

North American Response Centers

HP 3000 APPLICATION NOTE #7

COBOL II / 3000 PROGRAMS: TRACING ILLEGAL DATA USING ERROR 710/711 DOCUMENTATION



June 1, 1986
Document P/N 5958-5824/2623

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This document contains proprietary information which is protected by copyright. All rights are reserved. Permission to copy all or part of this document is granted provided that the copies are not made or distributed for direct commercial advantage; that this copyright notice, and the title of the publication and its date appear; and that notice is given that copying is by permission of Hewlett-Packard Company. To copy otherwise, or to republish, requires prior written consent of Hewlett-Packard Company.

Copyright © 1986 by HEWLETT-PACKARD COMPANY

COBOL II / 3000 Programs: Tracing Illegal Data Using ERROR 710/711 Documentation

A number of calls received by the Response Centers involve COBOLII programs encountering COBOL run-time errors 710 (Illegal Decimal Digit) or 711 (Illegal ASCII Digit). This note, a revised version of an article published in *Communicator (Issue #28)*, explains these errors. It will show you how to track down their source using the information in the error messages in conjunction with compiler MAP and VERBS listings, and a Segmenter PMAP listing. Three examples are given to help you to locate such errors whether in a main program, a non-dynamic subprogram, or a dynamic subprogram in a Segmented Library (SL).

WHAT IS ILLEGAL DATA?

Simply stated, illegal data is any data which when operated upon by the object program code does not conform to its defined type.

HOW IS IT DETECTED?

Illegal data is detected by the hardware trap system when executing some type of data conversion, comparison, arithmetic, shift or move instruction which involves decimal or ASCII data. To be considered legal, such data must be a digit between 0 and 9. In some special cases, such as conversion of ASCII data to decimal (using the CVAD instruction), blank characters are also allowed provided they are leading blanks. These operations are implemented by the Extended Instruction Set and Language Extension Instructions which are part of the Decimal Firmware and Microcode hardware for COBOLII. For more detail on these instructions see the *HP 3000 Machine Instruction Set Manual (P/N 30000-90022)*.

WHERE DOES ILLEGAL DATA OCCUR?

Actually illegal data can occur anywhere within the scope of the object program's environment, and can thus appear in an input file, working-storage, or linkage section data item which can itself be a part of a main segmented main program, dynamic/non-dynamic subprogram, segmented dynamic/non-dynamic subprogram and/or user's group/public library.

ERROR 710/711 MESSAGE OUTPUT FORMATS:

There are three output formats that may be produced by 710/711 errors depending upon the type of error and where it occurred. Figures 1, 2, and 3 show a typical message in each of the three formats.

```
*** ERROR 711  ILLEGAL ASCII DIGIT
SOURCE ADDRESS = %001034
SOURCE = '12=4'
STATUS = %060705  P REGISTER = %000112  INSTRUCTION = CVND
```

Figure 1. Message Format 1

```

*** ERROR 710  ILLEGAL DECIMAL DIGIT
SOURCE ADDRESS = %000624
SOURCE = '3F2F3D5+'
STATUS = %060310  P REGISTER = %000212  INSTRUCTION = CVDA

FIXUP, RESTART ATTEMPTED:
NEW SOURCE = '3020305+'

```

Figure 2. Message Format 2

```

*** ERROR 711  ILLEGAL ASCII DIGIT
SOURCE ADDRESS = %000215
SOURCE = '1A4E'
STATUS = %060627  P REGISTER = %000116  INSTRUCTION = CVAD

***  STACK  DISPLAY  ***

          S=001254  DL=177644  Z=003252
Q=001260 P=001202 LCST= P012 STAT=U,1,1,L,0,0,CCG X=000004

Q=001116 P=000115 LCST= G004 STAT=U,1,1,L,0,0,CCL X=000005
Q=000577 P=000126 LCST= 002 STAT=U,1,1,L,0,0,CCG X=000006

FIXUP, RESTART ATTEMPTED:
NEW SOURCE = '114E'

```

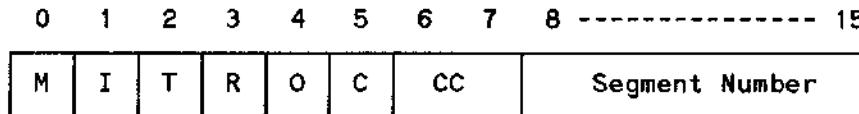
Figure 3. Message Format 3

NOTES PERTINENT TO ALL FORMATS:

SOURCE ADDRESS - This field provides the octal byte address of the source field that contains the illegal data. This address can be used to trace back to a data item via the compiler MAP listing.

SOURCE - This field gives the contents of the illegal data item between the single quotes. Unprintable characters appear as blanks. Usually, by looking at this field you can see which characters are illegal, e.g., non-digits between 0-9 or leading blanks (in some cases).

STATUS - This field holds the value of the hardware Status Register. You can use this to determine the segment number being executed at the time of the error trap. The format of the Status Register is:



Where:

- M = Mode: User/Privileged
- I = External Interrupts Enabled
- T = User Traps Enabled
- R = Right Stack Op Pending
- O = Overflow
- C = Carry
- CC = Condition Code (CCL=1; CCE=2; CCG=0)

SEGMENT NUMBER = Segment number of executing module

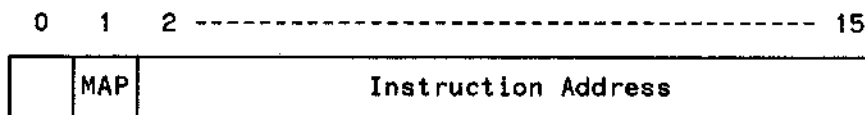
As you can see, the right half of the Status Register is used to hold the Segment Number. However, how these 8 bits are interpreted is dependent on whether your system has mapping firmware installed (which allows expanded tables on MPE versions V/E or later). Mapping firmware is standard on all Series 37 and 6x systems using MPE V/E or later. It is optional on Series 4x systems.

To obtain a program segment number for use with your PMAP listing, you must subtract 1 (if your system has mapping firmware) or %301 (if it does not) from the Status Register's 8-bit value. *Be careful to mask off the Bit 7 from the octal value shown in the STATUS field before subtracting since it may be set depending on the Condition Code.*

If the segment is a library segment, COBOLII provides a stack display which indicates the specific Logical Code Segment (LCST) and Segmented Library (G for Group SL; P for Account SL, in the PUB group; or S for the System SL). A PMAP of that library will be necessary to trace further, as you will see later in the examples.

P REGISTER - This field gives the NEXT address to be executed when the trap occurred. Using this address and the VERBS listing of the segment (obtained using the \$CONTROL VERBS compiler option) you can find the specific COBOL statement which tried to operate on the illegal data item.

To do this, you should subtract one from the address shown in the P REGISTER field to get the actual address of the instruction that caused the trap. In addition, on systems with mapping firmware (see above), the P REGISTER field will include a 'MAP' bit as shown below.



So, on systems with mapping firmware, you must also subtract %040000 to eliminate this MAP bit.

Example:

If the P REGISTER field in the error message shows %071234, subtract 1 giving %071233 (the address of the instruction which caused the trap) and then, if the system has mapping firmware, subtract %040000 to get the correct address of %031233.

INSTRUCTION - This field displays the particular machine instruction which is operating on the illegal data. The instructions which can result in illegal data traps are:

Extended Instruction Set

CVAD - ASCII to decimal conversion
CVDA - Decimal to ASCII conversion
CVDB - Decimal to binary conversion
ADDD - Decimal add operation
SUBD - Decimal subtract operation
CMPD - Decimal compare operation
SLD - Decimal shift left operation
NSLD - Decimal normalizing shift left operation
SRD - Decimal shift right operation
MPYD - Decimal multiply operation

Language Extension Instructions

EDIT Subinstructions:

MA - Move alphabetic operation
MN - Move numeric operation
MNS - Move numeric with zero suppression operation
MFL - Move numeric with floating insertion operation
MDWO - Move digit with overpunch operation

Numeric Conversion Operations:

ALGN - Align numeric
CVND - Convert numeric display
ABSN - Absolute numeric

NOTES PERTINENT TO FORMAT 1 MESSAGES:

This message indicates that the illegal source data could not be fully processed by the instruction and the results of the operation are unpredictable. Data errors of this type must be corrected and the program re-run for valid results.

NOTES PERTINENT TO FORMAT 2 MESSAGES:

This message indicates that the operation has performed a FIXUP of the illegal data and the NEW SOURCE field shows the result of the fixup. The fixup operation occurs in a temporary area so it only affects the target location and not the source data. If you decide the fixup result is not acceptable, the data should be corrected and the program re-run.

A fixup operation is not always performed because of 'bad' data that needs to be corrected. In fact, the program may actually depend on a fixup happening. Such is the case where programs have been written to conform to the COBOL 68 ANSI Standard and deliberately use overpunch characters within numeric fields, for some type of program data control and wish to have the overpunch ignored when the data is processed numerically. These programs are considered in violation of the COBOL 74 ANSI Standard, but

the compiler allows them (with an appropriate error message) so that such COBOL 68 programs may still be compiled using the COBOLII compiler. *It is not recommended that new application designs use such 'features', however, because they do violate the ANSI 74 standard.*

The algorithm used for the fixup operation is:

- 1) If the illegal character is a lower case alphabetic character, it is upshifted before continuing with the algorithm.
- 2) The resulting character is converted as follows:

0 thru 9	→	No conversion
A thru I	→	1 thru 9
J thru R	→	1 thru 9
S thru Z	→	2 thru 9
/	→	1

All other characters are replaced by a zero.

NOTES PERTINENT TO FORMAT 3 MESSAGES:

A stack display has been added to this message, as of library version A.00.05, to facilitate tracing errors which are detected within segmented library procedures. It is only provided for traps which occur within segmented libraries.

The information in the stack display is as follows:

The first line indicates the top of stack and stack boundary when the trap occurred.

The second line indicates the location of Q after the four-word marker has been placed at the top of the stack. Note that Q is four words greater than S. This line reflects the call to the STACKDUMP procedure by C'TRAP.

The third line identifies the procedure which was executing when the illegal data trap occurred and is the key line of information needed when tracing the error source segment, as you will see later.

Any additional lines shown in the stack display can be used to follow a series of segment calls leading to the detection of the illegal data. In the stack display shown in Figure 3, you can see that a call was made from LCST 002 (the user program) to LCST G004 (logical segment in a group library) to LCST P012 (logical segment in a public library), where the COBOL LIBRARY existed.

TRACING EXAMPLES

The following examples show how to locate an illegal data item by tracing through a main program, a non-dynamic subprogram, and a dynamic subprogram residing within a Segmented Library (SL). The partial MAP, VERBS, & PMAP listings are provided for reference purposes in the analysis of each trace.

Example 1: A Simple Trace - A Non-Segmented Main Program

SYMBOL MAP (MAIN PROGRAM)						
LINE#	LVL	SOURCE NAME	BASE DISPL	SIZE	USAGE	
WORKING-STORAGE SECTION						
00008	01	A	Q+2: 000010	000004	DISP	
00009	01	B	Q+2: 000014	000004	DISP	
00010	01	C	Q+2: 000020	000004	COMP-3	
00011	01	D	Q+2: 000024	000004	DISP	
PROCEDURE/VERB MAP						
LINE #	PB-LOC	PROCEDURE NAME/VERB	INTERNAL NAME			
00013	000003	START-IT	STARTIT00'			
00014	000003	MOVE				
00015	000012	MOVE				
00016	000031	STOP				
PROGRAM PMAP						
STARTIT00'	0					
NAME	STT	CODE	ENTRY	SEG		
STARTIT00'	1	0	0			
TERMINATE'	3			?		
QUIT	4			?		
COBTEST	2	34	34			
DEBUG	5			?		
COBOLTRAP	6			?		
SEGMENT LENGTH	144					

Figure 4. Symbol Map, Procedure/Verb Map, & PMAP for Example 1

Given the following error 711 message and the information in Figure 4:

```

*** ERROR 711 ILLEGAL ASCII DIGIT
SOURCE ADDRESS = %000014
SOURCE = '1/5A'
STATUS = %060701 P REGISTER = %000012 INSTRUCTION = CVND
    
```

Analysis and Trace:

1. Since no fixup information is shown in the error message you know immediately the resulting target field is undefined and therefore the illegal data must be corrected and the program re-run.
2. The illegal data is the "/" character shown in the SOURCE field and the length of the field is 4 digits. The STATUS field indicates a segment number of 0; therefore, the problem is within segment 0 of the user program. The P REGISTER indicates the CVND instruction is located at P-relative address 11 in segment 0. The CVND instruction itself indicates the problem occurred when converting a numeric display item.

3. To locate the item:

- a. The PMAP of the program indicates that the P REGISTER address 11 is within the CODE for the relocatable binary module (RBM) STARTIT00' within segment 0.
- b. The VERBS MAP from the source program compilation points to the COBOL MOVE statement, at line # 14, within relocatable binary module (RBM) STARTIT00', in segment 0.
- c. The SYMBOL TABLE MAP from the source program compilation tells you the name of data item by locating the SOURCE ADDRESS within the DISPL field.

In this example, the item is found to be "B" in the working-storage section displacement location 14.

Example 2: A Segmented Main and Segmented Subprograms

SYMBOL MAP (MAIN PROGRAM)						
LINE#	LVL	SOURCE NAME	BASE DISPL	SIZE	USAGE	
WORKING-STORAGE SECTION						
00008	01	A	Q+2: 000146	000004	DISP	
00009	01	B	Q+2: 000152	000007	DISP	
00010	01	C	Q+2: 000161	000004	DISP	
00011	01	D	Q+2: 000165	000004	DISP	
SYMBOL MAP (SUBA & SUBB SUBPROGRAMS)						
LINE#	LVL	SOURCE NAME	BASE DISPL	SIZE	USAGE	
LINKAGE SECTION						
00009	01	A	Q+21 000000	000004	DISP	
00010	01	B	Q+22 000000	000007	DISP	
00011	01	C	Q+23 000000	000004	DISP	
00012	01	D	Q+24 000000	000004	DISP	
PROCEDURE/VERB MAP (SUBA)						
LINE #	PB-LOC	PROCEDURE NAME/VERB	INTERNAL NAME			
00014	000070	SUBA-SEC1A	SUBASEC1A01'			
00015	000070	SUBA-PARA1A				
00016	000070	DISPLAY				
00017	000003	SUBA-SEC2A	SUBASEC2A02'			
00018	000003	SUBA-PARA2A				
00019	000003	DISPLAY				
00020	000003	SUBA-SEC1B	SUBASEC1B01'			
00021	000003	SUBA-PARA1B				
00022	000003	ACCEPT				
00023	000007	COMPUTE				
00024	000003	SUBA-SEC2B	SUBASEC2B02'			
00025	000003	SUBA-PARA2B				
00026	000003	CALL				
00027	000010	EXIT PGM				
PROGRAM PMAP						
SUBA		4				
NAME	STT	CODE	ENTRY	SEG		
SUBASEC1B01'	1	0	0			
C'ACCEPT	4			?		
SUBA	2	47	52			
SUBA'	5			3		
C'DISPLAY	6			?		
C'DISPLAY'FIN	7			?		
C'DISPLAY'INIT	10			?		
SUBA'S	3	47	47			
SEGMENT LENGTH		214				

Figure 5. Symbol Map, Procedure/Verb Map, & PMAP for Example 2

Program Environment: A Main and two Non-Dynamic Subprograms

Given the following error 711 message and information in Figure 5:

```
*** ERROR 711  ILLEGAL ASCII DIGIT
SOURCE ADDRESS = %001426
SOURCE = '3243 1E'
STATUS = %060705  P REGISTER = %000022  INSTRUCTION = CVAD
FIXUP, RESTART ATTEMPTED:
NEW SOURCE = '324301E'
```

Analysis and Trace:

1. Since a fixup has been done (as indicated in the error message), you immediately know the resulting target field has been modified, as shown in the NEW SOURCE field. If, after looking at the results of the fixup, you decide that it is appropriate, the resulting processing can be considered valid; otherwise the data should be corrected and the program re-run.
2. The illegal data is the space or unprintable character shown in the SOURCE field and the length of the field is 7 digits. The STATUS field indicates a segment number of 4; therefore, the problem is in segment 4 of the user's program file. The P REGISTER gives the location of the CVAD instruction as P Relative address 21 and the instruction indicates the problem occurred when trying to convert an ASCII value to decimal.
3. To locate the item:

Tracing the illegal item in this environment is quite similar to the simple main program trace. You need to be aware, however, of the non-dynamic subprograms and whether the data item is within the working-storage area of the module causing the trap message or is being passed, as a parameter, via the linkage section.

- a. The PMAP of the program indicates that the P REGISTER address 21 (22-1) is within the code for the relocatable binary module SUBASEC1B01' within segment 4.
- b. The VERBS MAP of the source program compilation for the SUBA subprogram points to the COBOL statement COMPUTE, within the RBM named SUBASEC1B01'.
- c. The SYMBOL TABLE MAP of the source program compilation tells you the name of the data item by locating the SOURCE ADDRESS within the DISPL field, provided that the item is in that module's working storage area. If the address is not shown on the MAP, it indicates the item has an origin elsewhere and has most probably been passed, as a parameter, to the detecting module. In this example, the MAP for SUBA does not show any working storage area; hence, the bad data is coming from another area.
- d. In order to determine the actual item address from the logical linkage address you must compute it using information from the PMAP and the source listing "DATA AREA IS %nnnnnn WORDS." for each module. An example of this computation is as follows, given the information:

<u>PMAP PGM SEQ</u>	<u>PGM DATA AREA (%WORDS)</u>	<u>W-S MAP ADDRESS (%BYTE)</u>
SUBP B	260 X 2 = 540 %BYTES	1426
SUBP A	246 X 2 = 514 %BYTES	1426 - 540 = 666
MAIN	252 X 2 = 524 %BYTES	666 - 514 = 152

By using the SOURCE ADDRESS = %1426 in the above error message and the PMAP sequence of the main and subprogram modules along with the DATA AREA values for each program unit you can calculate the possible W-S MAP ADDRESS for each program unit for the item you are looking for e.g., address 1426 for SUBP, 666 for SUBA and 152 for MAIN. By looking at the MAP for each program module you should now be able to identify what the item is, and the program module where it was defined by locating the working storage area containing the matching calculated address.

In this example, the item is found within working storage of the MAIN program, as item "B".

Example 3: A Segmented Main and Segmented Library

SYMBOL MAP (MAIN PROGRAM)					
LINE#	LVL	SOURCE NAME	BASE DISPL	SIZE	USAGE
WORKING-STORAGE SECTION					
00008	01	A	Q+2: 000370	000004	DISP
00009	01	B	Q+2: 000374	000004	DISP
00010	01	C	Q+2: 000400	000004	DISP
00011	01	D	Q+2: 000404	000010	DISP
SYMBOL MAP (SUBA & SUBB DYNAMIC SUBPROGRAMS)					
LINE#	LVL	SOURCE NAME	BASE DISPL	SIZE	USAGE
LINKAGE SECTION					
00009	01	A	Q+21 000000	000004	DISP
00010	01	B	Q+22 000000	000004	DISP
00011	01	C	Q+23 000000	000004	DISP
00012	01	D	Q+24 000000	000010	DISP
PROCEDURE/VERB MAP (SUBB)					
LINE #	PB-LOC	PROCEDURE NAME/VERB	INTERNAL NAME		
00014	000102	SUBB-SEC1	SUBBSEC101'		
00015	000102	SUBB-PARA1			
00016	000102	DISPLAY			
00017	000003	SUBB-SEC2	SUBBSEC202'		
00018	000003	SUBB-PARA2			
00019	000003	DISPLAY			
00020	000042	SUBB-PARA2A			
00021	000042	DISPLAY			
00022	000076	ACCEPT			
00023	000003	SUBB-SEC3	SUBBSEC303'		
00024	000003	SUB-PARA3			
00025	000003	COMPUTE			
00026	000056	EXIT PGM			
SEGMENTER SL PMAP					
SUBBSEC303'					
2					
NAME	STT	CODE	ENTRY	SEG	
SUBBSEC303'	1	0	0		
QUIT	3			?	
SUBB'	2	72	72		
SUBB'S	4			?	
SUBBSEC202'	5			?	
SEGMENT LENGTH		350			

Figure 6. Symbol Map, Procedure/Verb Map, & PMAP for Example 3

Program Environment: A Main and two Dynamic Subprograms in a SL

Given the following error 711 message and information in Figure 6:

```
ERROR 711  ILLEGAL ASCII DIGIT
SOURCE ADDRESS = %000374
SOURCE = '123 '
STATUS = %060627  .P REGISTER = %000016    INSTRUCTION = CVAD

***      STACK DISPLAY      ***

          S=001254  DL=177644    Z=003252
Q=001260 P=001202  LCST= P004   STAT=U,1,1,L,0,0,CCG X=000004
Q=001116 P=000015  LCST= G002   STAT=U,1,1,L,0,0,CCL X=000005
Q=000577 P=000007  LCST= G000   STAT=U,1,1,L,0,0,CCG X=000006
Q=000306 P=000013  LCST= 001   STAT=U,1,1,L,0,0,CCG X=000006

FIXUP, RESTART ATTEMPTED:
NEW SOURCE = '123{'
```

Analysis and Trace:

1. Since a fixup has been done (as shown in the error message), you must either accept its results or re-run the program, after correcting the data item.
2. The illegal data is the trailing space or unprintable character and the length of the field is 4 digits. The STATUS field indicates a segment number of 227 which, on a system without mapping firmware, immediately indicates the segment is within some segmented library. The P REGISTER indicates the CVAD instruction is at P Relative address 15 and the instruction itself indicates the problem occurred when trying to convert an ASCII value to decimal.
3. To locate the item in this environment you can use the STACK DISPLAY information to identify the dynamic subprogram where the error was detected and then use similar procedures, as described above, to locate the data item and location source address.
 - a. The top entry in the stack display (LCST=) field indicates the C'TRAP procedure is in segment 4 of the PUBLIC LIBRARY of the account (LCST= P004) and the dynamic subprogram, causing the trap, is in segment 2 of the GROUP LIBRARY of the account (LCST= G002).
 - b. The SL PMAP of the library for segment 2, indicates that the P REGISTER address 15 (16-1) is within the code for the RBM named SUBBSEC303'.
 - c. The VERBS MAP of the source dynamic subprogram points to the COBOL statement COMPUTE, within the RBM SUBBSEC303'.
 - d. The SYMBOL TABLE MAP of the source program compilation indicates the name of the data item by locating the SOURCE ADDRESS within the DISPL field, provided that the item belongs to that module's working storage area. If the address is not locatable on the MAP it indicates the item has an origin elsewhere and has most probably been passed, as a parameter, to the detecting library module.
 - e. In order to determine the actual item address we can refer to the STACK DISPLAY:

The stack trace indicates a main program (LCST= 001) called a procedure in a group SL (LCST= G000) which called another procedure within the group SL (LCST= G002), in which the trap procedure C'TRAP was called from the pub SL (LCST= P004).

Examination of the Q= values (base word address of the next procedures data area, if dynamic) indicates the SOURCE ADDRESS = %000374 (source byte address = word address %176) falls inside the data area for the Main procedure, since %176 is less than %306. This defines the illegal data as a passed parameter.

The actual data item can now be determined by locating the source address within the working storage section, of the main program's source compilation.

In this example, the item is "B" at displacement location %374.