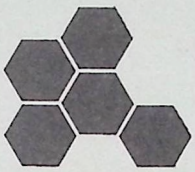


Customer
Support
Operation

ICE-51

VERSION 2.0

March 1984

122229-001

AN INTEL TECHNICAL REPORT

FROM SYSTEM SUPPORT SERVICES

ICE-51

VERSION 2.0

March 1984

122229-001

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051
Mail Stop SC6-325

Intel Corporation (U.K.) Ltd.
Piper's Way
Swindon, Wiltshire SN3 1RJ
United Kingdom

Intel Japan K.K.
5-6 Tokodai, Toyosato-machi
Tsukuba-gun
Ibaragi-Pref. 300-26
Japan

An Intel Technical Report
From Software Support Services

CONTENTS

Contents
Index to Symptoms in Troubleshooting Guide

OVERVIEW.....1
DIFFERENCES BETWEEN ICE-51 VERSION 1.3 AND ICE-51 VERSION 2.01
DEFINE COMMAND.....2
EVALUATE COMMAND.....2
GO/STEP COMMAND.....2
HELP COMMAND.....2
LOAD COMMAND.....2
SAVE COMMAND.....3
SYMBOLS COMMAND.....3
NEW COMMANDS.....3
DOMAIN COMMAND.....4
MODULES COMMAND.....4
LINES COMMAND.....4
ENABLE/DISABLE LINES AND ENABLE/DISABLE SYMBOLS.....4
REMOVE MODULE COMMAND.....5
CWORD/DWORD/XWORD COMMANDS.....5

ARTICLES.....7
DEBUGGING WITH THE ICE-51.....7
BREAKING EMULATION DURING ASYNCHRONOUS OPERATIONS.....9
ICE-51 CMOS DEBUG LIMITATIONS.....11
DIFFERENCES BETWEEN THE 8052 AND 8051.....13
EMULATING THE 8052 WITH ICE-51.....13
CONVERTING AN SDK-51 TO AN SDK-52.....16
INTERPRETING TRACE INFORMATION ON THE ICE-51.....18
VALUE INFORMATION IN EXTERNAL AND INTERNAL ACCESSES.....21
EXAMPLES OF EXTERNAL AND INTERNAL MEMORY ACCESSES.....22
APPLICATIONS OF COMPLEX TRACE AND BREAKPOINTS.....25
TRACING 1000 INSTRUCTIONS.....25
BREAKING ON AN INSTRUCTION THAT SETS A PARTICULAR PORT VALUE.....27
BREAKING ON A MOVX TO A PARTICULAR LOCATION.....28
SAVING INTERNAL REGISTERS IN ICE-51.....30

CONFIGURATIONS AND COMPATIBILITIES.....31

TROUBLESHOOTING GUIDE.....32

See the Index to Symptoms guide following the Table of Contents for a listing of all symptoms described in the Troubleshooting Guide.

AVAILABLE PUBLICATIONS.....34

INDEX TO SYMPTOMS IN TROUBLE SHOOTING GUIDE

1. Bond Out Continuously Failing.....32

2. Error 90: MEMORY OVERFLOW.....32

3. Match Conditions Set on External Memory Not Breaking Emulation.....33

(this page intentionally left blank)

OVERVIEW

DIFFERENCES BETWEEN ICE-51 VERSION 1.3 AND ICE-51 VERSION 2.0

The ICE-51 has undergone major changes to improve the debugging of PL/M programs. The major enhancements to the ICE-51 are :

1. High level PL/M debugging by module and line numbers
2. An enhanced loader to load code into user supplied byte wide rams
3. New commands for ease of use.
4. Enhanced symbolic entry and isplay to support symbols,lines and module names produced by PL/M programs.

The new features extend the capabilities of ICE-51 to debug large PL/M 51 applications.

The following commands were changed to implement the new features:

-DEFINE
-EVALUATE
-GO/STEP
-HELP
-LOAD
-PRINT
-REGISTERS
-SAVE
-SYMBOLS

The new changes in each command are covered individually here to help distinguish between the old and new ICE-51 software. For more detailed information on each command, refer to the latest ICE-51 manual. (See manual listings under Available Publications in this technical report.)

DEFINE COMMAND

The syntax for the DEFINE command has changed to allow symbols to be defined within specific modules. The new syntax is shown below

```
DEFINE [..module name] .symbol-name = expression
```

The module must already exit since new modules cannot be created.

EVALUATE COMMAND

The EVALUATE command has changed to symbolically display the address in the form ..module.symbol if the system can find a symbol that is equal or less then the expression. The evaluate command will display ..module#line number if LINES are ENABLED.

GO/STEP COMMAND

Execution of the GO and STEP commands, normally prints out the following headings:

```
EMULATION BEGUN (AND EMULATION TERMINATED), PC=XXXXH
```

The user can type DISABLE HEADINGS to suppress the print out of the headings. This provides the user with the ability to write custom headings with the use of a WRITE statement within macros.

HELP COMMAND

The help file has changed to add the new commands and incorporate the new syntax changes.

LOAD COMMAND

The LOAD command now includes an external load option for user supplied BYTE WIDE RAMS. The necessary hardware to accomplish the load is provided by the user. The syntax for the LOAD command is:

```
LOAD :Fn:filename NOCODE  [SELECTING module-name[TO module-name],...
                  NOSYMBOLS
                  NOLINES
                  EXTERNAL
```

The new options allow for selective loading of symbols, lines, or external code. The SELECTING option permits the loading of symbols and lines for specific modules or ranges of modules.

SAVE COMMAND

The save command has been changed to add a NOLINES option. The new syntax for the command is shown below:

```
SAVE :Fn:filename  NOCODE      [partition]
                   NOSYMBOLS
                   NOLINES
```

SYMBOLS COMMAND

The SYMBOLS command has changed to accomodate symbols within modules. When SYMBOLS command is entered, symbol names are displayed under each corresponding module. The SYMBOLS command can specify the display of a selective module.

```
SYMBOLS [..module]
```

NEW COMMANDS

The following commands have been added to help debug PL/M programs.

- -DOMAIN
- -MODULES
- -LINES
- -ENABLE & DISABLE LINES
- -ENABLE & DISABLE HEADINGS
- -REMOVE MODULE
- -CWORD, DWORD, XWORD

DOMAIN COMMAND

The DOMAIN command specifies a default module when referencing line number symbols. The syntax of this command is:

```
DOMAIN [..module-name]
```

MODULES COMMAND

The MODULES command lists all the module names linked together to form the user program.

LINES COMMAND

The LINES command displays the associated address for each line number within each module.

```
LINES [..module]
```

ENABLE/DISABLE LINES AND ENABLE/DISABLE SYMBOLS

Trace display is affected by the four conditions of enabling and disabling of the SYMBOLS and LINES commands. These conditions are:

```
ENABLE SYMBOLS / ENABLE LINES
DISABLE SYMBOLS / ENABLE LINES
ENABLE SYMBOLS / DISABLE LINES
DISABLE SYMBOLS / DISABLE LINES
```

Symbolic labels consist of module name, line number, and symbol name, or any combination of these elements.

When symbols and lines are enabled, the symbol table is searched for the line number address that is identical to the address of the trace instruction. The first line number, module name, and matching symbol name found (if any) is displayed on the line preceding its corresponding instruction. If no line number is found, then no label is displayed. The trace display shows no labels for modules with no line numbers defined (such as ASM modules).

With symbols disabled and lines enabled, the symbol table is searched for the line number address that is identical to the address of the trace instruction. Module names, line numbers, and matching symbol names (if any) are displayed, but instructions are suppressed. If no line numbers are found, then no label is displayed.

When symbols are enabled and lines disabled, the symbol table is scanned to see which module the instruction address belongs to. An instruction address is considerd to belong to a module if the address exactly matches any line number, or it is between two line numbers of the same module. When an exact line number match is found, the first matching symbol within that module is displayed. When the address lies between two line numbers of a module, the module containing the symbol cannot be determined. The first matching symbol from any module is displayed. If line numbers are not defined, the first matching symbol name and its module name are displayed.

Only code is displayed when symbols and lines are disabled.

REMOVE MODULE COMMAND

The REMOVE command allows the removal of one or more symbols from the table. The command can specify individual symbols or delete an entire module of symbols and lines. The syntax for the command is:

```
REMOVE SYMBOLS
       [..module-name].symbol-name[,[ ..module-name].symbol-name]
       MODULE ..module-name[,module-name]
       MACROS
       :macro-name [,:macro-name]
```

CWORD/DWORD/XWORD COMMANDS

The CWORD, DWORD, and XWORD commands are similar to the CBYTE, DBYTE and XBYTE except the commands operate on words instead of bytes.

ARTICLES

DEBUGGING WITH THE ICE-51

Designs utilizing an 8031 and 2732A need to configure the External-Access pin active, (EA/=0). This causes all memory access from address 0 to 4K to be external. Debugging this design with the ICE-51, prior to committing to EPROM, requires the user to load the program into ICE-51 memory. Mapping the lower 4K bytes of memory to ICE-51 requires EA/=1 (inactive). A switch needs to be placed in the prototype to facilitate ICE-51 debugging memory mapped to ICE and actual operation out of EPROM memory.

The bond-out chip on the ICE-51 must be thought of as an actual 8051 chip when under emulation. With memory mapped to ICE-51, memory fetches will be performed using the 16 additional "bond-out" lines connected between ICE memory and the bond-out chip.

If EA/=0 (active), the bond-out will also attempt to fetch as the 8031, through P0 and p2. Therefore, contention occurs and the ICE-51 will not operate correctly. If the code is committed to EPROM, (memory mapped to user), fetches will occur as they would for the 8031 (p0 and p2), not the bond-out lines.

The following steps should be done when debugging an 8031 prototype design:

1. STRAP EA/=1 (inactive)
2. Map lower 4K bytes = ICE
3. Debug 8031 code

4. Program EPROMs

5. Strap EA/=0 (active)

6. Debug 8031 circuitry by:

    a) MAP lower 4K bytes - User and use ICE-51
    b) Replace 8031 in socket

BREAKING EMULATION DURING ASYNCHRONOUS OPERATIONS

The TI bit (SCON.1 Transmit Interrupt flag) is set by the hardware of the 8051 when a byte has been transmitted by the serial port. Breaking emulation when the 8051E is in the process of transmitting the byte will result in the TI bit not being set.

Breaking emulation when asynchronous operations are bing performed by the processor raises interesting discussions on exactly how the emulator should deal with the situation. Ideally, the process of transmitting a byte should be frozen in time; the opeation of the bondout an instant before and after emulation is broken would be indestinguishable from the noninterrpted operation of the chip. The peripheral would receive parts of the byte at different points in time, which is an accepted consequence of leaving real time. The user knows this and re-enters emulation at a point which recaptures the program flow.

An alternate and less desirable solution would be to allow the serial port to continue transmitting the byte while program execution has stopped at the breakpoint, until the transmission is complete.

The present stepping of the bondout is the cause of this situation. With this condition in mind, the impact can be localized to one specific serial scenario.

The temporary solution that is recommended you use is explained followed by a program example of a TI set problem.

If the 8051 is servicing the serial port through the use of interrupts, the above situation will not be a problem when debugging the port. Breakthroughs can be placed in the service routine itself (the TI bit has already been set) and execution is not hindered by the break.

When a breakpoint is set to debug another section of code, but interrupts the serial transmission poses a problem that is easily handled. Since the transmission to the peripheral has been lost, the designer will usually go from 0 to re-establish the serial activity from the beginning. A potentially significant debugging situation occurs if the design calls for a polling testing method for serial activity.

As the example shows, it becomes very obvious that something is awry if your program is testing the TI bit. Knowing about this problem, you can easily overcome the situation by resetting the bondout and starting from 0. The problem becomes very significant and exceedingly hard to deal with when a program casually polls the TI bit.

The servicing of the serial port may be a secondary function of the 80951 in which polling occurs at large intervals of time. If a break had occurred by chance when a bytes was being transmitted, the designer has only an indirect way of knowing that the TI bit should be set. The next step is to determine the elaspsed time a byte would take to be transmitted, and then searching backward through trace to determine if SBUP was set within that time period. At 110 baud and 10 bits per byte transmission, there would be about 100 msec worth of transmission. The ICE-51 trace memory would have been flushed many times during transmission.

In that situation there is no way of knowing if the serial port should be transmitting or not. The bottom line in determining if this is the source of a problem is to investigate the polling method used in the design and see if the serial problem goes away if the chip is reset and emulation is started from 0. Have the testing debugger break on the address of the successful jump after testing the TI bit.

Example:

Program of TI Set Problem

Initialization of timers and serial modes was omitted.

```
        MOV   SBUF,A  ; Transmit the byte
LOOP:   JNB   TI,LOOP ; Poll the TI bit for completed
                      ;   transmission
        CLR   TI      ; Clear TI bit
LOCA:   MOV   A,#41H
        MOV   SBUF,A
LOCB:   LJNP  LOOP        ; Poll TI
```

ICE-51 CMOS DEBUG LIMITATIONS

The minor differences between the NMOS 8051 microcomputer and the CMOS version, the 8051C, do not prohibit the emulation of 80C51 with the current version of ICE-51. The specifications of the 8051 and 80C51 are very similar. Some of the specification differences are:

1. The operating voltage, Vcc, for 8051 ranges from 4.5V to 5.5V, versus 4V to 6V for the 80C51.

2. The 80C51 has specifications for the Input Low Voltage (VIL) and the Input High Voltage (VIH) that are Vcc dependent:

   VIL = 0.8V for Vcc > 4.5V; 0.5V for Vcc < 4.5V

   VIH = 2.5V for Vcc > 5.5V; 2.0V for Vcc < 5.5V

   The Output High Voltage (VOH) of the ICE-51 emulation processor is not specified at the 80C51 voltage levels (0.9Vcc at IOC = 10uA). The emulation processor should have no problems reaching 2/3 Vcc at current drain.

3. The Power-Down Voltage (VDP) for the 80C51 is 2.0V to 6.0V, versus 4.5V to 5.5V for the 8051. The power-down mechanism for the 80C51 is different than that of the 8051. The 80C51 enters power-down upon the setting of the PD bit in the PCON register; VPD is then derived from the Vcc line (pin 40).

4. The power supply current for the 8051 emulation processor is 160mA, versus24mA for that of the 80C51.

5. There is no mechanism to emulate the idle mode of the 80C51 component with the 8051-based emulation processor.

6. The TPLIV (PSEN to Valid Instruction IN) for the 80C51 is specified as 3TCLCL (the oscillator period) minus 100ns, versus 3 TCLCL minus 125ns for the 8051.

These specifications (which are also found in the 8051 and 80C51 data sheets) are provided for your information, as well as to explain the reasons for the following guidelines and limitations for emulation of the 8051 and 80C51:

1. Maintain Vcc at 5V + 10%. This ensures that VIL and VIH remain within the 8051 specifications and that the voltage limitations are not violated.

2. The user's power supply must be able to supply at least 160mA for the emulation processor.

3. The 80C51 CPU idle mode is not supported by the ICE-51 or ASM-51. You can use the following workaround to emulate the idle mode (with only a difference in power useage):

```
                SETB IDLE           ; set fake idle bit (where
                                    ; idle is user defined bit in
                                    ; bit addressable data memory)
          LOOP: JB IDLE, LOOP       ; Stay in loop until reset or
                                    ; interrupt occurs.
                .. ........
     INTERRUPT: .. ........
                .. ........

                CLR IDLE            ; Clear IDLE bit so on return
                RETI                ; from interrupt, the LOOP will
                                    ; be exited.
```

4. The power-down mechanism of the 80C51 is not supported by ICE-51 emulation processor. The power-down bit location has no affect on the chip, and the VPD source is not supported.

The ICE-51 is able to provide CMOS support except for the oprating limitations outlined above. The ICE-51 emulates the 8051 NMOS version, therefore, take a close look at the 8051 and 80C51 datasheets before you initiate your emulation procedures.

DIFFERENCES BETWEEN THE 8052 AND 8051

The 8052 provides three additional features over the 8051:

4K of extra internal code space
128 bytes of indirectly addressable RAM space
1 16 bit timer

The ICE-51 and EMV-51 are based on the 8051 and cannot emulate the 16 bit timer or the extra 128 bytes of indirectly addressable RAM space in real time. The purpose of this article is to provide debug alternatives for 8052 designs.

EMULATING THE 8052 WITH ICE-51

The ICE-51 can debug the extra 4K of code space in the 8052 in real time with the help of a currently available command called ROM. The ROM command sets the internal code boundary for the bond out. For 8051 designs, the boundary is set to 4K. Therefore, any code less then 4K will not present address information on port 0 and port 2. Code greater than 4k accesses external code memory and presents address information on port 0 and port 2. An 8052 design needs the boundary extended to 8K before execution out of external code space. The ROM command can extend the boundary to the required 8K of code space. The syntax for the command is:

```
                    4K
ROM     =
                    8K
```

Typing ROM returns the current setting

The ICE-51 cannot emulate the extra timer or data RAM of the 8052 in real time. The ICE-51 can emulate the extra data RAM if breakpoints are set on any instruction accessing the extra RAM. The following macro emulates the internal data RAM of the 8052 by simulating indirect moves to RAM locations above 07H.

```
*mac
DEFINE :RAM
br1=opcode is 0111x011xy    ; set breakpoint for execution on a MOV a,@r0,
br0=location is %1          ; MOV a,@r1, mov @r0,a, mov @r1,a or on address
g from %0 til br            ; set by the user.
repeat                      ; repeat macro until break caused by condition
while cause =02h            ; other than BR1.
```

```
tra=fra                          ; set trace to appropriate mode for macro
move -4                          ; move to instruction causing break

if cbyte frame addr=e6h and r0 >7f then ; if instruction is MOV a,@r0 and
rbyte .acc=cbyte (r0+1f00h)             ; and r0 > 7f then simulate
endif                                   ; instruction in saved code space.
if cbyte frame addr=e7 and r1 > 7f then ; Simulation for mov a,@r1
rbyte .acc=cbyte (r1+1f00h)
endif
if cbyte frame addr=f6 and r0 > 7f then ; Simulation for mov @r0,a
cbyte (r0+1f00h)=rbyte .acc
endif
if cbyte frame addr=f7 and r1 > 7f then ; Simulation for mov @r1,a
cbyte (r1+1f00h)=rbyte .acc
endif
g from (frame addr)+1 til br  ; continue emulation
endr
tra=ins                          ; return trace to instruction mode
em
```

Invoke this macro by typing in :RAM start address, end address. The first few lines in the macro set the breakpoint conditions to break on the instructions(MOV A,@R0, MOV A,@R1, MOV @R0,A, MOV @R1,A) and to break on the specified ending address. The rest of the macro breaks emulation on these instructions and tests to see if the break was caused by the opcode occurence or the ending address. If the ending address causes the break (cause =01h for br0), then the macro terminates. If one of the opcodes caused the break, then the macro determines which instruction caused the break and executes a simulation of the appropriate command. For locations less than 07FH the macro executes the proper command. The macro then skips the execution of the instruction and continues emulation. The following example illustrates the use of the macro.

```
; The execution of moving location 0BFH in internal data RAM is simulated
*dasm 100 len 10
0100H = MOV    R0,#BFH
0102H = MOV    A,@R0
0103H = NOP
*regs
PC    ACC B   SP  DPTR  R0  R1  PSW
010BH FFH 00H 07H 0000H FFH A2H 00000000Y
*rbyte .acc=00
; Set simulated internal ram location bf=99
*cbyte 1fbf=99
*:ram 100,104
EMULATION BEGUN
EMULATION TERMINATED, PC=0103H
EMULATION BEGUN
```

```
EMULATION TERMINATED, PC=0105H
*regs
PC    ACC B   SP  DPTR  R0  R1  PSW
0105H 99H 00H 07H 0000H BFH A2H 00000000Y
; as can be seen the macro executed the instruction as expected with the
; accumulator being set to the contents of shadow ram location B9.
; The following illustration demonstrates the moving of the accumulator to
; the simulated internal ram
*asm org 102h
0102H
*asm mov @r0,a
0103H
*rbyte .acc=33
*:ram 100,105
EMULATION BEGUN
EMULATION TERMINATED, PC=0103H
EMULATION BEGUN
EMULATION TERMINATED, PC=0106H
*CBYTE 1FBF
CBYTE 1FBFH = 33H
; The appropriate location 0BF has been changed
```

The macro will simulate the extra data ram although not in real time. The major limitation of this process is that 128 bytes of code space from 1f80 to 1fffh must be saved. Another point to remember is that the stack should not operate within the internal RAM. A POP or PUSH at locations 80 to 0FF in RAM does not work. The macro is best suited for debugging software in applications where real time emulation is not critical. Be aware of the following limitations with the macro approach.

The stack cannot extend beyond the 07FH boundary.
The macro only works with code mapped to ICE.
The macro uses up one of two possible match conditions for breakpoints.
Software using moves to internal RAM does not run in real time.
The extra timer is not emulated.

NOTE: The macro only works with code mapped to ICE because of the setting of the break condition BR1. The breakpoint is set to break on internal opcodes. The breaking on external opcode fetches require the setting of breakpoints on P0 since the opcodes are traced on P0 when executing out of external code space. The only problem with this is that low order addresses will also cause a matched break condition and improperly simulate a data RAM move.

The macro requires memory to be saved in the internal ICE-51 memory. This requires mapping at least 4K to save space for the simulated internal RAM. The RAM does not necessarily have to reside at 01F80H to 01FFFH. The offset address can be changed to an location. The only requirement is the offset + 128 bytes must reside in internally mapped ICE-51 memory. Any external code using internal data memory fetches cannot be emulated without changing the breakpoint match condition and mapping to accommodate the simulated shadow RAM.

This is a non-real-time solution for simulating software code accessing the extra data RAM of the 8052. By using this solution, a majority of an 8052 target system software and hardware can be debugged.

CONVERTING AN SDK-51 TO AN SDK-52

The differences between the 8051 and the 8052 allows the easy conversion of an SDK-51 to an SDK-52. By substituting the 8052 for the installed 8051, The current SDK-51 can access the extra timer registers and bit locations of the 8052 with the use of RBYTE and RBIT commands. However, the current SDK-51 cannot access the extra data RAM locations above 07FH available on the 8052. The SDK-51 flags any attempt to access locations from 80H to 0FFH with the error message 'ERR 12 ADDR OUT OF RANGE'.

The firmware requires a small change to the monitor routine called (I)FETCH/(I)STORE. The previous monitor code uses direct addressing of the internal DATA RAM and special function registers. The changes required eliminate the generated error message and use indirect addressing through R0 or R1 to access the extra data RAM. The code patch shown uses indirect addressing through R0 to read or write to the extra RAM.

The current monitor code for accessing register and data RAM is:

```
        CJNE   A,#(RBYTE TOKE AND 07H),DBYTE  ; JUMP IF NOT RBYTE
        MOV    A, PNTLOW
        JNB    ACC.7,ERR          ; ERROR IF ADDRESS BETWEEN 0 AND 07FH
        MOV    DPL,A
        JB     F0, XWRITE         ; JUMP TO STORE IF FLAG IS SET
        JMP    XREAD              ; EXIT FROM FETCH
DBYTE:  CJNE   A,#(DBYTE_TOKE AND 07H), RBIT ; JUMP IF RBIT SELECTED
        MOV    A, PNTLOW
        JB     ACC.7, ERR         ; ERROR IF ADDDR IS BETWEEN 80 AND 0FFH
        MOV    DPL,A              ; LOAD DPL WITH NEW POINT VALUE
        JB     F0,XWRITE          ; JUMP TO STORE IF FLAG IS SET
        JMP    XREAD              ; EXIT FROM FETCH
```

The modified SDK-52 code is as follows

```
          CJNE   A,#(RBYTE TOKE AND 07H, DBYTE ; JUMP IF NOT RBYTE
DIRECT:   MOV    DPL,PNTLOW        ; LOAD DPL WITH NEW POINT VALUE
          JB     F0, XWRITE        ; JUMP TO STORE IF FLAG IS SET
          JMP    XREAD             ; EXIT FORM FETCH
DBYTE:    CJNE   A,#(DBYTE_TOKE AND 07H), RBIT ; JUMP IF RBIT SELECTED
          MOV    A, PNTLOW         ; JUMP TO DIRECT ACCESS FOR
          JNB    ACC.7, DIRECT     ; DATA RAM < 07FH
          MOV    R0,A              ; STORE POINTER FOR INDIRECT ACCESS
          JB     F0,DWRITE         ; JUMP IF DATA WRITE
          MOV    A,@R0             ; MOVE DATA RAM VALUE INTO ACCUMULATOR
          JMP    FETEND            ; JUMP IF END OF FETCH
DWRITE:   MOV    A,PARAM1          ; DATA VALUE TO BE STORED
          MOV    @R0,A             ; STORE VALUE USING INDIRECT ADDRESSING
          JMP    FETEND            ; JUMP TO END OF FETCH
```

These code changes can be substituted in place of the old monitor routine.

Change the monitor PROM on the left of the two PROM monitor set by carefully removing and copying the contents to an internal buffer on a PROM programmer. To implement the code changes, substitute the following code at locations E6AC to E6C9.

B4,01,08,85,45,82,20,D5,E6,C1,97.B4,02,15,E5,45,30,E7,F0,F8,20,D5,03,E6, C1,9D,EA,F6,C1,9D

There is another consideration when making the above changes. The SDK-52 does a checksum on the two monitor PROMS and generates an error message if the generated checksum does not match a stored value. To change the checksum make the following code substitution at E049H.

8D

After the substitutions have been made, burn the new PROM and install it in the old PROM's location.

The replacement of the 8051 with the 8052 converts the old SDK-51 to an SDK-52. The SDK-52 provides the best way to debug the extra timer and extra data RAM of the 8052. The only limitations are those imposed by the SDK-52. The user should consider the use of a combination of ICE-51 and SDK-52 when debugging 8052 designs.

INTERPRETING TRACE INFORMATION ON THE ICE-51

The ICE-51 provides detailed frames information for the setting of breakpoints and conditional trace. The interpretation of this information depends on the instructions executed by the ICE51. The ICE-51 monitors two signals, ALE and PSEN/ to latch 8051 address and data information. Figure 1 illustrates a one cycle memory fetch.

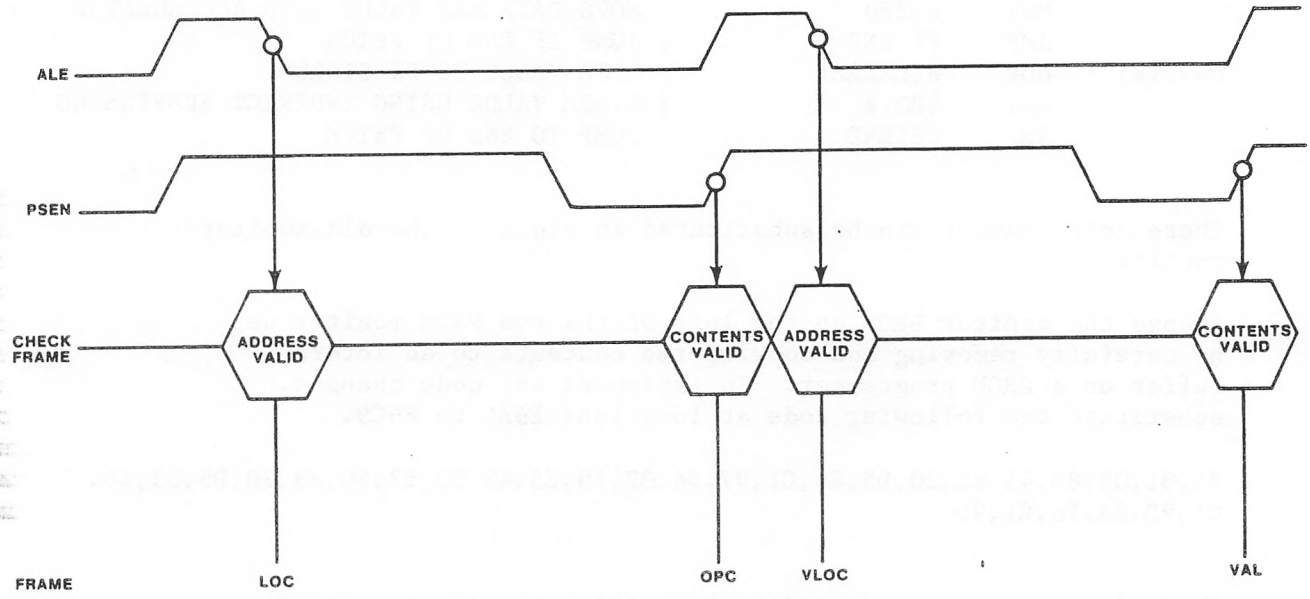


Figure 1. ONE CYCLE MEMORY FETCH

On each falling edge of ALE a valid address is latched and on each rising edge of PSEN/ a valid opcode, operand or next instruction opcode is traced. The 8051 fetches opcodes or operands twice a cycle except during a MOVX instruction. Whether an opcode, operand or next instruction opcode is fetched depends on the length in bytes of the instruction and the cycles it takes to execute. The simplest instruction is a one byte, one cycle instruction like INC A. The display of trace in frames mode shows the tracing of four different types of frames: LOCATION, VLOCATION, OPCODE and VALUE. The trace of the instruction at

1000H is shown in Figure 2.

```
FRAME TYPE ADDR  DATA  INSTRUCTION          P1   P2  P0   TOVF
0000: LOC  1000H (04H) (INC    A)           FFH  10H 00H   0
0001: OPC        04H                        FEH  10H 00H   0
0002: VLO  1001H                            FEH  10H 01H   0
0003: VAL        74H                        FEH  10H 00H   0
0004: LOC  1001H (74H) (MOV    A,#---)      FFH  10H 01H   0
0005: OPC        74H                        FEH  10H 00H   0
0006: VLO  1002H                            FEH  10H 02H   0
0007: VAL        44H                        FEH  10H 00H   0
```

Figure 2. FRAMES INFORMATION FOR A ONE BYTE, ONE CYCLE INSTRUCTION

FRAME 0 is a LOCATION frame collected on the falling edge of ALE. This frame displays the opcode address of the first cycle of an instruction. FRAME 1 is an OPCODE frame collected on the rising edge of PSEN/ and corresponds to the current instruction opcode. LOCATION and OPCODE frames always correspond to the first cyle opcode fetch of an instruction. FRAME 2, collected on the falling edge of ALE, captures the address of the next instruction. Since INC A is a one byte instruction, there are no further operands to fetch. Therefore, the frame shows the fetch of the next instructions opcode. The 8051 disregards the opcode if the current instruction has not completed execution. The fetch of 74H is again executed during frames 05 and 06.

FRAMES 5 and 6 display the first cycle opcode fetch of the instruction following the INC A instruction.(Notice the opcode has been fetched twice) Frames 6 and 7 correspond to the address and operand of the instruction MOV A, #44H. VLOCATION frames are the address of an opcode or operand. VALUE frames can correspond to operands or to the next instruction opcodes depending on the length and the cycles required for completion of the instruction being executed. The next example is an instruction requiring two cycles to execute.

INC DPTR is a one byte, two cycle instruction. In Figure 3, the trace in FRAMES mode shows the opcode fetch of the next instruction for three consecutive times. This occurs because the instruction requires only one cycle to fetch the single opcode. This is an illustration of the continuous fetching of the next opcode while the 8051 completes the execution of the current instruction.

FRAME	TYPE	ADDR	DATA	INSTRUCTION	P1	P2	P0	TOVF
0000:	LOC	1000H	(A3H)	(INC DPTR)	FFH	10H	00H	0
0001:	OPC		A3H		FEH	10H	00H	0
0002:	VLO	1001H			FEH	10H	01H	0
0003:	VAL		74H		FEH	10H	00H	0
0004:	VLO	1001H			FEH	10H	01H	0
0005:	VAL		74H		FEH	10H	00H	0
0006:	VLO	1001H			FEH	10H	01H	0
0007:	VAL		74H		FEH	10H	01H	0
0008:	LOC	1001H	(74H)	(MOV A,#---)	FFH	10H	01H	0
0009:	OPC		74H		FEH	10H	7FH	0
0010:	VLO	1002H			FEH	10H	02H	0
0011:	VAL		44H		FEH	10H	FFH	0

Figure 3. EXAMPLE OF A 1 BYTE, 2CYCLE INSTRUCTION e.g. INC DPTR

In the case of a branch instruction, the last operand is fetched twice instead of the next opcode. Figure 4 displays the frame information for a branch instruction. Frames 04 and 07 show the repetitive fetching of the final operand of the next instruction.

FRAME	TYPE	ADDR	DATA	INSTRUCTION	P1	P2	P0	TOVF
0000:	LOC	1000H	(02H)	(LJMP ---)	FFH	10H	00H	0
0001:	OPC		02H		FEH	10H	00H	0
0002:	VLO	1001H			FEH	10H	01H	0
0003:	VAL		20H		FEH	10H	00H	0
0004:	VLO	1002H			FEH	10H	02H	0
0005:	VAL		00H		FEH	10H	00H	0
0006:	VLO	1002H			FEH	10H	02H	0
0007:	VAL		00H		FEH	10H	00H	0
0008:	LOC	2000H	(A3H)	(INC DPTR)	FFH	20H	00H	0
0009:	OPC		A3H		FEH	20H	05H	0
0010:	VLO	2001H			FEH	20H	01H	0
0011:	VAL		91H		FEH	20H	05H	0
0012:	VLO	2001H			FEH	20H	01H	0
0013:	VAL		91H		FEH	20H	05H	0
0014:	VLO	2001H			FEH	20H	01H	0
0015:	VAL		91H		FEH	20H	35H	0

Figure 4. EXAMPLE OF A BRANCH INSTRUCTION

The MOVX instruction is a little different than most instructions. The trace in FRAMES mode shows the fetching of the next opcode at frames 2 and 3. However, in frames 4 through 6 the execution of the MOVX instruction is on ports 0 and 2. The trace displays the absence of an ALE during frames 5 and 6. This is expected since the 8051 skips an ALE during the second cycle of a MOVX instruction. Frame 4 is the ALE which latches the valid external data address and the following two frames are latched frames because PSEN/ is active while ALE is not active. The information latched in the VAL frames is the data on the internal ICE memory bus when PSEN/ goes low. The PSEN in the target system maintains a constant high on PSEN/ although the ICE has a transition.

```
0000: LOC  1000H (E0H) (MOVX   A,@DPTR)      FFH  10H 00H   0
0001: OPC         E0H                        FFH  10H FFH   0
0002: VLO  1001H                             FFH  10H 01H   0
0003: VAL         74H                        FFH  10H FFH   0
0004: VLO  0088H                             FFH  00H 88H   0
0005: VAL         11H                        FFH  00H FFH   0
0006: VAL         11H                        FFH  00H FFH   0
0007: LOC  1001H (74H)  (MOV A,#--)          FFH  10H 01H   0
0008: OPC         74H                        FFH  10H FFH   0
0009: VLO  1002H                             FFH  10H 02H   0
0010: VAL         44H                        FFH  10H FFH   0
```

Figure 5. EXAMPLE OF A MOVX INSTRUCTION

VALUE INFORMATION IN EXTERNAL AND INTERNAL ACCESSES

The VAL frames on ICE-51 displays different results depending on whether an internal ICE or external user memory fetch is being executed. The VALUE column in ICE-51 displays the proper operands and opcodes when accessing internal ICE memory. However, when accessing external code memory, 0FF is displayed during opcode and value frames. The correct opcode or operand is displayed on P0. P0 displays invalid operands and opcodes when executing out of code greater than the 4K mapped internally to ICE-51 memory. Figures 6-7 show the tracing of instructions using different map settings. The value and P0 data are different in each case. VALUE and OPCODE frames normally correspond to internal ICE memory activity. P0 and P2 correspond to external access of user memory or to port values when accessing the lower 4K of internal 8051 memory. This is an important consideration when setting complex break and trace points. To break or trace correctly, take into account the

information being traced.

EXAMPLES OF EXTERNAL AND INTERNAL MEMORY ACCESSES

Figure 6 is an example of an internal ICE memory access of less than 4K. The data displayed during OPC and VAL frames represents instruction opcode and operands. In this case, the code is executing out of internal ICE memory and the trace of the activity is displayed in the DATA column.

FRAME	TYPE	ADDR	DATA	INSTRUCTION	P1	P2	P0	TOVF
0000:	LOC	0100H	(04H)	(INC A)	FFH	FFH	FFH	0
0001:	OPC		04H		FFH	FFH	FFH	0
0002:	VLO	0101H			FFH	FFH	FFH	0
0003:	VAL		74H		FFH	FFH	FFH	0
0004:	LOC	0101H	(74H)	(MOV A,#---)	FFH	FFH	FFH	0
0005:	OPC		74H		FFH	FFH	FFH	0
0006:	VLO	0102H			FFH	FFH	FFH	0
0007:	VAL		44H		FFH	FFH	FFH	0

Figure 6. EXAMPLE OF INTERNAL ICE MEMORY ACCESS LESS THAN 4K.

Figure 7 is an example of external access of greater than 4k with memory mapped to ICE. The DATA displayed during OPC and VAL frames represents the instruction opcode and operands. The activity on P0 is the ICE presenting address information on the port since the 8051 is accessing external code memory. The opcode and operands are not used because the access is to internal ICE memory and traced in the DATA column.

FRAME	TYPE	ADDR	DATA	INSTRUCTION	P1	P2	P0	TOVF
0000:	LOC	1000H	(04H)	(INC A)	FFH	10H	00H	0
0001:	OPC		04H		FFH	10H	FFH	0
0002:	VLO	1001H			FFH	10H	01H	0
0003:	VAL		74H		FFH	10H	FFH	0
0004:	LOC	1001H	(74H)	(MOV A,#---)	FFH	10H	01H	0
0005:	OPC		74H		FFH	10H	FFH	0
0006:	VLO	1002H			FFH	10H	02H	0
0007:	VAL		44H		FFH	10H	FFH	0

Figure 7. EXAMPLE OF EXTERNAL ACCESS GREATER THAN 4K MAPPED TO ICE

Figure 8 is an example of an external access of memory mapped to USER. The valid opcode and operands are now traced on P0. OPCODE and VALUE frames display values of 0FFh.

FRAME	TYPE	ADDR	DATA	INSTRUCTION	P1	P2	P0	TOVF
0000:	LOC	E000H	(02H)	(LJMP ---)	FFH	E0H	00H	0
0001:	OPC		FFH	(MOV R7,A)	FFH	E0H	02H	0
0002:	VLO	E001H			FFH	E0H	01H	0
0003:	VAL		FFH		FFH	E0H	E2H	0
0004:	VLO	E002H			FFH	E0H	02H	0
0005:	VAL		FFH		FFH	E0H	65H	0
0006:	VLO	E002H			FFH	E0H	02H	0
0007:	VAL		FFH		FFH	E0H	65H	0

Figure 8 EXAMPLE OF EXTERNAL ACCESS WITH ICE MAPPED TO USER

The implication of the varying trace information is that breakpoints set for certain conditions may not be executed. For example, the setting of a breakpoint on the opcode 0A3Hh(INC A) does not execute as expected if the opcode resides in externally mapped memory. To set a breakpoint on the opcode, specifing a break on P0 IS 0A3H is needed. In addition, the setting of breakpoints or tracepoints on particular frame values may cause other frames to meet the match condition. To break on the execution of any opcode or operand independent of the memory location, specify a match condition on VAL IS 0A3H OR P0 IS 0A3H. However, the problem is that the port value is not restricted to opcodes or operands and matches when executing addresses with a lower address of 0A3H. Take into account the frames information

that the ICE is attempting to break or trace on especially when using match conditions on OPC and VAL frames. Take a close look at the trace of instructions in FRAMES mode prior to setting any complex trace or break points. A brief description of the match conditions available are shown in Table 1.

MATCH CONDITION	CORRESPONDING FRAME INFORMATION
LOCATION	addr of first opcode fetch
VLOCATION	addr of subsequent opcode and operand fetches
ADDR	addr of opcodes and operands for VLOC and LOC frames
OPCODE	instruction opcodes during internal access(OPCODE =0FFH during ext access)
VALUE	operands and opcodes during internal access (VALUE=FF during ext access)
P0	P0 column (See note 1)
P1	P1 column (displays actual Port value)
P2	P2 column (See note 2)
XADDR	P2 and P0 during external access.

Table 1. MATCH CONDITIONS AND FRAME INFORMATION

Note 1: During internally mapped ICE accesses of less than 4K, P0 displays the the port value. During external accesses, P0 displays the opcode, operand or lower address. During external accesses of ICE mapped memory greater than 4K, P0 displays the current state of the port on OPC and VAL frames and displays the lower address on the LOC and VLOC frames.

Note 2: During external accesses, P2 displays the upper address. P2 displays the port value during internal ICE memory accesses.

APPLICATIONS OF COMPLEX TRACE AND BREAKPOINTS

TRACING 1000 INSTRUCTIONS

Information gathering can be optimized by using a combination of conditional trace and macros. The following macro optimizes the number of instructions traced using the ICE-51:

```
DEFINE :TRACE
write 'ADDR   INSTRUCTION'
oldest
move %0
COUNT (%1-%0)
If (frame addr-1) = 03h  Then
write '***external 0 interrupt***'
ORIF (frame addr-1) = 0bh
write '***timer 0 interrupt***'
ORIF (frame addr-1) = 13h
write '***external 1 interrupt***'
ORIF (frame addr-1) = 1bh
write '***timer 1 interrupt***'
ORIF (frame addr-1) = 23h
write '***serial interrupt***'
ENDIF
DASM (FRAME ADDR-1)
MOVE 1
ENDC
EM
```

The commands to set up the macro are:

```
TR=OPCODE IS XXH     ;  Trace all VLOC frames following an OPC frame
TRA=FRAMES           ;  Set up trace to display only frame information
G                    ;  Start emulation
:TRACE 100,200       ;  Trace frames 100 to 200
```

The setting up of a match condition on all opcodes traces all VLOC frames following an opcode. The traced VLOC frames give the address of the first cycle of the instruction +1. The macro extracts the address of the instruction executed by subtracting one from the VLOC frame being traced. The disassembly of the address provides a trace of the instruction executed. The macro prints out a message for interrupt addresses to warn that the previous instruction traced may not have been executed. During an interrupt, the processor fetches the first cycle opcode more than once. The trace of an occuring interrupt shows the apparent execution of an instruction but the instruction is actually

fetched and not executed. The VLOC frame traced following the OPCODE is the actual address of the first cycle of the instruction instead of the ADDRESS+1. The trace macro subtracts one and disassembles either the opcode or the last operand of the previous instruction. Therefore, an unexpected instruction may be displayed. The message should warn that the previous instruction may not have been executed. The instruction should be executed following the RETI instruction at the end of the appropriate interrupt service routine. An example using the trace macro is shown here. An interrupt is also shown:

```
*tr=opc is 0xxh
*tra=frames
*g from .start
EMULATION BEGUN
WARN CA:PPC/OPCODE NOT VALID
PROCESSING ABORTED
*:trace 100,120
ADDR   INSTRUCTION
0146H = INC     R0
0147H = CJNE    R0,#5BH,.MIDOUT
.MIDOUT
014CH = POP     .PSW
014EH = SETB    .TR1
.ENDOUT
0150H = RETI
0132H = MOV     .P2,A
0134H = JNC     .LOOP
.LOOP           .
0130H = ADD     A,#01H
0132H = MOV     .P2,A
0133H = ORL     C,/50H
***timer 1 interrupt***
001BH = AJMP    .TIMOUT
.TIMOUT
0140H = CLR     .TR1
0142H = PUSH    .PSW
0144H = MOV     .P1,R0
0146H = INC     R0
0147H = CJNE    R0,#5BH,.MIDOUT
.MIDOUT
014CH = POP     .PSW
014EH = SETB    .TR1
.ENDOUT
0150H = RETI
0134H = JNC     .LOOP
```

NOTE: The ORL C,/50H is the disassembly of A0H. This is last operand of the instruction MOV .P2,A. A0H is ORL C,/bit. The VLOC address traced in frames mode is 134H which corresponds to the address of the JNC .LOOP instruction. This instruction is fetched while the interrupt is being processed. Subtracting one from the address results in 133H. This is the address of the operand A0H.

BREAKING ON AN INSTRUCTION THAT SETS A PARTICULAR PORT VALUE

Breaking on the setting of a port value by a particular instruction can be done using a complex breakpoint. Consider the case of setting a breakpoint on a MOV .P0,A instruction when the value latched is 59H. The information in trace shows the match condition required.

```
ADDR   INSTRUCTION
0100H = MOV    .P0,A
0102H = INC    A
0103H = AJMP   0100H

*print 15
FRAME TYPE ADDR  DATA  INSTRUCTION          P1   P2  P0   TOVF
0188: LOC  0103H (21H) (AJMP   ---)         FFH  FFH 52H   1
0189: OPC        21H                        FFH  FFH 52H   1
0190: VLO  0104H                            FFH  FFH 52H   1
0191: VAL        00H                        FFH  FFH 52H   1
0192: VLO  0105H                            FFH  FFH 52H   1
0193: VAL        19H                        FFH  FFH 52H   1
0194: VLO  0105H                            FFH  FFH 52H   1
0195: VAL        19H                        FFH  FFH 52H   1
0196: LOC  0100H (F5H) (MOV    ---,A)       FFH  FFH 52H   1
0197: OPC        F5H                        FFH  FFH 52H   1
0198: VLO  0101H                            FFH  FFH 52H   1
0199: VAL        80H                        FFH  FFH 52H   1
0200: LOC  0102H (04H) (INC    A)           FFH  FFH 52H   1
0201: OPC        04H                        FFH  FFH 53H   1
0202: VLO  0103H                            FFH  FFH 53H   1
```

The trace of the instruction shows that the port changes on the opcode frame of the next instruction. To set a breakpoint, the match condition should be set at OPC IS 04H and P0 IS 53H.

```
*res chi
*g from 100 til opc is 04h and p0 is 53h
EMULATION BEGUN
EMULATION TERMINATED, PC=0103H
```

```
*print -10
FRAME TYPE ADDR  DATA  INSTRUCTION          P1   P2  P0   TOVF
0990: VLO  0105H                           FFH  FFH 52H   1
0991: VAL        19H                       FFH  FFH 52H   1
0992: LOC  0100H (F5H) (MOV    ---,A)      FFH  FFH 52H   1
0993: OPC        F5H                       FFH  FFH 52H   1
0994: VLO  0101H                           FFH  FFH 52H   1
0995: VAL        80H                       FFH  FFH 52H   1
0996: LOC  0102H (04H) (INC    A)          FFH  FFH 52H   1
0997: OPC        04H                       FFH  FFH 53H   1
0998: VLO  0103H                           FFH  FFH 53H   1
0999: VAL        21H                       FFH  FFH 53H   1
```

The trace shows that the breakpoint was executed as expected. This is just one example of setting a complex breakpoint using information obtained from trace in FRAMES mode.

BREAKING ON A MOVX TO A PARTICULAR LOCATION

Breaking on a MOVX instruction to a particular address can be done using the SY0 and SY1 lines. To differentiate between a code address and a data address some external circuitry is required. This circuitry is shown in Figure 9.

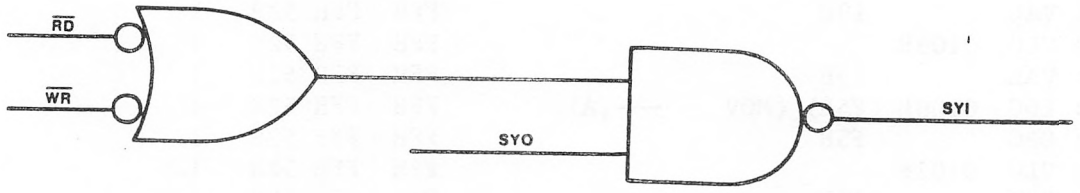


Figure 9. CIRCUITRY TO LATCH DATA MEMORY ADDRESS

The SY1 line indicates that a specific external address is on ports P0 and P1. The RD and WR are ORed together to latch the external matched by SY1. Therefore, only matches of data memory addresses occur. The latched SY1 signal is then used to halt emulation through the latched SY0 line. To set up the ICE for a break on MOVX to a specific address, the following commands and conditions need to be initiated:

```
*ENABLE SY1 OUT
*ENABLE SY0 LATCH
*TR=XADDR IS 1000H
*G F 0 TILL SY0
```

The ICE is now set up to break on a MOVX at external data memory location 1000H. The SY1 only traces when location 1000H is on ports P0 and P2. SY0 is driven low when SY1 goes high with a corresponding RD or WR signal. Therefore, emulation only breaks on a MOVX instruction. To break on a memory read, the WR line needs to be tied high. To break on a memory write, the RD line needs to be tied high.

SAVING INTERNAL REGISTERS IN ICE-51

The ICE-51 does not have any direct means for saving the internal data RAM or the special function registers on disk when exiting an emulation session. To save the state of the 8051 when exiting ICE-51, save the registers using block memory moves to internal ICE-51 code space and then save the code on diskette. The steps for saving internal data RAM and the special function registers are as follows:

```
*CBYTE 0 to 07FH =DBYTE 0 to 7FH    ; SAVES the internal data RAM
*CBYTE 80 to 0FFH=RBYTE 80 to 0FFH ; saves the special function registers
*SAVE REGSAV 0 to 0FFH              ; save data in file REGSAV
*EXIT
```

When entering ICE-51 enter:

```
*LOAD REGSAV
*DBYTE 0 to 07FH=CBYTE 0 TO 7FH
*RBYTE 80 to 0FFH=CBYTE 80 TO 0FFH
*; continue emulation
```

NOTE: Code residing at locations 0 to 0FFH is copied over. Therefore any saving of code to disk should be done before the registers are saved. Likewise, the user program should be loaded after the registers are loaded.

CONFIGURATIONS AND COMPATIBILITIES

The ICE-51 has two different software versions operating on the same hardware configurations. Version 2.0 is an upgrade to version 1.3 with additional features for debugging PL/M 51 programs. Either version can be used to run the ICE-51. However, version 2.0 includes several syntax changes to commands. The latest hardware and software configurations for ICE-51 are listed below:

Software Version Number	2.0
Firmware Version Number	.1

PWA numbers for the ICE-51

Control Processor	162380-003
Emulator	162249-001

The latest stepping of the emulator processor(bond out) is 8051E(G)

TROUBLESHOOTING GUIDE

The following Problem List has been compiled to assist you in identifying solutions to known product anomalies. To use this list, locate the symptom displayed by your software. Then read the cause of your problem and implement the solution.

An index to this list appears on the page following the Contents page.

If you detect a problem with your software, please submit a Software Problem Report (SPR) to Intel Corporation. We ask that you do this even if you believe you have encountered a known problem. We then can verify your diagnosis and obtain a measure of that problem's impact on our customers.

---

1. SYMPTOM 8051 bond out failure.

   CAUSE The majority of the ICE-51 bond out failures can be attributed to abuse in handling or to ground differentials between the ICE-51 and the target system.

   SOLUTION Take precautions to prevent static discharge by trying not to mishandle the bond out. Care should be taken whenever coming in contact with the part. In addition, the bond out is susceptible to destruction when ground differentials exist between the ICE-51 and the target system. The user should run a ground strap from the grounding pins on the end of the user cable to the target system. Any test equipment should also be grounded to help prevent presenting a large voltage potential on the bond out pins.

2. SYMPTOM This message appears: ERR 90: MEMORY OVERFLOW

   CAUSE Error 90 occurs when the workspace of ICE-51 has overflowed. This is usually caused by too many user defined symbols.

   SOLUTION When writing large programs, separate tasks into individual modules. Writing code in compact modules eases the task of debugging and compiling large programs. Specific tasks can be debugged at the module level and any changes required can be compiled quickly. The selective loading of line numbers and symbols by module or range of modules is designed to optimize the debugging of modular programs with ICE-51. Selective loading of symbols by module conserves the symbol table space available for the ICE-51 and should be used when the number of defined symbols exceeds the available workspace.

3. SYMPTOM Complex match conditions are not being recognized when executing out of external code memory. For example, setting a breakpoint for opcode is 00H (NOP) does not work when the opcode resides in external user code space.

   CAUSE The ICE-51 monitors internal and external program memory fetches for gathering trace information and comparing match conditions. When accessing internally mapped ICE-51 memory, the LOC, VLOC, OPC and VAL match conditions work properly. However, the operands, opcodes and addresses are traced on ports P0 and P2 when accessing external code memory. Match conditions like OPC IS do not work when executing out of external code space.

   SOLUTION The article on trace information in this technical report details the setting of complex breakpoints in external or internal code memory. To set a match condition for an opcode executing out of external code space set a match condition for P0 IS <opcode>.

AVAILABLE PUBLICATIONS

The following are the available manuals for the ICE-51 In Circuit Emulator.

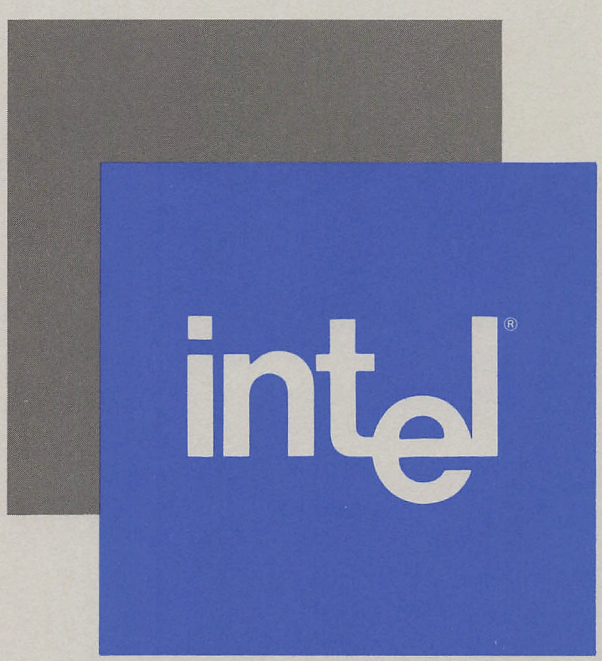
Manual Listings

- ICE-51/ICE-44 IN CIRCUIT EMULATOR OPERATING INSTRUCTIONS (SUPPORTING PL/M 51) FOR ISIS USERS
  Order No.: 164221

  This manual describes the operating instructions for the ICE-51 In-Circuit Emulator.

- ICE-51/ICE-44 IN CIRCUIT EMULATOR (SUPPORTING PL/M 51) COMMAND DICTIONARY
  Order No.: 164222

  This manual lists all the user commands for the ICE-51/ICE-44 in circuit emulators in alphabetical order. The dictionary provides a quick reference of the different commands used by the in circuit emulators.



SUPPORT SERVICES