RBF-89 REAL-TIME BREAKPOINT FACILITY OPERATING INSTRUCTIONS FOR ICE-86™ IN-CIRCUIT EMULATOR USERS

Manual Order Number: 9801018-01 Rev. A

intel®

RBF-89 REAL-TIME BREAKPOINT FACILITY OPERATING INSTRUCTIONS FOR ICE-86™ IN-CIRCUIT EMULATOR USERS

Manual Order Number: 9801018-01 Rev. A

A171/280/15K DD

PREFACE

This manual describes how to operate the RBF-89 Real-Time Breakpoint Facility. RBF-89 aids in testing and troubleshooting the software for application systems designed around the Intel 8086 Central Processing Unit (CPU) used in combination with the Intel 8089 Input/Output Processor (IOP).

Chapter 1, *General Information,* defines the primary capabilities of RBF-89 and gives a short functional description of this product. It also discusses RBF-89's relation to the 8086 CPU, the 8089 IOP, and the ICE-86 In-Circuit Emulator. In addition, this chapter defines the assumptions and prerequisites for using RBF-89.

Chapter 2, *RBF-89 Commands,* defines all RBF-89 commands and the syntax and rules for using them. The commands appear in functional groups so that you may easily grasp their interrelationship. Examples of command entries also appear.

Chapter 3, *Applications,* shows how to load and use RBF-89 in an operational environment. It illustrates, as examples, how two typical testing and debugging tasks are actually performed.

The appendixes present error messages and recovery procedures, a list of RBF-89 keywords and their definitions, a summary of command reference specifications, and the 8089 IOP instruction set (which is also summarized in *RBF-89 Real-Time Breakpoint Facility Pocket Reference,* Order No. 9801019).

Before using this manual, you should read the following manuals for definitions of terms and concepts not unique to RBF-89, and other prerequisite information:

- *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users,* Order No. 9800714.
- *The 8086 Family User's Manual,* Order No. 9800722. (Includes 8086, 8088, and 8089 microprocessor documentation.)

As an option, depending on your particular application, you may also wish to consult some of the following manuals for helpful background information:

- *Intellec MDS Operator's Manual,* Order No. 9800129
- *Intellec MDS Hardware Reference Manual,* Order No. 9800132
- *ISIS-II User's Guide,* Order No. 9800306
- *PL/M-86 Programming Manual,* Order No. 9800466
- *PL/M-86 Operator's Manual,* Order No. 9800478
- *Intellec Series II Hardware Reference Manual,* Order No. 9800556
- *MCS-86 Software Development Utilities Operating for ISIS-II Users,* Order No. 9800639
- *MCS-86 Assembly Language Reference Manual,* Order No. 9800640
- *MCS-86 Assembler Operating Instructions for ISIS-II Users,* Order No. 9800641
- *8089 Assembly Language and 8089 Assembler Operating Instructions for ISIS-II Users,* Order No. 9800938
- *The 8086 Family User's Manual,* Order No. 9800722

intel

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1 GENERAL INFORMATION

The RBF-89 Real-Time Breakpoint Facility is a software extension to the ICE-86 In-Circuit Emulator that aids in testing and troubleshooting applications designed around the Intel 8086 Central Processing Unit (CPU) used in combination with the Intel 8089 Input/Output Processor (IOP). During your testing sessions, RBF-89 extends the debugging power of the ICE-86 In-Circuit Emulator into the 8089 portion of your system.

Because RBF-89 has no hardware of its own, it operates by preparing special control blocks in system memory and then issuing channel-attention commands to the 8089 IOP in your application system to perform the requested functions. This technique requires that you build your prototype application system and, using the ICE-86 emulator and/or various hardware debugging tools, debug it to a certain level of reliability *before* using RBF-89. For more information about this requirement, see *Assumptions and Prerequisites for Using RBF-89* later in this chapter.

Features

RBF-89 provides the following features:

Program Loading, Execution, and Back-Up

RBF-89 allows you to load your application (channel) program from diskette into 8089 IOP memory and execute it in real time. The program can reside in either local (system) RAM (accessible by both the 8086 and 8089 microprocessors), or remote RAM (accessible by the 8089 IOP only). You may request execution to begin at any location and continue until normal termination, or until a specified breakpoint is reached or until the program is aborted. (A *breakpoint* is a pre-specified location where execution is to terminate.) If you modify your program, you may request RBF-89 to save the latest version by copying it in absolute object format from memory to diskette.

Breakpoints

With RBF-89, prior to program execution, you can specify up to six breakpoints in each breakpoint register. RBF-89 implements each breakpoint by inserting a HALT instruction at the breakpoint location, and saving the overwritten instruction in temporary storage. When a breakpoint is reached during program execution, the program halts. At this point RBF displays the adress reached and you can examine various registers and memory locations, and optionally resume program execution. The breakpoint addresses are recorded in one of two Breakpoint Registers—one register for each 8089 input/output channel. Through simple RBF-89 commands, you can display or change the contents of these registers.

Data Display and Modification

Through RBF-89, you can display and change the contents of:

- Local and remote 8089 memory.
- All 8089 registers except the channel control pointer (CCP), and status flags.

Disassembly of Object Code in Memory

RBF-89 allows you to disassemble object code in remote or local memory. This permits you to display and examine the machine instructions in 8089 assembly-language mnemonic form.

Multiprocessor Operation

Because the 8089 IOP is so often used with the 8086 CPU, RBF-89 permits you to run the 8089 and emulated 8086 simultaneously as well as sequentially. To facilitate this, RBF-89 lets you specify breakpoints and begin program execution in three different operating sequences:

a. Set the breakpoints, start the 8089, continue execution until a breakpoint occurs or the program runs to completion or is aborted, and then return control to the console. You use this sequence when you do not need to execute your 8086 user program and your 8089 channel program simultaneously.

b. Set the breakpoints, start the 8089, return control to the console, and start the 8086. This sequence lets you run both microprocessors simultaneously.

c. Set the breakpoints, start the 8086 program, and allow that program to drive the 8089 program in a master/slave relationship. You do this, for instance, to verify your 8086 driver program.

Chapter 2 provides more information about the effects and limitations of these operating sequences.

Command Interface

Because RBF-89 is a software substructure embedded within the ICE-86 In-Circuit Emulator software, you invoke RBF-89 simply by entering the **RBF** command while logged-on to the emulator. RBF-89 then accepts a series of short subcommands that initialize the 8089 IOP and verify that it is working properly. Then, you operate RBF-89 through a set of commands very much like the ICE-86 emulator commands.

Access to ICE-86 Emulator Commands

In addition to its own functions, RBF-89 lets you use most ICE-86 emulator commands by entering the appropriate emulator command without exiting from RBF-89.

Applications

RBF-89 can be used to debug most application systems that incorporate an 8089 IOP. Such systems must be multiprocessor configurations that also include the 8086 CPU. In these configurations, the CPU and IOP sometimes operate in a master/slave relationship. Typical 8089 applications include:

- Scan control, file, directory, and buffer management for magnetic disks. In these applications, code is often established for procedures that either allow retries or handle detected cyclic redundant code (CRC) errors—all without central processor intervention.
- Support of high-speed transfers on hard disks. These transfers are either buffered in 8089 memory on a local bus or sent directly to 8089 system memory,

- Support of graphic CRT terminals. For both CRT's and keyboards, the 8089 IOP handles linked-list data structures, maintains cursor control, performs automatic scrolling, translates characters, and recognizes special control characters—again, all without CPU involvement.
- Data formatting, protocol interpretation, and general-purpose input/output control. The 8089 IOP causes the peripheral controller to appear to the CPU as a block-oriented DMA device. A simple software interface, common to all peripherals and user programs, is often specified. The 8089's processing power allows users to customize the software to the particular requirements of each type of peripheral device.

System Description

RBF-89 is furnished as an integral part of the ICE-86 emulator software. RBF-89's main components, illustrated in figure 1-1, are:

- A *Host Program* that resides in Intellec development system RAM, where it serves as an extension of the ICE-86 emulator's software driver. The Host Program is executed by the development system's 8080 microprocessor. This program translates your keyboard input into command directives that can be processed by the RBF-89 MONITOR program (described below), and converts information supplied by the MONITOR into easily-understood display output. (The Host Program also handles input received from an ISIS-II SUBMIT file, and output directed to a mass-storage device or line printer.) In addition, the Host Program maintains appropriate 8086 register settings and memory allocation to allow the system to alternate between RBF-89 and 8086 application (user) program processing.
- A *Control Program,* named *MONITOR,* that resides in ICE-86 emulator memory. MONITOR runs on the emulator's 8086 microprocessor and monitors such operations as: preparing program control blocks for communication with the 8089 microprocessor; issuing channel commands to the 8089 IOP to start, terminate, and continue the 8089 channel program; and directing the 8089 to start execution of the RBF-89 Utility Program (described below).
- A *Utility Program* that resides in the 8089 RAM in your prototype application system. This program, running on the 8089, reads and writes data to and from 8089 memory and registers, and sets and removes breakpoints in your channel program.

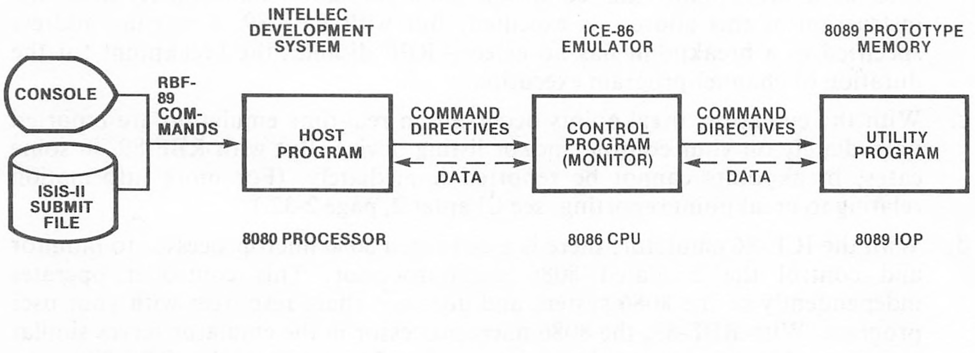


Figure 1-1. RBF-89 System Block Diagram

1018-01

Assumptions and Prerequisites for Using RBF-89

Before using RBF-89, you must develop and debug your application system to a certain level of reliability. Generally, you perform this debugging with the ICE-86 emulator and/or various hardware debugging tools. Specifically, the 8086/8089 microprocessor interface and the 8089 subsystem must be developed to the following level:

- CPU/IOP Interface
  a. The mechanism used by the 8086 CPU to issue a channel-attention request must be operational.
  b. The mechanism used by the 8086 CPU to toggle the SEL pin on the 8089 IOP must be operational.
  c. The application system RAM area, where the RBF-89 control blocks are stored, must be fully-installed. (The RAM area must include 200 bytes usable by RBF-89.)
- 8089 Subsystem
  a. The 8089 I/O bus must be operational.
  b. The memory or input/output port accessed by the 8089 IOP must be operational.
  c. The portion of your 8089 channel program where you wish to set breakpoints must NOT be stored in ROM.

ICE-86 Emulator/RBF-89 Differences

As someone accustomed to using the ICE-86 emulator, you should be aware of these functional differences between the emulator and RBF-89:

a. With the ICE-86 emulator, breakpoints are implemented in the hardware and can be triggered by a variety of events—access of a particular memory location, processing within a range of locations, reading of a specific data value at a particular location or port, and so forth. But in RBF-89, breakpoints are implemented through the software and can only be triggered by execution of one of the breakpointed instructions.

b. With the ICE-86 emulator, the address at which emulation begins can also be used as a breakpoint; this condition halts emulation immediately after the instruction at this address is executed. But with RBF-89, a starting address specified as a breakpoint has no effect—RBF disables the breakpoint for the duration of channel-program execution.

c. With the emulator, breakpoints occurring in real-time emulation are reported immediately on your console and/or listing device. But with RBF-89, in some cases, breakpoints cannot be reported immediately. (For more information relating to breakpoint reporting, see Chapter 2, page 2-32.)

d. With the ICE-86 emulator, there is a dedicated 8080 microprocessor to monitor and control the emulated 8086 microprocessor. This controller operates independently of the 8086 system and does not share resources with your user program. With RBF-89, the 8086 microprocessor in the emulator serves similar purposes—but it is not a dedicated controller. Thus, when using RBF-89 to run an 8086 user program concurrently with an 8089 channel program, you must allow for the shared use of the 8086 microprocessor; RBF-89 commands cannot always be processed while emulation is underway. Similarly, ICE-86 emulator commands cannot always be processed while RBF-89 is running.

Limitations

In preparing to use RBF-89, you should be aware of these limitations and restrictions:

- RBF-89 supports only 8086/8089 configuration—other configurations, such as 8085/8089 combinations, cannot be debugged with RBF-89. In a dual-8089 system, RBF-89 can communicate with only one 8089 during a particular RBF-89 debugging session.
- RBF-89 does not provide single-step execution or trace capabilities.
- If the 8089 IOP locks up the system bus during operation, RBF-89 takes no corrective action; the only way to free the system bus is to reset the 8089 IOP to its initial state.

The ICE-86 In-Circuit Emulator supports 8089 configurations in the remove mode only. The ICE-86 emulator does not employ RQ/GT (request/grant). RQ/GT is required in the local mode to provide system bus arbitration between the 8089 and 8086. In the remove mode, system bus arbitration is provided by the 8289 Bus Arbiter.

System Requirements

For its operation, RBF-89 requires the following supporting hardware and software:

Hardware

The minimum hardware configuration needed to support RBF-89 is:

- Intellec Microcomputer Development System:

  Model 800 (with CRT console, two single-density or double-density disk drive, 64k and RAM; or

  Model 220 or 230 (with 64k RAM; or

  Model 240 (with 64k RAM and hard-disk drive).

- ICE-86 (or ICE-88) Emulator Hardware, with ICE memory dedicated to the RBF-89 Monitor Program. (During an RBF-89 session, an attempt to map your application program into ICE memory results in an error.) RBF-89's requirements for ICE memory should not significantly restrict your activities, because your prototype application system should be developed to the point where you can map your entire channel program into application system memory.

- Two contiguous 1k-byte segments of the 8086 memory address space mapped to GUARDED status. RBF-89 will map this memory to the emulator's 2k of emulated memory. Thus, this memory need not exist physically in your prototype application system.

- Application System Hardware, developed to the level described under *Assumptions and Prerequisites for Using RBF-89,* with 200 bytes of RAM available to RBF-89 as a buffer for messages between the 8089 and the emulated 8086.

Software

The minimum software required to support RBF-89 is:

- ISIS-II Operating System (Version 3.4 or later)

For batch-processing operations, RBF-89 can be run under the ISIS-II SUBMIT facility.

RBF-89 recognizes and uses the American Standard Code for Information Interchange (ASCII) Character Set, X3.4, 1968, established by the American National Standards Institute (ANSI).

IOP channel programs loaded by RBF-89 must be formatted as Intel Standard absolute object files. Files saved by RBF-89 are always stored in this format. (For further information about this format, see *MCS-86 Absolute Object File Formats,* Order No. 9800821.)

CHAPTER 2
RBF-89 COMMANDS

To interact with RBF-89, you log-on to the ICE-86 In-Circuit Emulator, invoke and initialize RBF-89, and enter various RBF-89 commands at the console. These commands, which have a syntax very similar to that of the ICE-86 emulator commands, allow you to request the full range of RBF-89 operations.

NOTE

RBF-89 also permits you to request any ICE-86 emulator operation by entering the appropriate emulator command without exiting from RBF-89. For directions on using the emulator commands, please see *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.*

Entering Commands

Because RBF-89 and ICE-86 emulator commands are syntactically similar, most of the rules for entering both types of commands are identical. For example, the rules covering command-line structure, comment entry, line continuation and termination, and ISIS-II line-editing capabilities are the same for both systems. Because these command rules are presented and explained in *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users,* they will not be repeated herein. Where differences apply, however, they will be cited in this manual. For instance, although the command prompt character for both systems is an asterisk (*), RBF-89 prompts for special subcommands during its initialization by displaying a period followed by an asterisk (.*).

Correcting Errors

When RBF-89 detects a fatal error during command processing, it halts this processing, displays a brief error message, and prompts you for a new command entry. When RBF-89 detects a non-fatal error, it displays a message but allows processing to continue. The general error message format, plus a list of all error messages, their meaning, and recommended corrective action, appear in Appendix A.

In most cases when a fatal error occurs, RBF-89 detects the error and aborts the command before its execution is initiated. But in other cases, such as a program-loading or file-saving operation, RBF-89 cannot detect the error until the command is partially executed; in such instances, the command may affect the system before its termination. For instance, if RBF-89 detects an error during file loading, a partial load has already taken place.

Interrupting RBF-89 Command Processing

At any time you may cancel an RBF-89 command by pressing the console ESC key. If you press this key while entering a command, RBF-89 cancels this command and prompts you for a new entry. If you press the ESC key while RBF-89 is executing a command, RBF-89 aborts execution of that command and prompts you for a new entry. Typically, you use the ESC key to halt an 8089 channel program already in progress, interrupt program loading or saving, or abort lengthy CRT displays or printed output.

Alphabetical List of Commands

To help you understand their interrelationship, the commands discussed in this chapter are grouped according to functional classification. As a reference aid, these commands are also summarized in alphabetical order in table 2-1. With this table, you may quickly locate any command discussion.

Table 2-1. Alphabetical List of RBF-89 Commands

Command	Function	Page
CA CONTINUE	Continue interrupted 8089 channel program.	2-34
CA HALT	Halt 8089 channel program.	2-37
CA START	Start 8089 channel program.	2-31
CHANNEL	Display or change current 8089 channel.	2-22
communication block	Display 8086/8089 communication blocks.	2-52
ENABLE RBR	Enable breakpoints without starting channel program execution.	2-44
EXIT RBF	Terminate RBF-89.	2-24
item-reference	Selectively display and change registers and remote memory locations.	2-46
LOAD	Load program code and symbol table.	2-26
RBF	Invoke and prepare to initialize RBF-89.	2-7
RBR	Display or change Breakpoint Register contents.	2-39
RBYTE/RWORD	Display 8089 memory contents without disassembly, or change 8089 memory contents. (Complements BYTE/WORD command in ICE-86 emulator.)	2-48
RESET RBF	Reset 8089 to initial state.	2-23
RESET RBR	Reset (clear) Breakpoint Registers.	2-42
SAVE	Save program code and symbol table on diskette.	2-27
UPDATE	Update software copy of all 8089 registers.	2-54
89 ASM	Disassemble and display local or remote memory contents into 8089 mnemonics.	2-51
89 REGISTER	Display all 8089 registers.	2-50

Command Elements

RBF-89 commands, like ICE-86 emulator commands, are made up of combinations of *characters* that are grouped into *tokens.*

Character Set

In all commands, RBF-89 accepts the same characters accepted by the ICE-86 emulator. These characters are selected from the American Standard Code for Information Interchange (ASCII) Character Set, X3.4, 1968, established by the American National Standards Institute (ANSI). RBF-89 ignores all other characters.

Tokens

RBF-89 uses the same types of tokens as the ICE-86 emulator: keywords, user-names, and special-tokens, The same rules apply in both systems for forming these tokens, with the following specific extensions for RBF-89 keywords that access various RBF-89 parameters:

- *Register Names,* which uniquely identify all 8089 registers. The 20-bit registers for the currently selected channel are listed below. The contents of these registers are represented internally by a 32-bit address pair that defines their base and displacement.

Name Referenced	Meaning
RGA	General-purpose registers A, B, and C.
RGB	
RGC	
RTP	Task Pointer (TP).

  The 16-bit registers for the currently selected channel are:

Name Referenced	Meaning
RBC	Byte Counter (BC).
RIX	Index Register (IX).
RCC	Channel Control (CC).
RMC	Mask/Compare (MC).

- *Remote Memory References,* which address the remote memory accessible only to the 8089, are written as:

Name Referenced	Meaning
RBYTE *partition*	Byte reference
RWORD *partition*	Word reference

  Local (system) memory references, which address memory accessible to both the 8086 and 8089, are made by using the ICE-86 memory-reference syntax defined in *ICE-86 In-Circuit Emulator Operating Instructions*.

- *8089 System Variables,* which designate specific parameters, are written as follows:

Name Referenced	Meaning
CCW	Control Word for currently selected channel.
PBP	Parameter Block Pointer for currently selected channel.

- *Status Flags,* which designate the following 8089 flags:

Name Referenced	Meaning
PS	8-bit channel program status word.
TGA, TGB, TGC	Tag bit for 20-bit general-purpose register.

For a complete list of all RBF-89 keywords, see Appendix B. Some keywords have single-letter synonyms, also listed in Appendix B. As with ICE-86 emulator keywords, you can alternate all RBF-89 keywords by entering their first three letters only.

Command Descriptions

The RBF-89 commands are described in functional order throughout the remainder of this chapter. For each command, the following information appears:

- *Syntax*—the syntax of the command.
- *Command Elements*—the command name and parameter tokens that appear in the command, and their meaning, rules, and constraints.
- *Default*—the action taken if you omit the command.
- *Operation*—a complete description of the function and operation of the command, including its relationship to other RBF-89 commands.
- *Examples*—sample entries illustrating the syntax and format of the command, with explanatory comments where helpful.

In the syntax descriptions, the same metalinguistic notation that appears in *ICE-86 In-Circuit Emulator Operating Instructions* is used. This notation, summarized in table 2-2, is based primarily upon standard Backus-Naur Form notation.

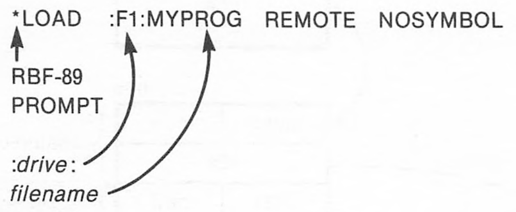
Table 2-2. Syntax Conventions

Notation	Description
UPPERCASE LETTERS	Elements in uppercase regular (Roman) type are specific keywords that must be entered exactly as shown (or as the abbreviations or synonyms listed in Appendix B).
lowercase italics	Elements in lowercase italics are class names that identify sets or classes of tokens. From each set, you select and enter a specific identifier.
. . .	Elements followed by an ellipsis (. . .) may be repeated indefinitely.
{ }	When elements are enclosed in braces, one (and only one) of these elements MUST be selected.
{ } . . .	When elements are enclosed in braces followed by an ellipsis, AT LEAST one of these elements MUST be selected. If more than one is selected, they may be entered in any order.
[ ]	When elements are enclosed in brackets, ALL elements are optional but only ONE may be entered. If only one element appears in brackets, that element is optional.
[ ] . . .	When elements are enclosed in brackets followed by an ellipsis, ALL elements are optional but MORE THAN ONE may be entered. Those elements selected may be entered in any order.

As an example of how this syntax notation is used, consider this RBF-89 command:

```
                                 [LOCAL  ] [NOCODE  ]
LOAD [:drive:] filename          [REMOTE ] [NOSYMBOL] . . .
```

In this command, LOAD and is a required literal keyword that must be entered exactly as shown. The *:drive:* parameter is a variable class name meaning "any disk drive"—you must specify which one; because it appears within brackets, *:drive:* is also an optional parameter that can be omitted. (The default for *:drive:* is :F0:.) The *filename* parameter is a class name meaning "any file name"; it is required. LOCAL and REMOTE are optional parameters denoted by common brackets. you may include one or the other, or neither. NOCODE and NOSYMBOL are optional parameters denoted by common brackets followed by an ellipsis; you may include both, one or the other, or neither. When you enter the LOAD command, it might appear as follows:



```
*LOAD   :F1:MYPROG   REMOTE   NOSYMBOL
```

After you have become generally familiar with the RBF-89 commands as described in this chapter, you may find the condensed syntax summary in Appendix C sufficient for reference while working at the console.

Invoking, Controlling, and Terminating RBF-89

To begin an RBF-89 debugging session, you must invoke RBF-89 and then initialize various system variables as described in the following pages. These variables provide the 8089 with a definition of the system configuration and environment to be used.

Invoking RBF-89

Before invoking RBF-89, first insert the ICE-86 emulator's 40-pin cable terminal into the 8086 CPU socket on your prototype system and invoke the ICE-86 emulator as directed in *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.* Then, after the emulator's sign-on message and initial prompt appear:

a. Enter the ICE-86 emulator **MAP** command to map 8086 RAM and the following 8089 control blocks to USER status:
   - System Initialization Block (SIB)
   - System Configuration Block (SCB)
   - Channel Control Block (CCB)

   The SIB resides at the fixed location FFFF:6H as shown in figure 2-1, but the SCB, CCB, and RAM may be mapped to any locations. More information about RBF-89's memory requirements appears under *Initializing RBF-89.*

b. Enter the **RBF** command to invoke RBF-89, using the syntax described in the command specification on the next page.

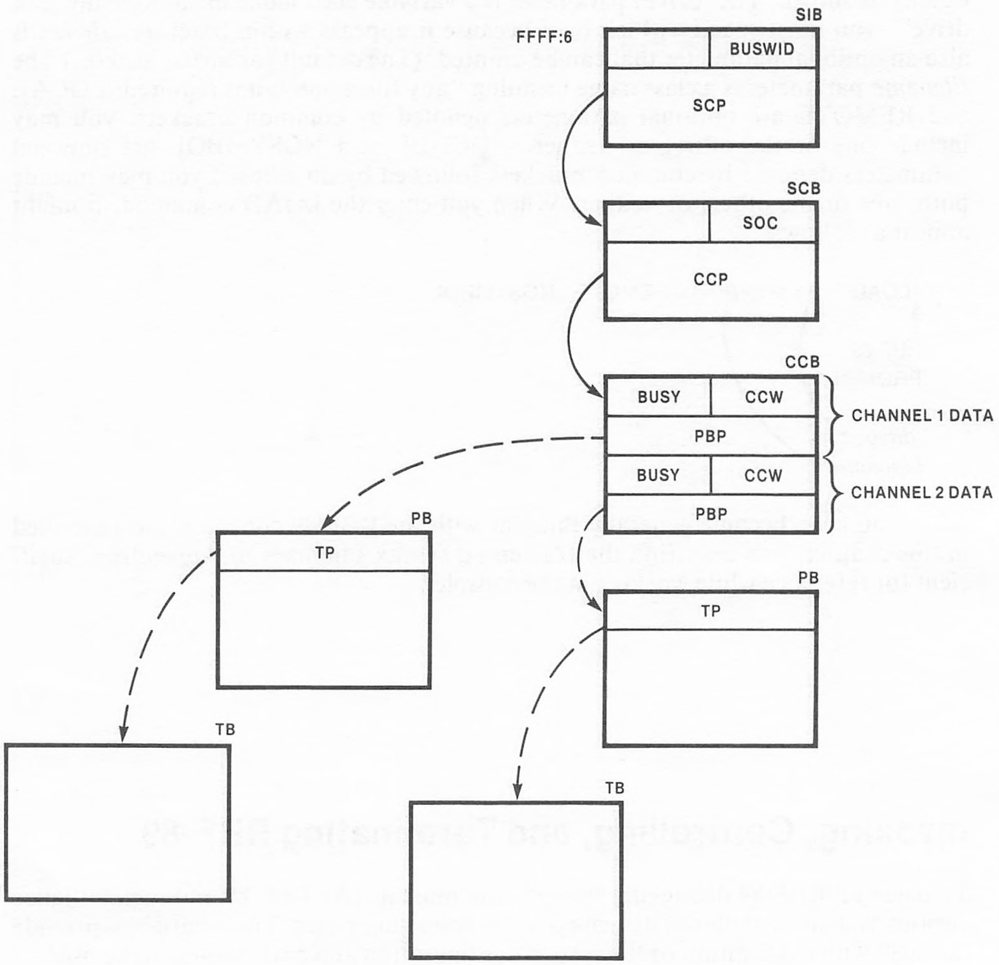


Figure 2-1. Memory Used by RBF-89

1018-02

RBF

Invokes RBF-89.

Syntax

```
RBF (CR)
subcommand (CR)
[subcommand] (CR)
      .
      .
      .
ENDRBF (CR)
```

Command Elements

RBF	Request to invoke RBF-89.
(CR)	Carriage-return that delimits each RBF command token (RBF command name and following subcommands).

NOTE

In this command format description, the notation (CR) is used for clarity to indicate a carriage-return. In subsequent format descriptions, however, (CR) is omitted even though the carriage-return is always implied at the end of each entry line.

subcommand	Specific RBF subcommand, as defined below, used to initialize one of several RBF-89 system variables.
ENDRBF	Request to terminate RBF subcommand list/entry sequence.

Default

None. (This command is required.)

Operation

The RBF command is a compound command that consists of a command name (RBF), followed by one or more subcommands that initialize RBF-89 system variables, followed by the terminating keyword ENDRBF. When you enter RBF followed by a carriage-return, the ICE-86 emulator invokes RBF-89 which then displays a period followed by an asterisk (.*) as a prompt for your first subcommand.

NOTE

The RBF subcommands are discussed after the conclusion of this description of the RBF command. You may enter these subcommands in any order, but may omit only those with default values. If you omit any required subcommand, an error message appears when you terminate the subcommand entry sequence.

At the end of the subcommand entry sequence (when you enter ENDRBF), RBF-89 transmits a channel-attention directive to initialize the 8089 input/output processor (IOP) and verify that it is functioning properly. After this verification, RBF-89 displays the following message on your console to confirm sign-on and issues an asterisk prompt for your next command:

```
ICE-86 RBF-89 Vx.y◄——SIGN-ON MESSAGE
*◄————————————————————PROMT FOR NEXT COMMAND
```

NOTE

In the sign-on message format, V*x.y* denotes the software version number.

After this message, you may begin your RBF-89 debugging session by entering any RBF-89 command desired. Because you can access the ICE-86 emulator from within RBF-89, you can also enter any ICE-86 emulator command at this point.

If the ICE-86 emulator memory is not available for RBF-89 use when you enter the RBF command, RBF-89 aborts this command and displays an error message. This occurs, for instance, if you have mapped your user program to emulator memory.

If the 8089 BUSY flag is not clear within 500 milliseconds after you enter the ENDRBF keyword, a timeout occurs and an error message appears. The timeout results from one of these possible causes:

a. Hardware not functioning properly.

b. {CH1/CH2} subcommand entered incorrectly.

c. SCP subcommand entered incorrectly.

After you have resolved any hardware problem and checked the appropriate subcommand syntax rules, re-enter the correct RBF command sequence.

Examples

```
*MAP 1023 = USER              ; Maps SIB, SCB, CCB, and RAM to USER
*                             ; status.
*RBF                          ; Invokes RBF-89.
.*RAM = FFC0:0                ; Specifies RAM available for
.*                            ; RBF-89.
.*CH1 = PORT 0                ; Denotes mechanism for
.*                            ; channel-attentions.
.*WIDTH = 16T                 ; Indicates system bus width.
.*SCP = FFF0:0                ; Initializes System Configuration
.*                            ; Pointer.
.*SOC = 1                     ; Initializes System Operation
.*                            ; Command Word.
.*CCP = FFF0:10               ; Initializes Channel Control
.*                            ; Pointer.
.*IOP = MASTER                ; Specifies Master/Slave status.
.*ENDRBF                      ; Terminates RBF subcommand entry
*                             ; sequence.
ISIS-II RBF-89 Vx.y
*                             ; Command entry prompt, preceded by
.*                            ; sign-on message.
```

NOTE

In this and all other examples, command input is *underlined* while system output is not. In addition, RBF-89 and ICE-86 emulator commands appear only on lines that begin with asterisk (*) prompt.

Always terminate each line by entering a carriage-return. Otherwise, the line is not transmitted to the system and processed.

Initializing RBF-89

Before you can use RBF-89, you must enter a group of *RBF subcommands* that provide information required for initializing and communicating with the 8089. Specifically:

- To initialize the 8089, RBF-89 must know how to issue channel-attention directives to the 8089. You provide this information by entering the **{CH1/CH2}** subcommand.
- When the 8089 receives the channel-attention directive, it fetches the system bus width (BUSWID) and the System Configuration Block (SCB) Pointer (SCP) from the System Initialization Block (SIB). (Refer again to figure 2-1.) The SIB always begins at fixed location FFFF:6H. You specify the BUSWID through the **WIDTH** subcommand (or use the default width of 16 bits), and the SCP setting through the **SCP** subcommand.
- Next, the 8089 fetches, from the SCB, the System Operation Command (SOC) word and the Channel Control Block (CCB) Pointer (CCP). You set these items with the **SOC** and **CCP** commands respectively.
- After RBF-89 initializes the 8089, the 8089 clears the BUSY flag in the CCB. (There is a CCB for each channel. If a channel program is running on a particular channel, the Parameter Block Pointer (PBP) in the CCB indicates the beginning location of the Parameter Block (PB) for that program. The PB, in turn, contains a Task Pointer (TP) that points to the Task Block for that program.)
- RBF-89 also needs to know the location of the RAM in your application system that is available for RBF-89 use. This memory block serves as a buffer for messages between the 8086 and 8089. You specify the location of this block with the **RAM** subcommand.
- Finally, as options, you may wish to specify the Master/Slave status of the 8089, the locations in system memory reserved for use by the RBF-89 MONITOR program, the relationship of the set of 8086 addresses to the set of 8089 addresses used to reference the same area of physical memory (in configurations involving the two-port memory concept of the SBC 86/12 board), and the base component for 20-bit register references. You specify these elements by entering the **IOP**, **RESERVE**, **RELOCATE**, and **PAGE** subcommands, respectively).

NOTE

For further information about the various elements mentioned above, please see *The 8086 Family User's Manual.*

To enter the RBF subcommands that initialize the 8089 for your debugging session, use the subcommand specifications shown in the following pages. Enter one subcommand after each subcommand prompt (.*), specifying the subcommands in any order you wish.

{CH1/CH2}

Specifies how RBF-89 issues channel-attention directives to 8089.

Syntax

```
        ⎧ PORT  ⎫
 ⎧CH1⎫  ⎪ WPORT ⎪
 ⎨   ⎬ =⎨ BYTE  ⎬  expression1 [VALUE expression2]
 ⎩CH2⎭  ⎪ WORD  ⎪
        ⎩       ⎭
```

Subcommand Elements

CH1	Request to direct channel-attention to channel 1.
CH2	Request to direct channel-attention to channel 2.
PORT	Mechanism is a 1-byte value transmitted to the addressed 8-bit input/output port.
WPORT	Mechanism is a 2-byte value transmitted to the addressed 16-bit input/output port.
BYTE	Mechanism is a 1-byte integer value written to a location in 8086 system memory.
WORD	Mechanism is a 2-byte integer value written to a location in 8086 system memory.
expression1	Expression whose value denotes port or memory location to which byte or word value is written.
VALUE	Request to specify value to be written to port or memory to cause channel-attention operation.
expression2	Expression whose value is to be written to port or memory. If omitted, default value is either *FF* (for byte value) or *FFFF* (for word value).

Default

None. (The CH1 [ATTENTION] form of this command is required in all cases.)

Operation

The mechanism which RBF-89 must use to issue a channel-attention signal to the 8089 depends on the way you have designed your system. As part of this mechanism, your application hardware delivers a pulse to the channel-attention (CA) pin of the 8089, and controls the value of the SEL pin during this pulse. The {CH1/CH2} subcommand tells RBF-89 how it can cause the pulse while the SEL pin is set to zero (directing the channel-attention to channel 1) and while the SEL pin is set to one (directing the channel-attention to channel 2).

The channel-attention mechanism involves three considerations:

a. Channel to be used for communication.

b. Physical entity (output port or 8086 system memory) used to trigger channel-attention.

c. Value transmitted to trigger channel-attention.

RBF requires communication with channel 1. Therefore you MUST enter a CH1 command as part of the initialization procedure. if you also plan to use channel 2, a CH2 command is necessary—otherwise no RBF commands for channel 2 are accepted.

If the 8089 is configured so that channel-attention directives are caused by output instructions to ports, you should specify that either:

a. A one-byte value will be transmitted to an 8-bit port (indicated by the PORT keyword).

b. A two-byte value will be transmitted to a 16-bit port (indicated by the WPORT keyword).

If the 8089 is memory-mapped, you should specify that either:

a. A one-byte integer value will be written to a location in 8086 system memory (indicated by the BYTE keyword).

b. A two-byte integer value will be written to a location in 8086 system memory (indicated by the WORD keyword).

Finally, with the VALUE *expression2* clause, you may specify the value that causes the channel-attention directive. If you omit this clause, the default value *FF* is assigned to eight-bit ports or memory-mapped input/output, and *FFFF* is assigned to sixteen-bit addresses.

Examples

```
.*CH1 = PORT 0          ; Enables channel 1, where
.*                      ; mechanism is one-byte value
.*                      ; (FF) transmitted to Port 0.
.*CH2 = WPORT 1;        ; Enables channel 1, where
.*                      ; mechanism is two-byte value
.*                      ; (FFFF) directed to
.*                      ; Port 1.
```

WIDTH

Specifies width of system bus.

Syntax

```
WIDTH = expression
```

Subcommand Elements

WIDTH	Request to specify width of system bus.
expresion	Expression whose value specifies bus width. This value must be either 8 or 16.

Default

WIDTH = 16T	T means the base is decimal.

Operation

If the CPU is an 8086 microprocessor, the system bus is 16 bits wide; if the CPU is an 8088, however, this bus is 8 bits wide. The **WIDTH** subcommand specifies the appropriate logical width of the bus to RBF-89, and also initilizes a flag in the 8089 System Initialization Block (SIB). If you omit the WIDTH subcommand, a default width of 16 bits is assigned.

Examples

```
.*WIDTH = .BUSWD   ; Assigns width denoted by the byte value
.*                 ; of the symbol BUSWD.
.*WID = 16T        ; Assigns width of 16 bits.
```

SCP

Sets System Configuration Block Pointer
(SCP).

Syntax

```
SCP = expression
```

Subcommand Elements

SCP	Request to initialize SCP.
expresion	Arithmetic expression that evaluates to an address that points to location of System Configuration Block in system memory.

Default

None. (This subcommand is always required.)

Operation

The SCP indicates the location of the System Configuration Block (SCB), which contains master/slave status information, the input/output bus width, and the pointer to the Channel Control Block (discussed below), used to initialize the 8089. To set the SCP, enter the **SCP** subcommand, specifying the desired SCB location. To determine the physical location of the SCB, the 8089 shifts the 16-bit relocation value left by four bits, and adds it to the address word. For more information about the SCP and SCB, see *The 8086 Family User's Manual.*

Example

```
.*SCP = 8000:0   ; Sets SCB address to 80000H.
```

SOC

Sets System Operation Command Word (SOC).

Syntax

```
SOC = expression
```

Subcommand Elements

SOC	Request to Initialize SOC word.
expression	Expression whose value is used to set SOC word.

Default

None. (This subcommand is required.)

Operation

Bit 1 (the R-bit) of the SOC word in the SCB controls the operation of the request/grant arbitration circuit in the IOP; Bit 0 (the I-bit) of this word sets the physical width of the input/output bus. The meaning of the bit settings are:

R Field

R=0

1. MASTER processor has the bus on initialization and normally has control of the bus.
2. SLAVE processor requests the bus from the MASTER when non-idle.
3. MASTER grants the bus to the SLAVE when idle.
4. SLAVE returns the bus to the MASTER when idle.

R=1

1. MASTER has the bus on initialization.
2. SLAVE requests the bus from MASTER when non-idle.
3. MASTER grants the bus to SLAVE only on an unchained instruction boundary, or wait for DMA (unless LOCK is active), or idle condition.
4. MASTER may request bus from SLAVE when non-idle. This may occur immediately.
5. SLAVE grants the bus to MASTER only on an unchained instruction boundary, or wait for DMA (unless LOCK is active), or idle condition.

I Field

I=0	The I/O bus width is 8 bits wide.
I=1	The I/O bus width is 16 bits wide.

RBF-89 uses no other bits in this word.

To set the SOC word, enter the **SOC** command, specifying the desired value.

Example

```
.*SOC = 1          ; Sets SOC word to value of 1.
```

CCP

Sets Channel Control Pointer (CCP).

Syntax

```
CCP = expression
```

Subcommand Elements

CCP	Request to Initialize CCP.
expression	Arithmetic expression that points to location of Channel Control Block (CCB).

Default

None. (This subcommand is required.)

Operation

The CCP denotes the location of the Channel Control Block (CCB) in the 8086's memory space, which contains the Channel Control Word (CCW), BUSY Flag, Parameter Block (PB) pointer, and other values that permit a channel to perform its activities. All 8086-to-8089 communications center on the CCB, half of which is dedicated to each channel. The CCW indicates the type of operation (command) that the 8089 is to perform. The BUSY flag indicates whether a channel program is executing on a given channel, and is cleared when program execution terminates. To set the CCP, enter the **CCP** subcommand, specifying the CCB location. For more information about the CCP and CCB, see *The 8086 Family User's Manual.*

Example

```
.*CCP = 8000:10    ; Sets CCB address to location 80010H.
```

RAM

Specifies location of RAM in application
system dedicated to RBF-89

Syntax

```
RAM = address
```

Subcommand Elements

RAM	Request to specify RAM available to RBF-89.
address	Starting address of RAM assigned to RBF-89.

Default

None. (This subcommand is required.)

Operation

To use RBF-89, you must reserve 200 decimal (C8 hexadecimal) bytes of application system RAM for use by RBF-89. The **RAM** subcommand indicates the location of this area. This memory block serves as a buffer for messages between the 8089 and the emulated 8086. When you enter the RAM subcommand, RBF-89 verifies that the map for this block of memory is set to USER status, and that the memory physically exists.

Examples

```
.*RAM = 9000   ; Assigns C8H bytes of RAM to RBF-89, beginning
.*             ; at Location 9000H.
.*RAM = A000   ; Assigns Locations A000H through A0C7H to
.*             ; RBF-89.
```

IOP

Specifies 8089's master/slave status.

Syntax

$$\boxed{\text{IOP} = \begin{Bmatrix} \text{MASTER} \\ \text{SLAVE} \end{Bmatrix}}$$

Subcommand Elements

IOP	Request to specify IOP master/slave status.
MASTER	8089 serves as master.
SLAVE	8089 serves as slave.

Default

```
IOP=MASTER
```

Operation

To satisfy the request/grant feature of the 8089, you must specify the status of the 8089 as either master or slave. (The first channel-attention directive received after system initialization or reset informs the 8089 through the SEL line whether this processor is a master or slave unit. If channel 1 is addressed, the 8089 is a master; if channel 2 is addressed, the 8089 is a slave.)

If you omit the IOP command from the initialization sequence, RBF-89 assigns the 8089 master status.

When the 8089 is assigned slave status, you must enter the CH2 subcommand, or an error results.

NOTE

A discussion of how the 8089 hardware SEL bit relates to set-up and hold-time during channel-attention operations, and a recommended circuit for satisfying set-up and hold-time requirements, appear in Chapter 3, (page 3-1).

Examples

```
.*IOP = MASTER   ; Assigns master status.
.*IOP = SLAVE    ; Assigns slave status.
```

RESERVE

Reserves a specific 2k bytes of memory for RBF-89 MONITOR.

Syntax

```
RESERVE expression
```

Subcommand Elements

RESERVE	Request to reserve 2k contiguous bytes of system memory for RBF-89 MONITOR.
expression	Expression whose value denotes segment number of first of two contiguous segments devoted to MONITOR.

Default

RBF-89 dedicates the first consecutive 2k bytes of GUARDED memory available after Location 1000H to MONITOR.

Operation

The **RESERVE** subcommand reserves 2k bytes of contiguous logical 8086 address space in system memory for the RBF-89 MONITOR program.

If you omit the **RESERVE** subcommand, RBF-89 allocates the first consecutive 2k bytes of GUARDED memory available after Location 1000H to MONITOR. But if you plan to locate your 8086 program in low memory, you should enter the RESERVE subcommand to avoid conflict with the space required by MONITOR—otherwise, you will not be able to use the area occupied by MONITOR. Once RBF-89 allocates the segments for MONITOR, either through the RESERVE subcommand or by default, you cannot remap the segments to your own use.

Example

```
.*RESERVE 10          ; Reserves area beginning with segment 10
.*                    ; (starting at Location 2800H).
```

RELOCATE

Permits reference of particular area of memory by two different sets of addresses.

Syntax

```
RELOCATE partition TO address
```

Subcommand Elements

RELOCATE	Request to permit reference to memory area by two different address sets.
partition	Expression whose value denotes range of addresses with respect to 8089. Must denote physical size of memory to be relocated.
address	Expression denoting beginning (base) of first byte or range with respect to 8086.

Default

None.

Operation

The **RELOCATE** subcommand supports the two-port memory concept of the SBC 86/12 Board. With this subcommand, you can specify that a given area of physical memory is referenced by one set of addresses from the viewpoint of the 8089, but a *different* set of addresses from the viewpoint of the 8086. Typically, you would use the RELOCATE subcommand primarily to allow RBF-89 to reference the System Initialization Block (SIB), System Configuration Block (SCB), Channel Control Block (CCB), and RAM buffer contents with addresses pertinent to the 8086.

Example

```
.*RELOCATE F800:0 LEN 32K TO 0  ; Logically relocates 32k of
.*                              ; address space to 8086
.*                              ; address space starting at
.*                              ; address 0.
```

PAGE

Defines base component for 20-bit register references.

Syntax

```
PAGE = expression
```

Subcommand Elements

PAGE	Request to define base component.
expression	Expression must evaluate to a simple integer.

Default

None (but this subcommand is not required—see below).

Operation

The PAGE subcommand defines the equivalent base address component for 20-bit values, resolving any ambiguities that may arise in base-displacement conversions of these values.

If you omit the PAGE subcommand, RBF-89 assumes the default base component of the address to be the most significant 4 bits of the reference specified, multiplied by 1000H. To illustrate, without the PAGE subcommand, the value 45678H is treated as 4000:5678. But, when you specify *PAGE=4500*, then 45678H is converted to 4500:0678.

NOTE

The PAGE subcommand, unlike any other RBF subcommand, can also be entered as a regular RBF-89 *command* outside the subcommand entry sequence.

Example

```
.*PAGE = 2000    ; Sets base to 2000H.
```

Controlling RBF-89

At any point in your RBF-89 session, you can control RBF-89 operation in the following ways:

- Determine which input/output channel is currently selected to receive command input, or switch command input to another channel (by using the **CHANNEL** command).
- Reinitialize 8089 to its pre-execution state (by using the **RESET RBF** command).
- Terminate RBF-89 operation (by using the **EXIT RBF** command.).

To use these commands, read the corresponding specifications on the next few pages.

CHANNEL

Displays or changes current channel.

Syntax

$$\text{CHANNEL}\left[=\begin{Bmatrix}\text{CH1}\\ \\ \text{CH2}\end{Bmatrix}\right]$$

Command Elements

CHANNEL	Request to display or change currently-selected channel.
CH1	Request to switch to Channel 1.
CH2	Request to switch to Channel 2.

Default

None. (Display only.)

Operation

The **CHANNEL** command, when entered with no subsequent parameters, displays the currently-selected channel in the following format (where *n* is the channel number):

CHANNEL=CH*n*

When you enter the CHANNEL command followed by an equal sign and keyword, all subsequent RBF-89 commands are directed to the channel you select. This CHANNEL command remains in effect until you enter another CHANNEL command.

NOTE

If you specify *CHANNEL=CH2*, you must have also specified the CH2 subcommand in the RBF command.

Example

```
*CHANNEL          ; Displays current channel.
CHANNEL=CH1
*CHANNEL = CH1    ; Sets current channel to 1.
*CHANNEL = CH2    ; Sets current channel to 2.
```

RESET RBF

Resets RBF-89 and 8089 to the state specified by the RBF command.

Syntax

```
RESET RBF
```

Command Elements

RESET RBF	Request to reset RBF-89.

Default

None.

Operation

The **RESET RBF** command re-establishes the parameters stored in the 8086/8089 communications block to the values given by the user in the RBF command and issues a channel-attention command to the 8089. This channel-attention command causes the 8089 to be re-initialized with parameters from the communications blocks. However, the breakpoint registers are unaltered and retain current breakpoint addresses. This command can restore control over a runaway 8089 channel program, but only if you first physically reset the 8086 (by pressing the reset function switch on your prototype system).

Examples

```
*RESET RBF
*RES RBF
```

EXIT RBF

Terminates RBF-89 operation.

Syntax

```
EXIT RBF
```

Command Elements

EXIT	Request to terminate RBF-89.
RBF	Request to return control to ICE-86 emulator.

Default

None.

Operation

To terminate RBF-89 but remain logged onto the ICE-86 emulator, use the EXIT command followed by the keyword RBF. When this command is executed, resources such as emulator memory and application system RAM are once again available to you. The original instructions at the breakpoints are restored.

To terminate both RBF-89 and ICE-86 emulator operation, returning control to ISIS-II, enter the EXIT command without the RBF keyword.

Examples

```
*EXIT RBF  ; Terminates RBF, remaining within ICE-86 emulator.
*EXIT      ; Terminates RBF and ICE-86 emulator, returning
*          ; control to ISIS-II.
```

Loading and Saving Channel Programs

Through RBF-89, you can load your channel program and its symbol table into remote 8089 RAM, and later save these items on any disk device.

LOAD

Loads 8089 channel program and/or symbol table into local or remote 8089 RAM.

Syntax

```
LOAD [:drive:] filename [LOCAL ] [NOCODE  ] ...
                        [REMOTE] [NOSYMBOL]
```

Command Elements

LOAD	Request to load object program code and/or symbol table.
:drive:	Diskette drive (such as :F0: or F1) where object code resides.
filename	Name of file in which object program resides.
LOCAL	Request to direct the program to local 8086 system RAM.
REMOTE	Request to direct the program to remote 8089 memory (the LOAD LOCAL is eqivalent to the ICE-86 LOAD command).
NOCODE	Request to suppress loading of program code.
NOSYMBOL	Request to suppress loading of symbol table.

Default

Default is LOCAL if LOCAL/REMOTE is omitted. If neither NOCODE or NOSYMBOL is entered, both program code and the symbol table are loaded.

Operation

Channel programs are written in ASM-89 assembler language and assembled into object code through the ASM-89 assembler. To load them into 8089 remote or local memory, use the **LOAD** command. This command performs either or both of the following functions:

- Loads your channel program, in object code form, from a disk device into 8086 system RAM or remote 8089 RAM.
- Appends symbols encountered in the object program file to the ICE-86 emulator symbol table (in the order they are encountered in the file).

The keywords NOCODE and NOSYMBOL direct RBF-89 to suppress loading the object program or symbol table, respectively.

If you wish to load the channel program into local (system) memory, you may do so with the LOAD LOCAL or the ICE-86 LOAD command. Before doing this, however, you should first map the memory area that will contain the program to USER status. If you wish to load the channel program into remote 8089 memory, you must do so with the LOAD REMOTE command.

Examples

```
*LOAD :F1:IOPROG REMOTE    ; Loads program IOPROG and symbol
*                          ; table.
*LOA MYPROG REM NOSYMBOL   ; Loads program MYPROG but not
*                          ; symbol table.
```

SAVE

Saves 8089 channel program and symbol table on ISIS-II file.

Syntax

```
                                ┌NOSYMBOL┐
SAVE [:drive:] filename [LOCAL ] │ NOCODE │
                        [REMOTE] └partition┘
```

Command Elements

SAVE	Request to save program code and/or current symbol table.
:drive:	Diskette drive on which program and/or symbol table are to be saved.
filename	Name of file in which program and/or symbol table is to reside.
LOCAL	Request to fetch program code from local 8086 system memory (equivalent to SAVE in ICE-86).
REMOTE	Request to fetch program code from remote 8089 memory.
NOSYMBOL	Request to suppress saving symbol table.
NOCODE	Request to suppress saving object code.
partition	Partition denoting locations in 8089 remote memory from which object code is to be saved.

NOTE

The SAVE command cannot contain both the NOCODE and *partition* parameters—these parameters are mutually-exclusive.

Default

Default is LOCAL if LOCAL/REMOTE is omitted.

Operation

The SAVE command performs either or both of these operations:

- Saves all or part of your channel program by copying it onto an ISIS-II file.
- Saves the entire ICE-86/RBF-89 symbol table in an ISIS-II file.

If you specify *:drive:* in this command, but the diskette on the denoted drive does not include the file *filename* in its directory, RBF-89 creates the file on this diskette. If you omit *:drive:*, RBF-89 writes this file to drive :F0:. If the file already exists, RBF-89 overwrites the existing data.

If you specify NOSYMBOL, RBF-89 does not save the symbol table. If you specify NOCODE, RBF-89 does not save the object code. (The NOCODE and *partition* parameters are mutually exclusive; if you include one, you must omit the other.) If you include *partition*, RBF-89 saves only the code stored in the address range indicated by *partition*. If you omit both NOCODE and *partition*, RBF-89 saves all code loaded by the last LOAD LOCAL or LOAD REMOTE command. (If no code was loaded, none is saved.)

Examples

```
*SAVE :F1:PROG1.SAV REMOTE NOSYMBOL .START TO .START + 50H
*                                                ; Saves code in
*                                                ; Locations .START
*                                                ; through .START + 50H.
*SAV :F1:PROG2.SAV REM .RESTART LEN 200H ;
*                                                ; Saves code in 200H
*                                                ; locations that begin
*                                                ; at location PROG2.SAV
*                                                ; .RESTART, and saves
*                                                ; symbol table.
```

Controlling 8089 Program Execution

RBF-89 allows you to control 8089 channel program execution in these ways:

- Control channel programs from the console (through channel-attention commands).
- Set, enable, and display breakpoints from the console (through breakpoint commands).
- Control channel programs from 8086 user programs (through the **ENABLE RBR** breakpoint command).

These capabilities are discussed in the following pages.

Controlling Programs from the Console

From your console, you can:

- Start execution of a channel program, directing the program to halt when a breakpoint is encountered.
- Continue an interrupted channel program.
- Halt a running channel program.

Breakpoint Operation

The main device for controlling channel program execution is the breakpoint, which specifies a location within the object code where execution is to terminate. Because each input/output channel is associated with one Breakpoint Register that can hold up to six addresses, you can establish up to six breakpoints in your program.

To use a breakpoint, you must both enter the breakpoint address value and enable the breakpoint. Entering the breakpoint value (or *setting* the breakpoint) means assigning the desired address value to the Breakpoint Register for the current channel. Enabling the breakpoint causes RBF-89 to write HALT instruction into the 8089 memory space at the breakpoint location, and save the overwritten instruction in temporary storage. When your program reaches a breakpoint during execution, the program halts. (Because the instruction at the breakpoint location is overwritten, this instruction itself is not executed.) At this point, you can examine 8089 registers and memory, perform various debugging analyses, and optionally continue program execution.

There are three ways to set and enable breakpoints:

a. Specify a *breaklist* parameter in a **CA START** or **CA CONTINUE** channel-attention command, as discussed below. This both sets the value of the breakpoint and enables that breakpoint.

b. Specify a value for *breaklist* with the **RBR** breakpoint command discussed later in this chapter, and then enable the breakpoint by using the *TILL RBR* clause in the CA START or CA CONTINUE command.

c. Specify a value for *breaklist* with the RBR command, and then enable it explicitly with the **ENABLE RBR** command discussed later in this chapter. This allows you to set breakpoints in the 8089 channel program without actually starting that program. Presumably in this case, your 8086 program will issue the necessary channel-attention signal to start the 8089 program.

   When channel program execution terminates, or when you enter the RBF-89 command, RBF-89 cancels the breakpoints by restoring the original instructions that were overwritten by the HALT instruction, and saved in temporary storage, disabling all breakpoints.

NOTE

If you request a breakpoint at the address where your program begins or resumes execution, this breakpoint is disabled for the duration of channel program execution and has no effect. To circumvent the problem of such temporarily-disabled breakpoints, you may set breakpoints in contiguous pairs so that the second breakpoint terminates execution when the first is disabled. You might wish to do this, for instance, when you want the 8089 to continue execution from the location at which it was last halted. For an example, see the discussion of the CA CONTINUE command.

To avoid confusion in your debugging operations, you should NOT set breakpoints in 8089 code common to both input/output channels.

Channel-Attention Command Operation

RBF-89 channel-attention commands let you start, stop, or continue channel programs from your console. When you enter a channel-attention command, RBF-89 places the command into the command field of the CCW, prepares the task pointer in the parameter block, sets the BUSY Flag in the CCB to 0FH to indicate that a program is about to begin, and issues a channel-attention directive to the appropriate channel. The channel reads the CCW from the CCB, examines the CCW's command field, sets the BUSY flag to FFH, and executes the command encoded there. (You can diagnose a non-functional channel by its failure to change the BUSY flag from its initial value of 0FH.) RBF-89 continually monitors the BUSY Flag until it is cleared to 00H by the channel, and then returns control to your console.

CA START

Begins channel program, optionally specifying one or more breakpoints.

Syntax

```
                                                   ┌ FOREVER        ┐
                  ┌ LOCAL  ┐                       │                │
CA START FROM  ─  ┤        ├  expression           │ TILL breaklist │   [RETURN]
                  └ REMOTE ┘                       │                │
                                                   └ TILL RBR       ┘
```

Command Elements

CA	Request to issue channel-attention directive.
START	Request to begin 8089 channel program.
FROM	Request to specify channel program starting address.
LOCAL	Specification that program is stored in local memory.
REMOTE	Specification that program is stored in remote memory.
expression	Arithmetic expression that denotes starting location of channel program in 8089 local or remote memory.
FOREVER	Request to continue program execution indefinitely (until normal termination or fatal error, or until ESC key is pressed).
TILL	Request to terminate program when any breakpoint specified in Breakpoint Register is encountered.
breaklist	List of one to six breakpoint addresses to be loaded into Breakpoint Register for current channel before program execution begins. Written in this format: $\left\{\begin{matrix}\text{LOCAL}\\\text{REMOTE}\end{matrix}\right\}$ *expression* $\left[,\left\{\begin{matrix}\text{LOCAL}\\\text{REMOTE}\end{matrix}\right\}\ \textit{expression}\right]$ . . . Execution at these breakpoint addresses will terminate program execution.
RBR	Request to use currently-loaded breakpoint addresses as conditions for terminating program execution.
RETURN	Request to return control to console after channel-attention directive is issued to 8089, without waiting for channel program completion.

Default

See below.

Operation

The **CA START** command begins 8089 channel program execution at the starting address you specify. It also allows you to specify up to six breakpoints for halting program execution.

Before entering the CA START command, you must explicitly set the Parameter Block Pointer (PBP) by using the **RBF-89 PBP =** command. For further information about setting the PBP, see Chapter 3, page 3-6.

The channel program must reside in either local or remote 8089 memory, as indicated by the keyword LOCAL or REMOTE respectively.

You may use the *breaklist* parameter to establish a breakpoint in either local or remote memory space.

If you neither specify a breakpoint (through the TILL *breaklist* or TILL RBR clause), nor indefinite execution (through the FOREVER keyword), the program-terminating conditions will be those established by the last CA START or CA CONTINUE command. (When you begin an RBF-89 debugging session, the initial terminating condition is FOREVER.)

Normally, control does not return to the console until the 8089 channel program completes execution. This, of course, prevents you from executing your 8086 user program simultaneously with your 8089 channel program. However, you may request immediate return of control to the console by specifying the RETURN keyword. With this option, RBF-89 starts the 8089 channel program and then immediately displays a prompt; this allows you to enter an ICE-86 emulator GO command to start your 8086 user program while the channel program is still running, thus executing both programs simultaneously.

NOTE

Under the RETURN option, once control returns to the console, channel program termination (via breakpoint or otherwise) is not reported to the console until the next RBF-89 command is issued.

When the channel program begins execution, the following message appears on the console:

```
8089 EXECUTION BEGUN
```

If the channel program terminates normally, this message appears:

```
8089 EXECUTION TERMINATED RTP = t nnnnn
```

In this message, *t* is the tag bit of the Task Pointer (RTP) and *nnnnn* is the RPT value.

If you selected the RETURN option, you may abort the channel program, by entering a CA HALT command. But if you did not specify the RETURN option, you must use the ESC key to abort the program or else wait for normal channel program termination.

If you halt the channel program by pressing the ESC key or entering a CA HALT command, this message appears:

```
8089 EXECUTION ABORTED RPT = t nnnnn
```

If the channel program reaches a breakpoint, this message appears:

```
8089 EXECUTION BREAKPOINT REACHED RPT = t nnnnn
```

Examples

```
*CA START FROM LOCAL F800:0000 TILL LOCAL F800:0050 &
**                                                     ; Begin program at
*                                                      ; location 0, break at
*                                                      ; location 50.
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 F8050H
*CA START FROM LOCAL F800:0 TILL LOCAL F800:10,LOCAL F800:20 &
**                                                     ; Break at location 10
*                                                      ; or 20.
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 F8010H
*RBR = LOCAL F800:10,LOCAL F800:20                     ; Set Breakpoint Register.
*CA START FROM LOCAL F800:0 TILL RBR                   ; Same as last CA START
*                                                      ; command.
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 F8010H
*CA START FROM LOCAL F800:0 TILL RBR RETURN            ; Same as above, with
*                                                      ; RETURN option.
8089 EXECUTION BEGUN
*CHANNEL = CH2                                         ; Redirects commands to
*                                                      ; Channel 2.
*CA START FROM LOCAL F800:0 TILL LOCAL F800:20,LOCAL F800:40 &
**                                                     ; Starts another program
*                                                      ; on channel 1.
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = .0 F8020H
```

CA CONTINUE

Resumes execution of a halted channel program.

Syntax

$$\boxed{\text{CA CONTINUE} \begin{bmatrix}\text{FOREVER}\\ \text{TILL } \textit{breaklist}\\ \text{TILL RBR}\end{bmatrix} \text{[RETURN]}}$$

Command Elements

CA	Request to issue channel-attention directive.
CONTINUE	Request to continue 8089 channel program.
FOREVER	Request to continue program execution indefinitely (until normal termination or fatal error, or until ESC key is pressed).
TILL	Request to terminate program when any breakpoint specified in breakpoint register is encountered.
breaklist	List of one to six breakpoint addresses to be loaded into breakpoint register for current channel before program execution resumes. Written in this format:

$$\begin{Bmatrix}\text{LOCAL}\\ \text{REMOTE}\end{Bmatrix} \textit{expression} \left[, \begin{Bmatrix}\text{LOCAL}\\ \text{REMOTE}\end{Bmatrix} \textit{expression} \right] \ldots$$

These breakpoints will then be used as conditions for terminating program execution.

RBR	Request to use currently-loaded breakpoints as conditions for terminating program execution.
RETURN	Request to return control to console after channel-attention directive is issued to 8089, without waiting for channel program completion.

Default

See below.

Operation

When you wish to resume execution of an 8089 channel program that was halted (prior to completion) by a breakpoint or a CA HALT command, use the **CA CONTINUE** command. This command continues execution from the location where the halt occurred. As in the CA START command, you may specify up to six breakpoints for halting the requested execution and may also request the RETURN option. You may NOT, however, specify a starting address. The rules for breakpoint specification and the messages indicating the beginning and termination of program execution are the same as noted in the CA START command discussion.

When you enter the CA CONTINUE command, RBF-89 copies the currently-saved TP and PSW values into the first two words of the parameter block and issues a channel-attention directive to continue 8089 execution.

NOTE

Do not attempt to change the content of the PSW—otherwise, unpredictable results may occur.

If you did not select the RETURN option, you may abort the channel program and any RBF-89 command in process by pressing the Escape (ESC) key on the console. But if you did specify RETURN, you must enter the CA HALT command to abort the channel program.

As noted earlier under *Breakpoint Operation,* if you request a breakpoint at the address where your program begins or resumes execution, this breakpoint is disabled for the duration of channel program execution and has no effect. But, suppose you want the 8089 to continue execution from the breakpoint at which it was last halted, and expect the program to loop through this temporarily-disabled breakpoint one or more times. To circumvent the problem of the disabled breakpoint, you may set another breakpoint at the *next* instruction in your program. Thus, when the first of these contiguous breakpoints is disabled, the second breakpoint terminates execution, and vice-versa. For instance, suppose your program contained the following loop, and you wished to enable breakpoints of the instructions labeled BRK1 and BRK2.

```
           .
           .
           .
LOOP:      LDPI [PP], PTR
           .
           .
           .
BRK1:      MOV GA [PP], 10
BRK2:      MOV GB [PP], 13
           .
           .
           .
           JNZ BC .LOOP
           .
           .
           .
```

You could set the breakpoints with the following RBR command:

```
*RBR = LOCAL .BRK1, LOCAL .BRK2
```

You could then run your program in increments, breaking alternately on .BRK1 and .BRK2, as follows:

```
             .
             .
             .
*CA START FROM .BEGIN TILL RBR
8089 EXECUTION BEGUN
8089 BREAKPOINT REACHED RTP = .BRK1
*CA CONTINUE
8089 EXECUTION BEGUN
8089 BREAKPOINT REACHED RTP = .BRK2
*CA CONTINUE
8089 EXECUTION BEGUN
8089 BREAKPOINT REACHED RTP = .BRK1
             .
             .
             .
```

Examples

```
*CA CONTINUE                                ; Resumes program execution.
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 F8040H
*CA CONTINUE TILL F800:30, LOCAL F800:40    ; Continues program until
*                                           ; 30, 40.
8089 EXECUTION BEGUN
8089 EXECUTION TERMINATED RTP = 0 F8072H
*CA CONTINUE FOREVER RETURN                 ; Resumes program
*                                           ; indefinitely with RETURN.
8089 EXECUTION BEGUN
*                                           ; ESC key pressed, but no
*                                           ; terminating message.
*
*CA HALT                                    ; Requests channel-attention
*                                           ; halt.
8089 EXECUTION TERMINATED RTP = 0 F8072H
*                                           ; Terminating message
*                                           ; appears after CA command.
```

CA HALT

Halts channel program.

Syntax

```
CA HALT
```

Command Elements

CA	Request to issue channel-attention directive.
HALT	Request to terminate 8089 channel program.

Default

None.

Operation

When an RBF-89 command prompt is displayed, you may use the **CA HALT** command to abort an 8089 channel program presently in execution. The CA HALT command also directs RBF-89 to dump the contents of TP, PSW, and all other registers (except CP and PP) in effect when the program halts. If the channel program has already terminated when you issue the CA HALT command, this command is ignored.

When an RBF-89 command prompt is NOT displayed, you must press the ESC key to abort the channel program. This action causes RBF-89 to implicitly issue a CA HALT command.

Example

```
*CA HALT          ; Halts channel program.
8089 EXECUTION ABORTED TP = 0 01020H
```

Setting, Enabling, and Disabling Breakpoints

As you have seen, you can set and enable breakpoints in your 8089 channel program through the CA START and CA CONTINUE channel-attention commands that initiate program execution. Alternatively, you also can set and enable breakpoints without requesting program execution, and display the contents of the Breakpoint Registers, by entering the *breakpoint commands* **RBR** and **RESET RBR**, described in the following pages.

When controlling the channel program from the console, you may wish to use breakpoint commands because of their convenience in particular cases. When controlling the channel program from an 8086 user program, you *must* use breakpoint commands to establish breakpoint register contents. (This function is discussed later in this chapter, under *Controlling IOP Program from CPU Program.*)

RBR

Displays or changes Breakpoint Register contents.

Syntax

```
RBR [=breaklist]
```

Command Elements

RBR	Request to access Breakpoint Register for currently-assigned channel.
breaklist	List of one to six breakpoints to be loaded into Breakpoint Register for current channel. Written in this format: $\left\{\begin{matrix}\text{LOCAL}\\\text{REMOTE}\end{matrix}\right\}$ *expression* $\left[,\left\{\begin{matrix}\text{LOCAL}\\\text{REMOTE}\end{matrix}\right\}\right.$ *expression*$\left.\right]$ ...

Default

None.

Operation

The **RBR** command displays the contents of the Breakpoint Register for the currently-assigned channel or loads this register with the breakpoints you specify. The breakpoints are not actually entered in your program, however, until the CA START, CA CONTINUE, or ENABLE RBR command is issued.

To display the current contents of the Breakpoint Register, enter the RBR command with no subsequent parameters. The display appears in this format:

```
RBR = breaklist
```

To either set the contents of the Breakpoint Register initially or to alter the current contents of this register, follow the RBR command name with an equal sign and the *breaklist* parameter. (The rules for breakpoint specification via *breaklist* are the same as those described for the CA START and CA CONTINUE command.) RBF-89 clears the current contents before entering the new addresses into the register. From this point on in your debugging session, you may start the channel program by simply referencing START and CA CONTINUE commands rather than explicitly setting the breakpoints with each instance of these commands. This technique is particularly useful when referring to the same group of breakpoints through several successive CA START or CA CONTINUE commands. For example, the following sequence of commands:

```
*CA START FROM LOCAL 1000 TILL LOCAL 1050, LOCAL 1060, LOCAL 1070, LOCAL 1080
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
*CA CONTINUE FOREVER
      .
      .
      .
*CA START FROM LOCAL 1010 TILL LOCAL 1050, LOCAL 1060, LOCAL 1070, LOCAL 1080
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
*CA CONTINUE FOREVER
      .
      .
      .
*CA START FROM LOCAL 1020 TILL LOCAL 1050, LOCAL 1060, LOCAL 1070, LOCAL 1080
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
      .
      .
```

could be entered more simply as:

```
*RBR=LOCAL 1050, LOCAL 1060, LOCAL 1070, LOCAL 1080
*CA START FROM 1000 TILL RBR
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
*CA CONTINUE FOREVER
  .
  .
*CA START FROM LOCAL 1010 TILL RBR
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
*CA CONTINUE FOREVER
  .
  .
  .
*CA START FROM LOCAL 1020 TILL RBR
8089 EXECUTION BEGUN
8089 EXECUTION BREAKPOINT REACHED RTP = 0 01050H
  .
  .
  .
```

Examples

```
*CHANNEL = CH1                        ; Selects Channel 1.
*RBR                                  ; Displays Channel 1 Breakpoint
                                      ; Register.
RBR = 0 F8050H
*CHANNEL = CH2                        ; Selects Channel 2.
*RBR                                  ; Displays Channel 2 Breakpoint
                                      ; Register.
NO BREAK POINT(S) SET
*RBR = LOCAL 0050H                    ; Sets Breakpoint Register to
                                      ; local address 0050H.
*RBR = LOC .START+10H,LOC .START+20H,LOC .START+30H,LOC .START+40H, &
**LOC .START+50H,LOC .START+60H       ; Sets Breakpoint Register for 6
*                                     ; locations.
*RBR                                  ; Displays Channel 2 Breakpoint
*                                     ; Register.
*
```

RESET RBR

Resets contents of Breakpoint Register for current channel, or both Breakpoint Registers.

Syntax

```
RESET [BOTH] RBR
```

Command Elements

RESET	Request to perform reset (clearing) operation.
BOTH	Request to perform this operation on both Breakpoint Registers.
RBR	Request to access one or both Breakpoint Registers.

Default

None.

Operation

The **RESET RBR** command resets one or both Breakpoint Registers by clearing them to zero. This command physically removes all breakpoints from your program.

If you include the *BOTH* keyword, RBF-89 resets the breakpoint registers for both input/output channels. If you omit *BOTH,* RBf-89 resets the register for the currently- assigned channel.

Examples

```
*RESET RBR        ; Resets the Breakpoint Register for
                  ; current channel.
*RES BOTH RBR     ; Resets Register for both channels.
```

Controlling IOP Program from CPU Program

In certain situations, it is necessary to set breakpoints without starting an 8089 channel program from the console. This is required, in fact, when you want to control the execution of the channel program from an 8086 program, allowing the 8086 program to drive the 8089 program in a master/slave relationship. You may do this, for instance, when verifying the 8086 driver program. For this type of operation, you:

a. Load breakpoint addresses into the Breakpoint Register for the selected 8089 channel (with the RBF-89 **RBR** command).

b. Enable the breakpoints in your 8089 program (with the RBF-89 **ENABLE RBR** command).

c. Start your 8086 program (with the ICE-86 emulator **GO** command).

This procedure requires that you either:

a. Establish a protocol between your 8086 and 8089 programs to differentiate between normal, error, and breakpoint termination conditions. This is necessary so that you can specify the ICE-86 emulator GO commands in such a way that emulation will break when the 8086 program detects that the 8089 program has reached a breakpoint.

b. Cause the ICE-86 emulator to break whenever the 8086 is about to issue a channel-attention directive, and check the cause of the 8089 program's termination. This will prevent the 8086 from issuing the channel-attention directive to the 8089 if an 8089 breakpoint has already been reached.

ENABLE RBR

Enables breakpoints.

Syntax

```
ENABLE [BOTH] RBR
```

Command Elements

ENABLE	Request to perform enable operation.
BOTH	Request to enable both Breakpoint Registers.
RBR	Request to access one or both Breakpoint Registers.

Default

None.

Operation

After you have set a breakpoint with the RBR command, you may explicitly enable it with the **ENABLE RBR** command. This allows you to set breakpoints in the 8089 channel program without actually starting this program. In this case, your 8086 program is to issue the necessary channel-attention signal to start the 8089 program.

The ENABLE RBR command places HALT instructions at the locations in your channel program (or programs) specified by the contents of one or both Breakpoint Registers. If you include the **BOTH** keyword, the breakpoints from both registers are established in the channel programs for the respective channels. If you omit **BOTH**, only the breakpoints from the register for the currently-selected channel are enabled in the program for that channel.

NOTE

The original instructions are restored to the breakpoint locations when you enter the next RBF-89 command. Thus, the ENABLE RBR command should be the *last* RBF-89 command you enter before starting your 8086 program.

Example

```
*RBR = .TASKAD + 50H  .TASKAD + 100H, TASKAD + 150H
                                         ; Loads 8089
*                                        ; breakpoints into
*                                        ;registers.
*ENABLE RBR                              ; Enables 8089
*                                        ; breakpoints.
*GO TILL .TASKDONE                       ; Starts 8086
*                                        ; program, which
*ENA BOTH RBR                            ; drives 8089
*                                        ; program.
```

Displaying and Changing 8089 Register and Memory Contents

RBF-89 recognizes unique keyword operators for all user-programmable 8089 registers (except CP and PP, as noted below), 8089 status flags, and other 8089 system variables. By using these keywords in RBF-89 *data-access commands,* you can examine or alter the contents of these variables. By using ICE-86 emulator display/change commands, you can also similarly access 8086 variables while logged onto RBF-89.

NOTE

This register is loaded only once for the System Control Block (SCB) used during 8089 initialization.

RBF-89 also lets you display or change the contents of remote 8089 memory and input/output ports. The memory displays may be requested in either numeric-data or disassembled mnemonic-code form. You may also apply ICE-86 emulator symbol-manipulation commands to 8089 symbolic references. To display or change 8086 memory and ports, you must use ICE-86 emulator memory-reference commands. Expressions and symbolic references are evaluated according to the syntactic and semantic rules described in *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.*

The RBF-89 commands for displaying or changing 8089 registers and memory and for displaying 8086/8089 communication blocks, are described in the following pages. The displays all appear in the current output base as specified by the ICE-86 emulator BASE command.

item-reference

Selectively displays or changes 8089 registers or memory locations, or 8086 registers, memory locations, or port values.

Syntax

$$\textit{item-reference} \begin{Bmatrix} [,\textit{item-reference}]\ \ldots \\ = \textit{expression} \end{Bmatrix}$$

Command Elements

item-reference	Item to be displayed or changed, or selected from:

- 8089 Register: RGA, RGB, RGC, RTP, RBC, RIX, RCC, RMC, PS, TGA, TGB, TGC.
- 8089 Memory Location, in this format:

```
RBYTE
         address
RWORD
```

- 8086 Register, as defined in *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.*
- 8086 Memory Location, in this format:

```
BYTE
WORD
POINT    address
INT
SINT
```

- 8086 Port, in this format:

```
PORT
         address
WPORT
```

expression	Expression whose value becomes the contents of *item-reference.*

Default

None.

Operation

This command displays or changes 8089 or 8086 registers, status flags, or memory locations, or 8086 input/output ports.

Typing the *item* name alone displays the value of the item. Entering the *item* name, followed by an equal sign, followed by a new value sets the item to that value.

In all displays, the data identifiers and their contents are presented consecutively, separated from each other by a space. If any line of the display would contain more than 80 characters, the references are continued on the next line.

The tag bits for the 20-bit registers appear in this format, where *bit-value* is either 1 or 0:

```
TGA = bit-value
TGB = bit-value
TGC = bit-value
```

The 20-bit register values appear in the format shown below. The *tag* portion is *0* when the register is loaded with a local memory pointer, or *1* when the register contains remote memory pointer. The *pointer-value* is a 20-bit quantity followed by an H suffix that denotes the hexadecimal output display base.

```
RGA = tag pointer-value
RGB = tag pointer-value
RGC = tag pointer-value
RTP = tag pointer value
```

The 16-bit registers appear in this format:

```
RBC = word-value
RIX = word-value
RCC = word-value
RMC = word-value
```

The Program Status Word is displayed in this manner:

```
PS = byte-value
```

In the displays of individual 8089 memory locations, the data appears in this format:

```
{RBY}
{RWO}  base:displacement = location-content
```

NOTE

For further information about 8089 registers and memory references, see *The 8086 Family User's Manual.*

For information about 8086 registers, memory references, and input/output ports, and their display formats, see *The 8086 Family User's Manual* and *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.*

Examples

```
*RGA                        ; Displays Register A.
RGA=0 00000H
*RGA, RGB, RGC              ; Displays Registers GA, GB, and GC.
RGA=0 00000H RGB=0 00000H RGC=0 00000H
*RBYTE 1083H                ; Displays byte 1083H in remote memory.
RBY 0000:1083H=00H
*RGA=20:40                  ; Sets Register RGA.
*TGA=1                      ; Sets Tag Bit of Register GA.
*RGA                        ; Displays contents of Register GA.
RGA=1 00240H
```

RBYTE
RWORD

Displays or changes contents of 8089 remote memory.

Syntax

```
{RBYTE}
{RWORD} partition [= expression [,expression] . . .
```

Command Elements

RBYTE	Request to display or change contents of one or more bytes of remote memory.
RWORD	Request to display or change contents of one or more words of remote memory.
partition	Range of locations to be displayed or changed, entered in one of three formats: a. *address* b. *address*1 LENGTH *address2* c. *address*1 TO *address2*
expression	The new value to be entered in the area of remote memory indicated by *partition.* May be an arithmetic expression, string, or memory input/output reference (entered in the format {RBYTE/RWORD} *partition*).

Default

None.

Operation

This command displays the contents of selected remote memory partition in numeric data form, or changes these contents. (In the display, disassembly does not take place.) Changes are normally made to patch-in changes to resident object code or data.

To produce the display, enter the command in this format:

```
{RBYTE}
{RWORD} partition
```

The display appears in this format:

```
*RBYTE 0 LENGTH 10
RBY 0000:0000H=11H 22H 33H 44H 55H 66H 77H 88H 99H AAH BBH CCH DDH EEH FFH 00H
*RWORD 10 TO 2E
RWO 0000:0010H=1111H 2222H 3333H 4444H 5555H 6666H 7777H 8888H
RWO 0000:0020H=9999H AAAAH BBBBH CCCCH DDDDH EEEEH FFFFH 0000H
```

To change the contents of memory, enter the command in this format:

```
{RBYTE}
{RWORD}  partition = expression [,expression] . . .
```

If you specify the *partition* to be changed in the *address* format, so that no upper limit or length is defined for the *partition*, storage of the new data begins at *address* and continues until all memory required by this data is used.

If you specify the *partition* to be changed in the *address1* TO *address2* or *address1* LENGTH *address2* format, so that an upper limit or length is specified, then the new data begins at *address1* and is repeated until the *partition* is filled. If the *partition* range is shorter than the new data entered, only the locations within the *partition* limits are changed and an error message is displayed.

Examples

```
*RBYTE 1052H                    ; Displays one byte.
RBY 0000:1052H=09H
*
*RWORD 1200H TO 1260H           ; Displays 62H bytes.
RWO 0000:1200H=000FH 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1210H=0000H 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1220H=0000H 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1230H=0000H 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1240H=0000H 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1250H=0000H 0000H 0000H 0000H 0000H 0000H 0000H 0000H
RWO 0000:1260H=0000H
*RBYTE 1000 LEN 4               ; Displays 4 bytes.
RBY 0000:1000H=0FH 00H 00H 00H
*RBYTE 1083H = 00H, 40H, 18H    ; Patches code beginning at
*                               ; Location 1083H.
*RWORD 1000 LEN 20 = 00H        ; Fills 10H words with zeros.
*
```

89 REGISTER

Collectively displays contents of all 8089 registers.

Syntax

```
89 REGISTER
```

Command Elements

89	Request to access 8089 registers.
REGISTER	Request to display registers.

Default

None.

Operation

This command displays the 8089 registers in the following order: RGA, RGB, RGC, RTP, RBC, RIX, RMC, and RCC. Where duplicate registers exist for both channels, only those for the currently-selected channel appear. In the displays, the first line shows the pointers in hexadecimal notation, and the remaining lines show the 16-bit registers in the current output base.

Example

```
*89 REGISTER
RGA=1 11111H RGB=0 22222H RGC=1 33333H RTP=0 44444H
RIX=6666H RBC=5555H RMC=8888H RCC=7777H
```

89 ASM

Disassembles and displays memory.

Syntax

```
89 ASM [LOCAL ] partition
       [REMOTE]
```

Command Elements

89	Request to access 8089 memory.
ASM	Request to display memory contents in disassembled form.
LOCAL	Request to display local memory.
REMOTE	Request to display remote memory.
partition	Range of locations to be displayed, entered in one of these formats: a. *address* b. *address1* LENGTH *address2* c. *address1* TO *address2*

Default

Default is to LOCAL when LOCAL/REMOTE is omitted.

Operation

This command disassembles and displays object code in remote or local 8089 memory, showing the code in assembly-language mnemonic form. If you specify REMOTE, RBF-89 displays remote memory. Otherwise, RBF-89 presents local memory. The *partition* parameter must define the boundary of a single instruction, or an address range that encompasses a discrete number of 8089 instructions.

NOTE

To aid you in interpreting this display, the complete 8089 IOP instruction set appears in Appendix D.

Examples

```
*89 ASM 1000H TO 100AH        ; Disassembles local memory 1000 through 100A.
ADDR       PREFIX         MNEMONIC  OPERANDS                          COMMENTS
0000:1000H                  INCB    [GA]
0000:1002H                  JBT     [GA],00H,$+12H                    ; SHORT
0000:1005H                  LPDI    GA,5678H                          ; 1234H
*89 ASM REMOTE 0021H LEN 0EH    ; Disassembles a 0EH-byte remote partition.
ADDR       PREFIX         MNEMONIC  OPERANDS                          COMMENTS
0000:0021H                  MOVI    [GA],1234
0000:0025H                  NOTB    [GA]
0000:0027H                  TSL     [GA],12H,$+12H                    ; SHORT
0000:002BH                  WID     8,8
0000:002DH                  XFER
```

communication-block

Displays contents of System Initialization Block (SIB), System Configuration Block (SCB), and/or Channel Control Block (CCB)

Syntax

```
communication-block
```

Command Elements

communication-block	Item to be displayed, selected from these entries:

Entry	Item Referenced
SIB	System Initialization Block.
SCB	System Configuration Block.
CCB	Channel Control Block.

Default

None.

Operation

This command displays, in formatted output, the SIB, SCB, or CCB. The general format is illustrated below, using a CCB display as an example:

```
ADDRESS  OFFSET  MNEMONIC  DATA VALUE

80010:   +00:    CCW  CH1: 00H
80011:   +01:    BUSY CH1: 03H
80012:   +02:    CPB1 ADR: 1F00H
80014:   +04:    CPB1 REL: 0000H
```

In this format:

- *Address* shows the location of this byte in local memory, displayed as five hexadecimal digits (without a suffix denoting the base).
- *Offset* indicates the displacement of this byte from the beginning of the block displayed. The offset appears as a plus sign followed by two decimal digits.
- *Mnemonic* is an eight-character abbreviation that identifies the function of the data value displayed in the next column.
- *Data Value* is the value of the data described in the preceding column. It appears as an 8- or 16-bit quantity followed by a single-letter suffix that denotes the current display base.

NOTE

For further information about the contents of the SIB, SCB, and CCB, see *The 8086 Family User's Manual*.

Examples

```
*SIB
 FFFF6:  +00:  SYSBUS   :  01H
 FFFF8:  +02:  SCB ADDR:  8000H
 FFFFA:  +04:  SCB REL :  0000H

*SCB
 08000:  +00:  SYS OP C:  03H
 08002:  +02:  CCB ADDR:  8100H
 08004:  +04:  CCB REL :  0000H

*CCB
 08100:  +00:  CCW  CH1:  08H
 08101:  +01:  BUSY CH1:  00H
 08102:  +02:  CPB1 ADDR: 8200H
 08104:  +04:  CPB1 REL:  0000H
```

UPDATE

Updates software copy of all 8089 registers.

Syntax

```
UPDATE
```

Command Elements

UPDATE	Request to update software copy of all 8089 registers.

Default

None.

Operation

The UPDATE command reads the current contents of all 8089 registers, including RTP and PS, and updates the software copy of these registers used by RBF-89. You should enter this command when you run the 8089 through a mechanism other than RBF-89, but want to use RBF-89 to examine register contents. (You may wish to do this, for example, in the case of an 8089 program driven by the 8086.) In such instances, you should issue the UPDATE command immediately after the 8089 program breaks, just before entering the RBF-89 command to display the desired register contents; this ensures that current, valid RTP and PS register contents appear in the display.

Example

```
*UPDATE
```

CHAPTER 3 APPLICATIONS

This chapter explains how to load RBF-89 and use this product to perform typical debugging tasks on 8086/8089 microprocessor-based systems.

8089 Configuration Designing Hints

During the RBF-89 initialization sequence, the SEL pin setting (and the R bit in the System Operation Command word, SOC) determine the MASTER/SLAVE status of the IOP. After the initialization sequence is complete, the SEL pin setting determines which channel is being addressed. The SEL signal must be stable for 50 nanoseconds before the CA signal is transmitted. To toggle the CA and SEL pins, you may connect one of the available address (ADDR) lines to the SEL pin and feed the remaining ADDR lines into a combination of logic circuits to produce a signal that is gated with the $\overline{\text{IOWC}}$ or $\overline{\text{MWTC}}$ signal to result in a CA signal. If you use the $\overline{\text{IOWC}}$ signal, you must transmit output to an output port to produce the CA command. If you use the $\overline{\text{MTWC}}$ signal, you must write to a memory location to produce the CA command; this memory location is determined by the logic combination used in your design.

As an example, using the $\overline{\text{IOWC}}$ signal with the logic shown in figure 3-1, an *OUT 0* instruction produces a CA command for Channel 1 during normal dispatching of an input/output task. With this same logic, an *OUT 1* instruction produces a CA command for Channel 2.

It is important that you design your prototype circuit to transmit an $\overline{\text{XACK}}$ signal to the 8086 to acknowledge receipt of CA commands from the IOP. The easiest way to do this is to invert the CA signal back to produce the $\overline{\text{XACK}}$ signal.

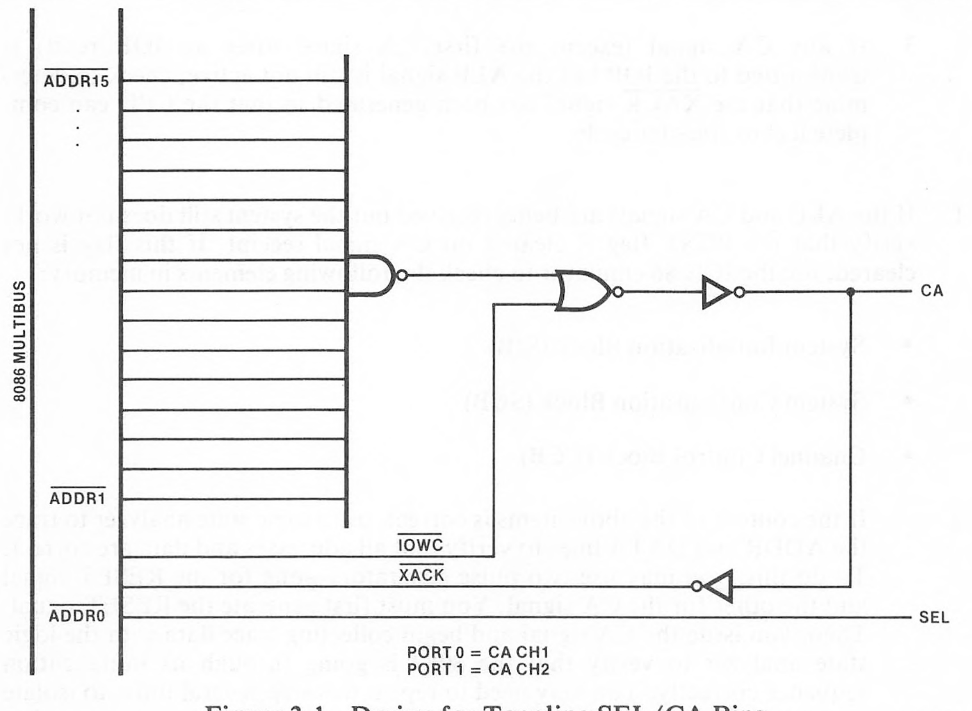


Figure 3-1. Design for Toggling SEL/CA Pins

Assumptions and Prototype Verification

Before using RBF-89, you must develop and debug your application system to the level of readiness described under *Assumptions and Prerequisites for Using RBF-89* in Chapter 1. (Generally, you perform this debugging with the ICE-86 emulator and various hardware debugging tools.) To verify that your prototype system hardware is functioning properly, follow these steps in the sequence shown:

a. Check all interconnections between the main components of your system to ensure that they are not loose.

b. Check all power supplies for short circuits.

c. Plug the 8089 IOP and other components into the system, turn on the power to all components, and check all power supply levels.

d. Try to invoke and initialize RBF-89. If you succeed, you may start your software debugging session. Otherwise, proceed to Step e below.

e. Check to determine that there is an ALE signal from the 8288 bus controller or that the Status signals (S0, S1, and S2) are in an active state. If there is no ALE signal or if the Status signals are in a passive state:

   1. Use an oscilloscope or other suitable tool to check the CL$\overline{\text{OCK}}$, ST$\overline{\text{ATUS}}$, READY, and RESET signals on the IOP. Also, check the $\overline{\text{INIT}}$ and $\overline{\text{BCLK}}$ signals into the 8289 bus arbiter, and the priority of the bus arbiter.

   2. Check the CA and SEL signals to ensure that they are being issued correctly.

   3. If any CA signal (except the first CA signal after an IOP reset) is transmitted to the IOP but the ALE signal is still not active, check to determine that the $\overline{\text{XACK}}$ signal has been generated so that the CPU can complete its bus transfer cycle.

f. If the ALE and CA signals are being received but the system still does not work, verify that the BUSY flag is cleared on CA signal receipt. If this flag is not cleared, use the ICE-86 emulator to check the following elements in memory:

   - System Initialization Block (SIB)
   - System Configuration Block (SCB)
   - Channel Control Block (CCB)

g. If the content of the above items is correct, use a logic state analyzer to trace the ADDR and DATA lines to verify that all addresses and data are correct. To do this, you may use two pulse generators—one for the RESET signal and the other for the CA signal. You must first generate the RESET signal. Then, you issue the CA signal and begin collecting trace data with the logic state analyzer to verify that the 8089 is going through its initialization sequence correctly. You may need to repeat this step several times to isolate a problem, resetting the IOP and reissuing a channel-attention command to the IOP each time.

Sample RBF-89 Sessions

To illustrate how RBF-89 is actually used in an operational environment, the following examples are presented:

Example 1: Getting Started With RBF-89

Before using RBF-89 in an actual product debugging task, you should familiarize yourself with the system by reading this example which illustrates some simple on-line operations. These operations demonstrate some of the most commonly-used RBF-89 commands. They also let you observe some of the most often-requested displays. In this example, you will run a program named UPROG that uses local memory on a system with a remotely-configured 8089 IOP. (A source listing of this program appears in figure 3-2.) UPROG is an error-free sample program, run for demonstration purposes only. It executes for a period of time determined by a value established in the parameter block—this may indicate a finite number of seconds, or an indefinite period. This program is given a pointer into a variable-length work area. The first word of the work area gives the length of that area minus 7. The program uses all of the work area but the first six bytes as a long binary number. It initializes the number to zero, counts until all bits are set to one, and then halts. Thus, by adjusting the work area length, you can cause the program to run over a long or short duration.

UPROG is run on a hardware configuration that includes an SBC 86/12A Board and an 8089-based board. (This configuration is documented in Application Note AP89.) Attach a logic probe to the S-1 pin. When UPROG is running, the S-1 pin signal causes the logic probe to flash continuously.

```
;
; File :f1:Uprog1.a89;This is a test program which simulates a user 8089
;program being debugged by RBF. Through the M89
;standard pblk parameter block, it is given a ptr
;to a variable length work area. The first word
;of the work area gives its length minus 7.
;UPROG uses all the work area but the first 6 bytes
;as a long binary number. It initializes the number
;to 0, counts until it is all 1's, then halts.
;UPROG can thus be made to finish in a short time or
;a long time by adjusting the area length.

Uprog1Segment
Public  Uprog1, Len_Uprog1


$Include(:f1:typeq.a89)
$Eject

$Include(:f1:pblk.a89)
$Eject

Area Struc
Maxbit:ds Word
T1:ds Word
T2:ds word
CTequ $
Area    Ends

Reset:1pd GA,[PP].arg_blk;ptr to work area
lpd GB,[PP].arg_blk;ptr to work area

mov BC,[GA].Maxbit;highest bit number in count

addi GA,CT;ptr to b0
addi GB,CT+1;ptr to b1

$Include(:f1:Utilcc.a89)

wid 8,8
Xfer
movbi [GA],0;set 0 in first digit of count.
;overlapping block move will propagate it.
;Sets binary number b0.....b(maxbit) to 0's.
```

Figure 3-2. UPROG Program Listing

1018-04

```
lpd GA,[PP].arg_blk;reconstruct ptr in ga to area+0.
lpd GC,[PP].arg_blk;construct in GC ptr to b0 of count.
addi GC,CT

$Eject

;Now count the number up until it reaches 2**(maxbit+1)-1.

Count:movi IX,0;IX is bit number.  Start with 0.  Look for
;first 0 bit in number.

SeekLoop:
jzb [GC+IX], Incr;IF IX'th bit is 0, set it to 1 and set all
;lower bits to 0.

mov [GA].T1, IX;Save IX
not IX
inc IX
add IX,[GA].Maxbit;IX := Maxbit-IX
jz  IX,Done;If IX=Maxbit, b(Maxbit) is 1 and all lower
;bits are 1, so the
;full count has been attained.

mov IX,[GA].T1;Restore IX. Not Done. Still looking for lowest

;
UP_t0:Lpd GB,[PP].Arg_Blk;Now we screw around with GB a little bit.
Movp [GA].T1,GB;Here we are testing addbi with a neg arg.
Addbi GB,-2
Movp [GA].T1,GB
UP_t1:Nop

Lpd GB,[PP].Arg_Blk
Addbi GB,-1
Movp [GA].T1,GB
UP_t2:Nop

Movi GB,0F900h   ;I/O space adr F900h
Movp [GA].T1,GB
UP_t3:  Nop

Addbi GB,-2
Movp [GA].t1,GB
UP_T4:  Nop

Addi GB,0F00h            ;Enough to cause wrap-around
Movp [GA].T1,GB
UP_T5:  Nop
;

inc IX;order 0 bit in count.  Check next bit.
jmp Seekloop


Incr;Setb [GC+IX],0;Set IX'th bit to 1
jmp Incr1;Reset all lower bits

; Reset all lower-order bits to 0.

Zero;  Clr [GC+IX],0
Incr1:jz IX,Count
UP_Brk1: dec IX
UP_Brk2: jmp Zero

Done: HLt
Len_Uprog1  equ $
Uprog 1      Ends
End
```

Figure 3-2. UPROG Program Listing (Cont'd.) 1018-04

As you read through this example, put yourself in the place of the user sitting at the console and following the steps shown below:

NOTE

In these steps, you would enter only the *underlined* information—all other information is output by the system.

a. To begin, you proceed as follows:

1. Connect the ICE-86 In-Circuit Emulator to the Intellec Microcomputer Development System and to your prototype system hardware, ensuring that all connections are tight.

2. Turn on the power to the development system, console, all peripheral devices, and your prototype system.
3. Bootstrap the ISIS-II operating system.
4. Invoke the RBF-89 emulator. This initializes the ICE-86 hardware and software configuration that contains RBF-89.

NOTE

For complete details on Steps a1 through a4, above, see *ICE-86 In-Circuit Emulator Operating Instructions.*

b. Next, you map to USER status the portion of prototype system memory where the following elements are to reside:

- System Intialize Block (SIB)
- System Configuration Block (SCB)
- Channel Control Block (CCB)
- RAM buffer for RBF-89 use.

In this example, you allocate 32k bytes of memory for those elements, beginning at location 0 in user memory. To do this, you enter this ICE-86 emulator MAP command:

```
*MAP 0 LENGTH 32=USER
```

c. Next, you invoke RBF-89 by entering the RBF command. In response, RBF-89 displays a prompt for your first RBF-89 subcommand:

```
*RBF
.*        (RBF-89 subcommand prompt)
```

d. To perform the RBF-89 initialization sequence, you provide the following information through the RBF-89 subcommands shown:

1. To specify that the CCB begins at location FF00:0, you enter:

```
.*CCP=FF00:0
```

2. To indicate that the SCB starts at location FFF0:0:

```
.*SCP=FFF0:0
```

3. To denote that the physical width of the input/output bus is 16 bits:

```
.*SOC=1
```

4. To set the beginning location of the RAM area required by RBF-89 in your prototype system to Location FD00:0:

```
.*RAM=FD00:0
```

5. To tell RBF-89 that channel-attentions to Channel 1 are issued by writing to Output Port 0:

```
.*CH1=PORT 0
```

6. To tell RBF-89 that channel-attentions to Channel 2 are issued by writing to Output Port 1:

```
.*CH2=PORT 1
```

NOTE

The commands in Steps d1 through d6, above, could be entered in any order. For greater efficiency, you may wish to use macros to handle the initialization sequence in repeated applications.

7. To specify that the 8089 IOP is configured remotely, and that the logical width of the system bus is 16 bits, no RBF subcommands are required—the default values assigned by the system are appropriate.
8. To enable the 8086 to reference the SBC two-port memory with addresses pertinent to the 8086, you enter:

```
.*RELOCATE F800:0 LEN 32K TO 0
```

e. Now, you terminate the initialization sequence by entering the ENDRBF subcommand:

```
.*ENDRBF
```

In response, the following message is displayed, followed by a prompt for an RBF-89 (or ICE-86) command prompt.

```
ICE-86 RBF-89 1.0
*(RBF-89/ICE-86 Command Prompt)
```

f. Before you may run any channel program, you must establish the Parameter Block Ponter (PBP). In this case, you use the ***item-reference*** command to set PBP for Channel 1 to FB00:0:

```
*PBP=FB00:0
```

g. Because you plan to run programs on both channels, you must also set PBP for Channel 2. First, you enter a **CHANNEL** command to switch to Channel 2. Then, you enter an ***item-reference*** command to set the PBP for Channel 2 to FC00:0. And finally, you enter another **CHANNEL** command to return to Channel 1:

```
*CHANNEL=CH2
*PBP=FC00:0
*CHANNEL=CH1
```

h. Now, lyou load your IOP program, using the **LOAD** command. (This program resides in a file named UPROG1, on the diskette in disc drive :F1:.)

```
*LOAD :F1:UPROG1
```

i. To verify that the CCB is set up correctly, you decide to use the ***communication-block*** command to examine this element:

```
*CCB
FF000: +00: CCW  CH1: 07H
FF001: +01: BUSY CH1: 00H
FF002: +02: CPB1 ADR: 0000H
FF004: +04: CPB1 REL: FB00H
```

j. Now, you try to run your program by entering this **CA START** command:

```
*CA START FROM LOCAL F910:0
8089 EXECUTION BEGUN
```

k. This program continues execution indefinitely. It seems to be in a loop. Thus, to terminate its execution, you must press the ESC key. The program then halts, and the following messages appear:

```
8089 EXECUTION ABORTED RTP=0 F917FH
PROCESSING ABORTED
```

l. Next, you decide to set and enable two breakpoints in local memory (F910:D and F910:15) and execute your program in increments. To run the first increment, you enter this CA START command:

```
*CA START FROM LOCAL F910:0 TILL LOCAL F910:D, LOCAL F910:15
8089 EXECUTION BEGUN
```

The program begins execution, and continues until the breakpoint at location F910:D is encountered. At this point, execution halts and this message appears:

```
8089 EXECUTION BREAKPOINT REACHED RTP=0 F910DH
```

m. Now, you decide to display the contents of all 8089 IOP registers for further information. To do this, you enter the **89 REGISTER** command:

```
*89 REGISTER
RGA=0 F9206H RGB=0 F9200H RGC=0 F9206H RTP=0 F910DH
RIX=0003H RBC=0030H RMC=0000H RCC=C008H
```

n. Next, you decide to check the current contents of the Breakpoint Register for Channel 1:

```
*RBR
RBR=0 F910DH, 0 F9115H
```

o. Now, you continue program execution from the location at which the program is presently halted. To do this, you enter the **CA CONTINUE** command:

```
*CA CONTINUE
8089 EXECUTION BEGUN
```

The program breaks at the second breakpoint, and this message appears:

```
8089 EXECUTION BREAKPOINT REACHED RTP=0 F9115H
```

p. Again, you decide to examine all 8089 IOP registers:

```
*89 REGISTER
RGA=0 F9206H RGB=0 F9207H RGC=0 F9206H RTP=0 F9115H
RIX=0003H RBC=0030H RMC=0000H RCC=C008H
```

q. Now, you enter another **CA CONTINUE** command, this time requesting a program break at Location F910:2D.

```
*CA CONTINUE TILL LOCATION F910:2D
8089 EXECUTION BEGUN
```

When execution halts, this message appears:

```
8089 EXECUTION BREAKPOINT REACHED RTP=0 F912DH
```

r. After this break, you decide to examine the 16 bytes of local memory ranging from Location FB00:0 through Location FB00:0EH. To do this, you enter the following **BYTE** command:

```
*BYTE FB00:0 LEN 10H
BYTE FB00:0000H=2FH 91H F0H 00H 00H 00H 00H 00H 00H 20H F9H 00H FFH FFH FFH FFH
```

s. Next, you wish to disassemble some of the object code in local memory into assembly-language mnemonic form. To do this, you enter the **89 ASM** command:

```
*89 ASM F910:0 LENGTH 60
ADDR           PREFIX  MNEMONIC OPERANDS
F910:0000H                LPD     GA,[PP].07H
F910:0003H                LPD     GB,[PP].07H
F910:0006H                MOV     BC,[GA].00H
F910:0009H                ADDI    GA,0006H
F910:000DH                ADDI    GB,0007H
F910:0011H                MOVI    CC,C008H
F910:0015H                WID     8,8
F910:0017H                XFER
F910:0019H                MOVBI   [GA],00H
F910:001CH                LPD     GA,[PP].07H
F910:001FH                LPD     GC,[PP].07H
F910:0022H                ADDI    GC,0006H
F910:0026H                MOVI    IX,0000H
F910:002AH                JZB     [GC+IX],$+48H
F910:002DH                MOV     [GA].02H,IX
F910:0030H                NOT     IX
F910:0032H                INC     IX
F910:0034H                ADD     IX,[GA].00H
F910:0037H                JZ      IX,$+4AH
```

```
*89 ASM F910:0 LENGTH 60 (CONTINUED)
F910:003AH          MOV     IX,[GA].02H
F910:003DH          LPD     GB,[PP].07H
F910:0040H          MOVP    [GA].02H,GB
F910:0043H          ADDBI   GB,FEH
F910:0046H          MOVP    [GA].02H,GB
F910:0049H          NOP
F910:004BH          LPD     GB,[PP].07H
F910:004EH          ADDBI   GB,FFH
F910:0051H          MOVP    [GA].02H,GB
F910:0054H          NOP
F910:0056H          MOVI    GB,F900H
F910:005AH          MOVP    [GA].02H,GB
F910:005DH          NOP
F910:005FH          ADDBI   GB,FEH
```

t. Finally, you wish to demonstrate the simultaneous execution of two channel programs:

1. First, you run the program on the current channel, Channel 1, using the RETURN parameter of the **CA CONTINUE** command to return control to the console as soon as this command is entered:

```
*CA CONTINUE RETURN
8089 EXECUTION BEGUN
*
*
```

2. While the Channe 1 program is running, you issue a CHANNEL command to switch channels:

```
*CHANNEL CH2
```

3. Next, you display and check the CCB for the Channel 2 program:

```
*CCB
 FF008: +00: CCW  CH2: 07H
 FF009: +01: BUSY CH2: 00H
 FF00A: +02: CPB2 ADR: 0000H
 FF00C; +04: CPB2 REL: FC00H
```

4. Now, you start the Channel 2 program:

```
*CA START FROM LOC F910:0 FOREVER
8089 EXECUTION BEGUN
```

At this point, both Channel 1 and Channel 2 programs are running simultaneously.

5. To halt the Channel 2 program, you press the ESC key. When this program halts, the following messages appear:

```
8089 EXECUTION ABORTED RTP=0 F9151H
PROCESSING ABORTED
*
```

6. To halt the Channel 1 program, you:

   a. Again switch channels:

```
*CHANNEL=1
```

   b. Enter the **CA HALT** command:

```
*CA HALT
8089 EXECUTION TERMINATED RTP=0 FD002H
```

u. Now you are ready to terminate RBF-89 operation. To do so, returning to the ICE-86 emulator, you enter the EXIT command as follows:

```
*EXIT RBF
```

Example 2; Debugging a Simple Memory-Transfer Program

For your next debugging session, you plan to test a small program named *COPMIN* that copies a block of memory from system (local) memory to remote memory, or vice-versa. This program runs on a configuration that includes an SBC 86/12A Board, with 32k dual-port memory (mapped from Location 0 through 32k for the 8086, and from Locations F8000 through FFFFF for the remote 8089). This configuration also includes an 8089 prototype board that interfaces with the system in remote mode, and contains 6k of RAM (locations 0 through 6k). The 8089 bus is 16 bits wide; channel attentions to Channel 1 are generated by writing output to Port 80; channel attentions to Channel 2 are generated by writing output to Port 81.

Specifically, COPMIN transfers <src-len> byes of data from the partition beginning at Location <src-adr> to the partition beginning at Location <dst-adr>. (A source listing appears in figure 3-3.) The Parameter Block Pointer (PBP) is stored at Location F800:2000. Within the PBP, COPMIN picks up another pointer at Location F800:2007; this points to a block of parameters at Location F800:0000. The parameters in this block are:

- Source address (<src-adr>), a 20-bit pointer set to local memory address F800:0050 (which translates to 50H,40H,F0H in 8089 pointer format).
- Destination address (<dst-adr>), a 20-bit pointer set to remote memory location 0 (which translates to 00H,00H,08H in 8089 pointer format).
- Byte count (<src-len>), a 2-byte counter set to 32 bytes (20H,00H).

The data stored at <src-sdr> is:

```
0000:0050H=01H FFH 02H FEH 03H FDH 04H FCH 05H FBH 06H FAH 07H F9H 08H F8H
0000:0060H=09H F7H 0AH F6H 0BH F5H 0CH F4H 0DH F3H 0EH F2H 0FH F1H 10H F0H
```

In this example, you will execute the program twice: first, to move the data from the SBC 86/12 board to prototype memory, and second, to return the data to the SBC 86/12 board. Following the transfers, you may compare the transferred data to the source data to ensure that the transfer was successful.

a. After preparing and invoking the ICE-86 emulator (as in Example 1), you enter the **MAP** command to allocate 32k bytes of memory, beginning at Location 0, for your user program:

```
*MAP 0 LENGTH 32 = USER
```

b. Next, you use the ICE-86 emulator's **CLOCK** command to specify that the SBC 86/12A Board will use an externally-supplied clock. (This is necessary to let your program read the memory on this board.)

```
*CLOCK=EXTERNAL
```

c. Finally, you use the ICE-86 emulator's **DISABLE** command to disable the READY pin signal, normally used to acknwledge completion of data transfer from addresses memory.

```
*DISABLE READY
```

d. To initialize RBF-89 in this and subsequent sessions, you retrieve the following macro, using the ICE-86 emulator's **INCLUDE** command:

```
*INCLUDE :F1:RBFMAC                   ; Retrieves macro on disk
                                      ; file :F1:RBFMAC.
*DEFINE MACRO INIT                    ; Begins macro definition.
 *RBF                                 ; Invokes RBF-89.
 .*SCP=F800:1000                      ; Begins SCB at F800:1000.
 .*CCP=F800:1010                      ; Starts CCP at F800:1010.
 .*SOC=1                              ; Sets I|O bus width to 16
                                      ; bits.
 .*RAM=F800:5000                      ; Allocates RAM for RBF use.
 .*CH1=PORT 80                        ; Specifies CA mechanism for
                                      ; CH1.
 .*RELOCATE F800:0 LENGTH 32K TO 0    ; Specifies relative
                                      ; addresses for dual-port
                                      ; memory.
 .*ENDRBF                             ; Ends RBF initialization
                                      ; sequence.
*EM                                   Ends macro definition.
*
```

```

                 ;              C O P M I N
                 ;

                 ;This is the copy memory indirect
                 ;channel utility program.  It fits in a tiny
                 ;corner of the RBF-buffer.
                 .
                 ;8089 channel utility program COPMIN.  Copies memory to memory indirect
                 ; from <src.adr> to <dest.adr> for <byte.count> bytes.




                 copmin_ablk        STRUC






0000             src_adr:          ds MVPTR          ;Pointer to first byte of data to be copied.
0003             dst_adr:          ds MVPTR          ;Ptr to first byte of destination.
0006             src_len:          ds WORD           ;How many bytes to copy.
0008             copmin_ablk       ENDS





                 ;This is the first executable instruction of the COPMIN channel program.

0124             copmin:

0124    438B 07          lpd GC,[PP].arg_blk           ;load GC with pointer
                                                       ;to the command block. The
                                                       ;calling program
                                                       ;has prepared the block giving start
                                                       ;address, dest address, and byte count.

```

Figure 3-3. COPMIN Program Listing 1018-05

```
0127   D130 08C0 =       movi CC,0C008H            ;load Channel control register with
                 =                                 ;fixed data from immediate field of
                 =                                 ;instruction.  This bit pattern says:
                 =
                 =                                 ;   F = 11   block to block
                 =                                 ;  TR =  0   no translate
                 =                                 ; SYN = 00   no synch - ignore DRQ
                 =                                 ;   S =  0   src GA... dest GB
                 =                                 ;   L =  0   dont lock for duration DMA
                 =                                 ;   C =  0   use low priority for Ch Pgm
                 =                                 ;  TS =  0   not a single transfer DMA
                 =                                 ;  TX = 00   no external terminate
                 =                                 ; TBC = 01   use byte count terminate
                 =                                 ; TSH = 000  dont termainte on match

012B   038E 00              movp GA,[GC].src_adr   ;get source address.
012E   238E 03              movp GB,[GC].dst_adr   ;get dest address
0131   8000                 wid  8,8               ;set 8 bit logical width for DMA.
                                                   ;This is probably safest.

0133   6000                  XFER                  ;Perform following instruction, then
                                                   ;enter DMA transfer mode.

0135   6382 06               mov BC,[GC].src_len   ; Remember that this instruction is
                                                   ; executed before the XFER!
                                                   ; This instruction loads the byte count
                                                   ; for the DMA transfer from the command
                                                   ; block. It is done after the XFER in
                                                   ; favor of minimizing size of copmin,
                                                   ; admittedly at the expense of main-
                                                   ; tainability.


0138   2048                 hlt
0016               len_copmin     equ $ - copmin

                   ;  $Title('S E T M I M    Channel Utility')
```

Figure 3-3. COPMIN Program Listing (Cont'd.) 1018-05

e. Now, you invoke and initialize RBF-89 by calling the above macro:

```
*:INIT                                 ; Calls macro INIT.
*RBF                                   ; Invokes RBF-89.
.*SCP=F800:1000                        ; Begins SCB at F800:1000.
.*CCP=F800:1010                        ; Starts CCP at F800:1010.
.*SOC=1                                ; Sets I|O bus width to 16
                                       ; bits.
.*RAM=F800:5000                        ; Allocates RAM for RBF use.
.*CH1 PORT 80                          ; Specifies CA mechanism for
                                       ; CH1.
.*RELOCATE F800:0 LENGTH 32K TO 0      ; Specifies relative
                                       ; addresses for dual-port
                                       ; memory.
.*ENDRBF                               ; Ends RBF initialization
                                       ; sequence.
*EM                                    ; Ends macro definition.
*
```

f. Next, you use the ICE-86 emulator's LOAD command to load the COPMIN program into local memory.

```
*LOAD :F1:COPMIN
```

g. You set the Parameter Block Pointer (PBP), as follows:

```
*PBP=F800:2000
```

h. Now, you check the pointers at Locations F800:2000 and F800:2007 for validity:

```
*POI 2000              ; Starts COPMIN program.
POI 0000:2000H=F800:0000H
*POI 2007              ; Starts PBP
POI 0000:2007H=F800:0040H
```

You see that these pointers are set to the correct addresses.

i. Next, you check the parameters stored in the 8-byte area beginning at Location 0040:

```
*BYTE 0040 LENGTH 8
BYTE 0000:0040=50H A2H F0H 00H 00H 08H 20H 00H
```

This display shows that <src-adr> is set to 50H, A2H, F0H; <dst-adr> is set to 00H, 00H, 08H; and <src-len> is set to 20H, 00H. (In the third byte of each pointer, the lower nibble is 0 for local memory and 8 for remote memory.)

j. Now, you use the 89 ASM command to disassemble and check the COPMIN program:

```
*89 ASM  0  LEN 15
ADDR              PREFIX    MNENONIC  OPERANDS
0000:0000H                    LPD     GC,[PP].07H
0000:0003H                    MOVI    CC,C008H
0000:0007H                    MOVP    GA,[GC].00H
0000:000AH                    MOVP    GB,[GC].03H
0000:000DH                    WID     8,16
0000:000FH                    XFER
0000:0011H                    MOV     BC,[GC].06H
0000:0014H                    HLT
```

The display indicates the correct source code.

k. You run COPMIN, using the **CA START** command, to transfer the data from local to remote memory.

```
*CA START FROM LOCAL F800:000 FOREVER
8089 EXECUTION BEGUN
8089 EXECUTION TERMINATED RPT=0 F8016H
```

The program runs to completion, halting as expected at Location F800:2216.

l. You now wish to run COPMIN again, transferring the data back to local memory. Before you do this, however, you must respecify the contents of the two pointers beginning at Location 0040.

```
*BYTE 0040=10,00,08,70,80,F0
```

m. Now, you run COPMIN:

```
*CA START FROM LOCAL F800:0000 FOREVER
8089 EXECUTION BEGUN
8089 EXECUTION TERMINATED RTP=0 F8016H
```

n. You attempt to verify that the data was transferred correctly by COPMIN, by displaying this data:

```
*BYTE 0070 LENGTH 20
0000:0070:09H F7H 0AH F6H 0BH F5H 0CH F4H 0DH F3H 0EH F2H 0FH F7H 10H F0H
0000:0080:FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH FFH
```

By comparing this data with that stored in the 20-byte area beginning at Location 0050, you see that the data transfer was not completed successfully. You suspect that you might have respecified the pointers incorrectly.

o. To verify your suspecion, you:

1. Set a breakpoint at the XFER instruction at Location 000FH in COPMIN that performs the data transfer:

   ```
   *RBR=LOCAL F800:000F
   ```

2. Run COPMIN again:

   ```
   *CA START FROM LOCAL F800:0000 TIL RBR
   8089 EXECUTION BEGUN
   8089 EXECUTION BREAKPOINT REACHED  RTP=0  F800FH
   ```

3. Display the contents of all 8089 registers:

   ```
   *89 REGISTER
   RGA=1 00010H RGB=0 F8070H RGC=0 F80C0H RTP=0 F800FH
   RIX=0000H RBC=0000H RMC=0000H RCC=C008H
   ```

   In this display, you verify your suspicion by noting an erroneous pointer to <src-adr> in the RGA register.

4. You correct this pointer by entering:

   ```
   *RGA=0
   ```

5. You verify the new RGA setting by entering:

   ```
   *RGA
   RGA=1 00000H
   ```

p. Next, continue the program to its conclusion by entering:

   ```
   *CA CONTINUE
   8089 EXECUTION BEGUN
   8089 EXECUTION TERMINATE TP=0  F8016H
   ```

q. Again, you check the data beginning at Location 0070:

   ```
   *BYTE 0070 LENGTH 20
   ```

   This time, the display shows that the data was transferred correctly.

r. Finally, with your debugging session concluded, you terminate RBF-89 execution:

   ```
   *EXIT RBF
   ```

APPENDIX A
RBF-89 ERROR CONDITIONS AND RECOVERY

During RBF-89 command processing, three types of error may occur:

- *Syntax error in command input,* detected before command execution begins. In response, RBF-89 aborts the command at once, displays an error message, and returns control to the console (prompting you for a new command entry).

- *Fatal run-time error,* detected after command execution begins. RBF-89 aborts command execution, displays an error message, and returns control to the console. If the error occurs in a compound command group such as a macro, RBF-89 aborts all commands in the group. Some fatal run-time errors leave the system in an uncertain state. For instance, when an error occurs during a file-loading or memory-write operation, the information written after this error is incorrect.

- *Non-fatal run-time error,* detected during command execution. RBF-89 displays a warning message but command execution continues even though spurious results might occur.

Because the system processes commands on a line-by-line basis, it does not report any error until the entire line that contains the error is entered.

RBF-89 error messages contain the following elements, in the order listed:

- The notation *ERR* to indicate a syntax or fatal run-time error, or *WARN* to denote a non-fatal run-time error.

- A hexadecimal *error number* that identifies the particular type of error.

- *Text* that describes the error.

An example of an error message is:

```
ERR 7F:CHANNEL BUSY
ERR NOTATION
TEXT
ERROR NUMBER
```

A list of error messages, their meaning, and recommended corrective action, appears in table A-1.

NOTE

This table presents only messages unique to RBF-89. Messages generated by both RBF-89 and the ICE-86 emulator, or by the ICE-86 emulator only, appear in *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users.*

Table A-1. Error Messages, Causes, and Corrective Action

ERR/Warn	Error	Message Text	Probable Cause and Recovery Procedure
ERR	6F	INVALID RELOCATION ASGN	In RELOCATE subcommand, you specified block that does not fall within a paragraph boundary. Correction: Re-enter subcommand with correct block specification.
ERR	70	RESERVE PAGE CONFLICT	In RESERVE subcommand, you specified for MONITOR an addres space already allocated to your user program. Correction: Re-enter subcommand with new address space specification.
ERR	71	INVALID RESERVED PAGE	In RESERVE subcommand, you specified a page that does not lie within range bounded by Locations 0 through 1023. Correction: Re-enter subcommand with correct page specification.
ERR	72	RBF NOT INVOKED YET	You entered an RBF-89 command, but did not yet invoke RBF-89 in this session. Correction: Invoke and initialize RBF-89; then re-enter desired command.
ERR	73	BREAKPOINT AFTER XFER	You specified an RBF-89 breakpoint immediately following an XFER instruction in your user program. Correction: Re-specify the breakpoint at a different location.
ERR	74	NO ADDR SPACE AVAILABLE	You entered a RESERVE subcommand but no contiguous 2k-byte segment is available for MONITOR. Correction: Map additional space to GUARDED status to allow room for MONITOR, and re-enter subcommand.
ERR	75	MONITOR ERROR	RBF-89 did not operate properly. Correction: Terminate and re-invoke RBF-89. If this does not work, call Intel for assistance.
ERR	76	INSUFICIENT PARAMETERS	You included fewer parameters than the number required in your last command. Correction: Check command syntax described in Chapter 2 and re-enter command.
ERR	77	INVALID BUS WIDTH	In WIDTH subcommand, you specified a value other than 8 or 16T. Correction: Re-enter subcommand with proper value.
ERR	79	CH2 ATTN ASSGN REQUIRED	You spelcified the 8089 as a slave in an IOP subcommand, but did not issue CH2 subcommand. Correction: re-enter CH2 subcommand, followed by IOP subcommand.

Table A-1. Error Messages, Causes, and Corrective Action (Cont'd.)

ERR/Warn	Error	Message Text	Probable Cause and Recovery Procedure
ERR	7A	INVALID SOC VALUE	In an SOC subcommand, you specified an SOC value greater than or equal to 4. Correction: Re-enter subcommand with correct value.
ERR	7C	CANNOT SWITCH CHANNEL	You attempt to change current input/output channel without specifying CH2 subcommand. Correction: Enter CH2 subcommand, then CHANNEL=2 command.
ERR	7D	ICE MEM NOT AVAILABLE	You attempted to map user application program into ICE-86 emulator memory during RBF-89 session. (While RBF-89 session is in progress, emulator memory is dedicated to RBF-89 Control Program.) Correction: Map your program into application system memory.
ERR	7F	CHANNEL BUSY	Channel is currently busy, preventing acceptance of any RBF-89 command but CA HALT. Correction: Wait until channel not busy or enter CA HALT, then enter desired command.
ERR	B4	POTENTIAL BUS LOCKOUT	During channel-initialization BUSY flag was not reset within 500-millisecond time-period. Correction: Verify that correct address and value were given in CHANNEL subcommand. If this assignment was correct, check hardware for proper connection. Then re-initialize channel.
WARN	CD	BREAKPOINT CHANGED	RBF-89 tried to remove breakpoint that no longer exists because it was changed by user program. Correction: None necessary.
WARN	CE	TOO MANY BREAKPOINTS	You specified more than six breakpoints in a command. Correction: None necessary; RBF-89 uses only first six breakpoints you specified.
WARN	CF	DUPLICATE BREAKPOINTS	You specified two or more breakpoints at same location. Correction: None necessary; RBF-89 uses only one of these breakpoints.

APPENDIX B
RBF-89 KEYWORDS

The keywords used in RBF-89, and their single-letter synoymns (if applicable) and meanings appear in table B-1. Because any keyword can be abbreviated by entering its first three letters only, these letters are capitalized in this table.

Table B-1. RBF-89 Keywords, Synonyms, and Meanings

Keyword	Synonym	Meaning
ASM		Request to display memory contents in disassembled form.
ATTention		Request to define channel-attention mechanism.
CA		Channel-attention command.
CCB		Channel-Control Block.
CCP		Channel Control Pointer.
CCW		Channel Control Word.
CH1		Channel 1.
CH2		Channel 2.
CHAnnel		Command to change current channel.
CONtinue		Command to resume channel program execution.
ENAble		Command to enable element denoted by following keyword.
ENDrbf		Subcommand to terminate subcommand entry sequence.
EXIt		Command to terminate RBF-89 and/or ICE-86 emulator operation.
FORever		Request to continue channel-program execution indefinitely.
FROm	F	Starting-address specifier for channel-program execution.
HALT		Command to halt program execution.
IOP		Subcommand to specify 8089 master/slave status.
LENgth		Memory-partition length specifier.
LOAd		Command to load channel program.
MASter		Master status specifier.

Table B-1. RBF-89 Keywords, Synonyms, and Meanings (Cont'd.)

Keyword	Synonym	Meaning
NOCode		Request to omit loading or saving code in LOAD or SAVE command.
NOSymbol		Request to omit loading or saving symbols in LOAD or SAVE command.
PAGe		Subcommand to specify base for 20-bit register references.
PBP		Parameter Block Pointer.
PS		Program Status Word.
RAM		Command to indicate 200 bytes of RAM available to RBF-89 in application system.
RBC		Byte Count (BC) Register.
RBF		Command to invoke RBF-89.
RBR		Reference to Breakpoint Register for current channel.
RBYte		Remote byte specifier.
RCC		Channel-Control Register.
REGister	R	Command to display all 8089 Registers if preceding 89.
RELocate		Command to relocate memory references.
REMote		Remote configuration specifier.
REServe		Command to reserve memory for RBF-89 MONITOR.
RESet		Command to reset element denoted by following keyword.
RETurn		Request to return control to console immediately after starting channel program.
RGA		General-Purpose Register A.
RGB		General-Purpose Register B.
RGC		General-Purpose Register C.
RIX		8089 Index Register.
RMC		Mask/Compare Register.

Table B-1. RBF-89 Keywords, Synonyms, and Meanings (Cont'd.)

Keyword	Synonym	Meaning
RTP		Task Pointer.
RWOrd		Remote word specifier.
SAVe		Command to save program on diskette.
SCB		System Configuration Block.
SCP		System Configuration Pointer.
SIB		System Initialization Block.
SLAve		Slave status specifier.
SOC		System Operation Command Word.
STArt		Channel-attention command to begin channel program.
TGA		Tag bit of GA Register.
TGB		Tag bit of GB Register.
TGC		Tag bit of GC Register.
TIL/	T	Request to terminate program when breakpoint encountered.
UPDate		Command to update software copy of 8089 registers.
WIDth.		Subcommand to specify logical width of system bus.
89		8089 register or memory specifier.

APPENDIX C RBF-89 REFERENCE SPECIFICATION SUMMARY

Command Syntax Notation

In the command syntax description that follow, this symbolic notation is used:

Notation	Meaning
UPPERCASE LETTERS	Keyword that must be entered exactly as shown (or as three-letter abbreviation).
lowercase italics	Class name for set or class of token for which you select specific identifier (variable name).
. . .	Previous entry may be repeated indefinitely.
{ }	One (and only one) of the enclosed elements MUST be selected.
{ } . . .	AT LEAST one of the enclosed elements must be selected.
[ ]	All enclosed elements are optional, but only ONE may be entered.
[ ] . . .	All enclosed elements are optional, but MORE THAN ONE may be entered.

Command Syntax

ICE-86 Emulator Command to Invoke RBF-89

RBF CR
subcommand CR
[*subcommand* CR]
.
.
.
ENDRBF CR

Example:

```
*RBF
.*RAM=FFC0:0
.*CH1 = BYTE FF00:0
.*WIDTH=16T
.*SCP=FFF0:0
.*SOC=1
.*CCP=FFF0:10
.*IOP=MASTER
.*ENDRBF
```

RBF-89 Initialization Subcommands

CCP = *expression*

Example: `*CCP = 8000:10`

$\left\{\begin{array}{l}\textbf{CH1}\\ \textbf{CH2}\end{array}\right\} = \left\{\begin{array}{l}\text{PORT}\\ \text{WPORT}\\ \text{BYTE}\\ \text{WORD}\end{array}\right\}$ *expression1* [VALUE *expression2*]

Example:
```
*CH1 = BYTE.MEMSYM+10 VALUE 3
*CH2 = WPORT 2
```

IOP = $\left\{\begin{array}{l}\text{MASTER}\\ \text{SLAVE}\end{array}\right\}$

Examples:
```
*IOP=MASTER
*IOP=SLA
```

PAGE = *expression*

Example: `*PAGE=2000`

RAM = *address*

Examples:
```
*RAM=9000
*RAM=A000
```

RELOCATE *partition* TO *address*

Example: `*RELOCATE F800:0 LEN 32K TO 0`

RESERVE *expression*

Examples:
```
*RESERVE 10
*RES 8
```

SCP = *expression*

Example: `*SCP=8000:0`

SOC = *expression*

Example: `*SOC=1`

WIDTH = *expression*

Examples:
```
*WIDTH=.BUSWD
*WID=16T
```

RBF-89 Commands

CA CONTINUE $\left[\begin{array}{l}\text{FOREVER}\\ \text{TILL } \textit{breaklist}\\ \text{TILL RBR}\end{array}\right]$ [RETURN]

Examples:
```
*CA CONTINUE
*CA CONTINUE TILL LOCAL 1030, LOCAL 1040
*CA CONTINUE FOREVER RETURN
```

CA HALT

Example:
```
*CA HALT
```

CA START FROM $\left\{\begin{array}{l}\text{LOCAL}\\ \text{REMOTE}\end{array}\right\}$ *expression* $\left[\begin{array}{l}\text{FOREVER}\\ \text{TILL } \textit{breaklist}\\ \text{TILL RBR}\end{array}\right]$ [RETURN]

Examples:
```
*CA START FROM LOCAL 1000H
*CA START FROM LOCAL 1000H TILL LOCAL 1010, LOCAL 1020
*CA START FROM LOCAL 1000H TILL RBR
*CA START FROM REMOTE 1000H TILL RBR RETURN
```

CHANNEL $\left[= \left\{\begin{array}{l}\text{CH1}\\ \text{CH2}\end{array}\right\}\right]$

Examples:
```
*CHANNEL
*CHANNEL=CH1
*CHA=CH2
```

communication-block

Examples:
```
*SIB
*SCB
```

ENABLE [BOTH] **RBR**

Examples:
```
*ENABLE RBR
*ENA BOTH RBR
```

EXIT RBF

Example:
```
*EXIT RBF
```

item-reference $\left\{\begin{array}{l}[\textit{,item-reference}]\\ \textit{=expression}\end{array}\right\}$

Examples:
```
*RGA
*RGA,RGB,RGC
*RBYTE 1083H
*RGA=20:40
*TGA=1
```

LOAD [:*drive*:] *filename* $\left[\begin{array}{l}\text{LOCAL}\\ \text{REMOTE}\end{array}\right] \left[\begin{array}{l}\text{NOCODE}\\ \text{NOSYMBOL}\end{array}\right] \ldots$

Examples:
```
*LOAD :F1:IOPROG REMOTE
*LOA MYPROG REM NOSYMBOL
```

RBR [=*breaklist*]

Examples:
```
*RBR
*RBR = LOCAL 1000H
```

$\left\{\begin{array}{l}\textbf{RBYTE}\\ \textbf{RWORD}\end{array}\right\}$ *partition* [=*expression*[,*expression*] . . .]

Examples:
```
*RBYTE 1052H
*RWORD 1200H TO 1260H
*RBYTE 1000H LEN 4
*RBYTE 1083H=00H,40H,18H
*RWORD 1000H LEN 20 = 00H
```

RESET RBF

Examples:
```
*RESET RBF
*RES RBF
```

RESET [BOTH] **RBR**

Examples:
```
*RESET RBR
*RES BOTH RBR
```

SAVE [:drive:] filename $\left[\begin{array}{l}\text{LOCAL}\\ \text{REMOTE}\end{array}\right]$ $\left[\begin{array}{l}\text{NOSYMBOL}\\ \left\{\begin{array}{l}\text{NOCODE}\\ \textit{partition}\end{array}\right\}\end{array}\right]$

Examples:
```
*SAVE MYPROG REMOTE NOSYMBOL .START TO .START+50H
*SAVE YURPRG REM .RESTART LEN 200H
```

UPDATE

Example:
```
*UPDATE
```

89 ASM $\left\{\begin{array}{l}\text{LOCAL}\\ \text{REMOTE}\end{array}\right\}$ *partition*

Examples:
```
*89 ASM 1000H TO 1050H
*89 ASM REMOTE 1200H LEN 50H
```

89 REGISTER

Example:
```
*89 REGISTER
```

APPENDIX D 8089 INPUT/OUTPUT PROCESSOR INSTRUCTION SET

This appendix lists the instruction set for the Intel 8089 Input/Output Processor. In this list, all instructions appear in alphabetical order according to their names (ASM-89 mnemonic opcodes). For every combination operand types, the list shows the instruction's execution time and length in bytes, plus a coding example. The operand identifiers in the list are explained in table D-1. A key to the operand types in the list appears in table D-2. The Instruction Set List itself appears in table D-3.

In table D-3, the instruction timing figures show the number of clock periods required to execute the instruction with the given combination of operands. At 5 MHz, one clock period is 200 ns; at 8 MHz, a clock period is 125 ns. Two timings are provided when an instruction operates on a memory word. The first (lower) figure indicates execution time when the word is aligned on an even address and is accessed over a 16-bit bus. The second figure is for odd-addressed words on 16-bit buses and any word accessed via an 8-bit bus.

Instruction fetch time is shown in table D-4 and should be added to the execution times shown in table D-3 to determine how long a sequence of instructions will take to run. External delays such as bus arbitration, wait states, and activity on the other channel will increase the elapsed time over the figures shown in tables D-3 and D-4. These delays are application-dependent.

Table D-1. Key to ASM-89 Operand Identifiers

Identifier	Used In	Explanation
destination	data transfer, arithmetic, bit manipulation	A register or memory location that may contain data operated on by the instruction, and which receives (is replaced by) the result of the operation.
source	data transfer, arithmetic, bit manipulation	A register, memory localtion, or immediate value that is used in the operation, but is not altered by the instruction.
target	program transfer	Location to which control is to be transferred.
TPsave	program transfer	A 24-bit memory location where the address of the next sequential instruction is to be saved.
bit-select	bit manipulation	Specification of a bit location within a byte; 0=least-significant (rightmost) bit, 7=most-significant (leftmost) bit.
set-value	TSL	Value to which destination is set if it is found 0.
source-width	WID	Logical width of source bus.
dest-width	WID	Logical width of destination bus.

Table D-2. Key to Operand Types

Identifier	Explanation
(no operands)	No operands are written
register	Any general register
ptr-reg	A pointer register
immed8	A constant in the range 0-FFH
immed16	A constant in the range 0-FFFFH
mem8	An 8-bit memory location (byte)
mem16	A 16-bit memory location (word)
mem24	A 24-bit memory location (physical address pointer)
mem32	A 32-bit memory location (doubleword pointer)
label	A label within −32,768 to +32,767 bytes of the end of the instruction
short-label	A label within −128 to +127 bytes of the end of the instruction
0-7	A constant in the range: 0-7
8/16	The constant 8 or the constant 16

Table D-3. Instruction Set Reference Data

ADD destination, source		Add Word Variable	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem16	11/15	2-3	ADD BC, [GA].LENGTH
mem16, register	16/26	2-3	ADD [GB], GC

ADDB destination, source		Add Byte Variable	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem8	11	2-3	ADDB GC, [GA].N CHARS
mem8, register	16	2-3	ADDB [PP].ERRORS, MC

ADDBI destination, source		Add Byte Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed8	3	3	ADDBI MC,10
mem8, immed8	16	3-4	ADDBI [PP+IX+].RECORDS, 2CH

ADDI destination, source		Add Word Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed16	3	4	ADDI GB, 0C25BH
mem16, immed16	16/26	4-5	ADDI [GB].POINTER, 5899

AND destination, source		Logical AND Word Variable	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem16	11/15	2-3	AND MC, [GA].FLAG WORD
mem16, register	16/26	2-3	AND [GC].STATUS, BC

ANDB destination, source		Logical AND Byte Variable	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem8	11	2-3	AND BC, [GC]
mem8, register	16	2-3	AND [GA+IX].RESULT, GA

Table D-3. Instruction Set Reference Data (Cont'd.)

ANDBI destination, source — Logical AND Byte Immediate

Operands	Clocks	Bytes	Coding Example
register, immed8	3	3	GA, 01100000B
mem8, immed8	16	3-4	[GC+IX], 2CH

ANDI destination, source — Logical AND Word Immediate

Operands	Clocks	Bytes	Coding Example
register, immed16	3	4	IX, 0H
mem16, immed16	16/26	4-5	[GB+IX].TAB, 40H

CALL TPsave, target — Call

Operands	Clocks	Bytes	Coding Example
mem24, label	17/23	3-5	CALL [GC+IX].SAVE, GET_NEXT

CLR destination, bit select — Clear Bit To Zero

Operands	Clocks	Bytes	Coding Example
mem8, 0-7	16	2-3	CLR [GA], 3

DEC destination — Decrement Word By 1

Operands	Clocks	Bytes	Coding Example
register	3	2	
mem16	16/26	2-3	DEC [PP].RETRY

DECB destination — Decrement Byte By 1

Operands	Clocks	Bytes	Coding Example
mem8	16	2-3	DECB [GA+IX+].TAB

HLT (no operands) — Halt Channel Program

Operands	Clocks	Bytes	Coding Example
(no operands)	11	2	HLT

INC destination — Increment Word by 1

Operands	Clocks	Bytes	Coding Example
register	3	2	INC GA
mem16	16/26	2-3	INC [GA].COUNT

INCB destination — Increment Byte by 1

Operands	Clocks	Bytes	Coding Example
mem8	16	2-3	INCB [GB] POINTER

JBT source, bit-select, target — Jump if Bit True (1)

Operands	Clocks	Bytes	Coding Example
mem8, 0-7, label	14	3-5	JBT [GA].RESULT REG, 3, DATA VALID

Table D-3. Instruction Set Reference Data (Cont'd.)

JMCE source, target		Jump if Masked Compare Equal	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	14	3-5	JMCE [GB].FLAG, STOP_SEARCH

JMCNE source, target		Jump if Masked Compare Not Equal	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	14	3-5	JMCNE [GB+IX], NEXT_ITEM

JMP target		Jump Unconditionally	
Operands	**Clocks**	**Bytes**	**Coding Example**
label	3	3-4	JMP READ_SECTOR

JNBT source, bit-select, target		Jump if Bit Not True (0)	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, 0-7, label	14	3-5	JNBT [GC], 3, RE_READ

JNZ source, target		Jump if Word Not Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, label	5	3-4	JNZ BC, WRITE_LINE
mem16, label	12/16	3-5	JNZ [PP].NUM_CHARS, PUT_BYTE

JNZB source, target		Jump if Byte Not Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	12	3-5	JNZB [GA], MORE_DATA

JZ source, target		Jump if Word is Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, label	5	3-4	JZ BC, NEXT_LINE
mem16, label	12/16	3-5	JZ [GC+IX].INDEX, BUF_EMPTY

JZB source, target		Jump if Byte Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	12	3-5	JZB [PP].LINES_LEFT, RETURN

LCALL TPsave, target		Long Call	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem24, label	17/23	4-5	LCALL [GC].RETURN_SAVE, INIT_8279

LJBT source, bit-select, target		Long Jump if Bit True (1)	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, 0-7, label	14	4-5	LJBT [GA].RESULT, 1, DATA_OK

LJMCE source, target		Long jump if Masked Compare Equal	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	14	4-5	LJMCE [GB], BYTE_FOUND

Table D-3. Instruction Set Reference Data (Cont'd.)

LJMCNE source, target		Long jump if Masked Compare Not Equal	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	14	4-5	LJMCNE [GC+IX+], SCAN_NEXT

LJMP target		Long Jump Unconditional	
Operands	**Clocks**	**Bytes**	**Coding Example**
label	3	4	LJMP GET_CURSOR

LJNBT source, bit-select, target		Long Jump if Bit Not True (0)	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, 0-7, label	14	4-5	LJNBT [GC], 6, CRCC_ERROR

LJNZ source, target		Long Jump if Word Not Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, label	5	4	LJNZ BC, PARTIAL_XMIT
mem16, label	12/16	4-5	LJNZ [GA+IX].N_LEFT, PUT_DATA

LJNZB source, target		Long Jump if Byte Not Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	12	4-5	LJNZB [GB+IX+].ITEM, BUMP_COUNT

LJZ source, target		Long Jump if Word Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, label	5	4	LJZ IX, FIRST_ELEMENT
mem16, label	12/16	4-5	LJZ [GB].XMIT_COUNT, NO_DATA

LJZB source, target		Long Jump if Byte Zero	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, label	12	4-5	LJZB [GA], RETURN_LINE

LPD destination, source		Load Pointer With Doubleword Variable	
Operands	**Clocks**	**Bytes**	**Coding Example**
ptr-reg, mem32	20/28*	2-3	LPD GA, [PP].BUF_START

*20 clocks if operand is on even address; 28 if on odd address

LPDI destination, source		Load Pointer With Doubleword Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
ptr-reg, immed32	12/16*	6	LPDI GB, DISK_ADDRESS

*12 clocks if instruction is on even address; 16 if on odd address

MOV destination, source		Move Word	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem16	8/12	2-3	MOV IX, [GC]
mem16, register	10/16	2-3	MOV [GA].COUNT, BC
mem16, mem16	18/28	4-6	MOV [GA].READING, [GB]

Table D-3. Instruction Set Reference Data (Cont'd.)

MOVB destination, source		Move Byte	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem8 mem8, register mem8, mem8	8 10 18	2-3 2-3 4-6	MOVB BC, [PP].TRAN_COUNT MOVB [PP].RETURN_CODE, GC MOVB [GB+IX+], [GA+IX+]

MOVBI destination, source		Move Byte Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed8 mem8, immed8	3 12	3 3-4	MOVBI MC, 'A' MOVBI [PP].RESULT, 0

MOVI destination, source		Move Word Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed16 mem16, immed16	3 12/18	4 4-5	MOVI BC, 0 MOVI [GB], 0FFFFH

MOVP destination, source		Move Pointer	
Operands	**Clocks**	**Bytes**	**Coding Example**
ptr-reg, mem24 mem24, ptr-reg	19/27* 16/22*	2-3 2-3	MOVP TP, [GC+IX] MOVP [GB].SAVE_ADDR, GC

*First figure is for operand on even address; second is for odd-addressed operand.

NOP (no operands)		No Operation	
Operands	**Clocks**	**Bytes**	**Coding Example**
(no operands)	4	2	NOP

NOT destination/destination, source		Logical NOT Word	
Operands	**Clocks**	**Bytes**	**Coding Example**
register mem16 register, mem16	3 16/26 11/15	2 2-3 2-3	NOT MC NOT [GA].PARM NOT BC, [GA+IX].LINES_LEFT

NOTB destination/destination, source		Logical NOT Byte	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8 register, mem8	16 11	2-3 2-3	NOTB [GA].PARM_REG NOTB IX, [GB].STATUS

OR destination, source		Logical OR Word	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem16 mem16, register	11/15 16/26	2-3 2-3	OR MC, [GC].MASK OR [GC], BC

ORB destination, source		Logical OR Byte	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, mem8 mem8, register	11 16	2-3 2-3	ORB IX, [PP].POINTER ORB [GA+IX+], GB

ORBI destination, source		Logical OR Byte Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed8 mem8, immed8	3 16	3 3-4	ORBI IX, 00010001B ORBI [GB].COMMAND, 0CH

Table D-3. Instruction Set Reference Data (Cont'd.)

ORI destination, source		Logical OR Word Immediate	
Operands	**Clocks**	**Bytes**	**Coding Example**
register, immed16 mem16,immed16	3 16/26	4 4-5	ORI MC, 0FF0DH ORI [GA], 1000H

SETB destination, bit-select		Set Bit to 1	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, 0-7	16	2-3	SETB [GA].PARM_REG, 2

SINTR (no operands)		Set Interrupt Service Bit	
Operands	**Clocks**	**Bytes**	**Coding Example**
(no operands)	4	2	SINTR

TSL destination, set-value, target		Test and Set While Locked	
Operands	**Clocks**	**Bytes**	**Coding Example**
mem8, immed8, short-label	14/16*	4-5	TSL [GA].FLAG, 0FFH, NOT_READY

*14 clocks if destination ≠ 0; 16 clocks if destination = 0

WID source-width, dest-width		Set Logical Bus Widths	
Operands	**Clocks**	**Bytes**	**Coding Example**
8/16, 8/16	4	2	WID 8, 8

XFER (no operands)		Enter DMA Transfer Mode After Next Instruction	
Operands	**Clocks**	**Bytes**	**Coding Example**
(no operands)	4	2	XFER

Table D-4. Instruction Fetch Timings (Clock Periods)

Instruction Length (Bytes)	**Bus Width**		
	8	**16**	
		(1)	**(2)**
2	14	7	11
3	18	14	11
4	22	14	15
5	26	18	15

(1) First byte of instruction is on an even address.

(2) First byte of instruction is on an odd address. Add 3 clocks if first byte is not in queue (e.g., first instruction following program transfer).

APPENDIX E
GLOSSARY OF RBF-89 TERMS

The following glossary presents definitions of commonly-used RBF-89 terms:

Breakpoint	Prespecified Location where channel program execution is to terminate.
Busy Flag	Flag in IOP memory (second and tenth bytes in CCB), used to indicate presence or absence of channel-program activity.
CA	Channel-atention directive—special interrupt to 8089 IOP, used to prepare for channel-program execution.
CA CONTINUE	Channel-attention command to resume execution of interrupted channel program.
CA HALT	Channel-attention command to halt channel-program execution.
CA START	Channel-attention command to begin channel-program execution.
CCB	Channel Control Block, part of communication data structure. Provides pointer to parameter block, BUSY flag, and CCW.
CCP	Channel Control Pointer, which points to CCB.
CCW	Channel Command Word in Channel Control Block, which indicates type of operation IOP is to perform.
CP	Channel Control Pointer (see CCP).
Host Program	Part of RBF-89 that resides in Intellec development system RAM. Translates keyboard input into low-level directives that can be processed by RBF-89 MONITOR program, and converts information supplied by MONITOR into display output. Also maintains appropriate 8086 register settings and memory allocation to allow system to alternate between RBF-89 and 8086 user program processing.
IOP	Intel 8089 Input/Output Processor.
Local Memory	Memory on CPU/IOP system bus that is addressable by both CPU and IOP microprocessors.
Local Mode	CPU/IOP configuration where IOP is closely-coupled to CPU. Also known as system memory.
MASTER/SLAVE	SEL-pin toggle mode that determines whether CPU or IOP has control of system bus upon system start-up.
MONITOR Program	Part of RBF-89 that resides in ICE-86 memory. Monitors such operations as: preparing program control blocks for communication with IOP; issuing commands to start, terminate, and continue channel-program; and directing IOP to start execution of RBF-89 Utility Program.

PP	Parameter Block Pointer, which points to the Parameter Block.
PS	Program Status Word that records state of channel so that channel operation may be suspended and later resumed.
R	Prefix for all 8089 register references in RBF-89 commands.
RBC Register	Byte Count Register for IOP, used as a general register during channel-program execution; contains byte count used to terminate DMA transfer.
RBYTE	Byte in IOP remote memory.
RCC	Channel Control Register in IOP.
Remote Memory	Memory addressable by 8089 IOP only.
Remote Mode	CPU/IOP configuration where IOP is isolated from CPU.
RGA	General-Purpose Register A for IOP. Includes 20-bits plus Tag pointer. Points to source or destination of data during DMA transfer.
RGB	General-Purpose Register B for IOP. Functionally interchangeable with RGA.
RGC	General-Purpose Register C for IOP. Functionally interchangeable with RGA and RGB.
RIX	Index REgister (16 bits) for IOP.
RMC	Mask/Compare Register for IOP. (Upper eight bits for mask; lower eight bits for compare.)
RTP	Task Pointer in IOP, a 20-bit register with tag bit. Analogous to Program Counter (PC) in CPU.
RWORD	Word in IOP memory.
SCB	System Configuration Block which contains master/slave status information, input/output bus width, and pointer to CCB.
SCP	System Configuration Pointer, which indicates location of SCB.
SIB	System Initialization Block which contains the System Configuration Bus Pointer and system bus width.
SOC	System Operation Command word, which controls operation of request/grant arbitration circuit in IOP and sets physical width of input/output bus.
Tag	One-bit pointer to local or remote memory space, used in RGA, RGB, RGC, and RTP register settings.
TGA	Tag bit of RGA register.

TGB	Tag bit of RGB register.
TGC	Tag bit of RGC register.
Utility Program	Part of RBF-89 that resides in 8089 RAM in prototype application system. Reads and writes data to and from 8089 memory and registers, and sets and removes breakpoints to channel program.

INDEX

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.

NO POSTAGE
NECESSARY
IF MAILED
IN U.S.A.

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1040 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
Attn: Technical Publications M/S 6-2000
3065 Bowers Avenue
Santa Clara, CA 95051

intel®

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

Printed in U.S.A.