4 CREDIT DEBUGGING PROGRAM

4.1 Introduction

The CREDIT debugging program (CREBUG) is an interactive diagnostic task which runs under the control of the TOSS Monitor, on systems with and without memory management. It runs in parallel with a CREDIT application program being tested.

CREBUG may be used to control the execution of the application program in the following ways:

- Traps may be inserted.
- Verification may be started and stopped.
- The application program may be started or stopped.
- Variables may be examined and modified.
- Trace may be turned on or off.

In addition, CREBUG may be used to:

- Perform calculations (e.g. on addresses).
- Dump Memory.

Readers of Section 4 should be familiar with the following DOS6800 System Software concepts:

- Linkage editor
- Control command
- User library
- TOSS system operation

These concepts are explained in the DOS6800 System Software PRM (M11).

4.2 Running CREBUG

CREBUG is automatically built into the application load module by the Linkage Editor. If CREBUG is not required (e.g. for production versions of the application program) it must be explicitly excluded. This procedure is described below.
CREBUG has a task identity of TB and runs at priority level 55, specified in the configuration file for system loading (SYSLOD). The CREDIT interpreter calls CREBUG immediately before executing each instruction in the application program. The application program status is then checked. If certain conditions specified by the programmer are met, either the program is stopped or specified memory locations are printed.
The programmer communicates with CREBUG via one of the following device configurations:

General printer PTS6321, together with alphanumeric keyboard PTS6234 or 6331.
Console typewriter PTS6862.
Visual display unit PTS6344.

The chosen device configuration must be assigned TOSS file code /21 for input and /31 for output. File code /16 must be assigned to a (line) printer, when tracing is used. This is done during TOSS system generation.

The memory size of the application program is increased when CREBUG is included. For this reason it is advisable to explicitly exclude CREBUG when linkage editing the production version of the program. The following command sequence is recommended:

```
USERID : X
S:INC␣/OBJCT,INT:PROD
S:KPF/O
SCR
S:INC␣/OBJCT,Y
KPF/O
MOV␣MAIN,/S,USER
S:KPF /S,MAIN
S:TRA MAIN,NL
S:TLK␣MIX
S:LKE␣U,M
S:KPF␣/L,MOD NAM
```

where: X is an empty user library and Y is the user library containing the CREDIT Linker object modules.
MAIN is the module containing the data-division.

4.3 CREBUG Input

4.3.1 *General*

Various single character commands may be keyed-in by the programmer to control the testing process. These commands are:

T Set trap
P Proceed from trap
L Loop through trap
Y Remove trap
V Start or stop verification
G Go
H Halt
I Open data item
J Open boolean data item
Q Open Relocation Register
S Open Task Control Area/Condition Register
W Open memory word
/ Open byte
R Turn trace on or off
= Calculate
M Dump memory
KL Lock segment
KU Unlock segment

4.3.2 *CREBUG Modes*

CREBUG operates in one of two modes known as T mode and U mode. In T mode CREBUG is running and the application program is stopped. In U mode both CREBUG and the application program are running. T mode is selected whenever the application program stops.

A stop occurs when:

- a halt (H) command is keyed-in, or
- a trap is encountered in the application program, or
- a verification halt condition is detected.

U mode is selected when one of the following commands is keyed-in:

- proceed from trap (P)
- loop through trap (L)
- go (G)

Commands other than H, P, L and G will not result in a change of mode. The current mode is indicated by the letter T or U printed at the left of each line of output. Immediately after System start CREBUG is in T mode.

4.3.3 *Current Task Identifier/Current Segment number*

In certain commands a task identifier/segment number may be specified. If no task identifier or segment number is specified in these commands, the 'current' task identifier/segment number is assumed. This is the identifier of the task or segment number which was executed when the program halted last.

4.3.4 *Relocation register*

CREBUG maintains 16 relocation registers for the whole application, also when memory management is used. The registers are numbered 0 to F and may be used in commands as indices when referring to memory locations in the segments. The contents of relocation registers may be examined and/or modified using the Q command. The following description illustrates the way in which relocation registers are normally used. The start address of the module, currently being tested, and its segment number are loaded into a relocation register. Commands then refer to locations within this module by quoting the address relative to the start of the module, listed by the translator under the heading LOC, together with the segment number. Loading relocation register 4 with the start address of the module, which is present in segment 1, is done as follows:

4Q/0000 11E.1

A trap in this module and segment is set as : 72, 4T.
For non segmented program in memory always segment number zero must be specified.
When starting an application, relocation register D contains the start address of the program code (P:PIL), but this register is not used when setting traps. Program code is found by using the segment number and for non segmented applications, segment zero is specified. Register F contains minus eight (X'FFF8') which can be used to bias addresses from the linkage editor map.

4.3.4 *Addressing*

The following commands contain references to memory addresses:

I,J
G,P,T,Y
M,V,W,/

Either relative or absolute addresses may be used in these commands.
If an absolute address is used it may be written in either of the following ways:

hexadecimal-number[index]
$decimal-number[index]

The absolute address of a memory word is its displacement from word zero of memory. If the address refers to the start of an array, an index value must be given to specify the word within the array (counting the first word as one).

If relative address is used it may be written in the above manner. But if the relative address refers to a word in a procedure division, it must be modified by the appropriate segment number as follows:

hexadecimal-number[index].segmentnumber
$decimal-number[index].segmentnumber

The relative address of a memory word is its displacement from word zero of the CREDIT module of which it forms a part. Relative addresses are shown in a CREDIT module under the heading LOC (Location counter).

Commands I and J may only refer to addresses in the data division.
Commands G,P,T and Y may only refer to addresses in the procedure division.
Commands M,V, W and / may refer to addresses in either division.

In order to differentiate between the index and the location counter generated by the CREDIT translator, addresses referring to the data division must be prefixed by a # character (e.g. #18W).

Indirect addressing may be used in the following open variable commands:

Open data item (I)
Open task control area/condition register (S)
Open memory word (W)

If an address is indirect it must be prefixed by an asterisk (e.g. *10W). In case the address will point to a memory word which contains the address of the variable to be opened. If the resulting address is odd the next (lower) even address will be used.

4.3.6 *Command syntax*

Commands are keyed-in immediately after the T or U prompt. The prompt is printed at the left of each line by CREBUG.

Commands have the following syntax:

[argl;] [arg2] [:{tid|pha}] [.segnr] com

com is one of the single character commands listed above.
tid is the task identifier of the task to which the command applies. Task identifiers are defined during system loading time (SYSLOD).
segnr is the segment number.
pha is a physical memory area. The current user is default. (Only with MMU systems.) pha may be defined as :S, :X, :Y or :Z. Each value assigns a specific physical memory area.
:S System area, 0--64Kb
:X Extended area 64Kb—128Kb
:Y Extended area 128Kb—192Kb
:Z Extended area 192Kb—256Kb

Arg1 and arg2 are defined as follows:

arg1 :: = { term / terms }

arg2 :: = arg1

term :: = [{ # / * }] address[index] [,relocation-register]

terms :: = term{ + / − }term

index :: = ({ hexadecimal-integer / $decimal-integer } [,hexadecimal-integer]) $decimal-integer

address :: = { hexadecimal-integer / $decimal-integer }

relocation-register :: = 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

The following words, used in the above syntax definition, have the same meaning as that given in Appendix 1.

decimal-integer
hexadecimal-digit
hexadecimal-integer

4.4 CREBUG Output

4.4.1 *Program Stop Message*

Whenever the application program stops CREBUG prints the following message:

c pp=loc,rel.segnr:tid

The significance of these fields is as follows:

c — This code indicates why the program stopped. It may have the following values:
S — System start is complete and the application program may be started.
T — A trap has been encountered.
H — An H (halt) command has been keyed in.
V — A verification halt condition has been detected.
E — The application program is in error.

pp — This is the value of the program pointer. It points to the interpretive instruction which will be obeyed when the program is restarted.

loc — This is the location counter value for the instruction pointed to by the pp. It is the value found under the heading "LOC" for this instruction in the CREDIT module listing. This value, when added to the contents of the relevant relocation register, gives the value of the program pointer. If no relocation register is found for the current segment "loc,rel" will be replaced by the word "SPACE".

rel — This is the relevant relocation register. When added to "LOC" it gives the value of the pp.

segnr — The segment number in which the program is halted. for non segmented programs, segment number zero is printed.

tid — This is the task identifier. It is the identifier of the application task which is currently executed.

4.4.2 *Command Responses*

The response to commands I, J, Q, S, W, / and = is printed immediately after the command on the same line. For example, open relocation register (Q):

1Q/0000 0006.2 (CR)

In this example, the programmer keys in the underlined characters and CREBUG prints the others.

The response to the other commands, appears on the line after the command. For example, dump memory (M):

0004.0M (CR)
208C 0153 1081 3002 831C 0053 2181 4910 4000

If an undefined command is keyed-in, CREBUG responds with a question mark and no action is taken.

If an illegal command is keyed-in, CREBUG responds with "NO!". An example of an illegal command is a go (G) command issued while the application program is actually running.

4.4.3 *Curtail Printout*

If the output from a CREBUG command is unexpectedly long (e.g. a large memory dump) it can be curtailed by depressing any SOP switch or by switching the terminal computer power off and on.
CREBUG uses file code /14 for SOP input, which can be included at system generation time.

4.4.4 *Error messages*

One of the following messages may be output, when the CREDIT debugger detects an error:

ILLEGAL OPERATION
MIXED ARITHMETIC
ILLEGAL TYPE
ODD WORD ADDRESS
MISSING ENTRY
ILLEGAL REQUESTED LENGTH
STACK UNDERFLOW
ARITHMETIC OVERFLOW
STACK OVERFLOW
INDEX OVERFLOW
ILL INDEX TYPE
DEV BY ZERO
EDIT BUFFER OVFL
EDIT PICTURE OVERFLOW
ALLOCATION ERR AT INITIALIZATION
ILLEGAL INSTRUCTION ADDRESS
ILLEGAL FORMAT CODE
NO FIX BUFFER ALLOCATED
FIX BUFFER NOT ALLOWED
ILLEGAL CONTROL CODE
ILLEGAL INDEX VALUE
FORMAT CONDITIONS CHANGED
ILLEGAL PARAMETER
LENGTH ERROR
DATASET BUSY
ILLEGAL DATASET REFERENCE
DISK ERROR NO REENTER POINT

4.4.5 *Trap Commands*

In order to examine the state of a program at a certain stage in the execution, the programmer may insert traps. A trap is an address specified to CREBUG at which the program is stopped. A stop message (described above) is printed by CREBUG and the system then waits for a further command from the programmer.

The instruction on which the trap is set is not executed until the program is re-started. However, it is possible to loop a number of times through a certain trap before the program is stopped.

A maximum of 16 traps at a time may be set.

4.4.6 *Open variable commands*

These commands are:

- Open data item (I)
- Open boolean data item (J)
- Open relocation register (Q)
- Open task control area/condition register (S)
- Open memory word (W)
- Open byte (/)

An open and modify command has the following form:

...../xxxx[arg1[: tid|pha]] [.segnr] com
xxxx is the contents which will be replaced optionally by arg1.

The function of these commands is to print the contents of the specified variable, and if requested, modify those contents. This is done in the following manner:

- the programmer specifies the variable and keys in one of the above commands. This is done according to the normal command syntax rules.
- CREBUG responds by printing the contents of the variable (/xxxx). The variable is now "open", i.e. it can be modified.
- If necessary the programmer can now key-in a new value for the variable. If the variable is not to be modified, the programmer simply keys in an LF or CR character and the variable is "closed".

Where possible the above dialogue is all printed on a single line, for example:

EQ/0000 2942,2 (CR)

In this example the programmer keys in the underlined characters and CREBUG prints the others.
"E" indicates that relocation register E is to be opened (command Q). CREBUG responds by printing out "/" followed by the contents of relocation register E "0000". The programmer then keys in the new value "2942" followed by the segment number and a carriage return character. The CR indicates that the variable is to be closed.

If a line feed character (LF) had been keyed-in instead of CR in the above example, it would indicate that the current variable is to be closed and that the variable at the next (higher) address is to be opened.
If an @ character has been keyed-in, it would indicate that the current variable is to be closed and used as the address of the next variable to be opened (i.e. indirect addressing). This variable will be automatically opened. The @ character is meaningful only in commands I, S and W.

Any other non-hexadecimal character in this position would result in the current variable being closed without modification and without opening the next variable.

If the programmer does not wish to modify the contents of the opened variable, it can be closed without modification by simply keying-in a CR or LF character immediately after the current variable contents.

The possible values of the terminating character are summarised as follows:

- CR – close current variable
- LF – close current variable and open next variable
- @ – close current variable and open indirectly addressed variable.

4.4.7 *Command reference*

G *Go* **G**

Syntax: (i) arg2 [.segnr] G
(ii) G

Description: A go command enables the programmer to start an application program which has stopped.

(i) Start the program at address "arg2". Current segment is default.
(ii) Start the program at the address pointed to by the program pointer.

Example:

0005G	start at location 5 in segment 0
0007.3G	start at location 7 in segment 3
0004,2.4G	start at location 4 modified with relocation register 2, in segment 4.

H *Halt* **H**

Syntax: (i) :tid H
(ii) H

Description: (i) Halt the task specified by "tid".
(ii) Halt the program (may be any task).

Example: :BOH halt for task "BO"
H halt program

I	*Open data Item*	I

Syntax: arg2 [:tid] I

Description: This command can be used with binary, decimal or string data items. If the item is binary it will be opened. If it is decimal or string, it will be printed but not opened. The current "tid" is default. To open string or decimal data items, the "IX" value from the listing, must be preceded by a # sign, and followed by "W".

Example:

37(3)I/0005	Open element 3 of the array, referenced in the listing by 37.
50I/0002	Open binary data item referenced by 50, for current task.
#31W/2020	Open string data item referenced by 31
#42W/B137	Open decimal data item referenced by 42
50I:B0I/0521	Open binary data item referenced by 50, for task B0.

J *Open boolean data item* J

Syntax: arg2 [:tid] J

Description: This command is used to open a boolean data item. The current "tid" is default.

Example: 10J/0 Open boolean data item referenced by 10 for the current task.
10J:B0J Open boolean data item referenced by 10 for task B0.

KL *Lock segment* **KL**

Syntax: arg2 KL

Description: The segment specified in arg2 becomes main memory resident. It stays resident until an unlock segment (KU) command is executed. The command KL can be used before making a patch in a segment.

Example: 2KL Segment 2 becomes main memory resident.

KU *Unlock segment* **KU**

Syntax: arg2KU

Description: The segment with the number specified in arg2 will no longer be main memory resident.

Example: 2KU Segment 2 released from main memory.

L — *Loop through trap* — L

Syntax: (i) arg2 L
(ii) L

Description: (i) Loop "arg2" times through the current trap.
(ii) Loop once through the current trap.

Note: This command can only be given when the application program is stopped at a trap.

Example: 5L loop 5 times through current trap
L loop once through current trap

M *Dump memory* M .

Syntax:
(i) arg1; arg2 [:{tid|pha}] [.segnr] M
(ii) arg2 [:{tid|pha}] [.segnr] M
(iii) M

Description: The contents of selected memory words can be printed. When one or more lines have been printed CR (carriage return) or LF (linefeed) may be given. If CR is keyed-in the command is terminated. If LF is keyed-in the next eight memory words will be printed.

(i) Dump memory words from address "arg1" to address "arg2" inclusive. Current "tid" is default.
(ii) Dump eight memory words from address "arg2". Current "tid" is default.
(iii) Dump eight memory words from the address of the last word dumped.

Examples:

0004; 0018 M	Dump memory words 4 to 18 of the user area of the current task
0004; 0018:B0M	Dump memory words 4 to 18 of the user area of task B0
0004.5; 0018.5M	Dump memory words 4 to 18 of segment 5
0004; 0018::$M	Dump memory words 4 to 18 in the System area.

P *Proceed from trap* **P**

Syntax:
(i) arg1; arg2 [:tid] [.segnr] P
(ii) arg2 [:tid] [.segnr] P
(iii) P

Description:
(i) Set a new trap at address "arg1" and set the loop counter to "arg2". Remove current trap and proceed with program execution. Current segment is default.
(ii) Set a trap at address "arg2" and set the loop counter to zero. Remove current trap and proceed with program execution. Current segment is default.
(iii) Remove the current trap and proceed with program execution.

Note: This command can only be given when the application program is stopped at a trap.

Q — *Open relocation register* — Q

Syntax: (i) arg2Q
(ii) Q

Description: (i) Open relocation register "arg2". Only the last four bits are significant. 16 relocation registers are available, numbered from hexadecimal '0' to 'F'.
(ii) Print all relocation registers.

Example: 5Q/0000 ␣ 3D.0 Relocation register 5 is loaded with the start address of MOD2 (3D, from the load map), modified with the start address of segment 0.

R

Trace

R

Syntax:
(i) arg2R
(ii) R

Description: This command is used to trace the execution sequence of some specific instructions or all. The trace mode is defined in arg2 and can be:

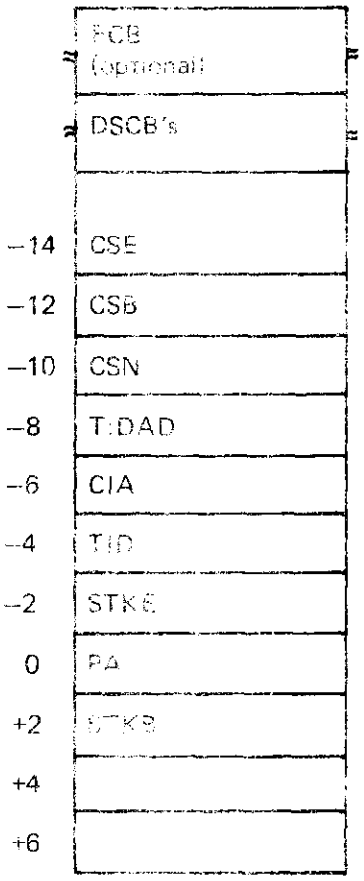
0 – switch trace off
1 – trace branch instructions
2 – trace all instructions
3 – trace arithmetic instructions.

(i) Set trace mode to "arg2". This affects all tasks.
(ii) Stop tracing.

S — *Open task control area / Condition register* — **S**

Syntax: (i) arg2 [:tid] S
(ii) [:tid] S

Description: This command can be used to open a word in the task control area (TCA) or to open a condition register.
The current "tid" is default.
The addresses of the items in the TCA are shown below.



Format Control Block (33 words)

Data set Control Blocks.
10 words per data set. The layout corresponds to an ECB.

Current Segment End.

Current Segment Base.

Current Segment Number.

Pointer to task descriptor table.

Current Instruction Address.

Task identifier.

Stack end pointer.

Auxiliary Stack pointer

Stack Base pointer

Descriptor table address.

Work block address.

(i) Word "arg2" (hexadecimal) within the TCA is opened.
Current "tid" is default.

(ii) The condition register belonging to task identifier "tid" is opened.
Current "tid" is default.

Example:
−4,A0/−4:S2 Open word −4 of TCA of task A0.
S/1 Open condition register of current task
A0/S:2 Open condition register of task A0.

T

Set trap

T

Syntax: (i) arg1; arg2 [:tid] [.segnr] T
(ii) arg2 [:tid] [.segnr] T

Description: (i) Set a trap at address "arg1" and set the loop counter to "arg2".
(ii) Set a trap at address "arg2" and set the loop counter to zero.

Note: If a "tid" is specified the trap affects only that task. If no "tid" is specified, the trap affects all tasks. Current segment is default.

V *Verify* V

Syntax:
(i) arg1, arg2 [:tid] V xx
(ii) arg2 [:tid] V
(iii) V

Description: The verify command instructs CREBUG to continuously monitor the contents of a specific memory word, referenced by arg1, during program execution. When the memory word assumes a specific relationship (equal to, greater etc.) to a given value, in arg2, the program is stopped.
A stop message is printed by CREBUG and the system waits for further command from the programmer. Verification is only permitted on complete 16 bit memory words.

(i) "XX" specifies the relationship to be tested for, and can have the following values:
EQ meaning: (arg1) = arg2
GR meaning: (arg1) > arg2
LT meaning: (arg1) < arg2
NE meaning: (arg1) ≠ arg2
NG meaning: (arg1) ≯ arg2
NL meaning: (arg1) ≮ arg2

Stop program when the contents of "arg1" and the value in "arg2" meets the specified relationship in XX.
Current ' tid" is default.
(ii) Stop verification at "arg2",
(iii) Stop all verifications.

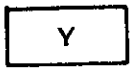
W

Open Memory Word

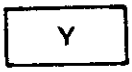
W

Syntax:
(i) arg2 [:{tid | pha}] [.segnr] W
(ii) W

Description:
(i) Open memory word at address "arg2". Current "tid" is default.
(ii) Reopen the memory word specified in the preceding W command.



Remove trap



Syntax: (i) arg2Y
(ii) Y

Description: (i) Remove the trap at address "arg2".
(ii) Remove all traps.

/ *Open Byte* /

Syntax: (i) arg2 [:tid] [.segnr] /
(ii) /

Description: (i) Open byte at address "arg2".
Current segment is default.
(ii) Re-open the byte specified in the preceding / command.

=	*Calculate*	=

Syntax: arg2=

Description: This command is used to calculate (add/subtract) with hexadecimal values.

Example: