# CICS Application Dump Reading

Russ Evans
russevans@evansgroupconsulting.com
The Evans Group, Inc.
Session 1068

# Objectives

- Why
- What is Available
- Controlling Dump and Trace
- AEI0 Abend
- ASRA Abend

# Why Read Dumps?

- Detailed snapshot of time of failure
- Sometimes difficult to recreate problem
- Sometimes dangerous to recreate problem
- Only way to identify problem with certainty
- Can be faster and easier than recreating
- Dumps are free!

# What Tools Are Available?

- CICS Transaction Dump
- CICS Aux Trace
- Language Environment Dump and Messages
- CICS System Dump

# Useful Resources

Before starting the debugging process, you will need:

- Compile listing with the LIST option
- Link map of the program load module with MAP and XREF
- CICS Application Dump

# Useful Resources (continued)

Before starting the debugging process, you may want:

- CICS Diagnosis Reference manual
  - Trace point layouts
- CICS Application Programming Reference
  - Command options
  - Response and reason codes
- CICS System Programming Reference
  - Command options
  - Response and reason codes
  - For commands outside the standard API
- CICS Information Center
  - All of the above
  - Allows searching across all manuals

# Looking at the Dump

The title of the dump contains identifying data:

```
CICSD224   --- CICS TRANSACTION DUMP ---  CODE=AEIM
TRAN=ESDM   ID=1/0047    DATE=08/01/10
TIME=05:26:18
```

Review to ensure you have the correct dump:

- VTAM APPLID of the region
- Trans ID of the abending transaction
- Abend Code
- Date and Time

# Looking at the Dump (continued)

```
REGISTERS AT LAST EXEC COMMAND
REGS 0-7   18326158  18326728  18305730  18326754        183038D0  18326158  1832616C   18326168
REGS 8-15  1832615C  1832673A  00031088  00032087        18307BC8  18326690  8003131C   00000000
```

Do NOT rely on these registers!

In this dump, they are from the last LE EXEC command issued while creating the CEEMSG dump, not from the user program.

You can test the reliability of these registers by checking to see if R14 points into your program.

# Looking at the Dump (continued)

The transaction environment section displays a variety of useful information...

*So, what's a start code, anyway?*



```
Transaction environment for transaction_number(0000783)
 transaction_id(ESDM)       orig_transaction_id(ESDM)
 initial_program(ESPYDEMO) current_program(ESPYDEMO)
 facility_type(TERMINAL)   facility_name(CP92)           Start_code(TP)
 netname(S01TCP92)         profile_name(ESPYPRF )
 userid(CICSD224)          cmdsec(NO)                    ressec(NO)
 spurge(NO)                dtimeout(0000000)             tpurge(NO)
 taskdatakey(USER)         taskdataloc(BELOW)
 twasize(00000)            twaaddr(        )
 remote(NO)                dynamic(NO)
 priority(001)             Tclass(NO)                    runaway_limit()
 indoubt_wait(YES)         indoubt_wait_mins(000000)
 indoubt_action(BACKOUT)   cics_uow_id(C1C6E7031297F846) confdata(NO)
 system_transaction(NO)    restart_count(00000)          restart(NO)
```

# Looking at the Dump (continued)

D : The task was initiated to process a distributed programming link (DPL) command that did not specify the SYNCONRETURN option. (The task is not allowed to issue syncpoints.)

DS: The task was initiated to process a distributed programming link (DPL) command containing the SYNCONRETURN option. (The task is allowed to issue syncpoints).

QD: CICS initiated the task to process a transient data queue that had reached trigger level.

S: Another task initiated this one, using a START command that did not pass data in the FROM option. The START command may or may not have passed a channel.

SD: Another task initiated this one, using a START command that passed data in the FROM option.

SZ: The task was initiated with a FEPI START command (see the CICS Front End Programming Interface User's Guide for further information).

TO: The task was initiated to process unsolicited input from a terminal (or another system), and the transaction to be executed was determined from the input.

**TP: The task was initiated to process unsolicited input or in response to a RETURN IMMEDIATE command in another task. In either case, the transaction to be executed was preset (in the RETURN command or in the associated TERMINAL definition) without reference to input.**

U: CICS created the task internally.

(From the CICS Systems Programming Reference manual)

# Looking at the Dump (continued)

## Analysis of the Exec Interface Block

```
EXEC INTERFACE BLOCK
00000000   0052614F 0108010F C5E2C4D4 0000783C  C3D7F9F2 000000EA 00047D06 02810000
00000020   000000C5 E2D7E8C3 E3D34000 00000000  000000C5 E2D7E8C3 E3D34000 00000000
00000040   00000000 00000000 00000000 0000000D  00000050 00
```

```
  0000 EIBTIME   DS    PL4 05:26:14
  0004 EIBDATE   DS    PL4 2008 010*
  0008 EIBTRNID  DS    CL4 ESDM
  000C EIBTASKN  DS    PL4
  0010 EIBTRMID  DS    CL4 CP92
  0014 EIBRSVD1  DS    H
  0016 EIBCPOSN  DS    H
  0018 EIBCALEN  DS    H   0004
  001A EIBAID    DS    CL1
  001B EIBFN     DS    CL2 0602*
  001D EIBRCODE  DS    CL6 810000…
  0023 EIBDS     DS    CL8 ESPYCTL
  002B EIBREQID  DS    CL8
  0033 EIBRSRCE  DS    CL8 ESPYCTL
  003B EIBSYNC   DS    C
  003C EIBFREE   DS    C
  003D EIBRECV   DS    C
  003E EIBSEND   DS    C
  003F EIBATT    DS    C
  0040 EIBEOC    DS    C
  0041 EIBFMH    DS    C
  0042 EIBCOMPL  DS    C
  004C EIBRESP   DS    F   0000000D*
  0050 EIBRESP2  DS    F   00000050*
```

# Looking at the Dump (continued)

Interpreting values in the Exec Interface Block

| Field Name | Value | Meaning |
|---|---|---|
| EIBFN | 0602 | Read |
| EIBDS | ESPYCTL | DD name of file read |
| EIBRCODE | 81 | NOTFND (Record not found) |
| EIBRESP | 0D | NOTFND (Record not found) |
| EIBRESP2 | 50 | Record not found |

# Looking at the Dump (continued)

## About EIBDATE

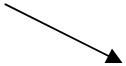
0108010F

EIBDATE is in format 0 C YY DDD F where:

- 0 is a place holder
- C indicates century:
  - “1” indicates century 20
  - “0” indicates century 19
- YY DDD is the Julian date
- F makes it a positive packed number

# Looking at the Dump (continued)

When viewing the trace table, look first at EIP ENTRY/EXIT pairs



```
000298 QR    AP 00E1 EIP   ENTRY WRITEQ-TD                                    =000019=
000298 QR    DD 0301 DDLO  ENTRY LOCATE                08A4DBC0,00059AC7,DCTE,CESE  =000020=
000298 QR    DD 0302 DDLO  EXIT  LOCATE/OK             08ABE270 , C4C3E3C5          =000021=
000298 QR    AP F600 TDA   ENTRY WRITE_TRANSIENT_DATA  CESE,08AEB2E0 , 00000001,YES  =000022=
000298 QR    DD 0301 DDLO  ENTRY LOCATE                08A4DBC0,00070BE4,DCTE,CESE  =000023=
000298 QR    DD 0302 DDLO  EXIT  LOCATE/OK             08ABE270 , C4C3E3C5          =000024=
000298 QR    AP F601 TDA   EXIT  WRITE_TRANSIENT_DATA/OK                            =000025=
000298 QR    AP 00E1 EIP   EXIT  WRITEQ-TD             OK                           =000026=
```

Remember, the most recent trace entry is the first

# Looking at the Dump (continued)

The expanded trace entries have more details….

```
AP F600 TDA  ENTRY - FUNCTION(WRITE_TRANSIENT_DATA) QUEUE(CESE) FROM_LIST(08AEB2E0 , 00000001) RSL_CHECK(YES)
 TASK-00298 KE_NUM-0014 TCB-QR   /008CFE88 RET-88CE76F4 TIME-11:52:29.0992101250 INTERVAL-00.0000019062     =000022=
  1-0000  00500000 00000035 00000000 00000000  BC200000 00000000 01AE0103 C3C5E2C5 *.&.........................CESE*
    0020  08AEB2E0 00000001 00000002 08A83070  01004000 02201101 00680000 00000028 *...\.........y.... .............*
    0040  00000000 01000000 B5C00000 00000000                                     *.........{......                *
  2-0000  40C3D7F0 F2C1D7C3 E340F2F0 F0F3F0F9  F1F7F1F1 F5F2F2F9 40404040 40404040 * CP02APCT 20030917115229        *
    0020  404EF0F0 F0F040F0 F9F2F0F6 F7C2F840  40F0F0F0 F0F0F0F0 F040F0F9 F2F0F6F5 * +0000 092067B8  00000000 092065*
    0040  C2F040F0 F9F2F0F6 F8F3F840 F8F9F5F1  F5C5F4F2 4040F0F9 F5F1F6F0 F5F840F0 *B0 09206838 89515E42  09516058 0*
    0060  F9F2F0F3 F7F5F040 F0F8C1C5 C1C6C4F4  40F0F9F2 F0F6F8F3 F840404F 4B4B4B4B *9203750 08AEAFD4 09206838  |....*
    0080  4B4B4B4B                                                               *....                            *
```

# Looking at the Dump (continued)

The Program Information section of the dump contains information related to all of the programs currently active in the transaction.

If the transaction has issued EXEC CICS LINKs, then each logical link level will be displayed.

This section has the most recent entries first, and is read from the last entry backward.

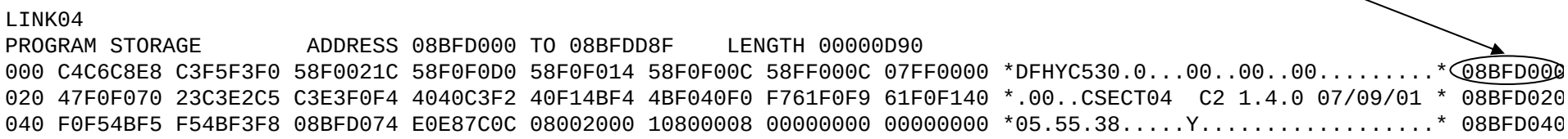
```
PROGRAM INFORMATION FOR THE CURRENT TRANSACTION
   Number of Levels 00000004

INFORMATION FOR PROGRAM AT LEVEL 00000003 of 00000004
   Program Name     LINK03       Invoking Program LINK02
   Load Point       098692A0     Program Length   00000C70
   Entry Point      898692C0     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced Unknown
   Commarea Address 09211650     Commarea Length  0000000A
   Execution Key    USER         Data Location    ANY
   Environment      User application
INFORMATION FOR PROGRAM AT LEVEL 00000002 of 00000004
   Program Name     LINK02       Invoking Program ABEND1
   Load Point       093FDF70     Program Length   00000C70
   Entry Point      893FDF90     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced Unknown
   Commarea Address 09208650     Commarea Length  00000009
   Execution Key    USER         Data Location    ANY
   Environment      User application
INFORMATION FOR PROGRAM AT LEVEL 00000001 of 00000004
   Program Name     ABEND1       Invoking Program CICS
   Load Point       093FC260     Program Length   00001C70
   Entry Point      893FC280     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced Unknown
   Commarea Address 00000000     Commarea Length  00000000
   Execution Key    USER         Data Location    BELOW
   Environment      User application
```

# Looking at the Dump (continued)

```
PROGRAM INFORMATION FOR THE CURRENT TRANSACTION
   Number of Levels 00000004
INFORMATION FOR PROGRAM AT LEVEL 00000004 of 00000004
   Program Name     LINK04       Invoking Program LINK03
   Load Point       08BFD000     Program Length   00000D90
   Entry Point      88BFD020     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced Unknown
   Commarea Address 09217C70     Commarea Length  0000000A
   Execution Key    USER         Data Location    ANY
   Environment      User application
```

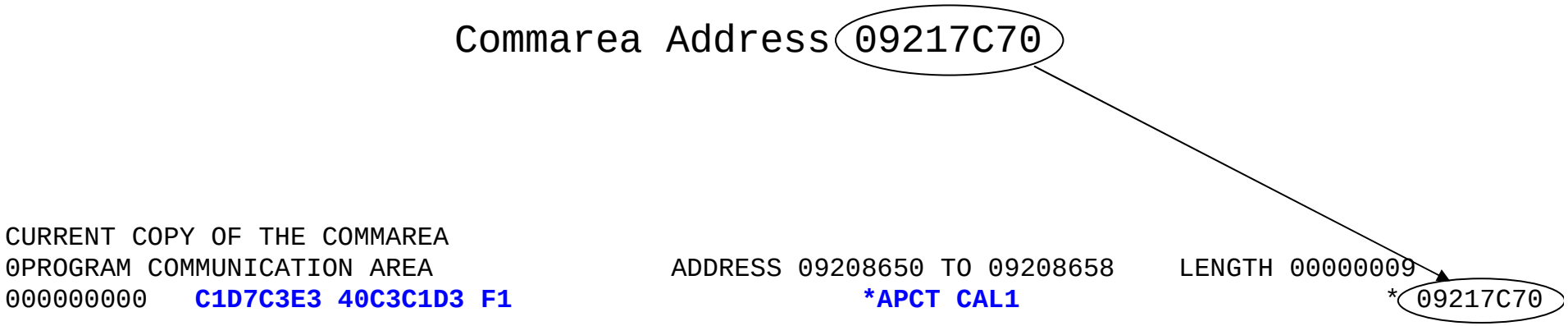
# Looking at the Dump (continued)

Load Point 08BFD000



```
LINK04
PROGRAM STORAGE        ADDRESS 08BFD000 TO 08BFDD8F    LENGTH 00000D90
000 C4C6C8E8 C3F5F3F0 58F0021C 58F0F0D0 58F0F014 58F0F00C 58FF000C 07FF0000 *DFHYC530.0...00..00..00.........* 08BFD000
020 47F0F070 23C3E2C5 C3E3F0F4 4040C3F2 40F14BF4 4BF040F0 F761F0F9 61F0F140 *.00..CSECT04  C2 1.4.0 07/09/01 * 08BFD020
040 F0F54BF5 F54BF3F8 08BFD074 E0E87C0C 08002000 10800008 00000000 00000000 *05.55.38.....Y.................* 08BFD040
```

# Looking at the Dump (continued)



# Printing CICS Dumps

- For transaction dumps only.  Use IPCS to read system dumps
- Program name specific to CICS release producing the dump
  - Includes the “internal” release number, not CICS/TS
- Use to format and print the DFHDMPA and DFHDMPB datasets
- Active dump dataset controlled by CEMT I DUMP
  Must de-activate dump dataset before printing
- Control transaction dumps via CEMT S TRD

# Controlling CICS Dumps

Use CEMT I DUMP to switch the dump datasets prior to printing

```
I DUMP
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Dum Cur(A) Ope
```

# Printing CICS Dumps

(continued)

Program name must match CICS release

```
// EXEC PGM=DFHDU640
//STEPLIB  DD
   DSN=SYS2.CICSTS31.CICS.SDFHLOAD,DISP=SHR
//DFHTINDX DD SYSOUT=X
//SYSPRINT DD SYSOUT=X
//DFHPRINT DD SYSOUT=X
//DFHDMPDS DD DISP=SHR,DSN=CICS.DFHDMPA
//SYSIN    DD *
```

DFHTINDX receives dump index output

DFHDMPDS is the dump dataset to be formatted.

If no control statements are needed, use //SYSIN DD DUMMY

# Printing CICS Dumps (continued)

## Useful Dump Control Statements

- TRANID=(xxxxx)
  Limits output to one or more specific transactions
  TRANID may be wild carded
- DUMPCODE=(xxxx)
  Limits output to one or more specific abend types
- TIME=(hh:mm)
  Limits output to dumps produced at a specific time or time range.

See Operations and Utilities Guide for syntax and details!

# Solving An AEI0 Abend

- Unhandled errors result in AEIx abend
- Much useful information in EIB fields and Trace Table
- CICS Messages and Codes provides clear description of error
- CICS Application Programming Reference Manual describes conditions each command can receive

# Solving An AEI0 Abend

(continued)

**From the CICS Messages and Codes manual:**

> **Explanation:** PGMIDERR condition not handled.
>
> This is one of a number of abends issued by the EXEC interface program. Because of their similar characteristics these abends are described as a group.
>
> See the description of abend AEIA for further details.

Note that this information can also be found using the CMAC transaction.

# Solving An AEI0 Abend

(continued)

Because an AEI* abend is the result of a failure in an EXEC CICS command, the first place to look is the EIB:

```
EIBTIME  DS    PL4 0115228C
EIBDATE  DS    PL4 0103260F
EIBTRNID DS    CL4 APCT
EIBTASKN DS    PL4 0000298C
EIBTRMID DS    CL4 CN02
EIBRSVD1 DS    H
EIBCPOSN DS    H
EIBCALEN DS    H
EIBAID   DS    CL1
EIBFN    DS    CL2 0E06
EIBRCODE DS    CL6
```

```
EIBDS    DS    CL8
EIBREQID DS    CL8
EIBRSRCE DS    CL8 ESPYDEMX
EIBSYNC  DS    C
EIBFREE  DS    C
EIBRECV  DS    C
EIBSEND  DS    C
EIBATT   DS    C
EIBEOC   DS    C
EIBFMH   DS    C
```

EIBFN = 0E06 = LOAD

EIBRSRCE = ESPYDEMX

# Solving An AEI0 Abend

(continued)

Another good place to look for the cause of AEI* abends is the trace table:

- Shows command that failed
- Provides detail on the failure
- Provides the R14 value to the program, identifying the failing line of code

# Solving An AEI0 Abend

(continued)

## An internal trace of the pgmiderr

```
000158 QR    DD 0301 DDLO  ENTRY LOCATE                160EEF80,0007C364,DCTE,CE
000158 QR    DD 0302 DDLO  EXIT  LOCATE/OK             1701A150 , C4C3E3C5
000158 QR    AP F601 TDA   EXIT  WRITE_TRANSIENT_DATA/OK
000158 QR    AP 00E1 EIP   EXIT  WRITEQ-TD             OK
000158 QR    AP 00E1 EIP   ENTRY WRITEQ-TD
000158 QR    DD 0301 DDLO  ENTRY LOCATE                160EEF80,0005D4C7,DCTE,CE
000158 QR    DD 0302 DDLO  EXIT  LOCATE/OK             1701A150 , C4C3E3C5
000158 QR    AP F600 TDA   ENTRY WRITE_TRANSIENT_DATA  CESE,16FF2DF0 , 00000001,
000158 QR    DD 0301 DDLO  ENTRY LOCATE                160EEF80,0007C364,DCTE,CE
000158 QR    DD 0302 DDLO  EXIT  LOCATE/OK             1701A150 , C4C3E3C5
000158 QR    AP F601 TDA   EXIT  WRITE_TRANSIENT_DATA/OK
```

Unfortunately, this CICS region was running with TERMTHDACT=TRACE.

# View and Modify LE options in CICS

Use transaction CLER to view and modify many of the LE runtime options in CICS

```
CLER                                                              D224
   CICSD224
               Language Environment Region Level Run-time Options

Type in your Choices.

Runtime option     Choice      Possible choices.

ALL31          ==>  ON          ON, OFF
CBLPSHPOP      ==>  ON          ON, OFF
CHECK          ==>  ON          ON, OFF
INFOMSGFILTER ==>  OFF         ON, OFF - ON equates to
   INFOMSGFILTER(ON,CICS)
RPTOPTS        ==>  OFF         ON, OFF
RPTSTG         ==>  OFF         ON, OFF
TERMTHDACT     ==>  quiet
   QUIET,MSG,TRACE,DUMP,UAONLY,UATRACE,UADUMP,UAIMM
TRAP           ==>  ON          ON, OFF

When finished, press ENTER.
      PF1=Help    3=Quit    5=Current Settings   9=Error List
```

# Solving An AEI0 Abend

(continued)

An aux trace of the pgmiderr with TERMTHDACT(QUIET)

```
AP 00E1 EIP   ENTRY LINK                                    0004      =000030=
PG 1101 PGLE  ENTRY LINK_EXEC            ESPYDEMX,1850BCF8 , 00000064,NO,      =000031=
DD 0301 DDLO  ENTRY LOCATE               16029E60,16FF2DBC,PPT,ESPYDEMX        =000032=
DD 0302 DDLO  EXIT  LOCATE/OK            D7D7E3C5 , 17287A80                   =000033=
PG 1102 PGLE  EXIT  LINK_EXEC/EXCEPTION  PROGRAM_NOT_LOADABLE,,,,              =000034=
PG 0700 PGHM  ENTRY INQ_CONDITION        1B =000030=
PG 0701 PGHM  EXIT  INQ_CONDITION/OK     AEI0,00000000,00000000,,0,SYSTEM      =000035=
AP 00E1 EIP   EXIT  LINK                 PGMIDERR                    00F4      =000036=
```

Recommendation: set **TERMTHDACT(QUIET)** or **(MSG)** to keep LE from overwriting your trace table

# Solving An AEI0 Abend

(continued)

## Another helpful trace item

```
00E1 EIP ENTRY LINK                          REQ(0004) FIELD-A(18508978 .&i.) FIELD-B(09000E02 ....)

          TASK-00162 KE_NUM-0089 TCB-QR   /008D2658 RET-96D55BFC TIME-11:16:37.9689106572
```

The RET value shown in the EIP ENTRY trace is the address of the instruction that follows the call to CICS for this command. We can use it to find the COBOL source statement that issued the EXEC CICS LINK.

Note that not all commands are issued by your program! LE clutters the trace table with its own commands. The RET value for these commands will point to an LE program.

# Identifying the EXEC CICS Statement That Failed

Step 1: Find the load point of the program.

Step 2: Identify the CSECT in question

Step 3: Determine the offset of the return into the CSECT.

Step 4: Identify the COBOL source statement.

# Identifying the EXEC CICS Statement That Failed

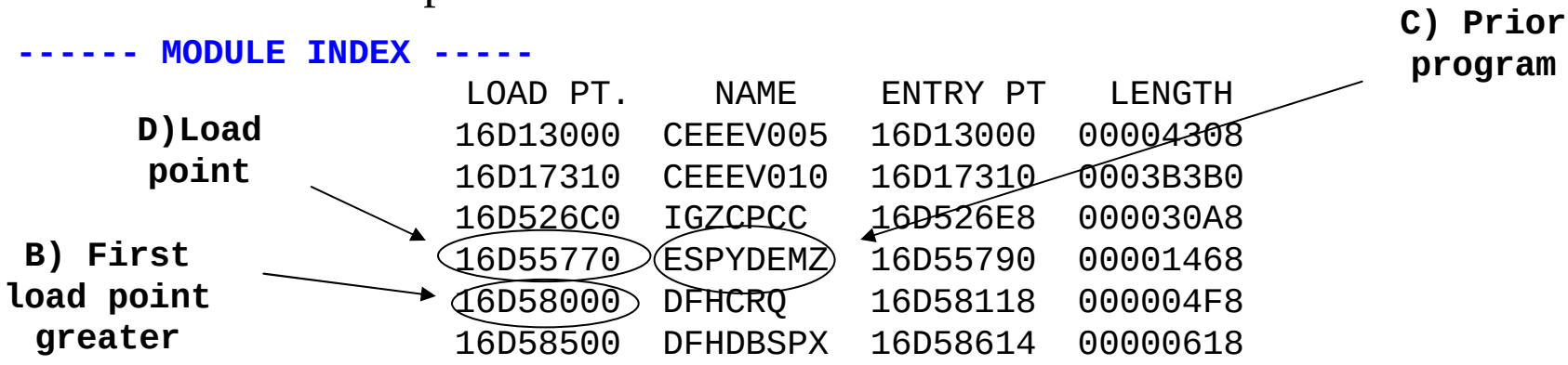
Step 1: Find the load point of the program.

A. Find the MODULE INDEX in the dump

B. Scan the index to find the first LOAD PT. greater than the RET address (**16D55BFC**). Note the high-order bit in the address should be ignored; it is used to track the AMODE of the program

C. The **PRIOR** module is the program the RET is pointing to

D. Get the load point from the index and write it down



# Identifying the EXEC CICS Statement That Failed

Step 2: Identify the CSECT in question

A. Subtract the RET value from the load point to get the offset into the load module

B. In the compile/link output, find the link map

C. Find the first “class offset” that is greater than the value derived in (A).

D. The **PRIOR** entry is the CSECT in question.

E. Write down the offset of this CSECT.

# Identifying the EXEC CICS Statement That Failed

## Step 2: Identify the CSECT in question

```
 *** M O D U L E  M A P ***
--------------
CLASS  B_TEXT           LENGTH =     1468  ATTRIBUTES = CAT,
                        OFFSET =        0 IN SEGMENT 001
--------------
 SECTION    CLASS
  OFFSET   OFFSET  NAME                 TYPE    LENGTH  DDNAME
                0  DFHECI              CSECT        1E  SYSLIB
        8       8     DFHEI1              LABEL
        8       8     DLZEI01             LABEL
        8       8     DLZEI02             LABEL
        8       8     DLZEI03             LABEL
        8       8     DLZEI04             LABEL
               20  ESPYDEMZ             CSECT       7F4  SYSLIN
              818  CEESG005        *    CSECT        18  SYSLIB
              830  CEEBETBL        *    CSECT        28  SYSLIB
```

A:

| | |
|---|---|
| RET value: | 16D55BFC |
| Load pt: | - 16D55770 |
| Offset into load module | 48C |

D) CSECT in question

E) Offset

C) First offset greater

# Identifying the EXEC CICS Statement That Failed

Step 3: Determine the offset of the return into the CSECT.

Subtract the CSECT offset (step 2(E) in the earlier slide) from the load module offset (step 2(A) in the earlier slide. The result is the offset into the CSECT.

```
Load module offset    48C
CSECT offset        - 20
Offset into CSECT     46C
```

# Identifying the EXEC CICS Statement That Failed

Step 4: Identify the COBOL source statement.

A. Locate the Cross-Reference in the COBOL compile output

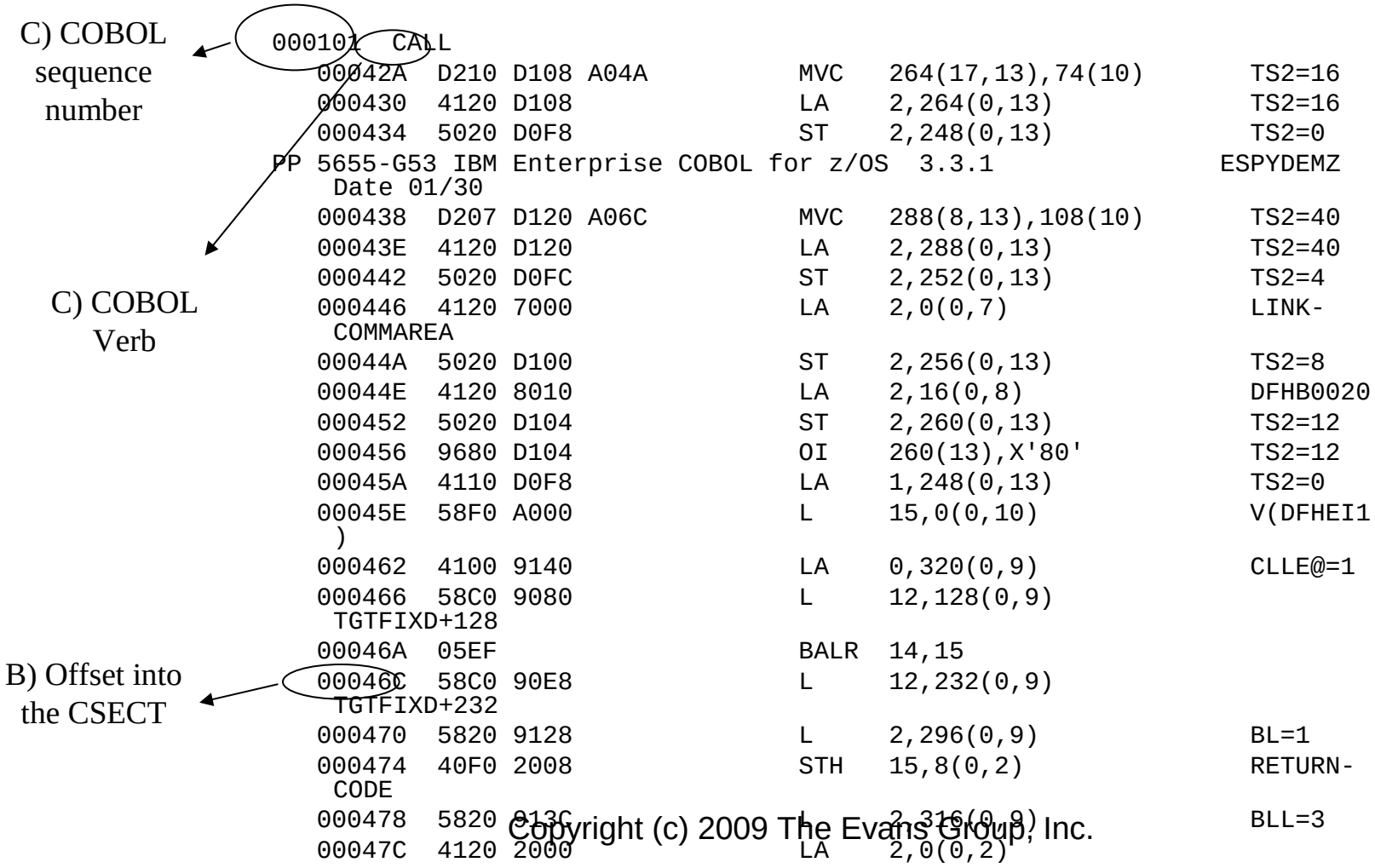
B. Locate the instruction at the offset identified in Step 3 in the previous slide

C. Search backward in the instruction listing to find the COBOL verb and sequence number that generated this offset

D. Locate the COBOL source statement associated with the sequence number

# Identifying the EXEC CICS Statement That Failed

## Step 4: Identify the COBOL source statement.



# Identifying the EXEC CICS Statement That Failed

## Step 4: Identify the COBOL source statement.

```
000093         000042     Call 'DFHEI1' using by content
    x'0c0290002700008c00f0f0f0f4f2 V510
000094                -    '404040' by reference address of Link-commarea by content
    x'0
000095                -    '000' by content x'0000' by content length of Link-
    commarea
000096                     end-call
000097
000098                *exec cics link program('ESPYDEMX') commarea(link-commarea)
000099                *end-exec
000100         000047     Move length of link-commarea to dfhb0020
    V510
000101                     Call 'DFHEI1' using by content
    x'0e02e0000700000100f0f0f0f4f7
000102                -    '404040' by content 'ESPYDEMX' by reference link-commarea
    by
000103                     reference dfhb0020 end-call
000104
000105                *exec cics return end-exec
```

The CICS command that failed.

C) COBOL sequence number

D) COBOL verb

Remember, all EXEC CICS statements are converted into COBOL CALLs by the CICS translator

# RETurn Address to Program Offset Cheat Sheet

1. Enter the RET value…….. 16D55BFC
2. Subtract the program load point…….. - 16D55770
3. Equals the offset into the load module = 48C
4. Subtract the class offset of the CSECT - 20
5. Equals the offset into the program of the EXEC CICS command = 46C
6. In the compile listing xref, find the COBOL sequence number associated with the offset in (5) 000101
7. In the compile listing procedure division, locate the source EXEC CICS command.

# Solving An ASRA Abend

An ASRA abend is the result of an 0Cx system abend in an application program.

Unlike an AEI* abend:

1. The EIB doesn’t point to the cause of the abend
2. The trace table doesn’t show the failing instruction

# Solving An ASRA Abend

For ASRA* abends CICS provides:

1. The current PSW
2. Registers at the time of the abend
3. A message to the CICS joblog describing the abend
4. The offset into the program where the abend occurred

*** - Note that in addition to ASRA abends, the PSW and registers are provided for some other CICS abend codes, such as ASRB and AICA**

# Solving An ASRA Abend

**From the CICS Messages and Codes manual:**

**Explanation:** The task has terminated abnormally because of a program check.

**System Action:** The task is abnormally terminated and CICS issues either message DFHAP0001 or DFHSR0001.

Message DFHSR0622 may also be issued.

**User Response:** Refer to the description of the associated message or messages to determine and correct the cause of the program check.

Note that this information can also be found using the CMAC transaction.

# Solving An ASRA Abend

**From the CICS Job Log:**

DFHAP0001 CICSD225 An abend (code **0C7**/AKEA) has occurred at offset X'00004BB6' in module ESPYDEMO.

**From the CICS Messages and Codes Manual:**

DFHAP0001 applid An abend (code aaa/bbbb) has occurred at offset X’offset’ in module modname.

Explanation: An abnormal end (abend) or program check has occurred in module modname. This implies that there may be an error in CICS code.

# Solving An ASRA Abend

## More From the CICS Messages and Codes Manual:

Alternatively:

• Unexpected data has been input,

• Storage has been overwritten, or

• There has been a program check within a user program.

The code aaa is, if applicable, a 3-digit hexadecimal MVS system completion code (for example, 0C1 or D37). If an MVS code is not applicable, this field is filled with three hyphens. The 4-digit code bbbb, which follows aaa, is a user abend code produced either by CICS or by another product on the user's system.

If X'offset' contains the value X'FFFF', then module modname was in control at the time of the abend, but the program status word (PSW) was not addressing this module.

System Action: An exception entry is made in the trace table. A system dump is taken, unless you have specifically suppressed dumps in the dump table.

Either this is a critical error and CICS is terminated, even if you have specified in the dump table that CICS should not terminate.

Or CICS will continue unless you have specified in the dump table that CICS should terminate.

Message DFHME0116 is normally produced containing the symptom string for this problem.

# Solving An ASRA Abend

**Even More From the CICS Messages and Codes Manual:**

**User Response:** Notify the system programmer.

# Identify The Failing COBOL Statement

We locate the failing COBOL statement in the same way we found the failing EXEC CICS command earlier.

Sometimes, CICS will save us a step by providing the offset of the failing instruction within the program:

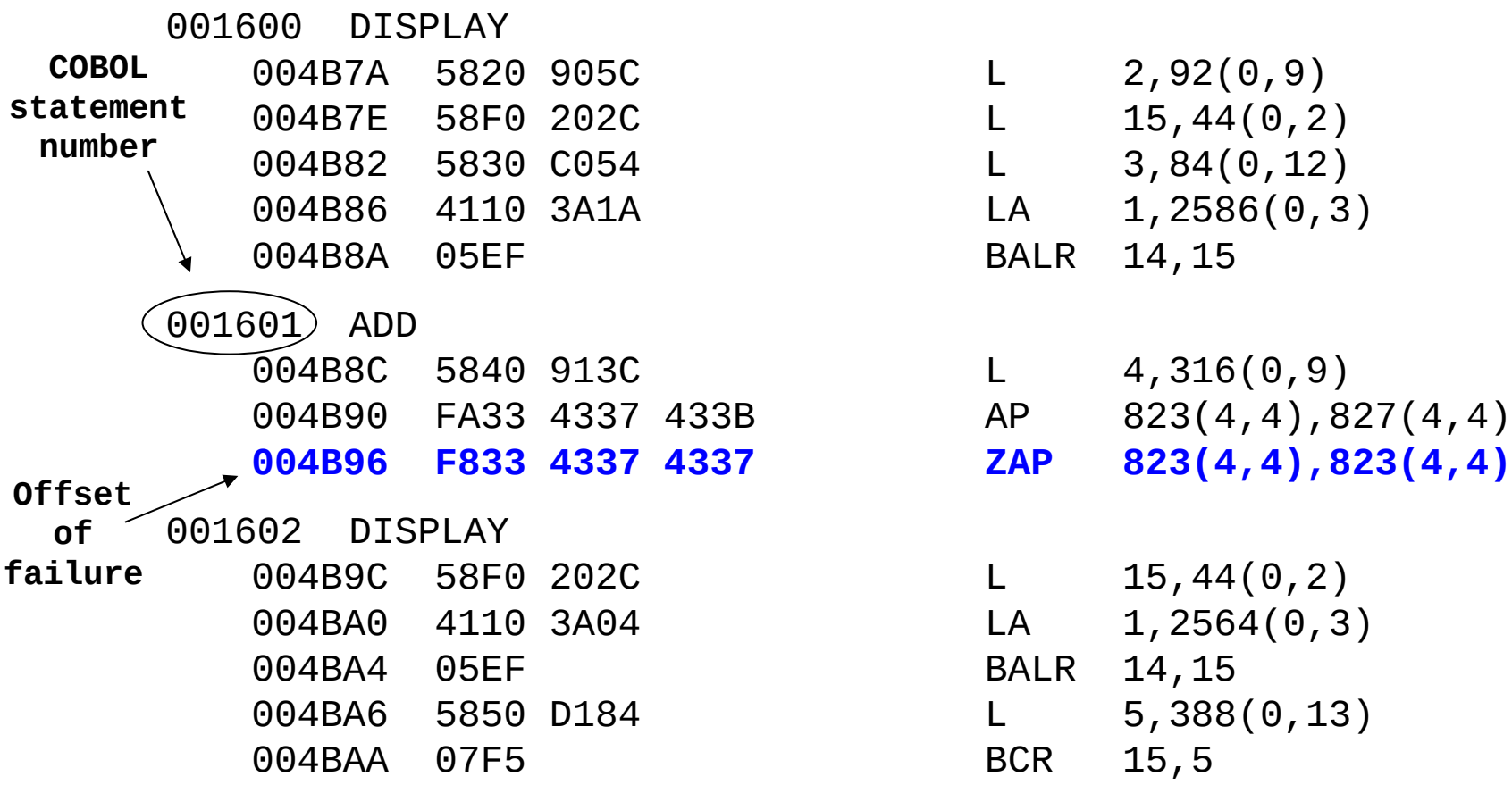
DFHAP0001 CICSD225 An abend (code **0C7**/AKEA) has occurred at offset X'00004BB6' in module ESPYDEMO

# PSW Address to Program Offset Cheat Sheet

**We don't have to worry about (1) and (2) because CICS gave us the offset (3) in the joblog message**

1. Enter the PSW value…… _____
2. Subtract the program load point…….. - _____
3. Equals the offset into the load module = 4BB6
4. Subtract the class offset of the CSECT - 20
5. Equals the offset into the program of the failing statement = 4B96
6. In the compile listing xref, find the COBOL sequence number associated with the offset in (5) _____
7. In the compile listing procedure division, locate the source EXEC CICS command.

# Identifying the COBOL Statement That Failed



# PSW Address to Program Offset Cheat Sheet

1. Enter the PSW value…… _____
2. Subtract the program load point…….. - _____
3. Equals the offset into the load module = 4BB6
4. Subtract the class offset of the CSECT - 20
5. Equals the offset into the program = 4B96
   of the EXEC CICS command
6. In the compile listing xref, find the 1601
   COBOL sequence number associated
   with the offset in (5)
7. In the compile listing procedure division, locate the
   source statement.

# Identifying the COBOL Statement That Failed

```
1595         001252*---------------------------------------------------
1596         001253*    Generate a S0C7 abend where the input data has an invali
1597         001254*    digit.
1598         001255*---------------------------------------------------
1599         001256 PA4-Abend-Asra-BadDigit section.
1600         001257     display MyProgramId ',PA4-Abend-Asra-BadDigit entry.'
1601         001258     add BadDigit1 to GoodField
1602         001259     display MyProgramId ',PA4-Abend-Asra-BadDigit exit.'
1603         001260     .
1604         001261*---------------------------------------------------
```

**Statement number from step (6) in previous slide**

# Identifying the COBOL Statement That Failed

```
 05 ASRA-S0C7-AbendData.
        10 GoodField              pic s9(7) comp-3 value 0.
*       Define a field with bad digits.
        10 BadDigit1C.
           15 BadDigit1C-Part1    pic x(3) value 'Bad'.
           15 BadDigit1C-Part2    pic x(1) value x'0F'.
        10 BadDigit1 redefines BadDigit1C pic s9(7) comp-3.
```

# Identify The Failing COBOL Statement

What to do when CICS doesn't supply the offset

```
DFHAP0001 CICSD225 An abend (code 0C7/AKEA) has
occurred at offset X'FFFFFFFF' in module ESPYDEMO
```

CICS is not aware of COBOL dynamic calls!

Abends in dynamically called programs show an offset of FFFFFFFF

Because no valid offset is provided. We must:

1) Manually identify the program which abended
2) Manually calculate the offset into that program

# Identify The Failing COBOL Statement

Step 1: Find the load point of the program.

Step 2: Identify the CSECT in question

Step 3: Determine the offset of the return into the CSECT.

Step 4: Identify the COBOL source statement.

# Identifying the COBOL Statement That Failed

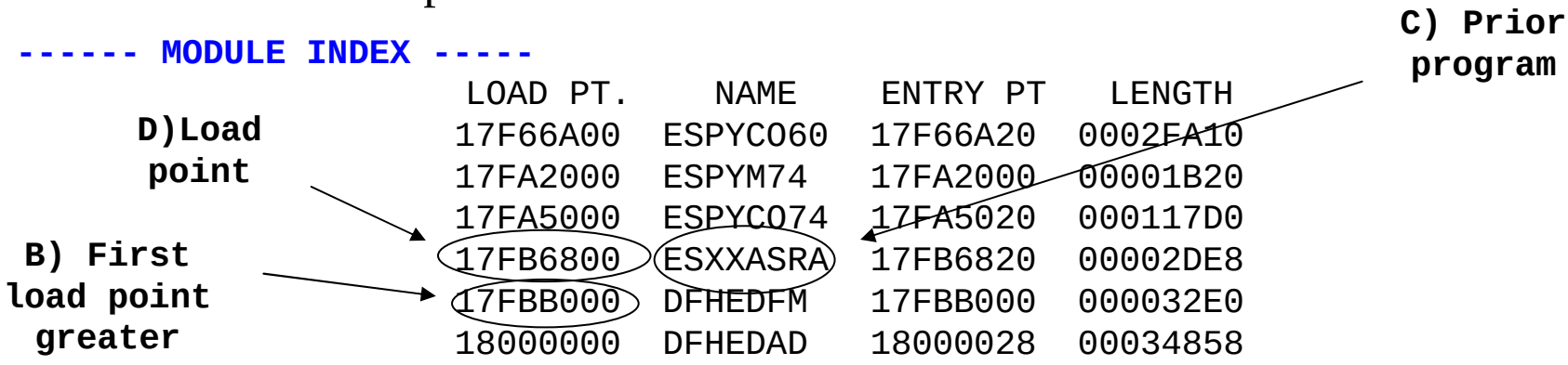
Step 1: Find the load point of the program.

A. Find the MODULE INDEX in the dump

B. Scan the index to find the first LOAD PT. greater than the PSW value in the dump (97FB7780). Note the high-order bit in the address should be ignored; it is used to track the AMODE of the program

C. The PRIOR module is the program the PSW is pointing to

D. Get the load point from the index and write it down



# PSW Address to Program Offset Cheat Sheet

1. Enter the PSW value…… _97FB7780
2. Subtract the program load point…….. - _17FB6800
3. Equals the offset into the load module = F80
4. Subtract the class offset of the CSECT - 20
5. Equals the offset into the program of the abending instruction = F60
6. In the compile listing xref, find the COBOL sequence number associated with the offset in (5)
7. In the compile listing procedure division, locate the source statement.

## Determine the Value of the Failing COBOL Statement's Arguments

We know that the failing statement has two arguments:

```
add BadDigit1 to GoodField
```

Now, we have to identify the field with the invalid data.

To do so, we have to determine the type of data mapped by the field, the length of the data mapped by the field, and the current value of the data at the time of the abend.

## Determine the Value of the Failing COBOL Statement's Arguments

COBOL has two types of data storage:

1. Working Storage
   - Comes in one contagious block of storage
   - Allocated and (optionally) initialized before the program is given control
   - Addressed by BLW cells (Base Locater for Working storage)
2. Linkage section
   - Storage passed via USING parm on CALL statement
   - Storage acquired by program (ex., GETMAIN, READ SET.)
   - Addressed by BLL cells (Base Locater for Linkage
   - Not in contiguous storage – each 01 level starts a new storage block

# Determine the Value of Working Storage Fields

Step 1: Find the program's TGT.

Step 2: Determine the BLW cell number, the offset of the field, the field's data type and length

Step 4: Find the address of the start of the BLW cell from Step (2).

Step 5: Find the field value in the dump.

# Determine the Value of Working Storage Fields

Step 1: Find the program's TGT

There are several methods for locating the COBOL TGT in a CICS dump:

A. Register 9 from the 'REGISTERS AT TIME OF INTERRUPT' contains the address of the TGT
B. Add +5C to the value in Register 13 from the 'REGISTERS AT TIME OF INTERRUPT', find the resulting address in the dump. At that location will be the address of the TGT
C. In the trace table, find an "EIP ENTRY" trace entry that was issued by your program. Add +5C to the value found in "Field A". Find the resulting address in the dump. At that location will be the address of the TGT

# Determine the Value of Working Storage Fields

Step 1: Find the program's TGT (using register 9)

**Hint: to quickly find an address in the dump, search on only the first three bytes, in columns 120 133: f '1830EB' 120 133**

```
REGISTERS AT TIME OF INTERRUPT
REGS 0-7     174F5E48  174F5036  000607FC  174F461C     181000C0  1810A0C0  001000D0  174F461C
REGS 8-15    174F261C  17129B88  1712DB88  174F6216     174F25AC  1712FF30  974F703C  97B7BC40

00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17129B80
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17129BA0
00000000 00000000 00000000 00000000  F3E3C7E3 00000000 06000000 68030260   *................3TGT..........-*   17129BC0
171297D0 000607FC 1712DEF0 00000000  01014588 00000000 00000000 18100030   *..p........0.......h...........*   17129BE0
00000000 00000000 17127BC8 00004368  00000000 00000000 00000000 00000001   *...........H...................*   17129C00
```

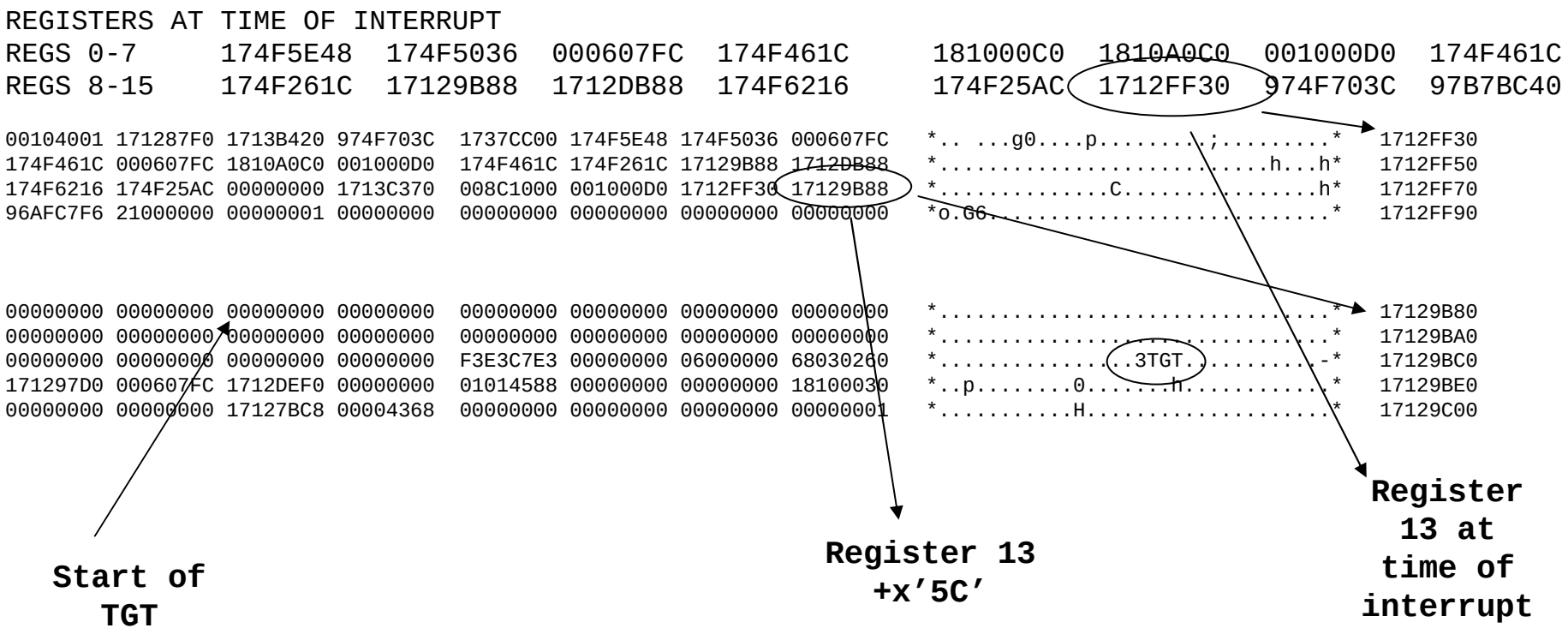
**Start of TGT**

**Register 9 at time of interrupt**

**Note literal "3TGT" at +x'48' into TGT**

# Determine the Value of Working Storage Fields

Step 1: Find the program's TGT (using register 13)



# Determine the Value of Working Storage Fields

Step 2: Determine the BLW cell number, the offset of the field, the field's data type and length

In the compile listing for the abending program:

A. Locate the "Data Division Map"

B. Find the field name(s) in question.

C. The BLW cell number, offset, data type, and length are displayed

```
Data Division Map
Source   Hierarchy and                              Base       Hex-Displacement  Asmblr Data
LineID   Data Name                                  Locator    Blk   Structure   Definition      Data Type
   465      3 BADDIGIT1 . . . . . . . . . . . . . .BLW=00000  33B   0 000 183   DS 4P           Packed-Dec
```

# Determine the Value of Working Storage Fields

Step 3: Find the address of the start of the BLW cell from Step (2)

A. In the compile listing for the abending program, find the "TGT Memory Map" and identify the offset within the TGT where the addresses are stored

# Determine the Value of Working Storage Fields

```
*** TGT MEMORY MAP ***
TGTLOC

000000  RESERVED - 72 BYTES
000048  TGT IDENTIFIER
00004C  RESERVED - 4 BYTES
000050  TGT LEVEL INDICATOR
000051  RESERVED - 3 BYTES
000054  32 BIT SWITCH
000058  POINTER TO RUNCOM
00005C  POINTER TO COBVEC
000060  POINTER TO PROGRAM DYNAMIC BLOCK TABLE
000064  NUMBER OF FCB'S
000068  WORKING-STORAGE LENGTH
00006C  RESERVED - 4 BYTES
000070  ADDRESS OF IGZESMG WORK AREA
000074  ADDRESS OF 1ST GETMAIN BLOCK (SPACE
      MGR)
000078  RESERVED - 2 BYTES
000110  POINTER TO FIRST FCB CELL
000114  WORKING-STORAGE ADDRESS
000118  POINTER TO FIRST SECONDARY FCB CELL
00011C  POINTER TO STATIC CLASS INFO BLOCK 1
000120  POINTER TO STATIC CLASS INFO BLOCK 2

*** VARIABLE PORTION OF TGT ***

000124  TGT OVERFLOW AREA ADCONS
000134  BASE LOCATORS FOR SPECIAL REGISTERS
00013C  BASE LOCATORS FOR WORKING-STORAGE
004190  BASE LOCATORS FOR LINKAGE-SECTION
004210  CLLE ADDR. CELLS FOR CALL LIT. SUB-PGMS.
```

Offset within the TGT of where the BLW cell list starts

Note that BLL cells work the same way. We'll come back to them later!

# Determine the Value of Working Storage Fields

Step 3: Find the address of the start of the BLW cell from Step (2)

A. In the compile listing for the abending program, find the “TGT Memory Map” and identify the offset within the TGT where the addresses are stored

B. In the dump, find the address that is stored at the offset found in (3) (A)

# Determine the Value of Working Storage Fields

TGT Address 17129B88

BLW cell offset +13C

Start of BLW cell addresses 17129CC4

```
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17129B80
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17129BA0
00000000 00000000 00000000 00000000  F3E3C7E3 00000000 06000000 68030260   *................3TGT...........-*   17129BC0
171297D0 000607FC 1712DEF0 00000000  01014588 00000000 00000000 18100030   *..p........0.......h............*   17129BE0
00000000 00000000 17127BC8 00004368  00000000 00000000 00000000 00000001   *...........H....................*   17129C00
E2E8E2D6 E4E34040 C9C7E9E2 D9E3C3C4  00000000 00000000 00000000 00000000   *SYSOUT  IGZSRTCD................*   17129C20
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17129C40
00000000 00000000 00000000 00000000  174F25AC 00000000 1712DEDC 17129AA8   *.............................y*   17129C60
174F43E6 00000000 174F24B0 174F2724  1712DE98 174F2604 00000000 181000C0   *...W..............q............*   17129C80
00000000 00000000 00000000 1712AB88  1712BB88 1712CB88 1712DB88 181060C0   *................h...h...h...h..-.*   17129CA0
18100040 181000C0 181010C0 181020C0  181030C0 181040C0 181050C0 181060C0   *... ................... ........-.*   17129CC0
181070C0 181080C0 181090C0 1810A0C0  1810B0C0 1810C0C0 1810D0C0 1810E0C0   *................................*   17129CE0
1810F0C0 181100C0 181110C0 181120C0  181130C0 181140C0 181150C0 181160C0   *..0................... ........-.*   17129D00
```

BLW0 BLW1 BLW2 BLW3

# Determine the Value of Working Storage Fields

Step 3: Find the address of the start of the BLW cell from Step (2)

A. In the compile listing for the abending program, find the “TGT Memory Map” and identify the offset within the TGT where the addresses are stored
B. In the dump, find the address that is stored at the offset found in (3) (A)

C. Add the offset you found in Step (2) (C) to the BLW address from step (3) (B)

# Determine the Value of Working Storage Fields

| Hierarchy and Data Name | Base Locator | Hex-Displacement Blk | Structure | Asmblr Data Definition | Data Type |
|---|---|---|---|---|---|
| 3 BADDIGIT1 . . | BLW=00000 | 33B | 0 000 183 | DS 4P | **Packed-Dec** |

BLW Address 181000C0

Offset of field +33B

Start of field 181003FB

```
C1C2C5D5 C4D5D940 C3C9C3E2 404040C5  E2D7E8C3 E3D34000 00000000 00000000   *ABENDNR CICS   ESPYCTL .........*   181003A0
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   181003C0
00000000 00000000 00000000 00000000  00000000 00000000 00000CC2 81840F01   *............................Bad..*   181003E0
2345EF01 234562C4 94F10FC5 E2D7E8C4  C5D4F1C4 94F20FC4 94F30F11 22334455   *.......Dm1.ESPYDEM1Dm2.Dm3.....*   18100400
66778899 11223344 55667788 99AABBCC  DDEE4040 D3C2E6A2 60D38995 9260F5E6   *..hr.......hr.....  LBWs-Link-5W*   18100420
```

# Locate field value in dump Cheat Sheet – page 1

In the COBOL compile listing Data Division Map, find:

1. The field name _____**Baddigit1**__
2. The field's Base Locator number…….. ____**0**_____
3. The type of locator …… BLL/BLW
4. The field's offset within the cell …… ____**33B**__
5. The field's length …….. _____**4**___
   and type …….. __**packed**__

In the COBOL compile listing TGT Memory map, find:

6. Offset of the start of the Base Locators
   for the type in (3) …….. _____**13C**__

# Locate field value in dump Cheat Sheet – page 2

7. Enter the TGT address ……..__1**7129B88**
8. Plus the BLW cell offset (6)…..…….. + _____**13C**
9. Equals the address of the BLW cells __**17129CC4**
10. Multiply (2) X 4 and enter the HEX value +_____0
11. (9) plus (10) provides the address of the address of the start of the BLW cell __**17129CC4**
12. Enter the address found at (11) __**181000C0**
13. Plus the field offset from (4) ……. +_____**33B**
14. Equals the address of the field value __**181003F**B

# Determine the Value of Linkage Section Fields

Two main differences between locating Linkage fields and Working Storage fields:

1) Linkage Section storage is not contiguous
2) Linkage Section storage uses BLL cells, not BLW cells

Step 1: Find the program's TGT.

Step 2: Determine the BLL cell number, the offset of the field, the field's data type and length

Step 4: Find the address of the start of the BLL cell from Step (2).

Step 5: Find the field value in the dump.

# Determine the Value of Linkage Section Fields

Step 2: Determine the BLL cell number, the offset of the field, the field's data type and length

In the compile listing for the abending program:

A. Locate the "Data Division Map"

B. Find the field name(s) in question.

C. The BLL cell number, offset, data type, and length are displayed

```
Data Division Map
Source   Hierarchy and                              Base       Hex-Displacement  Asmblr Data
LineID   Data Name                                  Locator    Blk   Structure   Definition      Data Type
  1327     2  COMM-TRANID .  . . . . . . . . . . . BLL=00002  000   0 000 000   DS 4C           Display
```

# Determine the Value of Linkage Section Fields

Step 3: Find the address of the start of the BLL cell from Step (2)

A. In the compile listing for the abending program, find the "TGT Memory Map" and identify the offset within the TGT where the addresses are stored

# Determine the Value of Linkage Section Fields

```
*** TGT MEMORY MAP ***
TGTLOC

000000  RESERVED - 72 BYTES
000048  TGT IDENTIFIER
00004C  RESERVED - 4 BYTES
000050  TGT LEVEL INDICATOR
000051  RESERVED - 3 BYTES
000054  32 BIT SWITCH
000058  POINTER TO RUNCOM
00005C  POINTER TO COBVEC
000060  POINTER TO PROGRAM DYNAMIC BLOCK TABLE
000064  NUMBER OF FCB'S
000068  WORKING-STORAGE LENGTH
00006C  RESERVED - 4 BYTES
000070  ADDRESS OF IGZESMG WORK AREA
000074  ADDRESS OF 1ST GETMAIN BLOCK (SPACE
      MGR)
000078  RESERVED - 2 BYTES
000110  POINTER TO FIRST FCB CELL
000114  WORKING-STORAGE ADDRESS
000118  POINTER TO FIRST SECONDARY FCB CELL
00011C  POINTER TO STATIC CLASS INFO BLOCK 1
000120  POINTER TO STATIC CLASS INFO BLOCK 2

*** VARIABLE PORTION OF TGT ***

000124  TGT OVERFLOW AREA ADCONS
000134  BASE LOCATORS FOR SPECIAL REGISTERS
00013C  BASE LOCATORS FOR WORKING-STORAGE
004190  BASE LOCATORS FOR LINKAGE-SECTION
004210  CLLE ADDR. CELLS FOR CALL LIT. SUB-PGMS.
```

Offset within the TGT of where the BLL cell list starts

# Determine the Value of Linkage Section Fields

Step 3: Find the address of the start of the BLL cell from Step (2)

A. In the compile listing for the abending program, find the “TGT Memory Map” and identify the offset within the TGT where the addresses are stored

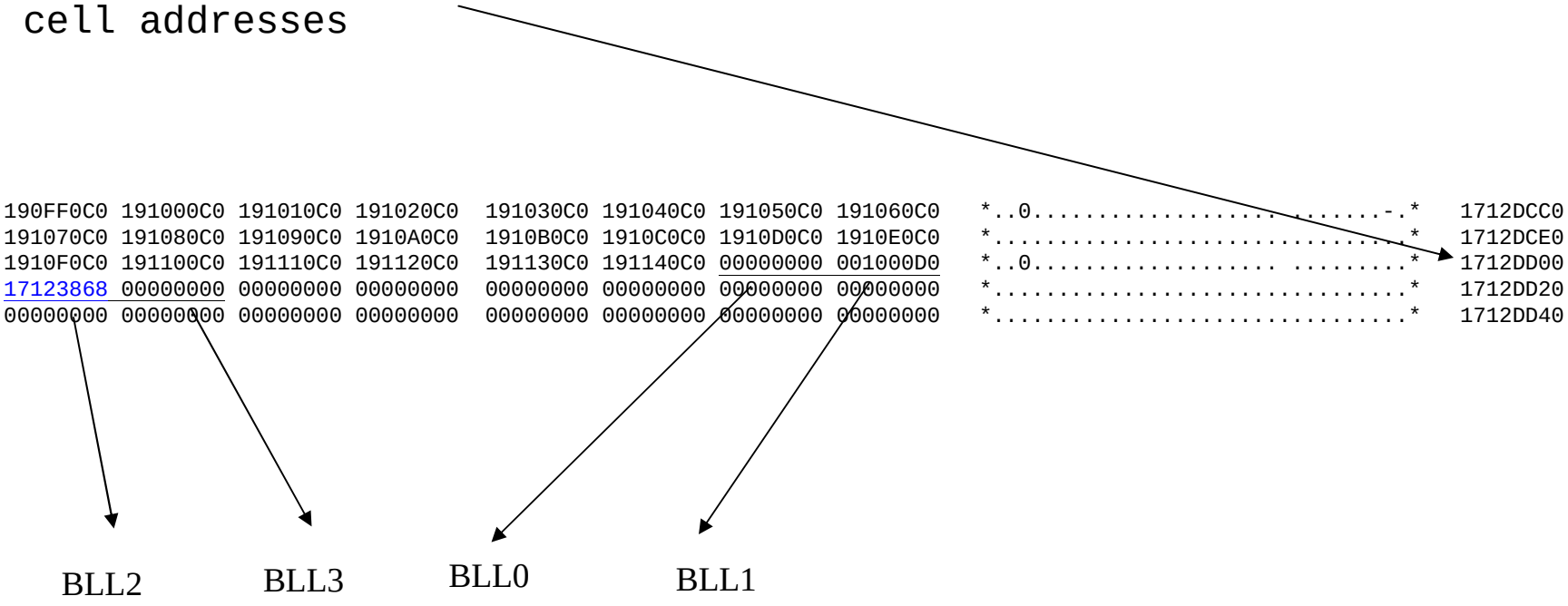
B. In the dump, find the address that is stored at the offset found in (3) (A)

# Determine the Value of Linkage Section Fields

TGT Address  17129B88

BLL cell offset + 4190

Start of BLL cell addresses  1712DD18



# Determine the Value of Linkage Section Fields

Step 3: Find the address of the start of the BLW cell from Step (2)

A. In the compile listing for the abending program, find the “TGT Memory Map” and identify the offset within the TGT where the addresses are stored
B. In the dump, find the address that is stored at the offset found in (3) (A)

C. Add the offset you found in Step 2 (C) to the BLL address from step (3) (B)

# Determine the Value of Linkage Section Fields

| Hierarchy and Data | Name | Base Locator | Hex-Displacement Blk | Structure | Asmblr Data Definition | Data Type |
|---|---|---|---|---|---|---|
| 2 | COMM-TRANID | BLL=00002 | 000 | 0 000 000 | DS 4C | Display |

BLL Address 17123868

Offset of field + 0

Start of field 17123868

```
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17123800
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000   *................................*   17123820
00000000 00000000 00000000 00000000  00000000 00000000 E4F0F0F0 F0F0F4F9   *........................U0000049*   17123840
E4F0F0F0 F0F3F0F2 C5E2C4D4 00000000  00000000 00000000 E4F0F0F0 F0F3F0F2   *U0000040ESDM..........U0000049*   17123860
```

# Locate field value in dump Cheat Sheet – page 1

In the COBOL compile listing Data Division Map, find:

1. The field name _____COMM-TRANID_____
2. The field's Base Locator number…….. ____2_____
3. The type of locator …… (BLL)/BLW
4. The field's offset within the cell …… ____0_____
5. The field's length …….. ____4_____
   and type …….. __Display__

In the COBOL compile listing TGT Memory map, find:

6. Offset of the start of the Base Locators
   for the type in (3) …….. ___4190___

# Locate field value in dump Cheat Sheet – page 2

7. Enter the TGT address …….__17129B88
8. Plus the BL cell offset (6)…..…….. + _____**4910**
9. Equals the address of the BLW cells __1712DD18
10. Multiply (2) X 4 and enter the HEX value +_____**0**
11. (9) plus (10) provides the address of the address of the start of the BLW cell __1712DD18
12. Enter the address found at (11) __17123868
13. Plus the field offset from (4) ……. +_____**0**
14. Equals the address of the field value __1712DD18

# Working With EXEC CICS Links

When a program issues an EXEC CICS LINK, CICS preserves the current program environment and builds an extension for the new program. When a LINKed-to program issues an EXEC CICS LINK, another extension is created, and so on.

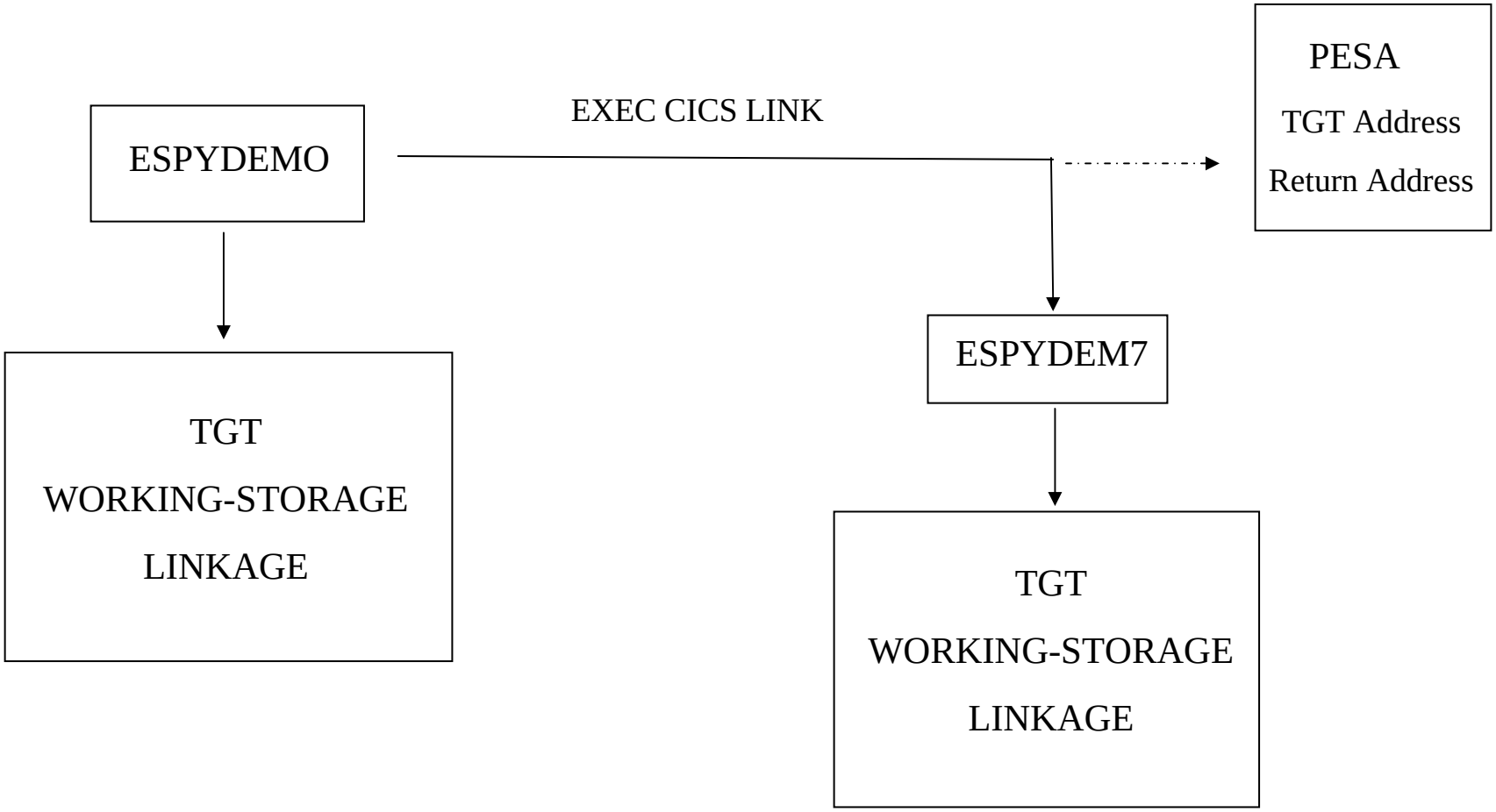
The PROGRAM INFORMATION portion of the dump provides details for each level.

If the “number of levels” is greater than 1, the abending program was linked to.

# Working With EXEC CICS Links

```
PROGRAM INFORMATION FOR THE CURRENT TRANSACTION
   Number of Levels 00000002
INFORMATION FOR PROGRAM AT LEVEL 00000002 of 00000002
   Program Name     ESPYDEM7     Invoking Program ESPYDEMO
   Load Point       180AF000     Program Length   00001900
   Entry Point      980AF000     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced COBOL II
   Commarea Address 181005B4     Commarea Length  00000001
   Execution Key    USER         Data Location    BELOW
   Concurrency      QUASIRENT    Api              CICSAPI
   Runtime          LE370
   Environment      User application
INFORMATION FOR PROGRAM AT LEVEL 00000001 of 00000002
   Program Name     ESPYDEMO     Invoking Program CICS
   Load Point       174F2490     Program Length   0000BF08
   Entry Point      974F24B0     Addressing Mode  AMODE 31
   Language Defined COBOL        Language Deduced COBOL II
   Commarea Address 17123868     Commarea Length  00000004
   Execution Key    USER         Data Location    BELOW
   Concurrency      QUASIRENT    Api              CICSAPI
   Runtime          LE370
   Environment      User application
```

# Working With EXEC CICS Links



# Working With EXEC CICS Links

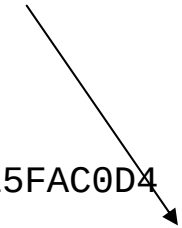
Locate the TGT and the EXEC CICS LINK statement from the linking program.

1. Locate the PESA for the linking program
2. Locate the TGT using data from the PESA
3. Locate the return value using data from the PESA
4. Locate this information for the program that linked to the linking program (if any)
5. Repeat as required

# Working With EXEC CICS Links

Locate the TGT and the EXEC CICS LINK statement from the linking program.

1. Locate the PESA for the linking program. The PESA address is found at +38 in the TCA System area



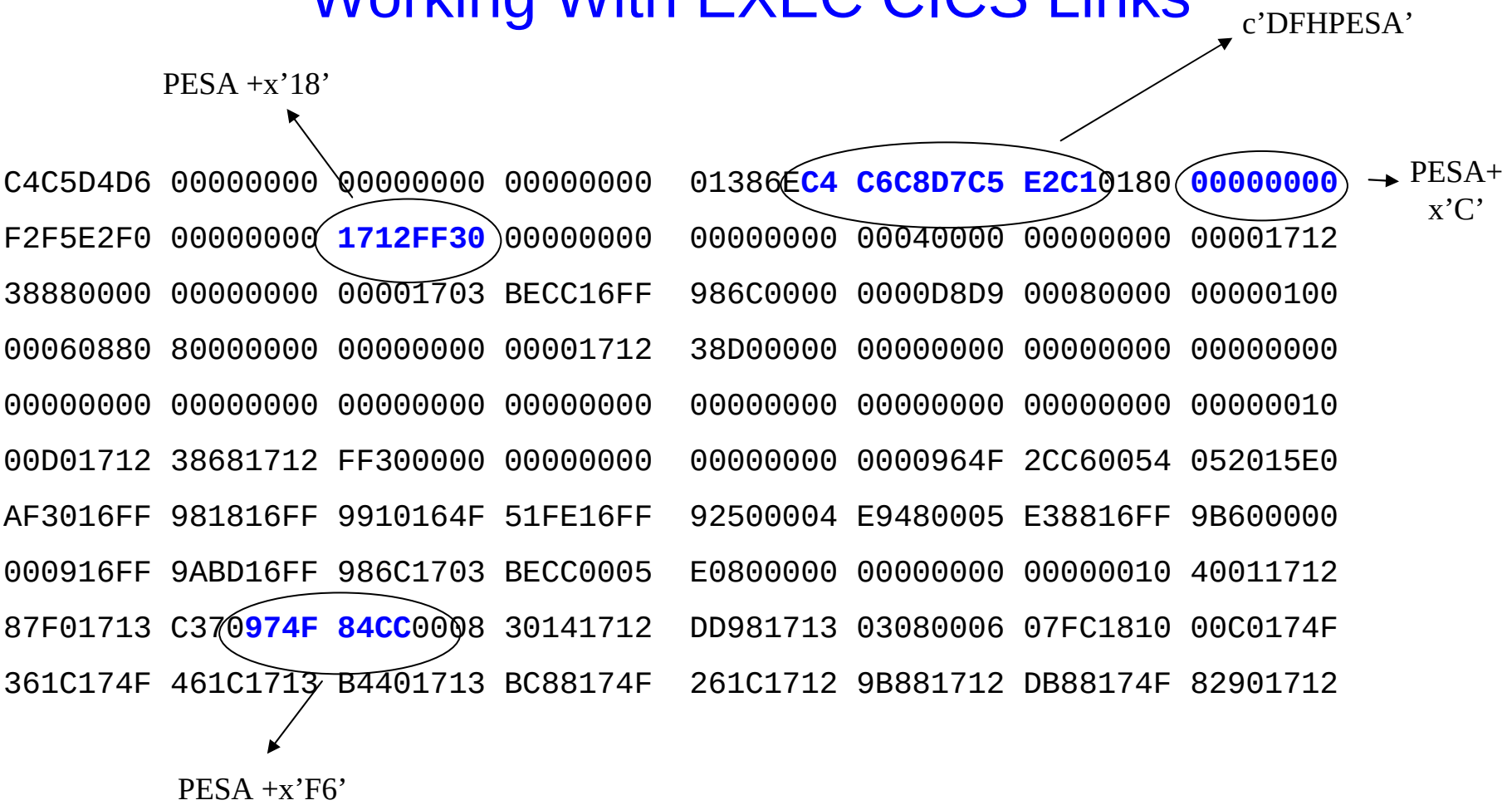
```
TASK CONTROL AREA (SYSTEM AREA)
0000   00000000 00000000 00000000 00000000  0000401C 15FAC0D4 00000087 00000000
0020   00000000 00000000 00000000 00000000  00000000 00000000 16FFAE08 00000000
0040   17120060 00000000 00000000 00000000  00000000 00000000 00000000 00000000
0060   00000000 00000000 00000000 00000000  00000000 00000000 00000000 C1E2D9C1
```

# Working With EXEC CICS Links

Important PESA Addresses:

| Offset | Value |
|---|---|
| x’02’ | Eyecatch >DFHPESA |
| x’0C’ | Address of previous PESA (if any) |
| x’18’ | R13 value from LINKING program |
| x’F6’ | R14 value from LINKING program |

# Working With EXEC CICS Links



# Working With EXEC CICS Links

Locate the TGT and the EXEC CICS LINK statement from the linking program.

2) Locate the TGT using data from the PESA

   Add +5C to the value in Register 13 (found at +x'18' in the PESA), find the resulting address in the dump. At that location will be the address of the TGT

3) Locate the return value using data from the PESA.

   The address at +F6' into the PESA is the return value into the LINKing program. Use the RETurn Address to Program Offset Cheat Sheet to identify the EXEC CICS LINK command.

# Identify TGT and return value of LINKing Program
# Cheat Sheet – Page 1

1. Enter the address found at +x'38' in the
   TCA System area……._16FFAE08_
2. Using the address from (1), find:
   A. At +x'02', Eyecatch ">DFHPESA"
   B. At x+0C, enter address of previous
      PESA (or zeros if none)…….. _00000000_
   C. At x'18', enter the value of R13
      at the time of the LINK …… _1712FF30_
   D. At +x'F6', enter the value of R14
      at the time of the LINK ……. _974F84CC _

# Identify TGT and return value of LINKing Program Cheat Sheet – Page 2

3. Use the address from 2(A) to fill out a Cheat Sheet for the previous PESA.
4. Use the address from 2(C) to use with the Locate Field Value Cheat Sheet.
5. Use the address from 2(D) to use with the Return Address to Program Offset Cheat Sheet.

# Conclusion

- Dumps and traces are valuable tools
- Identify the failing line of code
- Determine the values of fields at time of failure
- Help determine the actual cause of the failure
- Copy the “Cheat sheets” for later use

# RETurn Address to Program Offset Cheat Sheet

1. Enter the RET value……._____
2. Subtract the program load point…….. - _____
3. Subtract the class offset of the CSECT - _____
4. Equals the offset into the program = _____
   of the EXEC CICS command
5. In the compile listing xref, find the _____
   COBOL sequence number associated
   with the offset in (4)
6. In the compile listing procedure division, locate the source EXEC CICS command.

# Identify TGT and return value of LINKing Program
# Cheat Sheet – Page 1

1. Enter the address found at +x'38' in the TCA System area……._____
2. Using the address from (1), find:
   A. At +x'02', Eyecatch ">DFHPESA"
   B. At x+0C, enter address of previous PESA (or zeros if none)…….. _____
   C. At x'18', enter the value of R13 at the time of the LINK …… _____
   D. At +x'F6', enter the value of R14 at the time of the LINK ……. _____

# Identify TGT and return value of LINKing Program Cheat Sheet – Page 2

3. Use the address from 2(A) to fill out a Cheat Sheet for the previous PESA.
4. Use the address from 2(C) to use with the Locate Field Value Cheat Sheet.
5. Use the address from 2(D) to use with the Return Address to Program Offset Cheat Sheet.

# PSW Address to Program Offset Cheat Sheet

1. Enter the PSW value……._____
2. Subtract the program load point…….. - _____
3. Subtract the class offset of the CSECT - _____
4. Equals the offset into the program = _____
   of the next instruction
5. In the compile listing xref, find the _____
   COBOL sequence number associated
   with the offset in (4)
6. In the compile listing procedure division, locate the
   source statement.

# Locate field value in dump Cheat Sheet – page 1

In the COBOL compile listing Data Division Map, find:

1. The field name _____
2. The field's Base Locator number…….. _____
3. The type of locator …… BLL/BLW
4. The field's offset within the cell …… _____
5. The field's length …….. _____
   and type …….. _____

In the COBOL compile listing TGT Memory map, find:

6. Offset of the start of the Base Locators
   for the type in (3) …….. _____

# Locate field value in dump Cheat Sheet – page 2

7. Enter the TGT address ……._____
8. Plus the BLW cell offset (6)…..... + _____
9. Equals the address of the BLW cells _____
10. Multiply (2) X 4 and enter the HEX value +_____
11. (9) plus (10) provides the address of the address of the start of the BLW cell _____
12. Enter the address found at (11) _____
13. Plus the field offset from (4) ……. +_____
14. Equals the address of the field value _____