GETTING STARTED WITH THE ICE-51™ IN-CIRCUIT EMULATOR

Manual Order Number: 121595-001 Rev. A

intel®

GETTING STARTED WITH THE ICE-51™ IN-CIRCUIT EMULATOR

Manual Order Number: 121595-001 Rev. A

Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP	Int$_e$l	Megachassis
CREDIT	Intelevision	Micromap
i	Intellec	Multibus
ICE	iRMX	Multimodule
iCS	iSBC	PROMPT
im	iSBX	Promware
Insite	Library Manager	RMX/80
Intel	MCS	System 2000
		UPI
		μScope

and the combination of ICE, iCS, iRMX, iSBC, iSBX, MCS, or RMX and a numerical suffix.

A299/181/10K DD

PREFACE

This manual provides a hands-on introduction to the main features of the ICE-51™ in-circuit emulator for the 8051 microcontroller. It is intended for the user with little or no previous experience with Intel's in-circuit emulators. This manual shows how the emulator commands operate, using a short sample program that you enter with the emulator's one-line assembler. Although you are not required to use the ASM51 assembler for this manual, it is assumed that you are familiar with the 8051 and its assembler as described in the following manuals:

- *Intel MCS-51™ Family User's Manual,* Manual Order No. 121517.
- *MCS-51™ Macro Assembler User's Guide,* Manual Order No. 9800937.

This manual assumes that you have installed the emulator hardware in the development system. The manual shows how to invoke the emulator system, and how to enter commands. Operation of the commands is illustrated through examples with a minimum of discussion. For installation procedures, details on commands, and other information on the emulator system, please refer to the following manuals:

- *ICE-51™ Operating Instructions,* Manual Order No. 9801004.
- *ICE-51™ Command Dictionary,* Manual Order No. 9801005.

For information on the ISIS-II operating system, refer to the following manual:

- *ISIS-II User's Guide,* Manual Order No. 9800306.

CONTENTS

INTRODUCTION

This manual contains practical examples of many typical ICE-51 emulator commands. These examples are designed to get you started with a minimum of explanation. To use this manual as a hands-on learning guide, install the emulator hardware in the development system, and attach the crystal power accessory for stand-alone operation. Power up the development system and the disk drives, and insert an ISIS-II system diskette in drive 0. Insert the emulator software diskette in drive 1.

In this manual, lines that begin with asterisks (*) are the user commands; the asterisk is the emulator system prompt. Lines with no prompts are the displays produced by the emulator system. To make these lines stand out, they are set in a typeface different from the standard text; for example:

```
*MAP                          ;This is the command you enter.
MAP=0000H,1000H               ;This is the display that results.
```

Some lines contain comments to help explain the commands and displays. Each comment begins with a semicolon (;), as in the examples shown above. Enter the commands as shown, omitting the comments. Press the RETURN (CR) key at the end of each command line.

The following controls are useful for correcting commands as they are being entered (that is, before the CR):

RUBOUT	Delete last character typed; repeat RUBOUT to delete more than one character.
CTRL X	Delete current line of command being entered.
ESC	Delete entire command being entered.
CTRL R	Echo command line being entered.
CR	Carriage return ends command line.
LF	Line feed also ends command line.

Once a command line has been ended with CR or LF that line can no longer be corrected. The following additional controls are used to pause and continue during lengthy displays:

CTRL S	Pause console display.
CTRL Q	Continue console display.

This manual is organized into four "sessions". Session one presents a basic orientation, display of memory and registers, and use of the one-line assembler. The program entered in session one is saved on diskette file for use in the later sessions. Session two illustrates loading code from file, and demonstrates the emulation and trace controls. Session three shows how to define macros to automate all or part of the debugging process. The macros defined in session three are saved on diskette for use in session four. Session four shows how to exercise the system using command sequences brought in from diskette file.

NOTE

Several displays in this manual contain "random" values that will differ each time the examples are executed. These displays are screened to emphasize that your result can be different from that shown.

DEMONSTRATION PROGRAM LISTING

The examples in this manual involve the sample program described in this section. The listing shows the label (if present), the address, and the instruction mnemonic for each instruction.

The program is organized in three blocks: START, LOOP, and TIMOUT. START initializes the program, a demonstration of on-chip timer (Timer 1) in auto-reload mode. When initialization is complete, START starts the timer and jumps to LOOP. LOOP marks instruction time by incrementing the accumulator. To allow us to trace this activity, the accumulator is written out to a port on each iteration of the loop. The Data Pointer (DPTR) is used to count overflows from the accumulator. TIMOUT is the timer interrupt service routine. This routine outputs the characters A through Z repetitively to a port. The character hold time is determined by the auto-reload value of the timer. The program comments give further details.

Note that the program listing uses the emulator system's version of symbolic reference (.START for the label START, for example).

In session one you will be shown how to enter this program, labels and all.

```
LABEL       LOC     INSTRUCTION             COMMENTS

.RESET      0000H   AJMP    .START          ;Reset vector to initializa-
                                            ;tion routine.

.TIMER1     001BH   AJMP    .TIMOUT         ;Timer 1 interrupt vector
                                            ;to service routine.

.START      0100H   CLR     A               ;Clears accumulator, carry
            0101H   CLR     C               ;flag, data pointer, and
            0102H   MOV     DPTR, #0000H    ;port 1 used by the main
            0105H   MOV     .P1, #00H       ;loop.
            0108H   MOV     .P2, #00H       ;Clears ports 2 and sets R0
            010BH   MOV     R0, #41H        ;to letter 'A', both for timer
                                            ;service routine.
            010DH   MOV     .TMOD, #20H     ;Set timer 1 in auto-reload
                                            ;mode.
            0110H   MOV     .TCON, #00H     ;Clear all timer control bits.
            0113H   MOV     .TL1, #F4H      ;Timer 1 initial value.
            0116H   MOV     .TH1, #F4H      ;Timer 1 reload value.
            0119H   MOV     .IP, #08H       ;Set timer 1 interrupt to
                                            ;priority 1.
            011CH   MOV     .IE, #88H       ;Enable timer 1 interrupt.
            011FH   SETB    .TR1            ;Start timer 1.
.ENDSTART   0121H   AJMP    .LOOP           ;End of initialization.

.LOOP       0130H   ADD     A, #01H         ;Loop counter.
            0132H   MOV     .P2, A          ;Output the counter.
            0134H   JNC     .LOOP           ;The carry is set when the
                                            ;accumulator overflows.
            0136H   INC     DPTR            ;"Millisecond" counter.
            0137H   CLR     C               ;Set up for next 256
                                            ;iterations.
.ENDLOOP    0138H   SJMP    .LOOP           ;Restart the loop.
```

(**NOTE:** The first three instructions, ADD MOV, and JNC, require a total of four cycles or four microseconds to execute. Thus, 256 times through this loop represents approximately 1 millisecond.)

```
.TIMOUT     0140H   CLR     .TR1            ;Stop the timer while we
                                            ;service the interrupt.
            0142H   PUSH    .PSW            ;Save the carry flag for the
                                            ;main loop.
            0144H   MOV     .P1, R0         ;Output current character.
            0146H   INC     R0              ;Next character.
            0147H   CJNE    R0, #5BH,.MIDOUT ;Character 'Z' = 5AH should
                                            ;be the last in the series.
            014AH   MOV     R0,#41H         ;Start at 'A' again.
.MIDOUT     014CH   POP     .PSW            ;Restore the carry flag.
            014EH   SETB    .TR1            ;Start the timer again.
.ENDOUT     0150H   RETI                    ;End of interrupt.
```

(NOTE: The CJNE instruction sets the carry flag if R0 is less than 5BH. This is the reason for saving and restoring the carry flag in this routine.)

In session one you learn how to invoke the emulator system, display and change the contents of memory and registers, obtain help at the console, operate the one-line assembler, save a program from code memory to disk file, and exit the emulator system.

To invoke the emulator system, boot the ISIS-II system and obtain the ISIS-II prompt. Enter the command:

```
>:F1:ICE51
ISIS-II ICE-51 Vn.n
FOR COMMAND ENTRY ASSISTANCE, TYPE HELP
*
```

The asterisk (*) is the emulator system prompt; the system is waiting for a command. (Before the prompt appears, the system requires a few moments to download system software to the emulator hardware; the red light on the buffer box is on during the download.)

We wish to record this session on a file, so we enter:

```
*LIST :F1:NOV12A.LOG
*
```

Let's look at code memory, starting with the emulator's code memory map; enter:

```
*MAP
MAP = 0000H,1000H
```

This shows the initial location of the emulator's two 4K blocks of code memory. You can move these blocks around to serve the need for emulating from higher memory locations, but for these sessions we leave them as is.

To display or change code memory bytes as numeric values, use the CBYTE commands:

```
*CBYTE 0 TO 4K = 0                ;Clear the low block.
*CBYTE 0                          ;Display the contents of address 0.
CBYTE 0000H=00H
*CBYTE 0 TO 1FH                   ;Display a partition of addresses.
0000H=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
0010H=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
*
```

To display or change the contents of the on-chip data memory, use the DBYTE commands:

```
*DBYTE 0                          ;Display contents of address 0.
DBYTE 0000H=14H                   ;Initially the contents are 'random'.
*
*DBYTE 0 TO 7FH = 0               ;Clears the entire on-chip data memory.
*
*DBYTE 0 TO 1FH                   ;Display a partition of bytes.
0000H=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
0010H=00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
```

To display or change the contents of the on-chip special function registers, use the RBYTE commands. You can use the numeric address of the register:

```
*RBYTE 81H
RBYTE 0081H=07H
```

Or, you can use the system symbol for the register. The emulator requires you to precede the name of the symbol with a period (.) to distinguish it from other kinds of entries:

```
*RBYTE .SP                          ;Stack Pointer register.
RBYTE 0081H=07H
*
*RBY .PSW                           ;Program Status Word register
RBYTE 00D0H=00H
```

To display or change the values of the bit-addressable memory and registers, use the RBIT commands:

```
*RBIT 0
RBIT 0000H=0
*
*RBIT 0 TO 1FH
0000H=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0010H=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*
*RBIT .CY                           ;Using a system symbol for the bit
RBIT 00D7H=0                        ;address.
*
*RBIT .PSW+7                        ;Accessing a bit-addressable register by
RBIT 00D7H=0                        ;register name and bit number.
```

Certain registers can be displayed without using DBYTE or RBYTE:

```
*REGISTERS
PC     ACC  B    SP   DPTR   R0   R1   PSW
0000H  00H  00H  07H  0000H  00H  00H  00000000Y
*
*R0                                 ;Working registers in current bank.
R0=00H
*R1
R1=00H
*R7
R7=00H
*
*RBS                                ;Bank select for R0 through R7.
RBS=00H
*
*RBS = 3                            ;Change to bank 3.
*R0
R0=00H
*R0=55H
*R0
R0=55H
*RBS = 0
*R0
R0=00H
*STACK                              ;Display the stack and stack pointer.
SP STACK
07H 00H                             ;07H is the initial stack pointer, so this
06H 00H                             ;display means the stack is empty.
```

(Display continues on next page.)

```
05H 00H
04H 00H
03H 00H
02H 00H
01H 00H
00H 00H
*INTERRUPT                              ;Display interrupt flags.
        EA          SINT    TIMER1    EXTI1    TIMER0    EXTI0
IIP0                0       0         0        0         0
IIP1                0       0         0        0         0
IE      0           0       0         0        0         0
IP                  0       0         0        0         0
*
*PC                                     ;Program Counter
PC=0000H
*
*DPTR                                   ;Data Pointer
DPTR=0000H
*
```

We'll examine some of these displays further in later sessions.

Here are some commands that control the number bases used for console input and display:

```
*SUFFIX                                 ;SUFFIX controls the console input
H                                       ;Initially hexadecimal.
*
*CBYTE 1F                               ;With SUFFIX=H we can omit the H from
CBYTE 001FH=00H                         ;numbers entered in hex at the console.
*
*SUFFIX = T                             ;Switch to decimal (T) suffix.
*CBYTE 10                               ;Now numbers without suffixes are
CBYTE 00AH=00H                          ;assumed to be decimal.
*CBYTE 1F                               ;All digits must be valid for the current
CBYTE 1F#                               ;suffix, otherwise use an explicit suffix.
ERR 81:INVALID TOKEN
*
*SUFFIX = H                             ;The suffix can be set to hex (H),
*                                       ;decimal (T), octal (Q), or binary (Y).
*
*CBYTE 0 = 48H,45H,4CH,4CH,4FH          ;Sets up for next example.
*BASE                                   ;BASE controls the console displays.
H                                       ;Initially hexadecimal.
*BASE = ASCII                           ;BASE can be H, T, Q, Y, or ASCII.
*CBYTE 0 TO F
0000H=HELLO                             ;Displays bytes as ASCII characters.
*
*BASE = H
*
*EVALUATE 44H + 273T +345Q              ;EVALUATE does calculations.
1000111010Y 1072Q 570T 23AH ':' 023AH
```

The following two reset commands are useful for program control and error recovery:

```
*RESET ICE                              ;Resets emulator hardware and MAP.
*
*RESET CHIP                             ;Resets emulation processor.
```

To generate real data for the registers, we need a program to emulate. The remainder of session one shows you how to enter the demonstration program listed earlier in the manual. Let's start with the program labels. To define a symbol for an address, type:

```
*DEFINE .TEMP = FFFFH
*
```

To display and change the symbol, once defined:

```
*.TEMP
.TEMP=FFFFH
*
*.TEMP = ABCDH
*.TEMP
.TEMP=ABCDH
*
*SYMBOLS                    ;Display symbols.
.TEMP=ABCDH
```

To remove any user-defined symbol:

```
*REMOVE .TEMP               ;Remove one symbol.
*
*REMOVE SYMBOLS             ;Remove all symbols.
*SYMBOLS                    ;Table is empty.
*
```

Now let's define the seven symbols to be used as labels in the demo program:

```
*
*DEFINE .START = 100H       ;Refer to Program Listing.
*DEF .ENDSTART = 0121H
*DEF .LOOP = 0130H
*DEF .ENDLOOP = 138H
*DEF .TIMOUT = 0140H
*DEF .MIDOUT = 014CH
*DEF .ENDOUT = 150H
*
*SYMBOLS                    ;Displays the entire user symbol table.
.START=0100H
.ENDSTART=0121H
.LOOP=0130H
.ENDLOOP=0138H
.TIMOUT=0140H
.MIDOUT=014CH
.ENDOUT=0150H
*
```

The emulator lets you display and change code memory using the assembly language mnemonic instructions. To display code memory as disassembled instructions, type:

```
*DASM 0 TO 10H
0000H=ORL       A,R0
0001H=ORL       A,4CH
0003H=ORL       A,R4
0004H=ORL       A,R7
0005H=NOP
0006H=NOP
0007H=NOP
0008H=NOP
0009H=NOP
```

(Display continues on next page.)

```
000AH=NOP
000BH=NOP
000CH=NOP
000DH=NOP
000EH=NOP
000FH=NOP
0010H=NOP
*
```

The first four instructions are the characters 'HELLO' (entered to show BASE ASCII in an earlier example) interpreted as opcode and operand bytes. We want to use the one-line assembler to enter the "real" program. To get assistance in using the one-line assembler, ASM, we use the HELP command:

```
*HELP
Help is available for the following items. Type HELP followed by the item name. The
help items cannot be abbreviated. (For more information, type HELP HELP or HELP
INFO.)
Emulation:          Trace Collection:          Misc:          <address>
G0 GR  SY0          TR QR QR0 QR1 SY1          BASE           <CPU$keyword>
BR BR0 BR1                                     DISABLE        <exor>
STEP                Trace Display:             ENABLE         <ICE51$keyword>
                    TRACE MOVE PRINT           ERROR          <identifier>
                    OLDEST NEWEST              EVALUATE       <instruction>
                                               HELP           <masked$constant>
Change/Display/Define/Remove:                  INFO           <match$cond>
<CHANGE>  REMOVE CBYTE RBIT                    <LIGHTS>       <numeric$constant>
<DISPLAY> SYMBOL DBYTE DASM                    LIST           <partition>
REGISTER  RESET  PBYTE ASM                     LOAD           <string>
SECONDS   WRITE  RBYTE MAP                     SAVE           <string$constant>
DEFINE    STACK  XBYTE SY                      SUFFIX         <symbolic$ref>
                                               SYMBOLIC       <system$symbols>
Macro:                    Compound                            <trace$reference>
DEFINE    DIR             Commands:                           <unlimited$match$cond>
DISABLE   ENABLE          COUNT                               <users$symbols>
INCLUDE   PUT             IF
<MACRO$DISPLAY>           REPEAT
<MACRO$INVOCATION>
*
```

```
*HELP  ASM
ASM — Command to assemble instructions into 8051 code memory.
      (1) To display the current value of the assembly program counter, type:
          ASM
      (2) To change the value of the assembly program counter, type:
          ASM ORG <address> (EX: ASM ORG 400H)
      (3) To assemble an instruction into 8051 code memory at the address in the
          assembly program counter, type:
          ASM <instruction> (EX: ASM MOV A,R0)
          After the instruction has been assembled into memory, the updated
          assembly program counter will be displayed.
*
```

Now begin entering the program:

```
*ASM ORG 0                    ;Start assembly at 0.
0000H
*ASM                          ;Display ASM pointer.
0000H                         ;Initially 0.
*A AJMP .START                ;User-defined label.
0002H                         ;ASM pointer increments.
*A ORG .TIMER1                ;System-defined label.
```

(Display continues on next page)

```
001BH
*A AJMP .TIMOUT                 ;Note the abbreviation of ASM to A.
001DH
*
*A ORG .START                   ;Initialization routine.
0100H
*A CLR A                        ;Clear accumulator.
0101H
*A CLR C                        ;Clear carry flag.
0102H
*A MOV DTPR,#0000H              ;Clear data pointer.
A MOV DTPR#
ERR 80:SYNTAX ERROR             ;Oops, should be DPTR, not DTPR.
*ASM
0102H                           ;Note pointer not changed.
*A MOV DPTR,#0000H              ;Re-enter instruction.
0105H
*A MOV .P1, #00H                ;Clear port P1.
0108H
*A MOV .P2, #00H                ;Clear port P2.
010BH
*A MOV R0,#41H                  ;Character 'A'.
010DH
*A MOV .TMOD, #20H              ;Auto-reload timer mode.
0110H
*A MOV .TCON, #00H              ;Clear timer control.
0113H
*A MOV .TL1, #F4H               ;Initial timer value.
0116H
*A MOV .TH1, #F4H               ;Timer reload value.
0119H
*A MOV .IP, #08H                ;Timer 1 priority 1.
011CH
*A MOV .IE, #08H                ;Enable timer 1 interrupt.
011FH
*A SETB .TR1                    ;Start timer.
0121H
*A AJMP .LOOP                   ;Jump to main loop.
0123H
```

Let's display the program so far:

```
*
*DASM .START TO .ENDSTART
.START
0100H=CLR       A
0101H=CLR       C
0102H=MOV       DPTR,#0000H
0105H=MOV       .P1,#00H
0108H=MOV       .P2,#00H
010BH=MOV       R0,#41H
010DH=MOV       .TMOD,#20H
0110H=MOV       .TCON,#00H
0113H=MOV       .TL1,#F4H
0116H=MOV       .TH1,#F4H
0119H=MOV       .IP,#08H
011CH=MOV       .IE,#08H        ;Oops!
011FH=SETB      .TRI
.ENDSTART
0121H=AJMP      .LOOP
*
```

In scanning the instructions, we notice the error at location 011CH. Although the instruction assembled without error, we wanted "#88H", not "08H". To correct this error, type:

```
*A ORG 011CH
011CH
*A MOV .IE,#88H
011FH
*
```

Note that the display has picked up our user-defined labels .START (0100H) and .ENDSTART (0121H). Continue entering the program:

```
*
*A ORG .LOOP
0130H
*A ADD A, #01H
0132H
*A MOV .P2, A
0134H
*A JNC .LOOP
0136H
*A INC DPTR
0137H
*A CLR C
0138H
*A SJMP .LOOP
013AH
*
*D .LOOP TO .ENDLOOP                ;Note abbreviation of DASM to D.
.LOOP
0130H=ADD       A,#01H
0132F=MOV       .P2,A
0134H=JNC       .LOOP
0136H=INC       DPTR
0137H=CLR       C
.ENDLOOP
0138H=SJMP      .LOOP
*
*
*A ORG .TIMOUT
0140H
*A CLR .TR1
0142H
*A PUSH .PSW
0144H
*A MOV .P1, R0
0146H
*A INC R0
0147H
*A CJNE R0, #5BH, .MIDOUT
014AH
*A MOV R0, #41H
014CH
*A POP .PSW
014EH
*A SETB .TR1
0150H
*A RETI
0151H
```

(Display continues on next page)

```
*
*D .TIMOUT TO .ENDOUT
.TIMOUT
0140H=CLR       .TR1
0142H=PUSH      .PSW
0144H=MOV       .P1,R0
0146H=INC       R0
0147H=CJNE      R0,#5BH,.MIDOUT
014AH=MOV       R0,#41H
.MIDOUT
014CH=POP       .PSW
014EH=SETB      .TR1
.ENDOUT
0150H=RETI
*
```

Now that the program has been entered, save it in a file for use in the next session:

```
*SAVE :F1:DEMO.HEX 0 TO 150H
*
```

The SAVE saves the code in the partition 0 through 150H, and our symbol table. This is the end of session one. To exit the emulator system and return to ISIS-II, type:

```
*EXIT
```

In session two you learn how to load a program from disk, emulate in real time and single step, and display the trace information collected during emulation. You will encounter the basic forms of emulation and trace controls, emphasizing techniques to maximize the amount of useful information captured in the buffer.

To begin, invoke the emulator as before:

```
>:F1:ICE51
ISIS-II ICE-51 Vn.n
FOR COMMAND ENTRY ASSISTANCE, TYPE HELP
*
*LIST :F1:NOV18A.LOG                ;This LIST command preserves a
*                                   ;record of the session.
```

Next, load the program entered in session one:

```
*LOAD :F1:DEMO.HEX
```

Verify that the program and its symbol table have been loaded correctly:

```
*DASM .START TO .ENDOUT
.START
0100H=CLR       A
0101H=CLR       C
0102H=MOV       DPTR,#0000H
0105H=MOV       .P1,#00H
0108H=MOV       .P2,#00H
010BH=MOV       R0,#41H
010DH=MOV       .TMOD,#20H
0110H=MOV       .TCON,#00H
0113H=MOV       .TL1,#F4H
0116H=MOV       .TH1,#F4H
0119H=MOV       .IP,#08H
011CH=MOV       .IE,#88H
011FH=SETB      .TR1
.ENDSTART
0121H=AJMP      .LOOP
0123H=NOP
0124H=NOP
0125H=NOP
0126H=NOP
0127H=NOP
0128H=NOP
0129H=NOP
012AH=NOP
012BH=NOP
012CH=NOP
012DH=NOP
012EH=NOP
012FH=NOP
.LOOP
0130H=ADD       A,#01H
0132H=MOV       .P2,A
0134H=JNC       .LOOP
```

(Display continues on next page)

```
0136H=INC       DPTR
0137H=CLR       C
.ENDLOOP
0138H=SJMP      .LOOP
013AH=NOP
013BH=NOP
013CH=NOP
013DH=NOP
013EH=NOP
013FH=NOP
.TIMOUT
0140H=CLR       .TR1
0142H=PUSH      .PSW
0144H=MOV       .P1,R0
0146H=INC       R0
0147H=CJNE      R0,#5BH,.MIDOUT
014AH=MOV       R0,#41H
.MIDOUT
014CH=POP       .PSW
014EH=SETB      .TR1
.ENDOUT
0150H=RETI
*
```

Here are the initial values of the registers and displays we shall be using during session two:

```
*TM1
TM1=0000H
```

The high and low bytes of Timer 1 are both 00H.

```
*
*R
PC      ACC  B    SP   DPTR   R0   R1   PSW
0000H   00H  00H  07H  0000H  FFH  89H  00000000Y
```

The REGISTERS display (abbreviation: R) shows the program counter PC at 0000H, the accumulator (ACC) at 00H, stack pointer at 07H, R0 and R1 with 'random' values, and the program status word all zeros. Our program does not use the multiply register (B), initialized at 00H.

```
*
*INT
        EA      SINT    TIMER1  EXTI1   TIMER0  EXTI0
IIP0            0       0       0       0       0
IIP1            0       0       0       0       0
IE      0       0       0       0       0       0
IP              0       0       0       0       0
*
```

The INTERRUPTS display (abbreviation: INT) shows the interrupt priority register (IP) cleared (all interrupts are priority zero). The interrupt enable register (IE) is likewise cleared; no interrupts are enabled. No interrupts are in progress; IIP0 is the interrupt-in-progress flag for priority zero interrupts, and IIP1 refers to priority one interrupts in progress. Note that the EA (Enable All interrupts) bit is part of the IE register.

```
*RBYTE .TCON
RBYTE 0088H=00H
*
*RBYTE .TMOD
*RBYTE 0089H=00H
```

```
*
*RBY .P1
RBYTE 0090H=FFH
*
*RBY .P2
RBYTE 00A0H=FFH
*
```

Using the RBYTE commands (abbreviation: RBY), we display the initial values of the timer control register, the timer mode register, port P1, and port P2. The two timer registers are initially 00H, and ports are initialized to FFH.

Here are the initial values of the emulation and trace controls we will be using in session two:

```
*GR
GR=FOREVER
BR0 = RESET
BR1 = RESET
```

The "GO register" (GR) shows the breakpoints that are currently enabled to halt real-time emulation. Initially, no breakpoints are enabled. Once begun, emulation continues until we press the ESC key.

```
*TR
TR=FOREVER
QR0 = RESET
QR1 = RESET
```

The "trace register" (TR) shows the factors that are enabled to control trace collection during real-time emulation. Initially, no factors are enabled. Every cycle of every instruction executed will be collected in the trace buffer.

```
*BUFFERSIZE
BUFFERSIZE=0T
*
*SECONDS
0 MICROSECONDS
```

The BUFFERSIZE register shows the number of valid frames of trace information in the trace buffer (initially 0, maximum 1000T). The SECONDS register shows the value of the real-time emulation timer, initially 0.

Now we're ready to start emulating. The simplest emulation command is the GO command:

```
*GO                                   :Press the ESC key to halt.
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT
PROCESSING ABORTED
*
```

Let's look at some results of emulation (the results you obtain will be different).

```
*SECONDS
2,601,981 MICROSECONDS
*
*BUFFERSIZE                           ;The buffer has overflowed
BUFFERSIZE=1000T
*
*R
PC     ACC  B    SP   DPTR   R0   R1   PSW
0140H  04H  00H  09H  023EH  4DH  C8H  00000001Y
```

(Display continues on next page)

```
*
*TM1
TM1=F4FBH
*
*INT
          EA        SINT    TIMER1    EXTI1    TIMER0    EXTI0
IIP0                 0        0         0        0         0
IIP1                 0        1         0        0         0
IE        1          0        1         0        0         0
IP                   0        1         0        0         0
*
```

The INTERRUPTS display reminds us that the .TIMOUT interrupt in progress flag may still be set. This will prevent any interrupts from happening if we start over. One way to clear this flag, is:

```
*RESET CHIP
*
*INT
          EA        SINT    TIMER1    EXTI1    TIMER0    EXTI0
IIP0                 0        0         0        0         0
IIP1                 0        0         0        0         0
IE        0          0        0         0        0         0
IP                   0        0         0        0         0
*
```

The simplest sequence is GO ... ESC. This sequence allows the buffer to overflow, so that we retain the most recent 1000 frames. However, these may not be the frames with the most interest to us. The remainder of this chapter explores ways to emulate and trace, retaining the maximum amount of useful information.

First, we can restrict the number of instructions emulated, using a command like:

```
*GO FROM .START TILL .LOOP
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP+0002H
*
*BUF
BUFFERSIZE=100T
```

To see these frames, type:

```
*
*PRINT ALL
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
.START
0000:  0100H  E4       CLR   A                  FFH  FFH  FFH  0
0004:  0101H  C3       CLR   C                  FFH  FFH  FFH  0
0008:  0102H  900000   MOV   DPTR,#0000H        FFH  FFH  FFH  0
0016:  0105H  759000   MOV   .P1,#00H           FFH  FFH  FFH  0
0024:  0108H  75A000   MOV   .P2,#00H           FFH  FFH  FFH  0
0032:  010BH  7841     MOV   R0,#41H            00H  FFH  FFH  0
0036:  010DH  758920   MOV   .TMOD,#20H         00H  00H  FFH  0
0044:  0110H  758800   MOV   .TCON,#00H         00H  00H  FFH  0
0052:  0113H  758BF4   MOV   .TL1,#F4H          00H  00H  FFH  0
0060:  0116H  758DF4   MOV   .TH1,#F4H          00H  00H  FFH  0
0068:  0119H  75B808   MOV   .IP,#08H           00H  00H  FFH  0
0076:  011CH  75A888   MOV   .IE,#88H           00H  00H  FFH  0
0084:  011FH  D28E     SETB  .TR1                00H  00H  FFH  0
.ENDSTART
0088:  0121H  2130     AJMP  .LOOP              00H  00H  FFH  0
.LOOP
0096:  0130H  2401     ADD   A,#01H             00H  00H  FFH  0
*
```

In the display, the number at the left is the frame number; the buffer holds 1000 (decimal) frames. There are four frames for each cycle; all these instructions are either one or two cycles. The columns marked LOC, OBJ, and INSTRUCTION are the opcode address, hexadecimal value, and mnemonic disassembly of each instruction. P1, P2, and P0 are the ports. TOVF is the trace buffer overflow flag (0 for the first 1000 frames, 1 thereafter).

This example shows the effect of the FROM and TILL clauses in the GO command. The clause FROM .START caused emulation to begin at 0100H. The clause TILL .LOOP caused emulation to break after executing the instruction at 0130H. Since that's all we emulated, that's all we traced.

Another way to keep trace from overflowing (and losing the first instructions emulated) is to use a "trace trigger":

```
*TR=AFTER 0                          ;Executing address 0 turns
*                                    ;trace on for 1000 frames.
*TR                                  ;Display the trace register.
TR=AFTER QR0                         ;The trigger mode uses one of
QR0 = LOCATION IS 0000H              ;the two qualifier registers.
QR1 = RESET
*
*
```

Now we'll emulate for one millisecond, using TILL .ENDLOOP as the breakpoint:

```
*RESET CHIP                          ;Turn off the timer.
*GO FROM 0 TILL .ENDLOOP
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP
*
```

Let's see the most recent 25 instructions in the buffer:

```
*NEWEST
*
```

(This command really isn't necessary since the "display pointer" is already at NEWEST, the most recent instruction, after emulation breaks.)

```
*
*PRINT -25
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0843:  0132H  F5A0    MOV    .P2,A             48H  0CH  FFH   0
0847:  0134H  50FA    JNC    .LOOP             48H  0CH  FFH   0
WARN CC:UNEXPECTED TRACE
.LOOP
0855:  0130H  2424    ADD    A,#24H            48H  0DH  FFH   0
0863:  001BH  2140    AJMP   .TIMOUT           48H  0DH  FFH   0
.TIMOUT
0871:  0140H  C28E    CLR    .TR1              48H  0DH  FFH   0
0875:  0142H  C0D0    PUSH   .PSW              48H  0DH  FFH   0
0883:  0144H  8890    MOV    .P1,R0            48H  0DH  FFH   0
0891:  0146H  08      INC    R0                48H  0DH  FFH   0
0895:  0147H  B85B02  CJNE   R0,#5BH,.MIDOUT   49H  0DH  FFH   0
.MIDOUT
0903:  014CH  D0D0    POP    .PSW              49H  0DH  FFH   0
0911:  014EH  D28E    SETB   .TR1              49H  0DH  FFH   0
.ENDOUT
0915:  0150H  32      RETI                     49H  0DH  FFH   0
.LOOP
0923:  0130H  2401    ADD    A,#01H            49H  0DH  FFH   0
0927:  0132H  F5A0    MOV    .P2,A             49H  0DH  FFH   0
```

(Display continues on next page)

```
0931:    0134H  50FA    JNC    .LOOP              49H  0DH  FFH   0
WARN CC:UNEXPECTED TRACE
.LOOP
0939:    0130H  2424    ADD    A,#24H             49H  0EH  FFH   0
0947:    001BH  2140    AJMP   .TIMOUT            49H  0EH  FFH   0
.TIMOUT
0955:    0140H  C28E    CLR    .TR1               49H  0EH  FFH   0
0959:    0142H  C0D0    PUSH   .PSW               49H  0EH  FFH   0
0967:    0144H  8890    MOV    .P1,R0             49H  0EH  FFH   0
0975:    0146H  08      INC    R0                 49H  0EH  FFH   0
0979:    0147H  B85B02  CJNE   R0,#5BH,.MIDOUT    4AH  0EH  FFH   0
.MIDOUT
0987:    014CH  D0D0    POP    .PSW               4AH  0EH  FFH   0
0995:    014EH  D28E    SETB   .TR1               4AH  0EH  FFH   0
.ENDOUT
0999:    0150H  32      RETI                      4AH  0EH  FFH   0
*
```

NOTE

The warning messages have no effect on the command operation. See "Frame Mode Trace Displays" in chapter 6 of the *Operating Instructions* for an explanation.

Now let's see if the first instruction (at address 0) is still in the buffer:

```
*OLDEST
*
*P 1
FRAME  LOC    OBJ    INSTRUCTION          P1   P2   P0   TOVF
.START
0007H  0100H  E4     CLR    A             00H  00H  FFH   0
*
```

But it is not. The reason is that we were using a qualifier register match to trigger trace on, and the match turns trace on starting with the next frame *after* the one that matched. Since we lost this first frame of the instruction at 0, the instruction is not displayed in the INSTRUCTION mode. Note that the first instruction in the buffer starts at frame number 7, not frame 0.

Let's try another trigger mode, this time to turn trace off at a point of interest, and retain the 1000 frames prior to that point. First, however, we'd better turn off the timer:

```
*
*RESET CHIP                          ;Turn off timer.
*
```

Here's the trigger mode:

```
*TR = TILL .ENDOUT                   ;Halt trace at the end of the
*                                    ;first interrupt.
*TR
TR=TILL QR0
QR0 = LOCATION IS 0150H
QR1 = RESET
*
*GO FROM 0                           ;The system will remember our
EMULATION BEGUN                      ;breakpoint, .ENDLOOP.
EMULATION TERMINATED, PC=.LOOP
```

```
*
*GR                                   ;Here it is, just as a reminder.
GR=TILL BR0
BR0 = LOCATION IS 0138H
BR1 = RESET
*
*P ALL
FRAME   LOC     OBJ       INSTRUCTION              P1   P2   P0   TOVF
0000:   0000H   2100      AJMP   .START            FFH  FFH  FFH   0
.START
0008:   0100H   E4        CLR    A                 FFH  FFH  FFH   0
0012:   0101H   C3        CLR    C                 FFH  FFH  FFH   0
0016:   0102H   900000    MOV    DPTR,#0000H       FFH  FFH  FFH   0
0024:   0105H   759000    MOV    .P1,#00H          FFH  FFH  FFH   0
0032:   0108H   75A000    MOV    .P2,#00H          FFH  FFH  FFH   0
0040:   010BH   7841      MOV    R0,#41H           00H  FFH  FFH   0
0044:   010DH   758920    MOV    .TMOD,#20H        00H  00H  FFH   0
0052:   0110H   758800    MOV    .TCON,#00H        00H  00H  FFH   0
0060:   0113H   758BF4    MOV    .TL1,F4H          00H  00H  FFH   0
0068:   0116H   758DF4    MOV    .TH1,#F4H         00H  00H  FFH   0
0076:   0119H   75B808    MOV    .IP,#08H          00H  00H  FFH   0
0084:   011CH   75A888    MOV    .IE,#88H          00H  00H  FFH   0
0092:   011FH   D28E      SETB   .TR1              00H  00H  FFH   0
.ENDSTART
0096:   0121H   2130      AJMP   .LOOP             00H  00H  FFH   0
.LOOP
0104:   0130H   2401      ADD    A,#01H            00H  00H  FFH   0
0108:   0132H   F5A0      MOV    .P2,A             00H  00H  FFH   0
0112:   0134H   50FA      JNC    .LOOP             00H  00H  FFH   0
.LOOP
0120:   0130H   2401      ADD    A,#01H            00H  01H  FFH   0
0124:   0132H   F5A0      MOV    .P2,A             00H  01H  FFH   0
0128:   0134H   50FA      JNC    .LOOP             00H  01H  FFH   0
.LOOP
0136:   0130H   2401      ADD    A,#01H            00H  02H  FFH   0
0140:   0132H   F5A0      MOV    .P2,A             00H  02H  FFH   0
0144:   0134H   50FA      JNC    .LOOP             00H  02H  FFH   0
WARN CC:UNEXPECTED TRACE
.LOOP
0152:   0130H   2424      ADD    A,#24H            00H  03H  FFH   0
0160:   001BH   2140      AJMP   .TIMOUT           00H  03H  FFH   0
.TIMOUT
0168:   0140H   C28E      CLR    .TR1              00H  03H  FFH   0
0172:   0142H   C0D0      PUSH   .PSW              00H  03H  FFH   0
0180:   0144H   8890      MOV    .P1,R0            00H  03H  FFH   0
0188:   0146H   08        INC    R0                00H  03H  FFH   0
0192:   0147H   B85B02    CJNE   R0,#5BH,.MIDOUT   41H  03H  FFH   0
.MIDOUT
0200:   014CH   D0D0      POP    .PSW              41H  03H  FFH   0
0208:   014EH   D28E      SETB   .TR1              41H  03H  FFH   0
.ENDOUT
0212:   0150H   32        RETI                     41H  03H  FFH   0
*
*INT
        EA          SINT   TIMER1      EXTI1      TIMER0      EXTI0
IIP0                 0       0          0          0           0
IIP1                 0       0          0          0           0
IE       1           0       1          0          0           0
IP                   0       1          0          0           0
*
```

Do you have time for just one more trace magnification technique? This one allows the buffer to overflow, but it maximizes the number of instructions in the buffer. An instruction is displayed in INSTRUCTION mode if its first (LOC) frame is in the buffer. To maximize the number of LOC frames, the command is:

```
*TR = VALUE IS XXH
*
*TR
TR=QR0
QR0 = VALUE IS XXXXXXXXY
QR1 = RESET
*
*RESET CHIP                          ;Turn off timer.
*GO FROM 0
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP
*
*PRINT -10
FRAME  LOC     OBJ      INSTRUCTION              P1   P2   P0   TOVF
.TIMOUT
0978:  0140H   C28E     CLR    .TR1              56H  00H  FFH    1
0979:  0142H   C0D0     PUSH   .PSW              56H  00H  FFH    1
0982:  0144H   8890     MOV    .P1,R0            56H  00H  FFH    1
0985:  0146H   08       INC    R0                56H  00H  FFH    1
0986:  0147H   B85B02   CJNE   R0,#5BH,.MIDOUT   57H  00H  FFH    1
.MIDOUT
0989:  014CH   D0D0     POP    .PSW              57H  00H  FFH    1
0992:  014EH   D28E     SETB   .TR1              57H  00H  FFH    1
.ENDOUT
0993:  0150H   32       RETI                     57H  00H  FFH    1
0996:  0137H   C3       CLR    C                 57H  00H  FFH    1
.ENDLOOP
0997:  0138H   80F6     SJMP   .LOOP             57H  00H  FFH    1
*
```

Let's see that in FRAME mode:

```
*TRACE = FRAMES
*
*PRINT -10
FRAME  TYPE  ADDR   DATA   INSTRUCTION      P1   P2   P0   TOVF
0990:  VLO   014DH                          57H  00H  FFH    1
0991:  VLO   014DH                          57H  00H  FFH    1
0992:  LOC   014EH  (D2H)  (SETB  - - -)    57H  00H  FFH    1
.ENDOUT
0993:  LOC   0150H  (32H)  (RETI)           57H  00H  FFH    1
0994:  VLO   0151H                          57H  00H  FFH    1
0995:  VLO   0151H                          57H  00H  FFH    1
0996:  LOC   0137H  (C3H)  (CLR   C)        57H  00H  FFH    1
.ENDLOOP
0997:  LOC   0138H  (80H)  (SJMP  - - -)    57H  00H  FFH    1
0998:  VLO   0139H                          57H  00H  FFH    1
0999:  VLO   013AH                          57H  00H  FFH    1
*
```

(In the column labeled "TYPE", LOC and OPC frames are the opcode address and the opcode byte, respectively, and VLO and VAL frames are the operand address and operand values, respectively.)

Each instruction occupies a maximum of three frames, compared to a maximum of eight without this control. The buffer thus can contain between 300 and 1,000 instructions.

To return to INSTRUCTION mode of display, type:

```
*TRACE=INSTRUCTION
```

In addition to real-time emulation (GO), you can execute your program one instruction at a time, breaking between each instruction. Let's get back to the beginning of the program:

```
*RESET CHIP
*
*PC
PC=0000H
*
```

Now, to execute one instruction, type:

```
*STEP
EMULATION BEGUN
EMULATION TERMINATED, PC=.START
*
```

Here are some of the results of this emulation:

```
*
*PRINT -1
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0000:  0000H  2100    AJMP  .START             FFH  FFH  FFH    0
*
*SECONDS
0 MICROSECONDS
*
*REGISTERS
PC     ACC  B    SP   DPTR   R0   R1   PSW
0100H  00H  00H  07H  0000H  00H  C8H  00000000Y
```

Trace is always collected during single step (you can't turn it on and off); the emulation timer (SECONDS) is inoperative. The REGISTERS display shows every register still reset except the program counter, since the instruction did not affect these registers.

We'll present more examples of STEP in the next session. This is the end of session two:

```
*EXIT
```

SESSION THREE

In session three you will learn how to create compound commands and macro command blocks to assist in program development and to automate system testing.

NOTE

Errors used in compound commands and macro definitions can be corrected by using the BACKSPACE key as described in the Introduction to this manual, *but only* until the RETURN key for a given line has been pressed. After a line has been terminated, you cannot correct any errors in that line. A macro definition can be saved on a file, as described in session four, and edited off-line. During session three, however, if you find an error in a previous line while you are within a compound command or macro definition, you should press the ESC key to abort the command entry and start over.

To begin, invoke the emulator as before:

```
>:F1:ICE51
ISIS-II ICE-51 Vn.n
FOR COMMAND ENTRY ASSISTANCE, TYPE HELP
*
*LIST :F1:NOV19A.LOG                ;Record the session, if desired.
*
*LOAD :F1:DEMO.HEX                  ;Load sample program
*
```

To introduce the concept of compound commands, enter the following command sequence:

```
*STEP
EMULATION BEGUN
EMULATION TERMINATED, PC=.START
*PRINT -1
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0000:    0000H  2100    AJMP  .START            FFH  FFH  FFH    0
*REGISTERS
PC      ACC  B    SP   DPTR  R0   R1   PSW
0100H   00H  00H  07H  0000H 51H  00H  00000000Y
*
```

This sequence of STEP, PRINT, REGISTERS can be repeated indefinitely by entering the following compound command:

```
*REPEAT                             ;REPEAT starts the compound command
                                    sequence.
.*STEP                              ;The period (.) before the prompt
.*PRINT -1                          ;shows that we are within the compound
.*REGISTERS                         ;command. End each inner line with a
                                    RETURN.
.*END                               ;END terminates the block and starts
                                    execution.
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0001H
```

(Display continues on next page)

```
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
.START
0008:    0100H  E4       CLR    A                 FFH  FFH  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
0101H  00H   00H   07H   0000H  51H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0002H
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0012:    0101H  C3       CLR    C                 FFH  FFH  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
0102H  00H   00H   07H   0000H  51H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0005H
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0016:    0102H  900000   MOV    DPTR,#0000H       FFH  FFH  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
0105H  00H   00H   07H   0000H  51H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0008H
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0024:    0105H  759000   MOV    .P,#00H           FFH  FFH  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
0108H  00H   00H   07H   0000H  51H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+000BH
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0032:    0108H  75A000   MOV    .P2,#00H          00H  FFH  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
010BH  00H   00H   07H   0000H  51H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+000DH
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0040:    010BH  7841     MOV    R0,#41H           00H  00H  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
010DH  00H   00H   07H   0000H  41H   00H   00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0010H
FRAME  LOC    OBJ      INSTRUCTION              P1   P2   P0   TOVF
0044:    010DH  758920   MOV    .TMOD,#20H        00H  00H  FFH   0
PC      ACC  B     SP    DPTR   R0    R1    PSW
PROCESSING ABORTED
*
```

Because this repeat loop does not terminate automatically, we press the ESC key to abort processing.

To make the loop terminate on condition, we can use an UNTIL clause:

```
*RESET CHIP
*REPEAT                          ;Start of command block.
.*STEP                           ;Execute one instruction.
.*UNTIL PC = .TIMOUT             ;Check the program counter.
.*END                            ;End of loop.
EMULATION BEGUN
EMULATION TERMINATED, PC=.START
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0001H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0002H
EMULATION BEGUN
```

```
EMULATION TERMINATED, PC=.START+0005H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0008H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+000BH
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+000DH
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0010H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0013H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0016H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0019H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+001CH
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+001FH
EMULATION BEGUN
EMULATION TERMINATED, PC=.ENDSTART
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP+0002H
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP+0004H
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP
EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP+0002H
EMULATION BEGUN
EMULATION TERMINATED, PC=001BH
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT
*
```

Another way to terminate a loop automatically is to use COUNT instead of REPEAT:

```
*RESET CHIP
*
*COUNT 5                          ;Loop to execute five steps.
.*STEP
.*END
EMULATION BEGUN
EMULATION TERMINATED, PC=.START
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0001H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0002H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0005H
EMULATION BEGUN
EMULATION TERMINATED, PC=.START+0008H
*
```

We'll see more of REPEAT and COUNT later in the session. A third kind of compound command, the IF command, produces conditional execution of commands. To show how the IF command works, we'll introduce another new command, the WRITE command:

```
*WRITE 'HOW ABOUT THAT?'
HOW ABOUT THAT?
*
```

WRITE allows you to create console displays. Now for the IF command:

```
*IF PC > 0 THEN
.*WRITE 'I AM SOMEWHERE'
.*ELSE
.*WRITE 'I AM NOWHERE'
.*END
I AM SOMEWHERE
*
```

Instead of "PC > 0" you can use any expression. When the expression after IF is "true" (i.e., when the result has a 1 in the least significant bit), the commands after the IF are executed. When the IF expression is "false", the commands after the ELSE are executed instead.

REPEAT, COUNT, and IF are the basic compound commands. Now we are ready to create some command macros and "automate" some of our debugging tasks. Here's the first one:

```
*DEFINE :S1                    ;This macro is named :S1.
.*REPEAT                       ;The period shows we are "within" the
.*STEP                         ;macro.
.*PRINT -1
.*REGISTERS
.*END                          ;END terminates the REPEAT loop.
.*EM                           ;EM terminates the macro definition.
*
```

Note that the macro does not begin to execute immediately. To have the macro execute, we must call it by name. Let's use it to look at the beginning of the TIMOUT routine:

```
*
*RESET CHIP
*GO FROM .START TILL .TIMER1
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT
*
*:S1                           ;This is the macro call.
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0002H
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
.TIMOUT
0160:  0140H  C28E    CLR   .TR1              00H  03H  FFH   0
PC     ACC  B    SP   DPTR  R0   R1   PSW
0142H  03H  00H  09H  0000H  41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0004H
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0164:  0142H  C0D0    PUSH  .PSW              00H  03H  FFH   0
PC     ACC  B    SP   DPTR  R0   R1   PSW
0144H  03H  00H  0AH  0000H  41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0006H
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0172:  0144H  8890    MOV   .P1,R0             00H  03H  FFH   0
```

```
PC     ACC  B    SP   DPTR  R0   R1   PSW
0146H  03H  00H  0AH  0000H 41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0007H
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0180:  0146H  08      INC    R0                41H  03H  FFH  0
PC     ACC  B    SP   DPTR  R0   R1   PSW
0147H  03H  00H  0AH  0000H 42H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.MIDOUT
FRAME  LOC    OBJ     INSTRUCTION
PROCESSING ABORTED
*
```

When we call the :STEP macro, the sequence of STEP, PRINT, REGISTERS begins to repeat indefinitely. We press ESC to halt the loop. Here's another macro; this one is based on the COUNT command:

```
*DEF :S5                    ;This macro is named :S5
.*COUNT 5                   ;Begins COUNT loop (five times).
.*STEP                      ;Execute one instruction.
.*REGISTERS                 ;Display registers after each one.
.*END                       ;END of the COUNT loop.
.*PRINT -5                  ;Display five instructions from trace.
.*EM                        ;EM ends the macro definition.
*
```

Let's look at TIMOUT again:

```
*
*RESET CHIP
*GO FROM .START TILL .TIMER1
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT
*
*:S10
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMEOUT+0002H
PC     ACC  B    SP   DPTR  R0   R1   PSW
0142H  03H  00H  09H  0000H 41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0004H
PC     ACC  B    SP   DPTR  R0   R1   PSW
0144H  03H  00H  0AH  0000H 41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0006H
PC     ACC  B    SP   DPTR  R0   R1   PSW
0146H  03H  00H  0AH  0000H 41H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0007H
PC     ACC  B    SP   DPTR  R0   R1   PSW
0147H  03H  00H  0AH  0000H 42H  A1H  00000000Y
EMULATION BEGUN
EMULATION TERMINATED, PC=.MIDOUT
PC     ACC  B    SP   DPTR  R0   R1   PSW
014CH  03H  00H  0AH  0000H 42H  A1H  10000000Y
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
.TIMOUT
0160:  0140H  C28E    CLR    .TR1              00H  03H  FFH  0
0164:  0142H  C0D0    PUSH   .PSW              00H  03H  FFH  0
0172:  0144H  8890    MOV    .P1,R0            00H  03H  FFH  0
0180:  0146H  08      INC    R0                41H  03H  FFH  0
0184:  0147H  B85B02  CJNE   R0,#5BH,.MIDOUT   41H  03H  FFH  0
```

Macro :S5 works all right, but suppose we want to execute some number of steps other than five? Rather than creating a separate macro for each desired number of steps, the next macro uses a parameter in the definition so that we can change the number each time we call the macro:

```
*DEFINE :S                    ;This macro is named :S.
.*COUNT %0                    ;"%0" is the parameter.
.*WRITE ' '                   ;WRITE a blank (puts a blank line in the
.*STEP                        ;display.)
.*WRITE ' '
.*REGISTERS
.*END                         ;END terminates the COUNT loop.
.*WRITE ' '
.*PRINT -%0                   ;PRINT the same number that we
                              ;COUNT.
.*EM                          ;EM terminates the macro definition.
```

In order to see more clearly how the parameter operates, we turn on the display of the macro "expansion":

```
*ENABLE EXPANSION
*
```

One more time, let's look at TIMOUT:

```
*RESET CHIP
*GO FROM .START TILL .TIMER1
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT
*
*:S 6                         ;We call the macro with parameter = 6.
.*COUNT 6                     ;The 6 is substituted wherever "%0"
..*WRITE ' '                  ;appears in the definition.
..*STEP                       ;The periods show that we're within
..*WRITE ' '                  ;two blocks (COUNT inside a macro
..*REGISTERS                  ;definition).
..*END                        ;END of count loop.
.*WRITE ' '                   ;
.*PRINT -6                    ;Parameter 6 goes here too.
.*EM                          ;End of macro expansion, execution
                              ;begins.
EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0002H

PC     ACC  B    SP   DPTR  R0   R1   PSW
0142H  03H  00H  09H  0000H 41H  00H  00000000Y

EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0004H

PC     ACC  B    SP   DPTR  R0   R1   PSW
0144H  03H  00H  0AH  0000H 41H  00H  00000000Y

EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0006H

PC     ACC  B    SP   DPTR  R0   R1   PSW
0146H  03H  00H  0AH  0000H 41H  00H  00000000Y

EMULATION BEGUN
EMULATION TERMINATED, PC=.TIMOUT+0007H

PC     ACC  B    SP   DPTR  R0   R1   PSW
0147H  03H  00H  0AH  0000H 42H  00H  00000000Y
```

(Display continues on next page.)

```
EMULATION BEGUN
EMULATION TERMINATED, PC=.MIDOUT

PC     ACC  B    SP   DPTR   R0   R1    PSW
014CH  03H  00H  0AH  0000H  42H  00H   10000000Y

EMULATION BEGUN
EMULATION TERMINATED, PC=.MIDOUT+0002H

PC     ACC  B    SP   DPTR   R0   R1    PSW
014EH  03H  00H  09H  0000H  42H  00H   00000000Y

FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
.TIMOUT
0160:  0140H  C28E    CLR   .TR1               00H  03H  FFH  0
0164:  0142H  C0D0    PUSH  .PSW               00H  03H  FFH  0
0172:  0144H  8890    MOV   .P1,R0             00H  03H  FFH  0
0180:  0146H  08      INC   R0                 41H  03H  FFH  0
0184:  0147H  B85B02  CJNE  R0,#5BH,.MIDOUT    41H  03H  FFH  0
.MIDOUT
0192:  014CH  D0D0    POP   .PSW               41H  03H  FFH  0
*
```

With macro expansion displayed on the screen, we can see how the parameter "6" we gave in the macro call is substituted wherever we used "%0" in the macro definition. After the substitutions have been made, the resulting commands are executed. The only command we entered was ":S 6", and the system does the rest. You can leave macro expansion enabled, or you can turn it off with:

```
*DISABLE EXPANSION
*
```

To see the names of the macros we have defined so far, type:

```
*DIR
S1
S5
S
*
```

To review the definition of a macro (say, :S5), type:

```
*MACRO :S5
DEFINE :S5
COUNT 5
STEP
REGISTERS
END
PRINT -5
EM
```

To remove a macro definition, type:

```
*REMOVE :S1
*
```

To remove all macro definitions, type:

```
*REMOVE MACROS
*
```

These basic forms of macros are all we need to create our own "design aids" and diagnostic tests. Let's continue with a macro to initialize the emulator:

```
*DEFINE :INIT
.*RESET CHIP
.*RESET ICE
.*REMOVE SYMBOLS
.*LOAD :F1:DEMO.HEX
.*DBYTE 0 TO 7FH = 0
.*WRITE 'ALL SET & READY TO GO'
.*EM
*
*:INIT
ALL SET & READY TO GO
*
```

Any time we want to start over, we can call :INIT.

This next macro uses a "trace reference" to locate a frame in the trace buffer that contains a value of interest to us. Here's how a trace reference to port P1 works. Suppose the buffer is at OLDEST, so that the PRINT command displays frame 0000:

```
*STEP FROM 0                          ;Put one instruction in buffer.
*
*OLDEST
*
*PRINT 1
FRAME  LOC    OBJ     INSTRUCTION              P1   P2   P0   TOVF
0000:  0000H  2100    AJMP   .START            FFH  FFH  FFH    0
*
```

Now, to reference the value of P1 from this frame, we use a trace reference, FRAME P1. This reference is not a command, however, so we need to use EVALUATE to see the value:

```
*EVALUATE FRAME P1
1111111111111111Y 177777Q 65535T FFH ' '  00FFH
```

Here's the macro to look through the buffer for a value we seek (such as P1 = 41H); the parameter %0 lets us specify what we're looking for each time we call the macro:

```
*DEFINE :FIND
.*OLDEST
.*TRACE = FRAMES
.*COUNT BUFFERSIZE
.*MOVE 1
.*IF FRAME %0
.*PRINT 1
.*ENDIF
.*UNTIL FRAME %0
.*ENDCOUNT
.*TRACE = INSTRUCTIONS
.*EM
```

Before we run this macro, let's put something in the buffer:

```
*TR = AFTER 0
*GO FROM 0 TILL .ENDLOOP

EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP
*
```

Now let's find the frame that records when P1 first equals 41H (the character 'A'):

```
*:FIND P1 = 41H
FRAME   TYPE   ADDR   DATA   INSTRUCTION      P1    P2    P0    TOVF
0188:   OPC           08H    (INC     R0)     41H   03H   FFH     0
*
```

(This macro takes several seconds to execute.)

Now we can combine these two macros to simulate an automated test. This final macro, :TEST, shows how one macro can call another.

```
*
*DEFINE :TEST
.*:INIT                                ;Calls :INIT.
.*WRITE ' '
.*TR = AFTER 0
.*GO FROM 0 TILL .ENDLOOP              ;Emulate and trace.
.*WRITE ' '
.*:FIND P1=41H                         ;Find 'A'.
.*WRITE ' '
.*:FIND P1=42H                         ;Find 'B'.
.*WRITE ' '
.*WRITE 'END OF TEST'                  ;That's the test.
.*WRITE ' '
.*EM
*
```

To see the names of the macros you have defined, type:

```
*DIR
INIT
FIND
TEST
```

The earlier macros (:S1, :S5, and :S) were removed, so they are no longer in the table.

We are going to save these macro definitions in a file for use in the last session. Type:

```
PUT :F1:TEST.MAC MACROS
```

Instead of "TEST.MAC", you can use any filename you wish.

This is the end of session three.

```
*EXIT
```

In session four you will learn how to include a file with macro definitions, and run the test created in session three. To begin, invoke the emulator as before:

```
>:F1:ICE51
ISIS-II ICE-51 Vn.n
FOR COMMAND ENTRY ASSISTANCE, TYPE HELP
*
*LIST :F1:NOV24A.LOG                    ;If desired to record the session.
*
```

First we read in the macro definitions from the file TEST.MAC that we created in session three.

```
*
*INCLUDE :F1:TEST.MAC                   ;This is all you type.
*DEFINE :INIT                           ;The system reads in
.*RESET CHIP                            ;the macro definitions.
.*RESET ICE
.*REMOVE SYMBOLS
.*LOAD :F1:DEMO.HEX
.*DBYTE 0 TO 7FH = 0
.*WRITE 'ALL SET & READY TO GO'
.*EM
*DEFINE :FIND
.*OLDEST
.*TRACE = FRAMES
.*COUNT BUFFERSIZE
.*MOVE 1
.*IF FRAME %0
.*PRINT 1
.*ENDIF
.*UNTIL FRAME %0
.*ENDCOUNT
.*TRACE = INSTRUCTIONS
.*EM
*DEFINE :TEST
.*:INIT
.*WRITE ' '
.*TR = AFTER 0
.*GO FROM 0 TILL .ENDLOOP
.*WRITE ' '
.*:FIND P1=41H
.*WRITE ' '
.*:FIND P1=42H
.*WRITE ' '
.*WRITE 'END OF TEST'
.*WRITE ' '
.*EM
*
*
```

Now we ready to run the test. Type:

```
*:TEST
ALL SET & READY TO GO

EMULATION BEGUN
EMULATION TERMINATED, PC=.LOOP

FRAME   TYPE   ADDR   DATA   INSTRUCTION     P1      P2    P0    TOVF
0188:   OPC           08H    (INC R0)        41H     03H   FFH   0

FRAME   TYPE   ADDR   DATA   INSTRUCTION     P1      P2    P0    TOVF
0272:   OPC           08H    (INC R0)        42H     04H   FFH   0

END OF TEST
*
```

With the "successful" run of our automated test, we conclude session four. You may calculate the character hold times for the tests, if you wish.

```
*EXIT
```

We hope you have enjoyed getting started with the ICE-51 emulator.

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Parkway
Hillsboro, Oregon 97123

M.C.S.O. Technical Publications

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in U.S.A.