

JOHN'S DEBUGGER

BY

JOHN BRODERICK, CPA

FOR
ASSEMBLY LANGUAGE PROGRAMMING
ON THE **APPLE II** COMPUTER

OPTIONS:

TRACE LOGIC W/ INSTRUCTIONS - NOTING
ALL JUMPS, BRANCHES, JSR, ETC.

STEP EACH INSTRUCTION DISPLAYING:
-ALL REGISTERS, STATUS AND POINTER
-ACCUMULATOR IN BINARY
-LAST 8 BYTES ON THE STACK
-DISPLAY ALL FLAGS SET
-DISPLAY WHAT IS IN ANY 12 MEMORY
POSITIONS WITH YOUR LABELS

BREAKPOINT BREAK ON KEYPRESS,
CYCLE COUNTER, ETC (6 OPTIONS) INCLUDES
TIMING DELAY FROM 0.0 TO 255 SECONDS
(ALL OF ABOVE SAVES ENTIRE PAGE OF STACK)

JOHN'S DEBUGGER

by

JOHN BRODERICK, CPA
BRODERICK & ASSOCIATES
PROFESSIONAL SOFTWARE HOUSE
8635 Shagrock
DALLAS, TEXAS 75238
(214) 341-1635

PURPOSE: TO ASSIST PROGRAMMERS WITH ASSEMBLY AND
MACHINE LANGUAGE PROGRAMMING TO HELP FIND
LOGIC ERRORS AND OTHER MACHINE LANGUAGE
BUGS THAT SOMEHOW GET INTO THE SYSTEM.

COMPUTER: APPLE II COMPUTER MANUFACTURED BY APPLE
COMPUTER, INC.

LANGUAGE: APPLESOFT & MACHINE LANGUAGE.

DOS: REQUIRES AT LEAST 1 DISK DRIVE.

MEMORY: 48 K IS REQUIRED

JOHN'S DEBUGGER IS A MACHINE LANGUAGE PROGRAM
THAT RESIDES IN MEMORY FROM \$8600 to \$9600.

INTRODUCTION

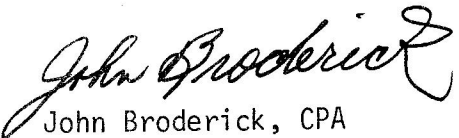
Hundreds of hours of my time has been expended with the design, development and programming of this debugging system so that each program may adhere to the high level of quality which I demand from computer software. And almost every critical decision was made keeping the user in mind so that these programs would not only function properly but be enjoyable and fun to use.

As a result you will find my debugger very educational for the beginning programmer - for he can see how each instruction affects such things like the stack and which flags are set and cleared to mention just a few of the displays.

Advanced programmers will like the various logic trace options along with the ability manually define the registers and then step through the monitor routines all the time seeing what is happening in up to 12 memory positions located anywhere in memory. With my debugger within a matter of hours I can thoroughly analyze someones's machine language routines and completely understand what they are doing and how it is being done.

My first trace program was written for a 16 bit 1410 Autocoder (40 K) - a big machine in 1967 for it took up over half of the entire computer dept. and almost supported a full time repairman. We've come a long way since then and I expect the next generation to be full of more surprises and I hope that my work can contribute to them.

Thanks for purchasing my software - I really appreciate it and want you to know that if you have any questions just feel free to call me.


John Broderick, CPA

INDEX

HOW TO RUN - AT A GLANCE	(Trace & Step)	1
	(Breakpoint)	1
	(Store Demo)	1
BREAKPOINT OPTIONS		2
TRACE & STEP OPTIONS		2
BREAKPOINT & HOW TO USE IT		3
BREAKPOINT (Additional Notes)		5
HOW TO RUN TRACE & STEP WITHOUT LABELS		6
HOW TO RUN TRACE & STEP WITH LABELS		8
PROGRAMS ON THE DISK INCLUDED WITH THIS DOCUMENTATION		10
TIMING DELAY ROUTINE AND HOW TO USE IT		11
MEMORY USAGE		12
REGISTRATION FORM		13
DISPLAY INFORMATION	Appendix A	

*Original Copy
Michael Schaffer
Feb 2024*

HOW TO RUN - AT A GLANCE

TRACE & STEP

RUN WITHOUT LABELS

BRUN JOHN'S DEBUGGER

or

*8600G (if already in memory)

RUN WITH LABELS

RUN LABEL MANAGER

or

*8600G (if already in memory) *9000G is same entry
*9003G after M)ONITOR break.

BREAKPOINT

BLOAD JOHN'S DEBUGGER

JSR \$8603 (put in your program to stop processing)
RETURN with *8606G after M)ONITOR break.

STORE DEMO FOR TRACE & STEP

RUN DEMO MANAGER

NOTE: IF HIMEM is not set to less than \$8600 and you are using an assembler or an applesoft program after you load in JOHN'S DEBUGGER then you will probably place your program on top of the debugger and cause strange output displays.

INTEGER BASIC and many assemblers use the HIMEM at 4C & 4D
set by 4C:FF 85

APPLESOFT use HIMEM at 73 & 74 set by 73:FF 85 or
HIMEM: 34304

The trace and step program will automatically set basic HIMEM to \$85FF if it discovers that it is still set to \$9600

NOTE: THIS DISK CAN BE COPIED - SO WE RECOMMEND THAT YOU MAKE EXTRA COPIES FOR EACH PROGRAM YOU ARE DEBUGGING.

BREAKPOINT OPTIONS

<u>O)PTIONS</u>		<u>TIMING DELAY</u>	<u>DISPLAY INFORMATION</u>
'0'	NO ACTION TAKEN (always return)	-	-
'1'	STOP ALWAYS (every time thru)	-	ON
'2'	STOP ON (ESC) KEYPRESS	ON	-
'3'	" " " "	ON	ON
'9'	NEVER STOP (to be used with basic) poke 34336,9 : CALL 34307 \$8620 \$8603	ON	ON

ONCE YOU HAVE STOPPED KEYPRESS

- '0' O)PTION is reset by next number keyed (0,1,2,3,9)
- 'M' M)ONITOR exit and return with *8606G to continue
- 'R' R)ESET all options to normal: always stop first time thru and reset delay and counter to zero
- 'C' C)YCLE COUNTER is reset to zero
- 'S' S)ECONDS DELAY are set by next number keyed (0-9)
- 'T' T)ENS OF SECONDS DELAY are set by next number keyed (0-9)

=====
TRACE & STEP OPTIONS

KEYPRESS AT ANY TIME DURING RUNNING

- 'T' T)RACE LOGIC with execution and displaying one page of instruction leaving spaces for all break in logic path like jumps, jsr, branches taken,etc
- 'S' S)TEP INSTRUCTION one at a time displaying all of the information listed on the cover of this documentation
- 'L' L)INE only one line is traced or stepped at a time
- (ESC) ACTIVATES continuous tracing or stepping until any key is pressed.
- 'M' M)ONITOR EXIT to examine or change memory *9003G to return and continue processing.

BREAKPOINT - AND HOW TO USE IT

I really don't know where to start in explaining BREAKPOINT because it does so much and has so many options and different combinations of options that if you are not careful when using them or the program the processor may disappear into never land. If that happens I explain how to manually reset the options.

ENTER

WHAT THIS DOES

BLOAD JOHN'S DEBUGGER This places the breakpoint program into memory. If you are subsequently using applesoft or an assembler that stores data below DOS then it is best to make sure that HIMEM is set to \$8600 or less or JOHN'S DEBUGGER will be wiped out.

I always *4D:85 which sets the high order byte of HIMEM to less than 86. Now I can run the assemble language assembler which I purchased.

Since the breakpoint program resides in JOHN'S DEBUGGER it has been loaded and is ready to use.

JSR \$8603

You must insert this instruction in a place in your program where you are debugging and would like to just stop processing (not necessarily break) and analyze the data in memory and in the regs and then at your option break or continue. HIT RETURN to continue processing or press 'M' to go into the monitor (return with *8606G).

Now please made a note of the BREAKPOINT options on one of the preceding pages. Say we don't want the processor to stop but to continue with the timing delay on(say set to 1 second) and then we can see out program running slower. OK lets do it.

'0' LETTER

Nothing. But the next number enter will change the option. '0' stands for option.

'3'

Scrolls the display and now right below the break address you should see the number '3' in inverse. This means to not stop processing unless the escape key is pressed but to use DELAY and DISPLAY.

NOW THIS IS IMPORTANT

if we don't set the timing DELAY to long enough to sense the keyboard strobe then the stop will not occur since I do not clear the strobe because you may be using it in your program.

BREAKPOINT - AND HOW TO USE IT (Cont)

ENTER

WHAT THIS DOES

'S'

Nothing. But it does let the debugger know that the very next keypress must be a number between 0 and 9 which will reset the DELAY to that number of seconds (actually times 5/6). So lets enter the delay in seconds.

'2'

Scrolls up and DISPLAYS. You should now see the delay window read '2.0' (2 seconds and 0 tenths of a second). To set the DELAY for less than 1 second just enter 'T' followed by the number.

ANY KEY

Will return you to your program and process using option 3 (DELAY AND DISPLAY).

(ESC)

Will now stop processing. Otherwise processing will continue. Please note here that if you used option 2 (DELAY ONLY) then nothing will appear on the screen each cycle (JSR \$8603) but your own programmed data - but the DELAY would still slow up your program so we can sense the ESCAPE key being pressed.

*8620:01

WILL MANUALLY SET THE BREAKPOINT OPTION TO '1' WHICH WILL CAUSE THE PROGRAM TO STOP ON EVERY JSR CYCLE.

DON'T FORGET TO FILL OUT AND RETURN THE REGISTRATION FORM - PG 13

I had not intended to release the breakpoint counter options with this release because they will require much more time to enable the user to use them easily. My next release will include them but it will sell for a retail price of about twice what this version sells for.

Still you can use the counter options if you wish to manually set memory position *8621 (check counter low byte) and \$8622 (check counter high byte) to the cycle number where you want a stop to occur. At this point you will have the same options as before however before you debug again you should press 'C' which will reset the cycle counter which is displayed on the screen just below the option and seconds inverse display. This will not effect the check counter - it will stay set to the bytes you entered.

You can set the counter options 4,5, and 6 thru the normal process of entering a '0' and then the number.

COUNTER OPTIONS:

- 4 - Stop only when the check counter equals the cycle counter.
- 5 - Stop only when the check counter equals the cycle counter but use the DELAY subrouting to slow processing.
- 6 - Stop only when the check counter equals the cycle counter but use the DELAY and DISPLAY routines each cycle.

HOW TO RUN TRACE & STEP WITHOUT LABELS

ENTER WHAT THIS DOES (((SCREEN MESSAGES)))

BRUN JOHN'S DEBUGGER This loads my binary debugging assembly language program into memory starting at location \$8600 to \$95FF (HEX) and begins execution of the first instruction at address \$8600 (which is a jump to my trace program starting at \$9000.

(((ENTER ADDRESS TO BEGIN TRACE)))

'XXXX' HERE YOU ARE TO ENTER 4 HEX CHARACTERS WHICH MAKE UP THE MEMORY ADDRESS OF THE FIRST INSTRUCTION OF YOUR PROGRAM.

In other words, I assume that your assembly language program is already in memory and waiting to be executed-so instead of *XXXXG which will go to the first line in your program and execute, you instead BRUN JOHN'S DEBUGGER which will ask you for the memory address which you would like to begin executing and tracing from that point forward.

Some examples to enter are:

03A5	4C5B
0800	F800
1CFA	4000

That's all there is to it -- DON'T HIT RETURN -- The debugger will grab that forth character out of your hands and immediately start stepping with the first instruction which can be found at that address.

Now the options can be entered depending on what you need:

- 'T' ----- Will trace the logic path listing your instruction and its memory position. You will notice that every time that there is a change to a new address the debugger will skip a line (for jumps, jsr, rts, branches taken, etc).
- 'S' ----- Will cause the debugger to begin immediately after you pressed the 'S' to display all of the step and breakpoint information. Please see the section of this documentation entitled "Display Information"
- 'M' ----- Breaks processing and puts you into the MONITOR. Now you can change or do anything that you want or need to do - you have complete control.
- *9003G ----- RETURNS you to continue to process your program at the exact point you were before you made your temporary look into the monitor.

HOW TO RUN TRACE & STEP WITHOUT LABELS - (Cont'd)

ENTER

WHAT THIS DOES

Continued from preceding page.

'L'

Stands for LINE and what this does is to trace or step just one line at a time. Pressing an 'L' will not change or disturb anything so use it freely as much as you wish.

(ESC)

THE ESCAPE KEY is very useful for debugging. What it does is to trace or step continuously without stopping for the normal page display or whatever.

TO STOP the trace or step after pressing (esc) just press any key and then you can continue tracing.

SAY YOU DON'T WANT TO "BRUN JOHN'S DEBUGGER" BECAUSE YOU ARE IN THE MONITOR.

ENTER:

*8600G

Provided that you have already 'BLOADED JOHN'S DEBUGGER' which would have loaded the program into memory then the next question on the screen will be:

(((ENTER ADDRESS TO BEGIN TRACE)))

Once the debugging program is loaded any of the four instructions below will begin execution:

*8600G or CALL 34304
*9000G or CALL 36864

Since the first instruction at location \$8600 is a jump to the trace program located at \$9000.

DON'T FORGET
'M'

Will put you into the monitor - to change the:

ACCUMULATOR *860B:XX
X REGISTER *860C:XX
Y REGISTER *860D:XX
PROCESSOR STATUS *860E:XX
STACK POINTER *860F:XX

*9003G

Will begin processing at the next instruction and you will notice that the accumulator or registers will be changed to what you want. This is a great way to find out how the various monitor routines work. Just put a jump anywhere in memory to the monitor routine that you want to test and then press 'M' right after the JMP so that you can change and enter the necessary data..

HOW TO RUN TRACE & STEP WITH LABELS

<u>ENTER</u>	<u>WHAT THIS DOES</u> <u>(((SCREEN MESSAGES)))</u>
RUN LABEL MANAGER	Loads JOHN'S DEBUGGER into memory at \$8600 and then proceeds to poke into the debugger the labels and related memory positions. (((WOULD YOU LIKE YOUR LABELS TO BE POKED INTO JOHN'S DEBUGGER))) Answer 'N' NO to this the first time thru because you have not yet defined your labels.
'N'	You will now see on the screen the label menu. So let's try out a few, first enter 'L' to list the labels.
'L'	Reads the label file off of the disk. In this case it is reading my sample file which I have setup. Now let's try the 'U' to update one of the 12 labels.
'U'	(((ENTER A=ADD C=CHG D=DELETE))) My manager wants to know whether you want to add a label in case you have not used all 12 of them, change one of the labels or memory positions, or delete a label so a blank will appear during debugging. Let's try a 'C' -Notice that you didn't have to hit return here.
'C'	(((CHANGE LABEL NUMBER))) So why don't you enter number 7.
'7'	(((ENTER LABEL # 7))) LABEL MANAGER wants to know what your 8 character label is to be which will replace the one in the file. Enter HIMEM or whatever wicked word you can think of.
'HIMEM'	(((HEX MEMORY # 7))) Now the manager wants to know <u>what</u> memory position from \$0001 to \$FFFF in hex should be analyzed and the byte displayed next to your label in the debugging program. Since we want to know what the high order byte of himem is why don't we enter 4D.
'4D'	Notice that it is not necessary to enter leading zeros.

HOW TO RUN TRACE & STEP WITH LABELS (Cont)

<u>ENTER</u>	<u>WHAT THIS DOES (((SCREEN MESSAGES)))</u>
	Continued from previous page.
'END'	You were so anxious to see my program work that you couldn't wait so go ahead and enter 'END'. This return you to the MENU so why don't we go ahead and poke the labels in the debugger.
'p'	The disk will read the labels off of the file and poke them into memory - this takes only a few seconds. The sound from the disk drive will be the closing and locking of the label file. At this point you can hit reset or you can allow my final goodbye question to properly sent you on your way.
But what do I do now?	<u>NOW JOHN'S DEBUGGER IS IN MEMORY AND READY TO RUN.</u> How do I run it you ask? Now turn to my documentation entitled: "HOW TO RUN TRACE & STEP WITHOUT LABELS" Do not, I repeat DO NOT perform the first step on that page because if you do you will load the debugger program right over you labels and have to poke them in again. Remember they are already in memory and ready to run so from this point on we can enter the 4 character hex address and use all of the options available for the trace without labels. The trace and step program is not aware of the fact whether or not your labels will be displayed in the DISPLAY sub-routine.

PROGRAMS ON THE DISK INCLUDED WITH THIS DOCUMENTATION

JOHN'S DEBUGGER

This is my primary debugging program. It is written 100% in assembly language and loads in starting at hex address \$8600. It consists of four separate programs:

1. BREAKPOINT - which is used to stop execution of your program and then resume. This program uses the DISPLAY and DELAY subroutines. (Memory \$8600 - 8BA0)
2. TRACE & STEP is completely separate from the BREAKPOINT in that it is primarily concerned with the proper execution of each instruction within the program itself. (\$9000-9500)
3. DISPLAY is a subroutine that is used by both of the two programs above. It display all of the information that is listed on the cover of this documentation. (\$8BAA-8FFF)
4. DELAY is a stand alone delay subroutine that can be set to cause an internal wait from 0.0 seconds up to 255 seconds. (\$95C0- 95FF)

DON'T FORGET TO FILL OUT AND RETURN THE REGISTRATION FORM ON PAGE 13

LABEL MANAGER

This program was written in applesoft to do all of the file management for the labels and memory positions which are used by the DISPLAY subroutine. Its sets up labels, deletes them. I hold the label manager to be completely responsible for all of the information under his care.

SURPRISE #1

Run this program and you will enjoy the extra program that I have included as a freebie.

SURPRISE #2

Same as the above except this program is much more complicated and requires thought on your part.

TIMING DELAY ROUTINE AND HOW TO USE IT

Within JOHN'S DEBUGGER is a very useful timing delay that you can use at any time. All it is is a series of loops that delay the processing of your program. There are three loops which need to be set and once they are set you need not set them again. Keep in mind though that I have set these loop and use them during debugging so you may need to relocate the program somewhere else in memory.

<u>ENTER</u>	<u>WHAT THIS DOES</u>
*95C0:01 01 01	sets the timing delay to 0.0 - no delay.
02 FF FF	Sets to 5/6 of 1 second
03 FF FF	2 seconds
04 FF FF	3 seconds
01 40 FF	.25 seconds
01 80 FF	.50 seconds
01 C0 FF	.75 seconds
03 80 FF	Sets to: (03) (80) 2 seconds x .5 seconds = 1 second x 5/6 seconds .833 seconds
JSR \$95C6	Will go to the delay subroutine and delay for the amount of time set above and then return to the original location.
JSR \$95C6	Will again cause the same delay as above-it is not necessary to reset the input positions because the delay subroutine stores them each time in memory positions \$95C3 to 95C5.

MEMORY USAGE

Extreme care has been taken to utilize only those portions of memory and monitor routines which are necessary. Below lists the memory that is being used by JOHN'S DEBUGGER.

PAGE ZERO:

\$ 3A Program Counter Low Address
\$ 3B Program Counter High Address

PAGE ONE to PAGE EIGHTY SIX (\$8600)

NONE

PAGE EIGHT SIX TO NINTY SIX (\$8600 - 9600)

All of this memory is utilized by the Debugger

MONITOR ROUTINES USED:

F8D0 Monitor instruction Display
FBFD Vidio Out
FC58 Home
FD35 Read Key
FD8E Carriage Return Out
FF3A Ring Bell
FF65 Monitor Entry
FF69 Monitor Entry

JOHN BRODERICK, CPA
BRODERICK & ASSOCIATES
PROFESSIONAL SOFTWARE HOUSE
8635 Shagrock
Dallas, Texas 75238

REGISTRATION FORM

I sure would appreciate it if you would take the time right now to fill out this form and return it to me with any additional comments or suggestions which you may have regarding JOHN'S DEBUGGER. Also I will put your name on my mailing list so that you can be informed of all improvement and new versions.

DATE PURCHASED: _____

PURCHASED FROM: _____

SERIAL NUMBER: _____

YOUR NAME: _____

YOUR ADDRESS: _____

CITY: _____

STATE: _____

PHONE _____

SUGGESTIONS AND COMMENTS:

DISPLAY INFORMATION

for the two programs BREAKPOINT and TRACE & STEP

```

A      4000 20 00 50      JSR $5000
B      ACC=97 X=21 Y=00 STATUS=C9 STACK=E7
C      ACC (BINARY) '1001 0111'      BREAK $ 4002      G
D      ON STACK: 01 20 00 F9 00 00 00 00 1-0.0      H
E      FLAGS SET:  NEG ORV DEC CRY      0000      I
F      05 LABEL #1      FA LABEL #5      2C LABEL #9
      44 LABEL #2      7D LABEL #6      20 LABEL #10
      21 LABEL #3      EE LABEL #7      02 LABEL #11
      CA LABEL #4      A0 LABEL #8      FF LABEL #12

```

- A Hex address of the instruction executed in TRACE & STEP. In the BREAKPOINT program it is the place in your program where you JSR \$8603 to breakpoint
- B ACC is the accumulator, X & Y are the two registers, STATUS is the processor status-this lets us know the condition of the flags, and STACK is the stack pointer.
- C The accumulator displayed in binary.
- D The last 8 bytes pushed on to the stack-(01) is last one pushed.
- E FLAGS SET is the same as STATUS except I have shown you that a 'C9' means the the 4 flags above are now on and set.
- F '05' is the byte that is in memory at a predetermined position which you defined the the program label manager. LABEL #1 is a 8 character label which you are to make up in LABEL MANAGER.
- G Used only in breakpoint to display place of break in your program.
- H 1 means break option 1 (always break & display). 0.0 tells us the the DELAY routine in BREAKPOINT is set to 0 seconds and 0 tenths of seconds.
- I Is the BREAKPOINT cycle counter.