THE Bugger

RS232 Debugger For The MYARC 9640
(C) 1989 T and J Software
Written By Jim Lohmeyer

Documentation by Jim Lohmeyer
Edited by Tom Freeman

INTRODUCTION

THE Bugger is an RS232 debugging system designed on and for the MYARC 9640 computer, and will run in either the GPL simulation, or MDOS modes. Great care has been taken to ensure that it is fully self-contained so it will neither write to any memory locations outside of its own boundaries, nor corrupt any other resource it borrows from the operating system during the course of its operations. It is for this reason that input/output the debugger handles is routed through the RS232 port to an external terminal. By implementing this I/O scheme, the program to be debugged will have access to all of the operating system's resources, including the host keyboard and screen, other than the RS232/2 port.

The Bugger is also a fully intelligent debugger in the sense that it knows, at all times, which mode the computer is running in (TI simulation or MDOS) and how the program has the memory mapped. By keeping a tab on the status of the operating conditions, THE Bugger is not confused by a program that switches the operating mode or juggles the memory map. However, it is possible for an executing program to map out THE Bugger during normal operating conditions, so the user should take care not to do so by accident. [Ed. Note: In MDOS, if THE Bugger precedes the program being debugged, then it resides entirely in execution page 0 and cannot be mapped out.]

THE Bugger is based upon RSBUG V3.0, which was used in the development of the TI 99/8 and MYARC 9640 computer operating systems, but was never released. However, through extensive modifications, revisions, bug fixes, and additions, less than 30% of the original code survives intact. In form and syntax it is much like TI's DEBUG and other debuggers based on it. The essential difference between them and THE Bugger is that the host screen (used by the program being debugged) is NOT used by THE Bugger either to input commands or display the information requested. Rather they are routed through the RS232/2 port to and from a terminal or another computer, thus not corrupting the host screen, changing VDP registers etc. Several important enhancements have also been added, in particular the ability to set permanent breakpoints and to single step several instructions at once, rather than just one, with one command.

THE Bugger is a joint effort between the author and T and J software to bring quality developmental products to the 9640 market. The author would also like to recognize Mr. Lou Philips for his support of this project, and for approaching T and J Software on the development of such a product.

Special cables may be required for use with THE Bugger. For instructions on this and changing terminal protocols, please see the Appendix on p. 20

For more information or questions, please contact:

T & J Software
515 Alma Real Drive
Pacific Palisades, CA 90272

CONTENTS
========

INTRODUCTION 1
CONTENTS 3
LOADING
  LOADING IN MDOS 4
  LOADING IN TI MODE 5
THE COMMAND LINE 6
COMMANDS
  A -- INPUT ASCII TO MEMORY 7
  B -- SET/RELEASE BREAKPOINT 7
  C -- CRU INSPECT/CHANGE 8
  D -- DISASSEMBLE INTO OPCODES 8
  E -- EXECUTE TASK 8
  F -- FIND WORD/BYTE EQUAL 9
  K -- FIND WORD/BYTE NOT EQUAL 9
  M -- MEMORY INSPECT/CHANGE 10
  N -- MOVE BLOCK 11
  P -- COMPARE BLOCKS 11
  R -- RTWP VECTOR INSPECT/CHANGE 12
  S -- SINGLE STEP 13
  T -- DISASSEMBLE AS DATA 13
  W -- WORKSPACE INSPECT/CHANGE 14
  X,Y,Z, -- BIAS VALUES 14
  . -- DECIMAL TO HEX CONVERSION 15
  , -- HEX TO DECIMAL CONVERSION 15
  ; -- HEX ARITHMETIC 15
  = -- FIND ASCII STRING 16
  @ -- MEMORY MAPPER 16
  / -- RESET THE Bugger 16
  ^R - RESET 9640 16
  ^B - SPECIAL BREAKPOINT 17
  ^C - PERMANENT BREAKPOINT 17
  ^T - VDP STATUS REGISTERS 18
  ^U - VDP STATUS REGISTERS - DYNAMIC 18
  ^V - WRITE TO VDP REGISTER 18
SWITCH OPTIONS
  ^E - COMPUTER MODE 19
  ^L - LINE LENGTH 19
  ^P - PRINTER OUTPUT 19
  ^S - DUMP STATUS 20
  ^W - DUMP WORKSPACE 20
  ^D - DUMP DISASSEMBLY 20
APPENDIX 1 - CONSTRUCTION OF A NULL MODEM CABLE 21
APPENDIX 2 - CHANGING THE BAUD RATE 21

LOADING

THE Bugger can be loaded from either of the two 9640 environments. Loading procedures will vary greatly in different circumstances and are described below. The disk provided contains the following files:

```
BUGDOS
BUGGER
BUGGER6
BUGGERA  \
BUGGERC   \ In order to fit on an SSSD disk, these files have
BUGGERE   / been compressed into a single file BUGGERARC
BUGGERLOW/
```

LOADING IN MDOS

The simplest method of loading THE Bugger in any environment is to load it by itself for poking around the operating system to find secrets and fix bugs. The file named BUGDOS has been provided for this purpose. Simply type devicename.BUGDOS from the MDOS command line, and the title screen will appear on your terminal or remote computer running a terminal emulator (henceforth referred to collectively as "terminal"). If the terminal is not running, the computer will appear to lock up and in fact will have to be restarted with CTRL-SHIFT-SHIFT. Note that the title screen also declares the address range used by THE Bugger. This information will be useful when you are actually debugging programs.

To actually debug your program (henceforth referred to as the "task"), THE Bugger and the task will need to reside in the computer simultaneously. Furthermore since the disk supplied contains only DIS/FIX 80 object code for use with debugging tasks, and MDOS will accept only memory image files, a little preparation is necessary. The simplest method is to use Paul Charleton's LINK program (a fairware program) or any other suitable program that assembles object code into MDOS image files. LINK requires purely relocatable object code, so you must write your program without any AORG's - the file BUGGER provided is also relocatable. Thus you would load THE Bugger, then your object code, and then (in LINK) type @devicename.OUTPUTFILE. You now have a MDOS image file suitable for loading, that will start immediately with THE Bugger and allow you to take control of the task.

You can also load THE Bugger <u>after</u> the task but you would then have to program the task to branch to the beginning of THE Bugger. In addition you would lose the advantage of being able to use the BIAS command to easily start and find addresses in the task (see pages 12 and 14 for explanations on how to do this). Furthermore, many programs use buffer areas outside of, and usually following, the address range of the program, so loading THE Bugger after the task could be dangerous as well.

Another method of using THE Bugger with your task is to leave a block at least >17EA in length at the beginning, with the first instruction being a branch to the area after the block, and

then LINKing it without THE Bugger. You can then run it alone, and if there is a problem, quit it, load BUGDOS over it (it will only overwrite the block) and then debug. You can make the task quit without running by replacing the first branch with BLWP @>0000 (0420 0000, if you sector edit the MDOS image file).

All things considered, LINKing THE Bugger and the task together is much easier.

LOADING IN TI MODE

In TI (GPL) mode the usual method of loading will be via E/A Option 3. THE Bugger can be loaded before or after the task, which should contain only relocatable code or AORG code which does not interfere with the loading of THE Bugger. The file to use is BUGGER. The program name for entry is BUGGER. You will then see the title screen with the address range for THE Bugger, and can begin to debug the task. Note that the task should not auto-start or you will not able to enter THE Bugger.

If the task fills too much of the available space for relocatable programs (>A000 to >FFFF) or occupies, along with any possible buffers, at least part of each of the three 8K blocks beginning at >A000, >C000, and >E000 then you may use the file BUGGER6, which AORG's at >6000. Entry is still via defined address BUGGER.

For use with Extended Basic, four files are provided on the disk: BUGGERA, BUGGERC, BUGGERE, and BUGGERLOW (they have been archived and compressed into one file BUGGERARC using Barry Boone's Archiver 3.02 - you will have to reverse the process first). These are uncompressed object code files, suitable for loading in XB, and which AORG at >A000, >C000, >E000 and RORG respectively. Use of one of these files should enable you to load almost any XB assembly program that you write. If your assembly code is relocatable you can use BUGGERA, which should leave plenty of room for the XB program at the top of high memory. If your code AORGs in high memory, BUGGERLOW will fit into the available low memory space. BUGGERC and BUGGERE are provided for unusual situations, but be careful with BUGGERE since it leaves little room for the XB program at the top of high memory (if the XB program serves only to load the assembly code then there will be no difficulty). Of course these files are loaded via the usual CALL INIT :: CALL LOAD("DSK1.TASK) :: CALL LOAD("DSK1.BUGGERA") :: CALL LINK("BUGGER") etc. that are outlined in the XB manual, and will not be reiterated here.

THE COMMAND LINE

THE Bugger uses one letter commands to perform operations. Some punctuation marks are also used, in addition to the control key together with a character. The latter will be denoted with a ^ preceding the character, e.g. ^R. The commands may be followed by up to three parameters labeled P1, P2, and P3, and if so, as soon as you type the command letter, without the return key, a space and P1= will follow the command letter on the terminal screen. The parameters may be followed by a special terminator character to alter the command's function, and are listed with each command. The spacebar will take you from one parameter to the next, assuming multiple parameter commands, and the return key will begin execution of the command. ^. (Ctrl period) will abort most commands, as well as abort displays before they are finished. In all following examples, < and > will refer to user input. For example:

<D> P1=<2000><space> P2=<4000><return>

would indicate that you typed D, then P1= appeared on the screen. You then typed 2000 and pressed the space bar, following which P2 appeared on the screen. After that you typed 4000 and pressed the return key. This would disassemble a block of memory from >2000 to >4000. When you finished the two inputs only D P1=2000 P2=4000 would appear.

Please note that all numbers input by the user in THE Bugger must be in hex notation, but that you do not need to (indeed must not) type the > character usually used to denote hex. If you make a mistake in inputting a number you cannot backspace. However only the last four characters input are used by THE Bugger, so just keep typing until you get what you want. For example if after P1= you type 1234AB78, THE Bugger will set P1 equal to AB78. If you type less than four characters, THE Bugger will insert leading 0's to make a total of four, e.g. 1A will be set by the program to >001A.

In the functions allowing inspection/changing with one parameter input, a "-" will back up to the previous address, and the space bar will advance to the next.

COMMANDS

A -- INPUT ASCII TO MEMORY

Examples: <A> P1=<A000><space> P2=<A00E><return>
or <A> P1=<A000><space> P2=<return>

Allows for typing of ASCII text from the specified address in P1 to the address in P2 (in the first example, 14 characters from >A000 to >A00D inclusive). As you reach the limit of characters on one line (16), the cursor will jump to the next and display the current address, however you may just continue typing continuously. If you type <return> only for P2 (as in the second example), then the input will be transferred to memory beginning at P1 until it is terminated by ^. (control period). If the address range is terminated by V then the transfer is to VDP memory. (You can see text typed on the screen by using V and the location of the screen image table.) [Ed. Note: For those familiar with DEBUG, there is no G termination, since the program runs in MDOS as well as GPL mode, and there is no GROM in MDOS.]

B -- SET/RELEASE BREAKPOINT

Examples: <B> P1=<29A4><return>
or <B> P1=<return>
or <B> P1=<29A4><->
or <B> P1=<->

Sets or releases breakpoints at the address specified in P1. If the address is terminated by <return> then the breakpoint is set, if by a <-> then it is cancelled. If no address is specified in P1 then <return> will list all breakpoints currently active, and <-> will release <u>all</u> breakpoints. The list of current breakpoints carries the following prefixes:

"N" - normal breakpoint
"S" - special breakpoint
"P" - permanent breakpoint

(for explanation of special and permanent breakpoints, see page 17). These are one-word breakpoints and can be set consecutively.

The B command sets <u>normal</u> breakpoints, that is execution of the task will halt at the first breakpoint it reaches, and the breakpoint <u>will be released</u>. You will see on the screen the work "break" plus information from any switch options that have been turned on (see page 20 for these). As you use THE Bugger, be sure that there is always at least one breakpoint active before you set the task running, or you will not be able to take control again.

Note that the task halts <u>before</u> execution of the instruction at the specified address, although that instruction is listed (if disassembly is turned on).

C -- CRU INSPECT/CHANGE

Example: <C> P1=<1100><return>

Reads the contiguous 16 bits starting at the base address specified by P1 (e.g. >1100, the disk controller card). The bits are displayed in hex, and you may change them if you wish by typing the appropriate hex number. <return> will exit the command. [Ed. Note: The author's original intent was for the <space> termination to advance the CRU bits displayed to the next 16, however this may not work properly]

D -- DISASSEMBLE INTO OPCODES

Examples: <D> P1=<A000><space> P2=<A020><return>
or <D> P1=<A000><return>

Will disassemble memory starting at the address specified in P1 to the address in P2. The address of the start of each instruction, as well as the contents of that address are listed before the opcode. If only P1 is given, the disassembly starts at that address, and continues one instruction at a time while the space bar is being pressed, and ends when the return key is pressed. For termination of the block disassembly before it is finished, press ^. (control period).

E -- EXECUTE TASK

Examples: <E> P1=<return>
or <E> P1=<1BEA><return>

Begins execution of the task using the RTWP vectors, which have been set using the R command. Be sure to set them before using the E command the first time, as they have arbitrary values when THE Bugger is entered the first time. You may specify the entry address by inputting it before pressing <return>. However the WP vector will not be set correctly unless the task does it right at the beginning. If you change the entry address from that present in the R command, be careful that the WP is still the same. Note that when the task is halted with a breakpoint, the PC (program counter) is correct, so that you may use the E command without P1 to continue after inspecting whatever you wish.

F -- FIND WORD/BYTE EQUAL

Examples: <F> P1=<A000> P2=<B000> P3=<4142><return>
or <E> P1=<A000> P2=<B000> P3=<41><->

Searches from the address in P1 to the address in P2 for the hex number (word in first example, byte in second example) specified in P3. <return> as terminator will search on even addresses for the word value in P3 and <-> will search at each address, odd or even, for the byte value in P3. All matches will be listed, so if the address range is large and the number of matches is also, then the first will scroll off the screen.

K -- FIND WORD/BYTE NOT EQUAL

Examples: <K> P1=<A000> P2=<B000> P3=<4142><return>
or <K> P1=<A000> P2=<B000> P3=<41><->

operates exactly like the F command except that it lists all addresses and their contents that do not match. Since in most cases there will be a large number of "non-matches," the list may be very large. In this command ^. does not appear to stop the display - you will just have to wait for it to end.

M -- MEMORY INSPECT/CHANGE

Examples: <M> P1=<C000> P2=<C100><return>
or <M> P1=<C000><return>

Lists all memory contents from the address in P1 to the address in P2 in a block display, with the number of words per line dependent on the line length set by THE Bugger for the terminal. Or, as in the second example, dislays the memory content of the address in P1 and allows you to change it or accept its present value. In the latter case THE Bugger will display, for example:

C000= 5252

You may then press <return> to accept the value and go on to another command, <space> to accept the value and go on to the next address where the contents will be similarly displayed, <-> to accept the value and back up to the previous address, or input a new value and then terminate with <return>, <space>, or <-> to exit, go on, or back up. In the last three cases you would see, for example:

```
      M P1=C000
C000= 5252 <5151><return>
```

or

```
      M P1=C000
C000= 5252 <5151><space>
C002= 5353 <?>
```

or

```
      M P1=C000
C000= 5252 <5151><->
BFFE= 5050 <?>
```

(where ? would represent your next input).

The M command is one of the most useful in THE Bugger, since in debugging the task you may find that you need to change a value at a given location, or you may find that an instruction was wrong and you can then actually change it (if you can determine what the hex value would be) without changing the source code and reassembling.

You may inspect or change VDP addresses in exactly the same way by appending a V to the address, except that only byte values are given.

S -- SINGLE STEP

Example: <S> P1=<return>
or <S> P1=<6><return>
or <S> P1=<C000><->
or <S> P1=<D000><+>

Causes THE Bugger to single step using the special capabilities of the 9640, and is one of the most powerful commands of the program. If <return> is pressed without input for P1 then THE Bugger will step a single instruction. If a number is input for P1 and <return> pressed then THE Bugger will single step that number of instructions. If the input for P1 is terminated with a <-> then P1 becomes the address to single step to, and if it is terminated with a <+> then single stepping will proceed until the contents of address P1 are altered from the current state. Note that because single stepping is done at an exceedingly slow rate, it is dangerous to single step through VDP routines and the operating system of the computer.

In all cases, as each instruction is stepped through, the word "step" is written to the screen and then additional information is displayed if any or all of the three switches ^S, ^W, and ^D have been turned on. Please see page 20 for an explanation of what is displayed.

There is one case in which you cannot single step. If you have stopped at a permanent breakpoint, then THE Bugger's special code (an XOP) is ready to go back into the address pointed to by the PC. Since the S command also uses that location, the single stepping cannot be done and if you try, execution will in fact proceed until the next breakpoint.

T -- DISASSEMBLE AS DATA

Examples: <T> P1=<A000><space> P2=<A020><return>
or <T> P1=<A000><return>

Functions exactly as does the D command, only the disassembly is one word of DATA.

W -- WORKSPACE INSPECT/CHANGE

Examples: <W> P1=<return>
or <W> P1=<space>
or <W> P1=<A><return>
or <W> P1=<A><space>

Will display and allow changing of the workspace registers. If P1= is followed by <return> all sixteen registers of the WP pointed to by the R command and their values are displayed. If <space> is used then each register beginning with R0 is displayed and you are allowed to change it. <space> will take you to the next register whether or not there has been new input, and <return> exits the command. If < > is pressed after a register is displayed or changed the previous register is displayed. You may also enter a register number (in hex, single digit) and then that register is displayed immediately. Subsequently the command behaves as if you had started with R0.

X,Y,Z, BIAS VALUES

Examples: <X> A000 <1BEA><return>
or <Y> 1000 <2000><return>
or <Z> 0040 <3000><return>

X, Y, and Z are optional bias values that can be set independently. The previous value is displayed first (note the defaults above, set by THE Bugger) and you are then given the option to change it, or accept the present value. If the bias letter is appended to any hex value, e.g. <0000X>, input for a parameter, THE Bugger will set the value of the parameter to the sum of X and the input value. For example, using the values listed above, if you type 0000X for P1= then THE Bugger sets P1=1BEA, or if you type 1234Y for P2 then THE Bugger sets P2=3234.

As explained under the R command, this can be particularly useful if you have a list file with all relocatable addresses on it, as the beginning of your program will begin with address 0000. In the example above, X was set to 1BEA because the length of THE Bugger is 17EA bytes, and in MDOS it begins at 0400 if loaded first.

. -- DECIMAL TO HEX CONVERSION

Example: <.> <512><return>
terminal will display =0200 on the same line

simply converts the decimal number input into a four digit hex number and displays it. Whole numbers from 0 to 65535 can be input, but not negative numbers. If a number larger than 65535 is input, which would result in a hex number with 5 or more places, only the last four are displayed.

There is a minor bug in THE Bugger in that P1= should be displayed after you input . but it isn't.

, -- HEX TO DECIMAL CONVERSION

Example: <,> P1=<0200><return>
terminal will display 512 on the next line

Converts the number input into P1 (one to four digits) into a decimal number and displays it. In this case, numbers from >8000 to >FFFF are displayed as negative decimal numbers.

; -- HEX ARITHMETIC

Example: <;> P1=<1234> P2=<567><return>
terminal will display:
H1=1234 H2=567
H1+H2=179B
H1-H2=0CCD
H1*H2=0062 56EC
H1/H2=0003 Remainder=01FF

displays the four major mathematical functions using the values input for P1 and P2.

= -- FIND ASCII STRING

Example: <=> P1=<A000> P2=<B000>
Enter String to Find (30 char max)
<your string><return>

Searches for an ASCII string of up to 30 characters, starting with the address in P1 and ending with that in P2. It will list only the first match, or inform you that it is not found. THE Bugger begins the search at each address in the range P1 to P2, so that if the string begins with the address at P2, then a match is made.

@ -- MEMORY MAPPER

Example: <@> P1=<EF> P2=<6><return>
or <@> P1=<return>

Maps the physical page number in P1 into the execution page number in P2. If <return> only is pressed for P1 then the current page map is displayed (eight execution page numbers). A minor bug has occurred in that the execution pages are sometimes listed as 10 to 17 rather than 01 to 07.

You must be very careful not to map out the page(s) in which THE Bugger is contained, or the computer will freeze up. However it does allow you to investigate all the physical pages of the 9640, by mapping them into an unused execution page.

/ -- RESET THE BUGGER

Displays the title screen and re-initializes the RS232 card.

^R - RESET 9640

Resets the computer by performing a BLWP @0.

^B - SPECIAL BREAKPOINT

Examples: <^B> P1=<29A4><return>
or <^B> P1=<return>
or <^B> P1-<29A4><->
or <^B> P1=<->

Functions like the normal breakpoint (B). only after breaking to output the word "break" and any data signalled by the switch options it does not return to the command line, but keeps on executing the task, and the breakpoint is cleared. Pressing <return>, <-> or <data><-> function to list all the breakpoints, clear all the breakpoints, or clear one breakpoint, as with the B command. If a breakpoint of one type is set at a given address , it must be cleared before using another type at that address, or the first type will remain.

^C - PERMANENT BREAKPOINT

Examples: <^C> P1=<29A4><return>
or <^C> P1=<return>
or <^C> P1=<29A4><->
or <^C> P1=<->

Functions like the normal breakpoint, but will remain in memory and not be reset after it is executed. THE Bugger takes great care to ensure that a permanent breakpoint will be removed from the task long enough for the original instruction to be executed when an E command is performed directly after a permanent breakpoint. Pressing <return>, <-> or <data><-> function to list all the breakpoints, clear all the breakpoints, or clear one breakpoint, as with the B command. If a breakpoint of one type is set at a given address , it must be cleared before using another type at that address, or the first type will remain.

This command is especially useful when you wish to repeatedly examine data after the same address in the task, but under different circumstances.

^T - VDP STATUS REGISTERS

Example: <^T> P1=<return>

Displays the contents of the nine VDP status registers.

^U - VDP STATUS REGISTERS - DYNAMIC

Example: <^U> P1=<6><return>

Dynamically displays the contents of the VDP status register contained in P1.

^V - WRITE TO VDP REGISTER

Example: <^V> P1=<XXYY><return>

Writes the value YY to the VDP register XX

SWITCH OPTIONS

The following "switches" which in each case toggle between two modes, are available at any time. Default mode is OFF in all cases.

^E - COMPUTER MODE

Toggles the computer between MDOS and TI modes and maps itself accordingly to accomodate the mode switch.

^L - LINE LENGTH

Toggles the length of the output line between 40 (OFF) and 80 (ON) columns to accomodate different terminals or computers.

^P - PRINTER OUTPUT

This option is not available. In order to print your output you should use the terminal to log to disk or printer. If ^L is toggled to 80 columns, the output will be in neat 80 column format, even if your terminal screen is 40 columns and does not appear even. In any case most output is less than 40 columns, except for memory dumps.

^S - DUMP STATUS

Toggles the display of the values of WS, PC, and ST, as well as symbols "disassembling" the value of the first 6 bits of the status register. These are:

L - Logical greater than
A - Arithmetic greater than
E - Equal
C - Carry
O - Overflow
P - Odd parity

Note that for a breakpoint, the listed values for WP and ST are those prior to execution of the instruction at the breakpoint (the PC indicates the address of the instruction). For single stepping they are listed twice, both before and after execution of the instruction. Thus you will see on the second line the PC after the instruction and the new (possibly changed) values of WS and ST.

^W - DUMP WORKSPACE

If ON dumps display of the values of the workspace registers (pointed to by the WS of the R command). A slight bug is present here in that in the case of single stepping the values prior to execution of the instruction are listed but not after. You can still inspect the WS after the step by entering the W command without parameters.

^D - DUMP DISASSEMBLY

When this switch is ON, disassembly of the current instruction that has just been single stepped, or the next instruction to be carried out after a breakpoint has been executed, is displayed to the terminal, as is the address and its contents (first one, if the instruction takes two or three words). NOTE: each time you execute the D command this switch is turned off!

APPENDIX 1 - CONSTRUCTION OF A NULL MODEM CABLE

The author used a TI99/4A running FAST-TERM at 2400 baud and even parity as a terminal during the writing and debugging of this program. [Ed. Note: TELCO was tested and worked at 7E1. We were unable to get MASS TRANSFER to work because of inability to change parity] To connect your terminal's port, or the second computer's RS232/1 port to the host computer's (the one running THE Bugger) RS232/2 port you will need a special cable. If you have a modem cable and TI's RS232 Y cable/splitter you should be able just to connect the modem cable to the RS232/2 plug of the Y cable (which for some strange reason is labelled "PORT 1" on ours!). This worked in our system. You may not want to disable your modem cable however, so we are including the correct pin connections to make a dedicated cable. Please note that the information in the TI RS232 manual is incorrect.

Terminal or computer with term. emulator	Computer with THE Bugger
1	1
2	16
3	14
5	19
7	7
20	13

APPENDIX 2 - CHANGING THE BAUD RATE

To change the baud rate for THE Bugger's output, find the following code near the beginning of the program: 4B00 0008 0000. The 0008 occurs at byte >D0 of sector 1 of BUGDOS, b. >6C of s. 2 in BUGGER and BUGGER6, and b. >1A of s. >4 in the uncompressed XB files. Change the 0008 according to the following table:

BAUD RATE	CHANGE TO
110	0000
300	0002
600	0004
1200	0006
2400	0008
4800	000A
9600	000C
19200	000E

Note that if you make a change in an uncompressed file you must change the 7 to an 8 before the checksum at the end of the line.

T And J Software - LIMITED WARRANTY

T and J Software warrants THE Bugger Ver. 1, which it manufactures, to be free from defects in materials and workmanship for a period of 90 days from the date of purchase.

During the 90 day warranty period T and J Software will replace any defective product at no additional charge, provided the product is returned, shipping prepaid, to T and J Software. The Purchaser is responsible for insuring any product so returned and assumes the risk of loss during shipping.

Ship to:

T and J Software
515 Alma Real Drive
Pacific Palisades, CA 90272

WARRANTY COVERAGE - THE Bugger program is warranted against defective material and workmanship. THIS WARRANTY IS VOID IF THE PRODUCT HAS BEEN DAMAGED BY ACCIDENT, UNREASONABLE USE, NEGLECT, TAMPERING, IMPROPER SERVICE OR OTHER CAUSES NOT ARISING OUT OF DEFECTS IN MATERIALS OR WORKMANSHIP.

REPLACEMENT AFTER WARRANTY - After the 90 day Warranty period has expired you may return any original defective diskette, along with a check for $4.00 to cover shipping and diskette costs, and we will replace it.