**CSCI 360 7. ABENDS, Dumps and the PSW**

If your program **ABEND**s (**AB**normally **END**s), ASSIST generally will provide you with a memory dump which can help you isolate the reason for the ABEND. The only instances in which you will not get a dump are those in which your job exceeds the time limit (no time remains to generate the dump), or the job generates more than the number of lines allowed (2000 is the default maximum number.)

**The information available to you in a dump includes:**

**The contents of the PSW**.

See Program Status Word (PSW) below

**The completion code**.

See your yellow card for a listing of program interruption codes - also, appendix D of your text gives a good explanation of the more commonly encountered completion codes, along with various programmer errors which can cause those types of interrupt.

**A trace of the last few instructions executed**.

**A trace of the last few branch instructions executed**.

**Contents of the 16 general purpose registers**

At the time of the ABEND. Since you are not going to be using the floating point registers, you can ignore them.

**The contents of user storage**

The portion of main storage used by your program and its save areas is dumped in hexadecimal. Each line contains 32 bytes of storage. In the left-hand margin you will find the address of the first of these 32 bytes (LOC.) On the right-hand margin you will find (between two \*s) a translation into character form, where alphabetic and numeric characters and blanks are identified whenever a byte contains the character's encoded form a period is printed to represent any other byte values.

**When your program ABENDs, you should be able to answer questions such as:**

1. What was the reason for the ABEND (interruption code)?
2. What does that interruption code mean?
3. What was the last instruction executed?
4. Did the registers contain the "right" values?
5. Were the contents of user storage correct?

**Program Status Word (PSW)**

The Program Status Word or PSW is a collection of data 8 bytes (or 64 bits) long, maintained by the operating system. It keeps track of the current state of the system.

We can usually ignore the PSW unless an ABEND has occurred. When an ABEND does occur, ASSIST will print out various information for us including the PSW. The PSW is printed out as 16 hex digits in two groups of 8.

**What information is in the PSW?**

You can find a detailed list of the fields in the PSW in the yellow card. We use the "BC Mode" of the PSW. Here is a list of some fields we will need in this course:

**Bytes Contents**

1 & 2: assorted data we can ignore for now

3 & 4: Interruption Code

5: 2 bits = Instruction Length Code (ILC)

2 bits = Condition Code (CC)

4 bits we can ignore for now

6 - 8: Address of the next instruction

**What are all these?**

* The Interruption Code indicates the type of ABEND that has occurred.
* The ILC gives us the length of the current instruction, measured in halfwords,
* The CC gives us the condition code as set most recently.
* The address of the next instruction gives us the location of the instruction that would have been executed if the program had not ABENDed.

**So what do we do with all this?**

One problem with an ABEND is to determine which instruction caused the ABEND. We can find it using the information in the PSW:

Address of ABENDing instruction = Address of next instruction - 2 \* ILC

**Example**

Suppose an ABEND occurs and the PSW has the value FFC50001 8000001A.

We can look at this and know that:

* The interrupt code is 0001 (Operation Exception).
* The ILC is 10 (binary) or 2 (decimal), so the ABENDing instruction is 4 bytes long.
* The CC is 00.
* The address of the next instruction is 00001A.
* The address of the ABENDing instruction is 00001A - 4 = 000016.

**Dump Example 1**

Type in and run the following program:

DUMP1 CSECT

USING DUMP1,15 ESTABLISH A BASE REGISTER

L 1,ONE LOAD THE FIRST NUMBER INTO R1

L 2,TWO LOAD THE SECOND INTO R2

AR 1,2 ADD THE TWO NUMBERS

ST 1,THREE STORE THE RESULT

XDUMP THREE,4 DUMP THE RESULT

BCR B'1111',14 RETURN TO CALLER

\*

ONE DC F'64' FIRST NUMBER

TWO DC F'32' SECOND NUMBER

EOFFLAG DC C'0' A FLAG SAVE AREA

THREE DS CL4' ' SUM OF THE TWO NUMBERS

END DUMP1

**After running the above program you should be able to answer the following questions:**

What is the address of the next instruction which will be executed?

What is the address of the instruction that caused the abend?

What type of error occurred?

What actually causes this error?

Correct the error by rewriting the section of code that caused it.

What is the contents of register 1 in decimal?

What does the value in reg 1 represent at the time of ABEND?

Why is the LOC address of the storage area with the label ONE on it 000018 when the branch statement before it whose LOC address is 000014 only takes up 2 bytes?

What are the contents of the two bytes of user storage starting at address 000016? What do they represent?

What are the contents of the byte saved at address 00001B? Does this byte represent the first byte of a full word?

If the dump program error were corrected, what value would the storage area at label THREE contain?

What two instructions have you worked with which cause data conversion to take place?

What is the decimal equivalent of hex 0002BA14?

**Dump Example 2**

Type in and run the following program:

DUMP2 CSECT

USING DUMP2,15

LA 2,TABLE

SR 3,3

XREAD DATA,80

LOOP1 BM ENDLOOP1

XDECI 4,DATA

ST 4,0(2,3)

LA 3,4(,3)

XREAD DATA,80

B LOOP1

ENDLOOP1 SR 3,3

LA 7,TABLE

LA 5,TABEND

LOOP2 CR 2,5

BE ENDLOOP2

L 6,0(,2)

ST 6,0(,7)

L 7,4(,7)

LA 2,4(,2)

B LOOP2

ENDLOOP2 BR 14  
\*

LTORG

\*

DATA DS CL80

TABLE DC 30F'-1'

TABEND DS 0X

END DUMP2

0

1 2

50

32 24 19 62

123 456 789

987 654 321

**Using the results from the program, answer the following questions:**

What was the interruption code?

What instruction caused the program to abend? Why?

What was the condition code at the time of the ABEND?

How many table entries were built? How did you figure this?

What is the return address to the calling routine? Where did you find this? Does your answer really make any sense?

What are the contents of register 7?

Was any object code changed by this program? If so, for which instructions?

Finally, explain why the program ABENDed.