ICE-85 IN-CIRCUIT EMULATOR OPERATING INSTRUCTIONS FOR ISIS-II USERS

Manual Order Number 9800463B

B43/0578/20K CP

PREFACE

This document describes the purpose and use of the ICE-85 In-Circuit Emulator for the Intel 8085 microprocessor.

The ICE-85 module is an optional addition to the Intellec Microcomputer Development System. The ICE-85 module aids in testing and modification of the hardware and software for new products designed around the 8085 microprocessor.

Chapter 1 describes the mission of ICE-85 as a development aid for system designs based on Intel's MCS-85 microprocessor family.

Chapter 2 gives step-by step instructions for installing the ICE-85 hardware in the Intellec chassis and connecting ICE-85 to the user prototype system.

Chapter 3 presents a hands-on debugging session with ICE-85.

Chapter 4 describes the meta-notation used to abbreviate the syntax of ICE-85 commands in this manual.

Chapter 5 contains discussions and reference summaries of all of the ICE-85 commands, grouped by function.

Appendix A is a list of all ICE-85 keywords (literals), and their abbreviations, in alphabetical order.

Appendix B is a list of ICE-85 error messages, with interpretations and recommended operator responses.

Appendix C is a list of 8080/8085 assembler instructions in order by opcode, for user reference.

The minimum configuration to run the ICE-85 module is an Intellec system with 32K of RAM, room for two boards, the ICE-85 hardware and software, a console input device, and a single diskette drive. A serial printer can be added for hard-copy output.

To use this manual effectively, you need to understand the 8085 architecture and the technique of programming and debugging. For background information on these subjects, refer to the following Intel publications.

8085 Microcomputer System User's Manual	9800366
8080/8085 Assembly Language Programming Manual	9800301
PL/M-80 Programming Manual	9800268
Intellec MDS Operator's Manual	9800129
Intellec MDS Hardware Reference Manual	9800132
Intellec Series II Installation and Service Manual	9800559
MDS-DOS Diskette Operating System Operator's Manual	9800206
ISIS-II System User's Guide	9800306
A Guide To Intellec Microcomputer Development Systems	

CONTENTS

PAGE

CHAPTER 1
INTRODUCTION TO THE ICE-85
The Evolution of Microprocessor Design Aids 1-1
ICE-85 Components 1-4
Generalized Development Cycle with ICE-85 1-6
A Generalized Emulation Session 1-8

CHAPTER 2
ICE-85 INSTALLATION PROCEDURES
ICE-85 Components 2-1
Required and Optional Hardware 2-2
Hardware Installation Procedures 2-2
Installation Procedure for Intellec Series I 2-2
Installation Procedure for Intellec Series II 2-3
Installing External Signal Cables 2-5
System Grounding 2-5
Ideal Grounding Arrangement 2-6
An Observation About Other Grounding Techniques 2-6

CHAPTER 3
A SAMPLE ICE-85 SESSION AT THE TERMINAL
How to Use This Chapter 3-1
Commands Used in the Examples 3-2
Memory and I/O Port Mapping Commands 3-2
LOAD Command 3-3
Symbolic References 3-3
Emulation Control Commands 3-4
Commands That Display or Change Memory and Register Contents 3-4
Trace Display Commands 3-5
Analysis of the Sample Program 3-6
A Debugging Session Using ICE-85 3-7

CHAPTER 4
ICE-85 METALANGUAGE
Introduction 4-1
Character Set 4-3
Tokens 4-3
Keywords 4-3
Reference Keywords 4-3
Command Keywords 4-6
Function Keywords 4-6
User-Names 4-7
Symbols 4-7
Module-Names 4-8
Statement-Numbers 4-9
User Group-Names 4-9
Constants 4-10
Numeric Constants 4-10
Masked Constants 4-10

PAGE

Special Tokens 4-11
Operators 4-11
Punctuation 4-11
Numeric Expressions 4-11
Operators 4-11
Operands 4-12
Expressions 4-12
Meta-Notation Used in the Manual 4-13

CHAPTER 5
THE ICE-85 COMMAND LANGUAGE
Entering Commands at the Console 5-6
Utility Commands Involving ISIS-II 5-7
Discussion 5-7
ICE-85 Command 5-9
EXIT Command 5-9
LOAD Command 5-10
SAVE Command 5-11
LIST Command 5-11
Number Bases and Radix Commands 5-12
Discussion 5-12
Console Input Radixes; SUFFIX Command 5-12
Console Output Radixes; BASE Command 5-13
The EVALUATE Command 5-13
Radixes Used in Trace Displays 5-14
Radixes Used for Displaying Breakpoint and Qualifier Settings 5-14
Set or Display Console Input Radix Commands .. 5-15
Set or Display Console Output Radix Commands . 5-15
EVALUATE Command 5-16
Memory and I/O Port Mapping Commands 5-16
Discussion 5-16
Mapping Memory 5-16
Mapping Input/Output Ports 5-24
MAP Mode Command 5-27
MAP Memory Command 5-28
MAP I/O Ports Command 5-29
Display MAP Status Command 5-30
RESET MAP Command 5-30
Hardware Register Commands 5-31
Discussion 5-31
Display Processor and Status Register Commands 5-36
Set Processor Register Command 5-37
RESET Hardware Commands 5-37
ENABLE/DISABLE TIMEOUT Commands 5-38
Memory and Port Content Commands 5-39
Discussion 5-39
Memory Content References 5-39
Setting Memory Contents 5-42
Port Content References 5-45
Display Memory and Port Contents Commands .. 5-46

CONTENTS (continued)

PAGE

CHAPTER 5
THE ICE-85 COMMAND LANGUAGE (continued)

Set Memory Contents Command 5-47
Set Input/Output Port Contents Command 5-48
Symbol Table and Statement Number Table Commands 5-49
Discussion 5-49
Display Symbol Table and Statement Number Table Commands 5-53
DEFINE Symbol Command 5-53
Change Symbol Command 5-54
REMOVE Symbol Command 5-54
Channel Group Commands 5-55
Discussion 5-55
DEFINE GROUP Command 5-59
Display GROUP Command 5-60
Change GROUP Command 5-60
REMOVE GROUP Command 5-61
Real-Time Emulation Control Commands 5-62
Discussion 5-62
Setting Breakpoint Registers 5-64
Setting the GO-Register 5-68
Emulation Timer 5-70
GO Command 5-71
Set GO-Register (GR) Command 5-72
Display Real-Time Emulation Registers Commands 5-73
Set Breakpoint Register Command 5-73
RESET Breakpoint Register Command 5-74
ENABLE/DISABLE SY0 OUT Command 5-74
Trace Control Commands 5-75

PAGE

Discussion 5-75
Trace Display Mode 5-76
Moving the Buffer Pointer 5-76
Displaying Trace Data 5-76
Trace Control Factors 5-79
Set TRACE Display Mode Command 5-84
MOVE, OLDEST, and NEWEST Commands ... 5-85
PRINT Command 5-86
Set Qualifier Register Command 5-87
RESET Qualifier Register Command 5-87
Display Trace Controls Commands 5-88
ENABLE/DISABLE Trace Factors Commands .. 5-88
Single-Step Emulation Control Commands 5-89
Discussion 5-89
Set Condition Register Command 5-96
SR Command (Set Step-Register) 5-97
STEP Command 5-98
Display Step-Register Commands 5-99
ENABLE/DISABLE DUMP Command 5-100
External Call Commands 5-101
Discussion 5-101
External Call Commands 5-103

APPENDIX A
ICE-85 KEYWORDS AND THEIR ABBREVIATIONS

APPENDIX B
ICE-85 ERROR CONDITIONS AND RECOVERY

APPENDIX C
8080/8085 CPU INSTRUCTIONS

TABLES

TABLE	TITLE	PAGE
3-1	Memory and I/O Port Mapping Examples	3-2
3-2	Symbol and Statement-Number References in Command Examples	3-3
3-3	Emulation Control Command Examples	3-4
3-4	Memory and Register Contents Command Examples	3-5
3-5	Trace Display Command Examples	3-6
5-1	ICE-85 Commands: Alphabetical Summary	5-3
5-2	Memory Blocks for Mapping	5-19
5-3	I/O Port Segments for Mapping	5-25
5-4	8085 8-Bit Registers	5-32
5-5	8085 16-Bit Registers	5-32
5-6	8085 1-Bit Status Flags	5-32
5-7	8085 Interrupt Mask Bits	5-32
5-8	ICE-85 Status Registers	5-34
5-9	External Synchronization Lines	5-35
5-10	Other External Signals	5-35
5-11	Symbolic References and Statement References	5-51
5-12	User Probe Channels, 8085 Processor Channels, and System-Defined Group Names	5-56
5-13	Mnemonic Match Conditions	5-65
5-14	Status Group Bit Settings	5-66
5-15	Halt Conditions in the GO-Register (GR)	5-69
5-16	Trace Display Headers	5-78
5-17	Mnemonic Match Conditions	5-80
5-18	Status Group Bit Settings	5-80
5-19	Halt Conditions in the Step-Register (SR)	5-92

ILLUSTRATIONS

FIGURE	TITLE	PAGE
Front	ICE-85 with SDK-85	
1-1	Logical Blocks in a Microcomputer System	1-1
1-2	8085 CPU Functional Block Diagram	1-2
1-3	Typical Microprocessor Development Cycle	1-3
1-4	ICE-85 Functional Block Diagram	1-4
1-5	Typical Development Cycle with ICE Module	1-5
1-6	MCS-85 System Bus	1-6
1-7	Design Cycle with Shared Resources	1-7
2-1	ICE-85 Cable Terminal Assembly	2-1
2-2	Control Board Jumper Pin Configuration	2-2
2-3	System Grounding for Intellec Series I	2-6
3-1	Traffic Light Controller Program Flow Chart	3-8
3-2	CARS1 Program Listing	3-9
3-3	DELAY Subroutine Listing	3-10
3-4	CARS2 Program Listing	3-24
4-1	Analysis of the GO Command	4-1
5-1	Intellec Shared Memory Locations with 64K of RAM	5-17
5-2	Intellec UNSHARED Memory	5-18
5-3	Memory Map Diagram	5-18
5-4	Mapping into User Memory	5-22
5-5	Mapping into User and Shared Intellec Memory	5-22
5-6	Mapping into User and Unshared Intellec Memory	5-23
5-7	IO Port Segments Mapped to User and Intellec	5-26
5-8	Trace Data Displays	5-77

intellec mds
intel
mds ice-85
CONDITION 1
CONDITION 2
MATCH
ADD HIGH 8T
intel
Regulated Power Supply

463-1

CHAPTER 1
INTRODUCTION TO THE ICE-85

This manual presents the operation of the In-Circuit Emulator for the Intel 8085 microprocessor, or ICE-85. As an introduction to the use of this microprocessor design aid, this chapter contains a brief review of the evolution of microprocessor design aids, a summary of the design process using the ICE-85, and a discussion of the main features of the ICE-85 in the context of a generalized emulation session.

The Evolution Of Microprocessor Design Aids

The microprocessor has revolutionized the electronics industry and the process of electronic design. Designers formerly built up their designs by interconnecting logic gates. Now you can design using general-purpose digital electronic blocks whose functions are determined by programming. Whether you choose a microprocessor or a one-chip microcomputer, it is clear that your design task has changed immensely. Logical blocks that were formerly built by the designer from small- or medium-scale integrated circuits are now part of a single silicon package.

Your design effort now includes the task of designing software programs to drive the LSI packages.

Figure 1-1 shows the logical architecture of an LSI microcomputer system. Figure 1-2 shows the functional block diagram of the 8085 CPU; (see the *MCS-85 User's Manual* for details on the 8085 operation).

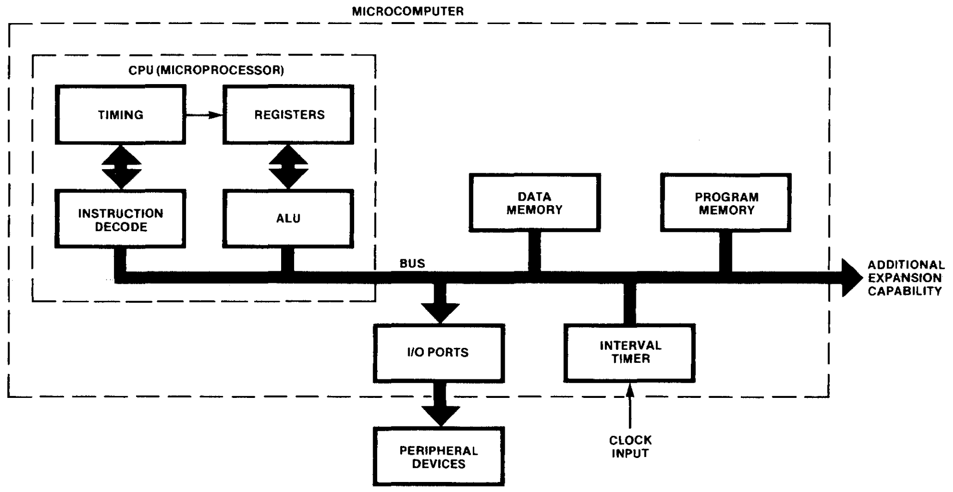


Figure 1-1. Logical Blocks in a Microcomputer System 463-2

System designs involving the integration of hardware and software long preceded the advent of the microprocessor. However, the emerging microprocessor design environment has created special problems of system design, test, and integration. For example, the number of logic signals available to you for test has been reduced from hundreds to 64 or fewer signals. Furthermore, the available design tools early in the evolution of LSI design imposed a separation of hardware and software design efforts.

Figure 1-3 is a diagram of a typical microprocessor development cycle (without ICE). In this design plan, the designers develop and verify the hardware and software components of the design separately. Late in the development cycle, the completed components are integrated for final test. This design method usually requires additional coding and hardware rework as the system pieces are eased into a final package.

System integration frequently involves special debug hardware and software, installed in the prototype system to give the designers the clearest possible view of the system interaction. The designers fit together the separate hardware and software designs and fine-tune them, working around the restrictions caused by extra boards, wires, and software. The need for separate verification of the debug components adds another level of complexity to the design task.

The Intellec Development System currently supports four in-circuit emulators, ICE-80, ICE-30, ICE-48, and ICE-85, for Intel's MCS-80, Series 3000, MCS-48 and MCS-85 microprocessor families, respectively. In-circuit emulators for other Intel microprocessor families are forthcoming. The Intellec system bus structure was designed specifically to support a multiprocessor configuration, allowing the system processor and ICE processor to share Intellec memory and I/O when desired. An ICE module is more than a microprocessor; it is a complete microcomputer system containing its own microprocessor, timing circuitry, memory, and I/O.

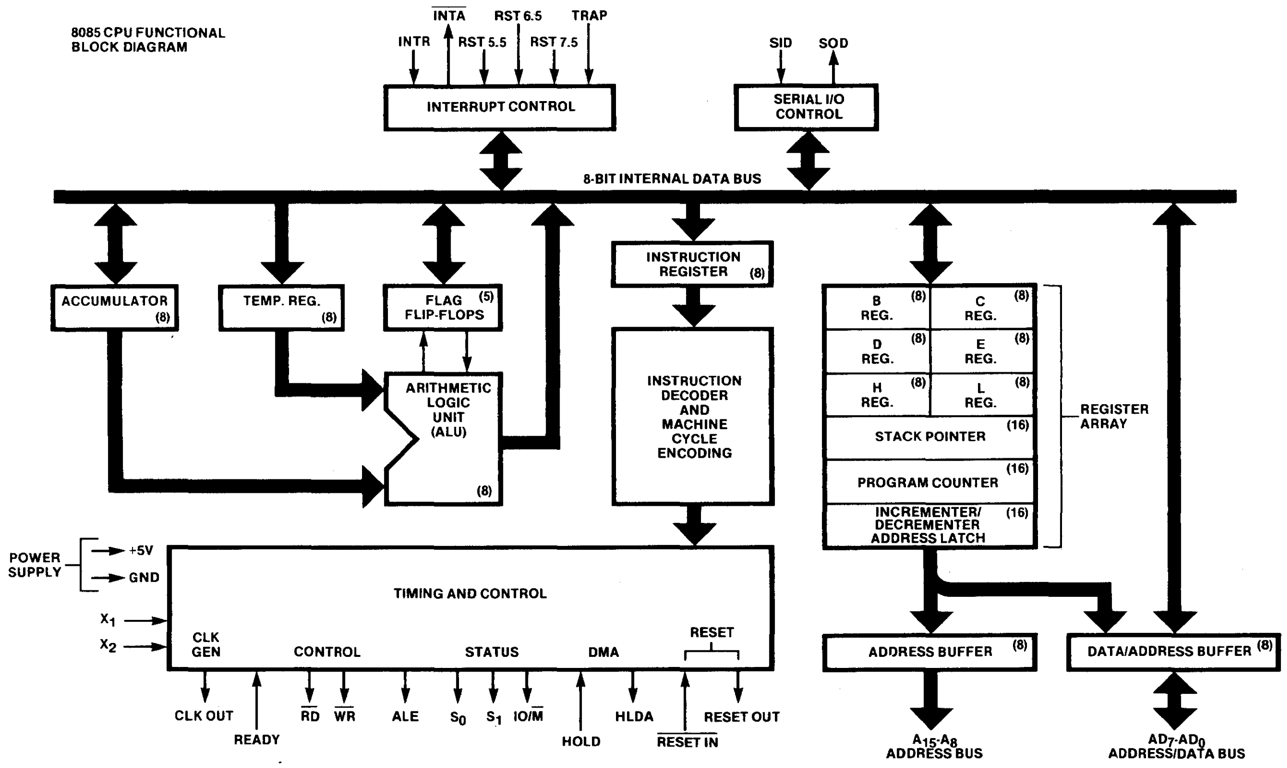


Figure 1-2. 8085 CPU Functional Block Diagram

463-3

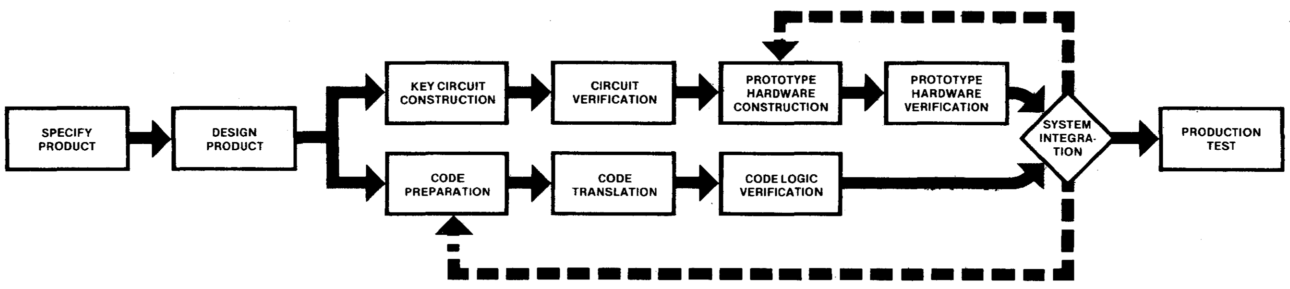


Figure 1-3. Typical Microprocessor Development Cycle 463-4

Thus, the introduction of the microprocessor soon led to the realization that the designer of microcomputer-based products required specialized design tools. Conventional design aids such as oscilloscopes and multimeters do not provide debug capabilities such as displaying and altering CPU registers and memory contents, simulating metal-masked ROM memory with easily-alterable RAM memory, or stopping program execution to display and alter systems status.

Early generations of microcomputer design aids included simulators that approximate microprocessor logic to aid in software debugging. Prototype boards made it easier for the hardware designer to check out key circuitry before building the actual system breadboards. Program assemblers and debugging systems were designed to run on the microprocessor itself, reducing the need for access to large computers for software development.

The Intellec Microcomputer Development Systems for Intel's 4040 and 8080 microprocessors introduced a powerful single package for designers. The hardware designer could construct key hardware (for example, memory, memory decode, and I/O circuitry) on prototype boards that could then be plugged directly into the development system backplane. This capability represented a great step forward in system design. The engineer could now evaluate new system concepts using development system resources.

An in-circuit emulator enhances the power of the Intellec system by inserting special development logic between the prototype system and its microprocessor, and by extending the range of controls that can be entered through the system console.

Only four years separate the introduction of the first microcomputer design aid, Intel's SIM-4-01, and the introduction of the Intellec MDS.

Seen in terms of later developments, the Intellec system alone is limited in the scope of its operations. Circuits to be tested must be compatible with the development system's bus architecture. The system debug support does not extend into the prototype system. Software and hardware development may be isolated from one another during most of the design effort.

The Intellec system with an in-circuit emulator option contains at least two microprocessors. The Intellec processor supervises system resources, executes system monitor commands, and drives system peripherals. A second processor, the ICE processor, interfaces directly to the designer's prototype or production system via an external cable. The ICE cable is terminated in a plug that fits the microprocessor socket in the user system. When you plug the ICE cable into your system's microprocessor socket, the development system debug aids are extended directly into your system.

ICE-85 Components

ICE-85 hardware consists of two printed circuit boards, the Control board and the Trace board, and two cable assemblies, the ICE-85 module and the External Trace module. The two PC boards are inserted in adjacent slots in the Intellec chassis, and connect to each other through an auxiliary connector furnished in the ICE-85 package.

The ICE-85 module, with its three ribbon cables, connects the Control and Trace boards and the microprocessor socket in the user system. The module, an enclosed PC card, contains buffering and timing logic, and also contains a set of connector pins for access to signals used to coordinate the ICE-85 operation with external events.

The External Trace module, an enclosed PC card on a single ribbon cable, presents a set of 18 hardware channel probes that you can connect to the signals on your system that you desire to monitor. The card contains buffers for these 18 probes. The 18 channels from the buffers are available to ICE-85 for the control of trace and emulation.

A block diagram of the ICE-85 hardware is shown in Figure 1-4. ICE-85 hardware is designed to emulate TTL inputs, and does not emulate MOS inputs.

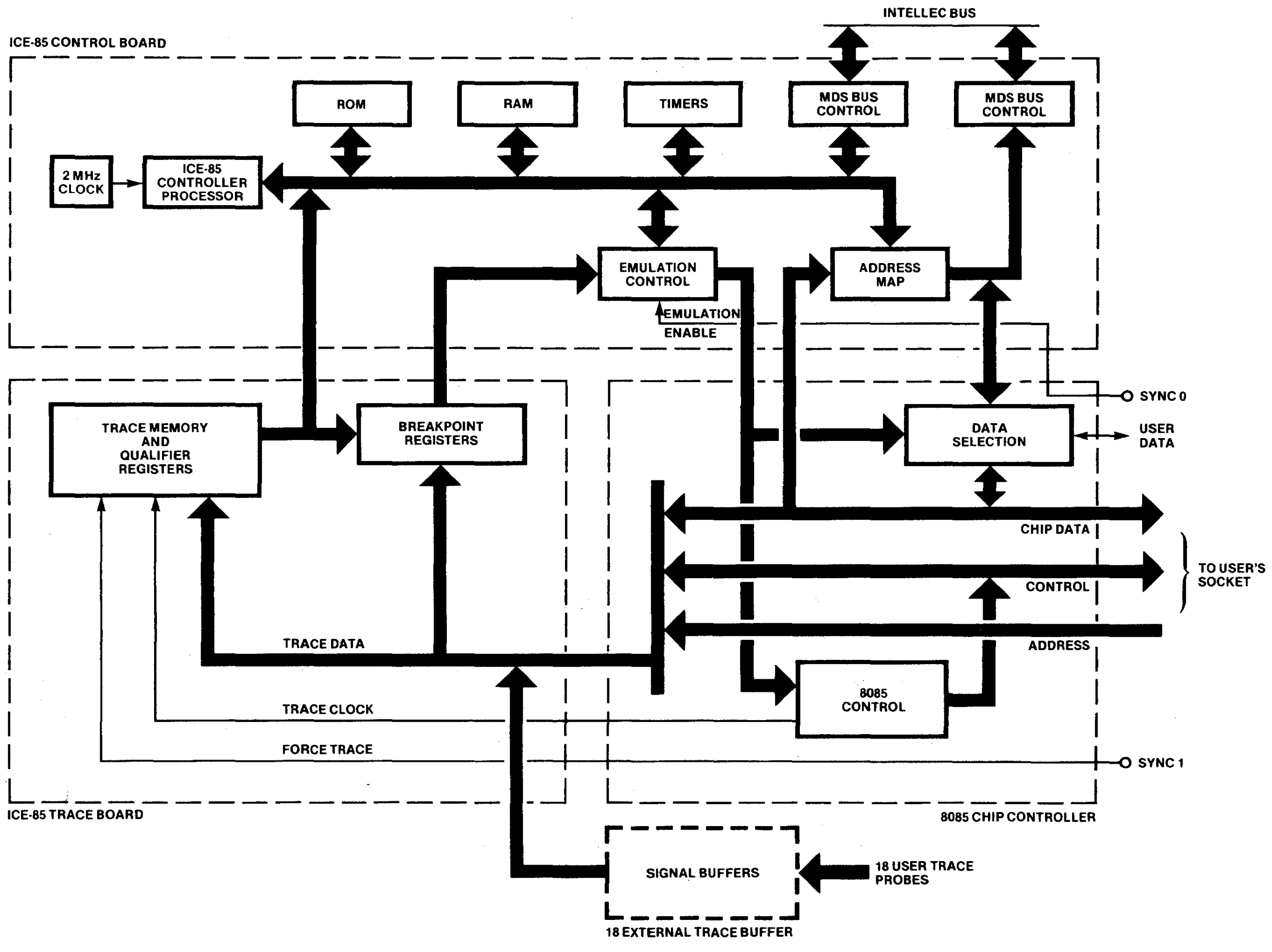


Figure 1-4. ICE-85 Functional Block Diagram 463-5

The ICE-85 program resides in Intellec memory along with the Intellec Monitor and diskette operating system software. This program is written in PL/M, Intel's high-level language, and runs under the Intel Systems Implementation Supervisor (ISIS-II). The ICE-85 program recognizes and translates your commands from the console, and places the encoded results in a control block for the hardware to read. It also retrieves the hardware-modified contents of control blocks and translates the contents into formats that you can easily understand.

However, the internal communication between ICE-85 and the Intellec system is transparent to the user. With ICE-85 installed and running, you enter ICE-85 commands interactively at the Intellec system console. Each command is executed immediately as it is entered. The commands provide a wide range of controls over all aspects of the operation of your system.

ICE-85 also performs extensive self-diagnosis during on-line operation. It checks the Intellec bus interface during initialization, checks for the absence of system clocks, verifies loading of ICE-85 registers, and verifies data written to user memory (data verification can be disabled for certain items at your option). When an error is detected, ICE-85 halts emulation or command processing, and displays an error message at the console.

To complement the ICE modules, the Intellec Development System hosts all other tools necessary for microprocessor development work. The Intellec text editor is standard system software; it enables the designer to create and edit source programs quickly. The ISIS-II diskette operating system provides high speed file handling and mass storage facility. The Universal PROM Programmer peripheral for the Intellec system can be used to program any Intel PROM.

PL/M-80, Intel's high-level language for the 8080 and 8085 systems, reduces the time required for software development by allowing the programmer to write code in a self-documenting, natural manner. The PL/M-80 compiler handles time- consuming tasks like managing register usage, allocating data memory, and optimizing code.

Using the relocating assembler, linker, and loader, programs can take advantage of PL/M-80 for general programming needs, and also incorporate assembly-language portions of code for critical timing loops or I/O procedures.

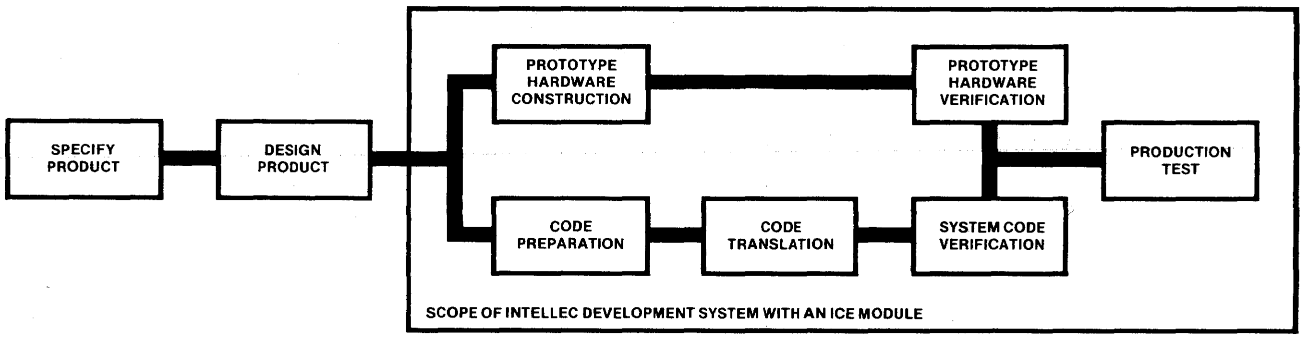

SPECIFY PRODUCT
DESIGN PRODUCT
PROTOTYPE HARDWARE CONSTRUCTION
PROTOTYPE HARDWARE VERIFICATION
CODE PREPARATION
CODE TRANSLATION
SYSTEM CODE VERIFICATION
PRODUCTION TEST
SCOPE OF INTELLEC DEVELOPMENT SYSTEM WITH AN ICE MODULE

Figure 1-5. Typical Development Cycle with ICE Module 463-6

Generalized Development Cycle With ICE-85

Figure 1-5 diagrams a generalized product development cycle using ICE-85 as a design aid. The sequence of events in developing a new product using the Intellec with ICE-85 is approximately as follows.

- Complete the specifications for the prototype hardware design, software control logic, and integrated system performance.
- Organize both the hardware and software designs into logical blocks that are readily understandable, have well-defined inputs and outputs, and are easy to test. Breaking down the design is an iterative process, but is extremely valuable in reducing the time required for prototyping, programming, testing, and modification.
- Program the software modules in PL/M-80 or in 8080/8085 assembly language, naming and storing the programmed modules as files under ISIS-II. Compile or assemble the modules, linking and loading the combinations you are ready to test, creating an object-code (machine language) version. Desk-check each module as it is completed.
- As software modules are ready for testing, load them into Intellec RAM, and emulate them via the ICE processor. If no hardware prototype is available, use the ICE-85 Adapter Socket to configure the ICE-85 for 'software' mode. The ICE-85 system allows you to use Intellec system memory and I/O ports as parts of the 'prototype' system. The advantages of this feature to software development include:

    1. You do not have to be concerned about overflowing your prototype system memory in the initial stages of software design. You have the freedom to test the program and compact it later without having to make room for extra memory in your prototype.
    2. You may test your program in RAM memory, and make patches quickly and easily without having to erase and reprogram PROM memory. In later test phases, the ICE module can control program execution from PROM or ROM in your prototype. The ICE module can map RAM memory in the Intellec to replace prototype memory in set increments, to test out software changes before reprogramming.

- Hardware prototyping can begin with just a microprocessor socket and a system bus (Figure 1-6). You can use I/O ports in the Intellec system to simulate inputs and outputs that later will come from and go to the 'real world', the environment in which your system is to 'live'. The Intellec system ports can also give your system access to external diagnostic routines.

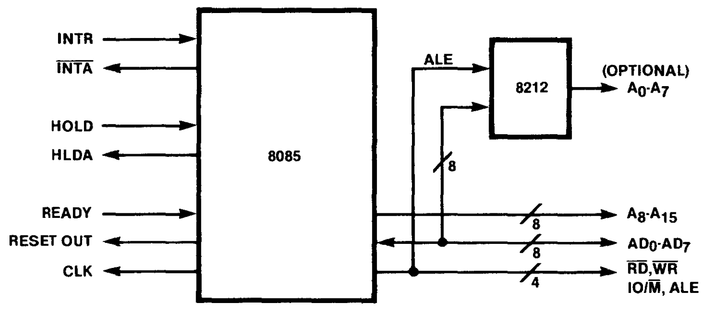


Figure 1-6. MCS-85 System Bus 463-7

- You can use memory in the Intellec system to check the interaction of prototype hardware and proven software. The ability to map memory and I/O is helpful in isolating system problems. You can exercise all prototype memory and I/O from a program residing initially in Intellec memory, and reassign memory block-by-block to the user system as code is verified. Hardware failures can then be isolated quickly, because interactions between prototype parts occur only at your command. You do not have to use the prototype to debug itself.

- When the hardware prototype is developed to include additional peripheral chips, access to 'real-world' signals, and perhaps some PROM memory and system I/O, you can attach the 18 ICE-85 hardware probes so as to monitor the coordination of these additional elements.

- The debugging/testing process can proceed through each hardware and software module, using ICE commands to control execution and to check that each module gets data or control information from the correct locations, and places correct data or other signals in the proper cells or output locations for subsequent modules to use.

- Eventually, you test all hardware and software together. The program can reside in RAM or PROM in your system, or in RAM in the Intellec. All other hardware can be in the prototype. ICE-85, connected to the system through the microprocessor socket and hardware probes, can emulate, test, and trace all the operations of the system.

- After the prototype has been completely tested, the ICE-85 can be used to verify the product in production test. The test procedures you developed for the final prototype testing can serve as the basis for production test routines, running the program from metal-masked ROM in the production system.

Figure 1-7, another diagram of the development cycle using ICE, shows the role of shared system resources in prototype development.

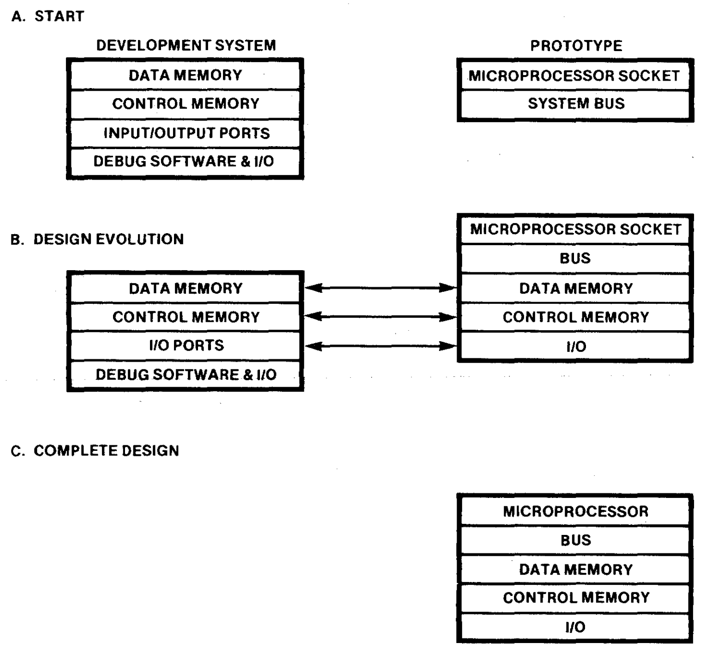


Figure 1-7. Design Cycle with Shared Resources 463-8

A Generalized Emulation Session

This section describes the main steps in an emulation session. You may not always perform all the procedures given here in every emulation session, but the main outline is the same in all sessions. The discussion emphasizes some of the features of ICE-85 that have not been presented earlier. For the details of the command language, see Chapters 4 and 5.

1. Install the ICE-85 hardware in the Intellec chassis (see Chapter 2).
2. Attach channel probes and external signal lines as desired.
3. Boot the system, and obtain the hyphen prompt from the ISIS-II system. Enter the ICE85 command, and obtain the asterisk prompt from ICE-85.
4. From the software to be tested, determine how many memory addresses in the Intellec system are required to perform the emulation. For example, if your program presently uses about 3K of memory but your prototype has only 1K installed, you need about 2K of Intellec memory to be devoted to your program.

   Intellec memory that is available for user program mapping is organized into blocks of 2K (2048) contiguous bytes. Thirty-two such blocks are logically available; the amount that is physically available depends on what you have installed in the Intellec.

   If your Intellec has only its main memory, the memory blocks mapped to the user program in the Intellec must share the memory space with the ISIS-II and ICE-85 system software, and with the Intellec Monitor. The memory locations for these programs are constant, and you can easily determine where free blocks of mappable memory are to be found.

   The systems software in shared memory occupies the lower 28K of Intellec shared memory, and Monitor uses addresses F800H to FFFFH. Typically, your program also occupies physical locations in low memory. Since you cannot overwrite the systems software, you must map the memory space used by your program into locations higher in Intellec memory; ICE-85 stores the displacement in its memory map, and refers each memory reference in your program to the proper physical location in Intellec memory. For example, suppose your code would occupy locations 0000H to 0FFFH (the 'H' means hexadecimal radix), or 4096 contiguous locations beginning at location 0; the lowest address in Intellec shared memory that represents the beginning of two free 2K memory blocks is always 7000H. Thus, the mapping command would be:

   ```
   MAP MEMORY 0000H TO 0FFFH = INTELLEC 7000H
   ```

   ICE-85 takes care of the rest. You (and your program) continue to use the memory references in the range 0000H to 0FFFH, and ICE-85 makes the correction transparently to you.

   You can also make use of optional unshared memory, up to 64K of extra memory, if you have the slots available in your Intellec chassis and the extra memory boards. Just as with the shared memory space, the Intellec Monitor software uses addresses F800H to FFFFH. The rest is free to be mapped; displacement is allowed, but is generally not required for unshared memory mapping.

5. From your program and the state of development of the prototype, determine how many Input/Output ports you would like to borrow from the Intellec system. I/O ports are available in segments of eight ports each; up to 32 such segments are logically available. Suppose you need to borrow twelve segments of eight ports each; the mapping command might be:

   ```
   MAP IO 00T TO 95T = INTELLEC
   ```

   (The 'T' means decimal radix.) The command makes available the lowest 96 ports. No displacement is allowed.

6. Load your program from diskette into the memory locations you have mapped, using the LOAD command.
7. ICE-85 has three modes of operation: interrogation, real-time emulation, and single-step emulation. The asterisk prompt signals that ICE-85 is in interrogation mode, ready to accept any command.
8. In interrogation mode, prepare the system for emulation by defining symbols and channel groups, and setting emulation breakpoints and trace qualifiers.

   ICE-85 software provides keywords for all 8085 registers and flags. In addition, you may define and use symbols to refer to memory locations and contents. The user symbol table is generated along with the object file during a PL/M compilation or assembly. This table can be loaded into Intellec shared memory when the user program is loaded.

   You are encouraged to add to this symbol table any additional symbolic values for memory addresses, constants, or variables that you may find useful during system debugging. Symbols may be substituted for numeric values in any of the ICE-85 commands.

   Symbolic reference is a great advantage to the designer. You do not need to recall or look up the addresses of key locations in your program, as they change with each assembly; you can use meaningful symbols from your source program instead. This facility is especially valuable for high-level language debugging. You can completely debug a program written in PL/M by referencing symbols defined in the source code. You do not need to become involved with the machine level code generated by the compiler. For example, the ICE-85 command:

   ```
   GO FROM .START TILL .RSLT WRITTEN
   ```

   begins real-time emulation of the program at the address referenced by the label START in the designer's PL/M-80 program. The command also specifies that the program is to break emulation when the microprocessor writes to the memory location referenced by RSLT. You do not have to be concerned with the physical locations of START and RSLT. The ICE-85 software supplies them automatiacally from information stored in the symbol table.

   The ICE-85 system provides twelve channel groups, representing logical clusters of the 18 hardware channel probes from the External Trace Module and the address, data, and control signals from the 8085 system bus (via the Interface cable). There are 43 channels in all. In addition to the system-defined groups, you can define any other groups you wish to use. The channel groups represent clusters of bit settings that can be used as match values for controlling emulation and trace data collection. In addition, you can specify the display format for any group you define, to make it easier for you to interpret the status of the group when it is displayed during trace display.

   Emulation breakpoints and trace qualifiers are identical in structure and complementary in function. Both are ICE-85 'pseudo-registers' containing 42 bits; the bits correspond to the lowest 42 of the 43 channels discussed above. You can set the bits in a breakpoint or qualifier register to zero, one, or don't-care. Then, whenever the states of the 42 channels match the settings of the corresponding bits in the register (not counting the don't-cares), ICE-85 recognizes the match and takes action. If a breakpoint register matches, emulation halts (the breakpoint register must be enabled); if a trace qualifier matches, trace data collection is enabled and runs whenever emulation runs.
9. Enter a GO command to begin real-time emulation. ICE-85 uses another pseudo-register called the GO-register to contain the halting conditions that you have specified, either in the GO command itself or previously.
10. When emulation halts, you display the trace data collected during that emulation. ICE-85 loads trace data into a trace buffer. Using ICE-85 commands, you can position the trace buffer pointer to the information you desire

to review, and display one, several, or all the entries in the buffer. You can set the display mode to one frame per line, one machine cycle per line, or one instruction per line of the display.

11. To control emulation more precisely and obtain more detailed trace data than with real-time emulation, you can command ICE-85 to begin single-step emulation. Under single-step emulation, you tell ICE-85 how many steps to emulate, and you can specify other conditions for halting. After each step emulated, you can obtain an automatic display of the current entry in the trace buffer, and the current settings of several 8085 registers.

12. When a halt is reached, you can examine and change memory locations, machine states, 8085 registers and flags, and I/O ports, to provide you with valuable information on program operation at the point of termination. You may alter data or register values to examine their effect on the next emulation, or you can patch in changes to your program code itself. You can display and change symbolic values in the symbol table, user-defined group names, and breakpoint and trace qualifier match values.

13. Alternate between interrogation and emulation until you have checked everything you want to check.

14. At the end of the emulation session, you can save your debugged code on an ISIS-II diskette file, using the ICE-85 SAVE command. The operation can be specified to save program code, symbol tables, and (for PL/M programs) the source code line number table.

    You can start another session immediately, resetting all parameters to their initial values with a few simple commands, or you can exit to ISIS-II to terminate the session.

This introduction is intended to show you some of the scope and power of the ICE-85 in operation, and to suggest how this integrated software/hardware design aid can fit into your development cycle. Chapter 2 contains detailed installation instructions. Chapter 3 contains a hands-on tutorial involving a sample program to be debugged. Chapter 4 describes the meta-notation used in this manual to specify command syntax and semantics. Chapter 5 presents the details of the command language in a format and sequence designed for reference.

CHAPTER 2
ICE-85 INSTALLATION PROCEDURES

This chapter contains information on the installation of the ICE-85.

ICE-85 Components

The following items are included in the ICE-85 package.

- ICE-85 Control board (PN 1001355): A circuit board that plugs into the Intellec chassis. The Control board contains the emulation controls, address map, timer, and internal clock.
- ICE-85 Trace board (PN 1001191): A circuit board that plugs into the Intellec chassis. It contains the breakpoint registers, trace memory, and qualifier registers for controlling trace data collection.
- ICE-85 Module and Interface Cable (PN 4001461): A cable assembly containing 8085 logic; the cable terminal is a 40-pin connector with an 8085 CPU chip installed (Figure 2-1). The terminal is inserted in the CPU socket of the user system.
- External Trace Module and Hardware Probe cable (PN 4001460): A cable assembly containing signal buffers for 18 probes that can be connected to any TTL signals you wish to monitor.
- ICE-85 Adapter Socket (PN 4001468): A 40-pin socket that fits the end of the Interface Cable. The socket contains a 6.144 MHz crystal and the circuits necessary to run the ICE-85 as a software development tool without user hardware. The Adapter Socket is delivered mounted on the end of the Interface Cable, to protect the pins on the connector.
- Dual Auxiliary Connector for the Intellec Series I chassis (PN 1000515), and Dual Auxiliary connector for the Intellec Series II chassis (PN 1000751): Each connector consists of a pair of parallel circuit board connectors that provide electrical interconnections between the Control and Trace boards when they are installed in the Intellec chassis.
- Two Sync Cables (PN 4001603): Two single-wire cables terminating in micro-hooks, used to connect the external synchronization lines SYNC0 and SYNC1 to external devices, including other ICEs.
- Some of the parts listed above may be contained in an Accessory Kit (PN 4001635).
- ICE-85 software, diskette-based version.

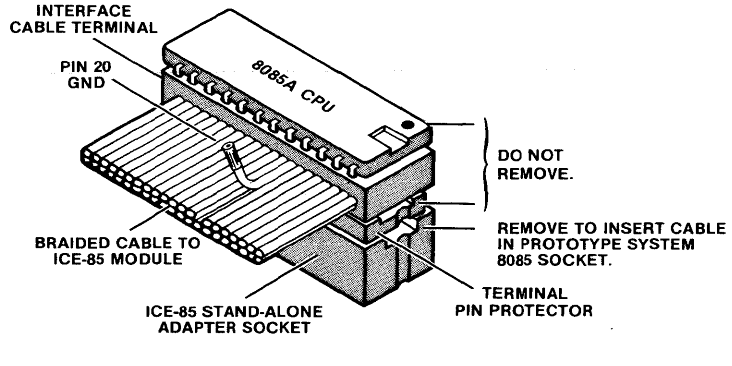

INTERFACE CABLE TERMINAL
PIN 20 GND
8085A CPU
DO NOT REMOVE.
REMOVE TO INSERT CABLE IN PROTOTYPE SYSTEM 8085 SOCKET.
BRAIDED CABLE TO ICE-85 MODULE
TERMINAL PIN PROTECTOR
ICE-85 STAND-ALONE ADAPTER SOCKET

Figure 2-1. ICE-85 Cable Terminal Assembly 463-9

Required and Optional Hardware

ICE-85 software requires the following minimum hardware configuration:

- Intellec Series I or Series II with 32K of RAM and available slots for two adjacent circuit boards.
- Teletypewriter, CRT, or equivalent for console input and output.
- Single diskette unit.
- ICE-85 hardware as described above.

The following are optional enhancements to an ICE-85 system:

- Serial printer for hard-copy output.
- One or more RAM boards (32K or greater) to provide up to 64K unshared memory in the Intellec chassis. If unshared memory is included, all RAM boards must be 32K or greater.

Hardware Installation Procedures

The installation of the ICE-85 hardware is presented in the next four sections, as follows: procedures for Intellec Series I; procedures for Intellec Series II; external signal cables; and grounding techniques.

Installation Procedure for Intellec Series I

1. Disconnect the power cords of the Intellec chassis and user system.
2. Install the Intellec peripherals (diskette drives, TTY, CRT, serial printer), following the installation guidelines given in the MDS Operator's Manual.
3. Inspect the ICE-85 components for damage.
4. Locate the group of six jumper pins at the lower center of the ICE-85 Control Board. These pins have the configuration shown in Figure 2-2.
5. Verify that the jumper is in position 0 as shown, selecting device code 10 to identify the ICE-85 module. The four positions select device codes as follows.

Position	Device Code
0	10 (standard setting)
1	11 (reserved for future ICE-85 use)
2	14 (reserved for future use)
3	15 (reserved for future use)

6. Remove the top cover of the Intellec chassis. The top cover is secured by four half-turn screws.
7. Locate the connector panel at the top rear of the Intellec chassis. Remove the four screws that attach the connector panel to the frame.
8. Lift the connector panel away from the frame. Insert the three ribbon cables from the ICE-85 Module and the ribbon cable from the External Trace Module through the slot beneath the connector panel. The corrugated sides of the ribbon cables should face upward.

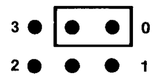


Figure 2-2. Control Board Jumper Pin Configuration

9. Replace the four screws in the connector panel. Replace any ground lugs from peripheral cables that were attached to these screws.
10. Expand Intellec memory as necessary to accept the ICE-85 and user software. Be sure to leave two adjacent card slots free for the ICE-85 boards.
11. Mount the Control Board and the Trace Board in the Dual Auxiliary connector. For ease of ribbon cable routing, the solder side of the Trace Board should face the component side of the Control Board (i.e., the Trace Board is 'in front of' the Control Board).
12. Insert the assembly of boards and connector into the Intellec chassis so that the Control Board is in an odd-numbered slot.
13. ATTACH THE RIBBON CABLE MARKED X FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE CONTROL BOARD MARKED X. When the connector is properly alighed, the triangles on connector and socket that designate pin 1 are lined up. Fold the cable as needed to fit inside the chassis with the top replaced.
14. ATTACH THE RIBBON CABLE MARKED Y FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE CONTROL BOARD MARKED Y, aligning the pin 1 markers as described in the previous step.
15. ATTACH THE RIBBON CABLE MARKED V FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE TRACE BOARD MARKED V, aligning the pin 1 markers as described above.
16. ATTACH THE RIBBON CABLE MARKED T FROM THE EXTERNAL TRACE MODULE TO THE RECEPTACLE ON THE TRACE BOARD MARKED T, aligning the pin 1 markers as described above.

WARNING

The ribbon cables must be connected exactly as described in steps 13, 14, 15, and 16 above. Any wrong connection can result in damage to the equipment.

17. Locate the 40-pin terminal at the end of the Interface Cable on the ICE-85 Module (Figure 2-1). The terminal has an 8085 mounted on the top side, and the Stand-alone Adapter Socket mounted on the lower side through a terminal pin protector. If the ICE-85 system is to be used without user prototype hardware, leave the Adapter socket mounted in the terminal.
18. If a user prototype is to be included, remove the Adapter Socket and insert the user cable into the 8085 socket on the user system. The 40-pin terminal protector guards the Terminal pins. Plug the terminal assembly in to the user system with this protector mounted. The pins on the cable terminal itself should never be plugged directly into the user system.
19. Inpect the system to ensure proper grounding. Grounding techniques are discussed later in this chapter.
20. Replace the top cover of the Intellec chassis.
21. Insert the power cords of the Intellec and user systems into their sockets. Connect both to power sources.

Installation Procedure for Intellec Series II

1. Disconnect the power cords of the Intellec chassis and user system.
2. Install the Intellec peripherals (diskette drives, TTY, serial printer), following the installation guidelines given in the Intellec Series II Installation and Maintenance Manual.
3. Inspect the ICE-85 components for damage.
4. Locate the group of six jumper pins at the lower center of the ICE-85 Control Board. These pins have the configuration shown in Figure 2-2.

5. Verify that the jumper is in position 0 as shown, selecting device code 10 to identify the ICE-85 module. The four positions select device codes as follows.

Position	Device Code
0	10 (standard setting)
1	11 (reserved for future ICE-85 use)
2	14 (reserved for future use)
3	15 (reserved for future use)

6. Remove the front cover panel of the Intellec. The panel is below the CRT screen, and is secured by two half-turn fasteners.
7. Expand the Intellec memory as required for the ICE-85 and user software. Be sure to leave two adjacent card slots free for the ICE-85 boards.
8. Mount the Control Board and the Trace Board in the Dual Auxiliary Connector, part number 1000751. The boards can be mounted in either order, but must be configured so that, when the component sides are up, the auxiliary connector is at the left.
9. Insert the assembly of boards and connector into any two adjacent slots in the EMDS chassis, component sides of the boards facing upward.
10. Locate the cable relief slot at the right front of the EMDS chassis. Bring the three ribbon cables from the ICE-85 Module and the ribbon cable from the External Trace Module through the relief slot and behind the vertical cable restraint just to the left of the slot.
11. ATTACH THE RIBBON CABLE MARKED X FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE CONTROL BOARD MARKED X. When the connector is properly aligned, the triangles on connector and socket that designate pin 1 are lined up. Fold the cable as needed to fit inside the chassis with the front cover replaced.
12. ATTACH THE RIBBON CABLE MARKED Y FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE CONTROL BOARD MARKED Y, aligning the pin 1 markers as described in the previous step.
13. ATTACH THE RIBBON CABLE MARKED V FROM THE ICE-85 MODULE TO THE CABLE RECEPTACLE ON THE TRACE BOARD MARKED V, aligning the pin 1 markers as described above.
14. ATTACH THE RIBBON CABLE MARKED T FROM THE EXTERNAL TRACE MODULE TO THE RECEPTACLE ON THE TRACE BOARD MARKED T, aligning the pin 1 markers as described above.

WARNING

The ribbon cables must be connected exactly as described in steps 11, 12, 13, and 14 above. Any wrong connection can result in damage to the equipment.

15. Locate the 40-pin terminal at the end of the Interface Cable on the ICE-85 Module (Figure 2-1). The terminal has an 8085A mounted on the top side, and the Stand-alone Adapter Socket mounted on the lower side through a terminal pin protector. If the ICE-85 system is to be used without user prototype hardware, leave the adapter socket mounted in the terminal.
16. If a user prototype is to be connected, remove the adapter socket and insert the cable terminal into the 8085 socket on the user system. The 40-pin terminal protector guards the terminal pins from damage. Plug the terminal assembly into the user system with the protector mounted. The pins on the cable terminal itself should never be plugged directly into the user system.
17. Inspect the system to ensure proper grounding. Grounding techniques are discussed later in this chapter.

18. Replace the front cover panel on the Intellec chassis.
19. Insert the power cords of the Intellec and user systems into their sockets. Connect both to power sources.

Installing External Signal Cables

The ICE-85 Module presents connectors for several external signals through a slot in the side of the module case. The signals are as follows.

SYNC 0	Enabled as input by ICE-85 command: halts emulation when set low. Enabled as output by ICE-85 command: goes low when emulation is not running.
SYNC 1	Enabled as input by ICE-85 command: forces trace data collection when set high. Enabled as output by ICE-85 command: goes low when trace is not running.
MATCH 0/ MATCH 1/	Both signals are always enabled. MATCH 0/ goes low when a condition in register BR0 occurs; MATCH 1/ goes low to signal a match on breakpoint register BR1.
EMUL	Always enabled. Goes high when emulation is running.
GND	Common ground for external signals.

Refer to Chapter 5 for the use of the external signals.

The installation procedure is the same for all of these signals. The cable consists of two wires terminated in micro-hooks. Connect one of the wires to the desired external signal post, through the slot in the side of the case. Connect the other wire of the pair to the GND post. The GND post can act as a common reference for several external signals.

System Grounding

On the rear of the Intellec Series I chassis there are two connectors that bring out Electronic Ground and Chassis Ground. These connectors are the only place in the Intellec Series I where these two grounds are tied together.

The AC frame ground is tied to the Intellec chassis and is the third wire on the AC power line (the other two wires being "hot" and "common").

Electronic Ground is routed through the Intellec onto the User Cable, where it is passed down the cable, through the Cable Card, and presented to the user system hardware on pin 20 of the 8085 40-pin socket.

On the Intellec Series II chassis, the chassis ground and electronic ground are tied together at the frame of the power supply. The location is internal, rather than on the rear of the chassis as on the Series I. The electronic ground wire is lime green with a yellow stripe. The exact location of the interconnection varies among the different models of the Series II.

Ideal Grounding Arrangement

The ideal grounding arrangement between the Intellec and the user system occurs if the user system's Electronic Ground and Chassis Ground are left unconnected (i.e., open). Thus, the user Chassis Ground and frame should be tied to the third wire of his AC power line, but his Electronic Ground should not be tied to it. This arrangement allows just one common electronic ground to the total user system. Figure 2-3 illustrates this ideal grounding arrangement, on the MDS chassis.

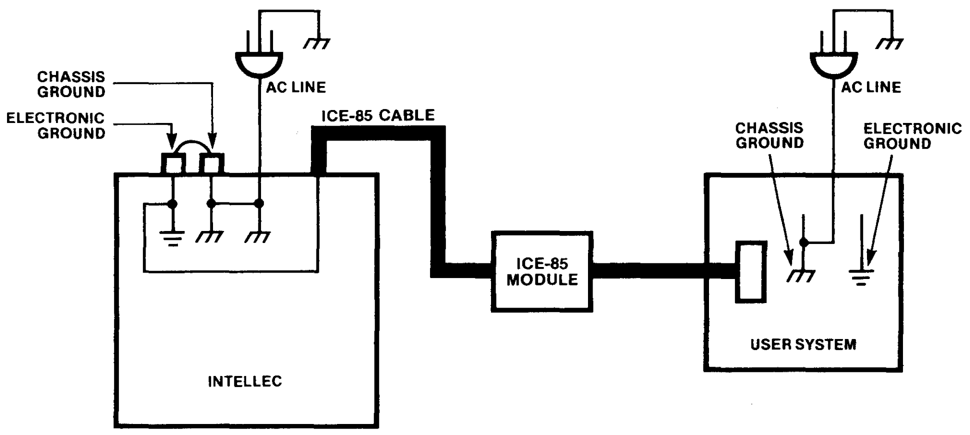


Figure 2-3. System Grounding for Intellec Series I 463-11

The above configuration is recommended in order to avoid a ground loop. Such a ground loop would cause DC current to flow through the user cable, producing a voltage drop between the Intellec System and the user system and introducing unnecessary noise levels.

An Observation About Other Grounding Techniques

It may not always be feasible for the user to break his Electronic Ground-Chassis Ground connection. In this situation, it is possible to avoid a ground loop by breaking the Intellec Electronic/Chassis Ground connection. This solution may prove unsatisfactory, however, because if other peripheral devices tied to the Intellec have their electronic and chassis grounds connected, you simply get a longer ground loop!

Although you can open all of the connections in the Intellec peripherals, with a large system this may be arduous.

If the Intellec AC line and the user system AC line plug into the same output, the ground loop problem may be of minor concern. If they plug into different AC supplies, though, serious problems may develop.

One possible way to solve a suspected long ground loop problem could be to connect the Intellec Electronic Ground and the user Electronic Ground by way of an external electronic cable. A braided ground strap should be used for this purpose.

Additionally, since pin 20 at the 8085 socket provides the only ground tie between the user system and the Intellec, you should ensure that a good ground exists at this point. In fact, Intel supplies a small grounding cable (PN 400514) that can be used to guarantee this ground. One end of this cable should be clipped to the user's Electronic Ground. The other end should be connected to the User Cable's 40-pin plug at the pin connector provided.

CHAPTER 3 A SAMPLE ICE-85 SESSION AT THE TERMINAL

The purposes of Chapter 3 are to introduce a few common ICE-85 commands, and to provide hands-on experience with ICE-85 in a sample software debugging session. To reduce the need for cross-reference to other chapters, this chapter includes brief discussions of the commands used in the examples. The program to be debugged is a simple traffic light controller. The program logic is presented before the hands-on session, to help you understand what is going on.

How To Use This Chapter

- To use this program as a hands-on tutorial, you must enter the programs as listed in Figures 3-2, 3-3, and 3-4, using the ISIS-II Text Editor. Omit the line number and nesting information that is on the listing; these values are assigned by the compiler.
- Compile CARS1 and CARS2 separately with the PLM80 compiler program, making two separate diskette files.
- Assemble the routine DELAY using ASM80.
- Link CARS1 and CARS2 separately to DELAY and to system libraries containing routines used by the programs. The link step should look like the following:

```
LINK :Fx:CARSy.OBJ,:Fx:DELAY.OBJ,SYSTEM.LIB,PLM80.LIB
    TO :Fx:CARSy.LNK
```

- Locate the two linked programs as desired. In the examples, the default location 36C0H has been used.
- Complete directions on how to perform these preliminary steps are contained in Dan McCracken's *A Guide to Intellec Microcomputer Development Systems*; this document also contains further hands-on examples using ICE-85.
- For details on program preparation, refer to the following manuals.

Text Editor:	*ISIS-II System User's Guide.*
PL/M-80 Compiler:	*PL/M-80 Programming Manual.* *ISIS-II PL/M Compiler Operator's Manual.*
8080/8085 Assembly Language:	*8080/8085 Assembly Language Programming Manual.* *ISIS-II 8080/8085 Macro Assembler Operator's Manual.*

- Read over the command discussions and tables of command examples.
- Study the logic of CARS1, the program to be debugged. The material includes text discussion, a flowchart, and program listings (CARS1, DELAY).
- Install the ICE-85 hardware, following the steps given in Chapter 2. Leave the Stand-alone Adapter mounted on the ICE-85 Interface Cable Terminal, for 'software mode' operation.
- Insert an ISIS-II system diskette in drive 0, and the ICE-85 software diskette in drive 1.
- Boot the system, and receive the ISIS-II hyphen prompt.
- Enter the directive :F1:ICE85 followed by carriage return (cr), to load and begin running the ICE-85 program. ICE-85 signals with an asterisk prompt.
- If you have a line printer, you can record your session by typing LIST :LP: (cr). All ICE-85 interactions are then printed as well as displayed at the console, but the output from the CARS programs is not captured in the hard copy.

- The session involves two versions of the same program. CARS1 (listing in Figure 3-2) has deliberate errors to be found during the first part of the session. CARS2 has the errors fixed (listing in Figure 3-4). The second part of the emulation session demonstrates the correct operation of the program.
- Insert the diskette containing the compiled versions of the two programs in drive 0, replacing the system diskette.
- Enter the ICE-85 commands in the order shown. Do not type the comments (comments are preceded by semicolons (;)). The commentary on the pages facing the printout of the emulation session gives the exact form of each command, and interprets the effect.

Commands Used In The Examples

Memory and I/O Port Mapping Commands

ICE-85 treats references to addresses in both the user program and in the ICE-85 commands as *logical* addresses. The *physical* address that corresponds to each logical address reference can reside either in the user system or in the Intellec system. By using the MAP commands, you tell ICE-85 where to look in physical memory to find the address corresponding to each logical address reference to be used. After that, ICE-85 refers to the map each time an address is used.

Logical addresses range from 0 to 65,535 (64K-1), and are partitioned for mapping into 32 blocks of 2K addresses each. In our examples, all blocks of memory used by the programs are mapped to the Intellec system. To avoid loading the program over pre-existing system software, we have used *displacement* for the blocks that contain the sample program. When displacement is used, the physical addresses differ from the logical addresses; this is permitted only when Intellec memory is used.

Logical I/O port references range from 0 to 255, and are partitioned into 32 segments of 8 ports each. Physical port segments can be mapped to either the user system or to the Intellec system, but no displacement is permitted. In our example, all I/O ports used have been mapped to the Intellec system.

Table 3-1 gives a sample of the MAP commands used in this chapter, with their effects.

Table 3-1. Memory and I/O Port Mapping Examples

Example	Effect
MAP MEMORY 0 = INTELLEC 0	Assign physical location in Intellec memory for the lowest block of logical addresses.
MAP IO F0 TO FF = INTELLEC	Assign physical location in Intellec system for highest two segments of 8 I/O ports each.
MAP	Display current memory mapping.
MAP IO	Display current I/O port mapping.

Load Command

The LOAD command directs ICE-85 to load one or more program modules from diskette into memory. The memory used is the one specified by the map. In other words, ICE-85 treats each address reference in the program as a logical address, looks up the physical address corresponding to that logical address in the map, then loads the program code into the physical address. The program (and you) continues to refer to the logical addresses, and ICE-85 makes the translation automatically and invisibly.

The command:

```
LOAD :F0:CARS1
```

loads the program (file) named CARS1 from diskette drive 0 into the physical area mapped for it. The command also causes ICE-85 to load the program symbol table and statement number table (if there is one), using physical locations high in memory.

Symbolic References

The ICE-85 symbol table gives the logical addresses corresponding to:

- Labels in program modules loaded.
- User-symbols defined with the ICE-85 DEFINE command.

To refer to a symbol in ICE-85, precede it with a single period.

The line number table gives the logical address corresponding to each statement number in a PL/M-80 program loaded. You cannot define any additional statement numbers.

To refer to a statement number in ICE-85, precede it with a number sign (#).

Note: ICE-85 uses the memory map to find the physical location of the logical address corresponding to a symbol or statement number.

Table 3-2 gives some symbolic references used in a sample commands in this chapter.

Table 3-2. Symbol and Statement Number References in Command Examples

Example	Effect
DEFINE .START = PC	Define symbol START to be added to symbol table, and set symbol equal to the address currently in the program counter (PC).
SYMBOL	Display all current symbols and corresponding (address) values.
.CARS$WAITING	Display address corresponding to symbol CARS$WAITING.
#62	Display address of first instruction on program statement 62.

Emulation Control Commands

The emulation control commands GO, STEP, and CALL cause ICE-85 to begin running the user program, using the 8085 at the end of the Interface Cable as the CPU. By contrast, the ICE-85 commands themselves (for example, MAP) are executed by the 8080 processor in the Intellec system.

The GO command begins real-time emulation. If a FROM clause follows the command word GO, the clause specifies the starting address for emulation. If a TILL clause is included in the GO command, the clause specifies one or more halting conditions. Emulation halts when one of the halting conditions becomes true. The halting conditions used in the examples involve instructions executed and variables written.

The STEP command begins single-step emulation. Single-stepping is much slower than real-time emulation. The FROM and TILL clauses control the start and stop conditions for single-stepping just as with the GO command, although the form of the halting condition used with STEP differs from that used with GO.

The CALL command saves the processor register contents, including the program counter, then begins emulation from the address given in the command; the address is the starting point of a procedure block. On return from the procedure, the original program counter value and all the other processor register values are restored, and emulation continues automatically. Any halting conditions previously in effect at the time of the CALL continue to control emulation, but no additional conditions can be specified as part of the CALL command itself.

Table 3-3 gives some examples of these commands.

Table 3-3. Emulation Control Command Examples

Example	Effect
GO FROM .START TILL #56 EXECUTED	Begin real-time emulation at the address corresponding to the symbol START, and halt when (if) the first instruction in program line 56 is executed.
STEP FROM #55	Begin single-step emulation starting with the first instruction on program line 55.
CALL .I4	Emulate the procedure that begins at the address corresponding to the symbol I4, then continue emulating from the (saved) value of the program counter as it was when the CALL was issued.

Commands That Display or Change Memory and Register Contents

The examples include only a small sample of the commands used to display and change memory and register contents. To display the content of any register, just type the ICE-85 token for that register, followed by carriage return. The content is displayed on the next line. To display the content of a single location in memory, use the token BYTE followed by the desired address; a symbol or line number may be used in the command. To display the content of two contiguous addresses, use the command token WORD followed by the lower of the two contiguous locations.

To assign (change) the content of a register or memory location, follow the 'display' reference with an equals sign (=), then give the desired new value.

Table 3-4 gives examples.

Table 3-4. Memory and Register Contents Command Examples

Example	Effect
BYTE .CARS$WAITING	Display the contents of the single location corresponding to symbol .CARS$WAITING, using the ICE-85 memory map to find the physical location.
BYTE .CARS$WAITING = 1	Set the content of the location given by .CARS$WAITING, via the memory map, to 1.
PPC	Display the content of the previous program counter register (last instruction executed before emulation halted).

Trace Display Commands

ICE-85 maintains a trace buffer that records information on the instructions executed during emulation. The information can be displayed, and the format of the display can be controlled, with ICE-85 commands.

Trace data is recorded in *frames*. Each machine *cycle* represents two frames of trace data; one frame is recorded when the ADDRESS lines are valid, and a second frame is recorded when the low address lines represent valid DATA. An *instruction* represents several cycles. Trace data collected during real-time emulation can be displayed as instructions, cycles, or frames; the default is instructions. Trace data collected during single step emulation is displayed (when so enabled) after each instruction, and is always displayed as cycles.

The ENABLE DUMP command enables the automatic display of trace data after each instruction executed in single-stepping. It can be further qualified to display only certain types of instructions; in our examples, we have commanded displays of this kind only for CALL instructions.

The TRACE command sets the trace mode for display of data from real-time emulation to instructions, cycles, or frames.

The OLDEST command sets the trace buffer pointer to the beginning of the buffer, so that the earliest items collected can be displayed. The NEWEST command sets the pointer to the end of the buffer.

The PRINT command displays one or more entries before or after the position of the buffer pointer.

Table 3-5 gives examples of trace control commands.

Table 3-5. Trace Display Command Examples

Example	Effect
ENABLE DUMP CALL	Display program cycles and register contents after each CALL instruction executed during single-step emulation.
TRACE = CYCLE	Set trace mode to display one machine cycle on each display line.
OLDEST	Move trace buffer pointer to the first entry in the trace buffer.
NEWEST	Move the trace buffer pointer to just after the latest entry in the trace buffer.
PRINT 25	Display 25 entries (lines of buffered data), beginning with the one pointed to by the current buffer pointer.

Analysis of the Sample Program

The application presented is a simple traffic light controller. Imagine an intersection of a main street and a side street. The desired operation is that the light should stay green on the main street until a decision involving the number of cars waiting on the side street and the amount of time they have been waiting has been satisfied. We suppose that there is a sensor in the pavement on the side street that sends an interrupt to the computer when a car arrives. We do not include the control of a yellow light on either street.

Refer to the following figures:

Figure 3-1, Traffic Light Controller Program Flow Chart
Figure 3-2, CARS1 Program Listing
Figure 3-3, DELAY Subroutine Listing
Figure 3-4, CARS2 Program Listing

Associated with each street is a time called the cycle length. In the program, the variable named SIDE$CYCLE$LENGTH controls the fixed length of time the light is green on the side street when that cycle is called into action. Even though the light stays green on the main street until the decision rule is satisfied, we need a variable MAIN$CYCLE$LENGTH that is involved in the decision rule.

The decision rule is as follows. The side street gets a green light if either of the following two conditions is satisfied.

1. Two or more cars are waiting on the side street, and the main street has had the green light for a period of time greater than or equal to the variable MAIN$CYCLE$LENGTH.
2. One car is waiting on the side street, and the main street has had the green light for a period of time equal to or greater than two times the variable MAIN$CYCLE$LENGTH.

The system has one input and one output. The input is a signal that a car has arrived on the side street since the last time we sampled the input. The variable CARS$WAITING contains the number of cars waiting on the side street. The output goes to the traffic light controller. We assume that sending the controller a 1 makes the light on the main street green and the light on the side street red; sending it a 0 makes the light on the main street red and the light on the side street green. The variable LIGHT$STATUS represents this output.

The program is initialized with constants and variables set as follows.

MAIN$CYCLE$LENGTH = 8 seconds (Declaration not shown on chart)
SIDE$CYCLE$LENGTH = 5 seconds (Declaration not shown on chart)
MAIN$TIME = 0 (Time since last change to MAIN GREEN, SIDE RED)
SIDE$TIME = not set yet. (Time since last change to SIDE GREEN)
LIGHT$STATUS = 1 (MAIN GREEN, SIDE RED)
CARS$WAITING = 0

To simulate the sensor interrupt during the emulation of the program, press the ESC key to halt emulation without losing the content of any processor registers. Then enter the ICE-85 CALL command; in the example, user symbol I4 has been defined to ICE-85 as a synonym for the interrupt procedure name SIDE$STREET$CAR, to save keystrokes. Upon receiving the CALL command, ICE-85 saves the processor register contents (including the program counter), emulates the called procedure in its entirety, then restores the registers and continues emulating from the point where the program was halted. The interrupt can occur at any point in program execution; it is shown at the top of the flowchart for illustration only.

The effect of the interrupt routine is to increment CARS$WAITING by 1 each time the routine is called.

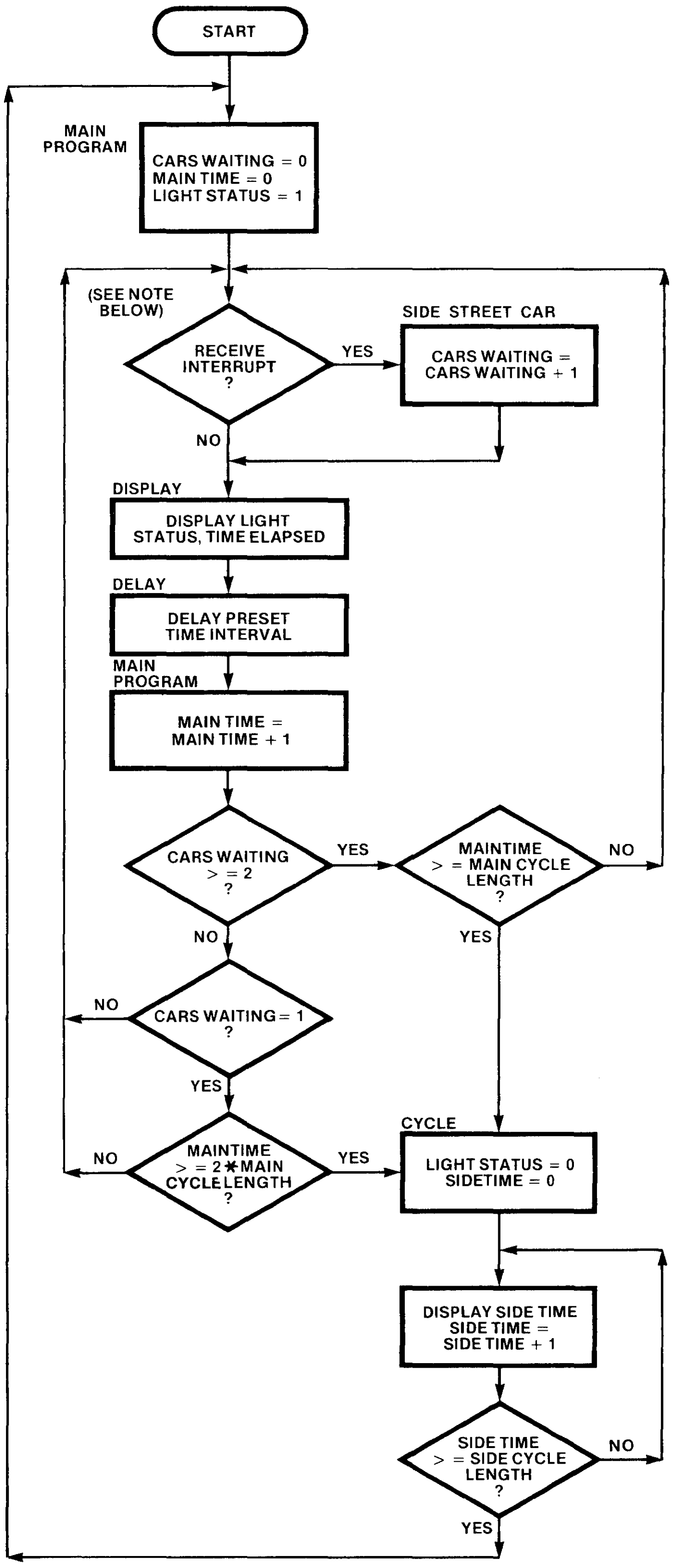
The procedure DISPLAY (lines 19 - 38) displays the current light status with one of two messages (declared in lines 7 and 8), then displays the number of seconds since the last light change in either direction. DISPLAY uses a procedure (CO) to display text characters; CO is a Monitor routine linked to the PL/M-80 program. DISPLAY also uses a procedure (function) LAST brought in from the PL/M-80 library at link time.

The procedure DELAY (Figure 3-3) produces a delay equal to .01 seconds times the value of the calling parameter. Line 54 in the main program and line 44 in the CYCLE module contain identical calls to DELAY: CALL DELAY (100). This produces a one-second delay. DELAY is written in 8080/8085 Assembly language and linked to the program as an external routine.

After each delay, MAIN$TIME is incremented by 1 (line 55).

Line 56 contains the decision rule. If it is sastisfied, CYCLE is called. If not, control reverts to the beginning of the DO FOREVER loop (line 52).

When CYCLE (lines 39 - 48) is activated, LIGHT$STATUS is set to 0, producing the 'SIDE GREEN' message when DISPLAY is called. After each one-second delay, SIDE$TIME is incremented until it exceeds SIDE$CYCLE$LENGTH (5 seconds). After the timeout, LIGHT$STATUS, CARS$WAITING, and MAIN$TIME are restored to their initial values and execution continues with a display of 'MAIN GREEN'.



NOTE: INTERRUPT PROCESSING CAN OCCUR AT ANY POINT. PROGRAM EXECUTION CONTINUES FROM THE POINT THE INTERRUPT OCCURRED.

Figure 3-1. Traffic Light Controller Program Flow Chart 463-12

```
                /* TRAFFIC LIGHT CONTROLLER PROGRAM */
                /* THIS PROGRAM CONTAINS DELIBERATE ERRORS! */

  1             CARS;
                DO;
  2     1          DECLARE (MAIN$TIME, SIDE$TIME) BYTE;
  3     1          DECLARE MAIN$CYCLE$LENGTH BYTE DATA(8), SIDE$CYCLE$LENGTH BYTE DATA(5);
  4     1          DECLARE CARS$WAITING BYTE;
  5     1          DECLARE LIGHT$STATUS BYTE;
  6     1          DECLARE FOREVER LITERALLY 'WHILE 1';
  7     1          DECLARE MAIN$GREEN$MESSAGE(*) BYTE DATA('MAIN GREEN, SIDE RED');
  8     1          DECLARE SIDE$GREEN$MESSAGE(*) BYTE DATA('SIDE GREEN, MAIN RED');
  9     1          DECLARE TIME$MESSAGE(*) BYTE DATA('  SECS SINCE LIGHT CHANGE');


                /* FOLLOWING PROCEDURE COUNTS CARS WAITING */

 10     1          SIDE$STREET$CAR: PROCEDURE;
 11     2             SIDE$TIME = SIDE$TIME + 1;
 12     2          END SIDE$STREET$CAR;

                /* FOLLOWING PROCEDURE DISPLAYS LIGHT STATUS AND ELAPSED TIME */

 19     1          DISPLAY: PROCEDURE(CYCLE$TIME);
 20     2             DECLARE CYCLE$TIME BYTE;
 21     2             DECLARE I BYTE;
 22     2             IF LIGHT$STATUS = 0 THEN
 23     2                DO I = 0 TO LAST(SIDE$GREEN$MESSAGE);
 24     3                   CALL CO(SIDE$GREEN$MESSAGE(I));
 25     3                END;
                      ELSE
 26     2                DO I = 0 TO LAST(MAIN$GREEN$MESSAGE);
 27     3                   CALL CO(MAIN$GREEN$MESSAGE(I));
 28     3                END;
 29     2             CALL CO(0DH);                         /* CARRIAGE RETURN */
 30     2             CALL CO(0AH);                         /* LINE FEED       */
 31     2             CALL CO( (CYCLE$TIME / 10) OR 30H );  /* TEN'S DIGIT     */
 32     2             CALL CO( (CYCLE$TIME MOD 10) OR 30H );/* UNIT'S DIGIT    */
 33     2             DO I = 0 TO LAST(TIME$MESSAGE);
 34     3                CALL CO(TIME$MESSAGE(I));
 35     3             END;
 36     2             CALL CO(0DH);                         /* CARRIAGE RETURN */
 37     2             CALL CO(0AH);                         /* LINE FEED       */
 38     2          END DISPLAY;

                /* FOLLOWING PROCEDURE CODED IN ASSEMBLY LANGUAGE AND LINKED IN */

 13     1          DELAY: PROCEDURE(TIME$HUNDREDTHS) EXTERNAL;
 14     2             DECLARE TIME$HUNDREDTHS BYTE;
 15     2          END DELAY;

                /* FOLLOWING PROCEDURE BORROWED FROM THE MONITOR */

 16     1          CO: PROCEDURE(CHAR) EXTERNAL;
 17     2             DECLARE CHAR BYTE;
 18     2          END CO;
```

Figure 3-2. CARS1 Listing

463-13A

```
                /* FOLLOWING PROCEDURE PERFORMS THE LIGHT CHANGE CYCLE */

39    1           CYCLE: PROCEDURE;
40    2              LIGHT$STATUS = 0;                     /* SIDE GREEN, MAIN RED */
41    2              SIDE$TIME = 0;
42    2              DO WHILE SIDE$TIME <= SIDE$CYCLE$LENGTH;
43    3                 CALL DISPLAY(SIDE$TIME);
44    3                 CALL DELAY(100);
45    3                 SIDE$TIME = SIDE$TIME + 1;
46    3              END;
47    2              LIGHT$STATUS = 1;                     /* MAIN GREEN, SIDE RED */
48    2           END CYCLE;

                /* MAIN PROGRAM -- EXECUTION BEGINS HERE */

49    1           LIGHT$STATUS = 1;                        /* START WITH MAIN GREEN */
50    1           CARS$WAITING = 0;
51    1           MAIN$TIME = 0;
52    1           DO FOREVER;
53    2              CALL DISPLAY(MAIN$TIME);
54    2              CALL DELAY(100);
55    2              MAIN$TIME = MAIN$TIME + 1;
56    2              IF (CARS$WAITING >= 2) AND (MAIN$TIME >= MAIN$CYCLE$LENGTH)
                        AND (CARS$WAITING = 1) AND (MAINTIME >= 2 * MAIN$CYCLE$LENGTH)
                        THEN
57    2                 DO;
58    3                    CALL CYCLE;
59    3                    CARS$WAITING = 0;
60    3                    MAIN$TIME = 0;
61    3                 END;
62    2           END;

63    1         END CARS;
```

Figure 3-2. CARS1 Listing (Continued)

```
LOC  OBJ            SEQ         SOURCE STATEMENT

                      1         ; TIME DELAY SUBROUTINE -- 0.01 SEC TIMES ARGUMENT
                      2         ;
                      3         CSEG
                      4         PUBLIC   DELAY
0000 79               5 DELAY:  MOV      A,C       ; ARGUMENT PASSED IN C REGISTER
0001 06FF             6         MVI      B,255     ; DELAY PARAMETER
0003 48               7 LAB1:   MOV      C,B
0004 0D               8 LAB2:   DCR      C
0005 221A00    C      9         SHLD     TEMP      ; WASTE 14 CYCLES
0008 221A00    C     10         SHLD     TEMP      ; DITTO
000B 221A00    C     11         SHLD     TEMP      ; DITTO
000E 221A00    C     12         SHLD     TEMP      ; DITTO
0011 00              13         NOP                ; WASTE 4 CYCLES
0012 C20400    C     14         JNZ      LAB2
0015 3D              15         DCR      A
0016 C20300    C     16         JNZ      LAB1
0019 C9              17         RET
                     18         ;
0002                 19 TEMP:   DS       2         ; DUMMY DATA STORAGE
                     20         END
```

Figure 3-3. DELAY Subroutine Listing

A Debugging Session Using ICE-85

We will now step through an emulation using ICE-85 as a software debugging tool. This exercise is divided into two parts. The first part isolates, identifies, and corrects program errors contained in the CARS1 program. The second part is a short exercise of CARS2 to verify the correction of the program errors found in CARS1.

1. Request a display of the memory map by entering the command:

   ```
   MAP
   ```

   This causes the display of the initial state of each of the thirty-two 2K memory blocks. The "G" denotes that the associated block is GUARDED. i.e., access to any memory address in that block is an error condition. All blocks are initially guarded.

2. Request a display of the IO may by entering the command:

   ```
   MAP IO
   ```

   This causes the display of the initial state of each of the thirty-two 8-port segments. The "G" denotes that the associated segment is GUARDED, i.e., access to any port in that segment is an error condition. All segments are initially guarded.

3. Enter the following "MAP =" commands:

   a. `MAP MEMORY 0 = INTELLEC 0`

   This provides access to ISIS-II reserved locations in block 0000 where ISIS-II variables used by the CARS1 program are stored. The warning message notifies you that you are mapping into the system area.

   b. `MAP MEMORY 3000 LENGTH 4K = INTELLEC 7000`

   This provides 4K address locations for the CARS1 program code.

   c. `MAP F800 = INTELLEC F800`

   This provides CARS1 access to the routine in MONITOR. Note that the term "MEMORY" is not mandatory in this command. "MEMORY" is the default specification and can be omitted.

4. Review the effects of the above map commands by entering:

   ```
   MAP
   ```

   This causes the display of the updated map. "I" means Intellec. Note the displacement of block 3000 into Intellec memory block 7000, and the displacement of the succeeding block, 3800, into Intellec memory block 7800.

```
1    *; DISPLAY INITIAL MEMORY MAP
     *
     *MAP
     SHARED
     0000=G          0800=G          1000=G          1800=G
     2000=G          2800=G          3000=G          3800=G
     4000=G          4800=G          5000=G          5800=G
     6000=G          6800=G          7000=G          7800=G
     8000=G          8800=G          9000=G          9800=G
     A000=G          A800=G          B000=G          B800=G
     C000=G          C800=G          D000=G          D800=G
     E000=G          E800=G          F000=G          F800=G
     *
     *
2    *; DISPLAY INITIAL I/O MAP
     *
     *MAP IO
     00=G            08=G            10=G            18=G
     20=G            28=G            30=G            38=G
     40=G            48=G            50=G            58=G
     60=G            68=G            70=G            78=G
     80=G            88=G            90=G            98=G
     A0=G            A8=G            B0=G            B8=G
     C0=G            C8=G            D0=G            D8=G
     E0=G            E8=G            F0=G            F8=G
     *
     *
3    *; MAP MEMORY BLOCK 0 FOR ACCESS TO ISIS-II VARIABLES
     *
     *MAP MEMORY 0 = INTELLEC 0
     WARN C1:MAPPING OVER SYSTEM
     *
     *; MAP SPACE FOR PROGRAM CODE
     *
     *MAP MEMORY 3000 LENGTH 4K = INTELLEC 7000
     *
     *; MAP BLOCK F800 FOR ACCESS TO MONITOR ROUTINE
     *; NOTE THAT 'MEMORY' IS THE DEFAULT, AND CAN BE OMITTED
     *
     *MAP F800 = INTELLEC F800
     *
     *
4    *; REVIEW EFFECTS OF MEMORY MAP COMMANDS
     *
     *MAP
     SHARED
     0000=I 0000     0800=G          1000=G          1800=G
     2000=G          2800=G          3000=I 7000     3800=I 7800
     4000=G          4800=G          5000=G          5800=G
     6000=G          6800=G          7000=G          7800=G
     8000=G          8800=G          9000=G          9800=G
     A000=G          A800=G          B000=G          B800=G
     C000=G          C800=G          D000=G          D800=G
     E000=G          E800=G          F000=G          F800=I F800
```

5. Map IO ports into Intellec by entering the command:

   ```
   MAP IO F0 TO FF = INTELLEC
   ```

   Note that no displacement is permitted. Review the updated IO may by entering:

   ```
   MAP IO
   ```

6. Load CARS1 from the diskette in drive 0 (:F0:) into Intellec memory:

   ```
   LOAD:F0:CARS1
   ```

7. Define two user symbols:

   ```
   DEFINE.START = PC
   ```

   The current contents of the program counter (PC) contains the starting address for CARS1. This sets START equal to the starting address.

   ```
   DEFINE.I4 = .SIDE$STREET$CAR
   ```

   This defines I4 as a synonym for SIDE$STREET$CAR. Therefore SIDE$STREET$CAR can be called using I4 in the call in place of the longer SIDE$STREET$CAR.

8. Display the symbol table:

   ```
   SYMBOL
   ```

   Note that START is set to the starting address (36C3H) and that I4 and SIDE$STREET$CAR are set to the same address (3727H).

```
5     *; MAP HIGHEST TWO SEGMENTS OF I/O PORTS
      *
      *MAP IO F0 TO FF = INTELLEC
      *
      *; REVIEW I/O MAP
      *
      *MAP IO
      00=G            08=G            10=G            18=G
      20=G            28=G            30=G            38=G
      40=G            48=G            50=G            58=G
      60=G            68=G            70=G            78=G
      80=G            88=G            90=G            98=G
      A0=G            A8=G            B0=G            B8=G
      C0=G            C8=G            D0=G            D8=G
      E0=G            E8=G            F0=I            F8=I
      *
      *
6     *; LOAD EMULATION PROGRAM, CARS1, CONTAINING ERRORS
      *; TO BE IDENTIFIED AND CORRECTED
      *
      *LOAD :F0:CARS1
      *
      *
7     *; DEFINE TWO USER SYMBOLS FOR USE IN EMULATION
      *
      *DEFINE .START = PC
      *DEFINE .I4 = SIDE$STREET$CAR
      *
      *
8     *; LOOK AT SYMBOL TABLE
      *
      *SYMBOL
      .START=36C3H
      .I4=3727H
      MODULE ..CARS
      .MEMORY=3889H
      .MAINTIME=3883H
      .SIDETIME=3884H
      .MAINCYCLELENGTH=3680H
      .SIDECYCLELENGTH=3681H
      .CARSWAITING=3885H
      .LIGHTSTATUS=3886H
      .MAINGREENMESSAGE=3682H
      .SIDEGREENMESSAGE=3696H
      .TIMEMESSAGE=36AAH
      .SIDESTREETCAR=3727H
      .DISPLAY=3735H
      .CYCLETIME=3887H
      .I=3888H
      .CYCLE=37E9H
      MODULE ..MODULE
      .DELAY=3816H
      .LAB1=3819H
      .LAB2=381AH
      .TEMP=3830H
      *
      *
```

9. Initiate emulation by entering the command GO.
10. Halt emulation after a few cycles by depressing the ESC key. Enter the command:

    ```
    CALL .I4
    ```

    This simulates an interrupt to enter a car on the side street. This call enters the first car on the side street. Emulation continues automatically after the call has been executed.
11. Let emulation continue until 18 seconds have elapsed. Then halt emulation via the ESC key. Main cycle time (2*MAIN$CYCLE$TIME) has been exceeded without a light change. This is a program error.
12. Enter the second car on the side street:

    ```
    CALL .I4
    ```

    Depress ESC key after 22 seconds of emulation.
13. Still no light change.
14. Display the contents of CARS$WAITING:

    ```
    BYTE .CARS$WAITING
    ```

    The command BYTE operator causes the display of the contents of the byte location specified by the parameter 'operator', in this case, CARS$WAITING. Note that the address of CARS$WAITING is 3885H (see symbol table). Therefore the display shows that CARS$WAITING = 0 (should be equal to 2).
15. Visually inspect statement 11 of the CARS$WAITING.LST listing. It contains the wrong variable, SIDE$TIME. It should be CARS$WAITING.
16. Check the address of SIDE$TIME by entering the command:

    ```
    SIDE$TIME
    ```

    The display contains the address of SIDE$TIME, 3884H.

```
9    *; PROGRAM IS NOW READY TO EMULATE, USING THE CURRENT PC
     *; CONTENTS AS THE START ADDRESS, AND 'FOREVER' AS THE
     *; DEFAULT HALTING CONDITION.
     *
     *GO
     EMULATION BEGUN
     (...)
10   EMULATION TERMINATED, PC=3824H
     PROCESSING ABORTED
     *
     *; HALTED MANUALLY (ESC KEY) TO ENTER FIRST CAR.
     *
     *CALL .I4
     EMULATION BEGUN
     (...)
11   EMULATION TERMINATED, PC=3821H
     PROCESSING ABORTED
     *
     *; STILL MAIN GREEN AFTER 18 SECONDS ELAPSED.  SHOULD HAVE
     *; CYCLED TO SIDE GREEN AT 16 SECONDS WITH ONE CAR WAITING.
     *; ENTER SECOND CAR AND TEST.
     *
12   *CALL .I4
     EMULATION BEGUN
     (...)
     EMULATION TERMINATED, PC=381EH
     PROCESSING ABORTED
     *
13   *; STILL MAIN GREEN AFTER 22 SECONDS WITH TWO CARS WAITING.
     *; CHECK CARS$WAITING; IT SHOULD BE EQUAL TO 2.
     *
14   *BYTE.CARS$WAITING
     3885H=00H
     *
15   *; CARS$WAITING IS NOT BEING INCREMENTED.  CHECK STATEMENT #11
     *; OF THE PROGRAM LISTING (FIGURE 3-2).  THIS IS WHERE CARS$WAITING
     *; SHOULD BE INCREMENTED.
     *
     *; THE INTERRUPT ROUTINE IS INCREMENTING THE WRONG VARIABLE --
     *; SIDE$TIME INSTEAD OF CARS$WAITING.
     *
16   *; CHECK THE ADDRESS OF SIDE$TIME.
     *
     *.SIDE$TIME
     3884H
     *
     *
```

17. Display the memory contents of statement 11 to statement 12 of CARS1 to determine which bytes contain the address:

    ```
    BYTE ..CARS#11 TO (..CARS#12 - 1)
    ```

    The command BYTE partition causes all of the bytes of object code generated by statement 11 to be displayed. The second and third bytes (84H 38H) displayed contain the address. The second byte contains the two low- order digits of the address (84H) and the third byte contains the two high- order digits (38H). Therefore the address is 3884H. This is the address of SIDE$TIME (see the symbol table). The entry ..CARS tells ICE-85 which module to search for the statement number.

18. The above bytes should be changed to contain the address of CARS$WAITING.

    a. Therefore, enter the following command to get the address of CARS$WAITING:

       ```
       .CARS$WAITING
       ```

       The display response (3885H) is the address of CARS$WAITING.

    b. Next, enter the following command to change the address in statement 11 to the address of CARS$WAITING:

       ```
       WORD (..CARS#11 + 1) = .CARS$WAITING
       ```

       WORD (..CARS#11 + 1) in the above command references a 16-bit word consisting of the second byte (..CARS#11 + 1) and the third byte (..CARS#11 + 2) of statement 11.

    c. Verify that the address has been changed correctly by entering the following command:

       ```
       BYTE ..CARS#11 TO (#12 -1)
       ```

       The display response shows that the address in statement 11 is now correct.

19. Enter the following command to restart CARS1:

    ```
    GO FROM .START FOREVER
    ```

    This command restores the initial start and halt condition.

20. After 3 seconds manually halt emulation via the ESC key and reenter the first car with the following command:

    ```
    CALL .I4
    ```

21. Manually halt emulation via the ESC key after 17 seconds have elapsed. Enter the following command to examine the contents of CARS$WAITING:

    ```
    BYTE .CARS$WAITING
    ```

```
17   *; DISPLAY MEMORY CONTENTS FROM STATEMENT #11 UP TO
     *; STATEMENT #12 TO FIND THE BYTES WHERE THIS ADDRESS
     *; IS STORED.
     *
     *BYTE ..CARS#11 TO (..CARS#12 - 1)
     372BH=21H 84H 38H 34H
     *
     *; THE MIDDLE TWO BYTES (84H 38H) ARE THE ADDRESS OF SIDE$TIME.
     *
18   *; REPLACE WITH THE ADDRESS OF CARS$WAITING.
     *
     *; FIRST, CHECK THE ADDRESS OF CARS$WAITING.
     *
     *.CARS$WAITING
     3885H
     *
     *; CHANGE SIDE$TIME TO CARS$WAITING IN INTERRUPT ROUTINE.
     *
     *WORD (..CARS#11 + 1) = .CARS$WAITING
     *
     *; VERIFY THAT THE CHANGE WAS MADE CORRECTLY.
     *
     *BYTE ..CARS#11 TO (..CARS#12 - 1)
     372BH=21H 85H 38H 34H
     *
     *; CHANGE WAS MADE CORRECTLY.  RESTART FROM .START.
     *
19   *GO FROM .START FOREVER
     EMULATION BEGUN
     (...)
     EMULATION TERMINATED, PC=3821H
     PROCESSING ABORTED
     *
     *
20   *; MAIN GREEN, 3 SECONDS ELAPSED.  ENTER FIRST CAR AGAIN.
     *
     *CALL .I4
     EMULATION BEGUN
     (...)
21   EMULATION TERMINATED, PC=3824H
     PROCESSING ABORTED
     *
     *; MAIN STILL GREEN AFTER 17 SECONDS.  SHOULD HAVE CYCLED
     *; AT 16 SECONDS WITH ONE CAR WAITING.
     *
     *; DOUBLE-CHECK CARS$WAITING
     *
     *BYTE . CARS$WAITING
     3885H=01H
     *
```

22. CARS$WAITING incremented correctly, enter second car:

    ```
    CALL .I4
    ```

23. Halt emulation manually after 21 seconds. Still no light change, check contents of CARS$WAITING again:

    ```
    BYTE .CARS$WAITING
    ```

    CARS$WAITING equal to 2, incrementing properly.

24. Since CARS$WAITING is incrementing correctly, enter the following command to determine if CYCLE is ever executed:

    ```
    GO TILL .CYCLE EXECUTED
    ```

    Emulation should not halt.

25. Manually halt emulation after 25 seconds.

26. Change to Single Step mode using automatic display.

    a. Enter the following command to enable automatic display of call instructions:

       ```
       ENABLE DUMP CALL
       ```

    b. Do single step beginning at statement 52, the beginning of the DO loop in the Main program:

       ```
       STEP FROM #52
       ```

       In the display, P = PC, the starting address of the called routine; the remainder of the display output can be ignored at this time.

27. Manually halt emulation with the ESC key. Note P = 3816H in the last line of display above. 3816H is the start of the DELAY subroutine. Single stepping through DELAY will take too much time.

28. Bypass DELAY. Start single stepping from statement 55. Statement 55 increments MAIN$TIME.

    ```
    STEP FROM ..CARS#55
    ```

```
22  *; THAT IS CORRECT.  ENTER A SECOND CAR.
    *
    *CALL .I4
    EMULATION BEGUN
    (...)
23  EMULATION TERMINATED, PC=381AH
    PROCESSING ABORTED
    *
    *; STILL MAIN GREEN AFTER 21 SECONDS WITH TWO CARS WAITING.
    *; SHOULD HAVE CYCLED.  CHECK CARS$WAITING AGAIN.
    *
    *BYTE .CARS$WAITING
    3885H=02H
    *
24  *; CARS$WAITING IS INCREMENTING WITH EACH CAR.
    *; IS CYCLE EVER EXECUTED?
    *
    *GO TILL .CYCLE EXECUTED
    EMULATION BEGUN
    (...)
25  EMULATION TERMINATED, PC=3824H
    PROCESSING ABORTED
    *
    *; MAIN STILL GREEN AT 25 SECONDS.  CYCLE IS NOT EXECUTING.
    *
26  *; TRY SINGLE-STEPPING FROM THE BEGINNING OF THE DO-LOOP
    *; IN MAIN PROGRAM, AND DISPLAY ALL 'CALL' INSTRUCTIONS EXECUTED.
    *
    *ENABLE DUMP CALL
    *STEP FROM ..CARS#52
    EMULATION BEGUN
     36D7-E-CD 36D8-R-35 36D9-R-37 3880-W-36 387F-W-DA
    P=3735H S=387FH A=59H F=B4H B=FFH C=19H D=00H E=30H H=43H L=19H I=08H
     377E-E-CD 377F-R-09 3780-R-F8 387E-W-37 387D-W-81
    P=F809H S=387DH A-13H F=10H B=36H C=4DH D=00H E=30H H=36H L=82H I=08H
     377E-E-CD 377F-R-09 3780-R-F8 387E-W-37 387D-W-81
    P=F809H S=387DH A=13H F=14H B=36H C=41H D=00H E=30H H=36H L=83H I=08H
    (...)

    P=F809H S=387DH A=0DH F=10H B=36H C=0AH D=00H E=30H H=38H L=88H I=08H
     36DC-E-CD 36DD-R-16 36DE-R-38 3880-W-36 387F-W-DF
    P=3816H S=387FH A=0AH F=10H B=36H C=64H D=00H E=30H H=38H L=88H I=08H
27  EMULATION TERMINATED, PC=381EH
    PROCESSING ABORTED
    *
    *; CAN'T USE STEP. DELAY ROUTINE TAKES TOO LONG.
    *
28  *; TRY STEPPING FROM THE NEXT STATEMENT AFTER THE RETURN FROM DELAY.
    *
    *STEP FROM ..CARS#55
    EMULATION BEGUN
     3709-E-CD 370A-R-65 370B-R-38 387C-W-37 387B-W0C
    P=3865H S=387BH A=1AH F=54H B=FFH C=FFH D=00H E=30H H=00H L=10H I=08H
     36D7-E-CD 36D8-R-35 36D9-R-37 387E-W-36 387D-W-DA
    P=3735H S=387DH A=00H F=54H B=00H C=1AH D=00H E=1AH H=43H L=1AH I=08H
     377E-E-CD 377F-R-09 3780-R-F8 387C-W-37 387B-W-81
    P=F809H S=387BH A=13H F=10H B=36H C=4DH D=00H E=1AH H=36H L=82H I=08H
    PROCESSING ABORTED
    *
```

29. Manually halt single step after four automatic single step displays. Note P = 3735H in the fourth line of display above. 3735H is the address of DISPLAY (see symbol table). Therefore CYCLE is not being executed. The IF statement (statement 56) is not branching correctly.
30. Restart emulation:

    ```
    GO FROM .START TILL ..CARS#56 EXECUTED
    ```

    This command sets PC = START and specifies a halt on statement 56 (IF statement).
31. After the halt on #56, set one of the test conditions of the IF statement:

    ```
    BYTE .CARS$WAITING = 1
    BYTE .MAIN$TIME = 2* (BYTE .MAIN$CYCLE$LENGTH)
    ```

32. Set up emulation to continue from statement 56 and to halt when statement 58 or 62 is executed:

    ```
    GO TILL ..CARS#58 E OR ..CARS#62 E
    ```

    Note the use of E, an abbreviation for EXECUTED. When emulation halts, look at the contents of the Previous Program Counter (PPC):

    ```
    PPC
    ```

    The display response is 3722H.
33. Display the address of the first byte location of object code for statement 62:

    ```
    ..CARS#62
    ```

    The address of statement 62 is 3722H, the address contained in the PPC in step 32. Therefore CYCLE was not executed. The IF statement took the "false" branch although you set the condition to the "true" state in step 31 above.
34. Repeat steps 30-32 with the second IF condition:

    ```
    GO FROM .START TILL ..CARS#56 EXECUTED
    ```

35. When emulation halts, set up the second IF condition:

    ```
    BYTE .CARS$WAITING = 2
    BYTE .MAIN$TIME = BYTE .MAIN$CYCLE$TIME
    ```

36. Repeat test:

    ```
    GO TILL ..CARS#58 E or ..CARS# 62 E
    ```

    Display PPC:

    ```
    PPC
    ```

    Display contains 3722H.

    Display the address of statement 62:

    ```
    ..CARS#62
    ```

    Address of statement 62 is 3722H, the same as the contents of the PPC. Therefore emulation bypassed CYCLE again. The IF statement failed to branch correctly under either branch condition.
37. Visually examine the IF statement in detail. Notice the second line of the IF statement is:

    ```
    AND (CARS$WAITING = 1) AND (MAIN$TIME>= 2*MAIN$CYCLE$LENGTH)
    ```

    The line is incorrect, the first "AND" should be "OR". As stated, the IF conditions contained in the first and second lines must both be true at the same time for the IF statement to branch to CYCLE. This is impossible. The test will always fail as stated. It must be one condition or the other. Therefore the AND must be changed to OR. To make this correction, the source code in CARS1 must be changed and CARS1 recompiled. CARS2 is a recompilation of CARS1 with the necessary corrections.

```
29   *; ADDRESS (P=3735H) IS THE BEGINNING OF DISPLAY. CYCLE NEVER EXECUTED.
     *
30   *; TRY EMULATING FROM .START, HALTING RIGHT BEFORE THE DECISION POINT.
     *; TEST THE 'IF' STATEMENT BY SETTING THE VALUES MANUALLY.
     *
     *GO FROM .START TILL ..CARS#56 EXECUTED
     EMULATION BEGUN
     EMULATION TERMINATED, PC=36E6H
     *
31   *BYTE .CARS$WAITING = 1
     *BYTE .MAIN$TIME = 2*(BYTE .MAIN$CYCLE$LENGTH)
     *
32   *; NOW TO EMULATE UNTIL CYCLE IS CALLED (STATEMENT #58) OR UNTIL THE
     *; END OF THE DO-LOOP IN THE MAIN PROGRAM IS EXECUTED (STATEMENT #62),
     *; HALTING EMULATION ON EITHER CONDITION.
     *
     *GO TILL ..CARS#58 E OR ..CARS#62 E
     EMULATION BEGUN
     EMULATION TERMINATED, PC=36D3H
     *
     *; LOOK AT PREVIOUS PROGRAM COUNTER (PPC).
     *
     *PPC
     3722H
     *
33   *; COMPARE WITH STATEMENT #62
     *
     *#62
     3722H
     *
     *; WE BRANCHED AROUND THE CALL TO CYCLE.  THE 'IF' CONDITION WAS NOT
     *; SATISFIED.  TRY THE ALTERNATE CONDITION.
34   *GO FROM .START TILL ..CARS #56 EXECUTED
     EMULATION BEGUN
     EMULATION TERMINATED, PC=36E6H
     *
35   *BYTE .CARS$WAITING = 2
     *BYTE .MAIN$TIME = BYTE .MAIN$CYCLE$LENGTH
     *
36   *GO TILL ..CARS#58 E OR ..CARS#62 E
     EMULATION BEGUN
     EMULATION TERMINATED, PC=36D3H
     *
     *PPC
     3722H
     *
     *#62
     3722H
     *
     *; SAME RESULT AS FOR ONE CAR.  THE 'IF' TEST IS FAILING.
     *
     *
37   *; EXAMINE THE 'IF' TEST.
     *
     *; NOTE THAT THE 'IF' STATEMENT HAS 'AND' INSTEAD OF 'OR'.
     *; MUST EXIT ICE-85, EDIT AND RECOMPILE PROGRAM.
     *; FIXED PROGRAM IS CARS2.
     *
     *EXIT
```

```
               /* TRAFFIC LIGHT CONTROLLER PROGRAM */

  1            CARS;
               DO;
  2    1          DECLARE (MAIN$TIME, SIDE$TIME) BYTE;
  3    1          DECLARE MAIN$CYCLE$LENGTH BYTE DATA(8), SIDE$CYCLE$LENGTH BYTE DA
  4    1          DECLARE CARS$WAITING BYTE;
  5    1          DECLARE LIGHT$STATUS BYTE;
  6    1          DECLARE FOREVER LITERALLY 'WHILE 1';
  7    1          DECLARE MAIN$GREEN$MESSAGE(*) BYTE DATA('MAIN GREEN, SIDE RED');
  8    1          DECLARE SIDE$GREEN$MESSAGE(*) BYTE DATA('SIDE GREEN, MAIN RED');
  9    1          DECLARE TIME$MESSAGE(*) BYTE DATA('  SECS SINCE LIGHT CHANGE');


               /* FOLLOWING PROCEDURE COUNTS CARS WAITING */

 10    1          SIDE$STREET$CAR: PROCEDURE;
 11    2             CARS$WAITING = CARS$WAITING + 1;
 12    2          END SIDE$STREET$CAR;


               /* FOLLOWING PROCEDURE CODED IN ASSEMBLY LANGUAGE AND LINKED IN */

 13    1          DELAY: PROCEDURE(TIME$HUNDREDTHS) EXTERNAL;
 14    2             DECLARE TIME$HUNDREDTHS BYTE;
 15    2          END DELAY;


               /* FOLLOWING PROCEDURE BORROWED FROM THE MONITOR */

 16    1          CO: PROCEDURE(CHAR) EXTERNAL;
 17    2             DECLARE CHAR BYTE;
 18    2          END CO;

               /* FOLLOWING PROCEDURE DISPLAYS LIGHT STATUS AND ELAPSED TIME */

 19    1          DISPLAY: PROCEDURE(CYCLE$TIME);
 20    2             DECLARE CYCLE$TIME BYTE;
 21    2             DECLARE I BYTE;
 22    2             IF LIGHT$STATUS = 0 THEN
 23    2                DO I = 0 TO LAST(SIDE$GREEN$MESSAGE);
 24    3                   CALL CO(SIDE$GREEN$MESSAGE(I));
 25    3                END;
                     ELSE
 26    2                DO I = 0 TO LAST(MAIN$GREEN$MESSAGE);
 27    3                   CALL CO(MAIN$GREEN$MESSAGE(I));
 28    3                END;
 29    2             CALL CO(0DH);                          /* CARRIAGE RETURN */
 30    2             CALL CO(0AH);                          /* LINE FEED       */
 31    2             CALL CO( (CYCLE$TIME / 10) OR 30H );   /* TEN'S DIGIT     */
 32    2             CALL CO( (CYCLE$TIME MOD 10) OR 30H );/* UNIT'S DIGIT    */
 33    2             DO I = 0 TO LAST(TIME$MESSAGE);
 34    3                CALL CO(TIME$MESSAGE(I));
 35    3             END;
 36    2             CALL CO(0DH);                          /* CARRIAGE RETURN */
 37    2             CALL CO(0AH);                          /* LINE FEED       */
 38    2          END DISPLAY;
```

Figure 3-4. CARS2 Program Listing

```
/* FOLLOWING PROCEDURE PERFORMS THE LIGHT CHANGE CYCLE */

   CYCLE: PROCEDURE;
      LIGHT$STATUS = 0;                       /* SIDE GREEN, MAIN RED *
      SIDE$TIME =0;
      DO WHILE SIDE$TIME <= SIDE$CYCLE$LENGTH;
         CALL DISPLAY(SIDE$TIME);
         CALL DELAY(100);
         SIDE$TIME = SIDE$TIME + 1;
      END;
      LIGHT$STATUS = 1;                       /* MAIN GREEN, SIDE RED *
   END CYCLE;

/* MAIN PROGRAM -- EXECUTION BEGINS HERE */

   LIGHT$STATUS = 1;                          /* START WITH MAIN GREEN
   CARS$WAITING = 0;
   MAIN$TIME = 0;
   DO FOREVER;
      CALL DISPLAY(MAIN$TIME);
      CALL DELAY(100);
      MAIN$TIME = MAIN$TIME + 1;
      IF (CARS$WAITING >= 2) AND (MAIN$TIME >= MAIN$CYCLE$LENGTH)
         OR (CARS$WAITING = 1) AND (MAIN$TIME >= 2 * MAIN$CYCLE$LENGT
         THEN
         DO;
            CALL CYCLE;
            CARS$WAITING = 0;
            MAIN$TIME = 0;
         END;
   END;

END CARS;
```

Figure 3-4. CARS2 Program Listing (Continued)

The following steps are a brief emulation of CARS2 to verify that the program changes made to CARS1 enable CARS2 to run properly.

1. Enter the same mapping commands as in CARS1.
2. Define I4 and START.
3. Load the corrected program.
4. Check the symbol table.
5. Start emulation from START and let CARS2 cycle for at least 20 cycles. Halt CARS2 manually with the ESC key. Note that there was no signal light change as no cars were entered on the side street.
6. Call I4 to enter one car on the side street. Allow emulation to continue for at least 30 cycles. Notice that the side street signal turns green for 5 cycles and then the main street turns green again.

```
1    *MAP 0 = INT 0
     WARN C1:MAPPING OVER SYSTEM
     *MAP 3000 LEN 4K = INT 7000
     *MAP F800 = INT F800
     *MAP IO F0 TO FF = INT
     *
2    *LOAD :F0:CARS2
     *
3    *DEFINE .I4 = .SIDE$STREET$CAR
     *DEFINE .START = PC
     *
4    *SYMBOL
     .I4=3727H
     .START=36C3H
     MODULE ..CARS
     .MEMORY=3889H
     .MAINTIME=3883H
     .SIDETIME=3884H
     .MAINCYCLELENGTH=3680H
     .SIDECYCLELENGTH=3681H
     .CARSWAITING=3885H
     .LIGHTSTATUS=3886H
     .MAINGREENMESSAGE=3682H
     .SIDEGREENMESSAGE=3696H
     .TIMEMESSAGE=36AAH
     .SIDESTREETCAR=3727H
     .DISPLAY=3735H
     .CYCLETIME=3887H
     .I=3888H
     .CYCLE=37E9H
     MODULE ..MODULE
     .DELAY=3816H
     .LAB1=3819H
     .LAB2=381AH
     .TEMP=3830H
     *
     *
5    *GO
     EMULATION BEGUN
     EMULATION TERMINATED, PC=3824H
     PROCESSING ABORTED
     *
6    *CALL .I4
     EMULATION BEGUN
     EMULATION TERMINATED, PC=3828H
     PROCESSING ABORTED
     *
     *
```

7. Halt emulation and call I4 to enter one car on the side street. Repeat for the second car. Let emulation continue for at least 30 cycles and then halt manually. Observe that the signal lights change in the proper sequence. In other words, CARS2 is sequencing correctly.
8. Display the address of DELAY.
9. Start emulation from DELAY and halt when MAIN$TIME is updated. Trace data is being collected in the trace buffer during this interval.
10. Set the trace buffer pointer to the top of the buffer. The following steps print the contents of the trace buffer in each of the print modes: instruction, frame, and cycle.
11. Print the first twenty-five instructions contained in the buffer. Since no print mode has been specified, the output is in the default mode, instruction format. Since emulation started from DELAY, the trace buffer has been filled and refilled a number of times by the DELAY subroutine. Therefore the first address in the listing is not meaningful at this time.

```
7       *CALL .I4
        EMULATION BEGUN
        EMULATION TERMINATED, PC=3824H
        PROCESSING ABORTED
        *
        *
        *CALL .I4
        EMULATION BEGUN
        EMULATION TERMINATED, PC=381AH
        PROCESSING ABORTED
        *
        *
8       *.DELAY
        3816H
        *
        *
9       *GO FROM .DELAY TILL .MAINTIME WRITTEN
        EMULATION BEGUN
        EMULATION TERMINATED, PC=36E3H
        *
        *
10      *OLDEST
        *
11      *PRINT 25
              ADDR INSTRUCTION ADDR-S-DA ADDR-S-DA ADDR-S-DA ADDR-S-DA
        0001: 381A DCR C
        0003: 381B SHLD 3830   3830-W-88 3831-W-38
        0013: 381E SHLD 3830   3830-W-88 3831-W-38
        0023: 3821 SHLD 3830   3830-W-88 3831-W-38
        0033: 3824 SHLD 3830   3830-W-88 3831-W-38
        0043: 3827 NOP
        0045: 3828 JNZ 381A
        0051: 381A DCR C
        0053: 381B SHLD 3830   3830-W-88 3831-W-38
        0063: 381E SHLD 3830   3830-W-88 3831-W-38
        0073: 3821 SHLD 3830   3830-W-88 3831-W-38
        0083: 3824 SHLD 3830   3830-W-88 3831-W-38
        0093: 3827 NOP
        0095: 3828 JNZ 381A
        0101: 381A DCR C
        0103: 381B SHLD 3830   3830-W-88 3831-W-38
        0113: 381E SHLD 3830   3830-W-88 3831-W-38
        0123: 3821 SHLD 3830   3830-W-88 3831-W-38
        0133: 3824 SHLD 3830   3830-W-88 3831-W-38
        0143: 3827 NOP
        0145: 3828 JNZ 381A
        0151: 381A DCR C
        0153: 381B SHLD 3830   3830-W-88 3831-W-38
        0163: 381E SHLD 3830   3830-W-88 3831-W-38
        0173: 3821 SHLD 3830   3830-W-88 3831-W-38
        *
        *
```

12. Move the buffer pointer to the end of the trace buffer.
13. Print the last twenty-five instructions stored in the buffer. Notice that each instruction listed consists of a number of frames. For example the first instruction listed extends from frame address 0853 to 0862, ten frames. The total span of frames covered by these twenty-five instructions are: 1020 - 0853 = 169 frames.

```
12   *NEWEST
     *
13   *PRINT -25
           ADDR INSTRUCTION ADDR-S-DA ADDR-S-DA ADDR-S-DA ADDR-S-DA
     0853: 381B SHLD 3830   3830-W-88 3831-W-38
     0863: 381E SHLD 3830   3830-W-88 3831-W-38
     0873: 3821 SHLD 3830   3830-W-88 3831-W-38
     0883: 3824 SHLD 3830   3830-W-88 3831-W-38
     0893: 3827 NOP
     0895: 3828 JNZ 381A
     0901: 381A DCR C
     0903: 381B SHLD 3830   3830-W-88 3831-W-38
     0913: 381E SHLD 3830   3830-W-88 3831-W-38
     0923: 3821 SHLD 3830   3830-W-88 3831-W-38
     0933: 3824 SHLD 3830   3830-W-88 3831-W-38
     0943: 3827 NOP
     0945: 3828 JNZ 381A
     0951: 381A DCR C
     0953: 381B SHLD 3830   3830-W-88 3831-W-38
     0963: 381E SHLD 3830   3830-W-88 3831-W-38
     0973: 3821 SHLD 3830   3830-W-88 3831-W-38
     0983: 3824 SHLD 3830   3830-W-88 3831-W-38
     0993: 3827 NOP
     0995: 3828 JNZ
     0999: 382B DCR A
     1001: 382C JNZ
     1005: 382F RET         3881-R-DF 3882-R-36
     1011: 36DF LXI H, 3883
     1017: 36E2 INR M       3883-R-03 3883-W-04
     *
     *
```

14. Change the print mode to CYCLES.
15. Print the last twenty-five cycles contained in the trace buffer. Notice that these twenty-five cycles cover fifty frames of trace data.

```
14   *TRACE = CYCLES
     *
15   *PRINT -25
           ADDR-S-DA SD RW M
     0973: 3821-E-22 00 10 1
     0975: 3822-R-30 00 10 1
     0977: 3823-R-38 00 10 1
     0979: 3830-W-88 00 01 1
     0981: 3831-W-38 00 01 1
     0983: 3824-E-22 00 10 1
     0985: 3825-R-30 00 10 1
     0987: 3826-R-38 00 10 1
     0989: 3830-W-88 00 01 1
     0991: 3831-W-38 00 01 1
     0993: 3827-E-00 00 10 1
     0995: 3828-E-C2 00 10 1
     0997: 3829-R-1A 00 10 1
     0999: 382B-E-3D 00 10 1
     1001: 382C-E-C2 00 10 1
     1003: 382D-R-19 00 10 1
     1005: 382F-E-C9 00 10 1
     1007: 3881-R-DF 00 10 1
     1009: 3882-R-36 00 10 1
     1011: 36DF-E-21 00 10 1
     1013: 36E0-R-83 00 10 1
     1015: 36E1-R-38 00 10 1
     1017: 36E2-E-34 00 10 1
     1019: 3883-R-03 00 10 1
     1021: 3883-W-04 00 01 1
     *
     *
     *
```

16. Change print mode to FRAMES.
17. Print the last twenty-five frames of trace data contained in the trace buffer.
18. Terminate program operation.

```
16   *TRACE = FRAMES
     *
17   *PRINT -25
           A/D  S D SD RW M
     0997: 381A R 1 00 10 1
     0998: 382B E 0 00 00 1
     0999: 383D E 1 00 10 1
     1000: 382C E 0 00 00 1
     1001: 38C2 E 1 00 10 1
     1002: 382D R 0 00 00 1
     1003: 3819 R 1 00 10 1
     1004: 382F E 0 00 00 1
     1005: 38C9 E 1 00 10 1
     1006: 3881 R 0 00 00 1
     1007: 38DF R 1 00 10 1
     1008: 3882 R 0 00 00 1
     1009: 3836 R 1 00 10 1
     1010: 36DF E 0 00 00 1
     1011: 3621 E 1 00 10 1
     1012: 36E0 R 0 00 00 1
     1013: 3683 R 1 00 10 1
     1014: 36E1 R 0 00 00 1
     1015: 3638 R 1 00 10 1
     1016: 36E2 E 0 00 00 1
     1017: 3634 E 1 00 10 1
     1018: 3883 R 0 00 00 1
     1019: 3803 R 1 00 10 1
     1020: 3883 W 0 00 00 1
     1021: 3804 W 1 00 01 1
     *
     *
     *
18   *EXIT
```

CHAPTER 4
ICE-85 METALANGUAGE

Introduction

The ICE-85 software provides you with an easy-to-use English language command set for controlling ICE-85 execution in a variety of functional modes.

The ICE-85 commands enable you to:

- Initialize the ICE-85 system, map your program to memory in your system or in the Intellec Microcomputer Development System, and load your program from a diskette file.
- Specify starting and stopping conditions for emulation.
- Execute real-time emulation of your software (and hardware).
- Execute single-step emulation.
- Specify conditions for trace data collection.
- Collect and display trace data on hardware conditions occurring during emulation.
- Display and alter 8085 registers, memory locations, and I/O ports.
- Copy the (modified) program from Intellec or user memory to a diskette file, and exit the ICE-85 system.

An example of one complete ICE-85 command, in this case one of the forms of the GO command, is shown in Figure 4-1. This command causes emulation to start and specifies the conditions that will halt emulation. The command is made up of nine separate mnemonic codes (character strings): GO, FROM, 0123H, TILL, LOCATION, 0400, EXECUTED, OR, and SY0. Each of these mnemonics provides a particular element of information necessary to inform ICE-85 of the specific command functions to be executed. Table 4-1 defines the function of each of these mnemonics as well as the match condition and the FROM and TILL clauses. This string of mnemonics requests ICE-85 to "start emulation at memory location 0123H and to continue emulation until either memory location 0400 is executed or until external signal Sync 0 goes high". Every ICE-85 command is composed of one or more such mnemonics.

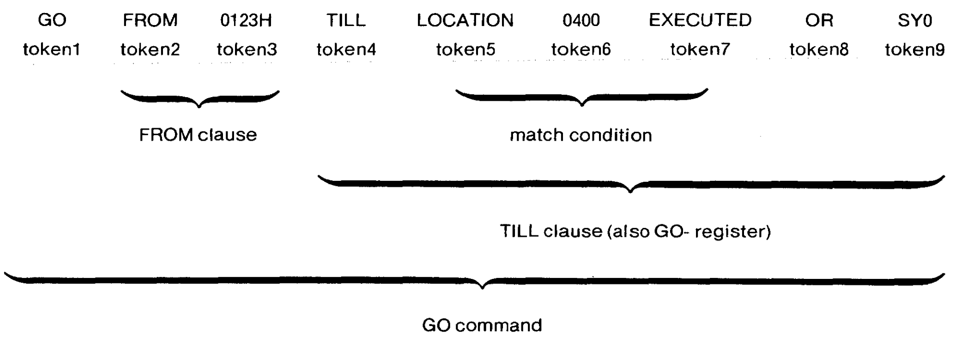


Figure 4-1. Example of a GO Command

Table 4-1. Definition of GO Command Functions

Token Number	Name	Function
1	GO	Go command specifier; requests and initiates emulation.
2	FROM	Indicates that the next token or expression is the starting address for emulation.
3	0123H	Starting address in hexadecimal radix.
2,3	FROM clause	FROM 0123H causes the program counter (PC) to be loaded with 0123H, the starting address for the emulation.
4	TILL	Indicates that breakpoint (halting) parameters are to follow.
5	LOCATION	Indicates that the address lines are to contain the memory address given by the token or numerical expression that follows.
6	0400	The address of the specified memory LOCATION.
7	EXECUTED	Match pattern for an instruction fetch.
5,6,7	match-condition	Emulation is to halt if an instruction fetch is executed from memory location 0400.
8	OR	Indicates that a second halt parameter is to follow.
9	SY0	Halt emulation if the external synchronization line SY0 goes low.
4 - 9	TILL clause	TILL LOCATION 0400 EXECUTED OR SY0 specifies that the emulation is to halt under either of the conditions given. This clause is also called the 'GO-register' in the ICE-85 language.

Thus, the ICE-85 command language is composed of a unique character set and vocabulary of mnemonics. The character set is used to construct mnemonics and in turn, the mnemonics are used to construct ICE-85 commands.

This chapter presents a metalanguage that classifies and identifies both character and mnemonic types. The metalanguage consists of a set of meta-terms. Each meta-term is a class-name for a specific type of character or mnemonic and is always shown in *lower-case italics*. Any character string not in lower-case italics is a specific mnemonic or character. For example, the meta-term *user-name* refers to the entire class of user names. The character string SAM is a particular user name.

As shown in Figure 4-1, all of the mnemonics are referred to as *tokens*. All ICE-85 mnemonics are divided into two classes: *tokens* and *special tokens. Tokens* encompass *constants, keywords, symbols,* and *user names. Special tokens* encompass *relational operators, arithmetic operators,* and *punctuation*.

The remainder of this chapter is devoted to a presentation of the character and mnemonic classes and the associated meta-terms that denote these classes. The meta-notation used in this manual for command syntax is also presented. This chapter will specify all primary meta-terms but will not present the entire set of meta-terms. Chapter 5 will introduce additional meta-terms that represent special subsets of tokens for use with particular commands. These additional meta-terms will be defined using the primary meta-terms contained in this chapter along with any particular qualifications and restrictions involved in the command usage.

Character Set

The valid characters in the ICE-85 command language include upper and lower case alphabetic ASCII characters A through Z and the set of digits 0 through 9. The space serves as a delimiter for tokens, and carriage-return/line-feed characters are also valid, delimiting command lines. A question mark, ?, @ sign, and $ sign are also valid in user-defined names entered in the command language.

The *algebraic operators* + and - (binary and unary), the asterisk (*), and slash (/), *relational operators* (=, <, >), the ampersand, semicolon, period, parentheses, pound sign (#), and comma constitute the only other valid ASCII characters for ICE-85: all other characters are treated as if a space was typed.

alphabetic characters:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

numeric characters:

0 1 2 3 4 5 6 7 8 9 (A B C D E F: hexadecimal characters)

special characters:

+ − < = > $ ' &) . (; * / #

This character set is used to construct the vocabulary that constitutes the command language.

Tokens

A *token* in ICE-85 command language is roughly equivalent to a 'word' in the English language. It consists of a string of alphanumeric characters that may be augmented by a one or two special character prefix that serves as a *token identifier*. Tokens are divided into the following types: *keywords, user-names,* and *constants*. Examples are: REGISTERS, .START, ..MODULE, GROUP, 0400, 123AH.

Keywords

ICE-85 recognizes a general class of predefined *tokens* that are fixed in the command language. They provide three functions. *Reference keywords* are used to specify locations having unique predefined functions. *Command keywords* specify command type. *Function keywords* specify subfunctions within a command. The following sections define and describe these keyword classes. Each class and associated keyword set is presented in the following paragraphs. Appendix A contains a listing of ICE-85 keywords and their abbreviations.

The reason for discussing the various classes and subsets here is to smooth the later discussions of commands, where the class-names are used to show what elements may appear in which commands.

Reference Keywords

The command language contains a set of system defined mnemonic *tokens* that are used to address system objects. Each device such as the accumulator or register B is assigned a specific mnemonic that is to be used to address and access the contents of that device. These identifiers are called *reference keywords. Reference keywords* are used in ICE-85 commands to refer to 8085 processor registers and flags, system defined channel groups, emulation registers, memory locations, and blocks, I/O ports and synchronization lines.

The total set of *reference keywords* is subdivided by types. Each type is referenced by a class name. For example, *register-name* denotes the set of 8085 8-bit registers. A reference *token* is assigned to each element within the set and is always shown in upper case. For example, 'RB' denotes the contents of the register B of the 8085 8-bit register set.

Registers

All of the registers are assigned reference keywords for addressing. The registers are composed of the following classes: 8085 8-bit registers, 8085 register pairs, 8085 1-bit status flags and 8085 interrupt bits.

Type	Class Name	Keywords
8-bit registers	*register-name*	RA,RB,RC,RD,RE,RF, RH,RL,OPCODE,CAUSE

The meta-term *register-name* is the name of the class of 8085 8-bit general purpose work registers that provide a variety of functions such as storing 8-bit data values, intermediate results of arithmetic operations, address pointers, and operation code and status data.

Type	Class Name	Keywords
16-bit registers	*register-pair-name*	RBC,RDE,RHL,SP,PC,P SW, UPPER,BUFFERSIZE,TIMER,HTIMER

The meta-term *register-pair-name* is the name of the class of 16-bit register pairs that provide such functions as storing 16-bit values, maintaining the stack and program pointers, and saving status information.

Type	Class Name	Keywords
Status flags	*flag-name*	CY,PY,ACY,Z-,SN

The meta-term *flag-name* is the class name of the set of five 1-bit condition flags that carry conditions resulting from arithmetic and logical operations.

Type	Class Name	Keywords
Interrupt bits	*i-bit-name*	M5,M6,M7,IE,I7,SID,SOD

The meta-term *i-bit-name* is the class name of the set of seven interrupt bits that are used to mask, enable/disable, and sense interrupts and to input/output serial data.

System Defined Channel Groups

Type	Class Name	Keywords
Channel group	*s-group-name*	DMUX,ADDR,ADDRL,ADDRH, DATA,STS,SD, RW,MTH, U0,U1,U2

The meta-term *s-group-name* is the class name of the set of twelve ICE-85 defined groups of external probe and 8085 processor channels that are used to set breakpoint and qualification registers.

Emulation Registers

The emulation registers consist of the breakpoint registers, qualification registers and condition registers.

Type	Class Name	Keywords
Breakpoint registers	*break-reg*	BR0,BR1,BR
Qualification registers	*qual-reg*	QR0,QR1,QR
Condition registers	*cond-reg*	CR0,CR1,CR2,CR3
GO register	*go-reg*	GR
Step register	*step-reg*	SR

The meta-term *break-reg* is the class name for the two breakpoint registers that are used to halt emulation. The meta-term *qual-reg* is the set name for the two qualification registers that control tracing. Each register is 42 bits long, one bit for each of the eighteen probe channels and twenty-four processor channels (except channel 43, MTH), and each bit can take any one of the three values 0, 1, or "don't care". The initial setting of these registers is 42 "don't care" values. The meta-term *cond-reg* is the set name for the four condition registers that are used to set the test conditions that control emulation in the step mode. The meta-term *go-reg* refers to the GO-register, an ICE-85 pseudo-register that controls the breaking of real-time emulation. The meta-term *step-reg* refers to the STEP-register, an ICE-85 pseudo-register that controls the stopping of single-stepping.

Synchronization Lines

Type	Keywords
Sync	SY0,SY1

ICE-85 can be synchronized with other ICE devices by means of two synchronization lines: SY0, which synchronizes real-time emulation and SY1, which synchronizes the trace. Both can be enabled or disabled for input or output or both.

Command Keywords

Type	Keywords
Command token	BASE,CALL,DEFINE,DISABLE ENABLE, EVALUATE,EXECUTE,EXIT,GO,GROUP,ICALL LIST,LOAD,MAP,MOVE, NEWEST,OLDEST, PRINT,REMOVE,RESET, SAVE,STEP, SUFFIX,SYMBOL,TRACE

These tokens are used as command verbs in that they specify the major function (action) to be executed as a result of the command.

Function Keywords

The function keywords are those *tokens* that are used as command modifiers. These are used as objects of commands, state or mode specifiers, specifiers of operational controls, and connectors.

Object Keywords

Type	Keywords
Modifier	ALL (print) HARDWARE (reset) DUMP (enable) STOPTRACE (enable)

These ICE-85 keywords are used as command modifiers. For example, in the command "PRINT ALL", the print command is modified to print all of the current contents of the trace buffer, as opposed to a single line, which is the default.

State or Mode Keywords

Type	Keywords
State/Mode	Y,Q,T,H,K,ASCII (base and radix) CYCLE,FRAME,INSTRUCTION (trace mode) SHARED,UNSHARED GUARDED,USER,INTELLEC (map mode) NOVERIFY IN,OUT (sync lines) NOLINES,NOSYMBOLS,NOCODE (load/save mode)

These keywords specify state or mode requirements. For example "TRACE = CYCLE" states that trace data is to be displayed in the cycle mode. "ENABLE SY0 OUT" states that the SY0 output mode is to be enabled.

Operational Control Keywords

Type	Keywords
Op Control	CALL,JUMP,RETURN (dump condition)
	FROM (GO and STEP start address)
	FOREVER (GO and STEP duration)
	COUNT (STEP duration)
	TILL (GO and STEP halting conditions)
	ON
	LOCATION,VALUE
	EXECUTED,WRITTEN,READ, INPUT,OUTPUT,HALT (Breakpoint state)

These keywords that specify operational conditions required. For example, "GO FROM 0400H TILL LOCATION 0600H READ" states that emulation shall start from memory location 0400 and will continue until memory location 0600H is read.

Connector Keywords

Type	Keywords
connector	AND (condition registers)
	OR (condition registers, break registers and synchronization lines)

These keywords connect multiple halting conditions for emulation or trace. For example, "GO FROM 0400 TILL BR1 OR SY0" states that the emulation is to start at 0400 and to continue until a match on breakpoint register 1, occurs, OR SY0 goes low.

User Names

The command language permits the programmer and operator to employ symbolic addressing through the use of user-generated tokens as opposed to system-generated tokens (keywords). The language permits four types of *user names: symbols, module names, statement numbers,* and *group names.*

Symbols

A *symbol* is a sequence of contiguous alphanumeric characters, prefixed by a period (.), that references a location in a symbol table. The *symbol* has two uses. The referenced table location always contains a number; it may be an address of an instruction or data value in a program module, or it may be used directly as a numerical value (variable). In the first case, the *symbol* is an alternative method of program addressing (symbolic as opposed to direct numeric addresses). In the second case, it provides a method for storing and retrieving data values symbolically into/from the table itself.

Symbols in the user-program symbol table may be entered, deleted, or referenced through the ICE-85 command language. *Symbols* (names) must conform to translator syntax: a sequence of one or more characters where the first character must be a let ter, an at-sign (@), or a question mark (?), and the characters following may be these or a numeral. However, ICE-85 truncates all *symbols* to 31 characters. Thus two different *symbols* must be unique in the first 31 characters. In addition, to refer to a *symbol* in ICE-85 command language, you must preface it with a period (.); *module names* require two periods (..) and *group names* must have none. Interven ing spaces are allowed between a period and the *symbol* following but are not part of the name.

Examples:

```
.VAR123.MSGXIZ..VOLUME.PRESSR.VAR666.SGNLTO
.@MYFLAG        .?WHICHPATH        .@WHATTIME?
```

The ICE-85 symbol table contains one or more modules; each has a name. When the Intellec system is initialized with ICE-85 system software, a symbol table is established in memory containing one "no-name" module with no *symbols* . Whenever a user object module is loaded, its modules are loaded subsequent to the modules currently in the symbol table. Each module will contain the programmer-specified procedure names and variable names as *symbols*.

A reference such a '.X.Y' means the first occurrence of the *symbol* Y that follows the first occurrence of *symbol* X.

One use for this type of construction is to identify one of several instances of the same *symbol-name* that actually refer to different variables because they were declared local to two different procedures in a block structured program. For example, suppose the program contains two procedures, named START and FINISH, both of which have their own variable COUNT. Note that START and FINISH are not *module names*. Now, you can reference the two instances of COUNT with the two constructions.

```
.START.COUNT
.FINISH.COUNT
```

The keywords BYTE and WORD can precede a reference such as .X.Y, to obtain the contents of the symbolic reference.

You can define and add *symbols* either to the no-name table or to any module symbol table currently in memory.

Module Names

ICE-85 permits multiple modules to reside in memory simultaneously, including the 'no-name' module. Moreover, the same *symbol* is permitted to occur in any or all of the modules currently in memory. Therefore it may be necessary to prefix a *symbol* with a *module name* to refer to the occurrence of that *symbol* in one particular named module. If no *module name* is present, ICE-85 scans all modules, starting with the no-name module and proceeding through the list of modules in the order in which they were entered into memory. The first occurrence of the symbol is selected.

When used, the *module name* is prefixed to the *symbol*. The *module name* conforms to the same PL/M syntax as a *symbol* except that it is prefixed with a double period (..).

As an example, consider the *symbol* .BEGIN in module ..MAINLOOP. The entire reference to this occurrence of .BEGIN is:

```
..MAINLOOP.BEGIN
```

Statement Numbers

In the process of compiling a source module, the PL/M compiler generates a set of (source) *statement numbers*, one for each source statement in the module. Each *statement number* is linked to the absolute address of the first instruction generated by the PL/M compiler for the associated source statement in the source program. Each compiled program will contain a table of *statement numbers* and absolute addresses. Items (addresses) in the table are referenced by entering the associated *statement number*.

The form of reference is

module-name # decimal-10

where:

> *#* is the 'number' sign; this designates the reference as a *statement number*.
>
> *decimal-10* is the (source) *statement number* (a numeric constant). The default suffix of *decimal-10* is always decimal.

For example:

```
..MAINLOOP#123
```

#123 is *statement number* #123 in the source program, ..MAINLOOP. This reference would obtain the *address* of the first instruction generated by source statement 123 of module ..MAINLOOP.

Statement numbers are an alternative to program addressing, as opposed to labels in the program.

User Group Names

ICE-85 allows the user to define a set of symbolically-named groups to reference subsets of the twenty-four 8085 processor channels and eighteen external probe channels when setting breakpoints and clock qualifiers, and when examining trace data.

The 18 external probe channels are numbered 1 to 18, and the 24 processor channels are numbered 19 to 42.

For example, if external probes 8, 11, 5, 10, 1, 7, 15, and 2 are attached to port A of in your system, you can define a group PORT$A to contain these probes. Then, when setting a breakpoint to match on a particular value on port A, when displaying the trace data from port A, etc., you can use the *group-name* PORT$A.

The same channel may appear in any number of different groups, but a channel may not appear more than once within a single group. All groups are limited to a maximum of 16 channels each.

A set of twelve groups are predefined by ICE-85, grouping the external probe and 8085 processor channels.

Constants

A *constant* is a token that represents a fixed numerical value (e.g., 24). ICE-85 recognizes two types of *constants: numeric constants* and *masked constants*.

Numeric Constants

A *numeric constant* consists of a sequence of numeric characters (digits) and optionally a suffix to specify the constant's radix. A *numeric constant* has the general form:

digit(s) [radix]

where:

digit(s) : are a string of one or more valid numeric digits (0123456789ABCDEF). Legal digits for a given *numeric constant* depend upon the radix specified.

radix : denotes the number base or representation for entering and displaying numeric values. The following are the valid ICE-85 radix indicators:

Radix	*Number Base*
Y	Binary (base 2)
Q	Octal (base 8)
T	Decimal (base 10)
H	Hexadecimal (base 16)
K	Decimal multiple of 1024

*The letter O can be substituted for the letter Q to denote octal.

Examples of valid numeric constants:

```
10010011Y
737Q
2049T
1FA9H
2049   (using a radix previously specified)
```

Masked Constants

A *masked constant* is syntactically identical to a *numeric constant* except it may not contain the "T" or "K" suffixes and must contain one or more "X" characters. Each "X" character represents a 'don't care' digit (1, 3, or 4 bits depending upon whether the *radix* is binary, octal, or hexadecimal). The *radix*, either explicit or implicit (i.e., previously specified), must be binary, octal or hexadecimal. The following are examples of *masked constants* :

10X1X01Y	(binary - 2 don't care bits shown explicitly)
3X4Q	(octal - 3 don't care bits implicit because each octal numeral represents 3 bits; equivalent to 011XXX100Y)
6FX1H	(hexadecimal - 4 don't care bits, implicit because each hexadecimal numerical represents 4 bits; equivalent to 01101111XXXX0001Y)

Typical use of a *masked constant* is to check for a match on specifically relevant bit positions.

When the *masked constant* is applied in a match, the zero and ones are checked, but the don't-care bits are ignored. If all the corresponding zeros and ones in the two numbers match, the match is successful.

Special Tokens

The command language contains two *special token* sets that provide special functions: *operators* and *punctuation.*

Operators

Type	Class Name	Operators
relational	*rel-op*	=, <, >, <=, >=, <>
plus	*plus-op*	+, –, (binary and unary)
mult	*mult-op*	*, /, MOD

Punctuation

Type	Class Name	Punctuation Characters
punctuation	*punct-op*	' & . , ;) ($ CR LF SP

The use of punctuation characters are defined in those sections that define command formats.

Numeric Expressions

A numeric expression is a construct of primaries and operators that evaluates to a numeric value. ICE-85 also makes use of *conditional* expressions that evaluate to 'true' or 'false'. However, when the term 'expression' is used with no qualifier in this manual, a numeric expression is intended.

ICE-85 evaluates in a left-to-right scan modified by operator precedence for an algebraic sequence in the form:

operand [*operator operand*]....

Operators

Operators are used in expressions and in commands. A summary of ICE-85 *operators* are shown below. The *binary (arithmetic) operators* are listed in their order of precedence from highest precedence to lowest.

Type	Operator	Interpretation
Precedence	( )	Controls order of evaluation
Binary (Arithmetic)	MOD	Modulo remainder
	*	Multiplication
	/	Division
	+	Addition
	–	Subtraction
	MASK	Bitwise AND
Relational	=	Is equal to
	>	Is greater than
	<	Is less than
	<>	Is not equal to
	>=	Is greater than or equal to
	<=	Is less than or equal to
Unary	+	Single positive quantity
	–	Negative quantity

Operands

Operands are numerical values, or references that are evaluated to numerical values. In an *operand*, all the bits in the machine (binary) version are set to zero or one, giving a unique value. *Masked constants* , however, have one or more bits set to 'don't care' (represented by X), and thus represent a multivalued entity.

An *operand* has the general form:

```
[unary-op] | (exp)                  |
           | numeric constant       |
           | statement number       |
           | [module-name] symbol   |
           | register name          |
           | pair-name              |
           | flag-name              |
           | i-bit-name             |
           | port-name              |
           | mem-desig address      |
```

where:

> (exp): an *operand* whose value is the value of the parenthesized expression e.g. (1+2+3) = 6 (operand value).

Expressions

An previously stated, every *expression* is of the form:

operand [operator operand]...

where each *operand* can be any type such as a *numeric constant, symbol, register name* , etc.

The following examples illustrate typical *expressions*. The *operands* are indicated by upper case *alphabetic characters* (A, B, C, ...) and can be of any valid *operand* type. These examples also illustrate the ICE-85 evaluation of each *expression*. The evaluation is accomplished by a left-to-right scan augmented by *operator* precedence. Each *expression* enclosed in parenthesis is treated as an *(exp)* type *operand* and is evaluated at that point. The examples show the resolution of each *expression* into a single *operand* value, R (resultant).

1. A+B*C → A+D → R
2. (A+B)*C → D*C → R
3. A+B*C/D → A+E/D → A+F → R
4. (A+B)*C/D → E*C/D → F/D → R
5. A+B*C/D MOD E → A+F/D MOD E
   → A+G MOD E
   → A+H → R
6. (A+B)*C/D MOD E → F*C/D MOD E
   → G/D MOD E
   → MOD E → R
7. A+B*C/ MOD E MASK F
   → A+G/D MOD E MASK F
   → A+I MASK F
   → J MASK F
   → R

8. (A+B)*C/D MOD E MASK F
   → G*C/D MOD E MASK F
   → H/D MOD E MASK F
   → I MOD E MASK F
   → J MASK F
   → R

Meta-Notation Used In The Manual

This manual employs a set of meta-notational symbols and conventions to describe the structure of commands and other language constructs. Items 1 and 2 below specify the use of meta-terms and tokens respectively. The features of this meta-notation system are as follows:

1. A lower-case italicized entry in the description of a command is the meta-term for a set or class of tokens. To create an actual operable command, you must enter a particular member of this class. For example, the lower-case entry:

   break-register

   means that the command will accept any of 3 tokens: BR0, BR1, or BR (which means BR0 and BR1)

   Each class-name given in the syntax description of a command is explained in the discusion of semantics that accompanies the command.

2. An upper-case entry is a token that must be used literally as given. A valid abbreviation of that token may substitute for the full token as given. The token may be a command word, or it may be a particular member of a class of references. For example, the upper-case entry:

   DEFINE

   is a command word that must be used as given unless abreviated. The abbreviation DEF may be used in place of DEFINE. As another example, the upper-case entry:

   BR1

   means breakpoint register 1 must be named as and where given.

3. A single-required entry is shown without any enclosures, whereas free options or choices among restricted alternatives are specially denoted. For example, in the following command syntax:

   GROUP [*group-name*]

   the token GROUP is required. The significance of the brackets around *group-name* is explained in (5) below.

4. Where only one entry must be selected from a menu of two or more items, the choices for the required entry are given in a vertical arrangement enclosed by two vertical lines. For example:

   ```
   TRACE =  | FRAME       |
            | CYCLE       |
            | INSTRUCTION |
   ```

   Indicates that FRAME or CYCLE or INSTRUCTION must be selected; the tokens TRACE and = are required as given.

5. An optional entry is enclosed in brackets [ ]. Where a choice exists for an optional entry, the choices are given in a vertical arrangement enclosed in brackets. For example, in the command:

   GROUP [*group-name*]

   the entry *group-name* (signifying the class of channel-group names) is optional. Omitting an optional entry usually alters the operation of the command, as explained under each command description.

6. A group of inclusive choices is given in vertical arrangement and enclosed in braces $\{\}$. "Inclusive" means that more than one of the items can be entered in the same command, and items can appear in any order; no item can be used twice. The menu of inclusive items represent an optional entry or entries. For example:

   $$\text{ENABLE DUMP} \left\{ \begin{array}{l} \textit{partition} \\ \text{CALL} \\ \text{JUMP} \\ \text{RETURN} \end{array} \right\}$$

   This notation indicates that none, one, or more that one choice of *partition*, CALL, JUMP, and/or RETURN may be included in one command; if more that one is used, the entries can be in any order.

   To complete the example:

   ENABLE DUMP RETURN CALL JUMP

   is a valid command.

7. Where two or more choices are mutually exclusive, and as a group they constitute one choice within an arrangement of inclusive entries, the mutually exclusive items are given on one line separated by slashes (/). For example:

   $$\text{SAVE pathname} \left\{ \begin{array}{l} \text{NOCODE / partition} \\ \text{NOSYMBOLS} \\ \text{NOLINES} \end{array} \right\}$$

   This example indicates that the optional entries are the items NOLINES, NOSYMBOLS, and *either* NOCODE *or* a memory partition; the entries can appear in any order. NOCODE and *partition* are mutually exclusive.

8. Where mutually exclusive entries can be shown on one line, the following shorthand notation can be used.

   SUFFIX = Y/Q/T/H

   This example is equivalent to

   $$\text{SUFFIX} = \left| \begin{array}{c} \text{Y} \\ \text{Q} \\ \text{T} \\ \text{H} \end{array} \right|$$

9. Where an entry can be *repeated* indefinitely at the user's option, the syntax is notated by enclosing the repeatable entry in brackets [ ] followed by three periods (...). For example,

   operand [*operator operand*]...

   indicates that *operator operand* can be repeated as many times as desired.

10. Some entries can be repeated up to a fixed maximum number of repeats, rather than indefinitely. The repeatable entries are enclosed in brackets [ ] followed by ...& n,

    where:

    &: denotes that the entries in the brackets may be repeated.
    n: a decimal number that specifies the number of times the entries enclosed within the brackets can be repeated.

    In addition, some repeatable entries require a particular item separator to precede each repeat after the first entry. When an item separator is required, it is also enclosed in the brackets, preceding the repeatable entry. Note that the maximum number of entries is one more than the number of repeats given by the number, n.

    Example:

    $$\text{TILL } \textit{cond-reg} \left[\left| \begin{matrix} \text{AND} \\ \text{OR} \end{matrix} \right| \textit{cond-reg} \right] \text{...\&3}$$

    In this example, up to four members of the class *cond-reg* can be named; each of the second, third, and fourth registers must be preceded by a token AND or OR.

CHAPTER 5
THE ICE-85 COMMAND LANGUAGE

Chapter 5 contains discussions, examples, and syntax summaries for each of the ICE-85 commands. The chapter begins with a summary of information on entering commands at the console. The commands have been classified into eleven sections for discussion; commands in a given section are related in function.

The following brief outline of Chapter 5 shows how the ICE-85 commands have been classified for discussion.

Utility Commands Involving ISIS-II

- Discussion
- ICE85 Command
- EXIT Command
- LOAD Command
- SAVE Command
- LIST Command

Number Bases and Radix Commands

- Discussion
- SUFFIX Command
- BASE Command
- EVALUATE Command

Memory and I/O Port Mapping Commands

- Discussion
- MAP Mode Command
- MAP MEMORY Command
- MAP IO Command
- Display MAP Status Command
- RESET MAP Command

Hardware Register Commands

- Discussion
- Display Processor or Status Register Command
- Set 8085 Processor Register Command
- RESET HARDWARE Command
- ENABLE/DISABLE TIMEOUT Command

Memory and Port Content Commands

- Discussion
- Display Memory and Port Contents Commands
- Set Memory Contents Commands
- Set Input/Output Port Contents Command

Symbol Table and Statement-Number Table Commands

- Discussion
- Display Symbol Table or Statement-Number Table Commands
- DEFINE Symbol Command
- Change Symbol Command
- REMOVE Symbol Command

Channel Group Commands

Discussion
DEFINE GROUP Command
Display GROUP Command
Change GROUP Command
REMOVE GROUP Command

Real-Time Emulation Control Commands

Discussion
GO Command
GR Command
Set Breakpoint Register Command
RESET Breakpoint Register Command
Display Real-time Emulation Controls Commands
ENABLE/DISABLE SY0 OUT Commands

Trace Control Commands

Discussion
Set Trace Display Mode Command
MOVE, OLDEST, and NEWEST Commands
PRINT Command
Set Qualifier Register Command
RESET Qualifier Register Command
Display Trace Controls Commands
ENABLE/DISABLE Trace Factors Commands

Single-Step Emulation Control Commands

Discussion
Set Condition-Register Command
SR Command
STEP Command
Display Single-Step Controls Commands
ENABLE/DISABLE DUMP Command

External Call Commands

Discussion
CALL Command
ICALL Command
EXECUTE Command

Table 5-1 lists the commands separately, in alphabetic order by command title. You can use this table to find the section containing any command or command function that you wish to locate.

Table 5-1. ICE-85 Commands: Alphabetic Summary

Command Title	Operations Included in Command	Section where discussed; page of syntax summary.
Change GROUP	Change channels assigned to previously-defined user group.	Channel Group Commands; page 5-60.
Change Symbol	Change address or variable value corresponding to a previously-defined symbol.	Symbol Table and Statement-Number Table Commands; page 5-54.
DEFINE GROUP	Define a group-name and assign a list of channels to it.	Channel Group Commands; page 5-59.
DEFINE Symbol	Define a symbol and assign it a corresponding value (typically an address).	Symbol Table and Statement-Number Table Commands; page 5-53.
Display Emulation Registers	Display the current settings of the GO-register, breakpoint registers, STEP-register, and condition registers.	Real-Time Emulation Control Commands; page 5-73. Single-Step Emulation Commands; page 5-99.
Display GROUP	Display channels assigned to one or more group names.	Channel Group Commands; page 5-60.
Display MAP Status	Display memory or I/O port mapping.	Memory and I/O Port Mapping Commands; page 5-30.
Display Memory and Port Contents	Display contents of one or more memory adresses or an I/O port.	Memory and Port Contents Commands; page 5-46.
Display Processor or Status Register	Display contents of 8085 and ICE-85 hardware registers.	Hardware Register Commands; page 5-36.
Display Radix	Display current input or output radix.	Number Bases and Radix Commands; page 5-15.
Display Symbol Table or Statement-Number Table	Display address or value from symbol table corresponding to a user-defined symbol. Display address corresponding to source program line number.	Symbol Table and Statement-Number Table Commands; page 5-53.
Display Trace Controls	Display current TRACE mode. Display qualification register setting.	Trace Control Commands; page 5-88.

Table 5-1. ICE-85 Commands: Alphabetic Summary (Continued)

Command Title	Operations Included in Command	Section where discussed; page of syntax summary.
ENABLE/ DISABLE	Enable or disable SY0 OUT, SY1 IN, SY1 OUT, STOPTRACE, or TIMEOUT	Hardware Register Commands; page 5-38. Emulation Control Commands; page 5-74. Trace Control Commands; page 5-88.
ENABLE/ DISABLE DUMP	Enable or disable automatic trace display after STEP.	Single-Step Emulation Control Commands; page 5-100.
EVALUATE	Evaluate and display the number entered, in the five bases Y, Q, T, H, and ASCII.	Number Base and Radix Commands; page 5-16.
EXIT	Terminate emulation session, return control to ISIS.	Utility Commands Involving ISIS-II; page 5-9.
External Call	Emulate or execute an external procedure.	External Call Commands; page 5-103.
GO	Begin emulation, accepting default halting and starting conditions, or specifying new ones.	Real-Time Emulation Control Commands; page 5-71.
GR	Set conditions for halting emulation.	Real-Time Emulation Control Commands; page 5-72.
ICE85	Load ICE-85 software into ISIS-II, transfer control to ICE-85 program.	Utility Commands Involving ISIS-II; page 5-9.
LIST	Send ICE-85 output to designated external device.	Utility Commands Involving ISIS-II; page 5-11.
LOAD	Load user program into ICE-85.	Utility Commands Involving ISIS-II; page 5-10.
MAP	Assign memory blocks or I/O port segments to user system or to Intellec system.	Memory and Port Mapping Commands; page 5-28, 5-29.
MAP Mode	Set MAP mode as SHARED or UNSHARED.	Memory and Port Mapping Commands; page 5-27.
MOVE, OLDEST NEWEST	Move trace buffer pointer to desired position.	Trace Controls Commands; page 5-85.
PRINT	Display trace data.	Trace Controls Commands; page 5-86.

Table 5-1. ICE-85 Commands: Alphabetic Summary (Continued)

Command Title	Operations Included in Command	Section where discussed; page of syntax summary.
REMOVE GROUP	Delete one or more group names.	Channel Group Commands; page 5-61.
REMOVE Symbol	Delete one or more user-defined symbols from the ICE-85 symbol table.	Symbol Table and Statement-Number Table Commands; page 5-54.
RESET	Restore breakpoint registers, qualification registers, MAP, ICE-85 HARDWARE, or RST 7.5 line to an initial state.	Hardware Registers Commands; page 5-37. Memory and Port Mapping Commands; page 5-30. Real-Time Emulation control Commands; page 5-74. Trace Control Commands; page 5-87.
SAVE	Save user program on external device.	Utility Commands Involving ISIS-II; page 5-11.
Set Breakpoint Register	Set designated breakpoint register to condition for halting real-time emulation.	Real-Time Emulation Control Commands; page 5-73.
Set Condition Register	Set condition register to condition for halting single- step emulation.	Single-Step Emulation Control Commands; page 5-96.
Set Input/Output Port Contents	Set contents of one I/O port.	Memory and Port Contents Commands; page 5-48.
Set Memory Contents	Set the contents of one or more memory locations.	Memory and Port Contents Commands; page 5-47.
Set Processor Register	Set content of 8085 register.	Hardware Register Commands; page 5-37.
Set Qualification Register	Set qualification register to condition for enabling trace data collection.	Trace Controls Commands; page 5-87.
Set Radix	Set default input or output radix.	Number Base and Radix Commands; page 5-15.
Set TRACE Mode	Set mode for display of trace data.	Trace Control Commands; page 5-84.
SR	Set STEP-register with halting conditions for single-step emulation.	Single-Step Emulation Control Commands; page 5-97.
STEP	Begin single-step emulation, setting start and halt conditions.	Single-Step Emulation Control Commands; page 5-98.

Entering Commands At The Console

ICE-85 displays an asterisk prompt (*) at the left margin when it is ready to accept a command from the console.

Each command is entered as a *command line*. The command line consists of one or more *input lines;* the length of an input line is limited to the number of characters that one line of the console display can contain.

ICE-85 recognizes the carriage return as the terminator for a command line. If it is necessary to use more than one input line to enter a command, each intermediate input line should end with an ampersand (&). When ICE-85 encounters the ampersand, it suppresses the interpretation of the command that would occur on encountering the carriage return that follows. After the carriage return is executed, ICE-85 displays a double asterisk prompt (**) to acknowledge the continuation of the command line.

Tokens in the command are separated by blanks, unless the construct requires another form of separator. For example, tokens in a list are separated by commas; in this case, blanks may be inserted for clarity but are not required.

Any input line may include comments. A semicolon (;) precedes the comments. The comments must appear after any portion of the command that is on that input line; in other words, if the first character in an input line is a semicolon (;), the entire input line must consist of comments. Characters in a comment are not interpreted by ICE-85, and are not stored internally. The main use of comments is to document an emulation session while it is in progress.

Comments may not be continued from input line to input line. If an ampersand is used to continue a command line that also contains comments, the ampersand must come before the comment. An ampersand that is embedded in a comment is ignored by ICE-85.

You can use ISIS-II editing capabilities to correct errors in the current input line. The line-editing characters are as follows.

Characters	Result
RUBOUT	Delete last character entered in input line. The deleted character is echoed immediately. The RUBOUT function can be repeated, deleting one character each time it is pressed.
CTRL X	Delete entire input line. (CRTL Z gives the same result.)
CTRL R	Display entire input line as entered so far. This is useful after a RUBOUT, to review which characters have been deleted.
ESC	Cancel entire command being entered.
CRTL P	Input next character literally.
Carriage Return	Terminate input line or command line.
Line Feed	Terminate input line.

Once a line terminator (carriage return or line feed) has been entered, that line can no longer be edited.

The dollar sign ($) is ignored by ICE-85. You can use it as a separator when you want to combine two words into one token. For example, suppose you wanted to combine the two system groups DATA and STS into one group for your use. Instead of DATAANDSTS, you can use the $ character as a separator: DATA$AND$STS.

Utility Commands Involving ISIS-II

The Intel Systems Implementation Supervisor (ISIS-II) is the diskette operating system for the Intellec Microcomputer Development System. ICE-85 runs under ISIS-II control, and can call upon ISIS-II for file management functions.

The following commands are included in this section:

Command	Purpose	Page
ICE85	Load ICE-85 program from diskette.	5-9
EXIT	Return control to ISIS-II.	5-9
LOAD	Load user program into memory accessed by ICE-85.	5-10
SAVE	Copy user program from memory onto diskette.	5-11
LIST	Copy ICE-85 emulation output to printer or file.	5-11

Discussion

ICE-85 commands can use ISIS-II *pathnames* to direct ISIS-II to a desired diskette file or other output device.

For diskette files, the format of *pathname* is as follows.

:drive:filename

The entry *:drive:* stands for one of the references to ISIS diskette drives, as follows.

:F0:	Diskette drive 0
:F1:	Diskette drive 1
:F2:	Diskette drive 2
:F3:	Diskette drive 3
:F4: :F5:	Single-density drives on a double-density system

The entry *filename* must follow the second colon (after *drive*) without any intervening spaces. A filename has the following components.

identifier [. *extension*]

The entry *identifier* is a name assigned by the user, and is made up of one to six alphanumeric characters. The *extension* is an optional part of the filename, consisting of one to three alphanumeric characters preceded by a single period. The extension must be used if it is present in the directory listing of the file on the diskette. If used, the extension follows the identifier without any spaces. Some extensions (e.g., .BAK, .LNK) are assigned by system processors; others can be assigned at the desire of the user. An extension provides a second level of file identification; it can be used to identify different versions of the same program, or to give supplemental information about the file (e.g., author, data, version).

Fully compiled or assembled programs ready to run (emulate) do not have system-assigned extensions, although they may have extensions assigned by the user.

For devices other than diskette files, the format of *pathname* is as follows.

:device:

The following devices are commonly accessed in ICE-85 commands.

:Device:	*Output Device*
:LP:	Line printer
:HP:	High-speed tape punch
:TO:	Teletypewriter printer
:CO:	Console display

For more information on ISIS-II filenames and device codes, refer to the *ISIS-II System User's Guide*.

The ICE85 command, entered after an ISIS-II prompt, directs ISIS-II to load the ICE-85 program from the specified diskette drive, into a reserved area in Intellec memory. ICE-85 begins operation as soon as it is loaded, initializing its hardware and program variables, and signaling readiness to accept ICE-85 commands by displaying an asterisk prompt.

The EXIT command ends the emulation session and returns control to ISIS-II. The command halts all emulations and issues a hardware reset before exiting.

The LOAD command loads the object code from the named file and drive into the areas of memory specified by the memory map. Modules are loaded in the order of their appearance in the source file. If no qualifiers are included, the command also loads the program symbol table and statement number table (for PL/M-80 programs) along with the program code. The symbols and statement numbers are placed in reserved areas of Intellec Shared memory. Symbols are grouped in the tables by module, in the order that the modules are loaded into memory.

The command can include one or more modifiers to control what is to be loaded. If NOCODE is included, the program code is omitted from the load. If NOLINE is included, the program statement number table is not loaded. If NOSYMBOL is included, the program symbol table is not loaded. Any combination of one, two, or three modifiers may be included, although the command with all three modifiers represents a "null" command. No modifier may be named twice in the same load command.

The SAVE command copies the user program currently loaded from memory onto the specified file and drive. If the diskette installed on the given drive does not have the named file in its directory, ISIS-II creates the file and opens it for write. If no explicit drive number is given, drive 0 is assumed.

The command can include one or more modifiers to control what is to be saved. If NOSYMBOL is included, the symbol table is not copied from memory to diskette. If NOLINE is included, the statement number table (for PL/M-80 programs) is not saved. The modifiers NOCODE and *partition* are mutually exclusive: if one is used, the other may not be included. If NOCODE is included, the program object code is not copied to diskette. If *partition* is included, only the code stored in the memory addresses in the partition (range of addresses) are saved. When more than one modifier is used, separate them with spaces. No modifier may be used twice in the same SAVE command. The SAVE operation does not alter the program code, symbol table, or statement number table in memory.

The LIST command saves a record of the emulation session, including high-volume data such as trace data, on a hard-copy device or on a diskette file. Only one device or file other than the console can be specified (active) at a given time.

The initial device is :CO:, output to the console. Other devices that can be specified are a line printer (:LP:), high-speed paper tape punch (:HP:), and teletypewriter printer (:TO:).

Instead of a hard-copy device, a diskette file can be specified. If the output is to a diskette file, the file is opened when the LIST command is invoked, and output is stored from the beginning of the file, writing over any existing data. Specifying a new file or device in a later LIST command closes any existing open file and avoids over-writing any more data.

When LIST is in effect (with a device or file other than :CO:), all output from ICE-85, including system prompts, commands, and error messages, is sent both to the named device or file and to the console display. To restore output to the console only (no other device), use the command LIST :CO:.

One additional command, the EXECUTE command, uses ISIS-II pathnames in its syntax. This command is discussed under the External Call Commands, page 5-101.

ICE85 Command

*:drive:*ICE85

Example:

```
:F1:ICE85
```

:drive:	The number of the diskette drive containing the ICE-85 software diskette. The number is preceded by the letter F, and enclosed in colons.
ICE85	The name of the ICE-85 program file under ISIS-II. The filename follows the second colon without any intervening spaces.

EXIT Command

EXIT

Example:

```
EXIT
```

EXIT	A command keyword that returns control from ICE-85 to ISIS-II.

LOAD Command

```
LOAD :drive: filename  ⎧ NOCODE   ⎫
                       ⎨ NOSYMBOL ⎬
                       ⎩ NOLINE   ⎭
```

Examples:

```
LOAD :F0:TEST.VR1
LOAD :F1:MYPROG NOLINE
LOAD :F2:COUNT.ONE NOCODE NOLINE
LOAD :F3:NEWCOD NOSYMBOL
```

LOAD	A command keyword that loads the software on the given file and drive into the combination of prototype and Intellec memory specified by a previous MAP command.
: *drive*:	The diskette drive (:F0:, :F1:, :F2:, or :F3:) that contains the target file. If no drive is given, :F0: (drive 0) is the default.
filename	The name of the desired program as compiled or assembled, linked, and located. The filename follows the second colon with no intervening spaces.
NOCODE	A modifier specifying that program code is not to be loaded.
NOSYMBOL	A modifier specifying that the program symbol table is not to be loaded.
NOLINE	A modifier specifying that the program line number table (for PL/M-80 programs) is not to be loaded.

SAVE Command

```
SAVE : drive: filename  ⎧ NOCODE / partition ⎫
                        ⎨ NOSYMBOL           ⎬
                        ⎩ NOLINE             ⎭
```

Examples:

```
SAVE :F1:TEST
SAVE :F0:MYPROG 0800 TO 0FFF NOLINE
SAVE :F2:COUNT.TWO NOLINE NOSYMBOL
SAVE :F3:NEWSYM NOCODE NOLINE
```

SAVE	The command keyword that directs ICE-85 to write the designated software elements to the indicated file and drive.
: *drive*:	The diskette drive (:F0:, :F1:, :F2:, :F3:) holding the diskette that is to contain the saved software. If no explicit drive number is given, drive 0 is the default.
filename	The name of the file that is to receive the saved information. The name of the file, including the extension if present, must follow the rules for naming files under ISIS-II. The filename immediately follows the second colon. If the filename does not exist on the designated diskette, ISIS-II creates the file and opens it for write.
NOCODE	A modifier specifying that program code is not to be saved.
partition	A construct specifying a range of one or more contiguous locations in memory; the contents of the specified locations are saved, but code in other locations is not copied. Refer to the discussion under the Memory Contents Commands (page 5-39) for the forms used to specify memory partitions.
NOSYMBOL	A modifier specifying that the symbol table is not to be saved to diskette.
NOLINE	A modifier specifying that the line number table (for PL/M-80 programs) is not to be saved.

LIST Command

```
(a)    LIST :device:

(b)    LIST :drive:filename
```

Examples:

```
LIST :LP:
LIST :CO:
LIST :F1:ICEFIL
```

LIST	The command keyword directing all ICE-85 output to be sent to the specified device or file, and to the console.
: *device*:	An ISIS-II device code, indicating a hard-copy output device to receive the output.
: *drive*:	The diskette drive holding the diskette on which output is to be written. If no explicit drive is given, drive 0 is assumed.
filename	The name of the file on the target diskette. The filename immediately follows the second colon, without intervening spaces.

Number Bases and Radix Commands

ICE-85 commands and displays involve several different number bases (*radixes*). This section describes the various radixes used by ICE-85, and the commands used to control some of them. Most radixes are set by ICE-85 and cannot be changed, but others are under your control.

This section gives details on the following commands.

Command	Purpose	Page
SUFFIX	Set or display console input radix.	5-15
BASE	Set or display console output radix.	5-15
EVALUATE	Display numeric constant or expression in all five possible output radixes.	5-16

Discussion

The commands given in detail in this section refer to the radixes used for console input and console output. In addition to these two main contexts, this section reviews the radixes used for the display of trace data and for the display of breakpoint and qualifier register settings.

Console Input Radixes; SUFFIX Command

Any number entered from the console can include an *explicit* input radix. An explicit input radix consists of one of the following alphabet characters appended directly to the number as entered.

Explicit Radix	Example	Radix Specified
Y	1001Y	Binary (base 2)
Q	11Q	Octal (base 8)
T	9T	Decimal (base 10)
H	9H	Hexadecimal (base 16)
K	2K	Multiple of 1024

The *implicit* input radix is the base used by ICE-85 to interpret numbers entered from the console without an explicit radix. The initial implicit radix for most numbers is hexadecimal (H).

To display the current implicit input radix, enter the command token SUFFIX followed by a carriage return. The implicit input radix can be Y, Q, T, or H, as defined earlier.

You can change the implicit input radix by entering a command with the form SUFFIX = X, where X is any of the characters Y, Q, T, or H. This SUFFIX command can be used where several numbers are to be entered in the same radix.

Note that K (multiple of 1024) cannot be specified as an implicit input radix. Numbers with explicit radix K are used mainly to refer to blocks of memory addresses for mapping; see the MAP commands for examples.

For some kinds of entries from the console, the implicit input radix is always decimal (T). Entries with implicit decimal radix are:

- Numbers entered after MOVE, PRINT, and COUNT keywords.
- Channel numbers.
- Program statement numbers.

An explicit radix always takes precedence over the implicit radix. If the digits in the number entered cannot be interpreted in either the explicit or the implicit radix, an error message is displayed.

Console Output Radixes; BASE Command

Numeric information such as addresses, memory and register contents, and data, is displayed in the current console output radix. The console output radix can be one of the following.

Output Radix	Radix Specified
Y	Binary
Q	Octal
T	Decimal
H	Hexadecimal
ASCII	ASCII character for each byte

The initial output radix is hexadecimal (H).

To display the current console output radix, enter the command token BASE followed by carriage return. The display consists of a single character, Y, Q, T, H, or A (for ASCII).

You can change the console output radix by entering a command with the form BASE = X, where X is one of the single characters Y, Q, T, or H, or the token ASCII. Once the radix is set with a BASE command, it stays in effect until another BASE command is entered.

The EVALUATE Command

The EVALUATE command handles the arithmetic computation involved in translating from one radix to another. This command has the form EVALUATE X, where X is any numeric constant or numeric expression. Upon receiving this command, ICE-85 evaluates any expression to a single number, and displays the result in the four bases Y, Q, T, and H, and the corresponding ASCII characters, all on one line. For ASCII, the characters are enclosed in single quotes ('); printing characters are displayed (ASCII codes 20H through 7EH after bit 7 is masked off), while non-printing characters are suppressed.

Here are three examples of the use of the EVALUATE command, with the display produced by each one.

Example 1:

```
EVALUATE 123T
```

Display:

```
1111011Y     173Q     123T     7BH     '['
```

Example 2:

```
EVALUATE 4142H
```

Display:

```
100000101000010Y    40502Q    16706T    4142H    'AB'
```

Example 3:

```
EVALUATE FFH + 256T
```

Display:

```
111111111Y    777Q    511T    1FFH    ' '
```

Note that the addition was performed first, then the result was displayed in the four bases. The result contained only non-printing ASCII characters, displayed as empty quotes.

Radixes Used in Trace Displays

Entries in the trace buffer are displayed in groups. Some groups are system-defined; the radixes used for displaying these groups are also set by the system. If you define any additional groups, they are also displayed as part of trace data, and you can specify the group-radix to be used for each such group. Refer to the Group Commands for more detail on system groups and user-defined groups.

System groups are displayed with the following radixes.

System Group	Trace Display Radix
ADDR, ADDRL, ADDRH, DATA	H (Hexadecimal)
STS	One-letter mnemonic (H=HALT, W=WRITTEN, R=READ, E=EXECUTED, O=OUTPUT, I=INPUT)
DMUX, SD, RW, MTH	Y (Binary)

The group-radix to be used for displaying a user-defined group in trace data is specified by the IN-clause that is part of the DEFINE GROUP command. Group-radixes can be one of the following.

Y	Binary
Q	Octal
T	Decimal
H	Hexadecimal

The default when no IN-clause is used is H (hexadecimal).

Radixes Used for Displaying Breakpoint and Qualifier Settings

When you enter a token representing one of the ICE-85 breakpoint or qualifier registers, the setting of that register is displayed. The display includes any system group or user-defined group having one or more care bits set; the setting of each group is displayed as a binary masked or numeric constant. Refer to the Emulation Commands, and to the Trace Commands for more details on breakpoint and qualifier registers.

Set or Display Console Input Radix Commands

```
SUFFIX

SUFFIX  =  |Y / Q / T / H|
```

Examples:

```
SUFFIX

SUFFIX  =  Y
```

SUFFIX	A command keyword referring to the implicit console input radix. The token SUFFIX alone displays the current setting (Y, Q, T, or H).
=	The assignment operator, indicating that the new setting is to follow.
Y	Binary radix.
Q	Octal radix.
T	Decimal radix.
H	Hexadecimal radix.

Set or Display Console Output Radix Commands

```
BASE

BASE  = |Y / Q / T / H / ASCII|
```

Examples:

```
BASE

BASE  =  Q
```

BASE	A command keyword referring to the system console output radix. The token BASE alone displays the current setting (Y, Q, T, H, or A).
=	The assignment operator, indicating that the new setting is to follow.
Y	Binary radix.
Q	Octal radix.
T	Decimal radix.
H	Hexadecimal radix.
ASCII	Each byte represented by its corresponding ASCII character, without separators.

EVALUATE Command

```
EVALUATE numeric
```

Examples:

```
EVALUATE 123T

EVALUATE 4142H

EVALUATE FFH + 256T
```

EVALUATE	A command keyword that directs ICE-85 to evaluate the expression and display the result in all four number bases and ASCII.
numeric	A *numeric expression* or *numeric constant*.

Memory and I/O Port Mapping Commands

The commands in this section control the ICE-85 memory and I/O port maps. ICE-85 uses these maps to determine what memory and I/O are installed on the prototype system, and what memory and I/O resources are being "borrowed" from the Intellec system for testing purposes.

This section gives details on the following commands.

Command	Purpose	Page
Map Mode	Identify Intellec memory to be borrowed as SHARED or UNSHARED.	5-27
MAP Memory	Assign up to 32 blocks of 2K locations per block segment to USER, INTELLEC, or GUARDED status.	5-28
MAP IO Ports	Assign up to 32 segments of eight I/O ports per segment to USER, INTELLEC, or GUARDED status.	5-29
Display MAP Status	Display status of one or more memory blocks, or of one or more port segments, and display the current map mode for Intellec memory.	5-30
RESET MAP	Restore Intellec map mode to SHARED, and the status of all memory blocks and port segments to GUARDED.	5-30

Discussion

Mapping Memory

ICE-85 can access program code loaded into user prototype memory, Intellec RAM memory, or a combination of the two. A maximum of 64K addresses are accessible with the addressing scheme used by the 8085 processor. If a combination is used, the address spaces assigned in each memory do not overlap; a given address must be assigned to either the prototype or to the Intellec system.

Intellec memory comes in two configurations. The SHARED memory space is standard. An optional UNSHARED memory space can be installed. Either configuration can include a maximum of 64K addresses. ICE-85 can access either the SHARED or the UNSHARED Intellec RAM, but not a combination of the two.

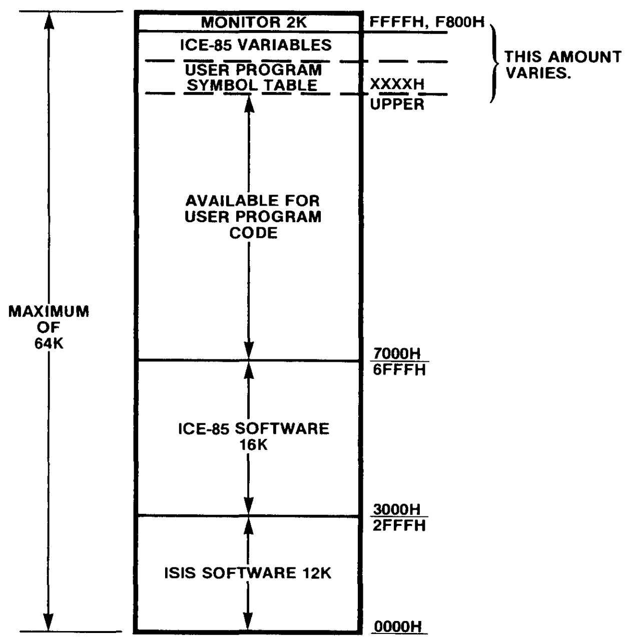


Figure 5-1. Intellec Shared Memory Locations with 64K of RAM 463-17

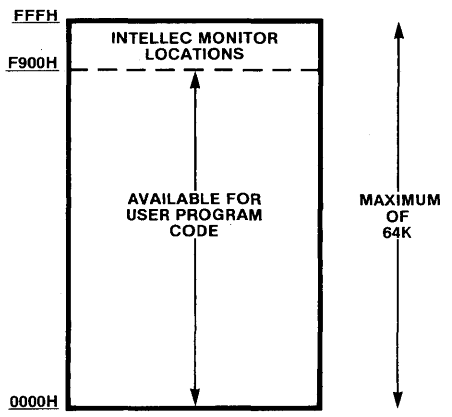
Intellec SHARED memory always contains the following system software and dedicated locations. Refer to Figure 5-2 for a diagram of Intellec SHARED memory.

Intellec Monitor (2K)	The Intellec Monitor occupies locations F800H through FFFFH in SHARED memory regardless of memory size as installed. All references to addresses in this range are sent to the Monitor 'shadow' PROM, and thus are unavailable to the memory map.
ISIS-II Software (12K)	ISIS-II software occupies the lowest 12K in Intellec SHARED memory (addresses 0000H through 2FFFH).
ICE-85 Software (16K)	The ICE-85 program occupies the 16K locations immediately above the ISIS-II software (addresses 3000H through 6FFFH).
ICE-85 Workspace and Symbol Tables (Amount varies)	The workspace for ICE-85 variables and the symbol and statement number tables occupy an area of SHARED memory whose highest address is the highest permissible address in installed memory that is below F800 (Monitor). In a 64K system, the workspace and symbol tables occupy the contiguous memory addresses just below the Monitor. In systems with less than 64K, these items take up the highest locations in installed memory. The amount of memory taken up by the workspace and tables varies. The ICE-85 command UPPER displays the highest available address below the workspace/table area.

The range of addresses from 7000H through the value of UPPER is available for user program code.

Intellec UNSHARED memory consists of up to 64K of RAM, installed in the Intellec chassis (or in an extender chassis) in addition to the UNSHARED memory boards. When UNSHARED memory is to be used, all the RAM boards in both SHARED and UNSHARED memories must be 32K or 64K in capacity. Intellec UNSHARED memory is restricted only by the Monitor locations (F800H through FFFFH; see Figure 5-1).

The ICE-85 memory map tells the system where to find the physical location corresponding to an address reference in the program code or in commands. ICE-85 treats most address references as logical addresses. The map translates any logical address into its corresponding physical address, given that the addresses (logical and physical) have been set in correspondence with a previous MAP command. Figure 5-3 diagrams the memory map operation. Some commands or command keywords use physical addresses directly; such addresses always are referred to Intellec SHARED memory.



5-2. Intellec UNSHARED Memory 463-18

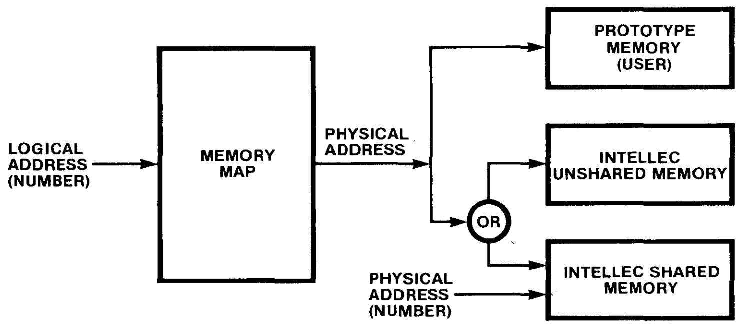


Figure 5-3. Memory Map Diagram 463-19

The address references shown in the diagram are simple numeric references. Symbolic references and numeric expressions used as addresses are resolved into numeric references by ICE-85 before the map is consulted.

The MAP Mode command tells ICE-85 to use either SHARED or UNSHARED Intellec memory for any addresses mapped to the Intellec system. The command has the form MAP = X, where X is one of the two tokens SHARED or UNSHARED. The initial mode is SHARED.

The MAP Memory command assigns blocks of logical addresses to blocks of physical locations either in the prototype or in the Intellec system. Each memory block (logical and physical) contains 2K (2048 decimal) contiguous locations. Refer to Table 5-2. A 64K system represents 32 blocks that can be mapped. You can map only in terms of entire blocks; partial blocks cannot be mapped. Of course, an ICE-85 command or program statement can reference a single address, or a range of addresses that does not correspond to block boundaries, once the block or blocks containing the referenced address has been mapped.

All memory blocks in the prototype and Intellec memories are initially GUARDED (logically nonexistent). Any attempt to access a guarded location produces an error message at the console. Guarded locations are write-protected in Interrogate mode; if you refer to such a location in a command entered from the console, no access occurs. During emulation, however, an attempted write to a guarded location will succeed before the error message is displayed. You can use the MAP Memory Command to reset any block to GUARDED.

Table 5-2. Memory Blocks for Mapping

Block No.	Lowest Address (block-name) K	Lowest Address (block-name) Hex	High Address Hex	Decimal Range
0	0K	0H	7FFH	0 - 2047
1	2K	0800H	0FFFH	2048 - 4095
2	4K	1000H	17FFH	4096 - 6143
3	6K	1800H	1FFFH	6144 - 8191
4	8K	2000H	27FFH	8192 - 10239
5	10K	2800H	2FFFH	10240 - 12287
6	12K	3000H	37FFH	12288 - 14335
7	14K	3800H	3FFFH	14336 - 16383
8	16K	4000H	47FFH	16384 - 18431
9	18K	4800H	4FFFH	18432 - 20479
10	20K	5000H	57FFH	20480 - 22527
11	22K	5800H	5FFFH	22528 - 24575
12	24K	6000H	67FFH	24576 - 26623
13	26K	6800H	6FFFH	26624 - 28671
14	28K	7000H	77FFH	28672 - 30719
15	30K	7800H	7FFFH	30720 - 32767
16	32K	8000H	87FFH	32768 - 34815
17	34K	8800H	8FFFH	34816 - 36863
18	36K	9000H	97FFH	36864 - 38911
19	38K	9800H	9FFFH	38912 - 40959
20	40K	A000H	A7FFH	40960 - 43007
21	42K	A800H	AFFFH	43008 - 45055
22	44K	B000H	B7FFH	45056 - 47103
23	46K	B800H	BFFFH	47104 - 49151
24	48K	0000H	C7FFH	49152 - 51199
25	50K	C800H	CFFFH	51200 - 53247
26	52K	D000H	D7FFH	53248 - 55295
27	54K	D800H	DFFFH	55296 - 57343
28	56K	E000H	E7FFH	57344 - 59391
29	58K	E800H	EFFFH	59392 - 61439
30	60K	F000H	F7FFH	61440 - 63487
31	62K	F800H	FFFFH	63488 - 65535

The details on using the MAP Memory command fall into three areas:

1. The ways to refer to blocks (*block-name*), and the number of contiguous blocks to be mapped in a single MAP command.
2. The use of displacement with Intellec SHARED memory.
3. The NOVERIFY option for mapping to USER or INTELLEC.

One or more contiguous blocks can be mapped to the same memory state (GUARDED, USER, or INTELLEC) in a single MAP command. To map a single block, use a command with the form MAP *block-name* = X, where X is one of the tokens GUARDED, USER or INTELLEC. (For now, we shall omit the required displacement for Intellec SHARED memory, and the NOVERIFY option.)

The *block-name* is the lowest-numbered address in the desired block. Since each block contains 2048 (decimal) addresses, all block-names are numbers that are multiples of 2048 decimal. In hexadecimal radix, block-names are multiples of 0800H, starting with 0000H. However, the simplest way to refer to a block is to use the explicit input radix K (multiple of 1024); any block-name is a number with the form (*n* times 2K), or 2nK, where *n* is the block number given in Table 5-2 (0⩽n⩽31).

To map more than one contiguous block in one command, use a form of *block- partition* instead of a single *block-name*. The forms of *block-partition* are as follows.

block-name TO *block-name*− 1

block-name TO *high-address*

block-name LENGTH *number-of-addresses*

The first two forms use the keyword TO. The *block-name* to the left of the keyword TO represents the low address of the low block in the partition; the block-name is specified as discussed above. The entry to the right of the keyword TO represents the highest address in the high block in the partition. You can either look up the *high-address* in the high block by referring to Table 5-2, or you can use an expression with the form (*block-name*− 1) and let ICE-85 calculate the high address. All the blocks included in the range of addresses given by this form are mapped to the designated memory state. For example, to map the lowest four blocks to USER (prototype) memory, either of the following two commands can be used.

```
MAP 0000H TO 2000 - 1 = USER

MAP 0K TO 1FFFH = USER
```

The third form of *block-partition* uses the keyword LENGTH. With this form, you supply the low address in the range (the block-name of the lowest block, as before), and (following LENGTH) the number of addresses in the range. The number of addresses *includes* the starting address, so that no additional calculation is required. Morever, since each block contains exactly 2K addresses, the number given for *number-of-addresses* is a multiple of 2K, and is identical in form to *block-name*. Thus, to map the first four blocks to USER, you can use the following command.

```
MAP 0K LENGTH 8K = USER
```

Any block is valid for mapping, including those dedicated to Monitor, ISIS-II, and ICE-85. If you map into ISIS-II or ICE-85, a warning message is displayed, but the map operation is carried out. You should not map into these areas unless your program uses some specific Monitor or ISIS-II routines.

If the range of addresses given in a *block-partition* contains more than 64K locations, or if the express or implied high address is higher than 64K, an error is displayed, and the MAP command is not executed.

When a block or range of blocks is mapped as GUARDED, the block or blocks become unavailable to ICE-85.

When the block or blocks are mapped to USER, the physical addresses in the prototype memory are the same as the logical addresses in the program or command. For example, if block 0K (locations 0000H to 07FFH) are mapped to USER, any reference to logical address 0100H is directed to physical location 0100H in the prototype memory.

When the blocks are mapped to INTELLEC, the prevailing map mode determines whether SHARED or UNSHARED memory is to be used. With UNSHARED memory, physical addresses are the same as logical addresses. With SHARED memory, physical addresses are generally not the same as the logical addresses. The default is SHARED.

In SHARED Intellec memory, the lowest available address for user program code is 7000H; this is block 28K. Typically, user program code is located in low areas of memory. Thus, for Intellec SHARED memory, logical addresses are less than physical addresses; the difference is called the 'displacement.'

When you are mapping into INTELLEC, you must specify the starting address in Intellec memory that is to correspond to the lowest logical address in the *block-partition* you wish to map. Subsequent logical address blocks are mapped to contiguous physical blocks higher in Intellec memory; you do not need to specify anything but the size of the partition of logical blocks.

For UNSHARED Intellec memory, the starting address (*block-name*) of the logical blocks can be the same as the physical address in Intellec memory. For example, to map the lowest block to Intellec UNSHARED memory, the following two commands are valid when entered in the order shown.

```
MAP = UNSHARED

MAP 0K = INTELLEC 0K
```

For SHARED Intellec memory, physical blocks start with block 28K, as mentioned earlier. Thus, to map the lowest logical block (0K) into the lowest available physical block in Intellec SHARED memory, use one of the following commands.

```
MAP 0K = INTELLEC 28K

MAP 0000H = INTELLEC 7000H
```

When memory is mapped to USER or to INTELLEC, you may use the NOVERIFY option. NOVERIFY suppresses write verification that normally is performed each time the contents of a memory location are changed with an ICE-85 console command.

To display the current map mode, and the state (GUARDED, USER, or INTELLEC) of all 32 blocks, enter the command token MAP followed by a carriage return. To display the status of a range of blocks, enter the token MAP followed by the desired partition.

The RESET MAP command restores the map mode to SHARED, and the status of all 32 memory blocks to GUARDED.

Figures 5-4, 5-5, and 5-6 illustrate three cases using the MAP command.

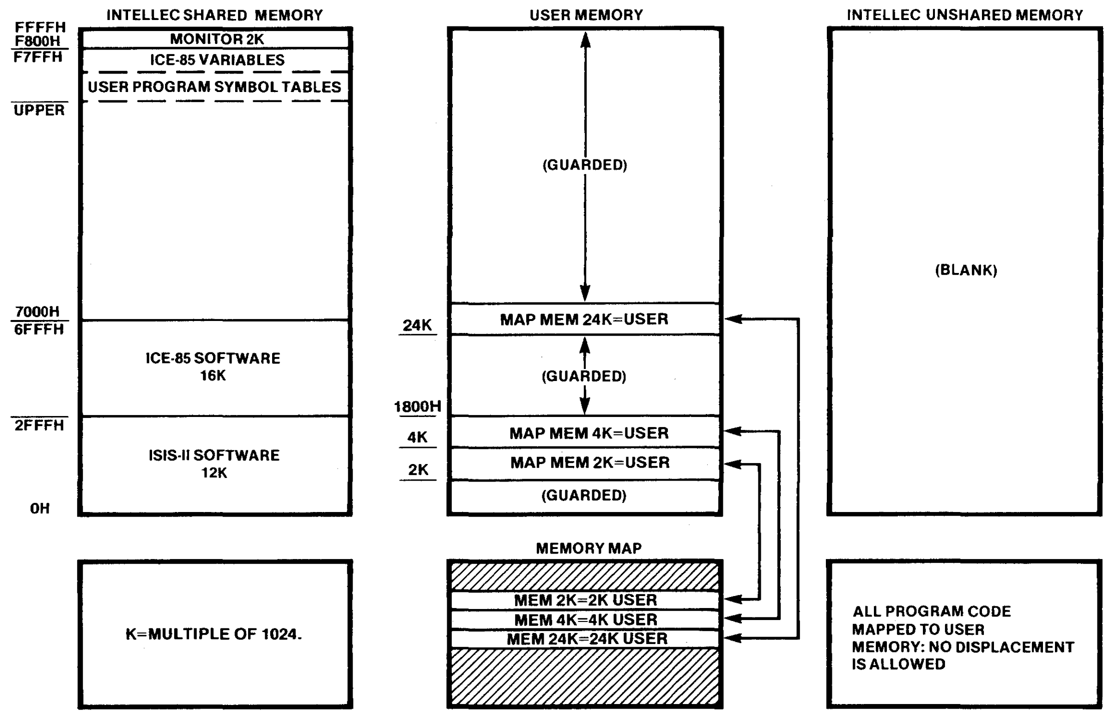


Figure 5-4. Mapping Into User Memory 463-20

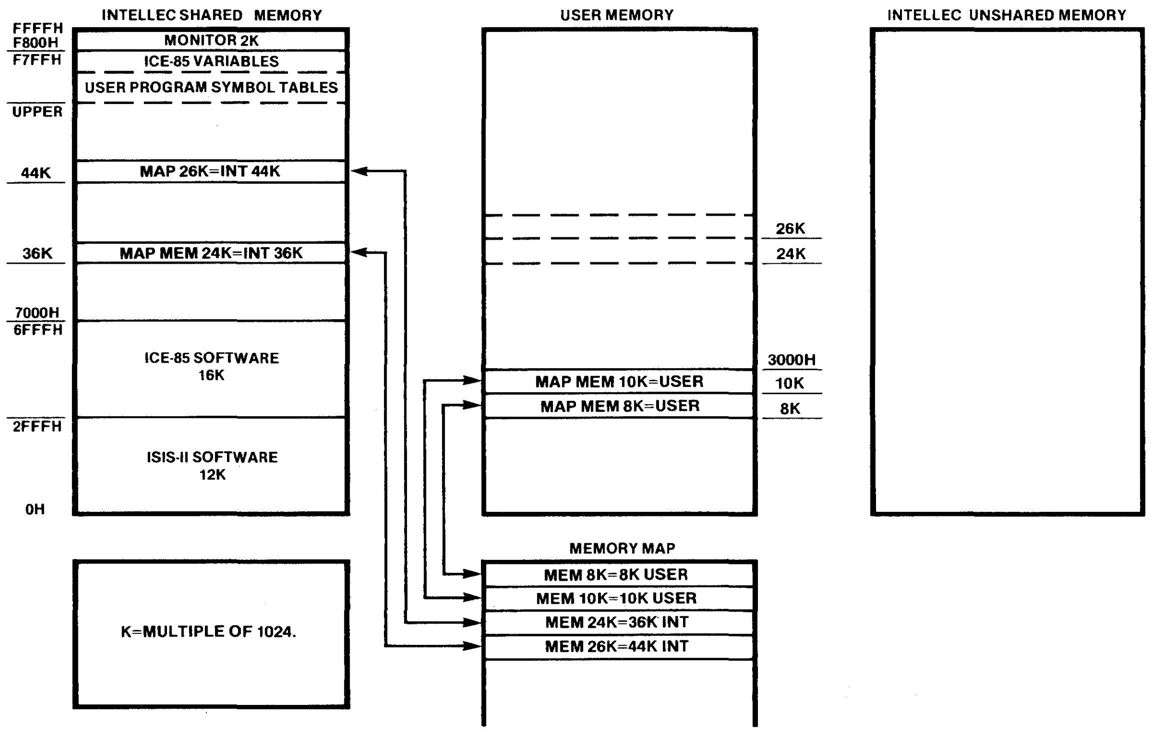


Figure 5-5. Mapping Into User and Shared Intellec Memory 463-21

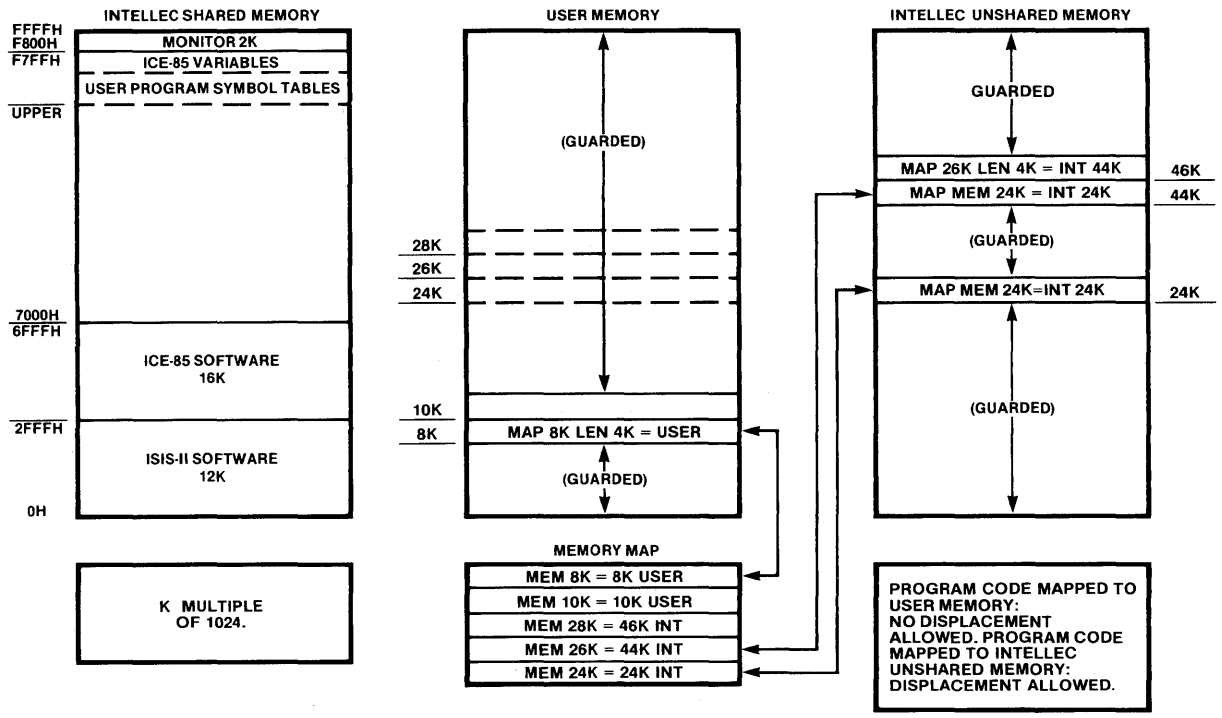


Figure 5-6. Mapping Into User and Unshared Intellec Memory 463-22

Case I: Mapping into User Memory Only (Figure 5-4).

Commands:

```
MAP 2K TO 1800 - 1 = USER NOVERIFY
MAP 24K = USER
```

User program (logical) address blocks 2K, 4K, and 24K are mapped into USER (prototype) memory. Blocks 2K and 4K have read-after-write verification suppressed.

Case II: Mapping into User and Shared Intellec Memory (Figure 5-5).

Commands:

```
MAP = SHARED
MAP 8K TO 3000H - 1 = USER NOVERIFY
MAP 24K = INTELLEC 36K NOVERIFY
MAP 26K = INTELLEC 44K
```

Logical blocks 8K and 10K are mapped to prototype memory with verification suppressed. Logical block 24K is mapped to Intellec shared memory, at physical block 36K, verification suppressed. Logical block 26K is mapped to Intellec shared memory at logical block 44K.

Case III: Mapping into User and Intellec Unshared Memory (Figure 5-6).

Commands:

```
MAP = UNSHARED
MAP 8K LENGTH 4K = USER
MAP 24K = INTELLEC 24K
MAP 26K LENGTH 4K = INTELLEC 44K
```

Logical blocks 8K and 10K are mapped to user memory; no displacement is allowed. Logical block 24K is mapped to physical block 24K in Intellec unshared memory. Logical blocks 26K and 28K are mapped to Intellec unshared memory, at physical blocks 44K and 46K respectively. With Intellec unshared memory, displacement is optional.

In summary, the ICE-85 map is used to decode logical addresses into physical address. Logical addresses correspond to the program and data locations in your program as LOCATED; physical addresses correspond to the address space as seen internally by ICE-85

How does ICE-85 decide if an address reference in a command is a logical or physical address? ICE-85 decides on the basis of the keyword that precedes the address reference. Some keywords take logical addresses as "arguments" or "parameters", while other keywords require (assume) physical addresses.

For reference, here are the ICE-85 command keywords that use address references. They are classified according to the type of address used.

The tokens BYTE, WORD, CALL, FROM, TO, and LOCATION use logical addresses. The address reference that follows any one of these tokens is translated through the memory map into its corresponding physical address in user or Intellec memory. For example, suppose you map location 2000H from your program code to address 7000H in Intellec SHARED memory. To refer to the content of this address, you use the expression BYTE 2000H.

The tokens IBYTE, IWORD, and ICALL use physical addresses in Intellec SHARED memory. The map is not used to translate the address, and the physical location is always Intellec SHARED memory, even when the current MAP mode for memory mapping is UNSHARED. For example, to refer to the content of address 7000H in Intellec SHARED memory, use the expression IBYTE 7000H.

Mapping Input/Output Ports

ICE-85 can access input/output ports in the prototype system, in the Intellec system, or in a combination of the two. A maximum of 256 decimal ports can be accessed.

The map command for I/O ports has the form:

MAP IO *segment-partition* = X

where X is one of the tokens GUARDED, USER, or INTELLEC.

Input/output ports are mapped in segments of eight ports each. Table 5-3 gives details on the segments of ports for mapping.

The *segment-name* is the number of the lowest port in the desired port segment. Segment names are multiples of 8H or 8T. To map a single port segment, the MAP IO command has the simpler form:

MAP IO *segment-name* = X

The forms of *segment-partition* (range of port segments) resemble those given above for *block-partition*. The three forms are as follows.

segment-name TO *segment-name* −1

segment-name TO *high-port*

segment-name LENGTH *number-of-ports*

Table 5-3. Input-Output Port Segments for Mapping

	Hex Range (segment-name)		Decimal-Range (segment-name)	
Segment-Number	Low	High	Low	High
0	0H	7H	0T	7T
1	8H	FH	8T	15T
2	10H	17H	16T	23T
3	18H	1FH	24T	31T
4	20H	27H	32T	39T
5	28H	2FH	40T	47T
6	30H	37H	48T	55T
7	38H	3FH	56T	63T
8	40H	47H	64T	71T
9	48H	4FH	72T	79T
10	50H	57H	80T	87T
11	58H	5FH	88T	95T
12	60H	67H	96T	103T
13	68H	7FH	104T	111T
14	70H	77H	112T	119T
15	78H	7FH	120T	127T
16	80H	87H	128T	135T
17	88H	8FH	136T	143T
18	90H	97H	144T	151T
19	98H	9FH	152T	159T
20	A0H	A7H	160T	167T
21	A8H	AFH	168T	175T
22	B0H	B7H	176T	183T
23	B8H	BFH	184T	191T
24	C0H	C7H	192T	199T
25	C8H	CFH	200T	207T
26	D0H	D7H	208T	215T
27	D8H	DFH	216T	223T
28	E0H	E7H	224T	231T
29	E8H	EFH	232T	239T
30	F0H	F7H	240T	247T
31	F8H	FFH	248T	255T

The first two forms use the keyword TO. The *segment-name* to the left of the keyword TO represents the low port number in the low segment in the partition; as discussed above, the segment name is a number with the form (*n* times 8H), where *n* is the segment number from Table 5-3. The entry to the right of the keyword TO represents the highest port number in the highest segment in the partition. You can either look up the *high-port* in the high segment by referring to Table 5-3, or you can use an expression with the form (*segment-name* - 1), and let ICE-85 calculate the high port number. For example, to map the lowest 4 port segments (32 ports total) to the user system, either of the two following commands may be used.

```
MAP 0H TO 20H - 1 = USER
MAP 0H TO 1FH = USER
```

The third form of *segment-partition* uses the keyword LENGTH. With this form, you specify the low port in the range (the segment-name of the lowest segment, as before), and (following the keyword LENGTH) the number of ports in the range. The number of ports includes the starting port-number, so that no additional calculation is required. Morever, since each segment contains 8 (H or T) ports, the number given for *number-of-ports* is always a multiple of 8, and is thus identical in form to *segment-name*. For example, to map the lowest four port segments to USER, you can use the following command.

```
MAP IO 0H LENGTH 32T = USER
```

Displacement is not allowed with I/O port segments; the logical port number always equals the physical port number, both in USER and INTELLEC systems. The NOVERIFY option is invalid with I/O ports, since data written to I/O ports is never verified.

Figure 5-7 diagrams an example of mapping I/O port segments to a combination of USER and INTELLEC. The commands that produce this result might be:

```
MAP IO 18H LENGTH 16T = USER
MAP IO 40H = USER
MAP IO 60H = INTELLEC
```

To display the current status (GUARDED, USER, or INTELLEC) of all 32 I/O port segments, enter the tokens MAP IO, followed by carriage return. To display the status of a partition of port segments, use MAP IO *segment-partition.*

The RESET MAP command restores all 32 I/O port segments to their initial GUARDED status.

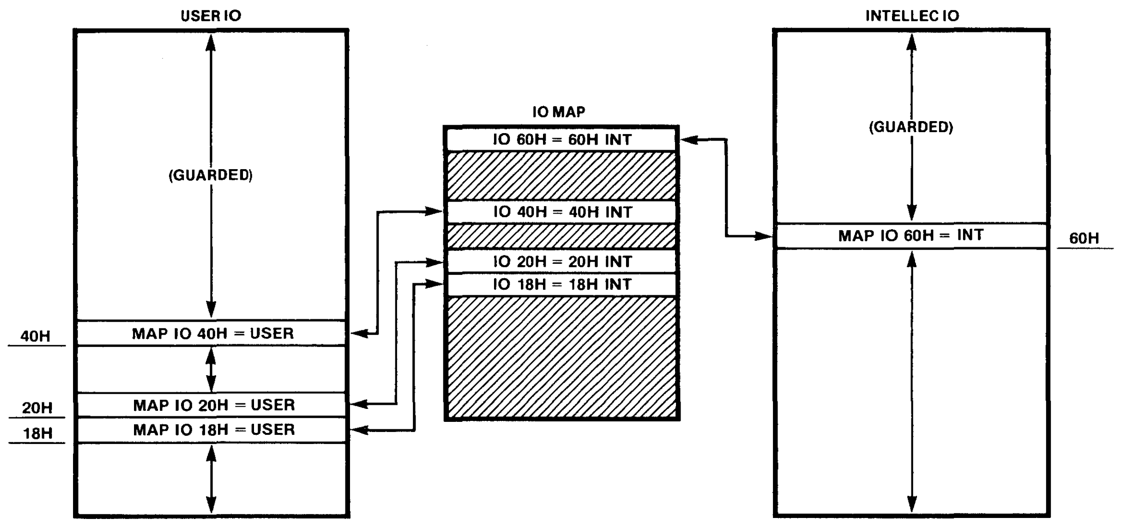


Figure 5-7. IO Port Segments Mapped to User and Intellec 463-23

MAP Mode Command

```
MAP     [MEMORY] =  | SHARED   |
                    | UNSHARED |
```

Examples:

```
MAP MEMORY = SHARED

MAP = UNSHARED
```

MAP	A command keyword referring to some operation on the ICE-85 map that relates each address in the user program absolute code to a location either in user memory or in Intellec memory.
MEMORY	A function keyword referring to the memory blocks used for program code. It may be omitted from this command.
=	The assignment operator.
SHARED	Refers to shared Intellec memory space. This memory space always contains the system software (Monitor, ISIS-II, and ICE-85), an ICE-85 system workspace, and the user symbol tables, in addition to any code mapped into it. SHARED is the initial mode
UNSHARED	A function keyword referring to unshared memory space in the Intellec system. This space requires an additional memory board or boards. When present, it is available for user program code only.

MAP Memory Command

```
MAP   MEMORY   block-partition = [ GUARDED                          ]
                                 [ USER   [NOVERIFY]                ]
                                 [ INTELLEC block-name   [NOVERIFY] ]
```

Examples:

```
MAP MEMORY 0000H TO 0FFFH = GUARDED

MAP MEMORY 6K = USER

MAP 2000H TO 2FFFH = INTELLEC 4K NOVERIFY

MAP MEMORY 5000 LENGTH 4K = USER NOVERIFY
```

The command contains the following semantic elements.

MAP	A command keyword that relates each address in the user program absolute code to a location either in user memory or in Intellec memory, and each input/output port to the user system or to the Intellec system.
MEMORY	A function keyword indicating that program code is to be mapped. (MEMORY is the default).
block-partition	One or more contiguous blocks of memory addresses (2048 addesses per block).
=	The assignment operator.
GUARDED	The initial state of all memory blocks. Any reference to a guarded address causes an error message. Guarded locations are write-protected in interrogate mode, but not write-protected in either real-time or single-step emulation.
USER	Refers to locations in user prototype memory.
INTELLEC	Refers to locations in Intellec memory (shared or unshared, as specified in an earlier Map Mode command).
block-name	Gives the starting address in Intellec memory for the blocks of contiguous memory identified in partition. The lowest block in partition will be mapped to the block location given by *block-name*, and succeeding blocks are mapped to contiguous memory blocks above the first block.
NOVERIFY	A function keyword that suppresses the normal read-after-write verification of data loaded into memory locations in user or Intellec memory.

MAP I/O Ports Command

```
MAP IO segment-partition = | GUARDED  |
                           | USER     |
                           | INTELLEC |
```

Examples:

```
MAP IO 00H TO 17H = INTELLEC

MAP IO 32T TO 63T = USER

MAP IO 88H LENGTH 16T = GUARDED
```

MAP	A command keyword referring to the memory and I/O port maps.
IO	Command keyword designating the I/O port map.
segment-partition	One or more contiguous segments of I/O ports. Each segment contains eight ports.
GUARDED	The initial state of all I/O port segments. Any reference to a guarded port causes an error message. Guarded ports are write-protected in interrogate mode, but not write-protected in either real-time or single-step emulation.
USER	Maps port segments in the partition to the user (prototype) system.
INTELLEC	Maps port segments in the partition to the Intellec system.

Display MAP Status Command

```
MAP  [MEMORY]    [block-partition]

MAP  IO  [ segment-partition]
```

Examples:

```
MAP

MAP MEMORY 0000

MAP MEMORY 0000 TO 07FF

MAP MEMORY 0000 LENGTH 2K

MAP IO

MAP IO 8
```

The command contains the following semantic elements.

MAP	A command keyword calling for a display of the current setting of the map for memory blocks or for input/output port segments.
MEMORY	A function keyword calling for a display of the memory block in the partition. MEMORY is the default.
IO	A function keyword calling for a display of the I/O port segments in the partition.
block-partition	One or more blocks of memory addresses.
segment-partition	One or more segments of I/O ports.

RESET MAP Command

```
RESET MAP
```

Example:

```
RESET MAP
```

RESET	A command keyword that restores its object to its initial state, as after an initial ICE-85 invocation.
MAP	As the object of RESET, the token MAP causes all 32 memory blocks and all 32 I/O port segments to be reset to GUARDED, and the mode for memory mapping to be reset to SHARED.

Hardware Register Commands

This section presents the keywords used in ICE-85 to refer to the following types of hardware registers and signals.

- 8085 Processor Registers
- ICE-85 Status Registers
- ICE-85 External Signals

The following commands that refer to hardware registers are discussed in this section.

Command	Purpose	Page
Display Processor or Status Register	Display the current contents of any of the 8085 processor registers or ICE-85 status registers.	5-36
Set 8085 Processor Register	Set (change) the contents of any of the writeable 8085 processor registers.	5-37
RESET Hardware	Reset ICE-85 hardware to initial state; reset bit RST-7.5 low.	5-37
ENABLE/DISABLE TIMEOUT	Enable or disable error message on absence of 8085 READY or CLK signals.	5-38

Discussion

Tables 5-4 through 5-7 show the tokens to use to refer to any 8085 8-bit register, 16-bit register pair, 1-bit status flag, or interrupt mask bit.

In the command syntaxes, the following meta-terms have been adopted to save writing out the complete list of actual tokens.

Meta-term	Class of tokens
processor-register	Any of the 8085 registers, register pairs, status flags, or interrupt bits.
register-name	Any 8085 8-bit register (Table 5-4).
pair-name	Any 8085 16-bit register pair (Table 5-5).
flag-name	Any 8085 1-bit status flag (Table 5-6).
i-bit-name	Any 8085 interrupt mask bit (Table 5-7).

To display the current contents of any of these registers or bits, enter the token for the desired register followed by a carriage return. The current setting is displayed on the next line, in the current output radix.

Table 5-4. 8085 8-Bit Registers

register-name	8085 Register and Interpretation
RA	Accumulator
RB	Register B
RC	Register C
RD	Register D
RE	Register E
RF	Status flag register
RH	Register H
RL	Register L

Table 5-5. 8085 Register Pairs

pair-name	8085 Register Pair and Interpretation
RBC	Register pair BC
RDE	Register pair DE
RHL	Register pair HL
SP	Stack pointer
PC	Program counter

Table 5-6. 8085 1-Bit Status Flags

flag-name	Status Flag
CY	Carry bit (bit 0 of RF)
PY	Parity bit (bit 2 of RF)
ACY	Auxiliary carry bit (bit 4 of RF)
Z	Zero bit (bit 6 of RF)
SN	Sign bit (bit 7 of RF)
	(Bit 3 is always set to one)

Table 5-7. 8085 Interrupt Mask Bits

i-bit-name	Interpretation (Bit Number)
M5	RST-5.5 mask (bit 0)
M6	RST-6.5 mask (bit 1)
M7	RST-7.5 mask (bit 2)
IE	Interrupt enable (bit 3)
I7	Interrupt 7.5 pending (read-only) (bit 6)
SID	Serial input data (read-only) (input bit 7)
SOD	Serial output data (write-only) (output bit 7)

The command REGISTER displays eleven different values on one line. The values are identified with single letters; here is an example:

(Command:)

```
    REGISTER
```

(Display:)

```
P=FF0FH S=FFFEH A=00H F=00H B=00H C=00H D=00H E =00H H=00H L=00H I=00H
```

This display is interpreted as follows.

Identifier	Element displayed
P	PC (program counter)
S	SP (stack pointer)
A	RA
F	RF (status flags)
B	RB
C	RC
D	RD
E	RE
H	RH
L	RL
I	Interrupt mask (byte formed by bits SID, I7, (I6), (I5), IE, M7, M6, and M5; M5 is bit 0 and SID is bit 7. I5 and I6 are invalid (always zero). The result is identical to that returned by the 8085 RIM instruction.)

To set (change) the content of one of the processor registers, use a command with the form:

processor-register= X

where X is a numeric constant or numeric expression giving the desired new contents. Note that interrupt bits I7 and SID are read-only bits, and cannot be set from the console; any attempt to set a read-only register produces an error message.

Each of the registers that can be changed with this command has a definite size (16, 8, or 1 bits). If the new contents represent fewer bits than the destination register, the bits are right-justified in the register, and the remaining bits in the register are set to zero; in other words, ICE-85 assumes that the quantity represents the lowest-order bits, and sets any unspecified high-order bits to zero.

If the new contents represent more bits than the register can hold, the least significant bits of the quantity are loaded into the register, and the rest of the bits in the quantity are ignored.

Table 5-8 shows the ICE-85 status registers, and the tokens used to refer to them. To display the current content of any one of these registers, enter the token for that register followed by carriage return. The content is displayed on the next line, in the current output radix.

The meta-term *status-register* in a command syntax means any one of the eight tokens for ICE-85 status registers.

Note that the ICE-85 status registers are all read-only registers, and cannot be set or changed from the console.

The RESET HARDWARE command is used to restore the ICE-85 and 8085 hardware to the initial program load condition. One use for this command might be to reset the hardware when reconfiguring (for example, moving the user plug from the stand-alone adapter to the user system. The EXIT command includes the RESET HARDWARE function.

The RESET I7 command causes the external interrupt line RST 7.5 to the 8085 to be reset to zero (low).

TIMEOUT calls for ICE-85 to display an error message whenever either the 8085 READY signal or the 8085 CLK (clock) signal is absent. TIMEOUT is initially enabled.

Table 5-8. ICE-85 Status Registers

Status-register	ICE-85 Status Register and Interpretation
OPCODE	Opcode fetched in last opcode-fetch cycle in trace data (8 bits, read-only)
CAUSE	Conditions that were true at the time of the last break in emulation (read-only). The 8 bit values are: Bit 0 on if breakpoint 0 matched Bit 1 on if breakpoint 1 matched Bit 2 on if SY0 caused break Bit 3 on if guarded memory or I/O accessed Bit 4 on if user aborted processing Bit 5 on if timeout on HOLD Bit 6 on if timeout on READY Bit 7 on if no user Vcc (power supply voltage)
PPC	Previous program counter; address of last instruction-fetch cycle in trace data (16 bits, read-only).
PSW	Program status word; accumulator in low byte, status flag register in high byte (16 bits, read-only).
UPPER	Highest location in Intellec memory available to map user memory into (16 bits, read-only).
BUFFERSIZE	Number of frames of valid trace data; initially zero, always between zero and 1022 (16 bits, read-only).
TIMER	Low 16 bits of 2- MHz emulation timer.
HTIMER	High 16 bits of timer.

Table 5-9 summarizes the operation of the external synchronization lines SY0 and SY1. SY0 is used to give hardware control of emulation; commands that enable and disable SY0 are discussed under the GO command. SY1 is used to control the collection of trace data; the commands that enable and disable SY1 are discussed under Trace commands. The meta-term *sync* refers to either synchronization line.

Table 5-9. External Synchronization Lines

Sync	IN	OUT
SY0	Enabled by GO or GR command. Halts emulation when low. Driven low by external event. Reset high by external event.	Enabled by ENABLE command. Driven low and held low by ICE-85 when emulation halts. Reset high by external event. If external event set SY0 OUT low, ICE-85 does not hold it low.
SY1	Enabled by ENABLE command. Forces trace data collection by going high.	Enabled by ENABLE command. Set high when trace is running.

Table 5-10 lists several other external signals presented on the ICE-85 Interface cable module. These signals are not controllable with ICE-85 commands; they cannot be set or displayed.

Table 5-10. Other External Signals

Signal	**Interpretation**
MATCH 0/	Goes low when breakpoint register BR0 matches.
MATCH 1/	Goes low when BR1 matches.
EMUL	Goes high during emulation.
GND	Common ground for external signals.

Display Processor and Status Register Commands

register-name

pair-name

flag-name

i-bit-name

REGISTER

status-register

Examples:

RA

RBC

CY

M5

REGISTER

OPCODE

register-name	Any one of the tokens RA, RB, RC, RD, RF, RF, RH, RL, OPCODE, or CAUSE representing the 8085 and ICE-85 8-bit registers (see Table 5-3).
pair-name	Any one of the tokens RBC, RDE, RHL, SP, PC, PPC, PSW, UPPER, or BUFFERSIZE representing 8085 and ICE-85 16-bit register pairs (see Table 5-4).
flag-name	Any one of the tokens CY, PY, ACY, Z, or SN representing 8085 1-bit status flags (see Table 5-5).
i-bit-name	Any of the tokens M5, M6, M7, IE, I7, SID or SOD, representing bits in the 8085 interrupt mask (see Table 5-6).
REGISTER	A command keyword that displays the contents of registers PC, SP, RA, RF, RB, RC, RD, RE, RH, RL, and a byte formed by the interrupt bits given by i-bit-name above.
status-register	Any of the tokens OPCODE, CAUSE, PPC, PSW, UPPER, BUFFERSIZE, TIMER, or HTIMER, representing ICE-85 status registers (see Table 5-8).

Set Processor Register Command

```
processor-register  =  contents
```

EXAMPLES:

```
RA = 00H
PC = 0800H
PC = PC + 1
IE = CY
RBC = WORD .SAM
```

processor-register	The keyword name of any of the writeable 8085 processor registers, as follows: 8-bit registers (see Table 5-4). 16-bit registers (see Table 5-5). Status bits (see Table 5-6). Interrupt bits (see Table 5-7).
=	The assignment operator.
contents	A *numeric-constant* or *numeric-expression.*

RESET Hardware Commands

```
RESET  | HARDWARE |
       | I7       |
```

Examples:

```
RESET HARDWARE

RESET I7
```

RESET	Command keyword restoring its object to a reset condition.
HARDWARE	A function keyword restoring ICE-85 hardware to the reset condition that occurs after the initial ICE-85 invocation.
17	A token specifying that the RST 7.5 interrupt line is to be reset to zero (low).

ENABLE/DISABLE TIMEOUT Commands

```
| ENABLE  |   TIMEOUT
| DISABLE |
```

Examples:

```
ENABLE TIMEOUT

DISABLE TIMEOUT
```

ENABLE	A command keyword that activates its object as a controlling element in emulation or trace.
DISABLE	A command keyword that cancels the effect of its object on emulation or trace.
TIMEOUT	A function keyword that, when enabled, causes ICE-85 to display an error message on the absence of either the 8085 READY signal or the 8085 CLK (clock) signal.

Memory and Port Content Commands

The commands in this section give access to the content or current value stored in designated memory locations or input/output ports. The commands discussed in this section are as follows. The purpose of each command is indicated by its title.

Command	Page
Display Memory and Port Contents	5-46
Set Memory Contents	5-47
Set Input/Output Port Contents	5-48

Discussion

Memory Content References

A memory content reference has the form:

mem-type address

The meta-term *mem-type* means one of the following 'content-of' modifiers for memory locations.

BYTE	The content of a single memory location. The *address* following BYTE is treated as a logical address; the physical address whose content is referenced is determined by look-up in the ICE-85 memory map (see Memory and I/O Port Mapping Commands, page 5-16).
WORD	The content of two adjacent bytes in memory. The most significant byte is located in the high address of the address pair; the least significant byte is stored in the low address of the pair. The *address* following WORD is treated as a logical address; the ICE-85 memory map is consulted to find the physical address whose content is referenced.
IBYTE	The content of a single memory location. The *address* following IBYTE is always treated as a physical address in Intellec SHARED memory; the ICE-85 memory map is not used.
IWORD	The content of two adjacent bytes in memory. The most significant byte is located in the high address of the address pair; the least significant byte is stored in the low address of the pair. The *address* following IWORD is always treated as a physical address in Intellec SHARED memory; the ICE-85 memory map is not used.

NOTE

IBYTE and IWORD allow you to display any location in Intellec SHARED memory, but do not allow you to change the content of any of the reserved locations in Intellec SHARED memory (Monitor, ISIS-II, ICE-85). Non-reserved locations in Intellec SHARED memory can be changed directly by using IBYTE or IWORD.

The meta-term *address* means one of the following types of entries.

numeric-constant	A single number in any *input-radix.* ICE-85 treats all numbers modulo 65532 (64K); thus any number represents an address.
numeric- expression	The forms for numeric expressions are presented in Chapter 4. The result obtained when the expression is evaluated becomes an address modulo 64K.
symbolic- reference	The ICE-85 symbol table lists all symbols loaded with the test program or defined by the user after program load. Corresponding to each symbol is a number that can be used as an address.
statement-number-reference	The ICE-85 statement number table gives the address of the first instruction generated by the statement with the designated number.
processor-register	The name of one of the 8085 processor registers (refer to Hardware Register commands, page ?). The content of the named register becomes the address.
status-register	The name of one of the ICE-85 status registers (see Hardware Register commands). The content of the named register becomes the address.
(*mem-type address*)	A memory content reference with a form such as BYTE (WORD 1000) represents an indirect reference. The content of the address or address-pair inside the parentheses is treated as the address for the *mem-type* outside the parentheses.

To display the content of one or more locations in memory, enter the appropriate memory content reference followed by a carriage return. We discuss the ways to refer to a range of addresses later in this section. Here are some examples of display commands that involve single addresses and pairs of addresses, using BYTE and WORD.

For the following examples, assume these equalities and conditions:

.AA = 1000H (Symbolic reference)

#56 = 2000H (Statement number reference)

Location 1000H contains 11H.

Location 1001H contains 22H

Location 2000H contains 33H

Location 2001H contains 44H

Note that the addresses 1000H, 1001H, 2000H, and 2001H are logical addresses. We need not consider the physical addresses that actually contain the values shown, since the ICE-85 memory map handles the conversion automatically for BYTE and WORD.

The commands are shown as they would be entered. The terminating carriage return is not shown. The content displayed by each command is shown on the next line, as it is displayed by ICE-85.

```
*BYTE 1000H
1000H = 11H

*WORD 1000H
1000H = 2211H
```

```
*BYTE .AA
1000H = 11H

*BYTE .AA + 1
1001H = 22H

*WORD #56
2000H = 4433H
```

To obtain the content of the bytes or words in a range of addresses, use a reference of the form

mem-type partition

A *partition* can be a single address, or one of the following types of constructs.

address TO *address*
address LENGTH *number-of-bytes* (for BYTE or IBYTE)
address LENGTH *number-of-words* (for WORD or IWORD)

The first form of *partition* uses the keyword TO. The address on the right of the keyword TO must be greater than the one to the left. With BYTE or IBYTE, this form allows you to access the content of each location in the range; the range includes both the first and last address in the partition. With WORD or IWORD, the first address is treated as the low address of the first address pair in the range; subsequent pairs of addresses are accessed until the second address is reached. If the second address is the low address of a pair, the word formed from the content of that address and the next consecutive higher address is accessed; if the second address is not the low address of a pair (that is, if it turns out to be the high address of a pair already accessed in the range), the access halts after the last complete pair has been accessed. Word-length accesses can begin on either an even-numbered or an odd-numbered address.

The second and third forms of *partition* use the keyword LENGTH. The *address* preceding the keyword LENGTH is the starting address in the range, as with the first form (using TO). The number or expression following the keyword LENGTH gives the number of addresses (when the controlling *mem-type* is BYTE or IBYTE), or the number of address pairs (for WORD or IWORD) in the range.

Here are two examples, using memory content references to display the contents of a range of addresses and address pairs. The contents shown could represent a section of program code.

```
BYTE 1000H TO 1010H
1000H=67H 43H 3FH 01H 32H 0BH 44H 01H 5EH 66H B5H 56H 03H 7BH 67H 39H
1010H=4DH

WORD 1000H LENGTH 10H
1000H=4367H 013FH 0B32H 0144H 665EH 56B5H 7B03H 3967H
1010H=014DH
```

Setting Memory Contents

To assign a new content to a single address or address pair, use a command with the form:

mem-type address = new-content

The meta-terms *mem-type* and *address* represent the types of entries discussed earlier in this section.

The meta-term *new-content* represents one of the following types of entries (for single address or address pairs; setting the content of ranges of address and address pairs will be discussed later on).

numeric-constant	A single number in any radix.
numeric-expression	A numeric expression evaluated by ICE-85 to a single number.
processor-register	One of the 8085 processor registers listed in the Hardware Register commands section of this chapter. The content of the named register becomes the new content of the designated address.
status-register	One of the ICE-85 status registers listed with the Hardware Register commands earlier in this chapter. The content of the named register becomes the new content of the designated address.
'string'	A string of alphabetic characters enclosed in single quotes. The ASCII value of each character in the string is treated as a byte value. To include a single quote as a character in the string, enter it as a pair of single quotes (' ').
mem-type address	This form of *new-content* allows you to copy a byte or word value from one address or address pair to another.

When a single address (byte) is to be set, ICE-85 treats the *new-content* as an 8-bit quantity. If *new-content* has more than eight bits, the least significant eight bits in the quantity are used as the new contents, and the other (higher) bits are lost. If *new-content* has fewer than eight bits, the bit values in the quantity are right-justified (placed in the low-order bits in the address), and the remaining (high) bits in the locations are set to zeroes.

Here are some examples of setting byte contents. The first line of each example shows the command that sets the new contents; the second line gives a command that produces a display of the contents just set; the third line shows the resulting display. The output radix is assumed to be H (hexadecimal).

```
*BYTE 1000H = FFH
*BYTE 1000H
1000H=FFH

*BYTE 1010H = RA + 1
*BYTE 1010H
1010H=F1H

*BYTE 1020H = FF11H
*BYTE 1020H
1020H=11H

*BYTE 1030H = 1Y
*BYTE 1030H
1030H=01H
```

```
*BYTE 1040H = 'A'
*BYTE 1040H
1040H=41H

*BYTE 1050H = BYTE 1000H
*BYTE 1050H
1050H=FFH
```

You can change the radix used to display the contents, using the Set Radix command (see Number Bases and Radix commands earlier in this chapter).

When a single address pair (word) is to be set, ICE-85 treats the *new-content* as a pair of bytes. The least significant byte is loaded into the low address in the pair, and the most significant byte is loaded into the high address in the pair. If *new-content* has fewer than 16 bits, the bit values present are loaded starting with the low address, and right-justified. The remaining (high) bits in the address pair are set to zeroes. The following examples demonstrate some of the possibilities for setting address pairs.

```
*WORD 1000H = 1122H
*WORD 1000H
1000H=1122H

*WORD 1010 = FFH
*WORD 1010H
1010H = 00FFH

*WORD 1030H = WORD 1000H
*WORD 1030H
1030H=1122H
```

The rules given above for numeric and ASCII values also apply when the *new- content* is of the form *memtype*.

A command of the form BYTE X = BYTE Y copies the content of address Y to the content of address X. A command of the form WORD X = WORD Y copies the content of address Y to location X, and the content of address (Y+1) to location (X+1). A command of the form BYTE X = WORD Y copies the content of address Y to location X; the content of location (X+1) is not changed. A command of the form WORD X = BYTE Y copies the content of address Y to location X; the content of location (X+1) is set to a byte of zeroes.

The commands used to set a range of addresses or address pairs differ in some details.

One way to set a range of addresses or address pairs is with a command of the form:

mem-type address = list of new-content values

With this form, the *address* on the left of the equals sign gives the starting location (or low address of the starting pair), and the number of values in the list to the right of the equals sign tells ICE-85 how many consecutive addresses or address pairs to set. Consecutive locations or pairs starting with the one given are changed to the values of the *new-contents* in the list, in left-to-right order.

Here are some examples showing the use of this form of the set memory contents command.

```
*BYTE 1000H = 11H, 22H, 33H, 44H, 55H, 66H
*BYTE 1000H LENGTH 6T
1000H=11H 22H 33H 44H 55H 66H

*WORD 2000H = FFFFH, 'AB', WORD 1000H
*WORD 2000H LENGTH 4T
2000H=FFFFH 0041H 0042H 2211H
```

Note in the last example that ICE-85 treats a string like 'AB' as a series of byte quantities, not as a word made up of the two ASCII values. Each character in the string is a one-byte *new-content* value.

To set a range of addresses or address pairs all to the same new value, use a command of the form:

mem-type partition = new-content

The forms of *partition* are discussed above in this section. All addresses or address pairs in the partition are set to the single *new-content*. The following examples show some of the possible results with this command form.

```
*BYTE 1000H TO 1005H = FFH
*BYTE 1000H LENGTH 5H
1000H=FFH FFH FFH FFH FFH

*WORD 2000H LENGTH 6T = AA00H
*WORD 2000H TO 200AH
2000H=AA00H AA00H AA00H AA00H AA00H AA00H
```

The last form of the set memory contents command sets the contents of each address or address pair in a range (*partition*) to the corresponding *new-content* in a list of values. This form is as follows.

mem-type partition= list of new-content values

This form combines the two forms first discussed above.

If the number of addresses or address pairs in the *partition* is equal to the number of values in the *new-content* list, the addressed bytes or words are set to the corresponding values in the list, in left-to-right order.

If the number of addresses or address pairs in the range is greater than the number of new values in the list, the addresses or address pairs are filled with the values from left-to-right, repeating the values in left-to-right order as necessary to fill all the locations. The maximum number of bytes that can be repeated is 128. With more than 128 bytes, the data is transferred but not repeated, and an error message is displayed.

If the number of new values in the list is greater than the number of addresses or address pairs in the *partition*, the lowest address or address pair in the range receives the first value, and successive addresses or address pairs in the range receive values in left-to-right order until all locations in the range have received values. The excess values are then detected by ICE-85 as an error condition, and an error message is displayed. The excess values are lost.

Here are a few examples showing this form of command.

```
*BYTE 1000H TO 1005H = 'ABCDEF'
*BYTE 1000H LENGTH 6T
1000H=41H 42H 43H 44H 45H 46H

*WORD 2000H LENGTH 6T = 1122H, 'AB'
*WORD 2000H TO 200AH
2000H=1122H 0041H 0042H 1122H 0041H 0042H

*BYTE 1000H TO 1002H = 11H, 22H, 33H, FFH
(An error message such as EXCESS VALUES is displayed.)
*BYTE 1000H LENGTH 4T
1000H=11H 22H 33H 44H
```

In the third example, note that the byte at location 1003H retains the value set in the first example in the group of examples given (44H rather than the FFH given in the command).

Port Content References

A reference to the content of an input/output port has the form:

PORT *port-number*

The keyword PORT is equivalent to the keyword BYTE used for memory contents. The content of a single port is a one-byte value. In ICE-85, port numbers are mapped, but the logical port number is always the same as the physical port number. A *port-number* is any entry of the types given above for *address*, except that the range of port numbers is 00H to FFH (0T to 255T). Any reference to a port number outside this range produces an error message.

To display the current content of a port, enter the appropriate port content reference, followed by a carriage return. The content is displayed in the prevailing output radix.

To set the content of a single port, use a command of the form:

PORT *port-number = new-content*

The forms of *new-content* are as discussed earlier in this section. The value given is treated as a byte quantity. If the value contains fewer than eight bits, it is right-justified in the low bits of the port and the remaining (high) bits in the port are set to zeroes. If the *new-content* has more than eight bits, the excess high-order bits in the value are truncated, and the least significant eight bits in the value are used as the content of the port.

One and only one port can be set in one command. A range or partition of ports cannot be accessed, either for display or for setting new contents.

Display Memory and Port Contents Commands

```
mem-type partition

PORT port-number
```

Examples:

```
BYTE 1000H
WORD .AA TO .BB
IBYTE 1000H LENGTH 10H
IWORD 1000H
PORT 8H
```

mem-type	One of the four 'content-of' keywords BYTE, WORD, IBYTE, or IWORD, used with memory addresses.
partition	One or more contiguous memory locations.
PORT	The 'content-of' keyword for input/output ports.
port-number	A *numeric constant, numeric expression,* or *symbolic reference* representing a port number (range: 00H - FFH).

Set Memory Contents Command

mem-type partition = new content [, *new-content*] ...

Examples:

```
BYTE 0800H = FFH
BYTE 7000H LENGTH 16T = 00H
BYTE 0800H TO 0805H = 12H, 34H, 56H, 78H, 9AH, BCH
WORD 70FFH = PC
WORD 7000H = PPC + 1
BYTE 0800H = 'ABCDEF'
BYTE 0800H = IBYTE 4000H
WORD 7000H = WORD 4000H LENGTH 20H
BYTE #56 = FAH
```

mem-type	One of the four memory 'content-of' modifiers BYTE, WORD, IBYTE, or IWORD.
partition	One or more contiguous locations in memory.
=	The assignment operator.
new-content	One of the following types of entries, to be used as the new contents of the BYTE or WORD addresses:

numeric-constant
numeric-expression
processor-register
status-register
'*string*'
mem-type partition
PORT *port-number*

Set Input/Output Port Contents Command

PORT *port-number*= *new-content*

Example:

PORT 8 = FFH

PORT	The 'content-of' modifier for I/O ports.
port-number	A *numeric-constant* or *numeric-expression* in the range 00H to FFH (0T to 255T).
new-content	One of the following types of entries, to be used as the new contents of the designated port:

numeric-constant
numeric-expression
processor-register
status-register
'*string*'
PORT *port-number*
mem-type address

Symbol Table and Statement-Number Table Commands

ICE-85 maintains a symbol table and source program statement number table to allow you to refer to memory addresses and other values by using symbolic references and statement references in the ICE-85 commands.

This section gives details on the following commands.

Command	Page
Display Symbol Table or Statement-number Table	5-53
DEFINE Symbol	5-53
Change Symbol	5-54
REMOVE Symbol	5-54

Discussion

The ICE-85 symbol table receives symbols from two sources; the symbol table associated with the user program can be copied to the ICE-85 symbol table when the program is loaded, and the user can define additional symbols for use during the emulation session.

Corresponding to each symbol in the table is a number that you can interpret and use either as an address or as a numeric value (variable or constant). The next few paragraphs discuss the kinds of symbols that can appear in the table, and the interpretation of the corresponding symbol table quantity (address or value).

Instruction and statement labels are loaded with the program code. The symbol table gives the address of the instruction corresponding to the label.

A program variable is a symbol for a quantity that can have its value changed as a result of an instruction in the program. Program variables are loaded with the program code. The symbol table gives the address where the variable value is stored.

A program constant is a symbol for a label set to a constant value (for example, using the assembler directives EQU or SET). Program constants are loaded into the symbol table when the program code is loaded. The symbol table gives the constant value associated with the symbol.

A module name is the label of a simple DO block that is not nested in any other block (for PL/M-80), or a label that is the object of a NAME directive (in 8080/8085 assembly language). A module name itself does not have a corresponding address value in the symbol table. However, symbols contained in a module are considered to be 'local' to that module; ICE-85 thus allows you to reference multiple occurrences of the same symbol name in different modules, by using the module name as a modifier in the *symbolic reference*.

The ICE-85 symbol table is organized to preserve any modular structure present in the program. Initially (before any code is loaded), the symbol table consists of one 'no-name' module. Any symbols loaded or defined without a specific module name are stored in the no-name module in the order they were loaded or defined. The no-name module is always the first module in the symbol table. Following the no-name module, named modules are stored in the symbol table in the order that the modules were loaded into ICE-85. Symbols local to each named module are stored in the order they appear in the module.

In addition to the symbols stored when the program code is loaded, you can use the DEFINE Symbol command to define new symbols for your use during the emulation session. The rules for user-defined symbols are as follows.

The name of the new symbol (*symbol-name*) can be defined with a maximum of 122 characters. However, ICE-85 truncates each symbol-name to the first 31 characters. Thus, to be different, two symbols must be unique in the first 31 characters.

The first character in the new *symbol-name* must be an alphabetic character, or one of the two characters @ or ?. The remaining characters after the first can be these characters or numeric digits.

You can specify the module that is to contain the new symbol you define. Symbols defined without a module are placed in the no-name module at the head of the table, in the order they were defined. Symbols defined with an existing module name are placed in that module's section of the table; the module named must already exist in the table.

The new symbol name cannot duplicate a symbol name already present in the module specified. You can, however, have two or more symbols of the same name in different modules.

When you define a new symbol, you also specify the value corresponding to it in the table. You can treat the value you assign as an address or as a numeric value for use other than addressing.

The DEFINE Symbol command has the following form.

DEFINE *symbolic-reference = address/value*

The forms of *symbolic-reference* are shown in Table 5-11. The meaning of each form is as follows. Not all forms can be used in a DEFINE Symbol command.

A simple *symbolic-reference* has the form *.symbol-name*. ICE-85 searches for this form of reference starting with the first symbol in the no-name module. If the symbol is not in the no-name module, ICE-85 searches through the named modules in the order they were loaded, and takes the first occurrence of the symbol in the first (earliest) module that contains it.

When you define a symbol without a module, it is placed in the no-name module.

The *symbolic-reference* can include a *module-reference*. The module reference immediately precedes the symbol name; the *module-name* is identified by a prefix consisting of a double period (. .). When you define a symbol with a module reference, the symbol is added to the symbols under that module. A later reference to a symbol with a module name restricts the search to that module.

A multiple symbolic reference has the form *.symbol-name.symbol-name*. This form causes ICE-85 to search for the first occurrence of the first *symbol-name* that follows the first occurrence of the second *symbol-name*. You cannot define a symbol in terms of this form. This form can be used to identify a symbol that occurs in several procedures that are themselves not modules, since procedure labels are stored as symbols in the table in the order they appear in the program modules. For example, if you have two symbols with the same name, say I1, one declared in procedure ADD and one in procedure SUB, you can guarantee access to the one in ADD by using .ADD.I1.

Table 5-11. Symbolic References and Statement References

Type of Reference	Meta-notation	Example	Display	DEFINE	Change	REMOVE
Symbolic	*. symbol-name*	.ABC	YES	YES	YES	YES
Symbolic	*.. module.symbol-name*	..MAIN.DEF	YES	YES, if *module* is already present in table.	YES	YES
Symbolic	*. symbol-name.symbol-name*	.XX.YY	YES	NO	YES	YES
Statement	*# statement-number*	#56	YES	NO	NO	NO
Statement	*.. module# stmt- number*	..MAIN#44	YES	NO	NO	NO

The meta-term *address/value* as used in the DEFINE Symbol command means one of the following types of entries.

numeric-constant	A single number in any *input-radix.*
numeric-expression	Any of the forms of numeric expressions given in Chapter 4.
processor-register	The name of one of the 8085 processor registers (see Hardware Register commands earlier in this chapter). The content of the named register becomes the address or value corresponding to the symbol.
status-register	The name of one of the ICE-85 status registers (see Hardware Register commands). The content of the named register becomes the address or value corresponding to the symbol.
mem-type address	A memory content reference using one of the keywords BYTE, WORD, IBYTE, or IWORD. The content of the *address* following the *mem-type* becomes the address of value corresponding to the symbol.
symbolic-reference	Any of the three forms of *symbolic-reference* shown in Table 5-11. The effect of this form is to establish one symbol as a synonym for another symbol; referencing either symbol produces the same corresponding value.

Once a symbol has been defined or loaded, any reference to that symbol is equivalent to supplying its corresponding address or value.

To display the value from the symbol table corresponding to any symbol, enter the appropriate symbolic reference followed by a carriage return. ICE-85 displays the symbol table value on the next line.

To display the entire ICE-85 symbol table, enter the command SYMBOL followed by a carriage return. Symbols are displayed module by module, starting with the no-name module. The *address/value* corresponding to each symbol is also displayed.

You can change the *address/value* corresponding to an existing symbol by entering a command of the form:

symbolic-reference = address/value

Any of the three forms of *symbolic-reference* shown in Table 5-11 can be used to identify the symbol whose value is to be changed. The symbol must already exist as referenced.

The forms of *address/value* are discussed earlier in this section. Any of these forms may be used to change the value of an existing symbol.

Where multiple occurrences of the same symbol name exist in the table, the rules for table search given earlier determine which of the several instances of the symbol is to receive the new *address/value*.

To delete one or more symbols from the table, use a command of the form:

REMOVE *list of symbolic-references*

The *symbolic-references* in the list are separated by commas. ICE-85 searches the table for each reference using the search rules given earlier, deleting the first occurrence of each symbol name that fits the type of reference given.

Note that deleting a symbol from the ICE-85 symbol table makes that symbol inaccessible to ICE-85, but does not affect the program code.

To delete the entire ICE-85 symbol table and the statement number table, enter the command REMOVE SYMBOL.

ICE-85 also maintains a statement number table for user programs written in PL/M-80 source code. The statement numbers are assigned by the PL/M-80 compiler. Corresponding to each source statement number in the table is the address of the first instruction generated by that source statement.

Table 5-11 shows the forms used to refer to statement numbers in ICE-85. The simplest form is the statement-number prefixed by a number sign (#). A *module-reference* can precede the statement reference, since the statement number table preserves any modular structure in the program. Thus, two modules compiled separately can have the same statement numbers; the module reference tells ICE-85 which statement number to use.

To display the address corresponding to a statement-number, enter the appropriate statement number reference followed by a carriage return.

ICE-85 does not allow you to change the address corresponding to any existing statement number, to define any new statement numbers, or to delete (REMOVE) any statement numbers.

Display Symbol Table and Statement-Number Table Commands

```
symbolic-reference
SYMBOL
statement-reference
```

Examples:

```
.ABC
..MAIN.DEF
.XX.YY
SYMBOL
#56
..MAIN#44
```

symbolic-reference	Any of the forms given in Table 5-11. The address of value corresponding to that symbol is displayed.
SYMBOL	A command keyword calling for the display of the entire ICE-85 symbol table, module by module.
statement-reference	Any of the forms shown in Table 5-11. The first instruction generated by the source statement with the number given is displayed.

DEFINE Symbol Command

```
DEFINE symbolic-reference = address/value
```

Examples:

```
DEFINE .ABC = 1000H
DEFINE ..MAIN.DEF = PC
DEFINE .TEMP = .ABC + 2
```

DEFINE	A command keyword that tells ICE-85 to enter the new symbol in the appropriate module table, and assign it the initial value given.
symbolic-reference	Any of the forms given in Table 5-11. The symbol defined may not duplicate a symbol already in the module given.
=	The assignment operator.
address/value	One of the following types of entries: *numeric-constant* *numeric-expression* *processor-register* *status-register* *mem-type address* *symbolic-reference*

Change Symbol Command

```
symbolic-reference = address/value
```

Examples:

```
.ABC = 2000H
..MAIN.DEF = AAFFH
.TEMP = .ABC + ..MAIN.DEF
```

symbolic-reference	Any of the forms shown in Table 5-11.
=	The assignment operator.
address/value	One of the following types of entries. *numeric-constant* *numeric-expression* *processor-register* *status-register* *mem-type address* *symbolic-reference*

REMOVE Symbol Command

```
REMOVE symbolic-reference [, symbolic-reference ] ...
REMOVE SYMBOL
```

Examples:

```
REMOVE .ABC
REMOVE ..MAIN.DEF, .TEMP
REMOVE SYMBOL
```

REMOVE	A command keyword causing the symbols that follow to be deleted from the ICE-85 symbol table.
symbolic-reference	Any of the forms shown in Table 5-11. Several symbols can be deleted with one command by entering the symbols to be deleted as a list with the *symbolic-references* separated by commas.
SYMBOL	A keyword that (as the object of REMOVE) deletes all symbols and statement numbers for all modules in the symbol table.

Channel Group Commands

Channel groups are used to control emulation breakpoints, trace data qualifiers, and trace data display. Some groups are system-defined; system-defined groups can be displayed and used in commands, but cannot be changed, removed, or defined. The commands in this section allow you to define groups of your choosing, display system-defined and user-defined groups, and change or delete any user-defined group. In addition, this section defines several meta-terms that are also used in commands discussed in other sections. The commands are as follows.

Command	Page
DEFINE GROUP	5-59
Display GROUP	5-60
Change GROUP	5-60
REMOVE GROUP	5-61

Discussion

ICE-85 *channels* represent input signals from the 18 user probes and from the address, data, and status lines of the 8085 emulation processor. Table 5-12 shows the interpretation of the ICE-85 channels, and gives the system-defined group names that refer to commonly-used groups of channels. Channel 43 (MTH) is set by ICE-85, and is displayed as part of trace data.

ICE-85 maintains a pseudo-register called the Channel Status Register (CSR), that contains the current value of channels 1 to 42, organized in terms of the system- and user-defined groups.

Time slices available to ICE-85 to record the 'current' value of the channels are called *frames* . Each machine cycle is divided into two frames: the first frame is that portion of the cycle when lines AD0 to AD7 represent the low address byte; the second frame is the portion of the cycle when those lines represent a byte of data. The 'current' values of channels 1 to 42 on each frame are stored for the duration of that frame in the CSR. The 'current' value of MTH (channel 43) is based on a breakpoint match on the previous frame, as discussed below.

There are four ICE-85 *match-registers*, as follows.

BR0, BR1: Emulation breakpoint registers.
QR0, QR1: Trace qualifier registers.

Each of the *match-registers* contains 43 bits, duplicated as necessary to represent all system- and user-defined groups. Using ICE-85 commands, you can specify a *match-value* for any channel or channel group. The *match-value* of any bit can be 0 (zero), 1 (one), or X (don't-care). The breakpoint registers are enabled by including them in a GO or GR command; the qualifier registers are always enabled. ICE-85 compares the settings of the bits in the breakpoint and qualifier registers with the state of the CSR, on every frame during real-time emulation. A match on an enabled breakpoint register halts emulation; a match on a qualifier register enables trace data collection.

Table 5-12. User Probe Channels, 8085 Processor Channels and System-Defined Group Names

System Group Name	Channel-numbers	8085 Pin or Signal	Interpretation					Trace Radix
U0	1 - 8		User probe channels					H
U1	9 - 16		User probe channels					H
U2	17, 18		User probe channels					H
DMUX	19	ALE	1 = AD0 to AD7 are data 0 = AD0 to AD7 are low-order address					Y
ADDR	20 - 35	AD0 - AD7 A8 - A15	Low-order address lines (DMUX=0) High-order address lines					H
DATA	20 - 27	AD0 - AD7	Data Lines (DMUX=1)					H
ADDRL	20 - 27	AD0 - AD7	Low-order address lines (DMUX=0)					H
ADDRH	28 - 35	A8 - A15	High-order address lines					H
STS	36	S0	**Action**	**$IO/\overline{M}$**	**S1**	**S0**	**Mnemonic**	Mnemonic
	37	S1	HALT	0	0	0	H	
	38	$IO/\overline{M}$	WRITTEN	0	0	1	W	
			READ	0	1	0	R	
			EXECUTED	0	1	1	E	
			OUTPUT	1	0	1	O	
			INPUT	1	1	0	I	
SD	39 40	SOD SID	Serial output data line Serial input data line					Y
RW	41 42	WR RD	WR line RD line					Y
MTH	43	MATCH 0 or MATCH 1	1 = breakpoint register matched in previous frame (trace data only)					Y

When a breakpoint register (enabled or not) matches the CSR on a given frame, channel MTH of the CSR is set to 1 in the next frame.

When trace data is being collected, the CSR is copied to the trace data buffer after each frame except for the first frame of the first instruction.

Details on the breakpoint registers is given in the section on Emulation Control commands (page 5-62); qualifier registers are discussed in the section on Trace Control Commands (page 5-75).

You can define channel groups in addition to the system-defined groups. A user group can be defined, for example, to obtain a particular ordering of channels and channel groups for display during trace data display or for specifying a *match-register* setting.

Groups are composed in terms of the channels they contain and the order of the channels from left to right as they are to be displayed or matched. The settings of any channels are not part of the group definition.

The DEFINE GROUP command has the following form.

```
DEFINE GROUP group-name= channel-list [IN group-radix]
```

A *group-name* must begin with an alphabetic character (A to Z), or with a special character (@ or ?), followed by other alphabet characters and/or numerals (0 to 9), up to a total of 31 characters. The *group-name* in the DEFINE command may not duplicate any names defined earlier, including the system channel group names. Once defined, a channel *group-name* may not be defined again, unless the *group-name* has been deleted with a REMOVE GROUP command.

Channels in *channel-list* are specified in two ways.

1. By *channel-number*. ICE-85 hardware probes are numbered 1 to 18, and 8085 processor channels are numbered 19 through 42. Channel 43 is displayed as part of trace data, but cannot be assigned to a user channel group (Table 5-12).
2. By *group-name*. The channel-list can include system- or user-defined channel *group-names*. When a *group-name* are included in the list (and thus in the group being defined), in the order previously specified for the included group. If a *group-name* is included in the list, the channels assigned to the included group must not duplicate any other channels in the list.

When a *group-name* is part of the *channel-list* used to define another group, the channels in the included *group-name* are assigned to the new group, but the name of the included group is *not* retained as part of the definition of the new group.

Enter the channels and groups in the list in the left-to-right order that you wish them to be displayed.

The *group-radix* given in the (optional) IN-clause tells ICE-85 what number base to use when displaying the group later as part of trace data. The default *group-radix* is hexadecimal. A *group-radix* is one of the following characters.

```
Y (binary)
Q (octal)
T (decimal)
H (hexadecimal)
```

In defining (or changing) groups, you should note the following limitations.

1. No group can contain more than 16 channels. If you attempt to assign more than 16 channels to a group, an error message is displayed and no channels are assigned.
2. You can define a maximum of 36 groups, if no group has more than eight channels assigned to it. Groups with more than eight channels count as two groups toward the total of 36.
3. The total of all channels assigned to user-defined groups may not exceed 103.

To display the channels assigned to any *group-name*, enter a command of the following form, followed by a carriage return.

GROUP *group-name*

The display obtained by this command gives the group name, an equals sign, the list of channels in left-to-right order as they were assigned, and an IN-clause giving the output-radix specified when the group was last defined or changed.

The details of this command are discussed by use of examples. Suppose you have defined a group as follows.

```
DEFINE GROUP PORTA = 10,11,12
```

Now, to display this group, enter the command:

```
GROUP PORTA
```

(Display:)

```
PORTA=10,11,12  IN H
```

Note that since no *group-radix* was specified in the DEFINE command, ICE-85 used the default *group-radix* H (hexadecimal).

We can change our group to include a system-defined group.

```
GROUP PORTA = PORTA,DATA IN Y
...
GROUP PORTA
```

(Display:)

```
PORTA=10,11,12,27,26,25,24,23,22,21,20 IN Y
```

Note that the channels in the system-defined group DATA were listed individually in the display, and that the *group-radix* specified in the Change GROUP command has been applied to our group.

To display several groups at once, use a command of the form:

GROUP *group-list*

The meta-term *group-list* means a list of system-defined and user-defined *group-names*, separated by commas.

To display the channels assigned to all *group-names*, enter the command GROUP (without a *group-list*). System-defined groups are displayed first, then user-defined groups in the order they were defined.

One more example. Assume group PORTA has channels assigned as in the previous example.

```
DEFINE GROUP LOOKSEE = PORTA,DMUX IN Q
...
GROUP LOOKSEE,PORTA
```

(Display:)

```
LOOKSEE=10,11,12,27,26,25,24,23,22,21,20,19 IN Q
PORTA=10,11,12,27,26,25,24,23,22,21,20 IN Y
```

Note that the *group-radix* specified for LOOKSEE overrides the *group-radix* specified for PORTA, without changing the earlier setting defined for PORTA alone.

To change the channels assigned to a user-defined *group-name*, use a command of the form:

GROUP *group-name* = *channel-list* [IN *group-radix*]

The meta-terms *group-name, channel-list* , and *group-radix* have the meanings described above for the DEFINE GROUP command.

To delete any user-defined group, use a command of the form:

```
REMOVE GROUP group-list
```

The entry *group-list* is a list of one or more *group-names* separated by commas. All the *group-names* in the list are removed. The settings of the channels in a deleted group are not affected; particularly, if a group has been used to set a *match-register*, the register setting is not affected by deleting the *group-name* . Where the same channel has been assigned to two different *group-names*, deleting one of the groups does not affect the group that remains.

The command REMOVE GROUP (without a *channel-list*) deletes all user-defined channel *group-names*.

Note that you cannot define, change, or remove any of the system-defined channel groups.

DEFINE GROUP Command

```
DEFINE GROUP group-name = channel-list  [IN group-radix]
```

Examples:

```
DEFINE GROUP INPUT$PROBE = 5,3,4,2,18
DEFINE GROUP LOOK$SEE = 17,1,DATA IN Y
DEFINE GROUP CHECKOUT = INPUT$PROBE, 14, 13, ADDRH IN H
```

DEFINE	Command keyword indicating that this is the first assignment of channels to this group.
GROUP	Function keyword identifying the next token as a channel group name.
group-name	The channel group name assigned by the user. The user name may not duplicate a system group name, or a user group name previously defined.
=	The assignment operator.
channel-list	A list of channels and/or pre-defined channel group names, separated by commas. The list can contain a maximum of 16 channels.
IN	A function keyword introducing the *group-radix* for use when this channel group is displayed as part of trace data. The IN-clause is optional. (The default radix is hexadecimal.)
group-radix	A single letter denoting a number base for for trace data display, as follows: Y Binary (base 2) Q Octal (base 8) T Decimal (base 10) H Hexadecimal (base 16)

Display GROUP Command

```
GROUP  [group-list]
```

Examples:

```
GROUP
GROUP PORTA
GROUP PORTA, DATA
```

GROUP	A command keyword calling for the display of the channels assigned to one or more channel groups.
group-list	A list of system-defined and/or user-defined channel *group-names*, separated by commas.

Change Group Command

```
GROUP group-name= channel-list  [IN group-radix]
```

Examples:

```
GROUP INPUT$PROBES = 16,14,12,10,8,4,2 IN Y
GROUP MAINLINES = 7,1,3,5,DATA,DMUX
GROUP MAINLINES = 7,1,3,5,27,26,25,24,23,22,21,20,19
GROUP LOOK$SEE = DATA IN H
```

GROUP	Function keyword identifying the user name that follows as a channel group name.
group-name	A channel group name previously defined by the user.
=	The assignment operator.
channel-list	A list of channels and/or previously defined channel group names, separated by commas.
IN	A function keyword introducing the *group-radix* for use when this channel group is displayed as part of trace data. The IN-clause is optional. Hexadecimal is the default radix.
group-radix	A single letter denoting a number base for trace data display, as follows:

Y	binary (base 2)
Q	octal (base 8)
T	decimal (base 10)
H	hexadecimal (base 16)

REMOVE GROUP Command

```
REMOVE GROUP   [group-list]
```

Examples:

```
REMOVE GROUP
REMOVE GROUP LOOK$SEE
REMOVE GROUP LOOK$SEE, MAINLINES, INPUT$PROBE
```

REMOVE	A command keyword indicating that the user-defined group names that follow are to be deleted as ICE-85 tokens.
GROUP	A function keyword introducing the user-defined channel groups to be deleted.
group-list	A list of user-defined channel group names separated by commas. If the command is entered without a *group-list* , the effect is to delete all user-defined channel group names.

Real-Time Emulation Control Commands

The emulation processor is the 8085 at the end of the ICE-85 Interface cable. During real-time emulation, this processor performs the instructions in the user program that has been mapped and loaded into the ICE-85 system. The operations of the system under emulation can be monitored through the processor signals and user probes. The commands in this section allow you to specify the starting address where emulation is to begin, and to specify and display the software or hardware conditions for halting emulation and returning control to the console for further commands.

The commands in this section are as follows.

Command	Purpose	Page
GO command	Begin real-time emulation.	5-71
GR command	Enable or set and enable breakpoint registers to halt emulation.	5-72
Display Emulation Controls	Display GO-register and breakpoint register settings.	5-73
Set Breakpoint-Register	Set match condition for halting emulation.	5-73
RESET Breakpoint-Register	Set breakpoint register to match on any condition.	5-74
ENABLE/DISABLE SY0 OUT	Enable or disable external signal SY0 as an output.	5-74

Discussion

The emulation control commands tell ICE-85 where to start emulation and when to halt emulation.

To initialize for emulation, you map the locations in prototype and Intellec memory that are to be accessible to ICE-85, and load your program code into mapped locations. After the code has been loaded, ICE-85 initializes for emulation as follows.

- The program counter (PC) is loaded with the address of the first executable instruction in your program.
- The GO-register (GR) is set to FOREVER. The setting of GR identifies the combination of factors that are enabled to halt emulation. The setting FOREVER means no factors are enabled.
- All 42 bits in both breakpoint registers (BR0 and BR1) are set to don't-care. The breakpoint registers contain match-settings of any of the 42 channels that are of interest. A don't-care bit in a breakpoint register matches either a zero or a one in the corresponding channel.

Now you can begin emulation by entering the command GO, followed by a carriage return. At the command GO, the following occurs.

- Emulation begins with the instruction at the address that is in PC; this is the first executable instruction in your program after initialization.
- External signal EMUL is set high (1) to tell an external device that emulation is occurring.
- Emulation continues until you press the ESC key, or until a fatal error occurs (see Appendix B for error messages).
- The message EMULATION BEGUN is displayed at the console.

Now, if you press the ESC key, the following happens.

- ICE-85 completes executing the current instruction.
- Emulation halts; the PC contains the address of the next instruction to be executed.
- The message EMULATION TERMINATED, PC = nnnnH is displayed. The value of PC displayed is the address of the next instruction to be executed.
- The message PROCESSING ABORTED is displayed, acknowledging the user abort (ESC key).

This is the simplest case of starting and stopping emulation. When the GO-register is set to FOREVER, you can enter the command GO to start emulation at the current PC address, and press the ESC key to halt emulation.

Instead of starting wherever the PC happens to be, you may specify the address you want for each GO command. There are two ways to do this. First, you can set the PC directly to any desired address with a command of the form PC = ***address***, then enter the GO command to start emulation at that ***address***. Second you can specify the starting address as part of the GO command; this form of the GO command is as follows.

```
GO [FROM address]
```

The meta-term *address* means any one of the following types of entries.

numeric-constant	A numeric in any *input-radix*.
numeric-expression	A numeric expression is evaluated to give the address (see Chapter 4 for the forms of *numeric-expression* and *numeric-constant*).
status-register	Any of the keywords for ICE-85 status registers shown in Table 5-8. The content of the named register becomes the address.
processor-register	Any of the keywords for 8085 processor registers shown in Tables 5-4 through 5-7. The content of the named register becomes the address.
symbolic-reference	Any of the three forms of symbolic reference shown in Table 5-11. The symbol table value corresponding to the named symbol is used as the address.
(*mem-type address*)	In the GO command, an *address* such as (WORD 1000) causes the content of location 1000 to be used as the address. Parentheses must be used to enclose this type of indirect reference.
statement-reference	Either of the forms of statement-reference shown in Table 5-11. The address of the first instruction generated by that source program statement is the address used.

For example, to start emulation with the instruction at location 3000, you could enter:

```
PC = 3000H
GO
```

Or, you could enter:

```
GO FROM 3000H
```

The effect is the same either way.

Setting Breakpoint Registers

Instead of accepting the default halting condition FOREVER, you can specify that either a match on one of the two breakpoint registers, or a low state on the external signal SY0 IN, is enabled to halt emulation.

The two breakpoint registers are named BR0 and BR1. Each of the two breakpoint registers contains 42 bits corresponding to the 18 user probe channels and the 24 8085 processor channels. The bits are duplicated as necessary to represent all the system-defined and user-defined channel groups. (Refer to Table 5-11 for a description of the channels and system-defined groups.)

To use a breakpoint register to halt emulation, *set* the bits in the register to the desired match value, then *enable* the register by including it in a GR (GO- Register) command or in a TILL clause in the GO command itself.

Breakpoint register bits can be set to 0 (zero), 1 (one) or X (don't-care). A don't-care bit matches either a zero or a one in its corresponding channel.

ICE-85 maintains a pseudo-register, the Channel Status Register (CSR), that contains the current actual value of all the channels. The CSR is updated after every frame during real-time emulation. Each machine cycle represents two frames: the first frame is the interval of time when the 8085 lines AD0 through AD7 represent the low address byte, and the second frame is the succeeding interval when those lines represent a byte of data. On each frame during emulation, ICE-85 compares the breakpoint register settings with the CSR. If either breakpoint register matches the CSR on a frame. the following occurs.

- Channel 43 (system group MTH) in the CSR is set to 1 in the frame following the frame that matched.
- External signal MATCH 0/ or MATCH 1/ is set low, depending on whether BR0 or BR1 produced the match.
- If the breakpoint register that matched was enabled in the GR, emulation halts. If a match with BR0 caused the halt, bit 0 in ICE-85 status register CAUSE is set to 1; if BR1 matched, CAUSE bit 1 is set to 1. (See Table 5-8).

Initially, both breakpoint registers are set to all don't-care bits. This setting matches all possible states of the CSR; if a breakpoint register is enabled with all bits set to don't-care, emulation halts immediately after the GO command is entered.

ICE-85 offers several ways to set the breakpoint registers. The simplest way is to use a command with the form:

break-reg = mnemonic-match

The meta-term *break-reg* means either of the tokens BR0 or BR1, or the token BR to set both breakpoint registers to the same match setting.

The meta-term *mnemonic-match* means one of the forms shown in Table 5-13. The entries in this table are explained in the following paragraphs.

For many applications, the channels of interest in controlling emulation are the ones contained in system channel groups ADDR, DATA, DMUX, and STS (refer to Table 5-12). The forms of *mnemonic-match* give you short and simple ways to specify settings for these groups.

Table 5-13. Mnemonic Match Conditions

	System Channel Groups			
Mnemonic-Match	**DMUX**	**ADDR**	**DATA**	**STS**
HALT	XY	XXXXH	XXH	000Y
[LOCATION] *address*	0	*address*	XXH	XXXY
[LOCATION] *address status*	0	*address*	XXH	*status*
[LOCATION] *address-mask*	0	*address-mask*	XXH	XXXY
[LOCATION] *address-mask status*	0	*address-mask*	XXH	*status*
VALUE *data*	1	XXXXH	*data*	XXXY
VALUE *data status*	1	XXXXH	*data*	*status*
VALUE *data-mask*	1	XXXXH	*data-mask*	XXXY
VALUE *data-mask status*	1	XXXXH	*data-mask*	*status*

NOTE:		
	X =	don't-care digit in radix shown
	Y =	binary radix
	H =	hexadecimal radix
	0 =	zero (any radix)
	1 =	one (any radix)

A command such as BR0 = HALT resets all bits in the named register to don't-care, then sets the three bits in system group STS to zeros. This setting represents the 8085 HALT state. If the breakpoint register is now enabled with a command such as GR = TILL BR0, and emulation is started with the GO command, emulation proceeds until a HALT state occurs, causing the STS channels in the CSR to take the value 000Y. This matches the setting of BR0, and emulation halts. If no HALT state ever occurs, emulation proceeds until some other factor halts it.

To match on an *address*, use one of the four forms of *mnemonic-match* that uses the token LOCATION. The forms of *address* are the ones discussed previously in this section.

The meta-term *address-mask* is a *masked-constant*. A masked constant is a number containing one or more don't care digits. In binary radix, each X digit represents one bit; in octal radix, each X digit stands for three adjacent bits; in hexadecimal radix, each X digit represents four adjacent bits.

The meta-term *status* means one of the types of entries shown in Table 5-14. If no *status* is specified, the three bits in STS are set to don't-care, and any action involving the *address* (or *data*, discussed later on) produces a match.

The token LOCATION may be omitted from the command. If neither LOCATION nor VALUE is entered, LOCATION is assumed as the default, and the first value given is treated as an *address* or *address-mask*.

To specify a match on a DATA value, use one of the forms of *mnemonic-match* that include the token VALUE. The meta-term *data* is identical to *address* in the forms it can take, but is a one-byte quantity. The meta-term *data-mask* is likewise identical in form to *address-mask*, and the meta-term *status* has the same meaning as that given above for LOCATION.

Table 5-14. Status Group Bit Settings

Status	STS Bits (Channels) 38	37	36
HALT	0	0	0
WRITTEN	0	0	1
READ	0	1	0
EXECUTED	0	1	1
OUTPUT	1	0	1
INPUT	1	1	0

Here are a few examples of setting breakpoint registers using LOCATION and VALUE.

To match any access to location 3000H, you can enter:

```
BR0 = LOCATION 3000H
```

Or you can simply enter:

```
BR0 = 3000H
```

since the token LOCATION is the default.

To match on a memory write to location 3000H, you can enter:

```
BR1 = LOCATION 3000H WRITTEN
```

To match on a memory read from any location in the range from 3000H to 30FFH, you can enter:

```
BR0 = LOCATION 30XXH READ
```

To match on any frame in which DATA value 11H occurs, enter:

```
BR1 = VALUE 11H
```

To specify a match on any data value read from any input port, enter:

```
BR0 = VALUE XXH INPUT
```

Two points to remember when using this form of the Set Breakpoint Register command are:

1. Anytime you use LOCATION, VALUE, or HALT, ICE-85 resets all 42 bits in the named breakpoint register to don't-care, then sets the DMUX, DATA, ADDR, and STS bits as specified.
2. An *address* match includes a match on DMUX = 0, and a *data* match includes a match on DMUX = 1.

In addition to the command forms involving LOCATION, VALUE, and HALT, ICE-85 offers two ways to set individual bits and specific groups directly. One way to do this is to use as command of the form:

break-reg= $\left| \begin{matrix} \textit{numeric} \\ \textit{mask} \end{matrix} \right|$ ON *channel-list*

The meta-terms in this command form are as follows.

break-reg	BR0 or BR1, or BR to set both registers to the same match value.
numeric	Commonly, a *numeric-constant* or *numeric-expression.* Generally, any of the forms of *operand.* See Chapter 4 for details.
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas; see channel group commands earlier in this chapter for details.

The channel list tells ICE-85 which bits in the register to set. If a channel group is included in the list, the channels assigned to that group are set in the order they appear in the group. ICE-85 'decomposes' the included group into its channels, then inserts those channels into the list at the place in the list where the group was named.

The channel list can contain a maximum of 16 channels.

The last (rightmost) channel bit in the list receives the value of the least significant bit in the *numeric* or *mask* . Succeeding channel bits in the list are then set to corresponding bits in the *numeric* or *mask* in right-to-left order. If the number or mask represents fewer bits than there are channels in the list, the rightmost channel bits in the list are set to their corresponding bits in the numeric or mask, and the remaining channel bits in the list are set to *zero* . If the numeric or mask contains more bits than there are channels in the list, the channel bits are set from right-to-left until all have received values; the extra leftmost bits are lost.

Any bits in the breakpoint register whose corresponding channels are not included in the channel list are reset to *don't-care* by this form of the Set Breakpoint Register command.

For example, suppose you are monitoring two signals from your prototype with user probe channels 1 and 2, and you want emulation to halt whenever both signals are high (1) simultaneously. You can use the following command to set this condition into BR0.

```
BR0 = 11Y  ON  1,2
```

All the other bits in BR0 except those corresponding to channels 1 and 2 are reset to don't-care as a result of this command.

The last form of the Set Breakpoint Register command differs from the two previous forms in that it does *not* reset any other channels or groups to don't-care. The form is:

$$\textit{break-reg channel-list} = \left| \begin{matrix} \textit{numeric} \\ \textit{mask} \end{matrix} \right|$$

All the meta-terms in this form have been discussed previously.

With this form of the command, only the breakpoint register bits corresponding to the channels in *channel-list* receive new values from the numeric or mask. Any breakpoint bits (or groups) whose corresponding channels are *not* included in the channel-list retain whatever setting they had before this form of the command was entered.

To reset some or all of the bits in a breakpoint register to don't-care, use a command with the form:

```
RESET break-reg [channel-list]
```

If *channel-list* is included in the command, only those bits corresponding to channels in the list are reset to don't-care, and any other bits retain their previous settings. If no channel list is given, all 42 bits are reset in whatever groups contain them.

The external signal SY0 can be enabled as an input and as an output. As an input, SY0 halts emulation when it is set low (0) by an external device. To enable SY0 as a halt factor (input), include it in one of the forms of the GR or GO command, discussed later in this section. To enable SY0 as an output, use the command ENABLE SY0 OUT. Both SY0 IN and SY0 OUT are initially disabled.

Setting The Go-Register

To enable a combination of BR0, BR1, and SY0 IN as a halt condition, you can use a set GR command of the form:

```
GR = halt-go-condition
```

The meta-term *halt-go-condition* means any one of the types of entry shown in Table 5-15. Table 5-15 gives all the forms of halt condition as they are to be entered, including the token TILL where applicable. The table contains all valid combinations of the halt factors.

Note that the entry TILL BR is equivalent to BR0 OR BR1. You cannot specify a match on BR0 AND BR1 simultaneously. The entry OR SY0 enables a halt on either SY0 high or a breakpoint match.

With the form of the set GR command just given, any breakpoint registers named must have received their settings from previous commands. The following form of the GR command can be used to set *and* enable one or both registers.

```
GR = TILL match [OR match] [OR SY0]
```

The meta-term *match* means one of the following two forms of entries:

mnemonic-match	Any of the forms shown in Table 5-13 and Table 5-14.
| *numeric* | ON *channel-list* | *mask* |	Discussed above under Set Breakpoint Register commands.

Both these forms have the same effect in the set GR command: the *match* given first (after TILL) sets the designated bits in BR0, and any bits in that register not specifically mentioned are reset to don't-care; then the GR is set to show that BR0 is enabled. The second *match* (after OR) similarly sets and enables BR1. The entry OR SY0 was discussed above.

You can also enable BR0, BR1, and SY0 IN by including them in a form of the GO command. The form is:

```
GO [FROM address] [halt-go-condition]
```

Table 5-15. Halt Conditions in the GO-Register (GR)

halt-go-condition	GO-Register				
	BR0	BR1	BR	SY0	OR SY0
FOREVER					
TILL BR0	E				
TILL BR1		E			
TILL BR0 OR BR1 TILL BR			E		
TILL SY0				E	
TILL BR0 OR SY0	E				E
TILL BR1 OR SY0		E			E
TILL BR0 OR BR1 OR SY0 TILL BR OR SY0			E		E

NOTE:
E = enabled
blank = not enabled
BR = BR0 OR BR1 (logical OR)

Where *address* is as discussed previously, and *halt-go-condition* is one of the entries (including the TILL) shown in Table 5-15. For example, the single GO command:

```
GO FROM 3000H TILL BR0 OR SY0
```

has the same effect as the pair of commands:

```
GR = TILL BR0 OR SY0
GO FROM 3000H
```

You can also set and enable one or both breakpoint registers, and/or enable SY0, with a GO command of the form:

GO [FROM*address*] TILL *match* [OR *match*] [OR SY0]

The first *match* sets and enables BR0; the second (if present) sets and enables BR1.

To display the current setting of the GO-register, enter the token GR followed by a carriage return. The *halt-go-condition* currently in GR is displayed. For example, suppose you had previously entered the command:

```
GO TILL LOCATION AOFFH READ OR SY0
```

Now, if you enter the command GR, the following is displayed.

```
TILL BR0 OR SY0
```

Note that the setting of BR0 is not part of the GO-register setting. To display the setting of a breakpoint register, enter the name of the register followed by carriage return. The display shows the setting of any system of user-defined groups that have at least one care bit (0 or 1) set; the setting of each group is displayed as a masked binary number. To display the setting given above for BR0, enter the command BR0 followed by a carriage return. The display obtained is the following; the parenthesized remarks are not part of the display.

```
BR0:                            (Remarks)

ADDRH=10100000
ADDR=1010000011111111           (A0FFH)
DATA=11111111
ADDRL=11111111
STS=010                         (READ)
DMUX=0                          (LOCATION)
```

Any don't-care bits in the groups are displayed as X.

Emulation Timer

An emulation timer is enabled when trace data is being collected. The timer can be used to determine how long it takes ICE-85 to emulate a given segment of code. The timer is a 2-MHz clock (i.e., counts are intervals of 500 ns), derived from the crystal on the Control board.

With trace enabled, the timer starts when the GO command is entered, starting emulation. The timer starts counting at the first T3 state of the first instruction emulated.

The timer is reset to 0 (before starting to count) when the GO command is entered with a FROM clause. If you want to reset the timer without changing the current program counter, enter a command such as GO FROM PC.

The timer continues counting until the first match on breakpoint BR0, halting at the instant the match is made. The timer halts on the first match on BR0, whether or not BR0 has been enabled to halt emulation.

After emulation halts, you can display the value of the timer in the current output radix. The display command TIMER displays the low 16 bits of the timer value; the command HTIMER displays the high 16 bits of the timer value. The tokens TIMER and HTIMER can also be used as keyword references in commands and expressions (see Table 5-8).

One common use for the timer facility is to estimate the deviation from real-time speed that occurs when ICE-85 emulates from Intellec memory. Each bus access incurs several wait states when Intellec shared or unshared memory is used for user program code. System speed degradation under these conditions is not a constant percentage, since each instruction takes a different number of cycles and number of bus accesses.

With the timer, you can measure the real elapsed time required to emulate a given code sequence. The elapsed time can then be compared to the calculated time based on the number of clock states in each instruction and the speed of the system clock. Note that code mapped to user runs at real-time; the timer value for code mapped to prototype memory is the real-time value.

The ICE-85 timer requires an initial run to initialize its operation at the beginning of emulation or after a RESET HARDWARE command. Thus, you should run the timer for some short period at the start of the emulation session, and discard this value. The timer is now initialized. Further, the timer is derived from the crystal on the Control board; the longer the emulation timed, the closer to the accuracy of this crystal the timer becomes.

Although the timer may halt when emulation halts, accuracy is preserved only when a match on BR0 is used to halt the timer. Note that if BR0 contains all don't-care bits as initialized, the timer halts immediately after emulation starts; under this condition, the timer value is meaningless.

GO Command

```
(1)     GO [FROM address] [halt-go-conditions]
(2)     GO [FROM address] [TILL match [OR match] [OR SY0] ]
```

Examples:

```
GO
GO FROM 3000H
GO FROM .START TILL BR0
GO FROM 3000H TILL LOCATION A0FF READ OR SY0
GO TILL LOCATION A0FFH READ OR 11Y ON 1,2
GO FROM #56 TILL BR
```

GO	Command keyword that starts emulation, subject to the current start and halt conditions.
FROM	Keyword introducing a starting address.
address	One of the following types of entries, to be used as the starting address for the emulation. *numeric-constant* *numeric-expression* *status-register* *processor-register* *symbolic-reference* *statement-reference* *(mem-type address)*
halt-go-condition	One of the types of entry (FOREVER or TILL-clause) shown in Table 5-15.
TILL	A keyword introducing one or more match or halt conditions.
match	One of the two following forms of breakpoint register match settings. (1) *mnemonic-match* (Table 5-13) (2) | *numeric* / *mask* | ON *channel-list*
numeric	A *numeric-constant* or *numeric-expression*, (see Chapter 4).
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas.
OR SY0	Low state on external signal SY0 OR breakpoint match enabled to halt emulation.

Set GO-Register (GR) Command

```
(1)     GR = halt-go-condition
(2)     GR = TILL match [OR match] [OR SY0]
```

Examples:

```
        GR = TILL BR0 OR BR1
        GR = TILL SY0
        GR = TILL BR WITH SY0
        GR = TILL LOCATION A0FFH READ
        GR = TILL LOCATION A0FFH READ OR 11Y ON 1,2
```

GR	Command token referring to the GO-register (halting conditions for real-time emulation).
halt-go-condition	One of the types of entry (FOREVER or TILL-clause) shown in Table 5-15.
TILL	A keyword introducing one or more match or halt conditions.
match	One of the two following forms of breakpoint register match settings. (1) *mnemonic-match* (Table 5-12) (2) |*numeric* / *mask*| ON *channel-list*
numeric	A *numeric-constant* or *numeric-expression*; (see Chapter 4).
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas.
OR SY0	Low state on external signal SY0 OR breakpoint match enabled to halt emulation.

Display Real-Time Emulation Registers Commands

```
GR
break-reg [channel-list]
```

Examples:

```
GR
BR0
BR1 31,32,DATA
BR
```

GR	A command keyword that displays the content of the GO-register (factors enabled to halt emulation).
break-reg	One of the breakpoint registers BR0 or BR1, to obtain a display of the setting, or the token BR to display the settings of both registers. The display includes only those groups that have at least one care bit set (zero or one).
channel-list	A list of channel numbers and/or channel group names, separated by commas. When this entry is included, only the settings of the bits corresponding to the channels in the list are displayed.

Set Breakpoint Register Command

(1) *break-reg = match*

(2) *break-reg channel-list* = $\left|\begin{matrix}\textit{numeric}\\ \textit{mask}\end{matrix}\right|$

NOTE: Form (1) resets to don't-care any channel bits in the register that are not specified. Form (2) does not reset any channel bits that are not included in the *channel-list*

Examples:

```
BR0 = LOCATION A0FFH READ
BR1 = 11Y ON 1,2
BR0 1,2,STS = 11000Y
```

break-reg	The name of one of the breakpoint registers (BR0, BR1), or BR to set both registers to the same match setting.
match	One of the two following forms of breakpoint register match settings. (1) *mnemonic-match* (Table 5-12) (2) $\left|\begin{matrix}\textit{numeric}\\ \textit{mask}\end{matrix}\right|$ ON *channel-list*
numeric	A *numeric-constant* or *numeric-expression*, (see Chapter 4).
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas.

RESET Breakpoint Register Command

```
RESET break-reg [channel-list]
```

Examples:

```
RESET BR0
RESET BR1 31,32,DATA
RESET BR
```

RESET	Command keyword that resets its object to an initial state. The reset state of any breakpoint register bit is X (don't-care).
break-reg	Either of the tokens BR0 or BR1 to reset a single breakpoint as named, or the token BR to reset both BR0 and BR1.
channel-list	A list of channel numbers and/or channel group names, separated by commas. When this entry is included in the RESET command, only the register bits in the list are reset, and bits not mentioned retain their previous settings. If *channel-list* is not included, all bits in the *break-reg* are reset to don't-care.

ENABLE/DISABLE SY0 OUT Commands

```
ENABLE SY0 OUT
DISABLE SY0 OUT
```

ENABLE SY0 OUT	Specifies that ICE-85 is to set external signal SY0 low when emulation halts, and hold it low until it is set high by an external device.
DISABLE SY0 OUT	Restores SY0 OUT to its initial condition.

Trace Control Commands

ICE-85 can record the value of each of the channels in all system and user-defined groups in a trace data buffer, during real-time emulation. The commands in this section allow you to control the display of trace data, and specify conditions for enabling and disabling trace data collection during emulation.

The commands in this section are as follows.

Command	Purpose	Page
Set TRACE Display Mode	Cause trace data to be displayed as frames, cycles, or instructions.	5-84
MOVE, OLDEST, NEWEST	Set trace buffer pointer to entry to be displayed.	5-85
PRINT	Display one or more entries from the trace buffer.	5-86
Set Qualifier Register	Specify channel match setting to be recorded.	5-87
RESET Qualifier Register	Reset channel match setting to match on every frame.	5-87
Display TRACE Controls	Display current TRACE Display mode, qualifier register settings.	5-88
ENABLE/DISABLE Trace Factors	Enable or disable SY1 IN, SY1 OUT, and STOPTRACE.	5-88

Discussion

The 'unit' of emulation is the *instruction*. Each instruction executed represents one or more machine *cycles*. Each cycle contains two *frames*: the first frame in each cycle is the period of time during which the 8085 lines AD0 through AD7 represent the low address byte; the second frame in a cycle is the time period during which those lines represent a byte of data. The *frame* is the 'unit' of trace data collection; trace can be enabled or disabled on any given frame, or you can collect as many consecutive frames as the buffer will hold.

The buffer contains a maximum of 1022 frames, or 511 cycles, of trace data. Each frame in the buffer contains the values of all the channels in all system-defined and user-defined groups. The trace buffer *pointer* controls the display of data from the buffer.

The buffer is cleared of data, and the buffer pointer reset to the 'top' (just before earliest entry) after any command that changes the program counter. Commands that change PC are the GO FROM and STEP FROM commands, and the Set Program Counter command (PC = *address*). The buffer is initially empty.

During emulation (GO or STEP) any new trace data collected is appended to any already in the buffer, and the most recent 1022 frames are retained in the buffer. After the emulation halts, the buffer pointer is at the 'bottom' of the buffer, just past the most recently collected frame.

Trace Display Mode

The trace display mode controls the size of an *entry* to be displayed or located in the trace buffer. An *entry* can be a frame, a cycle, or an instruction. The initial trace display mode is INSTRUCTION. To set the trace display mode, use one of the following commands.

```
TRACE = FRAME
TRACE = CYCLE
TRACE = INSTRUCTION
```

To display an entry from the buffer, move the pointer to the desired entry and enter a PRINT command.

Moving The Buffer Pointer

The pointer movement commands are MOVE, OLDEST, and NEWEST.

The command OLDEST (followed by carriage return) moves the pointer to the top of the buffer, in any trace display mode. The NEWEST command moves the pointer to the bottom of the buffer.

The MOVE command has the following form:

MOVE [[+ / −] *decimal*]

The meta-term *decimal* means any numeric quantity; if no explicit input-radix is given, ICE-85 assumes decimal radix. The value of *decimal* is the number of entries between the current pointer position and the desired position. Movement in a plus (+) direction is toward the bottom of the buffer; if neither (+) nor (−) is entered, a downward movement is assumed as the default. Movement in a minus (−) direction is toward the top of the buffer. The size of the move does not count the entry under the pointer when the MOVE command is given.

For example, assuming FRAME mode, if the pointer is pointing at frame 100 and you issue the command 'MOVE 10', the pointer is moved to point to frame 110. Under the same initial conditions, if you issue the command 'MOVE −10', the pointer is moved to point to frame 90. If *decimal-number* is larger than the number of entries between the current pointer location and the bottom (for '+') or top (for '−'), the pointer is moved only to the bottom or top, respectively. In short, you cannot move the pointer outside the range of buffer locations.

If the MOVE command has no decimal number following it, 'MOVE 1' is executed.

The trace display mode in effect controls the size of each move. Under FRAME mode, the command MOVE 10 moves down ten frames; under CYCLE, the same command moves down ten cycles; under instruction, the same command moves down ten instructions.

Displaying Trace Data

The PRINT command displays one or more entries from the buffer. This command has the form:

PRINT [[+ / −] *decimal*]

The meta-term *decimal* was defined under the MOVE command.

With (+) or no sign, *decimal* entries lower (toward the bottom) than the current pointer position are displayed. With (−), decimal entries above (toward the top) the current pointer position are displayed. The command PRINT without a *decimal* modifier is equivalent to PRINT 1 (one entry is displayed).

The PRINT command displays the number of entries requested, then moves the pointer to point to the next entry just past the last one displayed. As an illustration, the commands:

```
OLDEST
PRINT 10
PRINT 10
```

are equivalent to the commands

```
OLDEST
PRINT 20
```

The command PRINT ALL displays the entire trace buffer; PRINT ALL is equivalent to the commands:

```
OLDEST
PRINT 1022
```

Figure 5-8 shows two instructions as they are displayed in each of the three trace display modes. The headers shown are the ones displayed in the different modes. The interpretation of the headers is given in Table 5-16.

Instruction Mode:

```
       ADDR INSTRUCTION           ADDR-S-DA ADDR-S-DA ADDR-S-DA ADDR-S-DA
1005:  382F  RET                  3881-R-DF  3882-R-36
1011:  36DF  LXI  H, 3883
```

Cycle Mode:

```
       ADDR-S-DA SD   RW   M
1005:  382F-E-C9  00   10   1
1007:  3881-R-DF  00   10   1
1009:  3882-R-36  00   10   1
1011:  36DF-E-21  00   10   1
1013:  36E0-R-83  00   10   1
1015:  36E1-R-38  00   10   1
```

Frame Mode:

```
       A/D   S  D  SD   RW   M
1004:  382F  E  0  00   00   1
1005:  38C9  E  1  00   10   1
1006:  3881  R  0  00   00   1
1007:  38DF  R  1  00   10   1
1008:  3882  R  0  00   00   1
1009:  3836  R  1  00   10   1
1010:  36DF  E  0  00   00   1
1011:  3621  E  1  00   10   1
1012:  36E0  R  0  00   00   1
1013:  3683  R  1  00   10   1
1014:  36E1  R  0  00   00   1
1015:  3638  R  1  00   10   1
```

Figure 5-8. Trace Data Displays

In INSTRUCTION mode, the frame number at the left of each line is the second frame in the first cycle under that instruction. Each line of the display gives the address of the instruction, the assembler mnemonic for the instruction executed, and may show additional cycles (memory reads, for example) under the headers ADDR-S-DA.

In CYCLE mode, the frame number at the left is the second frame in each cycle. Each line contains the settings of the system (and user) groups produced by the cycle. For example, the cycle displayed after frame number 1005 in Figure 5-8 shows the opcode fetch for the RET instruction.

In FRAME mode, each line of the display is a frame as collected by ICE-85. Frames where D (DMUX) = 0 are address frames; those with D = 1 show the byte of data appearing in the low-order byte of A/D. Thus, the complete opcode fetch shown on one line of CYCLE mode takes two frames in FRAME mode; the corresponding frames are 1004 and 1005.

Table 5-16. Trace Display Headers

Header	**Interpretation**	**Radix**
ADDR	System group ADDR; address channels 35 - 20, DMUX (ALE) = 0.	Hexadecimal
INSTRUCTION	Assembler mnemonic for instruction.	None
DA	System group DATA; data channels 27 - 20, DMUX (ALE) = 1.	Hexadecimal
A/D	Word formed from ADDRH, ADDRL when DMUX = 0; ADDRH, DATA when DMUX = 1.	Hexadecimal
S	STS group (38 - 36), interpreted as follows.	None

Display	*status*	Channels 38	37	36
H	HALT	0	0	0
W	WRITTEN	0	0	1
R	READ	0	1	0
E	EXECUTED	0	1	1
O	OUTPUT	1	0	1
I	INPUT	1	1	0

Header	**Interpretation**	**Radix**
D	System group DMUX. When DMUX = 1, DATA (27 - 20) group is valid; when DMUX = 0, ADDRL is valid.	Binary
SD	System group SD; left bit = SID, right bit = SOD.	Binary
RW	System group RW; left bit = RD, right bit = WR.	Binary
M	System group MTH; M = 1 if either breakpoint register matched in the previous frame.	Binary
user-groups	*Group-names* defined by the user; each group-name is truncated to the number of digits in the display for that group.	*group-radix* given by IN-clause, or Hexadecimal (default)

INSTRUCTION mode is the initial mode. Only one mode can be in effect at a given time.

INSTRUCTION mode is useful for displaying a series of instructions. CYCLE mode is useful for displaying individual read and write operations. FRAME mode is useful for identifying the particular frame where some event (such as a breakpoint match) occurred.

Note that trace data is always collected as frames; cycles and instructions are reconstructed from the frames collected by the ICE-85 program. As discussed later on in this section, trace collection can be enabled or disabled on a frame-by-frame basis. When individual frames rather than complete cycles or instructions are recorded, the reconstruction of these frames into cycles or instructions may produce a meaningless display.

Trace collection is initially enabled on every frame when emulation is running. The trace factors that can be used to control trace are as follows.

Trace Control Factors

The qualifier registers QR0 and QR1 are identical to the breakpoint registers BR0 and BR1 in structure. Initially, both qualifier registers are set to all don't-care bits. Both qualifier registers are always enabled to control trace. When either qualifier register matches on a frame (subject to the other trace factors to be discussed), that frame is recorded in the buffer. The initial setting of both qualifier registers (all bits don't-care) matches on every frame. Note that setting just one of the two qualifier registers to contain some zero or one bits, without setting the other register, has no effect; the register with all don't-care bits still qualifies trace on every frame.

Each of the two qualifier registers contains 42 bits corresponding to the 18 user probe channels and the 24 processor channels. The bits are duplicated as necessary to represent all the system-defined and user-defined channel groups. (Refer to Table 5-12 for a description of the channels and system-defined groups.)

Qualifier register bits can be set to 0 (zero), 1 (one), or X (don't-care). A don't-care bit matches either a zero or a one in its corresponding channel.

ICE-85 maintains a pseudo-register, the Channel Status Register (CSR), that contains the current actual value of all the channels. The CSR is updated after every frame during real-time emulation. Each machine cycle represents two frames: the first frame is the interval of time when the 8085 lines AD0 through AD7 represent the low address byte, and the second frame is the succeeding interval when those lines represent a byte of data. On each frame during emulation, ICE-85 compares the qualifier register settings with the CSR. If either qualifier register matches the CSR on a frame, that frame is recorded in the trace buffer.

ICE-85 offers several ways to set the qualifier registers. The simplest way is to use a command with the form:

qual-reg = mnemonic-match

The meta-term *qual-reg* means either of the tokens QR0 or QR1, or the token QR to set both qualifier registers to the same match setting.

The meta-term *mnemonic-match* means one of the forms shown in Table 5-17. The entries in this table are explained in the following paragraphs.

For many applications, the channels of interest in controlling emulation are the ones contained in system channel groups ADDR, DATA, DMUX, and STS (refer to Table 5-12). The forms of *mnemonic-match* give you short and simple ways to specify settings for these groups.

A command such as QR0 = HALT resets all bits in the named register to don't-care, then sets the three bits in system group STS to zeroes. This setting represents the 8085 HALT state.

Table 5-17. Mnemonic Match Conditions

Mnemonic-Match	**System Channel Groups**			
	DMUX	**ADDR**	**DATA**	**STS**
HALT	XY	XXXXH	XXH	000Y
[LOCATION] *address*	0	*address*	XXH	XXXY
[LOCATION] *address status*	0	*address*	XXH	*status*
[LOCATION] *address-mask*	0	*address-mask*	XXH	XXXY
[LOCATION] *address-mask status*	0	*address-mask*	XXH	*status*
VALUE *data*	1	XXXXH	*data*	XXXY
VALUE *data status*	1	XXXXH	*data*	*status*
VALUE *data-mask*	1	XXXXH	*data-mask*	XXXY
VALUE *data-mask status*	1	XXXXH	*data-mask*	*status*

NOTE:

X = don't-care digit in radix shown
Y = binary radix
H = hexadecimal radix
0 = zero (any radix)
1 = one (any radix)

Table 5-18. Status Group Bit Settings

Status	**STS Bits (Channels)**		
	IO/$\overline{\text{M}}$(38)	**S1(37)**	**S0(36)**
HALT	0	0	0
WRITTEN	0	0	1
READ	0	1	0
EXECUTED	0	1	1
OUTPUT	1	0	1
INPUT	1	1	0

To match on an *address,* use one of the four forms of *mnemonic-match* that uses the token LOCATION. The forms of *address* are the ones discussed previously in this section.

The meta-term *address-mask* is a *masked-constant*. A masked constant is a number containing one or more don't-care digits. In binary radix, each X digit represents one bit; in octal radix, each X digit stands for three adjacent bits; in hexadecimal radix, each X digit represents four adjacent bits.

The meta-term *status* means one of the types of entries shown in Table 5-18. If no *status* is specified, the three bits in STS are set to don't-care, and any action involving the *address*(or *data*, discussed later on) produces a match.

The token LOCATION may be omitted from the command. If neither LOCATION nor VALUE is entered, LOCATION is assumed as the default, and the first value given is treated as an *address* or *address-mask*.

To specify a match on a DATA value, use one of the forms of *mnemonic-match* that include the token VALUE. The meta-term *data* is identical to *address* in the forms it can take, but is a one-byte quantity. The meta-term *data-mask* is likewise identical in form to *address-mask*, and the meta-term *status* is as shown in Table 5-17.

Here are a few examples of setting qualifier registers using LOCATION and VALUE.

To match any access to location 3000H, you can enter:

```
QR0 = LOCATION 3000H
```

Or you can simply enter:

```
QR0 = 3000H
```

since the token LOCATION is the default.

To match one a memory write to location 3000H, you can enter:

```
QR1 = LOCATION 3000H WRITTEN
```

To match on a memory read from any location in the range from 3000H to 30FFH, you can enter:

```
QR0 = LOCATION 30XXH READ
```

To match on any frame in which DATA value 11H occurs, enter:

```
QR1 = VALUE 11H
```

To specify a match on any data value read from any input port, enter:

```
QR0 = VALUE XXH INPUT
```

Two points to remember when using this form of the Set Qualifier Register command are:

(1) Anytime you use LOCATION, VALUE, or HALT, ICE-85 resets all 42 bits in the named qualifier register to don't-care, then sets the DMUX, DATA, ADDR, and STS bits as specified.

(2) An *address* match includes a match on DMUX = 0, and a *data* match includes a match on DMUX = 1.

In addition to the command forms involving LOCATION, VALUE, and HALT, ICE-85 offers two ways to set individual bits and specific groups directly. One way to do this is to use a command of the form:

$$\textit{qual-reg} = \left|\begin{array}{l}\textit{numeric}\\ \textit{mask}\end{array}\right| \text{ ON } \textit{channel-list}$$

The meta-terms in this command form are as follows.

qual-reg	QR0 or QR1, or QR to set both registers to the same match value.
numeric	Commonly, a *numeric-constant* or *numeric-expression.* Generally, any of the forms of *operand.* See Chapter 4 for details.
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas; see channel group commands earlier in this chapter for details.

The channel list tells ICE-85 which bits in the register to set. If a channel group is included in the list, the channels assigned to that group are set in the order they appear in the group. ICE-85 'decomposes' the included group into its channels, then inserts those channels into the list at the place in the list where the group was named.

The channel list can contain a maximum of 16 channels.

The last (rightmost) channel bit in the list receives the value of the least significant bit in the *numeric* or *mask*. Succeeding channel bits in the list are then set to corresponding bits in the *numeric* or *mask* in right-to-left order. If the numeric or mask represents fewer bits than there are channels in the list, the rightmost channel bits in the list are set to their corresponding bits in the numeric or mask, and the remaining channel bits in the list are set to *zero*. If the numeric or mask contains more bits than there are channels in the list, the channel bits are set from right-to-left until all have received values; the extra leftmost bits are lost.

Any bits in the qualifier register whose corresponding channels are not included in the channel list are reset to *don't-care* by this form of the Set Qualifier Register command.

For example, suppose you are monitoring two signals from your prototype with user probe channels 1 and 2, and you want trace to run whenever both signals are high (1) simultaneously. You can use the following command to set this condition into QR0.

```
QR0 = 11Y ON 1,2
```

All the other bits in QR0 except those corresponding to channel 1 and 2 are reset to don't-care as a result of this command. Note that in order to record only those frames in which the match on channels 1 and 2 occurs, QR1 must be set to some setting other than don't-care; setting both registers to the same match value also produces this result.

The last form of the Set Qualifier Register command differs from the two previous forms in that it does *not* reset any other channels or groups to don't-care. The form is:

$$\textit{qual-reg channel-list} = \left| \begin{array}{l} \textit{numeric} \\ \textit{mask} \end{array} \right|$$

All the meta-terms in this form have been discussed previously.

With this form of the command, only the qualifier register bits corresponding to the channels in *channel-list* receive new values from the numeric or mask. Any qualifier bits (or groups) whose corresponding channels are *not* included in the channel-list retain whatever setting they had before this form of the command was entered.

To reset some or all of the bits in a qualifier register to don't-care, use a command with the form:

```
RESET qual-reg [channel-list]
```

If *channel-list* is included in the command, only those bits corresponding to channels in the list are reset to don't-care, and any other bits retain their previous settings. If no channel list is given, all 42 bits are reset in the *qual-reg*.

To summarize trace control as discussed so far, trace data is collected on any frame when emulation is running AND either qualifier register matches.

The other two trace factors are STOPTRACE and SY1 IN. These two factors can be combined to turn trace on and off, given that emulation is running and one of the qualifier registers matches.

STOPTRACE is initially disabled. To enable this factor, use the command ENABLE STOPTRACE. To disable it after it has been enabled, enter the command DISABLE STOPTRACE. When STOPTRACE is enabled, a match on either of the two breakpoint registers BR0 or BR1, at any time during the emulation, halts trace data collection. STOPTRACE halts trace until something else restarts it; in other words, trace halts when the breakpoint matches but does not restart on later frames, even though the breakpoint does not match on those later frames. STOPTRACE does not apply to single-stepping.

External signal SY1 can be enabled as an input to ICE-85 from an external device. This trace factor is called SY1 IN. When SY1 IN is enabled, trace is forced by setting SY1 high via the external source; emulation must be running, and one or both qualifier registers must match, to allow SY1 IN to force trace. In other words, SY1 IN can force trace over STOPTRACE, but not over the lack of a qualifier register match. SY1 IN is initially disabled. To enable this factor, enter ENABLE SY1 IN. To disable it again, enter DISABLE SY1 IN.

Independent of SY1 IN, you can use the same signal as an output by entering the command ENABLE SY1 OUT. SY1 OUT then is set high by ICE-85 whenever trace data is being collected. To disable it again, enter DISABLE SY1 OUT. SY1 OUT is not a factor in controlling trace.

The operation of the trace control factors can be summarized in the following logic equation.

$$T = E \cdot Q \cdot [(\overline{ST} + ST \cdot \overline{B}) + (\overline{S1} + S1 \cdot S1H)]$$

NOTE

The instruction mode of trace is not guaranteed to work properly if you are setting the QR with anything but don't cares or if you are using an oscillating SY1 when it is enabled.

Where:

$\cdot$	= Logical AND (higher precedence than OR)
$+$	= Logical OR
T	= Trace data is collected.
E	= Emulation is running.
Q	= One or both qualifier registers match on the frame.
$\overline{ST}$	= STOPTRACE is not enabled.
ST	= STOPTRACE is enabled.
$\overline{B}$	= Neither breakpoint register has matched since emulation began.
$\overline{S1}$	= SY1 IN is not enabled.
S1	= SY1 IN is enabled.
S1H	= SY1 IN is high.

Set TRACE Display Mode Command

```
TRACE =   |FRAME      |
          |CYCLE      |
          |INSTRUCTION|
```

Examples:

```
TRACE = FRAME
TRACE = CYCLE
TRACE = INSTRUCTION
```

TRACE	A command keyword indicating that the mode of display for trace data is to be set.
FRAME	A function keyword indicating that data in the trace buffer is to be displayed frame by frame.
CYCLE	A function keyword indicating that data in the trace buffer is to be displayed by machine cycles. One machine cycle is equivalent to two frames.
INSTRUCTION	A function keyword indicating that data in the trace buffer is to be displayed by instructions. Each instruction is equivalent to one or more machine cycles.

MOVE, OLDEST, and NEWEST Commands

```
MOVE [ [ +/ -] decimal]
OLDEST
NEWEST
```

Examples:

```
MOVE
MOVE +6
MOVE -11
OLDEST
NEWEST
```

MOVE	A command keyword that moves the buffer pointer one or more entries forward (toward the most recent entries) or backward (toward the earliest entries). An entry is a frame, cycle, or instruction, depending on the TRACE mode in effect.
+	A unary operator specifying a forward movement. Plus is the default.
−	A unary operator specifying a backward movement.
decimal	A number, evaluated in decimal radix (if no explicit suffix is given), that gives the number of entries to be included in the MOVE.
OLDEST	A command keyword that moves the pointer to the earliest entry in the buffer.
NEWEST	A command keyword that moves the pointer to the latest entry in the buffer.

PRINT Command

```
(1) PRINT ALL
(2) PRINT [[+ / -] decimal]
```

Examples:

```
PRINT
PRINT ALL
PRINT +5
PRINT 5
PRINT -10
```

PRINT	A command keyword calling for a display of one or more entries from the trace data buffer. The entries are displayed as frames, cycles, or instructions, depending on the current trace mode.
ALL	A function keyword indicating that the entire trace buffer contents are to be displayed.
+	A unary operator directing the display of *decimal* entries below (entered later than) the current buffer pointer location. See DISCUSSION for details. Plus is the default.
–	A unary operator directing the display of decimal-number entries above (entered earlier than) the current buffer pointer location. See DISCUSSION for details.
decimal	A numeric constant, evaluated in decimal suffix, giving the number of entries to be displayed.

Set Qualifier Register Command

(1) *qual-reg = match*

(2) *qual-reg channel-list* = $\left|\begin{matrix}\textit{numeric} \\ \textit{mask}\end{matrix}\right|$

NOTE: Form (1) resets to don't-care any channel bits in the register that are not specified. Form (2) does not reset any channel bits that are not included in the *channel-list*.

Examples:

```
QR0 = LOCATION A0FFH READ
QR1 = 11Y ON 1,2
QR0 1,2,STS = 11000Y
```

qual-reg	The name of one of the qualifier registers (QR0, QR1), or QR to set both registers to the same match setting.
match	One of the two following forms of qualifier register match settings. (1) *mnemonic-match* (Table 5-17) (2) $\left|\begin{matrix}\textit{numeric} \\ \textit{mask}\end{matrix}\right|$ ON *channel-list*
numeric	A *numeric-constant* or *numeric-expression*; (see Chapter 4).
mask	A *masked-constant* (see Chapter 4).
channel-list	A list of channel numbers and/or channel group names, separated by commas.

RESET Qualifier Register Command

RESET *qual-reg* [*channel-list*]

Examples:

```
RESET QR1 1, 13, 15
RESET QR
```

RESET	Command keyword restoring its object to a reset condition.
qual-reg	QR0 or QR1 to reset one or more bits in a single qualification register, or QR to reset one or more corresponding bits in both qualification registers.
channel-list	A list of channel group numbers (1 to 42), and/or channel group names, separated by commas. When *channel-list* is included, only the bits in the breakpoint or qualification register corresponding to the channels in the list are reset to 'don't-care'. If no channel-list is included, all bits in the breakpoint or qualification register are reset to 'don't-care'.

Display Trace Controls Commands

```
TRACE
qual-reg [channel-list]
```

Examples:

```
TRACE
QR0
QR1, 31,32
```

TRACE	A command keyword that displays the current TRACE mode (F for FRAME, C for CYCLE, I for INSTRUCTION).
qual-reg	The name of one of the two qualification registers QR0 or QR1, to display the qualification setting of that register, or the token QR to display the settings of both registers. The display includes only those groups that have at least one care bit set.
channel-list	A list of channels or channel group names, separated by commas. When a *channel-list* is given, the settings of those channels are displayed as a single masked number.

ENABLE/DISABLE Trace Factors Commands

```
|ENABLE |      |SY1 IN    |
|DISABLE|      |SY1 OUT   |
               |STOPTRACE |
```

Examples:

```
ENABLE SY1 OUT
ENABLE STOPTRACE
DISABLE SY1 IN
```

ENABLE	A command keyword that activates its object as a controlling factor for trace.
DISABLE	A command keyword that cancels the effect of its object on trace.
SY1 OUT	A keyword clause that, when enabled, sets external synchronization line SY1 high whenever trace is running.
SY1 IN	Enables SY1 as an input to force trace when SY1 is set high by an external source.
STOPTRACE	A function keyword that, when enabled, causes trace to stop when either of the two breakpoint registers (BR0, BR1) has matched the state of the 42 channels since emulation began.

Single Step Emulation Control Commands

During single step emulation, the emulation processor performs the user program instructions under the control of the single step commands. The commands in this section permit you to specify the starting address where single stepping is to begin, and to specify and display the control conditions for halting processing and returning control to the console for further commands.

The commands in this section are as follows.

Command	Purpose	Page
Set Condition-Register	Set match for halting single stepping.	5-96
SR command	Enable or set and enable condition registers to halt single stepping.	5-97
STEP command	Begin single step emulation.	5-98
Display Single Step Controls	Display STEP-register and condition register settings.	5-99
ENABLE/DISABLE DUMP	Enable or disable or enable automatic display.	5-100

Discussion

The single step control commands tell ICE-85 where to start single step emulation and where to halt single stepping.

To initialize for emulation, you map the locations in prototype and Intellec memory that are to be accessible to ICE-85, and load your program code into mapped locations. After the code has been loaded, ICE-85 initializes for emulation as follows.

- The program counter (PC) is loaded with the address of the first executable instruction in your program.
- The STEP-register (SR) is set to FOREVER. The setting of SR identifies the combination of factors that are enabled to halt single stepping. The setting FOREVER means no factors are enabled.
- The condition registers (CR0, CR1, CR2, and CR3) are cleared and disabled.
- The parameter, COUNT, that controls the maximum number of single steps to be executed is ignored until loaded with a specific value.
- Automatic display is disabled.

Now you can begin single stepping by entering the command STEP, followed by a carriage return. At the command STEP, the following occurs.

- Single stepping begins with the instruction at the address that is in the PC; this is the first executable instruction in your program after initialization.
- Emulation will continue until you press the ESC key, or until a fatal error occurs (see Appendix B for error messages).

Now, if you press the ESC key, the following happens.

- ICE-85 completes executing the current instruction.
- Emulation halts with PC set to the address of the next instruction to be executed.
- The message EMULATION TERMINATED, PC = nnnnH is displayed. The value of PC displayed is the address of the next instruction to be executed.
- The message PROCESSING ABORTED is displayed, acknowledging the user abort (ESC key).

This is the simplest case of starting and stopping of single stepping. Whenever the STEP-register is set to FOREVER, you can enter the command STEP to start single stepping at the current PC address, and press the ESC key to halt emulation.

Instead of starting single stepping at the address currently in the PC, you may specify a new starting address. There are two ways to do this. You can set the PC directly to any desired address with a command of the form PC = *address*, then enter the STEP command to start emulation at that *address*. Or you can specify the starting address as part of the STEP command; this form of the STEP command is as follows.

```
STEP [FROM address]
```

The meta-term *address* means any one of the following types of entries.

numeric-constant	A number in any *input-radix.*
numeric-expression	A numeric expression is evaluated to give the address (see Chapter 4 for the forms of *numeric-expression* and *numeric-constant*).
status-register	Any of the keywords for ICE-85 status registers shown in Table 5-8. The content of the named register becomes the address.
processor-register	Any of the keywords for 8085 processor registers shown in Tables 5-4 through 5-7. The content of the named register becomes the address.
symbolic-reference	Any of the three forms of symbolic reference shown in Table 5-11. The symbol table value corresponding to the named symbol is used as the address.
(mem-type address)	In the STEP command, an address such as (WORD 1000) causes the content of location 1000H to be used as the address. The parentheses must be used to enclose this type of indirect reference.
statement-reference	Either of the forms of statement-reference shown in Table 5-11. The address of the first instruction generated by that source program statement is the address used.

For example, to start single stepping with the instruction at location 3000, you could enter:

```
PC = 3000H
STEP
```

Or, you could enter:

```
STEP FROM 3000H
```

The effect is the same either way.

Instead of accepting the default halting condition FOREVER, you can specify that a match on one or more of the condition registers is enabled to halt single stepping.

There are four condition registers, named CR0, CR1, CR2, and CR3. Each register can be loaded with a logical condition that can be used to halt stepping. To set a condition register to halt stepping, enter a conditional expression that represents a desired halt condition into the register with a set condition register command. Then *enable* the register by including it in a SR command or in a TILL clause in the STEP command itself.

Initially, all condition registers are set to don't-care conditions. This setting does not cause a halt.

ICE-85 provides several ways to set the condition registers. The simplest way is to use the Set Condition-Register command. It has the form:

condition-register = conditional-expression

The Set Condition-Register command sets a *condition register* to the *conditional- expression* given. One condition register can be set with one such command. Once a condition register has been set, it can be enabled to control single-stepping by including it in the STEP-register via a STEP or SR command. The Set Condition-Register command does not affect the STEP-register.

The *conditional-expression* has the form:

```
content-reference relational-operator    | content-reference  |
                                         | numeric-constant   |
                                         | numeric-expression |
```

Relational operators are as follows.

rel-op	Meaning
=	Is equal to
>	Is greater than
<	Is less than
>=	Is greater than or equal to
<=	Is less than or equal to
<>	Is not equal to

The *relational operator* divides the *conditional expression* into a left side and a right side.

The *content-reference* on the left side of the *conditional expression* can be any one of the following.

- *memory-type address*
- PORT *port-name*
- *processor-register*
- *status-register*

The term *content-reference* was chosen to describe the left side of the conditional expression to emphasize that if a symbolic reference is used, it must be preceded by a 'content-of' memory-type (BYTE, WORD, IBYTE, or IWORD). The memory or port *address* of a content-reference is evaluated only once, whereas the *content* of the addressed element is evaluated (examined) each time the condition is tested. The condition is tested after every instruction executed under STEP. A keyword reference such as PC on the left side refers to the contents of the hardware element (the program counter in this case) each time the condition is tested.

The right side of the conditional expression can be one of the following.

- *Content-reference* as described above.
- *Numeric-constant*
- *Numeric-expression*

The right side can be a reference to any system- or user-defined element, with or without a 'content-of' modifier. If the right side begins with a keyword, the entry must be a content-reference (not an expression). In this case, the condition refers to the current contents of the named element each time the condition is tested.

The right side can be a *numeric expression*. The expression must not begin with a keyword reference. *Masked constants cannot be used*. The expression is evaluated using the content of any reference as it stands at the time the condition is specified in a Set Condition-Register, SR, or STEP command. The evaluated result is then treated as a constant value; you cannot change the condition by later changing the content of one of the right side references in the original expression.

To enable any combination of CR0, CR1, CR2, and/or CR3 as a halt condition, you can use a SR (set STEP-Register) command of the form:

SR = *halt-step-condition*

The meta-term *halt-step-condition* means any one of the types of entry shown in Table 5-19. This table gives all the forms of halt conditions as they are to be entered, including the token TILL where applicable. The table contains all valid combinations of the halt factors.

Table 5-19. Halt Conditions in the Step Register (SR)

Halt Condition	STEP-Register COUNT	CR3	CR2	CR1	CR0
FOREVER					
COUNT n	[E]				
COUNT n TILL CR0	[E]				E
COUNT n TILL CR0 AND/OR CR1	[E]			E	E
COUNT n TILL CR0 AND/OR CR2	[E]		E		E
COUNT n TILL CR0 AND/OR CR1 AND/OR CR2	[E]		E	E	E
COUNT n TILL CR0 AND/OR CR3	[E]	E			E
COUNT n TILL CR0 AND/OR CR1 AND/OR CR3	[E]	E		E	E
COUNT n TILL CR0 AND/OR CR2 AND/OR CR3	[E]	E	E		E
COUNT n TILL CR0 AND/OR CR1 AND/OR CR2 AND/OR CR3	[E]	E	E	E	E
COUNT n TILL CR1	[E]			E	
COUNT n TILL CR1 AND/OR CR2	[E]		E	E	
COUNT n TILL CR1 AND/OR CR3	[E]	E		E	
COUNT n TILL CR1 AND/OR CR2 AND/OR CR3	[E]	E	E	E	
COUNT n TILL CR2	[E]		E		
COUNT n TILL CR2 AND/OR CR3	[E]	E	E		
COUNT n TILL CR3	[E]	E			
COUNT n TILL cond-exp	[E]				E
COUNT n TILL cond-exp AND/OR cond-exp	[E]			E	E
COUNT n TILL cond-exp AND/OR cond-exp AND/OR cond-exp	[E]		E	E	E
COUNT n TILL cond-exp AND/OR cond-exp AND/OR cond-exp AND/OR cond-exp	[E]	E	E	E	E

Note: E = ENABLED; blank = not enabled.
[E] = COUNT n is optional and may be omitted from any of the above halt conditions in the table. In this event COUNT = FOREVER.

The SR command can specify one or more conditional expressions to be loaded into the condition registers. The first such expression is loaded into CR0, and the second, third, and fourth expressions (if present) are loaded into CR1, CR2, and CR3, respectively. In a given SR command, you can specify either the CR's to be enabled, or the conditions to be loaded; the two forms cannot be combined in one SR command.

Each condition register holds a single conditional expression. AND takes precedence over OR.

The STEP-register includes both the COUNT clause and the TILL clause as halting conditions for single-step emulation. With the SR command, you can change or delete either or both of these clauses.

The initial state of the STEP-register is FOREVER. With this setting, the STEP command starts single-stepping, and continues until the user presses the ESC key or performs a hardware reset.

The COUNT clause establishes one condition for halting. The number *n* (following COUNT) is evaluated to a decimal number; single-stepping halts after that number of instructions has been executed, unless some other condition causes an earlier halt. In other words, the COUNT condition is implicitly OR'd with any other conditions given in the command. If STEP is entered with a TILL clause and no COUNT, then COUNT is set to FOREVER.

The TILL clause can include up to four condition registers (as previously set) or up to four conditional expressions. Each new TILL clause overrides any previous TILL clauses. If a STEP command is entered without a TILL clause, the conditions previously stored in the STEP-register are used. You cannot mix conditional expressions and condition registers in the same TILL clause.

Each of the four condition registers (CR0, CR1, CR2, CR3) can contain a conditional expression. When the state described by the combination of active condition registers becomes true, with AND > OR in precedence, single-step emulation halts after completing any instruction currently being executed.

Like the breakpoint registers used to control real-time emulation, each condition register must be both set and enabled to control single-step emulation. The SR and Set Condition-Register commands set the condition registers; the second form of the STEP command (with *TILL cond-reg*) enables the condition registers it names; the third form of the STEP command (with TILL cond-exp) both sets and enables one or more condition registers.

The halt conditions shown in Table 5-19 may also be set through the use of the STEP command. The STEP command causes ICE-85 to emulate your program at a single step or several steps at a time. Additionally, any conditions given in COUNT and TILL clauses are loaded into the STEP register for use by later STEP commands.

The operation of the STEP command parallels that of the GO command. The command can specify both a starting address and one or more halting conditions. Under STEP, trace data can be displayed after each instruction; refer to the ENABLE DUMP command for details on this feature of the STEP operation.

The STEP command has the general form:

STEP [FROM *address*] [*halt-step-condition*]

When the FROM clause is included, the value of *address* is loaded into the PC to start the single-step emulation. If the FROM clause is omitted, the current value of PC is used as the starting address.

The COUNT clause and any other conditions for halting single-step emulation (FOREVER or TILL clause) constitute the STEP-register.

The initial state of the STEP-register is FOREVER. With this setting, the STEP command starts single-stepping, and continues until the user presses the ESC key or performs a hardware reset.

The STEP command can specify up to four condition registers to be enabled. Each condition register holds a single conditional expression, as discussed previously.

Another form of the STEP command gives one more more conditional expressions to be loaded into the condition registers. The first such expression is loaded into CR0, and the second, third, and fourth expressions (if present) are loaded into CR1, CR2, and CR3 respectively. Any registers set with this form of the STEP command are also enabled in the SR.

For example:

```
STEP FROM .START COUNT 33 TILL .CRT = 55 AND .PRT <.SAM OR WORD.AA
> FFFFH AND BYTE. SAVE < E2H
```

In this event, the contents of the condition registers are as follows:

```
CR0: .CRT = 55
CR1: .PRT < .SAM
CR2: WORD.AA > FFFFH
CR3: BYTE.SAVE < E2H
```

The display STEP-register and display condition-register commands cause the display of the current setting of the STEP register and the designated condition register(s) respectively.

The SR and condition register display commands work exactly like the GR and breakpoint register display commands. Suppose you set the STEP-register as follows:

```
SR = COUNT 10 TILL PC=100 OR BYTE 1000 > AB
```

To display what you have done, enter the token SR followed by the carriage return. You get the following display.

```
COUNT 10 TILL CR0 OR CR1
```

To get the full details, you must display CR0 and CR1.

```
*CR0
PC=0100H

*CR1
BYTE 1000 > 00ABH
```

If you enter the token BR as a display command, both breakpoint register settings are displayed, using the same format as for a single register.

By contrast, the condition registers must be displayed singly; the token CR is invalid.

You can enable the automatic display of register contents and trace data after each instruction executed under single-step emulation. To enable this facility, use an ENABLE DUMP command. The form of this command is:

$$\text{ENABLE DUMP} \left\{ \begin{array}{l} \textit{partition} \\ \text{CALL} \\ \text{JUMP} \\ \text{RETURN} \end{array} \right\}$$

To disable this facility again, enter the command DISABLE DUMP.

Under automatic display for single-step emulation, the trace data for the last instruction executed, and the contents of registers RA, RB, RC, RD, RE, RH, RL, RF, PC, and SP are displayed after every single step. ENABLE DUMP turns this facility on; DISABLE DUMP turns it off.

The optional entries following ENABLE DUMP are called *selectors.* As indicated by the braces around them, each of the selectors may be used once in the command, in any order, but none may be used twice. When DUMP is enabled with one or more selectors, the information is displayed only when at least one selector was satisfied by the last instruction (that is, the last instruction was a JUMP, CALL, or RETURN, or the previous PC was between the bounds of *partition*).

Selectors from the previous ENABLE DUMP command are cleared by the next ENABLE DUMP command, and thus are not cumulative from command to command. DISABLE DUMP also clears all selectors.

The command tokens ENABLE DUMP or DISABLE DUMP are required. Following ENABLE DUMP, the selectors are optional.

Partition is specified in any of three formats, as follows.

expression	(evaluates to a single address)
expression TO *expression*	(range of addresses)
expression LENGTH *expression*	(the first expression evaluates to the first address in the partition; the second expression gives the number of contiguous address in the partition, including the first address.

Refer to the Memory Contents commands for more details on *partition.*

The CALL selector represents any of the following 8085 instructions.

CALL	Call to routine
Ccondition	Conditional call
PCHL	Jump H and L indirect (move H and L to PC)
RST n	Restart

The JUMP selector represents any of the following 8085 instructions.

JMP	Jump to address
Jcondition	Conditional jump
PCHL	Jump H and L indirect (move H and L to PC)

The RETURN selector represents any of the following 8085 instructions.

RET	Return
Rcondition	Conditional return

Set Condition-Register Command

condition-register = conditional-expression

Examples:

```
CR0 = BYTE.CTR > 55
CR1 = PC = 1EDH
CR2 = WORD.AA <>FFFFH
```

condition-register	One of the four *condition registers* (CR0, CR1, CR2, or CR3) used to bring about a halt in single-step emulation.
=	The assignment operator.
conditional-expression	An *expression* involving a *relational operator* , that 'evaluates' to true or false. Single-stepping halts when the conditional expression becomes true. See DISCUSSION for details on forming conditional expressions in this command.

SR Command (Set STEP-Register)

```
(1) SR =  FOREVER

(2) SR =  [COUNT expr-10]  [TILL cond-reg  [[AND cond-reg]... & 3]]
                                             OR

(3) SR =  [COUNT expr-10]  [TILL cond-exp  [[AND cond-exp]... & 3]]
                                             OR
```

Examples:

```
SR = FOREVER
SR = COUNT 4
SR = COUNT 33 TILL BYTE .CTR > 55
SR = TILL CR0 AND CR1 OR CR2 AND CR3
SR = TILL PC=1EDH OR BYTE.CTR > 55 AND WORD.AA<FFH
```

SR	A command *keyword* indicating that the STEP-register setting is to be changed.
=	The assignment operator.
FOREVER	The initial setting of the STEP-register. When the STEP-register is set to FOREVER, single-step emulation can be halted only by external user abort (ESC key or hardware reset).
COUNT	A function keyword specifying a halt after *expr-10* instructions have been executed. If COUNT is omitted, it is implicitly set to FOREVER.
expr-10	A *numeric constant* or *numeric expression* representing the maximum number of instructions to be emulated (executed) under the current STEP command. The default radix for evaluating *expr-10* is decimal.
TILL	A function keyword introducing one or more conditions for halting single-step emulation (in addition to the COUNT clause).
cond-reg	One of the four *condition-registers* (CR0, CR1, CR2, or CR3) used to control single-step emulation.
AND	Logical AND. Two conditions connected by AND must both be true to halt single-step emulation.
OR	Logical OR. When two conditions are connected by OR, emulation halts when either condition becomes true.
. . . & 3	A notational symbol indicating that *cond-reg* or *cond-exp* can be repeated up to three times (a total of four conditions or condition registers); separate conditions or condition registers must be connected with AND or OR, where AND has precedence over OR.
cond-exp	A *conditional expression*. Emulation halts when the condition becomes true. See DISCUSSION for details on forming conditional expressions in STEP (and SR) commands.

STEP Command

```
(1) STEP   [FROM addr]   [FOREVER]

(2) STEP   [FROM addr]   [COUNT expr-10] [TILL cond-reg  [ AND  cond-reg ] ...&3 ]
                                                         [ OR            ]

(3) STEP   [FROM addr]   [COUNT expr-10] [TILL cond-exp  [ AND  cond-exp ] ...&3 ]
                                                         [ OR            ]
```

Examples:

```
STEP
STEP FROM 1FFFH
STEP FOREVER
STEP FROM 1FFFH FOREVER
STEP COUNT 4
STEP FROM .START COUNT 33 TILL BYTE .CTR > 55
STEP TILL CR0 AND CR1 OR CR2 AND CR3
STEP TILL PC = 1EDH OR BYTE.CTR > 55 AND WORD.AA<FFFFH FROM 1FF0H
```

STEP	A command keyword that starts single-step emulation, subject to the starting and stopping conditions given.
FROM	A function keyword introducing the address where single-step emulation is to begin.
addr	A *numeric constant, statement,* or *numeric expression* that evaluates to an address value, to be used as the starting address for the single-step emulation.
FOREVER	The initial setting of the STEP-register. When the STEP-register is set to FOREVER, single-step emulation can be halted only by external user abort (ESC key).
COUNT	A function keyword specifying a halt after expr-10 instructions have been executed. If COUNT is omitted, it is implicitly set to FOREVER.
expr-10	A *numeric constant* or *numeric expression* representing the maximum number of instructions to be emulated (executed) under the current STEP command. The default radix for evaluating *expr-10* is decimal.
TILL	A function keyword introducing one or more conditions for halting single-step emulation (in addition to the COUNT clause).
cond-reg	One of the four *condition-registers* (CR0, CR1, CR2, or CR3) used to control single-step emulation.
AND	Logical AND. Two conditions connected by AND must both be true to halt single-step emulation.
OR	Logical OR. When two conditions are connected by OR, emulation halts when either condition becomes true.

...&3	A notational symbol indicating that *cond-reg* or *cond-exp* can be repeated up to three times (a total of four conditions or condition registers); separate conditions or condition registers must be connected with AND or OR, where AND has precedence over OR.
cond-exp	A *conditional expression.* Emulation halts when the condition becomes true. See DISCUSSION for details on forming conditional expressions in STEP (and SR) commands.

Display STEP Register Commands

```
SR

condition-register
```

Examples:

```
SR
CR0
```

SR	A command keyword that calls for a display of the content of the STEP-register.
condition-register	One of the four *condition registers* CR0, CR1, CR2, or CR3, causing the setting of that register to be displayed.

ENABLE/DISABLE DUMP Command

```
ENABLE DUMP   ⎧ partition ⎫
              ⎨ CALL      ⎬
              ⎨ JUMP      ⎬
              ⎩ RETURN    ⎭

DISABLE DUMP
```

Examples:

```
ENABLE DUMP
DISABLE DUMP
ENABLE DUMP 100H
ENABLE DUMP 100H TO 200H CALL JUMP RETURN
ENABLE DUMP RETURN CALL
```

ENABLE	A command keyword causing its object (in this case automatic display after single-step emulation) to become activated.
DISABLE	A command keyword that cancels the operation of its object.
DUMP	A function keyword for the automatic display of trace information after each step of a single-step emulation.
partition	A memory address or a range of contiguous addresses. If partition is included, the DUMP is enabled when an address in the range specified is accessed.
CALL	A function keyword representing a class of instructions. If CALL is included, DUMP is enabled when one of the instructions represented by CALL is executed.
JUMP	A function keyword representing a class of instructions. If JUMP is included, DUMP is enabled when one of the instructions represented by JUMP is executed.
RETURN	A function keyword representing a class of instructions. If RETURN is included, DUMP is enabled when one of the instructions represented by RETURN is executed.

External Call Commands

The External Call commands emulate or execute procedures located in mapped or unmapped memory. The three commands differ in the assumed location of the called procedure, and the conditions they establish for the return from the procedure.

This section gives details on the following commands.

Command	Page
CALL	5-103
ICALL	5-103
EXECUTE	5-103

Discussion

The CALL command emulates the called procedure, using the ICE-85 microprocessor in the umbilical cable to execute the instructions. This command saves the current value of PC, and continues emulation from that address upon return from the procedure. The second and third parameters of the command are word-type parameters. If these parameters are passed, registers BC and DE are used, destroying their earlier contents.

Breakpoints activated prior to the CALL command can halt emulation of the called procedure. Trace data is collected, subject to its normal controls.

CALL thus enables you to insert emulation of a selected procedure at any point; all pre-specified controls and displays apply to that emulation and to the continuation of normal emulation upon return.

A common use is to simulate an external interrupt *n*, using a command CALL 8* *n*. One use for this command is to answer questions such as: "What if a certain type of interrupt were to occur here? Would correct parameters be passed and acted on correctly? Does control return correctly, with appropriate flags set?".

The ICALL command executes the called procedure, using the 8080 microprocessor in the Intellec system. The starting address given by address lies in Intellec shared memory; the memory map is not used to determine the location. The procedure must be loaded separately from the rest of the user program code. One or two additional word-type parameters may be passed; if they are present, Intellec 8080 registers BC and DE are used to hold them.

The EXECUTE command loads and emulates the procedure in the file entered. Two address parameters may be passed, as discussed above. The value returned by the routine in register HL is displayed. The procedure called by EXECUTE must have start address 6F00H, and must lie between 6F00H and 6FFFH.

Other differences between EXECUTE and CALL are as follows. Under EXECUTE:

1. Breakpoints and trace are disabled before emulation begins, and are restored to their earlier condition after this emulation returns.
2. All registers are automatically saved before the emulation is started, and are automatically restored when the procedure returns.
3. Emulation is halted upon return from the procedure, rather than continuing as under CALL.

The EXECUTE command can be used to access peripheral chips in the user system, by writing a procedure to read or write the appropriate memory or I/O locations.

The meta-term *address* means one of the following types of entries.

numeric-constant	A single number in any *input-radix.* ICE-85 treats all numbers modulo 65532 (64K); thus any number represents an address.
numeric-expression	The forms for numeric expressions are presented in Chapter 4. The result obtained when the expression is evaluated becomes an address modulo 64K.
symbolic-reference	The ICE-85 symbol table lists all symbols loaded with the test program or defined by the user after program load. Corresponding to each symbol is a number that can be used as an address.
statement-number-reference	The ICE-85 statement number table gives the address of the first instruction generated by the statement with the designated number.
processor-register	The name of one of the 8085 processor registers (refer to Hardware Register commands, page 5-31).
status-register	The name of one of the ICE-85 status registers (see Hardware Register commands). The content of the named register becomes the address.
(*mem-type address*)	A memory content reference with a form such as BYTE (WORD 1000) represents an indirect reference. The content of the address or address-pair inside the parentheses is treated as the address for the *mem-type* outside the parentheses.

External Call Commands

```
(1) CALL
    ICALL   address [(address[, address])]

(2) EXECUTE :drive:filename [(address[, address])]
```

Examples:

```
CALL 0500H
CALL 0500H (1000H)
CALL 0500H (1000H, 2000H)
ICALL F6FF (F710H, F7FAH)
EXECUTE :F1:TEST
```

CALL	A command keyword initiating a call to a procedure in user (prototype) memory, to be emulated.
ICALL	A command keyword initiating a call to a procedure in Intellec memory, to be executed.
EXECUTE	A command keyword initiating a call to a procedure from an external file to be emulated, and halting emulation upon return.
address	A *numeric-constant* or *numeric-expression* that evaluates to an address value, the starting address for the called procedure.
:drive:	An ISIS-II diskette drive number enclosed in colons (see Utility commands).
filename	An ISIS-II diskette filename (see Utility commands). The file must contain an absolute object file which cannot be a main module.

APPENDIX A ICE-85 KEYWORDS AND THEIR ABBREVIATIONS

ACY	ACY	INPUT	INP	RA	RA
ALL	A	INSTRUCTION	INS	RB	RB
AND	AND	CTION	INS	RBC	RBC
ASCII	ASC	INTELLEC	INT	RC	RC
BASE	BAS	IO	IO	RD	RD
BR	BR	IWORD	IWO	RDE	RDE
BR0	BR0	JUMP	JUM	RE	RE
BR1	BR1	LENGTH	LEN,L	READ	REA,R
BUFFERSIZE	BUF	LIST	LIS	REGISTER	REG,R
BYTE	BYT	LOAD	LOA	REMOVE	REM
CALL	CALL	LOCATION	LOC	RESET	RES
CAUSE	CAU	M5	M5	RETURN	RET
COUNT	COU,C	M6	M6	RF	RF
CR0	CR0	M7	M7	RH	RH
CR1	CR1	MAP	MAP	RHL	RHL
CR2	CR2	MASK	MAS	RL	RL
CR3	CR3	MEMORY	MEM	SAVE	SAV
CY	CY	MOD	MOD	SHARED	SHA
CYCLE	CYC	MODULE	MOD	SID	SID
DEFINE	DEF	MOVE	MOV,M	SN	SN
DISABLE	DIS	NEWEST	NEW,N	SOD	SOD
DUMP	DUM	NOCODE	NOC	SP	SP
ENABLE	ENA	NOLINE	NOL	SR	SR
EVALUATE	EVA	NOSYMBOLS	NOS	STEP	STE,S
EXECUTE	EXE	NOVERIFY	NOV	STOPTRACE	STO
EXECUTED	EXE,E	OLDEST	OLD,O	SUFFIX	SUFFIX
EXIT	EXI	ON	ON	SY0	SY0
FOREVER	FOR	OPCODE	OPC	SY1	SY1
FRAME	FRA	OR	OR	SYMBOL	SYM
FROM	FRO,F	OUT	OUT,O	TILL	TIL
GO	G	OUTPUT	OUT,O	TIMEOUT	TIM
GR	GR	PC	PC	TO	TO
GROUP	GRO	PORT	POR	TRACE	TRA
GUARDED	GUA	PPC	PPC	UNSHARED	UNS
H	H	PRINT	PRI,P	UPPER	UPP
HALT	HAL,H	PSW	PSW	USER	USE
HARDWARE	HAR	PY	PY	VALUE	VAL
I7	I7	Q	Q,O	WORD	WOR
IBYTE	IBYTE	QR	QR	WRITTEN	WRI,W
ICALL	ICALL	QR0	QR0	Y	Y
IE	IE	QR1	QR1	Z	Z

APPENDIX B ICE-85 ERROR CONDITIONS AND RECOVERY

Error conditions encountered by the ICE-85 module cause numbered error messages to print on your console.

Since commands are read on a line-by-line basis, the ICE-85 module will not flag any error until after an entire line has been entered. When the first command error is found, (except for C-series errors), command processing stops and the offending line has no further effect on any internal variables: all program values remain unchanged.

The following list of error messages is not complete, but does cover most foreseeable possibilities for field errors as opposed to factory maintenance diagnostic messages. Some possible error numbers have no associated error or message. Also not included here are many messages which can come from ISIS; these are explained in the ISIS Operator's Manual. Any of several different error messages might result from faulty memory boards or connections.

In some rare cases, the error number may be printed without its associated message, as in:

```
ERR 80: ?
```

This means an Error 80 was detected but some other problem prevented the ICE-85 from printing the message. Usually it means file INSERT.ERR was not on the diskette that you invoked ICE-85 from.

Error numbers up through 7F are specifically hardware problems. 'C' series errors (C0 through CF) cause warning messages to be issued but command processing is not halted.

ERR 10:RSLTS BLK INACCESSIBLE

A bus timeout was detected on an attempt to write the results block or an incorrect installation of memory boards.

ERR 11:XMIT BLK INACCESSIBLE

A bus timeout was detected on an attempt to read the transmit block or an incorrect installation of memory boards.

ERR 21:COMMAND NOT ALLOWED NOW

The command code in the parameter block cannot be processed at this time.

Possibly an attempt was made to read register or data status during an emulation. Otherwise this error indicates a hardware failure.

ERR 30:PGM MEMORY FAILURE

Data read back from program memory did not agree with data written.

ERR 34:CABLE FAILURE

Cable diagnostic program detected a failure in the cable.

Check that the cable is oriented properly and connected properly at each end.

ERR 35:CONTROL CIRCUIT FAILURE

Control diagnostic program detected a failure in the control circuitry, or found itself in an ambiguous situation.

An ICE-85 hardware error has been detected. The cause of the error is not determinable. Try to reset the entire system.

ERR 41:NO USER VCC

The user VCC is not present.

Power is off in the user system.

ERR 42:GUARDED ACCESS

Access was made to a guarded memory or I/O location.

In Interrogation mode, no read or write is done to a GUARDED location. In Emulation, one or two (memory) cycles may have been executed before the guarded state was invoked.

ERR 43:PROCESSOR NOT RUNNING

Either there is no clock or the READY line is still low.

ERR 80:SYNTAX ERROR

The word flagged is not one that is allowed in the current context.

ERR 81:INVALID TOKEN

The word flagged does not follow the rules for a well-formed token.

The line is ignored and you must re-enter your intended command. Check the correctness of the syntax and variable-names used.

ERR 82:NO SUCH LINE NUMBER

The specified line number does not exist in the current module.

Perhaps it lies in another module or a different version of this one.

ERR 83:INAPPROPRIATE NUMBER

The value printed on the preceding line is not appropriate in the current context.

Some contexts allow only certain numbers, as in the MAP command, e.g. MAP IO 13 TO 17 is not permissible because IO blocks must start on multiples of 8 in hexadecimal, e.g. 8, 10, 18, 20, ...

ERR 84:PARTITION BOUNDS ERROR

The partition values entered in a command are not correct. Either the left part of the partition is greater than the right part or the values of the partition extremes are out of range in the current context.

ERR 85:ITEM ALREADY EXISTS

The symbol or group entered in a define command is currently defined in the symbol table.

You may need to validate the current usage of this symbol in your program, or perhaps merely use a different spelling to maintain the distinction.

ERR 86:ITEM DOES NOT EXIST

The item printed on the preceding line does not reside in the symbol table.

It may have been removed in an earlier test session, prior to saving the code, or it may be in a change you haven't inserted yet. Possibly you've loaded the wrong version of your code.

ERR 87:DUPLICATE CHANNEL

The channel specified appears more than once in a channel list.

ERR 8F:NON-NULL STRING NEEDED

a null string was used where a non-null string is required.

Perhaps in initializing a memory partition, as in IBYTE 1 TO 5 = ' ', where there is no character in the string between the single quotes.

ERR 90:MEMORY OVERFLOW

Memory requirements of all dynamic tables exceed the amount of memory available.

In a 32K Intellec the symbol table can contain a maximum of about 900 two-character symbols; fewer if they are more than two characters long. If some symbols are no longer needed for debugging, use the REMOVE command to make space. The code itself is unchanged by this.

ERR 91:STACK OVERFLOW

This is probably due to an excessively complicated command, e.g. one with 20 parenthesis pairs.

ERR 92:COMMAND TOO LONG

Probably due to inordinate number of operators. Break it up into several smaller commands.

ERR 93:MODULE DOES NOT EXIST

Module specified does not exist in symbol table.

ERR 94:NON-CHANGEABLE ITEM

An attempt was made to change an item that may not be changed.

ERR 95:INVALID OBJECT FILE

File specified in a load command is not a valid object file.

Perhaps it is a text file, or a wrong extension such as PRGFIL.HEX or PRGFIL.OBJ for object file.

ERR 97:EXCESSIVE DATA

The amount of data attempted to be inserted into a partition exceeded the size of the partition.

The *n* cells in the partition were filled with the first *n* data items supplied, after which the data given was ignored.

ERR 98:MORE THAN 16 CHANNELS

More than 16 channels specified in a channel list.

ERR 99:EXCESSIVE ITERATED DATA

The amount of data to be repeated throughout a range of memory exceeds the size of the buffer allocated to hold such data.

The limit is 128 decimal, e.g.,

```
BYTE 1 TO 256T = IBYTE 4 TO 131T
```

will work, using the right side twice but

```
BYTE 1 TO 256T = IBYTE 1 TO 131T
```

will fail, being too large.

ERR 9A:TOO MANY GROUPS

Number of groups defined by user may not exceed 36.

ERR 9B:TOO MANY CHANNELS

Number of channels defined by user may not exceed 103T.

ERR 9C:UNSUITABLE EXECUTE FILE

The file referenced in an execute command either contains code that is out-of-bounds for the execute command or it is a main module.

ERR 9D:LINE TOO LONG

Command line was longer than 122 characters.

WARN C0:UNSATISFIED EXTERNALS

The program just loaded contains externals which were not satisfied at link time. The program was correctly loaded except for references to the unresolved externals.

WARN C1:MAPPING OVER SYSTEM

The user has modified the map so that part of his address spaces includes either the ISIS system software or the ICE-85 module software.

This can later cause erroneous operation and undesirable results.

WARN C2:INSERT MISSING

An attempt was made to initialize the device whose generic device code number is printed on the previous line but no device responded. A generic device code is the first of four consecutive device codes reserved for a specific type of device, in this case the ICE-85 module hardware.

This means the ICE-85 module hardware is not installed correctly or the hardware module that is installed is not an ICE-85.

WARN C3:MULTIPLE INSERT

Two diagnostic devices have the same device code.

ERR E7:ILLEGAL FILENAME

The filename specified does not conform to a well-formed ISIS filename.

See ISIS manual for valid formulation and device labels.

ERR E8:ILLEGAL DEVICE

Illegal or unrecognized device in filename.

An invalid device lable was used, e.g. :DO: instead of :CO:, or something unrelated such as :PQ: see ISIS manual for valid list.

ERR E9:FILE OPEN FOR INPUT

Attempt to write to a file open for input.

E.g. SAVE :CI:, a file pre-defined as console input.

ERR EA:FILE ALREADY OPEN

Attempted to open a file that was already open

ERR FO:NO SUCH FILE

The file specified does not exist.

Possibly a wrong or missing device label, as in typing :F2:FILE when you meant :F3:FILE, or a file missing due to forgetting to copy it onto a new disk.

ERR F1:WRITE-PROTECTED FILE

Attempt to open a write-protected file for the purposes of writing data into it.

One of the system files, ISIS.DIR, for example, you must use a different name.

ERR F3:CHECKSUM ERROR

A checksum error in a hex object file was encountered during loading. Either a wrong or damaged file needing re-creation.

ERR F9:ILLEGAL ACCESS

Attempt to open a read-only file for the purpose of storing data (e.g. specifying :CI: as the list device) or a write-only file as a source of data (e.g. :LP: in a load command).

ERR FA:NO FILE NAME

No filename specified for a diskette file (e.g. no filename following :F1:).

ERR FD:"DONE" TIMED OUT

The device whose device code is printed on the preceeding line was invoked but failed to return done within five seconds.

Possibly a command requiring such a long time, some missing memory, or perhaps a hardware problem.

ERR FE:"ACKNOWLEDGE" TIMED OUT

The device whose device code number is printed on the preceding line was invoked but failed to acknowledge within 5 milliseconds.

Incomplete installation, faulty connections, or failed hardware.

ERR FF:NULL FILE EXTENSION

A file was specified so as to contain an extension, but no extension was specified.

E.g. LOAD :F1:MYPROG.

with no extension typed after the period.

APPENDIX C
8085 CPU INSTRUCTIONS

OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC	OP CODE	MNEMONIC
00	NOP	2B	DCX H	56	MOV D,M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B,D16	2C	INR L	57	MOV D,A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E,B	83	ADD E	AE	XRA M	D9	–
03	INX B	2E	MVI L,D8	59	MOV E,C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E,D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	SIM	5B	MOV E,E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B,D8	31	LXI SPD16	5C	MOV E,H	87	ADD A	B2	ORA D	DD	–
07	RLC	32	STA Adr	5D	MOV E,L	88	ADC B	B3	ORA E	DE	SBI D8
08	–	33	INX SP	5E	MOV E,M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E,A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAXB	35	DCR M	60	MOV H,B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M,D8	61	MOV H,C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H,D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	——	63	MOV H,E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C,D8	39	DAD SP	64	MOV H,H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H,L	8G	SUB B	BB	CMP E	E6	ANI D8
10	——	3B	DCX SP	66	MOV H,M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D,D16	3C	INR A	67	MOV H,A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L,B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A,D8	69	MOV L,C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L,D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B,B	6B	MOV L,E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D,D8	41	MOV B,C	6C	MOV L,H	97	SUB A	C2	JNZ Adr	ED	——
17	RAL	42	MOV B,D	6D	MOV L,L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	——	43	MOV B,E	6E	MOV L,M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B,H	6F	MOV L,A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAXD	45	MOV B,L	70	MOV M,B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B,M	71	MOV M,C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B,A	72	MOV M,D	9D	SBB L	C8	RZ	F3	DI
1D	DRC E	48	MOV C,B	73	MOV M,E	9E	SBB M	C9	RET Adr	F4	CP Adr
1E	MVI E,D8	49	MOV C,C	74	MOV M,H	9F	SBB A	CA	JZ	F5	PUSH PSW
1F	RAR	4A	MOV C,D	75	MOV M,L	A0	ANA B	CB	——	F6	ORI D8
20	RIM	4B	MOV C,E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H,D16	4C	MOV C,H	77	MOV M,A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C,L	78	MOV A,B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C,M	79	MOV A,C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C,A	7A	MOV A,D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D,B	7B	MOV A,E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H,D8	51	MOV D,C	7C	MOV A,H	A7	ANA A	D2	JNC Adr	FD	——
27	DAA	52	MOV D,D	7D	MOV A,L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	——	53	MOV D,E	7E	MOV A,M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D,H	7F	MOV A,A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D,L	80	ADD B	AB	XRA E	D6	SUI D8		

D8 = constant, or logical/arithmetic expression that evaluates to an 8 bit data quantity.

D16 = constant, or logical/arithmetic expression that evaluates to a 16 bit data quantity

Adr = 16-bit address

INDEX

Accessory kit, 2-1
ACY, 4-4, 5-32
ADDR, 4-5, 5-56
Address, 5-40
ADDRH, 4-5, 5-56
ADRL, 4-5, 5-56
Algebraic operators, 4-3
ALL, 4-6, 5-86
Alphabetic characters, 4-3
Alphabetic summary table of ICE-85
  commands, 5-3 thru 5-5
AND, 4-7
Automatic display, 5-100

BASE, 4-6, 5-15
Binary operators, 4-11
Block-name, 5-20
Block-partition, 5-20
break-reg, 4-5
Breakpoint registers, 4-5
BR, 4-5
BR0, 4-5
BR1, 4-5
BUFFERSIZE, 4-4, 5-34

CALL, 4-6, 4-7
CALL command, 5-101, 5-103
CALL selector, 5-95
Carriage return, 5-6
CARS1 program listing, 3-9
CARS2 program listing, 3-24
CAUSE, 4-4, 5-34
Change Group command, 5-3, 5-60
Change Symbol command, 5-3, 5-54
Channel group, 4-5
Channel group commands, 5-2, 5-55
Channel-list, 5-57
Channel Status Register (CSR), 5-55
Character set, 4-3
Command keywords, 4-6
Command-name, 4-6
Cond-reg, 4-5
Condition-expression, 5-96
Condition registers, 4-5, 5-96
Connector keywords, 4-7
Console input radixes, 5-12
Console output radix, 5-13
Constants, 4-10
Control board jumper pin
  configuration, 2-2
COUNT, 4-7, 5-97
COUNT clause, 5-93
CR0, 4-5, 5-92, 5-96
CR1, 4-5, 5-92, 5-96
CR2, 4-5, 5-92, 5-96
CR3, 4-5, 5-92, 5-96
CTRL P, 5-6
CTRL R, 5-6
CRTL X, 5-6
CY, 4-4, 5-32
CYCLE, 4-6, 5-84
CYCLE mode, 5-76

DATA, 4-5, 5-56
DEFINE GROUP command, 5-3, 5-59
DEFINE Symbol command, 5-3, 5-50, 5-53
DELAY program listing, 3-10
Digits, 4-10
DISABLE, 4-6, 5-4
DISABLE DUMP command, 5-100
DISABLE SY0 OUT command, 5-74
DISABLE TIMEOUT command, 5-38
DISABLE Trace Factors command, 5-88
Display GROUP command, 5-3, 5-60
Display Memory and Port Contents
  command, 3-4, 5-3, 5-46
Display Processor and Status Register
  commands, 5-3, 5-36
Display Real-time Emulation Registers
  commands, 5-3, 5-73, 5-99
Display Map status command, 5-3, 5-30
Display STEP Register commands, 5-99
Display Symbol Table and Statement-
  Number Table commands, 5-3, 5-53
Display Trace Controls
  command, 5-3, 5-88
DMUX, 4-5, 5-56
Dual auxiliary connector, 2-1
Dollar sign ($), 5-6
Double asterisk prompt, 5-6
DUMP, 4-6, 5-100

EMUL, 5-35
Emulation control commands, 3-4, 5-62
Emulation registers, 4-5, 5-62
ENABLE, 4-6
ENABLE DUMP command, 5-100
ENABLE SY0 OUT command, 5-74
ENABLE TIMEOUT command, 5-38
ENABLE Trace Factors command, 5-88
Entering commands at the console, 5-6
ESC, 5-6
EVALUATE, 4-6
EVALUATE command, 5-4, 5-13, 5-16
EXECUTE, 4-6
EXECUTE command, 5-101, 5-103
EXECUTED, 4-7
EXIT, 4-6
EXIT command, 5-4, 5-9
Explicit input radix, 5-12
Explicit output radix, 5-13
Expr-10, 5-97
Expressions, 4-12
External call commands, 5-4, 5-101
External signal cables, 2-5
External trace module, 2-1
External synchronization lines, 5-35

Filename, 5-7
Flag-name, 4-4
FOREVER, 4-7, 5-97
FRAME, 4-6, 5-84
FRAME mode, 5-76
FROM, 4-1, 4-2, 4-7
Functional keywords, 4-6

Generalized Development Cycle With
  ICE-85, 1-6
Generalized emulation session, 1-8
GND, 5-35
GO, 4-6
GO command, 4-1, 5-4, 5-71
GO command example, 4-1
Go-reg, 4-5, 5-72
GO register, 4-5, 5-72
GR, 4-5, 5-4, 5-72
GROUP, 4-6, 5-59
Group-names, 5-57
GUARDED, 4-6

HALT, 4-7
Halt condition in the GO register, 5-69
Halt conditions in the STEP register
  (SR), 5-92
HARDWARE, 4-6
Hardware installation procedures, 2-2
Hardware register commands, 5-1, 5-31

I-bit-name, 4-4
ICALL, 4-6
ICALL command, 5-101, 5-103
ICE85 command, 5-4, 5-9
ICE-85 adapter socket, 2-1
ICE-85 cable terminal assembly, 2-1
ICE-85 command language, 4-2
ICE-85 commands, 5-3
ICE-85 components, 2-1
ICE-85 control board, 2-1
ICE-85 functional block diagram, 1-4
ICE-85 installation procedures, 2-1
ICE-85 memory map, 5-18
ICE-85 metalanguage, 4-1
ICE-85 module, 1-4
ICE-85 module and interface cable, 2-1
ICE-85 mnemonics, 4-2
ICE-85 program, 1-5
ICE-85 software, 5-17
ICE-85 symbol tables, 5-17
ICE-85 trace board, 2-1
ICE-85 workspace, 5-17
Ideal grounding arrangement, 2-6
IE, 4-4
Implicit input radix, 5-12
IN, 4-6
In-circuit emulator, 1-1
INPUT, 4-7, 5-80
Input lines, 5-6
Input/output port segments for
  mapping, 5-25
Installing external signal cables, 2-5
Installation procedure for Intellec
  Series I, 2-2
Installation procedure for Intellec
  Series II, 2-3
INSTRUCTION, 4-6, 5-84
INSTRUCTION mode, 5-76
INTELLEC, 4-6
Intellec memory, 5-17, 5-18
Intellec Monitor, 5-17
Intellec Series I, 2-2
Intellec Series II, 2-3
Intellec text editor, 1-5
Interrupt bits, 4-4
Introduction to the ICE-85, 1-1
ISIS-II software, 5-17
I7, 4-4, 5-32

JUMP, 4-7
JUMP selector, 5-95

Keywords, 4-3

Line editing characters, 5-6
Line feed, 5-6
Line number references, 3-3
LIST, 4-6
LIST command, 5-4, 5-11
LOAD, 4-6
Load command, 3-5, 5-4, 5-10
LOCATION, 4-7, 5-65

MAP, 4-6
MAP I/O Ports command, 5-4, 5-29
MAP Memory command, 5-4, 5-19, 5-28
MAP Mode command, 5-4, 5-19, 5-27
Mapping Input/Output, 5-24
Masked constants, 4-10
Match-registers, 5-55
Match0, 5-35
Match1, 5-35
Memory and I/O port mapping
  commands, 3-2, 5-1, 5-16
Memory blocks for mapping, 5-19
Memory and Port Contents
  commands, 5-1, 5-39
Mem-type partition, 5-41
Metanotation used in the manual, 4-13
Mnemonic match condition, 5-65
Mode keywords, 4-6
Module names, 4-8
MOVE, 4-6
MOVE command, 5-4, 5-76, 5-85
MTH, 4-5, 5-56
Mult-op, 4-11
M5, 4-4, 5-32
M6, 4-4, 5-32
M7, 4-4, 5-32

NEWEST, 4-6
NEWEST command, 5-4, 5-85
NOCODE, 4-6, 5-8
NOLINES, 4-6, 5-8
NOSYMBOLS, 4-6, 5-8
NOVERIFY, 4-6
Number base, 4-10
Number bases and radix
  commands, 5-1, 5-12

Numeric characters, 4-3
Numeric constant, 4-10
Numeric expressions, 4-11

Object keywords, 4-6
OLDEST, 4-6
OLDEST command, 5-4, 5-85
ON, 4-7
OPCODE, 4-4, 5-34
Operands, 4-12
Operational control keywords, 4-7
Operators, 4-11
OR, 4-7
OUT, 4-6

Partition, 5-41
Pathname, 5-7
PC, 4-4, 5-32
Plus-op, 4-11
Port content references, 5-45
Precedence, 4-11
PRINT, 4-6
PRINT command, 5-4, 5-76, 5-86
Program constant, 5-49
Program variable, 5-49
Prototype memory, 5-18, 5-19
PSW, 4-4, 5-34
Punctuation, 4-11
PY, 4-4, 5-32

QR, 4-5
QR0, QR1, 4-5
Qual-reg, 4-5
Qualification registers, 4-5, 5-82, 5-87

RA, 4-4, 5-32
Radix, 4-10
Radixes used in trace displays, 5-14
Radixes used in displaying Breakpoint and
  Qualifier settings, 5-14
RB, 4-4, 5-32
RBC, 4-4, 5-32
RC, 4-4, 5-32
RD, 4-4, 5-32
RDE, 4-4, 5-32
RE, 4-4, 5-32
READ, 4-7
Real-time emulation control
  commands, 5-2, 5-62
Reference keywords, 4-3
Register-name, 4-4
Register-pair-name, 4-4
Registers, 4-4
Relational operator, 4-11
Rel-op, 4-11
REMOVE, 4-6
REMOVE GROUP command, 5-5, 5-61
REMOVE Symbol command, 5-4, 5-54
Required and optional hardware, 2-2
RESET, 4-6, 5-5
Reset Breakpoint Register command, 5-74
Reset Hardware command, 5-37
Reset Map command, 5-30
Reset qual-reg command, 5-87
RETURN, 4-7
RETURN selector, 5-95
RF, 4-4, 5-32
RH, 4-4, 5-32
RHL, 4-4, 5-32
RL, 4-4, 5-32
RUBOUT, 5-6
RW, 4-5, 5-56

Sample ICE-85 Emulation Session, 3-1
SAVE, 4-6
SAVE command 5-5, 5-11
S-group-name, 4-5
SD, 4-5, 5-56
Segment-name, 5-24
Segment-partition, 5-24
Set Breakpoint Register
  command, 5-5, 5-73
Set Condition Register command, 5-5, 5-96
Set GO Register (GR) command, 5-72
Set Input/Output Port Contents
  command, 5-5, 5-48
Set Memory Contents command, 5-5, 5-47
Set or Display Console Input Radix
  commands, 5-15
Set or Display Console Output Radix
  command, 5-15
Set Processor Register command, 5-5, 5-37
Set Qualifier Register command, 5-5, 5-87
Set Trace Display Mode
  command, 5-5, 5-84
SHARED, 4-6
SHARED memory, 5-17
SID, 4-4, 5-32
Single-step emulation control
  commands, 5-2, 5-89
SN, 4-4, 5-32
SOD, 4-4, 5-32
SP, 4-4, 5-32
Special characters, 4-3
Special tokens, 4-11
SR, 4-5
SR COMMAND, 5-5, 5-97
State keywords, 4-6
Statement numbers, 4-9
Status flags, 4-4
Status group bit settings, 5-80
STEP, 4-6, 5-90
STEP Command, 5-5, 5-90, 5-98
Step-reg, 4-5
Step register, 4-5, 5-93
STOPTRACE, 4-6, 5-88
STS, 4-5, 5-56
SUFFIX, 4-6, 5-15
SYMBOL, 4-6
Symbol table and statement number
  commands, 5-1, 5-49
Symbolic references, 3-3, 4-7, 5-50
Symbolic references and statement number
  references, 5-51
Symbols, 4-7
Sync cables, 2-1
Sync-line, 4-5
Synchronization lines, 4-5, 5-35
System defined channel groups, 5-56
System grounding, 2-5

System grounding for Intellec Series I, 2-6
System groups, 5-56
SY0, 4-5, 5-35
SY1, 4-5, 5-35

Terminators, 5-6
TILL, 4-7, 5-68
Tokens, 4-3
TRACE, 4-6, 5-76, 5-84, 5-88
Trace control commands, 5-2, 5-75
Trace display commands, 3-5
Trace display headers, 5-78
Traffic Light Controller program flow
chart, 3-8

Unary operators, 4-11
UNSHARED, 4-6
UNSHARED memory, 5-18, 5-19
UPPER, 4-4, 5-34
USER, 4-6
User names, 4-7
User group names, 4-9, 5-56
Utility commands, 5-1, 5-7
U0, 4-5, 5-56
U1, 4-5, 5-56
U2, 4-5, 5-56

VALUE, 4-7, 5-65

WRITTEN, 4-7

Z, 4-4, 5-32

8-bit registers, 4-4
8085 CPU functional block diagram, 1-2
8085 interrupt mask, 5-32
8085 processor channels and system-
defined group names, 5-56
8085 1-bit registers, 5-32
8085 8-bit registers, 5-32
16-bit registers, 4-4, 5-32

NOTES

NOTES

SOFTWARE PROBLEM REPORT

SUBMITTED BY:	**FOR INTERNAL USE ONLY**
Name __________	No. Fix Date
Company __________	Date Vers/System
Address __________	Notes
__________	
Phone __________ Date __________	

CHECK ONE ITEM IN EACH CATEGORY

Product	**Product Type**		**Machine Line**	**System**
☐ Software	☐ Monitor	☐ Simulator	☐ 4004/4040	☐ Intellec
☐ Manual	☐ Assembler	☐ Editor	☐ 8008	☐ Timeshare Co.
	☐ Compiler	☐ Utility	☐ 8080	__________
		☐ ______	☐ 3000	☐ In-House Computer
			☐ ______	__________

Exact Product/Manual Name __________

Version Number (If not known, give date of receipt) __________

PROBLEM:

REPLY:

PROBLEM DOCUMENTATION ATTACHED IS: ☐ Output Listing ☐ Paper Tape Program Sou

WE'D LIKE YOUR COMMENTS . . .

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

First Class
Permit No. 1040
Santa Clara, CA

BUSINESS REPLY MAIL

No Postage Stamp Necessary if Mailed in U.S.A.

Postage will be paid by:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Attention: MCS Systems Marketing

REQUEST FOR READER'S COMMENTS

The Microcomputer Division Technical Publications Department attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. ____________

NAME ____________________ DATE ____________
TITLE ____________________
COMPANY NAME/DEPARTMENT ____________________
ADDRESS ____________________
CITY ____________ STATE ____________ ZIP CODE ____________

Please check here if you require a written reply. ☐

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.

First Class
Permit No. 1040
Santa Clara, CA

BUSINESS REPLY MAIL

No Postage Stamp Necessary if Mailed in U.S.A.

Postage will be paid by:

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Attention: MCD Technical Publications

intel®

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

Printed in U.S.A.