iceMASTER®-WA
COP8-EM/DM/IM-Flash
User's Manual
for Microsoft® Windows®

Document Version: 1.1

Table of Contents

Chapter 1: Introduction

The Emulator

The iceMASTER-WA COP8-*x*M-Flash emulator is a tool for designing, debugging, programming and evaluating COP8 Flash Microcontroller Unit (MCU) devices. By providing all of the essential MCU timing and I/O circuitry, the iceMASTER-WA simplifies evaluation of the prototype hardware/software product.

The iceMASTER-WA is an in-system emulator controlled by an IBM PC (or compatible) running the Windows operating system. The iceMASTER-WA is an integral part of the development engineer's toolbox, with applications in device evaluation, software development and hardware integration.

The iceMASTER-WA can be operated in one of three configurations:

- Connected to a standard 2x7 header in your Target System.
- Connected to an optional Probe Card that plugs into a socket for the COP8 Flash device in your Target System. The Probe Card contains the necessary 2x7 header for connection to the iceMASTER-WA emulator.
- Connected to an optional Null Target Board for Stand-Alone (independent) Operation. Stand Alone Operation mode allows you to emulate hardware and/or execute code without a target system (provided no interaction with external devices is needed).

Hardware designers may use the iceMASTER-WA to develop and debug their designs, including programming their software into COP8 Flash devices. All available features of a given device are accessible interactively, as well as through your application programs. Software designers have complete emulation capability as well.

Recommended References

Several additional references can be of help to you as you progress through the development process. The data book and programmers guide for the microcontroller you are using provide essential information. You will also need the programmer's manual for the development language you are using.

What You Need To Know

Throughout this manual, it is presumed that you have a working knowledge of:

- The family of microcontrollers you are emulating
- The IBM PC (or compatible) as an engineering tool
- A development language (e.g., Assembly Language or C)
- Microsoft Windows 3.11, Windows 95, Windows 98 or Windows NT 4.0.

A few of these topics are discussed in this manual as a means of illustrating a particular feature or facet of the iceMASTER-WA's capabilities; however, basic programming knowledge and familiarity with the microcontroller architecture are assumed.

Chapter 2: Hardware Installation

iceMASTER-WA COP8-EM/DM/IM-Flash Hardware

Connect one of the modular adapters to a serial communication port on the Host Computer. Be sure you are using Communication Port 1 (COM1), 2 (COM2), 3 (COM3) or 4 (COM4).

- Use the modular-to-DB-25 adapter for a PC type Host Computer (one with a 25-pin serial communication port connection at the back of the computer).
- Use the modular-to-DB-9 adapter for a Pentium/486/386 type Host Computer (one with a 9-pin serial communication port connection at the back of the computer).

Connect one end (either end) of the RS-232 cable to the emulator and the other end to the modular adapter attached to the Host Computer.

Connect the power supply to the emulator by inserting the power supply's connector into the emulator power receptacle. For safety, we recommend that all items in your system, including emulator, Host Computer and target, be connected to the same outlet. Different outlets (even though near one another) may be connected to different circuits in the building, resulting in large potential differences between grounds.

Connect the14-pin flat cable to one of:

- A 2x7 header on your Target System.
- The 2x7 header on the optional Probe Card.
- The 2x7 header on the optional Null Target Board.
  Note that the emulator will supply 2.7V to the Null Target Board if you have not connected the Null Target Board to an external power supply.

Note that the emulator end of this 14-pin flat cable is permanently attached to the iceMASTER-WA emulator.

You should never turn the power to the emulator base on when the 14-pin cable coming from the emulator is not connected to your Target System or to the Probe Card or to the Null Target Board. Note that the emulator will not run if it is not connected in one of the above three ways.

Warnings

Hot-Plugging

The emulator is not designed to be hot-plugged. The emulator and target system must be turned off when attaching or removing the emulator from the target system. In addition, the emulator must be turned off when attaching or removing the probe card. Hot-plugging may cause CMOS latch-up and can void the warranty.

Power On/Off Sequence

We recommend that you apply/remove power to the components of the system in the following order:

Power-Up

1. Turn the emulator on.
2. Turn the target system or Null Target Board on.

Power-Down

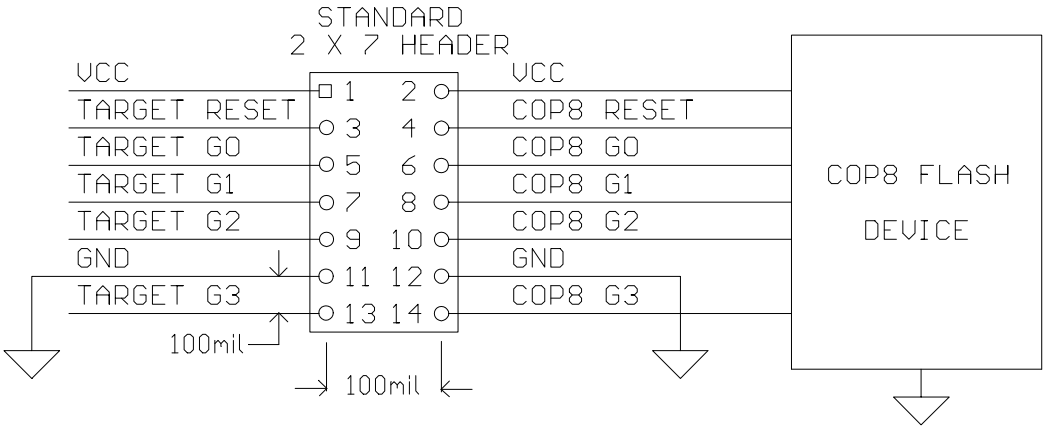
1. Turn the target system or Null Target Board off.
2. Turn the emulator off.

If you apply power in a different sequence and your target system does not have a “good” Power-On Reset circuit, or if you are using the Null Target board, flash memory in the COP8 flash device could be corrupted. Without a “good” Power-On Reset circuit, the COP8 processor could start executing “randomly” in the device’s Boot ROM code; it could execute some code that writes to, or mass erases, the flash memory in the part. The recommended Reset circuit description can be found in the National Semiconductor data book for the COP8 device you are using.

Chapter 3: Hardware Description

Target Board to Emulator Connection

The physical connection between the target system printed circuit board (PCB), containing the COP8 flash device, and the emulator is accomplished with a standard 2x7 header. To use the emulator with the COP8 device soldered on the target system PCB, you would design your PCB with a 2x7 header pattern located next to the G Port of the COP8 flash device. The connections to this header pattern are:



The characteristics of the 2x7 header are:

- 100 mil spacing between all pins. (100 mils = 0.1")
- Pins are square, 25 mils on each side.
- The header has a 230 mil insertion length (how much of the pin inserts into the connector).
- Example manufacturer and part numbers:

  3M: 2380-5121TN

  Molex: 10-88-1801

  Note that both the 3M and the Molex part numbers are for 2x40 headers. You must "break" them to create a 2x7 header. This is easy to do, as the parts have been designed for this.

In development mode, you would assemble your PCB with the header. A mating connector from the emulator would be placed on the header. The signals from the COP8 flash device on your target system PCB are connected to the emulator through one side of the header. The recreated ports and target reset signal from your target system PCB are connected to the emulator through the other side of the header.

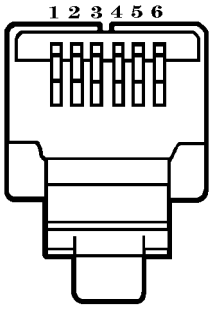
In non-development mode, you would short the pins across the header using standard shorting blocks. This would connect your target system logic directly to the pins on the COP8 flash device on your PCB. For production, the header need not be inserted and the PCB would then be assembled with shorting wires connecting adjacent holes (1 to 2, 3 to 4, etc.) in the header pattern.

RS-232 Modular Interface

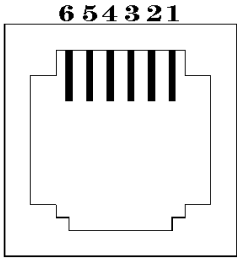
Host PC Cable Connector			Cable		Emulator Base Cable Connector	
Signal	Pin		Function	Direction	Pin	Signal
	Female DB-25	Female DB-9			Male Modular	
TxD	2	3	Data to ICE	→	3	RxD
RxD	3	2	Data to Host	←	4	TxD
RTS	4	7	Reset ICE – active low	→	6	RTS
CTS	5	8	Handshake	←	5	CTS
Ground	7	5	DC Ground	↔	2	Ground
DTR	20	4	Handshake	→	1	DTR

The communication link to the Host Computer is based on the serial RS-232-C specification. The serial baud rates are established entirely under software control. Therefore, you do not need to adjust your serial ports baud rate manually.

A 6-Conductor Reversed Telephone Cable Assembly is provided with the emulator; this cable has a 6-pin male modular plug on each end. The cable mates via a 6-pin male modular plug on the cable at the emulator end:



This plugs into a 6-pin female modular jack on the emulator chassis:



At the host end, the 6-pin male modular connector plugs into a 25-pin female DB-25 adapter for a PC type Host Computer or a 9-pin female DB-9 adapter for a Pentium/486/386 type Host Computer. Note that DB-9 Pins 2 and 3 are reversed from their normal 25-pin D connector assignments in the 9-pin RS-232-C interface of the PC AT. Also, note that the total length of the cable should not exceed 25 feet.

Note that some problems have been observed when using communication ports (on the host computer) that have an 8250 type UART. If you are having communication problems and all attempts at fixing the problem have failed and your communication port uses an 8250 type UART you might try replacing the communication port with one that uses

a 16450 or 16550 type UART. The program MSD.EXE from Microsoft can be used to determine what type of UART your communication port uses.

LEDs

NOTE: The Red and Green LEDs behave in the way described here only if the "Firmware ID" displayed by the ***Configure|Identification*** command is 21.07 or greater.

Red LED (Power)

The red LED blinks quickly (4 times per second) after the power switch on the emulator is turned on, but before any communication with the host computer and debugger has been established over the RS-232 communication link/line.

Once RS-232 communication with the host computer and debugger has been established, the red LED stays on (steadily).

Green LED (Status)

The green LED turns on when the interface between the emulator and the COP8 flash device in the target system is established.

The green LED turns off when the COP8 processor is "inactive":

- Upon RESET
- In Halt/Idle mode.
  This occurs both in emulation condition (when the COP8 processor is executing your code) and in break condition (when the COP8 processor has stopped (e.g., after reaching a breakpoint or a host-break)).

The green LED blinks slowly (2 times per second) during emulation (when the COP8 processor is executing your code).

Switches

SW1—POWER

The POWER rocker switch controls power coming in on the power connector/jack.

Power Jack

5 Volt-Powered Emulators

Emulators shipped before June 2000 are configured to use a 5 volt power supply. These units support only 5 volt operation of the COP8 flash device.

Power is supplied with a standard 2.5mm x 5.5mm x 9.5mm **barrel** plug and jack, center positive (Switchcraft 760). The power supply must provide +5 volts 5%, at 1 ampere. The ripple voltage must be no greater than 50 millivolts, peak-to-peak.

9 Volt-Powered Emulators

Emulators shipped after June 2000 are configured to use a 9 volt power supply. These units support a COP8 flash device operating over the full, wide voltage range (2.7 volts – 5.5 volts).

Power is supplied with a standard 2.5mm **phone** plug and jack, tip positive. The power supply must provide +9 volts 5%, at 300 milliamps. The ripple voltage must be no greater than 300 millivolts, peak-to-peak.

Chapter 4: Software Guide

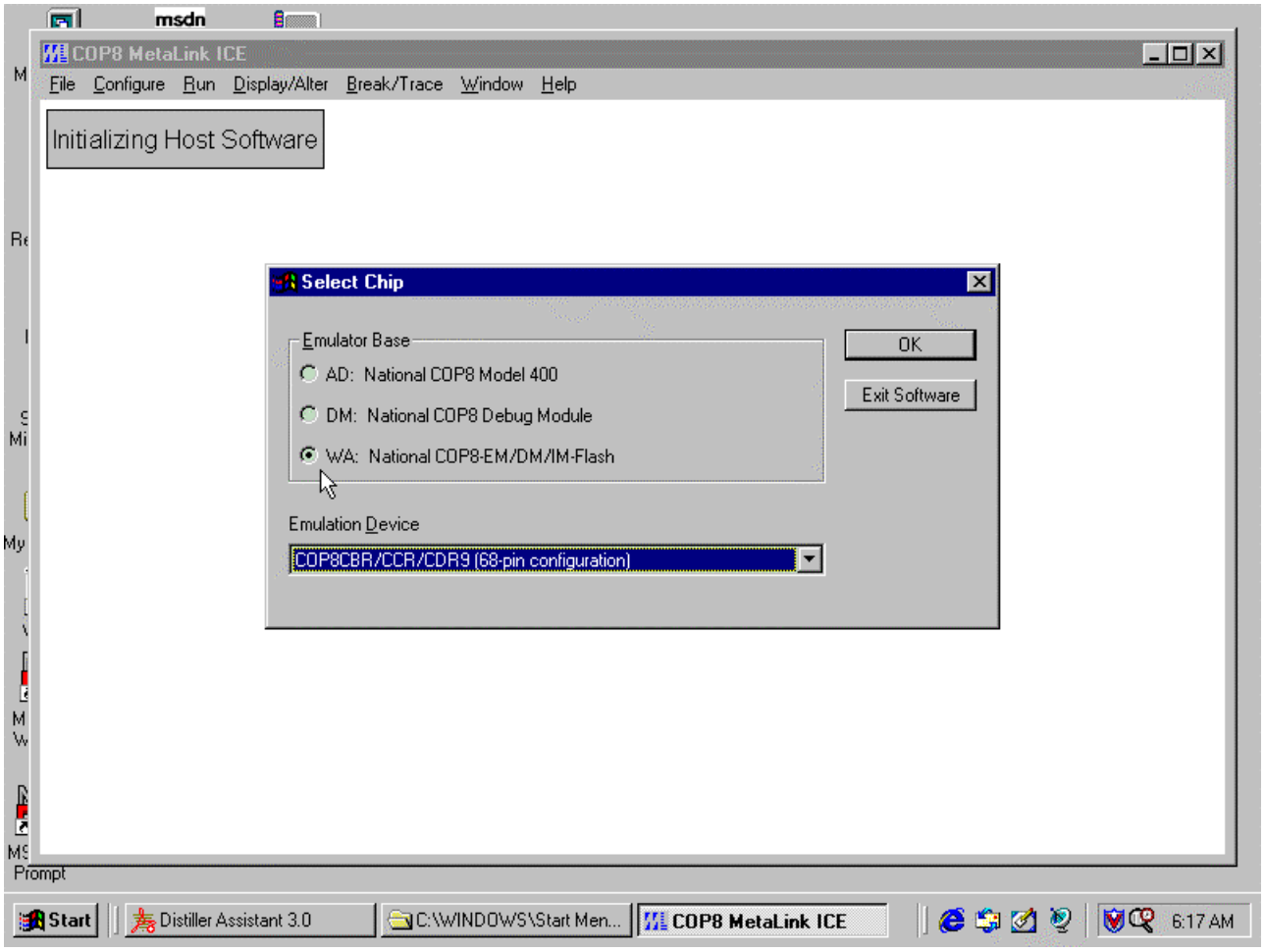
Installation and Initial Startup

The host software package that comes with the iceMASTER-WA COP8-EM/DM/IM-Flash emulator can be used with the following products:

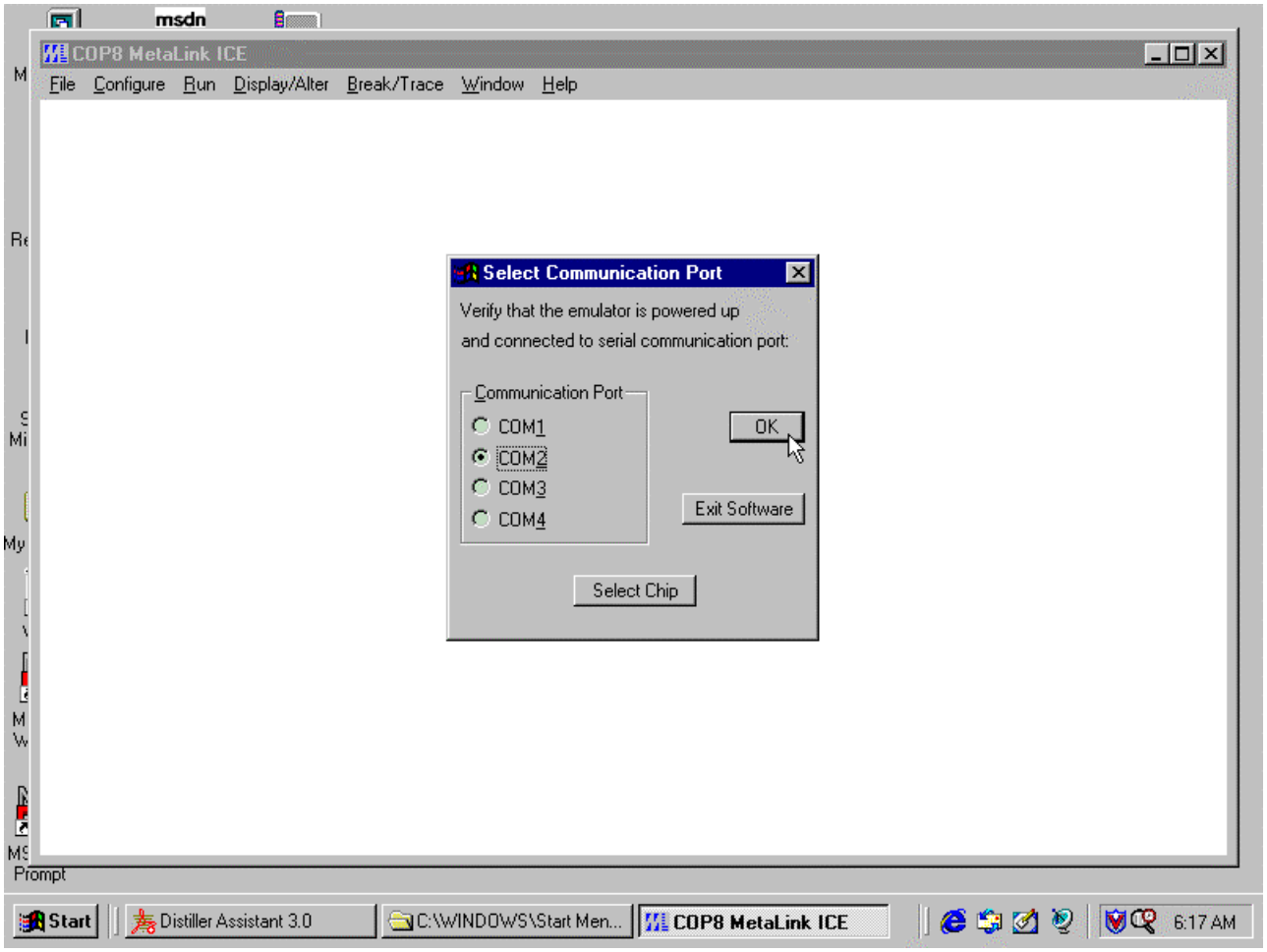
- iceMASTER-AD Model 400 Emulator
- iceMASTER-DM Debug Module
- iceMASTER-WA / COP8-EM/DM/IM-Flash products:

iceMASTER-WA Model 100	COP8-EM-Flash
iceMASTER-WA Model 400	COP8-DM-Flash
iceMASTER-WA Model 500	COP8-IM-Flash

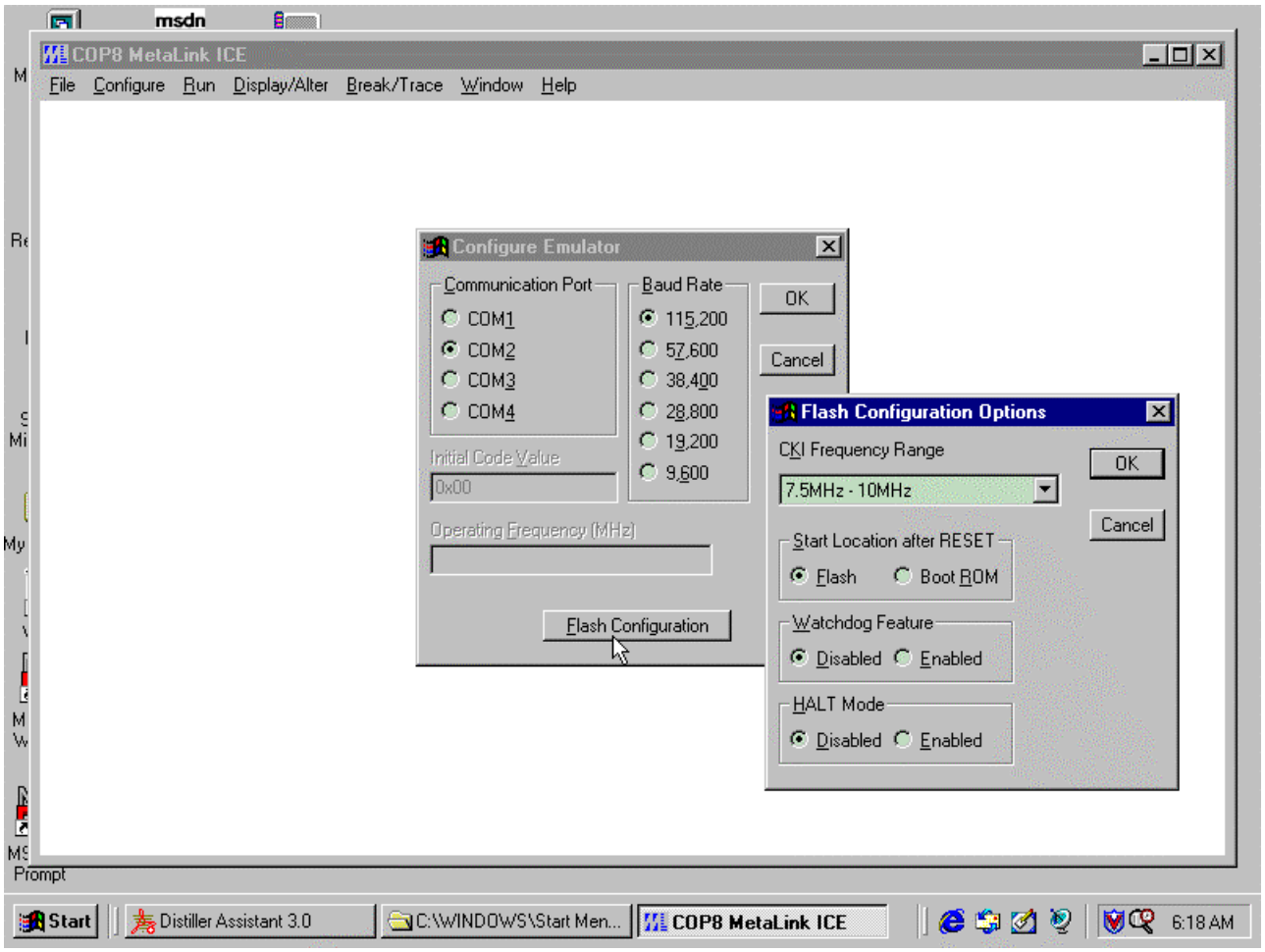
When you first start the software after installing it, you should select the emulator base type and emulation processor (device), as follows:



After that, you must select the COM*n* port on your PC that you are using for the emulator:



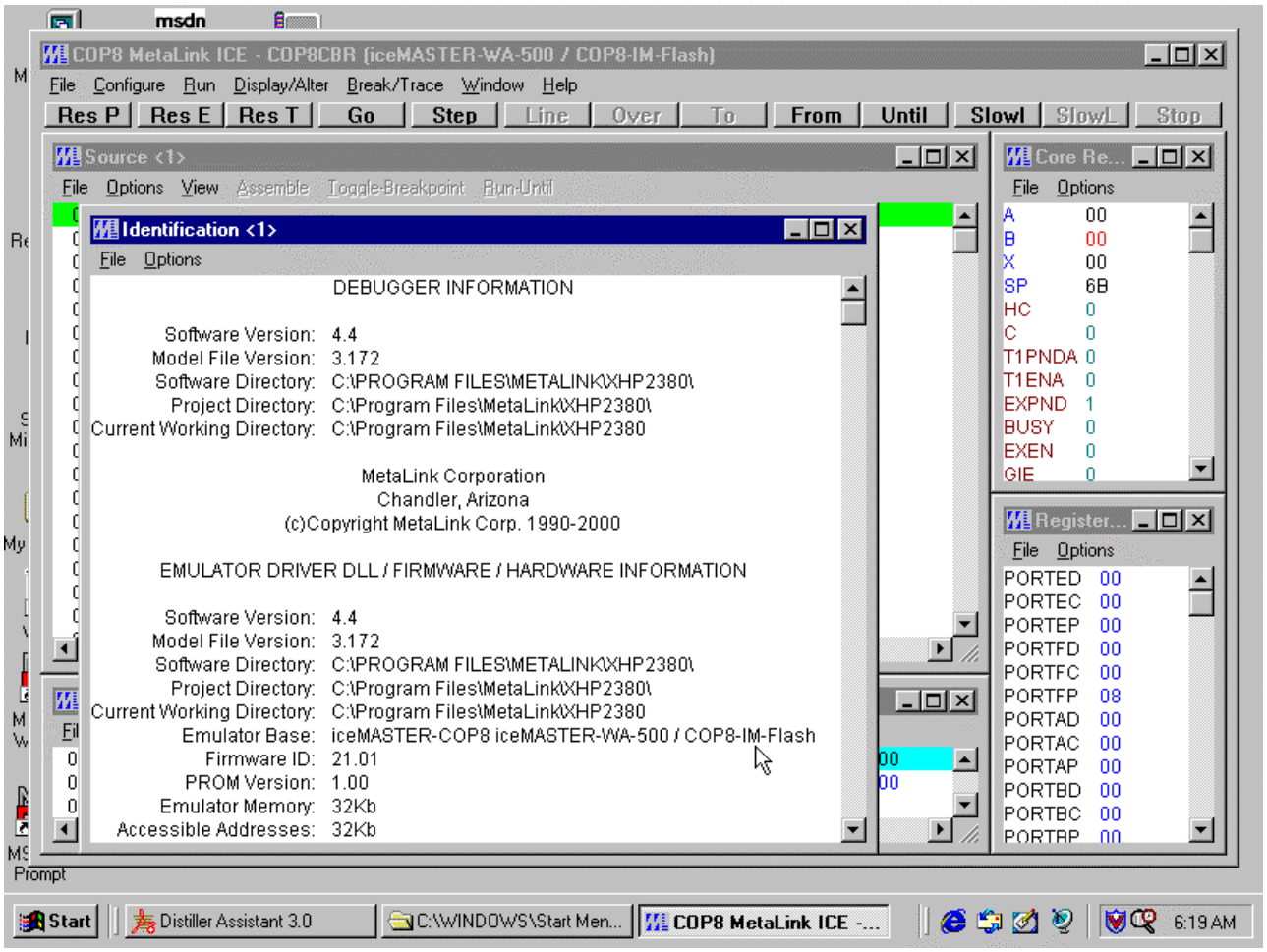
The software will then attempt to communicate with the emulator hardware to determine exactly which emulator model is being used (e.g., iceMASTER-WA Model 400 / COP8-DM-Flash). Once determined, you will be presented with the following dialog (***Configure|Emulator***), which contains any other configuration options relevant to the operation of the particular hardware (emulator model):



For the iceMASTER-WA, the following unique configuration options are present:

- **CKI Frequency Range**: Select the appropriate range from the drop-down list that corresponds to the CKI frequency that you are using in your target system hardware. The selection made here allows the software to automatically set the Write Timing Register (PGMTIM) to the correct value to control the width of the timing pulses for write and erase operations on the flash memory.
- **Start Location after RESET**: The selection made here is, if necessary, written to the OPTION register and specifies where the COP8 flash device will start executing following a RESET (***Run|Reset|Emulator*** or ***Run|Reset|Target*** commands). Normally, you would choose 'Flash' to start execution from location zero in the flash memory. If your application is designed so that you want to program the flash memory using the Microwire ISP routine or external programming, then you should select 'Boot ROM' so that the COP8 processor will start executing in the Boot ROM following a RESET.
- **Watchdog Feature**: The selection made here is, if necessary, written to the OPTION register and enables or disables the Watchdog feature in the COP8 device.
- **HALT Mode**: The selection made here is, if necessary, written to the OPTION register and enables or disables HALT mode in the COP8 device.
- The **Initial Code Value** is disabled (low-lighted) and shows the value (0x00) that the COP8 chip uses for a Flash Erase operation.

After the system initializes completely, the information describing the particular emulator and model is displayed in the ***Configure|Identification*** window, as well as in the main window's title bar:



Overview/Features

This chapter contains an overview of some of the features of the software and a guide to help you understand the layout of the screen. Note that context sensitive help is available for every command.

Windows

You can have any number of windows of any type, including multiple windows of the same type, open at the same time.

All windows are updated after an emulation cycle or after something changes (e.g., after you explicitly change the value in a register or memory location during Break Condition).

To cycle through the existing windows:

- CTRL+F6: cycle forward
- CTRL+SHIFT_F6: cycle backward

This is the Microsoft Windows convention for cycling through MDI (Multi-Document Interface) child windows.

The software implementation "model" most closely resembles an MDI application, with the following enhancements:

- The "child" windows (all windows except the Main Window) are not constrained to be within ("be clipped by") the Main Window. They can be anywhere on the screen. (For comparison: Open several documents in a word processor and move the document windows around.)
- You can have as many "child" windows as you want, of a given type, open at the same time (e.g., 5 Code Memory windows, each viewing the same/different/overlapping memory addresses).
- If you turn off the ***Configure|Preferences|Move Windows with Main*** option, the Main Window can be moved independently of any "child windows". Currently, re-sizing the Main Window is always done independently of any "child" windows.

Additionally, the Main Menu window will never be "on top of" any of the "child" windows.

You can also pick a specific existing window by selecting it from the bottom of the ***Windows*** pull-down menu.

Double Click

In general, double-click (left mouse button) on a symbol name, register name or memory-location to pop-up the ***Display/Alter|Expression*** dialog box, which will allow you to view, change or browse the value.

NOTE: Currently, right-clicking in a Browse Window (browse data structure) does pop-up a properties menu showing everything you can do to the selected item (e.g., change (modify), or browse further (if it is a pointer, structure or array).

Save Settings On Exit

Currently, the ***Configure|Preferences|Save Settings on Exit*** function saves the preferences, the name of the previously loaded program file and the position, size and other information for each window.

These settings can optionally be restored (re-established) whenever you restart the software based on the ***Configure|Preferences|Restore Settings at Startup*** and ***Configure|Preferences|Reload Code File at Startup*** settings.

A detailed list of exactly what settings are saved is displayed in the on-line help for the ***Configure|Preferences|Save Settings on Exit*** command.

Host Break

There are several ways to stop emulation if no breakpoint is set or reached during emulation:

- Click on the ***Stop*** toolbar button.
- Select the ***Stop(Esc)*** command in the Emulating window.
- Select the ***Run|Stop*** command from the Main Menu window.
- Press the ***Esc*** key.

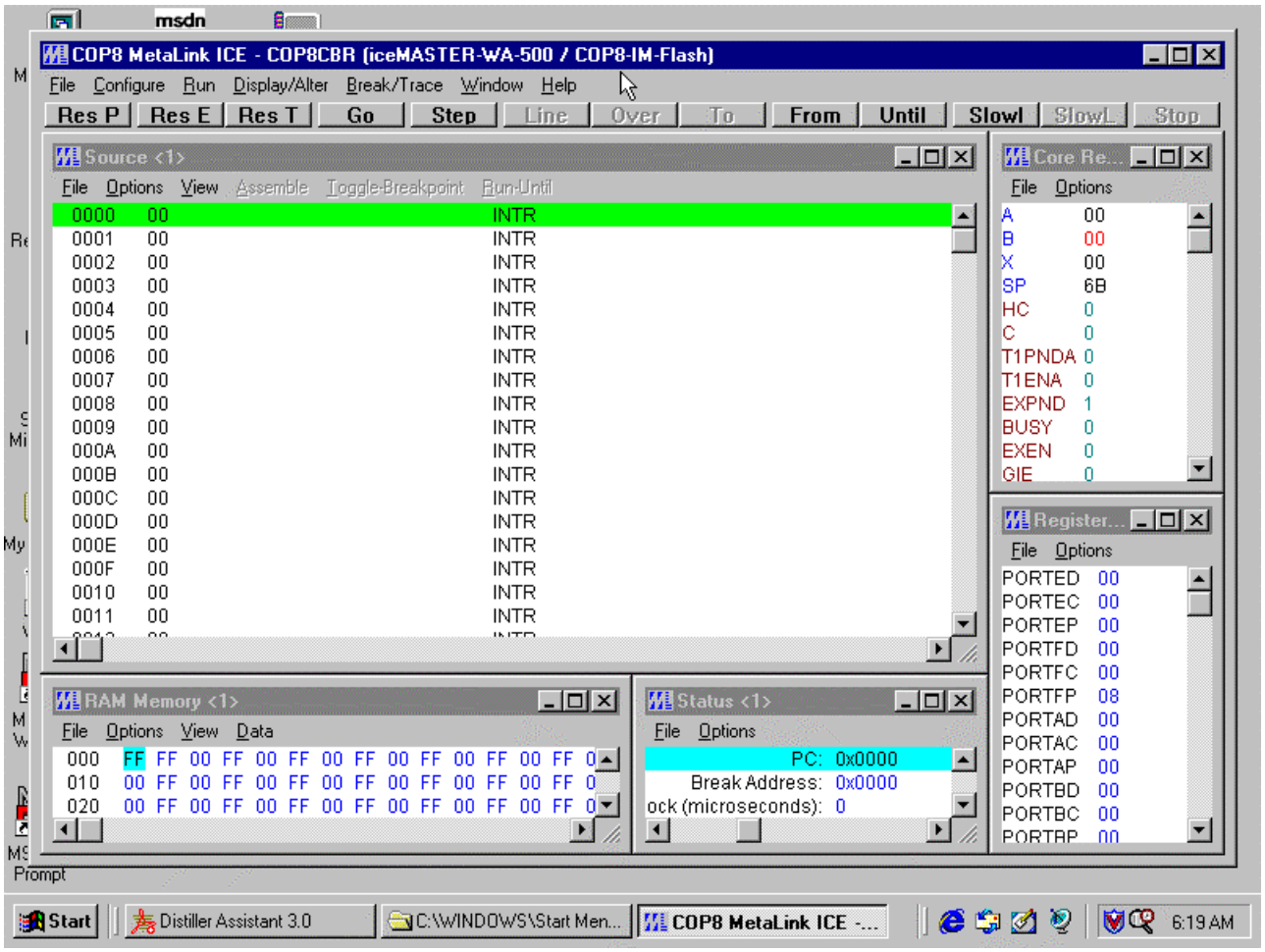
Help

Detailed and context sensitive Help is available on-line using the Help Key (F1).

Dynamically Annotated Code

When emulating using single steps (***Step*** command) or slow motion mode (***Slow Motion*** command), the right side of the Source Window is used to display a history of execution for each instruction executed. This history contains the value (before execution of the instruction) of any address, register and bit used by the instruction. In addition, if the instruction is an unconditional jump instruction or a conditional jump, where the condition is met (TRUE), the Target Address and an arrow indicating direction of program flow will be displayed. If the instruction is a conditional jump, where the condition is not met (FALSE), a * is displayed.

Default Screen Layout



The picture above shows the default screen layout. The windows shown (Source, RAM Memory, Core Registers, Registers (SFR), and Status) are just a subset of the available windows. The available windows are described below.

Available Windows

Break

The Break Window is used to display, add, remove or edit breakpoints. Breakpoints are evaluated and transmitted to the emulator as they are created or edited.

Browse

The Browse Window is used to inspect or change structures, unions, arrays, pointers and bit-fields. For each element of the object being browsed, at least the address, data type and value are displayed and where meaningful, array subscript values and member names are also displayed.

Code Memory

The Code Memory Window displays memory as hexadecimal bytes and their ASCII equivalent. In addition, from the Code Memory Window you may fill or copy blocks of memory, compare two blocks of memory and search memory for a value (or values) match/mismatch.

Core Register

The Core Register Window is used to display the core architecture registers and bit flags.

Emulating

The Emulating Window is displayed when an emulation is started using any run-type commands. It displays status information about the current emulation cycle.

Identification

The Identification Window describes the properties of your emulator system. The information displayed includes such things as hardware, firmware and software versions numbers, memory sizes, model numbers and option configurations. Such information is useful, for example, when you call for technical support.

Program Structure

The Program Structure Window displays module, source line number or source scope information for the loaded program, if available.

RAM Memory

The RAM Memory Window displays memory as hexadecimal bytes and their ASCII equivalent. In addition, from the RAM Memory Window you may fill or copy blocks of memory, compare two blocks of memory and search memory for a value (or values) match/mismatch.

Register (SFR)

The Register (SFR) Window is used to display the special function registers (SFRs).

Source

The Source Window is used to display code memory as assembly level instructions, optionally with HLL source images, if available.

Status

The Status Window displays status information such as the PC address, break address, real-time clock, inactive count, pass count, repetition count, emulation status, trace status, trace read percentage and trace trigger type.

Symbols

The Symbols Window displays symbolic information for the loaded program, if available. Several display formats are available.

Trace

The Trace Window is used to display a trace of the most recent emulation. Several display formats are available. Note that the Trace Window is available only on those emulator systems that have trace memory.

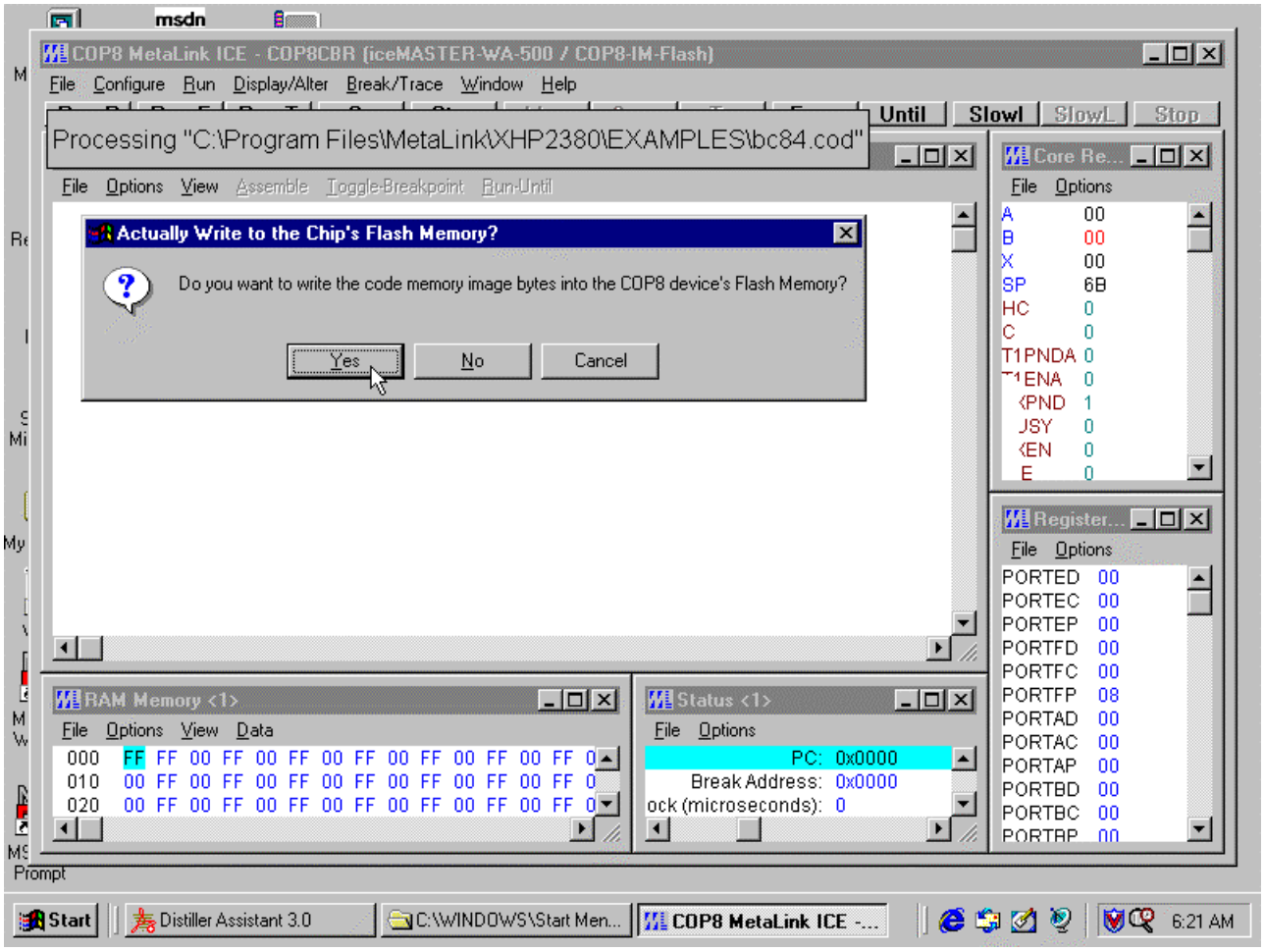
Watch

The Watch Window is used to display information about watch expressions. You can think of watch expressions as peepholes into memory where you specify the starting address symbolically (the name of a program variable, register or bit) or numerically (an expression) and where the displayed values are interpreted according to the data type of the expression (if available).

Programming Flash Memory

There is no explicit “Program the Flash” command. The flash memory in the COP8 chip is automatically programmed when you load a file (***File|Load***), or explicitly change a memory location (e.g., ***Code Memory|Data|Fill***), or patch or assemble into code memory.

When you use the ***File|Load*** command, the software reads the debugging information, if any, in the file and builds the internal symbol table and other debugging information (e.g., source line number table, etc.). Then, just before the software attempts to actually write the code memory initialization bytes from the file into the flash memory in the chip, you will be asked:



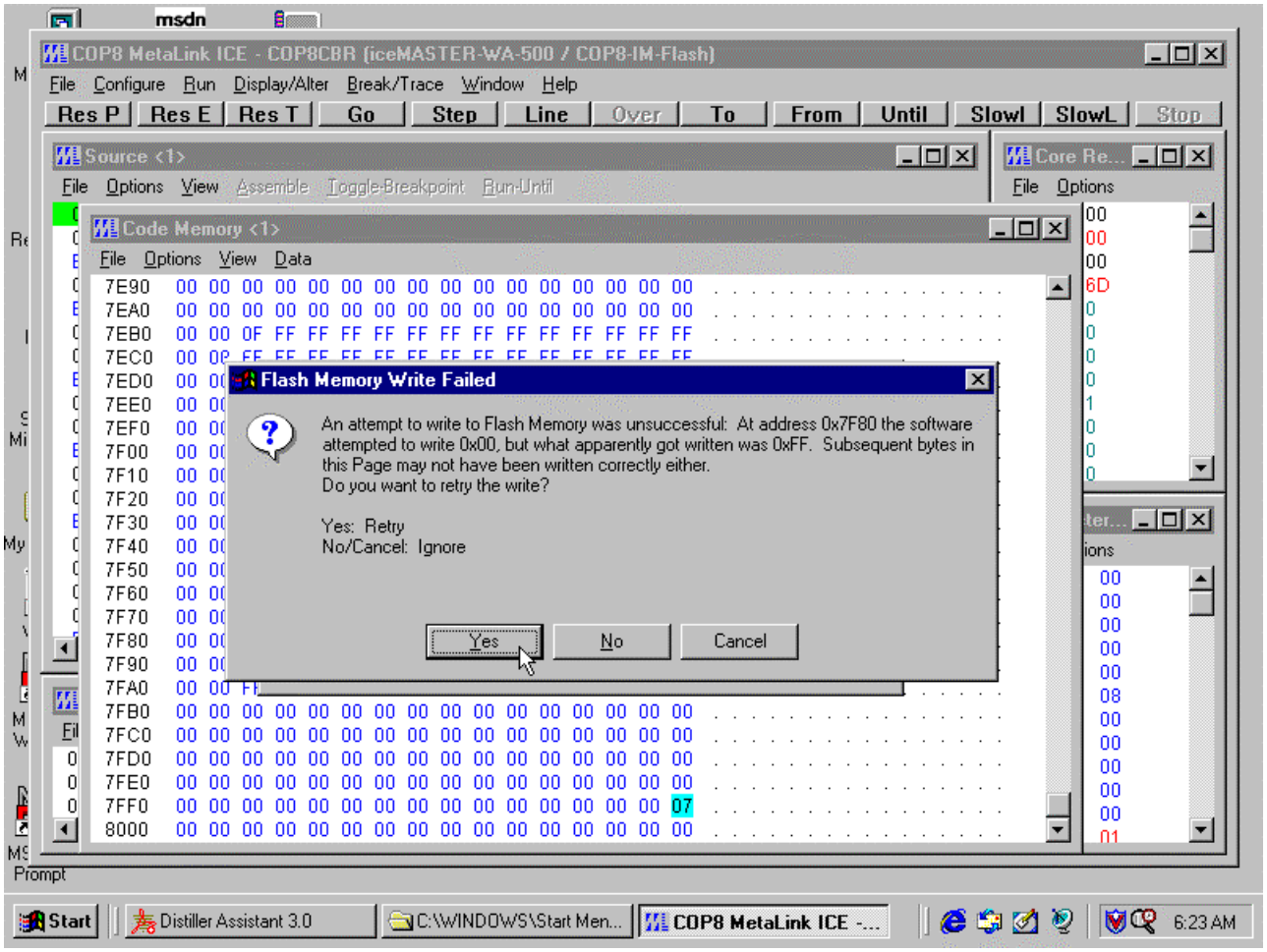
In this way, you can load all the debugging information from the program file, but avoid actually writing to flash memory if you are sure that the contents of the flash memory in the COP8 device already matches the contents of the file.

> Actually, this “manual” intervention is somewhat unnecessary because the software employs the following heuristic optimization at the lowest level to maximize the useful life (endurance) of the flash memory in the part: Before writing a Page of flash memory (128 bytes in the COP8CBR), the software first reads the current contents of that Page in flash memory. If the current contents of the

Page of flash memory about to be written exactly match the “new” contents for that Page, the “write-to-flash” operation is not actually performed.

NOTE: The above confirmation is not requested when you load a file using a Command File (script) in a Command Window. In that situation, the initialization bytes in the file will be written unconditionally into the flash memory in the COP8 device.

After the debugger software performs any “write-to-flash” operation, it reads the Page of flash memory just written, to verify that the Page was written correctly. If any byte mismatches, the following message will be displayed:



The message indicates the starting address of the first mismatch, and shows what was written and what was read back. There could be other mismatches in the Page. At this point, you can either retry the “write to flash” operation or ignore the error. In any case, this indicates that something is wrong.

There is one exception to the above verification. If

- The Page of flash memory just written is the page containing the OPTION register, and
- The value written to the OPTION register has the Security (SEC) bit set (enabled),

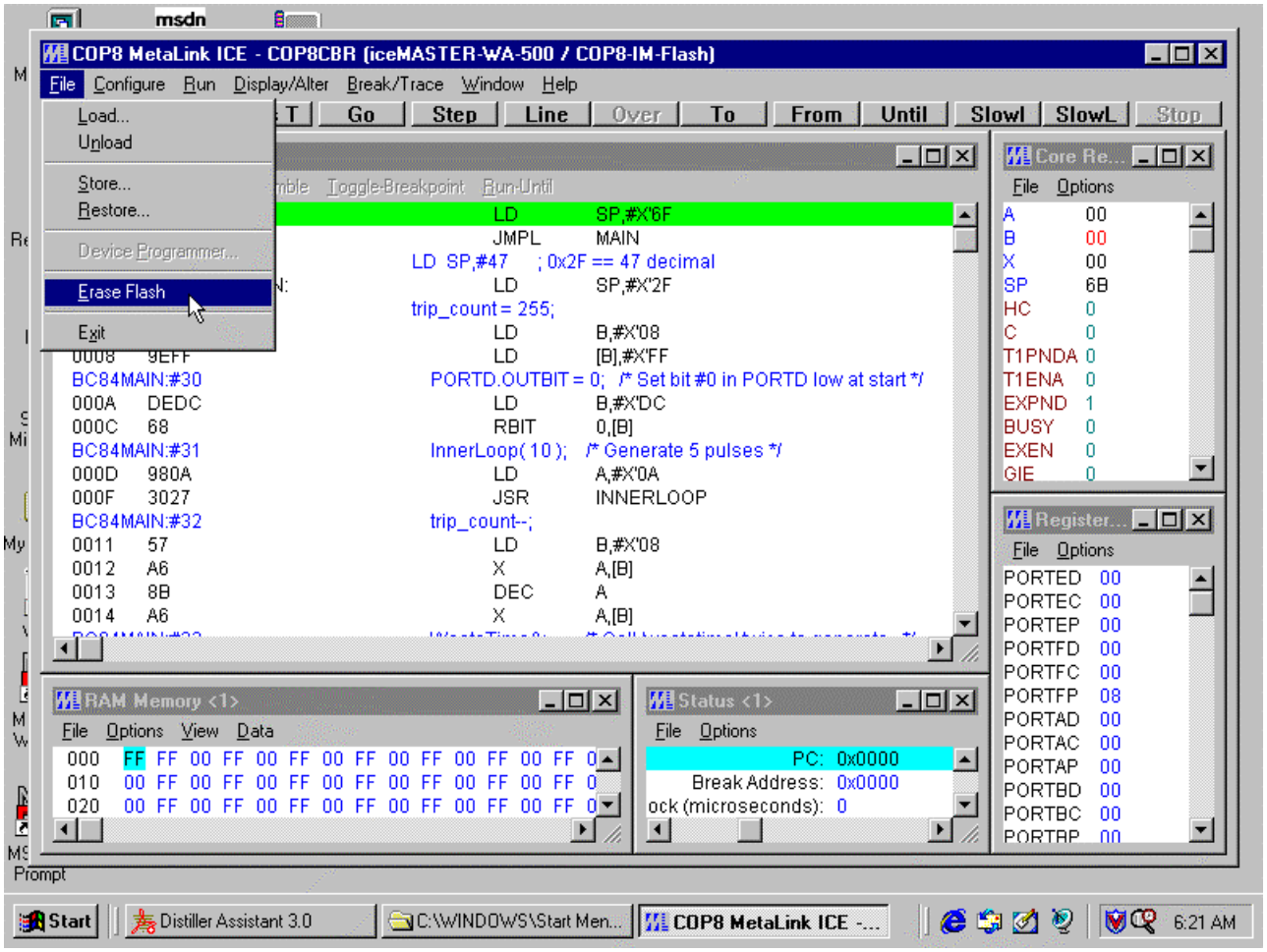
then the software verifies that all the values in that Page now read as 0xFF (i.e., Security enabled). In the COP8CBR, the OPTION register is at address 0x7FFF and the Page containing the OPTION register is the Page at 0x7F80-0x7FFF. (The above screen snapshot was taken before the software was changed to handle the Page containing the OPTION register specially.)

Mass Erasing Flash Memory

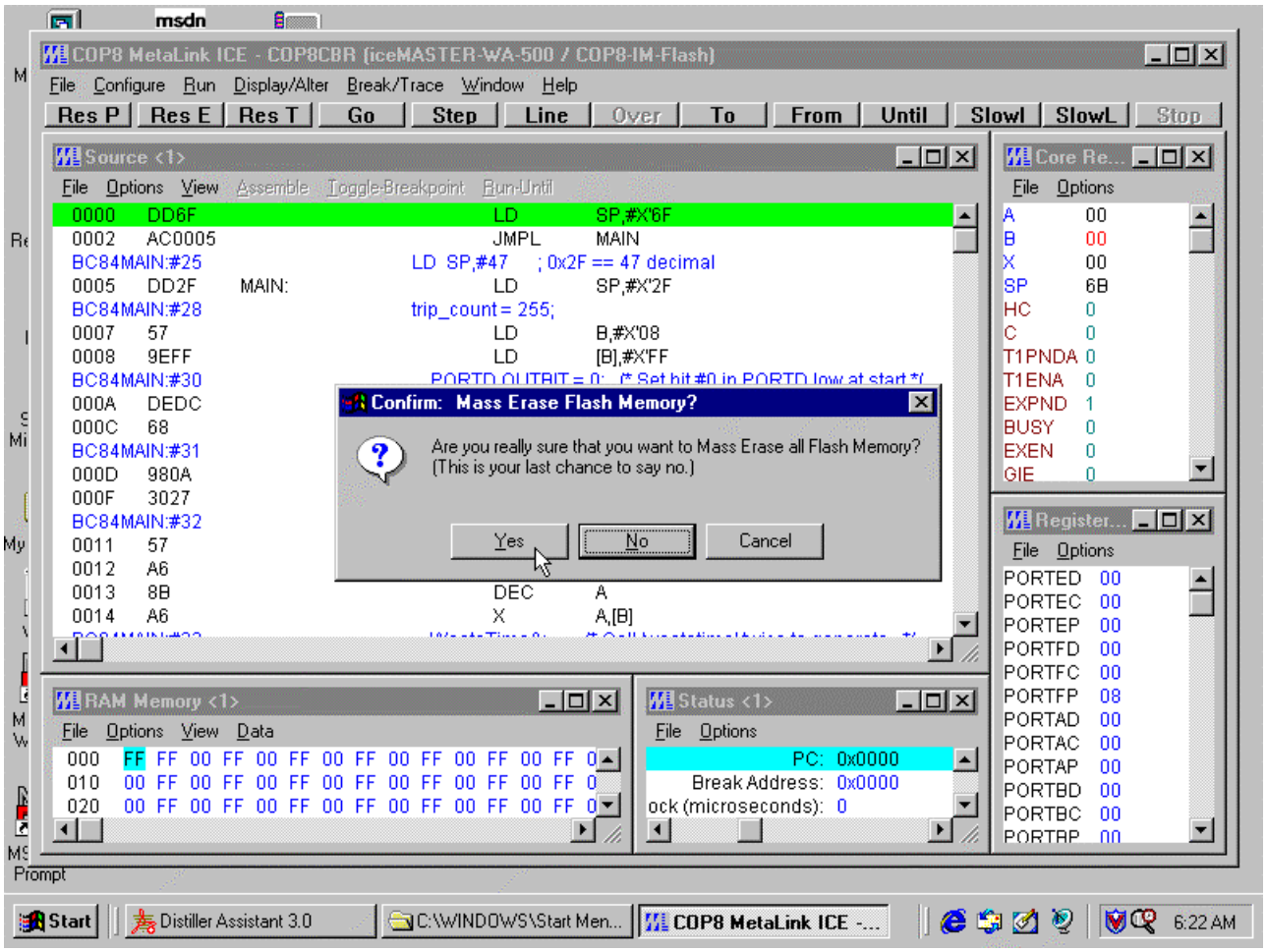
There are only two cases where the software automatically Mass Erases the flash memory in the COP8 device:

- The ***File|Load*** command: If you respond 'No' to the "Merge into current application environment?" prompt, then flash memory will be mass erased and the OPTION register will be restored to its state prior to mass erasing, before the new program file is loaded.
- The ***File|Unload*** command mass erases flash memory and then restores the OPTION register to its value prior to the mass erase operation.

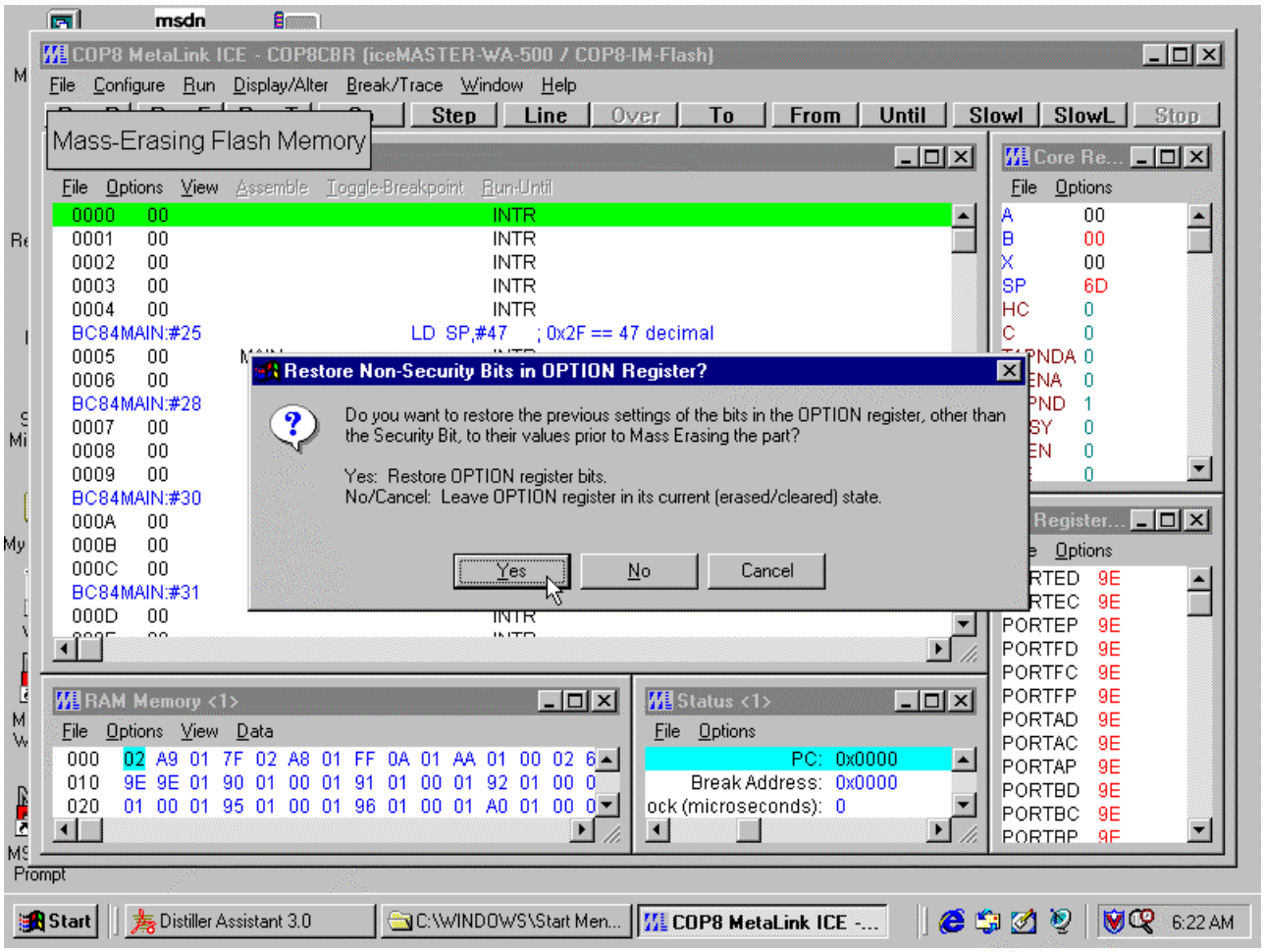
Additionally, the ***File|Erase Flash*** command is provided so that you can explicitly Mass Erase the flash memory in the COP8 part whenever you want:



Because this command clears your program from flash memory (i.e., all bytes in the flash code memory in the COP8 device are set to a value of 0x00), you must confirm the command selection before the flash memory will actually be Mass Erased:



If you do choose to Mass Erase flash memory, the debugger will remember the value in the OPTION register before erasing. Then, after Mass Erasing and if appropriate, the software will prompt you to see if you want to restore all bits in the OPTION register (except the Security (SEC) bit, which will remain off):



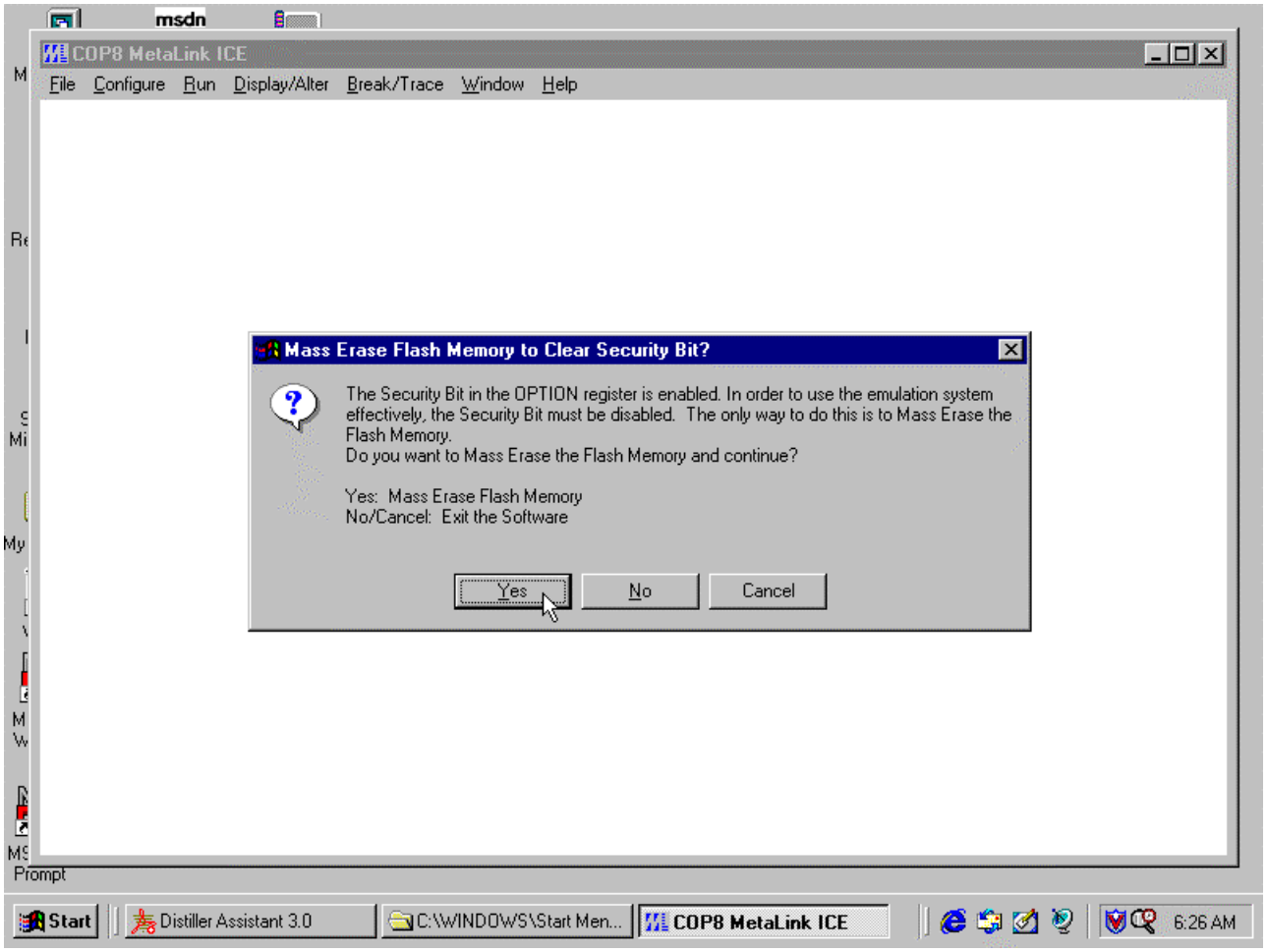
In this way, following a Mass Erase, you can easily restore your settings for:

- WD – Watchdog feature enabled/disabled.
- HALT – Halt mode enabled/disabled.
- FLEX – Start executing in flash or Boot ROM following a RESET.

Flash Memory Security

When the Security bit (SEC) in the OPTION register is set, security in the COP8 device is enabled. In such a case, the debugger software can neither read/write flash memory, nor can it update the FLEX bit in the OPTION register to reflect your ***Configure|Flash...*** choices.

If the Security bit is set (enabled), the software will prompt you as follows:



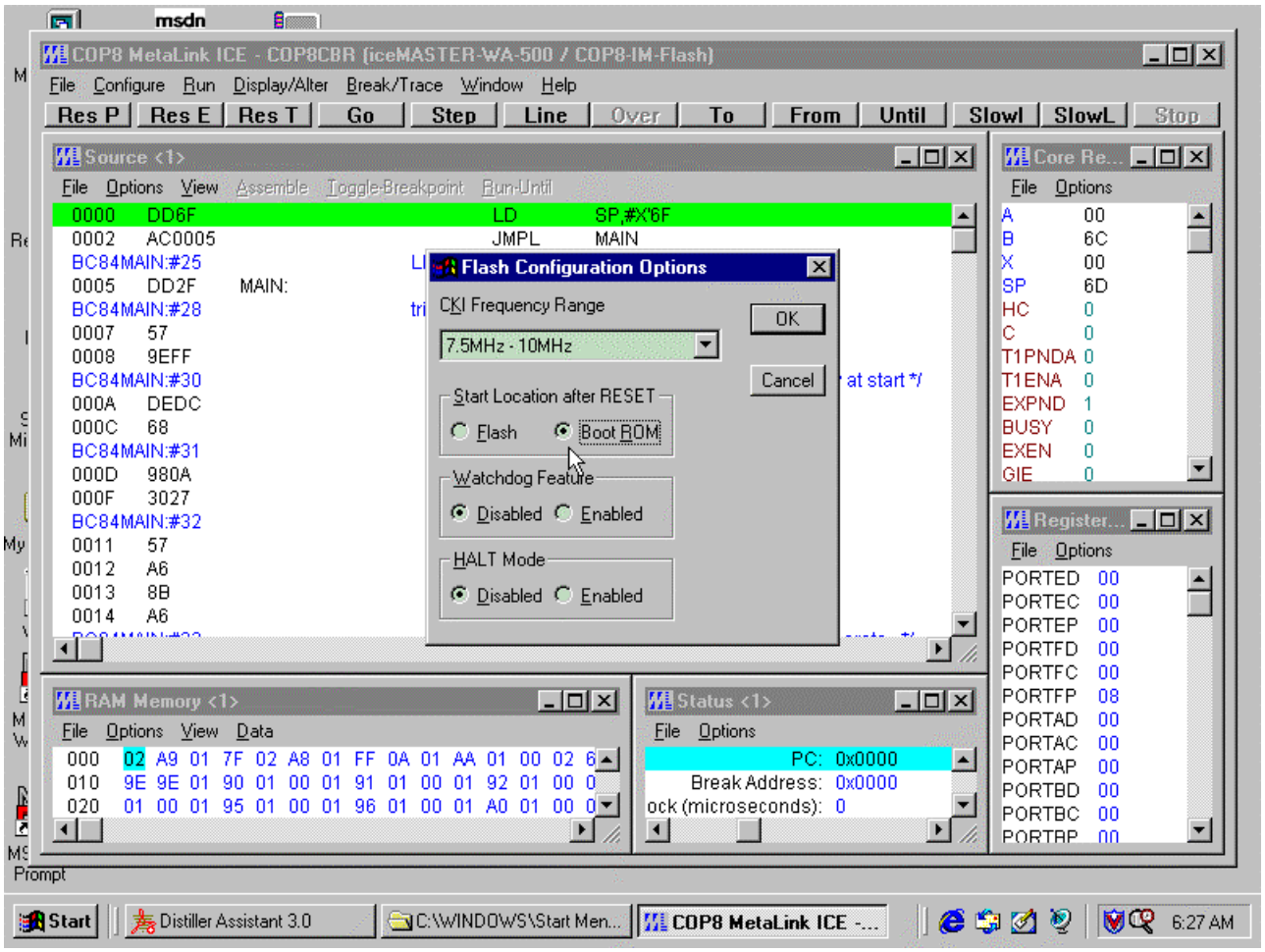
Again, because this command clears your program from flash memory (i.e., all bytes in the flash code memory in the COP8 device are set to a value of 0x00), you must confirm the command selection before the flash memory will actually be Mass Erased:

The debugger performs the above check (to determine if Security is enabled) at the following times:

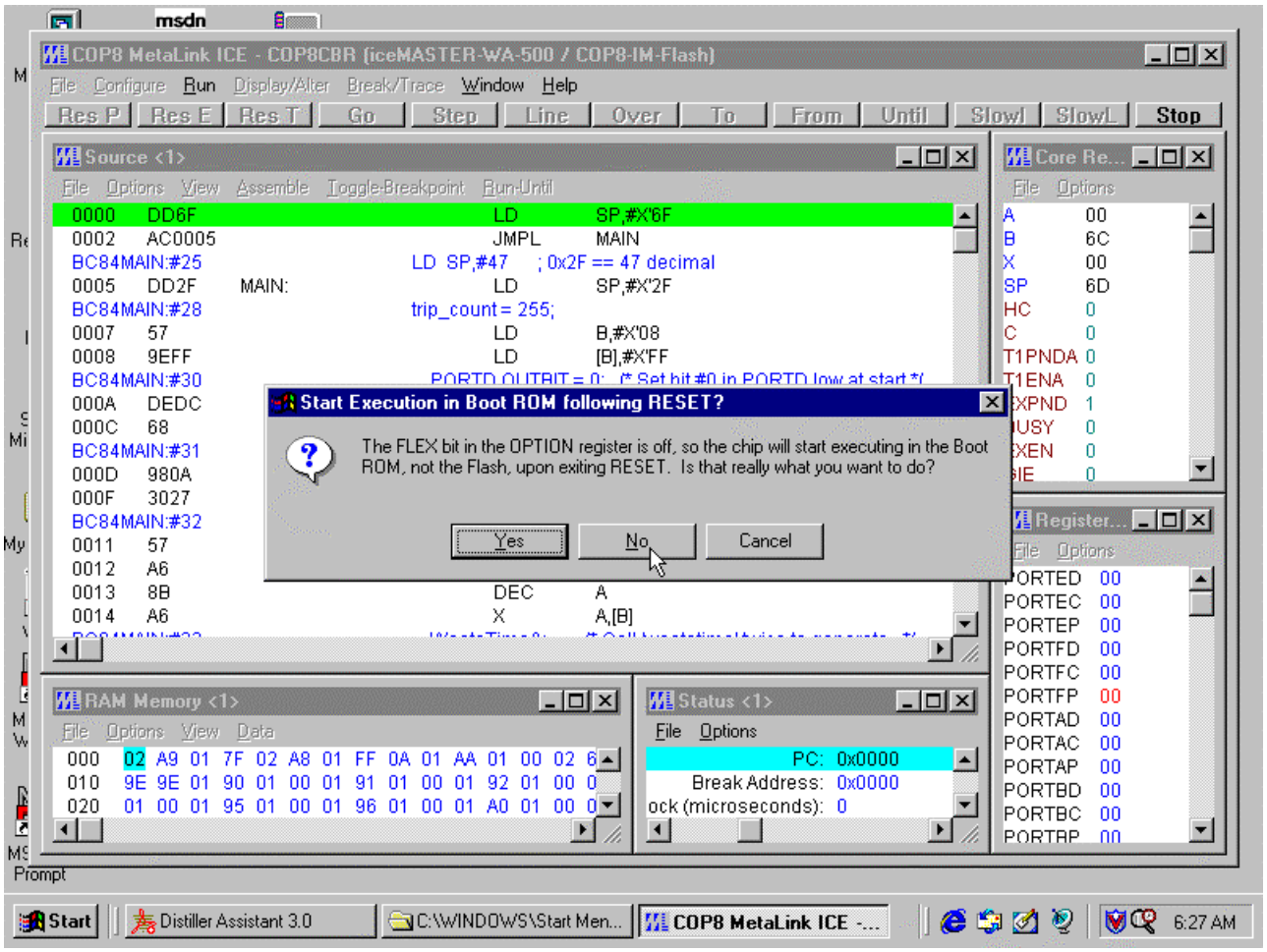
- When you first invoke the debugger software and it initially establishes communication with the emulator.
- When you select the ***Configure|Emulator...|OK*** command button.
- When the debugger attempts to write to any location in the COP8 code flash memory (for any reason).

FLEX Bit in the OPTION Register

The FLEX bit in the OPTION register determines whether the COP8 device will start executing in the flash memory or in the Boot ROM upon exiting reset. This bit is set to reflect the choice you make in the ***Configure|Flash...*** dialog:



If the part is configured to start executing in the Boot ROM upon exiting RESET, when you select either the ***Run|Reset|Emulator*** or the ***Run|Reset|Target*** command, you will be prompted to confirm the command selection:

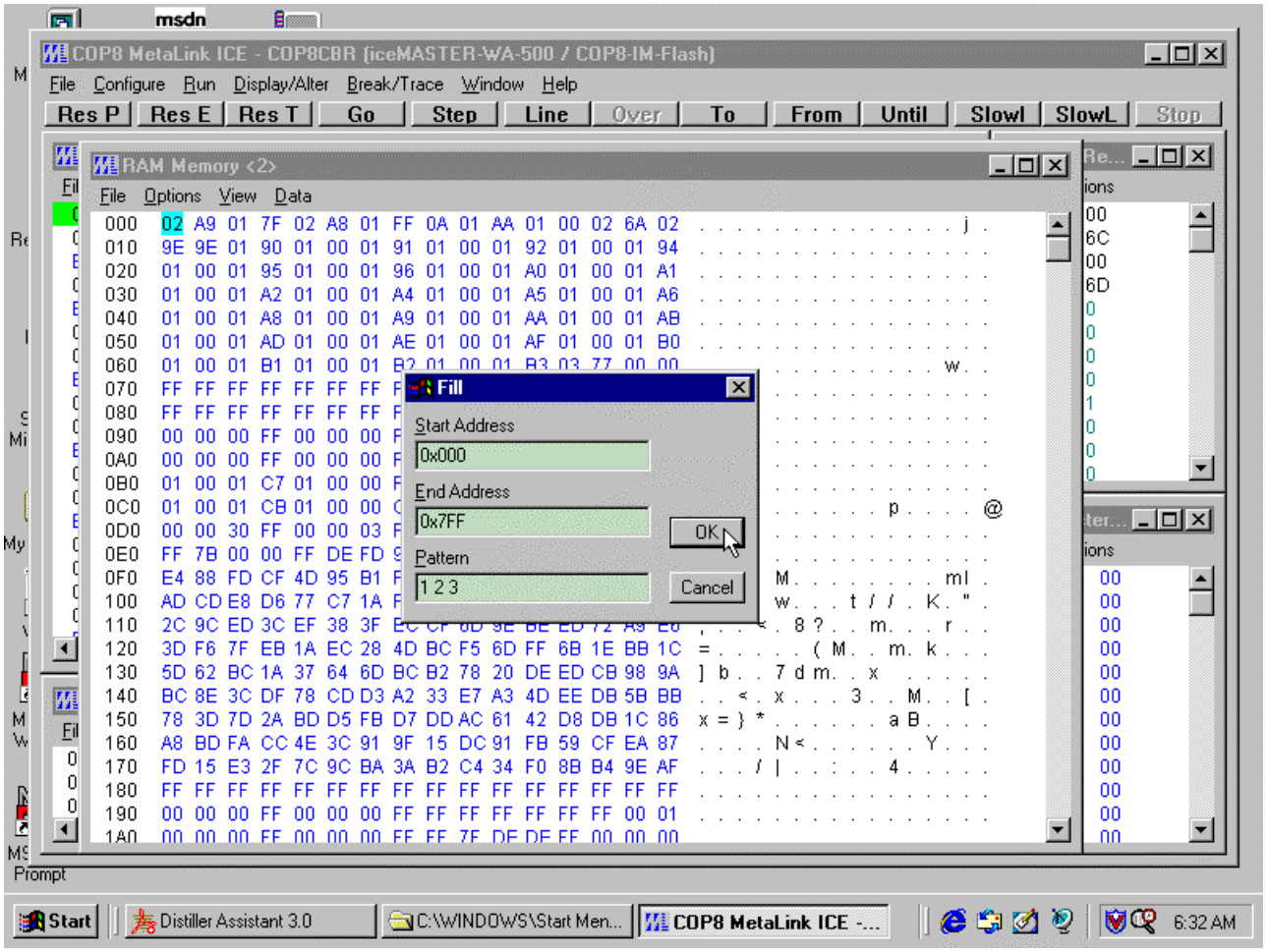


This confirmation is requested because it can be confusing if the part starts executing in the Boot ROM and that is not what you are expecting.

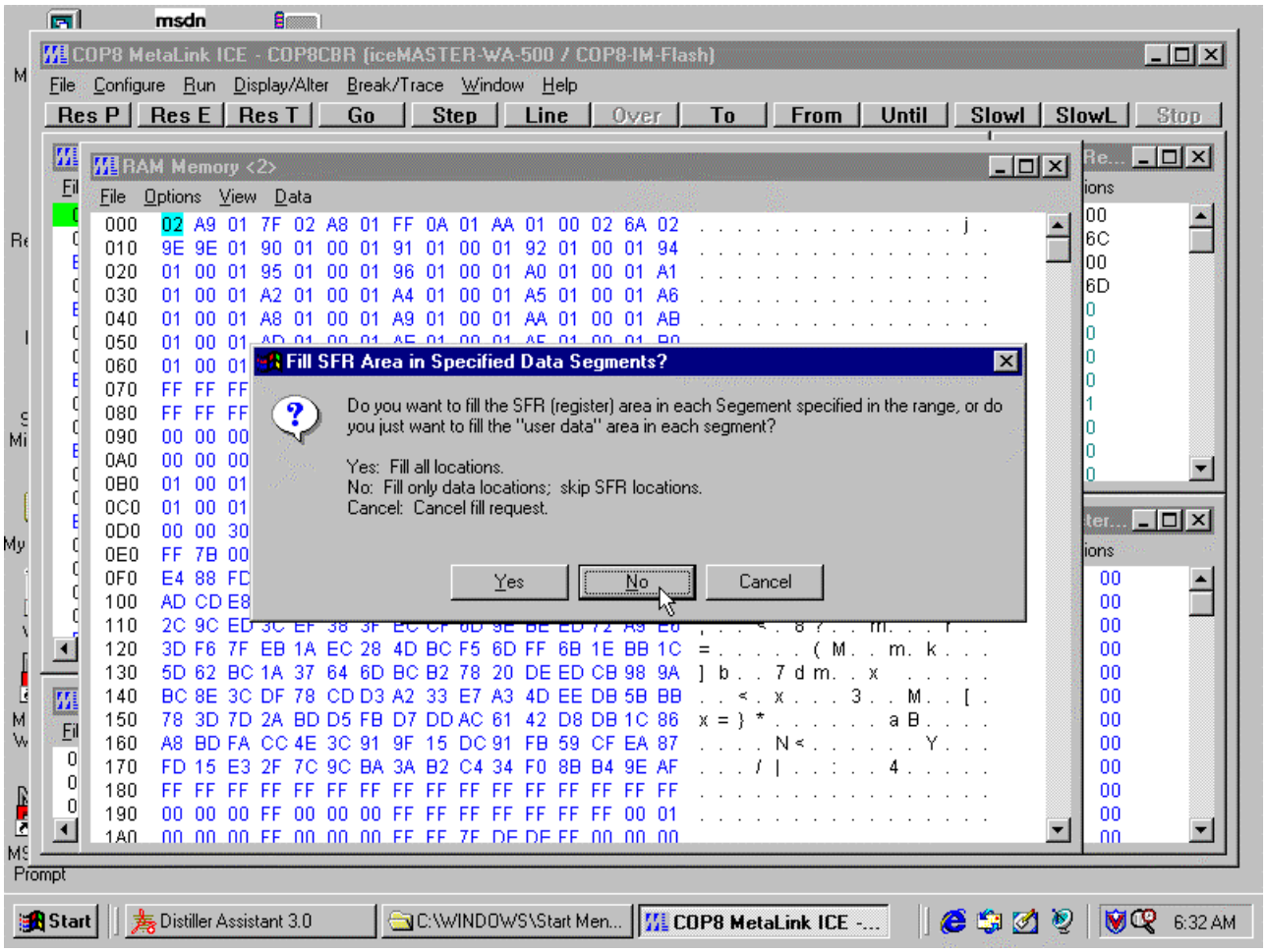
In the Raw Mode ***Window|Trace*** display, the data byte for a Boot ROM access will display as two question marks (“??”).

RAM Memory|Data|Fill Command

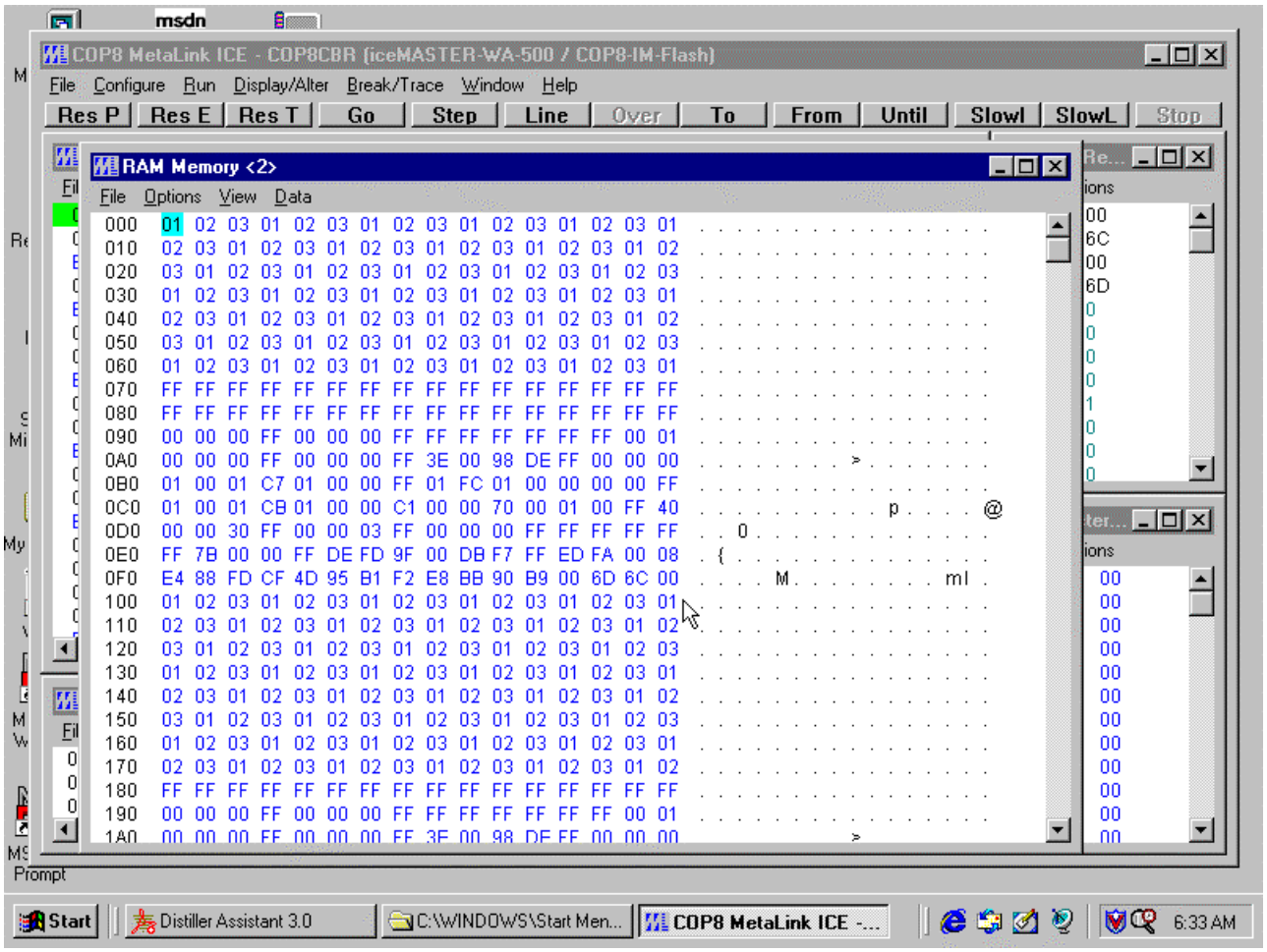
If you specify an address range in the ***Fill*** dialog that spans the SFR (Register, I/O) area in one or more data segments:



The software now gives you the option of filling only the “true” data locations in each segment (lower end) and skipping the SFR (register) portion of each segment (upper end):

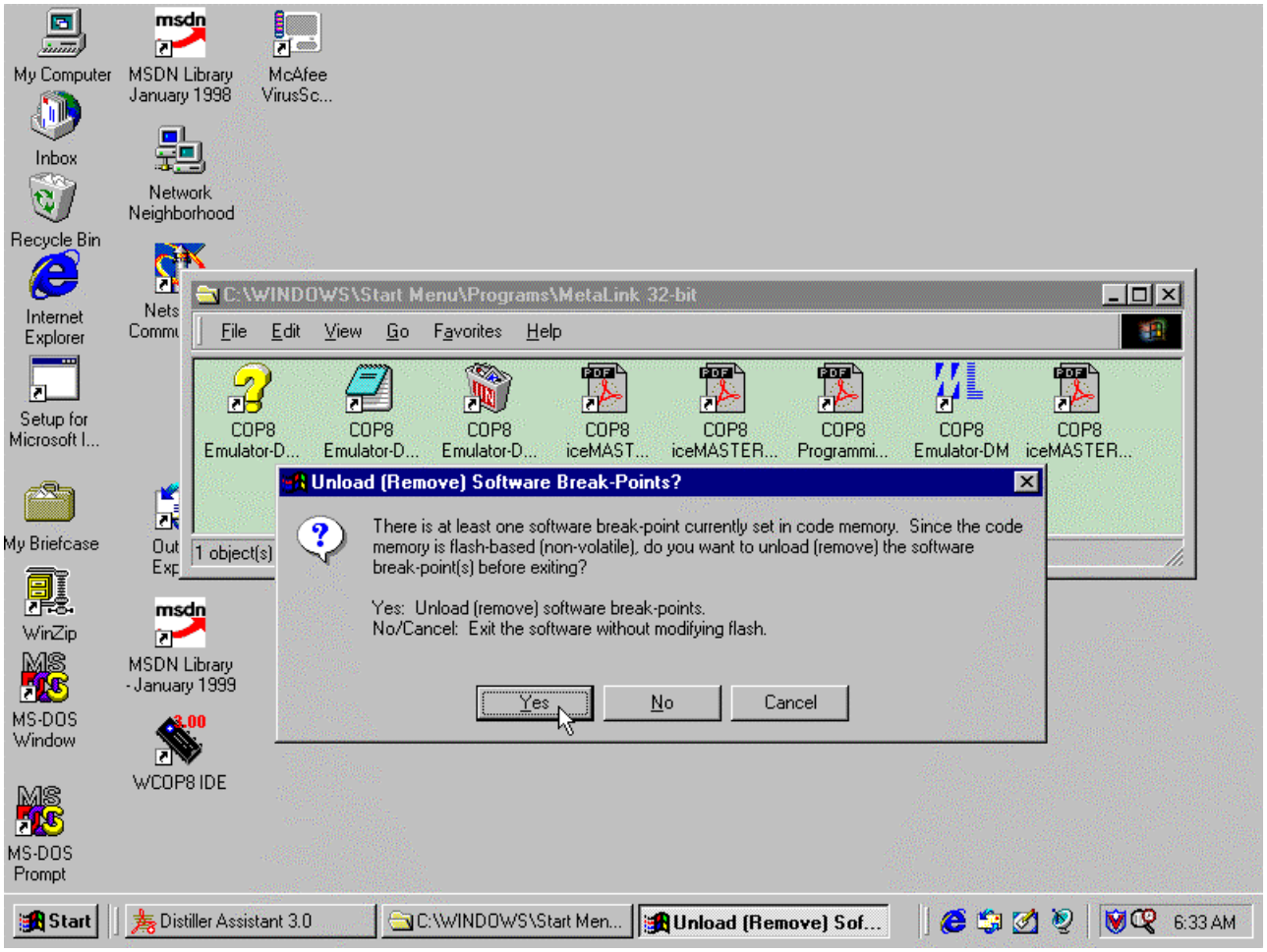


The result of pressing ‘No’ above looks something like the following:



Exiting the Debugger Software

When you exit the debugger, there may be software breakpoints set in the flash memory of the COP8 flash device. If that is the case, you will be prompted as follows:



In general, it is usually a good idea to respond 'Yes' to this prompt (unload the software break-points). In this way, the flash memory in the COP8 flash device will contain exactly the code memory image bytes that you last loaded there.

If you respond 'No' or 'Cancel' and leave the break-points (**BRK** instructions) set/present in the COP8 device's flash memory, then, when you next restart the debugger software, you may see the following diagnostic:

> **Unsolicited breakpoint encountered at 0x*hhhh*.**
> **(The program contains a non-breakpoint BRK instruction.)**

under the following conditions:

1. Restart (invoke) the debugger software.
2. Don't reload/rewrite the flash memory during initialization.
3. Begin emulation.

The reason is that the **BRK** instructions that the debugger set ("planted") in the COP8 flash memory during your previous debugging session are still present in flash memory during this new debugging session (since flash memory is

"persistent" (non-volatile)). However, the instance of the debugger that you are running in this new debugging session did not set ("plant") those breakpoints. Also, see the Troubleshooting chapter.

Run|Go *From a Break-Point*

The implementation of the ***Run|Go*** command (***Go*** button) is different depending on whether or not there is a breakpoint set at the current PC value.

1. If there is no break-point set at the current PC, the ***Run|Go*** command starts a real-time emulation cycle that continues until some break-point is reached or until you select the ***Run|Stop*** command (***Stop*** button).
2. If there is a break-point set at the current PC, then two emulation cycles are performed. The detailed sequence is as follows:

    a. The software break-point (a 'BRK' instruction) at the current PC is removed and it is replaced by the original ("real") instruction.

    b. A machine-instruction single-step emulation cycle is performed.

    c. After single-stepping, the software break-point removed in 2a. above is reinserted into the code.

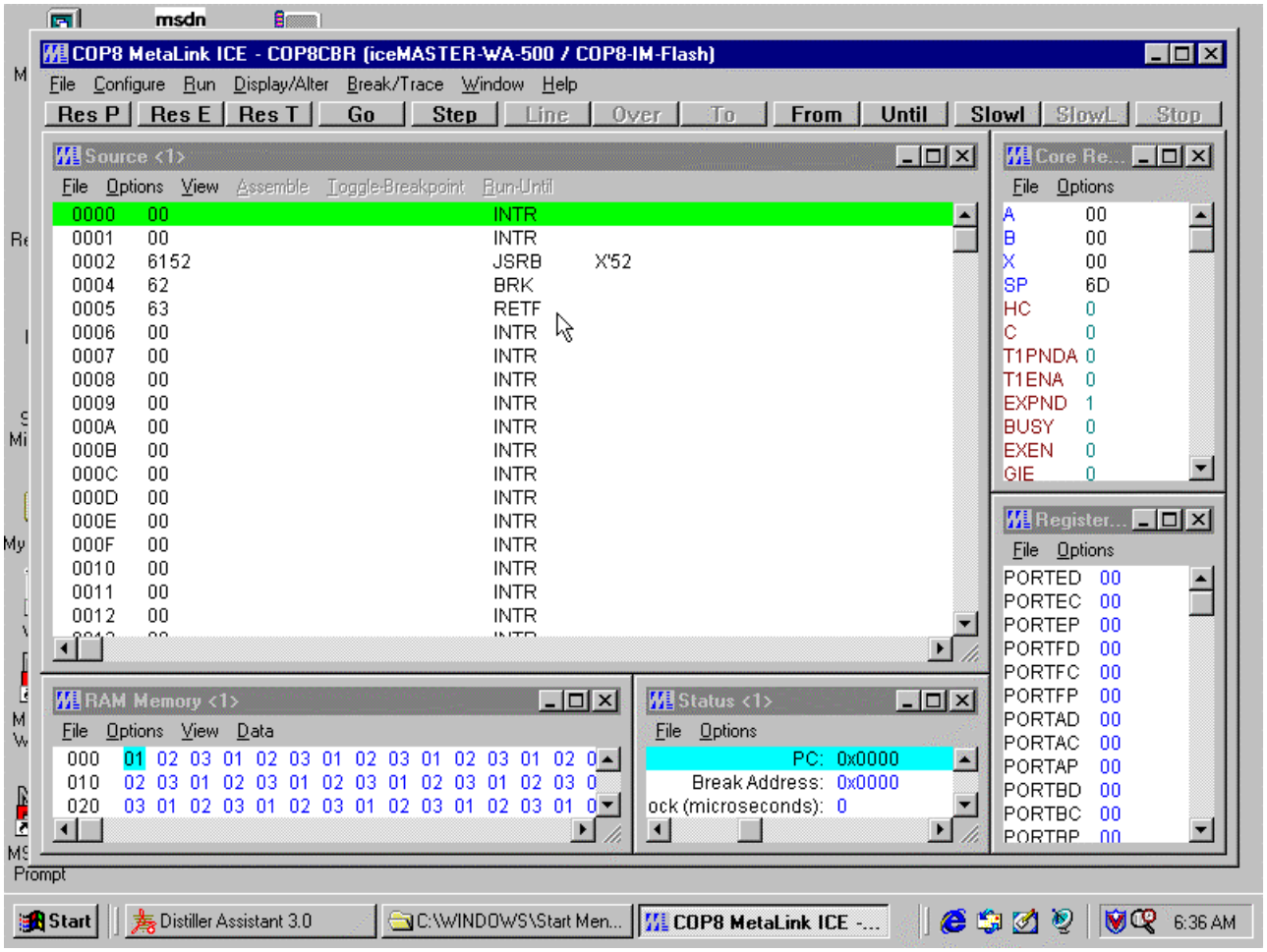
    d. A "Go" emulation cycle (***Run|Go*** command) is then performed (starting at the new PC). Emulation continues until some break-point is reached or until you select the ***Run|Stop*** command (***Stop*** button).

There are several implications of this process:

- The process, as a whole, implemented in 2. above is not real-time, although the "Go" in Step 2d. is real-time.
- The Trace Window will not contain the trace of the single-step of the first instruction in 2a. above, because two emulation cycles were actually performed by the hardware (a "Single-Step" and a "Go").

New Instructions

The debugger's Disassembler supports the 3 new instructions in the COP8 flash devices (**BRK**, **JSRB** and **RETF**):



However, the Single-Line Assembler does not support them.

If you do "patch in" a **BRK** instruction (by changing the value in a code memory location to 0x62), the emulator will break (stop emulating) when the COP8 processor executes the **BRK** instruction at that location. However, in such a case, the software will issue the following diagnostic:

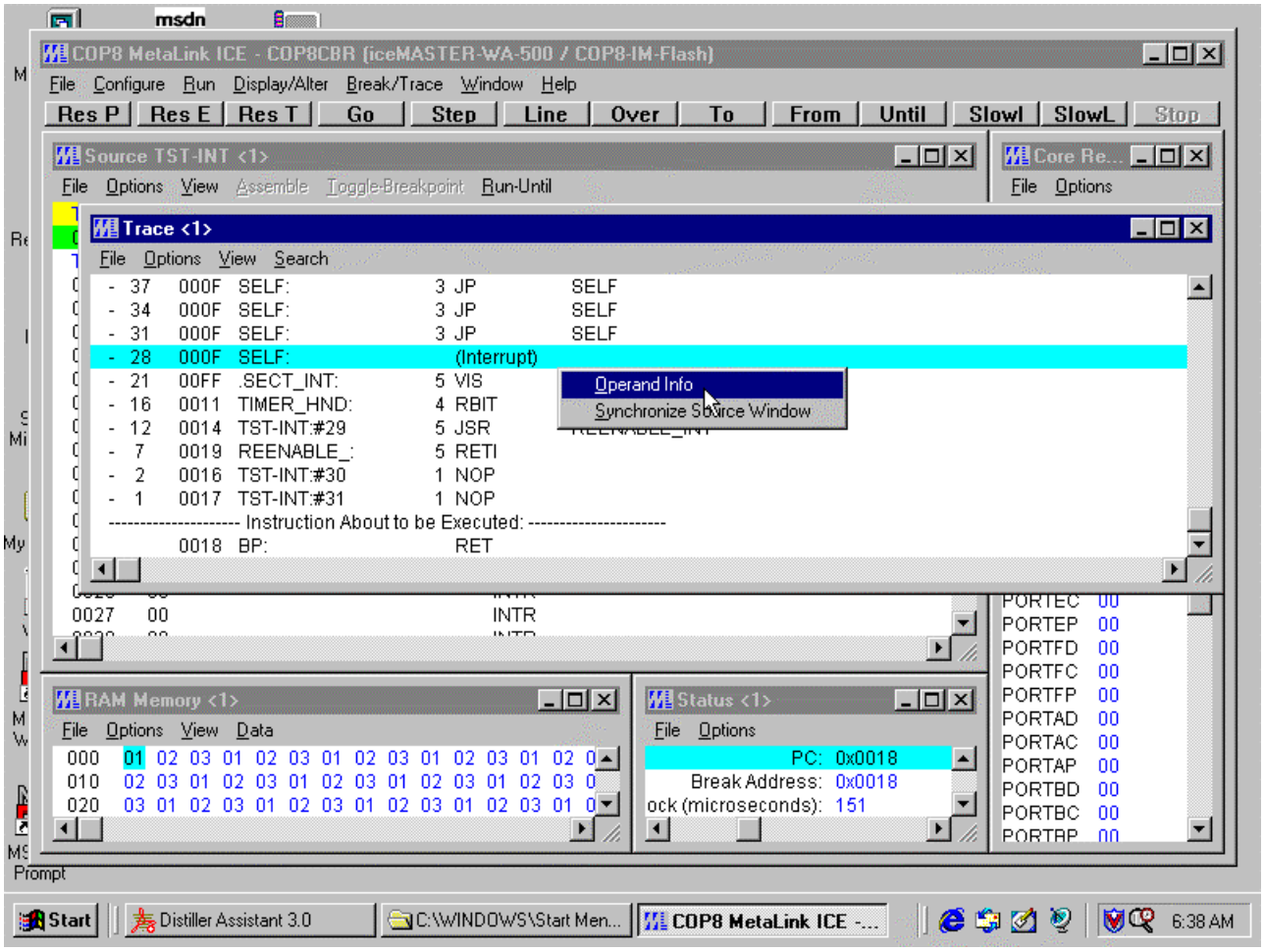
> **Unsolicited breakpoint encountered at 0x*hhhh*.**
> **(The program contains a non-breakpoint BRK instruction.)**

because the debugger did not set the break-point in the "normal" manner. (See the Operational Considerations chapter.)

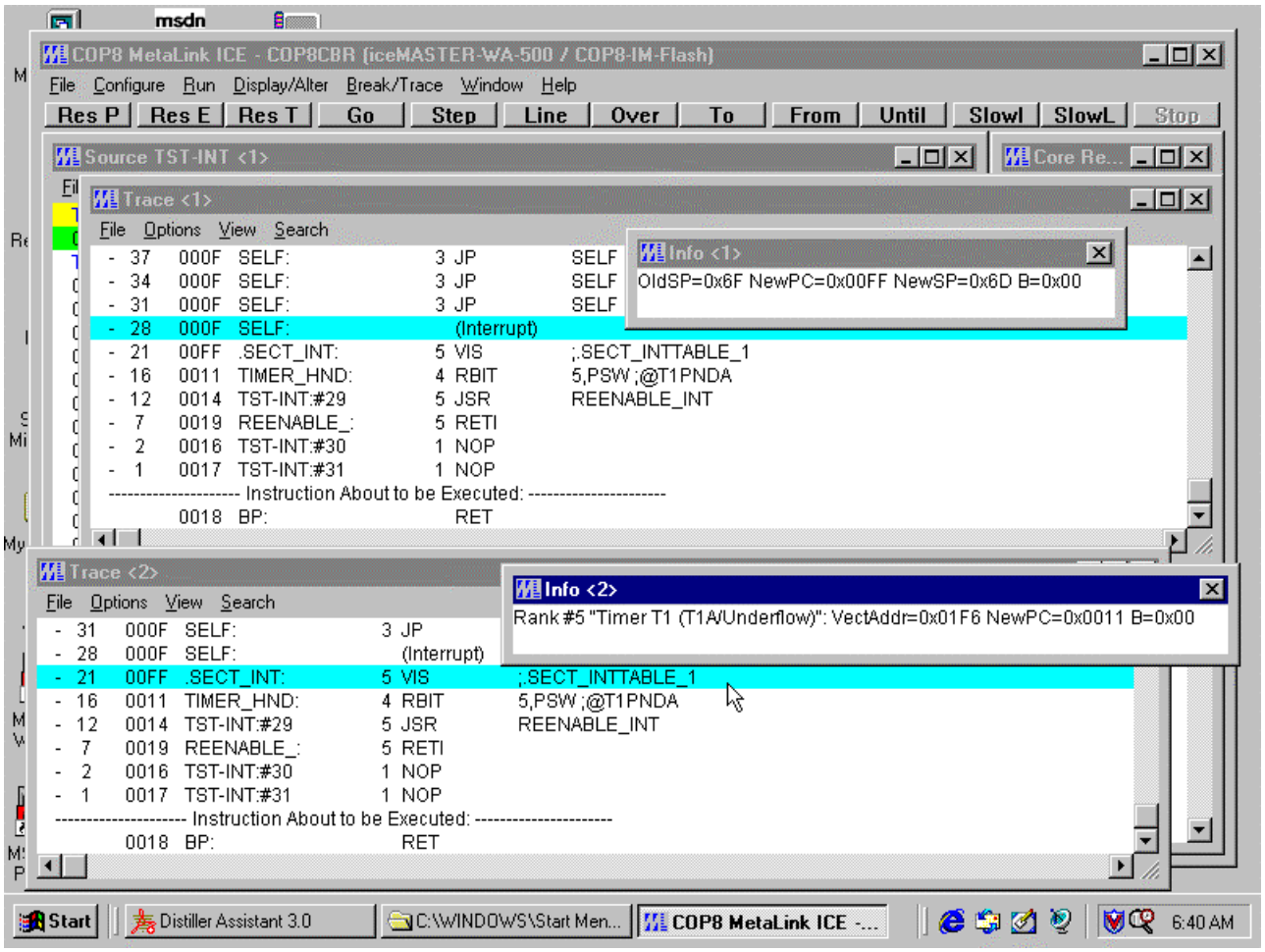
If you try to execute a **RETF** instruction in program code flash memory, it will behave exactly like a **RET** instruction, per the NSC device specification.

Trace Operand/Instruction Value Information

In the Model 500 emulators (COP8-IM-Flash), the Trace contains information about the instruction that was executed. This includes such things as operand values (memory contents), Stack Pointer “before” and “after” values, etc. Right-click on the currently-selected line and select the ‘Operand Info’ command to view this information:



An example of the information shown follows:



As you scroll the currently-selected line through the Trace Window, the information in the pop-up information box will be updated to reflect the value(s) for the instruction at the currently-selected line.

Command Window

Several commands are available in the Command Window to configure and manipulate the COP8 flash device:

FLASH_INIT

Initialize the emulator based on previously specified **FLASH_...** commands (described below). Note that none of the settings specified using any of the **FLASH_...** commands following in this section will be set in the emulator until this **FLASH_INIT** command is executed.

FLASH_HALT *str*

Configure|Flash|Halt Mode: Specify the state of Halt Mode (the HALT bit in the OPTION register), where ***str*** is one of the following:

ENABLE	Enable Halt Mode.
DISABLE	Disable Halt Mode.

DFLASH_HALT

Display the current setting of the **FLASH_HALT** value (see above). This is the value that will be used when the next **FLASH_INIT** command is executed.

FLASH_WDT *str*

Configure|Flash|Watchdog Feature: Specify the state of Watchdog Feature (the WDT bit in the OPTION register), where ***str*** is one of the following:

ENABLE	Enable the Watchdog feature.
DISABLE	Disable the Watchdog feature.

DFLASH_WDT

Display the current setting of the **FLASH_WDT** value (see above). This is the value that will be used when the next **FLASH_INIT** command is executed.

FLASH_START *str*

Configure|Flash|Start Location after RESET: Specify whether the COP8 processor will begin executing from the Flash or the Boot ROM following a RESET (the FLEX bit in the OPTION register), where ***str*** is one of the following:

FLASH	Begin execution at location zero in the Flash memory following a RESET.
ROM	Begin execution in the Boot ROM following a RESET.

DFLASH_START

Display the current setting of the **FLASH_START** value (see above). This is the value that will be used when the next **FLASH_INIT** command is executed.

FLASH_CKI *n*

Configure|Flash|CKI Frequency Range: Specify the CKI frequency being used in your target system. The value of ***n*** is one of the number values displayed by the **DFLASH_CKI_ALL** command below. The value specifed here will be used to update the PGMTIM register appropriately.

DFLASH_CKI

Display the current setting of the **FLASH_CKI *n*** value (see above). The format of the display is "*number*: *description*". This is the value that will be used when the next **FLASH_INIT** command is executed.

DFLASH_CKI_ALL

Display all possible (legal) values and descriptions that can be used for the **FLASH_CKI *n*** command (see above). The format of the display is "*number*: *description*".

FLASH_ERASE

File|Erase Flash: Mass Erase the flash memory in the COP8 device. No questions are asked. The OPTION register is left in its erased state, so you must follow this command with any appropriate **FLASH_...** and **FLASH_INIT** configuration commands.

SWBREAKPOINTS *str*

Configure|Preferences|Use Software Breakpoints: (iceMASTER-WA Model 500 only) Specify whether you want to use software code breakpoints or hardware code breakpoints, where ***str*** is one of the following:

ON	Use software code breakpoints.
OFF	Use hardware code breakpoints.

Chapter 6: Operational Considerations

The iceMASTER-WA COP8-EM/DM/IM-Flash emulator is designed to be as close as is possible to an actual device. In most cases you will not be aware of any difference since the devices used to emulate are the actual microcontrollers. Since we need to know what is going on during emulation there are a few constraints placed upon us, and a few precautions you can take to prevent problems.

Emulation Notes

The following characteristics apply to all iceMASTER-WA emulators except where noted:

- Model 100 (COP8-EM-Flash) and Model 400 (COP8-DM-Flash):
  The **BRK** instruction is used to implement software breakpoints. There are no hardware breakpoints. When emulation starts, the program code at all breakpoints will be replaced by a **BRK** instruction and possibly **NOP**s to fill out a multi-byte instruction.

  Note that the iceMASTER-WA Model 500 lets you chose whether you want to use software breakpoints or hardware breakpoints. All the items concerning software breakpoints here apply when you elect to use software breakpoints.

  Also note that when you use hardware breakpoints in the Model 500, the breakpoints are “break after”. The emulator will stop after the instruction at the breakpoint location is executed. If the instruction at the hardware breakpoint location is a single-cycle instruction, the emulator will stop 2 instructions after the breakpoint.

- If your application code contains **BRK** instructions, the emulator behaves as though a breakpoint occurred when the COP8 CPU executes a **BRK** instruction, but will display the following message to alert you:

  Unsolicited breakpoint encountered at 0x*hhhh*.
  (The program contains a non-breakpoint BRK instruction.)

- If a breakpoint is set on an instruction that could potentially be skipped, emulation will break only when that instruction is actually executed. Emulation will never break on a skipped instruction.
- In addition, when a breakpoint is set on an instruction which could potentially be skipped and that instruction is skipped during emulation, the address and data values captured in the trace of execution will not reflect the program code at that address. Instead, the first cycle will be that of the skipped **BRK** instruction. If the original program code instruction was a multi-byte instruction, subsequent cycles will be the same as that of an executed **NOP**.
- When a breakpoint is set on an instruction, emulation will break before that instruction executes. The two (2) bytes below the stack pointer will be overwritten when a breakpoint is reached because the **BRK** instruction used for the breakpoint is actually executed.
- Timer operation: There is no delay in stopping or restarting running timers upon reaching a breakpoint or when emulation resumes.
- HALT Mode. To allow clock re-synchronization in the COP8 microcontroller, it is necessary to program two **NOP**s immediately after the microcontroller comes out of HALT mode. When the multi-input wake-up interrupt is enabled, the first two instructions of the interrupt routine must be **NOP**s. If no interrupt is used, then two **NOP**s must follow the instruction that put the microcontroller into HALT mode.
- IDLE Mode. Like HALT mode, it is necessary to program two **NOP**s to allow clock re-synchronization upon return from IDLE mode. The **NOP**s are placed either at the beginning of the IDLE Timer interrupt routine or follow the instruction that put the microcontroller into IDLE mode.

Static

Perhaps the most difficult problem anyone who uses MOS devices will face is static. You may go for years with no fault traceable to static, or you may blow every part you touch. The iceMASTER-WA emulator can be as sensitive to static as any other circuit. The microcontroller devices in the emulators are especially vulnerable since adding extra protection would change response characteristics. This would be a step away from the real world. The built-in protections internal to the devices are operative.

We do still recommend, however, that you take every precaution regarding static. The use of grounding straps, static free workstations, and a little extra care in handling the emulator (and any MOS part) can prevent troubles later.

Power

5 Volt-Powered Emulators

Emulators shipped before June 2000 are configured to use a 5 volt power supply. These units support only 5 volt operation of the COP8 flash device.

Power is supplied with a standard 2.5mm x 5.5mm x 9.5mm barrel plug and jack, center positive (Switchcraft 760). The power supply must provide +5 volts 5%, at 1 ampere. The ripple voltage must be no greater than 50 millivolts, peak-to-peak.

9 Volt-Powered Emulators

Emulators shipped after June 2000 are configured to use a 9 volt power supply. These units support a COP8 flash device operating over the full, wide voltage range (2.7 volts – 5.5 volts).

Power is supplied with a standard 2.5mm phone plug and jack, tip positive. The power supply must provide +9 volts 5%, at 300 milliamps. The ripple voltage must be no greater than 300 millivolts, peak-to-peak.

Clock Drivers

Many of the external clock drivers commonly used provide TTL level outputs. This can be a problem for not only the emulator but your target design as well. COP8 microcontrollers require a CMOS oscillator or clock levels to function reliably.

Microcontroller Serial Port

After breaking emulation, all interrupts are disabled. A serial port transmit or receive interrupt which occurs after the breakpoint has been encountered will be ignored.

IDLE and HALT Modes

During emulation, the Emulating Window may show a status of **Inactive**. This status indicates that the COP8 microcontroller is not executing instructions. Normally this can be due to an extended RESET pulse, or the microcontroller is in HALT or IDLE mode. If this condition occurs for any other reason turn to the Troubleshooting Chapter.

If the microcontroller is in HALT or IDLE mode and you execute the host-break command (by pressing either the Stop button or the ESC key), the processor will break on the instruction immediately after the instruction that put the processor into HALT or ILDE mode.

Memory Corruption

If you notice that all memory contents (RAM, Code, Registers) suddenly change to a value of 0x37 or 0x45, it means that the synchronization of the interface between the control processor in the emulator and the COP8 flash device in your target system has been lost.

This condition could be caused by having too low a voltage going to the COP8 flash device in your target system.

OPTION Register

The OPTION register is located at address 0x7FFF in the COP8CBR device. If you "manually" change bits in that byte, some of the settings specified in the ***Configure|Flash...*** dialog may be affected.

If you manually enable (turn on) the Security bit (SEC) in the OPTION register, the system will oblige and the emulator and COP8 flash device will continue to function properly (see page 17). However:

- The data in all source and code memory windows will display as 0xFF.
- You cannot set a software breakpoint.
- If you begin emulation, then ***Stop*** emulation and then attempt to view the Trace, all opcode and data bytes in the Trace will display as 0xFF, but the addresses in the Trace will be correct.
- Any software break-points which were set in the COP8 device's flash memory at the time you manually set the Security bit will be "permanent" (i.e., BRK instructions will be present in the COP8 device's flash memory at all locations where you had break-points set). The only way to remove these software break-points after setting the Security bit is to Mass Erase (***File|Erase Flash***) the device.

Note: The debugger will not check again to see if the Security bit is set until you perform, or indirectly cause the debugger to perform, one of the actions described on page 21.

Clock Monitor / Watchdog Reset

If your application causes a clock monitor reset, recreated port G1 will not issue a reset pulse until the clock has started up again. At that time, G1 will be driven low for 16-32 instruction cycles in order to issue a reset.

When G1 is connected to the reset pin and a watchdog timeout or clock monitor reset occurs, the reset pulse may briefly de-assert. That is, instead of one reset pulse lasting 16-32 instruction cycles, you may see up to three shorter pulses that total 16-32 instruction cycles.

Null Target Board

When using the null target powered by the emulator, do not attempt to drive anything with the port pins. The emulator supplies just enough current for the COP8 and the power LED. The port pins can only be used to sense external events or to be probed by a scope – not to drive a resistive load. If however, you attach an external power supply between DVCC and DGND, you may use the port pins to drive external hardware.

Note that the null target's probe pin footprints for CKI and G7 are not attached to the COP8's CKI and G7. If you wish to probe these signals, you must probe the COP8's pins directly.

Chapter 7: Troubleshooting

Before starting any fault investigation, make sure that all connectors are in good condition and fully seated. Take note of any physical damage to the unit.

Several common problems are covered in this chapter. If this isn't enough to get the emulator back into operation, contact MetaLink.

Before Calling

Have the system near the phone so that problems may be walked through. If the unit needs to be sent in for repair, you will also need the following information:

- The emulator's model and serial numbers (on the bottom of the emulator chassis)
- The software revision level (see the ***Configure|Identification*** Window)
- The address to which MetaLink will ship the repaired unit

Power

Ensure that your target system is powered correctly.

Ensure that the emulator itself is powered correctly:

5 Volt-Powered Emulators

Emulators shipped before June 2000 are configured to use a 5 volt power supply. These units support only 5 volt operation of the COP8 flash device.

Power is supplied with a standard 2.5mm x 5.5mm x 9.5mm barrel plug and jack, center positive (Switchcraft 760). The power supply must provide +5 volts 5%, at 1 ampere. The ripple voltage must be no greater than 50 millivolts, peak-to-peak.

9 Volt-Powered Emulators

Emulators shipped after June 2000 are configured to use a 9 volt power supply. These units support a COP8 flash device operating over the full, wide voltage range (2.7 volts – 5.5 volts).

Power is supplied with a standard 2.5mm phone plug and jack, tip positive. The power supply must provide +9 volts 5%, at 300 milliamps. The ripple voltage must be no greater than 300 millivolts, peak-to-peak.

Communications Failure

First, check the RS-232 connection. Compare the RS-232 cable with the illustration in the Hardware Description Chapter. Replace the cable if needed. A break-out box will facilitate a check on the presence and level of the RS-232RS-232 signals on the cable. Active signals will be at a nominal 10V and the polarity may be plus or minus depending on the hardware. If no activity is seen on Pin 2 (Transmit), the Host Computer has a fault in its interface card. If Pin 2 is active and Pin 3 is inactive (not toggling), the emulator has a fault.

A persistent fault may indicate that the emulation microcontroller has been damaged. For example, failure of the oscillator to start properly will result in a communication failure after, or during sequencer file loading.

Ensure that the COP8 flash device in the target system is receiving a clock.

Emulation Problems

In stand-alone mode, load and run one of the demo programs that are supplied on the distribution diskettes. If the program runs correctly, the problem may not be with the emulator but with the target interface. Keep an open mind. Even in known good target systems, failures occur. It may even be possible for a real device to work in your target where the emulator has trouble. This is usually a problem with tolerances, not differences. If you encounter this, please call us so we can determine quickly where the problem lies.

Carefully consider the application:

- Verify that the emulator is configured properly
- Look for unexpected resets (e.g., watchdog timers)
- Check interrupt routines for proper returns to normal code execution

If these procedures restore operation in the stand-alone mode, or if the unit worked in the stand-alone mode without correction, it is necessary to determine if the target is causing the fault or if the emulator has a fault that doesn't manifest in stand-alone mode.

Troubles to watch for include:

- Bus contention
- Excessive loading
- Failed components in the target system (especially failures that are likely to damage the emulator)

If you have any questions contact MetaLink for technical assistance.

Chapter 8: Feature Comparison

The following table is a comparison of the features in the various models of the iceMASTER-WA COP8-EM/DM/IM-Flash emulator:

iceMASTER-WA			
	Models		
	iceMASTER-WA-101/102 (Model 100) COP8-EM-Flash	iceMASTER-WA-400 (Model 400) COP8-DM-Flash	iceMASTER-WA-500 (Model 500) COP8-IM-Flash
Basic Features			
Real-time Emulation	Yes	Yes	Yes
Maximum Frequency (MHz)	20MHz	20MHz	20MHz
Map Clock to Target	Always	Always	Always
User-Programmable Clock	N/A	N/A	N/A
Low Voltage Support	Yes	Yes	Yes
Programming Capability	Yes	Yes	Yes
Pinouts Supported	All	All	All
Packages Supported	All	All	All
Connection to PC	RS-232	RS-232	RS-232
Maximum Baud Rate	115.2Kb	115.2Kb	115.2Kb
Power Indicator LED	Yes	Yes	Yes
Status Indicator LED	Yes	Yes	Yes
Field Upgradeable	Yes	Yes	Yes
Accessories			
Enclosed in Case	No	Yes	Yes
Power Supply Included	Yes, 110 or 220	Yes	Yes
Serial Cable Included	Yes	Yes	Yes
Breakpoints			
Number of Software Breakpoints	16	32K	32K[1]
Number of Hardware Breakpoints	None	None	32K[1,2]
Pass Counter	No	No	Yes[3]
Trace			
Real-Time Trace	No	Yes	Yes
Trace Memory Size (Frames)	N/A	32K	32K
Trace Search	N/A	Yes	Yes
Read Data (Operand Value) Trace	N/A	No	Yes
Adjustable Trace Trigger Points	N/A	No	Yes[3]
Transparent Trace (view trace while emulation continues)	N/A	No	Yes
Trace Filtering (On/Off), Real Time	N/A	No	Yes
Compilers/Assemblers			
Symbolic Debug	Yes	Yes	Yes
Source-Level Debug	Yes	Yes	Yes
Byte Craft	Yes	Yes	Yes
IAR	Yes	Yes	Yes
National Semiconductor	Yes	Yes	Yes
User Interface			
Changed-Value Highlighting	Yes	Yes	Yes
Watch Window(s)	Yes	Yes	Yes
Code Memory Window(s)	Yes	Yes	Yes
Internal Data Memory Window(s)	Yes	Yes	Yes
File|Load	Yes	Yes	Yes
File|Unload	Yes	Yes	Yes
File|Store	Yes	Yes	Yes
File|Restore	Yes	Yes	Yes
Run|Reset|Processor	Yes	Yes	Yes

Run|Reset|Emulator	Yes	Yes	Yes
Run|Reset|Target	Yes	Yes	Yes
Run|Go	Yes	Yes	Yes
Run|From...	Yes	Yes	Yes
Run|Until...	Yes	Yes	Yes
Run|Slow Motion|Instruction	Yes	Yes	Yes
Run|Slow Motion|Line	No	Yes	Yes
Run|Step	Yes	Yes	Yes
Run|Line	No	Yes	Yes
Run|Over	No	Yes	Yes
Run|To	No	Yes	Yes
Run|Repetition|Count	Yes	Yes	Yes
Run|Stop	Yes	Yes	Yes
Execution Timer	Yes	Yes	Yes
Reset Processor During Emulation	No	No	Yes

[1]: The breakpoint type is selectable dynamically: either 32K hardware breakpoints or 32K software breakpoints.

[2]: Hardware breakpoints are “break after”. The emulator stops after the instruction at the hardware breakpoint location is executed. If the instruction at the hardware breakpoint location is a one-cycle instruction, the emulator will execute two instructions, rather than just one, before stopping.

[3]: Only available when you are using hardware breakpoints, not software breakpoints.