68Ø9 DEBUGGER

USER'S MANUAL

1st Printing

NOTICE

This manual describes IDBØ9 Version 1.Ø. Software Dynamics has carefully checked the information given in this manual, and it is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Software Dynamics reserves the right to change the specifications without notice.

TABLE OF CONTENTS

IDBØ9 68Ø9 DEBUGGER USER'S MANUAL

INTRODUCTION

IDB is a small memory, stand-alone debugger for 68Ø9 microprocessor systems. It is appropriate for debugging assembly language object programs.

IDB allows the programmer to: display and change memory, to display and change registers, and to set breakpoints.

This manual describes IDB Version 1.Ø.

NOTATION:

In the examples included in this manual, underlined characters are typed by the operator. Comments to the right do not appear as output of the debugger; all other printed data is typical debugger output. Many of the examples use previous examples to set up a known situation.

OPERATION

The debugger gains control automatically in some ROM-based systems. Usually, control is initially passed to the debugger by an SDOS DEBUG command or ^D typed to SDOS (see SDOS Manual).

The programmer interacts with IDB via commands given at the keyboard. IDB gives has no prompt but does dump the registers on startup. IDB checks command input character by character. If an entry is incorrect, it is diagnosed immediately by a print-out of "??" followed by a carriage-return. When a command error occurs any opened location is closed. If a valid command is entered, IDB executes the command and accepts another command.

All IDB commands and hexadecimal numbers can be entered in either upper or lower case. In this manual only uppercase commands are shown. A small letter immediately to the left of a command represents a numeric value entered by the operator.

COMMAND FORMAT

All commmands to IDB fit one of the following forms:

C	(No Parameter)
nC	(Single Parameter)
n;C	(Single Parameter)

where n is a hex number up to ten digits depending upon the command. <CR> is a carriage-return, <LF> is a line-feed and C is a command character (letter, punctuation mark, <CR>, or <LF>). ";" is a semicolon and "," is a comma.

VALUES ENTERED INTO THE DEBUGGER

IDB accepts four formats for numbers:

Hex numbers, a string of hex letters or digits:
ØA BC9 22 BD3FA9

. (Period), meaning the address of the last opened memory location, whether it is open now or not. This is referred to as the open location marker.

* (asterisk), meaning the value that is displayed as the P register contents on a register dump (location of next instruction to execute).

'c (single quote, followed by any character), meaning "the ASCII value of the character c". 'A is equivalent to typing in 41 (hex); likewise, 'b = hex 62.

SIGNIFICANCE

Numbers entered into IDB have significance (size in bytes) based on the number of digits keyed in. This significance is used by commands which store into memory.

1 or 2 digits gives 1 byte significance
3 or 4 digits gives 2 byte significance
5 or 6 digits gives 3 byte significance
7 or 8 digits gives 4 byte significance
9 or 1Ø digits gives 5 byte significance
. (Period) has 2 bytes of significance
* (Asterisk) has 2 bytes of significance
'c has 1 byte of significance

IDB COMMANDS

IDB commands fall into the following categories:

Examine and Modify Memory
Set Register
Breakpoints
Execute

Examine and Modify Commands

The examine and modify commands are used to display and/or change memory locations and registers.

COMMAND	OPERATION
n/	Open Location n
<LF>	Display Next
n<LF>	Deposit and Display Next
<CR>	Close This Location
n<CR>	Deposit and Close Location
?	Display Registers

The n/ command is used to open location n and display its content. "Opening a location" means to make it available for examination and/or modification.

The <LF> (line-feed) command is used to advance the open location address and display the contents of the new location. The open location address is bumped by one, and the next byte is displayed. <LF> is only valid when a location is open.

The n<LF> command is used to deposit from one to five bytes. The open location address is bumped by the significance of n. n<LF> is only valid when a location is open.

The <CR> (carriage-return) command is used to close the currently open location. The open location address is not advanced. <CR> is a no-op when a location is not open.

The n<CR> command is used to deposit from one to five bytes into the open location. The open location address is not advanced and the location is closed. n<CR> is only valid when a location is open.

The ? Command is used to display the registers. This display is referred to as a register dump elsewhere in this manual. In a register dump, the contents of the registers follow the letter naming that register. "Z" refers to the page register.

Examples:

```
1ØØ/FF 45<CR>          DEPOSIT 45 OVER FF IN LOCATION 1ØØ
./ 45 <LF>             EXAMINE LOCATION 1ØØ, EXAMINE NEXT
./ 46 <LF>             OPEN THE LAST LOCATION; EXAMINE NEXT.
Ø1Ø2/ 43 BD <LF>       CHANGE VALUE AND EXAMINE NEXT.
*/ 7E3608 39<CR>       FIX INSTRUCTION AT P COUNTER
?                      DUMP REGISTERS
C=CØ A=Ø1 B=FE Z=ØØ X=3Ø31
Y=ØØØØ U=C27D P=3ØØ5 S=4Ø73
```

SETTING REGISTERS

The following commands are used to change the contents of a specific register by name.

COMMAND	OPERATION
n;A	Set A Register to n
n;B	Set B Register to n
n;C	Set C Register to n
n;X	Set X Register to n
n;S	Set S Register to n
n;P	Set P Register to n
n;U	Set U Register to n
n;Y	Set Y Register to n
n;D	Set D Register to n
n;Z	Set Z register to n

The n;A n;B n;C n;Z commands set registers A B C Z respectively to the rightmost byte of n.

The n;X n;S n;P n;U n;Y and n;D commands set registers X S P U Y and D respectively to the rightmost two bytes of n. If a one byte value is given, a leading zero byte is assumed.

When the stack pointer is set, IDB assumes that the value given, minus 12, points to a interrupt context block (i.e., n-12 points to a condition code byte). The contents of this context block are used as the values of the registers.

When IDB starts up, it allocates a 12 byte stack for the user's context block.

Examples:

```
1;A              SET THE A REGISTER TO Ø1
FE;B             SET THE B REGISTER TO FE
CØ;C             SET THE C REGISTER TO CØ
?               SHOW REGISTERS
C=CØ A=Ø1 B=FE Z=ØØ X=3Ø31
Y=ØØØØ U=C27D P=3ØØ5 S=4Ø73
1234;A           SET THE A REGISTER TO 34
123456;B         SET THE B REGISTER TO 56
?                SHOW REGISTERS
C=CØ A=34 B=56 Z=ØØ X=3Ø31
Y=ØØØØ U=ØØØØ P=3ØØ5 S=4Ø73
1;X              SET X TO ØØØ1
?                SHOW REGISTERS
C=CØ A=34 B=56 Z=ØØ X=ØØØ1
Y=ØØØØ U=ØØØØ P=3ØØ5 S=4Ø73
1234;X           SET X TO 1234
?                SHOW REGISTERS
C=CØ A=34 B=56 Z=ØØ X=1234
Y=ØØØØ U=ØØØØ P=3ØØ5 S=4Ø73
FE;P             SET P REGISTER TO ØØFE
FE/ ØØ           LOOK AT LOCATION FE
FE/ ØØ 2245<CR> MAKE IT AN INSTRUCTION
?               SHOW REGISTERS
C=CØ A=34 B=56 Z=ØD X=1234
Y=ØØØØ U=ØØØØ P=ØØFE S=4Ø73
```

Caution: setting the stack pointer (S register) causes the remaining registers to take on arbitrary new values according to their positions in the context block pointed to by the new value of the S register!

```
FE;S             SET THE STACK POINTER TO ØØFE
?                SHOW REGISTERS
C=F3 A=Ø4 B=Ø4 Z=ØD X=9762
Y=ØØØØ U=C27D P=ØØ67 S=ØØFE
```


BREAKPOINT COMMAND

"Breakpoints" are used to stop a program at a certain place so that the state of the machine can be examined. The programmer places a breakpoint in his program where he wishes; then he tells IDB to run his program (see Execution Commands). When the program hits a breakpoint, control is passed to IDB, which results in a register dump. The programmer then can examine or change memory, place a new breakpoint, start his program again or continue execution from where it left off.

COMMAND	OPERATION
n!	Set Unconditional Breakpoint on Address n
Ø!	Clear Breakpoint

An IDB breakpoint consists of a "JSR" instruction. The JSR instruction is "planted" at the breakpoint location to regain control when encountered.

Since a JSR takes three bytes, no breakpoint may be set within two bytes of another breakpoint. Setting breakpoints in ROM doesn't work. This may not be obvious since the breakpoints can't be seen in the user's code while IDB is in the command input mode. The breakpoint JSR instruction is not "planted" in the user code. Execution is requested (see Execution Commands), For example, if a breakpoint were set at location 2ØØ (by entering "2ØØ!"), examination of location 2ØØ will still show the original user code rather than IDB'S breakpoint JSR instruction.

When the user's program is executing, and it encounters a breakpoint JSR, then the breakpoint is "hit"; the breakpoint JSR is removed,the original user code is restored and a register dump is displayed. IDB then enters command input mode.

When a breakpoint hits, the next instruction to execute is the one at the breakpoint address (the instruction at the breakpoint has not yet been executed). Entering the G command on the console after hitting the breakpoint will result in an immediate breakpoint "hit" without having executed any instructions because the P register still points to the breakpoint location. The convenient way to continue from a breakpoint is to use the set new breakpoint and proceed (nX) command.

Using a JSR instruction for a breakpoint causes some peculiarities.

Example:

```
        BRA     L1
        .
        .
        .
LØ      BEQ     L3
L1      LDAA    #5
```

Breakpointing on LØ is hazardous during realtime execution if the "BRA L1" is executed. The reason for this is that the breakpoint JSR is planted at LØ and it will take up the first byte of L1, so that during realtime execution, L1 does not contain a "LDAA #5" instruction!

Example:

```
        BSR     XYZ
```

Breakpointing the BSR is fatal when the RTS in subroutine XYZ is executed because the third byte of the breakpoint JSR covers the first byte of the instruction following the BSR. When the called subroutine returns, the instruction will most likely be invalid, and at the very least will cause unpredictable results.

The set breakpoint command (n!) is used to set the breakpoint on a particular location.

Examples:

```
100!                    SET BREAK AT LOCATION 100
4852!                   SET BREAK AT 4852
```


EXECUTION COMMANDS

The execution commands are used for proceeding from a breakpoint, or restarting a program.

COMMAND	OPERATION
G	Start Realtime Execution (GO)
nG	Set P Register and GO
nX	Set Breakpoint at n and proceed

The G command is used to start realtime execution from the current context block (the context block consists of all the registers displayed by the "?" command). All of the registers are loaded including the S register and control is transferred to the user program. Instruction execution begins with the instruction pointed to by the P register, and execution continues in real time. If a breakpoint JSR is encountered, IDB will regain control and give a register dump and enter command mode.

The nG command sets the P register in the context block to n, then does a G command. If the significance of n is one, a leading zero byte is assumed.

The nX command is used to continue realtime execution from a breakpoint. It also sets a new breakpoint at location n.

COMMAND SUMMARY

n/	Open Location n and Display
<LF>	Display Next
n<LF>	Deposit and Display Next
<CR>	Close This Location
n<CR>	Deposit and Close Location
?	Display Registers, Current Instruction, and Last Opened Location
n;A	Set A Register to n
n;B	Set B Register to n
n;C	Set C Register to n
n;D	Set D Register to n
n;X	Set X Register to n
n;U	Set U Register to n
n;Y	Set Y Register to n
n;Z	Set Z Register to n
n;S	Set S Register to n
n;P	Set P Register to n
n!	Set Breakpoint on Location n
G	Start Realtime Execution (GO)
nG	Set P Register and GO
nX	Set new breakpoint and GO to location in P register