Larry Fish
123 E. Arkansas
Denver CO 80210

Troubleshoot Your Software

... a trace program for the 6502

Here's a rather significant piece of software for you 6502-types. It should make your software debugging a real fun exercise (who am I kidding?). A little note of caution is in order. Larry mentioned that he did assemble the program by hand, therefore the format isn't directly compatible with MOS Technology's Cross Assembler . . . I guess there's only one, right? — John.

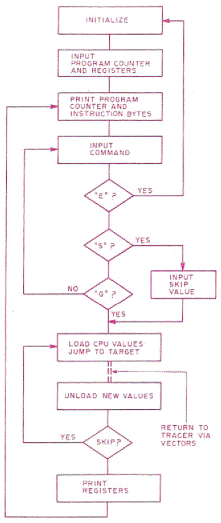
INITIALIZE
INPUT PROGRAM COUNTER AND REGISTERS
PRINT PROGRAM COUNTER AND INSTRUCTION BYTES
INPUT COMMAND
"E" ?
YES
"S" ?
YES
INPUT SKIP VALUE
NO
"G" ?
YES
LOAD CPU VALUES JUMP TO TARGET
UNLOAD NEW VALUES
RETURN TO TRACER VIA VECTORS
YES
SKIP?
PRINT REGISTERS

Fig. 1. Program flowchart.

One of the most frustrating parts of programming is the program that doesn't do what it is supposed to do. This is especially true with machine language programs where a single misplaced bit in a forgotten register can destroy a whole program. The usual arsenal of debugging tools include setting break points, examining memory, and careful analysis. These techniques are fine if you have lots of time and patience, but there is a simpler method: Tracer.

Overview

Tracer is a program that provides a simple, straightforward method of debugging faulty programs. It allows you to single-step execute each instruction in a program under software control. Tracer prints each address, each instruction, and after each execution, prints the contents of the registers. All branches and jumps are followed or not followed in exactly the same manner as normal program execution. The Tracer program also allows you to skip ahead, executing any number of instructions at almost full speed. This allows you to skip past subroutines and long program loops.

Most tracer or emulator programs are exceedingly long and complex since they must duplicate in software all hardware functions of the CPU. Large chunks of memory must be used to imitate addressing modes, follow branches and duplicate instructions. Tracer, on the other hand, is simple and requires only 300 bytes of memory.

The secret to Tracer's simplicity is the hardware timer that is available on the TIM, KIM-1, and JOLT ROMs. This timer is tied directly to the CPU clock and can be set to interrupt the CPU anywhere between one and 262,144 clock cycles. Tracer uses this clock to control the CPU's execution of each instruction.

The program uses a dummy Program Counter, Stack Pointer, X, Y and Status Registers. At the beginning of a trace run the user types values into these registers. With each single execution the Program Counter and register values are loaded into the CPU. The timer is then set and the CPU jumps to the target instruction in the program being debugged. At this point, the timer goes off and CPU is interrupted mid-instruction. The processor as a part of its normal interrupt sequency finishes the instruction, saves the program counter and status register, and jumps back to Tracer, which finishes the process of saving the resulting registers. In this way Tracer steps its way through the target program.

Operation

Once Tracer is started it types a P character. This is a request for the starting address and register values of the program to be debugged. The user simply types in these values in the following order: Program Counter, Status Register, Accumulator, X register, Y register, and Stack Pointer. Tracer automatically supplies spaces between each value typed. After the Stack Pointer is typed, Tracer prints the first address and one, two, or three bytes of instruction. Tracer is now ready for a command. The following commands control the execution of the trace sequence: Typing a G command causes the program to execute a single instruction. Typing an E command causes the program to *escape* from the current trace sequence so that new values maybe loaded. Typing an S command allows the program to skip ahead at nearly full speed from the current location to a preset

```
 1.  .: 1FC0 00 00 00 00 FE
 2.  .G
 3.  P 7000 00 00 00 00 FA
 4.  7000 85 F9      G 22 00 00 00 FA
 5.  7002 A9 23      G 20 23 00 00 FA
 6.  7004 D0 55      G 20 23 00 00 FA
 7.  705B 85 FE      G 20 23 00 00 FA
 8.  705D D8         G 20 23 00 00 FA
 9.  705E 4A         G 21 11 00 00 FA
10.  705F 86 FA      E
11.  P 7000 00 00 00 00 FA
12.  7000 85 F9      S705B
13.    20 23 00 00 FA
14.  705B 85 FE      G 20 23 00 00 FA
15.  705D D8         G 20 23 00 00 FA
16.  705E 4A         E
17.  P↓
18.  .
```

Example 1. Tracer execution. User input material is underlined. Line numbers are added for clarification.

location.

Example 1 will clarify the usual trace sequence.

Lines 1 and 2 are the starting procedure used by the TIM and JOLT ROMs. Tracer begins on line 3. When the program is started, Tracer responds with a Carriage Return, a Line Feed, and prints P followed by a space. The user then types in the program counter of the first address in the target program. This is followed by the Status Register, the Accumulator, X register, Y register and the Stack Pointer. As you can see, Tracer is now set to begin execution at address 7000, with all registers set to 00 and the Stack Pointer set to FA. After the Stack Pointer is typed in, the program prints another Carriage Return and Line Feed, then prints the Program Counter of the first location and one, two, or three bytes of instruction. The program is now waiting for a command. Typing a G as illustrated in line 4 causes Tracer to execute a single instruction, print the resulting registers and print the new Program Counter value and instruction bytes. Spacing is automatically set for one, two, and three byte instruction. Lines 4 through 9 illustrate a short trace sequence. Notice that the register values change in response to certain instructions. Also, in lines 6 and 7 Tracer follows an 85 byte forward branch.

Line 11 illustrates the use of an E command to restart the sequence with a new address and/or new register values. When E is typed the program prints P and the user types in the new values.

An S command allows the execution to skip ahead at almost full speed, without printing registers, instruction bytes, or the program counter. This is accomplished (as in line 12) by typing S and then typing the stop location. This *must* be the address of the *first* byte of an instruction, otherwise the program will run away with unpredictable results. If you wish to escape from Tracer back into the TIM or JOLT monitor, type an E command, then a Carriage Return (lines 16, 17 and 18). A period indicates that you are back in the monitor.

Debugging Hints: Naturally, the exact debugging procedure depends on the program and the nature of the fault. Bad programs give clues. They will often execute parts of the routine correctly before blowing up. The usual trace procedure involves starting Tracer at a point in the program that is clearly ahead of the fault. It is then a simple matter to single step until an error is found. If the error is subtle, think out what each instruction is supposed to do; then carefully watch the registers after each instruction.

Branches are a common source of problems. They can be tested by typing an E command and running through the branch with both branch and nonbranch conditions. Check to see that the branch lands on the *first* byte of the intended instruction.

Tedious subroutines involving hundreds of steps away from the main body of the program can be handled using the skip command. By setting Tracer to skip to the first instruction immediately following the subroutine call, Tracer will execute the subroutine at almost full speed and return with all of the registers set by the subroutine.

Input and output routines present special problems for the skip command. For example, if you use the S command to skip through a subroutine that inputs a character from the keyboard, the character loaded will be incorrect. This is caused by the fact that Tracer does not execute at full speed (about 12,000 instructions per second in the skip routine) and time constants used by the input routines are altered. If the correct character must be used to test a portion of a program, it can be hand loaded by resetting a register or loading the character directly into memory. Similar speed-related problems can be handled in the same way.

Tracer is written to run directly on an MOS Technology 6502 TIM or JOLT system having at least 300 bytes of memory starting at location 1F00 hex. It also uses the 55 bytes of unused memory on the 6530 (TIM-JOLT) chip. Adapting the program to KIM-1 systems should be a simple matter since KIM has two timers. Just change the input and output subroutine calls and the timer address. Tracer could be rewritten for any system that has a hardware timer (the flowchart in Fig. 1 should prove an aid in such an effort). For systems that don't have built-in timers, MOS Technology has several timer-port combinations for under $20. Also possible is some kind of TTL flip-flop counter to interrupt the processor after several cycles. Further information on programming and addressing the MOS Technology timer can be found in MOS Technology's hardware and software manuals and in the JOLT manual. ■

Program Listings — Tracer. (Note: Not all zero page locations are consecutive.)

```
Location    Contents           Label   Inst.   Operand      Comments
00D6        00                 XPCL                         ;Storage for Program Counter
00D7        00                 XPCH                         ;Accumulator, Stack Pointer
00D8        00                 XSTA                         ;and X,Y and Status Reg.
00D9        00                 XACC
00DA        00                 XX
00DB        00                 XY
```

```
00DC    00              XSP
00DD    01              BITO                          ;Bit Masks
00DE    07              BIT012
00DF    04              BIT2
00E0    08              BIT3
00E1    10              BIT4
00E2    80              BIT7

00E6    FF              FLAG                          ;Flag to indicate multiple
                                                      ;instructions (FF=clear)
00F4    00              STO                           ;Stores #of bytes used by opcode
00F5    00              SPACER                        ;Used to calculate #of spaces
                                                      ;after opcode printout
                                                      ;(Note 00E6, 00F4 & 00F5 are
                                                      ;within TIM's zero page,but
                                                      ;are unused by TIM

                        ;MAIN BODY OF PROGRAM
1F00    A9 9C           BEG     LDA     #9C           ;Initialize vectors and FLAG
1F02    8D F8 FF                STA     UINT
1F05    A9 1F                   LDA     #1F
1F07    8D F9 FF                STA     UINT+1
1F0A    A9 FF                   LDA     #FF
1F0C    85 E6                   STA     FLAG
1F0E    20 8A 72                JSR     CRLF (TIM)    ;Input PC,S,A,X,Y, & SP
1F11    A9 50                   LDA     "P"
1F13    20 C6 72                JSR     WRT (TIM)
1F16    20 77 73                JSR     SPACE (TIM)
1F19    20 A4 73                JSR     RD0A (TIM)    ;Input 2 byte hex address
1F1C    20 DE 1F                JSR     ALPC
1F1F    A0 00                   LDY     #00
1F21    20 77 73        P1      JSR     SPACE (TIM)
1F24    20 B3 73                JSR     RDOB (TIM)    ;Read one hex byte
1F27    99 D8 00                STA(Y)  XSTA          ;Indexed store of reg.
1F2A    C8                      INY                   ;Count
1F2B    C0 05                   CPY     #05           ;Test for 5 reg. loaded
1F2D    D0 F2                   BNE     P1            ;Loop until done
1F2F    20 E5 FF                JSR     PPC           ;Print Program Counter
1F32    B1 D6                   LDA(Y)  XPCL          ;Pick up opcode of 1st
                                                      ;instruction through indirect
                                                      ;pointer
                        ;This routine calculates the number of bytes required
                        ;by each opcode

1F34    F0 25                   BEQ     1BYTE         ;Tests for BRK inst.
1F36    C9 60                   CMP     $60           ;Test for RTS
1F38    F0 21                   BEQ     1BYTE         ;Branch if true
1F3A    EA                      NOP
1F3B    24 E0                   BIT     BIT3
1F3D    F0 0E                   BEQ     HALFOP
1F3F    24 DF                   BIT     BIT2
1F41    D0 16                   BNE     3BYTE         ;Branch if 3 Byte op
1F43    24 DD                   BIT     BITO
1F45    F0 14                   BEQ     1BYTE         ;Branch if 1 Byte op
1F47    24 E1                   BIT     BIT4
1F49    F0 0F                   BEQ     2BYTE         ;Branch if 2 Byte op
1F4B    D0 0C                   BNE     3BYTE         ;Branch if 3 Byte op
1F4D    24 DE           HALFOP  BIT     BIT012
1F4F    D0 09                   BNE     2BYTE         ;Branch if 2 Byte op
1F51    24 E1                   BIT     BIT4
1F53    D0 05                   BNE     2BYTE         ;Branch if 2 Byte op
1F55    24 E2                   BIT     BIT7
1F57    D0 01                   BNE     2BYTE         ;Branch if 2 Byte op
1F59    E8              3BYTE   INX                   ;Count #of Bytes
1F5A    E8              2BYTE   INX
1F5B    E8              1BYTE   INX
1F5C    86 F4                   STX     STO           ;Save count

                        ;This routine prints the opcode and operand
                        ;and a variable number of spaces according to the # of bytes
1F5E    A9 07                   LDA     #07           ;Initial spacer
1F60    85 F5                   STA     SPACER
1F62    B1 D6           P2      LDA(Y)  XPCL          ;Load op index indirect
1F64    20 B1 72                JSR     WROB (TIM)    ;Print it
1F67    C8                      INY                   ;Count
1F68    C4 F4                   CPY     STO           ;Test if done
1F6A    F0 0C                   BEQ     A1            ;Branch when done
1F6C    C6 F5                   DEC     SPACER        ;Subtract three spaces
1F6E    C6 F5                   DEC     SPACER
1F7D    C6 F5                   DEC     SPACER
1F72    20 77 73                JSR     SPACE (TIM)
1F75    4C 62 1F                JMP     P2            ;Loop
1F78    A6 F5           A1      LDX     SPACER        ;Routine prints spaces
1F7A    20 77 73        A2      JSR     SPACE (TIM)
1F7D    CA                      DEX                   ;Count spaces
1F7E    D0 FA                   BNE     A2            ;Loop until done
1F80    20 C0 FF                JSR     COM           ;Input and process commands
1F83    20 77 73                JSR     SPACE (TIM)

                        ;This routine sets up the registers to execute the single
                        ;instruction. The TIM timer is used to interrupt the CPU
                        ;so that only one instruction is executed.
```