

# Reference Guide For D-Bug12 Version 2.1.x

A Debug Monitor  
For  
The MC68HC912B32 Microcontroller

Written By  
Gordon Doughman  
Software Specialist

## Introduction

D-Bug12 has undergone considerable revision since the introduction of version 1.0.x for the M68HC812A4 EVB. Version 2.x.x of D-Bug12 was developed for the MC68HC912B32 EVB to provide an economical yet powerful debugging tool that can be utilized to develop M68HC12 applications or simply to evaluate the M68HC12 architecture. Most of the improvements made to D-Bug12 version 2.x.x since its original release are related to D-Bug12's operation in 'POD' mode. In this operating mode, D-Bug12 communicates with a target M68HC12 microcontroller through the Single-Wire Background Debug Mode (BDM) Interface to allow true emulation of an application in the target microcontroller's operating environment.

The BDM firmware communications primitives utilized by version 2.0.x of D-Bug12 require the M68HC912B32 on the EVB to operate at the same clock speed as the target M68HC12. This restriction requires developers utilizing a target clock other than 8.0 MHz (16.0 MHz crystal) to replace the crystal on the EVB and modify D-Bug12 for operation at the alternate operating frequency. While these modifications are not difficult, it requires the modifications to be performed each time the EVB is connected to a target that operates at a different clock frequency. Among other features described in this document, D-Bug12 version 2.1.x utilizes new firmware communications primitives that allow the target M68HC12 microcontroller to operate with a crystal frequency between 32.768 kHz and the crystal frequency of the MC68HC912B32 EVB (normally 16.0 MHz).

The remainder of this document describes the enhanced features of D-Bug12 version 2.1.x

## New Features

- Supports BDM communication for target crystal frequencies less than or equal to the EVB crystal frequency down to 32 kHz.
- Supports on-chip hardware breakpoint module providing two program only hardware breakpoints.
- RESET command now loads program counter with the contents of the target MCU's reset vector (\$FFFE and \$FFFF).

- Improved register display format - disassembles code at current program counter
- Improved FLoad, FBulk, Load, Verify and Device commands to support M68HC12 devices with greater than 64K bytes of memory.
- FBulk command supports Motorola's newly specified erase pulse timing of 10 mS.
- Improved target memory read and write routines to support aligned word access of 16-bit wide memory and peripherals.
- D-Bug12 utilizes the XIRQ interrupt input for a program abort function
- Command line buffer length was reduced to 50 characters from 80 characters.
- Maximum number of command line arguments increased from 10 to 11.
- Maximum S-Record code/data field length was reduced to 32 bytes from 64 bytes.
- Support for MC68HC912BC32 CAN interrupt vectors when operating in EVB mode.

### Upgrading to Version 2.1.x

Previous versions of D-Bug12 allowed the D-Bug12 firmware to be upgraded using either the serial bootloader, described in Appendix E of the M68EVB912B32 Evaluation Board Users Manual or by erasing and reprogramming the entire 32K bytes of on-chip Flash using a BDM programming tool such as Motorola's SDI12 or a second MC68HC912B32 EVB. The serial bootloader method allowed the main portion of D-Bug12, residing in the lower 30K bytes of the Flash memory, to be completely replaced while retaining the bootloader residing in the 2K byte erase protected boot block. Because of the additional features contained in version 2.1.x, some of the 2K byte boot block that was previously unused by the bootloader is required by D-Bug12 for the BDM firmware communications primitives. **Therefore, upgrading to version 2.1.x will require the use of a BDM programming tool.**

To upgrade to version 2.1.x of D-Bug12, follow these steps:

- 1.) Connect the BDM programming tool to the target EVB as described in the programming tool's documentation.
- 2.) Erase the entire 32K byte Flash EEPROM array.
- 3.) Program the Flash array with the contents of the supplied S-Record file containing D-Bug12 version 2.1.x.
- 4.) After successfully programming the Flash with the new version of D-Bug12, disconnect the programming hardware and configure jumpers W3 (0) and W4 (0) for the EVB operating mode.
- 5.) Connect the EVB to a suitable power source and terminal as described in Chapter 2 of the M68EVB912B32 Evaluation Board Users Manual.

- 6.) Applying power to the EVB should produce the following response on the terminal:

```
D-Bug12 v2.1.0
Copyright 1996 - 1998 Motorola Semiconductor
For Commands type "Help"
```

>

If the prompt does not appear, check all connections and verify that the Flash memory was properly programmed with the new version of D-Bug12.

- 7.) Enter the Bulk command on the command line followed by a carriage return.

---

---

**Note:** Because special configuration information is stored in the on-chip EEPROM, failure to completely erase the on-chip EEPROM before operating the EVB in POD mode for the first time after the upgrade will result in the incorrect operation of many of D-Bug12's commands and/or features.

---

---

- 8.) The EVB may now be reconfigured for POD mode operation by reconfiguring jumpers W3 and W4 as described in Chapter 2 of the M68EVB912B32 Evaluation Board Users Manual.

---

---

**Note:** There will be a several second pause before a prompt appears the first time the EVB is powered-up or reset in POD mode after the D-Bug12 firmware upgrade. This pause is a result of D-Bug12 updating the on-chip EEPROM with the required configuration information. If the pause lasts longer than several seconds, follow the trouble shooting suggestions in Chapters 2 and 3 of the M68EVB912B32 Evaluation Board Users Manual.

---

---

### POD Mode Startup Procedure

On power-up or reset D-Bug12 attempts to establish communications with a target system connected to the BDM OUT (W11) connector. Initially, communications is attempted without resetting the target system. This feature allows the POD EVB to be 'hot connected' to a running system without disturbing the target microcontroller. However, if communications cannot be established, the following menu of choices is displayed:

```
Can't Communicate With The Target Processor
```

- ```
1.) Set Target Speed (16000 KHz)
2.) Reset Target
3.) Reattempt Communication
?
```

Entering the number '1', '2' or '3' from the keyboard allows the developer to configure D-Bug12 for an alternate target frequency, reset the target and attempt to establish communications or attempt to establish communications without resetting the target M68HC12. Entering a character other than

the choices provided will result in the target being reset and an attempt to establish communications. The frequency displayed in parenthesis is the current setting for the target crystal frequency.

Entering a '1' to set the target speed will cause the following prompt to be displayed:

```
Enter Target Crystal Frequency (KHz):
```

Note that the entered number must be the target's crystal frequency and **not** the target's E-clock frequency. The entered frequency must be in kilohertz and not hertz. Valid target frequencies range from a low of 32 KHz to a high equal to the crystal frequency of the EVB being used as the POD. Numbers outside this range will result in an error message being displayed and cause the menu of choices to be redisplayed. Each time a valid target crystal frequency is entered, the new value is saved in the EVB's on-chip EEPROM. The saved value is used to initiate communications each time the EVB is powered-up or connected to a new target system.

---

---

**Note:** Because of the timing tolerance inherent in the BDM communications protocol and the implementation of the BDM firmware communications primitives, an exact value for the target crystal need not be specified. However, the entered value should be as accurate as possible. For very low frequencies, such as a 32.768 KHz crystal, a value of 32 or 33 will result in proper communication. In reality, the BDM firmware communications primitives will communicate properly with the target microcontroller even if the entered crystal frequency is as much as  $\pm 20\%$  different from the actual target crystal frequency.

---

---

After a valid target crystal frequency has been entered, D-Bug12 will attempt to establish communications with the target processor **without** resetting the target. If the menu of choices is redisplayed, communication could not be established. If communication cannot be established after several attempts, check for the following possible problems:

- The EVB's BDM OUT connector (W11) must be properly connected to the target systems BDM connector. If the target system is another MC68HC912B32 EVB, make sure that the POD EVB's BDM OUT connector (W11) is connected to the target EVB's BDM IN connector (W9).
- Check for the proper orientation of the BDM cable with the BDM connectors on both the EVB and the target.
- If the target system is not another EVB, verify that its BDM connector is wired to the proper MCU signals on each pin.
- If the target MCU does not have any firmware to execute, the CPU will most likely "run away", possibly executing a STOP instruction, preventing BDM communications with the target MCU. Thus it is strongly recommended that if a target system does not have firmware to execute at power-up or reset, that the target MCU be configured to operate in Special Single Chip mode.

## POD EVB Baud Rates and Target Memory Downloads

The operating speed of the target MCU and hence the communication speed of the BDM interface directly affect the maximum terminal baud rate that can be used to download or verify S-Records in

either the on-chip or externally connected RAM. Earlier versions of D-Bug12 would support download baud rates as high as 38,400 with a target crystal frequency of 16.0 MHz. However, because of the additional software overhead of the variable speed BDM primitives, version 2.1.x is limited to a download baud rate of 19,200 with a target crystal frequency of 16.0 MHz. The maximum usable baud rate for download into target RAM will be directly proportional to the target crystal frequency.

The reason for this restriction is the lack of hardware or software handshaking supported by the basic SCI routines used by D-Bug12. The FLOAD command does not have the baud rate restriction because of the special software handshake protocol used to control the flow of S-Record data from the host computer.

### **Hardware Breakpoint Support**

D-Bug12 versions 2.0.1 and earlier supported 10 software breakpoints that allowed developers to halt program execution on instruction opcode boundaries. Unfortunately, the placement of software breakpoints are restricted to programs that reside in alterable memory. This restriction is not a problem for small programs placed in the on-chip RAM or EEPROM when operating the EVB in EVB mode. However, when the EVB is utilized in POD mode to test and debug code in a target M68HC12's Flash, software breakpoints cannot be used.

To facilitate debugging in an M68HC12's on-chip flash, many M68HC12 family members include an on-chip hardware breakpoint module. D-Bug12 versions 2.0.2 and later support the hardware breakpoint module by providing two hardware breakpoints in place of the 10 software breakpoints. Even though the breakpoint module is capable of providing data access breakpoints, D-Bug12 only supports the module's dual address program breakpoint operating mode. In this operating mode, the hardware breakpoints utilize the CPU12's instruction tagging mechanism. Much like software breakpoints, this restricts the placement of the hardware breakpoints to the address of an instruction opcode.

The hardware breakpoints may be used in both POD and EVB operating modes. Utilizing the hardware breakpoints in EVB mode is especially important when developing code in the on-chip EEPROM. The hardware breakpoints prevent D-Bug12 from erasing and reprogramming the EEPROM each time an instruction is traced or breakpoints are placed in memory.

Each time the EVB is powered up or reset, D-Bug12 defaults to using the 10 software breakpoints. To utilize the hardware breakpoints, the USEHBR command must be entered on the command line.

### **Improved Register Display Format**

Any time the CPU12 register contents are displayed, memory at the location pointed to by the current value of the program counter is displayed in assembly source format. This change affects not only the Register Display (RD) command but each of the register setting commands.

### **Additional Flash Programming Support**

The FLOAD, FBULK, VERIFY and DEVICE commands have been enhanced to support on-chip Flash programming for two additional M68HC12 family members - the MC68HC912D60 and the MC68HC912DA/DG128. In addition, the LOAD command, which supports loading of S-Records into RAM, supports the extended memory space of the MC68HC812A4. For details of the S-Record format required for parts supporting greater than 64K bytes of program memory, refer to the LOAD, FLOAD and VERIFY commands.

---

---

**Note:** Please refer to the section titled “FLOAD, LOAD and VERIFY S-Record Format” at the end of this document for a complete description of the S-Record Format utilized by these commands for M68HC12 devices supporting more than 64K bytes of memory.

---

---

### **FBULK Erase Pulse Time Reduced**

Motorola has recently made a change to the erase pulse timing specification,  $t_{EPULSE}$ , reducing it from a nominal value of 100 mS to a nominal value of 10 mS. The FLOAD command has been modified to reflect this change.

### **16-bit Aligned Target Memory Access Supported**

All versions of D-Bug12 prior to 2.1.x access memory a byte at a time through low-level drivers. Because all on-chip memory modules support byte access, utilizing this method simplified the low level driver code. However, this access method presents some potential problems for 16-bit registers that reside in the on-chip peripherals. Because the data bus connection to the on-chip peripherals is 16-bits wide, with a few exceptions, the peripherals are designed in such a way that 16-bit registers must be read or written with a single 16-bit access to ensure data coherency.

D-Bug12's low level memory access drivers have been rewritten to perform aligned word reads whenever possible. For instance, if the Memory Modify Word (MMW) command is used with an even address, all reads and writes will be performed as aligned word accesses. However, if the MMW command is used with an odd address, each memory access will be performed as two individual byte read or write operations. Because the Memory Display commands (MD and MDW) always display an even multiple of 16 bytes, all memory read operations are performed as aligned word accesses.

### **XIRQ Interrupt Usable As Program Abort Input**

When testing and debugging programs that reside in the internal RAM or EEPROM of the MC68HC912B32 when operating the EVB in EVB mode, it is possible for the program to become 'hung-up' and never return to the D-Bug12 prompt. In these cases, it is desirable to abort the user program execution and return control to D-Bug12. Unfortunately, pressing the reset switch, S1, causes a complete reinitialization of D-Bug12 resulting in a complete loss of information about the state of the executing user code. All versions of D-Bug12 utilize the XIRQ interrupt input as a program abort function. Even though a program abort switch is not present on the MC68HC912B32 EVB, the XIRQ interrupt input (PE0) may be utilized for a program abort function. One side of a normally open momentary contact push button should be wired to the XIRQ input, the other side of the push button should be wired to Vss.

Utilizing the program abort function will return control back to D-Bug12, displaying the CPU register contents at the point where the users program was terminated.

### **Command Line Buffer Length**

The command line buffer was reduced from 80 characters to 50 characters to make more memory available for the additional features of version 2.1.x. This change does not affect the operation of any of the D-Bug12 commands.

## **Maximum Number of Command Line Arguments**

The maximum number of command line arguments was increased from 10 to 11 (including the command name itself) to accommodate the additional data required by the DEVICE command. This change also allows all 10 software breakpoints to be set utilizing a single BR command.

## **Maximum S-Record Length Reduced**

Previous versions of D-Bug12 permitted the use of S-Records containing a code/data field of up to 64 bytes with the LOAD, VERIFY and FLOAD commands. To make more memory available for the additional features of version 2.1.x, this number was reduced to a maximum 32 bytes. This should not cause any problems for most developers. If an attempt is made to utilize an S-Record with a longer code/data field, an error message is issued.

## **MC68HC912BC32 CAN Interrupt Vector Support**

When D-Bug12 is operated in EVB mode, it provides default interrupt handlers for all of the on-chip peripherals. Earlier versions of D-Bug12 only supported the MC68HC912B32 as a host CPU. Version 2.1.x now fully supports the use of the MC68HC912BC32 as a host CPU by providing default interrupt handlers for the on-chip CAN communication module. The MC68HC912BC32 is identical to the MC68HC912B32 except that the MC68HC912B32's BDLC module was replaced by the CAN module.

## D-Bug12 Command Set

The following list summarizes the D-Bug12 command set. Each command's function and command line syntax are described in detail.

- ASM - Single line assembler/disassembler.
- BAUD - Set the SCI communications BAUD rate
- BF - Block Fill user memory with data.
- BR - Set/Display user breakpoints.
- BULK - Bulk erase on-chip EEPROM
- CALL - Execute a user subroutine, return to D-Bug12 when finished.
- DEVICE - Select/define a new target MCU device.
- EEBASE - Inform D-Bug12 of the target's EEPROM base address
- FBULK - Erase the target processor's on-chip Flash EEPROM
- FLOAD - Program the target processor's on-chip Flash EEPROM from S-Records
- G - Go. Begin execution of user program.
- GT - Go Till. Set a temporary breakpoint and begin execution of user program.
- HELP - Display D-Bug12 command set and command syntax.
- LOAD - Load user program in S-Record format.
- MD - Memory Display. Display memory contents in hex bytes/ASCII format.
- MDW - Memory Display Words. Display memory contents in hex words/ASCII format.
- MM - Memory Modify. Interactively examine/change memory contents.
- MMW - Memory Modify Words. Interactively examine/change memory contents.
- MOVE - Move a block of memory.
- NOBR - Remove one/all user breakpoints.
- RD - Register Display. Display the CPU register contents.
- REGBASE - Inform D-Bug12 of the target's I/O register's base address
- RESET - Reset the target CPU
- RM - Register Modify. Interactively examine/change CPU register contents.
- STOP - Stop the execution of user code in the target processor and place the target processor in background mode.
- T - Trace. Execute an instruction, disassemble it, and display the CPU registers.
- UPLOAD - Display memory contents in S-Record format.
- USEHBR - Use EVB/Target Hardware breakpoints
- VERF - Verify memory contents against S-Record Data.
- <RegisterName> <RegisterValue> - Set CPU <RegisterName> to <RegisterValue>

## ASM - Single Line Assembler/Disassembler Command

### Command Line Format

ASM <Address>

### Parameter Description

<Address> - A 16-bit hexadecimal number

### Command Description

The assembler/disassembler is an interactive memory editor that allows memory contents to be viewed and altered using assembly language mnemonics. Each entered source line is translated into machine language code and placed into memory at the time of entry. When displaying memory contents, each instruction is disassembled into its source mnemonic form and displayed along with the hexadecimal machine code and any instruction operands.

Assembler mnemonics and operands may be entered in any mix of upper and lower case letters. Any number of spaces may appear between the assembler prompt and the instruction mnemonic or between the instruction mnemonic and the operand. Numeric values appearing in the operand field are interpreted as *signed* decimal numbers with one exception. Placing a \$ in front of any number will cause the number to be interpreted as a hexadecimal number.

When an instruction has been disassembled and displayed, the D-Bug12 prompt is displayed following the disassembled instruction. If a carriage return is entered immediately following the prompt, the next instruction in memory is disassembled and displayed on the next line.

If a CPU12 instruction is entered following the prompt, the entered instruction is assembled and placed into memory. The line containing the new entry is erased and the new instruction is disassembled and displayed on the same line. The contents of the next memory location(s) is disassembled and displayed on the screen.

The instruction mnemonics and operand formats accepted by the assembler follows the syntax as described in the *M68HC12 Family CPU12 Reference Manual*.

There are a number of M68HC11 instruction mnemonics that appear in the *M68HC12 Family CPU12 Reference Manual* that do not have direct equivalent CPU12 instructions. These mnemonics, listed in the table below, are translated into functionally equivalent CPU12 instructions. To aid the current M68HC11 users that may desire continue to use the M68HC11 mnemonics, the disassembler portion of the assembler/disassembler recognizes the functionally equivalent CPU12 instructions and disassembles those instructions into the equivalent M68HC11 mnemonics.

When entering branch instructions, the number placed in the operand field should be the absolute destination address of the instruction. The assembler will calculate the two's complement offset of the branch.

The assembly/disassembly process may be terminated by entering a period (.) following the assembler prompt.

## Restrictions

None.

| M68HC11 Mnemonic | CPU12 Instruction | M68HC11 Mnemonic                    | CPU12 Instruction                   |
|------------------|-------------------|-------------------------------------|-------------------------------------|
| CLC              | ANDCC #\$FE       | INS                                 | LEAS 1,S                            |
| CLI              | ANDCC #\$EF       | TAP                                 | TFR A,CC                            |
| CLV              | ANDCC #\$FD       | TPA                                 | TFR CC,A                            |
| SEC              | ORCC #\$01        | TSX                                 | TFR S,X                             |
| SEI              | ORCC #\$10        | TSY                                 | TFR S,Y                             |
| SEV              | ORCC #\$02        | XGDX                                | EXG D,X                             |
| ABX              | LEAX B,X          | XGDY                                | EXG D,Y                             |
| ABY              | LEAY B,Y          | SEX R <sub>8</sub> ,R <sub>16</sub> | TFR R <sub>8</sub> ,R <sub>16</sub> |
| DES              | LEAS -1,S         |                                     |                                     |

## M68HC11 to CPU12 Instruction Translation

### Example

>ASM 700

```
0700 CC1000          LDD    #4096
0703 1803123401FE  MOVW  #$1234,$01FE
0709 0EF9800001F1  BRSET -32768,PC,$01,$0700
070F 18FF          TRAP  $FF
0711 183FE3          ETBL  <Illegal Addr Mode> >.
```

>

## Assembly Operand Format

This section describes the operand format used by the assembler when assembling CPU12 instructions. The operand format accepted by the assembler is described separately in the *CPU12 Reference Manual*. Rather than describe the numeric format accepted for each instruction, some general rules will be used. Exceptions and complicated operand formats are described separately.

In general, anywhere the assembler expects a numeric value in the operand field, either a decimal or hexadecimal value may be entered. Decimal numbers are entered as signed constants having a range of -32768..65535. A leading minus sign (-) indicates negative numbers, the absence of a leading minus sign indicates a positive number. A leading plus sign (+) is not allowed. Hexadecimal numbers must be entered with a leading dollar sign (\$) followed by one to four hexadecimal digits. The default number base is decimal.

For all branching instructions, (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, TBNE) the number entered in the address portion of the operand field must be the *absolute address of the branch destination*. The assembler will calculate the two's complement offset to be placed in the assembled object code.

The D-Bug12 assembler allows an optional # symbol to precede the 8-bit mask value in all bit manipulation instructions (BSET, BCLR, BRSET, BRCLR).

## Disassembly Operand Format

This section describes the operand format for the disassembler that is used in conjunction with the single line assembler. The operand format used by the disassembler is described separately in the *CPU12 Reference Manual*. Rather than describe the numeric format used for each instruction, some general rules will be applied. Exceptions and complicated operand formats will be described separately.

All numeric values disassembled as hexadecimal numbers will be preceded by a dollar sign (\$) to avoid being confused with values disassembled as signed decimal numbers.

For all branch (Bcc, LBcc, BRSET, BRCLR, DBEQ, DBNE, IBEQ, IBNE, TBEQ, TBNE) instructions the numeric value of the address portion of the operand field will be displayed as the hexadecimal *absolute address of the branch destination*.

All offsets used with indexed addressing modes will be disassembled as *signed* decimal numbers.

All addresses, whether direct or extended, will be disassembled as four digit hexadecimal numbers.

All 8-bit mask values (BRSET/BRCLR/ANDCC/ORCC) will be disassembled as two digit hexadecimal numbers.

For bit manipulation instructions (BSET, BCLR, BRSET, BRCLR), the disassembler always displays the # symbol preceding the 8-bit mask value.

All 8-bit immediate values will be disassembled as hexadecimal numbers.

All 16-bit immediate values will be disassembled as hexadecimal numbers.

## BAUD - Change The Communications BAUD Rate

### Command Line Format

BAUD <BAUDRate>

### Parameter Description

<BAUDRate> An unsigned 16-bit decimal number

### Command Description

The BAUD command is used to change the communications rate of the SCI that is used by D-Bug12 to communicate with the user.

### Restrictions

Because the <BAUDRate> parameter supplied on the command line is a 16-bit unsigned integer, BAUD rates greater than 65535 baud cannot be set using this command. The SCI BAUD rate divider value for the requested BAUD rate is calculated using the M clock value that is supplied in the *CustomizationData* area. Because the SCI BAUD rate divider is a 13-bit counter, certain BAUD rates may not be supported at particular MCU clock frequencies.

### Example

```
>baud 50
```

```
Invalid BAUD Rate
```

```
>baud 38400
```

```
Change Terminal BR, Press Return
```

```
>
```

## **BF - Fill memory with data**

### **Command Line Format**

BF <StartAddress> <EndAddress> [<Data>]

### **Parameter Description**

<StartAddress> A 16-bit hexadecimal number  
<EndAddress> A 16-bit hexadecimal number  
<Data> An 8-bit hexadecimal number

### **Command Description**

The Block Fill command is used to place a single 8-bit value into a range of memory locations. <StartAddress> is the first memory location written with data and <EndAddress> is the last memory location written with data. If the <data> parameter is omitted the memory range is filled with the value \$00.

### **Restrictions**

None.

### **Example**

```
>bf 400 fff 0  
>bf f00 fff 55  
>
```

## BR - Set/Display User Breakpoints

### Command Line Format

BR [<Address> <Address>...]

### Parameter Description

<Address>A 16-bit hexadecimal number

### Command Description

The BR command is used to set a breakpoint at a specified address or to display any previously set breakpoints. The function of a breakpoint is to halt user program execution when the program reaches the breakpoint address. When a breakpoint address is encountered, D-Bug12 will disassemble the instruction at the breakpoint address, print the CPU12's register contents, and wait for the next D-Bug12 command to be entered by the user.

Breakpoints are set by entering the breakpoint command followed by one or more breakpoint addresses. Entering the breakpoint command without any breakpoint addresses will display all the currently set breakpoints.

A maximum of 10 breakpoints may be set at one time when using software breakpoints (default). A maximum of 2 breakpoints may be set when using the EVB or target CPU's hardware breakpoint capability. For additional information on D-Bug12's hardware breakpoint support, see the USEHBR command description.

### Restrictions

D-Bug12 implements the software breakpoint function by replacing the opcode at the breakpoint address with an SWI instruction when operating in the EVB mode or the BGND instruction when operating in the POD mode. A breakpoint may not be set on a user SWI instruction when operating in EVB mode. In either mode breakpoints may only be set at an opcode address and breakpoints may only be placed at memory addresses implemented as RAM.

When using the on-chip hardware breakpoints, D-Bug12 utilizes the the breakpoint module in either SWI Dual Address (EVB) or BDM Dual Address (POD) mode. Both of these breakpoint module modes utilize the CPU12 instruction fetch tagging mechanism which only allows breakpoints to be set on instruction opcodes.

When operating in the POD mode, new breakpoints may not be set with the BR command when the 'R>' prompt is being displayed. However, the BR command may be used to display breakpoints that are currently set in the user's running program.

### Example

```
>br 35ec 2f80 c592  
Breakpoints: 35ec 2f80 c592
```

```
>br  
Breakpoints: 35EC 2F80 C592
```

```
>
```

## **BULK - Bulk Erase on-chip EEPROM**

### **Command Line Format**

BULK

### **Parameter Description**

No parameters are required

### **Command Description**

The BULK command is used to erase the entire contents of the on-chip EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip EEPROM location shall be checked for contents of \$FF.

### **Restrictions**

None.

### **Example**

```
>BULK  
  
F/EEPROM Failed To Erase  
>BULK  
  
>
```

## CALL - Execute A User Subroutine

### Command Line Format

CALL [<Address>]

### Parameter Description

<Address> A 16-bit hexadecimal number

### Command Description

The CALL command is used to execute a subroutine and return to the D-Bug12 monitor program when the final RTS of the subroutine is executed. When control is returned to D-Bug12, the CPU register contents will be displayed. All CPU registers contain the values at the time the final RTS instruction was executed with the exception of the program counter (PC). The PC will contain the starting address of the subroutine. If a subroutine address is not supplied on the command line, the current value of the Program Counter (PC) will be used as the starting address.

NOTE: No breakpoints are placed in memory before execution is transferred to user code.

### Restrictions

If the called subroutine modifies the value of the stack pointer during its execution, it **MUST** restore the stack pointer's original value before executing the final RTS of the called subroutine. This restriction is required because D-Bug12 places four bytes of data on the users stack that causes control to return to D-Bug12 when the final RTS of the subroutine is executed. Obviously, any subroutine must obey this restriction to execute properly.

The CALL command cannot be issued when the 'R>' prompt is being displayed indicating that the target system is already running a user program.

### Example

```
>call 820
```

```
Subroutine Call Returned
```

```
PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
0820   0A00   057C   0000      0F:F9           1001 0000
0820   CCFFFF           LDD    #$0FFF
```

```
>
```

## DEVICE - Specify a target MCU device type

### Command Line Format

```
DEVICE
DEVICE ?
DEVICE <DeviceName>[<EEStart> <EEEnd> <FStart> <FEnd> <RAMStart> <RAMEnd>
<IOBase> <PPageAddr> <NumPages>]
```

### Parameter Description

|              |                                                                         |
|--------------|-------------------------------------------------------------------------|
| <DeviceName> | Maximum of 7 ASCII characters used to select/define a target MCU device |
| <EEStart>    | on-chip EEPROM starting address; a 16-bit hexadecimal number            |
| <EEEnd>      | on-chip EEPROM ending address; a 16-bit hexadecimal number              |
| <FStart>     | on-chip Flash EEPROM starting address; a 16-bit hexadecimal number      |
| <FEnd>       | on-chip Flash EEPROM ending address; a 16-bit hexadecimal number        |
| <RAMStart>   | on-chip RAM starting address; a 16-bit hexadecimal number               |
| <RAMEnd>     | on-chip RAM ending address; a 16-bit hexadecimal number                 |
| <IOBase>     | Base address of the on-chip I/O registers; a 16-bit hexadecimal number  |
| <PPageAddr>  | I/O address of the PPage register at reset; a 16-bit hexadecimal number |
| <NumPages>   | Number of 16K memory pages; a 16-bit hexadecimal number                 |

### Command Description

Selecting the proper target MCU with the DEVICE command provides D-Bug12 the information necessary to allow transparent alteration of the target MCU's on-chip EEPROM using any D-Bug12 commands that modify memory. It also allows provides the necessary information to allow the programming and erasure of on-chip Flash EEPROM. In addition, it allows D-Bug12 to initialize the stack pointer to the top of on-chip RAM when the target MCU is reset by use of the RESET command. The DEVICE command has four command line formats that allows for the display, selection and/or definition of target device parameters.

Entering "DEVICE" on the command line followed by a carriage return will display the name of the currently selected device, the on-chip EEPROM's starting and ending address, the on-chip Flash EEPROM's starting and ending address, the on-chip RAM's starting and ending address, and the I/O Base address. This form of the command may be used when D-Bug12 is operating in either EVB or POD mode.

When D-Bug12 is operated in the POD mode, the device command may also be used to select or define a new target device. Entering the DEVICE command followed only by a device name will configure D-Bug12 for operation with the selected target device. The default device list contains entries for the MC68HC912B32, MC68HC912BC32, MC68HC812A4, MC68HC912D60 and theMC68HC912DA/DG128. The table below shows the command line name to use for the default MCU devices.

| <u>Device Name</u> | <u>Target MCU</u> |
|--------------------|-------------------|
| 912B32             | MC68HC912B(C)32   |
| 812A4              | MC68HC812A4       |
| 912D60             | MC68HC912D60      |
| DA128              | MC68HC912DA/DG128 |

Entering the DEVICE command followed by a device name and nine hexadecimal parameters allows new devices to be added to the target device table or existing device table entries to be

modified. When a new device is added or when an existing device entry is modified, it becomes the currently selected device. If a new device does not contain a particular on-chip resource, such as Flash EEPROM, a value of zero should be entered for the starting and ending addresses

Because the target device data and the current device selection are stored in the probe MCU's on-chip EEPROM, new device information and the device selection are retained when power is removed from the POD. If the MC68HC912B32EVB is operated in EVB mode and the contents of **ANY** locations of the on-chip EEPROM are altered it is **STRONGLY** recommended that the on-chip EEPROM be completely erased by using the BULK command before using the EVB in POD mode again. Erasing the on-chip EEPROM will cause D-Bug12 to reinitialize the the device table with the default MCU devices. The information for any new devices that were added to the table will be lost.

The <PPageAddr> and <NumPages> parameters are used to provide D-Bug12 with information it requires to program Flash devices with greater than 64K bytes of memory. The <PPageAddr> parameter must specify the I/O address of the PPAGE register at reset. The <NumPages> parameter is used to specify the number of 16K byte pages that are visible in the \$8000 - \$BFFF memory window. For a device such as the the MC68HC912DA/DG128, that contains 128K of Flash, the <PPageAddr> parameter would be \$FF and the <NumPages> parameter would be 8 (128K ÷ 16K). If a device does not contain more than 64K of Flash program memory, a value of zero must be provided for these two parameters.

### **Restrictions**

When operating the M68EVB912B32 in EVB mode, the DEVICE command may only be used to display the current device information.

The DEVICE command maintains a 16-bit checksum on the contents of the entire on-chip EEPROM to maintain the integrity of the device table. If any of the on-chip EEPROM locations are altered while operating the M68EVB912B32 in EVB mode, D-Bug12 will reinitialize the device table with the default device information contained in the on-chip Flash. However, it is possible for the checksum verification to fail (one case where the checksum will fail is if the entire contents of the on-chip EEPROM is programmed with zeros). Therefore, it is **STRONGLY** recommended that the on-chip EEPROM be completely erased by using the BULK command before using the EVB in POD mode again. Using the EVB in POD mode with a corrupt device data table may cause D-Bug12 to operate in an unpredictable manner.

The 768 bytes of on-chip EEPROM will allow a total of 29 entries in the device table. **DO NOT** exceed this number.

When adding a new device to the device table, the addresses provided for the on-chip Flash EEPROM, on-chip RAM and the I/O Registers should reflect the locations of these resources when the part is reset. This requirement is necessary for the FBULK and FLOAD command to work properly.

## Example

>device

Device: 912B32  
EEPROM: \$0D00 - \$0FFF  
Flash: \$8000 - \$FFFF  
RAM: \$0800 - \$0BFF  
I/O Regs: \$0000  
S>device 912b32 1d00 1fff 8000 ffff 800 bff 0

Device: 912B32  
EEPROM: \$1D00 - \$1FFF  
Flash: \$8000 - \$FFFF  
RAM: \$0800 - \$0BFF  
I/O Registers: \$0000

S>device 812a4

Device: 812A4  
EEPROM: \$1000 - \$1FFF  
RAM: \$0800 - \$0BFF  
I/O Registers: \$0000

S>device da128

Device: DA128  
EEPROM: \$0800 - \$0FFF  
Flash: \$8000 - \$BFFF Pages: 8 PPAGE at: \$00FF  
RAM: \$2000 - \$3FFF  
I/O Regs: \$0000

S>

## EEBASE - Specify the EEPROM base address

### Command Line Format

EEBASE <Address>

### Parameter Description

<Address> A 16-bit hexadecimal number

### Command Description

Each time D-Bug12 performs a memory write, it will automatically perform the necessary register manipulations to program the on-chip EEPROM if the write operation falls within the address range of the target's on-chip EEPROM. Because user code may change the EEPROM's base address may be changed by writing to the INITEE register, D-Bug12 must be informed of the EEPROM's location if automatic EEPROM writes are to occur. The EEBASE command is used to specify the base address of the target processor's on-chip EEPROM.

When operating in EVB mode, the default EEPROM base address and range are specified in the Customization Data variables `CustomData.EEBase` and `CustomData.EESize`. The value in `CustomData.EEBase` is used by the startup code to remap the EEPROM. The EEBASE command may not be used to relocate the I/O registers.

When operating in POD mode, the target's default EEPROM base address and range are specified by the currently selected device (See the DEVICE command description for additional details).

The EEBASE command does not check to ensure that the parameter is a valid base address for the selected M68HC12 family member. If an improper base address is provided, automatic programming of the on-chip EEPROM will not operate properly.

---

---

**Note:** The EEBASE command does not automatically modify the INITEE register. It is the responsibility of the user to ensure that the INITEE register is modified either manually or through the execution of user code.

---

---

### Example

S>device

```
Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000
```

S>eebase 1d00

Device: 912B32

EEPROM: \$1D00 - \$1FFF

Flash: \$8000 - \$FFFF

RAM: \$0800 - \$0BFF

I/O Regs: \$0000

S>mm 12

0012 01 11

0013 0F .

S>md 1d00

1D00 FF FF FF FF - FF FF FF FF - FF FF FF FF - FF FF FF FF .....

S>

## **FBULK - Erase target on-chip Flash EEPROM Memory**

### **Command Line Format**

FBULK

### **Parameter Description**

No parameters are required

### **Command Description**

The FBULK command is used to erase the entire contents of the on-chip Flash EEPROM in a single operation. After the bulk erase operation has been performed, each on-chip Flash location shall be checked for contents of \$FF. The target processor's Flash memory is erased by resetting the target processor and then loading a small 'driver' program into the target processor's on-chip RAM. For this reason, the previous contents of the target processor's On-chip RAM is lost.

### **Restrictions**

When operating in the 'EVB' mode, the FBULK command cannot be used. If the FBULK command is entered while in 'EVB' mode, an error message is displayed and command execution will be terminated.

Before using the FBULK command, a target device must be selected (see the DEVICE command description) that reflects the locations of the on-chip Flash EEPROM, on-chip RAM and the I/O Registers when the part is reset. Failure to follow this restriction will cause the FBULK command to fail and may require that the EVB be reset.

Because the FBULK command downloads a small 'driver' program into the target MCU's on chip RAM, D-Bug12's breakpoint table is cleared before beginning execution of the 'driver'. This is necessary to prevent previously set breakpoints from accidentally halting the execution of the driver program.

### **Example**

```
S><u>fbulk</u>
Flash Programming Voltage Not Present
S><u>fbulk</u>
F/EEPROM Failed To Erase
S><u>fbulk</u>
S>

><u>fbulk</u>
Command Not Allowed In EVB Mode
>
```

## FLOAD - Program on-chip Flash memory from S-Records

### Command Line Format

FLOAD [<AddressOffset>]

### Parameter Description

<AddressOffset>            A 32-bit hexadecimal number

### Command Description

The FLoad command is used to program a target device's Flash EEPROM memory with the data contained in S-Record object files. The address offset, if supplied, is added to the load address of each S-Record before an S-Record's data bytes are placed in memory. Providing an address offset other than zero allows object code or data to be programmed into memory at a location other than that for which it was assembled or compiled. An offset greater than \$FFFF may only be used with devices that support more than 64K bytes of memory.

---

---

**Note:** Please refer to the section titled "FLOAD, LOAD and VERIFY S-Record Format" at the end of this document for a complete description of the S-Record Format utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

---

---

The programming of the on-chip Flash memory uses an algorithm where the time required to program each byte or word can vary from as little as 60  $\mu$ S to as long as 3.5 mS (Note, however that the programming time for each byte or word should typically take no more than 120  $\mu$ S - 180  $\mu$ S). Because of this variability, the FLOAD command uses a software handshaking protocol to control the flow of S-Record data from the host computer. When the FLOAD command is ready to receive an S-Record, an ASCII asterisk character (\*) is sent to the host computer. The host computer should respond by sending a single S-Record. The S-Record may include a carriage return and/or line feed character(s). Most commercial terminal programs that are capable of sending ASCII text files have the ability to wait for a specific character or characters before sending a line of text.

The FLoad command is terminated when D-Bug12 receives an 'S8' or 'S9' end of file record. If the object file being loaded does not contain an 'S8' or 'S9' record, D-Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing a system Reset will return D-Bug12 to its command line prompt.

### Restrictions

As mentioned previously, the host program used to send the S-Record data must be capable of waiting for an ASCII asterisk character (\*) before sending each S-Record line.

Because the on-chip Flash EEPROM is only bulk erasable, the FBULK command should be used before attempting to program the Flash memory using the FLOAD command.

The FLOAD command cannot be used with target MCUs operating with crystal speeds lower than 3.0 MHz (E-clock speeds less than 1.5 MHz).

The FLOAD command cannot be used with S-Records that contain a code/data field longer than

32 bytes. Sending an S-Record with a code/data field longer than 32 bytes will cause D-Bug12 to terminate the FLOAD command the issue an error message.

Before using the FLOAD command, a target device must be selected (see the DEVICE command description) that reflects the locations of the on-chip Flash EEPROM, on-chip RAM and the I/O Registers when the part is reset. Failure to follow this restriction will cause the FLOAD command to fail and may require that the EVB be reset.

Because the FLOAD command downloads a small 'driver' program into the target MCU's on chip RAM, D-Bug12's breakpoint table is cleared before beginning execution of the 'driver'. This is necessary to prevent previously set breakpoints from accidentally halting the execution of the driver program.

Supplying an address offset greater than \$FFFF for an M68HC12 family member that contains less than 64K of addressable program memory will result in termination of the FLOAD command and an error message being issued.

### Example

```
S><u>fload</u>
Flash Programming Voltage Not Present
S><u>fload</u>
*****
*****
*****
S>
```

## Go, begin execution of user code

### Command Line Format

G [<Address>]

### Parameter Description

<Address>A 16-bit hexadecimal number

### Command Description

The G command is used to begin the execution of user code in real time. Before beginning execution of user code, any breakpoints set using the BR command are placed in memory. Execution of the user program will continue until a user breakpoint is encountered, a CPU exception occurs or the reset switch on the HC12EVB is pressed. When user code halts for one of these reasons and control is returned to D-Bug12, a message shall be displayed explaining the reason for user program termination. In addition, D-Bug12 displays the CPU12's register contents, disassembles the instruction at the current PC address, and waits for the next D-Bug12 command to be entered by the user.

If a starting address is not supplied in the command line parameter, program execution will begin at the address defined by the current value of the Program Counter.

### Restrictions

The G command cannot be issued when the 'R>' prompt is being displayed indicating that the target system is already running a user program.

### Example

```
S>g 800
R>md 1000

1000 FF FF FF FF - FF FF FF FF - FF FF FF FF - FF FF FF FF .....
R>
User Breakpoint Encountered

  PC   SP   X    Y    D = A:B  CCR = SXHI NZVC
0820 09FE 057C 0000 00:00      1001 0100
0820 08                INX
S>
```

## GT - Go Until, Execute user code until temporary breakpoint

### Command Line Format

GT <Address>

### Parameter Description

<Address>A 16-bit hexadecimal number

### Command Description

The GT command is similar to the G command except that a temporary breakpoint is placed at the address supplied on the command line. Any breakpoints that were set by the BR command are NOT placed in the user's code before program execution begins. Program execution begins at the address defined by the current value of the Program Counter. When user code reaches the temporary breakpoint and control is returned to D-Bug12, a message is displayed explaining the reason for user program termination. In addition, D-Bug12 displays the CPU12's register contents, disassembles the instruction at the current PC address, and waits for the next D-Bug12 command to be entered by the user.

### Restrictions

The GT command cannot be issued when the 'R>' prompt is being displayed indicating that the target system is already running a user program.

### Example

```
S>gt 820
```

```
R>
```

```
Temporary Breakpoint Encountered
```

```
PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
0820  09FE  057C  0000      00:00          1001 0100
0820  08                INX
S>
```

## **HELP - Display D-Bug12 command summary**

### **Command Line Format**

HELP

### **Parameter Description**

No parameters are required

### **Command Description**

The HELP command is used to display a summary of the D-Bug12 command set. Each command is shown along with its command line format and a brief description of the command's function. The commands are listed in alphabetical order.

### **Restrictions**

None.

### **Error Conditions**

None.

## Example

```
>help
ASM <Address> Single line assembler/disassembler
  <CR> Disassemble next instruction
  <.> Exit assembly/disassembly
BAUD <baudrate> Set communications rate for the terminal
BF <StartAddress> <EndAddress> [<data>] Fill memory with data
BR [<Address>] Set/Display breakpoints
BULK Erase entire on-chip EEPROM contents
CALL [<Address>] Call user subroutine at <Address>
DEVICE [<DevName> [<Address>...<Address>]] display/select/add target device
EEBASE <Address> Set base address of on-chip EEPROM
FBULK Erase entire target FLASH contents
FLOAD [<AddressOffset>] Load S-Records into target FLASH
G [<Address>] Begin/continue execution of user code
GT <Address> Set temporary breakpoint at <Address> & execute user code
HELP Display D-Bug12 command summary
LOAD [<AddressOffset>] [;d] Load S-Records into memory
MD <StartAddress> [<EndAddress>] Memory Display Bytes
MDW <StartAddress> [<EndAddress>] Memory Display Words
MM <StartAddress> Modify Memory Bytes
  <CR> Examine/Modify next location
  </> or <=> Examine/Modify same location
  <^> or <-> Examine/Modify previous location
  <.> Exit Modify Memory command
MMW <StartAddress> Modify Memory Words (same subcommands as MM)
MOVE <StartAddress> <EndAddress> <DestAddress> Move a block of memory
NOBR [<address>] Remove One/All Breakpoint(s)
RD Display CPU registers
REGBASE <Address> Set base address of I/O registers
RESET Reset target CPU
RM Modify CPU Register Contents
STOP Stop target CPU
T [<count>] Trace <count> instructions
UPLOAD <StartAddress> <EndAddress> S-Record Memory display
USEHBR Use Hardware Breakpoints
VERF [<AddressOffset>] Verify S-Records against memory contents
<Register Name> <Register Value> Set register contents
  Register Names: PC, SP, X, Y, A, B, D
  CCR Status Bits: S, XM, H, IM, N, Z, V, C
>
```

## LOAD - Load user program in S-Record format

### Command Line Format

LOAD [<AddressOffset>] [;d]

### Parameter Description

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| <AddressOffset> | A 32-bit hexadecimal number                                                  |
| “;d”            | Load S-Records into ‘data’ memory (for devices with > 64K of program memory) |

### Command Description

The Load command is used to load S-Record object files into user memory from an external device. The address offset, if supplied, is added to the load address of each S-Record before an S-Record’s data bytes are placed in memory. Providing an address offset other than zero allows object code or data to be loaded into memory at a location other than that for which it was assembled. An offset greater than \$FFFF may only be used with devices that support more than 64K bytes of memory.

---

---

**Note:** Please refer to the section titled “FLOAD, LOAD and VERIFY S-Record Format” at the end of this document for a complete description of the S-Record Format utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

---

---

During the loading process, the S-Record data is not echoed to the control console. However, for each ten S-Records that are successfully loaded, an ASCII asterisk character (\*) is sent to the control console. When an S-Record file has been successfully loaded, D-Bug12 will issue its prompt.

The Load command is terminated when D-Bug12 receives an ‘S8’ or ‘S9’ end of file record. If the object file being loaded does not contain an ‘S8’ or ‘S9’ record, D-Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing a systems Reset button will return D-Bug12 to its command line prompt.

The ‘;d’ option is used to load S-Records, containing program or data, into target memory such as RAM or EEPROM that is outside of the normal program memory range. This option is only required by devices that support more than 64K bytes of memory and have a device definition where the number of 16K memory pages is greater than zero. This option allows the S-Record loader to distinguish between S-Records that are to be loaded into paged program memory and those destined for other areas of on- or off-chip memory.

### Restrictions

When operating in POD mode, the LOAD command will not support standard baud rates above 19,200 when the target MCU is operating at an E-clock frequency of 8.0 MHz (16.0 MHz crystal). This restriction is due to the overhead involved in the implementation of the Custom Serial Protocol required by the Single Wire Background Debug pin. For target MCUs operating at lower E-clock frequencies, the maximum baud rate that can be used with the LOAD command will be proportionally lower.

## Example

```
>load 1000  
*****  
>
```

## MD - Display memory in hexadecimal bytes and ASCII format

### Command Line Format

MD <StartAddress> [<EndAddress>]

### Parameter Description

<StartAddress>            A 16-bit hexadecimal number  
<EndAddress>             A 16-bit hexadecimal number

### Command Description

The memory display command displays the contents of memory in both hexadecimal bytes and ASCII, 16-bytes on each line. The <StartAddress> parameter must be supplied, however, the <EndAddress> parameter is optional. When the <EndAddress> parameter is not supplied, a single line is displayed.

The number supplied as the <StartAddress> parameter is rounded down to the next lower multiple of 16. While the number supplied as the <EndAddress> parameter is rounded up to the next higher multiple of 16 - 1. This causes each line to display memory in the range of \$xxx0 through \$xxxF. For example if the user entered \$205 as the start address and \$217 as the ending address, the actual memory range displayed would be \$200 through \$21F.

### Restrictions

None.

### Example

```
>md 800
0800 AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20    ..7j..'5.x..Vx

>md 800 87f
0800 AA 04 37 6A - 00 06 27 F9 - 35 AE 78 0D - B7 56 78 20    ..7j..'5.x..Vx
0810 B6 36 27 F9 - 35 AE 27 F9 - 35 9E 27 F9 - 35 BE B5 28    .6'.5.'.5.'.5..(
0820 27 F9 35 D6 - 37 B8 00 0F - 37 82 01 0A - 37 36 FF F0    '.5.7...7...76..
0830 7C 10 37 B3 - 00 00 37 B6 - 00 0F AA 04 - A5 02 37 B6    |.7...7.....7.
0840 00 0F 27 78 - 37 6A 00 06 - 27 F9 35 78 - 27 F9 35 56    ..'x7j..'5x'.5V
0850 78 0D B7 10 - 78 3B 37 86 - 00 DC 27 F9 - 35 48 78 57    x...x;7...'5HxW
0860 37 86 00 DE - F5 01 EA 09 - 37 B5 0D 0A - 27 F9 36 2A    7.....7...'6*
0870 A5 00 37 65 - 00 02 27 F9 - 35 E8 37 9C - 37 4C F5 02    ..7e..'5.7.7L..
```

## MDW - Display memory in hexadecimal words and ASCII format

### Command Line Format

MDW <StartAddress> [<EndAddress>]

### Parameter Description

<StartAddress> A 16-bit hexadecimal number

<EndAddress> A 16-bit hexadecimal number

### Command Description

The memory display command displays the contents of memory in both hexadecimal words and ASCII, 16-bytes on each line. The <StartAddress> parameter must be supplied, however, the <EndAddress> parameter is optional. When the <EndAddress> parameter is not supplied, a single line is displayed.

The number supplied as the <StartAddress> parameter is rounded down to the next lower multiple of 16. While the number supplied as the <EndAddress> parameter is rounded up to the next higher multiple of 16 - 1. This causes each line to display memory in the range of \$xxx0 through \$xxxF. For example if the user entered \$205 as the start address and \$217 as the ending address, the actual memory range displayed would be \$200 through \$21F.

### Restrictions

None.

### Example

```
>mdw 800
0800 AA04 376A - 0006 27F9 - 35AE 780D - B756 7820 ..7j...'.5.x..Vx

>mdw 800 87f
0800 AA04 376A - 0006 27F9 - 35AE 780D - B756 7820 ..7j...'.5.x..Vx
0810 B636 27F9 - 35AE 27F9 - 359E 27F9 - 35BE B528 .6'.5.'.5.'.5..(
0820 27F9 35D6 - 37B8 000F - 3782 010A - 3736 FFF0 '.5.7...7...76..
0830 7C10 37B3 - 0000 37B6 - 000F AA04 - A502 37B6 |.7...7.....7.
0840 000F 2778 - 376A 0006 - 27F9 3578 - 27F9 3556 ..'x7j...'.5x'.5V
0850 780D B710 - 783B 3786 - 00DC 27F9 - 3548 7857 x...x;7...'.5HxW
0860 3786 00DE - F501 EA09 - 37B5 0D0A - 27F9 362A 7.....7...'.6*
0870 A500 3765 - 0002 27F9 - 35E8 379C - 374C F502 ..7e...'.5.7.7L..
>
```

## MM - Modify memory bytes in hexadecimal format

### Command Line Format

MM <Address> [<data>]

### Parameter Description

<Address> A 16-bit hexadecimal number

<data> An 8-bit hexadecimal number

### Command Description

The memory modify word command allows the contents of memory to be examined and/or modified as 8-bit hexadecimal data. If the 8-bit data parameter is present on the command line, the byte at memory location at <Address> is replaced with <data>. If not, D-Bug12 will enter the interactive memory modify mode. In the interactive mode, each byte is displayed on a separate line following the data's address. Once the memory modify command has been entered, several sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:

|                 |                                                                       |
|-----------------|-----------------------------------------------------------------------|
| [<Data>] <CR>   | Optionally update current location and display the next location      |
| [<Data>] / or = | Optionally update current location and redisplay the current location |
| [<Data>] ^ or - | Optionally update current location and display the previous location  |
| [<Data>] .      | Optionally update current location and exit Memory Modify             |

With the exception of the carriage return, the sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message will be issued and the contents of the current memory location shall be redisplayed.

### Restrictions

While there are no restrictions regarding the use of the MM command, caution should be used when modifying target memory while user code is running. Accidentally modifying target memory containing program code could lead to program run away.

### Example

```
>mm 800
0800 00 <CR>
0801 F0 FF
0802 00 ^
0801 FF <CR>
0802 00 <CR>
0803 08 55 /
0803 55 .
>
```

## MMW - Modify memory words in hexadecimal format

### Command Line Format

MMW <Address> [<data>]

### Parameter Description

<Address>    A 16-bit hexadecimal number  
<data>        A 16-bit hexadecimal number

### Command Description

The memory modify word command allows the contents of memory to be examined and/or modified as 16-bit hexadecimal data. If the 16-bit data parameter is present on the command line, the word at memory location at <Address> is replaced with <data>. If not, D-Bug12 will enter the interactive memory modify mode. In the interactive mode, each byte is displayed on a separate line following the data's address. Once the memory modify command has been entered, several sub-commands are used for the modification and verification of memory contents. These sub-commands have the following format:

[<Data>] <CR>        Optionally update current location and display the next location  
[<Data>] / or =        Optionally update current location and redisplay the current location  
[<Data>] ^ or -        Optionally update current location and display the previous location  
[<Data>] .            Optionally update current location and exit Memory Modify

With the exception of the carriage return, the sub-command must be separated from any entered data with at least one space character. If an invalid sub-command character is entered, an appropriate error message will be issued and the contents of the current memory location shall be redisplayed.

If the <Address> parameter corresponds to an even byte address, values read from and/or written to memory will be performed as aligned word accesses. This guarantees data coherency for peripherals that require a single access to their 16-bit registers.

### Restrictions

While there are no restrictions regarding the use of the MMW command, caution should be used when modifying target memory while user code is running. Accidentally modifying target memory containing program code could lead to program run away.

### Example

```
>mmw 800
0800 00F0 <CR>
0802 0008 AA55 /
0804 843F ^
0802 AA55 <CR>
0804 843F <CR>
0806 C000 .
>
```

## MOVE - Move a Block of Memory

### Command Line Format

MOVE <StartAddress> <EndAddress> <DestAddress>

### Parameter Description

<StartAddress> A 16-bit hexadecimal number  
<EndAddress> A 16-bit hexadecimal number  
<DestAddress> A 16-bit hexadecimal number

### Command Description

The MOVE command is used to move a block of memory from one location to another a byte at a time. The number of bytes moved is one more than the <EndAddress> - <StartAddress>. The block of memory created beginning at the destination address may overlap the memory block defined by the <StartAddress> and <EndAddress>.

One of the uses of the MOVE command might be to copy a program from RAM into EEPROM memory.

### Restrictions

A minimum of one byte may be moved if the <StartAddress> is equal to the <EndAddress>. The maximum number of bytes that may be moved is  $2^{16} - 1$ . In addition, caution should be exercised when moving target memory while user code is running. Accidentally modifying target memory containing program code could lead to program run away.

### Example

```
>move 800 8ff 1000  
>
```

## **NOBR - Remove one/all user breakpoints**

### **Command Line Format**

NOBR [<Address> <Address>...]

### **Parameter Description**

<Address>A 16-bit hexadecimal number

### **Command Description**

The NOBR command is used to remove one or more of previously entered breakpoints. If the NOBR command is entered without any arguments, all user breakpoints are removed from the breakpoint table.

### **Restrictions**

When operating in the POD mode, breakpoints may not be removed with the NOBR command when the 'R>' prompt is being displayed.

### **Example**

```
>br 800 810 820 830  
Breakpoints: 0800 0810 0820 0830
```

```
>nobr 810 820  
Breakpoints: 0800 0830
```

```
>nobr  
All Breakpoints Removed
```

```
>
```

## RD - Display CPU12 Register Contents

### Command Line Format

RD

### Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

### Command Description

The Register Display command is used to display the CPU12's registers. The registers are displayed in the same format used when a breakpoint is encountered.

### Restrictions

When operating in the POD mode, the CPU registers may not be displayed when the 'R>' prompt is being displayed.

### Example

```
S>rd
```

```
PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
C028  4000  0000  0000      00:00      1101 0000
C028  790016      CLR  $0016
S>
```

## REGBASE - Specify the Register base address

### Command Line Format

REGBASE <Address>

### Parameter Description

<Address> A 16-bit hexadecimal number

### Command Description

Because D-Bug12 supports the ability to transparently program the on-chip EEPROM of the target MCU, it must know the base address of the I/O registers. Because user code may change the register block's base address by writing to the INITRG register, D-Bug12 must be informed of the register block's base address for transparent EEPROM writes to occur. The REGBASE command is used to specify the base address of the target processor's on-chip registers.

The REGBASE command does not check to ensure that the <Address> parameter is a valid base address for the selected M68HC12 family member. If an improper register base address is provided, automatic programming of the on-chip EEPROM will not operate properly.

When operating in EVB mode, the default register base address is specified in the Customization Data variables `CustomData.IOBase`. This value is used by the startup code to remap the I/O registers. The REGBASE command may not be used to relocate the I/O registers.

---

---

**Note:** The REGBASE command does not automatically modify the INITRG register. It is the responsibility of the user to ensure that the INITRG register is modified either manually or through the execution of user code.

---

---

### Restrictions

The REGBASE command may not be used when D-Bug12 is operated in the EVB mode.

### Example

S>device

```
Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $0000
```

S>regbase 2000

```
Device: 912B32
EEPROM: $0D00 - $0FFF
Flash: $8000 - $FFFF
RAM: $0800 - $0BFF
I/O Regs: $2000
```

## **RESET - Reset the target system MCU**

### **Command Line Format**

RESET

### **Parameter Description**

No parameters are required. Any parameters on the command line will be ignored.

### **Command Description**

The RESET command is used to reset the target system processor when operating in D-Bug12's POD mode. The target processor's reset pin is held low for approximately 2 mS. When the reset line is released, BDM commands are sent to the target processor to place it in active background mode. With the exception of the program counter (PC), the target processor's registers are initialized with the same values used for the registers when operating in EVB mode. The PC is initialized with the contents of the target processor's reset vector, memory locations \$FFFE and \$FFFF

### **Restrictions**

When operating in the 'EVB' mode, the RESET command cannot be used. If the RESET command is entered while in 'EVB' mode, an error message will be displayed and command execution will be terminated.

### **Example**

```
S>reset
Target Processor Has Been Reset
S>q 4000
R>reset
Target Processor Has Been Reset
S>
```

## RM - Interactively Modify CPU12 Register Contents

### Command Line Format

RM

### Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

### Command Description

The register modify command is used to examine and/or modify the contents of the CPU12's registers in an interactive manner. As each register and its contents is displayed, D-Bug12 allows the user to enter a new value for the register in hexadecimal. If modification of the displayed register is not desired, entering a carriage return causes the next CPU12 register and its contents to be displayed on the next line. When the last of the CPU12's registers has been examined and/or modified, the RM command will redisplay the first register giving the user an opportunity to make additional modifications to the CPU12's register contents. Typing a period (.) as the first non space character on the line will exit the interactive mode of the register modify command and return to the D-Bug12 prompt.

The registers are displayed in the following order, one register per line: PC, SP, X, Y, A, B, CCR.

### Restrictions

When operating in the POD mode, the CPU registers may not be modified when the 'R>' prompt is being displayed.

### Example

```
>RM
PC=0206 200
SP=03FF <CR>
X=1000 1004
Y=3700 <CR>
A=27 <CR>
B=FF <CR>
CCR=D0 D1
PC=0200 .
>
```

## STOP - Stop Execution of user code in the target MCU

### Command Line Format

STOP

### Parameter Description

No parameters are required. Any parameters on the command line are ignored.

### Command Description

When operating in D-Bug12's POD mode, the STOP command is used to halt target program execution and place the target processor in active background debug mode.

### Restrictions

When operating in the 'EVB' mode, the STOP command cannot be used. If the STOP command is entered while in 'EVB' mode, an error message is displayed and command execution will be terminated.

### Example

```
S>asm 4000
4000  CCFFFF          LDD  #$FFFF
4003  830001          SUBD #$0001
4006  26FB           BNE  $4003
4008  20F6           BRA  $4000
400A  00             BGND
                                     >_
S>g 4000
R>stop
Target Processor Has Been Stopped

  PC   SP   X   Y   D = A:B   CCR = SXHI NZVC
4003  0A00  0000  0000   37:3F   1101 0000
4003  830001          SUBD  #$0001
S>
```

## T - Trace (Execute) CPU12 Instruction(s)

### Command Line Format

T [<Count>]

### Parameter Description

<Count> An 8-bit decimal number in the range 1..255

### Command Description

The Trace command is used to execute one or more user program instructions beginning at the current Program Counter (PC) location. As each program instruction is executed, the CPU12's register contents are displayed and the *next* instruction to be executed is displayed. A single instruction may be executed by entering the trace command followed immediately by a carriage return.

### Restrictions

When operating in 'EVB' mode, all branch instructions (Bcc, LBcc, BRSET, BRCLR, DBEQ/NE, IBEQ/NE, TBEQ/NE) containing an offset that branches back to the instruction opcode will NOT execute because of the method used to execute a single instruction. The monitor will appear to become 'stuck' at the branch instruction and will not execute the instruction even if the condition for the branch instruction is satisfied. This limitation can be overcome by using the GT (GoTill) command to set a temporary breakpoint at the instruction following the branch instruction.

This restriction **DOES NOT** apply when using D-Bug12 on a target system in the POD mode.

### Example

```
>t
PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
0803  09FE  057C  0000      10:00      1001 0000
0803  830001      SUBD  #$0001
>t 2
PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
0806  09FE  057C  0000      0F:FF      1001 0000
0806  26FB      BNE   $0803

PC      SP      X      Y      D = A:B      CCR = SXHI NZVC
0803  09FE  057C  0000      0F:FF      1001 0000
0803  830001      SUBD  #$0001

>
```

## UPLOAD - Display Memory In S-Record Format

### Command Line Format

UPLOAD <StartAddress> <EndAddress>

### Parameter Description

<StartAddress> A 16-bit hexadecimal number  
<EndAddress> A 16-bit hexadecimal number

### Command Description

The UPLOAD command is used to display the contents of memory in Motorola S-Record format. In addition to displaying the specified range of memory, the UPLOAD command also outputs an S9 end-of-file record. The output of this command may be captured by the users terminal program and saved to a disk file.

### Restrictions

None.

### Example

```
>upload 400 5ff
S123040000F0000843FC0000F50F379F37BF43FCF50F27FA757F177AFA047504177AFA21C5
S123042037B500FF37FAFB0437B5400037FAFB061735FB0037B500C137FAFA003715379C01
S1230440F50F379D37BC012C37BD400085009A003C023D02377C0140B6EE7A0F400037B583
S1230460000337FAFA4C37FAFA5037FAFA5437B5502037FAFA4E37B5302037FAFA5237B58A
S1230480682037FAFA5637BD014037BC000095008A003C023D02377D0172B6EE37BD017259
S12304A037BC020095008A003C023D02377D018EB6EE27F937B0F50F379C37BC00CE27F901
S12304C000FC27F9104C27F90E68378000BE0A0D442D42756731362056312E3033202D20E3
S12304E04465627567204D6F6E69746F7220466F7220546865204D363848433136204661ED
S12305006D696C790A0D2843292031393932204D6F746F726F6C612053656D69636F6E64BD
S12305207563746F7220496E632E000037B5FF0237FAFA4837B578B037FAFA4A7A0F005E52
S1230540000000000000000000020002040208020C021000000000000000000000000002144F
S1230560002187A0F3BAC7A0F3BBC7A0F11E87A0F62
S12305803C727A0F3C847A0F3C967A0F3CA8F50F379C379D379E27FAF50F379F37BF43FCE8
S12305A07501177A4054173540523604361C27F90088B0D637BC01BC360227F70A0D3E00A9
S12305C04500B70427F936BC3C01B0F027F7277537BC400017BC405027F936CC780DB60477
S12305E027F936A0274A27F77803B6FEB03A7808B6162776B7DE3730000127F93686752002
S9030000FC
>
```

## USEHBR - Use EVB/Target Hardware Breakpoints

### Command Line Format

USEHBR

### Parameter Description

No parameters are required. Any parameters on the command line will be ignored.

### Command Description

Entering the USEHBR command causes D-Bug12 to use the hardware breakpoint capability of the MC68HC912B32 on the EVB, in EVB mode, or the breakpoint capability of the target microcontroller in POD mode. Using hardware breakpoints allows two, program only breakpoints to be set in Flash or other non-volatile memory. To revert to the 10 software breakpoints, the EVB reset button must be pressed.

Using the hardware breakpoints of the MC68HC912B32 when operating in EVB mode allows the developer to trace through the user accessible routines in D-Bug12 that are located in the on-chip Flash memory. Further, when debugging small programs located in the MC68HC912B32's on-chip EEPROM, it is recommended that hardware breakpoints be used. Using hardware breakpoints will prevent D-Bug12 from repeatedly erasing and reprogramming the on-chip EEPROM when using the T, G or GT commands or when setting breakpoints.

Entering the USEHBR command will reinitialize the breakpoint table causing any previously set breakpoints to be removed from the breakpoint table.

### Restrictions

When operating in POD mode, D-Bug12 has no way of detecting whether the target processor contains a hardware breakpoint module. If the USEHBR command is issued when running in POD mode and the target processor does not contain a hardware breakpoint module, D-Bug12 breakpoint capability will be lost. In addition, unpredictable behavior of the target may occur if breakpoints are set with the BR command.

When operating in the POD mode, the USEHBR command cannot be issued when the 'R>' prompt is being displayed indicating that the target system is running a user program.

### Example

```
S>usehbr
Using Hardware Breakpoints
S>br 810 835
Breakpoints: 0810 0835
S>br 957
Breakpoint Table Full
S>
```

## VERF - Compare S-Record File To The Contents of Memory

### Command Line Format

VERF [<AddressOffset>]

### Parameter Description

<AddressOffset>            A 16-bit hexadecimal number

### Command Description

The VERF command is used to compare the data contained in an S-Record object file to the contents of target memory. The address offset, if supplied, is added to the load address of each S-Record before an S-Record's data bytes are compared to the contents of memory. Providing an address offset other than zero allows the S-Record's object code or data to be compared against memory other than that for which the S-Record was assembled. An offset greater than \$FFFF may only be used with devices that support more than 64K bytes of memory.

---

---

**Note:** Please refer to the section titled "FLOAD, LOAD and VERIFY S-Record Format" at the end of this document for a complete description of the S-Record Format utilized by this command for M68HC12 devices supporting more than 64K bytes of memory.

---

---

---

---

**Note:** The VERF command DOES NOT require the software handshaking protocol used by the FLOAD command. Before using the VERF command, make sure that the terminal's line-at-a-time handshaking is disabled.

---

---

During the verification process, an ASCII asterisk character (\*) shall be sent to the control console for each ten S-Records that are successfully verified. When an S-Record file has been successfully verified, D-Bug12 will issue its prompt.

The VERF command is terminated when D-Bug12 receives an 'S8' or 'S9' end of file record. If the object file being loaded does not contain an 'S8' or 'S9' record, D-Bug12 will not return its prompt and will continue to wait for the end of file record. Pressing a system Reset will return D-Bug12 to its command line prompt.

### Restrictions

When operating in POD mode, the VERF command will not support standard baud rates above 19,200 when the target MCU is operating at an E-clock frequency of 8.0 MHz (16.0 MHz crystal). This restriction is due to the overhead involved in the implementation of the Custom Serial Protocol required by the Single Wire Background Debug pin. For target MCUs operating at lower E-clock frequencies, the maximum baud rate that can be used with the VERF command will be proportionally lower.

## Example

```
>verf 1000  
*****  
>
```

## <RegisterName> - Modify a CPU12 Register Value

### Command Line Format

<RegisterName> <RegisterValue>

### Parameter Description

Where <RegisterName> is one of the following CPU12 register names:

| <u>Register Name</u> | <u>Description</u>      | <u>Legal Range</u> |
|----------------------|-------------------------|--------------------|
| PC                   | Program Counter         | \$0..\$FFFF        |
| SP                   | Stack Pointer           | \$0..\$FFFF        |
| X                    | X-Index Register        | \$0..\$FFFF        |
| Y                    | Y-Index Register        | \$0..\$FFFF        |
| A                    | A Accumulator           | \$0..\$FF          |
| B                    | B Accumulator           | \$0..\$FF          |
| D                    | D Accumulator (A:B)     | \$0..\$FFFF        |
| CCR                  | Condition Code Register | \$0..\$FF          |

Each of the fields in the CCR may be modified by using the following field Names:

| <u>CCR Bit Name</u> | <u>Description</u>            | <u>Legal Range</u> |
|---------------------|-------------------------------|--------------------|
| S                   | STOP Enable                   | 0..1               |
| H                   | Half Carry                    | 0..1               |
| N                   | Negative Flag                 | 0..1               |
| Z                   | Zero Flag                     | 0..1               |
| V                   | Twos Complement Overflow Flag | 0..1               |
| C                   | Carry Flag                    | 0..1               |
| IM                  | IRQ Interrupt Mask            | 0..1               |
| XM                  | XIRQ Interrupt Mask           | 0..1               |

### Command Description

This set of “commands” uses the CPU12 register names as individual commands to allow changing the contents of individual registers. Each register name or Condition Code Register bit name is entered on the command line followed by a space, then followed by the new register or bit value. The successful alteration of a CPU register or CCR will cause the CPU12’s register contents to be displayed.

### Restrictions

None.

If a value outside the range for a given register is entered, an error message is displayed and command execution is terminated leaving the register contents unaltered.

## Example

>pc 700e

| PC   | SP   | X    | Y    | D = A:B | CCR = SXHI NZVC |
|------|------|------|------|---------|-----------------|
| 700E | 0A00 | 7315 | 7D62 | 47:44   | 1001 0000       |

|      |        |  |     |        |  |
|------|--------|--|-----|--------|--|
| 700E | 790016 |  | CLR | \$0016 |  |
|------|--------|--|-----|--------|--|

>x 1000

| PC   | SP   | X    | Y    | D = A:B | CCR = SXHI NZVC |
|------|------|------|------|---------|-----------------|
| 700E | 0A00 | 1000 | 7D62 | 47:44   | 1001 0000       |

|      |        |  |     |        |  |
|------|--------|--|-----|--------|--|
| 700E | 790016 |  | CLR | \$0016 |  |
|------|--------|--|-----|--------|--|

>c 1

| PC   | SP   | X    | Y    | D = A:B | CCR = SXHI NZVC |
|------|------|------|------|---------|-----------------|
| 700E | 0A00 | 1000 | 7D62 | 47:44   | 1001 0001       |

|      |        |  |     |        |  |
|------|--------|--|-----|--------|--|
| 700E | 790016 |  | CLR | \$0016 |  |
|------|--------|--|-----|--------|--|

>z 1

| PC   | SP   | X    | Y    | D = A:B | CCR = SXHI NZVC |
|------|------|------|------|---------|-----------------|
| 700E | 0A00 | 1000 | 7D62 | 47:44   | 1001 0101       |

|      |        |  |     |        |  |
|------|--------|--|-----|--------|--|
| 700E | 790016 |  | CLR | \$0016 |  |
|------|--------|--|-----|--------|--|

>d adf7

| PC   | SP   | X    | Y    | D = A:B | CCR = SXHI NZVC |
|------|------|------|------|---------|-----------------|
| 700E | 0A00 | 1000 | 7D62 | AD:F7   | 1001 0101       |

|      |        |  |     |        |  |
|------|--------|--|-----|--------|--|
| 700E | 790016 |  | CLR | \$0016 |  |
|------|--------|--|-----|--------|--|

>

## **FLOAD, LOAD and VERIFY S-Record Format**

The S-Record object file format was designed to allow binary object code and/or data to be represented in printable ASCII hexadecimal format to allow easy transportation between computer systems and development tools. For M68HC12 family members supporting less than 64K bytes of address space, S1 records, which contain a 16-bit address, are sufficient to specify the location in the device's memory space where code and/or data are to be loaded. The load address contained in the S1 record generally corresponds directly to the address of on-chip or off-chip memory device. For M68HC12 devices that support an address space greater than 64K bytes, S1 records are not sufficient.

Because the M68HC12 family is a 16-bit microcontroller with a 16-bit program counter, it cannot directly address a total of more than 64K bytes of memory. To enable the M68HC12 family to address more than 64K bytes of program memory, a paging mechanism was designed into the architecture. Program memory space expansion provides a window of 16K byte pages that are located from \$8000 through \$BFFF. An 8-bit paging register, called the PPAGE register, provides access to a maximum of 256, 16K byte pages or 4 megabytes of program memory. While there may never be any devices that contain this much on-chip memory, the MC68HC812A4 is capable of addressing this much external memory. In addition, the MC68HC912DA/DG128 contains 128K bytes of on-chip Flash EEPROM that must be programmed by various development tools.

While many high-level debuggers are capable of directly loading linked, absolute binary object files into a target system's memory, D-Bug12 does not have that capability. D-Bug12 is only capable of loading object files that are represented in the S-Record format. As mentioned previously, because S1 records only contain a 16-bit address, they are inadequate to specify a load address for a memory space greater than 64K bytes. S2 records, which contain a 24-bit address, were originally defined for loading object files into the memory space of the M68000 family. It would seem that S2 records would provide the necessary load address information required for M68HC12 object files. However, as those who are familiar with the M68000 family know, the M68000 has a linear (non-paged) address space. Thus, development tools, such as non-volatile memory device programmers, interpret the 24-bit address as a simple linear address when placing program data into memory devices.

Because the M68HC12 memory space expansion is based on 16k byte pages, there is not a direct one-to-one mapping of the 24-bit linear address contained in the S2 record to the 16K byte program memory expansion space. Instead of defining a new S-Record type or utilizing an existing S-Record type in a non-standard manner, the D-Bug12 FLOAD, LOAD and VERIFY commands view M68HC12 memory blocks larger than 64K bytes as a simple linear array of memory that begins at an address of \$00000. This is the same format in which S-Records would need to be presented to a stand alone non-volatile memory device programmer. For example, from the view of the FLOAD, LOAD and VERIFY commands, the 128K bytes of on-chip Flash would have addresses from \$00000 through \$1FFFF. The mapping between the linear address contained in the S-Record and the 16K byte page viewable through the window at addresses \$8000 through \$BFFF is shown in Figure 1 below.

The generation of S-Records that meet these requirements is the responsibility of the linker and/or S-Record generation utility provided by the compiler/assembler vendor. Cosmic Software's linker and S-Record generation utility is capable of producing properly formatted S-Records that can be used by D-Bug12. Other vendor's tools may or may not possess this capability.

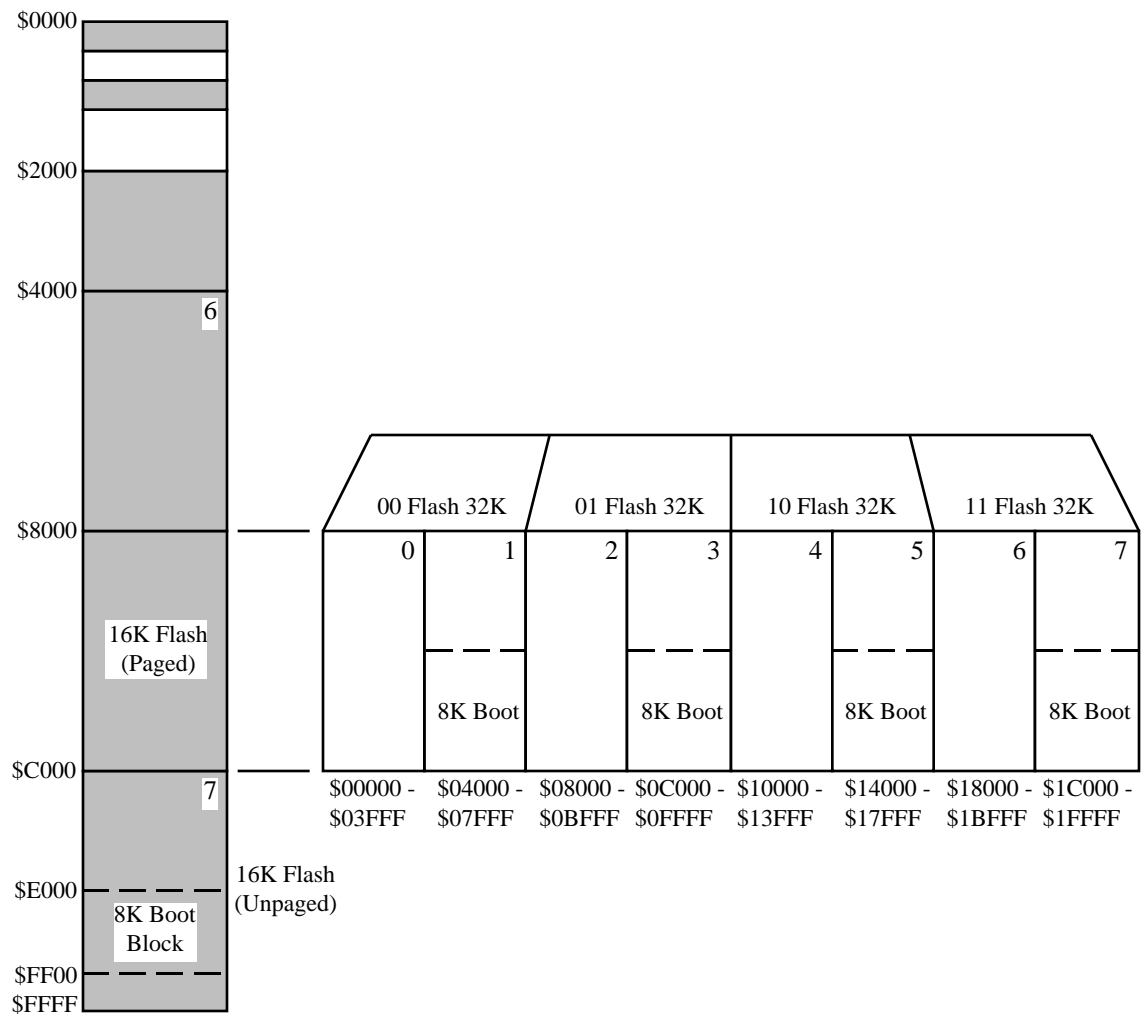


Figure 1, MC68HC912DA/DG128 Flash Memory Paging