Multi-Modular Dump Solving

Multi-Modular Dumps

Introduction

In principle multi-modular dumps are no more difficult to solve than single module dumps. The main difference when solving a multi-modular dump is that we must first find which module the abend occurred in. In order to do this we need to understand a little more about the Linkage Editor report.

The Linkage Editor Report

This report gives a list of the various modules which have been linked together to form the program and their relative displacements within that program.

It is important to note that the 'origin' of the program is not always the same as the entry point. i.e. module 00 will not always have a displacement of 00 within the program.

```
MVS/DFP VERSION 3 RELEASE 1 LINKAGE EDITOR     15:11:44  TUE  DEC 08, 1992
 JOB DEVADTR    STEP LINK       PROCEDURE STEPB
 INVOCATION PARAMETERS - XREF,LIST,LET
 ACTUAL SIZE=(317440,86016)
 OUTPUT DATA SET SYS92343.T151143.RA000.DEVADTR.EXECLIB IS ON VOLUME USR008
IEW0000     INCLUDE TESTLIB(TR9MM2,TR9MM1)
IEW0000     ENTRY TR9MM1

                                                CROSS REFERENCE TABLE

  CONTROL SECTION                          ENTRY
    NAME    ORIGIN  LENGTH                   NAME   LOCATION     NAME   LOCATION
 NAME
  TR9MM2        00     4B2
  TR9MM1       4B8     EDA
  BPSAIOP *   1398      34
                                           BPSOPEN     1398
               ...

 ENTRY ADDRESS      4B8
 TOTAL LENGTH      1CD0
 ** TR9ADC   DID NOT PREVIOUSLY EXIST BUT WAS ADDED AND HAS AMODE 24
 ** LOAD MODULE HAS RMODE 24
 ** AUTHORIZATION CODE IS         0.
```

In the above example, the Entry Point (the start of the program) is TR9MM1 which begins at displacement 4B8 within the program.

This is normally because one or more of the modules within the program has been recompiled and linked to the original program at a later stage, i.e. after the program has been made live. The most recently linked modules will be placed at the beginning of the program.

Finding the statement in error

The procedure will be :

1) Find the relative interrupt address.

2) Find the module in which the abend occurred.

3) Find the corresponding COBOL statement.

Finding the Relative Interrupt Address

There are many ways of finding the Relative Interrupt Address, the Address of the abending instruction within your program. In this course we will show you the two main alternatives.

a) The easiest way to use the OFFSET and ILC from the JOB LOG as for single modular dumps.

```
15.11.47 JOB06505  IEA995I SYMPTOM DUMP OUTPUT
                   SYSTEM COMPLETION CODE=0C7  REASON CODE=00000007
                    TIME=15.11.46  SEQ=44174  CPU=0000  ASID=0038
                    PSW AT TIME OF ERROR  078D1000   00006F86  ILC 6  INTC 07
                                ACTIVE  LOAD  MODULE=TR9ADC      ADDRESS=00006330
OFFSET=00000C56
                      DATA AT PSW  00006F80 - FC41D1F3  D1FEF833  9498D1F4
                      GPR  0-3  00025FF8  00028E70  30026610  00026374
                      GPR  4-7  00026190  700070A8  008CAFF8  FD000000
                      GPR  8-11 00028E70  00026310  00006894  00006B08
                      GPR 12-15 00006868  00025E88  7000BD14  8000BD24
                    END OF SYMPTOM DUMP
```

RELATIVE INTERRUPT ADDRESS = OFFSET - ILC

C50 = C56 - 6

b) The traditional way of finding the relative interrupt address is a little more complex but may be useful if you want to check your result (or in some rare cases where OFFSET is not given!).

In order to find the Relative Interrupt Address we must find the **Absolute address of the Origin** of the program, the address of the start of the program:

The easiest way to look at the ADDRESS is in the SYMPTOM DUMP OUTPUT on the Job Log (file 1).

```
15.11.47 JOB06505  IEA995I SYMPTOM DUMP OUTPUT
                   SYSTEM COMPLETION CODE=0C7  REASON CODE=00000007
                    TIME=15.11.46  SEQ=44174  CPU=0000  ASID=0038
                    PSW AT TIME OF ERROR  078D1000   00006F86  ILC 6  INTC 07
                                ACTIVE  LOAD  MODULE=TR9ADC      ADDRESS=00006330
OFFSET=00000C56
                      DATA AT PSW  00006F80 - FC41D1F3  D1FEF833  9498D1F4
                      GPR  0-3  00025FF8  00028E70  30026610  00026374
                      GPR  4-7  00026190  700070A8  008CAFF8  FD000000
                      GPR  8-11 00028E70  00026310  00006894  00006B08
                      GPR 12-15 00006868  00025E88  7000BD14  8000BD24
                    END OF SYMPTOM DUMP
```

In this example the ADDRESS is 6330 ; this is the 'absolute origin' of the program.

In order to find the Relative Interrupt Address we now need to look up the Absolute Interrupt Address in the DATA AT PSW on the Symptom Dump.

The Absolute Address of the Origin is then subtracted from the Absolute Interrupt Address to give the Relative Interrupt Address with the program.

RELATIVE		ABSOLUTE		ORIGIN
INERRUPT	=	INTERRUPT	-	ADDRESS
ADDRESS		ADDRESS		
C50	=	6F80	-	6330

Finding the module in which the abend occurred

To do this we need to look at the Linkage Editor Report.

```
                    *** M O D U L E   M A P ***
                                                ---------------
          CLASS  B_TEXT            LENGTH =     3828  ATTRIBUTES = CAT,   LOAD, RMODE= 24 ALIGN = DBLWORD
---------------

SECTION CLASS                                  ------- SOURCE --------           OFFSET  OFFSET  NAME            TYPE
   LENGTH   DDNAME   SEQ  MEMBER
              0  TR1000LG           CSECT       402  TESTLIB   01  TR1000LG

          408  TR1001LG           CSECT      11E6  TESTLIB   01  TR1001LG

        15F0  TR1002LG           CSECT       604  TESTLIB   01  TR1002LG

        1BF8  TR1003LG           CSECT       6A2  TESTLIB   01  TR1003LG

        22A0  TR1099LG           CSECT       886  TESTLIB   01  TR1099LG

        2B28  MAGCODE         *  CSECT       188  SYSLIB    01  MAGCODE

           ..............
```

The Relative Interrupt Address can then be compared with the OFFSET or starting address of each module within the program to find the module in which the abend occurred.

In our example the Relative Interrupt Address is C50 and therefore the abend occurred in module TR1001LG which starts at address 408 and has a length of 11E6. The next module starts at address 15F0; as C50 is greater than 408 but less then 15F0 we can be sure that the abend occurred before that module was entered.

Finding the statement causing the abend

To find this, once again we need to use the Linkage Editor Report and the Relative Interrupt Address, as well as the module compilation listing.

To find the statement in error we simply subtract the origin or starting address of the module, from the Relative Interrupt Address.

DISPLACEMENT WITHIN MODULE	=	RELATIVE INTERRUPT ADDRESS	-	START OF MODULE (ORIGIN FROM LINK EDITOR REPORT)
848	=	C50	-	408

We can now find the COBOL statement in error by looking in the CONDENSED LISTING of the appropriate module.

```
 000185                   MULTIPLY HRS-WKD BY PAY-RATE GIVING WS-WAGE.

LINE #  HEXLOC  VERB                        LINE #  HEXLOC  VERB
000185 000840 MULTIPLY                      000187 0008A4 CALL
```

To determine the contents of the fields involved, we follow exactly the same procedure as we would for a single-module program: we consult the Data Division Map to find the appropriate base locators, displacements and data types, then the VS COBOL II Abend information File to discover the addresses of the base locators and finally we look up the fields in the dump.

```
Data Division Map
Source   Hierarchy and                Base      Hex-Displacement  Asmblr Data
LineID   Data Name                   Locator    Blk   Structure   Definition
Data Type
    31     02 HRS-WKD . . . . . . . . BLF=0000   004   0 000 004   DS 2C
Disp-Num
    32     02 PAY-RATE. . . . . . . . BLF=0000   006   0 000 006   DS 4C
Disp-Num
   141      02 WS-WAGE . . . . . . . . BLW=0000    498    0 000 000    DS 4P
Packed-Dec

--- VS COBOL II ABEND Information ---
Contents of base locators for files are:
    0-00028E70
Contents of base locators for working storage are:
    0-00026310
```

⇒ address of field in dump = base locator address + displacement

HRS-WKD= 28E70 + 4= 28E74

PAY-RATE= 28E70 + 6= 28E76

WS-WAGE= 26310 + 498= 267A8

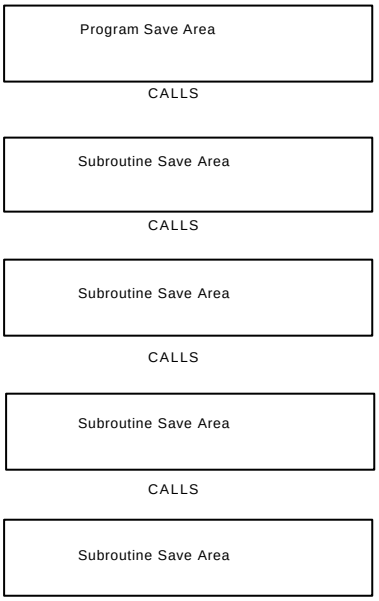
Looking up the addresses in the dump, we find:

```
00028E60 00000000 00000000 00028EC0 00050050    F4F3F3F3 F67BF8F9 F1F24040 40404040
*.................43336.8912      *
000267A0 40404040 00000000 0059994C F0F8F1F2    F9F20000 00000002 0F00004F 00000F00
                                                  *    .........081292..............*
```

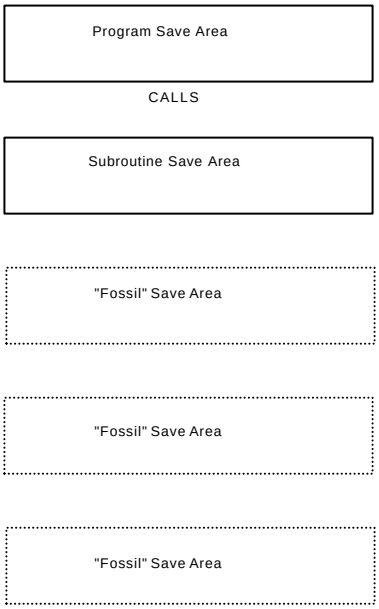
HRS-WKD contains 6#
PAY-RATE contains 8912
WS-WAGE contains +59994

⇒

Save Area Trace after first call



Save Area Trace after subsequent call



THE SAVE AREA CHAIN

Whenever a call is made to a module or subroutine, the **MVS** operating systems sets up a portion of memory known as the **Save Area**. The **Save Area Chain** can be found in your compile job. This is done principally so that the operating system knows where to return control to once the module or subroutine has finished executing. If a subroutine or module then calls a further sub-module, a new save area is set up for that call. However, once control is finally returned to the top most module, any subsequent calls to modules will **reuse** the save area chain. This can lead to "fossil" save areas, as shown on the diagram below:

In this example, the first call has lead to a chain of 5 save areas being built up. The next call, however, only results in a chain of two - leaving three "fossil" save areas left over from the preceding call.

You can tell how many save areas are currently active (i.e. not "fossils") by seeing how many are listed at the end of the Save Area chain. Here, the operating system is attempting to re-trace its steps back to the top most module. In the example on page 9, there are two currently active save areas. Those at the bottom of the trace are the same as the first two in the trace but in reverse order.

The Save Area Chain may be useful in solving abends, particularly if the abend occurred in a subroutine rather than the main body of your program. It will show you what was the last module called and the contents of the general purpose registers (GPR) at the time of that call. Each save area consists of 18 fields, each a full word (4 bytes binary), formatted as follows:

| WD1 | HA | LSA | R14 | R15 | R0 | R1| .. | R11 | R12 |

(Rnn is the general purpose register nn)

The HA field is a pointer to the save area of the **H**igher or calling module, and the LSA is a pointer to the save area of the **L**ower or called module. Thus the save areas form a chain linking the various modules together.

For example, in the save area trace on page 9, the first save area (i.e. that set aside for the top most program) starts at 00005FA8. It has a HA of 00000000 - which is normal for the first save area. The LSA of 00025E88 points to the SA in the save area below. In turn, the HA in that, 00005FA8, points to the SA of the calling module.

```
SAVE AREA TRACE
PROCEEDING FORWARD FROM TCBFSAB
TR9ADC   WAS ENTERED VIA LINK         AT EP TR9MM1...C2.1.3.0.12.08.92.14.42.27
SA   00005FA8  WD1 00000000 HA 00000000 LSA 00025E88 RET 80FDAC10 EPA 000067E8 R0
FD000008
               R1  00005FF8 R2  00000040 R3  008F69A4 R4  008F6980 R5  008F3470 R6
008CAFF8
               R7  FD000000 R8  008FE010 R9  808FF408 R10 00000000 R11 008F3470 R12
00D77CDA
TR9ADC   WAS ENTERED VIA CALL         AT EP BPSAIWR.84.298.BPSAIWR.
SA   00025E88  WD1 00108001 HA 00005FA8 LSA 00005000 RET 50007074 EPA 00007738 R0
00025FF8
               R1  0002607C R2  00026610 R3  00026374 R4  00000021 R5  700070A8 R6
008CAFF8
               R7  FD000000 R8  00028E70 R9  00026310 R10 00006894 R11 00006B08 R12
00006868
TR9ADC   WAS ENTERED VIA CALL         AT EP BPSAILK.......BPSAILK
SA   00005000  WD1 00000000 HA 00025E88 LSA 00005054 RET 60007982 EPA 000078F8 R0
00000008
               R1  0002607C R2  00025E88 R3  00026374 R4  00000021 R5  700070A8 R6
008CAFF8
               R7  FD000000 R8  00028E70 R9  00026310 R10 00000008 R11 00005000 R12
0002B970
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSINSA.87.043.BPSINSA.
SA   00005054  WD1 00000000 HA 00005000 LSA 00032420 RET 6002BB78 EPA 0002DB58 R0
00000008
               R1  0002607C R2  00005054 R3  00026374 R4  0002607C R5  700070A8 R6
00005000
               R7  FD000000 R8  00028E70 R9  00026310 R10 00000008 R11 00032388 R12
0002B970
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSINLR.87.043.BPSINLR.
SA   00032420  WD1 00000000 HA 00005054 LSA 00032468 RET 5002DCA8 EPA 0002E2B0 R0
00000008
               R1  00026088 R2  00000000 R3  80026610 R4  00000000 R5  80026610 R6
0002DEA0
               R7  FD000000 R8  00028E70 R9  00026310 R10 00000008 R11 00032388 R12
4002DC56
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSSVML.86.323.BPSSVML.
SA   00032468  WD1 00000000 HA 00032420 LSA 000324B0 RET 4002E670 EPA 0002E980 R0
00000008
               R1  00026088 R2  00000000 R3  80026610 R4  00000000 R5  80026610 R6
0002DEA0
               R7  FD000000 R8  00028E70 R9  00026310 R10 00033470 R11 00032388 R12
4002E63E
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSWRPR.92.260.BPSWRPR.
SA   000324B0  WD1 00000000 HA 00032468 LSA 000324F8 RET 4002EBEE EPA 00029B78 R0
00000008
               R1  00026088 R2  0002EC08 R3  00029B78 R4  00000000 R5  00000000 R6
0002DEA0
               R7  FD000000 R8  00028E70 R9  00026310 R10 00033470 R11 00032388 R12
6002E9C8
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSWRLO.92.260.BPSWRLO.
SA   000324F8  WD1 00000000 HA 000324B0 LSA 00032540 RET 70029DB0 EPA 00029ED0 R0
00000008
               R1  00026088 R2  0002EC08 R3  00029B78 R4  0000000C R5  0000003C R6
0002DEA0
               R7  00033358 R8  00028E70 R9  00026310 R10 00033470 R11 00032388 R12
60029BC0
BPSFILE  WAS ENTERED VIA CALL         AT EP BPSWRPL.92.260.BPSWRPL.
SA   00032540  WD1 00000000 HA 000324F8 LSA 00032588 RET 70029F56 EPA 0002B6F0 R0
00000008
```

Finding the Absolute Address of Each Module

When using the save area trace it is useful to know the Absolute Address of the start of each module.

First find the Origin of the program. This is the value given in the ADDRESS field in file one of the job (see page 3). To find the absolute start address, we simply add on the displacement given in the linkage editor report.

Thus:

TR9MM2 starts at	6330	+	0	=	6330
TR9MM1 starts at	6330	+	4B8	=	67E8
BPSAIOP starts at	6330	+	1398	=	76C8
BPSAIOL starts at	6330	+	13D0	=	7700
BPSAIWR starts at	6330	+	1408	=	7737

The three modules beginning BPS... are the actual external names of the BPS subroutines. The subroutine names that we are more familiar with are given in the linkage editor report to the right and below the external names; thus, BPSAIWR actually refers to the subroutine we call BPSWRITE.

Entry Points

The **EPA** field contains the Entry Point Address of the module. By working out these addresses, the modules last called can be traced. In our example save area trace, the first two EPAs are:

EPA 67E8 which is the start of module TR9MM1
EPA 7738 which is the start of the BPSWRITE module

Therefore, the last call made by program TR9MM1 was to BPSWRITE. This method could be used to find the module which called a subroutine in which an abend has occurred.

To find the module in which the abend occurred, you will need to use the WD1 field in the save area:

1) Find the save area chain and go to the bottom of it.

2) Working your way up the chain, search for the string 00108001 in the WD1 field.

3) When you have found the string, look at the save area **following the block that contains the string 00108001**. Look at the EPA field in that block - it will be the entry point of the module in which the abend occurred.

Using our example save area, we can see that the last but one save area contains the string 00108001. The EPA field in the following block contains the value 000067E8, which corresponds with the entry address of module TR9MM1; therefore, the abend occurred in that module.

Return Addresses

This can be used to find the statement following the call to an abending subroutine/module.

The **RET** field indicates the return address within the higher or calling module. However, only the **last three bytes** of the field are significant. To find the Relative Return Address within the module all you need to do is to subtract the EPA of the higher module from the RET of the lower module. This address can then be looked up in the Condensed Verb Listing.

From our example:

BPSWRITE RET = 50007074 ⇒ discard first byte = 007074

TR9MM1 EPA = 000067E8

Return Address = 7074 - 67E8 = 88C (in TR9MM1)

```
000230 000810 MOVE                 000232 00083A ADD                  000196 000862
CALL
000198 000896 SET                  000200 0008A8 PERFORM              000244 0008A8
READ
   000196                                CALL  'BPSWRITE'  USING  COIN-REP,  SPACE2,
REP-DETAIL-LINE.
  000197
  000198                    SET MONEY-INDEX COIN-INDEX TOTAL-INDEX TO BASE-INDEX.
```

We can see from the compiler listing that this return address falls just after a call to BPSWRITE - just as we deduced from the linkage editor listing and the save area trace.

Finding Parameters Passed to Modules

It is often useful to find the values in the variables passed as parameters between program modules. There are several ways of doing this but if you have a compiler listing and a VS COBOL II abend information file (ddname SYSABOUT) the simplest method is as follows:

The field we want to look up is B-WAGE. From the module compilation listing:

```
000049                LINKAGE SECTION.
000050               *----------------*
000051                01  B-TABLE.
000052                    05 B-QUANTITY   PIC 999 OCCURS 11 TIMES
000053                                    INDEXED BY B-INDEX.
000054
000055                01  B-WAGE             PIC S9999V99 COMP-3 VALUE +0.
```

From the condensed verb listing:

```
    55   01 B-WAGE. . . . . . . . . . . . BLL=0002   000         DS 4P
Packed-Dec
```

From the COBOL II abend information file:

```
Contents of base locators for the linkage section are:
    0-00000000       1-00006330       2-00005FFE
```

this field can be found in the dump at address 5FFE (base locator + displacement):

```
00005FE0 808F32A0 00000000 008F3470 00D77CDA     00000000 00000000 80005FFE 0000000
   00006000   0000000  00000000  00000000  00000000       00000000  00000000  00000000
00000000
```

B-WAGE contains x00000000

This method will not work if your program has abended within a subroutine which you do not have a compilation listing for (e.g. a BPS routine). Such occasions are fortunately rare - subroutines are generally written such that they do NOT abend! In such situations, we can exploit a feature of COBOL - that a subroutine using parameters uses the same area of memory to store the values as the calling program. Hence, if you look up the parameters in the storage area of the calling program, you will be able to see the contents of the parameters used in the subroutine.

✶✶✶ Note: this method will NOT work if the BY CONTENT clause is specified on the CALL statement which invokes the subroutine.

EXERCISE

Import MMDUMP1 and MMDUMP2 from DEV@@TR.TEST.SORCLBN, and submit both jobs. They will produce a multi-modular dump which you can then solve.

- What module did the program abend in?
- At what relative offset within that module did the abend occur?
- On which line did the abend occur?
- What was the cause of the abend?

Show your results to your tutor including an explanation of why the abend occurred.

APPENDIX A - ADVANCED DUMP SOLVING

The methods outlined above will be sufficient for solving all but the most complicated storage dumps. What follows is intended to act as additional information for the very keen! For more information, please consult the IBM Application Programming Debugging guide, SC26-4049.

The Task Global Table (TGT)

The TGT contains information about the program that was running at the time of the abend. It keeps track of information pertinent to the program, such as:

- the location of the register save area
- the location and length of working storage
- pointers to the location of FCBs (File Control Blocks).

The TGT can be found in one of two ways:

- from the SYSABOUT file
- from the Save Area Trace.

The TGT starts with the program save area, the location of the TGT will be the same as the value given in the SA field of the save area block of the module in which the program abended (see page 10 for a description of how to find the abending module). So, for example:

```
--- VS COBOL II ABEND Information ---
Program = 'TR9MM1' compiled on '12/08/92' at '14:42:27'
    TGT = '00025E88'

INTERRUPT AT 00006F86
PROCEEDING BACK VIA REG 13
TR9ADC   WAS ENTERED VIA CALL         AT EP BPSAIWR.84.298.BPSAIWR.
SA   00025E88 WD1 00108001 HSA 00005FA8 LSA 00005000 RET 50007074 EPA 00007738 R0
00025FF8
             R1  0002607C R2  00026610 R3  00026374 R4  00000021 R5  700070A8 R6
008CAFF8
             R7  FD000000 R8  00028E70 R9  00026310 R10 00006894 R11 00006B08 R12
00006868
TR9ADC   WAS ENTERED VIA LINK         AT EP TR9MM1...C2.1.3.0.12.08.92.14.42.27
SA   00005FA8 WD1 00000000 HSA 00000000 LSA 00025E88 RET 80FDAC10 EPA 000067E8 R0
FD000008
             R1  00005FF8 R2  00000040 R3  008F69A4 R4  008F6980 R5  008F3470 R6
008CAFF8
             R7  FD000000 R8  008FE010 R9  808FF408 R10 00000000 R11 008F3470 R12
00D77CDA
```

You can identify a VS COBOL II TGT because it starts with the string "00108001" and contains the text "C2TGT+48" at offset x48 from the start of the TGT.

The figure below contains the start of a TGT:

```
00025E80 00026030 00025E30 00108001 00005FA8     00000000 00000000 00000000 00000000
  *.................................*
00025EA0 00000000 00000000 00000000 00000000     00000000 00000000 00000000 00000000
  *.................................*
00025EC0 00000000 00000000 00000000 00000000     C3F2E3C7 E34EF4F8 02000000 61100220
  *................C2TGT.48........*
```

Using the TGT

There are a number of uses to which the information in the TGT can be put; however, these notes will concentrate on just one - the File Control Blocks (FCBs) associated with a program. The FCB is the control block that VS COBOL II uses to control a dataset or file. In the FCB, you can find information on how the file was opened or closed, a pointer to the File Information Block (FIB) and information on the progress of the I/O processing on a file.

The TGT contains:

- at offset x64, the number of FCBs associated with the program.
- at offset x110, a pointer to the address of the FCB pointer.

If we use the TGT described on page 14 as an example:

- ☞ the number of the FCBs may be found at location 25E88 + 64 = 25EEC
- ☞ the FCB pointer will start at location 25E88 + 110 = 25F98.

```
00025EE0 00025CC8 000052E4 000260E8 00000001     000004C8 00000000 00000000 00009020
  *...H...U...Y.......H............*
00025F80 00006969 00000000 000067E8 000068B0     00026010 00006890 00026064 00026310
  *...........Y....................*
```

Hence, we can see that there is 1 FCB associated with this program and it's location is pointed to by the four bytes beginning at address 26064 in the dump.

```
00026060 00000000 00026190 C0000000 00000000    00000000 000262A8 00026700 0008912F
```

If there were any more FCBs associated with this program, they would be pointed to by the next four bytes of the pointer list (i.e. location 26068).

Going to location 26190, we find the FCBID field. The first three bytes are the letters "FCB", the next two the FCB number and the next byte after that the FCB level number.

```
00026180 00026290 00026100 00026A0 00026110  C6C3C200 0102 0000 FFFFFFFF FFFFFFFF
```

Hence this is FCB number 0001 and the level number is 02.

For more information concerning the TGT and FCBs, refer to the **IBM VS COBOL II** manual, **Application Programming Debugging (SC26-4049).**

Finding What Parameters have been Passed to Modules

On page 12, we looked at two ways of determining what parameters had been passed between programs. However, both rely on you having a compilation listing of the abending module. This is not always the case! This third method will work without the aid of a compilation listing.

The method involves using the Save Area Trace:

❶ firstly, go to the **bottom** of the trace. Register 1 (R1) points to list of addresses of the parameters being passed to the called module.

❷ Go to the address shown in R1 in the save area. This will contain a list of addresses. These addresses correspond to the location of the parameters passed between the subroutine and the calling module. For example:

0000A788 000093D8 8000A768

Note that the third address has got its first bit set on, i.e. it begins with '80', this indicates that it is the last parameter passed.

❸ The three addresses:

A788 93D8 A768

can now be looked up in the dump to find their current values.