# DEBUGGING COBOL II/3000 PROGRAMS

Greg Gloss
Hewlett-Packard General Systems Division

This paper provides some tips for debugging programs compiled with the new COBOL II/3000 compiler. A discussion of compile time aids is followed by a description of the run time environment and guidance for using DEBUG/3000 to debug your programs interactively.

## COMPILE TIME AIDS

Users of COBOL/3000 will notice three major changes in compile time debugging aids with COBOL II/3000. The first change is that diagnostic messages are now listed together at the end of the listing. In the following sample program the hyphen was left out of N-2 in line 22. Note that the diagnostic listing references both a compiler generated line number (5 digits) and the 6 digit sequence field.

```
PAGE 0001   HEWLETT-PACKARD 32233A.00.00  COBOL II/3000

00001           001000$CONTROL CROSSREF,VERBS,MAP
00002           001100 IDENTIFICATION DIVISION.
00003           001200 PROGRAM-ID. SAMPLE.
00004           001300 ENVIRONMENT DIVISION.
00005           001400 DATA DIVISION.
00006           001500 WORKING-STORAGE SECTION.
00007           001600 01  GRP.
00008           001700     05  N-1 PIC S9(4) VALUE 0
00009           001800            SIGN IS LEADING SEPARATE.
00010           001900     05  N-2 PIC S9(4) VALUE 0
00011           002000            SIGN IS LEADING SEPARATE.
00012           002100     05  N-3 PIC S9(4) COMP-3.
00013           002200 PROCEDURE DIVISION.
00014           002300 SEC-1 SECTION 01.
00015           002400 BEGIN-HERE.
00016           002500     DISPLAY "ENTER N-1, 4 DIGITS".
00017           002600     ACCEPT N-1 FREE.
00018           002700     DISPLAY "N-1 = ", N-1.
00019           002800 SEC-2 SECTION 02.
00020           002900 PAR-2.
00021           003000     DISPLAY "ENTER N-2, 4 DIGITS".
00022           003100     ACCEPT N2 FREE.
00023           003200     DISPLAY "N-2 = ", N-2.
00024           003300     MOVE N-2 TO N-3.
00025           003400 SEC-1A SECTION 01.
00026           003500 STOP-HERE.
00027           003600     STOP RUN.
```

```
COBOL ERRORS:

 LINE #  SEQ #    COL  ERROR  SVRTY  TEXT OF MESSAGE
-------------------------------------------------------------

 00022  003100    21    356     Q     UNDEFINED DATA NAME N2

0 ERRORS, 1 QUESTIONABLE,  0 WARNINGS
```

The other new features which have been added are a cross reference capability and a verb map listing. The CROSSREF option on the $CONTROL command generates an alphabetical listing of user-defined symbols in the program with the line numbers in which they are referenced. The following listing shows the cross reference generated after the above program was corrected.

```
COBOL CROSS REFERENCE LISTING


BEGIN-HERE
00015

GRP
00007

N-1
00008    00017    00018

N-2
00010    00022    00023    00024

N-3
00012    00024

PAR-2
00020

SEC-1
00014

SEC-1A
00025

SEC-2
00019

STOP-HERE
00026
```

For example, N-2 is referenced in 4 places: lines 10, 22, 23, and 24.

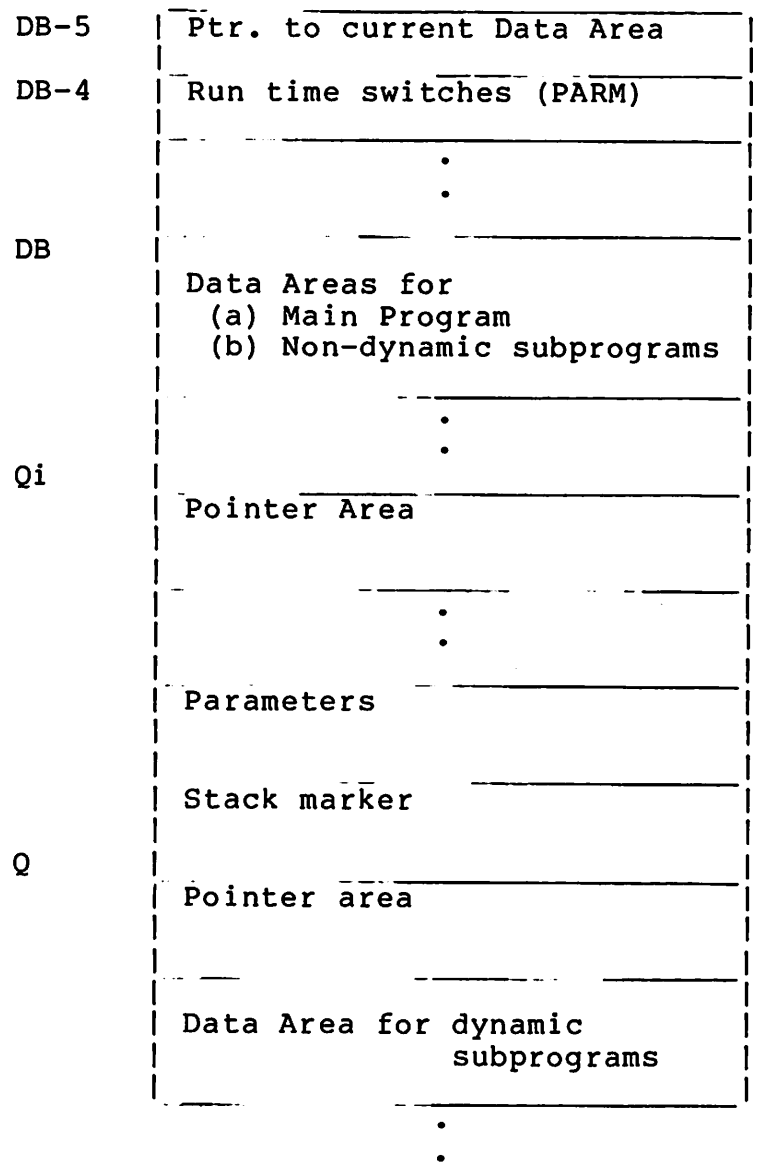
The third change is the capability to list the starting address of code generated for each verb instead of just each paragraph. The VERBS option of the $CONTROL command lists these addresses which are relative to the starting address of the RBM (Relocatable Binary Module) as shown in the PMAP. The following listing shows the address of the ACCEPT statement in line 17 to be 33 relative to the start of the RBM SEC101'. This address can be used with DEBUG/3000 to set a breakpoint at the start of this statement.

```
 LVL SOURCE NAME                    BASE DISPL    SIZE    USAGE


 WORKING-STORAGE SECTION

 01  GRP                           Q+2: 000350   000015 DISP
 05   N-1                           Q+2: 000350   000005 DISP
 05   N-2                           Q+2: 000355   000005 DISP
 05   N-3                           Q+2: 000362   000003 COMP-3

PAGE 0003/COBTEXT  SAMPLE          SYMBOL TABLE MAP
 LINE #    PB-LOC        PROCEDURE NAME/VERB      INTERNAL

 00014     000003      SEC-1                      SEC101'
 00015     000003          BEGIN-HERE
 00016     000003              DISPLAY
 00017     000033              ACCEPT
 00018     000042              DISPLAY
 00019     000003      SEC-2                      SEC202'
 00020     000003          PAR-2
 00021     000003              DISPLAY
 00022     000033              ACCEPT
 00023     000042              DISPLAY
 00024     000071              MOVE
 00025     000003      SEC-1A                     SEC1A01'
 00026     000003          STOP-HERE
 00027     000003              STOP
```

RUN TIME ENVIRONMENT

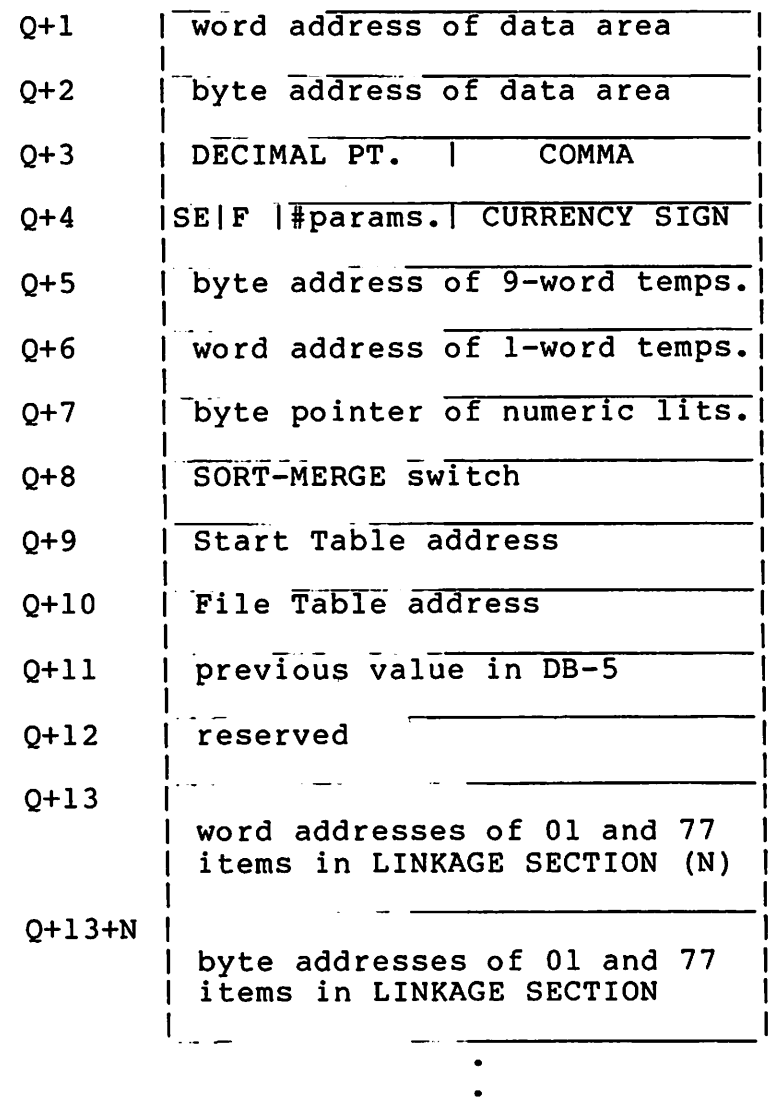
[NOTE: All information presented in this area is for help in debugging programs. The run time structure is subject to change in future releases so applications should not be developed which depend on this structure remaining constant.]

The stack layout for a COBOL II program is shown below. Each program unit (Main program or subprogram) has two areas in the data stack: a data area and a pointer area. For main programs and non-dynamic subprograms, the data areas are located between DB and Q initial. The pointer areas for all types of program units are dynamically allocated starting with Q+1 upon entry to the program unit. For dynamic subprograms, the data area is allocated after the pointer area.



Pointer Area

The pointer area contains the addresses of data structures in the data area and other information local to the program unit. All types of COBOL II program units have the same format for the pointer area except that main programs do not have an area for Linkage Section addresses. For subprograms, each 01 or 77 level item in the Linkage Section is assigned a 1-word location at or above Q+13 for the word address. This address is moved from the appropriate Q-negative location upon entry to the subprogram. In addition, another 1-word location is allocated for the corresponding byte address. If a Linkage Section item is not referenced in the USING phrase of the Procedure Division header or ENTRY statement, these locations will contain an illegal address. The format of the pointer area is shown below:

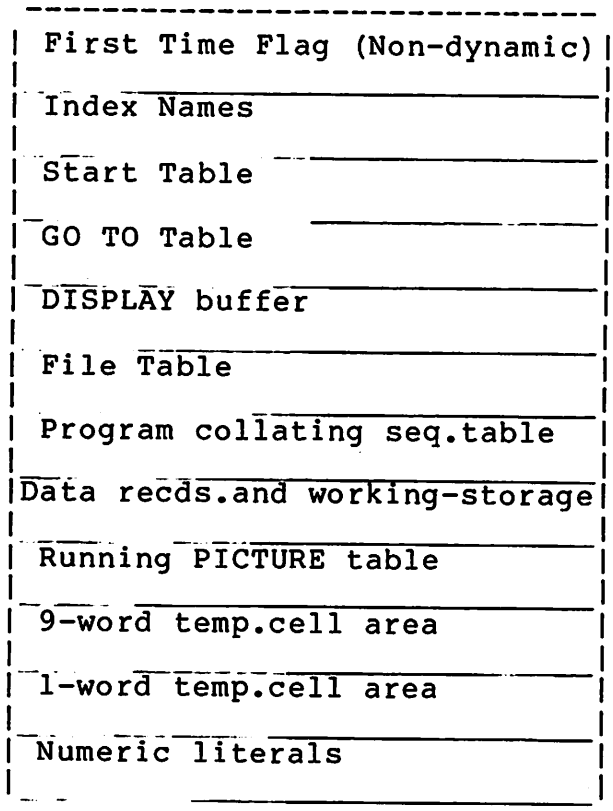



DECIMAL PT. and COMMA are character representations of DECIMAL-POINT and COMMA.

F(1 bit)=First-time flag for PERFORM statements
SE(1 bit)= SIZE ERROR flag: 1- size error occurred
#params(6 bits)= no.of parameters
CURRENCY SIGN = character representing currency sign

Data Area

The data area contains a combination of compiler generated structures and user defined data areas. These areas are described below:




1. First Time Flag (Non-dynamic Subprograms only). A 1-word value indicating whether or not it is the first call to a Non-dynamic subprogram.

2. Index Names (Optional) One word is allocated for the value of each index-name declared in the program.

3. Start Table. Each paragraph/section generates a two-word entry containing its starting PB-address and segment number.

4. GO TO Table (Optional). Each Alterable GO TO (a paragraph whose first or only statement is a simple GO TO) generates a three word entry for use with the ALTER statement.

5. DISPLAY Buffer (Optional) A 200-byte buffer for values to be DISPLAYed.

6. File Table (Optional). Contains information pertaining to each user specfied file.

7. Program collating seq. Table (Optional). A table containing the mapping for characters if a PROGRAM COLLATING SEQUENCE is specified in the Environment Division.

8. Data Records and Working Storage (Optional). This area contains the values of User defined data items declared in the File and Working-Storage Sections.

9. Running Picture Table (Optional). Contains information pertaining to data items and is used by the run-time library for certain constructs.

10. 9-word temp cell area (Optional). Used for intermediate values in arithmetic computations.

11. 1-word temp cell area (Optional). Used for intermediate values in arithmetic computations.

12. Numeric Literals (Optional). Contains representations of some of the numeric literals used by the program.

Code Segmentation

Each COBOL II program unit generates at least two RBMs. The first RBM is an initialization module. For main programs, this unit is the Outer Block where control is transferred when the program is started. The name of this module is the same as the PROGRAM-ID name (stripped of hyphens). For subprograms, an apostrophe is appended to the PROGRAM-ID name to form the name of the initialization module. The Procedure Division will generate one or more RBMs depending on the segment numbers (if any) which are specified in the section headers. If no segment numbers are specified, the entire procedure division goes in one RBM. Consecutive sections with the same segment number are placed in the same RBM. The RBM name is listed in the compile-time MAP under the heading "INTERNAL NAME." The initialization RBM is put into the same code segment as the last RBM of the Procedure Division. Non-contiguous sections with the same segment number will generate multiple RBMs but they all will be put into the same code segment.

The PMAP from the sample program is shown below:

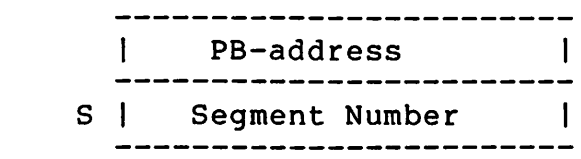
```
SEC202'            0
   NAME             STT  CODE ENTRY SEG
   SEC202'            1     0     0
   C'DISPLAY          2                 ?
   C'DISPLAY'FIN      3                 ?
   C'DISPLAY'INIT     4                 ?
   C'DISPLAY'L        5                 ?
   ACCEPT'FREE'C      6                 ?
   SEGMENT LENGTH          124
SEC101'            1
   NAME             STT  CODE ENTRY SEG
   SEC1A01'           1     0     0
   TERMINATE'         4                 ?
   QUIT               5                 ?
   SAMPLE             2     6     6
   DEBUG              6                 ?
   COBOLTRAP          7                 ?
   SEC202'           10                 0
   SEC101'            3   216   216
   C'DISPLAY         11                 ?
   C'DISPLAY'FIN     12                 ?
   C'DISPLAY'INIT    13                 ?
   C'DISPLAY'L       14                 ?
   ACCEPT'FREE'C     15                 ?
   SEGMENT LENGTH          334
```

Notice that sections SEC-1 and SEC-1A generate different RBMs (SEC101' and SEC1A01'), but are both in the same code segment because they both have a segment number of 01. Since SEC-1A is the last section of the Procedure Division, the initialization module SAMPLE is also put in this segment. If the program were changed such that SEC-1A came between SEC-1 and SEC-2 then both SEC-1 and SEC-1A would be in the same RBM and the initialization module would go in the same segment as SEC-2.

Implementation of PERFORM and GO TO

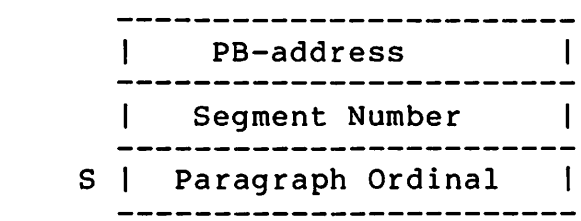
The implementation of PERFORM and GO TO statements has been changed with COBOL II/3000. Three new instructions have been added to the machine instruction set to perform these operations. For GO TO statements across section boundaries, the new External Branch (XBR) instruction is used as follows:

The two word value from the Start Table for the paragraph being branched to is loaded onto the top of the stack

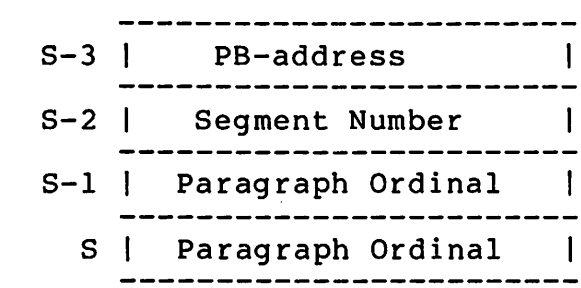



and the XBR instruction transfers control to the specified location, deleting the 2 words from the stack.

For all PERFORM statements, the Paragraph Call (PARC) instruction is used to initiate a PERFORM. First, the two word Start Table entry for the starting paragraph of the PERFORM is loaded onto the top of the stack followed by the ordinal of the last paragraph being performed.




The PARC instruction replaces the PB address and Segment Number with the appropriate return values and then transfers control. To return from a PERFORM statement, the End Paragraph (ENDP) instruction is used. First the ordinal of the paragraph being concluded is loaded onto the stack.




The ENDP instruction compares the two paragraph ordinals and if equal returns control to the location specified in S-2 and S-3 and deletes the four words from the stack. If the paragraph ordinals are not equal only one word is deleted from the stack and control falls through to the next paragraph.

USING DEBUG/3000

To use DEBUG/3000 you should have a compilation listing of the program units with the MAP and VERBS options specified on the $CONTROL command together with the PMAP for the program file. The CROSSREF option on the compilation listing may also prove useful. There are two ways to get into DEBUG with a COBOL II program. The first is to specify DEBUG on the :PREPRUN or :RUN command. The second is to specify the DEBUG option on the $CONTROL command when the program is compiled. This option sets the CONTROL-Y trap so that pressing Control-Y when the program is executing will invoke DEBUG.

To debug the sample program shown earlier,

```
:RUN SAMPPROG;DEBUG
```

To set a breakpoint, get the offset from the VERBMAP for the statement and add the starting address of the code module as shown in the PMAP. For example, to set a breakpoint at the ACCEPT statement in line 17, get the logical segment number of the segment containing SEC101' (which is 1) and the starting address of the module (which is 216). From the VERBMAP, the offset for the ACCEPT statement is 33. So, when the program starts,

```
*DEBUG* 1.6
?B1.216+33
```

sets a breakpoint at the start of the first ACCEPT statement. To set another breakpoint at the STOP RUN statement in line 27, you will notice that module SEC1A01' starts at PB+0 and the STOP RUN statement starts at 3.

```
?B1.3
?R
```

The R will resume execution of the program.

```
ENTER N-1, 4 DIGITS

*BREAK* 1.251
?
```

Now we are at the start of the ACCEPT statement. To verify the initial value of N-1 as 0, display its contents.

```
?D ('Q+2'+350)/2,3,A
DB+164    +0000+
```

The 'Q+2' means the contents of Q+2 which contains the byte address of the data area. The 350 is the byte address of N-1 as shown in the MAP. The /2 converts the byte address to a word address and the 3 specifies the length as 3 words. Since the item is USAGE DISPLAY, the A specifies to display the value in ASCII.

Since the item is actually only 5 bytes long, displaying 3 words picks up the first byte of the next item also. For items in the Linkage Section the Q+2 would have to be replaced by the value shown in the BASE column of the MAP. This value is currently displayed as a positive decimal value (Q14 means Q+14 (decimal)). To use this value in a DEBUG command, precede the numeric offset by #; for example, Q+#14. A future release will probably change this to an octal representation to ease using DEBUG.

To continue running until the other breakpoint, the following is keyed in:

```
?R

123                  Value of N-1 keyed in
N-1 = +0123
ENTER N-2, 4 DIGITS
-1                   Value of N-2 keyed in
N-2 = -0001

*BREAK* 1.3
?
```

Now we are at the STOP RUN statement. To verify that N-3 did get the proper value from the MOVE statement,

```
?D ('Q+2'+362)/2,2,H
DB+171    0000 1D00
```

Since N-3 is a COMP-3 item, the H (for Hexadecimal) representation was displayed. The value of N-3 is shown as 00001D (The D being the representation of minus).

Using the same information sources (the MAP, VERBMAP, and PMAP), values of data items may also be changed in the middle of executing a program.