage unavailable for options area. Storage Obtain RC={*return code*}

Indicates an attempt was made to conditionally acquire virtual storage for the Logger Services PGMLG2II program names work area mapped by the LGCPLSWA copybook, and there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. This area is used to process the input Logging options specified by the user in the //LGRSYSIN DD. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'STORAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging.

The response is to increase the value in the REGION= parameter in your JCL, and rerun the job.

If increasing the region size does not correct the problem, refer to the general guidelines in message LGA01101A.

10.3 LGA01103A

LGA01103A Unable to open data set for DDnme LGRECOUT

An error occurred attempting to open the //LGRECOUT DCB to enable logger records to be printed to an output data set.

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging.

The response is to check the system log to determine if a 913 abend occurred attempting to open the data set. It is most likely you will see system message IEC150I associated with the data set. Refer to IBM manual "MVS System Messages Volume 7 (IEB - IEE)". The most likely cause is you are not authorized to access the data set. This is not related to being APF authorized, or supervisor state, or system key, but to RACF not allowing access. Check for a ICH408I message. In your job refer to the JESYSMSG data set which contains any system messages for your job, and refer to the JESMSGLG data set which contains allocation messages and any JES2 and operator messages for your job.

Possible causes may be:

1. Your attempt to access violated security access rules under RACF or ACF2 or other security product, or the attempt to access violated resource access rules (e.g, OPEN OUTPUT but access allows read only). Verify that your data set naming convention is consistent with classroom requirements and follows the guidelines set by your instructor. Notify your instructor who may need to contact the RACF security administrator with information from your ICH408I message which will indicate the profile, class, and access allowed for the protected resource.

2. The data set name specification for the //LGRECOUT was misspelled or is incorrect, but was for a valid existing data set you did not intend to access which was protected by security access rules.

10.4 LGA01104A

LGA01104A Unknown operation code for logger option parameters, card input is ignored.

An invalid operation was specified for one of the input control statements in the //LGRSYSIN data set. Logger Services examines the input parameters from //LRGSYSIN and validates the operands. In this instance an invalid operand was specified for an operation field.

For diagnostic purposes the input control statements in //LRGSYSIN are written to the //LGRECOUT output data set. Immediately preceding the LGAO1104A message will be the input control statement containing the error.

The operation field must be specified as LOGEVENT.

For example:

```
LOGEVENT PROGRM,NAME=(*)
```

The response is review the control statement and check that the operation field is spelled correctly, and that it is followed by at least one blank if the starting position is from column 1, or if indented it is preceded and followed by at least one blank. For additional information, refer to 'Section 5 Input Control Card Format: //LGRSYSIN'

After review make any necessary corrections, and rerun the job.

Upon encountering this error condition, Logger Services will ignore the input control statement and proceed to the next statement. The user program is not interrupted, and runs to normal completion without logging enabled for the requested operation. This will result in only a partial report being generated which will not contain all the information you expected. Upon encountering this error, no return code is generated nor is one propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing.

10.5 LGA01105A

LGA01105A No operand supplied with logger option parameters, card input is ignored.

An operation was specified with no keyword options for one of the input control statements in the //LGRSYSIN data set. Logger Services examines the input parameters from //LRGSYSIN and validates the operands. In this instance no keyword options were specified for an operation field.

For diagnostic purposes the input control statements in //LRGSYSIN are written to the //LGRECOUT output data set. Immediately preceding the LGAO1105A message will be the input control statement containing the error.

The response is review the control statement and check that the LOGEVENT operation field is specified with valid keywords.

For example:

```
LOGEVENT PROGRM,NAME=(*)
```

Keywords are PROGRM for program logging and NAME= for Csect names. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

After review make any necessary corrections, and rerun the job.

Upon encountering this error condition, Logger Services will ignore the input control statement and proceed to the next statement. The user program is not interrupted, and runs to normal completion without logging enabled for the requested operation. This will result in only a partial report being generated which will not contain all the information you expected. Upon encountering this error, no return code is generated nor propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing.

10.6 LGA01106A

LGA01106A Invalid keyword for logger option parameter, card input is ignored.'

An operation was specified with an invalid keyword for one of the input control statements in the //LGRSYSIN data set. Logger Services examines the input parameters from //LRGSYSIN and validates the operands. In this instance an invalid keyword was specified.

For diagnostic purposes the input control statements in //LRGSYSIN are written to the //LGRECOUT output data set. Immediately preceding the LGAO1106A message will be the input control statement containing the error.

The response is review the control statement and check that the PROGRM, SUBRTN, or LOGPNT keyword fields are specified correctly.

For example:

```
LOGEVENT PROGRM,NAME=(*)
LOGEVENT SUBRTN,NAME=(*)
LOGEVENT LOGPNT,NAME=(*)
```

Valid keywords are PROGRM for program logging, SUBRTN for subroutine logging, and LOGPNT for Log Point logging. The name keyword is specified as NAME= to designate all Csects as in NAME=(*), to specify a single Csect name as in NAME=MYCSECT1 or NAME=(MYCSECT1), or to specify a list of Csect names as in NAME=(MYCSECT1,MYCSECT2) which are subject to logging. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

After review make any necessary corrections, and rerun the job.

Upon encountering this condition, Logger Services will ignore the input control statement and proceed to the next statement. The user program is not interrupted, and runs to normal completion without logging enabled for the requested operation. This will result in only a partial report being generated which will not contain all the information you expected. Upon encountering this error, no return code is generated nor propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing.

10.7 LGA01107A

LGA01107A Unknown keyword specified for logger option parameter, card input is ignored.

An operation was specified with an invalid keyword for one of the input control statements in the //LGRSYSIN data set. Logger Services examines the input parameters from //LRGSYSIN and validates the operands. In this instance the NAME parameter was not properly specified.

For diagnostic purposes the input control statements in //LRGSYSIN are written to the //LGRECOUT output data set. Immediately preceding the LGAO1107A message will be the input control statement containing the error.

The response is review the control statement and check that the NAME= keyword field is specified correctly.

For example:

```
LOGEVENT PROGRM,NAME=(*)
LOGEVENT SUBRTN,NAME=(*)
LOGEVENT LOGPNT,NAME=(*)
```

The keyword is specified as NAME=. For the PROGRM parameter you could designate all Csects as in NAME=(*), or specify a single Csect name as in NAME=MYCSECT1 or NAME=(MYCSECT1), or specify a list of Csect names as in NAME=(MYCSECT1,MYCSECT2) which are subject to logging. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

After review make any necessary corrections, and rerun the job.

Upon encountering this condition, Logger Services will ignore the input control statement and proceed to the next statement. The user program is not interrupted, and runs to normal completion without logging enabled for the requested operation. This will result in only a partial report being generated which will not contain all the information you expected. Upon encountering this error, no return code is generated nor propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing.

10.8 LGA01108A

LGA01108A Error establishing recovery environment. ESTAEX return code={*return code*}, reason code={*reason code*}

Indicates an attempt was made during Logger Services initialization to setup the recovery environment with an ESTAEX to receive control in the event of abnormal termination occurring in the user's program or in Logger Service programs. The return code indicated in the message is the completion code returned in register 15 by the ESTAEX macro, and the reason code is from register 0.

The response is to refer to subsection 'Return and Reason Codes' in section 'ESTAEX' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 2 (ENFO-IXGWRITE) " for an explanation of the return code and reason code.

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging and without a recovery environment. Upon encountering this error, no return code is generated nor propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing. However, the job will run without Logger Services and without ESTAEX recovery enabled.

10.9 LGA01109A

LGA01109A Storage unavailable for work area. Storage Obtain RC={*return code*}

Indicates an attempt was made by Logger Services initialization program LGMHLRCI to conditionally acquire virtual storage for a temporary work area, and there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'STORAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging.

The response is to increase the value in the REGION= parameter in your JCL, and rerun the job.

If increasing the region size does not correct the problem, refer to the general guidelines in message LGA01101A.

Upon encountering this condition, Logger Services will stop any further initialization processing and Logger Services will be disabled for the job run. The user program runs to normal completion without logging and without a recovery environment. Upon encountering this error, no return code is generated nor propagated back to the user program. This is to ensure all decisions made by the user's Assembler program are not biased by Logger Services, and are based solely on its own processing.

10.10 LGA01110I

LGA01110I Assembler program logging has completed.

This message is issued as an informational message confirming that the Logger Services termination program LGMGLRCT has completed program logging, released Logger resources, and has returned to the user's program for end of job.

10.11 LGA01111A

LGA01111A Primary index block overrun on secondary index block split.

The LPGMNTRY entry was specified with the LOGOUT=BUFR option to direct the buffering of Logger records, and the user set the region size to 0M not subject to an IEFUSI. There existed a sufficiently large private area to allow a volume of more than 8 million Logger records to accumulate in the above-the-line memory buffers.

This resulted in the Logger Services primary index block for the log record buffers to fill causing program LGMGLRCB to issue the LGA01111A block overrun WTO message. Program LGMHLRCB will set a return code 12 for the severity for this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the overrun bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error producing a truncated Logger report.

The response is to use LOGOUT=PRNT to write logger records to DASD immediately upon generation without buffering, and ensure the //LGRECOUT data set is sufficiently large to hold the anticipated volume of records.

If you need to create a large result set in memory, try using LOGOUT=BUF64 which will buffer the records in storage above-the-bar, offer more available storage subject to the MEMLIMIT set for the job, and will allow substantially more than 8 million records to be cached in memory.

However, it is a best practice to use //LRGSYSIN input control statements to implement filtering. Filtering can significantly reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of records. For additional information on filtering, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'. Also refer to ‘Chapter 9 Basic Problem Solving’.

10.12 LGA01112A

LGA01112A Storage exhausted for log buffer area. Storage RC = {*return code*}

The LPGMNTRY entry was specified with the LOGOUT=BUFR option to direct the buffering of Logger records. During the insertion of log records into the buffers the last buffer area in the chain of buffer areas had insufficient room remaining to insert the current log record. During the attempt to dynamically acquire an additional page aligned area of storage to hold more Logger records, there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'STORAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

The response is to increase the value in the REGION= parameter in your JCL, and rerun the job. If increasing the region size does not correct the problem, refer to the general guidelines in message LGA01101A.

Consider using LOGOUT=PRNT to write logger records directly to DASD immediately upon generation of a record without buffering to significantly reduce memory. In this case also ensure that the //LGRECOUT data set is sufficiently large enough to hold the anticipated volume of records.

Consider using //LRGSYSIN input control statements with filtering to reduce the volume of log records to a manageable size both for space considerations and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

Upon encountering this error condition, Logger Services program LGMHLRCB will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program attempting to proceed to normal completion, but no further log records will be written after this error producing a truncated Logger report. However, even though Logger Services has stopped obtaining any more storage, in some instances there may be so little region area left in the address space that even z/OS system services may get 878-010 abends, CVAF errors, and the job eventually abend 40D and terminate at end of memory.

10.13 LGA01113A

LGA01113A Storage exhausted for secondary index. Storage RC = {*return code*}

The LPGMNTRY entry was specified with the LOGOUT=BUFR option to direct the buffering of Logger records. During the insertion of log records into the buffers an expansion of a secondary index block was required on account of an insert of a log record into a buffer area was pending because a secondary index block which indexes the buffer area had filled up. During the process to expand the filled secondary index block into two secondary index blocks, the attempt to dynamically acquire an additional above-the-line page aligned area of storage for the next secondary index area failed because there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'STORAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

The response is to increase the value in the REGION= parameter of your JCL, and rerun the job. If increasing the region size does not correct the problem, refer to the general guidelines in message LGA01101A.

Consider using LOGOUT=PRNT to write logger records to DASD immediately upon generation without buffering to significantly reduce memory, and ensure the //LGRECOUT data set is sufficiently large to hold the anticipated volume of records.

If you need to create a large result set in memory, try using LOGOUT=BUF64 which will buffer the records in storage above-the-bar, and offer more available storage subject to the MEMLIMIT set for the job. You will need to ensure that the //LGRECOUT data set is sufficiently large to hold the anticipated volume of records when using LOGOUT=BUF64.

It would be a best practice to limit log record output by using //LRGSYSIN input control statements with filtering to reduce the volume of log records to a manageable size both for space considerations and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

Upon encountering this error condition, Logger Services program LGMHLRCB will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error producing a truncated Logger report.

10.14 LGA01114A

LGA01114A Storage exhausted for primary index. Storage RC = {*return code*}

The LPGMNTRY entry was specified with the LOGOUT=BUFR option to direct the buffering of Logger records. During the process to dynamically acquire memory for the primary index block there was insufficient contiguous virtual storage available within the private area of the user program's address space to satisfy the request for more storage. The return code indicated in the message is the completion code returned by STORAGE OBTAIN for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'STORAGE-Obtain and Release' in IBM manual "MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFRR-WTOR)".

The response is to increase the value in the REGION= parameter in your JCL, and rerun the job. If increasing the region size does not correct the problem, refer to the general guidelines in message LGA01101A.

Consider using LOGOUT=PRNT to write logger records to DASD immediately upon generation without buffering to significantly reduce memory, and ensure the //LGRECOUT data set is sufficiently large to hold the anticipated volume of records.

If you need to create a large result set in memory, try using LOGOUT=BUF64 which will buffer the records in storage above-the-bar, and offer more available storage subject to the MEMLIMIT set for the job. You will need to ensure that the //LGRECOUT data set is sufficiently large to hold the anticipated volume of records when using LOGOUT=BUF64.

It would be a best practice to limit log record output by using //LRGSYSIN input control statements with filtering to reduce the volume of log records to a manageable size both for space considerations and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

Upon encountering this error condition, Logger Services program LGMHLRCB will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error producing a truncated Logger report.

10.15 LGA01115A

LGA01115A IARV64 GetStore error for log. RC= {*return code*} Reason Code = {*reason code*}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object using the IARV64 GETSTOR for a log record buffer area, and there was an error on the attempt to create a memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from the IARV64 GETSTOR service for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation' in the IBM manual "z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT".

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 'System Completion Codes' in the IBM manual "z/OS V1R13 MVS System Completion Codes". If you have no hardcopy manual available you can Google the phrase "z/OS MVS System Completion Codes" to reference this document online.

The reason code described in the manual will be a 4-byte hexadecimal value in the format 'xxRRRRyy', where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

A likely condition is there may have been insufficient above-the-bar storage available within the private area of the user program's address space to satisfy the request for storage. In this event the LGA01115A message may show a return code of 008 and a reason code (0016) indicating the memory limit was exceeded on an attempt by Logger Services program LGMHLB64 to acquire more storage above the bar than what the current MEMLIMIT would allow for the job. The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your overall above-the-bar storage needs, then increase the value in the MEMLIMIT= parameter on the JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

However, even if a higher MEMLIMIT is set, the maximum amount of above-the-bar storage that will be used by the LGMHLB64 program for buffering the log indices and log records is capped at 10 gigabytes. The cap by LGMHLB64 is set at 10 gigabytes for purposes of buffering log records because it is considered a large enough number to be conceptual infinity.

Another response would be to consider using the LOGOUT=PRNT option to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

However, it would be a best practice to limit log record output by using //LRGSYSIN input control statements with the filtering option to reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error which will result in producing a truncated Logger report.

10.16 LGA01116A

LGA01116A MEMLIMIT is zero value indicating no above the bar storage.

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. Before a conditional request could be made to create a memory object for a log record buffer area, the LGMHLB64 program determined there was no above-the-bar storage available, and the pending request by LGMHLB64 to use IARV64 GETSTOR services would not succeed. For this condition verify that the virtual storage MEMLIMIT is set properly for your job.

Some possible condition which could cause this type of error:

(1)You did not include a MEMLIMIT= parameter on the JOB or EXEC statements in your JCL, you were executing on a z/OS V1R10 or higher system where the MEMLIMIT default is 2 GB, and a zero MEMLIMIT was specified in an active SMFPRMxx member of parmlib to establish a system wide setting of zero resulting no virtual storage above-the-bar being available. Response would be to check if a MEMLIMIT value would be accepted by the z/OS system for your job, then include a nonzero MEMLIMIT= value on the JOB or EXEC statements in your JCL, and then rerun the job. Recommended minimum would be 20 megabytes.

(2)You were executing on a z/OS V1R10 or higher system where the MEMLIMIT default is 2 GB or a MEMLIMIT value was specified in an active SMFPRMxx member of parmlib to override the default, and you specified a MEMLIMIT=0 explicitly on the JOB or EXEC statements in your JCL to override the system default that was set or allowed to default to in SMFPRMxx. This resulted in an address space setting of zero being established causing no virtual storage above-the-bar being available for your job. Response would to specify a nonzero MEMLIMIT= value on the JOB or EXEC statements in your JCL, and then rerun the job.

Recommended minimum would be 20 megabytes, or higher if necessary to have sufficient memory to hold your anticipated volume of records.

(3)You explicitly specified REGION=0M or a MEMLIMIT value on the JOB or EXEC statements in your JCL to override the system-wide default MEMLIMIT, and there was an active IEFUSI system exit being used to establish a system default MEMLIMIT for different job classes, and you explicitly specified a CLASS= parameter in your JCL or were assigned a job class by the system where the IEFUSI exit would limit the amount of above-the-bar that can be used for your job to zero. Response would to specify a different job class value for the CLASS= parameter on the JOB card in your JCL that would allow the resources required to run your job, and rerun the job.

Also, you can reference the prolog section at the very beginning of the log output report, and check the 'MEMLIMIT Source' detail line which will indicate from which source the MEMLIMIT value was established such as from JCL, SMF, or an IEFUSI exit.

If above-the-bar storage is unavailable to you, another response would be to consider using the LOGOUT=BUFR option instead of LOGOUT=BUF64 to buffer records above-the-line. Ensure that you increase the value in the REGION= parameter in your JCL to handle your anticipated volume of records, check that the maximum above-the-line private area that could be made available to you is even capable of providing sufficient above-the-line storage, and attempt to rerun the job.

If above-the-bar storage is unavailable to you, another response would be to consider using the LOGOUT=PRNT option instead of LOGOUT=BUF64 to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.17 LGA01117A

LGA01117A MEMLIMIT less than 20 meg and insufficient for processing.

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. Before a conditional request could be made to create a memory object for a log record buffer area, the LGMHLB64 program determined there was insufficient above-the-bar storage available. The request by LGMHLB64 to use IARV64

GETSTOR services is not executed, and no memory objects will be created. For this condition verify that the virtual storage MEMLIMIT is set properly for your job.

When using the LOGOUT=BUF64 option the minimum amount you can specify for MEMLIMIT is 20 megabytes. The response would be to determine if the MEMLIMIT value for the job could be raised to above 20 megabytes to accommodate your storage needs for the log records you expect to produce, and if so increase the value in the MEMLIMIT= parameter on the JOB or EXEC statements in the JCL to higher than 20 Megabytes, and rerun the job.

If above-the-bar storage is unavailable to you, consider using the LOGOUT=BUFR option instead of LOGOUT=BUF64 to buffer records above-the-line. If you expect 20 megabytes to be sufficient to handle the number of log records in a job run, this storage amount should easily fit in the private area above-the-line in your address space. Ensure that you increase the value in the REGION= parameter in your JCL to handle your above-the-line storage needs, and rerun the job. If above-the-bar storage is unavailable to you, another response would be to consider using the LOGOUT=PRNT option instead of LOGOUT=BUF64 to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.18 LGA01118A

LGA01118A IARV64 Getstore error 2nd Indx. RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object using the IARV64 GETSTOR for a secondary index area, and there was an error on the attempt to create a memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from the IARV64 GETSTOR service for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation' in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online.

The reason code described in the manual will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

A likely condition is there may have been insufficient above-the-bar storage available within the private area of the user program's address space to satisfy the request for storage. In this event the LGA01118A message may show a return code of 008 and a reason code (0016) indicating the memory limit was exceeded on an attempt by Logger Services program LGMHLB64 to acquire more storage above the bar than what the current MEMLIMIT would allow for the job. The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, both for the memory objects created by Logger Services and for any memory objects that your own user program may be using. Then increase the value in the MEMLIMIT= parameter on the JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

However, even if a higher MEMLIMIT is set, the maximum amount of above-the-bar storage that will be used by the LGMHLB64 program for buffering the log indices and log records is capped at 10 gigabytes. The cap by LGMHLB64 is set at 10 gigabytes for purposes of buffering log records because it is considered a large enough number to be conceptual infinity.

Another response would be to consider using the LOGOUT=PRNT option to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

However, it would be a best practice to limit log record output by using //LRGSYSIN input control statements with the filtering option to reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of log records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.19 LGA01119A

LGA01119A IARV64 Getstore error 1st Indx. RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object using the IARV64 GETSTOR for a primary index area, and there was an error on the attempt to create a memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from the IARV64 GETSTOR for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation' in the IBM manual "z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT".

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 'System Completion Codes' in the IBM manual "z/OS V1R13 MVS System Completion Codes". If you have no hardcopy manual available you can Google the phrase "z/OS MVS System Completion Codes" to reference this document online.

The reason code described in the manual will be a 4-byte hexadecimal value in the format 'xxRRRRyy', where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

A likely condition is there may have been insufficient above-the-bar storage available within the private area of the user program's address space to satisfy the request for storage. In this event the LGA01119A message may show a return code of 008 and a reason code (0016) indicating the memory limit was exceeded on an attempt by Logger Services program LGMHLB64 to acquire more storage above the bar than what the current MEMLIMIT would allow for the job. The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, both for the memory objects created by Logger Services and for any memory objects that your own user program may be using. Then increase the value in the MEMLIMIT= parameter on the JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

However, even if a higher MEMLIMIT is set, the maximum amount of above-the-bar storage that will be used by the LGMHLB64 program for buffering the log indices and log records is capped at 10 gigabytes. The cap by LGMHLB64 is set at 10 gigabytes for purposes of buffering log records because it is considered a large enough number to be conceptual infinity.

Another response would be to consider using the LOGOUT=PRNT option to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

However, it would be a best practice to limit log record output by using //LRGSYSIN input control statements with the filtering option to reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of log records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.20 LGA01120A

LGA01120A IARV64 Getstor abended on Log {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object for buffering of Logger records to storage above-the-bar, and there was an error on the attempt by the IARV64 GETSTOR service to create the memory object.For this condition check the completion code and reason code.

As part of the process to create the log record storage area, a conditional request was made using the IARV64 GETSTOR service to create a log record memory object. However, the GETSTOR request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the GETSTOR request to end abnormally in an abend. However, a Logger Services recovery routine is used on the GETSTOR call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01120A message is the completion code returned by the IARV64 GETSTOR request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01120A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the GETSTOR request was abnormally terminated. The reason code in the LGA01120A message will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. Attempt to correct the error indicated by the reason code, and rerun the job. If unable to resolve the issue, contact Sam Golob at sbgolob@cbttape.org.

Another response would be to consider using the LOGOUT=PRNT option instead of LOGOUT=BUF64 which will write logger records to DASD immediately upon generation without any buffering to above-the-bar storage. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold your anticipated volume of records.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion, but the IARV64 GETSTOR failure will prevent any further buffering of log records. This means in the absence of any other problems the user program runs to normal completion except for the log report which will be cut short, and only the log records which were created previous to the IARV64 abend will eventually be written to the output //LGRECOUT data set when the user program ends.

10.21 LGA01121A

LGA01121A IARV64 Getstor abended on Index2 {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object for indexing of Logger records, and there was an error on the attempt by the IARV64 GETSTOR service to create the memory object for the secondary index area in above-the-bar storage. For this condition check the completion code and reason code.

A secondary index is used to process the log records. A conditional request was made using the IARV64 GETSTOR service to create a secondary index memory object. However, the GETSTOR request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the GETSTOR request to end abnormally in an abend. However, a Logger Services recovery routine is used on the GETSTOR call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01121A message is the completion code returned by the IARV64 GETSTOR request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01121A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the GETSTOR request was abnormally terminated. The reason code in the LGA01121A message will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. Attempt to correct the error indicated by the reason code, and rerun the job. If unable to resolve the issue, contact Sam Golob at sbgolob@cbttape.org.

Another response would be to consider using the LOGOUT=PRNT option instead of LOGOUT=BUF64 which will write logger records to DASD immediately upon generation without any buffering to above-the-bar storage. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold your anticipated volume of records.

Upon an abend in the IARV64 service, abend recovery will interrupt the abend, return to the retry address in Logger Services program LGMHLB64, program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. User program execution continues to completion, but the IARV64 GETSTOR failure will prevent any further indexing of log records. This means in the absence of any other problems the user program will run to normal completion except for the log report which will be cut short, and only the log records which were created previous to the IARV64 abend will eventually be written to the output //LGRECOUT data set when the user program ends.

10.22 LGA01122A

LGA01122A IARV64 Getstor abended on Index1 {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to create a memory object for indexing of Logger records, and there was an error on the attempt by the IARV64 GETSTOR service to create the memory object for the primary index area in above-the-bar storage. For this condition check the completion code and reason code.

A primary index is used to hold pointers to entries in the secondary index. A conditional request was made using the IARV64 GETSTOR service to create a primary index memory object. However, the GETSTOR request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the GETSTOR request to end abnormally in an abend. However, a

Logger Services recovery routine is used on the GETSTOR call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01122A message is the completion code returned by the IARV64 GETSTOR request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01122A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the GETSTOR request was abnormally terminated. The reason code in the LGA01122A message will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. Attempt to correct the error indicated by the reason code, and rerun the job. If unable to resolve the issue, contact Sam Golob at sbgolob@cbttape.org.

Another response would be to consider using the LOGOUT=PRNT option instead of LOGOUT=BUF64 which will write logger records to DASD immediately upon generation without any buffering to above-the-bar storage. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold your anticipated volume of records.

Upon an abend in the IARV64 service, abend recovery will interrupt the abend, return to the retry address in Logger Services program LGMHLB64, program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. User program execution continues to completion, but the IARV64 GETSTOR failure will prevent any further indexing of log records. This means in the absence of any other problems the user program will run to normal completion except for the log report which will be cut short, and only the log records which were created previous to the IARV64 abend will eventually be written to the output //LGRECOUT data set when the user program ends.

10.23 LGA01123A

LGA01123A IARV64 Changeguard error: * RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers an expansion of a secondary index block was required on account of an insert of a log record into a buffer area was pending because a secondary index block which indexes the buffer area had filled up.

As part of the process to expand the filled secondary index block by splitting it into two secondary index blocks, a conditional request was made using the IARV64 CHANGEGUARD service to change one megabyte in a secondary index memory object from the guard state to a usable state to add more usable above-the-bar area storage to hold the next secondary index area.

The severity of the error is indicated by the return code from the IARV64 CHANGEGUARD for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation’ in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for this IARV64 conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online.

The reason code described in the manual will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

A likely condition is there may have been insufficient above-the-bar storage available within the private area of the user program's address space to process a secondary index block split. In this event the LGA01123A message may show a return code of 008 and a reason code (0016) indicating the MEMLIMIT was exceeded on an attempt by Logger Services program LGMHLB64 to use the IARV64 CHANGEGUARD service to change a one megabyte segment in the secondary index memory object from the guard state to a usable state. The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, both for the memory objects created by Logger Services and for any memory objects that your own user program may be using. Then increase the value in the MEMLIMIT= parameter on the JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error which will result in producing a truncated Logger report.

10.24 LGA01124A

LGA01124A IARV64 Changeguard error: - RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers the last buffer area in the chain of buffer areas had insufficient room remaining to insert the current log record. As part of the process to expand the log record area, a conditional request was made using the IARV64 CHANGEGUARD service to change eight megabytes in the log record memory object from the guard state to a usable area to add more usable above-the-bar area storage to hold the next block of incoming log records.

The severity of the error is indicated by the return code from the IARV64 CHANGEGUARD for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation’ in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for this IARV64 conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online.

The reason code described in the manual will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

A likely condition is there may have been insufficient above-the-bar storage available within the private area of the user program's address space to add more log records. In this event the LGA01124A message may show a return code of 008 and a reason code (0016) indicating the MEMLIMIT was exceeded on an attempt by Logger Services program LGMHLB64 to use the IARV64 CHANGEGUARD service to change an eight megabyte segment in the log record memory object from the guard state to a usable state. The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, both for the memory objects created by Logger Services and for any memory objects that your own user program may be using. Then increase the value in the MEMLIMIT=

parameter on the JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no further log records will be written after this error which will result in producing a truncated Logger report.

10.25 LGA01125A

LGA01125A IARV64 Changeguard abended with {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers an expansion of a secondary index block was required on account of an insert of a log record into a buffer area was pending because a secondary index block which indexes the buffer area had filled up. As part of the process to expand the filled secondary index block by splitting it into two secondary index blocks, a conditional request was made using the IARV64 CHANGEGUARD service to change one megabyte in a secondary index memory object from the guard state to a usable state. This was required in order to add to the usable area to hold the next secondary index block.

However, in this instance the secondary index CHANGEGUARD request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the CHANGEGUARD request to end abnormally in an abend. However, a Logger Services recovery routine is used on the CHANGEGUARD call where the abend will not terminate the user program, and processing continues to normal completion.

The CHANGEGUARD error sequence is where the CHANGEGUARD call is made in program LGMHLB64, the CHANGEGUARD fails, CHANGEGUARD calls the Abend macro to terminate, RTM intercepts the abend, followed by invocation of the Logger Services ESTAEX recovery program LGMHESTA, and then its a SETRP call back to the retry address in program LGMGLB64 to issue the LGA01125A error message to describe the cause of the IARV64 failure, and after the message is issued return back to the user program.

The abend code indicated in the LGA01125A message is the completion code returned by the IARV64 CHANGEGUARD request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01125A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the CHANGEGUARD request was abnormally terminated. This reason code will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. If unable to resolve the issue and technical support is required, contact Sam Golob at sbgolob@cbttape.org.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion, but the IARV64 CHANGEGUARD failure will prevent any further buffering of log records. This means in the absence of any other problems the user program runs to normal completion except for the log report which will be cut short, and only the log records which were created previous to the IARV64 abend will eventually be written to the output //LGRECOUT data set when the user program ends.

10.26 LGA01126A

LGA01126A IARV64 Changeguard abended with {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers the usable area in the log record memory object had filled with log records, and an expansion of the usable area was required to hold additional incoming log records. As part of the process to expand the buffers, a conditional request was made using the IARV64 CHANGEGUARD service to convert an 8 megabyte portion of the guard area from the guard state to a usable state. This was required in order to add more storage to the usable area to hold the increasing volume of log records being generated by Logger Services.

However, in this instance the log record CHANGEGUARD request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the CHANGEGUARD request to end abnormally in an abend. However, a Logger Services recovery routine is used on the CHANGEGUARD call where the abend will not terminate the user program, and processing continues to normal completion.

The CHANGEGUARD error sequence is where the CHANGEGUARD call is made in program LGMHLB64, the CHANGEGUARD fails, CHANGEGUARD calls the Abend macro to terminate, RTM intercepts the abend, followed by invocation of the Logger Services ESTAEX recovery program LGMHESTA, and then its a SETRP call back to the retry address in programLGMHLB64 to issue the LGA01126A error message to describe the cause of the IARV64 failure, and after the message is issued return back to the user program.

The abend code indicated in the LGA01126A message is the completion code returned by the IARV64 CHANGEGUARD request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01126A message there is a reason code associated with the abend. Within the 4-byte reason code will be a 2-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the CHANGEGUARD request was abnormally terminated. This reason code will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. If unable to resolve the issue and technical support is required, contact Sam Golob at sbgolob@cbttape.org.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion, but the IARV64 CHANGEGUARD failure will prevent any further buffering of log records. This means in the absence of any other problems the user program runs to normal completion except for the log report which will be cut short, and only the log records which were created previous to the IARV64 abend will eventually be written to the output //LGRECOUT data set when the user program ends.

10.27 LGA01127A

LGA01127A Changeguard for record area exceeds available guard area.

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers the usable area in the log record memory object had filled with log records, and an expansion of the usable area was required to hold additional incoming log records.

As part of the process to expand the buffers, a verification process was performed. This verification was done just before the request would be made to the IARV64 CHANGEGUARD service to convert an 8 megabyte portion of the guard area from the guard state to a usable state in order to add more usable above-the-bar storage. This error indicates that the verification check had found that the maximum number of segments had already been obtained for the log record memory object. What this means is you ran out of segments in the log record memory object before you reached the MEMLIMIT because the log record memory object is just one of three memory objects used for managing log records where all three memory objects contribute to the MEMLIMIT.

The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, then increase the value in the MEMLIMIT= parameter on JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

However, even if a higher MEMLIMIT is set, the maximum amount of above-the-bar storage that will be used by the LGMHLB64 program for buffering the log indices and log records is capped at 10 gigabytes. The cap by LGMHLB64 is set at 10 gigabytes for purposes of buffering log records because it is considered a large enough number to be conceptual infinity.

Another response would be to consider using the LOGOUT=PRNT option to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

However, it would be a best practice to limit log record output by using //LRGSYSIN input control statements with the filtering option to reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.28 LGA01128A

LGA01128A Changeguard for 2nd Index area exceeds available guard area.

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made to change a specified range in a memory object from the guard state to a usable area, and there was an error on the attempt to change the existing memory object for above-the-bar storage. For this condition check the return code and reason code.

During the insertion of log records into the buffers an expansion of a secondary index block was required on account of an insert of a log record into a buffer area was pending because a secondary index block which indexes the buffer area had filled up. As part of the process to expand the filled secondary index block by splitting it into two secondary index blocks, a conditional request would be made using the IARV64 CHANGEGUARD service to change one megabyte in a secondary index memory object from the guard state to a usable state to add more usable above-the-bar area storage to hold the next secondary index area.

As part of the process to expand the secondary index memory object, a verification process was performed. This verification was done just before the request would be made to the IARV64 CHANGEGUARD service to convert a 1 megabyte portion of the guard area from the guard state to a usable state in order to add more usable above-the-bar storage. This error indicates that the verification check had found that the maximum number of segments had already been obtained for the secondary index memory object. What this means is you ran out of segments in the secondary index memory object before you reached the MEMLIMIT because the secondary index memory object is just one of three memory objects used for managing log records where all three memory objects contribute to the MEMLIMIT.

The response would be to determine if the MEMLIMIT value for the job could be raised to a size sufficiently large enough to accommodate your storage needs, then increase the value in the MEMLIMIT= parameter on JOB or EXEC statements in the JCL to meet your anticipated needs, and rerun the job.

However, even if a higher MEMLIMIT is set, the maximum amount of above-the-bar storage that will be used by the LGMHLB64 program for buffering the log indices and log records is capped at 10 gigabytes. The cap by LGMHLB64 is set at 10 gigabytes for purposes of buffering log records because it is considered a large enough number to be conceptual infinity.

Another response would be to consider using the LOGOUT=PRNT option to write logger records to DASD immediately upon generation without buffering to significantly reduce memory. You must ensure the //LGRECOUT data set is allocated sufficiently large enough to hold the anticipated volume of records.

However, it would be a best practice to limit log record output by using //LRGSYSIN input control statements with the filtering option to reduce the volume of log records to a manageable size both for space considerations, and to decrease the effort of viewing and searching a large collection of records. For additional information, refer to 'Chapter 5 Input Control Card Format: //LGRSYSIN'

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set the out-of-storage bit flag, and then program LGMGLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion, but no log records will be written to the //LGRECOUT Logger report data set.

10.29 LGA01129A

LGA01129A IARV64 Detach error Log Area. RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a log record buffer area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from IARV64 DETACH for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation’ in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. You should attempt to correct the error indicated by the reason code, and rerun the job.

The reason code described in the manual will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error. The response is to check the reason code represented by the bytes at RRRR to explain why the error occurred.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

10.30 LGA01130A

LGA01130A IARV64 Detach error 2nd Index. RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a secondary index area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from IARV64 DETACH for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation’ in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. You should attempt to correct the error indicated by the reason code, and rerun the job.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

10.31 LGA01131A

LGA01131A IARV64 Detach error 1st Index. RC= {return code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a primary index area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

The severity of the error is indicated by the return code from IARV64 DETACH for a conditional request. For an explanation of the return code, refer to subsection 'Return and Reason Codes' in section 'IARV64 – 64-bit Virtual Storage Allocation’ in the IBM manual “z/OS V1R13 MVS Programming: Assembler Services Reference IAR-XCT”.

To find the reason code you will need to reference another IBM manual. The reason code for the conditional request will be one of the reason codes listed in abend code DC2 for an unconditional IARV64 request. For information on interpreting the reason code, refer to the explanation for abend code DC2 in Section 2.0 ‘System Completion Codes’ in the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. You should attempt to correct the error indicated by the reason code, and rerun the job.

Upon encountering this error condition, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. This will result in the user program proceeding to normal completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

10.32 LGA01132A

LGA01132A IARV64 Detach Log Area abended {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a log record buffer area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

In this instance the IARV64 DETACH request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the DETACH request to end abnormally in an abend. However, Logger Services uses a recovery routine on the DETACH call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01132A message is the completion code returned by the IARV64 DETACH request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01132A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the DETACH request was abnormally terminated. This reason code will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. If unable to resolve the issue and technical support is required, contact Sam Golob at sbgolob@cbttape.org.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

10.33 LGA01133A

LGA01133A IARV64 Detach 2nd Index abended {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a secondary index area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

In this instance the IARV64 DETACH request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the DETACH request to end abnormally in an abend. However, Logger Services uses a recovery routine on the DETACH call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01133A message is the completion code returned by the IARV64 DETACH request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01133A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the DETACH request was abnormally terminated. This reason code will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. If unable to resolve the issue and technical support is required, contact Sam Golob at sbgolob@cbttape.org.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

10.34 LGA01134A

LGA01134A IARV64 Detach 1st Index abended {abend code} Reason Code = {reason code}

The LPGMNTRY entry was specified with the LOGOUT=BUF64 option to direct the buffering of Logger records to storage above-the-bar. A conditional request was made by the LGMHLB64 Logger Services program to release a memory object using the IARV64 DETACH service for a primary index area, and there was an error on the attempt to release the memory object for above-the-bar storage. For this condition check the return code and reason code.

In this instance the IARV64 DETACH request encountered an environmental issue of such severity that the condition was not recoverable by the use of the COND=YES parameter, and the severity of the error was sufficient to cause the DETACH request to end abnormally in an abend. However, Logger Services uses a recovery routine on the DETACH call where the abend will not terminate the user program, and processing continues to normal completion.

The abend code indicated in the LGA01134A message is the completion code returned by the IARV64 DETACH request. The IARV64 service call will abnormally terminate with a DC2 abend. In the LGA01134A message there is a reason code associated with the abend. The reason code will be a 4-byte hexadecimal value in the format ‘xxRRRRyy’, where the two-byte hex value in byte positions RRRR will describe the error.

The response is to check the reason code represented by the bytes at RRRR to explain why the DETACH request was abnormally terminated. This reason code will be one of the reason codes listed for abend code DC2 in Section 2.0 ‘System Completion Codes’ of the IBM manual “z/OS V1R13 MVS System Completion Codes”. If you have no hardcopy manual available you can Google the phrase “z/OS MVS System Completion Codes” to reference this document online. If unable to resolve the issue and technical support is required, contact Sam Golob at sbgolob@cbttape.org.

Upon an abend in the IARV64 service, Logger Services program LGMHLB64 will set a return code 12 for the severity of this condition, and pass the return code back to calling Logger Services program LGMHLRCE to set an error bit flag, and then program LGMHLRCE will clear the return code to ensure it is not propagated back to impact the user program. Program execution continues to completion. The request for IARV64 DETACH occurs after all Log records from the memory object have been written to output, and the log records will be available to view in the //LGRECOUT Logger report data set.

11.0 General Overview

To create the LOGGRASM environment, LOGGRASM performs the setup of the user program entry protocol and user program exit protocol, and makes the user program re-entrant. Ensure you link-edit with the RENT attribute. LOGGRASM always sets up your program to be re-entrant.

LOGGRASM is basically designed to work only with new programs where you are starting from scratch, such as when a student begins an Assembler course in a college or university environment. LOGGRASM will work fine with an old program, but you will need to retrofit the old program to include LOGGRASM services.

LOGGRASM provides a series of standardized functions for user program entry. LOGGRASM allows a called program also using Logger Services to inherit the LCA storage from the calling program also using Logger Services. The use of LOGGRASM frees the subordinate lower level programs from issuing constant Getmain requests to acquire dynamic storage in order to maintain re-entrant status. LOGGRASM performs linkage processing, generates log points, defines the entry and exit protocols, sets up programs to be re-entrant, performs basic parameter validation, establishes program addressability, and defines a program prefix area in the Logger Control Area (LCA). The Logger Control Area (LCA) is created by the initial call to Logger Services, and allows a called program also using Logger Services to pick up a dynamic storage area from the calling program.

This means Assembler programs using Logger Services are always re-entrant because LOGGRASM automatically sets up your program to be that way. The proper entry and exit housekeeping is done for you regardless of your addressing mode. LOGGRASM will automatically assign the base registers, save the registers of the calling program, obtain a save area, set up R13 to contain the address of the save area, set the forward chain and back chain pointers for the save area addresses, and set up the Logger Services environment to support the generation of log points specified by the user in their Assembler program.

The Logger Service macros do not use in-line work areas, the macros do not require any MF=E or MF=L generations to maintain re-entrant status, and do not require an IEBUPDTE process. All registers from the user’s Assembler program including R0, R1, R14, and R15 remain non-volatile across the call to Logger Services. Logger Services use the RSECT instruction to tell the Assembler to check for non-reentrant code in a control section during the assemble of your program.

There is a prefix area for each program's dynamic storage area which is defined by Logger Services. This prefix area is owned by the program entry (LPGMNTRY) and exit (LPGMEXIT) process of Logger Services. The first 216 bytes in the prefix area is a register save area to support either an OS/390 72-byte standard save area (24/31), or a z/Architecture 144-byte Format-4 save area, or a z/Architecture 216-byte Format-5 save area (64-bit split).

Upon program program entry for your main user program where you specify LPGMNTRY TYPE=MAIN, Logger Services has no control of the caller in regards to what save area format the caller may provide, and Logger Services assumes a 72-byte save area even if the caller provided a 144-byte save area. There is no requirement that the caller provide a format id (e.g., FS1A, FS4A, FS5A) in bytes 4-7 of the save area, and you cannot rely on one being there except at your own peril. So the Logger services protocol on main program entry will be to save the low halves of the caller's registers in the caller provided 72-byte save area, and save the high halves of the caller's general purpose registers in the called program's save area. The called program will be your user program that was setup automatically by Logger Services with a F5SA format save area within the get-main'ed LCA to maintain re-entrancy. Logger Services will then ensure that the high halves of the general purpose registers remain unchanged upon return to the calling program. When LPGMNTRY TYPE=SUB, Logger Services has control of the caller, and the caller's registers are saved in the caller's 144 byte FS4A format save area within the caller's LCA.

Next the save area in the Logger Services LCA is followed by additional variables which define the LCA stack, and is then followed by an 80-byte carry-over storage area that includes definitions to support Logger Services. Whenever a program using Logger Services is called these 80-bytes are copied from the caller's dynamic area into the called program's dynamic area by LOGGRASM protocols. This allows the 80-byte data area initialized in the LCA prefix area by the Main program to be carried over to include all other called programs (LPGMNTRY TYPE=SUB) in order to support Logger Services. With the exception of the LCA prefix area, all storage in the dynamic area is cleared to binary zeros each time a program is entered ensuring that all data items in the LCA are initialized to zeros to ensure a clean storage area upon entry to a main program or called sub-program using Logger Services.

LOGGRASM offers a method in which to imbed Logger Services into a user's Assembler program, support processes which are performed automatically, and to allow Logger Services to exist transparently within the source Assembler program code. Logger Services provide a series of standardized functions for program entry. All program state information, including the condition code is maintained over the call. Logger Services may be used in any AMODE and in any RMODE with the exception that no executable code can run above the bar. With Logger Services no registers or storage areas are modified or affected in any manner. No SMPE install, hooks, SVCs, 0C1 traps, front-ends, IEBUPDTE, authorized libraries, or system environmental changes are required to use LOGGRASM.

LOGGRASM service programs always execute in 64-bit addressing mode. User programs may operate in a different addressing mode from that of LOGGRASM without restriction. LOGGRASM will return in the addressing mode of the user program. When you use 31-bit or 64-bit addressing mode, you must decide whether your new program should reside above or below the 16 megabyte line unless it is so large that it will not fit below. Your decision depends on what programs and system services the new program invokes and what programs may invoke it. Make every effort to have your programs reside above-the-line. If you need to invoke IBM service routines that use system services requiring residence below the line, then you can use the LCA in your program provided by LOGGRASM to build the execute form of the IBM service macros to invoke your service below-the-line which will also maintain the re-entrant status of your program. Otherwise you can perform your own Getmain for a separate storage area below-the-line. Then utilize the list and execute forms of the macro to invoke your service in your area below-the-line which will also maintain the re-entrant status of your program. However, if you are required to have your program reside below-the-line because you are utilizing services that must accept control directly from unchanged 24-bit addressing mode programs, then your program must reside below-the-line. If your program resides above-the-line in 31-bit or 64-bit addressing mode, then you must include specific logic in your own program to support the ability to accept control directly from unchanged 24-bit addressing mode programs.

As z/OS support matures, AMODE 64 toleration and AMODE 64 enabled z/OS interfaces will continue to grow.

LOGGASM uses a "log point" as a means for interrogating the execution of a user's Assembler program, and causing control to pass to Logger Services. Log points are created at user program entry and user program exit, and log points can be defined by the user, and placed essentially anywhere in a user program's execution path. When a log point is executed, control is passed to LOGGRASM resulting in user program information being saved. LOGGRASM creates log points in a user's program through macros which generate small areas of code in-line to the source at assembly time. Upon user program execution and the user program encountering a log point defined by the user, LOGGRASM captures user program information and shows this information by writing records to the //LGRECOUT DD data set. This offers a method to provide complete debugging of your program.

A lot of information is required to understand and then fix a problem or a failure. LOGGRASM in addition to allowing you to see the logic flow of your program can display register contents, data fields, control block structures, and set storage watches. However, even this can be of limited value because you also have to more or less look forward to understand the logic of your entire program, and not just see the immediate logic path that is in error in order to get the full benefit of log points. However, sometimes with students it is helpful to work backwards and have them see what happened through a log point. The instructor can then go back and explain to them what they see displayed in a Logpoint in order to help them to understand the logic of what they coded or the manner in which they used an instruction.

One of the goals here was to prevent a condition where if you set a log point at an particular place in your Assembler program, you don't encounter a situation that when you look at the data you shouldn't have made it there because a log point had changed the data in the meantime.

One of the goals here was that LOGGRASM passes no information about its processing back to the user program. It displays its information in the //LGRECOUT output data set. Logger Services does not modify the return code or pass one back even if it abends allowing the user program to make branch or termination decisions based solely on its own processing.

One of the goals here with LOGGRASM was to allow students to get started early with coding a complete (although simple) Assembler program that will assemble and execute. The install of LOGGRASM is basically upload copybooks, JCL, and some source, assemble to produce objects, and go. Getting students involved in programming Assembler at an early stage of a class helps gets their attention, their interest, their buy-in, and their commitment to the subject. Instructors can then guide students in learning proper Assembler programming techniques from the beginning, before bad habits are developed.

One of the goals here was to reduce the need to use DUMP or SNAP or ABEND macros, or use WTO's, or set DC H'0' or EX R0,* snares and traps with a SYSMDUMP DD or SYSABEND DD. These techniques are still available if you desire to use them, or to teach students how to use them. The use of PER/GTF/SLIP assumes access to operator commands where students will not have that level of authority. LOGGRASM attempts to help students who frequently learn in an environment where they may not have access to HLASM IDF, HLASM Toolkit, XDC, Xpeditor, IPCS, ABENDAID, Dumpmaster, or Fault Analyzer, their use of memory is throttled back, and their disk space allocations may be limited.

One thing for college instructors to take note is that LOGGRASM by its design hides a fair amount of Assembler program setup, housekeeping (e.g., save-area linkage), and re-entrant logic from the students in order to get them started in Assembler. However, it is important to explain these concepts from the beginning, and then gradually introduce Assembler instructions to get students to build working programs that actually solve a problem.

One of the goals here was no assist type commands were designed into LOGGRASM to help students with reading a data set, writing to a data set, performing decimal conversions, finding the end of a line, or other like functions. This is to get students from the beginning to use real Assembler instructions to better understand how their Assembler program is working, and get a better handle on what the mainframe hardware is doing. This helps students understand that just knowing some Assembler does not make you a Systems Programmer or a developer. The idea is to get students to use Assembler instructions where they can commit to what they learn at the time that they learn it, and later in a second semester course not be in a position where they have to back out what they learned or unlearn a command that did things for them when faced with having to convert to the concept that one Assembler instruction is one machine instruction, millicode notwithstanding. The intent with LOGGRASM is to allow students with greater ease to see the specific results of the use of an instruction whether for better understanding of the language or for debugging an error.

With the help of JCL examples supplied to the student, they need to do a simple build of the JCL to perform the assembly, get a listing, and produce the object. Encourage students to look at the listing of a simple Assembler program, and have them play Assembler by 'walking' the location counter on the left hand side of the Assembler listing. Then with help from JCL examples build the JCL to do the link-edit and run the program to understand the Binder and the loader. Then as part of your curriculum in your second-semester course expand the discussion on save-area linkage and re-entrant logic, include a basic introduction to IPCS dump reading, and go into a little more depth on the hardware.

LOGGRASM incorporates an AMODE64 ESTAEX recovery exit to intercept abends which may occur in Logger Services when the user program at a log point specifies invalid data. A Specify Task Abnormal Exit Extended (ESTAEX) is issued by the Logger Service program LGMHLRCI that defines a program named LGMHESTA as the Logger Services abend error recovery routine. When program LGMHLRCI previously issued the ESTAEX macro, the Recovery Termination Manager created an SCB (STAE Control Block) containing a pointer to the LGMHESTA exit routine. The addresses of the SCB's are chained on a LIFO stack which is anchored by the TCBSTABB field off the TCB. LGMHESTA should be the newest ESTAEX-type routine in existence at the moment that an abend occurs or that a log point is reached. If another ESTAEX macro has been issued by the user program, it will be more recent than the ESTAEX macro that was issued to setup Logger Services. This is due to the LIFO stack order of the SCB's. Just be aware that when an abend occurs or when a log point is executed which itself may be subject to an abend, the newer ESTAEX will receive control ahead of Logger Services which may result in the abend being handled differently than what Logger Services is expecting depending on how you may have set up your own recovery exit.

The LOGGRASM ESTAEX is only for recovering from possible abends in Logger Services, and not for recovering from abends which may occur in your user program. Students should see the consequences of their programming errors in what they code, and take the abend and the dump. They will learn more that way as they will be forced to look up z/OS system completion codes and attempt to interpret their meaning. If they get stuck, then the instructor can explain why their program ended up with a dead stick and no rudder and crashed. Then students can insert Logpoints in their program to diagnose the error.

If a student makes a coding error in their program leading to an abend, LOGGRASM will still intercept the abend. However, this is done only to allow LOGGRASM to produce a diagnostic report before proceeding with percolation of the abend. Upon completion of the diagnostic report, LOGGRASM will set the ABEND macro with the original system completion code and reason code from the student's program if available, and then the student's program will terminate.

However, if an abend occurs in Logger Services due to the student specifying a Logpoint with an invalid address or a bad length on a SHOW= parm or on a WATCH= parm, Logger Services will intercept the abend and fully recover from the abend. The abend condition will be reported in the //LGRECOUT output data set.

Upon an abend Logger Services will issue a message that an abend has occurred, display the place in the user program where the Log Point was defined, indicate the type of activity Logger Services was performing, and show the address it was attempting to reference (within a 4-byte range) which resulted in an abend. Logger then returns to the user program and normal execution continues. Students need to be aware of these messages, and go back to find out what happened. For example when a #LGPOINT was coded was an invalid address specified or an improper length supplied for a data area? When a #LGPOINT was coded was an address specified that was valid at the time, and after several iterations through the program path due to a design bug in the user program the storage area was released too soon when it was still needed?

If a user program bug is suspected the user can review their existing //LRECOUT output, add new Log Points in their Assembler code based on what is observed in the previous output report, re-assemble and relink, and run the program again to recreate the problem with the benefit of having additional diagnostic output from the new Log Points.

Example of message issued when a Log Point abends:

```
Log) ==>Log Point**: ABCD0200
   ???> An Abend Occurred Referencing Storage Data at Address
   000000004FDECA30
```

Logger Services is designed to be used in the development of new Assembler programs. Logger Services does not support the display of data in dataspaces. If you were to write a new Assembler program and you needed large amounts of storage, you should obtain storage above-the-bar instead of from a dataspace unless there is a compelling reason to use a dataspace. With the advent of above-the-bar storage you should not need to use dataspaces in the design of future code. All your future design efforts should be biased towards thinking only in terms of 64-bit.

However, dataspaces are perfectly valid and there is nothing wrong with using dataspaces. For existing code the use of dataspaces are acceptable, dataspaces are still supported, and the current design of the existing code may be dependent upon them (i.e., you cannot map a VSAM LDS above-the-bar). Also for the existing code the use of dataspaces may still satisfy all storage requirements where changing to above-the-bar would be a wasteful, expensive, and unnecessary conversion effort.

12.0 Appendix

Disclaimer:

A reasonable effort has been made to ensure the accuracy, completeness, and correctness of LOGGRASM processing. However, LOGGRASM is distributed on an "as is" basis, without any expressed or implied warranty of any kind. Use entirely at your own risk.

Permission is granted to the recipient to make, distribute, or print unlimited copies, to alter or modify the supplied source code in any manner they wish to suit their personal needs, and/or use the material for personal use only. Teachers, instructors, faculty, and academic institutions may use as many copies of this material in any manner they wish as are needed for any college or university course requirements. All other rights reserved.

If you should have any suggestions for improvement or to report any errors, you may send comments or questions to:

mailto:khf00@sbcglobal.net

mailto:sbgolob@cbttape.org

Quotes:

"The greatest artist was once a beginner."

[Unknown]

"May we continue to exist and enjoy what we do."

Carmine A. Cannatello, "Advanced Assembler Language and MVS Interfaces", 1999

"No data yet... It is a capital mistake to theorize before you have all the evidence. It biases the judgment."

Arthur Conan Doyle, "A Study in Scarlet", 1887

Heard on the street:

Why ever use a mainframe?

"There are just some things you can do more efficiently with a bulldozer than a thousand people with shovels."

What is a mainframe?

"The mother of all servers."

References:

z/OS V1R13 MVS Programming: Assembler Services Guide SA22-7605-14

z/OS V1R13 MVS Programming: Assembler Services Reference, Volume 1 (ABEND-HSPSERV) SA22-7606-12

z/OS V1R13 MVS Programming: Assembler Services Reference, Volume 2 (IAR-XCT) SA22-7607-17

z/OS V1R13 MVS Programming: Authorized Assembler Services Guide
SA22-7608-16

z/OS V1R13 MVS Programming: Authorized Assembler Services Reference, Volume 1 (ALESERV-DYNALLOC) SA22-7609-12

z/OS V1R13 MVS Programming: Authorized Assembler Services Reference, Volume 2 (EDTINFO-IGXWRITE) SA22-7610-19

z/OS V1R13 MVS Programming: Authorized Assembler Services Reference, Volume 3 (LLACOPY-SDUMPX) SA22-7611-13

z/OS V1R13 MVS Programming: Authorized Assembler Services Reference, Volume 4 (SETFFR-WTOR) SA22-7612-13

z/OS V1R13 MVS Programming: Extended Addressability Guide SA22-7614-08

z/OS V1R13 MVS JCL Reference SA22-7597-15.

z/Architecture Principles of Operation SA22-7832-08

High Level Assembler Programmer's Guide, Release 6 SC26-4941-05

The above are IBM publications may be referenced through the z/OS Website at:

http://www-03.ibm.com/systems/z/os/zos/software/index.html

http://www-03.ibm.com/systems/z/os/zos/bkserv/

Carmine A. Cannatello
Advanced Assembler Language and MVS Interfaces, Second Edition
John Wiley & Sons, Inc, 1999 ISBN 0-471-36176-3

Professor Spotwood D. Stoddard, University of Nebraska - Lincoln
Principles of Assembler Language Programming for the IBM 370
McGraw-Hill Book Company, 1985 ISBN 0-07-061561-6

Professor Donald E. Knuth, Stanford University
The Art of Computer Programming, Volume 3, 2nd Edition
Sorting and Searching
Addison-Wesley Publishing, 1998. ISBN 0-201-89685-0

Department of Computer Engineering
School of Engineering
Santa Clara University
Santa Clara, CA 95053

IBM, HLASM, z/OS, z/Architecture, and zEnterprise are trademarks or registered trademarks of International Business Machines Corporation.

For Students:

IBM Redbooks: Introduction to the New Mainframe: z/OS Basics SG24-6366-02

www.redbooks.ibm.com/abstracts/sg246366.html

CBT:

CBT Tape is an MVS-related software repository containing many utilities which can potentially offer much help to students and new programmers. Even if you do not use the utilities, in reviewing the CBT Tape contents it will illustrate many Assembler coding examples which can be a good learning experience for students and new programmers.

http://www.cbttape.org

zNextGen:

zNextGen is a cooperative effort from both SHARE and IBM. It is oriented towards students and new IT professionals working with mainframe related technologies.

http://www.share.org

IBM System z Academic Initiative:

The IBM Academic Initiative is an offering by IBM to assist colleges and universities with educational resources for their computer science curriculum in order to support expanded course offerings in mainframe related computer technologies.

http://ibm.com/university/systemz
https://www.ibm.com/developerworks/university/systemz/

IBM z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is intended to assist students and new users with learning the basic concepts of z/OS. z/OS is the operating system that runs most of the IBM mainframe computers in use today. This Web based resource will introduce you to the basic concepts of the z/OS mainframe operating system.

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp

For more information on z/OS topics, there is the IBM-MAIN mailing-list/newsgroup. You can subscribe to the IBM-MAIN mailing list. Some commonly asked questions about using the IBM-MAIN mailing-list/newsgroup can be found at the following Web site. Though it is not a site officially endorsed by the IBM-MAIN list provider, it does provides basic information on using this valuable IBM-MAIN reference site.

http://planetmvs.com/ibm-main/faqg.html