ICD-378
for
80186/80188
USER'S MANUAL

Part No. ZTP-103-00, Rev. B. Printed: November 1986

Limitation on Warranties and Liability

ZAX Corporation warrants this equipment to be free from defects in materials and workmanship for a period of 1 (one) year from the original shipment date from ZAX. This warranty is limited to the repair and replacement of parts and the necessary labor and services required to repair this equipment.

During the 1-year warranty period, ZAX will repair or replace, at its option, any defective equipment or parts at no additional charge, provided that the equipment is returned, shipping prepaid, to ZAX. The purchaser is responsible for insuring any equipment returned, and assumes the risk of loss during shipment.

Except as specified below, the ZAX Warranty covers all defects in material and workmanship. The following are not covered: Damaged as a result of accident, misuse, abuse, or as a result of installation, operation, modification, or service on the equipment; damage resulting from failure to follow instruction contained in the User's Manual; damage resulting from the performance of repairs by someone not authorized by ZAX; any ZAX equipment on which the serial number has been defaced, modified, or removed.

Limitation of Implied Warranties

ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO THE LENGTH OF THIS WARRANTY. IN NO EVENT WILL ZAX BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS EQUIPMENT. THIS EXCEPTION INCLUDES DAMAGES THAT RESULT FROM ANY DEFECT IN THE SOFTWARE OR MANUAL, EVEN IF THEY HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

Disclaimer

Although every effort has been made to make this User's Manual technically accurate, ZAX assumes no responsibility for any errors, omissions, inconsistencies, or misprints within this document.

ZAX Corporation
Technical Publications Department
2572 White Road, Irvine, California 92714

IBM is a registered trademark of International Business Machines Corporation.

Written by Mark Johnson of ZAX Technical Publications.

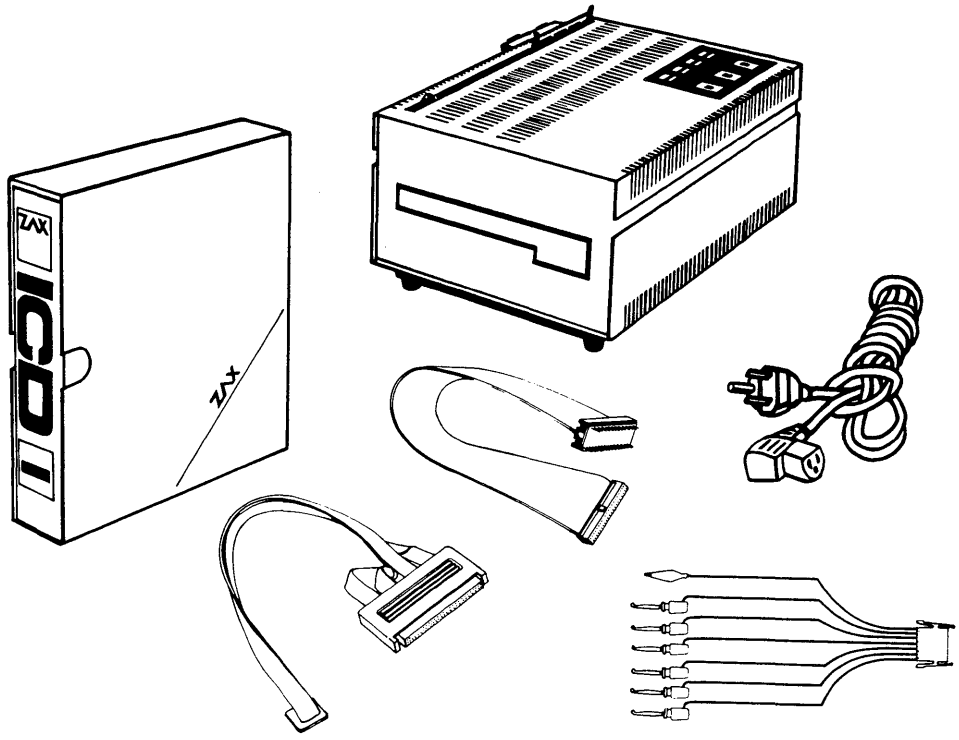
Reorder User's Manual ZTP-103-00

Reorder Command Reference Guide ZTP-606-00

Contents

SECTION 2 - MASTER COMMAND GUIDE

SECTION 3 - TECHNICAL REFERENCES


ZAX
ICD I
ZAX

ICD for 80186/80188 Features

General Characteristics

* 80186/80188 CPU and 8087 NDP support * Host computer support * All memory available * All I/O ports (64K bytes) available

User Interface

* You control all functions from terminal or computer * Mnemonic command names * Setup emulation controls from batch file on host computer * Symbolic debugging available with ZICE * In-line assembler

Emulation Controls

* Internal or external clock * Disable interrupt inputs * Disable bus request input

Memory Mapping

* 128K bytes standard emulation memory * 1M byte maximum emulation memory * 1K-byte mapping resolution * Read-only or read/write emulation memory * "No memory" mapping specification * Programmable wait states * Map override input * Control from keyboard

Address and Data Specifications

* Four offset registers * One bit "don't care" resolution

Breakpoints

* Four hardware breakpoints * Eight software breakpoints
* Break on a specified address or data * Break on range
* Break on access to non-memory area * Break on write to read-only area
* Sequential break (A then B) * Break on opcode fetch only
* Break on instruction execution * Break on Nth occurrence
* Break on wait state timeout * External break input (triggers from HI or LO signal edge) * External break output

Non-Real-time Trace

* Single step * Step n steps * Trace Jump instructions only

Real-time Trace

* Stores addresses, data, and status * 4K bytes deep x 40 bits wide trace memory size * Trace control modes include: Begin Monitor, End Monitor, Begin Event, End Event, Center Event, Multiple Event * Real-time counter * Adjustable delay

Disassembly Capabilities

* Disassemble from program memory
* Disassemble trace memory from any selected area

Special Features

* Assemble into memory * Use ICD's serial interface from user program
* Search program memory for pattern * Search trace memory for pattern

Package Includes :

1) ICD-378 Emulator with 80186 or 80188 processor
1) CPU In-circuit Emulation Probe
1) NDP In-circuit Emulation Probe
1) Monitor Probe Set
1) AC Power Cord
1) User's Manual

About This Manual

Thank you for choosing a ZAX in-circuit emulator! Your ZAX emulator is one of the most powerful and sophisticated microprocessor development tools in the industry - as you will soon discover. But for all the things your emulator can do, it's still very simple to use. In fact, you don't have to know a thing about ZAX emulators to use this manual. The information presented in this manual is structured for first-time users, so you'll be learning about emulation techniques and applications as well. If you're already familiar with the principles of emulation, you can use this manual now to learn a few basic emulator skills, and then use the section on commands as a reference.

What This Manual Will Show You

* How to identify the parts (controls, components & accessories) of your emulator and understand what they do (Section 1).

* How to connect the emulator to your terminal, host computer and target system (Section 1).

* How to find out more about special emulator controls and learn how to use them for your specific applications (Section 1).

* How to use the accessories that come with your emulator (Section 1).

* How to use each of the emulator commands (Section 2).

* How to learn more about how your emulator works, by examining the internal control modules (Section 3).

* How to write support software programs for interfacing the emulator with a host computer (Section 4).

How To Use This Manual

There are really only two things you must know to use a ZAX emulator: the first is how to connect it to your present system; the second is how to control the emulator's operation by using the commands. These two subjects are presented in the first two sections of this manual, and of these two, you'll be using the section on "commands" particularly.

So first, read Section 1 to learn about the various controls and components of your emulator. (Before you can operate the emulator, you'll have to set certain switches and make some minor adjustments so that it performs correctly with your system.) Then, continue on to learn how to connect your emulator to other devices, such as a console terminal or a host computer, and your target system.

Once your emulator is working properly, you can refer directly to Section 2 to find out how to enter any of the emulator commands. Each command's function is examined along with the format needed to use the command. Once you're familiar with the command syntax, you can use the fold-out Command Reference Guide located in the front of the manual.

If you need a refresher course on emulation principles, turn to Appendix A. If you're not sure session (we call it "debugging"), turn to Appendix B for a demonstration. Use Section 3 for a reference (it contains technical information that you may find useful later on). You can use Section 4 if you're writing your own support software programs to interface your host computer (if it's not already supported by ZAX's ZICE communication software) to the emulator.

Oh by the way, any time a word or phrase is used and you don't understand its meaning, turn to the Glossary at the back of this manual. It contains definitions for a number of common engineering terms as well as many specialized microprogramming terms.

Terminology

In this manual, you will see the initials "ICD" used whenever we mean the ICD-378, in-circuit debugger, emulator, or in-circuit emulator. You will also see the terms "user" and "target system" used to describe your prototype hardware.

NOW TURN TO SECTION 1 AND GET STARTED.

Introduction

In Section 1, you'll learn about the different parts of your ICD, what they do, and how to use them. You'll also learn how to connect the ICD to your system (terminal, host computer, target system) and find out how to use the accessories that come with the ICD. Your ICD has a few find information about these features in this section as well.

A Word Of Caution

You shouldn't try to attach the ICD to any external device before you finish reading this section. As long as the power cord is disconnected you can't hurt anything internally, but don't connect the ICD to your target system before you read "How To Connect Your ICD To Other Devices," later in this section. Although it's difficult, it is possible to get the cables to the target system reversed, which could result in damage to the ICD's internal components.

Getting Acquainted With Your ICD

Your ZAX ICD-series in-circuit emulator is a microprocessor emulation device that can be used for developing and maintaining 80186/80188 microprocessor-based systems as well as the 8087 Numeric Data Processor (NDP). It does this by letting you direct and test activities in your prototype ("target") system. You perform these operations by entering one or more debugger commands.

All ZAX ICD-series emulators are controlled by a separate terminal, or in conjunction with your existing host computer system. You can use the debugger commands for your hardware or software projects by simply inputting the command mnemonics and parameters from just about any terminal or popular computer you might own.

A Few Features

Here are just a few things you can do using the debugger commands:

* Use the ICD's emulation memory to simulate or take the place of memory (or future memory) in your target system.

* Use a single-step trace operation to move through your program, one step at a time, and examine the registers' contents after each step.

* Set a combination of hardware and software breakpoints to stop your program when: data is written or read into a specific address; an event point is passed; a non-existent memory access is attempted; or an interrupt is acknowledged by the CPU. Hardware breakpoints can also generate triggers for instruments such as logic analyzers and oscilloscopes.

* Record ("trace") a portion of your program (beginning and ending anywhere within the program) and store it in the ICD's real-time trace buffer without affecting the emulation process. Later you can display the recorded memory contents in either machine code or in its disassembled format.

* Translate symbolic codes into machine instructions, item for item, using the in-line assembler.

* Selectively enable and disable the interrupt or bus request inputs - including non-maskable interrupts.

You can turn to Section 2 for a complete list of the ICD's debugger commands. To find out about other things your ICD can do, turn to "More About Your ICD," in this section.

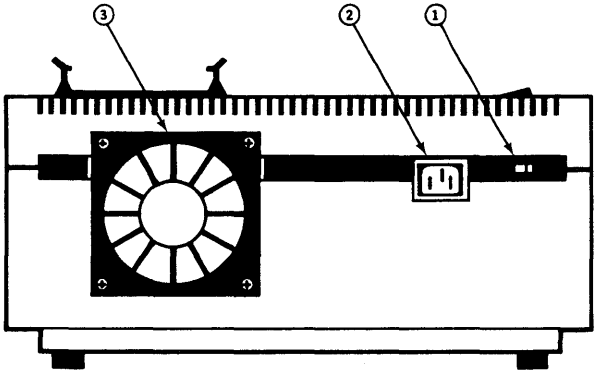
PROCEED TO THE NEXT PAGE TO LEARN ABOUT THE PARTS OF YOUR ICD.

The Controls And Component Functions Of Your ICD

1) AC POWER Select Switch. This switch is used to select the power requirements for the ICD. Set the switch to 110V/117V to run on a power supply of 110-120VAC or select 200V/240V to run on a power supply of 200-240.

2) AC POWER Cord Receptacle. Accepts female end of the supplied three-wire power cord. Be sure to disconnect the power cord before moving the ICD.

3) COOLING FAN. Provides air circulation for the control modules. The cooling fan is an integral-type design; there are no external connections required.



4) TERMINAL Port Connector. Accepts male end of an RS-232 cable to attach the ICD to a terminal in a stand-alone (LOCAL mode) configuration. When using the ICD in the REMOTE mode, this port can be used as an auxiliary I/O.

5) HOST/AUX Port Connector. Accepts male end of an RS-232 cable to attach the ICD to a host computer system when the ICD is operating in the REMOTE mode. ICD commands can then be entered using the computer's keyboard. When using the ICD in a stand-alone (LOCAL mode) configuration, this port dumps object code, registers, or memory to a host computer or printer.

6) LOCAL/REM (Local/Remote) Select Switch. This switch is used to select which port (TERMINAL or HOST/AUX) the ICD will use to receive commands.

7) BAUDRATE Switches (TERMINAL and HOST/AUX). These switches are used to set the baud rates for the TERMINAL and HOST/AUX ports. The factory setting is #1 (9600bps). To change the baud rates for the ports, see "Changing the Baud Rate Settings," in Section 3.

8) DCE/DTE Select Switch. This switch is used to set the HOST/AUX port to either RS-232 data terminal equipment (DTE) or data communications equipment (DCE). Use the DCE setting if the ICD is used with a host computer. Use the DTE setting if a printer is connected to the HOST/AUX port. (The TERMINAL port is always DCE.)

9) NDP In-circuit Probe Receptacle. Accepts female end of the NDP In-circuit Probe for emulating the 8087 co-processor.

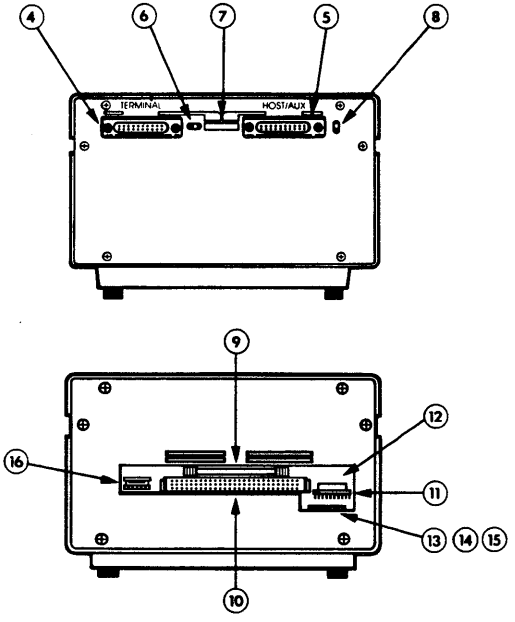
10) CPU In-circuit Probe Receptacle. Accepts female end of the CPU In-circuit Probe. (The connector end of the probe is keyed for proper polarity recognition.)

11) Emulation Method Select Switch #1. This switch is used to set the machine cycle operation to the target system. (See "More About Your ICD," later in this section, for details about this switch.)

12) Emulation Method Select Switch #2. This switch is an extension of the Emulation Method Select switch #1, which is used to set the machine cycle operation to the target system. (See "More About Your ICD," later in this section, for details about this switch.)

13) - 15) Trigger Cable Receptacle. Provides external sensing through three different probes, including the EXT.BRK. (External Break), EVENT TRG. (Event Trigger) and EXT.L/E. (Level/Edge Trigger) Probes. (See "More About Your ICD," later in this section, for details about this cable.)

16) CLKS (Clock) Simulator Module. When the ICD runs off the external clock from a target system, the clock circuit must be simulated on this module. (See "More About Your ICD," later in this section, for more details about this feature.)



17) POWER On/Off Switch. This switch is used to supply power to the ICD.

18) CLOCK INT/EXT Switch. This switch is used to select either the ICD's clock (INT=internal) or the target system's clock (EXT=external).

19) HALT Lamp. This LED comes on after the CPU has stopped executing a HALT instruction or when a BUSAK (BUS ACKNOWLEDGE) is in progress.

20) RESET Switch. This switch is used to reset the ICD monitor. You can push it any time the MONITOR lamp is lit. After you push the RESET switch, you'll see the ICD's identification message on your terminals's monitor.

21) MONITOR Lamp. This LED comes on to indicate that control is currently in the ICD's monitor. It will not be lit during emulation.

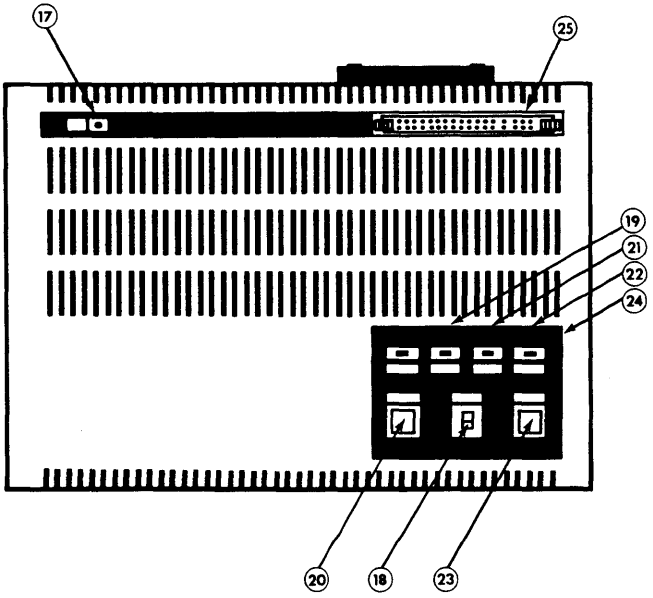
22) ICE (In-Circuit Enable) Lamp. This LED comes on when the ICD is operating in the in-circuit mode I1 or I2.

23) MONITOR Break Switch. This switch is used to return control to the ICD monitor during emulation.

24) POWER Lamp. This LED comes on to indicate that power is being supplied to the ICD.

25) STATE SIGNAL Output Connector. Allows the ICD to be interfaced with a logic-state analyzer. (See the appendices for more information on this feature.)

NOW TURN TO THE NEXT CHAPTER TO LEARN HOW TO CONNECT THE ICD TO YOUR SYSTEM.


17
25
19
21
22
24
20
18
23

How To Connect Your ICD To Other Devices

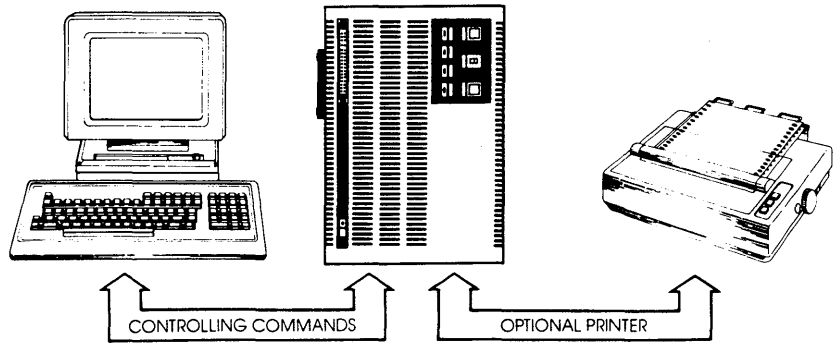
In the main introduction, you read that properly connecting the ICD to your system was one of the most important things you would learn in this manual. The following information will show you how to connect the ICD's components, what cables to connect and where they go, and which switches are set to what positions. Once you've completed the procedures outlined in this section, you'll have what is called a "Microprocessor Development System" (MDS). By using the commands and applications found in Section 2, you'll be able to perform a remarkable variety of debugging operations with your MDS.

About Your System's Environment

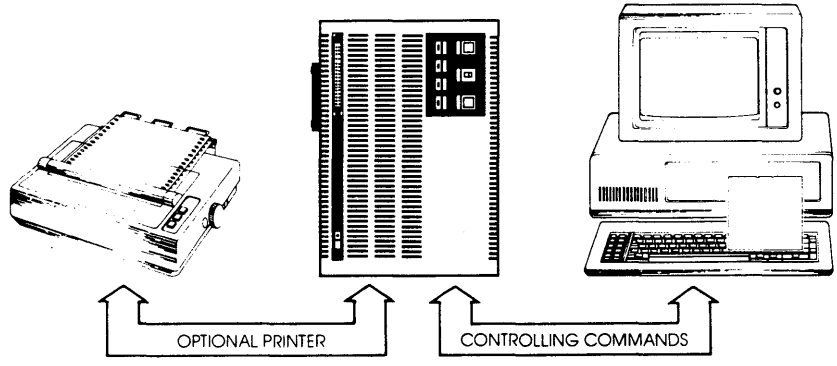
Before you connect your ICD to anything, you'll need to answer three questions about your system's environment. First, will you control the system with a terminal or a host computer? Second, if a terminal is used to control the ICD, will a host computer be used as a source for data files? And third, will your system be used to develop/debug hardware or software?

Terminal or Host Computer Controlled?

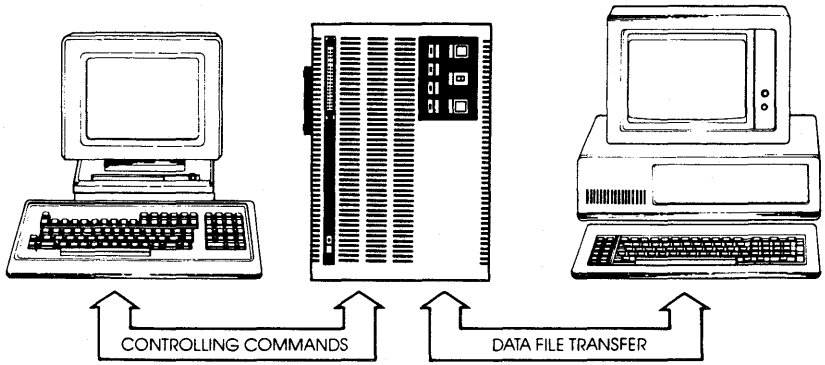
If you'll be controlling the ICD by a console terminal, it's called TERMINAL CONTROL OF THE ICD.In this configuration, the ICD "stands alone" (hence the name, stand-alone emulator), or apart from the auxiliary control of a host computer system. The ICD assumes a stand-alone mode of operation when you place the LOCAL/REM switch to the LOC (LOCAL) position.



If you'll be controlling the ICD with a host computer and using the utility software program ZICE, it's called HOST COMPUTER CONTROL OF THE ICD. The ICD assumes this mode of operation when you place the LOCAL/REM switch to the REM (REMOTE) position.



You may choose to control the ICD with a terminal and use a separate host computer to store data files. A printer can also be connected to the host computer to dump data for hard copies. This configuration is called TERMINAL CONTROL OF THE ICD (WITH HOST DATA FILES). In this configuration, the ICD is still under direct control of the terminal, while the host computer serves as a data storage device. You can also cause the ICD to assume a "transparent" condition, which allows direct communication between the terminal and host computer.

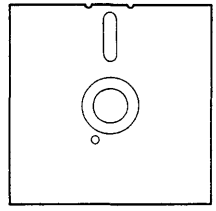


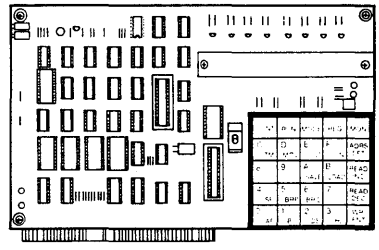
NOTE: ZICE software may be used in the LOCAL mode - TERMINAL CONTROL OF THE ICD (WITH HOST DATA FILES) - for performing symbolic dedug and accessing the ZICE commands (help files, "Z" commands, etc.). To use this LOCAL "host computer assisted" mode, see the HOST command, in Section 2.

Hardware Or Software?

Your hardware is called a "target system." By physically removing the 80186/80188 CPU (or 8087 your system and electronically replacing it with the ICD's internal microprocessor, you can control, test and check almost all possible functions in your target system.

Can you use your ICD without a target system? Of course! Whenever you develop and debug software, you'll be doing it without the use of a target system. This mode is also an effective way to demonstrate some of your ICD's features.





SYSTEM CONFIGURATION CHARACTERISTICS

SYSTEM CONFIGURATION CHARACTERISTICS	TERMINAL CONTROL OF THE ICD	TERMINAL CONTROL OF THE ICD (WITH HOST DATA FILES)	COMPUTER CONTROL OF THE ICD
Operation Mode	LOCAL	LOCAL	REMOTE
Controlling Device	Console Terminal	Console Terminal	Computer
Recommended Baud Rate (bps)	9600	TERMINAL=19200 HOST/AUX=9600	9600
Memory Storage Facility	ICD Internal Only	Computer	Computer
Computer's Role	Not Used	Memory storage, ZICE access	Controlling device, ZICE access, memory storage
Optional Target System?	Yes	Yes	Yes
Optional Printer?	Yes	Yes, if connected to computer	Yes, if connected to computer
Can Emulate CPU and NDP?	Yes	Yes	Yes
Number Of RS-232 Cables Needed	1 (2 if printer is used)	2	1 (2 if printer is used)
Uses ZICE Software?	No	Optional	Yes
Is ZICE Software Used For ICD Interface?	No	No	Yes
Can Access ZICE Commands?	No	Yes	Yes

Summing It All Up ...

* Your ICD can function in any of three different system configurations.

* Your ICD can be used to debug hardware or software.

* Your ICD can operate with or without a target system.

* Your ICD can dump data directly to a printer.

* Your ICD can dump data to a printer attached to a host computer.

* Your ICD can be controlled by just a terminal or by a host computer.

* Your ICD can be controlled by a terminal and use a separate hostcomputer for storing data files.

* Your ICD can be controlled by a terminal and use a separate host computer for accessing the ZICE commands.

NOW TURN THE PAGE AND READ ABOUT PREPARING A SITE FOR YOUR SYSTEM.

About Grounds

System Preparation Read this chapter before you connect anything!

Your ICD is equipped with a 3-wire polarized receptacle that accepts a 3-wire cord. This cord connects to a power source and protective ground. Make sure that you plug the power cord into a properly grounded 115 VAC receptacle. Do not try to bypass the 3-prong plug with an adapter (3- into 2-prong adapter).

About Power

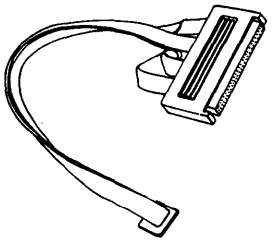
WARNING: THE GROUND TERMINAL OF THE 3-PRONG PLUG IS USED TO PREVENT SHOCK HAZARDS - DO NOT BYPASS IT!

Your ICD is normally set to operate on a voltage supply of 110-120 VAC, but this can be changed to 200-240 VAC by setting the Power Select switch to the 200V/240V position.

In most cases a multiple power outlet strip should be used to provide voltage to the entire system (host computer, terminal, printer, target system). Most power outlet strips are equipped with a circuit breaker in case of an overload, and all are properly grounded.

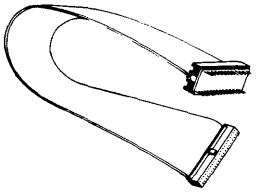
No matter what type of power source you use, always apply power after connecting the ICD to an electrical outlet, and always apply power in the same sequence: switch on the power supply first, and then press the POWER On/Off switch to On.

Important Facts About The In-Circuit Probes



The CPU and NDP in-circuit probes are used to interface the ICD to your target system. The CPU in-circuit probe consists of four 16.5-inch ribbon cables that are joined to two end connectors. The small end connector attaches to your prototype design (target system) and the large end connector attaches to the ICD's CPU 96-pin receptacle. The CPU carrier on your prototype board is a 68-pin JEDEC type A ceramic unit, which is the same configuration as the unit within the ICD. The CPU connector end of the in-circuit probe is designed to attach to the CPU carrier in the same manner as the 80186/80188 processor - no other connections are necessary. The CPU in-circuit probe connector is keyed for proper polarity recognition.

The NDP in-circuit probe is used to connect the ICD to your target system when you are emulating the 8087 Numeric Data Processor. This probe features a single cable and 40-pin connector and must be plugged into the receptacle labeled NDP.



NOTE: In many cases the NDP in-circuit probe may not need to be used because both the CPU and NDP are connected, internally, within the ICD. By adjusting the settings of the Emulation Method Select switch, the NDP in-circuit probe can usually be omitted, and NDP emulation can be performed using only the CPU in-circuit probe.

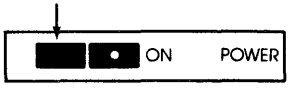
For more information on this subject, and to see if you can omit using the NDP in-circuit probe, see the chapter on "NDP Emulation," in Section 3.

NOW TURN TO THE NEXT PAGE TO LEARN HOW TO PREPARE YOUR ICD FOR OPERATION.

Preparing Your ICD

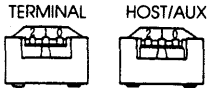
Before you attach a system-controlling device (terminal or host computer) or your target system to the ICD, complete the following steps:

First -
Make sure that the POWER On/Off switch is set to Off.



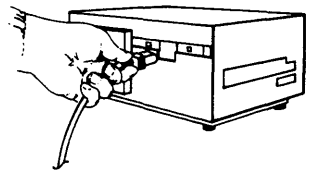
Now -
Check the Power Select switch on the side of the ICD. Set this switch to the same voltage as your power source.

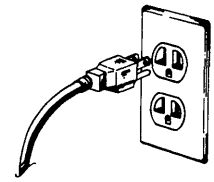
Optional: The ICD's baud rates for the TERMINAL and HOST/AUX communication ports are factory-set at 9600bps. To change the baud rates, see "Changing The Baud Rate Settings," in Section 3.



Then -

Plug the AC POWER CORD into the ICD's power receptacle and connect the other end of the cable to a power source.





Now turn to the appropriate heading - which you'll find on one of the following pages - to construct your microprocessor development system.

System Configuration	**Terminal Control Of The ICD**
Operation Mode	LOCAL
Controlling Device	Console Terminal
Optional Printer?	Yes
Optional Target System?	Yes
Number Of RS-232 Cables Needed	1 ——— 2 if printer is used
Recommended Baud Rates (bps)	9600
Uses ZICE Software?	No

Use the illustration on the opposite page and the information below to construct this system configuration. Then adjust the switches as indicated in the bottom-right column.

CONSTRUCT YOUR SYSTEM

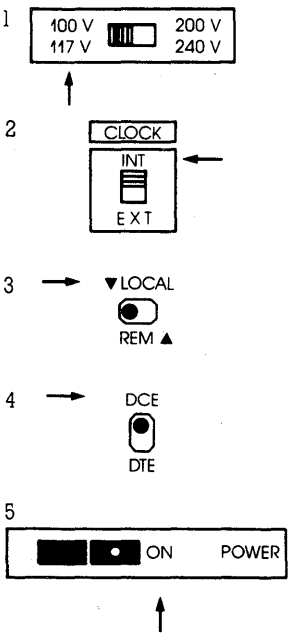
1) Connect your terminal to the ICD by using an RS-232 cable. Attach the cable from your terminal's serial (EIA RS-232) port to the ICD's TERMINAL port connector. The ICD defaults to 9600 baud, 8 data bits, 2 stop bits and no parity: set your terminal to these specifications.

2) [Optional] Connect your printer to the ICD by using an RS-232 cable. Attach the cable from your printer to the ICD's HOST/AUX port connector.

3) [Optional] If you're debugging a target system, remove the existing CPU (80186/80188) from your target system and insert the CPU IN-CIRCUIT PROBE into the target system's chip carrier. Connect the other end of the CPU IN-CIRCUIT PROBE to the ICD's in-circuit probe receptacle.

If you're debugging a target system containing an NDP (8087) and you wish to use the NDP in-circuit probe, remove the existing NDP from your target system and insert the NDP IN-CIRCUIT PROBE (40-pin end) into the target system's NDP socket.

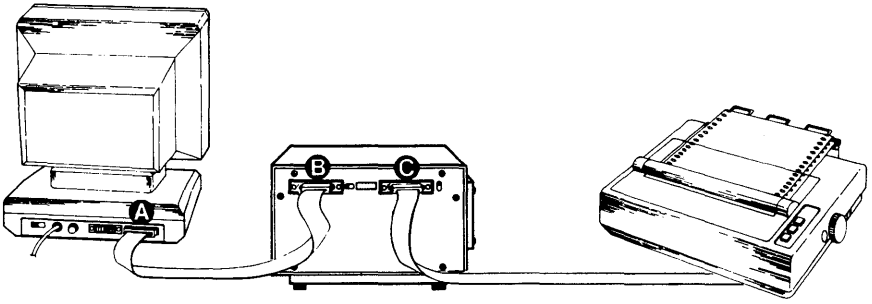
ADJUST THESE SWITCHES



The following message should now appear on your monitor's screen (you may have to press the RESET switch on the ICD): **ICD-378 for 80186 V1.X**

NOW TURN TO **"WHAT CAN YOU DO WITH YOUR MDS?"** IN THIS SECTION.

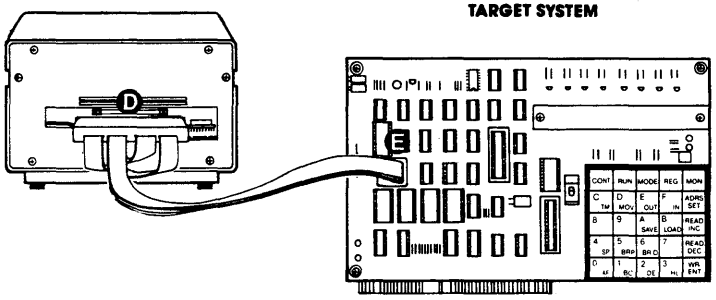
Ⓐ **Terminal's EIA RS-232 port**
Ⓑ **ICD's TERMINAL port**
Ⓒ **ICD's HOST/AUX port**
Ⓓ **ICD's in-circuit probe receptacle (CPU/NDP)**
Ⓔ **Target system's CPU/NDP socket**



CONTROL CONFIGURATION

ICD

TARGET SYSTEM



System Configuration	Terminal Control Of The ICD (With Host Data Files)
Operation Mode	LOCAL
Controlling Device	Console Terminal
Optional Printer?	Yes
Optional Target System?	Yes
Number Of RS-232 Cables Needed	2
Recommended Baud Rates (bps)	Terminal: 19200, Host: 9600
Uses ZICE Software?	Optional
Can Access ZICE Commands?	Yes

Use the illustration on the opposite page and the information shown below to construct this system configuration.

CONSTRUCT YOUR SYSTEM

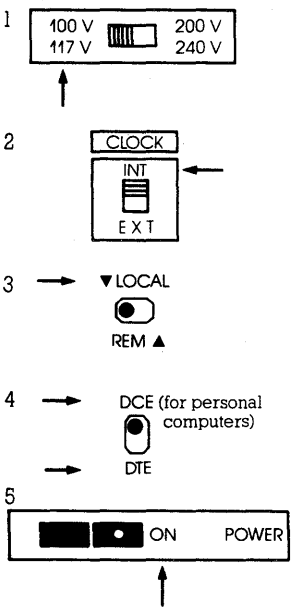
1) Connect your terminal to the ICD by using an RS-232 cable. Attach the cable from your terminal's serial (EIA RS-232) port to the ICD's TERMINAL port connector. The ICD defaults to 9600 baud, 8 data bits, 2 stop bits and no parity: set your terminal to these specifications.

2) Connect your host computer to the ICD by using an RS-232 cable. Attach the cable from your host computer's serial (EIA RS-232) port to the ICD's HOST/AUX port connector.

3) [Optional] If you're debugging a target system, remove the existing CPU (80186/80188) from your target system and insert the CPU IN-CIRCUIT PROBE into the target system's chip carrier. Connect the other end of the CPU IN-CIRCUIT PROBE to the ICD's in-circuit probe receptacle.

If you're debugging a target system containing an NDP (8087) and you wish to use the NDP in-circuit probe, remove the existing NDP from your target system and insert the NDP IN-CIRCUIT PROBE (40-pin end) into the target system's NDP socket.

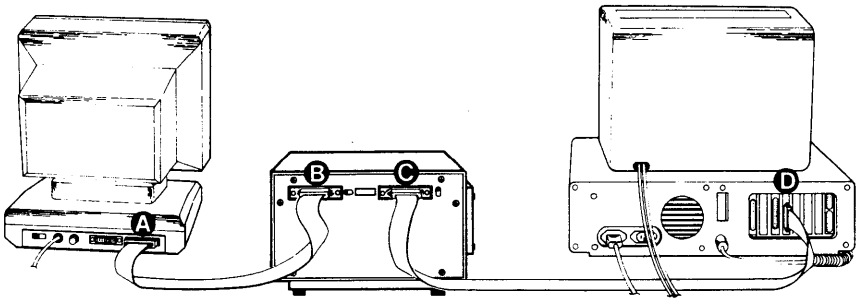
ADJUST THESE SWITCHES

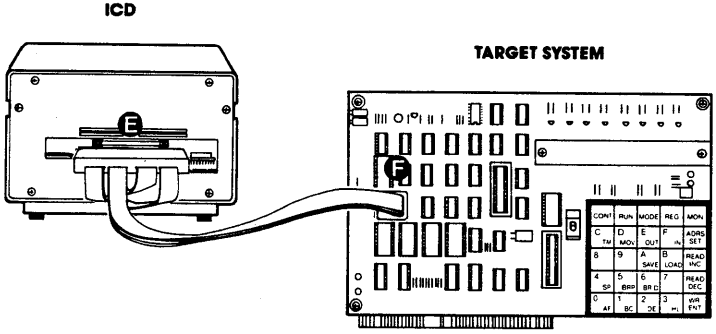


The following message should now appear on your monitor's screen (you may have to press the RESET switch on the ICD): **ICD-378 for 80186 V1.X**

NOW TURN TO "WHAT CAN YOU DO WITH YOUR MDS?" IN THIS SECTION.

Ⓐ Terminal's EIA RS-232 port
Ⓑ ICD's TERMINAL port
Ⓒ ICD's HOST/AUX port
Ⓓ Computer's SIO port
Ⓔ ICD's In-circuit probe receptacle
Ⓕ Target system's CPU socket





System Configuration	**Host Computer Control Of The ICD**
Operation Mode	REMOTE
Controlling Device	Host Computer
Optional Printer?	Yes
Optional Target System?	Yes
Number Of RS-232 Cables Needed	1 —— 2 if printer is used
Recommended Baud Rates (bps)	Host: 9600
Uses ZICE Software?	Yes
Can Access ZICE Commands?	Yes

Use the illustration on the opposite page and the information shown below to construct this system configuration.

CONSTRUCT YOUR SYSTEM

1) Connect your terminal to the ICD by using an RS-232 cable. Attach the cable from your terminal's serial (EIA RS-232) port to the ICD's TERMINAL port connector. The ICD defaults to 9600 baud, 8 data bits, 2 stop bits and no parity: set your terminal to these specifications.

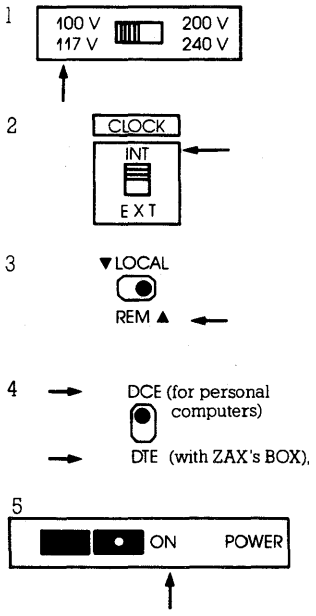
2) Connect your host computer to the ICD by using an RS-232 cable. Attach the cable from your host computer's serial (EIA RS-232) port to the ICD's HOST/AUX port connector.

3) [Optional] If you're debugging a target system, remove the existing CPU (80186/80188) from your target system and insert the CPU IN-CIRCUIT PROBE into the target system's chip carrier. Connect the other end of the CPU IN-CIRCUIT PROBE to the ICD's in-circuit probe receptacle.

If you're debugging a target system containing an NDP (8087) and you wish to use the NDP in-circuit probe, remove the existing NDP from your target system and insert the NDP IN-CIRCUIT PROBE (40-pin end) into the target system's NDP socket.

4) Load the ZICE software program necessary for interfacing the ICD with your host computer. Execute the program loading commands as outlined in the ZICE software documentation.

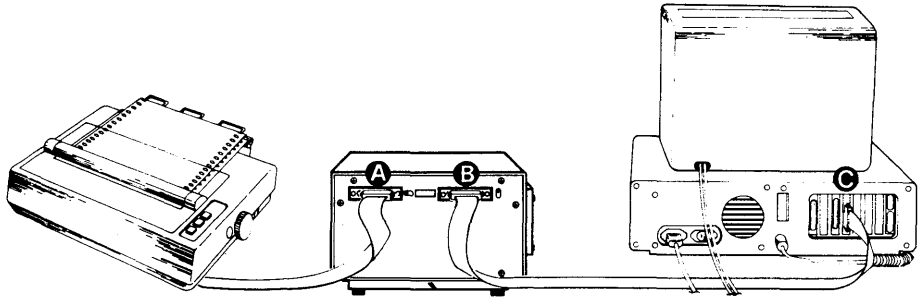
ADJUST THESE SWITCHES



The following message should now appear on your monitor's screen (you may have to press the RESET switch on the ICD): **ICD-378 for 80186 V1.X**

NOW TURN TO "WHAT CAN YOU DO WITH YOUR MDS?" IN THIS SECTION.

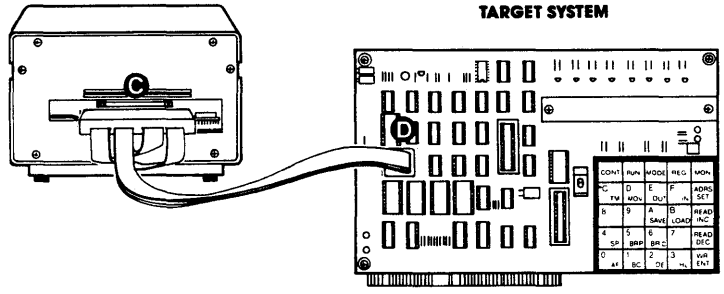
Ⓐ **ICD's TERMINAL port**
Ⓑ **ICD's HOST/AUX port**
Ⓒ **Computer's SIO port**
Ⓓ **ICD's in-circuit probe receptacle (CPU/NDP)**
Ⓔ **Target system's CPU/NDP socket**



CONTROL CONFIGURATION

ICD

TARGET SYSTEM



What To Do With Your MDS

You should now have a fully operational Microprocessor Development System (MDS) capable of developing and debugging your hardware or software designs. If your MDS is functioning correctly, and the ICD's identification message appears on your monitor's screen, you can now:

* Turn to the "Master Command Guide," Section 2, for a complete analysis of your ICD's debugger commands.

* Turn to Appendix B for a demonstration of the features and functions of your ICD.

* Use the fold-out "Command Reference Guide" (from the front of this manual) as a source for the various command formats.

What To Do If Your MDS Is Not Working

If your MDS is not functioning correctly or gives you problems during emulation, turn to "Trouble Shooting," on the next page. Start by reading "Checking Electrical Connections," and then proceed to "Diagnosing ICD Interface Problems" if you encounter problems when you're emulating.

Trouble Shooting

Introduction: The Problem ...

Your ICD must be controlled by either a separate terminal or a host computer's keyboard. Because you must connect the ICD to these external devices to form your development system, the possibility always exists of misplacing a cable, setting a switch to the wrong position, or bypassing a procedure.

... And The Solution!

"Trouble Shooting" is designed to get you through the problems you may have encountered in "How To Connect Your ICD To Other Devices." It begins with a typical example of what the ICD should do when the system is operating correctly. Then the ICD by itself is tested, followed by the ICD and terminal, together. ICD, terminal and target system configuration is then tested.

What Should Happen

When the ICD is connected to a terminal the following should happen:

When the ICD's POWER On/Off switch is pressed to On, the PWR (power) and MONITOR lamps should come on and the external cooling fan should be operating. The terminal's monitor should then show the ICD's identification message:

ICD-378 for 80186 V1.0

If the ID message does not appear, try pressing the RESET switch. A prompt (>) should also appear, indicating that the system is working properly and the ICD is ready to accept commands. At this point, any of the "status commands" (command name followed by a RETURN) can be entered.

They include: B, EV, H, I, MA, O, PI, R, SU, T

Try entering a few of the status commands. If the response from the ICD is the command's status, then the system is probably functioning properly. Otherwise, continue reading and following the procedures outlined in this chapter.

How To Get Your ICD Working

In this trouble-shooting session, you'll start by disconnecting the ICD from all external devices such as the target system, host computer or terminal. You'll check the ICD by itself (just connect its power cord), and then attach a terminal. If that configuration works properly, you can connect your target system for final testing.

NOTE: If you're using a host computer to control the ICD, be sure to check the ICD and host computer operation (together) BEFORE connecting your target system.

Checking Electrical Connections

Now begin with "Checking Electrical Connections."

a. Press the ICD's POWER On/Off switch to Off.

b. Turn the power Off on all externally attached devices (terminal, host computer, target system, etc.)

c. Disconnect all externally attached devices from the ICD.

d. Unplug the AC power cord from the ICD and from the wall outlet or power supply.

e. Check the wall outlet or power supply by plugging in a working device (lamp, terminal, logic analyzer, etc.). If the outlet or power supply is controlled by a switch, is the switch On?

f. Disconnect and reconnect each device's AC power cord to ensure a proper electrical connection.

PROCEED WITH "DIAGNOSING ICD INTERFACE PROBLEMS," ON THE NEXT PAGE.

Diagnosing ICD Interface Problems

Before you begin, make sure your terminal is working properly (i.e., the cursor on the screen should be visible). Then use an RS-232 cable to connect the ICD to the terminal.

PROBLEM:	SOLUTION:
The terminal does not respond when the RESET switch is pressed.	What's Probably Wrong: There is is either an interface problem or a defective component in the system. What To Do: First make sure that the RS-232 cable is firmly attached to both the ICD and terminal connectors. Is the cable defective? If the cable is OK, check that the INT/EXT clock switch is set to INT and that the LOCAL/REM switch is set to LOCAL. Make sure that both the ICD and terminal are transmitting at the same baud rates.
Terminal responds "gibberish" when the RESET switch is pressed.	What's Probably Wrong: The baud rates are different for the ICD and terminal. What To Do: Make sure that the baud rates for the ICD and the terminal are the same (your ICD's baud rate was factory-set at 9600).
Terminal responds with a C?> error message when any of the commands are entered.	What's Probably Wrong: On some terminals the ICD will only recognize a command which is stated with capital letters (e.g. R not r). What To Do: Press the Lock or Caps Lock key on your keyboard to the locked position.
ICD and Host Computer	Before you begin, make sure your computer is working properly (i.e., the cursor on the screen should be visible). Then use an RS-232 cable to connect the ICD and computer.

PROBLEM:	SOLUTION:
The computer does not respond at all when the ICD's RESET switch is pressed.	What's Probably Wrong: There is either an interface problem or a defective component in the system. What To Do: First make sure that the RS-232 cable is firmly attached to both the ICD and computer (if you are operating in the REMOTE mode). Is the cable defective? If the cable is Ok, check the position of the DCE/DTE switch on the ICD. Set this switch to the alternate position and press the RESET switch again. Make sure that both the ICD and computer are transmitting at the same baud rate. If you've reached this point with no problems, your difficulty probably lies in the ICD failing to emulate your target system. Now connect the ICD to your target system and read through the next check-out procedure.
ICD With Target System Connected	Connect the target system to the ICD, using the CPU in-circuit probe. Use a terminal to control the ICD.
PROBLEM:	SOLUTION:
Terminal doesn't work	What's Probably Wrong: There is either an interface problem or a defective component in the system. What To Do: Check that the ICD is properly connected to your target system, that the target system has power, and that the terminal is adjusted correctly. Select the EXTERNAL (EXT) clock, and press the RESET switch on the ICD. The ICD's identification message and prompt should appear. If a prompt fails to appear when the clock is set to EXT, switch to the INTERNAL (INT) clock and press RESET again. (With INT selected, the ICD and terminal should work independently of your target system.) If the ICD operates on the INT setting, the problem is probably a poor clock signal from your target system. It is possible to use the ICD with the INT setting, but you will lose real-time operation.
What To Do If The ICD Still Doesn't Work	In most cases, the procedures just listed will solve all but the most stubborn problems. However, it is possible that the ICD is still not functioning correctly. In this case, you should consult directly with ZAX Corporation.

More About Your ICD

Introduction

In this chapter, you'll learn how to use the accessory cables that come with your ICD, how to use the Clock Simulator Module, and what functions are performed by the Emulation Method Select switches (#1, #2 & #3). By taking advantage of these components, you'll be able to further expand your ICD's debugging capabilities.

The three accessory cables can be used to input and output pulses to and from the ICD. By using the six probes that are attached to the ends of these cables, you can:

* Determine if the ICD is emulating.

* Cause a breakpoint in your program to output a pulse to an external device.

* Selectively access either ROM or RAM.

* Cause the ICD to insert a break in your program when an external pulse is detected.

* Cause an event to occur within the emulated program when the HI/LO edge of an external signal is detected.

* Cause an event to occur within the emulated program when the HI/LO level of an external signal is detected.

The Clock Simulator Module may be used to simulate a crystal, RC or LC circuit by fabricating the associated circuit on the module.

The Emulation Method Select switches alter signal I/O between the ICD and target system during emulation.

Accessory Cables & Probes

To use the probes, see the chart on the following pages. Attach the probe connector to the ICD's TRIGGER CABLE receptacle (it is keyed for proper polarity recognition. and then attach the probes to the desired peripheral devices.

Probe Functions

Probe Name	Probe Color	Cable Color	What The Probe Does	How It's Used
Emulation Qualify	WHITE	ORANGE	Outputs a HIGH level signal from the ICD to the Emulation Qualify probe during emulation. During the MONITOR mode (after encountering a breakpoint or when the MONITOR button is pressed) the signal level is LOW.	The EQ signal can be used as an "emulation in progress" indicator or to remove unwanted signals during emulation.
Event Trigger	GREEN	YELLOW	Outputs a LOW level signal from the ICD to the Event Trigger probe when an event point is encountered during emulation.	The Event Trigger output is useful when a timing analysis of some external circuitry (not controlled by the ICD) is desired. In this application, the LOW level signal could be used to trigger a logic analyzer or oscilloscope.
Map Control	YELLOW	GREEN	Accepts a LOW level input signal from the target system to dynamically select between ROM and RAM within the ICD. A LOW level signal causes the ICD to set all memory as user (target) memory.	The ROM/RAM selection process is helpful when developing a system which uses phantom ROM (ROM that operates for the system bootstrap procedure and then "hides" behind the main memory). The Map Control signal allows you to access the same user memory address space that is occupied by the phantom ROM.

Probe Color	Probe Color	Cable Color	What The Probe Does	How It's Used
External Break	RED	BLUE	Accepts a LOW level input signal from an external component to trigger a break during program execution.	The External Break input is useful in capturing information (usually on the hardware level) that exists outside the control of the microprocessor.
External Level	WHITE	BROWN	Accepts a HIGH or LOW level signal from the target system to causes an event during program execution. The level sensitivity is controlled by the EVENT command.	The event can be used in the same manner as the normal EVENT command.
External Edge	GREEN	RED	Accepts a signal's HIGH-going or LO-going input from the target system to cause an event during program execution. The edge sensitivity is controlled by the EVENT command.	The event can be used in the same manner as the normal EVENT command.

Clock Simulator Module

The 80186/188 provides an integrated clock generator which generates the main clock signal for all 80186/188 integrated components, as well as all CPU synchronous devices in the system. The clock generator consists of a crystal-controlled oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

The ICD also supports an LC or RC circuit to be used with the oscillator. This circuitry may be fabricated on the ICD's Clock Simulator Module (CSM).

NOTE: The ICD will not operate on its own internal clock if the CSM is removed. Remove, modify and replace the CSM only if an external clock signal is required.

Removing The Clock Simulator Module

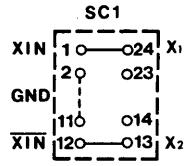
To modify or fabricate the CSM circuitry, remove the module by inserting the tip of a slot-headed screwdriver or similar pointed object into thehole located at the corner of the board. Gently pull the module away from the ICD mainframe. Next, remove the standard DIP header from the CSM and install the optional DIP header for fabricating the desired circuitry. Installing the CSM is the reverse.

NOTE: The standard DIP header may be used when either the ICD's internal clock is selected, or if the target system is supplying a TTL level clock. In this case, the clock setting is external.

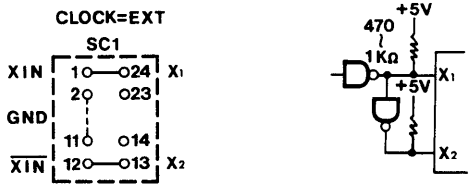
Modifying The Clock Simulator Module

Use the diagrams below to fabricate the associated circuit on the CSM. Then, set the ICD's INT/EXT clock switch to EXT in order to transfer clock timing to the CSM.

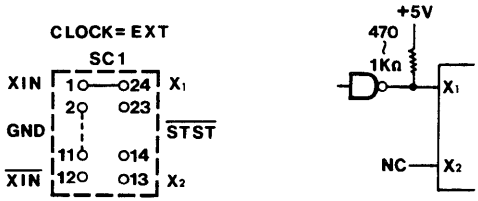
Internal Clock Configuration (Standard Circuitry)



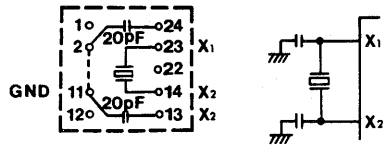
External Target Clock (1-10MHz Input Frequency; X1 Low Time > 40ns)

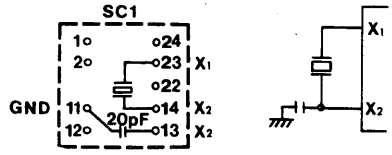


External Target Clock (1-6MHz Input Frequency; X1 Low Time > 60ns)

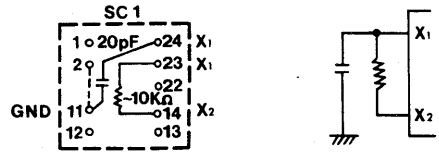


Quartz Crystal Clock
(1-6MHz; Clock Switch = INT or EXT)

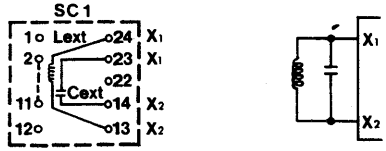




RC Circuit
(Clock Switch = INT or EXT)



LC Circuit
(Clock Switch = INT or EXT)



Emulation Method Select Switch #1

Description	The Emulation Method Select switch #1 is an 8-bit, On/Off type switch that controls signal I/O between the ICD and target system during emulation.
Location	The NDP/CPU connector end of the ICD. Switch #1 is located below switch #2. (See "The Controls And Component Functions Of Your ICD," earlier in this section.)
Function	See the individual bit settings that follow.
Adjustment	To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.
Factory Settings	All bits are Off.
Bit 1	On - RD signal outputs during user memory access only.
	Off - RD signal outputs during both user and ICD memory access.
Bit 2	On - WR signal outputs during both user and ICD memory access.
	Off - WR signal outputs during user memory access only.
Bit 3	On - DEN signal outputs during user and ICD memory operations.
	Off -DEN signal outputs during user memory access only.
Bit 4	On - ALE signal outputs during emulation.
	Off - ALE signal outputs in all states.
Bit 5	On - DT/R signal outputs during emulation.
	Off - DT/R signal outputs in all states.
Bit 6	On - DT/R signal set to low-level during a break.
	Off - DT/R signal set to high-level during a break.

Bit 7	On - User TEST signal connected to ICD TEST signal.
	Off - User TEST signal disconnected with ICD TEST signal.
Bit 8	Off/On - Don't care bit (not connected).

Emulation Method Select Switch #2

Description	The Emulation Method Select switch #2 is an 8-bit, On/Off type switch that controls signal I/O between the ICD and target system during emulation.
Location	The NDP/CPU connector end of the ICD. Switch #2 is located above switch #1. (See "The Controls And Component Functions Of Your ICD," earlier in this section.)
Function	See the individual bit settings that follow.
Adjustment	To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.
Factory Settings	All bits are Off.
Bits 1, 2, 3, 4, 5	On/Off - Don't care bits (not connected).
Bit 6	On - ARDY signal effective during both user and ICD memory access.
	Off - ARDY signal effective during user memory access only.
Bit 7	On - SRDY and READY signals effective during both user and ICD memory access.
	Off - SRDY and READY signals effective during user memory access only.
Bit 8	On - Automatic 2 clock wait state inserted into each machine cycle.
	Off - Wait state suppressed.

Emulation Method Select Switch #3

Description	The Emulation Method Select switch #3 is an 8-bit, On/Off type switch that controls signal I/O between the ICD's CPU and NDP processor and the target system during emulation.
Location	In the central section of the CPU Control module. NOTE: This switch is not externally accessible. The ICD must be fully disassembled and the CPU Control module removed before this switch can be adjusted. To remove the CPU Control module and gain access to this switch, see "How To Disassemble Your ICD," in Section 3.
Function	See the individual bit settings that follow.
Adjustment	To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.
Factory Settings	Bit 1 = On Bit 2 = On Bit 3, 4, 5, 6 = Off Bit 7 = On Bit 8 = Off
Bit 1	On - NDP's BUSY line is tied to the CPU's $\overline{\text{TEST}}$ line. Off - NDP's BUSY line and CPU's $\overline{\text{TEST}}$ line function normally.
Bit 2	On - NDP's $\overline{\text{RQ/GT0}}$ line is tied to the CPU's HOLD:HLDA line. Off - NDP's $\overline{\text{RQ/GT0}}$ line and CPU's HOLD:HLDA line function normally.
Bit 3, 4, 5, 6	On/Off - Don't care (not connected)

Bit 7	On - NDP's READY line is tied to the CPU's SRDY line.
	Off - NDP's READY line and CPU's SRDY line function normally.
Bit 8	On - NDP's READY line is tied to the CPU's ARDY line.
	Off - NDP's READY line and CPU's ARDY line function normally.

ICD Commands

Program Control

GO - Starts program execution

BREAK - Stops program execution on a variety of different parameters

EVENT - Signals an event in the program, triggers the trace feature, or sends out an external signal at a point in the program

HISTORY - Records program execution in real time, and then displays it in either machine or disassembled format

TRACE - Displays program execution in non-real time

NEXT - Displays "n" instruction lines as executed in non-real time

OFFSET - Sets an offset in the emulator for relative program addressing

Memory Control

ASSEMBLE - Converts the mnemonics entered from the keyboard to machine language in memory

DISASSEMBLE - Converts the memory contents to assembly language mnemonics

DUMP - Displays the memory contents in hexadecimal/ASCII format

ALLOCATION - Transfers memory from the ICD program memory to the CPU's memory space

COMPARE - Compares the memory contents and displays the non-matching data

MOVE - Moves the memory contents between the ICD and the target system

EXAMINE - Examines and changes the memory contents

FILL - Fills the memory contents with data

SEARCH - Searches the memory contents for either matched or unmatched data

Debug/Emulation Control

REGISTER - Displays or changes the registers' data

SUPERVISOR - A "system call" to allow access to the serial input/ output ports

PRINT - Sends the display to a printer

PRINT - Resets the target system via the ICD

PIN - Enables or disables selected input signals

PORT - Examines one or more I/O port locations and optionally modifies them

IDENTIFICATION - Identifies the type of emulator in use and the firmware version

IN-CIRCUIT - Sets the ICD mapping mode

USER - Allows one terminal to communicate with both the ICD and a host computer

MAP - Sets the ICD/target system memory map

CALCULATION - Performs subtraction, addition, and conversion of hex and decimal data.

Host & File Handling Commands

LOAD - Loads an Intel Hex file from the host computer to the ICD memory

SAVE - Saves an Intel Hex file to the host computer

VERIFY - Checks a file in the host computer against a file in the ICD

HOST - Initiates or terminates LOCAL "Host Computer Assisted" mode

Introduction

ZAX ICD-series emulators respond to commands which you enter from a console terminal or host computer. The commands enable the ICD to perform a variety of complex debugging tasks for you. In this section, you'll learn how to use the debugger commands and how to perform actual debugging and development operations.

In order to use the commands effectively, you'll need to become familiar with three different areas:

* The language needed to implement the commands

* What each command does

* How to use the commands to perform debugging or development operations

Command Language

All ZAX ICD-series emulators execute operations in response to "command statements" made up of the "command name" and "parameters." The command name refers to a symbol or group of symbols that designate the basic emulation operation to be performed (e.g., G for GO, MA for MAP, T for TRACE, etc.). Parameters refer to any additional information that complements the command name, such as a specific address, an address range, or a base value. Together, the command name and the parameters can be combined to execute a variety of complex debugging operations.

The control firmware within the ICD requires that the command statements be entered in a concise and logical manner, and that all required elements of the command statement be used. The elements of the command statement are described below. The elements shown here represent all possible items within a command statement. Of course, not all commands require the presence or absence of each element.

Elements Within A Command Statement

The Prompt Character. The prompt character lets you know that the ICD is ready to accept a command statement. The prompt character is supplied by the ICD - you do not enter it - and is always displayed on the left side of the console's screen.

Example of prompt character: >

The Command Name. Commands are represented by the first, or first two letters of the command name. The commands are displayed by upper-case typeface and should be entered using capital letters.

Examples of command names: B (for BREAK), CO (for COMPARE), SA (for SAVE).

Command Qualifiers. The slash key (/) acts to signal a qualifier for the command whenever it appears immediately following the command mnemonic.

Examples of qualifiers: B/0 B/E F/W

The Space Character. The space character is an invisible character that not only improves the readability of a sentence, but in the case of the command format, it is recognized as a delimiter for the command name. Spaces must be interpreted from the command format; there is no symbol used to indicate spacing.

Example of space character in use: EV ON

In this example, the space between EV and ON allows the ICD to interpret EV as the EVENT command, and ON as a directive to enable the command.

Keywords are items which you must enter as shown. These items are displayed by upper-case typeface, but usually any combination of upper-case or lower-case letters may be used to enter them.

NOTE: Some terminals must use upper-case letters only. If the ICD responds with an error message, try using upper-case letters.

Examples of keywords: UP EN LO ON OFF

User-Supplied Items. Lower-case letters in *italic typeface* show items which you may supply; these are called user-supplied items.

Examples of user-supplied items include the name of your file (TEST.H86), a beginning address (0), an ending address (3FF), a comparison address (100), and data (55).

Address And Data Parameters. The common numerical parameters for the ICD commands are described below:

addr, beg_addr, comp_addr, mov_addr, stop_addr, search_addr = hexadecimal numbers in 16 bits (0-FFFF). These parameters specify a memory address with 16-bit hexadecimal characters. These parameters can be specified in an addition or subtraction equation, or a bias can be added if offset registers (0, 1, 2 or 3) are provided.

"Don't care" conditions may be specified for the BREAK and EVENT commands, on a bit or nibble basis, by entering "X" at the desired position. Examples include:

1A3X - Don't care condition in hexadecimal notation. May be specified in 4-bit units (0-F or X).

101X_X1XX_010X_1XX0 - Don't care condition in binary notation. May be specified in 1-bit units (0, 1 or X).

end_addr = hexadecimal numbers in 16 bits (0-FFFF), or number of bytes in 16 bits (0-FFFF).

NOTE: The byte format is; Lnnnn where nnnn = (0-FFFF).

data, mod_data, and search_data = hexadecimal/binarynumber in 8/16 bits (0-FFFF). These parameters canbe specified in an addition or subtraction equation, but the offset registers cannot be used.

"Don't care" conditions may be specified for the EVENT command, on a bit or nibble basis, by entering "X" at the desired position. Examples include:

7X - Don't care condition in hexadecimal notation. May be specified in 4-bit units (0-F or X).

01XX-X001 - Don't care condition in binary notation. May be specified in 1-bit units (0, 1 or X).

The Equal Sign. The equal sign (=) causes the value or information on its right to assume a relationship with the value on its left.

Example of the equal sign: P 100=55

In this example, the ICD does not display anything in response to this entry, but the value entered on the right (which represents a data value of 55H) is now assigned a relationship with the value on the left (an address value of 100H).

The Comma Character. The comma character (,) is used to separate parameters when more than one parameter is required to form a command statement.

Example of the comma character: DI 0,100

NOTE: A space may be substituted for a comma (e.g., DI 0,100 =DI 0 100), but a space cannot be used where a comma acts as the separator (e.g., DI 0,100).

Brackets. Items in square brackets ([]) are optional. If you choose to include the information, you should not enter the brackets, only the information inside the brackets.

Examples of brackets: [D=data] [,bias]

The Return Key. The return key is used to terminate statements and execute commands, and it must be entered after every statement. It is assumed that the return key must be pressed after the command statement is entered; there is no symbol used to indicate the return key in the command format.

NOTE: Other parameters are defined and explained in each command. See Terms and Notes for an explanation about these parameters.

Example Of The Command Format

Each command is presented in the same format as shown below. This format makes it easy to find the name of a command and what it does, and then how to enter it correctly. An example (sometimes more than one) shows how the command is used in a debug/ development session.

The example below illustrates the DUMP command and includes many elements of a typical command statement. This command is also used as the syntax example in "How To Enter A Command."

Command

DUMP

Operation

Displays the memory contents in both hexadecimal and ASCII code.

Syntax

D[/W] ***beg_addr[,end_addr]***

Terms

W = Displays the memory contents in word units arranged in MSB/ LSB (most significant bit/least significant bit) order. The default is byte unit display.

beg_addr = beginning address of display.

end_addr = ending address of display.

Syntax Example

D/W 100,1FF
D 120

Notes

The *end_addr* is an optional parameter. If it is omitted, 16 bytes are displayed starting with *beg_addr*.

Command Example

See Syntax Example above. The first example shows that the memory contents are displayed in work units, beginning with address 100 and ending with address 1FF. The second example shows that the last 16 bytes are displayed beginning at address 120.

Explanations

1) The "Command Name" is always found at the top of the page. If a command performs more than one task, a description of the various command functions can be found after the command name, for example, "OFFSET: Specification" and "OFFSET: Status."

2) "Operation" describes the action of the command, and emulation practices and principles that involve the command.

3) "Syntax" shows the characters and elements that are needed to implement the command. However, the characters and elements in Syntax may not provide enough information in themselves to correctly enter the command (the parameters may only represent an address or data value). The information in Terms should then be used to define the parameters.

4) "Terms" describes the characters and elements used in Syntax. The lower-case characters in italic typeface show items which you must supply. Upper-case characters in bold typeface show what these items are and how they should be entered.

5) "Syntax Example" shows how the command might be entered using various characters and elements, and the correct spacing between them.

NOTE: If a command cannot be entered, or the ICD responds with an error message, try entering the example shown in Syntax Example.

6) "Notes" explains important facts about the command. It usually contains information about the parameters shown in Terms, or it may include an explanation of how the command is used in a debug/development application. Spacing describes the correct spacing of the elements of the syntax.

7) "Command Example" shows how the command might be used in an actual debug/development session.

How To Enter A Command

Before you can enter a command, you'll need to know what operation(s) the command performs. This information can be found in two different places: "ICD COMMANDS" in this section, and "Operation," found in the Command Format.

After selecting the command, examine the information in Syntax and Terms. Enter the parameters needed to perform the task you desire. Examine the Syntax Example to see the proper spacing and how the characters and elements are used. An example of this procedure is shown below using the DUMP command.

Command Example

The syntax for the DUMP command is:

D[/W] *beg_addr*[,end_addr]

The terms used in the syntax are:

W = Display the memory contents in word units (default is byte units).

beg_addr = beginning address of display.

end_addr = ending address of display.

Entering The Example Command

To use this command, first enter D (the mnemonic for DUMP). Now decide (after examining the definitions in Terms) if the memory contents should be displayed in word or byte units. Since W is in brackets, it represents an optional parameter (if it was omitted, the display would be in byte units). For this example, we'll use a word display and enter W, preceded by a slash and followed by a space. The first user-supplied item is the beginning address for the display (we'll supply the value of 100). The next item is an optional (because it's in brackets, []) ending address. In this example we'll specify 1FF for this parameter, preceded by a comma.

At this point, the display on the console's screen should look like:

>D/W 100,1FF

This input now forms a command statement, complete with the command mnemonic, usable parameters, elements and proper spacing. To send the command statement to the ICD for execution, press the return key on your keyboard.

What To Do If You Make An Input Error

If you make an error when entering a command statement, merely backspace over the error (which cancels the character) and enter the new information. You can also press the Delete (Del) key, which not only cancels out the error, but displays the canceled character as well*.

If you've already entered a command statement into the ICD but you meant something else, press Ctrl-U (Control-U)*, then just re-enter the correct command statement and the ICD will execute the latest command.

** NOTE: These features are available in the LOCAL mode only (i.e., when a console terminal is used to control the ICD directly).*

Error Messages

If you enter a parameter incorrectly, use an invalid address, or forget to use a space at the appropriate place, the ICD will respond with an error message. The error messages and causes are shown below and on the back of the fold-out Command Reference Guide.

Error Message	Displayed when
C?>	an unrecognizable command is entered
P?>	a parameter code error occurs
/?>	a modifier code error occurs
** Break Busy	the break specification exceeds the limit
** Unable Soft Break	a software break is set at the address presently not mapped in RAM
** Multi Break Address	a software break is set at the same address
** Input Error	an input error occurs
** Check Sum Error	a check sum error occurs
** File Name Error	a parameter code error occurs with the LOAD or VERIFY commands
** Not Local Mode	a LOCAL mode command is used when the system is in the REMOTE mode
** Not Remote Mode	a REMOTE mode command is used when the system is in the LOCAL mode
** Memory Write Error at ####	there is a memory modification error
** I/O Timeout Error at ####	a timeout error occurs at a specific address
** Memory Timeout Error at ####	memory or I/O in the target system does not respond to an ICD access
** Memory Guarded Access Error at ####	when a user program attempts to access an area mapped as NO memory
** Software Break Instruction Misrecovered at ####	an error has occurred while attempting to replace original contents of a software break location

NOTE: #s refer to address locations in the program.

Command	ALLOCATION: Status
Operation	Displays a logical block or sequence of blocks by the address range and by the corresponding beginning physical block number for the block series.
Syntax	**AL**
Command Example	See the "ALLOCATION: Specification" command.

Command ALLOCATION: Specification

Operation Allocates any 1K-byte block from the ICD program memory to any address space in the CPU's memory space.

Syntax **AL *beg_addr*[,*end_addr*]=*block_no***

Terms *beg_addr* = Transposes the supplied address to the first 1K-block address space where the address resides.

end_addr = Transposes the supplied address to the last 1K-block address space where the address resides.

block_no = The allocation beginning block number (range = 0 to 3FFH).

Syntax Example AL 80000,803FF=3F

Notes Ending Address. If the *end_addr* exceeds the first 1K-block specification, the end of the entire block in which the address resides will be assigned as the ending address. For example:

>AL 0,400=5 <— 400 exceeds first 1K-block specification. End of second 1K block now assumes position as the ending address. Allocation status will then show:

```
>AL
00000-007FF = 005 <—      first allocation block is now
00800-FFFFF = 002         actually 2K long
>
```

Block Number. Each 1K block of memory that is available in the ICD is assigned a sequential block number beginning with 0; these physical memory blocks may be allocated to any logical block address. The block number parameter represents the beginning of the block, but if the allocation beginning and ending address parameters define more than one logical block, sequential block numbers will be assigned to each subsequent logical block. It is possible to assign more than one logical block address to a single physical memory block, so beware.

The block number range depends on the amount of emulation memory in the ICD:

128K memory: range = 0 to 7FH
256K memory: range = 0 to 0FFH
384K memory: range = 0 to 17FH
512K memory: range = 0 to 1FFH
1M memory: range = 0 to 3FFH (maximum)

Spacing: A space is required between AL and [*beg_addr*]. Spaces are not permitted where commas are used to separate the parameters.

Command Example

Press the RESET switch on the ICD to initialize the allocation block number to 0, then enter:

```
>AL <-------------------------  displays the current allocation status
00000-FFFFF = 000 <-----------  shows all memory to be un—
>                              allocated
>AL 0,3FF=5 <-----------------  assigns the first 1K memory space
>                              to block #5
>AL <-------------------------  displays new allocation status
00000-003FF = 005 <-----------  shows block #5's memory
00400-FFFFF = 001              assignment
>AL 400,8FF=12 <--------------  assigns next 4FFH memory space
>                              to block #12
>AL <-------------------------  displays new allocation status
00000-003FF = 005
00400-00BFF = 012 <-----------  notice how memory range
00C00-FFFFF = 003              includes next 1K block as well
>
```

Command ASSEMBLE

Operation Translates simple-to-understand mnemonic instructions into machine language. The opposite translation (machine language to assembly language mnemonics) is accomplished using the DISASSEMBLE command.

Applications Note: The In-line Assembler in the ICD is a powerful software tool that can be used for writing patches into program code that has either been downloaded from a host computer or originated in the target system. This feature also allows you to quickly write your own routines, develop small programs, etc.

Syntax

A *mem_addr* <cr>
xxxx:xxxx (80186/80188 assembly instruction) <cr>
xxxx:xxxx <cr>

Terms

mem_addr = The beginning memory address where assembled code is stored. The logical address assumes the current code segment is used unless otherwise specified.

xxxx:xxxx = The next storage location.

80186/80188 assembly instruction = The mnemonic instruction to be assembled and stored. Operand may include number or .sym (.sym value must be pre-defined).

<cr> = Exits the assemble mode.

Syntax Example >A 100

Notes All number operands are assumed to be decimal unless specified as hexadecimal.

Spacing: A space is required between A and *mem_addr*. A space is required between opcode and operand of mnemonic instruction (no tab).

Command Example

```
Execute this sequence:
>A 0:0   <———————— starts assembling the
0000:0000 MOV BX,1000H   program into address 0
0000:0003 MOV AL,0
0000:0005 MOV [BX],AL
0000:0007 INC BX
0000:0008 INC AL
0000:000A JNZ 5H
0000:000C HLT
0000:000D <———————— press the return key here to
>                end the program input
>DI 0:0,000C <— displays the program just entered
```

Command BREAK

Introduction

The best way to safely stop a moving car is to use the brakes. In emulation, the best way to stop a program for examination is to use BREAKpoints. You can use the BREAK commands to set breakpoints anywhere within a program, and you can specify many different types of breaks to stop program execution. Breakpoints differ from event points (see the EVENT command) in that they actually cause the program to stop execution; event points are used to trigger various external events, including stopping execution, without necessarily affecting the emulation process.

Software breakpoints replace program instructions automatically with monitor calls, in order to stop program execution at a particular point in the program. This provides real-time operation until the break. Several software breakpoints can be set throughout the program and selectively enabled and disabled. Also, an unlimited number of user breakpoints can be assembled into the code throughout the program.

The ICD can also implement hardware breakpoints, which recognize machine cycles but do not disturb normal software execution. Hardware breakpoints can cause the ICD hardware to monitor the address and status signals for a specified condition. When the conditions are met, a break occurs.

Both hardware and software breakpoints can be activated (enabled), and then temporarily deactivated (disabled), without affecting their location addresses within the program or their parameter specifications.

Another break feature allows the ICD to use a probe to receive a signal from a peripheral, which can then cause a break in the program. (See "More About Your ICD," in Section 1.)

There are 15 different BREAK command formats. See each format for an explanation and example.

Command BREAK: Status

Operation Displays the current status of the break command. Use this command to check the condition of the breakpoint settings.

Syntax **B**

Command Example This command example shows what the break status might reveal after several break parameters are defined:

```
                                        |pass count
                                        |  |elapsed count
                                        |  |  |INDependent of or ARMed by event
                                        |  |  |  |processor access
                                        |  |  |  |  |bit-wise physical address
>B                                      |  |  |  |  |
A (ON)      EX 0040:0600 1              0 IND BOTH (0000_0000_1010_0000_0000)
B (OFF)     MR 003XX/DS                 1        0 ARM NDP
(0000_0000_0011_0XXX_XXX0)
0 (ON)        0040:0213                 1        0
1 (ON)        01000                     1        0
2 (0N)        01FFE                     1        0
S (DI)      HLT   |
X (ON)      HI    |                     1        0
E (OFF)     |     |                     1        0
T (ON)      |     |
W (ON)      |     |
>   |       |     |     |address
|   |       |     |break operation
|   |break status
|break identification
```

NOTE: A,B,C=hardware break names, 0,1,2=software break names (up to 8 names), S=software break opcode, X=external break at high edge of signal, E=event break, T=ready time-out break, W=write-protect break.

Command BREAK: Hardware Breakpoint Qualification

Operation Enables, disables, or clears the settings of the hardware breakpoints.

Applications Note: This command can be used to temporarily disable pre-set hardware breakpoints without affecting their locations within the program or their parameter specifications.

Syntax **B[*/name*] *switch***

Terms *name* = A, B, or C

switch = ON, OFF, or CLR

Syntax Example B/A ON

B OFF

Notes A, B, or C identifies hardware breakpoint names, and more than one name can be specified at a time (e.g., B/A/C CLR). If the breakpoint name is omitted, all hardware and software breakpoints are affected.

ON enables the breakpoint(s), OFF disables the breakpoint(s), and CLR clears the break condition.

Hardware breakpoints automatically default to "ON" after they are specified by the "BREAK: Hardware Breakpoint Specification" command.

Spacing: A space is required between *name* and *switch*. If *name* is omitted, a space is required between B and *switch*.

Command Example See Syntax Example and the "BREAK: Hardware Breakpoint Specification" command.

Command BREAK: Hardware Breakpoint Specification

Operation Sets a hardware breakpoint within the user program. Setting a hardware break configures the emulator hardware to monitor the address and status signals for the specified condition to occur. When the conditions are met in the program, a break occurs.

Syntax **B[/*name*] *status*,*addr*[/*segment*][,*passcount*]**

Terms *name* = A, B, or C

status = Any one of eight types of break status, including:

M (memory access)
MR (memory read)
MW (memory write)
P (port access)
PR (port read)
PW (port write)
OF (operation code fetch)
IA (interrupt acknowledge)
EX (command execution)

addr = The address to break on.

segment = Any one of four segments for the address, including:

CS (code segment)
DS (data segment)
SS (stack segment)
ES (extra segment)

passcount = The number of times the condition occurs before breaking, from 1 to 65535.

Syntax Example B/C M,1111_001X_0011_XX10_110X

Notes A, B, or C identifies hardware breakpoint names.

If *name* is omitted, the next available breakpoint is used; if all the breakpoints are in use, an error message will be displayed.

The *addr* can be specified by a binary or hexadecimal notation. To specify a "don't care" condition in 1-bit units (binary notation), or in 4-bit units (hexadecimal notation), write X at the required position. If passcount is specified, real-time operation is momentarily lost each time the condition occurs. If the passcount specification is omitted, 1 is assumed.

Spacing: A space is required between name and status. If name is omitted, a space is required between B and status. Spaces are not permitted wherecommas are used to separate the parameters.

Command Example

Execute this sequence:

```
>B/B OF,200 <— specifies hardware breakpoint
>B     <— checks breakpoint status
B (ON)  OF 00200                1        0 IND BOTH (0000_000...
S (DI)  HLT
E (OFF)                         1        0
T (ON)
W (ON)
>B/B OFF  <— disables hardware breakpoint B
>B  <— checks the breakpoint status again
B (OFF)  OF 00200               1        0 IND BOTH (0000_000...
E (OFF)                         1        0
T (ON)
W (ON)
>
```

This example shows that a hardware breakpoint is placed at address 200 in the program and that thestatus to break on is an opcode fetch. The "BREAK: Status" command is then used to verify the breakpoint setting. Next, the breakpoint is temporarily disabled using the B/B OFF command. Again, the "BREAK: Status" command is used to verify the change.

Command BREAK: Event then Hardware Breakpoint

Operation Causes a break in the program at a hardware breakpoint (A, B, or C), but only after an event point is also passed (see EVENT command). The arm feature creates a simple level of sequencing: A-then-B relationship.

Applications Note: This command can be used to trigger a peripheral device (such as a logic analyzer) when an event point is passed in the program. The program then stops when a breakpoint is encountered.

Syntax **B[/*name*] *switch***

Terms *name* = A, B, or C

switch = ARM or IND

Syntax Example

B/C ARM
B IND

Notes A, B, or C identifies hardware breakpoint names, and more than one name can be specified (e.g.,B/A/C IND). If the breakpoint name is omitted, all three hardware breakpoints are affected.

If ARM is selected, the break occurs after an event trigger takes place. If IND is selected, the break occurs independently of any event trigger.

The ARMing event is not automatically reset. See the "BREAK: ARM Initialize" command.

Spacing: A space is required between *name* and *switch*. If *name* is omitted, a space is required between B and *switch*.

Command Example See Syntax Example.

Command BREAK: ARM Initialize

Operation Clears (initializes) the event pass condition and resets the ARM specification of the "BREAK: Event then Hardware Breakpoint" command.

Syntax **B INI**

Notes Once the ARMing event has occurred, the condition will remain ARMed until cleared by this command.

Spacing: A space is required between B and INI.

Command Example

Execute this sequence:

```
>A 0:0
0000:0000 MOV BX,1000H
0000:0003 MOV AL,0
0000:0005 MOV [BX],AL
0000:0007 INC BX
0000:0008 INC AL
0000:000A JNZ 5H
0000:000C HLT
0000:000D
>EV ST=MW,A=1012 <— sets event to trigger on memory write to location
1012H
>B/A EX,8 <— sets breakpoint A to break on execution of location 8H
>B/A ARM <— arms breakpoint A by event trigger
>B INI <— clears (initializes) event's arming condition
>G 0:0 <— begins execution

00008                          FECO              INC
  CS  IP ODITSZAPC AXBXCX  DXSP
0000:0005 000000010 0014 1014 0000 0000 0000 ...
<Break  Hardware A>
>
>G 0:0

00008                          FECO              INC
  CS  IP ODITSZAPC AX  BX  CX  DX  SP
0000:0005 000000010 0001 1001 0000 0000 0000 ...
<Break  Hardware A>
>B INI  <— clears event's arming condition again
>G 0:0
>
00008                          FECO              INC
  CS  IP ODITSZAPC AX      BX      CX  DXSP
0000:0005 000000010 0014 1014 0000 0000 0000 ...
<Break Hardware A>
```

Command BREAK: Software Breakpoint Specification

Operation Sets a software breakpoint within the user program.

Setting a software breakpoint causes the ICD to automatically replace the opcode at the specified address with a HLT or INT3 instruction opcode (see the "BREAK: Software/User Breakpoint Code" command). When this code is encountered during execution, a temporary break will occur, the original contents of this location will be replaced, and execution will restart at that same location for the duration of that one instruction. The ICD will then enter the monitor mode.

Setting a software breakpoint is a two-step process requiring both Specification and Recognition commands (see the "BREAK: Software Breakpoint Recognition" command).

Syntax **B[/*name*] *addr*[,*passcount*]**

Terms *name* = 0, 1, 2, 3, 4, 5, 6, or 7

addr = The address to break on.

passcount = The number of occurrences before a break, from 1 to 65535.

Syntax Example

B/4 100,3
B/7 1000

Notes 0, 1, 2, ... or 7 identifies software breakpoint names.

If *name* is omitted, the first available break name is used; if all available breakpoints are in use, an error message will be displayed.

For software breakpoints, the *addr* is specified by a hexadecimal notation and must be a single, specific address.

If *passcount* is specified, real-time operation is momentarily lost each time the condition occurs. If the passcount specification is omitted, 1 is assumed.

A software breakpoint cannot be specified in a USER-ROM area since the breakpoint requires changing the memory contents (at the specified location) to a HLT or INT3 instruction, and ROM cannot be changed. A hardware breakpoint must be used in this situation.

A software breakpoint must be specified for a location containing the first byte of an opcode; otherwise, the ICD will not break, and unpredictable results will occur within the program execution.

The monitor call is automatically placed at the specified locations when the program code is executed, but the program display will only show the original contents at that location. Anything that causes the contents of a location to be changed during program execution destroys the monitor call instruction.

Spacing: A space is required between *name* and *addr*. If *name* is omitted, a space is required between B and *addr*.

Command Example

Execute this sequence:

```
>B/5 1000 <— sets software breakpoint at addr 1000
>B S=EN <— enables the software breakpoints
>B  <— checks the status of the breakpoints
5 (ON)  OF 01000    1   0 <— shows that software
S (EN)  HLT                 breakpoint #5 is
E (OFF)          1   0    active at addr 1000
T (ON)
(ON)
>
```

This example shows that a software breakpoint labeled 5 is set at address 1000 in the program. The software breakpoint is enabled (software breakpoints must be enabled to function), and then the "BREAK: Status" command is used to verify the change.

Command BREAK: Software Breakpoint Recognition

Operation Enables or disables all software and user breakpoints. Setting a software breakpoint is a two-step operation requiring the software and user breakpoint to be enabled before any software breakpoints become operational.

Syntax **B S=*switch***

Terms *switch* = EN or DI

Syntax Example B S=EN

Notes EN enables the software and user breakpoints, causing a break in the program based on the software breakpoint specification or when a user break is encountered. DI disables the software and user breakpoints, causing them to be temporarily disabled, although their initial specification remains unaffected.

The ICD defaults to DI upon power-up or reset.

Spacing: A space is required between B and S. No spaces are permitted after S; the equal sign acts as the separator.

Command Example See Syntax Example and the "BREAK: Software Breakpoint Specification" command.

Command BREAK: Software/User Breakpoint Code

Operation Specifies which code the ICD uses to implement asoftware or user break.

Applications Note: The ICD can use HLT (0F4H), INT3 (0CCH), or any code from 0 to 0FFH (if specified by its correct hexadecimal code) to cause a software break within the user program. This allows you to conveniently cause a program break without continuously specifying breakpoint parameters.

Syntax **B S=*op_code***

Terms *op_code* = HLT, INT3, or any hexadecimal code

Syntax Example B S=INT3

Notes The ICD defaults to HLT upon power-up or reset.

Spacing: A space is required between B and S. No spaces are permitted after S; the equal sign acts as the separator.

Command Example Command

Execute this sequence:

```
>B     <—— checks the breakpoint status
S (DI) HLT <———— shows software break code is
E (OFF)    1    0  currently HLT
T (ON)
W (ON)
>B S=INT3 <—— changes software break code to INT3
>B S=EN <— enables all software breakpoints
>B     <—— checks the breakpoint status again
S (EN) INT3 <—— shows the software break code
E (OFF)    1    0
T (ON)
W (ON)
>
```

This example shows how the software break code is changed from HLT to INT3 and then enabled. The "BREAK: Status" command verifies the change.

BREAK: Software Breakpoint Qualification

Operation

Enables, disables, or clears the software breakpoints.

Applications Note: This command can be used to temporarily disable pre-set software breakpoints without affecting their address locations within the program or their parameter specifications.

Syntax

B[*/name*] *switch*

Terms

name = 0, 1, 2, 3, 4, 5, 6, or 7

switch = ON, OFF, or CLR

Syntax Example

B/3 ON
B OFF

Notes

0, 1, 2, ... or 7 identifies software breakpoint names, and more than one name can be specified ata time (e.g., B/1/2/3/4 OFF). If the breakpoint name is omitted, all the hardware and software breakpoints are affected.

ON enables the breakpoint, OFF disables the breakpoint, and CLR clears the break condition.

Spacing: A space is required between *name* and *switch*. No spaces are permitted between B/*name*.

Command Example

Execute this sequence:

```
>B <——— checks the breakpoint status
S (DI) HLT
E (OFF) 1   0
T (ON)
W (ON)
>B/2 7FF <— sets a software breakpoint at addr 7FF
>B S=EN <—— enables the software breakpoints
>B  <—— checks the breakpoint status again
2 (ON)  007FF <——— 1 — 0 - shows that soft—
S (EN) HLT                ware breakpoint #2 E (OFF)      1   0    is active
at addr
T (ON)          7FF
W (ON)
>B/2 OFF <—— disables software breakpoint #2
B <———— checks the status again
2 (OFF)  007FF <———— 1— 0 - shows software
S (EN)  HLT               breakpoint #2 is
E (OFF) 1   0    inactive
T (ON)
W (ON)
```

This command shows how a software breakpoint is set, enabled, and then disabled. After each operation, the status of the breakpoints is checked against the changes.

Command BREAK: Processor Access

Operation Specifies which processor access causes a hardware break in the user program.

Syntax **B/*name processor***

Terms *name* = A, B, or C

processor = CPU, NDP, DMA, or ANY

Syntax Example B/B NDP

Notes A, B, or C identifies hardware breakpoint names, and more than one name can be specified at a time (e.g., B/A/B/C CPU).

CPU means that accessing the main (80186/80188) processor causes a break; NDP means that accessing the Numeric Data (8087) Processor causes a break; DMA means that making a Direct Memory Access causes a break; and ANY means accessing either CPU, NDP, or DMA causes a break.

Spacing: A space is required between *name* and *processor*. No spaces are permitted between B/*name*.

Command Example

Execute this sequence:

Press the RESET switch on the ICD, then enter:

```
>B <——— checks the breakpoint status
S (DI) HLT
E (OFF)                 1       0
T (ON)
W (ON)
>B/A OF,1FF <— sets a hardware breakpoint
>B <————— shows the revised breakpoint status
A (ON) OF 001FF         1       0 IND BOTH (0000...
S (DI) HLT
E (OFF)                 1       0       T (ON)
W (ON)
>B/A NDP <—— changes BOTH specification to NDP
 >B <————— confirms the change
A (ON) OF 001FF         1       0 IND NDP (0000...
S (DI) HLT
E (OFF)                 1       0
T (ON)
W (ON)
```

Command BREAK: External Signal Qualification

Operation Allows the ICD to sense a signal (using the accessory probes) from an external source and cause a break in the user program. This command specifies how the break is triggered, from either the high-going or low-going edge of the external signal. To enable or disable this command, see the "BREAK: External BreakpointQualification" command.

Syntax **B/X *edge*[,*passcount*]**

Terms *edge* = HI or LO

passcount = The number of occurrences before a break, from 1 to 65535.

Syntax Example B/X LO

Notes HI causes the breakpoint to occur on the rising edge of the signal; LO causes the breakpoint to occur on the falling edge of the signal.

When edge is specified, the parameters for the "BREAK: External Breakpoint Qualification" command become effective.

If *passcount* is specified, real-time operation is momentarily lost each time the condition occurs. If the passcount specification is omitted, 1 is assumed.

Spacing: A space is required between B/X and edge. No spaces are permitted between B/X.

Command Example See the "BREAK: External Breakpoint Qualification" command.

Command BREAK: External Breakpoint Qualification

Operation Allows the ICD to sense a signal (using the accessory probes) from an external source and trigger a break in the user program during emulation. This command enables, disables, or clears that feature. (For more information on how to use the accessory probes, see "More About Your ICD," in Section 1.)

Syntax **B/X *switch***

Terms switch = ON, OFF, or CLR

Syntax Example B/X CLR

Notes ON enables the recognition of an external trigger, OFF disables the recognition of an external trigger, and CLR clears the external trigger specification.

Spacing: A space is required between B/X and switch. No spaces are permitted between B/X.

Command Example

Execute this sequence:

```
>B <——— checks the breakpoint status
S (DI) HLT
E (OFF)          1   0
T (ON)
W (ON)
>B/X HI <———— sets signal recognition to high
>B            edge of signal
S (DI) HLT
X (ON) HI <————————— 1 — 0 - shows external
E (OFF)          1   0  break feature is
T (ON)                  active
W (ON)
>B/X OFF <— disables external break feature
>B <— checks breakpoint status again
>B
S (DI) HLT
X (OFF) HI <————————— 1 — 0 - shows external
E (OFF)          1   0  break feature is
T (ON)                  inactive
W (ON)
>B/X CLR <— clears the external breakpoint feature
>B <———————— verifies the change
S (DI) HLT
E (OFF)          1   0
T (ON)
W (ON)
>
```

This example shows how the external breakpoint specification is set to occur at the high edge of an external signal. The external breakpoint is then temporarily disabled and finally cleared.

Command BREAK: Event Breakpoint

Operation Allows the ICD to use an event trigger as a breakpoint (see the EVENT command). This command enables or disables the event break feature but does not affect the event point specification in any way.

Syntax **B/E *switch***

Terms *switch* = ON or OFF

Syntax Example B/E OFF

Notes ON enables the event breakpoint and OFF disables the event breakpoint.

Spacing: A space is required between B/E and *switch*. No spaces are permitted between B/E.

Command Example

Execute this sequence:

```
>EV <------- displays event status
Event is Clear <— shows absence of event points
>EV ST=OF A=7FF <— sets an event point in program
>EV <— displays new event point setting
(ON)
Status  = OF
Address = 007FF  (0000_0000_0111_1111_1111)
>B/E ON <—     enables the event point to cause a break
>B             in execution
S (DI)  HLT
E (ON) <---------- 1— 0 —     shows event point
T (ON)                        setting is active
W (ON)
>
```

This example shows how an event in the program can be used to send out a signal to a peripheral device. First, the event point is set in the program at address 7FF, and then the status command is used to verify the setting. Next, the event break point is enabled by using a breakpoint command. The "BREAK: Status" command is used again to verify that the event point is enabled.

Command BREAK: Write Protect Breakpoint

Operation Causes a break in the user program if the program attempts to write into a protected memory area (see the MAP command). After the break, the ICD responds with an error message that reads: Break Write Protect.

If this break is disabled, any attempt to write to a protected memory location will fail, thereby preserving its integrity; however, program execution will continue without causing a break.

Syntax B/W *switch*

Terms switch = ON or OFF

Syntax Example B/W ON

Notes ON enables the write protect feature and OFF disables the write protect feature. (This feature is automatically activated when the ICD boots up.)

Spacing: A space is required between B/W and switch. No spaces are permitted between B/W.

Command Example

Execute this sequence:

```
>MA 0,FFF=RO <————————  sets memory as read-only
>MA                     from address 0 to FFF
In-Circuit Mode 0 (US=>RW)
00000-00FFF = RO (000) <——  shows status of memory
01000-FFFFF = RW (004)      is read-only from addr
>                  0 to FFF
>B/W ON <—— enables the write protect feature
>B
S (DI) HLT
E (OFF)        1   0
T (ON)
W (ON) <— shows write protect feature is active
>
```

This example shows how the write protect feature might be used. First, memory within the ICD is mapped from 0 to FFF as read-only. Because the in-circuit status is I 0 (debugging using the ICD's memory only), any area mapped as user (target system) memory is now re-mapped as read/write memory in the ICD; this causes the remaining memory areas (1000 - FFFFF) to act as read/write memory. The write protect feature is then enabled using the "BREAK: Write Protect Breakpoint" command. Finally, the break status is checked to verify the changes. The ICD now causes a break if an attempt is made to write into memory locations 0 to FFF.

Command BREAK: Timeout Breakpoint

Operation Causes a break in the user program when the ICD is unable to access the target memory contents within a certain time period (128 clock cycles). If the READY signal is negated for more than 128 clock cycles, a time-out condition will occur. After the break, the ICD responds with an error message that reads: Break Timeout.

Applications Note: This break command can be used to flag a failure by the target system to re-assert a ready condition. The failure could be caused by a problem in the hardware, or it could be inherent in the design. If the problem lies in the design, the Timeout Breakpoint feature should be disabled; but if it is a hardware problem, disabling this feature could cause the ICD to "lock-up" due to a continuously negated ready condition.

This feature can also act as a safeguard for the target's refresh period if dynamic RAMs are being used.

Syntax B/T *switch*

Terms *switch* = ON or OFF

Syntax Example B/T OFF

Notes ON enables the timeout feature and OFF disables the timeout feature. (This timeout feature is automatically activated when the ICD boots up.)

Spacing: A space is required between B/T and switch. No spaces are permitted between B/T.

Command Example See Syntax Example.

Command CALCULATION

Operation Performs subtraction and addition of hexadecimal and/or decimal numbers, and performs hexadecimal-to-decimal or decimal-to-hexadecimal conversions. The results of the particular operation are displayed in both decimal and hexadecimal notation.

Syntax C *operand#1*[+/-*operand#2*][... +/-*operand#n*]

Terms *operand#1*,#2... #n = -2147483648 to 2147483647 (signed) or 0 to 4292967295 (unsigned); or 0 to 0FFFFFFFFH.

Syntax Example C 427+351-2FFH

Notes Both addition and subtraction may be performed on the same line.

Negative decimal results are displayed as "unsigned/signed."

Spacing: A space is required between C and *operand#1*. No spaces are permitted after *operand#1*.

Command Example

```
>C 283-350 <—— subtraction of decimal numbers
FFFFFFBDH
4294967229-67
>C 100 <———————— conversion of decimal number
00000064H
100
>C FFH+1FF50H <—— addition of hexadecimal numbers
0002004FH
131151
>C 429-2EH+8+FF3DH <——— mixed calculations
000100C4H
65732
```

Command COMPARE

Operation Compares the contents of specified memory blocks within the ICD or target system, and then displays the non-matching data. The comparison can be made between different memory blocks as mapped to the ICD, or between one block of memory within the ICD and one in the target system.

Syntax CO ***beg_addr,end_addr,comp_addr[,direction]***

Terms *beg_addr* = The beginning address for comparison.

end_addr = The ending address for comparison.

comp_addr = The beginning memory address to be compared.

direction = UP or PU.

Syntax Example CO 100,3FF,1000,UP

Notes If UP is selected, *beg_addr* is user memory, and *comp_addr* is ICD program memory. If PU is selected, *beg_addr* is ICD program memory and *comp_addr* is user memory.

If *direction* is omitted, memory locations are specified by the MAP command.

This command displays non-matching data on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required between CO and *beg_addr*. No spaces are permitted after *beg_addr*; commas are used to separate the remaining parameters.

Command Example See Syntax Example. This example shows that a memory block (100 to 3FF) in the target system is compared with a block of memory in the ICD, beginning at address 1000. Any unmatching data will be displayed, along with the location addresses.

Command DISASSEMBLE

Operation Translates the memory contents from machine language to assembly language mnemonics, and then displays the converted contents. The opposite translation (assembly language mnemonics to machine language) is accomplished by using the ASSEMBLE command.

Syntax **DI [*beg_addr*][,*end_addr*]**

Terms *beg_addr* = The beginning memory address in the program.

end_addr = The ending memory address in the program.

Syntax Example

```
DI 100,1A6
DI FFF
DI
DI ,L40
```

Notes If *beg_addr* is omitted, disassembly begins at the current program counter (PC). If *end_addr* is omitted, 11 lines of instructions are automatically displayed.

This command displays items on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required between DI and *beg_addr* (if *beg_addr* is used). Spaces are not permitted where commas are used to separate the parameters.

Command Example See Syntax Example. The first example shows that the memory contents in the ICD are disassembled beginning from address 100 to address 1A6. In the second example, the ending address is omitted, which causes 11 lines of the memory contents to be disassembled starting from address FFF. The third example illustrates that 11 instruction lines are displayed from the current PC. The fourth example displays the current PC to PC + 40.

Command DUMP

Operation Displays the memory contents in both hexadecimal and ASCII code.

Syntax **D[/W] *beg_addr*[,*end_addr*]**

Terms W = Displays the memory contents in word units arranged in MSB/LSB (Most Significant Bit/Least Significant Bit) order (the default is byte unit display).

beg_addr = Beginning address of display.

end_addr = Ending address of display.

Syntax Example

```
D/W 100,1FF
D 1FFF
```

Notes The *end_addr* is an optional parameter; if it is omitted, 16 bytes are displayed starting with beg_addr.

The 80186/80188 arranges word data in memory as: low address = LSB, high address = MSB; therefore, the /W option effectively swaps the two bytes of each word.

This command displays items on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required between D or D/W and *beg_addr*. Spaces are not permitted where commas are used to separate the parameters.

Command Example See Syntax Example. The first example shows that the memory contents are displayed in word units, beginning with address 100 and ending with address 1FF. The second example shows that the last 16 bytes are displayed beginning at address 1FFF.

Command EVENT

Introduction

An event can be defined as a significant occurrence in time. That is, events take their respected place at a point in time, without affecting the presence or passing of time itself. And of course, the ICD's EVENT command works on the same principle.

This command allows an event to occur during the execution of a program, without necessarily stopping the program. In this way, an event point differs from a breakpoint because breakpoints always stop program execution.

The EVENT command can enact four different operations: trigger a peripheral device, such as a logic analyzer; trigger the real-time trace feature (which is defined by the HISTORY command); arm a hardware breakpoint in an A-then-B type sequence; and stop the program in a manner similar to the BREAK command.

Unlike the BREAK command, the EVENT command has the advantage of allowing you to specify a certain data pattern on the data bus, in addition to the normal address parameters, memory access, and I/O access conditions.

Events point can be enabled and disabled, just like breakpoints. This feature allows you to temporarily disable the event setting without affecting its address location within the program or its parameter specifications.

Using The Event Command

There are three EVENT commands: Status, Qualification, and Specification. To see how to use an event point as a breakpoint, see the "EVENT: Specification" and "BREAK: Event Breakpoint" commands. To arm a hardware breakpoint, see the "BREAK: Event Then Hardware Break" command. To use an event point to trigger the real-time trace, see the HISTORY command. To use an event point to trigger a peripheral device, see "More About Your ICD," in Section 1.

Command EVENT: Status

Operation Displays the current event point specifications. When changes are made to the event point specifications by using the "EVENT: Specification" command, this command is used to display the latest changes.

Syntax EV

Command Example

```
>EV
Event is Clear
```

This is the default condition for the EVENT command.The display shows the absence of any event points in the program. After specifying an event point, the"EVENT: Status" command might reveal a display such as the one shown below:

```
>EV (ON)  Status= MW
Address  = F035:0000 (1111_0000_0101_1000_0000)
Segment  = CS
Mode     = CPU
Data     = 55      (0101_0101)
>
```

This status display shows that the EVENT command is enabled (ON), the status of the event point is a memory write (MW), the port is located at address F035:0000 (which is also represented by its physical address in bit-wise notation), the segment value is CS (code segment), the EVENT command will access the CPU only, and the data to match for the event is 55. When these conditions are satisfied, an event occurs.

Command	EVENT: Qualification
Operation	Enables, disables, or clears the event trigger feature, or initializes the event counter. Applications Note: This command can be used to temporarily disable an event point without affecting its location within the program or its parameter specifications. Use this command after setting an event point with the "EVENT: Specification" command.
Syntax	EV *switch*
Terms	switch = ON, OFF, CLR, or INI
Syntax Example	EV CLR
Notes	ON enables the event trigger recognition feature, OFF disables the event trigger recognition feature, and CLR clears the event setting. (ON is the default when an event is set using the "EVENT: Specification" command.) INI resets the event counter. Spacing: A space is required between EV and switch.
Example Command	See Syntax Example and the "EVENT: Specification" command.

Command EVENT: Specification

Operation Sets the conditions for an event point trigger.

Syntax **EV [ST=*status*][,A=*addr*][,D[/*mode*]=*data*][,M=*proc*][,CNT=*cnt*] [,LE=*edge*][,ED=*edge*]**

Terms

status = The type of cycle to trigger event on. This can be one of nine different names, including:

M (memory access)
P (port access)
MR (memory read)
MW (memory write)
PR (port read)
PW (port write)
OF (operation code fetch)
IA (interrupt acknowledge)
EX (command execution)
ANY (any operation)

addr = Specifies the address value to match for the event.

mode = B (byte) or W (word) data.

data = Specifies the data value to match for the event.

W = Qualifier to specify word data.

proc = CPU, NDP, DMA, or ANY

cnt = Specifies event counter, from 1 to 99,999.

LE = Specifies the HI-going or LO-going level of an external signal received via the accessory probes.

ED = Specifies the HI-going or LO-going edge of an external signal received via the accessory probes.

edge = HI, LO, or ANY

Syntax Example

EV ST=MR
EV A=1111_1111_0000_0XXX_XXX0
EV ST=MW,A=EFD04,D/B=55,M=NDP,CNT=100,LE=HI

Notes

All parameters for this command are optional, and all parameters not defined remain unchanged. Both addr and data may be specified as "don't care," in 1-bit units (binary) or in 4-bit units, (hex) by writing "X" at the required position. Also, any undefined parameter defaults as "don't care."

If data is specified as other than "don't care," the address parameter must have the lowest address bit (A0) defined as 0 or 1 (not as "don't care").

When specifying a P, PR or PW cycle for the event, and the port address is defined, the address should be defined as a 16-bit address, with the upper 8 bits defined as "don't care." (Example: port address 34 = XX34.)

If CPU is specified, event occurs on main processor access (80186/ 80188). If NDP is specified, event occurs on co-processor access (8087). If DMA is specified, DMA causes event. If BOTH is specified, event occurs on either processor access.

Spacing: A space is required between EV and the first parameter. Spaces are not permitted where commas are used to separate the parameters.

Command Example

```
EV A=FF0XX <- Specify an address ("X" on nibble basis)

EV A=0000_0011_1111_1111_1110 <- Specify an address ("X" on bit basis)

EV ST=MR <- Specify an event status
EV ST=MW,A=EFD05,N=NDP <- Set event using status, address, and mode

EV ST=PR,D=XX46 <- Set event for port read of specified data
```

Command EXAMINE

Operation Examines one or more memory locations and optionally modifies them. The locations can be displayed and changed with either ASCII or hexadecimal values.

Syntax **E[/W][/N] *beg_addr*[=*mod_data*]**

Terms W = Use the word mode (the default is the byte mode).

N = No-verify (the default is to read-verify after write).

beg_addr = Starting address for display.

mod_data = Modified (new) data for this location.

Syntax Example

```
E/W 100=5555
E FFE
```

Notes When /W option is selected, the word will be displayed or entered in LSB/MSB (Least Significant Bit/Most Significant Bit) order (bytes swapped).

If mod_data is omitted, the command enters a repeat mode, which allows several locations to be changed.

The repeat mode includes:

return (cr) to display the next byte (word) of data.
comma (,) to display the same byte (word) of data.
caret (^) to display previous byte (word) of data.
slash (/) to exit the EXAMINE command.

Spacing: A space is required before *beg_addr*. No spaces are permitted between *beg_addr* and *mod_data*; the equal sign acts as the separator.

Command
Command Example

```
>E 0

00000 FF=74,    <- change value to 74H; re-examine
00000 74=       <- leave value unchanged; go to next address
00001 BF=       <- leave value unchanged; go to next address
00002 BF='A'    <- change value; go to next address
00003 72=34^    <- change value; go to previous address
00002 41=^      <- leave value unchanged; go to previous address
00001 BF=,      <- leave value unchanged; re-examine address
00001 BF=^      <- leave value unchanged; go to previous address
00000 74=/      <- leaves value unchanged; exit command

>E/W 20
00020 A9BF=4455  <- change word value; re-examine
00020 4455=      <- leave value unchanged; go to next location
00022 FDB2='HI', <- change value (ASCII); re-examine
00022 4948=      <- leave value unchanged; go to next location
00024 CFED=      <- leave value unchanged; go to next location
00026 F7F5=^     <- leave value unchanged; go to previous location
00024 CFED=0/    <—change value; exit command

>E 30
00030 06=  <- examine only
00031 A0=
00032 00=
00033 64=
00034 0C=
00035 0E=/ <- exit command
>
```

Operation FILL

Syntax Fills a block of memory with either hexadecimal or ASCII codes.

Terms **F[/W][/N] *beg_addr,end_addr,data***

W = Fill memory contents of a word basis (the default is a byte basis).

N = No-verify (the default is to read-verify after write).

beg_addr = The block beginning address to be filled.

end_addr = The block ending address to be filled.

data = Data that fills the block.

Syntax Example

F 100,3FF,55
F/N 4000,4FFF,0
F/W DS:0,FF,3412

Notes

When /W option is selected, the word will be displayed or entered in LSB/MSB (Least Significant Bit/Most Significant Bit) order (bytes swapped).

Spacing: A space is required before beg_addr. No spaces are permitted where the commas act as separators.

Command Example

See Syntax Example. The first example shows how memory is filled from address 100 to address 3FF, with a data value of 55; the second example shows how memory is filled without verifying the write; and the third example shows how memory is filled on a word basis, including a data segment value.

Command GO

Operation Executes the user's program.

Syntax G [*beg_addr*][,*end_addr*][,*end_addr#2*]

Terms *beg_addr* = The address to begin execution.

end_addr = The last address to execute.

end_addr#2 = Optional second ending address.

Syntax Example

G
G 100
G 0,2FFE

Notes All parameters for this command are optional. If *beg_addr* is omitted, the program continues from the current program counter. If *end_addr* is omitted, the program continues until a breakpoint or a monitor break. When *end_addr#2*is specified, the first location reached by execution (*end_addr* or *end_addr#2*) will cause a break. Two hardware breakpoints must be available to activate the *end_addr* or *end_addr#2* parameters.

Spacing: A space is required between G and any additional parameters. Spaces are not permitted where commas are used to separate the parameters.

Example Command See Syntax Example. The first example starts the program from the current program counter; the second example starts the program from address 100; and the third example starts the program from 0 and stops it at address 2FFE.

Command

HISTORY (Real-time tracing)

Introduction

The real-time trace is one of the most powerful and useful features of your ICD. It allows you to record (hence the name "History" command) and then analyze a specific section of program execution, rather than sift through the entire program looking for a problem. Event points may be used to trigger the real-time trace buffer to start or stop the data storage process. Breakpoints may also be used to stop the storage process.

By using the various trigger modes, the real-time trace can effectively capture any single instruction or any set of instructions within a program. After the program stops, the ICD can display the address, data, and control bus of the latest series of machine cycles (in either machine cycle or disassembled format). In this way,if a problem develops during program execution, the real-time trace provides a record that can be reviewed to determine what and where the problem is.

Trace Width And Depth

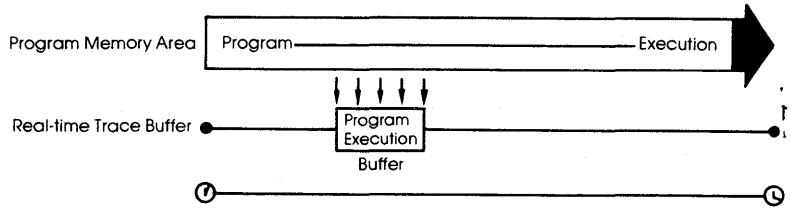
To be truly useful, an emulator's trace memory And Depth should be wide enough to accommodate all of the processor's address, data, and status lines. Your ICD features a trace memory that is 40 bits wide (16 bits data/16 bits address/8 bits status).

When it comes to the trace memory's depth, more is not always best. If too much depth is specified, it may be difficult to sift through all the data acquired. However, if the trace memory depth is insufficient, the chances of recording the trace section where a potential problem exists are significantly diminished. Your ICD contains a maximum trace memory depth of 4K (4095) machine cycles; and this may be reduced by specifying the "range" in the HISTORY command (except for the End Monitor and End Event modes). The ability to alter the size of the trace storage permits very specific tracing.

Real-time Trace Buffer

The data that is recorded from the program execution is stored in the ICD real-time trace buffer. The real-time trace buffer can be thought of as a data storage facility that moves parallel the user program, while storing the same data that is executed by the user program.

The maximum storage capacity of the real-time trace buffer is 4K machine cycles, but by using a "First-In/First-Out" (FIFO) recording technique, the buffer captures the latest program execution by discarding old data and replacing it with new data. By using this technique, the display reveals the latest data the buffer has stored.



Using The Real-time Trace

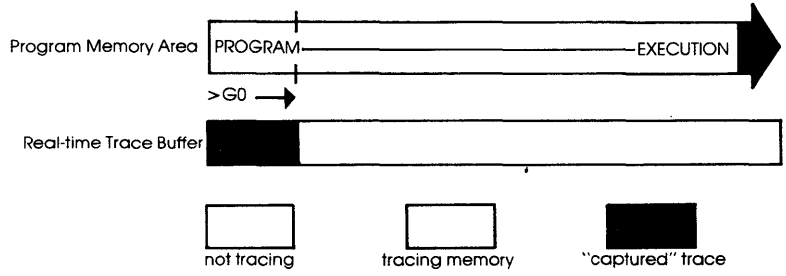
The ICD's real-time trace feature defaults to the End Monitor mode upon initialization; therefore, it is always active, that is, it records the program execution even if the HISTORY command parameters are omitted. The ICD can also display the recorded memory contents in four different modes (using the "HISTORY: Real-time Format Display" command).

The options, then, for the HISTORY command involve selecting the proper command format to trigger or halt the real-time trace feature. A discussion of each storage mode follows.

Simplest Case: Begin Monitor Mode

An easy way to understand how the real-time trace works is to examine the Begin Monitor mode. In this mode, the GO command (which begins emulation) also triggers the start of real-time tracing so that the data executed from the program memory area is simultaneously transferred to the real-time trace buffer.

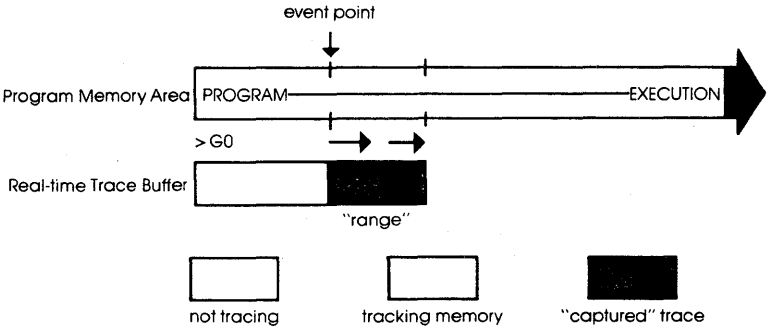
After the user program executes (and the buffer stores) the data equivalent of the range, the trace buffer fills to that point and then stops. The data that is now stored in the buffer is the "captured" trace section (the section that the ICD displays). The real-time trace then enters a non-trace mode and stops when a MONITOR break (accomplished by pressing the MONITOR switch) or breakpoint is encountered.



Begin Event Mode

The Begin Event mode works in the same way as the Begin Monitor mode except that an event point triggers the real-time trace instead of the GO command. The buffer stores the amount specified by the range (up to 4K) and then stops.

NOTE: The event itself is not stored in the buffer, but triggers the buffer to begin storing.

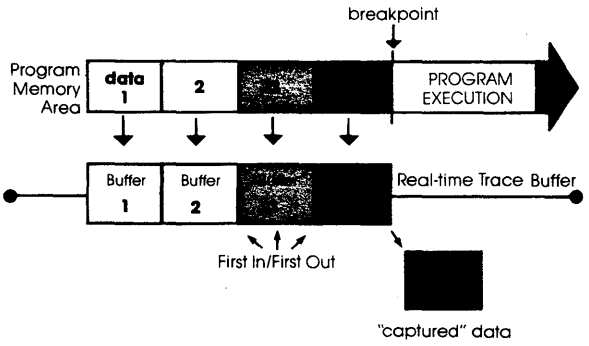


End Monitor Mode

The End Monitor mode begins storing all data, and then terminates the storage process when a breakpoint is encountered or when the MONITOR switch is pressed. The captured trace section is the last 4K before the breakpoint or MONITOR break.

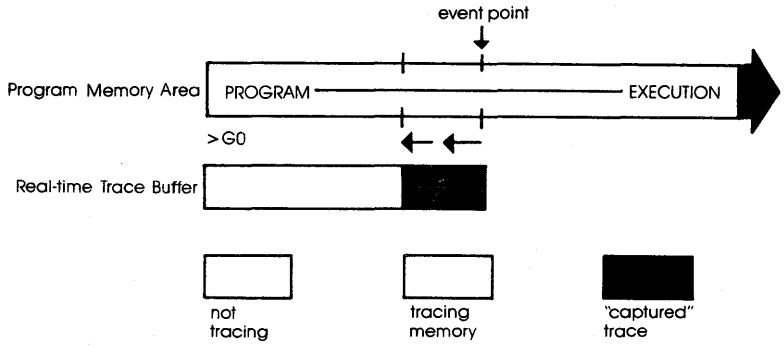
The ICD accomplishes this type of tracing by recording and storing data on a First-In/First-Out (FIFO) basis after the buffer is filled. By using this technique, the ICD displays the latest data in the trace buffer.

The End and Center Event modes use this same FIFO recording technique in their operation.



End Event Mode

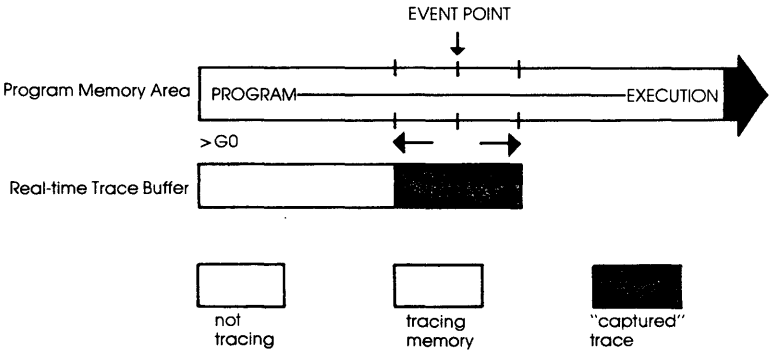
The End Event mode works in the same way as the End Monitor mode except that an event point (instead of a breakpoint) triggers the buffer to halt data storage. The captured trace section is the last 4K before and including the event point.



Center Event Mode

The Center Event mode is used when you desire the trace to surround a single event point in the program. It performs this task by reading the range specification and recording that number of cycles after the event point occurs. The remainder of the 4K buffer then contains cycles just prior to and including the event point. For example, if 1K is specified as the range, 1K of data would be captured after the event point, and the remaining 3K would be captured before the event point. If the specified range is 4000, 4000 cycles would be captured after the event, and the remaining 95 cycles would be captured before the event point. (4K = 4095 cycles.)

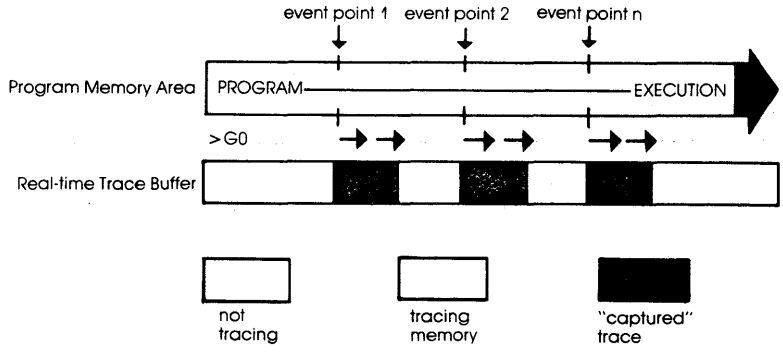
Just like the End Monitor and End Event modes, the Center Event mode causes the real-time trace to start recording data immediately after the GO command.



Multiple Event Mode

The Multiple Event mode is identical to the Begin Event mode, except that when the trace-range specification is reached, the tracing temporarily stops until another event point occurs. Then the buffer resumes storing another trace-range number of cycles. When the 4K buffer is completely filled, the event points are then ignored, and the buffer remains in a non-storage mode. This allows several event occurrences to trigger the History buffer, giving successive "snap-shots" of a particular routine.

NOTE: The smaller the trace-range, the more times an event can retrigger the buffer to store data.



TRIGGER NAME:	BEGIN MONITOR	END MONITOR	BEGIN EVENT
Command format	H BM	H EM	H BE,trace_range
Activated by	GO command	GO command	An event point
Terminates when	Buffer filled	Break in execution	Buffer is full
FIFO when buffer full?	No	Yes	Yes
Range affects	Storage size	Nothing (ignored)	Storage size
End result in buffer	First 4K cycles executed	Last 4K cycles executed	4K cycles following event

TRIGGER NAME:	CENTER EVENT	END EVENT	MULTIPLE EVENT
Command format	H CE,trace_range	H EE	H ME,trace_range
Activated by	GO command	GO command	An event point
Terminates when	Event point + range # of cycles	An event point occurs	Buffer is full
FIFO when buffer full?	Yes	Yes	No
Range affects	Offset of event from center	Nothing (ignored)	Temporary storage termination until
End result in buffer	4K surrounding events	Event point + 4K cycles	Several"snapshots" triggered by event points

Command HISTORY: Real-time Trace Status

Operation Displays the current status of the real-time trace buffer.

Applications Note: The real-time trace status can be used to analyze the condition of the real-time trace buffer, (i.e., storage mode name, size of the trace range, number of cycles executed, and number of cycles stored in the History buffer.)

When the real-time trace specifications are changed, the "HISTORY: Status" command will display their latest settings.

Syntax **H**

Command Example

Press the RESET switch on the ICD to initialize the HISTORY command, then enter:

```
>H
Clock Counts = 00000000/0 <— number of clock cycles
Storage Mode = EM <———— mode and trace range
Storage Size = 0/0 <— number of cycles passed
>
```

The ICD defaults to this condition whenever it is initialized. The ICD also automatically resets the clock counter to 0, selects End Monitor as the storage mode, sets the trace range to maximum, and initializes the storage size to 0.

In this example, "Clock Counts" shows the number of clock cycles (T-states) since the real-time trace was cleared. The number to the left of the slash (/) is the hexadecimal number of clock cycles, and the number to the right is its decimal equivalent. "Storage Mode" shows that the default specification is the "End Monitor" mode (the trace range for this mode is automatically set to 4095). "Storage Size" shows the number of cycles since the program was started (to the right of the slash) or since the program was resumed (to the left of the slash). If the Storage Size displayed "Full," it would indicate a full buffer, or 4095 cycles.

In the next example, the Begin Event mode is selected and the trace range is omitted. The status command now shows:

```
>H
Clock Counts = 00000000/0
Storage Mode = BE 4093
Storage Size = 0/0
>
```

The trace range for the Begin Event mode defaults to 4093 (4095, the maximum, must be specified).

Command	HISTORY: Real-time Trace Counter Reset
Operation	Clears (resets) the clock counter.
Syntax	**H CLR**
Notes	Spacing: A space is required between H and CLR.
Command Example	See "HISTORY: Real-time Trace" examples.

Command HISTORY: Real-time Trace Format Display

Operation Allows the contents of the real-time trace buffer to be displayed in either machine cycle format or disassembled format.

Syntax **H *mode*[*,int_point*][*,term_point*]**

Terms mode = M, D, C, or X

int_point = Initial point of display, from 1 to 4095.

term_point = Point at which display terminates, from 1 to 4095.

Syntax Example

```
H M,200,100
H D
```

Notes M specifies to display the program execution in machine cycle format. D displays the program execution in disassembled format (excludes opcode fetch), C specifies clock cycle format, and X specifies machine cycle with disassembly.

The *int_point* must be greater than or equal to *term_point*. The storage pointer is numbered by bus cycles - displayed from high to low - where "1" is the most recent bus cycle.

This command displays items on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required between H and *mode*. Spaces are not permitted where the commas are used to separate the parameters.

Command Example

See the following and the "HISTORY: Real-time Trace" examples.

If H M (machine cycle format) or H C (clock cycle format) is selected, the following headings will be shown on the display:

Point T Address St Data Seg IF QS_Instruction

Where: Point = address in HISTORY buffer
T = event point indicator
Address = cycle address
St = cycle status (type of cycle operation)
Data = cycle data
Seg = segment used in this cycle to derive address
IF = interrupt flag status
QS_Instruction = queue status_Hex machine code
F_xx = first instruction fetched from queue
S_xx = subsequent instruction fetched from queue
Emp = queue flush

If H D (disassembled format) or H X (machine cycle format with disassembly) is selected, the following headings will be shown on the display:

Point T Addr. Machine Code Prefix Opcode Operand

Where: Point = address in HISTORY buffer
T = event point indicator
Addr. = cycle address
Machine Code = entire instruction in Hex code
Prefix = disassembled prefix mnemonic, e.g., LOCK, REPNE, etc.
Opcode = disassembled opcode mnemonic
Operand = disassembled operand (may be symbolic if ZICE software is used)

Command HISTORY: Real-time Trace Storage Mode

Operation Specifies the trace mode for the real-time trace buffer. This is the command that specifies what type of mode activates the real-time trace feature.

Syntax **H *mode*[,*range*]**

Terms *mode* = Trace mode. This can be one of six different modes, including:

BM (begin monitor mode)
EM (end monitor mode)
BE (begin event mode)
CE (center event mode)
EE (end event mode)
ME (multiple event mode)

range = The trace range, from 1 to 4095.

Syntax Example H ME,2027

Notes The range specified for the EM and EE modes will be ignored; it defaults to the maximum 4K size.

Spacing: A space is required between H and mode. No spaces are permitted where the commas are used as separators.

Command Examples See "HISTORY: Real-time Trace" examples.

HISTORY: Real-time Trace Command Examples

NOTE: To illustrate the following examples, a sample program is first entered into the ICD memory, and then disassembled for inspection (the sample program starts at address 1000:0400). The same program is used throughout the HISTORY command demonstration.

The ICD defaults to the End Monitor mode when it boots up. The sample program is now disassembled for inspection.

```
>DI 400,L22  <— sample program is disassembled for inspection
1000:0400 B0FF                  MOV     AL,0FFH
11000:0402 F08606F0FF  LOCK     XCHG    AL,BYTE PTR 0FFF0H
1000:0407 22C0                  AND     AL,AL
1000:0409 7403                  JE      040EH
1000:040B 90                    NOP
1000:040C 90                    NOP
1000:040D 90                    NOP
1000:040E BB0080                MOV     BX,8000H
1000:0411 8A07                  MOV     AL,[BX]
1000:0413 E6FE                  OUT     0FEH,AL
1000:0415 43                    INC     BX
1000:0416 81FB0090              CMP     BX,9000H
1000:041A 75F5                  JNE     0411H
1000:041C B000                  MOV     AL,00H
1000:041E A2F0FF                MOV     BYTE PTR 0FFF0H,AL
1000:0421 E9DC0D                JMP     1200H
>
>H EM <——   sets HISTORY command trace mode to end monitor (although the end monitor mode is the
            default state). In this mode, the ICD captures all activity (up to 4K) before a breakpoint or
            monitor break.

>H <—— displays current trace settings
Clock Counts = 00000000/0 <— displays total # of clock cycles
Storage Mode = EM  <—— displays current trigger mode
Storage Size = 0/0 <—— displays # of clock cycles since program ran
>R CS=1000 <——— sets code segment to 1000
>G 400,40E <—— runs program from 400 to 40E, breaks, then displays:

1040E   BB0080              MOV   BX,8000H
CS  IP  ODITSZAPC  AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:0415 000010000 0000 0000 0000 0000 0000 0000 0000 0000 ...
<Break Hardware A>
>H <—— shows current trace settings
Clock Counts = 00000064/100
Storage Mode = EM
Storage Size = 54/54
>
```

```
>H C <—— displays captured trace section in clock cycle format
...
0003    0000A_W MR
0002                    8090  0  Emp
0001 * Pause <—   Pause indicates buffer storage termination/*
                  indicates trigger recognition
>
>H M <——— displays captured trace section in machine cycle format
Point T Address St  Data Seg  IF QS_Instruction
0054    10400_W OF  FF30  0
0052    10402_W OF  86F0  0  F_30 S_FF
0048    10404_W OF  F006  0  F_F0 F_86
0045    10406_W OF  22FF  0  S_06
0042    ...
0007    00008_W MR  040D  0
0005    10418_W OF  9000  0
0003    0000A_W MR  8090  0
0001 * Pause
>
>H D 9<— displays captured trace section in disassembled format
Point T Addr.  Machine Code      Prefix   Opcode Operand
0009    (0E607  MW  FE    0)
0007    (00008_W MR 040D  0)
0003    (0000A_W MR 8090  0)
0001 * Pause
>
>H BM <——   changes trigger to the begin monitor mode. In this mode, the ICD captures all activity from the
            start of program execution up to the range specification (in this case, 4K).

>G 400,421 <—  runs program from 400 to 421, breaks, then displays:

10421   E9DC0D                  JMP   $+0DDFH
CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:1200 000001010 0008 9000 0000 0000 0000 0000 0000 0000 ...
<Break Hardware A>
>
>H <— displays current trace settings
Clock Counts = 0018C0BF/1622207
Storage Mode = BM 4093 <— note new mode and maximum range specification
Storage Size = 4093/4093
>H D,4028 <— displays captured trace section from 4028 to 0001
>
>EV ST=PW A=XXXFE D=41 <—   sets event point in program for begin
>                           event trigger recognition
>H BE <— —  changes trigger to the begin event mode. In this mode,
            an event point in the program initiates the trace. The trace buffer then stores the range specified
            (in this case, 4K) and stops.
```

```
>G 400,421 <— runs program from 400 to 421, breaks, then displays:

10421  E9DC0D              JMP   $+0DDFH
CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:1200 000001010 0000 9000 0000 0000 0000 0000 0000 0000 ...    <Break Hardware A>
>H <— displays current trace settings
Clock Counts = 0031811A/3244314
Storage Mode = BE 4093 <— note new mode and maximum range specification
Storage Size = 4093/4093
>
>H D <— displays captured trace section of full buffer
>
>H ME,37 <—   changes trigger to the multiple event mode. In this mode, the trace is triggered each time an event
              point is passed. The trace buffer then fills up to the range specification (in this case, 37),
              temporarily stops tracing, then resumes tracing when another event point is encountered.
>G 400,421 <— runs program from 400 to 421, breaks, then displays:

10421  E9DC0D              JMP   $+0DDFH
CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:1200 000001010 0000 9000 0000 0000 0000 0000 0000 0000 ...    <Break Hardware A>
>
>H <— shows current trace settings
Clock Counts = 004A4175/4866421
Storage Mode = ME 37 <— note new mode and range specification
Storage Size = Full
>H D  <— displays captured trace section from event trigger
>
>H EE <——    changes trigger to the end event mode. In this mode, the
              trace stops on the event point. The trace buffer then
              displays the last 4K cycles before and including the event point.
>G 400,421 <— runs program from 400 to 421, breaks, then displays:

10421  E9DC0D              JMP   $+0DDFH
CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:1200 000001010 0000 9000 0000 0000 0000 0000 0000 0000 ...
<Break Hardware A>
>
>H <— shows current trace settings
Clock Counts = 006301D0/6488528
Storage Mode = EE <— note new mode and maximum range (default size)
Storage Size = Full
```

```
>H D <— displays captured trace section up to the event point
>
>H CE <—        changes to the center event mode. In this mode, the captured trace section surrounds a single event
                point in the program. The display will show the range specification after the event point, and 4K
                minus the range before the event point.
>G 400,421 <— runs program from 400 to 421, breaks, then displays:

10421   E9DC0D              JMP    $+0DDFH
CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS
1000:1200 000001010 0000 9000 0000 0000 0000 0000 0000 0000 ...    <Break Hardware A>
>
>H
Clock Counts  = 007BC22B/8110635
Storage Mode  = CE 2046 <— note new mode and range (the default size)
Storage Size  = Full
>H D <—— displays captured trace section surrounding event point
>
```

Command

HISTORY: Real-time Trace Search By Machine Cycle

Operation

Searches through the History trace buffer for certain specified operations. For example, "find all of the times a memory write operation to memory location 1234H occurred."

Syntax

H S,[*int_point*][,*term_point*][,ST=*status*][,A=*addr*] [,D=*data*][,T]

Terms

int_point = Initial point of display, from 1 to 4095.

term_point = Point at which display terminates, from 1 to 4095.

status = Type of status, and includes one of the following:

MR (memory read)
MW (memory write)
PR (port read)
PW (port write)
IA (interrupt acknowledge)
HA (halt acknowledge)
OF (operation code fetch)
NR (NDP read)
NW (NDP write)

addr = Value to search for ("addr_W" means to search for a word address).

data = Data to search for. (Must also specify address.)

Syntax Example

H S,200,100,ST=OF,A=CS:0,D=0,T

Notes

If *data* is specified, addr specification is also required. The *int_point* defaults to 4095, and term_point defaults to 1; otherwise, *int_point* must be specified as greater than or equal to *term_point*.

The storage pointer is numbered by bus cycles - displayed from high to low - where "1" is the most recent bus cycle.

The address may be preceded by a segment qualifier, including CS (code segment), DS (data segment), ES (extra segment) and SS (stack segment).

This command displays items on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required between H and S, and thereafter no spaces are permitted if commas are used to separate information.

Command Example

See Syntax Example.

Command	HOST
Operation	Initiates or terminates LOCAL "Host Computer Assisted" mode. Applications Note: This command enables the ICD to operate as though it is in the REMOTE mode when connected to a host computer running in the LOCAL mode (terminal control of the ICD with host computer access). Using this configuration, only one SIO port is required of a multi-user host computer (e.g., VAX), rather than two ports a required in the REMOTE mode.
Syntax	**HOST *switch***
Terms	switch = ON or OFF
Syntax Example	HOST ON
Notes	This command is only available with firmware versions 2.0 or greater, and only recognized when the ICD is in the LOCAL mode. ON enables the HOST feature and OFF disables the HOST feature. The QUIT command will also perform the equivalent of the HOST OFF command, but the HOST OFF command does not terminate ZICE. Spacing: A space is required between HOST and *switch*.
Command Example	See Syntax Example.

Command IDENTIFICATION

Operation Displays the current ICD device name and the firmware version.

Syntax **ID**

Notes This display is also shown when the RESET switch is pressed on the ICD.

Command Example

```
>ID
ICD-378 for 80188   V1.0
```

This example shows that the ICD emulates the 80188 processor and that the firmware version within the ICD is 1.0 (if the 80186 processor was installed, "80186" would be displayed). Your firmware version may be different than 2.0, depending on your purchase date.

Command IN-CIRCUIT: Status

Operation Displays the current in-circuit status, either 0, 1, or 2. The in-circuit status is also displayed when the "MAP: Status" command is used.

Syntax I

Command Example See the MAP command.

Command IN-CIRCUIT: Specification

Operation Sets the ICD mapping mode. See Notes (below) and the MAP command for an explanation and example of the different mapping modes.

Syntax **I [*mode*]**

Terms *mode* = 0, 1, or 2

Syntax Example I 0

Notes 0 = System mode. Debugging is performed using the ICD program memory only. The area specified as US (user memory) by the MAP command acts as RW (read/write memory) in the ICD. Target system I/O and interrupt signals are ignored.

1 = Partial mode. Debugging is performed using the ICD program memory and user (target system) memory, as defined by the MAP command. Interrupts can be disqualified by using the PIN command.

2 = All mode. Debugging is performed using only the target system memory. Memory now mapped as read/ write and read-only act as user (target system) memory. I/O and interrupts are enabled. Any area mapped as NO (non-memory) will act as NO memory regardless of the in-circuit mode.

In-circuit mode settings and memory specifications are shown below.

In-circuit	Memory Type			PIN Functions		
Mode/Description	RO	RW	US	NO	EN	DI
I0/System Mode	RO	RW	(RW)	NO	(DI)	DI
I1/Partial Mode	RO	RW	US	NO	EN	DI
I2/All Mode	(US)	(US)	US	NO	EN	(EN)

(). Items in parentheses show the revised memory or PIN specification for that particular in-circuit mode.

Spacing: A space is required between I and mode.

Command Example See the MAP command.

Command LOAD

Operation Downloads an Intel-Hex file from the host computer to the ICD's memory (or through the ICD to user memory).

Applications Note: This command can be used in both LOCAL (ICD controlled by a terminal, using a computer for storage) and REMOTE (ICD controlled by a host computer running ZICE software) modes.

Syntax L[*/source*] *filename*[*.ftype*][*,bias*][*,message*]

Terms *source* = T, P, A, or H

filename = Name of the file to download to the ICD.

.ftype = Optional filetype (.abs is the default).

bias = Memory address offset to be added to the object file being loaded (default is 0).

message = Any ASCII message (in 'single' quotes) or

hex data, or any combination separated by commas.

Syntax Example

```
L/H TEST.H86,100
L/A ,,'TYPE TEST.HEX',0D
L/A
```

Notes If *source* is omitted, command defaults to H in the REMOTE mode or LOCAL with HOST ON mode, and T in the LOCAL mode.

T specifies to use the TERMINAL port and X-ON/X-OFF protocol.
P specifies to use the TERMINAL port and software protocol.
A specifies to use the HOST/AUX port and X-ON/X-OFF protocol.
H specifies to use the HOST/AUX port and software protocol. (See software specifications in Section 4 for a description of the software protocol.)

When using XON-XOFF protocol options (T, A), it is necessary for the host to either recognize XON-XOFF, or delays must be inserted after each carriage return (end of each record). Otherwise, every second record may be lost. Also, if recognition of XOFF by the host computer is slow (more than two characters), the problem could exist as well. In certain instances, a slower baud rate may help to correct the problem (but is usually undesirable, due to extended download times, especially with long files). The *message* is sent out the source port at the beginning of the load operation to provide a way of prompting the host computer to begin transmitting a file.

Spacing: A space is required before *filename*; no spaces are permitted where commas act as separators.

Command Example

See Syntax Example. The first example shows how the LOAD command is used with ZICE (host software utilizing software protocol). If ZICE is used, H becomes the default, and may therefore be omitted. With this example, a bias of 100H is added to the load address.

The second example loads a file from a host computer not using ZICE software. For this application, the ICD's HOST/AUX port must be connected to a port on the host computer normally designated for a terminal (one having access to the OS command language).

The message is sent to the host computer, followed by a carriage return (specified by 0D - which is its ASCII code) prompting the host computer to transmit the file TEST.HEX to the ICD.

The third example is used when the host computer's OS command language cannot be accessed via the SIO port, but rather from a separate terminal. This command will be given to the ICD first, then the ICD will wait - ready to receive input prompted from the host terminal.

Command MAP: Status

Operation Displays the current memory assignments and address parameters as defined by the "MAP: Specification" command.

Syntax MA

Command Example

Execute this sequence:

```
>I 0 <———————— uses ICD's memory resources
>MA <——————— shows how memory is categorized
In-Circuit Mode 0 (US=>RW)
00000-FFFFF = RW (000)
>
```

In this example (default condition), the in-circuit mode is first set to 0 (debugging using ICD memory only), and then the MAP status command is entered. The display shows that the in-circuit mode is 0, that user (target system) memory now acts as read/write memory (US=>RW), and that the entire memory area (from 0H to FFFFFH) is categorized as read/write memory. The (000) ranges from 0H to FFFH, and indicates the block number in 1K-block increments. 0 to 32 (1FH) is the standard range.

A second example is shown below:

```
>I 2 <——— uses target system's memory resources
>MA <—— shows how the memory is categorized
In-Circuit Mode 2 (RW,RO=>US)
00000-FFFFF = RW (000)
>
```

In this example, the I 2 mode (debugging using system memory only) is selected, and then the MAP status is requested. The display shows that the in-circuit mode has changed to 2, and that all memory categorized as read/write or read-only (from 0H to FFFFFH) now functions as user (target system) memory.

Command MAP: Specification

Operation Categorizes the target system's memory functions as either read-only, read/write, user (target system) or non-memory area.

Applications Note: This command can be used to develop your target system's firmware (ROM) by allowing code in a mainframe system to be downloaded to the ICD, mapped as RO, and tested before being burned into the target's ROM.

Syntax MA *beg_addr*[,*end_addr*]=*area*

Terms *beg_addr* = The beginning address of mapping.

end_addr = The ending address of mapping.

area = RO, RW, US, or NO

Syntax Example
MA 1000,1FFF=US
MA 150=RO

Notes The target system or ICD memory is used in 1K-byte blocks. The parameters are only valid when the in-circuit mode is I1. (See IN-CIRCUIT command.)

If the *beg_addr* or *end_addr* does not coincide with the beginning or ending of a 1K-block location, the beginning or ending area is assigned a location that includes *beg_addr* or *end_addr*.

Two of the areas, RO and RW, refer to ICD user memory, and RW gives the user program free access to this memory. RO enables the user program to read this memory, but any attempt to write to this area will be blocked, and (unless the B/W breakpoint is disabled) will also cause a break during program execution.

US acts as target system memory area (US being RAM, ROM, I/O, etc. - whatever resides at those locations in the target). NO memory assignment is useful in debugging by causing a break in the emulated program if an attempt is made to access this non-existent memory area. A NO memory area is recognized as such, regardless of the in-circuit mode.

Command Example

Execute this sequence:

```
>I 1 <------------------- uses both ICD and target
>MA 0,FFF = RO           system memory resources
>MA 1000,1FFF = US <---- categorizes memory
>MA 2000,FFFF = RW       blocks
>
>MA <-------------------- shows how the memory
In-Circuit Mode 1 is categorized
00000-00FFF = RO (000)
01000-01FFF = US
02000-FFFFF = RW (008)
>
```

In this example, the I 1 (debugging using both ICD memory and target system memory) is selected, and then the memory blocks are categorized as read-only (0H to FFFH), user (1000H to 1FFFH), and read/write (2000H to FFFFFH). The MAP status command is then entered, showing how the memory was just specified. A second example is shown below:

```
>I 2 <------- uses target system memory resources
>MA <--------- shows how the memory is categorized
In-Circuit Mode 2 (RW,RO=>US)
00000-00FFF = RO (000)
01000-01FFF = US
02000-0FFFF = RW (008)
>
```

In this example, the I 2 (debugging using target system memory only) is selected, which automatically categorizes read/write and read-only memory areas (from 0H to FFFFFH) as user (target) memory (RW,RO=>US).

Command MOVE

Operation Moves the memory contents between different locations within the ICD, or between the ICD and the target system.

Syntax **M *beg_addr,end_addr,mov_addr*[,*direction*]**

Terms *beg_addr* = Beginning address of data source.

end_addr = Ending address of data source.

mov_addr = Beginning address for destination.

direction = UP or PU

Syntax Example M 100,3FF,100,UP

Notes UP means that the source is user (target system) memory and the destination is ICD program memory. PU means that the source is ICD program memory and the destination is user (target system) memory. If direction is omitted, data is relocated within the memory areas as specified by the MAP command.

Spacing: A space is required between M and *beg_addr*. No spaces are permitted where commas are used as separators.

Command Example See Syntax Example. In this example, a block of memory in the target system, beginning at address 100H and ending at address 3FFH, is moved to the ICD, beginning at address 100H.

For an application of the MOVE command, carry out the demonstration below:

```
>F 0,3FF,55 <— fills a 1K memory block with 55s
>D 0,3FF <——— displays the memory contents
>M 290,3A0,2EFF <— moves a section of the memory
>                          to address 2EFFH
>D 2EFF,300E <—— displays the transferred memory
>                          contents at the new location
```

Command NEXT

Operation This command is a subcommand of the TRACE command. It allows the next 1 to 65,535 instructions to be executed and traced in non-real time from the current CS:IP.

Syntax **N [*steps*]**

Terms **steps** = 1 to 65,535

Syntax Example N 5

Notes The *steps* means the number of instructions to execute from the current program counter, and may be any integer from 1 to 65,535. If *steps* is omitted, only a single instruction line is displayed.

When the registers' contents are displayed as a series of periods (....), it indicates that the contents of the registers are unchanged. The registers contents are displayed fully, however, at least once every 22 lines.

Spacing: A space is required between N and *steps*.

Command Example

Press the RESET switch on the ICD, then execute this sequence:

```
>F 0,FFF,90 <—— fills memory with NOPs
>G 0:0,1F0 <——— starts the program running from
address 0 and stops at address 1F0H, then displays:

001F0  90                          NOP
       CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP ...
0000:01F2 000000000 0000 0000 0000 0000 0000 00...
<Break Hardware A>
>N 3 <————————————— shows the next three instruction lines

       CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP ...
0000:01F3 000000000 0000 0000 0000 0000 0000 0000 001F3
90             NOP
0000:01F4      ........    .... .... .... .... .... ....
.... ....  .... ....    ..
001F4  90                          NOP
0000:01F5      ........    .... .... .... .... .... ....
.... ....  .... ....    ..
>
```

This example illustrates how the NEXT command is used after program execution halts. When the program stops at address 1F0, entering N3 causes the next three instruction lines to be displayed.

Command OFFSET: Status

Operation Displays the status of the "OFFSET: Specification" command.

Syntax **O**

Command Example

```
>O <——— shows the status of the offsets
&0 = 0000  <— shows the default conditions (all
&1 = 0000      offset registers = 0)
&2 = 0000
&3 = 0000
>
```

This example shows the default condition of the OFFSET command. Changing the address of any one of the four offset values (0-3) causes a change in the 0000 display.

Command OFFSET: Specification

Operation Sets an offset in the ICD for relative program addressing.

Applications Note: This command is useful when debugging a program that consists of a number of different modules. The procedure would be to assign the physical base address for each module to one of the offset registers. Any location in a module may be addressed by specifying its relative address to that module's base address, plus an offset register. The address parameter of any command will then be interpreted as the sum of the relative address and the offset register (physical base address).

Syntax O *&number[=addr]*

Terms *number* = 0, 1, 2, or 3

addr = Offset to place in the register.

Syntax Example O &2=FFF

Notes Any of the four offset registers can be used with any of the ICD command memory addressing parameters.
When *addr* is omitted, the offset register is cleared to zero.
Spacing: A space is required between O and &. No spaces are permitted between *&number=addr*; the equal sign (=) acts as the separator.

Command Example

```
Execute this sequence:
>O &1=351<—— sets #1 value to offset of 351
>O <———————— shows current offset values
&0 = 0000
&1 = 0351
&2 = 0000
&3 = 0000
C 1377H+351H <—— use calculation command to find
000016C8H       offset address location (1377H+
5832          351H=16C8H)
>
>DI 0:1377&1 <——— disassembles from address 1377H+ the offset value
0000:16C8 0000        ADD ....
0000:16CA 0000        ... ....
0000:16CE 0000        ... ....
0000:16CF 0000        ... ....
.... .... .... etc.
```

Command PIN: Status

Operation Displays the current status of the "PIN: Specification" command.

Syntax PI

Command Example

```
>PI <-------------      shows status of interrupt signals
All Pin Disable         in the I 0 mode
NMI  (EN) = H
HOLD (EN) = H
DRQ0 (EN) = H
DRQ1 (EN) = H
INT0 (EN) = H
INT1 (EN) = H
INT2 (EN) = H
INT3 (EN) = H
TEST/    = H
>
```

This example shows the status of the interrupt signals when the in-circuit mode is 0. "All Pin Disable" means that the interrupt signals cannot be enabled in this mode. "H" shows that the current logic levels of the signal are high. The slash (/) after the signal name signifies an "active-low" signal.

If the ICD was operating in the I1 mode, the display would show:

```
>PI
In-Circuit Mode 1
NMI  (EN) = H
HOLD (EN) = H
DRQ0 (EN) = H
DRQ1 (EN) = H
INT0 (EN) = H
INT1 (EN) = H
INT2 (EN) = H
INT3 (EN) = H
TEST/    = H
>
```

If the in-circuit mode was 2, all interrupt signals would automatically be disabled.

Command PIN: Specification

Operation Masks or unmasks selected input signals when the in-circuit mode is 1.

Syntax **PI *signal*=*switch***

Terms

signal =
NMI (Non-Maskable Interrupt)
HOLD (Hold)
DRQ0 (Data Request 0)
DRQ1 (Data Request 1)
INT0 (Interrupt 0)
INT1 (Interrupt 1)
INT2 (Interrupt 2)
INT3 (Interrupt 3)
TEST (Test)

switch = EN or DI

Syntax Example PI DRQ0=DI

Notes The parameters for this command are only valid when the in-circuit mode is 1. When the in-circuit mode is 2, all signals are valid. When the in-circuit mode is 0, all target system signals are ignored.

EN is used to enable the signal and DI is used to disable the signal.

Spacing: A space is required between PI and *signal*.
No spaces are permitted after *signal*.

Command Example See Syntax Example.

Command PORT

Operation Examines one or more I/O port locations and optionally modifies them. The locations can be displayed and replaced with either hexadecimal or ASCII values.

This command works on the same principle as the EXAMINE command, except that the port address accesses the I/O port space.

Syntax **P[/W] *port_addr*[=*mod_data*]**

Terms W = Word mode (default is the byte mode).

port_addr = Starting address for display.

mod_data = New data for this location.

Syntax Example

```
P FF=23
P 55
```

Notes If *mod_data* is omitted, the command enters a repeat mode, which allows several locations to be changed.

The repeat mode includes:

return (cr) to display the next byte (word) of data.
comma (,) to display the same byte (word) of data.
caret (^) to display previous byte (word) of data.
slash (/) to exit the PORT command.

Spacing: A space is required between P and *port_addr*. No spaces are permitted between *port_addr* and *mod_data*; the equal sign (=) acts as the separator.

Command Example

See Syntax Example. The first example illustrates how the port located at address FFH is changed to a data value of 23H. The second example allows the ports to be modified, beginning at address 55H.

Now examine the following display:

```
>P 12          <- start by examining port #12
0012 12=23,    <- change value to 23; re-examine
0012 12=       <- leave value unchanged; go to next
                  address
0013 00=       <- leave value unchanged; go to next address
0014 14='21'   <- change value; go to next address
0015 00=34^    <- change value; go to previous address
0014 14=^      <- leave value unchanged; go to previous
                  address
0013 00=,      <- leave value unchanged; re-examine address
0013 00=17     <- change value; go to next address
0014 14=       <- leave value unchanged; go to next address
0015 00=       <- leave value unchanged; go to next address
0016 16=00/    <- change value; exit command
>
```

Command	PRINT
Operation	Controls logging of ICD commands by sending the terminal display to an external serial printer.
Syntax	**PR *switch***
Terms	*switch* = ON or OFF
Syntax Example	PR ON
Notes	ON enables the printing feature and OFF disables the printing feature. The printing is routed to the HOST/AUX port when the ICD is in LOCAL mode, and to the host printer when the ICD is in REMOTE mode (using ZICE, or the LOCAL "HOST ON" mode using ZICE). Spacing: A space is required between PR and *switch*.
Command Example	See Syntax Example.

Command REGISTER: 80186/80188 Status

Operation Displays the current status of the 80186/80188 registers and any changes made after using the "REGISTER: Examine and Change" command.

Syntax R

Command Example

```
>R
  CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP  SI  DI  SS  DS  ES  RL
0000:0000 000000000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00FF
```

Notes This example shows the status of the 80186/80188 registers (currently all 0). Changing any of the registers with the "REGISTER: Examine and Change" command affects this display.

Register Structure

The CPU's have eight 16-bit general registers. They are divided into two files of four registers each: the data register flag and the pointer and index register file.

The 16-bit data register are named AX, BX, CX and DX; the 8-bit registers are named AL, AH, BL, BH, CL, CH, DL and DH (the H or L suffix designates high-order or low-order byte of the 16-bit register). The pointer and index registers consist of the 16-bit registers SP, BP, SI and DI. SP and DP are pointer subfiles while SI and DI are index subfiles.

The segment registers are also 16-bit registers. These registers specify the four currently addressable memory segments: CS (code segment), DS (data segment), SS (stack segment), and ES (extra segment).

The status and control registers consist of the instruction pointer (IP) and the status word or flags. Status word or flags include: OF (overflow flag),DF (direction flag), IF (interrupt enable flag), TF (single step flag), SF (sign flag), ZF (zero flag), AF (set on carry from or borrow to the low order four bits of AL), PF (parity flag) and CF (carry flag).

The control block base address is programmed via a 16-bit relocation register (RL) contained within the control block at offset FEH from the base address.

Command REGISTER: 8087 Status

Function Displays the current status of the 8087 registers.

Syntax **R[/*status*]**

Terms *status* = E or N

Syntax Example

R/E
R/N

Notes

E displays the 8087 registers ST(0) - ST(7), and N display all 8087 registers.

The 8087 register contents can be changed by using the "REGISTER: Specification" command.

Spacing: No spacing is allowed between parameters.

Command Example

```
R/N
Control Word:
        X  X  X IC  -RC-  -PC- IEM X PM UM OM ZM DM IM
        0  0  0  0   0   0   0   0  0  0  0  0  0  0
Status Word:
        B C3  —TOP— C2 C1 C0 IR  X PE UE OE ZE DE IE
        0  0   0   0  0  0  0   0  0  0  0  0  0  0
Instruction Address:
        00000
Last Operation:
        D8 00:
Operand Address:
        00000
Stack Register:
        ST(0) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(1) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(2) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(3) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(4) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(5) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(6) 00 00 00 00 00 00 00 00 00 00  Tag=0
        ST(7) 00 00 00 00 00 00 00 00 00 00  Tag=0
>

>R/E
        ST(1) +0.0000000000000000E+0  Valid
        ST(2) +0.0000000000000000E+0  Valid
        ST(3) +0.0000000000000000E+0  Valid
        ST(4) +0.0000000000000000E+0  Valid
        ST(5) +0.0000000000000000E+0  Valid
        ST(6) +0.0000000000000000E+0  Valid
        ST(7) +0.0000000000000000E+0  Valid
>
```

Command REGISTER: Reset

Operation Sets all the registers to zero (except for CS, which remains FFFFH).

Syntax **R RESET**

Notes To reset CS to 0000, use the "REGISTER: Specification" command: R CS=0.

Spacing: A space is required between R and RESET.

Command Example

Execute this sequence:

```
>R <———————— shows the status of the registers
>R CX=2FFE <— sets register CX to a value of 2FFEH
>R <———— shows the status of the registers again
>R RESET <—— resets all register values to 0
>R <—— verifies the change to 0
>
```

This example shows how register CX is changed from 0000 to 2FFE, and then set back to 0000 using the "REGISTER: Reset" command.

Command REGISTER: Examine and Change

Operation Examines and changes the contents of the 80186/80188 or 8087 internal registers.

Syntax **R[/*mode*] *reg_name*=[*data*]**

Terms *mode* = E or N. E displays the 8087 registers ST(0) - ST(7), and N display all 8087 registers.

reg_name = Any one of the following registers:

AX AH AL BX BH BL
CX CH CL DX DH DL
CS DS ES SS SP IP
BP SIDI FL RL ODITSZPC
RL

data = New value for register contents.

Syntax Example

```
R HL=A000
R DE
```

Notes If R *reg_name* is entered, this command displays the current contents of the specified register. If data is used, this command changes the contents of the specified register to the new value.

For reg_name(s) IF,S,Z,P,N, and CY, only 0 and 1 are valid data entries.

E displays the 8087 registers ST(0) - ST(7), and N display all 8087 registers.

Spacing: A space is required between R and *reg_types*. No spaces are permitted after *reg_types*; the equal sign (=) acts as the separator.

Command Example

Execute this sequence:

```
>R
      CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP
FFFF:0000 000000000 0000 0000 0000 0000 0000 000 ...
>R DX=1FFE
>R
      CS  IP ODITSZAPC AX  BX  CX DX  SP  BP
FFFF:0000 000000000 0000 0000 0000 1FFE 0000 000 ...
>R DX
1FFE
>
```

This example illustrates how the contents of a register are changed to a new value, and the two ways it can be checked.

Command RESET

Operation Resets the I/O of the target system via the CPU in-circuit probe. The RESET signal is asserted to the processor for 124 clock cycles.

Syntax **RES**

Notes When used singularly, the RESET command resets the target system I/O only. When the RESET command is used with the REGISTER command, it can reset the processor's registers as well (e.g., R RES).

Command SAVE

Operation Saves an Intel Hex file from the ICD memory to the host computer. (The file format is the same as the LOAD command.)

Syntax SA[/*destination*] *filename*[,*ftype*],*beg_addr*,*end_addr*, *entry_addr*[,*message*]

Terms

destination = T, P, A, or H.

filename = Name of the file to be used for saving the memory contents.

ftype = Optional three-letter filetype (.abs is the default).

beg_addr = First address to save.

end_addr = Last address to save.

entry_addr = Starting address of the user program.

message = Any ASCII message (in 'single' quotes) or hex data, or any combination separated by commas.

Syntax Example

```
SA/H TEST.H86,0,3FF,1000
SA/A TEST.HEX,0,1FFF,0,'CREATE TEST.HEX',0D
```

Notes If destination is omitted, command defaults to H in the REMOTE (host computer control of the ICD) mode or LOCAL with HOST ON (host computer assisted) mode, and T in the LOCAL (terminal control of the ICD) mode.

T specifies to use the TERMINAL port and X-ON/X-OFF protocol. P specifies to use the TERMINAL port and software protocol. A specifies to use the HOST/AUX port and X-ON/X-OFF protocol. H specifies to use the HOST/AUX port and software protocol. (See software specifications in Section 4 for a description of the software protocol.)

The *message* is sent out the destination port at the beginning of the save operation to provide a way of prompting the host computer to receive a file. (Remember to use the USER command to access the host and to terminate the file input.) Either XOFF-XON or DTR-DSR flow control will be accepted by the ICD when the destination option is T or A. If the host computer does not provide input flow-control, its input buffer will probably overflow.

Spacing: A space is required before *destination*; no spaces are permitted where commas act as separators.

Command Example See Syntax Example.

Command SEARCH

Operation Searches the memory contents and displays the matching or unmatching data, if any.

Syntax S[/*mode*][/D] ***beg_addr,end_addr,search_data***

Terms *mode* = W (perform word search) or B (perform byte search).

D = Search for unmatching data (if omitted, search is made for matching data).

beg_addr = Address to begin search.

end_addr = Address to end search.

search_data = Data to search for.

Syntax Example S/D 100,7FF,55

Notes This command displays items on a line-for-line basis. To control the scrolling of the display, alternately press the space bar. To exit the display, press the Escape (Esc) key.

Spacing: A space is required before *beg_addr*. No spaces are permitted where the commas act as separators.

Command Example See Syntax Example. This example illustrates that a search of the memory contents is made from address 100H to address 7FFH. The display will show all locations that contain data other than 55.

Command SUPERVISOR: Specification

Operation Provides a way to access the ICD's serial ports (TERMINAL or HOST/AUX) from the emulated program by using specified breakpoints as supervisor calls to the ICD system.

The breakpoints in the emulated program do not stop the program being emulated, but perform input/output to the ICD serial interface only.

IMPORTANT! Do not use the HOST/AUX port to output data during a supervisor call if ZICE software is being used, or the communication protocol will be disturbed.

Syntax SU[*/break switch*]

Terms *break* = C, 7, or U

switch = ON or OFF

Syntax Example

```
SU/7 ON
SU
```

Notes C specifies to use hardware breakpoint C as a supervisor call, 7 specifies to use software breakpoint 7 as a supervisor call, and U specifies to use a user software breakpoint as a supervisor call. ON enables the specified breakpoint (C, 7, or U), and OFF disables it.

If a user software breakpoint is specified, the supervisor call will occur at each user software breakpoint. In this way, multiple calls can be used throughout a program.

The function code of the supervisor call is specified in the DL register, and the I/O data is transferred via the AL register.

Omitting all parameters will display the current supervisor call settings.

Spacing: A space is required between *break* and *switch*. No spaces are permitted before *break*.

Command Example

Execute this sequence:

```
>R RESET <—— resets the registers to 0
>R CS=0 <—— sets the CS register to 0
>A 100  <——— starts assembling the sample
          program from address 100H
0000:0100 MOV SI,120H
0000:0103 MOV DL,2
0000:0105 CLD
0000:0106 LODSB
0000:0107 OR AL,AL
0000:0109 JNZ 106H
0000:010B HLT
0000:010C <——— <return> here to terminate input
>
>B S=EN <— enables all software breakpoints
>B/C EX 0:107 <——— sets hardware breakpoint C at address
                    107H
>SU/C ON <———— uses breakpoint C as a supervisor call
>F 120,139,'THIS IS A SUPERVISOR CALL ' <— call message
>F 13A,143,'MESSAGE',0D,0A,00 <— call message
>G 100 <——— runs program from address 100H

THIS IS A SUPERVISOR CALL  <—ICD  issues message
                                  then stops at
                                  breakpoint C

0010B   F4                                HLT
        CS  IP ODITSZAPC AX  BX  CX  DX  SP  BP ...
0000:010B 000001010 0000 0000 0000 0002 0000 00 ...
<Break  Software User>
>
```

Supervisor Function Code Key

Port Input Status Fetch

Entry Conditions:
Register DL = 01H Get input status from TERMINAL port
Register DL = 11H Get input status from HOST/AUX port

Exit Conditions:

Register DL = Unchanged
Register AL = 0H No data is available at specified port
Register AL = FFH Data has been received at specified port

Input Character From Port

Entry Conditions:
Register DL = 00H Input character from TERMINAL port
Register DL = 10H Input character from HOST/AUX port

Exit Conditions:
Register DL = Unchanged
Register AL = Character received from specified port

NOTE: If no character is available at the specified port, control will not return from the supervisor call until acharacter has been received.

Port Output Status Fetch

Entry Conditions:
Register DL = 03H Get output status from TERMINAL port
Register DL = 13H Get output status from HOST/AUX port

Exit Conditions:
Register DL = Unchanged
Register AL = 00H Port transmit buffer is busy (not ready)
Register AL = FFH Port transmit buffer is empty (ready)

Output Character from Port

Entry Conditions:
Register DL = 02H Output character to TERMINAL port
Register DL = 12H Output character to HOST/AUX port

Exit Conditions:
Register DL = Unchanged
Register AL = Unchanged

NOTE: If transmit buffer is busy when this call is made, control will not be returned until buffer is ready and character has been sent.

	FUNCTION CODE		DATA OUT	DATA IN
FUNCTION	DL-reg		AL-reg	
TERMINAL	Port data in	00	-	receive data
HOST/AUX	Port data in	10	-	receive data
TERMINAL	Port input status read	01	-	input status
HOST/AUX	Port input status read	11	-	input status
TERMINAL	Port data out	02	output data	-
HOST/AUX	Port data out	12	output data	-
TERMINAL	Port output status read	03	-	output status
HOST/AUX	Port output status read	13	-	output status

Command TRACE: Status

Operation Displays the current trace setting.

Syntax T

Command Example

Execute the following:

```
>T <———————— displays the current trace
Trace is Clear <—— shows inactive trace
>T A <———————— sets trace to all display
>T <———————— displays new trace setting
(ON)  ALL 0000:0000-FFFF:000F (0000-FFFFF)
>T J <———————— sets trace to jump only display
>T <———————— displays new trace setting
(ON)    ALL 0000-FFFF <— shows jump specification
```

Command	TRACE: Qualification
Operation	Enables, disables, or clears the trace setting. Applications Note: This command can be used to temporarily disable the software trace feature without affecting its location within the program or the parameter specifications.
Syntax	T *switch*
Terms	*switch* = ON, OFF, or CLR
Syntax Example	T ON
Notes	If ON is specified, the trace specification is valid. If OFF is specified, the trace specification is disabled. If CLR is specified, the trace specification is cleared. Spacing: A space is required between T and *switch.*
Command Example	See the Syntax Example and the "TRACE: Specification" command.

Command

TRACE: Specification

Operation

Performs a software trace of the program in non-real time.

Applications Note: This command allows a section of the user program to be displayed in a step-by-step manner by either automatically scrolling through the program, or moving through the program one line at a time.

Syntax

T[/S] ***mode*[,*beg_addr*][,*end_addr*]**

Terms

S = Single step mode.

mode = A or J

beg_addr = Beginning address of memory to trace (default = 0).

end_addr = Ending address of memory to trace (default = FFFFF).

Syntax Example

T/S J,100,300
T A,200,FFF

Notes

S causes a single instruction to be executed eachtime the space bar is pressed. The *mode* must be defined as either A or J. A means that all commands are traced and displayed, and J means all instructions are traced but only Jump instructions are displayed.

If *beg_addr* is omitted, the trace starts from address 0H. If *end_addr* is omitted, the trace ends at address FFFFFH. When *beg_addr* or *end_addr* is specified, all the instructions are traced, but only the instructions within the specified address range are displayed. The instructions that are located outside of the address parameters are executed in non-real time as well.

Spacing: A space is required between T and *mode* (orT/S and *mode*). No spaces are permitted where commas act as separators.

Command Example

Execute this sequence:

```
>F 0:100,2FF,90 <— fills memory with NOPs
>F 0:300,,E9 <—— fills one byte with a jump
                 instruction
>D 0:0,300 <— displays memory to address 300H
>T A <— traces and displays all instructions
>G 0:100 <— displays all of program as it runs
>T A 100,11F <— traces all instructions from
        address 100H to 11FH
>G 0:100 <— displays program per trace
specification
>T/S A 100,120 <— traces all instructions from
                  address 100H to 120H and
                  displays one line at a time
>G 0:100 <— displays one instruction line each
time space bar is pressed
>T J <—— displays only jump instructions
>G 0:100 <— runs program and displays jump
        instruction
>T CLR <—— clears the trace feature
```

This example first fills a range of memory with NOPs so that a trace can be performed on the data. After the data is entered it is inspected, and then the trace parameters are specified. The first trace is of allinstructions, the second trace is of all instructions from address 100H to 11FH, the third trace is of all instructions and display is line-by-line, and the fourth trace is of Jump instructions only. Finally, the trace feature is cleared from the ICD memory.

Command USER

Operation Allows a single console terminal to communicate with either the ICD or a host computer.

Applications Note: This command enables the ICD to assume a "transparent" condition when it is positioned between a console terminal and a host computer. In this mode, a console terminal (connected to the ICD's TERMINAL port) can communicate directly with a host computer (connected to the ICD's HOST/AUX port). Essentially, the transparent mode uses the ICD as an interface or conduit between the two ports.

Syntax U [*code*]

Terms *code* = A single printing character used to signal the ICD to terminate the transparent communication mode. Control returns to the ICD command mode when this character is entered from the terminal's keyboard.

Notes The Terminal-to-ICD baud rate should be at leastdouble that of the ICD-to-Host baud rate (recommended: host computer=9600; terminal=19,200).

U initiates the transparent mode and U *code* terminates the transparent mode.

Spacing: A space is required between U and *code*.

Command Example

```
U
U !
U ~
```

Command VERIFY

Operation Compares an Intel Hex format file on the host computer to the ICD memory (or through the ICD to the target memory).

NOTE: All parameters and uses are identical to the LOAD command, with the exception that the VERIFY command does not alter memory; it only compares the memory contents against the file and displays the difference.

Syntax V[/source] filename[.ftype][,bias][,message]

Terms source = T, P, A, or H.

filename = Name of the file to download to the ICD.

.ftype = Optional three letter filetype (.abs is the default).

bias = Memory address offset to be added to the object file being compared (default is 0).

message = Any ASCII message (in 'single' quotes) or hex data, or any combination separated by commas.

Syntax Example V/H TEST.HEX,100

Notes T specifies to use the TERMINAL port and X-ON/X-OFF protocol.
P specifies to use the TERMINAL port and software protocol.
A specifies to use the HOST/AUX port and X-ON/X-OFF protocol.
H specifies to use the HOST/AUX port and software protocol. (See software specifications in Section 4 for a description of the software protocol.)

The *message* is sent out the source port at the beginning of the verify operation to provide a way of prompting the host computer to verify a file.

If *source* is omitted, command defaults to H in the REMOTE (host computer controlled) mode and T in the LOCAL (terminal controlled) mode.

See the LOAD command Notes for additional information.

Spacing: A space is required before filename; no spaces are permitted where commas act as separators.

Command Example

See Syntax Example and the LOAD command examples for additional information.

Command Syntax Summary

ALLOCATION
AL
AL *beg_addr*[,*end_addr*]=*block#*

ASSEMBLE
A *mem_addr* <cr>
xxxx:xxxx (80186/80188 assembly instruction) <cr>
xxxx:xxxx <cr> ...

BREAK
B [INI]
B[/*name*] *switch*
B[/*name*] *status,addr*[/*segment*][,*passcount*]
B[/*name*] *addr*[,*passcount*]
B[/*name*] *processor*
B S=*switch*
B S=*op_code*
B/X *edge*[,*passcount*]
B/X *switch*
B/E *switch*
B/E *passcount*
B/W *switch*
B/T *switch*

CALCULATION
C *operand*[+/-*operand n*]

COMPARE
CO *beg_addr,end_addr,comp_addr*[,*direction*]

DISASSEMBLE
DI [*beg_addr*][,*end_addr*]

DUMP
D[/W] *beg_addr*[,*end_addr*]

EVENT
EV [*switch*]
EV [ST=*status*][,A=*addr*][,D[/*mode*]=*data*][,M=*proc*][,CNT=*cnt*][,LE=*edge*][,ED=*edge*]

EXAMINE
E[/W][/N] *beg_addr*[=*mod_data*]

FILL
F[/W][/N] *beg_addr,end_addr,data*

GO
G [*beg_addr*][,*end_addr*][,*end_addr#2*]

HISTORY
H [CLR]
H *mode*[,*int_point*][,*term_point*]
H *mode*[,*range*]
H S,[*int_point*][,*term_point*][,ST=*status*][,A=*addr*][,D=*data*][,T]

HOST
H *switch*

IDENTIFICATION
ID

IN-CIRCUIT
I [*mode*]

LOAD
L[/*source*] *filename*[,*ftype*][,*bias*]

MAP
MA [*beg_addr*[,*end_addr*]=*area*]

MOVE
M *beg_addr,end_addr,mov_addr*[,*direction*]

NEXT
N [*steps*]

OFFSET
O [*&number=addr*]
PIN
PI [*signal=switch*]

PORT
P[/W] *port_addr*[=*mod_data*]

PRINT
PR *switch*

REGISTER
R [RESET]
R[/*mode*] *reg_name*=[*data*]

RESET
RES

SAVE
S[/*destination*] *filename*[,*ftype*],*beg_addr*,*end_addr*,*entry_addr*

SEARCH
S[/*mode*][/D] *beg_addr*,*end_addr*,*search_data*

SUPERVISOR
SU[/*break switch*]

TRACE
T [*switch*]
T[/S] *mode*[,*beg_addr*][,*end_addr*]

USER
U *code*

VERIFY
V[/*source*] *filename*[,*ftype*][,*bias*]

NOTE: Items in brackets ([]) are optional.

Introduction

In this section you'll learn about the eight internal control modules (including the optional Expansion Memory module) which, with the power supply, make up your ICD. These modules are used to control the various processes that are required for emulation, including electronically substituting your target system's microprocessor with the ICD's processor, controlling communication between the ICD and host computer or terminal, and tracing (and storing) a portion of the program memory contents for analysis.

Special Environments

Although it's not necessary to read this section to use your ICD, you may find the information helpful if you require an examination of how the ICD operates under special conditions and in particular environments. In certain instances, modules may need to be modified to permit the ICD to operate at peak performance. All possible modifications are detailed on the following pages.

In order to modify the components and controls, or to change certain settings on the modules, the ICD must be partially or fully disassembled. At the end of this section is a procedure which explains how to disassemble your ICD and remove (and replace) the eight control modules.

IMPORTANT!

✳ This symbol defines the adjustments and modifications to the ICD which are permitted under the Warranty Policy. In order to preserve the warranty on this equipment, do not adjust, modify, and/or in any way alter the controls or components on the modules unless the written procedure for manipulating a particular module is marked by this symbol.

Overview: The Eight Control Modules

Indicator/Control Module

This module contains the Operator Panel switches, indicator lamps and logic-state analyzer interface connector. All controls are externally accessible. (There are no user-serviceable controls on this module.)

Serial Interface Output Module

This module contains the RS-232 serial interface connectors for the TERMINAL and HOST/AUX ports. A 20mA current loop or TTL level terminal may also be used by changing the configuration of this module. (There are several user-serviceable controls, components, and switches on this module - see "How To Disassemble Your ICD," at the end of this section, after reading about the module's components on one the following pages.)

Expansion Memory Module

This optional module expands the ICD's memory capabilities to 256K bytes (128K standard + 128K expansion). Components on this module include a 60-pin bus receptacle to connect with the Memory Mapping Unit module, and two 8-bit switches that control the module's functions. (To gain access to these components, see "How To Disassemble Your ICD," at the end of this section, after reading about the module's components on one of the following pages. To install the module, see the chapter on the Expansion Memory module.)

Break Comparator Memory Module

This module qualifies the conditions (address, data, status) for the BREAK command. (There are no user-serviceable controls on this module.)

CPU Control Module

This module contains the connectors, circuitry, CPU (80186/80188) processor, and NDP (8087) co-processor, which allow the ICD to emulate the target system's processors. (There are a few user-serviceable components on this module - see "How To Disassemble Your ICD," at the end of this section, after reading about the module's components on one of the following pages.)

Emulator Control Module

This module controls the emulation mode or monitor mode of operation for the ICD. (There are no user-serviceable components on this module.)

Real-time Storage Module

This module's circuitry includes the controller, memory, and real-time counter for performing tracing and storage of the user (There are no user-serviceable components on this module.)

Memory Mapping Unit Module

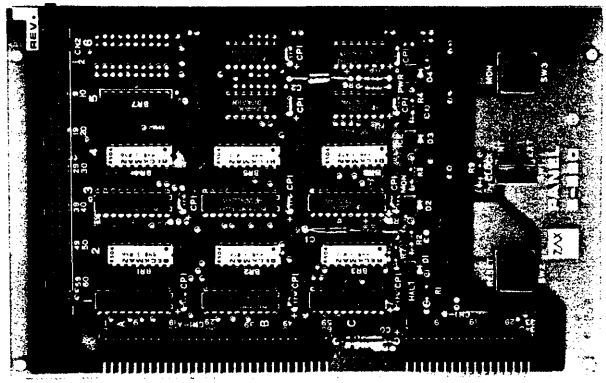
This module contains 128K bytes of high-speed static RAM (known as "emulation memory"), which can be used for downloading files, altering the memory contents, and loading future memory into the target system. (There are a few user-serviceable components on this module - see "How To Disassemble Your ICD," at the end of this section, after reading about the module's components on one of the following pages.)

Indicator/Control Module

Description

The Indicator/Control module contains three switches, four indicator lamps, one 60-pin bus receptacle, the logic-state analyzer interface connector and intermediary circuitry. Switch SW1 selects between the internal (INT) or external (EXT) clock; switches SW2 and SW3 activate the RESET and MONITOR functions, respectively. The indicator lamps D1, D2, D3, and D4 show the condition of the HALT, MONITOR, ICE (in-circuit enable), and POWER functions.

The three switches, four indicator lamps and logic-state analyzer connector are all accessible for operation (and viewing) from outside the ICD; there are no user-serviceable controls or components on this module.



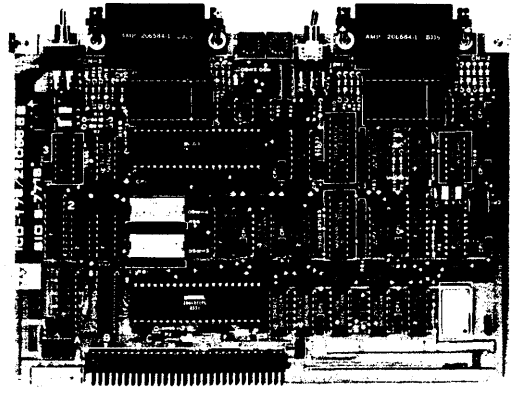
Serial Interface Output Module

Description

The Serial Interface Output (SIO) module controls communication between the ICD and various external devices (host computer, terminal, printer) through the TERMINAL and HOST/AUX ports. The SIO module's internal components feature jumper sockets and line drivers that can be modified to permit either RS-232, current loop, or TTL interface operation. There are also two transmission format switches (DSW3 and DSW4) that are used to set the data format and stop bits for the TERMINAL and HOST/AUX ports, and a special socket that allows any key on the console keyboard to activate the MONITOR break switch in the ICD.

These components are all user-serviceable; the ICD must be disassembled before they can be adjusted or modified. (See "How To Disassemble Your ICD," at the end of this section).

The module's remaining components are all externally accessible. These include the DCE/DTE and LOCAL/REMOTE switches, the TERMINAL and HOST/AUX port connectors, and two rotary switches that set the communication baud rates for the ports.



SIO S-791 Module Components

1) JA Socket. By connecting different pins with jumpers, this socket is used to select either RS-232, current loop, or TTL interface for the TERMINAL port.

2) JB Socket. Used the same way as the JA socket, but selects the interface for the HOST/AUX port.

3) TERMINAL Port Line Driver. The standard line driver is an SN75188, and is used with RS-232 and current loop interface operation. When TTL interface is used, the standard line driver must be replaced with an SN7438 line driver.

4) HOST/AUX Port Line Driver. Functions the same as the TERMINAL port line driver, except controls the HOST/AUX port.

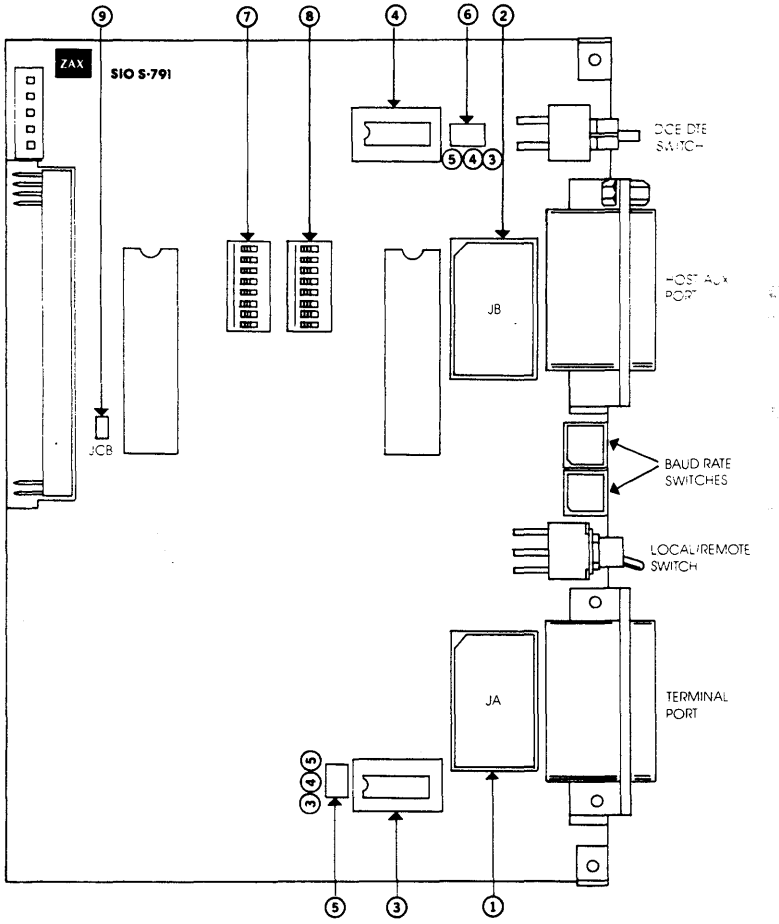
5) JA 5/4/3 Power Supply Jumpers. Supplies power to the TERMINAL port line drivers. Pins 3 and 5 supply +12V to the SN75188 line driver (when using RS-232 or current loop interface), and Pin 4 supplies +5V to the SN7438 line driver (when TTL interface is used).

6) JB 5/4/3 Power Supply Jumpers. Functions the same as JA 5/4/3, but supplies power to the HOST/AUX port line driver.

7) DSW3 Transmission Format Switch. Sets the data format and stop bits for the TERMINAL port. (See "How To Set The Transmission Format Switches.")

8) DSW4 Transmission Format Switch. Sets the data format and stop bits for the HOST/AUX port (See "How To Set The Transmission Format Switches".)

9) JCB Console Break Jumper Socket. When the pins of this socket are connected together, it allows any key on the terminal keyboard to activate the MONITOR break switch; it is essentially the same as pressing the MONITOR switch on the ICD. (The MONITOR switch is used to return control to the ICD monitor during emulation.)


ZAX
SIO S-791
JCB
JB
JA
DCE DTE SWITCH
HOST AUX PORT
BAUD RATE SWITCHES
LOCAL/REMOTE SWITCH
TERMINAL PORT

Baud Rate Switches

The Baud Rate switches are used to set the baud rates for the TERMINAL and HOST/AUX ports. The factory setting is #1 (9600 bps) for both ports. There are 13 other baud rate settings available; do not set the baud rate switches to E or F.

✳ Changing The Baud Rate Settings

The Baud Rate switches are rotary-type switches. To change the baud rates, turn the dials to the number or letter shown in the Baud Rate diagram below. Use a pointed object such as a pen tip or a small screwdriver.

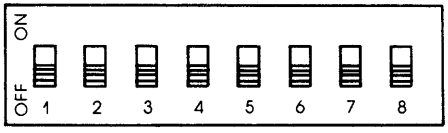
Baud Rate Switch No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Baud Rate (bps)	19.2K	9.6K	4.8K	2.4K	1.2K	600	300	150	75	110	134.5	200	1.8K	2K	—	—

✳ How To Set The Transmission Format Switches

The transmission format switches are used to set the data format and stop bits for the TERMINAL and HOST/AUX ports. Both 8-bit, On/Off type switches can be set by inserting a small, pointed tool and sliding the bits to the On or Off position.

Bit	Off	On
1	Data bit 8	Data bit 7
2	No parity bit	Enable parity bit
3	Even parity	Odd parity
4	Stop bit 2	Stop bit 1
5	Bit 8 always 0	Bit 8 always 1
6	Multi-ICD I/O disable	Multi-ICD I/O enable
7	Multi-ICD I/O disable	Multi-ICD I/O enable
8	TBMT & TEOC	TBMT only

Factory Settings All bits = Off



NOTE 1: When bit 8 is set to Off, the ICD transmits on a single buffer basis for monitoring the BUSY state. When this bit is set to On, the ICD transmits on a double buffer basis without monitoring the BUSY state.

NOTE 2: Facts about TBMT and TEOC signals:

TBMT - Transmitted Buffer Empty. The transmitted buffer empty flag goes to a logic "1" when the data bits holding register may be loaded with another character.

TEOC - Transmitted End of Character. This line goes to a logic "1" each time a full character is transmitted. It remains at this level until the start of transmission of the next character.

✳ Multiple ICDs

Signals for multiple ICDs can I/O through the HOST/AUX port by setting bits 6 and 7. When this feature is enabled, the External Break, Emulation Qualify, and Event Trigger signals can be monitored by more than one ICD. (I/O level is EIA.)

To activate this feature, set the following bits:

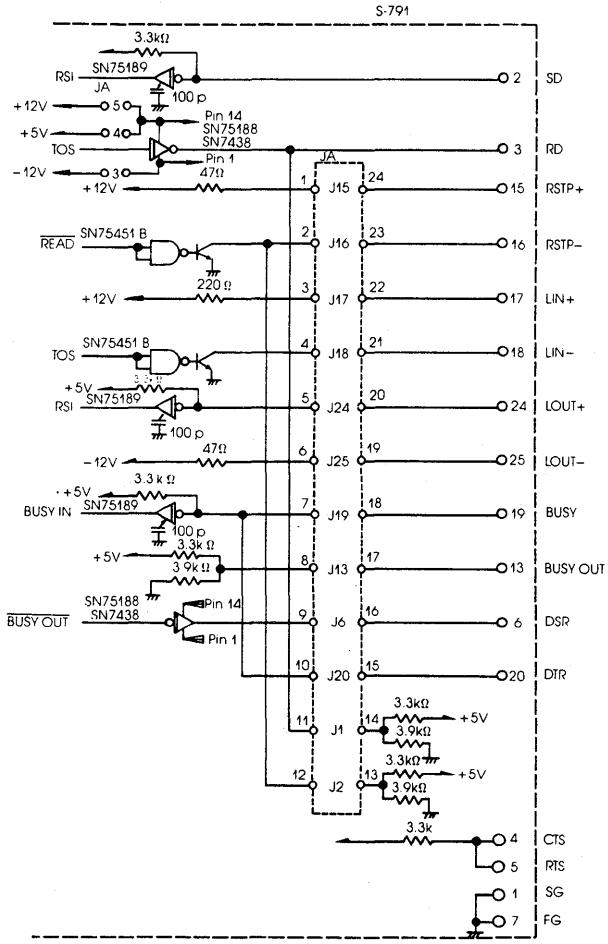
DSW3 bit 6 = On DSW4 bits 6 & 7 = On

This feature effects the following pins of the HOST/AUX port:

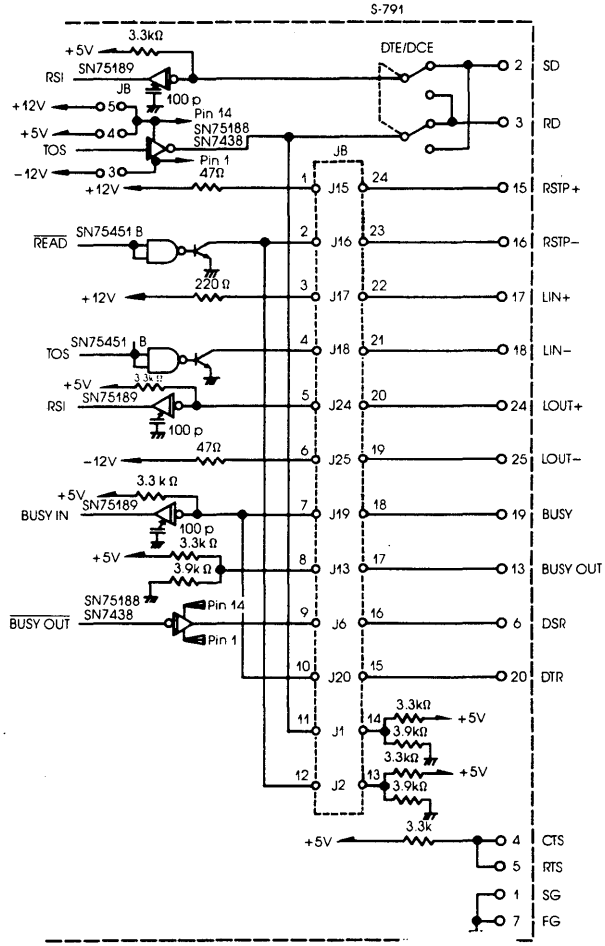
Pin No.	Signal Name	I/O
11	External Break	IN
18	Emulation Qualify	OUT
25	Event Trigger	OUT

NOTE: The multiple ICD feature is available on ICDs which use SIO module S-771B.

SIO Diagram Of
TERMINAL Port



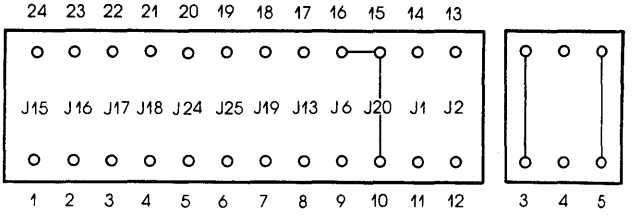
SIO Diagram Of HOST/AUX Port



RS-232 Interface

The RS-232 interface is the normal configuration for the ICD. The diagram below shows how the pins on the JA/JB sockets are arranged for the RS-232 setting. The tables show the status of the signals for both the TERMINAL and HOST/AUX ports.

RS-232 Pin Configuration
(Standard connection is shown)



RS-232 Interface I/O Signals - TERMINAL Port

PIN No.	SIGNAL NAME	DESCRIPTION	IN/OUT	JA No.
1	FG	Frame Ground		
2	SD	Send Data	IN	SN 75188N
3	RD	Receive Data	OUT	
4	RTS	Request To Send *2	IN	
5	CTS	Clear To Send *2	OUT	
6	DSR	Data Set Ready	OUT	
20	DTR	Data Terminal Ready	IN	J 6, J 20 *3
7	SG	Signal Ground		

RS-232 Interface I/O Signals - HOST/AUX Port

PIN No.	SIGNAL NAME	DESCRIPTION	IN/OUT	JB No.
1	FG	Frame Ground		
2	SD	Send Data	OUT (IN) *1	SN 75188N
3	RD	Receive Data	IN (OUT)	
4	RTS	Request To Send *2	OUT (IN)	
5	CTS	Clear To Send *2	IN (OUT)	
6	DSR	Data Set Ready	IN (OUT)	
20	DTR	Data Terminal Ready	OUT (IN)	J 6, J 20 *3
7	SG	Signal Ground		

NOTE 1: Values in () enabled when the DCE/DTE select switch is set to DCE.

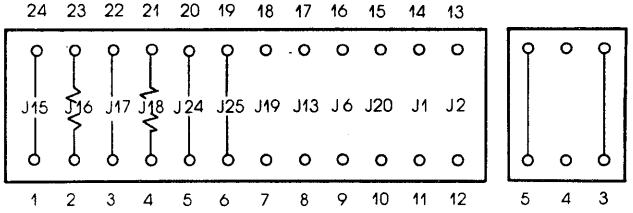
NOTE 2: CTS and RTS signals are looped back (null modem) within the ICD and pulled up to +5V.

NOTE 3: Connecting pins 15 and 16 (JA/JB socket) causes the DTR and DSR signals to be looped back (null modem) within the ICD.

NOTE 4: Connecting pins 10 and 15 (JA/JB socket) causes the DTR signal to be used as the BUSY signal to the terminal. Connecting pins JA6/JB6 causes the DSR signal to be used as the BUSY signal to the terminal.

Current Loop Interface

The current loop interface is an optional configuration that is enabled when the JA and JB sockets are modified. The diagram below shows how the pins on the JA/JB sockets are arranged for the current loop setting. The table shows the status of the signals for both the TERMINAL and HOST/AUX ports.



✳ Using The Current Loop Interface

a) Connect pin 4 to pin 21 (JA18/JB18) with a 220ohm, 1/4 watt resistor, or adjust the resistance to the associated circuit.

b) Connect pin 2 to pin 23 (JA16/JB16) with a 47 ohm, 1/4 watt resistor.

c) Connect the other pins as shown in the Current Loop Interface diagram.

d) Set the ICD's DCE/DTE select switch to DCE.

e) Adjust the baud rates for the TERMINAL and HOST/AUX ports to a maximum of 600 bps.

NOTE: Do not change the jumpers on the line driver power supply (JA3/JB3, JA5/JB5).

Current Loop Interface I/O Signals -
TERMINAL & HOST/AUX Ports

PIN NO.	SIGNAL NAME	DESCRIPTION	IN/OUT	JA/JB No.
24	LOUT+	Current Loop OUT(+)	IN	J 24
25	LOUT–	Current Loop OUT(–) *1	IN	J 25
17	LIN+	Current Loop IN(+) *2	J 17	
18	LIN–	Current Loop IN (–)	OUT	J 18 220 Ω
15	RSTP+	Reader Step (+)	OUT	J 15
16	RSTP–	Reader Step (–)	OUT	J 16 47 Ω

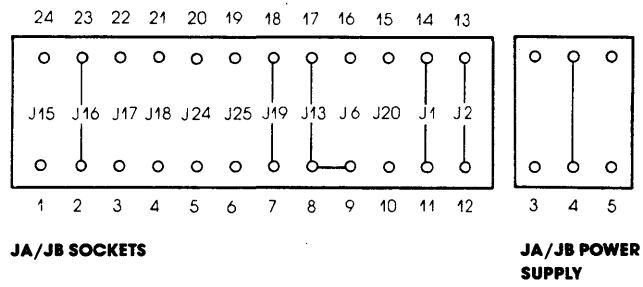
NOTE 1: Pin 25 is the current source pin for current loop input signals pulled down to -12V.

NOTE 2: Pin 17 is the current source pin for current loop input signals pulled up to +12V.

TTL Interface

The TTL interface is an optional configuration that is enabled when the JA/JB sockets are modified. The diagram below shows how the pins on the JA and JB sockets are arranged for the TTL Interface setting. The table shows the status of the signals for both the TERMINAL and HOST/AUX ports.

TTL Interface
(Modified connection is shown)



✳ Using The TTL Interface

a) Remove the jumpers from JA3/JB3 and JA5/JB5 of the line driver power supply, and insert a single jumper into JA4/JB4.

b) Connect the pins as shown in the TTL Interface diagram.

TTL Interface I/O Signals - TERMINAL Port

PIN No.	SIGNAL NAME	DESCRIPTION	IN/OUT	JA No.
1	FG	Frame Ground		
2	SD	Send Data	OUT (IN) *1	SN 7438
3	RD	Receive Data	IN (OUT)	
19	BUSY	BUSY Input	IN	J 19
13	BUSYOUT	BUSY Output	OUT	J 13, J 6 *2
16	RSTP	Reader Step	OUT	J 16
7	SG	Signal Ground		

TTL Interface I/O Signals - HOST/AUX Port

PIN No.	SIGNAL NAME	DESCRIPTION	IN/OUT	JA No.
1	FG	Frame Ground		
2	SD	Send Data	IN	SN 7438
3	RD	Receive Data	OUT	
19	BUSY	BUSY Input	IN	J 19
13	BUSYOUT	BUSY Output	OUT	J 13, J 6 *2
16	RSTP	Reader Step	OUT	J 16
7	SG	Signal Ground		

NOTE 1: Values in () enabled when the DCE/DTE select switch is set to DCE.

NOTE 2: Connecting pins 8 and 9 (JA/JB socket) causes the DTR signal to be used as the BUSY signal to the terminal.

Serial Interface Control Signals

XON and XOFF Protocol

XON/XOFF allows terminals or host computer systems to receive data from the ICD even if the baud rates between these devices are different.

The XON/XOFF protocol works in the following manner:

1. The host computer or terminal sends XOFF to the ICD before the reception buffer overruns.

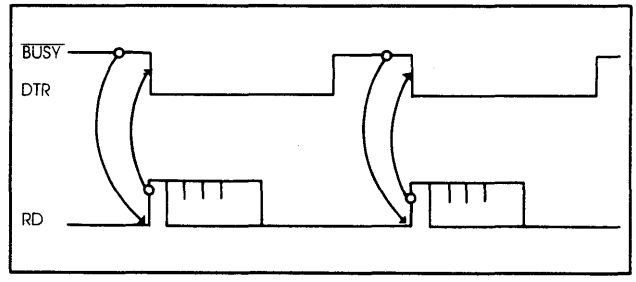
2. When the reception buffer is ready, the host computer or terminal sends XON to the ICD and resumes reception.

The control codes for XON/XOFF signals are:

XON - DC3 (CTRL-S: 13H)
XOFF- DC1 (CTRL-Q: 11H)

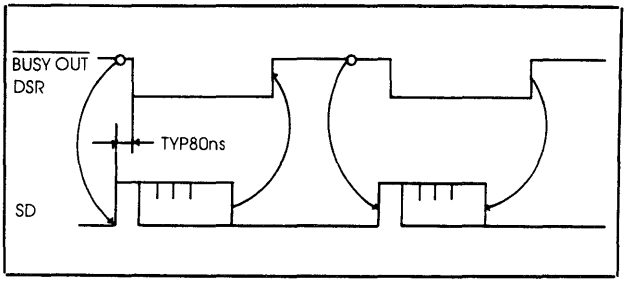
BUSY and DTR Input Signals

The BUSY signal sent from a low-speed terminal can be used to stop the ICD from transmitting data. Under normal conditions, the terminal sets the BUSY signal to low, from the leading edge of the RD signal starting bit, to the completion of data processing. The ICD suspends data transmission to the terminal as long as the BUSY signal is low.



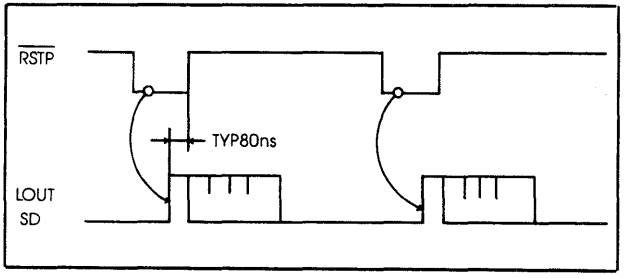
BUSYOUT and DSR Output Signals

When a host computer sends data at a higher speed than the ICD's internal monitor processor can accept, the BUSYOUT signal from the ICD must be monitored. The ICD sets the BUSYOUT signal to low until the ICD monitor reads the SD signal from the host computer.



RSTP Output Signal

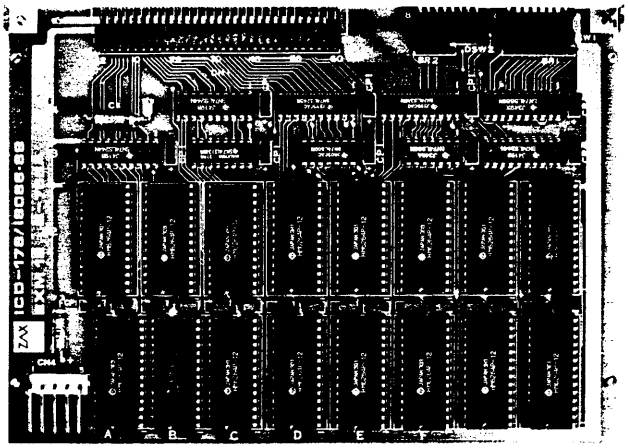
The ICD can transmit the RSTP signal to terminals that require a step signal for each data transmission. The ICD sets RSTP to low when it requests data to be read, and then returns RSTP to high when it detects the start bit signal from the terminal.



Expansion Memory Module

Description

The optional Expansion Memory module expands the ICD's memory capabilities to 256K bytes (128K standard + 128K expansion). Components on this module include a 60-pin bus receptacle to connect with the Memory Mapping Unit(MMU) module, a 5-pin power-supply connector, and two 8-bit switches that control the module's functions.



✳ Installing The Module

If your ICD does not contain the optional Expansion Memory module and you wish to install it, follow the procedure below:

a) Remove the ICD's top, bottom, and side covers as described in "How To Disassemble Your ICD," in this section. It is not necessary to remove the other modules from the mainframe to install the Expansion Memory module.

b) Slide the Expansion Memory module into the open slot that is located just below the SIO module. Position the module so that it fits between the aligning tabs, then fit the two small screws which attach the module to the mainframe.

c) Connect the power-supply socket to the module. (Use a pair of needle-nose pliers to push the socket onto the module's 5-pin plug.)

d) Connect the auxiliary bus cable to the module's 60-pin bus receptacle.

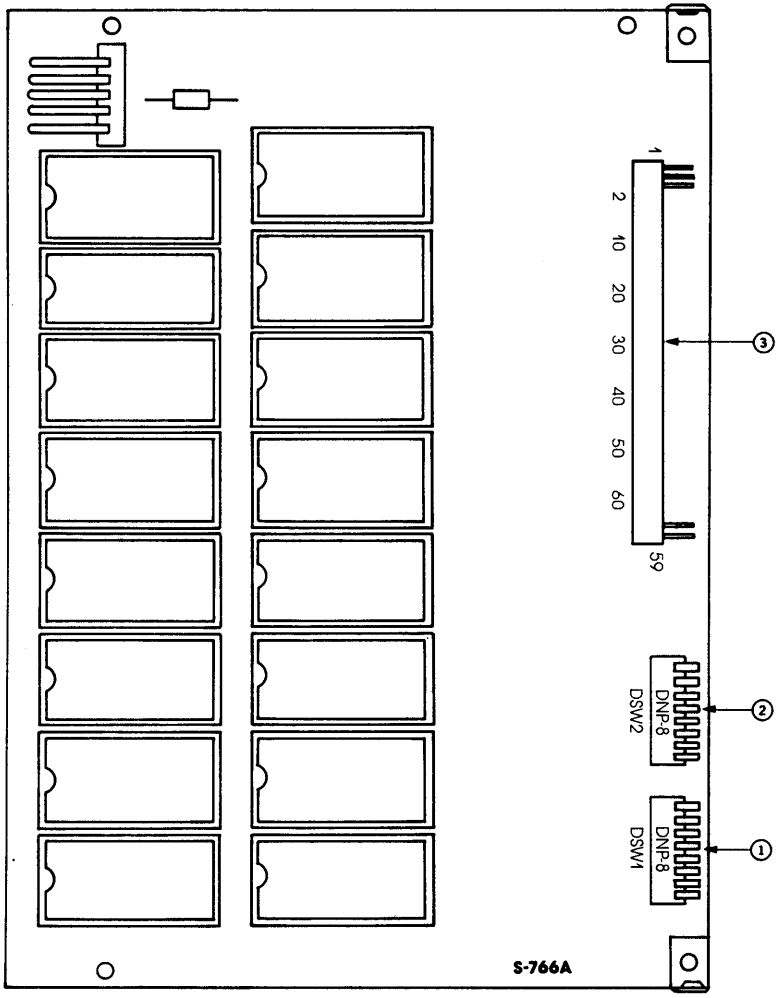
e) Attach the other end of the auxiliary bus cable to the 60-pin bus receptacle on the MMU module, located on the bottom of the ICD mainframe.

f) Replace the top, bottom, and side covers on the ICD mainframe.

✳ EXM-12 Module Components

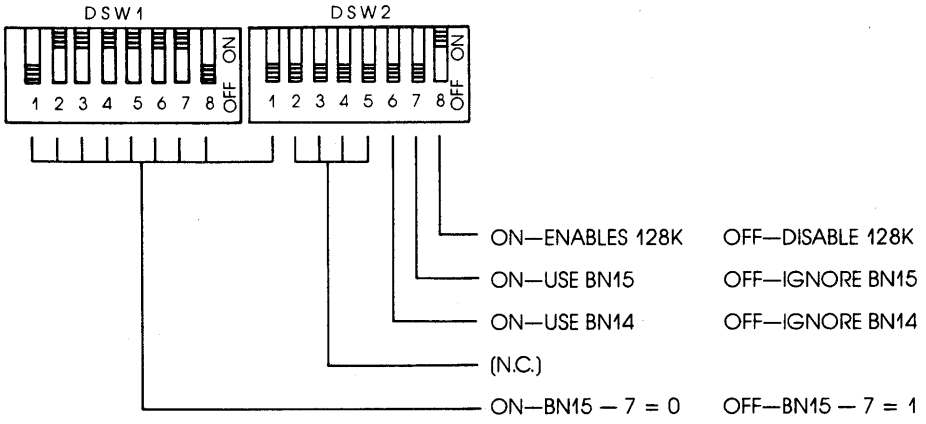
1) & 2) DSW1 & DSW2 ICD Program Memory Block Number switches. Sets the allocation block number of the 128K-byte Expansion Memory module in 1K-byte blocks. (For more information on memory allocation, see the ALLOCATION command in Section 2.)

2) EXM-12 60-pin Bus Connector. Connects the EXM-12 module with the MMU S-776 Memory Mapping Unit module.


1
2
10
20
30
40
50
60
59
3
DNP-8
DSW2
2
DNP-8
DSW1
1
S-766A

DSW1 & DSW2 Switch Settings

The following diagram shows the factory-adjusted switch settings for the DSW1 and DSW2 ICD Program Memory Block Number switches.



Notes on bit functions:

Bit 8 of DSW2 - Enables and disables the Expansion Memory module.

Bit 7 and bit 1 of DSW2 - When both bits are Off, the A25M address signal is suppressed. Signals A17M through A24M remain active.

Bit 6 of DSW2 and bit 7 of DSW1 - When both bits are Off, the A24M address signal is suppressed. Signals A17M through A23M and A25M remain active.

If bits 1 through 7 of DSW1 and bit 1 of DSW2 are On, block numbers 000-07F are selected. If bit 1 of DSW1 is Off and all other bits are On, block numbers 080-0FF are selected.

The following diagram shows the allocation block numbers and corresponding bit settings:

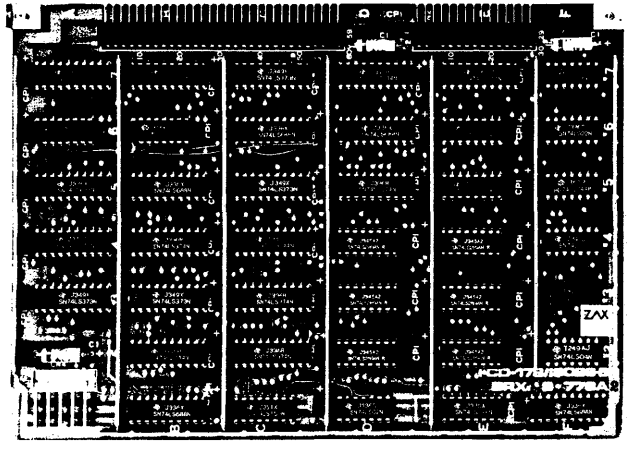
Bit 1 (DSW2) = BN 15 (A25M)
Bit 8 (DSW1) = BN 14 (A24M)
Bit 7 (DSW1) = BN 13 (A23M)
Bit 6 (DSW1) = BN 12 (A22M)
Bit 5 (DSW1) = BN 11 (A21M)
Bit 4 (DSW1) = BN 10 (A20M)
Bit 3 (DSW1) = BN 9 (A19M)
Bit 2 (DSW1) = BN 8 (A18M)
Bit 1 (DSW1) = BN 7 (A17M)

Break Comparator Memory Module

Description

The Break Comparator Memory module qualifies the conditions (address, data, status) for the BREAK command.

The BREAK command is used to control the functions of the Break Comparator Memory module; there are no user-serviceable controls or components.



CPU Contro Module

Description

The CPU Control module contains the connectors, circuitry, and 80186 or 80188 microprocessor and 8087 co-processor, which allow the ICD to emulate the target system's microprocessor and co-processor.

The user-serviceable but not externally accessible components on this module include: the CPU socket and NDP sockets, which contain the 80186 or 80188 processor and the 8087 processor; one of the three emulation method select switches; and various "jumpers" which can be used to alter clock speed, change wait states, and control signal timing.

To gain access to these components and/or change the processors, see "How To Disassemble Your ICD," located at the end of this section.

The remaining components on this module are externally accessible. These include the CPU probe connectors (which connect the ICD's internal processor to the target system), the NDP probe connector, and the two remaining emulation method select switches.

CPU Control Module Components

1) CPU Socket. Accepts either the 80186 or 80188 CPU.

2) NDP Socket. Accepts the 8087 Numeric Data Processor.

3) Internal Clock Jumper Pins. Changes the ICD's internal clock to either 8MHz or 4 MHz.

4) Wait State Jumper Pins. Inserts 1, 2 or 3 wait states into each of the SRDY and ARDY lines.

5) Timeout Jumpers. Sets the timeout (delay from ARDY/SRDY response) to 2048, 128 or 8 clock cycles.

6) VCCT Jumper. Provides a +5 volt reference line to the target system.

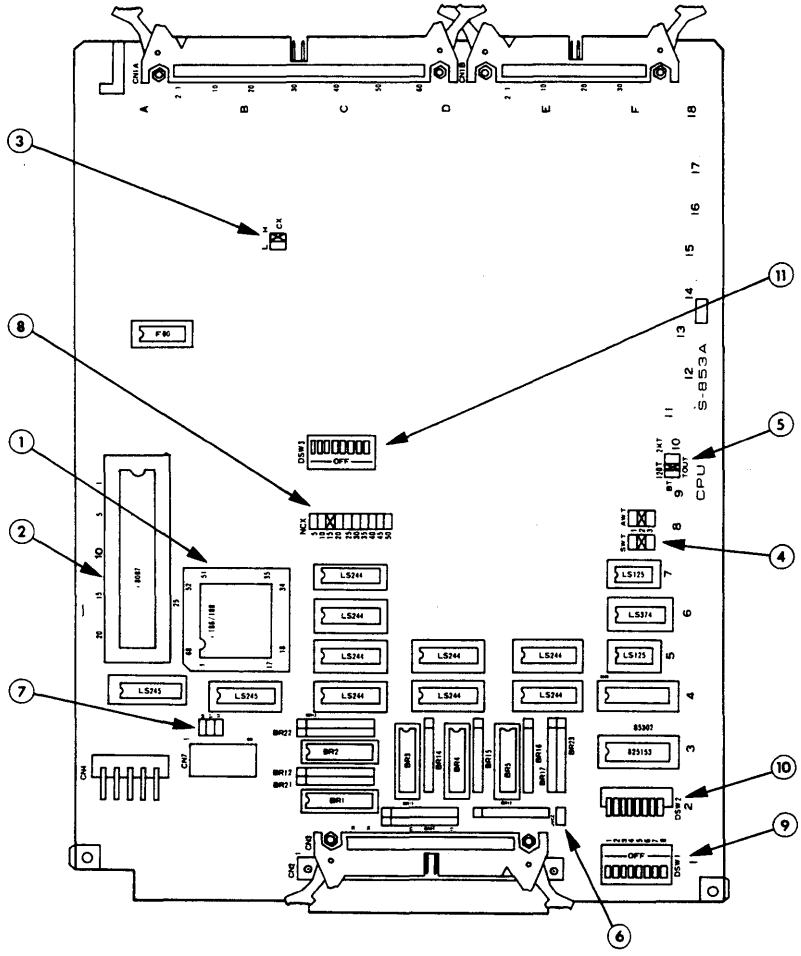
7) Status Jumpers. Controls output of the 80186 processor's S0, S1 and S2 status lines during an emulation break.

8) NCK Jumper. Permits synchronization of the clock signal between the ICD's CPU and NDP by adjusting the signal delay to the NDP from 5ns to 50ns.

9) Emulation Method Select switch #1. See the switch description in Section 1 and the information in this section.

10) Emulation Method Select switch #2. See the switch description in Section 1 and the information in this section.

11) Emulation Method Select switch #3. See the switch description in Section 1 and the information in this section.


3
11
8
5
1
4
2
7
10
9
6
S-853A
CPU
OFF

✳ Changing CPUs

To remove CPUs (80186 or 80188), carry out the following procedure:

a) Disassemble the ICD as shown at the end of this section, and then remove the CPU Control module from the ICD chassis.

b) Locate the square-shaped processor holder. To remove the existing CPU, press the retaining clip away from the cover plate and then lift the cover plate away from the holder.

c) Carefully grasp the edges of the CPU and lift the processor up and out of the holder.

To install CPUs (80186 or 80188), carry out the following procedure:

a) Locate the index marks on the CPU, then install the CPU in the holder with the index marks properly positioned and the contact side facing downward.

b) Reposition the cover plate onto the processor holder and press the cover plate firmly down to seat the CPU.

c) Press the retaining clip back onto the cover plate nibs.

d) Reinstall the CPU Control module and reassemble the ICD.

✳ Changing The NDP

To remove the NDP (8087), carry out the following procedure:

a) Disassemble the ICD as shown at the end of this section, and then remove the CPU Control module from the ICD chassis.

b) Locate the NDP socket. Slide the tab marked "On" into the socket until the tab marked "Off" appears on the other side of the socket.

c) Carefully remove the NDP from the socket.

To install the NDP (8087), carry out the following procedure:

a) Reinstall the NDP into the socket.

b) Push the tab marked "Off" into the socket until it locks into the original "On" position.

c) Reinstall the CPU Control module and reassemble the ICD.

✳ CPU Control Module Jumpers

The CPU Control module contains a number of plastic-encased, gold-plated connectors called "jumpers."The jumpers provide a convenient way of connecting (as opposed to soldering) the various pins on the module, which in turn control clock speeds, wait states, time outs, and signal I/O.

Internal Clock Jumpers. Sets the ICD's internal clock speed to either 4 or 8MHz. (The ICD internal clock is selected by setting the INT/EXT switch to INT.)

The internal clock normally runs at a speed of 8MHz with a 50% duty cycle, but it can be changed to 4MHz by modifying the jumper on the CPU Control module. The clock jumper is identified by CX, and the H and L jumpers specify the high (H = 8MHz) or (L = 4MHz) clock speed.

Settings: CX-L - Sets ICD's internal clock speed to 4MHz.
CX-H - Sets ICD's internal clock speed to 8MHz.

Factory Setting: CX-H

External Clock. Selecting the EXT setting on the INT/EXT switch enables the ICD to use an external clock. The external clock setting allows the peripheral LSI of the target system and the ICD's CPU to be synchronized for simultaneous operation.

NOTE: To ensure accurate operation of the ICD's CPU, a 50% duty cycle is required for high speed clocks greater than 4MHz.

Wait State Jumpers. Inserts 1, 2 or 3 wait states into each of the SRDY and ARDY lines. "SWT" controls the SRDY line and "AWT" controls the ARDY line. To activate feature, use the Emulation Method Select switch #2 and set bit 8 to the On position.

Settings: SWT-1 - Inserts 1 wait state into SRDY line.
SWT-2 - Inserts 2 wait states into SRDY line.
SWT-3 - Inserts 3 wait states into SRDY line. AWT-1 - Inserts 1 wait state into ARDY line. AWT-2 - Inserts 2 wait states into ARDY line. SWT-3 - Inserts 3 wait states into ARDY line.

Factory Settings: SWT-2
AWT-2

Timeout Jumpers. These jumpers set the (delay from ARDY/SRDY response) to 2048, 128 or 8 clock cycles. This constitutes a wait state. Wait states can be used to cause a break (using the "BREAK: Timeout" command) in the user program when the ICD is unable to access the target system contents within a certain time period.

Settings: TOUT-8T - Sets the timeout to 8 clock cycles.
TOUT-128T - Sets the timeout to 128 clock cycles.
TOUT-2KT - Sets the timeout to 2048 clock cycles.

Factory Setting: TOUT-128T

VCCT Jumper. Provides a +5 volt reference line to the target system.

Settings: VCCT (connected) - outputs +5 volts to the target system.

VCCT (open) - reference line is not connected to the target system.

Factory Setting: open (not connected)

Status Jumpers. Controls output of the 80186 processor's S0, S1 and S2 status lines during an emulation break.

Settings: S0 (connected) - Activates the S0 signal during an emulation break.

S1 (connected) - Activates the S1 signal during an emulation break.

S2 (connected) - Activates the S2 signal during an emulation break.

Factory Setting: all open (not connected)

NCK Jumpers. Permits synchronization of the clock signal between the ICD's CPU and NDP by adjusting the signal delay to the NDP from 5ns to 50ns.

Settings:
NCK-5 - Sets delay to 5ns.
NCK-10 - Sets delay to 10ns.
NCK-15 - Sets delay to 15ns.
NCK-20 - Sets delay to 20ns.
NCK-25 - Sets delay to 25ns.
NCK-30 - Sets delay to 30ns.
NCK-35 - Sets delay to 35ns.
NCK-40 - Sets delay to 40ns.
NCK-45 - Sets delay to 45ns.
NCK-50 - Sets delay to 50ns.

Factory Setting: NCK-15

✳ Emulation Method Select Switch #1

Description

The Emulation Method Select switch #1 is an 8-bit, On/Off type switch that controls signal I/O between the ICD and target system during emulation.

Location

The NDP/CPU connector end of the ICD. Switch #1 is located below switch #2. (See "The Controls And Component Functions Of Your ICD," earlier in this section.)

Function

See the individual bit settings that follow.

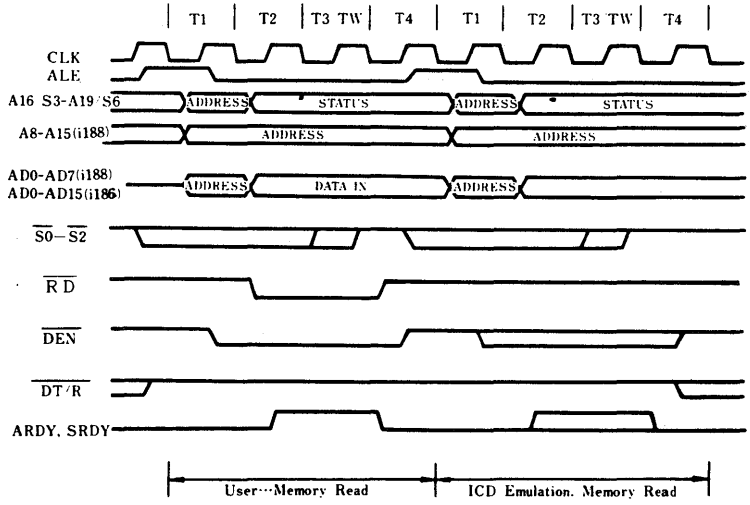
Adjustment

To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.

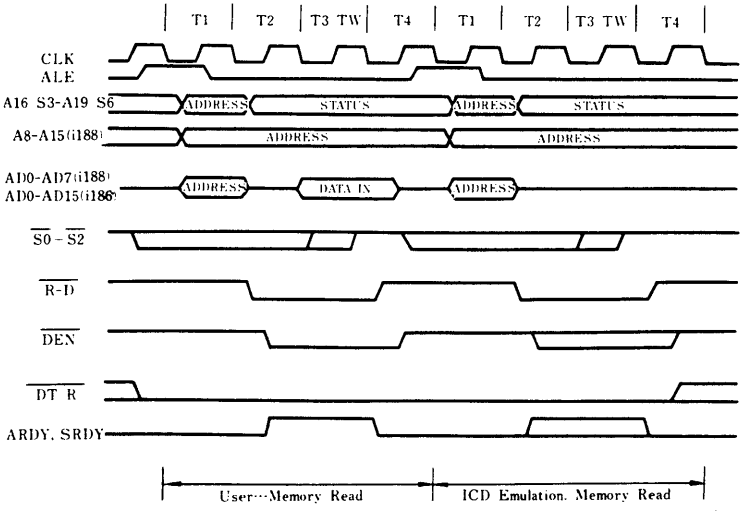
Factory Settings

All bits are Off.

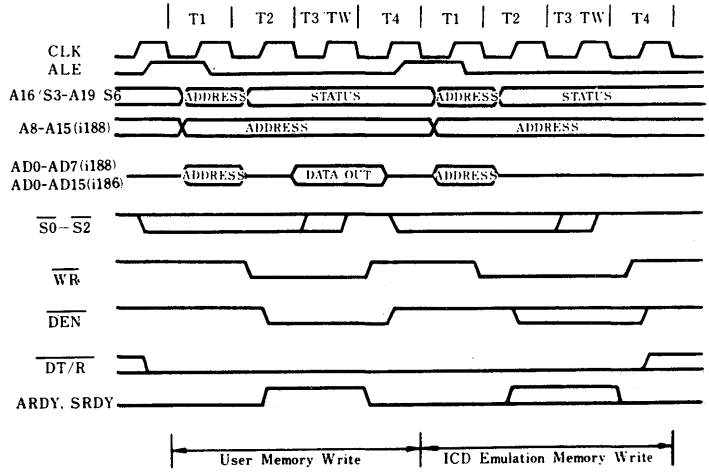
Bit 1: On - RD signal outputs during user memory access only.



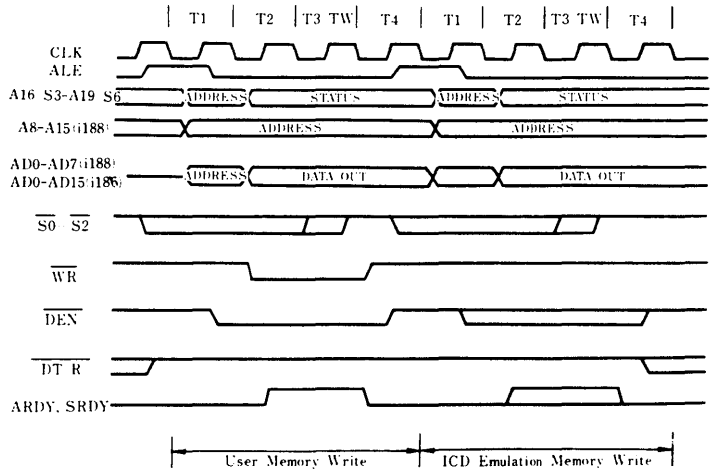
Bit 1: Off - RD signal outputs during both user and ICD memory access.



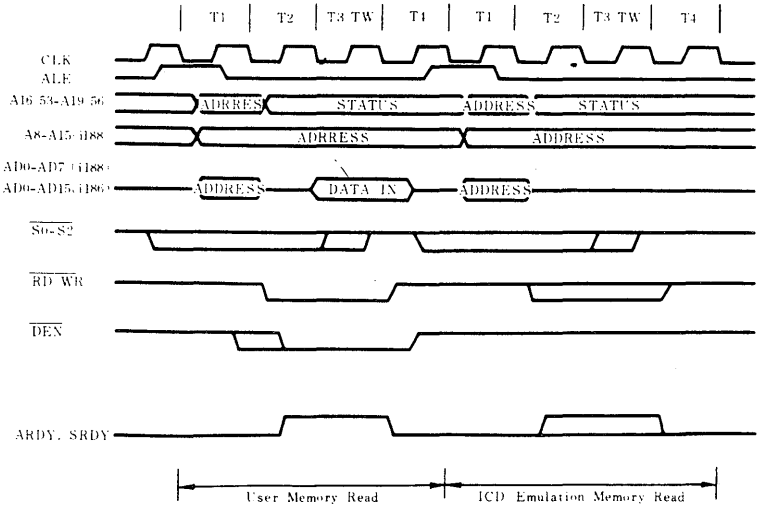
Bit 2: On - WR signal outputs during both user and ICD memory access.



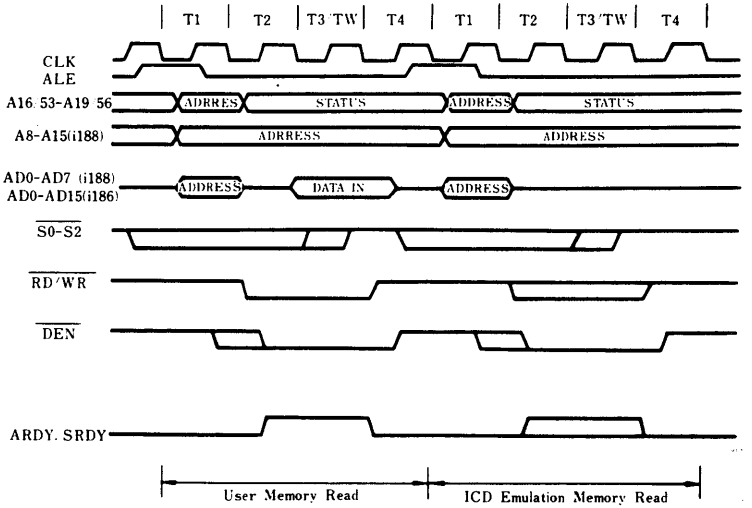
Bit 2: Off - WR signal outputs during user memory access only.



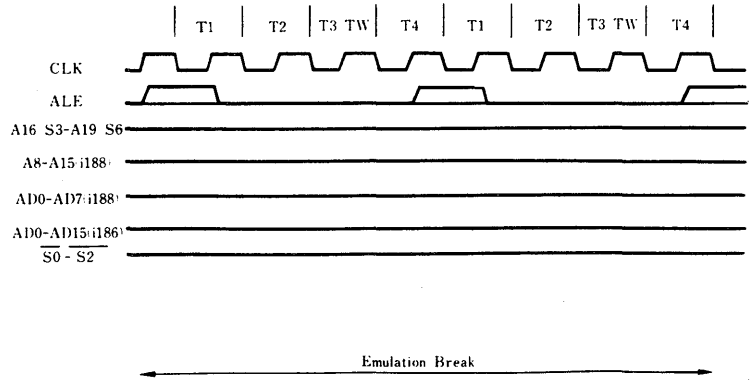
Bit 3: On - DEN signal outputs during user and ICD memory operations.



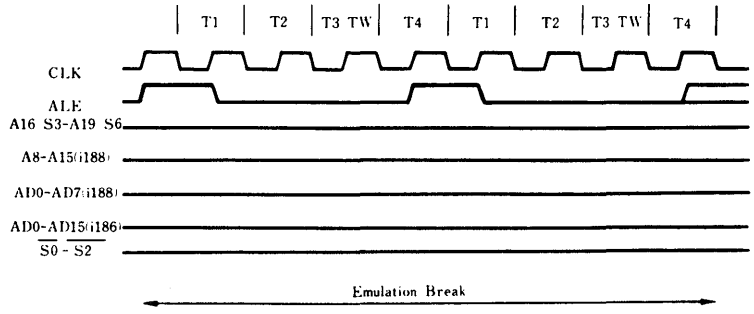
Bit 3: Off - DEN signal outputs during user memory access only.



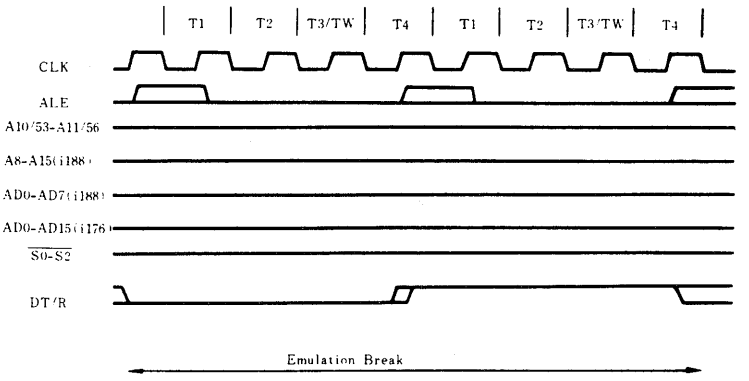
Bit 4: On - ALE signal outputs during emulation.



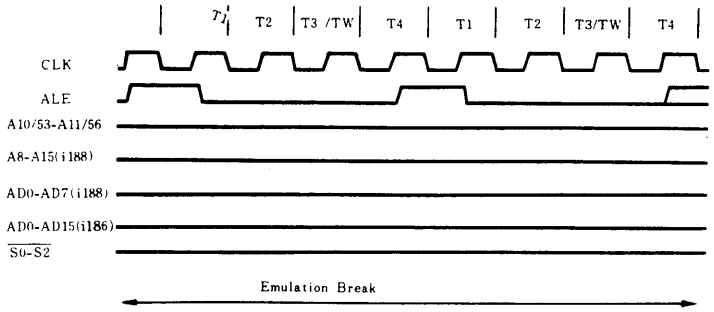
Bit 4: Off - ALE signal outputs in all states.



Bit 5: On - DT/R signal outputs during emulation.



Bit 5: Off - DT/R signal outputs in all states.



Bit 6: On - DT/R signal set to low-level during a break.

Bit 6: Off - DT/R signal set to high-level during a break.

Bit 7: On - User TEST signal connected to ICD TEST signal.

Bit 7: Off - User TEST signal disconnected with ICD TEST signal.

Bit 8: Off/On - Don't care bit (not connected).

✳ Emulation Method Select Switch #2

Description The Emulation Method Select switch #2 is an 8-bit, On/Off type switch that controls signal I/O between the ICD and target system during emulation.

Location The NDP/CPU connector end of the ICD. Switch #2 is located above switch #1. (See "The Controls And Component Functions Of Your ICD," earlier in this section.)

Function See the individual bit settings that follow.

Adjustment To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.

Factory Settings All bits are Off.

Bits 1, 2, 3, 4, 5 On/Off - Don't care bits (not connected).

Bit 6 On - ARDY signal effective during both user and ICD memory access.

Off - ARDY signal effective during user memory access only.

Bit 7 On - SRDY and READY signals effective during both user and ICD memory access.

Off - SRDY and READY signals effective during user memory access only.

Bit 8

On - Automatic 2 clock wait state inserted into each machine cycle.

Off - Wait state suppressed.

✳ Emulation Method Select Switch #3

Description

The Emulation Method Select switch #3 is an 8-bit, On/Off type switch that controls signal I/O between the ICD's CPU and NDP processor and the target system during emulation.

Location

In the central section of the CPU Control module.

NOTE: This switch is not externally accessible. The ICD must be fully disassembled and the CPU Control module removed before this switch can be adjusted. To remove the CPU Control module and gain access to this switch, see "How To Disassemble Your ICD," in Section 3.

Function

See the individual bit settings that follow.

Adjustment

To change the settings of this switch, insert a small, pointed tool (a pen tip works well) and adjust the switch bits to the On or Off position.

Factory Settings

Bit 1 = On
Bit 2 = On
Bit 3, 4, 5, 6 = Off
Bit 7 = On
Bit 8 = Off

Bit 1

On - NDP's BUSY line is tied to the CPU's $\overline{\text{TEST}}$ lin.
Off - NDP's BUSY line and CPU's $\overline{\text{TEST}}$ line function normally.

Bit 2

On - NDP's $\overline{\text{RQ}}$/$\overline{\text{GT0}}$ line is tied to the CPU's HOLD:HLDA line.

Off - NDP's $\overline{\text{RQ}}$/$\overline{\text{GT0}}$ line and CPU's HOLD:HLDA line function normally.

Bit 3, 4, 5, 6 On/Off - Don't care (not connected)

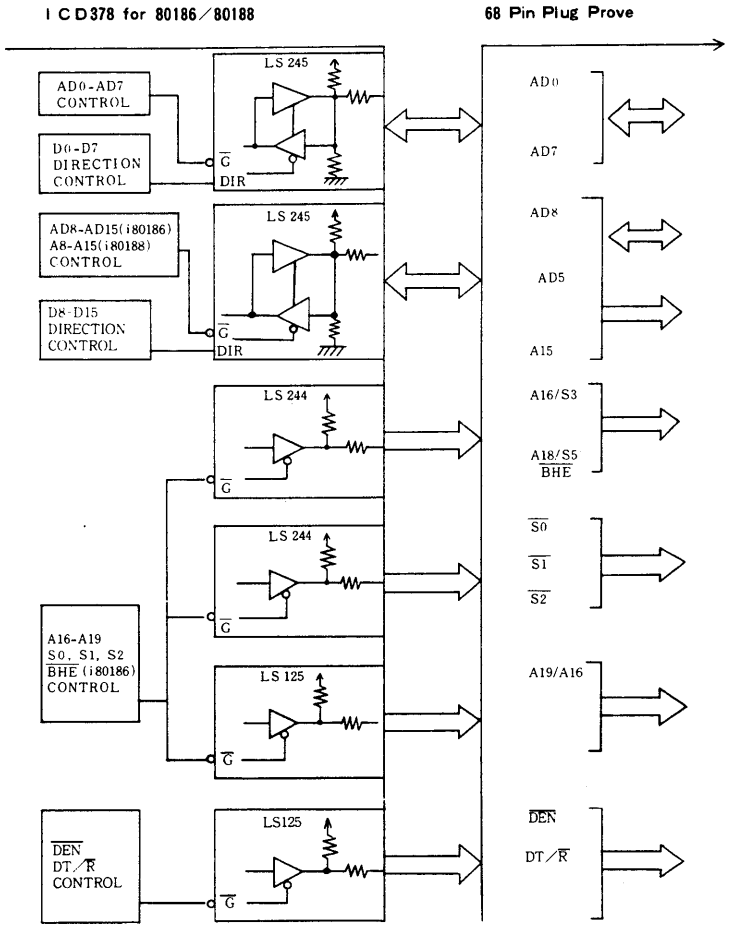
Bit 7 On - NDP's READY line is tied to the CPU's SRDY line.

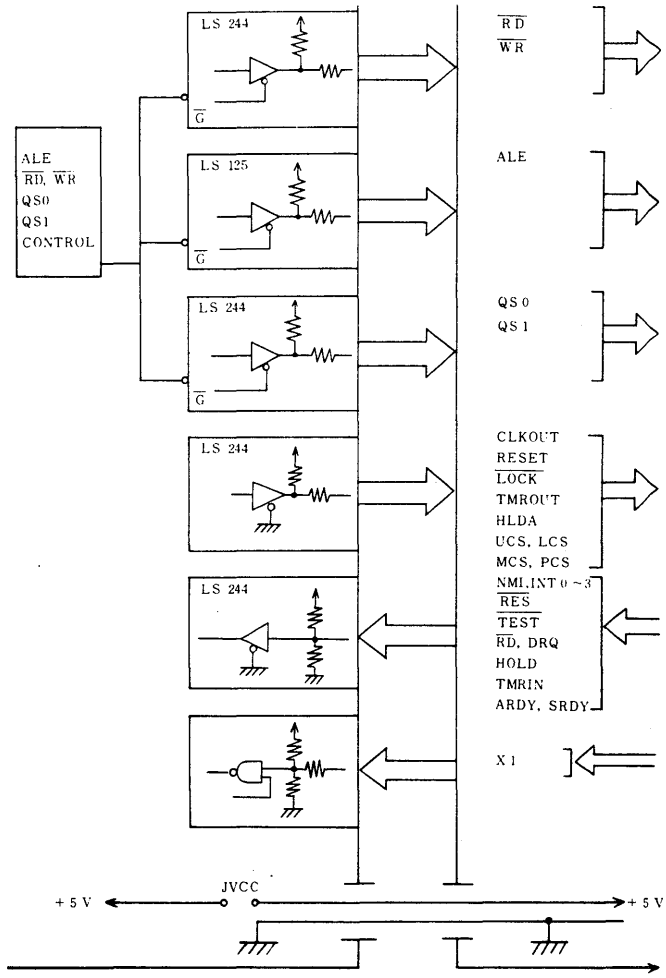
Off - NDP's READY line and CPU's SRDY line function normally.

Bit 8 On - NDP's READY line is tied to the CPU's ARDY line.

Off - NDP's READY line and CPU's ARDY line function normally.

ICD/Target System Interface




LS 244
LS 125
LS 244
LS 244
LS 244
ALE
RD, WR
QS0
QS1
CONTROL
RD
WR
ALE
QS 0
QS 1
CLKOUT
RESET
LOCK
TMROUT
HLDA
UCS, LCS
MCS, PCS
NMI,INT 0 ~3
RES
TEST
RD, DRQ
HOLD
TMRIN
ARDY, SRDY
X 1
JVCC
+ 5 V
+ 5 V

In-circuit Mode/CPU Emulation

In-cir cuit mode	I 0		I 1		I 2	
ICD State	Break	Emulation	Break	Emulation	Break	Emulation
X 1 、X 2	EXT or INT	EXT or INT	EXT or INT	EXT or INT	EXT or INT	EXT or INT
CLKOUT	AC	AC	AC	AC	AC	AC
ICD RESET SW	AC	N	AC	N	AC	
ICD MONITOR SW	N	AC	N	AC	N	AC
RESET		N	N *2	AC	N *2	AC
ALE	N	AC	AC*5	AC	AC*5	AC
NMI, INT 0 ~ 3	N	N	N *1.3	AC	N *3	AC
HOLD, HOLDA	N	N	AC*1	AC*1	AC	AC
ARDY, SRDY	N	N	N *4	AC	N *4	AC
TEST	N	AC	N	AC	N	AC
LOCK	N	AC	N	AC	N	AC
QS 0、1	AC	AC	AC	AC	AC	AC
DRQ 0、1	N	N	N	AC*1	N	AC
TMR IN 0 、1 TMR OUT 0 、1	AC*6	AC	AC*6	AC	AC*6	AC
PCS 0 ~ 7 MCS 0 ~ 2 UCS, CCS	N	N	N *4	AC	N *4	AC

In-circut mode		I 0	I 1	I 2
Memory by Mapping	RO	RO	RO	I C D access as U S
	RW	RW	RW	I C D access as U S
	U S	I C D access as U S	U S	U S
	NO	NO	NO	NO

Status:

AC = accept or active
N = not accepted or not active
RO = ICD emulation memory; read only
RW = ICD emulation memory; read/write
NO = ICD emulation memory; non memory
US = user memory (target system)

Notes:

*1 May be enabled/disabled with PIN command.

*2 Does not reset target CPU.

*3 NMI is edge-sensed. If NMI occurs during ICD emulation break, the NMI process sequence is emulated when the ICD resumes emulation.

*4 Accepted when the ICD monitor is in a user memory access cycle (e.g., EXAMINE, DUMP, etc.).

*5 Output may be controlled by Emulation Method Select switch #1 (bit 4).

*6 Timer contents not guaranteed during ICD monitor state.

Machine Cycle Operation

MACHINE CYCLE	STATUS CONTROL							A16 – A19	AD 0 - AD15 –(i80186)		READY
	S 2	S 1	S 0	DT / R	DEN	WR	RD	BHE (i80186) A 8 – A19 (i80188)	AD 0 - AD 7 (i80188) Address	Date	
Interrupt Acknowldge	L	L	L	L	L	H	H	OUT	I N	I N	AC
Read I / 0	L	L	H	L	L	H	L	OUT	OUT	I N	AC
Write I / 0	L	H	L	H	L	L	H	OUT	OUT	OUT	AC
Halt	L	H	H	H	L	H	H	OUT	OUT	OUT	AC
Instruction Fetch	H	L	L	L	L	H	L	OUT	OUT	I N	AC
Read Data from Memory	H	L	H	L	L	H	L	OUT	OUT	I N	AC
Write Data from Memory	H	H	L	H	L	L	H	OUT	OUT	OUT	AC
Passive	H	H	H	H	L	H	H	TS	TS	TS	AC
HOLD state	TS	TS	TS	TS	TS	TS	TS	TS	TS	TS	NA
RESET State	H	H	H	TS	TS	TS	TS	TS	TS	TS	NA
ICD Emulation	H	L	L	L	L	H	L	OUT	OUT	TS	NA
Memory Fetch *1					*4		*2	O			
ICD Emulation	H	L	H	L	L	H	L	OUT	OUT	TS	NA
Memory Read *1					*4		*2				
ICD Emulation	H	H	L	H	L	H	H	OUT	OUT	TS	NA
Memory Write *1					*4	*3					
Emulation Break	H	H	H	X	X	H	X	X	X	X	NA
I 0	TS	TS	TS	X	L	H	X	TS	TS	TS	NA

Status:

H = high level
L = low level
X = high or low level
TS = high impedance
AC = accepted
NA = not accepted
OUT = output
IN = input

Notes:

*1 In this cycle, the target system is not accessed although the ICD emulation memory is mapped.

*2 $\overline{RD}$ signal suppressed by Emulation Method Select switch #1 (signal status becomes high).

*3 $\overline{WR}$ signal suppressed by Emulation Method Select switch #1 (signal status becomes low).

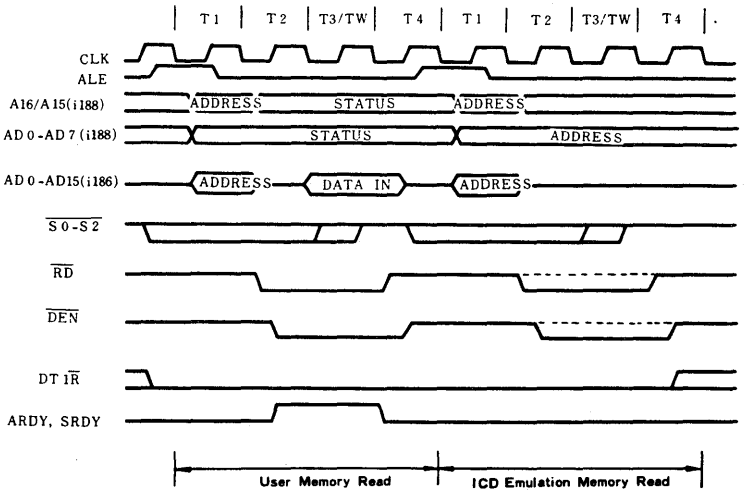
*4 If an NDP is installed, the status of the bus ground becomes read data or write data as viewed from the target system.

ICD Program Memory Cycles

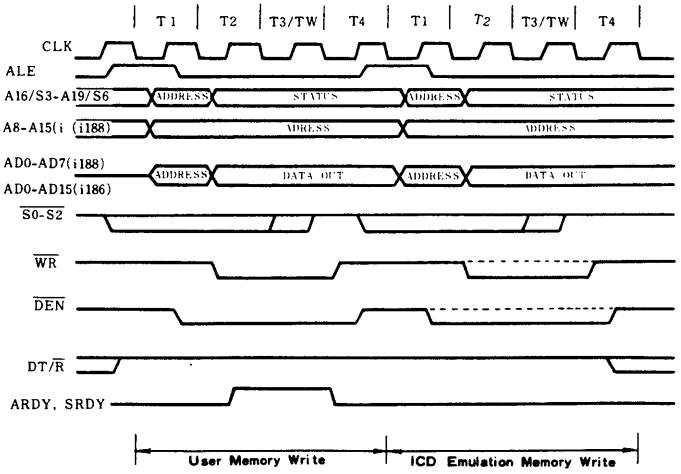
The ICD generates a cycle which does not access memory in the target system when the ICD program memory is selected (by memory mapping), or emulation is interrupted by a break state. This is the machine cycle of the ICD program memory fetch, read and write signals.

During the ICD program memory fetch and read cycles, each of the S0, S1 and S2 signals indicate the memory fetch/read cycle, but the ICD does not receive any data by setting the data bus to a 3-state. During the write cycle, the S0, S1 and S2 signals directly indicate the memory write cycle, and the write data outputs on the data bus. This allows the ICD's program memory write cycle to perform the same operations as the target memory write cycle.

ICD Program Memory Read/Fetch Timing Diagram

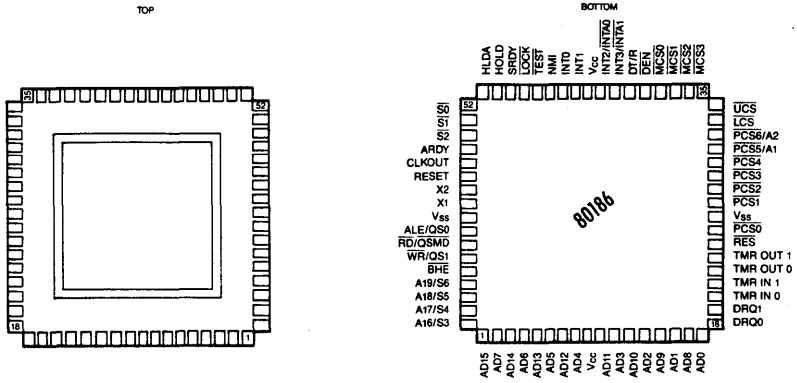


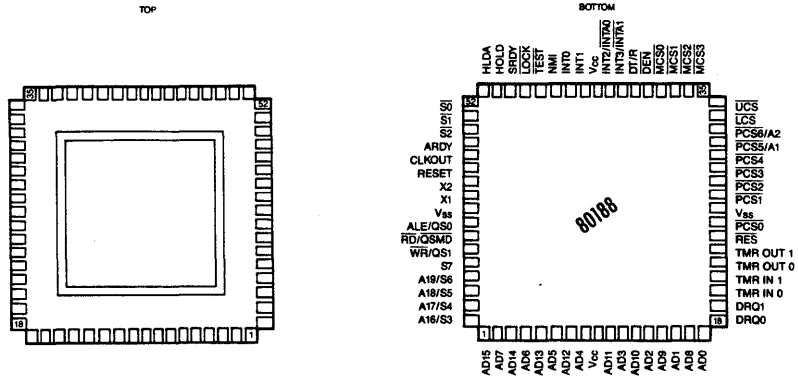
ICD Program Memory Write Timing Diagram



ICD/CPU Configuration

80186/188 Pin Configuration





ICD/CPU Probe Configuration

CPU Probe (1 / 3)

Signal Name	Buffer IC	Direction	State	Register Termination		
				Pull up	Pull down	Series
AD0 - AD3	4A (LS245)	IN/OUT	TS	BR13 (open)	BR22 (open)	BR2 (open)
AD4 - AD7				BR12 (open)	BR21 (open)	BR1 (open)
AD8 - AD11 (A8 - A11)	4B (LS245)	IN/OUT	TS	BR13 (open)	BR22 (open)	BR2 (open)
AD12 - AD15 (A12 - A15)				BR12 (open)	BR21 (open)	BR1 (open)
A16 - A18	4D (LS244)	OUT	TS	BR15 (open)	—	BR4 (open)
A19	5F - 8, 6 (LS125)	OUT	TS	BR15-4 (open)	—	BR4-3 (open)
$\overline{BHE}$ S7	4D - 3 (LS244)	OUT	TS	BR15-3 (open)	—	BR4-2 (open)
$\overline{S0} - \overline{S2}$	4C (LS244)	OUT	TS	BR14 (open)	—	BR3 (open)
$\overline{DEN}$	5F - 3 (LS125)	OUT	TS	BR16-8 (open)	—	BR5-2 (open)
DT / $\overline{R}$	7F - 3, 6 (LS125)	OUT	TS	BR16-7 (open)	—	BR5-6 (open)
ALE	7F - 11 (LS125)	OUT	—	BR14-9 (open)	—	BR3-8 (open)
QS0	5E - 3 (LS244)	OUT	—			
$\overline{WR}$	5E - 14 (LS244)	OUT	TS	BR15-2 (open)	—	BR4-1 (open)
QS1	5E - 7 (LS244)	OUT	—			
$\overline{RD}$	5E - 18 (LS244)	OUT	TS	R4 (10K)	—	BR3-7 (open)
$\overline{QSMD}$	4D - 8 (LS244)	IN	—			

CPU Probe (2 / 3)

Signal Name	Buffer IC	Direction	State	Register Termination		
				Pull up	Pull down	Series
X1	13A-10 (F00)	IN	—	R11 (10K)	R21 (39K)	R1 (100)
X2	—	No Connection		—	—	—
CLKOUT	7C-12 (LS244)	OUT	—	BR14-6 (open)	—	BR3-5 (open)
RESET	7C-9 (LS244)	OUT	—	BR14-7 (open)	—	BR3-6 (open)
$\overline{RES}$	4E-8 (LS244)	IN	—	BR17-5 (open)	BR23-5 (open)	—
NMI	4E-6 (LS244)	IN	—	BR18-7 (open)	BR24-7 (open)	—
$\overline{TEST}$	4E-17 (LS244)	IN	—	BR18-6 (open)	BR24-6 (open)	—
$\overline{LOCK}$	7C-18 (LS244)	OUT	—	BR15-7 (open)	—	BR4-6 (open)
SRDY	4E-15 (LS244)	IN	—	BR18-8 (open)	BR24-8 (open)	—
ARDY	4E-2 (LS244)	IN	—	BR18-3 (open)	BR24-3 (open)	—
HOLD	4E-4 (LS244)	IN	—	BR18-9 (open)	BR24-9 (open)	—
HLDA	7C-3 (LS244)	OUT	—	BR15-6 (open)	—	BR4-5 (open)
Vcc	JVCC No Jumper	No Connection		—	—	—
GND	Connection	Connect Common		—	—	—

CPU Probe (3 / 3)

Signal Name	Buffer IC	Direction	State	Register Termination		
				Pull up	Pull down	Series
TMRIN0, 1	4E-13, 11 (LS244)	IN	—	BR17-6, 7 (open)	BR23-6, 7 (open)	—
TMROUT0	7C-14 (LS244)	OUT	—	BR16-6 (open)	—	BR5-5 (open)
TMROUT1	7C-16 (LS244)	OUT	—	BR16-5 (open)	—	BR5-4 (open)
DRQ0, 1	3F-5, 6. (85302)	IN	—	BR17-8, 9 (open)	BR23-8, 9 (open)	—
INT0, 1	3F-7, 8 (85302)	IN	—	BR18-4, 5 (open)	BR24-4, 5 (open)	—
INT2/INTA0	4F-11 (85301)	IN/OUT	—	BR17-3 (open)	BR23-3 (open)	BR5-2 (jumper)
INT3/INTA1	4F-9 (85301)	IN/OUT	—	BR17-4 (open)	BR23-4 (open)	BR5-3 (jumper)
$\overline{UCS}$	6C-12 (LS244)	OUT	—	BR19-2 (open)	—	—
$\overline{LCS}$	6C-14 (LS244)	OUT	—	BR19-3 (open)	—	—
$\overline{MCS0}$-3	6C-3, 16 7, 5 (LS244)	OUT	—	BR11-5, 4 3, 2 (open)	—	—
$\overline{PCS0}$-1 $\overline{PCS3}$-6	5C-5, 9 7, 14 3, 16 (LS244)	OUT	—	BR19-9, 8 7, 6 5, 4 (open)	—	—
$\overline{PCS2}$	5C-12 (LS244)	OUT	—	BR16-4 (open)	—	—

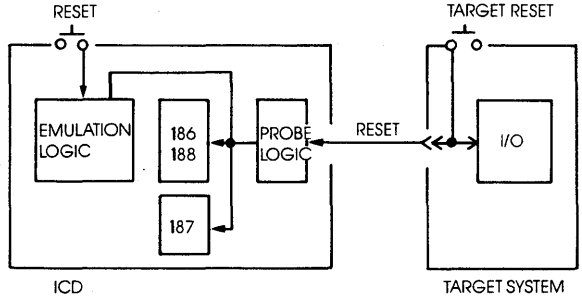
ICD/CPU Signals Examined

RESET Signal

The RESET signal is used to reset the ICD monitor. The signal is sent by pushing the Reset switch on the Indicator/Control panel. This action resets the ICD monitor, but does not reset the target system. Typically, the target system will have a manual reset switch that resets the entire system.

Resetting the target system also causes a hardware reset of the ICD's CPU registers. However, if an emulation break is in progress, resetting the target system will not have any affect on the ICD's CPU registers. The CPU registers must be reset by using the REGISTER RESET command.

NOTE: Resetting the ICD's CPU resets the NDP as well.



	I0		I1		I2	
	MONITOR	EMULATION	MONITOR	EMULATION	MONITOR	EMULATION
ICD MONITOR SWITCH	○	×	○	×	○	×
RESET	×	×	△ *1	○	△ *1	○

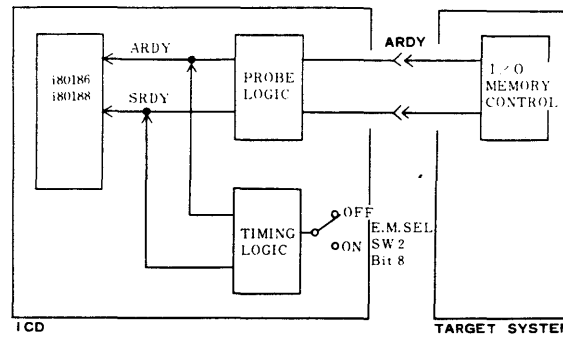
○: Effective △: Conditionally effective ×: Not effective
*1: Does not work as the hardware reset of the emulated CPU.

ARDY/SRDY Signals

The READY (ARDY and SRDY) are active anytime the target system or I/O is being accessed. This signal is useful when emulating target systems that operate a high clock speeds; when the target memory or I/O access time is short; or when a small margin of time is left in the READY set-up time of the target system.

If the ICD is unable to access the target system's memory contents within 128 clock cycles, you can make the ICD cause a break in program execution by using the "BREAK: Timeout Breakpoint" command. (See the BREAK command in Section 2.)

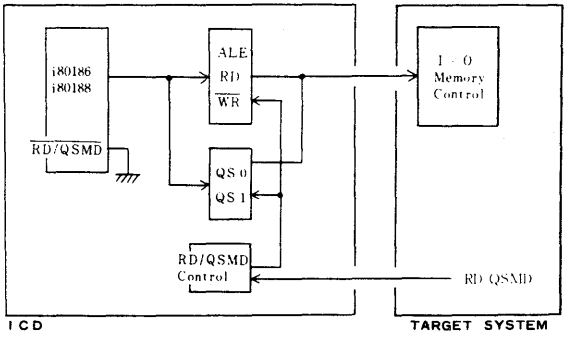
NOTE: Bit 8 of Emulation Method Select switch #2 has no affect if the external READY is prohibited by the READY generation logic of the emulated CPU.



ALE/QS0,WR/QSI, RD/QSMD Signals

If the $\overline{RD}/\overline{QSMOD}$ signal is on GND level during a reset (ICD or target system) and active during emulation mode, the ICD enters the "queue status mode." The probe logic of the ICD has two modes: queue status mode and ALE mode. The target system's RD/QSMD signal determines which type of probe logic is interfaced between the ICD and target system.

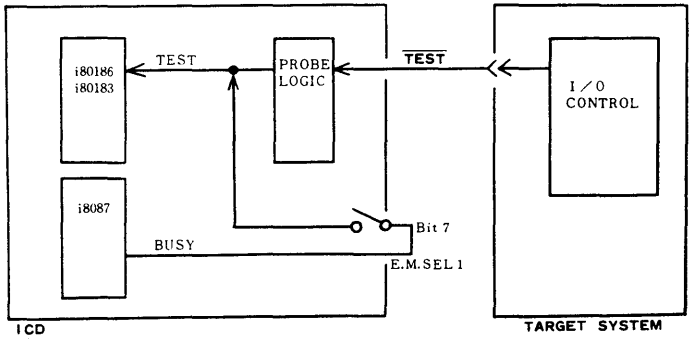
The $\overline{RD}/\overline{QSMOD}$ signal is pulled up within the ICD to prevent the operator from inadvertently establishing the queue status mode.



TEST Signal

The ICD can accept the TEST signal at any time if the in-circuit mode is I1 or I2. If, during emulation, the TEST signal is inactive when the WAIT instruction is executed, the processor does not proceed to the next instruction.

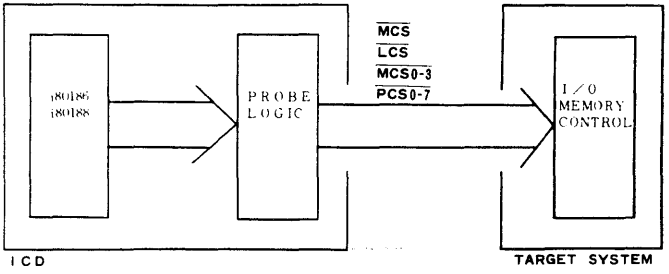
The NDP's BUSY signal can be interfaced with the CPU's TEST signal by using Emulation Method Select switch #1. If this link is chosen, the TEST input of the target system is "ORed" with the BUSY signal. (See "More About Your ICD," in Section 1.)



MCS LCS MCS0-3 PCS0-7 Signals

The chip select signals are effective while an access is being made to the target memory or I/O. The values of the memory select signals at the reset of the ICD's CPU are different from those when the CPU is reset. The peripheral chip select signals, however, have the same initial values between the ICD's CPU and CPU resets.

The table at the bottom shows the initial values for the ICD.



NDP Emulation

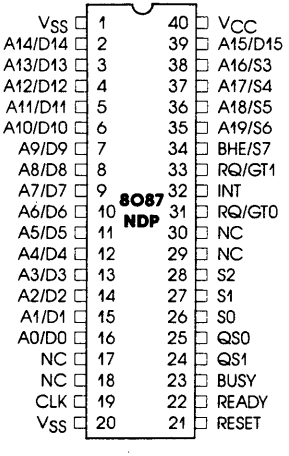
The 8087 Numeric Data Processor (NDP) performs arithmetic and comparative operations on a variety of numeric data types, as well as execution of numerous built-in transcendental functions (e.g., tangent and log functions).

The NDP does not operate as an independent device, but rather as an extension processor to the existing CPU (80186/188). By operating as a co-processor, the NDP effectively extends the register and instruction sets of the host CPU, and adds several new data types as well.

NDP Configuration

8087 Pin Configuration

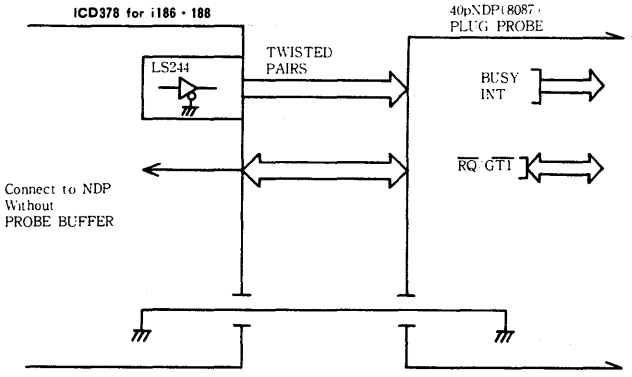
8087 Numeric Data Processor Pin Configuration



NC = NO CONNECT

NDP/CPU Interface

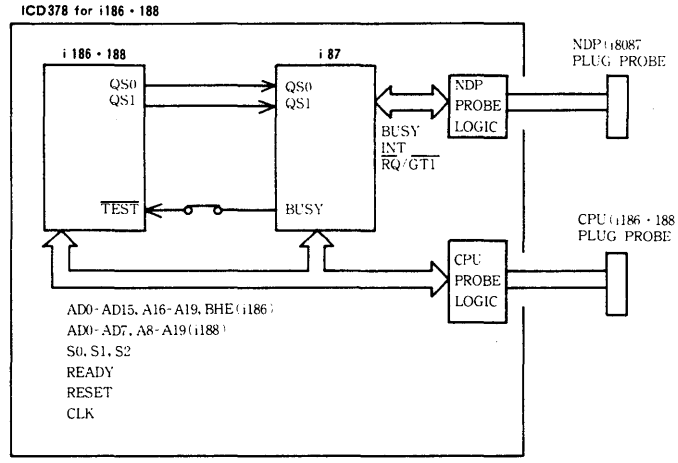
Because the NDP is wired directly to the host CPU, the two processors can be thought of as a single processor. The CPU's queue status lines (QS0 and QS1) enable the NDP to obtain and decode instructions in synchronization with the CPU. The NDP's BUSY signal informs the CPU that the NDP is executing. The WAIT instruction tests this signal to ensure that the NDP is ready to execute instructions. The NDP can interrupt the CPU when it detects an exception. The NDP also uses one of the host CPU's request/grant lines to obtain control of the local bus for data transfers. The CPU and NDP processors all utilize the same clock generator and system bus interface (bus controller, latches, transceivers, bus arbiter); no additional hardware is needed to interface the two processors.



Emulating The NDP

In most cases, a target system which incorporates an NDP in its design, can be emulated using only the ICD's CPU in-circuit probe. This is because the NDP and CPU interface signal lines are internally connected within the ICD. However, the RQ/GT0 and BUSY signals of the NDP are connected to the CPU via the Emulation Method Select switch #3, and this switch must be set correctly in order to emulate the target system's NDP without using the NDP in-circuit probe.

For a complete description of how the Emulation Method Select switch #3 affects the signal interface, refer to the "Emulation Method Select Switch #3" chapter a few pages back.



When To Use The NDP In-circuit Probe

There are certain conditions under which the NDP in-circuit probe must be used to emulate an NDP-equipped design. You should use the NDP in-circuit probe when any of the following conditions exist:

1. The NDP's INT signal is used for an interrupt to the CPU.

2. The NDP's BUSY signal is not used to monitor the TEST terminal of the CPU.

3. The NDP's RQ/GT1 signal is used to connect a bus request from another IOP (Input/Output Processor-8089) to the target IOP.

4. The NDP's RQ/GT signal is not used for a bus request to RQ/GT0 or RQ/GT1.

If the NDP in-circuit probe is required for your particular application, see "System Preparation," in Section 1.

NDP Machine Cycles

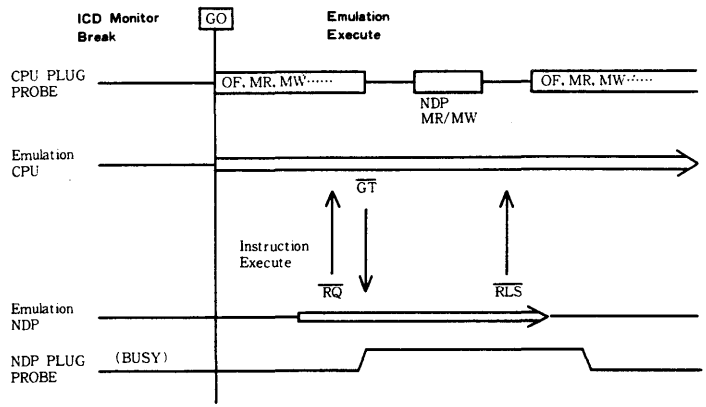
The QS1, QS2, S0, S1, S2 and CLK signals of the CPU and NDP processors are directly connected within the ICD. This allows the NDP to be synchronized with the CPU, regardless of the in-circuit mode or whether the ICD's NDP is connected to the target system. The NDP memory accesses sequence is executed in the following manner:

1. The RQ signal is generated from the NDP's RQ/GT0 line that is connected to the CPU via the Emulation Method Select switch #3.

2. After a GT response from the CPU, the CPU probe interface sets the bus signal to 3-state.

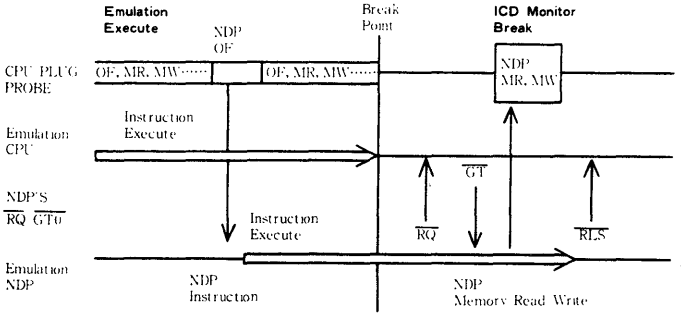
3. When the NDP starts memory access, the bus signal and S0/S1/S2 signals become active and emulate the NDP memory read/write operation.

When the RLS signal of the NDP is generated, the CPU, via the probe, resumes CPU memory access.



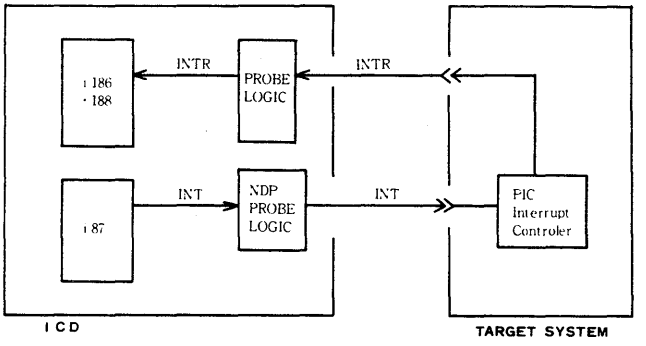
The NDP memory may be accessed after the ICD breaks emulation. The ICD breaks CPU emulation if 500 clock cycles pass, from the time of the NDP instruction fetch (eg., FST, FIST, EBSTP) to the time of memory data writing. In this case, the ICD can complete a bus request and memory access of the NDP.

The RQ/GT0 signal, connected from the NDP to the CPU via Emulation Method Select switch #3, is received by the CPU even during an emulation break; it cannot be inhibited by the PIN command.



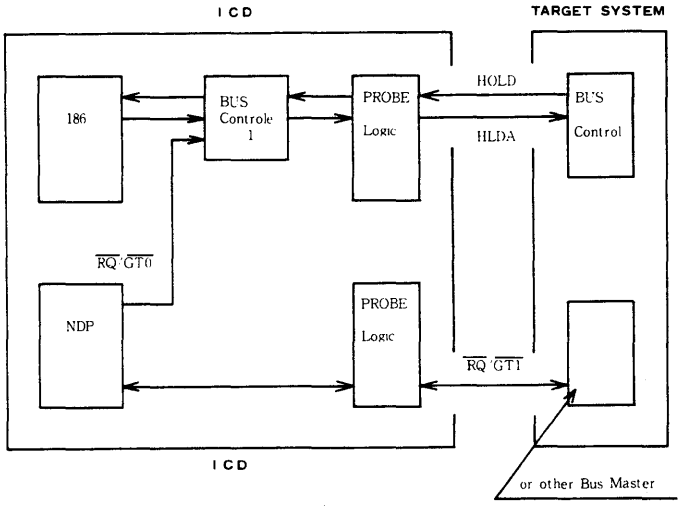
NDP Interrupt Signal

The ICD can output the INT signal of the NDP in all in-circuit modes (I1/I2/I3). The INT signal is used to generate an interrupt sequence to the CPU by the Programming Interrupt Controller (PIC). The CPU then transmits the interrupt signal to the target system in the I1 and I2 modes only.



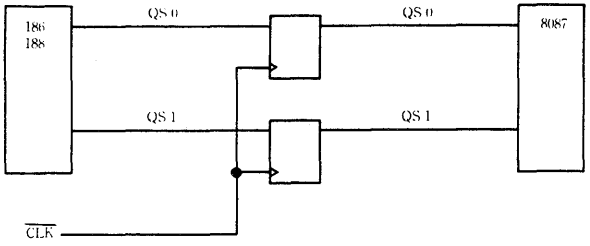
NDP Bus Control

Because the NDP and CPU are connected internally, the ICD can emulate the bus arbitration function by the NDP's RQ/GT0 signal in any of the I0, I1 or I2 modes. The RQ/GT0 signal of the NDP may be accepted during an emulation break state but not inhibited by the PIN command.



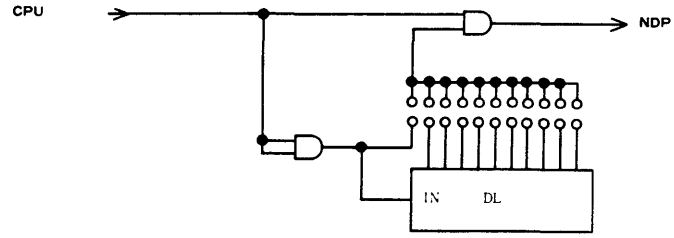
NDP QSO/QS1 Signal

Because the queue status signal of the NDP is issued a half clock cycle earlier than the CPU, the CPU's queue status cannot be directly connected to the NDP.



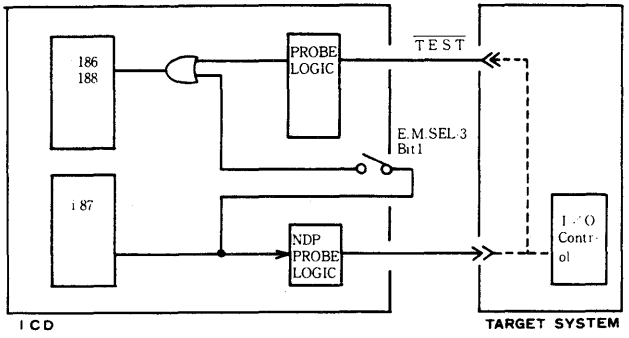
CPU/NDP Clock Delay

The NCK jumpers on the CPU Control module permit synchronization of the clock signal between the ICD's CPU and NDP by adjusting the signal delay to the NDP from 5ns to 50ns. Refer to the "NCK Jumpers" chapter for more information.



NDP BUSY Signal

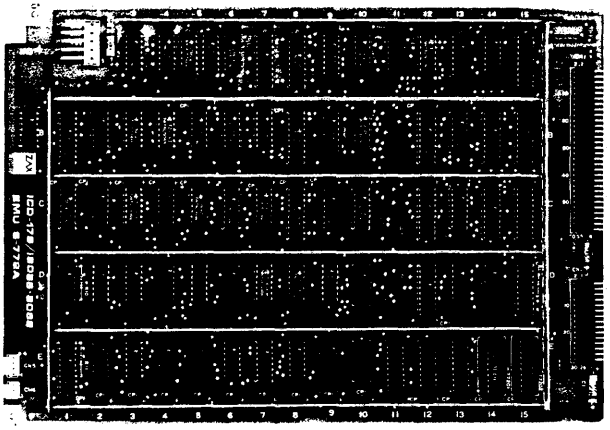
The NDP's BUSY signal can be accessed by the target system in all in-circuit modes (I0/I1/I2). Generally, the BUSY signal connects to the TEST terminal of the CPU, and the BUSY signal between the NDP and CPU is joined via Emulation Method Select switch #3.



Emulator Control Module

Description

The Emulator Control module controls the emulation and monitor modes of operation. The module also houses the Event Trigger connector and the External Break connector; both are externally accessible. There are no internal user-serviceable controls or components on this module.

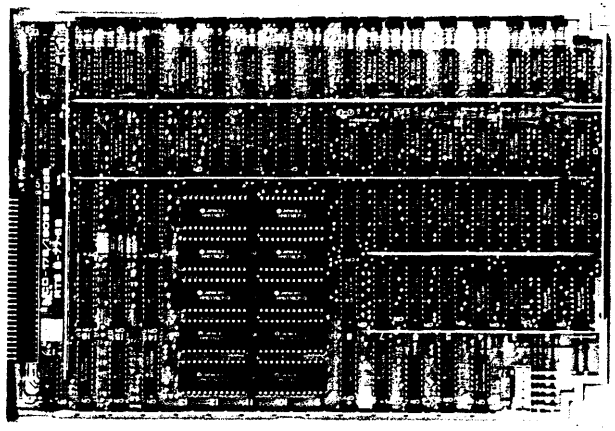


Real-time Storage Module

Description

The Real-time Storage (RTS) module includes the controller, memory, and real-time counter for managing program tracing. By using the HISTORY command, different sections in your program can be traced, stored, recalled and displayed.

The HISTORY command is used to control the functions of the real-time trace module; there are no user-serviceable controls or components on this module. (For a complete description of how the real-time trace feature works, see the HISTORY command in Section 2.)

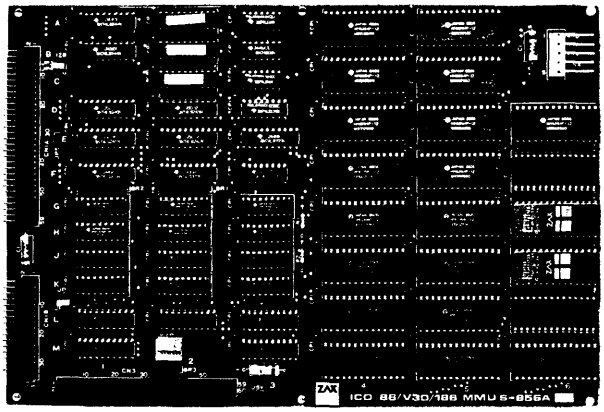


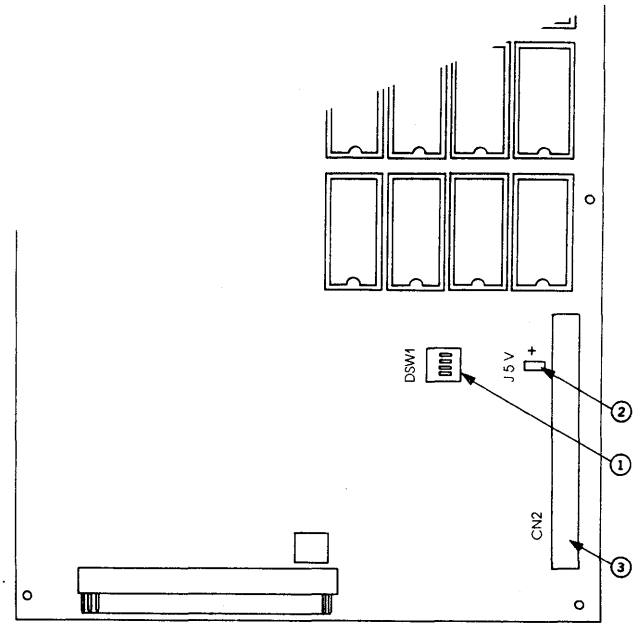
Memory Mapping Unit Module

Description

The Memory Mapping Unit module contains 128K bytes of high-speed static RAM (known as "emulation memory"), which can be used for downloading files, altering the memory contents, and loading future memory into the target system.

There are a few user-serviceable components on this module. See "How To Disassemble Your ICD," at the end of this section, after reading about the MMU's components on the following pages.





1) ICD Program Memory Block Number switch
2) +5V Jumper
3) CN2 Connector

✳ **MMU Components**

1) ICD Program Memory Block Number switch. This 4-bit switch sets the allocation block number of the 128K-byte memory mapping unit in 1K-byte blocks. (For more information on memory allocation, see the ALLOCATION command in Section 2.)

Bit	Operation	Enabled	Disabled
1	Block #7 (A17M)	On	Off
2	Block #8 (A18M)	On	Off
3	Block #9 (A19M)	On	Off
4	Selects the 128K-byte program memory for ICD	On	Off

Factory settings = All bits set to On.

2) +5V Jumper. Internally connects Vcc of the ICD to the Expansion Memory module. Under normal operating conditions, the +5V jumper is disconnected.

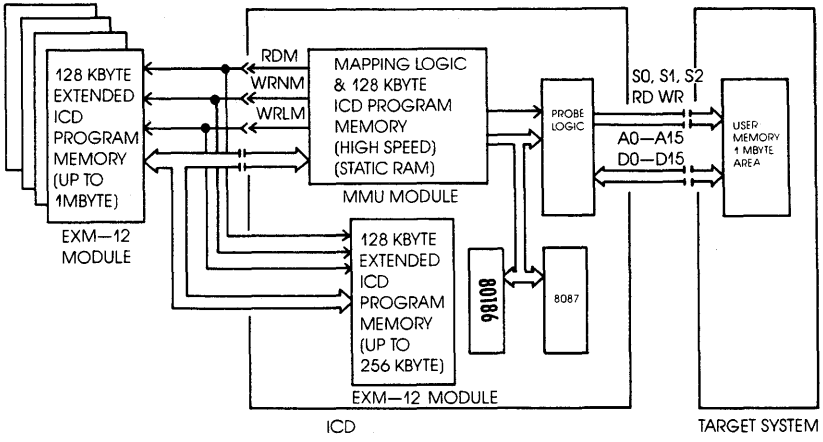
CAUTION: DO NOT USE THE JUMPER FOR A +5V POWER SUPPLY TO CN2. DOING SO WILL CAUSE DAMAGE TO THE ICD.

3) CN2 Connector. Used to connect the optional Expansion Memory module to the ICD.

ICD Emulation Memory

The ICD for 80186/80188 features 128K bytes of RAM (emulation memory) which can be used for downloading object files, as well as altering or manipulating the target system's memory. (Emulation memory contrasts to user memory in that user memory is typically contained within the prototype itself.) ICD emulation memory can be expanded internally to 256K bytes by adding the EXM-12 Expansion Memory module.

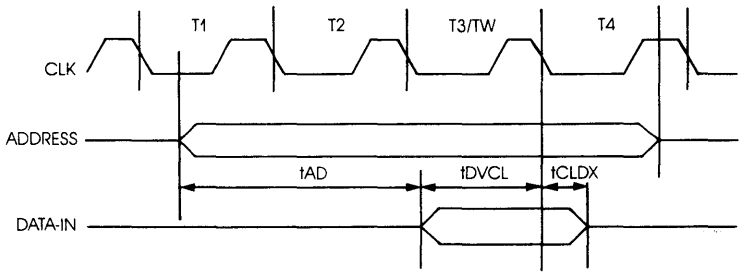
ICD emulation memory is composed of high-speed static RAM, which allows the support of multi-speed target systems. When viewed from the target system, emulation memory is different from a normal memory area in that it is contained within the 80186/80188 processor. Because of the special characteristics of emulation memory, DMA transfer between the target system and the ICD emulation memory is not possible; however, DMA transfer between the address spaces within the target system is permitted.



Target System (User) Memory

The memory contained in the prototype is called target system memory or user memory. The ICD can address any area of the 1M-byte target system memory space.

The access time required to write to the target system memory from the ICD is identical to that of the processor; however, the access time needed to read from the target system memory is slightly shorter than with the processor. Therefore, certain access conditions must be satisfied for accurate reading. These conditions are shown below:



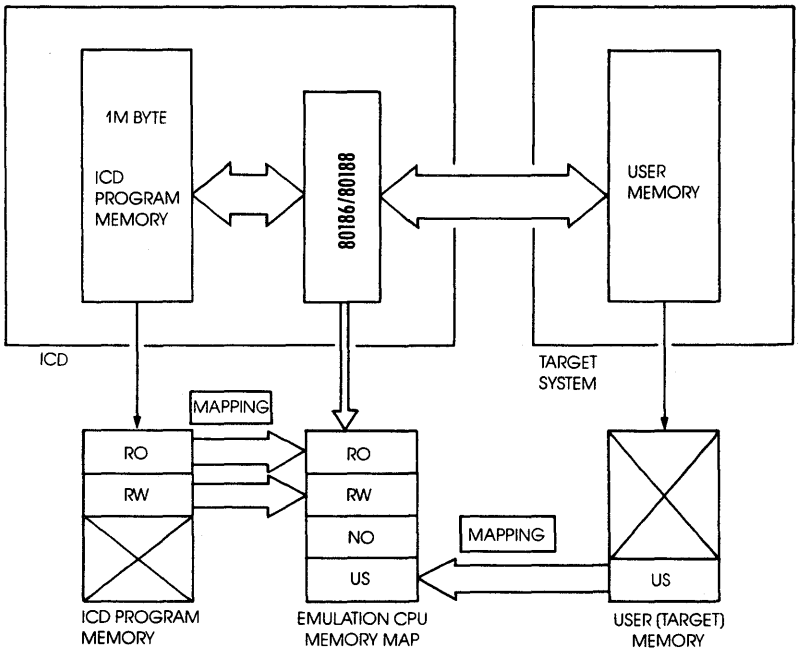
tAD	A0-A19 Valid to Valid Data In	max (3 + N) T—95 ns	
tDVCL	Data Setup Time	min	35 ns
tCLDX	Data Hold Time	min	10 ns

N = Total WAIT state T = CLK Cycle Period

Target Memory Timing Diagram

Mapping

You can use all or part of the ICD's RAM in place of target system memory by creating a memory map. The ICD's emulation memory can be mapped in increments of 1K bytes by using the MAP command. (For an explanation and example of how this works, see the MAP command in Section 2.)



Power Supply Specifications

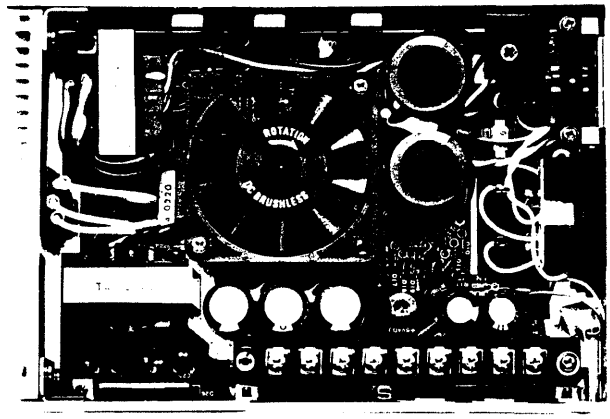
Line Voltage: 100 to 120 volts AC
200 to 240 volts AC

Frequency: 50 or 60 Hz

Power: 50 watts

Output voltage: +5 volts DC
+12 volts DC
-12 volts DC

The Power Supply provides +5 volts to the control modules and 24 volts (p-p) to the external cooling fan and its own cooling fan. The voltage to the control modules is filtered to reduce noise from the power supply line.



How To Disassemble Your ICD

Introduction

The ICD must be partially or fully disassembled in order to modify the components and controls, or to change certain settings on the control modules. In this chapter, you will find the procedure for disassembling the ICD and removing (and installing) the control modules.

IMPORTANT NOTICE!

Before you begin any disassembly of your ICD, you should be aware of certain guidelines which must be followed in order to preserve the Warranty Policy on this equipment.

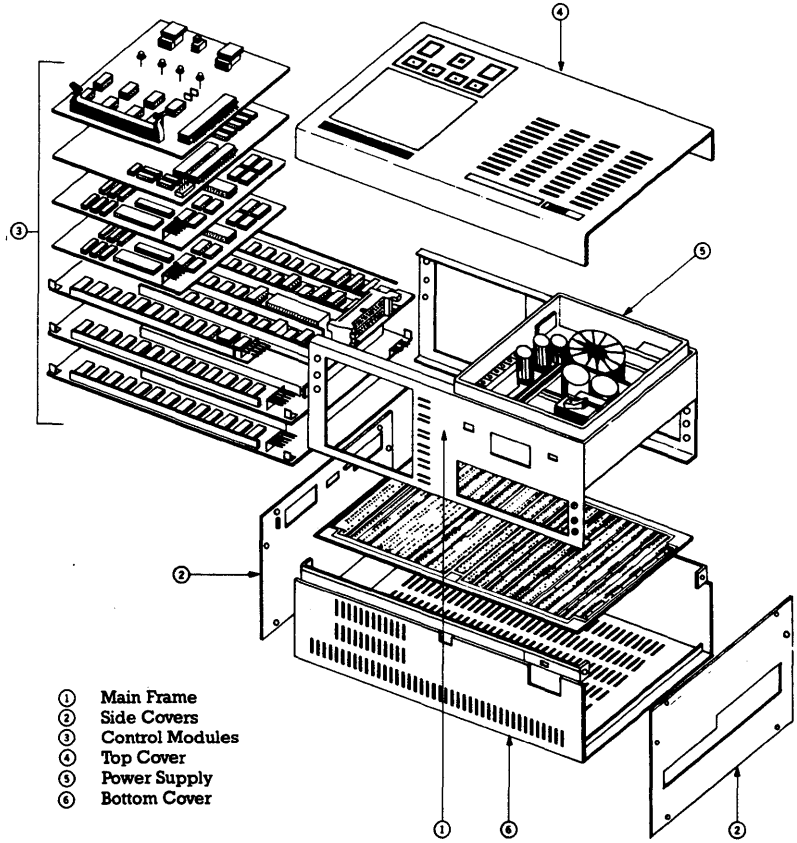
1. All adjustments and modifications to the ICD are limited to the Serial Interface Output (SIO) module, CPU control module, Memory Mapping Unit (MMU) module, and Expansion Memory (EXM-12) module. The adjustments and modifications which are authorized by ZAX are clearly identified in each of these chapters. Any other alterations or adjustments on the other control modules voids the Warranty Policy.

2. Do not adjust, modify, and/or in any way alter the controls or components on any of the three remaining modules (Indicator/ Control, Real-time Trace, or Emulation Memory Unit), or the power supply.

3. Follow the disassembly procedure described here. Damage may result if the ICD is disassembled, or the modules removed, in a manner other than that described here.

The Basic Parts Of Your ICD

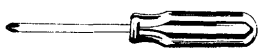
The construction of all ZAX ICD-series emulators is very similar. The basic ICD unit includes the mainframe, the eight control modules, the power supply, the Mother Bus cable, and the outside casing. The mainframe is a metal chassis that houses the control modules and the power supply. The eight control modules are circuit boards which do the actual work of emulating the target system. The power supply provides voltage for the modules. The Mother Bus cable permits the modules to communicate with each other. The ICD case consists of a top cover, bottom cover, and two side covers.



✳ Procedure For Disassembling The ICD

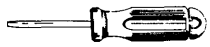
WARNING: HAZARDOUS VOLTAGE IS PRESENT WITHIN THE ICD. DISCONNECT THE AC POWER PLUG BEFORE BEGINNING ANY INTERNAL WORK ON THE ICD.

Disassembling the ICD requires the following tools:

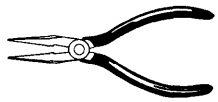


(1) medium Philips-type screwdriver; used for removing the outside case screws.

(1) small Philips-type screwdriver; used for removing the modules from the internal mainframe.



(1) small slot-type screwdriver; used for prying the bus cable sockets away from the pin connectors.



(1) pair of needle-nosed pliers: used for removing and attaching power-supply connectors from the 5-pin plugs.

1. Remove the two side covers and the top cover.

    a) Remove the four raised screws that connect the side and top covers.

    b) Remove the eight countersunk screws on the side covers.

    c) Detach the side covers and the top cover.

2. Gently turn over the ICD and remove the bottom cover.

NOTE: Place the ICD on a soft foam-type pad to protect the case and components.

    a) Remove the four screws that attach the bottom cover to the mainframe (it is not necessary to remove the two countersunk screws).

    b) Lift the bottom cover away from the ICD.

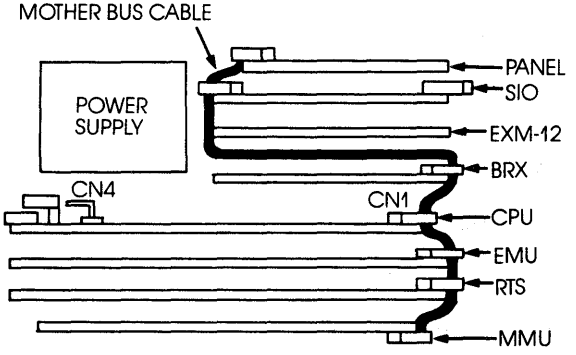
    c) Turn the ICD back over on the foam pad so the control panel and power supply are facing up.

The control modules are now accessible for removal.

How The Modules Are Connected

Each module is linked by the 60-pin Mother Bus cable. Power is supplied to each module by a 5-pin, plug-type power connector cable (except for the Indicator/Control Panel which receives its power from the Mother Bus cable). The power and Mother Bus cables must be detached before removing any of the control modules.

IMPORTANT: Note the position of the power connectors before removing them. Both the socket and plug have a black label on one side which marks the polarity of the connectors.



Removing The Modules

1. Remove the Indicator/Control Panel.

    a) Remove the four screws that secure the metal cover to the panel.

    b) Detach the Mother Bus cable from the Indicator/Control panel (location CN-1).

    c) Remove the four screws that attach the panel to the mainframe.

    d) Remove the Indicator/Control panel from the mainframe.

2. Remove the Serial Interface Output module.

    a) Remove the two small screws that attach the module to the mainframe.

    b) Detach the Mother Bus cable from the module.

    c) Detach the power-supply connector from the module.

3. Remove the optional Expansion Memory module, if supplied.

    a) Detach the auxiliary bus cable from the module.

    b) Detach the power-supply connector from the module.

    c) Remove the two small screws that attach the module to the mainframe.

    d) Remove the Expansion Memory module from the mainframe.

NOTE: When installing this module, carefully fold the bus cable to make a 90-degree turn so that it can attach to the 60-pin bus receptacle on the Memory Mapping Unit module (located on the bottom of the ICD).

4. Remove the Break Matrix module.

   a) Detach the Mother Bus cable from the module. Detach the 30-pin bus cable that connects the BRX module to the EMU module.

   b) Detach the power-supply connector from the module.

   c) Remove the two small screws that attach the module to the mainframe.

   d) Remove the Break Matrix module from the mainframe.

5. Remove the Central Processing Unit module.

   a) Detach the Mother Bus cable from the module by pushing out the retaining clips from the connector housing.

   b) Detach the power-supply connector from the module.

   c) Remove the two small screws that attach the module to the mainframe.

   d) Slide the module away from the mainframe at the power-supply connector-end of the ICD.

6. Remove the Emulator Control module.

   a) Detach the Mother Bus cable from the module.

   b) Detach the power-supply connector from the module.

   c) Remove the two small screws that attach the module to the mainframe.

   d) Slide the module away from the mainframe at the power-supply connector-end of the ICD.

7. Remove the Real-time Trace Storage module.

   a) Detach the Mother Bus cable from the module.

   b) Detach the power-supply connector from the module.

   c) Remove the two small screws that attach the module to the mainframe.

   d) Slide the module away from the mainframe at the power-supply connector-end of the ICD.

8. Remove the Memory Mapping Unit module.

   a) Turn the ICD over so that the MMU module is on top.

   b) Detach the Mother Bus cable from the module.

   c) Detach the power-supply connector from the module.

   d) Remove the four small screws that attach to module to the bottom of the mainframe.

   e) Lift the module away from the mainframe.

Installing The Modules

To install the modules, reverse the "removing the modules" procedure.

CAUTION: DO NOT REVERSE POWER CONNECTOR POSITION DURING INSTALLATION. CONNECTOR MISPLACEMENT WILL CAUSE DAMAGE TO THE ICD-278.

NOTE: When replacing the side panels, loosely position all the screws in place to allow the panels to align properly before tightening the screws.

Section 4: COMMUNICATION PROTOCOL, shows the communication programs for interfacing a computer to an ICD-series emulator NOT supported by ZAX's ZICE communication software. It is an optional section of this User's Manual. If you require this section for your application, call or write the following:

ZAX Corporation
Attn: Technical Publications Department
2572 White Road
Irvine, CA 92714
(714) 474-1170
(800) 421-0982
Telex #183829

APPENDIX A

IN-CIRCUIT EMULATORS SPEARHEAD THE NEW MICROPROCESSOR DEVELOPMENT SYSTEMS

by Mark D. Johnson

Although turnkey microprocessor development systems have been available for about a decade now, the new integrate microprocessor development systems (MDS) used for testing, debugging and integrating both hardware and software designs have only recently become a common sight in the design engineer's lab.

The new MDS, with its unbundled structure, is a multi-component, multi-vendor system pieced together by the user. It is the latest, and least expensive, approach to implementing and automating the design of microprocessor-based systems. These open-architecture development systems - designed to be compatible with existing development equipment - are the newest and fastest growing design aids that are more than adequately competing with the turnkey development systems supplied by the chip manufacturers themselves.

The latest tri-based systems incorporate a host computer, development software and diagnostic tools such as logic state analyzers and emulators. Of these components, it is the emulator that transposes the run-of-mill computer system into a full-fledged microprocessor-based development system for managing a project from the software coding phase right through to hardware/software integration and final testing.

What makes the new MDS so popular? Primarily, its low cost. By utilizing existing resources in their labs, engineers can now integrate the development software and diagnostic tools needed to form a complete MDS, while saving thousands of dollars over the price of dedicated systems.

PC Makes The Ideal Manager

The heart of the MDS is the management unit - the host computer. The host computer provides control for the system, acts as a pipeline to development software and provides mass memory storage. In the past, mainframe micro computers were the accepted standard for project control, but with the new MDS, this expense far outweighs the costs of the remaining components.

The most attractive alternative is the IBM personal computer - indisputably the design engineer's favorite choice in a personal computer. Other PCs can be utilized, but the IBM PC is really the key to the flexible MDS as other market leaders continue to tailor their software products and diagnostic tools to IBM's architecture.

Using the IBM PC, design engineers can now tap an extensive line of development software products for generating source code and compiling complete programs. This development parallels the longstanding ability of programmers to incorporate the use of assemblers, compilers, linkers, loaders and editors as software development tools.

The remaining members in the MDS structure are diagnostic tools; specifically, logic-state analyzers and in-circuit emulators. When the hardware prototype is constructed, it must be tested to determine whether the actual operation in real life is within the hardware specifications. This is where specialty diagnostic tools become a necessity, as they have the capability to monitor signals from the microprocessor and peripheral circuits. Since there are numerous microprocessor signals which require simultaneous monitoring, the use of an oscilloscope (capable of monitoring only a few signal lines) would be impractical for keeping track of all the necessary signals. This is where the logic state analyzer and emulator play such a vital role.

Emulation vs Simulation

A logic state analyzer (LSA) can vary from a simple (where some indication of a single multi-signal state is needed) to a very sophisticated device which can store several multi-signal states and display the results in various forms - even showing the instruction mnemonics. LSAs, however, lack the one feature available on virtually every emulator - the ability to "loan" resources. Thus, the emulator is an advance over the LSA because it permits the use of the more sophisticated facilities of the LSA, combined with the ability to transfer blocks of memory between the prototype and the emulator, and then modify the memory by categorizing it as read-only, read/write, and so on. Another popular configuration ties an LSA directly to an emulator. The combination merges a logic analyzer's sophisticated trigger, qualification, data-acquisition, and measurement capabilities with an emulator's debugging and loan facilities. The emulator/LSA interface permits simultaneous control

and monitoring of the prototype system under development.

For all their testing, debugging and management capabilities, emulators remain the most misunderstood and often intimidating devices associated with the MDS. Part of the mystique arises from their functionality (the ability to monitor as well as execute), part from the lack of application information (often vague and incomplete), and part from their control mechanism (emulators need to be told what to do and how to do it by a set or series of complicated and often confusing instructions.) Couple these problems with the fact that there are several different system configurations, all requiring a particular compatibility formula, and it's no wonder designers turn to the single-vendor development systems. Fortunately, however, the situation is changing for the better, since vendors are beginning to coordinate various manufacturers' equipment and development packages for their users.

Mimicking The Prototype

Emulation is the result of replacing the microprocessor in a prototype design with a piece of test equipment which incorporates an identical microprocessor to that found in the prototype. The strategy behind this exercise is to provide the test equipment with all the functions of a particular microprocessor, along with capabilities which assist in the integration of the prototype's hardware and software components. In a real-world testing environment, the emulator is attached to prototype circuitry by replacing the microprocessor with a multi-pinned header plug with the same pin configuration as the processor's. Thus, in its simplest operating mode, the emulator "tricks" the prototype into thinking nothing has changed with its relationship to the microprocessor. The prototype can then execute its functions while being monitored by the emulator.

The basic components of all emulators are very similar, usually consisting of a mainframe chassis, individual control and memory circuit boards, intermediary circuitry, and an isolated power supply. The controls and components on an emulator's external casing can range from virtually non-existent - where control and indication is regulated to an external terminal - to those possessing complete keyboard facilities, control switches and even an EPROM socket. The physical size of an emulator is usually

dictated by the amount of resident memory it contains, the control and indication mechanisms, its emulation probe arrangement and the sophistication level of its debugging facilities.

The emulator's microprocessor, which permits the actual emulation of the prototype hardware, is located on the emulation board or emulation pod. The board typically contains an emulation probe (or the means of attaching one), which connects the emulator's microprocessor to the prototype. The connection between the prototype and the emulator is necessary for hardware development but is not needed for software debugging and testing.

Although apparently simple in its construction, the design of the probe is critical if transparent emulation is to occur. (Transparency is the ideal emulation condition in which the operation of the prototype is unaffected when the emulator substitutes the microprocessor. An emulator should make no demands on any part of the prototype's resources such as interrupts and memory allocation, and the emulator should resemble as closely as possible the micropro-cessor's characteristics such as timing and clock speed.) Ideally, the emulation cable represents a compromise between user convenience and minimizing the propagation delays and capacitive loading that the length of cable introduces. Some emulator vendors try to eliminate the latter by locating the emulated processor in a pod positioned both physically and electrically as close to the prototype's socket as possible - sometimes a mere inch away. While this arrangement minimizes propagation delays, there are some disadvantages such as a bulkier probe design (especially when interfacing to pre-constructed designs) and the necessity for a larger buffer board within the emulator. If clock speeds are kept under 10 MHz, this type of cable provides virtually transparent emulation. If speeds are faster, the microprocessor pod arrangement is preferable.

Emulator Supplies The Speed

An emulator's resident memory is composed of high-speed static RAM, which is fast enough (i.e., sufficient access time) such that the program can run at the same speed as if the prototype were supplying the memory. High-speed static memory also eliminates the need for the emulator to insert wait states when accessing the memory. Emulators typically contain between 32 and 123 kbytes of internal RAM, and nearly all emulators allow

expansion to larger memory resources (some to 16 Mbytes via off-board memory modules or by accessing a portion of prototype memory.

Emulators can also allocate their memory to the prototype (or any location within the addressable memory space) in specified byte blocks. For an emulator featuring 64 kbytes of memory, the block size (resolution) is typically 1 kbyte. The resolution for a given emulator usually varies with the amount of memory available, with 2- or 4-kbyte sizes popular with emulators that have a memory capacity of 128 kbytes to 1 Mbyte. When resolution falls below the 1 k-byte block size, it often becomes too "fine," which necessitates several allocating operations. The opposite condition, "coarse" resolution, makes it impossible to accurately emulate prototype systems incorporating limited memory chips. Most vendors seem to agree on confining resolution capacities to between 1K and 4K bytes.

Since the allocated memory should have the same design characteristics of memory that will eventually exist in the final design, mapping is used to categorize the type of emulation memory that will be allocated to the prototype. Most emulators allow their resident memory to be specified as read-only and read/write, and some add a "user" memory specification (user being RAM, ROM, I/O, etc. - whatever resides at those memory locations in the prototype). A few emulators provide a "no memory" mapping option, where any memory not mapped as read/write or read-only is assumed to be non-existent prototype memory. This type of mapping turns out to be a very useful provision in software debugging since a common result of programming error is an attempted access to an area where no memory is present. During emulation, the recognition of an attempted access to a protected memory area results in a program halt.

The ability to loan memory to the prototype design is what sets the emulator apart from an ordinary computer system, but emulators can also manage several other memory functions. Most emulators feature an in-line assembler which converts the mnemonics entered from the keyboard to machine language in memory. The emulator's assembler is useful for writing software patches into program code that has either been downloaded from a mainframe or developed in the prototype.

Programmers can also use the assembler for writing their own routines or developing small programs.

Another memory control feature is the ability to display the emulator's memory contents (starting and ending anywhere that the user chooses). The ability to change the memory contents at individual locations and fill entire sections of memory with specified data is also available. Another memory control involves comparing specified blocks of memory with other blocks in the emulator's memory or with blocks residing in the prototype. The comparison can show all the matching or non-matching data and their locations. Instead of a complete block of memory, the emulator may also locate specific data in its memory or the prototype memory.

Breakpoints to Stop, Triggers to Flag

The key to effective program control is installing breakpoints and triggers within the user program so the emulator may show the condition on the processor's address, data, and control lines. Both software and hardware breakpoints are available within a given emulator, and each have their own functioning scheme. Triggers provide the emulator with "sensors" that can be used to initiate program recording (via the trace mechanism), terminate the program recording, or simply detect both operations.

Software breakpoints replace program instructions with monitor calls in order to stop program execution at a pre-determined location. Setting a software breakpoint causes the emulator to automatically replace the op code at the specified address with the processor's software interrupt instruction. When the code is encountered during program execution, a temporary break occurs while the original contents are replaced by the interrupt instruction; then the execution restarts at the same location (this causes the program to only run in real time up to the breakpoint). Software breakpoints are limited to address recognition only - there is no way for a software breakpoint to decipher between memory types.

Hardware breakpoints, which recognize machine cycles but do not disturb normal software execution, can monitor the address and status signals for a specified condition (for example, a memory read operation at address 1000H) and halt program execution when

those conditions are met. Emulators allow several bus transactions to be monitored for program breaks including accessing any memory operation, accessing any I/O operation, memory reads, memory writes, I/O reads, I/O writes, operation instruction fetches and interrupt acknowledgment. When a break condition occurs, most emulators can stop the program being executed and display the current contents of the microprocessor's registers, the instruction just executed, the location of the instruction and the reason the program was halted.

Triggers can be used to initiate actions during emulation or simply observing them and relay the pertinent information to other devices. For example, a trigger which is set in the program (according to address location, data type or bus transaction) and encountered during emulation may be used to halt program execution in a manner similar to a normal hardware breakpoint. But triggers are more flexible than breakpoints in that they may be specified to merely observe the pre-defined condition and act without disrupting the program being executed.

Real-Time Trace Stores Program Execution

Breakpoints and triggers come together under the debugging capability called real-time tracing. The real-time trace feature allows a user to record program execution in real time and then analyze (by replaying) a section of the program in non-real-time. The entire program can be traced (and reviewed), or just a portion - depending on your strategy. By using various combinations of triggers and breakpoints, the desired section where a potential problem exists can be recorded. The program can then be stopped, and the address, data and control lines of the latest series of machine cycles can be displayed or dumped to a printer for examination.

The trace buffer size of an emulator is determined by its width and depth. An emulator's trace buffer should be wide enough to accommodate the processor's address and data lines and possibly a few external lines. Typically, emulators offer between 32 and 64 bits of width. Some universal emulators, designed to work with several different processors, feature extremely wide trace buffers for allocating the various address, data, status and external lines for the entire line of processors supported.

When it comes to trace buffer depth (i.e., how many cycles of machine cycle execution the buffer can hold), larger is not always better. If the size is fixed at a large depth, problems may exist when sifting through all the information accumulated in the buffer. Conversely, if the buffer is too small, the chances of recording the section containing the error is reduced to a hit-and-miss situation. A good working buffer depth falls between 2k and 4k machine cycles. A large buffer with user-variable depth makes an ideal combination.

Emulator Enhancements

Functions in emulators which do not manipulate memory or affect emulation processes are simply debugging enhancements. Since no two vendors offer the same set of debugging enhancements, none have emerged as standards (as with breakpoints, for example). Among the more useful features are built-in test programs, which include memory tests, scope loop tests and signature analysis tests. Memory tests are useful in determining whether the system RAM is functioning properly. Scope loops provide repeated read or write operations to memory or I/O ports, which are useful when debugging these circuits with an oscilloscope, while signal analysis can be a useful test environment strategy.

Other feature include built-in PROM programmers, offset registers and calculation and conversion facilities. PROM programmers are used in microprocessor applications to place programs and data in ROM. Offset registers. Offset registers are helpful when debugging programs consisting of several modules. Each module listing typically shows the first address in the module as zero, with the linker/loader then relocating each module to the appropriate address. To determine the actual address for a given instruction, the load address must be added to the address shown in the listing. By setting the offset register to the load address, this procedure is handled by the emulator.

When debugging software, the use of a decimal and hex calculator/converter can be helpful. Some emulators have the ability to perform subtraction and addition of hex and/or decimal numbers, and perform hex-to-decimal or decimal-to-hex conversions. The results of a particular operation are displayed in both hex and decimal notation.

Command Formats Yet to be Standardized

Before any operation can be executed or any function monitored, the emulator must be told what to do, and told in its own precise language - a language that's different with almost every emulator. While many emulators use their own simple mnemonic command structure, others rely on complicated non-mnemonic logical statements. While one emulator contains a built-in control keyboard, another can only be activated from the remote keyboard of a terminal or computer. The two different command format styles and entry mechanisms influence the way debugging operations are both specified and executed. For example, assume a user wishes to move a block of memory from a location in the prototype memory space to a space in the emulator's memory. One emulator, with its built-in keyboard, would require the user to press a control keyswitch, punch in a start address, press another control keyswitch, punch in the end address, then press another control to initiate the transfer. With a mnemonic command-controlled emulator, the same operation can be executed with a single-line entry that specifies both the Move operation and the beginning and ending addresses that mark the block of memory in the prototype.

The Emulation Session

Assuming the user has an understanding of emulation principles and can interpret the tutorial material in the emulator's operation manual, he is now ready to begin the art of debugging and testing the prototype hardware and software.

Once the hardware prototype is available, the system may be configured for emulation by first removing the prototype's processor and then electronically replacing it with the emulator's processor via the emulation probe. The emulator may now attempt to emulate the prototype. (At this point, the prototype design takes on the role of a "target" for the emulator.)

Initially, the emulator is instructed to analyze certain conditions and act accordingly, for example, it may stop program execution when a read-only memory area is written to. The emulator then begins executing the prototype program either from an initial or reset state, a user-specified starting address or the location where the emulator was last stopped. During this time, the emulator is operating as if the prototype's processor were controlling the system as well as monitoring the address, data and control lines for a breakpoint, trigger or fault condition.

After a period of time, the emulator's circuitry stops the execution of the prototype program. It may have detected a breakpoint, an illegal instruction execution, a memory protection violation, a trigger acting as a breakpoint, or the user may have manually stopped program execution. If a breakpoint stopped program execution, the emulator can display the instruction just executed, the instruction location, and the status of the registers. If the emulator's real-time trace mechanism has been activated, it will have stored the latest series of machine cycles and is now available for viewing. The buffer will show all bus activity from the breakpoint or trigger.

Once the current status of the processor is examined, the user may need to alter the prototype program. This may be as simple as changing the contents of specified registers or more involved by writing a software patch using the emulator's in-line assembler or transferring complete sections of prototype memory to the emulator for testing. After the section is tested, it is moved back into the prototype at the same location. When the user is satisfied with the changes, the emulator starts executing the program again, looping through the above process until the prototype functions correctly.

Economic Considerations

In the MDS engineering environment, emulators play a special role by providing the unique ability to not only monitor the prototype under actual operating conditions, but also to loan facilities to the prototype system itself. During prototype execution, the emulator's testing abilities permit the address, data and control lines of the emulator's microprocessor to appear transparent to the prototype until a problem occurs or a breakpoint or trigger is encountered. The emulator then possesses the facilities to track down and fix the problem, then test and recheck the program.

From a monetary standpoint, emulators are cost-effective diagnostic tools in comparison to single-vendor turnkey development systems. Their flexible design allows them to interface to affordable and increasingly popular personal computers, which, in turn, can be used to develop software code as well as handle the editorial chores associated with a word processing system. They are also completely compatible with existing mini and mainframe computers - making them ideal design aids for both large and small design departments.

ICD Product Demonstration: Features & Functions Of The ICD

Introduction

If this is the first time you are using a ZAX emulator, you've turned to the right place! This appendix contains two exercises which you can use as a product training course. By following the exercises presented here, you'll not only demonstrate to yourself the powerful debugging capabilities of your ICD, but you'll learn more about emulation principles as well. Once you've familiarized yourself with some basic command functions and applications, you can go back to the Master Command Guide (Section 2) and become an emulation expert!

Two Different Exercises!

You have two exercises to choose from in this appendix, and each is designed to teach you something new about your ICD. In Exercise 1, the ICD executes commands entered from a terminal; a target system is not used. Then a computer is used to control the ICD so that save and load commands can be executed.

In Exercise 2, the ICD executes commands entered from a host computer, and a target system is emulated by the ICD. This exercise can also be performed using just a terminal to control the ICD.

System Configurations

In order to perform the exercises, the ICD must be configured for a particular operating environment. The details for connecting the ICD to a terminal, computer, and target system are discussed in "How To Connect Your ICD To Other Devices," in Section 1.

Which Exercise To Use?

If this is the first time you are using a ZAX emulator, you should read through and then complete Exercise 1: Using The ICD Without A Target System. This session reveals many of the ICD's capabilities, including performing actual emulation of a test program. (If you need a refresher course in emulation theory and practices, read through Appendix A before you try the exercises.)

Entering The Commands

You don't need to fully understand all the command rules to use the ICD feature demonstration. Just carry out the instructions listed under ACTION and read the display on your terminal's screen. You must remember to enter the exact items as shown in the exercise, including feature characters (,/=), and provide spaces at the appropriate places as shown in the instructions.

If you make a mistake when entering a command parameter, address, or data value, the ICD will probably respond with an error message. It's usually not a problem - just check to see that the proper characters, numbers, or spaces were used, and then re-enter the complete command statement.

Facts About Exercise 1

Name: ICD Product Demonstration - Exercise 1
Controlling Device(s): Terminal (preferred) or Host Computer
Operation Mode: LOCAL & REMOTE (if using computer)
Target System Used?: No
Printer?: Optional

Before you begin this exercise, make sure that the ICD is configured for terminal control (see "How To Connect Your ICD To Other Devices," in Section 1).

Any time you enter a command, it must be followed by a carriage return.

A scroll control is used with all commands that "roll" data onto the screen. Anytime you wish to stop or start the scrolling on the screen, simply press the space bar. Anytime you wish to exit the display, press the Esc key.

Now begin the demonstration by executing the instructions listed under ACTION; the command you are using is listed under COMMAND DEMONSTRATED; read the information under COMMENT for a description of what's taking place at that point in the demo.

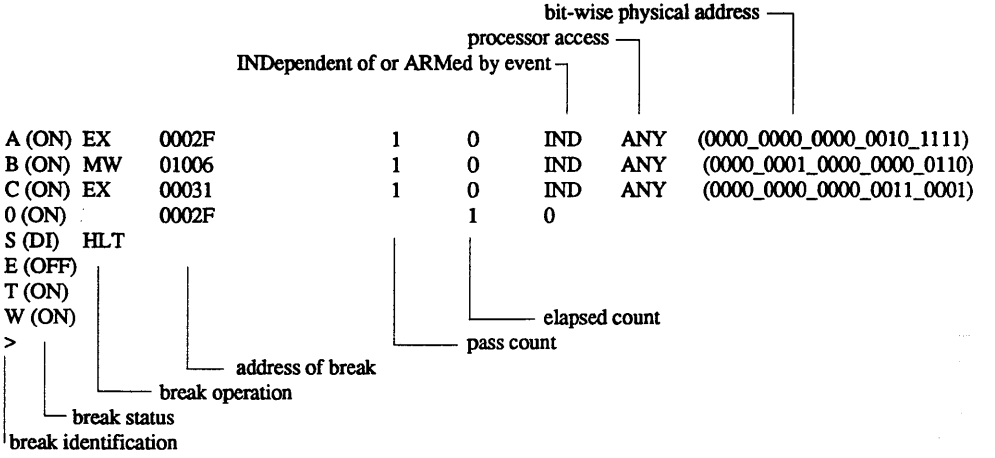
ACTION	COMMAND DEMONSTRATED	COMMENT
PRESS: The blue RESET button on the ICD and look at the screen.		You'll see the ICD's identification message followed by a prompt: ICD-378 for 80186 V1.X > The X represents the firm-ware version. The prompt (>) indicates to you that the ICD is waiting for a command. After you've executed a command or whenever an emulation process is completed, a new prompt will appear. Remember to enter a RETURN <cr> after each command. Now return to your terminal's keyboard.
ENTER: R RESET	REGISTER	Initializes (resets) the registers.
ENTER: R CS=0	REGISTER	Sets the code segment register to 0.
ENTER: R	REGISTER	Displays status of registers.
ENTER: F 0,8000,90	FILL	Fills 32K of internal memory, from address 0 to address 8000, with NOP instructions. It takes a few seconds for the ICD to do this - wait to see the prompt.
ENTER: D 0,13F	DUMP	Displays contents of memory in both hex and ASCII codes.
ENTER: D 140,9FF	DUMP - and the scroll feature	Displays contents of memory. Alternately press the space bar to stop and start the scrolling.
PRESS: The Esc key and a new prompt will appear.		
ENTER: DI 0,15	DISASSEMBLE	Disassembles contents of memory into assembly instructions.

ACTION	COMMAND DEMONSTRATED	COMMENT
		Now you'll enter a test program using the in-line assembler (this program will be used throughout the demonstration).
ENTER: R RES	REGISTER	Resets the registers.
ENTER: R CS=0	REGISTER	Initializes the code segment to 0.
ENTER: R	REGISTER	Displays status of registers.
		Now enter the test program.
ENTER: A 0:0	ASSEMBLE	This command begins assembling the test program into address 0.

ICD displays:	You enter:
0000:0000	MOV SP,0FFFFH
0000:0003	MOV BX,0H
0000:0006	MOV DS,BX
0000:0008	MOV CX,2H
0000:000B	CMP CX,1H
0000:000E	JL 8H
0000:0010	JE 17H
0000:0012	MOV AX,0AAAAH
0000:0015	JMP 1AH
0000:0017	MOV AX,05555H
0000:001A	MOV BX,1000H
0000:001D	MOV [BX],AX
0000:001F	MOV DX,[BX]
0000:0021	CMP AX,DX
0000:0023	JNE 31H
0000:0025	ADD BX,2H
0000:0028	CMP BX,8000H
0000:002C	JNE 1DH
0000:002E	DEC CX
0000:002F	JMP 000BH
0000:0031	JMP 31H
0000:0033	NOP
0000:0034	NOP
0000:0035	<———now enter a second return here to terminate the
>	test program input

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: DI 0,34	DISASSEMBLE	Displays the program just entered. This program tests memory from 1000H to 8000H by writing alternate data patterns of 55s and AAs. After writing to a memory location, a verification is made by a read. In the first part of this exercise, you will use the program to demonstrate how breakpoints are set and how emulation memory is manipulated. You will also perform a trace of the program memory using the real-time trace buffer. Later (still using the test program), you will trace instructions and display the data in a scrolling and single-step manner. The last part of this exercise demonstrates the remaining principal commands. Now back to the demonstration. First, you'll set breakpoints.
ENTER: B	BREAK	This display shows the status of the breakpoints. At this point, none have been set or enabled.
ENTER: B/A EX,2F	BREAK	Sets a hardware (A) breakpoint in the program at address 2FH.
ENTER: B/B MW,1006	BREAK	Sets a hardware (B) breakpoint in the program at address 1006H.
ENTER: B/C EX,31	BREAK	Sets a hardware (C) breakpoint in the program at address 31H.
ENTER: B/0 2F	BREAK	Sets a software (0) breakpoint in the program at address 2FH. (Hardware breakpoint "A" also resides at this same location, however, you won't enable its recognition at this time.)

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: B	BREAK	Displays status of the breakpoints as shown below.



		A,B,C=hardware breakpoint names; 0=software breakpoint names (up to eight names); S=software break opcode; E=event break; T=ready time-out break; W=write protect break.
ENTER: H CLR	HISTORY	Clears the real-time trace clock counter.
ENTER: H BM	HISTORY	Sets the trigger mode of the real-time trace buffer (called up using the HISTORY command) to the Begin Monitor mode. In this mode, the ICD captures all activity from the start of program execution up to the range specification.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: H	HISTORY	Displays the trace status of the real-time trace buffer. Compare with the display below: Clock Counts = 00000000/0 Storage Mode = BM 4093 Storage Size = 0/0 "Clock Counter" indicates the number of clocks (T-states) since the HISTORY command was cleared. "Storage Mode" indicates the trace mode and trace range. "Storage Size" indicates the number of cycles since a program began or since it was resumed.
ENTER: EV ST=M, A=4000, D/W=5555	EVENT	EV is the EVENT command. The event to be recognized is a memory access at address 4000. The data value to match for the event is 5555.
ENTER: EV	EVENT	Displays the status of the EVENT command. Compare with the display below:

```
>EV
 (ON)
Status  = M
Address = 04000          (0000_0100_0000_0000_0000)
Data    = 5555           (0101_0101_0101_0101)
Mode    = ANY
Count   =   1            0
Edge    = ANY
Level   = ANY (Low)
```

		"(ON)" indicates that the EVENT setting has been enabled and is currently active; "Status" indicates the memory type to trigger on; "Address" shows the hex and bit-wise equivalent address to trigger on;

ACTION	COMMAND DEMONSTRATED	COMMENT
		"Data" shows the data value to match for the event; "Mode" shows processor recognition; and "Count" shows the passcount setting. "Edge" and "Level" indicate probe recognition status (they will not be used in this demo). Now you're ready to emulate!
ENTER: G 0	GO	Starts the program. Stops at hardware breakpoint B and displays the current status of the registers.
ENTER: H D RETURN, and then press the space bar several times.	HISTORY	Displays the contents of the real-time trace buffer from the current instruction pointer. The asterisk ("*") indicates trigger recognition; the "Pause" indicates storage buffer termination. Remember to press the space bar to start and stop the scrolling action on the screen.
PRESS: Esc key.		Exits the routine and returns the prompt back to the screen.
ENTER: D 1000	DUMP	Now look at memory location 1000H (will contain "AA").
ENTER: B/B OFF	BREAK	Disables breakpoint B.
ENTER: H EM	HISTORY	Changes to the End Monitor (EM) trigger mode. In this mode, the ICD captures all activity before a breakpoint or monitor break.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: H	HISTORY	Displays the real-time trace buffer status. Compare with the display below: Clock Counts = 0000012F/303 Storage Mode = EM Storage Size = 185/185 Now let's continue emulation!
ENTER: G	GO	Starts the program again and stops when hardware breakpoint A occurs. Now look at the address range 1000-7FFFH - it should contain data of AA hex. Let's find out!
ENTER: D/W FF0,L30	DUMP	Displays a total of 30 bytes in word units (shows beginning of AA data value).
ENTER: D 7FE0,800F	DUMP	Displays end of AA data value.
ENTER: H D 100	HISTORY	Displays the last 100 locations in the real-time trace buffer. (Press space bar to start and stop scrolling.)
ENTER: H EE	HISTORY	Changes to the End Event (EE) trigger mode. In this mode, the trace stops on the event point. The trace buffer then displays the last range number of cycles before and including the event point.
ENTER: H	HISTORY	Displays the current trigger status. Compare with the display below: Clock Counts = 000D207B/860283 Storage Mode = EE Storage Size = Full
ENTER: B/E ON	BREAK	Enables the event (E) breakpoint.
ENTER: G	GO	Starts emulation again, and stops when an event occurs.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: H D 60	HISTORY	Displays the last 60 locations of the real-time trace buffer.
ENTER: D 4000	DUMP	Look at memory location 4000H (will contain "55").
ENTER: G	GO	Starts emulation again, and stops when hardware break A occurs. The address range 1000-7FFFH should now contain the data value 55 hex.
ENTER: D FF0,L30		Displays a total of 30 bytes in word units (shows beginning of 55 data value).
ENTER: D 7FE0,L30	DUMP	Displays end of 55 data value. There's one last breakpoint to examine - the software breakpoint.
ENTER: B/A OFF	BREAK	Disables hardware breakpoint "A" at location 2FH.
ENTER: B S=EN	BREAK	Enables the software breakpoint residing at location 2FH. (NOTE: software breakpoints must be enabled in order to be recognized; disabling them does not change their pre-set specifications in the program.)
ENTER: G	GO	Starts emulation again and stops when the software break occurs. Now you'll see how to use the TRACE command.
ENTER: R RES	REGISTER	Resets the registers.
ENTER: R CS=0	REGISTER	Initializes the code segment.
ENTER: R	REGISTER	Displays status of registers.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: DI 0,34	DISASSEMBLE	Checks to see that the program still exists.
ENTER: T A	TRACE	Traces all instructions (beginning at address 0) and displays them in a continuous manner.
ENTER: G 0	GO	Starts emulation from address 0 and displays all instructions in non-real time. Use the space bar to scroll through the display; press Esc to exit.
ENTER: T J	TRACE	Sets the trace to record all instructions but display only JUMP instructions.
ENTER: G	GO	Starts emulation and displays only JUMP instructions. Use the space bar to scroll through the display; press Esc to exit.
ENTER: T/S A	TRACE	Traces all instructions and displays in a single-step manner (one instruction at a time).
ENTER: T	TRACE	At this point, examine the trace status and compare with the display below:

<ON> /S All 0000:0000-F000:000F (00000-FFFFF)

		This display shows that the trace is active ("ON"), that the single-step ("/S") trace feature is active, that "All" instructions are to be traced and displayed, and that the trace range is 0000:0000H to F0000:FFFFH. (NOTE: You can also single step with the Trace Jump command.)
ENTER: G	GO	Starts program execution in non-real time and displays one line at a time. Press the space bar to display the next instruction; press Esc to exit.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: T CLR	TRACE	Clears the trace setting.
ENTER: T	TRACE	Displays the current trace setting.
		This next example will demonstrate other interesting features of the ICD. In this example, you will use different commands to MOVE, COMPARE and SEARCH through memory, and also examine and change memory locations. Other commands allow reading and modifying of I/O ports. A simple hex-to-decimal command is also shown.
		NOTE: During this example, the space bar and Esc key may be used to control scrolling and exit the display, as in the previous examples.
ENTER: R RES	REGISTER	Resets the registers.
ENTER: R CS=0	REGISTER	Initializes the code segment.
ENTER: R	REGISTER	Displays status of registers.
ENTER: F 1000,10FF,00,55,AA,FF	FILL	Fills memory from 1000 to 10FF with specified data. This command format shows that several values can be entered on the same line.
ENTER: D 1000,113F	DISASSEMBLE	Displays the specified data.
		Now you'll fill memory blocks in order to compare, search and move memory within the ICD.
ENTER: F 2000,L10,0	FILL	Fills 16 bytes of memory with 0, starting at address 2000H.
ENTER: D 2000	DUMP	Displays the 16 bytes of memory just filled.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: F 3000,L10,11	FILL	Fills 16 bytes of memory with 11, starting at address 3000H.
ENTER: D 3000	DUMP	Displays the 16 bytes of memory just filled.
ENTER: F 4000,L10,22	FILL	Fills 16 bytes of memory with 22, starting at address 4000H.
ENTER: D 4000	DUMP	Displays the 16 bytes of memory just filled.
ENTER: CO 2000,L10,3000	COMPARE	Compares memory contents (16 bytes worth) starting at 2000H with the memory contents at 3000H, and displays the locations which are different. (In this case, all the memory contents are different, therefore all addresses are displayed.)
ENTER: S 4000,L10,22	SEARCH	Searches through memory (beginning at address 4000H) and displays memory locations which are equal to the data value of 22. (Remember, three steps back you filled 16 bytes with 22, from address 4000H with 11; therefore, all 16 bytes will be displayed.)
ENTER: S/D 4000,L10, 22	SEARCH	Displays the locations which differ from the searched data. (Since the memory contents are the same at these locations, nothing is displayed.)
ENTER: M 2000,L10,3000	MOVE	Starting at location 2000H, moves 16 bytes of memory (the contents of which are "00") to address 3000H.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: D 3000	DUMP	Displays 16 bytes of memory at address 3000H. Remember, you originally filled address 2000H with 00 (16 bytes worth) and address 3000H with 11 (16 bytes worth). Then you moved the contents (16 bytes) of address 2000H to address 3000H (effectively "writing over" the original memory contents). The memory contents now display 00 at address 3000H.
ENTER: E 4000	EXAMINE	The ICD responds with 04000 22=. (Remember, you originally filled address 4000H with 22, so that's what the EXAMINE command will reveal. You can now change that value directly.
ENTER: AA/	EXAMINE	Changes the value at address 4000H from 22 to AA. The slash at the end terminates the EXAMINE command; however, you can perform several different address changes by using different control characters. This is explained in Section 2.
ENTER: E 4000	EXAMINE	Verifies the change that you made above. To exit the display, press the slash key (/).
ENTER: C 1000-300	CALCULATION	Subtracts two decimal numbers and displays the result in both hex and decimal notation.
ENTER: C 55	CALCULATION	Shows the equivalent hex value.
ENTER: C 289H	CALCULATION	Shows the equivalent decimal value.

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: C 351H-427	CALCULATION	Performs a mixed calculation of hex and decimal values.
ENTER: C 7373H-29+7284	CALCULATION	Performs mixed calculations of several values. This concludes the examples which feature the ICD controlled by a terminal (no target system used). If you have a computer available, you can now use it (through ZICE software) to control the ICD. To find out how to connect your ICD to a computer, see "How To Connect Your ICD To Other Devices," in Section 1. *NOTE: Do not switch off power to the ICD at this point or you will lose the sample program. If this does occur, you must reenter the program found at the beginning of this demo.*
Execute the ZICE software program (use the ZICE documentation to see how).		
PRESS: The RESET switch on the ICD.		The ICD will respond with an identification message and a prompt.
ENTER: SA TEST.H86,0:0,34	SAVE	Remember your test program? This just saved the program to your computer's disk. The SAVE command was summoned with SA; TEST.H86 was what you named it; and the range of the program was 34 bytes (beginning at address 0:0). Let's prove it!

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: Z D *.H86		Displays all of the files on the disk which end in .H86. *NOTE: ZD is the ZICE command to display the directory of ZICE files. Different versions of ZICE may require a different command syntax. See your ZICE documentation for the proper command syntax.* Now reload the program (TEST.H86) back into the ICD but at a different location.
ENTER: L TEST.H86,1000	LOAD	Downloads TEST.H86 to the ICD starting at a new location (address 1000H). The offset is optional; if omitted, 0 is assumed.
ENTER: DI 0:1000,L34	DISASSEMBLE	Displays the program after completing the download.
ENTER: Q	QUIT	Ends the ZICE program and returns to host system control. This is the end of Exercise 1.

Facts About Exercise 2

Name: ICD Product Demonstration - Exercise 2
Controlling Device: Host Computer
Operation Mode: REMOTE
Target System Used?: Yes
Printer?: Optional

Before you begin this exercise, make sure that the ICD is configured for host computer control (if you're continuing from the previous exercise, leave the operating system the same).

Connect the ICD and your target system as outlined in "System Preparation" in Section 1.

Now begin the demonstration.

ACTION	COMMAND DEMONSTRATED	COMMENT
Execute ZICE on the host computer.		
ENTER: I 2	IN-CIRCUIT	Sets the mapping mode to the target system memory only.
ENTER: DI 0:0	DISASSEMBLE	Disassembles 16 bytes of user code.
ENTER: G	GO	Now you're emulating!
PRESS: The MONITOR break switch on top of the ICD to stop emulation.		
ENTER: H D 100	HISTORY	Displays your code (100 locations) that was executed right before the monitor break.
ENTER: I 0	IN-CIRCUIT	"I Zero." No target emulation mode. This mode does not require the use of any target resources. It is used for software development when hardware isn't available to execute code. The ICD depends on it's own internal clock for operation.
ENTER: I 1	IN-CIRCUIT	"I One." Partial target emulation mode (median between I0 and I2 modes). This mode allows mapping of emulation memory to overlay portions of the target memory, and also allows masking out of certain control pins on the microprocessor.
ENTER: I 2	IN-CIRCUIT	"I Two." All mode. Debugging is performed using only the target system memory. Memory mapped as read/write and read-only act as user (target system) memory. I/O and interrupts are enabled. Any area mapped as NO (non-memory) will act as NO memory regardless of the in-circuit mode.

ACTION	COMMAND DEMONSTRATED	COMMENT
		The next example will demonstrate the ability to execute memory out of the ICD and out of the target system. In this example you will move the target's ROM into the ICD's memory space and then execute it out of the ICD. *NOTE: This example is purely theoretical, as it is impossible to duplicate the program in your prototype.* Assume the following theoretical memory map. 0 to 1FFFF is RAM 2000 to EFFFF is NO MEMORY F0000 to FFFFF is ROM Now it looks as though there could be a problem in that the target represents a 1M byte address range while the standard ICD has only 128K byte RAM area. However, this is easily handled by using the ICD's ALLOCATION command.
ENTER: I 1	IN-CIRCUIT	Sets the ICD to partial emulation mode (uses both ICD and target system memory resources).
ENTER: AL	ALLOCATION	Displays the status of the ALLOCATION command. Since no memory blocks have been allocated, the display will look like the one below: 00000-FFFFF = 000

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: MA	MAP	Displays the status of the MAP command. Compare with the display shown below: 00000-FFFFF = RW (000) Notice that all the memory space of the 80186/188 defaults to being internal to the ICD.
ENTER: AL F0000,FFFFF=0	ALLOCATION	*NOTE: The resolution of the MAP command is in increments of 1K byte blocks. Equates the top 64K bytes of target memory to a starting address of 00000H internal to the ICD.*
ENTER: AL	ALLOCATION	Notice the change after the allocation of memory space.
ENTER: MA	MAP	Notice the changes after allocation of the memory space.
ENTER: MA 0,1FFFF=US	MAP	Maps the lower 128K of memory to be accessed from the target system (user).
ENTER: MA 20000,EFFFF=NO	MAP	Maps to non-existent memory.
ENTER: F0000,FFFFF=RO	MAP	Maps to read-only instead of read/ write.
ENTER: MA	MAP	Notice the status of the MAP command. Now you could move the contents of the target ROM into the ICD. If you were using a host computer, you could now download a program to the ICD which was meant for the same address space as the target ROM. Here's how:

ACTION	COMMAND DEMONSTRATED	COMMENT
ENTER: M F0000,FFFFF,F0000,UP	MOVE	F0000,FFFFF is target address, F0000 is the ICD start address, UP means move User (target) memory to program (ICD) memory. Now that the contents of the target ROM are in ICD memory, you could begin emulation. With the contents of the target ROM internal to the ICD, the code can now be modified using the in-line assembler and then checked out for proper execution by setting breakpoints and using the real-time trace buffer, or simply tracing to the display. Once the new code has been checked, it can either be saved to a host computer or sent out the "HOST/ AUX" port to a prom programmer for burning a new prom. Lastly, let's examine the PIN command. This command allows you to manipulate certain control pins of the microprocessor when debugging in the I1 mode. *NOTE: The PIN command can only be used in the I1 mode.*
ENTER: I 1	IN-CIRCUIT	Sets mapping to I1 (partial) mode.
ENTER: PI	PIN	Displays the status of the PIN command. Notice that the signals that can be enabled and disabled show (EN) or (DI).
ENTER: PI INTR=DI	PIN	Disables the interrupt pin to the emulation processor.
ENTER: PI	PIN	Notice the difference from the previous status request. This is the end of exercise 2.

ICD General Specifications

Emulated Processors /Clock Speed	80186 CPU/8MHz 80188 CPU/8MHz 8087 NDP/8MHz
Memory Size Optional Memory	128K bytes static RAM Expandable to 1M byte, externally
Mapping Resolution	1K-byte blocks
Real-time Trace Buffer	4K bytes deep x 40 bits wide
Debugger Commands	30 (resident)
Breakpoints	4 hardware, 8 software
Usable I/O Ports	64K
Communication Ports	Two RS-232C/20mA current loop/TTL
Baud Rates	14 different rates; 75 to 19200bps (factory set at 9600bps for each port)
Physical Dimensions	300mm (11.8in) wide 210mm (8.2in) deep 140mm (5.5in) high
CPU Probe Length NDP Probe Length	490mm (19in) long 540mm (21in) long
Weight Power Requirements Power Consumption Operating Temperature Storage Temperature Humidity	5.5kg (12.2lb) 115VAC/230VAC; 50/60Hz 50 watt 0 to 45 degrees C -10 to 55 degrees C 30% to 85%; relative humidity (non-condensing)

ICD Emulation Specifications

Memory Area	Emulation Memory. The entire ICD memory area (128K bytes) is available to the target system (prototype hardware) in the form of high-speed static RAM. Mapping. The ICD internal memory can be mapped in 1K-byte blocks. The mapping modes include: user (target), emulation read/write memory, emulation read-only memory, and non memory.
I/O Ports	All 64K ports are open.
Breakpoints	4 hardware and 8 software Hardware Breakpoints. A, B, C and Event trigger. All hardware breakpoints can be individually enabled and disabled. A, B, C Breakpoints. Address 20 bits, BHE. Each bit may be specified 0, 1 or "don't care." Status may be specified as opcode fetch, memory access, memory read, memory write, I/O access, I/O read, and instruction execution. Event Trigger Breakpoint. Address 20 bits, BHE. Each bit may be specified 0, 1 or "don't care." Status may be specified as opcode fetch, memory access, memory read, memory write, I/O access, read/write, and instruction execution. Data: 16 bits. Each bit may be specified 0, 1 or "don't care." External Trigger Breakpoint. 1 channel-TTL level specified at hi or lo signal edge, or hi or lo signal level. Software Breakpoints. 8 points; 0-7. Any point may be specified as a software breakpoint by using the HLT or INT3 instruction. All software breakpoints can be individually enabled and disabled. A break is caused in the target system when the CPU read F4H as an opcode. Execution of a software breakpoint does not affect the registers or flags.

Real-time Trace	Operation: During emulation, all address, data and status lines are stored in the real-time trace buffer. Trace capacity: 4K bytes deep x 40 bits wide. Trigger functions include: End Monitor, Begin Monitor, End Event, Begin Event, Center Event and Multiple Event.

Technical Bulletins & Application Notes

Introduction

Things are constantly changing in the microprocessor industry, and ZAX wants to help you stay on top of these changes. New products, emulation methods, and applications are always being devised and tested by us in an effort to provide you with the latest and most effective equipment possible. In the same manner, revising your existing equipment keeps it current with the latest ICD designs from ZAX.

One of the best ways we have of keeping you up-to-date is by issuing Technical Bulletins and Application Notes.

Technical Bulletins

Technical Bulletins inform you of major changes or revisions to the equipment's hardware or firmware. Usually they are the result of a problem that's recently been solved, or they could be a feature that's been revised to improve the performance of the emulator.

Application Notes

Application Notes are the result of new methods or procedures derived from emulation practices. They may also caution you against doing something a certain way, or they may show you a new way of accomplishing an old task.

Both Technical Bulletins and Application Notes are sent to you as soon as they become available - you should never need to request them. When you receive your documents, insert them into this appendix for easy reference. That's all there is to it!

Logic-State Analyzer Interface

The ZAX ICD-378 series emulators now feature a logic-state analyzer (LSA) interface. This feature permits ZAX emulators to interface directly to various logic-state analyzers.

The combination merges an LSA's sophisticated trigger, qualification, data-acquisition and measurement capabilities with the ICD's debugging mechanisms and loan facilities.

The LSA connects to the ICD's top panel (labeled "STATE SIGNAL OUTPUT") where it gains access to all lines of the emulated microprocessor - no other connections are required. Two additional lines from the ICD are also available to act as an "emulation in progress" indicator to the LSA (pin #21), and provide event trigger recognition during program execution (pin #22). A third line clears the event trigger setting from the ICD (pin #1). (See the pin-out listing on the next page.)

The LSA interface provides many useful features. Interface software packages now permit a single computer, such as an IBM PC/AT/XT, to control both LSA and ICD. And because the interface is conducted through the ICD, the lines of the microprocessor may be monitored by the LSA during an emulation memory access cycle (the majority of emulators possessing interface capabilities lack this ability).

CD State Signal Interface Connector Specification

Pin	Name (Type)	Pin	Name (Type)
1	EVTCLR (Input Signal)	2	not connected
3	not connected	4	T4 (ICD signal)
5	not connected	6	not connected
7	ALE (Processor Signal)	8	not connected
9	GND	10	A16 (Processor Signal)
11	A17 (Processor Signal)	12	A18 (Processor Signal)
13	A19 (Processor Signal)	14	S3 (Processor Signal)
14	S4 (Processor Signal)	16	S5 (Processor Signal)
17	S6 (Processor Signal)	18	BHE (Processor Signal)
19	AD0 (Processor Signal)	20	OF (ICD Signal)
21	EMQUAL (ICD Signal)	22	EVENT (ICD Signal)
23	SO (Processor Signal)	24	S1 (Processor Signal)
25	S2 (Processor Signal)	26	GND
27	AD0 (Processor Signal)	28	AD1 (Processor Signal)
29	AD2 (Processor Signal)	30	AD3 (Processor Signal)
31	AD4 (Processor Signal)	32	AD5 (Processor Signal)
33	AD6 (Processor Signal)	34	AD7 (Processor Signal)
35	AD8 (Processor Signal)	36	AD9 (Processor Signal)
37	AD10 (Processor Signal)	38	AD11 (Processor Signal)
39	AD12 (Processor Signal)	40	AD13 (Processor Signal)
41	AD14 (Processor Signal)	42	AD15 (Processor Signal)
43	GND	44	AD0 (Processor Signal)
45	AD1 (Processor Signal)	46	AD2 (Processor Signal)
47	AD3 (Processor Signal)	48	AD4 (Processor Signal)
49	AD5 (Processor Signal)	50	AD6 (Processor Signal)
51	AD7 (Processor Signal)	52	AD8 (Processor Signal)
53	AD9 (Processor Signal)	54	AD10 (Processor Signal)
55	AD11 (Processor Signal)	56	AD12 (Processor Signal)
57	AD13 (Processor Signal)	58	AD14 (Processor Signal)
59	AD15 (Processor Signal)	60	GND

Suggested Reading

Microprocessor Development and Development Systems
TSENG, Vincent. Granada Publishing Limited, 1982

Presents the fundamentals of microprocessor development systems, their uses, basics of operation, and applications. Includes study of development software and the use of logic analyzers.

"In-circuit Emulation Makes Software Development Manageable," Computer Design, April 1, 1986

"Development Systems Target Software Intensive Projects," Computer Design, March 15, 1986

"Development Systems for 32-bit Microprocessors" EDN, June 13, 1985

"Debugging Techniques"
BYTE, June 1985

"Emulators, Backed By Strong Debugging, Mimic Latest Microprocessors" Electronic Design, October 18, 1984

"What Is Emulation And Where Is It Going?" Test & Measurement World, June, 1984

"In-circuit Emulation" Computers & Electronics, May, 1984

"Strategies And Tools For Developing A Microprocessor -based System" Electronics Test, April, 1984

address — A character or group of characters (such as a label, name, or number) that identifies a register, location, or unit where information is stored.

allocate — To assign blocks of data to specified blocks of storage.

argument — An independent variable upon whose value the value of a function depends. Usually the arguments of a function are listed in parentheses (sometimes within brackets) after the function name, if a function name is used.

ASCII — [ask-ee] American Standard Code for Information Interchange. A standard 8-bit information code used for information interchange between equipments of different manufacturers.

assemble — To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable address for symbolic addresses. (With ZAX ICD-series emulators, this operation is performed using the ASSEMBLE command.)

assembler — A computer program which operates on symbolic input data to produce machine instructions. An assembler generally translates input symbolic codes into machine instructions - item for item - and produces the same number of instructions or constants which were defined in the input symbolic codes.

baud rate — The number of bits that are transmitted per unit of time (seconds). By definition, a baud is the reciprocal of time - in seconds - occupied by the shortest element of the code being transmitted, e.g., if the duration of the shortest signal element is 20 milliseconds, the modulation rate of the code is 50 bauds per second. (ZAX ICD-series emulators can transmit up to 19,200 bits per second.)

bi-directional data bus — A data bus in which digital information is transferred in either direction, thus saving time and providing easy access to stored information.

binary — A numbering system based on 2's rather than 10's, in which only the digits 0 and 1 are written.

bit	A binary digit.
branch	To depart from the normal sequence of executing instructions in the computer. (Synonymous with a jump.)
breakpoint	A point in a program as specified by an instruction where the program may be interrupted by some external intervention or by a monitor routine. This program break permits a visual check, print out, or other analysis of the program before resuming with the normal sequence. Used extensively in debugging operations. (With ZAX ICD-series emulators, there are four hardware and eight software breakpoints available by using the BREAK command.)
buffer	A storage device in which data is assembled temporarily during data transfers. It is used to compensate for the differences in the rate of flow of information when transferring information from one device to another.
byte	A sequence of adjacent binary digits operated upon as a unit and usually shorter than a computer word.
C	A high-level programming language designed to optimize run time, size, and efficiency. It was developed as the systems programming language of the UNIX operating system on the PDP 11/70 minicomputer from Digital Equipment Corporation.
CLK	clock
clock	Devices or units which control the timing of bits sent in a data stream and the timing of the sampling of bits received in a data stream. One such clock device is a real-time clock, which measures the past or used time on the same scale as the external events it will be used to describe. Most microprocessor clocks operate in the range of 1 to 12 MHz.
code	A group of symbols that represent data or instructions in a computer. Digital codes may represent numbers, letters of the alphabet, control signals, etc. as a group of separate bits rather than continuous signals. (See microcode.)
compiler	A computer program, more powerful than an assembler, that will convert a higher level language into machine language.

computer control of the ICD — A remote mode in which the ICD's input/output is controlled by a host computer using a utility software program designed for that computer. In this mode, the host computer sends commands and receives data via an RS-232C interface.

controller — A device which interfaces a peripheral to a computer in order to relieve the processor of device-controlled responsibilities.

CPU — Central Processing Unit. The module within a computer that is responsible for fetching, decoding, and executing instructions. It contains a main storage unit, arithmetic unit, and special register groups.

cross assembler or cross program — A program run on one computer for the purpose of translating instructions from a different computer.

cross compiling/ assembling — A method in which an existing computer can be used to write and debug what will become a microcomputer program. The advantage to cross compiling/assembling is that designers can have access to all of the conventional peripherals so that the object code produced during development can then be loaded into the microcomputer system.

CRT — Cathode Ray Tube. A television tube used to display alpha-numeric characters and graphics.

DCE — Data Communications Equipment. Refers to devices used for the transmission of information from one point to another.

debugging — A process of eliminating "bugs" in a system by isolating and correcting all malfunctions and/or mistakes in a piece of equipment or a program of operations.

decimal, binary-coded — Describing a decimal notation in which the individual decimal digits are represented by a pattern of 1's and 0's, e.g., in the 8-4-2-1 coded decimal notation, the number 12 is represented as 0001 and 0010 for 1 and 2. This contrasts with a straight binary notation where 12 is represented as 1100.

default value — The choice among exclusive alternatives made by the system when no choice is made by the user.

development system — A system of devices, usually consisting of a diagnostic tool (such as an emulator), a computer, a printer, etc., that can be used together to develop and debug hardware and software for a given microprocessor.

development tools — Hardware and software devices that are used to develop and debug programs and/or microprocessor systems.

DIP — Dual In-line Package. A standard IC package with two rows of pins at 0.1" intervals.

DIP switches — A collection of small switches on a DIP that are used to select options on circuit boards without having to modify the hardware.

disassembly (disassembler) — Refers to a program that translates from machine language to assembly language. Usually used to decipher existing machine language programs by generating symbolic code listings of a program.

don't care — A term applied to an operation which can be changed or interrupted upon receipt of a control signal. The output of the operation is independent of the input.

downloading — A process whereby a file is loaded "down" to a device from a computer system.

DTE — Data Terminal Equipment. Equipment comprising of a data source (transmitter) or data sink (receiver) that provides for the communication control functions (protocol).

dump — The process of transferring the contents of memory at a given instant of time onto a screen for viewing, or outputting the memory contents to a printer.

duplex — A simultaneous two-way independent transmission.

dynamic RAM — Memory that requires constant refreshing in order to store memory.

EAROM — Electronically Alterable Read Only Memory. A specialized random access read/write memory with a special slow write cycle and a much faster read cycle.

echo check — An accuracy check of a transmission in which the transmitted information received by an output device is returned to the information source and compared with the original information.

editor — A general-purpose text-editing program that allows entry and maintenance of text in a computer system. The original text is entered and held in memory where it can then be changed and corrected by inserting, deleting, or changing lines of text or characters within a line.

EEPROM — Electronically Eraseable Programmable Read-Only Memory. An EEPROM is a device that can be erased electrically in one second and reprogrammed up to a million times.

EEPROM programmer — A unit that provides a means of programming a single EEPROM or an EEPROM module from a terminal.

EIA-RS-232C — A standard method adopted by the Electronic Industries Association to ensure uniform interface between data communications equipment and data processing terminal equipment. (All ZAX ICD-series emulators use the EIA-RS-232C standard interface.)

emulation — Techniques using software or microprogramming, in which one system is made to behave exactly like another system, i.e., the emulating system executes programs in the native machine language code of the emulated system.

emulation mode — The mode that the ICD assumes in order to execute instructions.

emulator — An instrument that imitates the control memory of future hardware. Also a device that causes a system (such as the target hardware) to accept certain software programs and routines and appear as if it were the other system.

emulator, stand-alone — An emulator whose execution is not controlled by a control program. It also does not share system resources with other programs and excludes all other jobs from the computing system when it is being executed.

EPROM — Eraseable Programmable Read-Only Memory. A specific type of ROM that can be programmed electrically. It can retain data even with the power disconnected but can be erased by exposure to short wavelength ultraviolet light, and may be reprogrammed many times thereafter. Other types of EPROMs may be electrically erased. (See EEPROM.)

field — A set of one or more characters which is treated as a whole.

firmware — Programs that are stored in a physical device (e.g. ROM) that can form part of a system or machine.

FIFO — First In, First Out.

First In, First Out — A method or technique of storing and retrieving items from a storage source. With this technique, the "oldest" data transmitted to the storage source is thrown out and replaced with the "newest" data. The technique is useful when the storage source capacity is smaller or less than the item quantity.

FORTRAN — FORmula TRANslator. A high-level language developed by the IBM Corporation, originally conceived for use on scientific problems but now used for many commercial applications as well. It requires the use of a compiler.

full duplex — A mode of communication in which data can be transmitted and received simultaneously.

gate — A device that has one output channel and one or more input channels such that the condition of the output state is determined by the state of the input channel. The NAND, NOR, AND, OR, XOR, and NOT functions are examples of gates.

GND — Ground

half-duplex — A mode of communication in which data may be transmitted in only one direction at a time.

halt — A condition which occurs when an operation in a program stops.

handshaking — A sequence of signals that are required for communication between different systems.

hardware — Physical (electric, electronic, or mechanical) equipment - as opposed to a computer program - used for processing data. Contrast with software.

hex, hexadecimal — Pertaining to a number system with a base of 16. The digits 0 through 9 are used, then A through F, to represent decimal numbers 0 through 15, e.g., "FF" represents "11111111" binary, and "0A" is "00001010" binary.

high-level language — Any group of computer languages which use symbols and command statements an operator can read. High-level languages allow a user to write in a familiar notation rather than the machine code language of a computer. BASIC, FORTRAN, FOCAL, and COBOL are all examples of high-level languages.

host computer — The primary or controlling computer in a system operation. A host computer can also be reduced to a simple memory storage facility.

ICE — In-Circuit Emulation.

in-circuit emulation — Hardware/software facilities for real-time I/O debugging of chips. With in-circuit emulation, the actual microprocessor is replaced by a connector whose signals are generated by an emulation program. The emulated microprocessor can be stopped, its registers examined or modified, etc.

instruction — A coded program step that tells the computer what to do for a single operation in a program.

instruction set — The basic set of instructions that a particular computer can perform.

interface — The physical connection between two systems or two devices.

interrupt — A break in the normal flow of a system or program that occurs in such a way that the flow can be resumed from that point at a later time.

journaling Refers to a process where all information generated in an emulation session on a host computer is output to a storage file. The entire session can then be reviewed, line for line, just as it was initially entered.

kilobaud Refers to the number of one thousand bits per second. (ZAX ICD-series emulators are capable of transmitting at speeds to 19.2 kilobaud.)

linking loader A loader used to link compiled/assembled programs, routines, and subroutines and turn the results in operations.

loader A program required on practically all systems that load the user's program along with system routines into the central processor for execution. Loaders transfer the object code from some external medium (tape or disk) into RAM.

logic state analyzer (LSA) A device that monitors a system or component board and displays the resulting information.

machine cycle The time interval in which a computer (or similar device) can perform a given number of operations.

machine language A set of symbols, characters, or signs, and the rules for combining them, that conveys instructions or information to a computer.

macro Pertains to a specific type of instruction in assembly language that is implemented in machine language by more than one machine-language instruction, e.g., a group of instructions often designed to serve as an additive command or group of commands.

macro assembler An assembler that is capable of assembling object programs from source programs written in symbolic language.

macro instruction An instruction which stands for a predefined sequence of other instructions, called the "body" of the macro. Whenever a macro instruction is encountered in program text, it is "expanded," i.e., replaced by its body.

mainframe, main frame — Usually refers to large-scale computers (as opposed to microcomputers, microprocessors, and minicomputers). May also mean the fundamental portion of a computer, i.e., the portion that contains the CPU and controller units within the computer system.

microcode — A set of control functions performed by the instruction decoding and execution logic of a computer system. The microcode defines the instruction set of a specific computer.

mnemonic code — Refers to techniques used to assist human memory. A mnemonic code resembles the original word and is usually easy to remember, e.g., mpy for multiply, acc for accumulator.

monitor mode — Refers to a process where monitor commands from the ICD are executed. Dump, Fill, Disassemble, and Examine are all examples of commands used in the monitor mode.

MOS — Metal-Oxide Semiconductor. A technology used for fabricating high-density ICs. The name refers to the three layers used in forming the gate structure of a field-effect transistor.

NOP, NOOP — NO-OPeration. An instruction used to force a delay of one instruction cycle without changing the status flags or the contents of the registers.

object code — The code produced by a compiler or special assembler which can be executed by the processor when it is loaded, as with most microcode, or it may require a linkage phase prior to loading and execution.

object program library — An organized set of computer programs, routines, or common or specifically designed software, containing various programs or routines, source or object programs classified for intelligence or retrieval.

operating system — Software that is required to manage the hardware and logical resources of a computer system. Also a part of a software package (program or routine) dedicated to simplifying input/output procedures, sort-merge generators, data-conversion routines, or tests.

operation code — The symbols that designate a basic computer operation to be performed. This can be a combination of bits specifying an absolute machine-language operator, or the symbolic representation of the machine-language operator.

operator — A symbol in programming language that represents an operation to be performed on one or more commands (e.g., "add x").

parameter — A constant or variable in an equation or statement that may be assigned an arbitrary value.

parity bit — A redundant bit added to a group of bits so that an inaccurate retrieval of that group of bits is detected.

Pascal — A language designed to teach programming and the elements of computer science. Based on the language ALGOL, it emphasizes aspects of structured programming.

peripheral devices — Various kinds of machines or devices that operate in combination with a computer but are not physically part of the computer. Peripheral devices typically display computer data, store data from the computer and return it to the computer on demand, prepare data for human use, or acquire data from a source and convert it to a form usable by a computer.

phantom ROM — A type of ROM that operates for the system bootstrap only and then "hides" behind the main memory.

PIO interface — Abbreviation for Parallel Input-Output interface. PIO interfaces allow the computer to input and output parallel data to and from an external parallel device such as a keyboard or printer. Parallel means that all the data bits output at the same time.

PROM — Programmable Read-Only Memory. A ROM that may be altered by the user. Some PROMs can be erased and reprogrammed through special physical processes.

PROM programmer — A module or external device used to program programmable read-only memories.

PROM programming — The process of altering PROMs (called "burning"), either by blowing (melting or vaporizing) fusible links in bipolar PROMs or by storing a charge on the floating gates of UVEPROMs.

RAM — Random Access Memory. This type of memory is random because it provides access to any storage location point in the memory by means of horizontal and vertical coordinates. Information can then be "written" in or "read" out very quickly.

read-only (RO) — Refers to a process where information can be read from memory only.

read-write (RW) — Refers to a process where information can be read from and written into memory.

real time — Pertains to the actual time during which a physical process transpires. In emulation, real-time operation is very important because of the necessity for the emulator to maintain a "transparent" condition with regard to the device being emulated. (All ZAX emulators are capable of real-time emulation with no wait-states introduced for accessing memory.)

register — A memory device capable of containing one or more computer bits or words to facilitate arithmetical, logical, or transferral operations.

ROM — Read Only Memory. A special memory that can be read into but not written into.

RS-232C — See EIA-RS-232C.

semantics — The relationship between symbols and their intended meanings independent of their interpretation.

source program — A program coded in something other than machine language that must be translated into machine language before use.

stand-alone — Pertains to a device that requires no other piece of equipment to execute and complete its own operation.

stand-alone system Usually, a microprocessor development system (MDS) that runs independent of the control of a computer. The MDS may contain a terminal and built-in display facility, which in effect makes it a full microcomputer with debugging capabilities.

statement An instruction (macro) to the computer or other related device, to perform some sequence of operations.

static RAM RAM that does not need to be refreshed or receive any further attention as long as power is applied.

step One instruction in a computer routine.

stop bit The last element of a character that defines the character space immediately to the left of the most significant character in accumulator storage.

symbolic debugging Symbolic commands that are used to assist in the debugging procedure. Symbolic refers to codes which express programs in source language, i.e., by referring to storage locations and machine operations by symbolic names and addresses that are independent of their hardware-determined names and addresses.

symbolic trace A process where addresses in a program trace are replaced with symbols. The symbol conversion process is performed in the host system using the appropriate software program.

syntax Rules that govern sentence structure in a language, or statement structure in a language such as that of a compiler program.

target system Refers to the processor under development.

trace Refers to the operation of the real-time trace buffer (storage facility) and its ability to capture and store a portion of the program memory area.

transparency The ideal emulation condition in which the operation of the target system is unaffected when the emulator is substituted for the microprocessor. Transparency can be broken down into two categories: functional and electrical. To be functionally transparent, the emulator should make no demands on any part of the target system's resources such as interrupts and memory allocation. To be electrically transparent, the emulator should duplicate as closely as possible the microprocessor's characteristics, such as timing and clock speed.

trigger Refers to a user-specified reference point (external to the user program) which determines where and when to initiate or terminate a program trace.

TTL Transistor Transistor Logic. A family of integrated circuit logic elements with a specific output structure, usually +5 volt "ones" and 0 volt "zeros."

universal emulator A single emulator that is able to support several different processors, even from different manufacturers.

uploading A process whereby a file is transferred from an device to the computer system.

virtual memory Refers to a technique that permits users to treat secondary memory (disk) storage as an extension of main memory and thus give the virtual appearance of a larger main memory.

XON/XOFF Transmitter ON/OFF.

ZICE, ZICE-II Refers to the series of communications program which permit symbolic debugging and also interface different computer systems to ZAX ICD-series emulators.