TMCBUG2 User's Manual

Ver. 1.0, Apr. 20, 1996
by Yasuo Arai
KEK, National Laboratory for High Energy Physics
1-1 Oho, Tsukuba, Ibaraki 305, JAPAN
araiy@kekvax.kek.jp

TMCBUG2 is a monitor program for the 32 channel TMC-VME module, and helps the user to debug, load program, execute program and supply useful routines. TMCBUG2 is a modified version of TMCBUG which was developed for 64ch TMC-VME module, and the TMCBUG has been developed based on the DSPBUG which is originaly developed by Motolola Inc. TMCBUG2 is tailored to the TMC-VME module. Several commands and functions which is not necesary to the module are removed to reduce the program size, and several new commands specific to the module are added. This manual describes the command and functions of the TMCBUG2. Some sentences for the commands which are exist in DSPBUG are copied from "DSPBUG V2 Manual" prepared by Motolola Inc..

CONTENTS
(Some of the chapters are missing since they describe non-supporting functions in TMCBUG2)

1 TMCBUG2

1.1 Introduction

TMCBUG2 is a standalone debug monitor for the TMC-VME module. TMCBUG2 is written in DSP56002 assembly code and allows the user to develop applications on a real time system.

1.2 Features

The TMCBUG2 commands enable the user to:

1. Load/verify programs.
2. Modify/display memory.
3. Modify/display user's registers.
4. Execute/debug programs.
5. Set/remove breakpoints
6. Trace program execution.
7. Download/upload user data.
8. Call many monitor utility subroutines.

The TMCBUG2 monitor is designed to allow a user to modify the system to meet specific hardware/software requirements:

- I/O device independence for the host and terminal ports.

1.3 Terminology

Throughout this document, several terms have special significance in describing the operation and use of TMCBUG2:

1. Terminal Port - The RS232C port in the front panel and connected to the SCI port of DSP. A user is attached to this port.The terminal port controls input and displays output associated with TMCBUG2.

2. Host Port - Eight bit VME registers which are connected to the Host Interface of the DSP. A VME host computer is attached to this port. The host port transfers programs or data between the DSP56002 memory and a host computer.

3. Host Interface - This refers to the 8 bit parallel I/O interface on the DSP56002. TMCBUG2 may be controlled through the DSP56002 host interface.

5. Numbers prefixed with a dollar sign are hexidecimal constants.

6. <CR> refers to a carriage return, ASCII code $0D.

7. <SP> refers to a space, ASCII code $20.

8. DELETE refers to a delete or RUBOUT, ASCII code $7F.

9. Special control characters are generated by holding down the CONTROL key (sometimes shown as CTRL) on the keyboard and simultaneously pressing a key. These sequences are indicated by Ctrl-? where ? is the key code. For example, Ctrl-S indicates depressing the control key and simultaneously depressing the S key.

1.4 Operating Environment

1.4.1 Serial Control

TMCBUG2 requires an I/O device to enter TMCBUG2 commands and display output. The debug switch on the board configure itself at power up to operate on one of two devices:

1. The SCI on the DSP56002. The SCI is initialized to 9600 baud.

2. The DSP56002 host interface.

The VME-TMC is controlled by a serial type of device (terminal or IBM-PC) or by memory mapped I/O when using the host interface.

1.4.2 Powerup

TMCBUG2 boots from one external 8Kx8 EPROM located at $4000-$FFFF in program memory. Located in the low bytes, the EPROMs hold the TMCBUG2, an intelligent loader and several utility programs. A minimum of 3.5K of RAM (24 bits wide) is required at $1300 to hold TMCBUG2 for execution.
When the boot takes place, a loader transfers TMCBUG2 from the EPROMs to the program RAMs and begins execution at starting address in the TMCBUG2. TheTMCBUG2 initialization message displays the highest memory address required by TMCBUG2. Any memory $80 - $1300 and beyond the end address is available to the user.

1.4.3 Memory Access

TMCBUG2 accesses external X and P memory spaces with no wait states, and Y and IO spaces with one wait states. Since TMCBUG2 has no knowledge of the access time of the user supplied external memory, TMCBUG2 accesses all external memory with the maximum number of wait states. The bus control register (BCR) in the user register display is not used until a context switch is performed by using a JSR or GO command.

When TMCBUG2 is switching its context between the monitor mode and the user mode, the user's BCR is restored which may prevent TMCBUG2 access to its own memory to complete the context switch.

1.4.4 Stack Usage

The software interrupt (SWI) is reserved for TMCBUG2 to implement breakpoints. When the GO command begins program execution, the user must reserve one location on the DSP56002 system stack for breakpoints. The breakpoint causes a software interrupt and pushes the return address and status register on the stack.

When using the JSR command, the user must reserve at least one stack location since TMCBUG2 pushes on a return address for the user's RTS from the subroutine. If the user has breakpoints in the subroutine, two stack locations must be reserved for TMCBUG2 since a SWI also pushes another address on the stack.

1.5 Interrupt Vectors Used By TMCBUG2

TMCBUG2 uses three interrupt vectors:

1. Stack Error - The stack error vector (P:$0002) is initialized by TMCBUG2 at powerup pointing to a TMCBUG2 routine. The routine reports the stack error and returns control to TMCBUG2. If a stack error occurs, the stack pointer is saved. Examination of the stack pointer indicates if the error is due to pushing a full stack or pulling an empty stack. The user may change the stack error vector at any time.
If the stack pointer has the stack error (SE) bit set, the user must reset the stack pointer (using the REG command) before TMCBUG2 allows execution of the user program.

2. Trace - The trace vector (P:$0004) is used by TMCBUG2 to implement tracing and transfer of control commands. This vector is constantly being changed by TMCBUG2 depending on the current user state. Since this vector is constantly being changed, it should not be used by the user.

3. SWI - The software interrupt vector (P:$0006) is used by TMCBUG2 to implement breakpoints. If breakpoints are not being used, the user may write this vector and perform a SWI operation.
If the user does not write a new SWI vector but executes a SWI, the registers are saved and control returns to TMCBUG2. TMCBUG2 notifies the user that an unexpected SWI has been encountered.This may be used to save the current user state and return to TMCBUG2 for debugging.

Other interrupt vectors are in program RAM and may be used at any time by the user program.

1.6 Command Syntax

All TMCBUG2 commands are one to three alphabetic characters long and may be entered in upper case or lower case. The delimiter characters for commands are the space and carriage return.TMCBUG2 takes different actions depending on the the command delimiter. Typically, if more parameters are needed for the command, a space is the delimiter before each parameter. If no parameters are needed, a carriage return executes the command. For example if the command DTP (define terminal port) is entered at the command level followed by <CR>, the command displays the current driver for the terminal port. If a <SP> is the delimiter, the command parser accepts a parameter (number) as the new value for the terminal driver.

During command input, errors are corrected using the DELETE (RUBOUT) key. When TMCBUG2 deletes an error (assumed to be on a video type of terminal) it backspaces over the error character, blanks it out and positions the cursor next to the previous character that was valid. When three characters have been input, no more alphabetic characters are accepted. At this point, only a delete or delimiter may be entered.
The command level parses commands alphabetically and executes the first command with sufficient distinguishing characters. For example, assume the user typed "B" at the command level. Several commands begin with "B":BDA, BDB, BDS and BRK. Since "B" matchs BDA first, the BDA command is executed.If "BR" is entered at the command level, the first command that matchs is BRK.

A special command at the command level is the decimal point "." command. This does not require a delimiter and traces one instruction immediately (see TRA command).

Some commands allow the Ctrl-Y or Ctrl-C characterto terminate the TMCBUG2 command.(Although both Ctrl-Y and Ctrl-C are used for command termination, Ctrl-Y is

used throughout this document).This is to allow the user to return to the command level if a command is started that does not normally terminate. For example, if the LOA command is entered to load an OMF file but the host is not ready to send data, then the LOA command waits indefinitely.The LOA command may be terminated with Ctrl-Y or Ctrl-C to return to the command level.

1.7 Numeric Input Syntax

Several commands require numeric input for their parameters. All TMCBUG2 input/output uses hexidecimal numbers.
TMCBUG2 accepts character input up to the amount of characters required for each data type. At this point TMCBUG2 does not allow any more characters to be input and waits for a delimiter or a DELETE character.The delimiter characters for numeric input are the space, carriage return and the up arrow (^). For example, for an eight bit value, TMCBUG2 allows up to 2 hex characters to be input. If the input is an address, then up to 4 ASCII hex characters may be input (all addresses are assumed to be 16 bits). Fewer characters may be input and leading zeros are not needed. The number "001A" is accepted the same as "1A" for an address.

The parsing is similar to the command level when a DELETE key is entered.When the numeric field is full, the only characters allowed are the DELETE or a delimiter.When a delimiter is entered, the numeric value input is complete.

If a Ctrl-C or Ctrl-Y character is typed when TMCBUG2 is expecting numeric input, TMCBUG2 returns to the command level without executing the command. This is a method of aborting the command.If a non-hex character is entered when numeric entry is required, TMCBUG2 outputs an error message and returns to the command level without processing the requested command.
On commands that require two addresses to specify a range of addresses (such as FIL or FND), the delimiter for the first address echos as a dash (-). This is to remind the user that the second address specifies the end of the range and the first address is the start of the range.

1.8 Output Control Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y

The TMCBUG2 output follows the Ctrl-S/Ctrl-Q (stop/start) flow control protocol used by DEC and other terminal manufacturers. If TMCBUG2 is outputting a considerable amount of data (such as on the DMP command), it is possible for TMCBUG2 to transmit faster than the terminal can receive the data. Most terminals respond with a Ctrl-S (stop) character to stop the flow of characters.TMCBUG2 monitors the input while outputting strings and numeric fields and stops its output if a Ctrl-S is received. When the terminal can accept more characters, a Ctrl-Q (start) command resumes outputting data. The user may also stop the output if needed by typing Ctrl-S and restart display scrolling using Ctrl-Q.

If a Ctrl-C or Ctrl-Y character is received when TMCBUG2 is outputting data, TMCBUG2 returns to the command level. This is used to terminate commands that print large amounts of data such as the DMP command.

Any character received during output other than Ctrl-C, Ctrl-Y, Ctrl-Q and Ctrl-S is ignored.

1.10 I/O Ports And Drivers

Two generic driver routines are available in TMCBUG2. Each device on the board is identified by a unique driver number and a three character name. The name is to help the user remember the device associated with the device number. The current devices are defined below:

1. $000000 SCI - This is the Serial Communications Interface (SCI) on the DSP56002. This driver initializes the SCI to 9600 baud.

2. $000001 HST - This is the host interfaceonthe DSP56002.

It is possible to control TMCBUG2 from any device. The default configuration is the terminal port assigned to driver 1 (SCI) and the host port assigned to driver 2 (HST).

1.11 I/O Control

The I/O is controlled by setting a variable called IODEV. This variable contains the driver number to perform I/O on. Any I/O performed references IODEV to find out what device is to be used and the appropriate driver is looked up in IOTBL and dispatched to. Two other variables, TDEV and HDEV hold the driver numbers for the terminal port and the host port respectively. If I/O is desired on the host port, the driver value from HDEV is copied to IODEV and any I/O subroutine called uses the host port. TMCBUG2 is constantly switching the value of IODEV depending on if it is reading/writing from the terminal port or host port.

1.12 Powerup Options

2 TMCBUG2 COMMANDS

2.1 Command Syntax

TMCBUG2 contains a powerful set of commands to allow the user flexibility in debugging and developing DSP56002 code. TMCBUG2 is at the command level when the prompt "TMCBG2>" is printed. Entering a <CR> at the command level (no command) reprompts the user for a command.

The notation to indicate parameters for the commands is described below:

1. <prm> - A parameter enclosed in angle brackets<> indicates a required value for the command.

2. [opt1,opt2,...] - Optional parameters are enclosed in square brackets [] and indicate a list of optional parameters for the command. One of the listed parameters may be selected or no parameter may be selected.

3. <[opt1,opt2,...]> - This indicates a list of options where one option must be supplied. These options are enclosed in angle and square brackets <[ ]>.

The <SP> and <CR> are the only delimiters that are allowed for commands. The <SP>, <CR> and ^ are the delimiters for numeric parameters. In commands that require a range to be entered, a dash ("-") echos between addresses specifying the range even though the addresses are delimited with a space or carriage return.

All command names and parameters may be entered in upper or lower case.

Command Overview:

Execution Control:

GO [ADDR]	Begin execution
JSR [ADDR]	Call a subroutine
GOB [BREAKPOINT]	Set temporary breakpoint and begin execution at current PC

Debugging:

BRK <[S,R,-,<CR>]>	Breakpoints
NBR [NUMBER OF BREAKPOINTS]	Number of breakpoints to execute before stopping
TRA [N INSTRUCTIONS]	Trace

Program/Data Transfers:

LOA	Load OMF records
UPL <[P,X,Y]>:<ADD1> <ADD2> <[O,D]>	Upload memory to the host

Device Configuration:

DHP [DEVICE]	Define host port device
DTP [DEVICE]	Define terminal port device

Display:

DIS	Display registers
DMP <[P,X,Y]>:<ADD1> [ADD2]	Dump memory
DST	Display Stack
DVD	Device display
FND <[P,X,Y]>:<ADD1> <ADD2> <VALUE> [MASK]	Find value in memory
RD <[P,X,Y]>:<ADDR>	Read memory location

Modify:

COP <[P,X,Y]>:<ADD1> <ADD2> <[P,X,Y]>:<ADDR>	Copy memory
FIL <[P,X,Y]>:<ADD1> <ADD2> [VALUE]	Fill memory
MEM <[P,X,Y]>:<ADDR>	Memory examine/modify
REG	Register examine/modify
STK	Stack examine/modify
WRT <[P,X,Y]>:<ADDR> <VALUE> [REPEAT]	Write without verify

Miscellaneous:

EVA <ARG1> <[+,-,*,/]> <ARG2>	Evaluate expression
HEL	Help
CSR	Display contents of TMC CSR register
TMC	Display contents of TMC internal memory
ROM <program>	Call second ROM program

2.5 BRK - Breakpoint Command

BRK <[-,S,R,<CR>]>

The BRK command is to set/remove/display breakpoints. The BRK command is followed by an option character:

1. - (Dash) - This removes all breakpoints.

2. S - This sets breakpoints. After "S" is entered, TMCBUG2 prompts for the breakpoint to be set with ">".

3. R - This removes breakpoints.After "R" is entered, TMCBUG2 prompts for the breakpoint to be removed with ">".

4. <CR> - This displays the current breakpoint list.

A total of ten breakpoints are possible. By default, TMCBUG2 stops execution after the first breakpoint is encountered. The NBR command is used to set a different number of breakpoints to be executed before stopping.

_____
Examples:

(remove all breakpoints)
TMCBG2>BRK -
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
(set a breakpoint at $1234)
TMCBG2>BRK S >1234
1234 0000 0000 0000 0000 0000 0000 0000 0000 0000
(set a breakpoint at $ABCD)
TMCBG2>BRK S >ABCD
1234 ABCD 0000 0000 0000 0000 0000 0000 0000 0000
(remove breakpoint from $1234)
TMCBG2>BRK R >1234
0000 ABCD 0000 0000 0000 0000 0000 0000 0000 0000

2.6 COP - Copy Memory

COP <[P,X,Y]>:<ADD1> <ADD2> <[P,X,Y]>:<DEST>

The COP command moves memory starting at ADD1 through ADD2 to address DEST.A read verify is performed after the write operation.

_____
Examples:

TMCBG2>COP P:E000-E105 X:0
ERR 0100 (this section is ROM or no memory)
ERR 0101
ERR 0102
ERR 0103
ERR 0104
ERR 0105
TMCBG2>

2.8 DHP - Define Host Port

DHP [DRVR #]

The DHP command defines the driver for the host port. The host port is used to upload memory from the prototype board or to download programs. If the command is delimited with a <CR>,the current number for the host port driver is displayed. If the command is delimited with a <SP>, the command accepts a new value for the host port driver and displays the new host port driver number. If no host computer is available, the user can assign the host port number to be the same as the terminal port. This allows the user to upload memory and download OMF records from the terminal port.

The DHP command also displays a threecharactername representing the device. This name is set in IOTBL. A list of devices is generated by using the DVD command.

_____
Examples:

(display host port number)
TMCBG2>DHP
000002 DRB
(change host port driver to be the DSP SCI)
TMCBG2>DHP 3
000003 SCI
TMCBG2>

2.9 DIS - Display User Registers

DIS

The DIS command displays the user registers. The old register values are saved when a GO, JSR or TRA command is executed. After control is passed back to TMCBUG2 by a RTS (if the JSR commandis used), a breakpoint or hardware abort, the registers that have changed are flagged with an asterisk.

_____
Examples:

(display user registers)
TMCBG2>DIS

X1= 000000 X0= 000000 R7= 0000 N7= 0000 M7= 0000
Y1= 000000 Y0= 000000 R6= 0000 N6= 0000 M6= 0000
A2= 00A1= 000000 A0= 000000 R5= 0000 N5= 0000 M5= 0000
B2= 00B1= 000000 B0= 000000 R4= 0000 N4= 0000 M4= 0000
R3=*F123 N3= 0000 M3= 0000
PC=*F001 SR= 0354 OMR= 02R2=*E000 N2= 0000 M2= 0000
LA= 0000 LC= 0000SP= 00R1= 0000 N1= 0000 M1= 0000
BCR= 0004 IPR= 000000 R0= 0000 N0= 0000 M0= 0000

In the above example, registers R2, R3 and the PC were changed
due to the user program.

2.10 DMP - Dump Memory

DMP <[P,X,Y]>:<ADD1> [ADD2]

The DMP command displays memory in a horizontal format.The DMP command rounds the first address to the lowest 8 word block and then dumps enough lines to include the data for the second address.

If the first address is delimited by a <CR> then the DMP command dumps four lines of memory (32 words). Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

Examples:

```
(dump P memory)
TMCBG2>DMP P:E003-E00B
P:E000 000000 000000 000000 000000 000000 000000 000000 000000
P:E008 000000 000000 000000 000000 000000 000000 000000 000000
TMCBG2>DMP P:E015(delimit with a <CR>)
P:E010 512314 321334 402010 1BFDA1 A8B9D9 FF00FF AB123A 123412
P:E018 001104 402010 1BFDA1 A8B9D9 FF00FF AB123A 123412 AABBDD
P:E020 402010 1BFDA1 A8B9D9 FF00FF AB123A 123412 78A91B B18D11
P:E028 1BFDA1 A8B9D9 FF00FF AB123A 123412 402010 1BBB1B DADDAD
TMCBG2>
```

2.11 DST - Display The User Stack

DST

The DST command displays the data in the user stack according to the user stack pointer. If the SE bit is set in the user stack pointer, an error is generated and the stack is not displayed.

Examples:

```
TMCBG2>DST(display the user's stack)
SSH  SSL
01 0121 5512(bottom of stack)
02 5919 5291
03 0501 5929
04 0101 2223(top of stack, SP=4)
TMCBG2>
```

2.12 DTP - Define Terminal Port

DTP [DRVR #]

The DTP command defines the driver for the terminal port.If the command is delimited with a <CR>, the command displays the current number for the terminal port driver. If the command is delimited with a <SP>, a new value for the terminal port driver is input. For example, if the user is operating with a terminal on DUART-A, the user can change TMCBUG2 to operate from the host interface by changing the terminal port driver to 4.

The DTP command also displays a threecharactername representing the device. This name is set in IOTBL. A list of devices is generated with the DVD command.

_____
Examples:

(display host port number)
TMCBG2>DHP
000001 DRA
(change host port driver to be the DSP host
interface (or host interface if enabled))
TMCBG2>DHP 4
000004 HST
TMCBG2>

2.13 DVD - Device Display

DVD

The DVD command lists the drivers and their names that are currently in the device table IOTBL. Following each device driver number, a three character name is printed.The three character name is an abbreviation of the device name. The name is to aid the user in remembering the physical device associated with the device number.

_____
Examples:

(display host port number)
TMCBG2>DVD
000000 NUL (Null Device)
000001 DRA (DUART-A)
000002 DRB (DUART-B)
000003 SCI (SCI on the DSP56002)
000004 HST (8 bit parallel host interface on the DSP56002)
TMCBG2>

2.14 EVA - Evaluate Expression

EVA <AAAAAA><[+,-,*,/,.]> <BBBBBB>

The EVA command implements a four function calculator.The operation is performed on the two operands AAAAAA and BBBBBB. Both operands are considered to be two's complement fractional numbers.

The divide operation is a 24 bit two's complement signed division as shown in the DSP56002 Digital Signal Processor User's Manual in the section describing the DIV instruction.

The decimal point command converts the hex fraction to a decimal number.

_____
Examples:

(add two numbers)
TMCBG2>EVA 123456 + 010203=133659
(subtract two numbers)
TMCBG2>EVA 123456 - 010203=113253
(multiply two numbers)
TMCBG2>EVA 123456 * 010203=0024B1 EA9204
(divide two numbers)
TMCBG2>EVA 002222 / 123456=00EFFF
(convert number to decimal)
TMCBG2>EVA 873410 . -.94372367
TMCBG2>

2.15 FIL - Fill Memory

FIL <[X,Y,P]>:<ADD1> <ADD2> [NNNNNN]

The FIL command fills X, Y or P memory from ADD1 through ADD2 with the value of $NNNNNN. A verify is performed after the data has been written to the memory location and if there is an error writing the address, an error message is printed. Errors occur if there is ROM or no memory at the specified address.

If <ADD2> is delimited with a <CR>, then a default value of zero is used for $NNNNNN.

_____
Examples:

(fill X memory from $5 through $1A with $1A2B3C)
TMCBG2>FIL X:5-1A 1A2B3C
(fill Y memory from 0 through $105 with $123456)
TMCBG2>FIL Y:0-105 123456
ERR 100(this section of Y memory is ROM)
ERR 101
ERR 102
ERR 103
ERR 104
ERR 105
TMCBG2>FIL Y:0-1F (fill Y with zeros)
TMCBG2>

2.16 FND - Find A Value In Memory

FND <[P,X,Y]>:<ADD1> <ADD2> <VALUE> [MASK]

The FND command searchs for VALUE from ADD1 through ADD2. The optional mask performs a logical AND to mask the data after it is read from memory. If VALUE is delimited with <SP> then the MASK is entered, otherwise the default value of $FFFFFF (match all bits) is used. The MASK is used as a wildcard specifier to search for a particular bit or fields in memory.

_____
Examples:

```
TMCBG2>FND P:E000-F000 AF080(Search for JMP instruction)
E000 0AF080
E002 0AF080
E004 0AF080
E006 0AF080
TMCBG2>FND X:0-FF 1300 FFFF00(search for 0013xx)
0003 001315
0005 001319
0006 001320
0009 001355
TMCBG2>
```

2.17 GO - Begin Execution

GO [ADDR]

The GO command begins execution of the user program.If no address is given, execution begins at the address specified by the user program counter. If an address is given, execution begins at the specified address.

If the current program counter (or address in the command) is pointing to a breakpoint, TMCBUG2 traces 1 instruction to move past the breakpoint and then begins execution.Execution of the user's program continues until the specified number of breakpoints have been executed (see NBR command) or a hardware abort is generated. The hardware abort is software disabled when the prototype board is in the monitor mode. When the GO command terminates due to a breakpoint or hardware abort, a register dump.

If no breakpoints are set, TMCBUG2 prints a warning message indicating that no breakpoints are set and then begins execution of the user program. If the SE bit is set in the stack pointer, TMCBUG2 prints out an error message and does not begin execution. The user must change the stack pointer and remove the SE bit (by using the REG command) before execution of the user program is allowed.

_____
Examples:

```
TMCBG2>GO E000 (begin execution at $E000)
BREAKPOINT
```

```
X1= 000000 X0=*123456  R7= 0000  N7= 0000  M7= 0000
Y1= 000000 Y0=*9A9A9A  R6= 0000  N6= 0000  M6= 0000
A2= 00A1= 000000 A0= 000000  R5= 0000  N5= 0000  M5= 0000
B2=*00B1=*000000 B0=*000000  R4= 0000  N4= 0000  M4= 0000
R3= 0000  N3= 0000  M3= 0000
PC=*F000  SR= 0354  OMR= 02R2= 0000  N2= 0000  M2= 0000
LA= 0000  LC= 0000SP= 00R1= 0000  N1= 0000  M1= 0000
BCR= 0004  IPR= 000000  R0= 0000  N0= 0000  M0= 0000

TMCBG2>
```

The PC in the register display is the location of the next instruction to be executed. The instruction at $F000 has not been executed.

2.18 GOB - Begin Execution With Temporary Breakpoint

GO [BREAKPOINT]

The GOB command sets a temporary breakpoint and begins execution of the user program at the current user PC. If no breakpoint is given, no breakpoint is set and execution begins at the address specified by the user program counter. After any breakpoint is encountered, the temporary breakpoint is removed. The GOB command is useful when it is desired to step through programs in large blocks without having to set a breakpoint, execute the program and then remove the breakpoint.

Execution of the user's program continues until the specified number of breakpoints have been executed (see NBR command) or a hardware abort is generated.

If the SE bit is set in the stack pointer, TMCBUG2 prints out an error message and does not begin execution. The user must change the stack pointer and remove the SE bit (by using the REG command) before execution of the user program is allowed.

Examples:

```
TMCBG2>REG(display current user PC)
PC=E000
SR=0254(Ctrl-Y)
TMCBG2>GOB E010 (begin execution at $E000)
BREAKPOINT
X1= 000000 X0=*123456  R7= 0000  N7= 0000  M7= 0000
Y1= 000000 Y0=*9A9A9A  R6= 0000  N6= 0000  M6= 0000
A2= 00A1= 000000 A0= 000000  R5= 0000  N5= 0000  M5= 0000
B2=*00B1=*000000 B0=*000000  R4= 0000  N4= 0000  M4= 0000
R3= 0000  N3= 0000  M3= 0000
PC=*E010  SR= 0354  OMR= 02R2= 0000  N2= 0000  M2= 0000
LA= 0000  LC= 0000SP= 00R1= 0000  N1= 0000  M1= 0000
BCR= 0004  IPR= 000000  R0= 0000  N0= 0000  M0= 0000

TMCBG2>
```

2.19 HEL - Help Command

HEL

The HEL command displays the commands currently in the command table followed by a one line description of the command. The first 3 characters of each line is the commandname. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

_____
Examples:

```
TMCBG2>HEL
BDA [BAUD] SET BAUD RATE FOR DUART-A
BDB [BAUD] SET BAUD RATE FOR DUART-B
BDS [BAUD] SET BAUD RATE FOR SCI
BRK <[S,R,-,<CR>]>  SET/REMOVE/REMOVE ALL/DISPLAY BREAKPOINTS
COP <[P,X,Y]>:<ADD1><ADD2> <[P,X,Y]>:<DEST> COPY MEMORY
.
.
.
TMCBG2>
```

2.22 JSR - Call A User Subroutine

JSR [ADDR]

The JSR command calls a user subroutine.TMCBUG2 transfers control to the user subroutine and control is returned to TMCBUG2 when the user executes a RTS or a breakpoint. If no address is given, the address of the routine is taken from the user program counter.If an address is given, the user program counter is changed to the new address and execution is started.

After the user RTS is executed, the program counter in the user registers is the starting address of the subroutine if no breakpoints were encountered. If a breakpoint was encountered during subroutine execution, the program counter is the first instruction following the last breakpoint executed.This occurs because the address of the program counter is not meaningful after the user RTS is executed.

If no breakpoints are set, TMCBUG2 prints out a warning message indicating that no breakpoints are set and then begins execution of the user program. If the SE bit is set in the stack pointer, TMCBUG2 prints out an error message and does not begin execution. The user must change the stack pointer and remove the SE bit (by using the REG command) before execution of the user program is allowed.

_____
Examples:

```
TMCBG2>JSR E000(the user calls a subroutine at address $E000)
USER RTS(the user RTS is executed)

X1= 000000 X0= 000000  R7= 0000  N7= 0000  M7= 0000
Y1= 000000 Y0= 000000  R6= 0000  N6= 0000  M6= 0000
```

```
A2= 00A1= 000000 A0= 000000  R5= 0000  N5= 0000  M5= 0000
B2= 00B1= 000000 B0= 000000  R4= 0000  N4= 0000  M4= 0000
R3=*F123  N3= 0000  M3= 0000
PC=*F001  SR= 0354  OMR= 02R2=*E000  N2= 0000  M2= 0000
LA= 0000  LC= 0000SP= 00R1= 0000  N1= 0000  M1= 0000
BCR= 0004  IPR= 000000  R0= 0000  N0= 0000  M0= 0000

TMCBG2>
```

2.23 LOA - Load OMF Records From The Host Port

LOA

The LOA command first sends a <CR> to the host port and then loads object module format (OMF) records from the host port. TMCBUG2 echos a P, X or Y to the terminal port corresponding to the memory space in a "_DATA" or "_BLOCKKDATA" record.
The host port is assigned as same as the terminal port at power up.

If it is desired to download OMF records from the terminal port, assign the same device driver number to the host port as is assigned to the terminal port using the DHP command.See Appendix C for the procedure to load a program using an IBM-PC.

If the command is accidentally entered or the host port is not sending data, the command is terminated by typing Ctrl-Y.

Examples:

```
TMCBG2>LOA(the user starts the load)
XYPPP(the user had a segment in X, Y and 3 P segments)
TMCBG2>(done)
```

2.24 MEM - Memory Display/Modify

MEM <[P,X,Y]>:<ADDR>

The MEM command allows memory to be displayed/modified one location at a time. After the MEM command is given, the memory space, memory address, and the data in decimal and hex aredisplayed.At this point, several subcommands are available:

1. <CR> - (carriage return) Display the next memory location.

2. <SP> - (space) Change the current memory location.A greater than sign (>) prompts for the new value. If no value is entered, the memory location is not changed.

3. ^ - (up arrow) Display the previous memory location.

4. Ctrl-Y, Ctrl-C - Exit the memory change mode.

If the input hex value during a change operation is delimited with an up arrow (^), the memory location is changed and the previous memory location is displayed.

_____
Examples:

TMCBG2>MEM X:0(display data at X:0000)
X:0000 +.00784313 010101 (the user enters a <CR>)
X:0001 +.03136254 040404 (the user enters a <CR>)
X:0002 +.03136254 040404 (the user enters a <CR>)
X:0003 +.03136254 040404 (the user enters a <CR>)
X:0004 +.03921568 050505 >123456 (enter a space to change the value)
X:0005 +.04705882 060606 (enter ^ to go to the previous location)
X:0004 +.14222216 123456 (a Ctrl-Y is entered)
TMCBG2>

2.25 NBR - Set/Display the Number of Breakpoints to Execute Before Stopping

NBR [# of breakpoints]

The NBR command sets/displays the total number of breakpoints to execute before stopping. By default, this number is set to one. If this value is changed with the NBR command, for example to three, then execution stops after three breakpoints have been executed. This is very useful when it is desirable to stop a program in a loop after many passes. If the command is terminated with a carriage return, the current number is displayed.If the command is terminated with a space, a new value is entered.

_____
Examples:

TMCBG2>NBR (display the number of breakpoints to execute)
NUM BRKS= 0001
TMCBG2>NBR 3 (set the number of breakpoints to execute to 3)
NUM BRKS= 0003
TMCBG2>

2.26 RD - Read Memory Location And Display Changes

RD <[P,X,Y]>:<ADDR>

The RD command reads the data at address ADDR and displays any changes. It is assumed that the initial value is zero. Every time the memory location changes value, the new value is displayed. This is useful for monitoring a status register, a PIA input port or a serial receive register to aid program and hardware debugging.Ctrl-Y terminates this command and returns to the command level.

Examples:

TMCBG2>RD X:FFF4 (monitor data at the serial receive register)
X:FFF4 000041(the character A is received)
X:FFF4 000042(the character B is received)
Ctrl-Y(the user terminates and returns to TMCBUG2)
TMCBG2>

2.27 REG - Modify User Registers

REG

The REG command allows the user to modify the registers of the current user state.Once the command is entered, the first register is displayed with its value. At this point several subcommands are available:

1. <CR> - (carriage return) Display the next register.

2. <SP> - (space) Change the current register.A greater than sign (>) prompts for the new value. If no value is entered, the register is not changed.

3. ^ - (up arrow) Display the previous register.

4. Ctrl-Y, Ctrl-C - Exit the register change mode.

If the input hex value during a change operation is delimited with an up arrow (^), the register is changed and the previous register is displayed.

Examples:

TMCBG2>REG (enter register change mode)
PC=E000 (enter a space)
SR=0254 >0255 (enter a space to change the register,
delimit with ^ to go to the PC)
PC=E000 (enter Ctrl-Y to exit)
TMCBG2>

2.28 STK - Edit The User Stack

STK

The STK command edits the user stack. After the command is entered, the following options are available:

1. <CR> - (carriage return) Display the next stack location.

2. <SP> - (space) Change the current stack location.A greater than sign (>) prompts for the new value. If no value is entered, the stack location is not changed.

3. ^ - (up arrow) Display the previous stack location.

4. Ctrl-Y, Ctrl-C - Exit the stack change mode.

If the input hex value during a change operation is delimited with an up arrow (^), the stack location is changed and the previous stack location is displayed.

If the SE bit is set in the user stack pointer, an error message is generated.

Examples:

```
TMCBG2>STK (enter stack edit mode)
01 SSH = 0512  (enter 3 carriage returns)
01 SSL = 0352
02 SSH = FB12
02 SSL = 0354  >0355 (enter a space and change the value)
03 SSH = F121  (user enters ^ to go to the previous)
02 SSL = 0355  (user enters Ctrl-Y to exit)
TMCBG2>
```

2.29 TRA - Trace Instructions

TRA [N]

The TRA command allows single step execution of the user program.The execution begins from the program counter specified in the user registers. If N is not given, TMCBUG2 traces one instruction. Tracing continues until the number of instructions is traced or a breakpoint is encountered.After tracing is complete, the user registers are displayed. After each instruction is traced, the current program counter is displayed.This program counter is the address of the next instruction to be executed. Tracing may be aborted by typing Ctrl-Y during tracing.

A short form of the trace command is to use a period (.) at the command level. This is equivalent to "TRA" or "TRA 1".

Examples:

```
TMCBG2>.(trace 1 instruction)
TR 6005

X1= 000000 X0= 000000  R7= 0000  N7= 0000  M7= 0000
Y1= 000000 Y0= 000000  R6= 0000  N6= 0000  M6= 0000
A2= 00A1= 000000 A0= 000000  R5= 0000  N5= 0000  M5= 0000
B2= 00B1= 000000 B0= 000000  R4= 0000  N4= 0000  M4= 0000
R3= 0000  N3= 0000  M3= 0000
PC=*6005  SR= 0354  OMR= 02R2= 0000  N2= 0000  M2= 0000
LA= 0000  LC= 0000SP= 00R1= 0000  N1= 0000  M1= 0000
BCR= 0000  IPR= 0000R0= 0000  N0= 0000  M0= 0000
TMCBG2>TRA 4
TR 6006
TR 6007
TR 6008
```

```
TR 6009
X1= 000000 X0= 000000  R7= 0000  N7= 0000  M7= 0000
Y1= 000000 Y0= 000000  R6= 0000  N6= 0000  M6= 0000
A2= 00A1=*550000 A0= 000000  R5= 0000  N5= 0000  M5= 0000
B2= 00B1=*880000 B0= 000000  R4= 0000  N4= 0000  M4= 0000
R3= 0000  N3= 0000  M3= 0000
PC=*6009  SR= 0354  OMR= 02R2= 0000  N2= 0000  M2= 0000
LA= 0000  LC= 0000SP= 00R1= 0000  N1= 0000  M1= 0000
R0= 0000  N0= 0000  M0= 0000
BCR= 0000  IPR= 0000R0= 0000  N0= 0000  M0= 0000
TMCBG2>
```

It should be noted that the address in the "TR AAAA" message refers to the address of the instruction that execution stopped at. The instruction at this address is not executed. If execution is to continue, this is the address of the next instruction to be executed. In the example above, the address in the "TR 6005" message indicates that execution has stopped at $6005 but the instruction at P:$6005 has not been executed.

2.30 UPL - Upload Memory To The Host Port

UPL <[P,X,Y]>:<ADD1> <ADD2> <[O,D]>

The UPL command allows memory to be uploaded to a host computer via the host port.Two different formats are available.

The O specifies OMF records to be sent. The first record is a "_DATA" record followed by the data in ASCII hexidecimal for each memory location on aseparateline.Thedata transmissionisterminatedwiththe"_END"record. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output to the host (see the section Output Control). The D specifies data format which is the same as OMF format without the "_DATA" and "_END" records.

_____
Examples:

```
TMCBG2>HST(connect to the host computer, assume a VAX)
$ CREATE DATA.MEM  (create a ASCII data file)
Ctrl-A(return to TMCBUG2)
TMCBG2>UPL X:50-70 O(upload memory from $50 to $70
to the host in OMF format)
TMCBG2>HST(connect to the host computer)
Ctrl-Z(exit the create command)
$(VAX/VMS prompt)
```

2.32 WRT - Write A Value Without Read Verification

WRT <[P,X,Y]>:<ADDR> <NNNNNN> [REPEAT]

The WRT command writes a value to a memory location without a read verification.This is useful for writing write-only registers such as transmit data registers, D/A converters or for hardware debugging. If the value NNNNNN is delimited with a <SP>, then the repeat count is

entered.The repeat count specifies how many times to write the value to memory. If the value NNNNNN is delimited with <CR>, the repeat count is infinite and the value is written to memory continuously until a Ctrl-C or Ctrl-Y is entered.

_____
Examples:

(write the letter A to the serial transmit register)
TMCBG2>WRT X:FFF4 41 1
TMCBG2>WRT Y:FF14 99 (write 99 to Y:FF14 continuously)
Ctrl-Y(exit the command)
TMCBG2>

2.34 CSR - Display contents of TMC-CSR

CSR

The CSR command is a specific command in the TMC-VME board. The command shows the contents of all CSR registers in the TMC chips.

_____
Examples:

```
TMCBG2>CSR
Chip   0  1  2  3  4  5  6  7
CSR0: 00 00 00 00 00 00 00 00
CSR1: 00 00 00 00 00 00 00 00
CSR2: 00 00 00 00 00 00 00 00
CSR3: 00 00 00 00 00 00 00 00
```

2.35 TMC - Display bit pattern of TMC memory

TMC

The TMC command is a specific command in the TMC-VME board. The command asks channnel number, and shows the contents of the TMC chip.

_____
Examples:

```
TMCBG2>TMC
tmcread:
Channel No. (0-31)? 2
Rising Edge
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 85 01 00 00 00 00 00 00 00 00 00 00 00 00 00
:
Falling Edge
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 00 00 86 01 00 00 00 00 00 00 00 00 00 00 00
:
```

2.35 ROM - call second ROM program

3 USER ACCESSIBLE MONITOR UTILITIES

TMCBUG2 has a jump table at the beginning to allow user access to subroutines.The entry point of these subroutines are defined by module "BUGEQU.ASM" and should be included at the beginning of any user program that requires access to TMCBUG2's utilities. All access to TMCBUG2 utilities are hrough subroutine calls to the symbols defined in BUGEQU.

3.1 The TMCBUG2 Utility Table

The following descriptions of the subroutineutilities reference the subroutines symbolically. All symbols are defined in upper case so all references to the subroutines shouldbe in upper case. Alternatively, the "opt ic" assembler directive may be used to inhibit case sensitivity.

Most utilities are called as subroutines. The user sets up the registers required by the subroutine and then performs a JSR to the address specified by BUGEQU. The first instruction in the subroutine is a JMP to the desired code module. The TMCBUG2 utility ends with an RTS instruction which returns to the user application.

An example is shown below. This example reads 6 ASCII hex characters and outputs the value followed by a carriage return and line feed.

```
        ORG   P:$6000
INCLUDE '   BUGEQU'

START
        JSR   INDAT       GET DATA
        JSR   OUTDAT      ;OUTPUT DATA
        JSR   CRLF        ;OUTPUT CR,LF
        JMP   START
        END
```

All of the subroutines save all the user's registers when called. Only if a value is to be passed back to the user is a register changed.

3.2 DSPBUG - TMCBUG2 Cold Start

This entry point resets TMCBUG2 to its initial powerup condition. This is similar to hitting the RESET switch on the DSP56002. This puts TMCBUG2 through its powerup sequence which includes clearing the breakpoint table, initializing the IO, setting default values for the ports, etc. This command does not perform a hardware reset on the DSP56002 or the IO. It does perform a software reset on the IO. It does not check the setting of debug switch, so not change the terminal and host port assignment.

It should noted that this is not a subroutine and no return to the user's program is possible. A JMP instruction can be used to transfer program control to this entry point.A JSR may also be used since the stack is reset.

3.3 CMD - Command Level Entry Point

This entry point returns control to the TMCBUG2 command level and the TMCBUG2 prompt appears. It should be noted that the stack pointer is reset at the command level. All user registers are destroyed and all address modifier registers are set for linear arithmetic.

This entry point should not be used by programs to return to TMCBUG2 if breakpoints are being used. If this entry point is used to return control to TMCBUG2 from a user program and breakpoints are being used, the SWIs are not removed from the user's program.

It should noted that this is not a subroutine and no return to the user's program is possible. A JMP instruction can be used to transfer program control to this entry point.A JSR may also be used since the stack is reset.

3.4 SET_TERM - Set The Terminal Port As The Current I/O Port

This subroutine sets the terminal port as the current I/O device. This copies the value in TDEV to IODEV.

3.5 SET_HOST - Set The Host Port As The Current I/O Port

This subroutine sets the host port as the current I/O device. This copies the value in HDEV to IODEV.

3.6 DEFINE_TERM - Define Terminal Port

This subroutine takes the integer value in B1 and saves it in TDEV as the current driver number for the terminal port. TMCBUG2 then executes the initialization routine for the device.

3.7 DEFINE_HOST - Define Host Port

This subroutine takes the integer value in B1 and saves it in HDEV as the current driver number for the host port. TMCBUG2 then executes the intialization routine for the device.

3.8 IN0 - Scan Input Device

This subroutine scans the input device pointed to by IODEV and returns the Z bit in the status register equal to 1 if there is no data. If there is data pending, the Z bit is cleared and the 8 bit data is returned in the lower byte of A1. The upper two bytes of A1 are zeroed.

```
.
.
        JSR         IN0         ;SCAN INPUT DEVICE
        JEQ         _NODATA     ;JUMP IF NO DATA
        MOVE        A,X:0       ;SAVE INPUT DATA
```

.
.

3.9 IN1 - Read Input Device

This subroutine scans the input device pointed to by IODEV and waits for data to be available. When this routine returns, the 8 bit data is in the lower byte of A1.The upper two bytes of A1 are zeroed.

3.10 IN2 - Input Data, Mask Parity, Top Byte Of A1

This subroutine scans the input device pointed to by IODEV and waits for data to be available. When data is read, the parity is masked (bit 7 set to zero) and the data is shifted into the top byte of A1. The lower two bytes of A1 are zeroed.

3.11 IN3 - Input Data, Mask Parity, Top Byte Of A1, Echo Character

This subroutine scans the input device pointed to by IODEV and waits for data to be available. When data is read, it is echoed to the device pointed to by IODEV, parity is masked off and the data is returned in the top byte of A1. The lower two bytes of A1 are zeroed.

3.12 INBYTE - Input A Byte

This subroutine reads the device pointed to by IODEV, inputs 2 ASCII hex characters and converts them to an 8 bit value. The value is returned in the lower byte of B1 with the upper two bytes zeroed.The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed . The number of characters input is returned in R7.

3.13 INADD - Input An Address

This subroutine reads the device pointed to by IODEV, inputs 4 ASCII hex characters and converts them to a 16 bit value. The value is returned in the lower two bytes of B1 with the upper byte zeroed.The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed. The number of characters input is returned in R7.

3.14 INDAT - Input Data

This subroutine reads the device pointed to by IODEV, inputs 6 ASCII hex characters and converts them to a 24 bit value. The value is returned in B1. The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed.The number of characters input is returned in R7.

3.15 INBYTE_NLS - Input A Byte And Ignore Leading Spaces

This subroutine reads the device pointed to by IODEV, inputs 2 ASCII hex characters and converts them to an 8 bit value. The value is returned in the lower byte of B1 with the upper two bytes zeroed.The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed. The number of characters input is returned in R7.

This subroutine is identical in operation to INBYTE except this subroutine ignores any leading spaces before the hex characters. The spaces that are ignored are not echoed. Once hex characters are detected, the space acts as a delimiter.

3.16 INADD_NLS - Input An Address And Ignore Leading Spaces

This subroutine reads the device pointed to by IODEV, inputs 4 ASCII hex characters and converts them to a 16 bit value. The value is returned in the lower two bytes of B1 with the upper byte zeroed.The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed. The number of characters input is returned in R7.

This subroutine is identical in operation to INADD except this subroutineignoresanyleading spaces before the hex characters. The spaces that are ignored are not echoed. Once hex characters are detected, the space acts as a delimiter.

3.17 INDAT_NLS - Input Data And Ignore Leading Spaces

This subroutine reads the device pointed to by IODEV, inputs 6 ASCII hex characters and converts them to a 24 bit value. The value is returned in B1. The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed.The number of characters input is returned in R7.

This subroutine is identical in operation to INDAT except this subroutine ignores any leading spaces before the hex characters. The spaces that are ignored are not echoed. Once hex characters are detected, the space acts as a delimiter.

3.18 OUT0 - 8 Bit Data Output

This subroutine outputs the 8 bit data from the lower byte of A1 to the I/O device pointed to by IODEV. This subroutine waits until the device is ready to send. The upper two bytes of A1 are ignored.

3.19 OUT1 - 8 Bit Data Output, Output Control

This subroutine outputs 8 bit data from the lower byte of A1 to the I/O device pointed to by IODEV. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

3.20 OUT2 - Mask Parity, Shift Data, Data Output, Output Control

This subroutine masks the parity of the top byte in A1, shifts the data to the lower byte of A1 and outputs the data to the device pointed to by IODEV.Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control). The output routine waits until the device is ready to send the data.

3.21 OUT3 - Output 3 Packed Characters

This subroutine outputs 3 packed binary characters from A1 to the I/O device pointed to by IODEV. The first character output is in the high byte of A1, the second character output is in the middle byte of A1 and the last byte output is in the low byte of A1. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control). The output routine waits until the device is ready to send the data.The parity is not masked for the data output.

```
.
.
        MOVE    #'ABC',A    ;LOAD "ABC" INTO A
        JSR     OUT3 ;OUTPUT "ABC"
.
.
```

3.22 OUT4 - Output A String Of Packed Characters

This subroutine outputs a string of packed characters to the device pointed to by IODEV. Address register R0 initially points to the location of the packed string in P memory space. Register N0 is the number of words to output and M0 is set for linear addressing. The packed string has 3 characters per 24 bit word.The first character output is the high byte, the second is the middle byte and the third is the low byte. Data packing of ASCII characters is performed by the DC assembly directive as shown below:

```
        ORG   P:$6000
M1
        DC    'THIS IS A TEST'          ; MESSAGE TEXT
M1L     EQU   *-M1                      ; MESSAGE LENGTH
```

An example program to output this text:

```
        MOVE    #M1,R0          ;POINT TO MESSAGE
        MOVE    #M1L,N0         ;MESSAGE LENGTH
J       SR      OUT4        ;OUTPUT STRING
```

Register R0 is left pointing to the memory location after the last word in the string.

3.23 OUT5 - Output A String Of Packed Characters

This subroutine outputs a string of packed characters to the device pointed to by IODEV. Address register R0 initially points to the beginning location of the packed string in P memory space and the string ends with a value of zero. Register M0 is set for linear arithmetic. The packed string has 3 characters per 24 bit word. The first character output is the high byte, the second is the middle byte and the third is the low byte.Data packing of ASCII characters is performed by the DC assembly directive as shown below:

```
        ORG   P:$6000
M1
        DC    'THIS IS A TEST',0        ; MESSAGE TEXT
```

An example program to output this text:

```
MOVE    #M1,R0          ;POINT TO MESSAGE
JSR     OUT5        ;OUTPUT STRING
```

Register R0 is left pointing to the memory location after the zero value in the string.

3.24 OUTBYTE - Output A Byte

This subroutine outputs a byte as 2 ASCII hex characters from the lower byte of B1 to the device pointed to by IODEV. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

3.25 OUTADD - Output An Address

This subroutine outputs a 16 bit address as 4 ASCII hex characters from the lower two bytes of B1 to the device pointed to by IODEV.Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

3.26 OUTDAT - Output Data

This subroutine outputs a 24 bit data value as 6 ASCII hex characters from B1 to the device pointed to by IODEV. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

3.27 OUTINT - Decimal Output Of A Value

This subroutine outputs a 16 bit value in B1 as an ASCII decimal number to the device pointed to by IODEV. Ctrl-S/Ctrl-Q, Ctrl-C, Ctrl-Y output control is observed during data output (see the section Output Control).

3.28 ININT - Decimal Input Of A Value

This subroutine reads the device pointed to by IODEV, inputs 6 decimal digits and convert them to a 56 bit value. The binaly value is returned in B. The input string must be terminated with a delimiter (<CR>, <SP> or ^). The delimiter is returned in the top byte of A1 with the lower two bytes zeroed.The number of characters input is returned in R7.

3.29 CRLF - Print A <CR><LF>

This subroutine outputs a carriage return and then a line feed to the device pointed to by IODEV.

3.30 PRTSPC - Print A Space

This subroutine outputs a space to the device pointed to by IODEV.

3.33 OUTFRAC - Output Value In Decimal

This subroutine outputs the value in B1 as a signed decimal fraction.The output includes a leading sign and decimal point. For example, if B1 has the value $CC0000, then the output of OUTFRAC would be " -.40625000".

3.34 Example Using The TMCBUG2 Subroutines

This section provides an example using the TMCBUG2 subroutines. The problem considered here is a DSP56002 program that requires a block of data from the host. The system configuration consists of a separate terminal, host computer and TMC-VME module connected with serial lines. The host runs a FORTRAN program to get the data from a file and the data is sent to TMCBUG2.

The FORTRAN program to send the data is shown below:

```
      CHARACTER*1      DUMMY
      OPEN (UNIT=1,FILE='TEST.DAT',STATUS='OLD')
10    CONTINUE
      READ(5,15) DUMMY                !WAIT FOR <CR> TO COME
15    FORMAT(A)
      READ(1,*,ERR=999) NUMBER        !INPUT DATA FROM FILE
      WRITE(6,20) NUMBER              !WRITE OUT TO TERMINAL (TMCBUG2)
20    FORMAT('+',Z8,' ')              !PREFIX CHAR, DATA,DELIMITER
      GOTO 10                         !CONTINUE LOOPING UNTIL OUT OF DATA
999   CLOSE (UNIT=1)
      END
```

The Z8 is a format field to write the integer value in hexidecimal. This program operates as follows:

1. A file "test.dat" is opened to obtain data.

2. The program reads a one character variable into DUMMY. This is to make the program wait until TMCBUG2 requests the data. A <CR> satisfies this read statement.

3. After TMCBUG2 sends the <CR> to indicate that it is ready to receive, the FORTRAN program reads the integer data from the input file and prints it. Since the host port of TMCBUG2 is connected to the host computer, TMCBUG2 reads the characters printed.

A plus sign (+) is a synchronization character. After the plussign, TMCBUG2 knows the next character is the beginning of the numeric field.

A space is printed after the hex digits to be the delimiter for the TMCBUG2 input subroutine.

4. The program continues reading until the end of file is detected.

The program for the TMC_VME module using TMCBUG2 subroutines is shown below:

```
      PAGE 132,60,1,1
      INCLUDE     'BUGEQU'
```

```
        ORG         P:$1000
;
;SUBROUTINE TO READ A DATA VALUE FROM THE HOST
;
GET
        JSR         SET_HOST            ;SET HOST PORT FOR I/O
        MOVE        #$0D,A              ;GET <CR>
        JSR         OUT2        ;SEND IT TO THE HOST
_WT
        JSR    IN2                      ;GET CHARACTER FROM HOST
        MOVE        #'+',X0     ;GET PREFIX CODE
        CMP         X0,A                ;SEE IF PREFIX CODE
        JNE    _    WT                  ;WAIT FOR PREFIX CODE
        JSR         INDAT_NLS ;GET DATA, IGNORE LEADING SPACES
        JSR         SET_TERM            ;SET THE TERMINAL FOR I/O
        RTS
;
;SUBROUTINE TO READ A BLOCK OF DATA FROM THE HOST
;AND TRANSFER IT TO X MEMORY
;
GET_BLOCK
        MOVE        #0,R1               ;PLACE TO PUT BLOCK
        DO          #20,_G              ;INPUT 20 DATA VALUES
        JSR         GET                 ;GET THE INPUT DATA VALUE
        MOVE        B,X:(R1)+           SAVE IN X MEMORY
_G
        RTS
```

Two subroutines are presented.The first subroutine GET performs the following operations:

1. The host port is set as the device for I/O.

2. A <CR> is sent to the host port.This character is received by the FORTRAN program on the host and indicates that TMCBUG2 is ready to receive data.

3. A single character is received from the host.If this character is not a plus sign (+), then more characters are read. The plus sign synchronizes communications.When the plus sign is detected, it is assumed that the next character is the beginning of the numeric field.

4. Six hex characters arereadusingtheINDAT_NLS subroutine.This subroutine ignores leading blanks and then reads the hex data from the host. The host sends a space as the delimiter for the number.

5. The terminal port is set as the I/O port.

The routine GET returns a single 24 bit value in B1 each time it is called.

The second routine GET_BLOCK calls GET several times and stores a block data in memory using a loop.

4 POWERUP BOOTING

The DSP56002 performs a double bootstrap before TMCBUG2 execution begins.

1. The first bootstrap is started by the DSP56002 when it comes out of reset in mode 1.

2. The DSP56002 special bootstrap ROM loads a program from the lower byte at P:$C000, assembles 3 bytes into a 24 bit instruction word and transfers it into programRAM startingatlocation P:$0000. The DSP56002 then performs a jump to location $0000 and begins execution. Thefirst program loaded is and an intelligent loader (I-loader) which loads TMCBUG2 and a user utility table which has jump address of user accessible subroutines. The I-loader occupies the lower byte EPROM from locations $C000-$C07F.

3. The I-loader begins reading from the lower byte of the EPROM at location $C800.This is the low byte of the first word of TMCBUG2. The I-loader reads 3 (8 bit) bytes at a time (low, middle and high) and assembles them by shifting into a 24 bit instruction word. The I-loader transfers each program word to RAM beginning at location P:$1300.

4. After both EPROMs have been read, the I-loader transfers control to the beginning of TMCBUG2.

APPENDIX A: KNOWN TMCBUG2 BUGS.

APPENDIX B: CURRENT LISTING OF BUGEQU.ASM

```
;
; THIS IS THE TMCBUG2 UTILITY DEFINITION TABLE.  GOOD PROGRAMMING
; PRACTICE WILL REFERENCE THE TMCBUG2 UTILITIES THROUGH THIS EQUATE
; TABLE TO PROVIDE UPWARD COMPATIBILITY WITH CHANGES IN THE
; MONITOR DEFINITION.
;
; TMCBUG2 Jump Address
DSPBUG          EQU   $40     ;MONITOR COLD RESET
CMD             EQU   $42     ;MONITOR WARM START
IN0             EQU   $44     ;SCAN INPUT DEVICE
IN1             EQU   $46     ;8 BIT DATA INPUT TO A1 (LOWER BYTE)
IN2             EQU   $48     ;INPUT CH, MASK PARITY, TOP BYTE OF A1
IN3             EQU   $4A     ;IN3 + ECHO CHARACTER
INBYTE          EQU   $4C     ;INPUT A 2 NIBBLE BYTE
INADD           EQU   $4E     ;INPUT A 4 NIBBLE ADDRESS INTO B1
INDAT           EQU   $50     ;INPUT A 6 NIBBLE DATA VALUE INTO B1
INBYTE_NLS      EQU   $52     ;INPUT A 2 NIBBLE BYTE
INADD_NLS       EQU   $54     ;INPUT A 4 NIBBLE ADDRESS INTO B1
INDAT_NLS       EQU   $56     ;INPUT A 6 NIBBLE DATA VALUE INTO B1
OUT0            EQU   $58     ;8 BIT OUTPUT FROM A1 (LOWER BYTE) NO CTRL/S/Q
OUT1            EQU   $5A     ;8 BIT DATA OUTPUT FROM A1 (LOWER BYTE)
OUT2            EQU   $5C     ;MASK PARITY, OUTPUT CH FROM TOP BYTE OF A1
OUT3            EQU   $5E     ;OUTPUT 3 PACKED CHARACTERS IN A1
OUT4            EQU   $60     ;OUTPUT N0 PACKED CHARS POINTED TO BY R0
OUT5            EQU   $62     ;OUTPUT CHARS POINTED TO BY R0 ENDING WITH 0
OUTBYTEEQU      $64     ;OUTPUT 2 NIBBLE BYTE FROM B1 IN HEX
OUTADD          EQU   $66     ;OUTPUT A ADDRESS FROM B1 IN HEX
OUTDAT          EQU   $68     ;OUTPUT A DATA VALUE FROM B1 IN HEX
OUTINT          EQU   $6A     ;OUTPUT 16 BITS FROM B1 AS DECIMAL INTEGER
CRLF            EQU   $6C     ;OUTPUT <CR><LF>
PRTSPC          EQU   $6E     ;OUTPUT A SPACE
OUTFRACEQU      $70     ;OUTPUT B AS A FRACTION
ININT           EQU   $72     ;INPUT 6 DIGITS AS DECIMAL INTEGE
SET_TERM        EQU   $74     ;SET TERMINAL AS I/O DEVICE
SET_HOST        EQU   $76     ;SET HOST AS I/O DEVICE
DEFINE_TERM     EQU   $78     ;SET TERMINAL WITH DEV IN B AND INIT DEV
DEFINE_HOST     EQU   $7A     ;SET HOST WITH DEV IN B AND INIT DEV
```

APPENDIX C: PROGRAM DOWNLOADING PROCEDURE USING AN IBM-PC

This appendix describes a procedure to download OMF records produced by the DSP56002 Assembler to the TMC-VME attached to an IBM-PC on a serial port. It is assumed that the IBM-PC has a communications program such as KERMIT and a serial port.

The following description explains how to load an OMF file on an IBM-PC to a prototype board using KERMIT:

```
C:\DSP> CD \KERMIT  (change directory from DSP to KERMIT)
C:\KERMIT> KERMIT(invoke the communications program KERMIT)
IBM-PC Kermit-MS V2.28
Type ? for help
Kermit-MS> SET SPEED 9600  (set serial port to 9600 baud)
Kermit-MS> CONNECT(connect to serial port of prototype)
[Connecting to host, type Control-] C to return to PC]
TMCBG2> DHP(display host port)
000000 SCI (current host port is SCI)
TMCBG2>LOA (invoke load command)
Ctrl-] C (exit KERMIT transparent mode)
Kermit-MS> EXIT (exit KERMIT)
C:\KERMIT> COPY \DSP\TEST.LOD COM1  (send file to serial port)
1 File(s) Copied
C:\KERMIT> KERMIT (invoke kermit)
IBM-PC Kermit-MS V2.28
Type ? for help
Kermit-MS> CONNECT(connect to prototype board)
[Connecting to host, type Control-] C to return to PC]
TMCBG2>
```

It is assumed that once the baud rate of the serial port is set with KERMIT that it does not change unless the IBM-PC is powered down or reset.