GMXBUG-09 User's Guide

Version 2

GIMIX, Inc.
1337 W. 37th Place
Chicago, Ill. 60609
(312) 927-5510

Revision A 2/26/82

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO,ILLINOIS 60609 • (312)927-5510 • TWX910-221-4055

GMXBUG-09 Version 2

GMXBUG-09 Version 2 is an enhanced version of the original GMXBUG-09 V1. Every effort has been made to maintain hardware and software compatibility with the earlier versions. Existing version 1 VIDEO PROMs and/or BOOT PROMs can be used with GMXBUG-09 V2. Any existing software, written for earlier versions of GMXBUG-09 should work without modification. The following commands and functions have been added or changed:

COMMAND	FUNCTION
B,P,K	Several minor changes have been made to the breakpoint system. See the manual for details.
D,F	These commands now dump between a starting and ending address or line by line from a starting address.
G	See the manual.
O	Boots OS-9™ in systems that have software selection between OS-9™ and FLEX™ or switches to an alternate monitor ROM. This eliminates the need to boot FLEX™ first, when switching to OS-9™.
R	The condition codes are now printed in binary rather than hexadecimal for easier interpretation.
W	Jumps to the FLEX™ warmstart address ($CD03).
NMI/ABORT	GMXBUG-09 now vectors NMIs to a trap routine that prints a register dump and allows the option of continuing execution of the interrupted program or entering the debugger. This allows the front panel ABORT switch to be used to halt runaway programs and as a debugging aid.

GMXBUG-09 User's Guide - Version 2

TABLE OF CONTENTS

Chapter 1 The GMXBUG-09 system monitor

1-1 Basic features and requirements
1-2 SBUG-E compatibility
GMXBUG-09 reset routine
1-5 GMXBUG-09 quick reference page
1-6 GMXBUG-09 memory map
1-7 Video driver storage

Chapter 2 The GMXBUG-09 debugger

2-1 Debugger operation
Entering a parameter
2-2 Aborting a command
2-3 Command descriptions
2-7 Memory mode

Chapter 3 The GMXBUG-09 breakpoint system

3-1 What is a breakpoint?
GMXBUG-09 breakpoint table
Setting a breakpoint
3-2 Listing the breakpoints
Execution of a breakpoint
3-3 Removing a breakpoint
Resuming execution after a breakpoint
3-4 Use of the Non-Maskable Interrupt (NMI) as a breakpo

Chapter 4 GMXBUG-09 utility subroutines

4-1 SBUG-E compatible routines
4-3 GMXBUG-09 specific routines

Chapter 5 GMXBUG-09 video drivers

5-1 Video board summary
Outline of video operation
5-2 Pointers and registers
5-3 Video board initialization routine
Control characters
5-4 Scrolling
Escape sequences
5-5 Escape code descriptions
5-7 Escape code quick reference page

The GMXBUG-09 system monitor

Basic features and requirements

The GMXBUG-09 monitor is a support program that provides basic operating and debugging capabilities for a 6809-based microcomputer system. GMXBUG-09 provides console I/O, serial or parallel printer output, a powerful interactive debugging package, and a package of utility routines.

GMXBUG-09 contains console I/O drivers for a serial terminal, however a Gimix 80x24 video board and parallel keyboard may be substituted. This is done by adding an extra ROM with the video board driver software to the CPU card. All GMXBUG-09 console I/O is vectored through locations in the scratchpad memory. On reset, GMXBUG-09 checks for the presence of a video ROM, and if it is present the vectors are switched to it. Other devices may be substituted for the Gimix video board and parallel keyboard, if drivers are provided in an appropriate ROM.

The minimum system required by GMXBUG-09 consists of the following:

GIMIX 6809 CPU board
2K of PROM at $F800 containing GMXBUG-09
1K of memory at $E400 as the scratchpad
34 bytes of memory at $DFC0 for SBUG-E compatibility
ACIA at $E004
serial terminal connected to the ACIA

In a video-board based system, the ACIA and serial terminal are replaced by the following:

1K driver PROM at $F400
PIA at $E040 - port 4 - A side connected to a parallel ASCII keyboard
GIMIX 80x24 video board at $E800 - registers at $E3F0

GMXBUG-09 also includes drivers for both serial and parallel printers. Default printer printer type is serial; printer type selection is made by flags in the scratchpad and can be changed under program control. The parallel and serial drivers do not interfere with each other: both types of printer may be connected to the same system and switched between under program control. (See the Note under PRINT on page 4-7) To use these drivers the following is required:

PIA at $E040 - port 4 - side B connected to parallel printer (other half of keyboard PIA)
- or -
ACIA at $E000 - port 0 - connected to serial printer (side A when using a two port serial card and sixteen bytes per I/O slot)

SBUG-E compatibility

With the 6809, monitor compatibility is much less important than it was with the 6800. However, Technical Systems Consultants' FLEX (tm) operating system is in wide use on 6809 systems, so to insure compatibility with FLEX, GMXBUG-09 includes subroutines and subroutine entrance vectors equivalent to those in SWTPc's SBUG-E (tm) monitor. Also, the interrupt vectors (except NMI, which is not defined under SBUG-E) are in the same locations as under SBUG-E, and the dynamic address translation image is set up the same as SBUG-E on reset. An unmodified FLEX system disk from TSC will boot up and run under GMXBUG-09 when using a terminal based system. However, to get full use of all the features of a Gimix system, we recommend the use of Gimix FLEX, produced under license from TSC and customized by us for Gimix hardware.

GMXBUG-09 will not run on the SWTPc CPU board because there is no scratchpad memory, but the GIMIX CPU card can be set up to run SBUG-E correctly.

GMXBUG-09 reset routine

When the system is turned on or the RESET button is pushed, the reset routine, pointed to by the reset vector at addresses $FFFE - $FFFF is executed. This routine initializes the vectors, drivers and flags as follows:

(1) The Dynamic Address Translator is initialized to output the same four bits to the four highest address lines of the bus as are output by the CPU for all 16 tasks (i.e., bank 0 will respond to address 0, bank 1 to address 1, etc.), and the Task Select Register is set to task 0. Version 2.0S sets up the SWTPc-type DAT the same way, and both versions create a DAT image at $DFD0-DFDF equivalent to S-BUG. The code for initializing the DAT and TSR is located in the top page of memory ($FF00-FFFF) as this is the only memory definitely available on power-up.

Once the DAT and TSR are set, execution continues at the "coldstart" address; that is, at the address pointed to by the first subroutine vector at $F800. This address is used as a transfer point by many programs; it restarts the system in all aspects except the DAT and TSR. Only reset and power-on cause the DAT and TSR to be reinitialized.

(2) The coldstart routine sets the processor stack pointer to $E7FF (the top of the CPU board scratchpad memory). Then the console is initialized. To determine whether a serial terminal or video board is to be used as the console device, location $F400 is checked. If the contents of $F400 are $12 (a NOP instruction), this indicates that a PROM with the video drivers is installed, and control is transferred to $F400 to initialize the video board hardware and parallel keyboard interface. If $F400 has any other value, then the default routine to initialize a serial terminal

interface is performed instead. Both the default routine and the video routine assume a serial printer interface at $E000, and initialize it. Parallel printer interface initialization is only done if the user selects parallel printer type with the I command. The I/O vectors at $E400 are also initialized by both routines: the default routines set them to point to serial I/O routines, and the video routine sets them to point to video/parallel I/O routines in the driver PROM. The table below gives the vector functions and default settings. See chapter 5, "The GMXBUG-09 Video Driver", for the video settings.

function	setting
$E400 - get character	$FEE3 (SERIN)
$E402 - get character without waiting	$FEE8 (SERKEY)
$E404 - test input status	$FF03 (SERTST)
$E406 - character output	$FF0C (SEROUT)

The terminal interface ACIA and serial printer ACIA are initialized with a control mask of $13 (8 data bits, 2 stop bits, no parity, interrupt disabled, /16 clock). Also the address of the terminal's ACIA is stored at $DFE0 for SBUG-E compatibility.

(3) After the console is set up execution continues at $F864 (the video initialization routine rejoins the mainline there). At this point the printer vector at $E408 is set to $FF1F (HARDC), the GMXBUG printer driver, and the NMI vector at $E40A is set to $FF80 (NMITRP), the GMXBUG NMI handler. The line buffer is set at $E700, and the area $E420-$E439 is zeroed out. The breakpoint table is not affected. The vectors for SWI2, SWI3, IRQ9 and FIRQ (located at $DFC2-DFC8 for SBUG-E compatibility) are set to $FF94 (an RTI). The SWI vector at $DFCA is set to $FEAE (BRKPT), the GMXBUG breakpoint handler. The SBUG-E variable SVCORG ($DFCC) is set to $FFFF to indicate no SVC table set up (this is purely for compatibility; GMXBUG has no facilities for handling SBUG-E-type SVC calls).

This table gives the exact effect of reset on GMXBUG's flags and pointers.

```
Stack pointer - $E7FF (top of the scratchpad)
LINBUF  -  $E700 (start of the last page)
NMIVEC  -  points to NMITRP
SWIVEC  -  points to BRKPT
UPCASE  -  0 (normal)
WAIT    -  0 (not suspended by CTL-S)
NULLS   -  0
PRTFLG  -  0 (serial printer)
DEST    -  0 (screen)
```

4) The setting of these flags and pointers completes GMXBUG-09's coldstart routine. The warmstart routine which follows is the normal entry to GMXBUG-09, and does not affect any of the above variables. The warmstart routine displays a message identifying GMXBUG-09, and pushes a set of registers on the stack. The PC is not pushed itself; rather the warmstart address is pushed in its place, so that an RTI would cause execution to start at the warmstart. Also the MMODE flag is cleared, turning off memory

mode.

5) Finally the mainline of the debugger is entered; the prompt "GMX:" is displayed, and GMXBUG-09 waits for the user to input a character to select a command.

GMXBUG-09 debugger commands

```
A - hexadecimal Arithmetic
B - set Breakpoint
C - Checksum
D - Dump hex & ASCII
F - Formatted disassembly dump
G - Go on from breakpoint
H - Hex locate
I - Initialize
   D - output Destination
   N - Null count
   P - Printer type
   R - Report current settings
   U - Universal reset
J - JSR to user program
K - Kill breakpoint
M - Memory examine/change  **
     (cr) - leave memory mode
     (sp) - change & move forward 1
       +  - move forward 1 byte
       -  - move backward 1 byte
       =  - change byte
       2  - binary display
       "  - continuous ASCII input
       $  - continuous hex input
O - switch to Other CPU ROM
P - Print breakpoint info
R - Register examine/change
T - Test memory
U - jump to $F000 (FLEX boot)
W - jump to FLEX Warmstart ($CD03)
X - transfer memory
Z - Zap (fill) memory
```

** All other GMXBUG-09 commands are available in this mode.

Function keys

```
ESC    abort entry              control-S   output stop/start
CR     complete entry
```

GMXBUG-09 entry points

SBUG-E compatible calls

```
F800 Cold start
F802 Warm start
F804 Get character (no echo)
F806 Get character (echo)
F808 Test input status
F80A Output character
F80C Output a string
F80E Output CR/LF
F810 Output string with leading CR/LF
F812 Load real address
F814 branch to warmstart
```

GMXBUG calls

```
F816 Output a space
F818 Output registers
F81A Output ACCA in hex
F81C Get character w/o waiting
F81E Get upper case character
F820 Get hex number from consol
F822 Convert ACCA to ASCII hex
F824 Convert string to binary
F826 Input a line
F828 Search a table
F82A Move a block of memory
F82C Output to hardcopy
```

Each entry holds the address of a subroutine, which is called with an indirect subroutine jump, like this:

```
            JSR    [$F81A] Output ACCA in hex
```

* There is no entry at $F814, instead there is a branch to the GMXBUG-09 warm start address. This is to maintain compatibility with FLEX.

GMXBUG-09 memory map
---

address	name	bytes	usage
-- in main memory			
-- SBUG-E compatible storage			
DFC2	SW3VEC	2	SW13 vector
DFC4	SW2VEC	2	SWI2 vector
DFC6	FIRVEC	2	FIRQ vector
DFC8	IRQVEC	2	IRQ vector
DFCA	SWIVEC	2	SWI vector
DFCC	SVCORG	2	SWTPE compatibility
-- in scratchpad memory on the CPU card			
-- GMXBUG-09 storage			
E400	INVEC	2	console input vector
E402	KEYVEC	2	fast input vector
E404	TSTVEC	2	console test vector
E406	OUTVEC	2	console output vector
E408	PRTVEC	2	hardcopy output vector
E40A	NMIVEC	2	NMI vector
E40C	LINBUF	2	address of line buffer
E40E	ENDLIN	2	address of last buffer byte
E410	BRKTBL	16	breakpoint data
E420	MMODE	1	memory mode flag
E421	UPCASE	1	force upper case input flag
E422	BEGIN1	2	pointer storage
E424	END1	2	
E426	BEGIN2	2	
E42A	SUBTOT	1	arithmetic work storage
E42F	WAIT	1	etl-S pending flag
E430	MEMPTR	2	pointer for memory mode
E432	POINTR	2	extra pointer
E434	NULLS	1	null count
E435	PRTFLG	1	printer type flag: 0=serial 1=parallel
E436	DEST	1	output control flag: 0=console 1=printer also
E437	MATCH1	1	match bytes for hex locate,command
E438	MATCH2	1	
E439	MATCH3	1	

-- video driver storage

E43A	CURSOR	2	cursor address
E43C	ESCVEC	2	pointer to ESC processing routine
E43E	WRAP	2	positions left on line
E43F	CHRSLT	1	character slot in use
E440	MOVMOD	1	positioning mode: 0=absolute 1=relative
E441	ESCSTR	1	byte count in escape string
E442	DURATN	2	bell tone duration in cycles
E444	PERIOD	1	bell tone cycle period
E445	ABSX	1	absolute screen X position
E446	ABSY	1	absolute screen Y position
E447	VIDADD	2	address of video board RAM
E448	VIDEND	2	address of end of video RAM
E449	LOWLFT	2	address of lower left of video RAM
E44A	REGADD	2	address of video registers

The GMXBUG-09 debugger

Debugger operation

The mainline of GMXBUG-09 is the debugger. This program constitutes a minimal operating system, and also a powerful tool for analysing programs at the machine code level.

Upon entry to the debugger, GMXBUG-09 displays a copyright message and a prompt, "GMX:". This indicates that the user may enter a command. To enter a command, type the letter for that command, followed by any parameters that may be required. For example, to perform the "register examine/change" command, type "R" on the keyboard.

Entering a numeric parameter

Some commands require parameters such as addresses or byte values; this information is normally entered in hexadecimal. Numeric parameter values are entered through the GMXBUG-09 line input utility and converted to binary with the GMXBUG-09 hex-to-binary conversion utility. To enter a parameter value, type the required information (such as a breakpoint number) then type carriage return (CR) to terminate entry. The carriage return will not be echoed.

The value received by GMXBUG-09 will be the value of the four rightmost digits to the left of the first non-hexadecimal character in the entry. If there are less than four digits, the entry is zero-filled from the left. If no digits are typed, the value is 0000. If more than four digits are typed, all but the rightmost four are ignored. During entry, the user may correct a typing mistake with the BACKSPACE key. See the description of the line input utility for the use of this key. Also available during entry is the ESC key; typing ESC aborts the command.

```
Examples:  1234(CR)     enters 1234
           111234(CR)   enters 1234
           +1234(CR)    enters 0000 ("+" stops conversion)
           22 55        enters 0022 (" " stops conversion)
           QWERK(CR)    enters 0000 ("IQ" stops conversion)
           1E(CR)       enters 001E (zero filled)
           123(ESC)     aborts command
```

Certain commands require two parameters. Both parameters are entered on the same line. When the user types CR to complete entry of the first parameter, GMXBUG-09 does not echo the CR. Instead, a space is displayed to prompt entry of the second parameter. No user-typed spaces are necessary. When the CR to complete the second parameter is entered, then the command is performed.

At any time during the typing of either parameter, ESC may be typed to abort the command.

Note: Once the CR to complete parameter 1 is typed and entry of parameter 2 has begun, backspace will not move the cursor past the beginning of parameter 2. The first parameter can not be corrected once the CR is typed. If parameter 1 is in error, the command must be aborted and restarted.

```
Examples:  1234(CR) 4567(CR)  enters 1234 and 4567
           1(CR) 23(CR)       enters 0001 and 0023
           1A2B(CR) 9E3(ESC)  aborts command
```

Certain commands required 3 parameters; these commands work the same as the commands requiring 2 parameters except that a third parameter is entered after parameter 2 is completed.

The H (hex locate) command requires 3, 4 or 5 parameters: the bounds of the memory area to search and 1, 2, or 3 byte values to search for. If only a CR is entered for the fourth parameter, then the fifth parameter will be skipped.

The I command has a different format. The parameter it requires is a single character "subcommand" with no CR. The subcommands may require parameters of their own; see the command description.

Aborting a command

When ESC is typed during the entry of a parameter or other data, or if an illegal parameter value is entered, the command is aborted. All data entered is ignored, and GMXBUG-09 starts a new line and displays a prompt. ESC does NOT cause GMXBUG-09 to leave the memory mode.

GMXBUG-09 command descriptions

A hexadecimal Arithmetic

Format: A param1 param2

The 16-bit sum and difference of the two parameters (param1+param2 and param1-param2) are calculated and displayed on the same line.

Examples:
```
GMX:A 1123 203E 3161 F0E5
GMX:A 1234 5 1239 122F
GMX:A 8089 80B2 013B FFD7
```

B set Breakpoint

Format: B param1 param2

Sets a breakpoint in memory: the first parameter is the breakpoint number, and the second is the address to set it at. The breakpoint number must be 0, 1, 2, or 3, or GMXBUG will display a "?" and abort the command. The address may have any value but 0000, which is used in the table to indicate a breakpoint not set. For further details, see chapter 3,, "The GMXBUG-09 Breakpoint System".

Examples:
```
GMX:B 0 1234      set breakpoint 0 at 1234
GMX:B 2 12        set breakpoint 2 at 0012
GMX:B 4?          illegal breakpoint - "?" displayed
                  by GMXBUG-09 and command ignored
GMX:B 2 0         illegal address - command ignored
```

WARNING: Use of this command in the address range $FF00 - $FFFF may change the DAT or TSR, causing unpredictable results.

C Checksum memory

Format: C param1 param2

The 24-bit sum of the 8-bit values contained in the bytes from param1 to param2 inclusive is calculated, and the result is displayed on the same line. This is useful for checking large blocks of code or data for accidental changes. When param2 is less then param1 the command will terminate after only one byte.

Note: This command causes a read of every address in the specified range. Therefore the range should not include I/O interfaces or certain GIMIX CPU options, such as the 6840 timer or the 9511/9512 arithmetic processor.

Examples:
```
GMX:C 1234 1FFF 02EF52
GMX:C 0 FFF 031DE1
```

D Dump memory as hexadecimal and ASCII

Format: D param1 param2

Causes the memory from param1 to param2 to be displayed as hex values and ASCII characters. 16 bytes are displayed on each line; if param2 is not the last byte of a line then extra bytes are displayed to fill out the line. If param2 is 0, the command stops at the end of each line and waits for the user to type a character; if this character is ESC the command is terminated, else the next line is displayed. If param2 is not 0, then the display runs continuously till param2 is reached or exceeded. (If param2 is less than param1, then the command displays one line and stops.)

During continuous display the command checks the console at the end of each line; if the user types ctl-S, output will be suspended till another ctl-S is typed. If the user types ESC, the command will terminate at the end of the line.

Notes: any value not a printing ASCII character will be displayed as a "." in the ASCII section of the dump. A header line giving the last address digit for bytes in each column is displayed above the first line and the starting address of each line is displayed at the left.

```
Examples: GMX:D 1000 11FF    dumps 1000-11FF inclusive
          GMX:D 1000 1032    dumps 1000 - 103F inclusive
          GMX:D 1000         dumps one line at a time from 1000
          GMX:D 1000 FFF     dumps 1000 - 100F inclusive
```

F disassembly Format dump

Format: F param1 param2

Displays a "disassembly" dump of the memory from param1 to param2 inclusive. Each line consists of the starting address and from 1 to 5 bytes in hexadecimal. The number of bytes is the number needed to complete whatever 6809 machine instruction begins at that address. (Undefined instructions generally are treated as equivalent to similar defined instructions.) This command functions identically to the D command with regard to starting, terminating, and pausing.

```
Example: GMX:F F400 F409
         F400 12
         F401 10 CE E7 FF
         F405 86 E4
         F407 1F 8B
         F409 CE E4 00
```

G Go on from breakpoint

Format: G (CR)

After G is typed, the command waits for another character, which is not echoed. A "CR" causes an RTI to be performed, and execution of a program halted by breakpoint or NMI will resume. If any other character is typed, the command has no effect. The G command can also be used to start execution of a program with a known machine state. See page 3-3, "Resuming execution after a breakpoint".

Examples: GMX:G (CR) exit from breakpoint
GMX:G (ESC) no effect

H Hex locate

Format: H param1 param2 param3 (param4) (param5)

Searches through memory from param1 to param2 inclusive for 1, 2 or 3 bytes equal to the param3, param4 and param5 values. The address of each match found is displayed on a separate line.

The fourth and fifth parameters are optional. If the user wants to search for only two bytes, he enters their values as param3 and param4, then enters a CR with no preceding characters for param5. This tells GMXBUG that only two bytes are to be matched. If the user wants to search for only one byte, he enters its value as param3, then enters CR with no characters for param4. GMXBUG will then skip entry of the unneeded fifth parameter.

Note: in the examples below the CR at the end of every parameter is shown for clarity.

Examples:

```
GMX:H 1000(CR) 10FF(CR) CE(CR) E4(CR) 00(CR) (3-byte search)
1033
108F
GMX:H 1000(CR) 10FF(CR) 31(CR) 08(CR) (CR)   (2-byte search)
1036
GMX:H 10FF(CR) 11FF(CR) 3F(CR) (CR)          (1-byte search)
1174
GMX:H 10FF(CR) 11FF(CR) 31(CR) 08(CR) (CR)   (2-byte search,
                                              no match)
GMX:
```

I Initialize system parameters

Format: I subcommand (param)

Initializes a selected system parameter. The subcommand selects a system value to set. The parameter is the value to

set it to. The values initialized by the subcommands, and the parameter required by each are given in the table below:

subcmd	param	function
D	B or S	set output device to both printer and screen (B) or to screen only (S). Any input other than B is treated as S.
N	00-FF	set printer null count to param
P	P or S	set printer type to parallel (P) or serial (S). Any input other than P is treated as S. P causes a reset of Port 4 PIA side B, but S does not affect Port 0 ACIA.
R	none	reports current system settings
U	none	reset the entire system (jump to COLDS)

```
Examples: GMX:I D B          set output device to screen & printer
          GMX:I D S          set output device to screen only
          GMX:I P P          parallel printer
          GMX:I P S          serial printer
          GMX:I N 12         set printer nulls to hexadecimal 12
          GMX:I R            report settings
          NL P D
          00 S S
          GMX:I U            reset system
```

J JSR to user program

Format: J paraml

Leaves a return address on the stack and starts execution at paraml. The return address is the address of a branch to the beginning of the debugger. If the routine at paraml ends with an RTS, then control will automatically return to the debugger at the end of the routine.

Note: the Direct Page Register is set to 00 by the J command before the jump to paraml is made.

Example: GMX:J 1234

K Kill breakpoint

Format: K param

Removes a breakpoint; the parameter is the breakpoint number, and must be 0, 1, 2, or 3. If it is greater than 3 a ? is displayed and the command is aborted. If X is entered in place of a number, then all four breakpoints are removed. For further explanation, see chapter 3, "The GMXBUG-09 Breakpoint

System".

```
Examples: GMX:K 0     remove breakpoint 0
          GMX:K 4?    invalid breakpoint number
          GMX:K X     remove all four breakpoints
```

M Memory examine and change

Format: M paraml

Enters memory mode with the memory pointer (MEMPTR) set to paraml.

Memory mode

When GMXBUG-09 is in memory mode, the normal prompt GMX: is not displayed at the start of each line. Instead, the value of the memory pointer and the contents of the byte at that address are displayed.

```
Example: 1234 56       where the memory pointer value is 1234
                       and the value of that byte is 56
```

Also, eight additional commands are available, as well as all the regular commands.

```
Example: 1234 56 C 1000 11FF 001E12
         1234 56
```

The eight additional commands are:

\+ Increment memory pointer

Adds 1 to the address in MEMPTR

```
Example: 1234 56 +
         1235 6A
```

\- Decrement memory pointer

Subtracts 1 from the address in MEMPTR

```
Example: 1235 6A -
         1234 56
```

2 Binary display

Displays the contents of the current byte in binary. Useful for examining interface control registers.

```
Example: 1234 56 2 0101 0110
         1234 56
```

(sp) param Change memory and increment pointer

Stores the parameter value in the current byte, then increments MEMPTR.

```
Example: 1234 56   7F
         1235 6A
```

= param Change memory

Stores the parameter value in the current byte, but does not change MEMPTR.

```
Example: 1234 56 = CE
         1234 CE
```

(CR) Leave memory mode

Returns to normal operation

```
Example: 1234 56 (CR)
         GMX:
```

" param Enter ASCII characters

The parameter is a string of ASCII characters which are stored in successive bytes starting at the current byte. Each character is stored as soon as it is typed and MEMPTR is also incremented immediately. The string is terminated by CR, which is not stored. Any other character, including control characters, is stored exactly as typed. Characters are echoed as typed; thus typing backspace would erase the previous character, but would be stored after it in memory. If the first character typed is CR, the command has no effect.

```
Example: 1234 56 " ABCDQ
         1239 07
```

$ param1 param2... continuous hex input

The parameters are hex values which are stored in successive bytes starting at the current byte. As soon each parameter is entered it is stored and MEMPTR is incremented. The command terminates only when the user types ESC. All the normal rules for parameter entry apply; the value stored is the lower 8 bits (last 2 digits) of the value entered.

```
Example: 1234 56 $ 12 34
         1236 9F $ 01 3    (stored as 01 03)
         1237 30
```

O switch to Other ROM

Format: O (dummy param)

Sets bit 5 of the Task Select Register to select the alternate CPU board ROM in place of the GMXBUG-09 ROM and jump to its reset address. This alternate ROM is normally OS-9 (TM of Microware Systems Corp.). Since the switch removes the GMXBUG-09 ROM from the bus, the switching must be done by a routine in RAM; the switch routine is therefore copied to $E500 in the CPU board scratchpad, and executed there. The dummy parameter is required only to give the user an opportunity to abort the command with ESC. To complete the command normally the user should just type CR; any characters before the CR are ignored.

Note: the O command sets the Direct Page Register to 00 before the transfer to the alternate monitor is made.

```
Examples: O (CR)     switches to alternate ROM
          O (ESC)    no effect
```

P Print breakpoints

Format: P

Displays the information in the breakpoint table. For each breakpoint, its number, the address at which it is set, and the saved byte value are displayed. An address of 0000 indicates a breakpoint not set. For futher details, see chapter 3. "The GMXBUG-09 Breakpoint System".

```
Example: GMX:P
         00 2343 33
         01 0000 00
         02 0000 00
         03 0000 00
```

where breakpoint 0 is set at 2343, the saved value for that byte is 33, and breakpoints 1. 2, and 3 are not set.

R Register examine and change

```
Format: R
         EFHI NZVC A  B  DP  X    Y    U    PC   SP
         0000 0000 00 00 00 0000 0000 0000 0000 0000
        Register to change? param1 param2
```

The set of CPU registers on the top of the stack are displayed and the user may change any one of them if he wishes. This set of registers will be used if a G command is executed. (The Stack Pointer value is the address of the top saved register.) The registers are saved when GMXBUG-09 is entered through a breakpoint or through the NMI trap routine. Also, when GMXBUG is warmstarted, a set of registers is pushed on

the stack.

Param1 is a single character identifying the register to be changed. The possible character and the registers they indicate are: C=CC, A=A, B=B, D=DP, X=X, Y=Y, U=U, P=PC, S=SP. If the user does not want to change any register, typing CR will terminate the command without effect. If any character other than CR or a register ID is typed, a ? will be displayed and the command is terminated.

If the user selects a register to change, a hexadecimal value must be entered for param2. This value will replace the present stacked register value. All the normal conditions of parameter entry apply. If the register is an 8-bit register, only the leftmost two digits are used. Note that while CC is displayed in binary, a replacement value for CC must be entered in hexadecimal. If the register changed is the Stack Pointer, then the program cannot return to the debugger mainline by an RTS, so a direct jump to the warmstart address is performed instead.

```
Example: GMX:R
          EFHI NZVC A  B  DP  X    Y    U    PC   SP
          0101 0110 22 0E 00 A349 65C0 7DF8 23DE C065
         Register to change?  X 1234
```

sets the stacked X register to 1234.

T Test memory

Format: T param1 param2

Performs cyclic check on memory from param1 through param2, inclusive. This test is repeated through 256 passes to provide complete convergence testing. After each pass, a "#" is displayed, and CR/LF is output every 64th pass to prevent line overflow.

Note: this test destroys the contents of the tested area. Therefore, do not test the area used by GMXBUG-09 in the scratchpad RAM, or the area used for stack storage, or the test routine will bomb.

```
Examples: GMX:T 1234 5678
          ################################################
          ################################################
          ################################################

          (no errors found)

          GMX:T 1234 5678
          ###############
          2345 0010 0000

          (bit 5 in byte 2345 is defective)
```

WARNING: Use of this command in the address range $FF00 - $FFFF may change the DAT or TSR, causing unpredictable results.

U jump to User program at $F000

Format: GMX:U (dummy param)

Jumps to address $F000. The dummy parameter is required to give the user an opportunity to abort the command, as with the O command. The memory at $F000 is normally a ROM; for instance, Gimix supplies a disk boot program in ROM which can be installed at $F000 and invoked by this command.

Note: the U command sets the Direct Page Register to 00 before the jump to $F000 is made.

Example: GMX: U (CR)
EXECUTING GIMIX DMA BOOTSTRAP

W Warmstart FLEX

Format: W (dummy param)

Jumps to address $CD03. The dummy parameter has the same function as with the O and U commands. $CD03 is the restart address for the Gimix FLEX disk operating system; this command is included to provide a handy way of returning to FLEX after a reset or MON command.

Note: the W command sets the Direct Page Register to 00 before the jump to $CD03 is made.

Example: GMX:W (CR)

X block transfer

Format: X param1 param2 param3

Transfers the contents of memory from param1 through param2 inclusive to memory starting at param3. If the source area and destination area overlap, the move will be done in reverse order if needed.

Examples: GMX:X 1000 1FFF 2000
GMX:X 3400 36FF 3600 (reverse move)

WARNING: Use of this command in the address range $FF00 - $FFFF may change the DAT or TSR, causing unpredictable results

Z Zap memory

Format: Z param1 param2 param3

Fills memory from param1 to param2 inclusive with param3 (which is considered only as an 8 bit value).

```
Examples: GMX:Z 100 1FF 41       (fills area with $41)
          GMX:Z 1000 23FF 1001   (fills area with 01)
          GMX:Z 380 451 (CR)     (fills area with 00)
```

WARNING: Use of this command in the address range $FF00 - $FFFF may change the DAT or TSR, causing unpredictable results.

The GMXBUG-09 breakpoint system

One of GMXBUG-09's most powerful facilities is its breakpoint system. This system provides the user with four independent breakpoints for use in program debugging and testing.

What is a breakpoint?

A breakpoint is a special instruction, available on most computers, which is provided as a debugging aid. The effect of a breakpoint is to halt execution of the current program, save the machine state (register contents, flags, etc.) and transfer control to a debugger program. The programmer can then examine and modify the saved machine state, and resume execution if desired, all transparent to the current program.

The Motorola 6809 instruction set has the software interrupt instruction (SWI - $3F) as its breakpoint instruction. SWI saves the registers and begins execution at the address in $FFFA. In GMXBUG-09, this is the address of an indirect jump instruction which jumps to the address in SWIVEC ($DFCA). This is initialized to point the GMXBUG-09 breakpoint handler, but the user may change this to substitute any other program, such as his own custom breakpoint handler.

GMXBUG-09 breakpoint table

GMXBUG-09 processes multiple breakpoints by means of a table of breakpoint data in the scratchpad (BRKTBL - $E410). The address of each breakpoint set with the B command is kept here, and also the byte which the breakpoint replaced. A breakpoint is identified by matching its address to one of those in the table. An address entry of 0000 indicates a breakpoint which is not set; for this reason setting a breakpoint at location 0000 is not permitted.

Setting a breakpoint

To set a breakpoint, use the B command in the debugger. Type B, then enter the number of the breakpoint (0-3), then enter the address where the breakpoint is to be set. GMXBUG-09 then puts the address and the byte of code from that address in the tables, and puts an SWI ($3F) at that address. Note that when the breakpoint is executed, the replaced byte is in the table, and has not been executed. This allows examination of the registers just before an instruction is performed.

A breakpoint which is already set may be set again without first clearing it with the K command; GMXBUG-09 will automatically

clear the first setting before making the new setting.

Listing the breakpoints

The P command may be used to display the status of the four system breakpoints. When P is entered, GMXBUG-09 prints the number of each breakpoint, the address at which it is set, and the byte saved from that address. Breakpoints not set have an address and saved byte of 0.

```
Example: GMX:P
         00 1234 26
         01 5562 7E
         02 0000 00
         03 178F 9B
```

When the system is first turned on, the contents of memory, including the breakpoint table, are purely random. GMXBUG-09 does not initialize the breakpoint table on power-up or reset; this is because it is sometimes necessary to reset the system during debugging of an assembly language program. If the breakpoints were reinitialized by reset, then any breakpoints set would be lost on reset. This has been avoided, but unfortunately it means the breakpoint table is in a random state on power-up, and will contain meaningless addresses and saved bytes. Before using the breakpoints, the user should clear the table with the K command. See the caution in the "Removing a breakpoint section".

Execution of a breakpoint

When a breakpoint is encountered during the execution of a program, it has the following effects:

1) All the registers are saved on the machine stack, and the I and F bits are set in the condition code register.

2) Execution begins at the address in location $FFD8. This is the address of an indirect jump vectored through $DFCA (SWIVEC). Thus execution will begin at the address in SWIVEC. At reset SWIVEC is set to point to BRKPT, the start of the GMXBUG-09 breakpoint handler. If SWIVEC is not changed, a breakpoint transfers control to BRKPT.

3) The saved PC is adjusted so that after the breakpoint is removed and execution is resumed execution will start at the breakpoint address, not the next byte.

4) The saved PC is compared to the address entries in the breakpoint table. If one of them matches, the entry number is displayed with a preceding # and a register dump. If not, the number is set to $FF (255) and the data is displayed.

5) Control is transferred to the debugger at the warmstart entry.

Examples: #01

```
 EFHI NZVC A  B  DP  X    Y    U    PC   SP
 1101 0100 41 01 00 1FFF 0300 0000 3132 E7DE
GMX:

(breakpoint 1 at address 3132)

#FF
 EFHI NZVC A  B  DP  X    Y    U    PC   SP
 1101 0100 01 A4 00 23C0 51CC 0310 1FE9 C069
GMX:

(unknown breakpoint at 1FE9)
```

Removing a breakpoint

Breakpoints can be removed with the K command of the debugger, by typing K and then entering the number of the breakpoint to be removed. If the byte at the address in the table for that breakpoint is an SWI ($3F), it will be replaced with the saved byte from the table. The breakpoint address and saved byte are both set to 0 to indicate that the breakpoint is not set.

The K command can be used to clear the entire breakpoint table at once. If the user enters X instead of a breakpoint number, then GMXBUG-09 removes all four breakpoints and clears all four entries, exactly as if the K command had been entered four times with the numbers 0, 1, 2, and 3. This is especially useful at power-up when the table contains garbage.

CAUTION: It is possible for the random data in the breakpoint table at power-up to point to an address that will contain a $3F after the program being debugged is loaded. To prevent possible alteration of valid data, the breakpoint table should be cleared, using K X, before the program is loaded.

If the user has inserted a breakpoint "by hand", i.e. with the memory examine and change commands of the debugger, the breakpoint must be removed the same way. GMXBUG-09 has no way of automatically removing such breakpoints.

Resuming execution after a breakpoint

The G command of the debugger can be used to resume execution after a program has been halted by a breakpoint or NMI (Non-Maskable Interrupt). Typing G and then CR to confirm the command causes GMXBUG-09 to execute an RTI instruction, pulling all registers off the stack, and causing execution to resume at the PC value previously stacked.

If execution was halted by a breakpoint, then the user must remove the breakpoint before resuming execution. If this is not done, the first instruction executed will be that same breakpoint, halting execution immediately.

Note: the debugger pushes a set of registers on the stack during its warmstart routine, with a PC value of the warmstart address. Thus, using the G command when there has been no breakpoint or NMI will cause a warmstart of the debugger. However, the user may modify these stacked register values with the R command; this allows the user to start execution of a routine not only at a specified address, but with all condition codes and other register values specified. Example: by setting the PC to 1044 and the DPR to 18, and then using the G command, the user would begin execution at 1044, as if he had typed J 1044, but would also have the DPR set to a value other than 0, which the J command sets it to.

Use of the Non-Maskable Interrupt (NMI) as a breakpoint

The NMI is one of three system interrupts available on the 6809 processor. Unlike the FIRQ and IRQ, it cannot be turned off, hence its name. On many 6809 systems, including the Gimix system and SWTPc S/09, the NMI line is connected to a front panel switch labeled "ABORT". Pressing this switch causes an NMI to the 6809 processor. The processor then saves the machine state and begins execution at the address in $FFFC. In GMXBUG-09, this is the address of an indirect jump instruction which jumps to the address in NMIVEC ($E40A). This is initialized to point the GMXBUG-09 NMI trap routine, but the user may change this to substitute any other program, such as his own NMI handler, or a program performing real-time I/O with peripherals connected to the NMI line.

If NMIVEC is not changed, then the NMI trap routine is executed. This routine displays the register values saved by the NMI and the message "NMI: restart?", and then inputs a character from the console. If the user types Y in response to the message, an RTI will be executed immediately, and execution will resume unaffected (except for the screen, which of course is disturbed). If the user types any other character, then the debugger is warmstarted, but without the extra register stacking. Therefore the user can use the debugger, and later use the G command to restart the program halted by the NMI.

Note: in video-based systems, pressing "ABORT" while the system is waiting for keyboard input has a harmless but confusing side effect. The cursor is turned on at the start of the keyboard input subroutine, and off at the end. If this subroutine is interrupted, the interrupt handler can safely do console input, but the cursor will be turned off, and will remain off after the interrupted keyboard input routine is restarted with an RTI. However, this has no effect on the input itself, and the cursor will appear normally the next time the keyboard input routine is called. The user should just resume typing, even if the cursor is not present.

Another note: NMIs may be nested safely (up to a point), with no effect other than the cursor disappearance mentioned above. If a defective NMI switch or user error causes multiple NMIS, all that is needed is to type Y once for each excess NMI to straighten things out. The cursor may or may not appear for each input, depending on whether the NMI trap routine was interrupted before the cursor was turned on, but nothing harmful will happen.

GMXBUG-09 utility subroutines

GMXBUG-09 includes 22 utility subroutines which perform basic system and I/O functions, and are set up to be easily used by external programs. A table of the addresses of these subroutines is set up at $F800; thus a utility subroutine may be called by a JSR indirect to the appropriate table entry.

```
Example:   JSR    [$F804]  input a character from the console
```

GMXBUG-09 has utility subroutines for character I/O through the console, and also for hexadecimal I/O, hex/binary conversion, printer output, and some useful system functions. Also, GMXBUG-09 cold and warm start are included in the table for convenience, although these are not subroutine calls. For compatibility with SBUG-E, the first ten entries in the address table point to subroutines which are equivalent to SBUG-E subroutines. All the subroutines end by returning to the calling program, except cold and warm start.

The descriptions which follow give the table address, registers affected and entering and exiting parameters for each of the utility subroutines. Any registers not named as affected will be preserved intact, except the CC register, which is always affected. However only the N, Z, V, and C bits are affected; the I and F interrupt mask bits are never altered. For the console I/O subroutines, the descriptions are of the serial terminal interface; for the video/parallel version, see chapter 5, "The GMXBUG-09 Video Drivers".

Note: any subroutine which is vectored through a location in the scratchpad memory may be replaced by changing the vector to point to a substitute subroutine. However the substitute subroutine must match the original subroutine in registers affected and parameters for correct operation.

SBUG-E compatible utilities

The following utilities are functionally identical to the utilities in SBUG.

MONITR $F800 GMXBUG-09 cold start

Registers affected: none

Parameters: none

Performs the GMXBUG-09 reset routine and enters the debugger. For complete details, see chapter 1, "The GMXBUG-09 reset routine".

NXTCMD $F802 debugger warm start

Registers affected: none

Parameters: none

Enters the debugger without performing any system initialization. MMODE is cleared, and a set of registers is pushed on the stack, with PC set to the actual warmstart address.

INCH $F804 console input

Registers affected: ACCA

Parameters: ACCA contains input on return.

Inputs one character from the console keyboard. The character is not echoed. If UPCASE ($E421) is non-zero and the character is a-z, it is converted to upper case. This subroutine is vectored through location $E400 (INVEC).

INCHE $F806 console input with echo

Registers affected: ACCA

Parameters: ACCA contains input on return.

Calls INCH to input one character from the console keyboard, and calls OUTCH to display it on the console screen, unless the character is ESC ($1B).

INCHEK F808 test for console input

Registers affected: none

Parameters: Z indicates presence or absence of input on return

Tests the RXDRF bit of the console ACIA. Returns Z=1 if RxDRF=0 (no input is present), returns Z=0 If RxDRF=1 (input is present). Does not affect the ACIA. This subroutine is vectored through location $E402.

OUTCH $F80A console output

Registers affected: none

Parameters: ACCA contains output on entry.

Displays a character on the console screen. If DEST ($E436) is non-zero, the character will also be printed on the hardcopy device. This subroutine is vectored through

location $E406, but the printer output is NOT vectored through location $E408.

PDATA $F80C output string

Registers affected: X

Parameters: X points to string to be outputted on entry.

Outputs the bytes pointed to by X to the console with successive calls to OUTCH. The bytes in the string may contain any value except 04, which terminates output. On return, X contains the address of the byte containing 04 plus 1.

PCRLF $F80E output carriage return and line feed

Registers affected: none

Parameters: none

Outputs a carriage return and a line feed to the console through OUTCH.

PSTRNG $F810 output string with CR/LF

Registers affected: X

Parameters: X points to string to be outputted on entry.

Calls PCRLF to output a carriage return and line feed to the console, then PDATA to output the string.

LRA $F812 load real address

Registers affected: ACCA

Parameters: X contains a 16-bit address on entry; ACCA contains extended address bits on exit.

Converts a 16-bit address to a 20-bit extended address in conformance with SBUG-E. If X is $E000 or greater, $0F is returned in ACCA, else 00 is returned.

GMXBUG-09 specific routines

In addition to the 10 "standard" subroutines listed above, GMXBUG-09 includes 12 additional utility subroutines. These provide additional I/O and other useful functions.

PSPACE $F816 output a space

Registers affected: none

Parameters: none

Outputs a space character ($20) to the console through OUTCH.

PREGS $F818 output registers

Registers affected: none (including CC)

Parameters: none

Outputs the current contents of the CPU registers in hexadecimal, with an identifying header line. The PC value printed is the address of the instruction after the call to PREGS; the SP value is the value after all the registers have been pushed on the stack. All registers are preserved by this subroutine, including the condition codes.

PBYTE $F81A output ACCA in hexadecimal

Registers affected: none

Parameters: ACCA contains byte to be outputted on entry.

Outputs the 8-bit value in ACCA as two hexadecimal digits. Calls BINHEX to do the conversion and OUTCH to do the output.

INKEY $F81C console keyboard test and input

Registers affected: ACCA

Parameters: Z indicates presence or absence of input and ACCA contains input if any on return.

Calls SERTST (the serial keyboard test) to check for input present. If an input byte is found, returns Z=0 and the byte in ACCA; else returns Z=1. If UPCASE ($E421) is non-zero and the character is a-z, the character is converted to upper case.This subroutine is vectored through $E402, but the call to the test subroutine is not

vectored through location $E404.

INCAPS $F81E console input with echo and capitalization

Registers affected: ACCA

Parameters: ACCA contains input on return.

Makes UPCASE ($E421) non-zero to force uppercase input, calls INECHO to input and echo a character, then clears UPCASE. Note that UPCASE is left as zero, regardless of its previous value.

HEXIN $F820 hexadecimal input from console

Registers affected: ACCA, ACCB

Parameters: ACCA and ACCB contain the input on return; V indicates whether the input was completed or aborted.

Sets UPCASE for uppercase only, calls LINEIN to get a line of input, clears UPCASE, checks for input terminated with ESC and returns with V=1 is so, else calls HEXBIN to convert the input to binary and returns with the more significant byte of the input value in ACCA, the less significant in ACCB, and V=0. Note that UPCASE is left as zero, regardless of its previous value.

BINHEX $F822 convert binary to hexadecimal ASCII

Registers affected: ACCA, ACCB

Parameters: ACCA contains value to convert on entry; ACCA and ACCB contain resulting hex digits on return.

Converts the more significant four bits of ACCA to a hexadecimal ASCII digit in ACCA, and the less significant four bits to a digit in ACCB.

HEXBIN $F824 convert hexadecimal ASCII to binary

Registers affected: ACCA, ACCB

Parameters: X contains start of hex ASCII string on entry; ACCA and ACCB contain resulting value on return.

Converts the string of hexadecimal ASCII digits pointed to by X to a 16-bit binary value in ACCA and ACCB. The string is terminated by the first character other than 0-9 or A-F. If the string contains more than four digits, all but the last four are ignored. If there are less than four digits, the value is zero filled from the left. If

the very first character is not a digit, the value will be 0000. The more significant byte is returned in ACCA, and the less significant in ACCB.

LINEIN $F826 input a line of characters from the console

Registers affected: ACCA

Parameters: LINBUF ($E40C) points to a buffer area on entry; ACCA contains the terminating character and ENDLIN ($E40E) points to the terminator on return.

Inputs up to 79 characters from the console through INCH. The characters are stored in successive bytes starting at the address in LINBUF. BACKSPACE both erases the last character on the screen and moves the storage pointer back one byte, unles s the buffer is empty, in which case it is ignored. All other characters except CR and ESC are echoed and stored unless the buffer is full, in which case they are ignored. CR and ESC terminate the subroutine; when one of them is typed, it is stored but not echoed and is returned in ACCA.

TBSRCH $F828 search command table

Registers affected: X

Parameters: ACCA contains the ID byte and X contains the table address on entry; if a match is found X points to the data part of the matching entry and Z=1 on return, else X points one byte past the end of the table and Z=0.

Searches a table of the form ID DATA ID DATA ...ID DATA 00 where the IDs are bytes to be matched and the DATAs are 16-bit values. The ID byte in ACCA is compared to each of the ID bytes in the table. If a match is found, then X points to the first byte of the data on return. If no match is found, the search continues till an ID byte of 00 is encountered, and the subroutine is terminated with Z=0.

MOVBLK $F82A move a block of memory

Registers affected: none

Parameters: BEGIN1, END1, and BEGIN2 ($E424-E427) contain the starting and ending addresses of the block to be moved and the address it is to be moved to on entry.

Copies the bytes from BEGIN1 to END1 inclusive to the area beginning at BEGIN2. If the source and destination areas overlap such that part of the source area would be overwritten before it was read, then the pointers are adjusted and the move is done in reverse order.

PRINT $F82C output a character to the hardcopy device

Registers affected: none

Parameters: ACCA contains the character to be output on entry.

Outputs the character in ACCA to the system hardcopy device. If PRTFLG ($E435) is non-zero, then the output is to the parallel printer interface; if PRTFLG is zero, then the serial interface is used. If the character is a CR, then NULL characters are outputted to allow the printer to return its carriage. The number of nulls to be outputted is stored in NULCNT ($E434). It is initialized to 0, but may be changed by the user with the I command or by a program. This subroutine is vectored through location $E408.

Note: The GMXBUG-09 parallel printer interface is not initialized unless the parallel printer is selected with the I command. Before switching between parallel and serial printers under program control, the parallel interface must first be initialized using the I command or by the program itself.