EASE64158

Program Development Support for the MSM64153 Family

User's Manual

Rev. 1.01

May. 1994

OKI

NOTICE

1. The information contained herein can change without notice owing to product and/or technical improvements. Please make sure before using the product that the information you are referring to is up to date.

2. The outline of action and examples of application circuits described herein have been chosen as an explanation of the standard action and performance of the product. When you actually plan to use the product, ensure that external conditions are reflected in the actual circuit and assembling designs.

3. **NO RESPONSIBILITY IS ASSUMED BY US FOR ANY CONSEQUENCE RESULTING FROM ANY WRONG OR IMPROPER USE OR OPERATION, ETC. OF THE PRODUCT.**

4. Neither indemnity against nor license of a third party's industrial and intellectual property right, etc. is granted by us in connection with the use of the product and/or the information and drawings contained herein. No responsibility is assumed by us for any infringement of a third party's right which may result from the use thereof.

5. The product described herein falls within the category of strategic goods, etc. under the Foreign Exchange and Foreign Trade Control Law. Accordingly, before exporting the product or any part thereof, you are required under the law to file an application for an export license by your domestic government.

6. Although we endeavor to ensure that the information contained herein is accurate and reliable, we welcome your comments and suggestions addressed to the following:

   1st Sales Engineering Section
   Product Development Department
   Logic LSI Division
   Electronic Devices Group
   OKI ELECTRIC INDUSTRY CO., LTD.
   7-5-25 Nishi-Shinjuku, Shinjuku-ku
   Tokyo 160   JAPAN
   Phone: 81-3-5386-8137 (direct line)

7. No part of the contents contained herein may be reprinted or reproduced without our prior permission.

8. MS-DOS is a registered trademark of Microsoft Corporation.

PREFACE

This manual explains the operation of the EASE64158 in-circuit emulator for Oki Electric's MSM64153 family of CMOS 4-bit microcontrollers. The EASE64158 is configured from the POD64158 evaluation module and the EASE-LP2 special-purpose control system.

The MSM64153 family is all 4-bit microcontrollers that incorporate and provide all functions included in the MSM64E153 evaluator chip, which was designed for the emulator. This family currently includes four devices:

- MSM64152
- MSM64153
- MSM64155
- MSM64158

The following are related manuals:

- MSM6415X User's Manual
    - MSM6415X hardware description
    - MSM6415X instruction set description
    - Addressing description

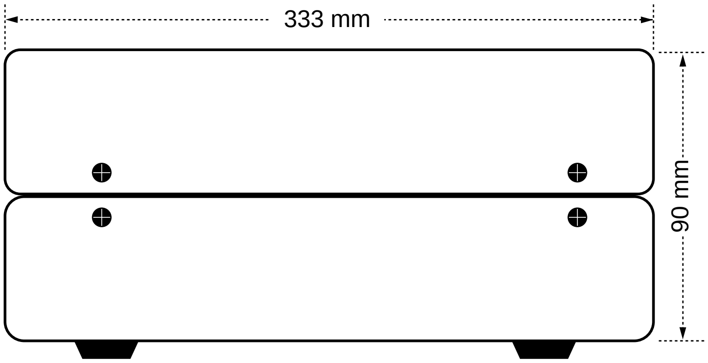
- ASM64K Cross-Assembler User's Manual
    - ASM64K assembler operation description
    - ASM64K assembly language description

! MSM6415X User's Manual is the user's manual corresponds to one of the MSM64153 family microcontroller that has been specified by customer.

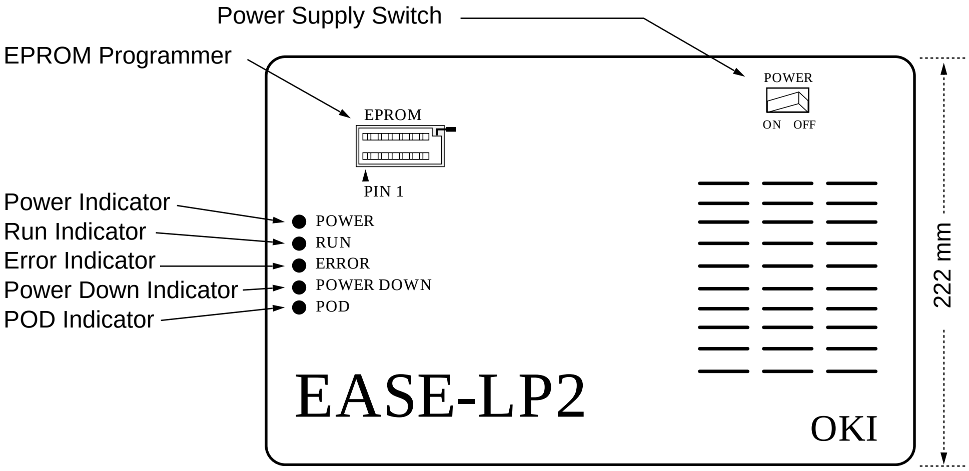
TABLE OF CONTENTS

Appendix

EASE-LP2 External Views (1)

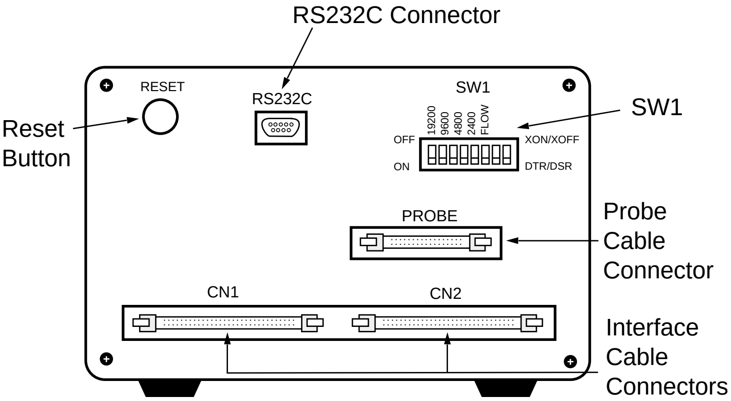


Front View

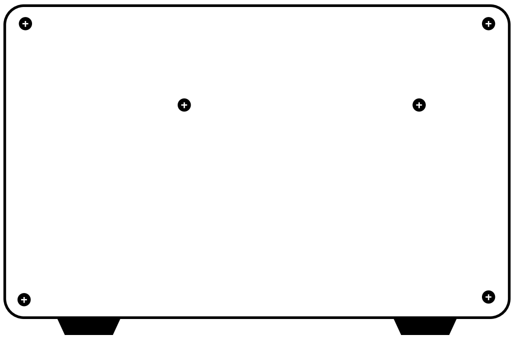


Top View

EASE-LP2 External Views (2)

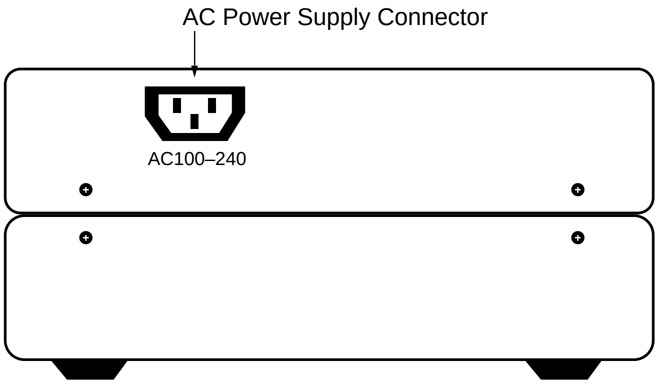


Left View



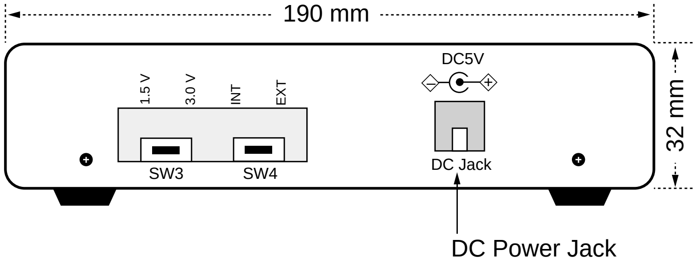
Right View

EASE-LP2 External Views (3)

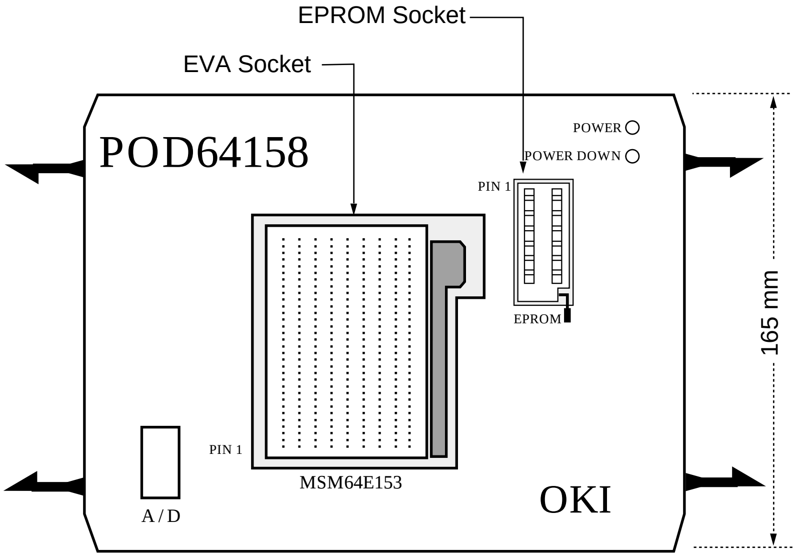


Rear View

POD64158 External Views (1)

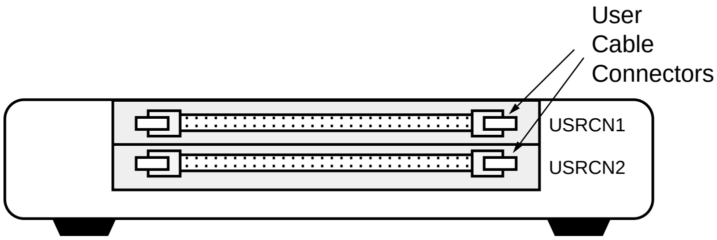


Front View

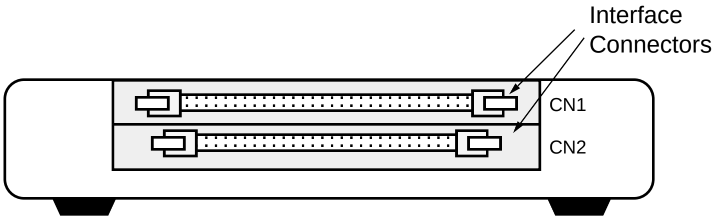


Top View

POD64158 External Views (2)

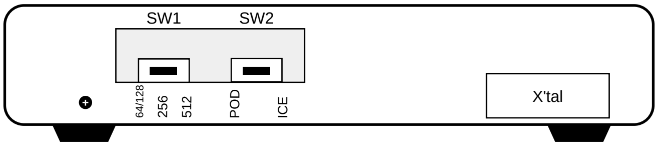


Left View



Right View

POD64158 External Views (3)



Rear View

Explanation of Symbols

❗ Indicates a supplemental explanation of particular importance that relates to the topic of the current text.

Example Indicates a specific example of the topic of the current text.

SEE ▷ Indicates a section number or page number to reference for related information on the topic of the current text.

(☞ 1) Indicates the number of a footnote with a supplemental explanation of particular words in the current text.

☞ **1** Indicates a footnote with a supplemental explanation of words marked with the above-described symbol. The numbers following each symbol correspond to each other.

Chapter 0

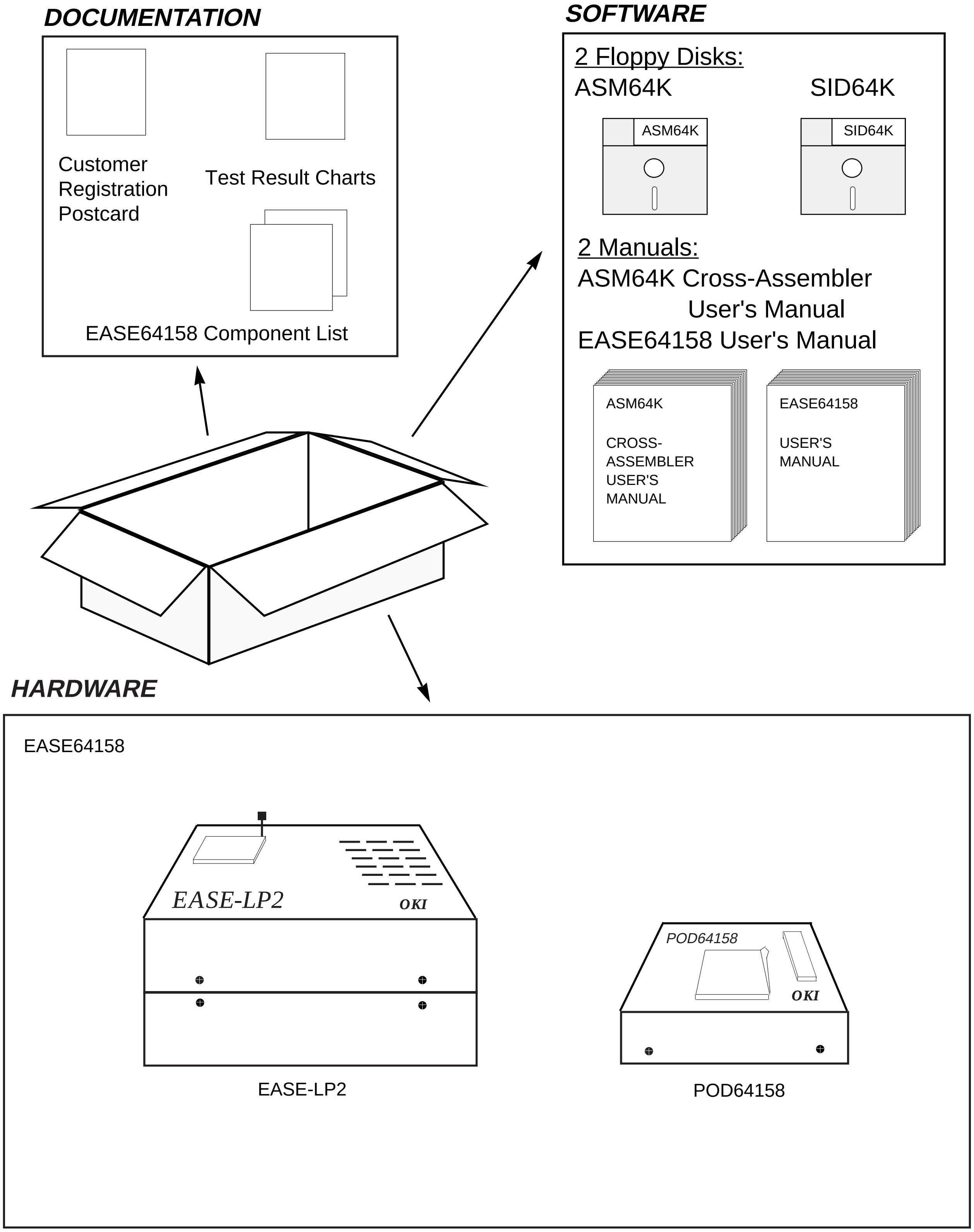
Before Starting

This chapter describes the first things you should do after taking delivery of an EASE64158 program development support system.

Thank you for buying Oki Electric's EASE64158 program development support system. When your system was shipped we made every effort to ensure that it would not be damaged or mispacked, but we recommend that you confirm once more that this did not occur following the explanations in this chapter.

The RS232C cable, floppy disks, or other items may differ depending on the model of host computer that you will use. Use with a different model could cause damage to the hardware, so please take particular care to avoid this. If the system shipped to you was damaged, if any components were missing, or if your host computer model is different, the please contact the dealer from whom you purchased the system or Oki Electric's sales department.

0-1. Confirm Shipping Contents (1)



DOCUMENTATION

Customer Registration Postcard

Test Result Charts

EASE64158 Component List

SOFTWARE

2 Floppy Disks:

ASM64K SID64K

2 Manuals:

ASM64K Cross-Assembler User's Manual

EASE64158 User's Manual

HARDWARE

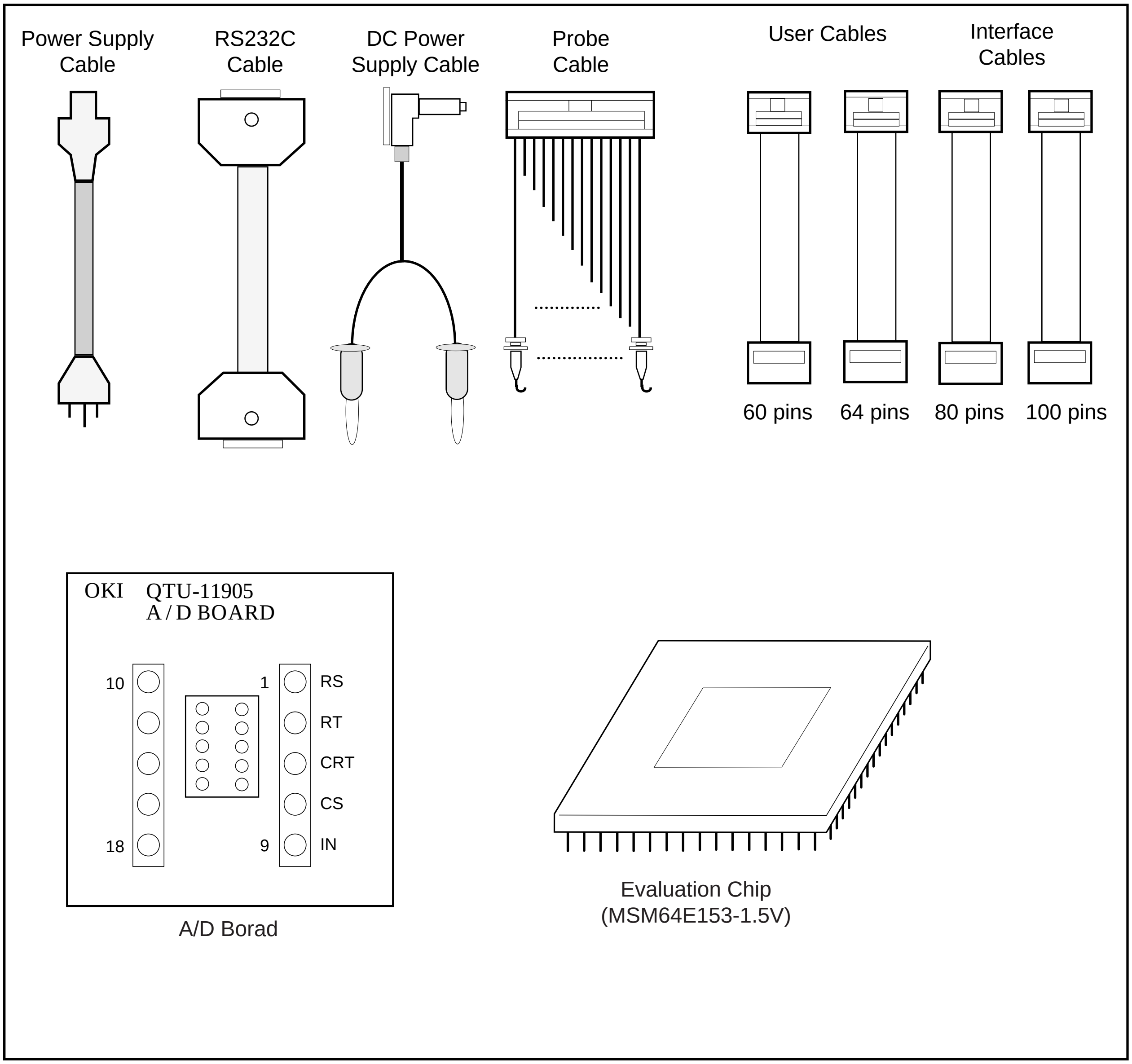
EASE64158

EASE-LP2

POD64158

0-1. Confirm Shipping Contents (2)

ACCESSORIES



Your purchase of the EASE64158 will be followed be delivery of the necessary hardware, software, and manuals in the shipping box illustrated in the upper left of page 2. After taking delivery, open the box and confirm that it contains all the contents illustrated on pages 2 and 3.

Each component is described below. Note that those marked with ☞ will differ depending on the model of host computer.

Documents

Customer Registration Postcard

Oki Electric uses this to record you in our customer list in order to inform you of product maintenance and version upgrades. Please fill out the requested items and send the postcard in as soon as possible. If you do not send in the registration postcard, it will it more difficult to provide you with maintenance and version upgrade service.

EASE64158 Components List

This is a list of the items shipped.

Test Results Charts

This chart shows that the EASE64158 passed all tests before shipping.

Hardware

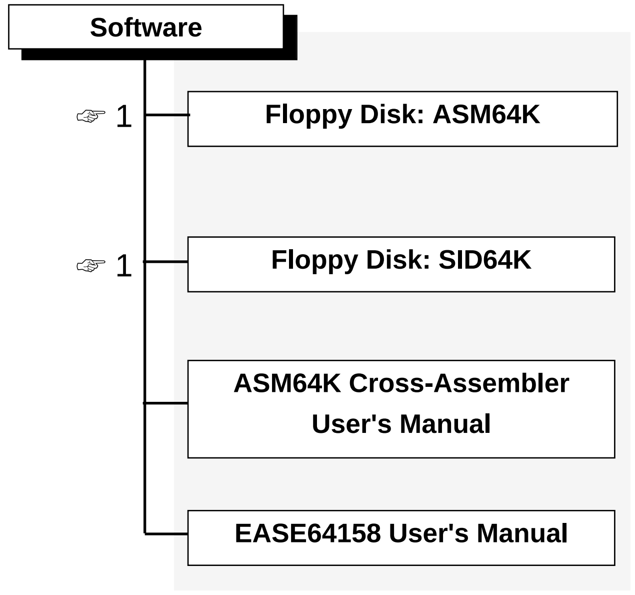
EASE-LP2

This is the EASE-LP2 control system. It contains hardware for host computer communications, EPROM programming, etc.

POD64158

This is the POD64158 evaluation module. It emulates the operation of the MSM64153 family.

! The EASE-LP2 and POD64158 will be called “EASE64158” or “emulation kit” for short.



Floppy Disk: ASM64K

This disk contains the ASM64K executable files. It can be supplied in the formats described below. Floppy disk contents are explained in Section 0-2.

Floppy Disk: SID64K

This disk contains the SID64K executable files. It can be supplied in the formats described below. Floppy disk contents are explained in Section 0-2.

ASM64K Cross-Assembler User's Manual

This is the user's manual for the ASM64K cross-assembler.

EASE64158 User's Manual

This is the user's manual (this manual) for the EASE64158.

☞ **1** Available floppy disk formats

MS-DOS format
  (1) 3.5-inch 2HD (1.21 Mbytes)
  (2) 5.25-inch 2HD (1.21 Mbytes)
PC-DOS format (for IBM PC/AT)
  (1) 3.5-inch 2HD (1.44 Mbytes)
  (2) 5.25-inch 2HD (1.232 Mbytes)

Accessories

Power Supply Cable

This cable connects to the power supply connector.

☞ 2 **RS232C Cable**

This cable connects the EASE-LP2 with a host computer. There are two types: for NEC-PC9801 and Oki if800 series computers, and for IBM-PC/AT computers.
If not specified before shipment, then the cable for NEC-PC9801 and Oki if800 series computers will be shipped.

Probe Cable

This cable connects to the EASE-LP2 probe connector.

User's Cables

These cables connect the POD64158 to the user's application system. Two cables are supplied: a 60-pin flat cable and a 64-pin flat cable.

Interface Cables

These cables connect the EASE-LP2 and the POD64158. Two cables are supplied: a 100-pin flat cable and an 80-pin flat cable.

DC Power Supply Cable

This cable supplies VDD to the POD64158 when used standalone. It connects to the POD64158's DC power jack.

A/D Board

Terminal board for A/D converter.
This board can be used only with the MSM64153 family microcontrollers that are equipped with A/D converter.

☞ 3 **Evaluation Chip**

An evaluation chip (MSM64E153) mounted on the POD64158.

☞ 2

Unless specified before the EASE64158 is shipped, a cable for the NEC-PC9801 series will be shipped. If you will use an Oki if800 series computer, then you can also use this cable. If you will use an IBM-PC, then please tell the responsible salesperson before your system is shipped so that a special-purpose cable will be included. If you forget to specify the personal computer that you will be using, then please contact the responsible salesperson to exchange cables.

To identify which type of cable was shipped to you, please refer to the features listed below.

(1) NEC-PC9801 series	25-pin D-SUB male connector on one side, and 9-pin male connector on the other side.
(2) IBM-PC/AT	9-pin male connector on one side, and 9-pin female connector on the other side.

If you will be using a host computer other than an NEC-PC9801 series, Oki if800 series, or IBM PC/AT, then the connectors and their cable connections may have to be changed. Refer to Appendix 3 and 4 to change the connectors or cable connections to match the host computer you will use.

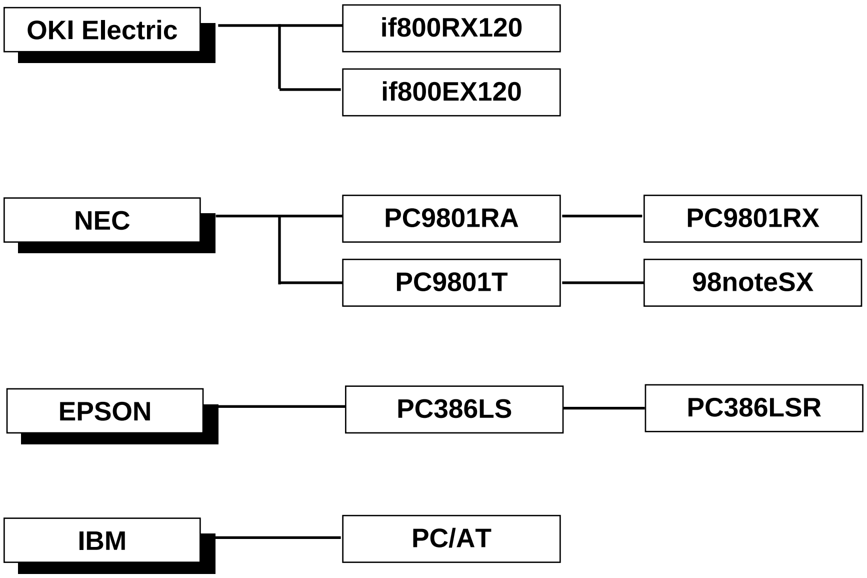
☞ 3

Two types of evaluation chips, a 1.5-V operating MSM64E153-1.5V, and a 3.0-V operating MSM64E153-3.0V, are provided. When the EASE64158 is shipped, the 3.0-V operating MSM64E153-3.0V is mounted on the POD64158.

0-2. Confirm Floppy Disk Contents

0-2-1. Host Computer

SID64K, the symbolic debugger for EASE64158, has been confirmed to operate with the following computer models.



All of the above models must have at least 640 Kbytes of memory.

Oki Electric has not confirmed direct operation with computers other than those listed above.

Before purchasing the EASE64158, your sales dealer or the Oki Electric sales department should verify the computer model that you will use. However, if after buying the system you want to consider a model other than those listed above, then please consult with Oki Electric’s application engineering section.

0-2-2. Operating System

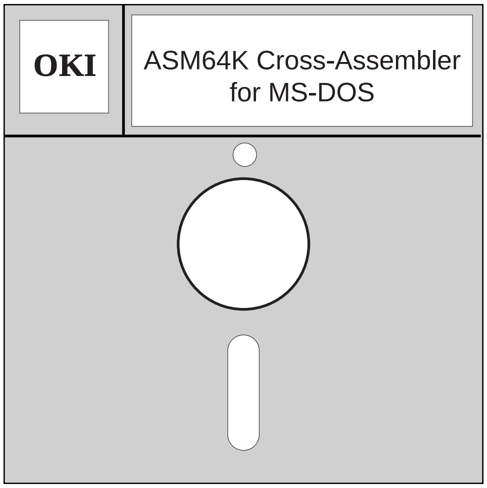
The operating system of computers other than IBM-PCs should be Japanese MS-DOS version 3.1 or later. For IBM-PCs, it should PC-DOS version 3.1 or higher.

0-2-3. Floppy Disk Contents

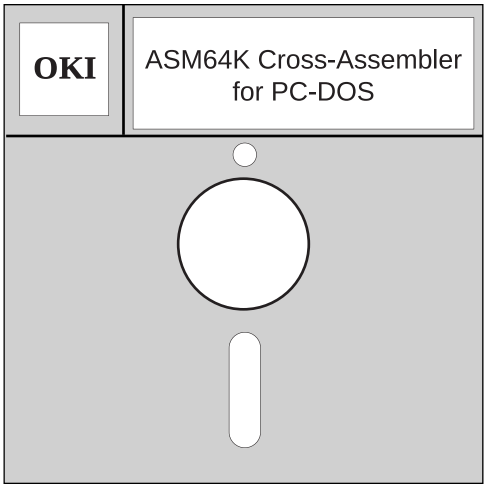
If the conditions described in Sections 0-2-1 and 0-2-2 are satisfied, then there will be no problem with your host computer model. Next, check the contents of the floppy disks.

(1) ASM64K floppy disk contents

As shown below, the label pasted on the floppy disk will differ for the PC9801/if800 series and the IBM-PC.

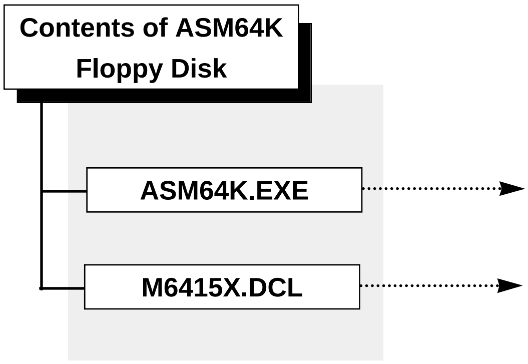


For PC9801/if800 Series



For IBM-PC Series

If you use the floppy disk for the wrong type of computer, then it will not be able to read the floppy disk contents, so check whether or not the correct disk is inserted. Each file included on the floppy disk and a brief explanation is given below.

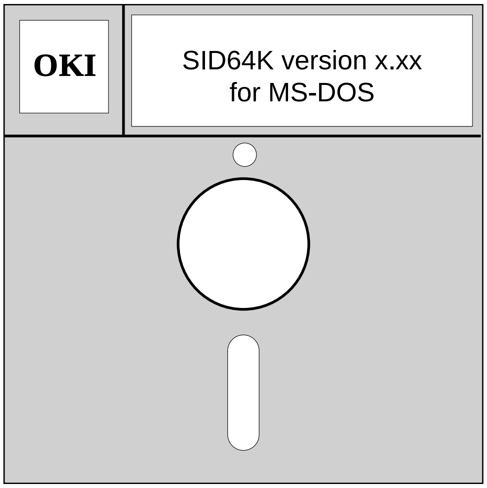


Executable file for ASM64K cross-assembler.

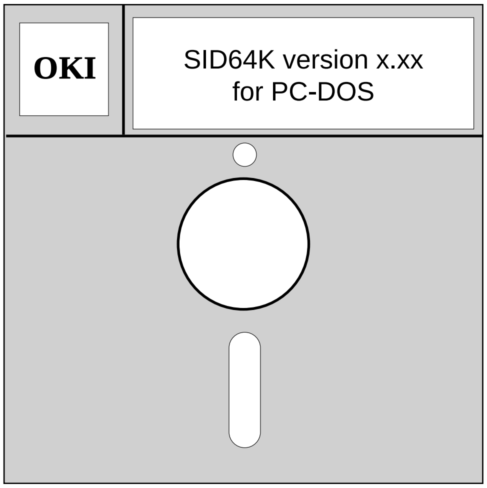
DCL file for ASM64K cross-assembler (☞3).
For details, refer to ASM64K Cross-Assembler User's Manual.

(2) SID64K Floppy Disk Contents

As shown below, the label pasted on the floppy disk will differ for the PC9801/if800 series and the IBM-PC.

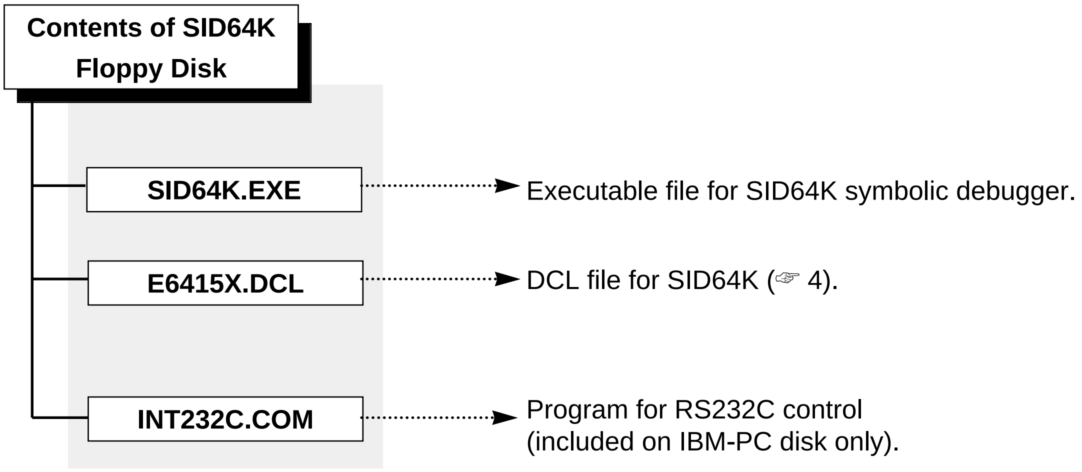


For PC9801/if800 Series



For IBM-PC Series

If you use the floppy disk for the wrong type of computer, then it will not be able to read the floppy disk contents, so check whether or not the correct disk is inserted. Each file included on the floppy disk and a brief explanation is given below.



SID64K.EXE: Executable file for SID64K symbolic debugger.

E6415X.DCL: DCL file for SID64K (☞ 4).

INT232C.COM: Program for RS232C control (included on IBM-PC disk only).

☞ 3 The DCL file for ASM64K defines the following items to match operation with the appropriate member of the MSM64153 family.

(a) SFR (special function register) addresses and access attributes.
(b) Code memory (program memory) address range.
(c) Data memory address range.

Currently, the following DCL files are provided for each device in the MSM64153 family. Note that the floppy disk contains all the DCL files for the device supported by ASM64K.

MSM64152:	M64152.DCL
MSM64153:	M64153.DCL
MSM64155:	M64155.DCL
MSM64158:	M64158.DCL

☞ 4 The DCL file for SID64K defines the following items to match operation with the appropriate member of the MSM64153 family. The DCL file is read when SID64K is invoked.

(a) SFR (special function register) addresses and access attributes.
(b) Code memory (program memory) address range.
(c) Data memory address range.

Currently, the following DCL files are provided for each device in the MSM64153 family. Note that the floppy disk contains all the DCL files for the device supported by SID64K.

MSM64152:	E64152.DCL
MSM64153:	E64153.DCL
MSM64155:	E64155.DCL
MSM64158:	E64158.DCL

! The DCL file used differs for SID64K symbolic debugger and ASM64K cross-assembler. Please ensure to use the correct DCL file:

DCL file for SID64K:	**E**64152.DCL (first character of the file name is "E")
DCL file for ASM64K:	**M**64152.DCL (first character of the file name is "M")

Chapter 1

Overview

This chapter provides an overview of EASE64158 system configuration, describes the program development procedure with the EASE64158 system.

1-1. EASE64158 Emulator Configuration

The EASE64158 in-circuit emulator is configured from:

(1) Control system (EASE-LP2)
(2) POD64158 evaluation module
(3) ASM64K cross-assembler
(4) SID64K symbolic debugger

1-1-1. Control System (EASE-LP2)

The EASE-LP2 is a general-purpose control system for in-circuit emulators for Oki Electric's MSM64153 family of CMOS 4-bit microcontrollers. The EASE64158 in-circuit emulator is constructed by connecting the control system to a POD64158 evaluation module.

The internal configuration of the EASE-LP2 control system is as follows.

	• System controller	MC68HC000
☞1	• Code memory	64K x 24 bits (☞2)
	• Trace memory	8K steps x 64 bits
	• Cycle counters	32-bit binary counter x 1 counter
☞1	• Attribute memory	64K x 8 bits
☞1	• Instruction executed bit memory	64K x 1 bit
	• EPROM programmer	For 2764/128/256/512
	• RS232C ports	1 channel
	• System power supplies	1

☞ 1 Refer to each microcontroller's User's Manual for the maximum addresses of code memory, attribute memory, and instruction executed memory for the MSM64153 family.

☞ 2 Up to 32K x 8 bits among the 64K x 24 bits code memory of the EASE-LP2 will be used by the EASE64158 as a program RAM .

! The emulator handles the test data area of the MSM64153 family program as an unusable area.

1-1-2. POD64158 Evaluation Module

The EASE64158 in-circuit emulator for Oki Electric’s MSM64153 family of CMOS 4-bit microcontrollers is constructed by connecting the POD64158 evaluation module to an EASE-LP2. The MSM64E153, a dedicated evaluation chip developed especially for emulation, is used in the POD64158.

! The POD64158 can be operated stand-alone by mounting an EPROM with a user program in the EPROM socket. Refer to Section 2-2-6, “Starting the EASE64158 Emulator.”

! Refer to Appendix 7 regarding user cable connectors and peripheral circuits for the evaluation chip.

1-1-3. ASM64K Cross-Assembler

ASM64K is a cross-assembler developed for the OLMS-64K series. It is stored on a floppy disk that comes with the purchase of an EASE64158 development support system.

Source files constructed from OLMS-64K instruction mnemonics and directives are converted to object files with ASM64K. Object files (machine language files) generated this way are read and executed by SID64K, explained in the next section.

ASM64K can be used with host computers that satisfy the following conditions.

- The operating system is MS-DOS or PC-DOS version 3.1 or higher.
- A transient program area of at least 128 Kbytes is available.

For details about ASM64K, refer to the ASM64K Cross-Assembler User’s Manual.

1-1-4. SID64K Symbolic Debugger

The SID64K symbolic debugger is software that operates on a host computer interfaced to the EASE64158. The EASE64158 operates through this software. SID64K also supports symbolic debugging.

SID64K is stored on a floppy disk that comes with the purchase of an EASE64158 development support system.

SID64K can be used with host computers that satisfy the following conditions.

- The operating system is MS-DOS or PC-DOS version 3.1 or higher.
- A transient program area of at least 250 Kbytes is available.
- A channel for an RS232C interface.

1-1-5. System Configuration

The system is used in the following two configurations.

- **EASE-LP2 mode** in which the POD64158 is connected to the EASE-LP2 and a host computer.
- **POD mode** in which the POD64158 is used stand-alone.

Figure 1-1 and Figure 1-2 shows each system configuration.

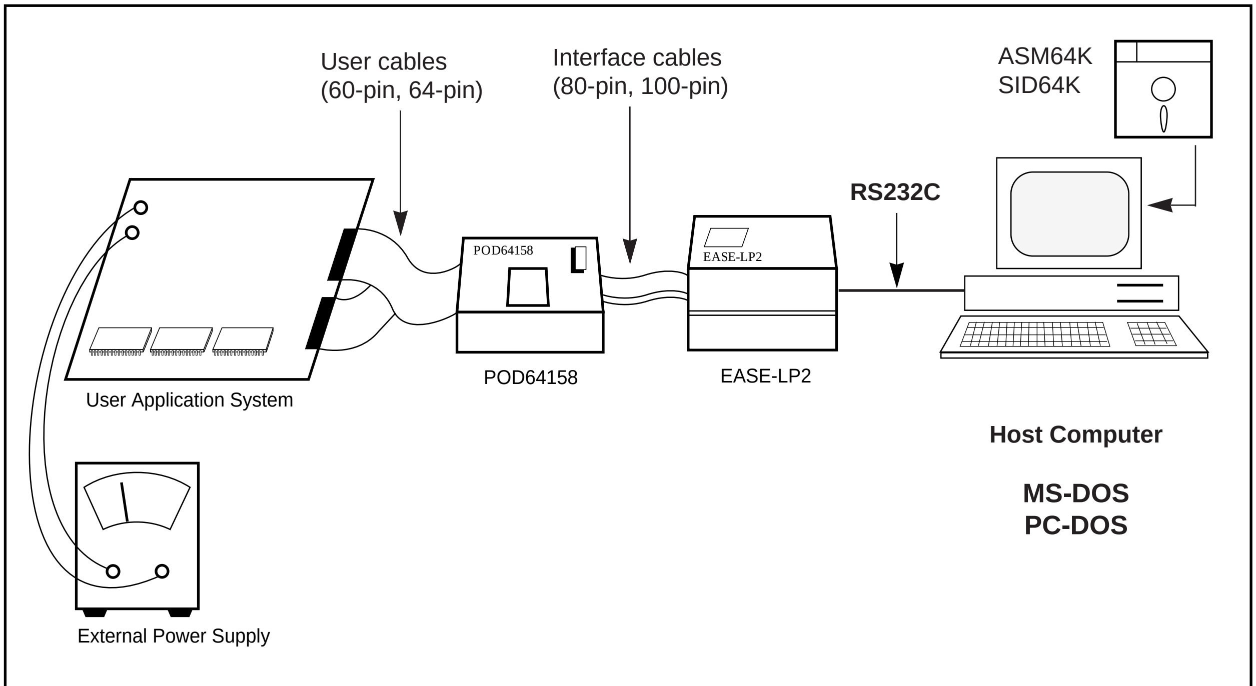


Figure 1-1. System Configuration in EASE-LP2 Mode

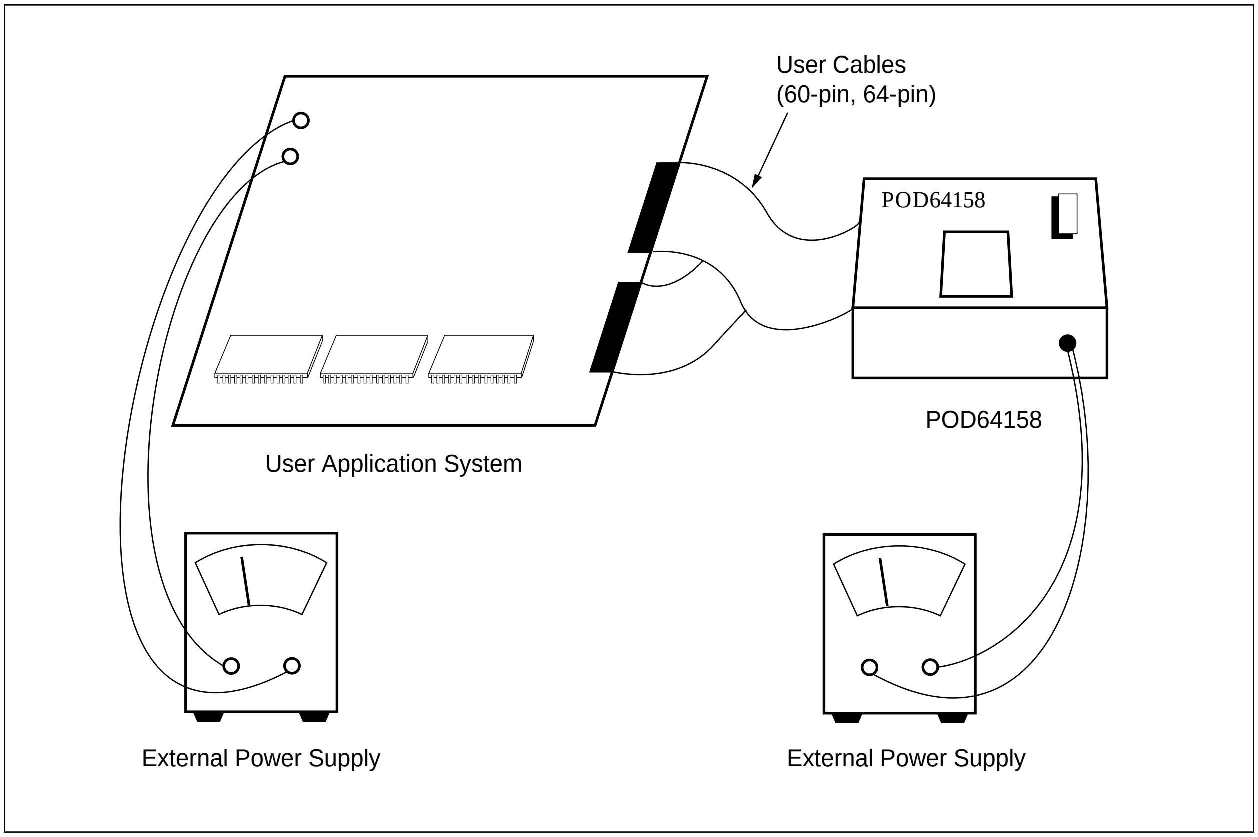


Figure 1-2. System Configuration in POD Mode

1-2. EASE64158 Parts and Functions

Each part of the EASE64158 and its function is summarized below.

1-2-1. Control System (EASE-LP2)

(1) **EPROM** programmer
Used to program the contents of the code memory to EPROM, or transfer the EPROM contents to the code memory.

(2) Indicators

POWER indicator: Lights when the EASE64158 is turned on using EASE-LP2 power switch.

RUN indicator: Lights when realtime emulation is executed (during continuous execution), and when the EPROM programmer is accessed.

ERROR indicator: Lights when the EASE64158 is not operating properly, or when an error is occured during operation. For details, refer to Appendix-5.

POWER DOWN indicator: Lights when the EASE64158 is in HALT mode during emulation (during continuous execution or during step execution).

POD indicator: Lights when the POD64158 is properly connected to the EASE-LP2.

(3) **RS232C** connector
A host computer is connected through the attached RS232C cable.

(4) **SW1**
Sets the baud rate of RS232C interface.

(5) **RESET** switch
Resets the EASE-LP2.

(6) **PROBE** cable connector
Connector for the attached probe cable to enable external break.

(7) Interface cable connectors (**CN1** and **CN2**)
The POD64158 evaluation module is connected through the attached 80-pin and 100-pin interface cables.

(8) AC power supply connector (**AC100**–**240**)
Connect the attached AC power supply cable. Note that EASE-LP2 rated power voltage is AC100–240V.

(9) **POWER** switch
Turns the EASE64158 ON/OFF.

1-2-2. POD64158 Evaluation Module

(1) **EPROM** socket
Mount the EPROM contains user program.

(2) Indicators

POWER indicator:	Lights when POWER switch is ON.
POWER DOWN indicator:	Lights when the EASE64158 is in HALT mode during emulation (during continuous execution or during step execution).

(3) **SW1**
Selects the type of EPROM mounted on EPROM socket.

(4) **SW2**
Selects the EASE-LP2 or POD operation mode.

(5) **SW3**
Switch the operation voltage of the MSM64E153 evaluation chip.

(6) **SW4**
Switch the operaion clock supplying method.

(7) **MSM64E153** evaluation chip socket
A socket to mount the MSM64E153 evaluation chip.

(8) **A/D board**
Terminal of A/D converter, to which resistors or capacitors are connected.

(9) Crystal board cover (**X'tal**)
A crystal oscillator board is inside.

(10) Interface cable connectors (**CN1** and **CN2**)
When the POD64158 is used in EASE-LP mode, connect the attached 80-pin and 100-pin interface cables to connect it to the EASE-LP2.

(11) User connectors (**USRCN1**and **USRCN2**)
Connect the user application system through attached 60-pin and 64-pin user cables.

(12) DC power jack (**DC JACK**)
When the POD64158 is used in POD mode, connect the external power supply.
DC5V must be supplied through attached DC power supply cable. Do not mix up DC polarity when connecting the DC power cable to the external power supply.

1-3. Program Development With EASE64158

1-3-1. General Program Development and EASE64158

Figure 1-3 shows the general flow of program development (☞1).

First, one decides on the functions of the product to be developed, and evaluates which hardware and software should be designed to implement them. Specific considerations include which MCU to use, how to allocate MCU interrupts, how much ROM and RAM to add, etc. This is called the *functional design process*.

Next is the *specification design process*. Here the functions to be implemented are evaluated in detail, and the methods to use those functions in the final product are decided. Specifically, commands are decided upon and a command input specification is written. The specification generated by this process is usually called the functional specification.

The process of creating a program based on the functional specification is called the *program design process*. Algorithms, flowcharts, and a program specification are created. This process can also include coding (source program creation) and assembly. In other words, ASM64K is used in this process.

Next is the *debug process*. This is the process for which the EASE64158 especially excels (☞2). An object file created in the program design process is downloaded to the EASE64158, and by using the various functions of the EASE64158 emulator, program bug analysis, fixing, and testing are performed.

The last position of the overall program development process is occupied by the *testing process*. The complete program from the debug process is operated in the actual product, and operation according to the functional specification is verified with test programs, etc. If there are bugs in the operation, then the flow from the program design process on is repeated until there are no more bugs.

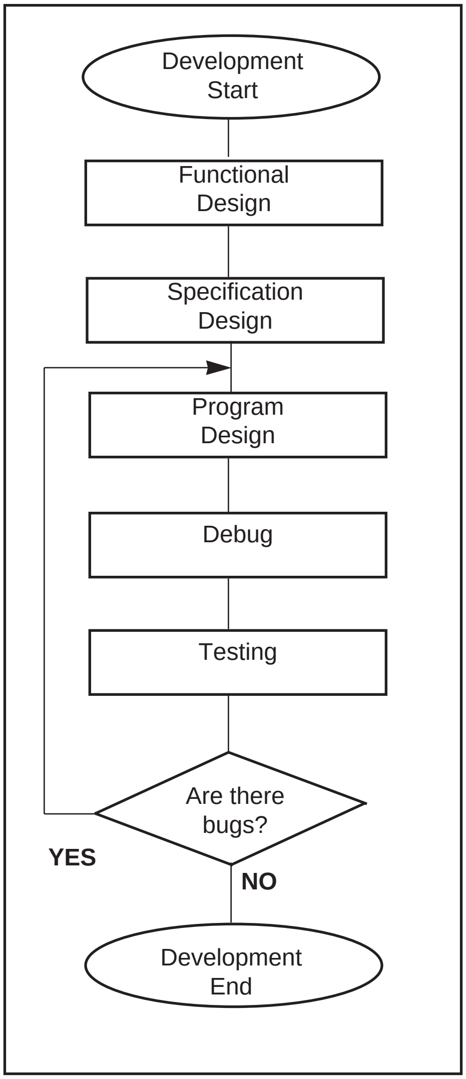


Figure 1-3. General Flow of Program Development

☞ 1 The general flow and terminology given here are typical, but other documents and manuals will have different expressions.

☞ 2 Refer to Chapter 2, “EASE64158 Emulator,” for details about the various function of the EASE64158 emulator.

1-3-2. From Source File To Object File

In order to perform debugging with the EASE64158 emulator, an object file for downloaded to the EASE64158 must be generated (☞3,4).

Figure 1-4 shows the process of generating an object file from a source program file coded in assembly language (hereafter called a source file).

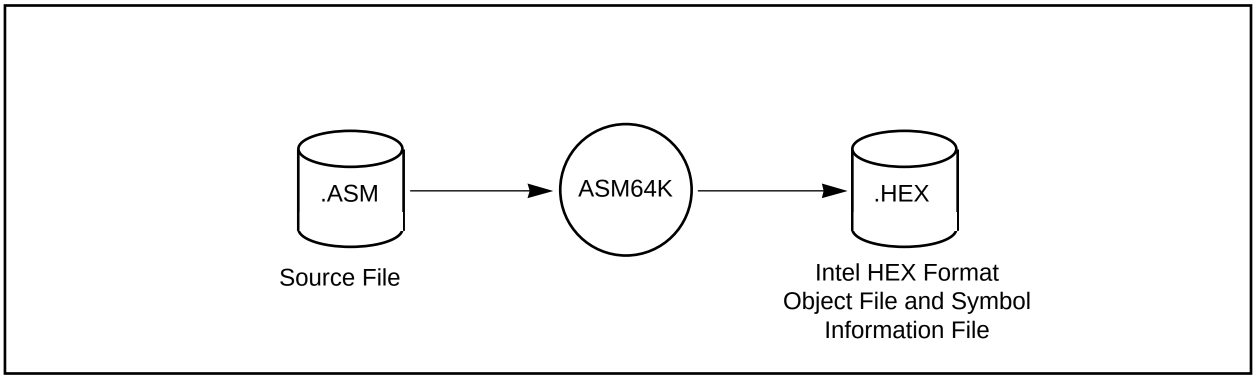


Figure 1-4. Process of Generating Object Files From Source Files

In the above figure, circles indicate operation of the ASM64K cross-assembler program, while cylinders indicate files generated by programs.

Object files that the EASE64158 emulator can handle are Intel HEX format object files that include symbol information, as shown in Figure 1-4.

☞ 3

Downloading means storing the contents of an object file in EASE64158 code memory with the SID64K **LOD** command. Refer to Section 3-1-2-4, “Load/Save/Verify Commands,” for details on the **LOD** command.

☞ 4

Object files in this document refer to Intel HEX format object files that include symbol information which the EASE64158 emulator can handle.

1-3-3. Files Usable With the EASE64158 Emulator

The files usable with the EASE64158 emulator are files generated by ASM64K, as explained in the previous section. This section describes these files.

(1) Files generated by ASM64K

These are object files generated by ASM64K from source files built from OLMS-64K mnemonics and various directives. These files include symbol information. Therefore, to perform symbolic debugging, loading must be done with the SID64K symbolic debugger’s **LOD** command with **/S** option (☞5).

☞ 5

Refer to Section 3-1-2-3, “Load/Save/Verify Commands,” about the **/S** option specification of the **LOD** command. Symbol information is supported by the ASM64K assembler version 1.00 and later versions. For details, refer to the ASM64K Cross-Assembler User’s Manual.

Chapter 2

EASE64158 Emulator

This chapter explains the actual use of the EASE64158 emulation kit and the SID64K symbolic debugger in detail.

In this chapter...

- Section 2-1 gives an overview of each group of functions that can be used with the EASE64158 emulation kit and the SID64K symbolic debugger
- Section 2-2 explains how to start the EASE64158. EASE64158 dipswitch settings (to set the communications mode with the host computer, etc.) are also explained in this section.
- Section 2-3 explains in detail the actual use of SID64K debugger commands with the EASE64158.
  - Section 2-3-1 describes the general input format of debugger commands and lists all debugger commands by function. This list also gives a reference page for each command, so it is convenient for use as a command index.
  - Section 2-3-2 gives a general explanation of symbolic input.
  - Sections 2-3-3 and 2-3-4 explain the history function and special-purpose keys respectively. These are provided to support efficient input of debugger commands.

2-1. EASE64158 Functions

2-1-1. Overview

Section 1-2 explained the program development process with the microcontrollers of the MSM64153 family. This section gives an overview of the actual emulator functions used to debug prototype programs created by that process.

The most basic function of the emulator is to read and execute a program (an Intel HEX format object code plus symbol information file generated by ASM64K). Here “execute” means to execute a program under the same electrical characteristics and at the same speed as the same volume-production microcontroller in the MSM64153 family. This is known as *emulation*, as distinguished from program simulation with large computers.

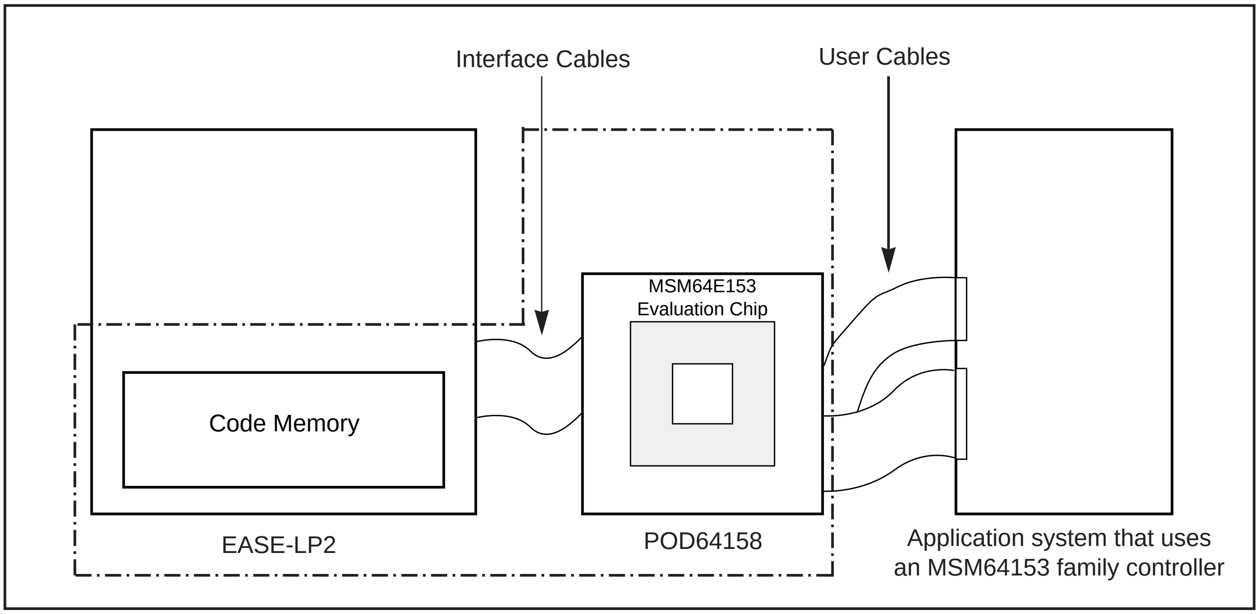


Figure 2-1

The volume-production MSM64153 family microcontrollers have mask ROM on-chip, but once mask ROM has been written it cannot be changed. However, program during the development stage is difficult to debug unless it is stored in rewritable memory (RAM).

Thus the EASE64158 has in internal 64K x 24-bit program storage RAM. This RAM is called *code memory* (☞ 1). Refer to Figure 2-1 on the previous page.

EASE64158 executes programs in this code memory instead of mask ROM (☞2). When the user application system is being produced in volume, it will be mounted with an MSM64153 family microcontroller, but at the debug stage it is replaced with a connector in the user application system. This connector is attached to an EASE64158 user cable (Refer to Figure 2-1).

Within the EASE64158 (strictly speaking, within the POD64158) is a special device, designated MSM64E153. The MSM64E153 has the same CPU circuit and the same external pins as MSM64153 family microcontrollers. It differs from MSM64153 family microcontrollers in that it has no internal mask ROM, but it does have some special control circuitry and control pins.

These additional circuits and pins are used to control execution of programs and reading of internal memory, registers, and flags. The MSM64E153 can read and execute the contents of code memory instead of mask ROM. In other words, with the MSM64E153 one can realize a system in which mask ROM is replaced by code memory, but its pins and electrical characteristics are identical to the MSM64153 family microcontrollers (☞3). The MSM64E153 is often called the *evaluation chip* in this manual.

The MSM64E153 evaluation chip's external pins include the same pins as volume-production MSM64153 family microcontrollers. These are connected to the connector on the user application system through the user cables. A/D converter pins are located on the A/D board.

As a result, the user application system sees the end of the user cable connector as identical to a MSM64153 family microcontroller (see Figure 2-1).

☞ 1

Refer to each MSM64153 family microcontroller's User's Manual regarding code memory addresses of the EASE64158. Within code memory, up to 32K x 8 bits can be used as RAM for program storage.

☞ 2

The POD64158 has an EPROM socket for code memory. If the POD64158 is used standalone, then the EPROM in this socket will be allocated to the program area. However, if used as an EASE64158, then do not use the EPROM socket.

☞ 3

The user cable connector’s pins cannot be said to be completely the same electrically as MSM64153 family microcontroller pins. It is very minor, but the leads on the user cable will add some resistance and floating capacitance. Port pins being traced also have one CMOS comparator load, and the EXT CLK and USER RESET pins have one CMOS load. These loads are tiny enough that they will be no problem to most systems, but note that problems can occur on systems with subtle electrical characteristics.

!

For information about the relation between the evaluation chip and the user cables, refer to Section 2-2-1, “Setting Operating Frequency,” and Section 4-1-2, “Resets,” and Appendix 7, “User Cable Peripheral Circuit.”

That the basic function of the emulator is to read and execute programs was already explained, but effective debugging is not possible with just simple execution. For example, one must be able to start and stop program execution at specified addresses. One needs to display and change the states of data memory (internal RAM), registers, and flags after execution. Furthermore, instead of just stopping execution at a specified address, one needs the ability to set complex conditions for stopping after a specified time has elapsed or some address has been passed a specified number of times (pass count). To meet these needs, EASE64158 has many functions beyond its basic one. These features are explained one by one in the following sections.

2-1-2. Changing the Target Chip

The EASE64158 is configured to each device of the MSM64153 family by setting the following items.

(a) Set the DCL file read when SID64K is invoked.

The DCL file defines symbol information needed to perform symbolic debugging, the code memory address range, and the data memory address range.

(b) Set the POD64158’s internal chip select dipswitch.

The POD64158's internal dipswitch 4 specifies which SFRs and data memory in the MSM64E153 are prohibited and which are permitted. It corresponds to the appropriate device in the MSM64153 family.

The dipswitches 2 and 3 specify which of the MSM64E153’s internal interrupt circuits are prohibited and which are permitted. They correspond to the appropriate device in the MSM64153 family.

The DCL file and dipswitches must all be set to the same device in the MSM64153 family being used. If the settings are mixed, then the EASE64158 will not operate properly.

Refer to Section 2-2-4, “Dipswitch Changes for Chip Select,” for setting the chip select dipswitches.

❑ *Reading the DCL file for the target chip* (☞1)

In order to start SID64K configured for the appropriate target chip, the chip-specific DCL file must be read. The DCL file read by SID64K is determined by power-on information (chip mode information) from the EASE64158. When the filename is determined, SID64K first searches for it in the *current directory*. If not found, then it searches the *directory which contains SID64K.EXE* and then the *directory specified by the DCL environment variable*. If still not found, then SID64K will not start.

☞ 1

Refer to Section 2-2-6, “Starting the EASE64158 Emulator,” for details about reading DCL files and about chip modes corresponding to DCL files.

2-1-3. Data Memory Space

The EASE64158 assigns the MSM64E153 evaluation chip’s internal memory and the SFR area to data memory space. Refer to each MSM64153 family microcontroller's User's Manual for detailed description.

2-1-4. Code Memory (Program Memory) Space

The EASE-LP2 has 64K x 24 bits as code memory space, but the EASE64158 used code memory in accordance with the devices of the MSM64153 family. Address range of the code memory is described in each MSM64153 family microcontroller's User's Manual.

Code memory size, attribute memory size, and instruction executed memory size can be expanded to 32K bytes (000–7FFFH) with the **EXPAND** command.

SEE **EXPAND**

! Note that the EASE64158 emulator handles the test data area in each MSM64153 family microcontroller's code memory (program memory) as unusable area.

2-1-5. Emulation Functions

The EASE64158 has two modes for its emulation functions (program execution functions).

(1) Single-step mode (**STP** command)

In this mode, program execution stops after each step (one instruction) is executed. After each instruction is executed, the state of the evaluation chip is read and displayed on the CRT. Single-step mode is realized with the **STP** command. The information to be displayed can be set with the **SSF** command.

SEE **STP, SSF**

(2) Realtime emulation mode (**G** command)

In this mode, program execution will continue until some specified break condition is satisfied or an **ESC** command is input. Realtime emulation mode is realized with the **G** command. Even during realtime emulation, the EASE64158 allows some of the debug commands to be input. For details, refer to Section 3-4-3, “Commands Usable During Emulation.”

SEE **G**

☞ 1 The emulation functions here are those of the EASE-LP2 mode. In the POD mode, in which the POD64158 operates standalone, only the continuous execution by the user program EPROM on the POD.

❑ *Operating Clock*

The operating clock of the EASE64158 can be selected from either a clock supplied by an internal oscillation circuit or a clock input from the user cable EXT CLK pin. Operating clock selection is performed by switching a dipswitch on the POD64158.

When the EASE64158 is shipped, it is set to operate using the clock supplied by its internal oscillation circuit. The internal oscillation circuit’s frequency is 32.768 kHz (typical). The internal oscillation frequency can be changed by changing the crystal oscillator on the X'tal board.

For details, refer to Section 2-2-1, “Setting Operating Clock Frequency.”

- The EASE64158 operating frequency is 32.768 kHz (typical). To use any other frequency, please contact Oki Electric’s engineering department.

- When changing the crystal oscillator, capacitors or resistors in the oscillation circuit might also be required to change in accordance with the manufacturer of the crystal oscillator or the oscillation frequency being used.

- Clock input selection is performed by switching a dipswitch 4 on the POD64158. Refer to Section 2-2-2, “EASE64158 Switch Settings.”

- However the MSM64153 family microcontrollers MSM64155 and MSM64158 can select CR oscillation circuit as the clock generator by using mask option, the EASE64158 cannot select the CR oscillation circuit as a clock generator. If this hinders your program development, please contact Oki Electric's engineering department.

2-1-6. Realtime Trace Functions

One EASE64158's principal functions is realtime tracing. Realtime tracing occurs during program execution under realtime emulation mode. It stores the executed addresses, the data and addresses in data memory used, and the states of evaluation chip port pins, registers, and flags in memory provided for tracing. The memory provided for tracing is called *trace memory*.

The EASE64158 has trace memory for 8K steps. It traces the following items.

Trace Contents
Program counter (PC) value
Data memory addresses
Data memory data
A register value
B register value
H (X) register value (☞1)
L (Y) register value (☞1)
Stack pointer (SP) value
State of any two ports among ports 0, 1, 2, 3, 4, 6, 7
MI flag value
Carry (C) flag
INT flag (☞2)
SKIP flag (☞2)

SEE ▷ **STF, DTM, DTP, RTP, CTO**

☞ 1 The **CTO** command selects whether the values of the H and L registers or X and Y registers are traced.

☞ 2 The INT flag indicates an interrupt transfer cycle. The SKIP flag indicates skip execution. Refer to Chapter 4, "EASE64158 Timing," for output timing of the INT flag and SKIP flag.

❏ *Controlling trace execution*

There are six ways to control realtime tracing.

(1) Free-running trace

Tracing is always performed during program execution.

(2) Trace on trace enable bits

Tracing is performed on particular portions of program memory specified with trace enable bits.

(3) Trace disable

Tracing is not performed during program execution.

(4) Trigger-based trace start/stop

Tracing starts when the trace start address is executed, and stops when the trace stop address is executed.

(5) Data match post-trace

Tracing starts when a probe or RAM value matches the specified value.

(6) Data match pre-trace

Tracing ends when a probe or RAM value matches the specified value.

Details of each tracing method are explained in Chapter 3, “SID64K Debugger Commands.” The following are related commands.

SEE **DTR, CTR, STT, DTT**

The address of trace memory written to is controlled by the *trace pointer*. The trace pointer is a 13-bit counter. It is incremented for each instruction execution in accordance with the control conditions for each way of realtime tracing (refer to Figure 2-2).

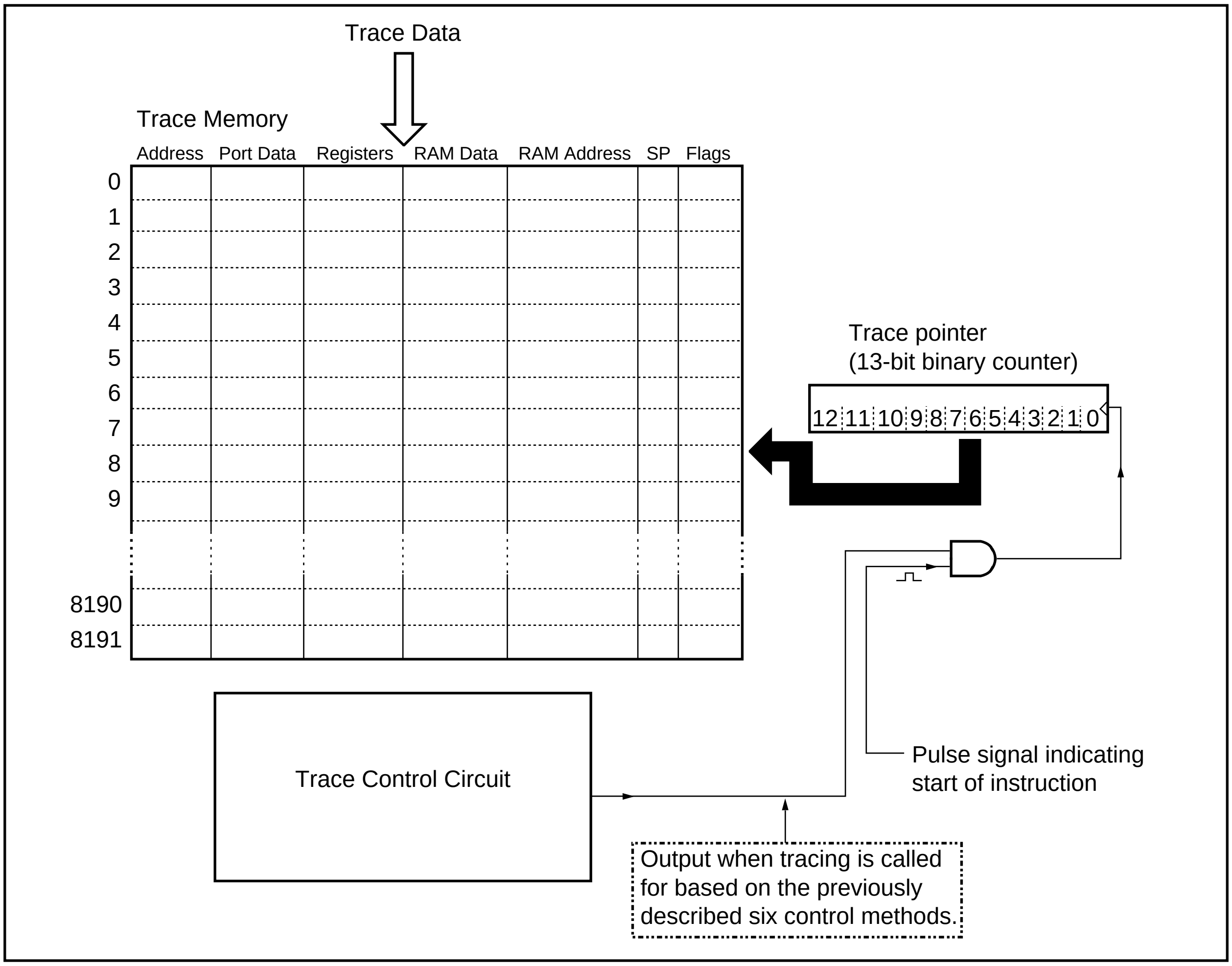


Figure 2-2. Trace Control Conceptual Diagram

The trace pointer's value indicates the address in trace memory to which data will be written. The trace pointer is incremented at the start of each instruction while the conditions of the previously described control methods are satisfied. As a result, the trace memory addresses written are updated one by one as trace data is stored at each.

The trace pointer is a 13-bit counter, so its value will be between 0 and 1FFFH (in decimal, 0 and 8191). When the trace pointer exceeds 1FFFH, it overflows and becomes 0. In other words, when traced data exceeds 8192 steps, it will be overwritten in order from the oldest data in trace memory.

2-1-7. Break Functions

The following methods for breaking program execution are available with the EASE64158.

(a) Breakpoint bit breaks

The EASE64158 has a 1-bit wide memory that corresponds 1-for-1 with the entire program memory address space (0-7FFFH). This memory is called *breakpoint bits memory* or *breakpoint bits*.

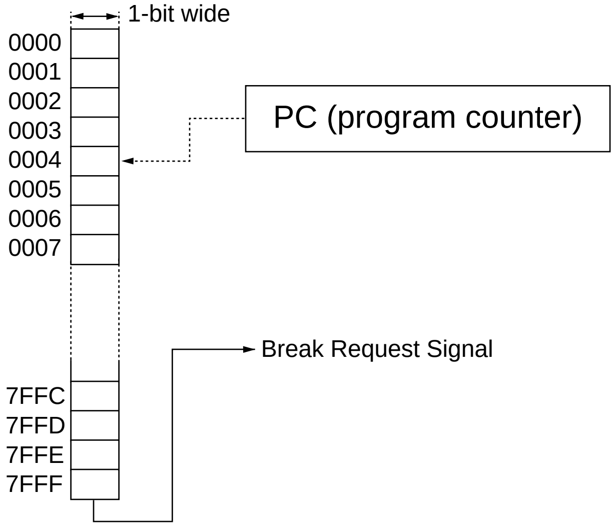


Figure 2-3. Breakpoint Bits Conceptual Diagram

Breakpoint bits can be set to 1 or 0 with the **CBP** (Change BreakPoint bit) command. During emulation execution, the breakpoint bit corresponding to each executed address is referenced, and if “1,” a break request signal is output (refer to Figure 2-3).

By using breakpoint bits, breakpoints can be set throughout the entire address space without a limit to their number. (In this manual breaks generated by breakpoint bits are called *breakpoint bit breaks* to clearly distinguish them from *address breaks*, which are generated by break addresses specified as break parameters of the **G** command.)

SEE **DBP, CBP, SBC, DBC**

(b) Trace full breaks

The EASE64158 can force a break using overflow of the trace pointer.

SEE **DTR, CTR, SBC, DBC**

(c) Cycle counter overflow breaks

The EASE64158 has a 32-bit counter that increments every machine cycle (called the *cycle counter*). The overflow of the cycle counter can be used as a break condition.

SEE **SCT, RCT, TIME, CCC, DCC, SBC, DBC**

(d) Address pass counter overflow breaks

The EASE64158 has four 16-bit address pass counters that are incremented when the program at a specified address is executed. Of these address pass counters, the overflow of counter 0 (C0) can be used as a break condition.

SEE **CAP, DAP, SBC, DBC**

(e) Break on execution of power-down instruction

This break occurs when an instruction is executed that sets to “1” bit 0 (HLT) of the Halt Mode Register (HALT), an SFR of all microcontrollers in the MSM64153 family. In other words, it occurs when a microcontroller in the MSM64153 family enters power-down mode.

SEE **SBC, DBC**

(f) **ESC** command breaks

Input an **ESC** command to forcibly stop **G** command execution (realtime emulation).

SEE **ESC**

(g) Breaks specified during **G** command input

- Break at specified address (with pass count)
- Break at specified address (with pass sequence)
- Break when specified data matches data at a specified address in data memory
- Break when specified data matches probe data

SEE **G**

(h) N area access break

The EASE64158 will forcibly break when it accesses an area that exceeds the maximum address for its respective chip modes.

(i) External breaks

An external break will occur when the signal on the external break pin of the probe cable transitions from “L” to “H.”

SEE **SBC, DBC**

❑ *Break request mask function*

The break conditions explained is (a)-(d) and (i) above can each be masked. As shown in Figure 2-5, masking of break conditions is performed using a register called the *break condition register*.

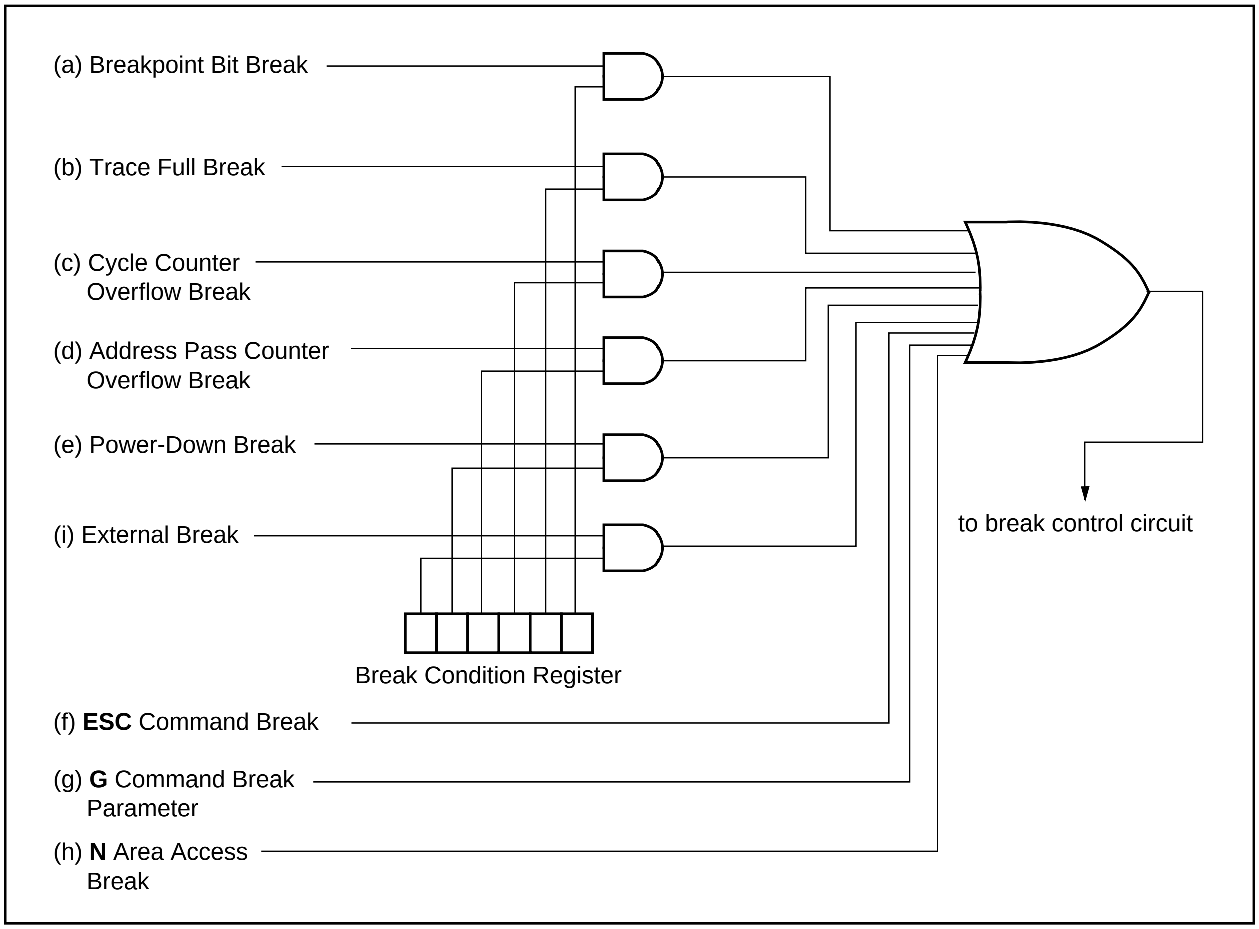

(a) Breakpoint Bit Break
(b) Trace Full Break
(c) Cycle Counter Overflow Break
(d) Address Pass Counter Overflow Break
(e) Power-Down Break
(i) External Break
Break Condition Register
(f) **ESC** Command Break
(g) **G** Command Break Parameter
(h) **N** Area Access Break
to break control circuit

Figure 2-4. Break Masking

!

The order of bits in the break condition register of Figure 2-4 does not necessarily match the order of bits in the actual register.

2-1-8. Performance/Coverage Functions

The EASE64158 has the following performance/coverage functions.

(a) Check for program areas not yet passed

The EASE64158 has a 32K x 1-bit *instruction executed bits memory* (or *IE bit memory*) that corresponds 1-for-1 to code memory's 32K addresses (0H-7FFFH). Whenever an instruction is executed, the contents of IE bit memory at the address corresponding to the instruction will be set to "1." By examining the contents of IE bit memory, one can see which program areas have not been passed (or debugged).

SEE **CIE, DIE**

(b) Measuring elapsed time

Elapsed execution time for a specified block can be measured by using the EASE64158 internal 32-bit cycle counter (CC).

SEE **CCC, DCC, SCT, RCT, DCT, TIME**

(c) Measuring execution passes

The number of times up to four specified addresses are executed can be measured by using the EASE64158's four 16-bit address pass counters (AP).

SEE **CAP, DAP**

2-1-9. Probe Cable Functions

The EASE64158 utilizes a probe cable with nine probe pins. The probe cable is connected to the EASE-LP2 probe connector. Refer to Appendix 8, “Probe Cable Configuration.”

The probe cable provides the following functions.

(a) Probe input, bits 0-7 (pins P1-P8)

❑ Data match break

Break when the probe value matches a specified value.

SEE ▷ **G**

For details, refer to Section 2-1-7, “Break Functions.”

❑ Data match post-trace

Tracing starts when the probe value matches a specified value.

❑ Data match pre-trace

Tracing ends when the probe value matches a specified value.

SEE ▷ **STT, DTT**

For details, refer to Section 2-1-6, “Realtime Trace Functions.”

(b) External break signal input (pin P9)

❑ External break

Break when the input signal on this pin transitions from”L” to “H.”

SEE ▷ **SBC, DBC**

For details, refer to Section 2-1-7, “Break Functions.”

! The probe cable can be used only when the MSM64E153 evaluation chip operating voltage is 3.0 V. It cannot be used when the voltage is 1.5 V.

2-1-10. EPROM Programmer

The EASE64158 has an internal EPROM programmer (EPROM writer). By using the EPROM programmer, EPROM contents can be transferred to code memory, and contents of a code memory area can be written to EPROM (☞1).

The types of EPROM that the EPROM programmer can write are as follows:

2764, 27128, 27256, 27512, 27C64, 27C128, 27C256, 27C512

SEE **TPR, VPR, PPR, TYPE**

! **DO NOT USE THE EPROM PROGRAMMER FOR PURPOSES OTHER THAN DEBUGGING PROGRAMS. IF RELIABILITY IN WRITE CHARACTERISTICS IS NECESSARY, THEN USE AN EPROM PROGRAMMER DESIGNED FOR THAT PURPOSE.**

☞ 1 Refer to Appendix 9, “Mounting EASE-LP2 EPROMs,” for information about how to handle EPROMs.

2-1-11. Symbolic Debugging Functions

The SID64K debugger supports symbolic debugging functions. These functions allow symbols to be input in addition to numbers as address and data input to all debugger commands, and as instruction operands within the **ASM** command.

Symbols defined by labels or assembler directives within the **ASM** command can also be used as command line input or assemble command input even after the defining assemble command terminates. Operators are permitted on input lines, so expressions constructed from symbols and operators can also be input.

SEE Section 2-3-2, “Symbolic Input.”

2-1-12. Assemble Command and Disassemble Command

Most line assemblers (assemble command) that come with emulator systems are designed to perform minimum necessary patches (modifications to programs). Normally they permit only instruction mnemonics and absolute addresses. However, the line assembler of SID64K alone is more powerful, providing nearly all the functionality of a standalone assembler. Its principal functions are as follows.

- Memory space can be coded as two logical segments: code segment, and data segment.
- The **ORG**, **EQU**, **SET**, **CODE**, **DATA**, **CSEG**, **DSEG**, **DB**, **DW**, **DS**, **NSE**, **END** and other directives can be used exactly as they are with ASM64K. Comment can also be input the same as they are in ASM64K.
- The full set of **C** language operators is supported.
- Because it is a complete 2-pass assembler, forward referenced labels can be used. Also, all symbols in a loaded program can be referenced. All symbols defined within the assemble command can be referenced on any command line.
- Up to 100 assembler lines can be input. When 100 lines have been input, an **END** will automatically be appended.
- By saving the code input with an assemble command to a file with the **LIST** command, the code can easily become a source file.

Furthermore, the disassemble command does just display simple mnemonics. If a symbol with the code segment attribute exists for an address being displayed, then that address will be displayed as a label. If a symbol exists for an address in an operand, then the operand will be displayed as that symbol, and its absolute address will be displayed as a comment. The disassemble command tries to create a display as close to a source file as possible.

SEE ▷ **ASM, DASM** commands (see details of Chapter 5, “Assemble Command”)

2-2. EASE64158 Emulator Initialization

2-2-1. Setting Operating Frequency

As explained in Section 1-5, the EASE64158 operates with the 32.768-kHz (typical) clock supplied from the POD64158's internal oscillation circuit when it is shipped. Oki Electric normally recommends that the EASE64158 be used as it is with this setting.

The clock setting can be changed with the following method. To change to an operating frequency other than 32.768-kHz, please contact Oki Electric's engineering department.

To change the clock setting

- Change the oscillation clock of the crystal board on the POD64158.

Selection of the clock from the POD64158 internal clock or the EXT•CLK pin of the user connector 2 is performed with the dipswitch 4 (SW4) of the POD64158.

The POD64158 crystal board, and the EXT•CLK pin of the user connector 2, are explained next.

(1) Crystal Board

The crystal board is mounted inside the X'tal board cover in the rear side of the POD64158. It generates a 32.768-kHz (typical) clock. Remove the X'tal board cover when disassembling/assembling the crystal board.

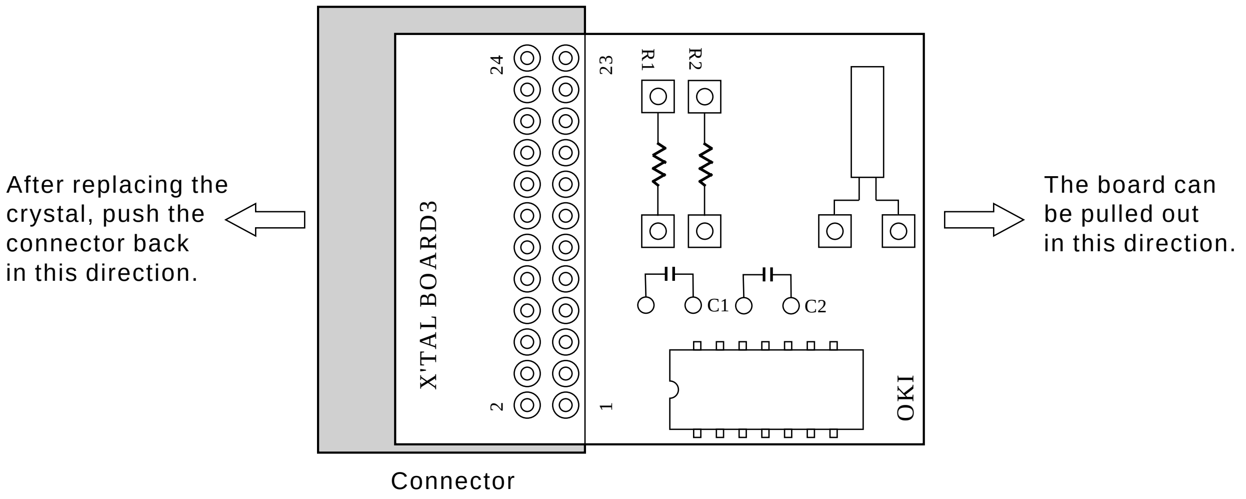

After replacing the crystal, push the connector back in this direction.
X'TAL BOARD3
24
2
23
1
R1
R2
C1
C2
OKI
Connector
The board can be pulled out in this direction.

Figure 2-5. Crystal Board

! When the crystal oscillator on the crystal board has been changed, always check that it is oscillating correctly. Depending on the crystal's manufacturer and type, it might not oscilate.

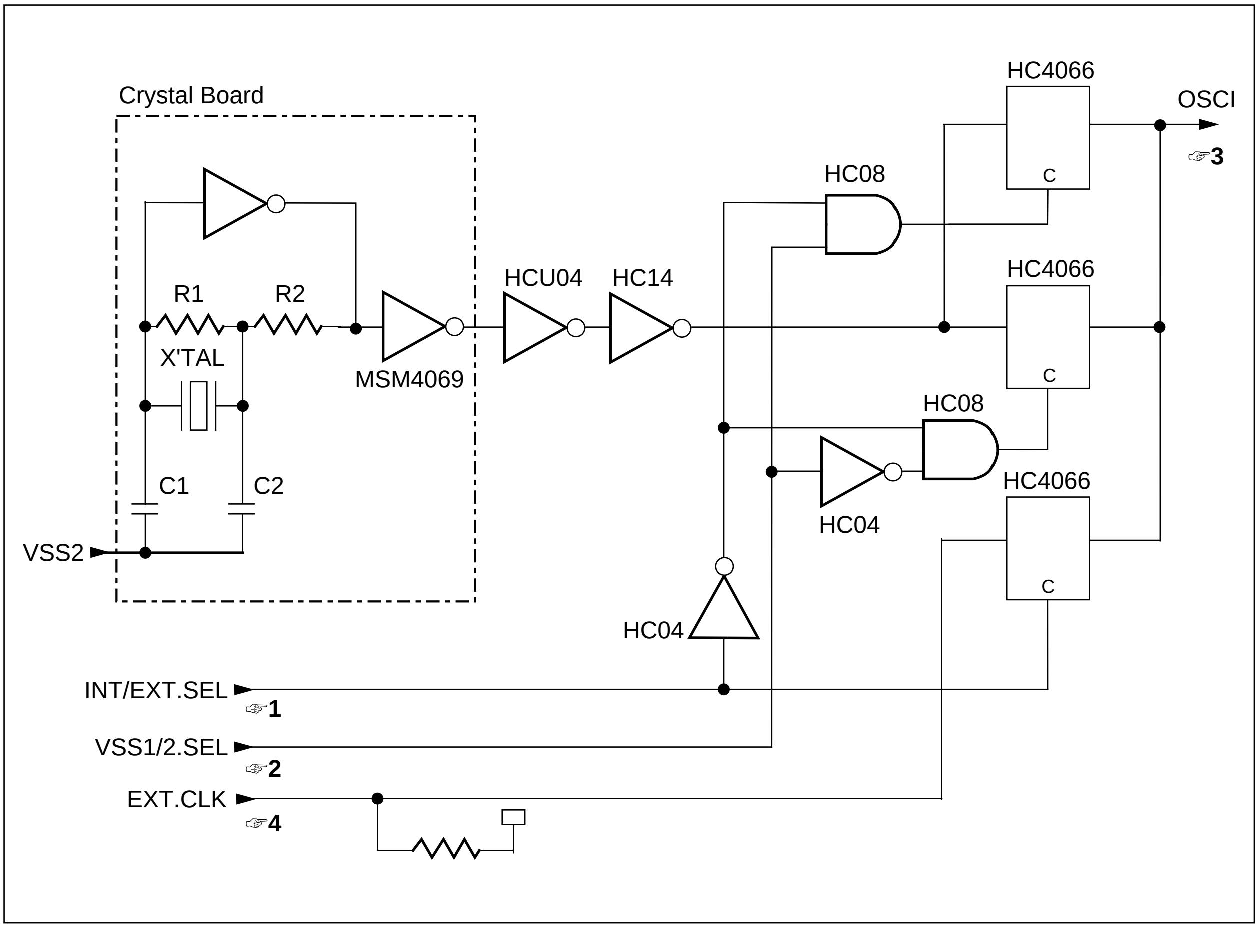


Figure 2-6. Crystal and Oscillation Circuit

☞ 1

This signal selects whether the EASE64158 operating clock is supplied from the POD64158 crystal board or the user cable’s EXT CLK pin. The selection of this signal is performed with POD64158 dipswitch 4 (SW4). For details, refer to Section 2-2-2, “EASE64158 Switch Settings.”

☞ 2

This signal switches the operating voltage of the MSM64E153 evaluation chip. The selection of this signal is performed with POD64158 dipswitch 3 (SW3). For details, refer to Section 2-2-2, "EASE64158 Switch Settings."

☞ 3

The MSM64E153 evaluation chip operates using this clock.

☞ 4

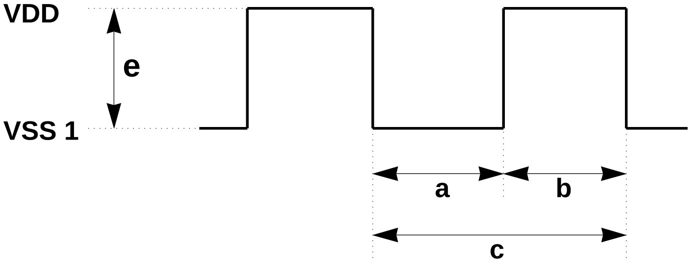
This is connected to the EXT•CLK pin of the user connector 2.

However the MSM64153 family microcontrollers MSM64155 and MSM64158 can select CR oscillation circuit as the clock generator by using mask option, the EASE64158 cannot select the CR oscillation circuit as a clock generator. If this hinders your program development, please contact Oki Electric's engineering department.

(2) <u>User cable EXT CLK pin input</u>

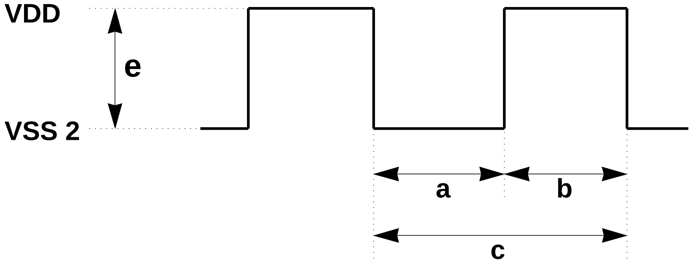
The emulator can be made to operate with an oscillating clock input on the EXT•CLK pin (pin 46) of the user connector 2. Use a clock like that shown below.

(1) <u>When evaluation chip operating voltage is 1.5 V</u>



Duty ratio	a:b = 1:1
Frequency	c = operating frequency 32.768 kHz (typical)
Voltage	e = 1.5 V (±5%)

(2) <u>When evaluation chip operating voltage is 3.0 V</u>



Duty ratio	a:b = 1:1
Frequency	c = operating frequency 32.768 kHz (typical)
Voltage	e = 3.0 V (±5%)

The input clock uses the pulse generator's output clock and the user application system oscillation circuit's clock. As shown in Figure 2-6, the EXT•CLK pin clock is input to an HC4066, so match its impedance to the HC4066.

The clock input clock to the EXT•CLK pin should be square-wave. Note that the operating stability is not guaranteed when sign-wave clock is used.

The input clock is not only supplied to the MSM64E153 evaluation chip, but also used for timing control inside the MSM64158 emulator.

The EASE64158 internal circuits always operate using this input clock during user program execution as well as the EASE64158 is waiting for a command input. So, interrupt of the clock input or an extraordinarilly distorted wave-form of the clock may cause abnormal operation or hung-up of the EASE64158.

2-2-2. EASE64158 Switch Settings

(1) EASE-LP2

There is an 8-bit dipswitch toward the top of the left panel of the EASE-LP2, labeled SW1 (refer to Figure 2-7). Each of the switch is explained below.

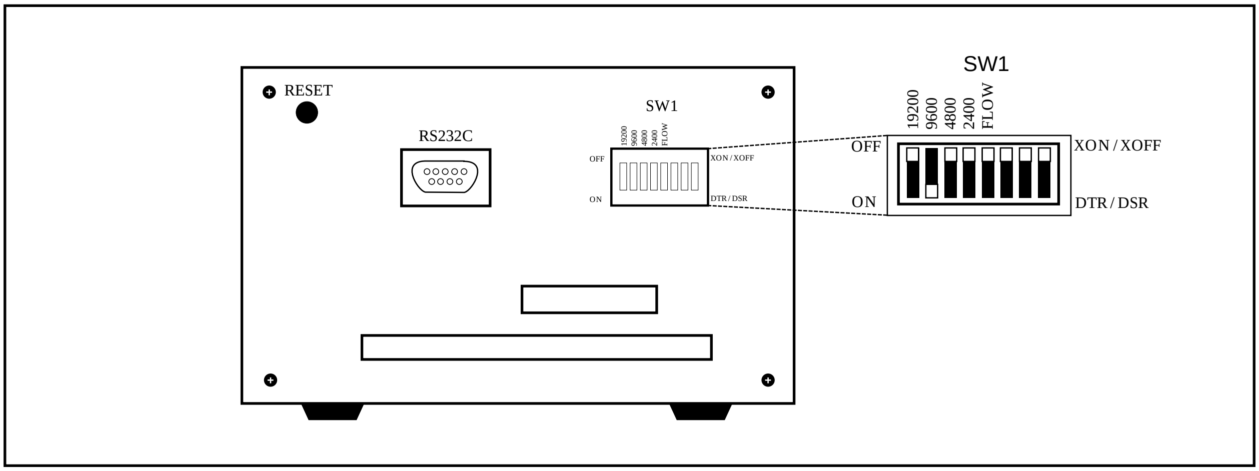


Figure 2-7. EASE-LP2 Dipswitches

❑ SW1

This dipswitch sets the EASE-LP2 interface parameters with the host computer. Each switch should be set appropriately.

- **19200 - 2400 (switches 1-4)**

These switches set the baud rate of the RS232C interface. Use them to match the EASE-LP2 baud rate with that of the host computer. When the EASE-LP2 is shipped, it is set to 9600 bps.

The baud rate can be set to 2400-19200 bps using switches 1-4 of DIP1. Table 2-1 shows the baud rate switch settings.

Table 2-1. Baud Rate Switch Settings

		19200	9600	4800	2400
SW1-1	19200	ON	OFF	OFF	OFF
SW1-2	9600	OFF	ON	OFF	OFF
SW1-3	4800	OFF	OFF	ON	OFF
SW1-4	2400	OFF	OFF	OFF	ON

- **Flow control (switch 5)**

This switch sets the flow control of the RS232C interface. Use the switch 5 to set the flow control to XON/XOFF control or DTR/DSR control. When the EASE-LP2 is shipped, it is set to XON/XOFF. Match the EASE-LP2 flow control with that of the host computer. Table 2-2 shows the flow control setting.

Table 2-2. Flow Control Switch Settings

	XON/XOFF	DTR/DSR
SW1-5	OFF	ON

Other than baud rate and flow control, the EASE-LP2's RS232C parameters are set as follows.

- **Other parameters**

° Transfer format	8 bits, 1 stop bit, no parity
° Other	Asynchronous, baud rate factor x 16

The above parameters on the host computer side must match those of the EASE-LP2, except for the stop bit (☞3).

! The INT232C program, described later, does not support 19200 bps with IBM-PC personal computers. If you are using it, set the baud rate to 9600-2400 bps. Refer to Section 2-2-6, "EASE64158 Emulator Initialization."

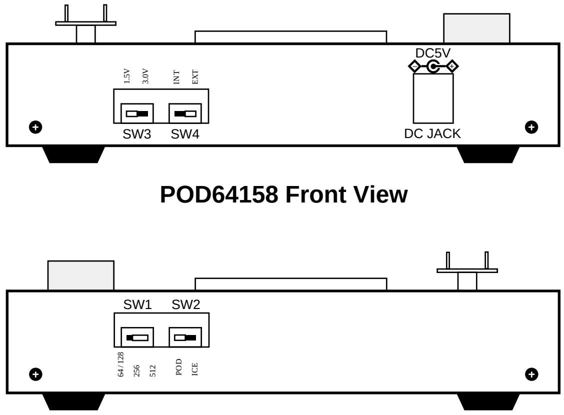
! In Table 2-1 and Table 2-2, ON means to flip the switch up, and OFF means to flip it down.

☞ 3 Oki if800 series computers are set using the **SWITCH** command.
PC9801 series computers are set using the **SPEED** command.
IBM-PC computer uses the INT232C program (described in Section 2-2-5).

! With the if800 series after changing parameters with the **SWITCH** command, if the if800 reset button is pushed once more to boot up the computer again, then be sure to note that the RS232C parameters will not be set correctly.

(2) POD64158

There are four dipswitches on the rear panel of the POD64158, labeled SW1, SW2, SW3, and SW4 (refer to Figure 2-8).



POD64158 Front View

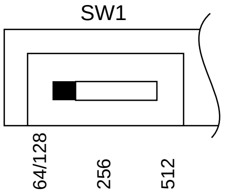
POD64158 Rear View

Figure 2-8. POD64158 Dipswitch

Each of the switches is explained below.

❑ **SW1**

This switch selects the type of EPROM that will be mounted in the POD64158's EPROM socket. The switch should be set appropriately according to the EPROM being used as shown below. Refer to Table 2-3 for the area written in each type of EPROM.



- **For 2764, 27C64, 27128, and 27C128 EPROM**
  Slide the SW1 to 64/128.

- **For 27256 and 27C256 EPROM**
  Slide the SW1 to 256.

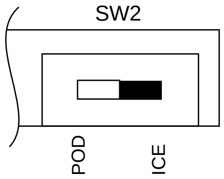
- **For 27512 and 27C512 EPROM**
  Slide the SW1 to 512.

Table 2-3. EPROM Write Area

EPROM type	Write area	SW1 setting
27512, 27C512	0000–7FFFH	512
27256, 27C256	0000–7FFFH	256
27128, 27C128	0000–3FFFH	64/128
2764, 27C64	0000–1FFFH	64/128

❑ SW2

This switch determines whether the POD64158 will be used in standalone mode (POD mode) or connected with the EASE-LP2 (EASE-LP2 mode). Set it as shown below.

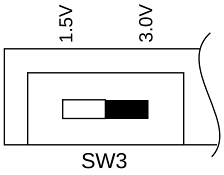


- **When used in EASE-LP2 mode**
  Set SW2 to ICE.

- **When used in POD mode**
  Set SW2 to POD.

❑ SW3

This switch determines the operating voltage of the MSM64E153 evaluation chip. Set it as shown below.



- When the operating voltage will be 1.5V
  Set SW3 to 1.5V.

- When the operating voltage will be 3.0 V
  Set SW3 to 3.0V.

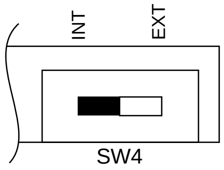
!

The MSM64E153 evaluation chip comes in two operating voltage versions: a 1.5-V version and a 3.0-V version. The 1.5-V operating voltage evaluation chip is labeled “MSM64E153-1.5V” on its top surface, while the 3.0-V operating voltage evaluation chip is labeled “MSM64E153-3.0V” on its top surface.

IF THE WRONG OPERATING VOLTAGE FOR THE EVALUATION CHIP IS SET, THE CHIP COULD BE DAMAGED.

❑ SW4

This switch determines whether the EASE64158 operating clock will be supplied from the POD64158’s internal oscillation circuit or the EXT•CLK pin of the user connector 2. Set it as shown below.



- When it will be supplied from the internal oscillation circuit
  Set SW4 to INT.

- When it will be supplied from the EXT•CLK pin
  Set SW4 to EXT.

Refer to Section 2-2-1. "Operating Frequency Setting" for details on the operating frequency.

2-2-3. Confirming EASE-LP2 Power Supply Voltage

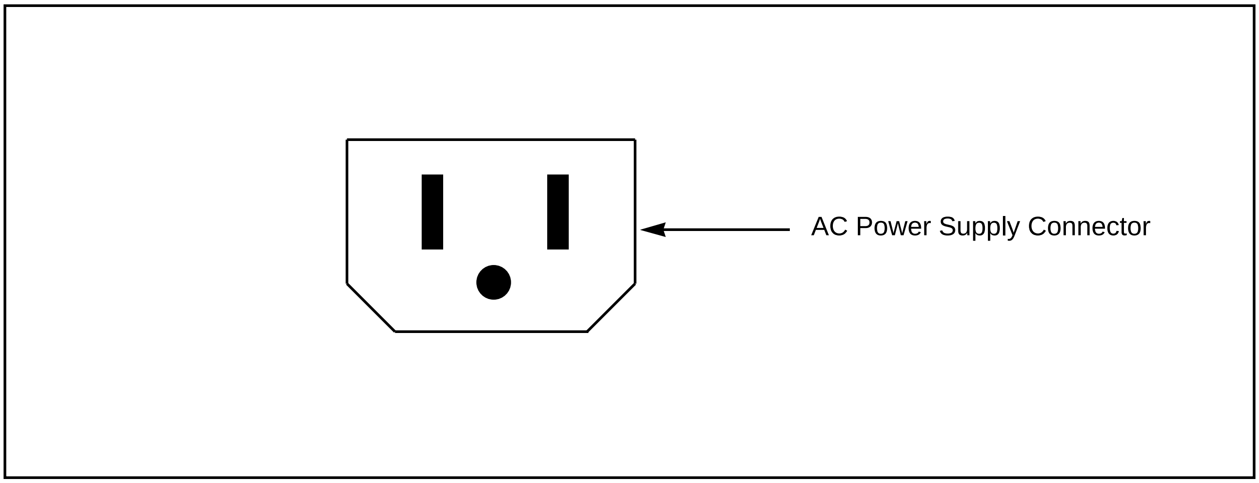
The power supply for POD64158 differs in EASE-LP2 mode in which the POD64158 will be connected to the EASE-LP2, and in POD mode in which the POD64158 will be used standalone.

(1) EASE-LP2 mode

In EASE-LP2 mode, both of the EASE-LP2 and the POD64158 will operate by the EASE-LP2 internal switching power supply circuit that uses normal household power. The switching power supply circuit automatically switches between the AC 100–240 V range.

EASE-LP2 Switching Power Supply Specifications

Input voltage	AC 100–240 V
Frequency and phase	47–63 Hz, single-phase



! **ABSOLUTELY DO NOT USE A VOLTAGE OTHER THAN AC 100–240 V. DOING SO COULD CAUSE A FIRE.**

(2) POD mode

In the POD mode, the POD64158 must be supplied from an external DC power supply using attached DC power supply cable. The DC power supply must conform to at least 5 V, 1 A. Connect the red plug of the DC power supply cable to + side of the DC power supply, black to – side.

! **ABSOLUTELY DO NOT MIX UP THE POLARITY OF THE INPUT DC POWER SUPPLY. DOING SO WILL DAMAGE THE POD64158.**

2-2-4. Changing the Chip Select Dipswitches

As explained in Section 2-1-2, the target chip can be changed with the DCL file read when the EASE64158 initialized, and the setting of the POD64158 dipswitches. This section explains how to change the chip select dipswitches.

The chip select dipswitch have the following respective functions.

(a) Chip Select Dipswitch 4 (SW4)

The dipswitch 4 specifies which SFRs and data memory in the MSM64E153 are prohibited and which are permitted. It corresponds to the appropriate device in the MSM64153 family.

(b) Chip Select Dipswitches 2 and 3 (SW2 and SW3)

The dipswitches 2 and 3 specify which of the MSM64E153's internal interrupt circuits are prohibited and which are permitted. They correspond to the appropriate device in the MSM64153 family.

When the system is shipped, its dipswitches will be set to correspond to the MSM64153.

The method for changing the chip select dipswitch is shown below. The chip select dipswitches are mounted on a board inside the POD64158. They can be changed by unscrewing the top case of the POD64158 and removing it.

1. Remove the top cover or the POD64158. Remove the four screws on the sides, and remove the top cover.

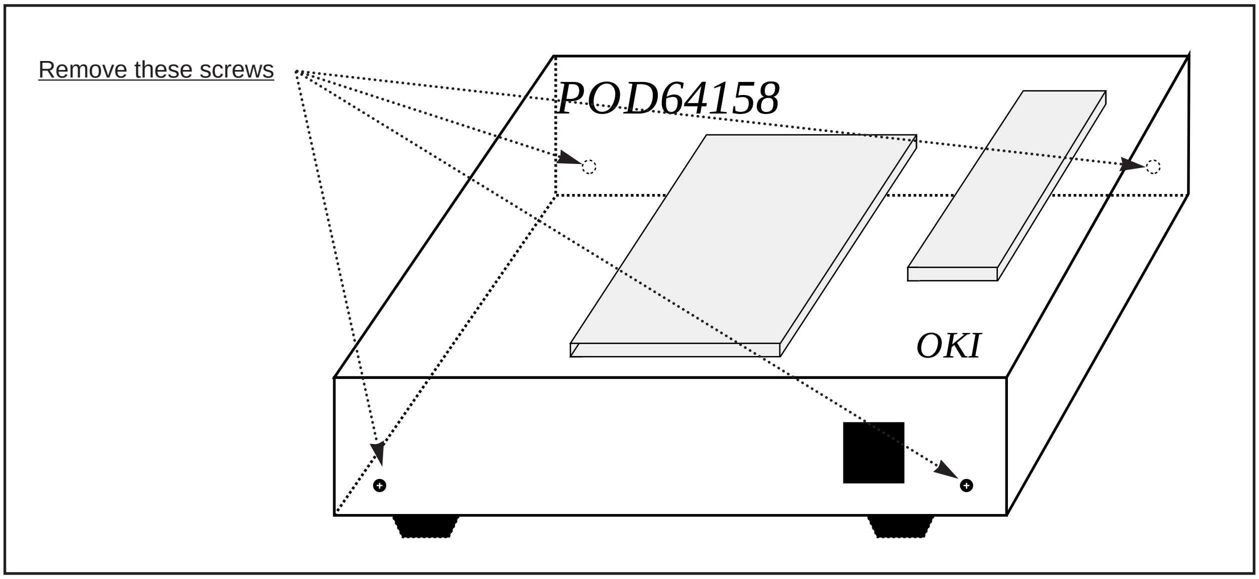
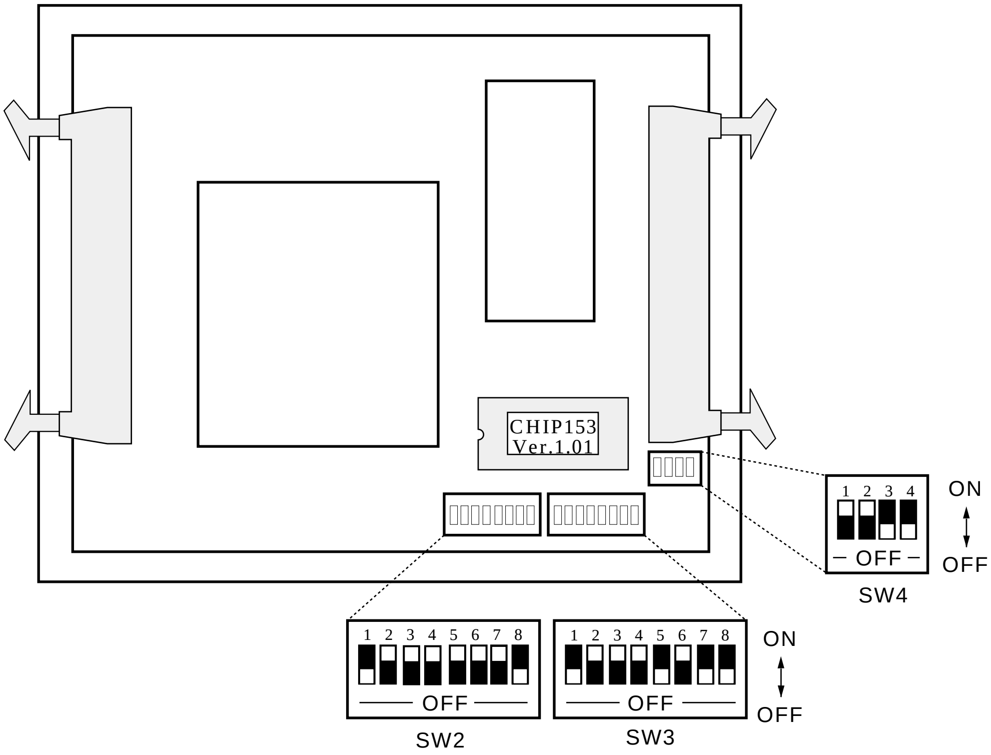


Figure 2-9. Changing the Chip Select Dipswitches (1)

2. Set the dipswitches shown in Figure 2-10.

Figure 2-10. Changing the Chip Select Dipswitches (2)



Set the SW4 in accordance with the target chip being used as shown in below.

Table 2-4. Setting SW4 Dipswitch

Target Chip	SW4-1	SW4-2	SW4-3	SW4-4
MSM64152	OFF	ON	OFF	OFF
MSM64153	ON	ON	OFF	OFF
MSM64155	ON	OFF	ON	OFF
MSM64158	OFF	OFF	OFF	ON

Set SW2 and SW3 in accordance with the target chip as shown below.

Table 2-5. Setting SW2 and SW3 Dipswitches

Target Chip	SW2-1	SW2-2	SW2-3	SW2-4	SW2-5	SW2-6	SW2-7	SW2-8
MSM64152	OFF	ON	OFF	OFF	ON	ON	OFF	OFF
MSM64153	OFF	ON	ON	ON	ON	ON	ON	OFF
MSM64155	OFF	ON	ON	ON	ON	ON	ON	OFF
MSM64158	OFF	ON	OFF	ON	ON	ON	OFF	OFF

Target Chip	SW3-1	SW3-2	SW3-3	SW3-4	SW3-5	SW3-6	SW3-7	SW3-8
MSM64152	ON	OFF	ON	ON	OFF	ON	OFF	OFF
MSM64153	OFF	ON	ON	ON	OFF	ON	OFF	OFF
MSM64155	OFF	ON	ON	ON	OFF	ON	OFF	OFF
MSM64158	OFF	OFF	ON	OFF	ON	ON	OFF	OFF

!

IF THE CHIP SELECT DIPSWITCHES ARE SET INCORRECTLY, INTERRUPTS WILL NOT BE EXECUTED PROPERLY!

!

Remove the user cable connector and the DC power supply cable when the top case of the POD64158 is removed.

!

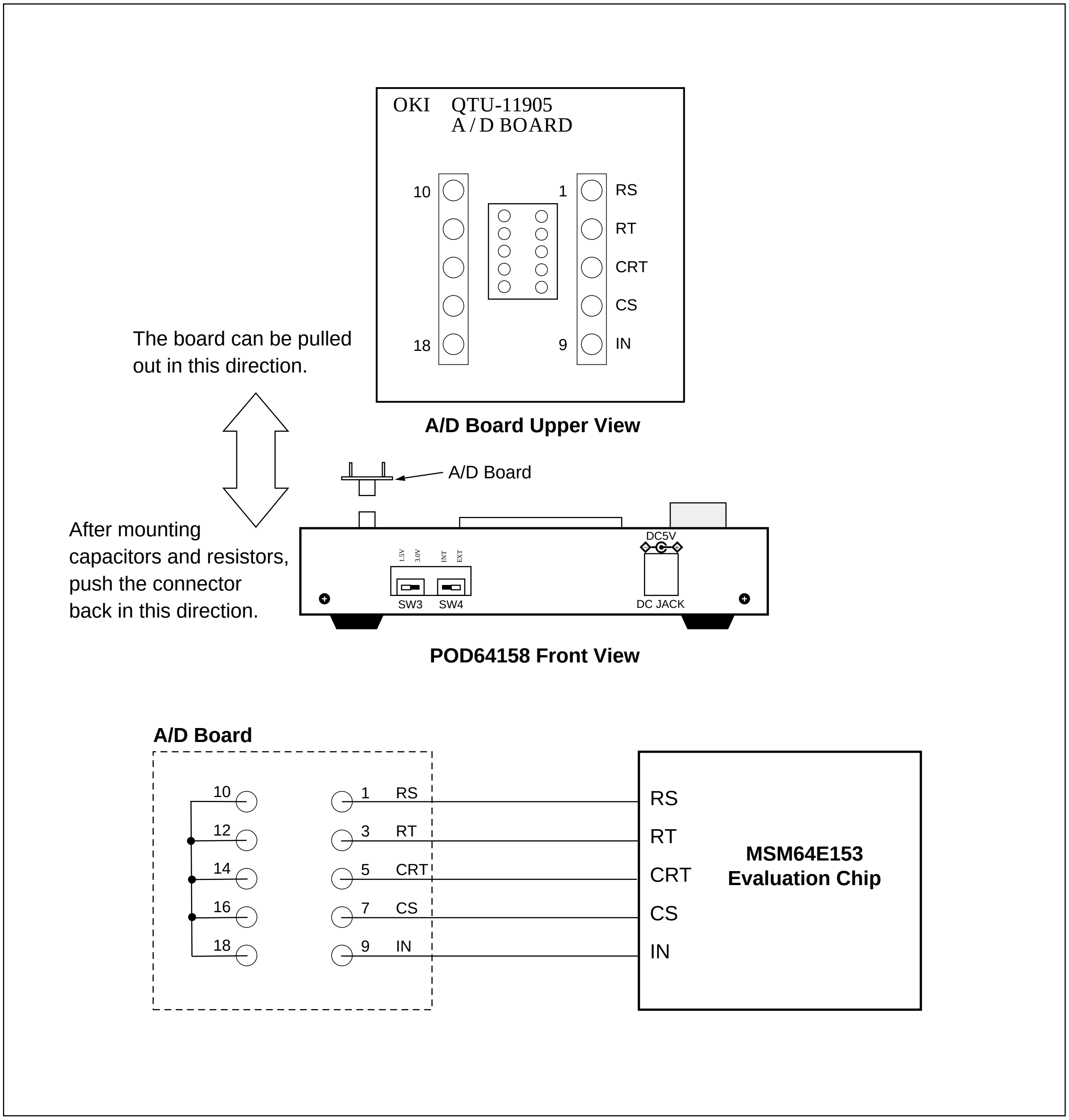
When the system is shipped, dipswitches are set to correspond to the MSM64153.
Set them to match the device being used.

!

The DCL file and dipswitches must all be set to the same device in the MSM64153 family being used. If the settings are mixed, then the EASE64158 will not operate properly.

2-2-5. A/D Board

The A/D board is only available for the MSM64153 family microcontrollers that are equipped with the A/D converter. Refer to the each microcontroller's User's Manual when the A/D board is used.



A/D Board Upper View

POD64158 Front View

Figure 2-11. A/D Board

As shown above, each pin of the A/D board is directly connected to the MSM64E153 evaluation chip's pin. So, the A/D conversion will made in accordance with the same electric characteristics of the MSM64153 family microcontrollers being used.

2-2-6. Starting the EASE64158 Emulator

The procedure for starting the EASE64158 emulator differs for each operation mode. Each starting procedure is as follows.

2-2-6-1. Starting the EASE64158 in EASE-LP2 mode

(1) Confirm that the necessary cables are connected to the EASE64158 emulator (hereafter called the emulation kit) .

(a) Connect the AC power supply cable to the AC connector.

- Confirm that the EASE-LP2 power switch is OFF.
- Note that the AC power supply rated voltage is AC 100–240 V.

(b) Connect the host computer to the EASE-LP2.

- Connect the RS232C cable to the EASE-LP2's RS232C connector and the host computer's RS232C connector.

(c) Connect the EASE-LP2 to the POD64158.

- Connect the interface cables (80-pin, 100-pin) between the EASE-LP2 and the POD64158.

(d) Connect the user cable.

- Connect the user cable when using the user application system.
- Connect the attached 60-pin and 64-pin user cables between the POD64158 USER connector and the user application system.

(e) Connect the probe cable.

- Connect the probe cable to the EASE-LP2's PROBE connector and the probe points of the user application system.

(f) Connect the external power supply.

- Connect the external power supply to the user application system.
- Confirm that the power switch of the external power supply is OFF.
- Note that the external power supply voltage should be the same as the operating voltage of the MSM64E153 evaluation chip being used.

(g) Mount the A/D board.

- The A/D board can be used only with the MSM64153 family microcontrollers that are equipped with the A/D converter. For details, refer to Section 2-2-5. "A/D Board."

! Do not mount the A/D board on the POD64158 when the A/D board is not used.

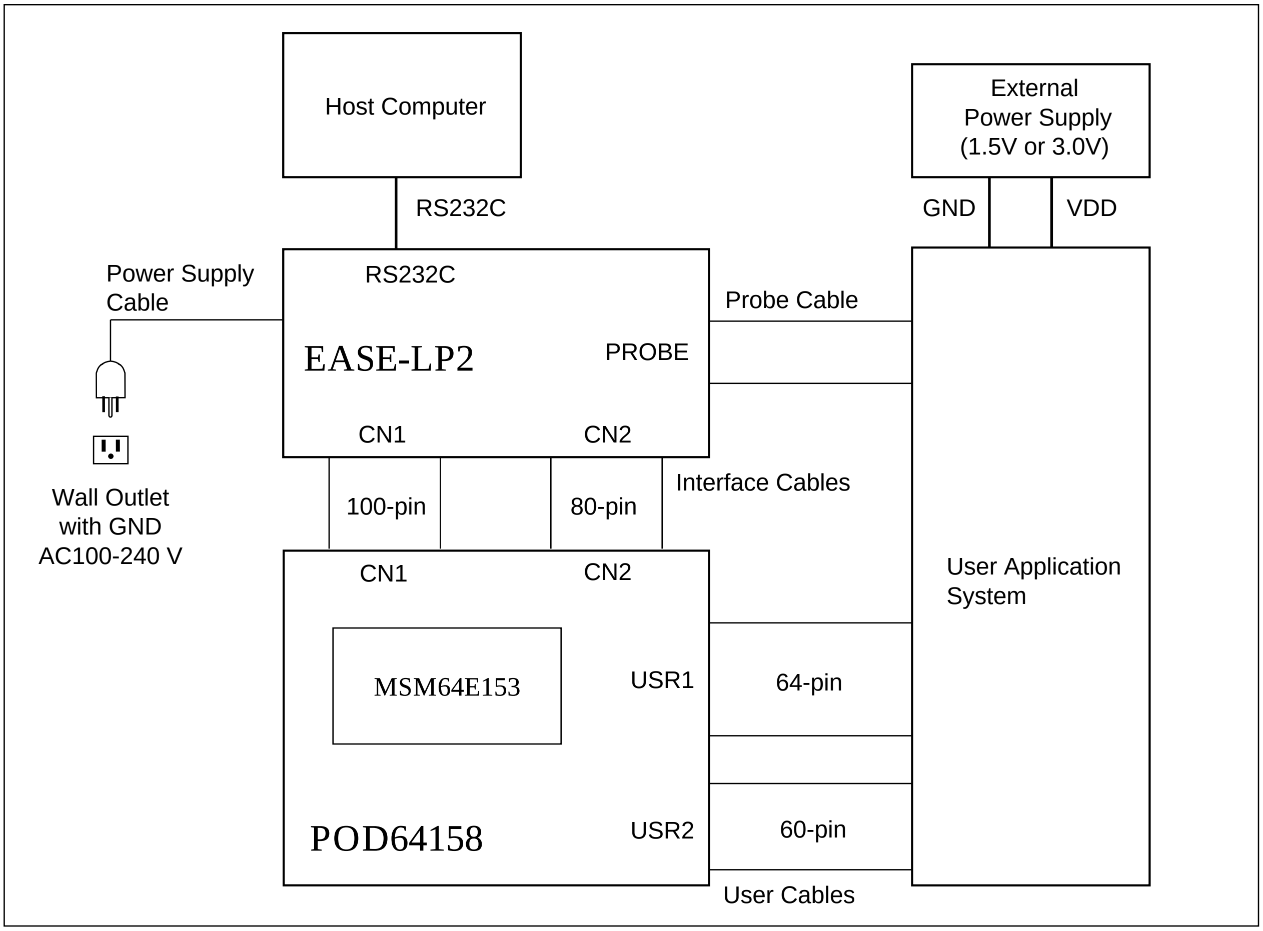


Figure 2-12. Cable Connection in EASE-LP2 Mode

! The emulation kit will start even if the user application system is not connected. In this case, do not connect the user cable and the probe cable.

! VDD is not supplied to the user application system from the user cables (however, VSS1 or VSS2 level is connected to the user application system through the user cables). If VDD must be supplied to the user application system, use a separate power supply.

! **IN EASE-LP2 MODE, DO NOT CONNECT THE DC POWER CABLE TO THE POD64158 DC JACK. IF IT IS CONNECTED, THE EMULATION KIT WILL NOT OPERATE PROPERLY.**

! **IN EASE-LP2 MODE, DO NOT MOUNT THE USER PROGRAM EPROM ON THE POD64158 EPROM SOCKET. DOING SO COULD CAUSE A MALFUNCTION.**

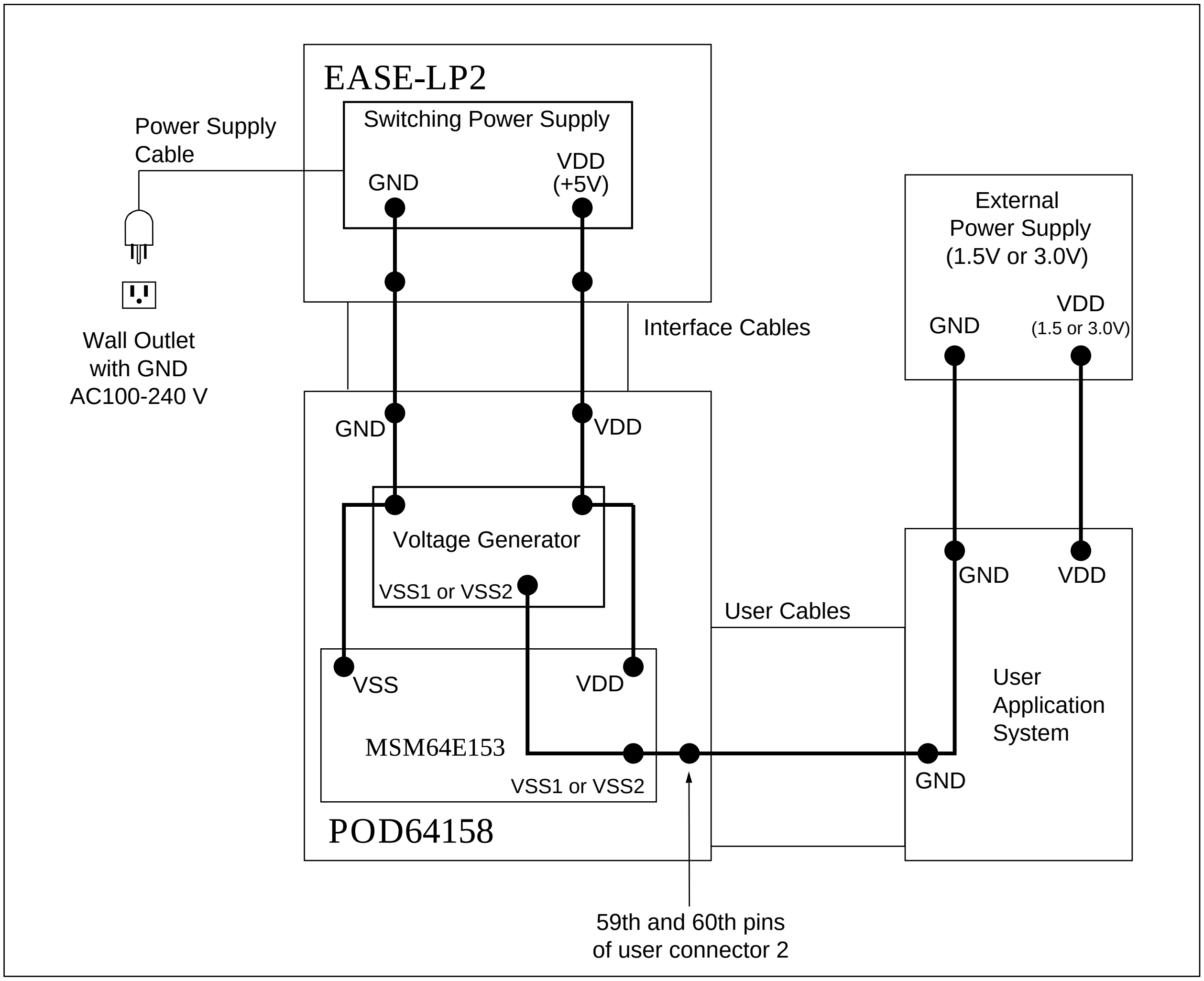


Figure 2-13. Power Supply Configuration in EASE-LP2 Mode

! The 59th and 60th pins of the POD64158 user cable connector provides VSS1 when the MSM64E153 evaluation chip's operating voltage is 1.5 V, and VSS2 when it is 3.0 V.

! The external power supply voltage should be same as the MSM64E153 evaluation chip's operating voltage (i.e. 1.5-V power supply is required for the 1.5-V operating evaluation chip, 3.0-V power supply for the 3.0-V operating evaluation chip).

! In EASE-LP2 mode, do not connect the DC power supply cable to the POD64158 DC jack. If it is connected, then the emulation kit will not operate properly.

(2) Verify that the emulation kit switches are set correctly.

❑ EASE-LP2

(a) Baud rate setting
- Set SW1-1 to SW1-4 to match the baud rate being used.

❑ POD64158

(a) Operating mode setting
- Set SW2 to ICE to select EASE-LP2 mode.

(b) Setting MSM64E153 evaluation chip's operating voltage
- Set SW3 to 1.5V when the evaluation chip's operating voltage is 1.5 V, and set it to 3.0V when the voltage is 3.0 V.

(c) Operating clock supply setting
- Set SW4 to select whether the operating clock is supplied from the POD64158 crystal board, or from the EXT•CLK pin of the user connector 2.

(d) Chip select dipswitch setting
- Set chip select dipswitches to match the target chip being used.

For details on switch settings, refer to Section 2-2-2, “EASE64158 Switch Settings.”

(3) Confirm that the MSM64E153 evaluation chip is mounted correctly.

- Be sure to match the 1-pin label on the evaluation chip to that of the POD64158.
- Refer to Appendix-11, "Handling the POD64158 Evaluation Chip," regarding the mounting procedure of the evaluation chip.

The 1.5-V operating evaluation chip has a label "MSM64E153-1.5V" on its top, and the 3.0-V operating chip "MSM64E153-3.0V."

(4) Turn on the host computer power supply, and start MS-DOS (PC-DOS).

! Use MS-DOS or PC-DOS version 3.1 or later.

(5) Set the host computer’s transfer parameters.

When the EASE64158 is shipped, its data transfer parameters are as follows. Except for baud rate, the EASE64158 parameters cannot be changed.

Baud rate	9600 bps
Transfer format	8 bits, 1 stop bit, no parity
Flow control	XON/XOFF control
Others	Asynchronous, baud rate factor x16

Oki if800 series computers are set using the **SWITCH** command.

PC9801 series computers are set using the **SPEED** command.
For details, refer to the manual of the host computer.

With the if800 series after changing parameters with the **SWITCH** command, if the if800 reset

button is pushed once more to boot up the computer again, then be sure to note that the RS232C parameters will not be set correctly.

! IBM-PC computer uses the INT232C program (described in step 6 below).

(6) Invoke INT232C.
This step should be executed only if you are using an IBM-PC computer.
For other computers, skip this step and go to step 7.

INT232C is a TSR (Transient but Stay Resident) program. It sets the RS232C interface operating conditions of the IBM-PC/AT, and simultaneously enables interrupt signals.

Invoking this program once will place it in host computer memory, where it will reside until removed. The method for invoking and removing INT232C is shown below.

❑ *Invoking INT232C*

First verify the settings of the baud rate switches on the EASE-LP2 unit. Assume that the verified baud rate is called **<baud>**. (☞1) Next, change to the directory that stores the INT232C.COM file and enter the following input.

```
A> INT232C X;<baud>,N,8,1 ↵
```

This will load INT232C into host computer memory, and display the following message.

```
INT232C has been loaded.
```

This ends the process of invoking INT232C. If INT232C had already been loaded, then the following message will be displayed instead.

```
INT232C has already been loaded.
```

In this case, it will not be newly loaded.

Example Input the following to use a baud rate of 4800 bps. After the INT232C has been loaded, the message will be displayed.

```
A> INT232C X;4800,N,8,1    ↵
   INT232C has been loaded.
```

☞ **1** The valid baud rates for the EASE64158 are listed below. However, INT232C.COM cannot set 19200 bps.

2400, 4800, 9600, 19200

❏ *Removing INT232C*

Because INT232C is a resident program, it will stay in memory even after you have finished with the symbolic debugger (SID64K). Input the following to remove it.

```
A> INT232C R  ↵
```

This will remove INT232C from memory, and display the following message.

```
INT232C has been removed from memory.
```

If it has already been removed, then the following message will be displayed instead.

```
INT232C has not been loaded.
```

The above simple explanations show how to use INT232C. Read the following page if you wish to understand each parameter in detail. If you do not need to know them in detail, go on to step 7.

! If you will use SID64K with IBM PC/AT, then add the appropriate ANSI escape sequence driver from your DOS system disk to CONFIG.SYS. If you forget to do so, then you will not be able to use the special editing keys.

Host computer	ANSI escape sequence driver name
IBM PC/AT	ANSI.SYS

❑ *Explanation of INT232C input format and parameters*

```
A>  INT232C [<options>[;<baud>,<parity>,<databits>,<stopbits>]] ↵
```

The brackets [ ] can be omitted. When omitted, the default values of the following explanations apply.

<*options*>

X Perform XON/XOFF control.
* Do not perform XON/XOFF control.
R Remove resident INT232C.

<*baud*>

Specifies the baud rate. Choose one of the following.

2400, 4800, 9600 (default)

<*parity*>

Specifies whether and what kind of parity checking to perform. Choose one of the following.

N Do not perform parity checking (default).
O Perform odd parity checking.
E Perform even parity checking.

<*databits*>

Specifies the number of data bits. Choose one of the following.

7, 8 (default)

<*stopbits*>

Specifies the number of stop bits. Choose one of the following.

1 (default), 2

If the command is executed with all parameters omitted, then the above explanation of INT232C usage will be displayed. This is convenient if you forget how to use INT232C.

Example

- **`INT232C *`** ↵
  This is the same as: **`INT232C *;9600,N,8,1`**
- **`INT232C X;1200,E,7,2`** ↵
  This initializes the RS232C port to **XON/XOFF** control, 1200 bps baud rate, even parity, 7 data bits, and 1 stop bit.
- **`INT232C R`** ↵
  This removes INT232C from memory.

❏ *List of messages*

INT232C outputs the following messages.

- **`INT232C has been removed from memory.`**
- **`INT232C has not been loaded.`**
- **`INT232C has already been loaded.`**
- **`INT232C has been loaded.`**

(7) Set the DCL file environment.

The DCL file has the symbol information necessary to perform symbolic debugging with SID64K. SID64K first searches for it in the current directory. If not found, then it searches the directory which contains SID64K.EXE and then the directory specified by the DCL environment variable. If still not found, then SID64K will not start.

❑ *Setting the environment*

There are three ways to set the environment so that SID64K will read the DCL file when it is started.

(1) Store the **DCL** file in the current directory.
(2) Store the **DCL** file in the directory which contains **SID64K.EXE**.
Copy the **DCL** file for the device to be used into the directory that contains **SID64K** with the **COPY** command of **MS-DOS/PC-DOS**.
(3) Set the **DCL** environment variable to the path name of the directory that contains the **DCL** file.
Set the **DCL** environment variable with the following input.

```
A> SET DCL = path-name ↵
```

The *path-name* here is the path name of the directory that contains the **DCL** file.

The **DCL** environment variable will be lost if the host computer is reset. If this happens, then set the **DCL** environment variable again.

If you feel setting the environment variable every time the host computer is started up, then you can eliminate this step by registering the **DCL** environment variable in your **AUTOEXEC.BAT** file. For information about **AUTOEXEC.BAT** files and registering environment variables, refer to the manual that came with your host computer.

! The SID64K symbolic debugger will read the DCL file corresponds to the evaluation chip specified with the POD64158 chip select dipswitches when it is started.

(8) Start the SID64K symbolic debugger.

The symbolic debugger executable file SID64K.EXE can be started from the directory that stores it or from another directory.

(1) Starting from the directory that stores **SID64K.EXE**

Input the following after the DOS prompt.

```
A> SID64K ↵
```

(2) Starting from another directory

If the **PATH** environment variable includes the directory that contains **SID64K.EXE**, then input is the same as in (1). If not specified by **PATH**, then the **SID64K** symbolic debugger is invoked as follows.

```
A> path-name\SID64K ↵
```

Here *path-name* is the absolute path name of the directory that contains **SID64K.EXE**.

! Confirm that the emulation kit power supply switch is turned off before starting the SID64K symbolic debugger.

(9) The following message will be displayed on the console, and the system will wait for a reset switch input from emulation kit.

```
SID64K Symbolic Debugger Ver. x.xx
Copyright (C) xxxx. OKI Electric Ind.Co.,Ltd.
```

(10) Turn on the emulation kit power supply switch and the power supply of the user application system. The following message will be displayed on the host computer, and emulator system initialization will end.

```
*** POWER ON INITIALIZATION START ***
*** RESET CODE ACCEPTED ***
```

(11) Next the following message will be displayed, the OLMS-64X series DCL file will be read, and memory mapping of the appropriate device will be performed.

```
DCL file (E64XXX.DCL) reading...
```

E64XXX.DCL is the name of the **DCL** file read.

(12) When the DCL file has been read, a prompt corresponding to the DCL file type will be displayed and the system will wait for command input (☞2).

```
64XXX>
```

The prompt will be the name of a chip in the MSM64153 family. For example, if **E64153.DCL** is read, then the prompt will be **64153>**.

(13) From then on, debugger commands can be input.

(1) For more information on the emulator's RS232C interface, refer to Section 2-2-2, "EASE64158 Switch Settings."

(2) The user application system cannot be supplied with VDD taken from the emulator.

(3) The VSS1 or VSS2 line in the user cable is connected, but the VDD power supply line is not (it is an open pin).

(4) If the emulator does not start, refer to Appendix 5.

(5) When the EASE-LP2 is connected to the POD64158, do not mount an EPROM in the POD64158 evaluation module's EPROM socket. If an EPROM is, incorrect operation may occur.

(6) When the EASE-LP2 is connected to the POD64158 with the interface cables, the POD64158 is connected to the VDD from the EASE-LP2. Therefore, do not connect the DC power supply cable to the POD64158's DC power jack. If it is connected, incorrect operation may occur.

(7) When the EASE-LP2 and POD64158 are correctly connected with the interface cables, the POD indicator on the top of the EASE-LP2 and the POWER indicator on the top of the POD64158 will light.

(8) Always turn on the emulation kit power supply switch before the user application system is turned on. If the user application system is turned on first, then the emulation kit will not start properly.

Table 2-5 (a)-(b) shows the items that are initialized when power is applied to the EASE64158, when the reset switch is pressed, when an **RST** command is executed, and when an **RST E** command is executed. Items in the table with an entry of "O" are initialized, while items with an entry of "-" are not.

Also, when the reset switch is pressed, all open files will be closed.

Once the EASE64158 is turned off, be sure not to make an attempt to turn it on again for approximately 5 seconds.

Table 2-5 (a). Initialization

Item	Contents Initialized	Power Applied	Reset Switch Pressed	RST Command	RST E Command
MSM64153 Evaluation Chip	Initializes to same state as when a reset is input to a microcontroller in the MSM64153 family.	O	O	O	O
Break Conditions	Breakpoint bit breaks (BP) and power-down breaks (PD) enabled.	O	–	–	–
Breakpoint Bits	All areas cleared to "0", disabling all breakpoint bit breaks.	O	–	–	–
Break Status	Cleared to state of no breaks generated.	O	O	O	–
Instruction Executed Bits	All areas cleared to "0", disabling all trace enable bit tracing.	O	–	–	–
Trace Pointer	Cleared to "0"	O	–	–	–
Trace Trigger	Set to free-running trace mode.	O	O	–	–
Trace Enable Bits	All areas cleared to "0", disabling all trace enable bit tracing.	O	–	–	–
Trace Execution Format (**STF** Format)	Set to default mode.	O	–	–	–
Trace Settings	Set to defaults.	O	–	–	–
Cycle Counter	Cleared to "0"	O	O	O	–
Cycle Counter Trigger	Cycle counter start/stop addresses are cleared, and counting is disabled.	O	O	–	–
TIME Command Display Units	Set to default value (91.0 µs)	O	–	–	–

Table 2-5 (b). Initialization

Item	Contents Initialized	Power Applied	Reset Switch Pressed	RST Command	RST E Command
Step execution Format (**SSF** Command)	Set to default mode.	O	–	–	–
Address Pass Counters 0 - 3	Cleared to "0."	O	O	O	–
Count Address of Address Pass Counters	Set to address "0000."	O	–	–	–
EPROM Programmer Setting	Set to type "I27512."	O	O	–	–
RADIX Command	Set to default (hexadecimal).	O	–	–	–
MAC Command	Removes registrations (☞ 1).	–	–	–	–
Symbol Registration	Removes registrations (☞ 1).	–	–	–	–

☞ 1

Because information about symbols registered with **LOD** and **CSYM** commands, and information about emulator commands registered with **MAC** commands are stored in the SID64K symbolic debugger on the host computer, they are not initialized by applying power or a reset to the EASE64158. However, if the SID64K terminates once, then all registered information will be lost.

2-2-6-2. Starting the POD64158 in POD mode

(1) Confirm that the necessary cables are connected to the POD64158.

(a) Connect the DC power supply cable to the DC power jack.
- Connect the attached DC power supply cable to the POD64158 DC power jack.
- Connect the DC power supply cable red plug to the plus side of the external power supply, black plug to the minus side. Be sure that external power supply is turned off before connecting the cable.
- Note that the rated operating voltage of the POD64158 is DC5V.

(b) Connect the user cable.
- Connect the attached 60-pin and 64-pin user cables between the POD64158 user connectors and the user application system.

(c) Connect the external power supply.
- Connect the external power supply to the user application system.
- Note that the external power supply voltage should be same as the operating voltage of the MSM64E153 evaluation chip (i.e. 1.5-V external power supply for the 1.5-V operating evaluation chip, 3.0-V power supply for the 3.0-V operating chip.)

(d) Mount the A/D board.
- The A/D board can be used only with the MSM64153 family microcontrollers that are equipped with the A/D converter. For details, refer to Section 2-2-5, "A/D Board."

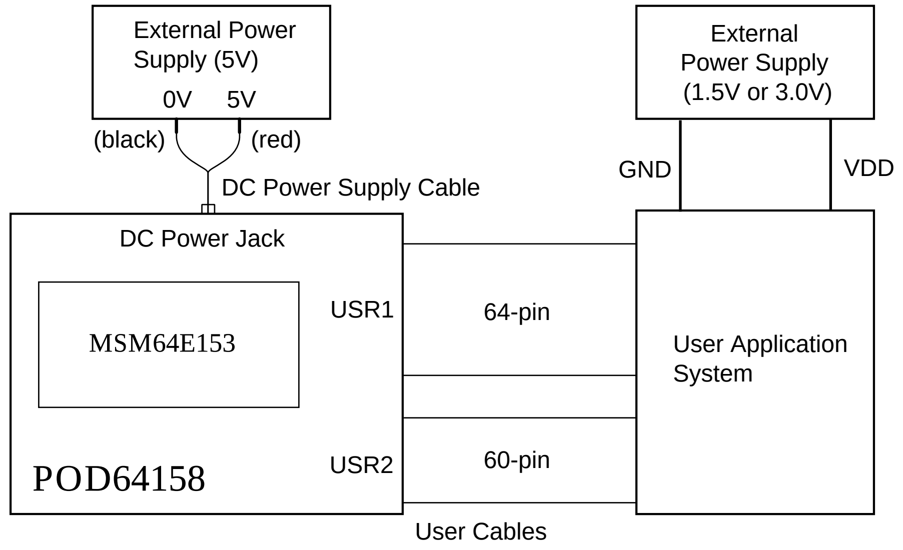


Figure 2-14. Cable Connection in POD Mode

! The VDD is not supplied to the user application system (however, VSS1 or VSS2 level is connected to the user application system through the user cable). If the user application requires VDD, user an appropriate separate power supply.

! In POD mode, do not connect the interface cable between the emulation kit to the EASE-LP2. If it is connected, incorrect operation may occur.

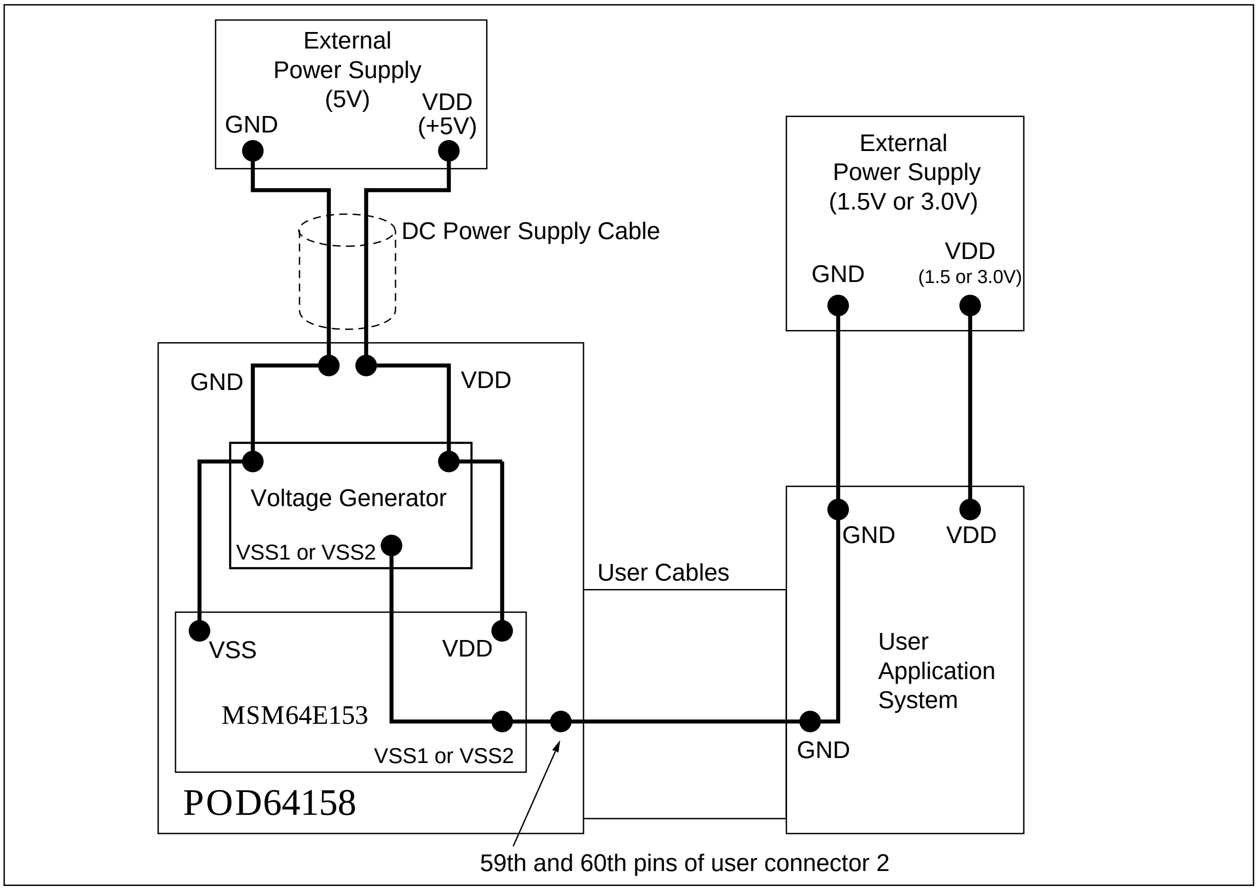


Figure 2-15. Power Supply Configuration in POD Mode

! The 59th and 60th pins of the user cable connector will provide VSS1 when the MSM64E153 evaluation chip's operating voltage is 1.5 V, VSS2 when it is 3.0 V.

! The external power supply voltage shoule be same as the operating voltage of the MSM64E153 evaluation chip (i.e. 1.5-V external power supply for the 1.5-V operating evaluation chip, 3.0-V power supply for the 3.0-V operating chip).

! In POD mode, do not connect the interface cable between the emulation kit and the EASE-LP2. If it is connected, incorrect operation may occur.

(2) Verify that the POD64158 switches are set correctly.

(a) Selecting the user EPROM type mounted on the EPROM socket.

- Set SW1 to match the user program EPROM to be used.

(b) Operating mode setting.

- Set SW2 to POD to select POD mode.

(c) Evaluation chip's operating voltage setting.

- Set SW3 to 1.5V when the MSM64E153 evaluation chip's operating voltage is 1.5 V, and set it to 3.0V when the voltage is 3.0 V.

(d) Operating clock supply setting

- Set SW4 to specify whether the operating clock is supplied from the POD64158 crystal board, or from the EXT•CLK pin of the user connector 2.

(e) Chip select dipswitch setting.

- Set chip select dipswitches to match the target chip being used.

For details on switch settings, refer to Section 2-2-2, “EASE64158 Switch Settings.”

(3) Verify that the MSM64E153 is mounted correctly.

- Mount the evaluation chip so that its pin 1 mark matches the pin 1 mark on the POD64158.
- For more on mounting the evaluation chip, refer to Appendix 11, “Mounting the POD64158 Evaluation Chip.”

The 1.5-V operating voltage evaluation chip is labeled “MSM64E153-1.5V” on its top surface, while the 3.0-V operating voltage evaluation chip is labeled “MSM64E153-3.0V” on its top surface.

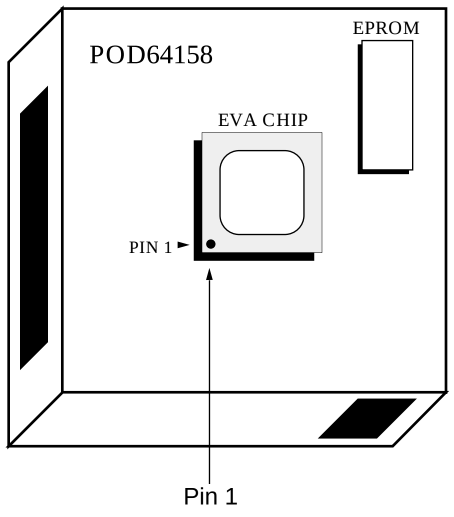


Figure 2-16. Mounting MSM64E153 Evaluation Chip

(4) Mount the EPROM that contains the user program into the EPROM socket.

Refer to Appendix 10, “Mounting POD64158 EPROMs.”

Usable EPROM types are: **2764, 27C64, 27128, 27256, 27512, 27C128, 27C256, 27C512.** Refer to Table 2-2 in Section 2-2-2, “EASE64158 Switch Settings.”

(5) Turn the POD64158 power supply switch on, and then turn the user application system on.

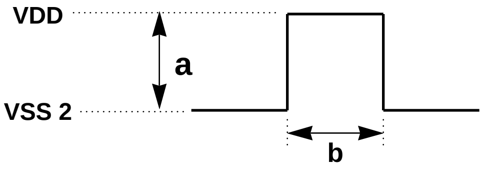
- POD64158 POWER indicator will light.

! Always turn on the external power supply to the POD64158 first, and then turn on the user application system power supply. If the user application system power supply is turned on first, then the POD64158 will not operate properly.

(6) Input a reset signal on the USER RESET pin.

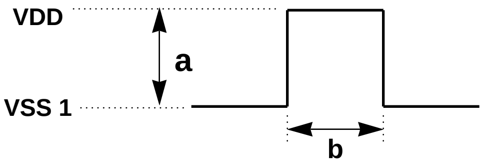
Use a signal like the one below to input on the USER RESET pin.

When evaluating chip operating voltage is 3.0 V (MSM64E153-3.0V)



a: Voltage 3V (±5%)
b: 10 ms or more

When evaluating chip operating voltage is 1.5 V (MSM64E153-1.5V)



a: Voltage 1.5V (±5%)
b: 10 ms or more

(7) At this point, all internal states are initialized, and the user program is executed from address 000H.

!

(1) When using the POD64158 standalone, do not connect the interface cables.

(2) Do not handle the evaluation chip or EPROM when power is on.

(3) The POD64158 can currently be used for the following four devices.

MSM64152
MSM64153
MSM64155
MSM64158

The target chip can be changed on the POD64158 by changing the chip select dipswitch. Refer to Section 2-2-4, “Changing the Chip Select Dipswitches.”

(4) The reset signal input on the USER RESET pin must be held at an “H” level after power is applied until the POD64158 internal oscillator begins oscillating. Therefore, if you change the oscillator mounted in your system when shipped, the a reset signal appropriate for the new oscillator. Reset operations can be performed at times other than when power is applied by inputting a reset signal wider than one clock. For details on reset operation, refer to the user’s manual of the MSM64153 family device.

2-3. SID64K Debugger Commands

2-3-1. Debugger Command Syntax

The explanations of this manual make use of the following symbols.

- UPPER CASE   Debugger command names are expressed with upper case letters.

Example **DCM, LOD, G**

- *italics*   *Italicized* expressions indicate user-supplied information (changes according to operator input). The following italicized words are used.

parm	This indicates a general parameter that follows after a command name. It includes *fname*, *expression*, *address*, *data*, *number*, *bank*, and *mnemonic*, explained below.
fname	This indicates a file name, including drive name, path name, primary name, and extension. Except for the extension, a file name is handled with the exact same processing as a DOS file name. Extensions are handled differently depending on the command (when omitted for some commands, default extensions exist).
expression	This indicates an expression. It can include operators and symbols. Types of expressions are *address*, *data*, *number*, and *bank*.
address	This indicates an address value input.
data	This indicates a data value input.

number *bank* *count*	These are types of expressions. They are recognized as decimal regardless of the **RADIX** command setting. A *number* is used to indicate a cycle counter value, step count, etc. A *bank* indicates an input value for a register bank number. A *count* indicates a pass count value of **G** command breakpoints.
mnemonic	This indicates an optional string input from a set of strings that is determined by the command type.
string	This indicates any string.
option	These are normally constructed with a slash "/" and a specific string. They are added as needed after command parameters (*parm*). They place restrictions or add functions to command operations.

- Special symbols These symbols have the following special meanings for explaining command syntax.

Δ	This indicates white space (☞1).
↵	This means a carriage return input.
[xxxx]	The xxxx means an optional string used within an explanation. The xxxx enclosed in [ ] means that it can be omitted.
[addressΔaddress]	This indicates an address range.
___ (underline)	When text displayed automatically by the debugger and operator input are mixed on one line, the underlined portion indicates user input.

☞ 1 White space is a string consisting of one or more spaces (ASCII code 20H) and/or tabs (ASCII code 09H) in any order.

2-3-1-1. Character Set

SID64K debugger commands can make use of the following character set.

1. **Characters with letter attributes** .... (☞ 2)

   (1) Alphabetic characters (upper and lower case)

   A B C D E F G H I J K L M

   N O P Q R S T U V W X Y Z

   a b c d e f g h i j k l m

   n o p q r s t u v w x y z

   (2) Special characters

   _ $

2. **Digits**

   0 1 2 3 4 5 6 7 8 9

3. **Delimiters**

   TAB space CR .... (☞ 3)

4. **Operators** .... (☞ 4)

   + - * / % & | ^ ˜ ! . < = > ( )

5. **Other special symbols**

   \ [ ] { } : ; ? @ # “ ‘ , ˜

☞ **2** Characters with letter attributes are those characters that can be used as the first character of a symbol. Anything that starts with another kind of character will be recognized as a number, operator, delimiter, or other special symbol. Characters bordered by the dotted rectangle can be used only by the **ASM** command (during command input, these characters are converted to upper case).

☞ **3** TAB is ASCII code 09H; space is ASCII code 20H; CR (carriage return) is ASCII code 0DH.

☞ **4** Of these operators, only +, -, (, and) are permitted in command line expressions. Other operators are permitted only within the **ASM** command.

! All characters usable with SID64K debugger commands are included in this character set. If any other character is encountered, then the “Illegal character” error message will be output. However, any character can be coded in commend fields, described later.

2-3-1-2. Command Format

❏ *Debugger Command Format*

command_name Δ *parm* Δ *parm . . . parm* Δ *option* Δ *option . . . option* ↵

Debugger commands consist of a command name followed by several parameters (*parm*). Depending on the command type, a command might further be followed by option parameters (*option*). White space always delimits between the command name, *parm*, and *option*. A command line is recognized as ending at the point a carriage return (↵) is input.

❏ *Comment Input*

The entire string following a semicolon (;) is recognized as a comment. It will be ignored during command parsing. For example, in the first line below the entire line is a comment, so the emulator will perform no operation. The second line is an example of a comment appended after a command.

Example

```
64153> ;;;; This is an example of whole line comment ;;;;
64153> LOD SAMPLE.HEX /S ; This is a sample of comment after
                           command
```

❏ *ESC Key Input*

To forcibly terminate a debugger command, press the ESC key. The ESC key is valid during the following commands.

D, DCM, LOD, SAV, VER, DASM, DDM, STP, DBP, DTM, DTR, DIE, VPR, BATCH, DSYM

❏ *Space Key Input*

When the following commands display data, pressing the space key can temporarily stop the display. To resume, press the space key again.

DCM, VER, DASM, DDM, STP, DBP, DTM, S, DTR, DIE, VPR, DSYM

❏ *Command Name Format*

Command names are strings consisting of 1-5 alphabetic characters. They express instructions given to the debugger. A command name's function is indicated by its first character. Second and following characters are keywords for MSM64E153 evaluation chip or emulator internal registers and memory.

D	**(Display)**	Data display commands
C	**(Change)**	Data change commands
E	**(Enable**)	Enable commands
R	**(Reset)**	Reset commands
S	**(Set)**	Set commands
P	**(Program)**	Commands for writing data to EPROM
T	**(Transfer)**	Commands for reading data from EPROM
V	**(Verify)**	Commands for comparing memory contents
M	**(Move)**	Data move commands
G	**(Go)**	Execute (emulation) commands

However, the following commands are exceptions.

EXPAND, LOD, SAV, ASM, DASM, STP, ESC, TIME, TYPE, PAUSE, RADIX, MAC, S, URST, BATCH, LIST, NLST, SH, EXIT

2-3-1-3. Command Summary

This section gives a summary table of all SID64K commands.

Detailed explanations of each command are given in Chapter 3. The table of this section was created with the purpose of first giving a quick overview of the commands, and then in the future serving as a command index.

The table of this section follows the format below.

<table>
<tr><th colspan="4">Command Group Name</th></tr>
<tr><td rowspan="3">No.</td><td><b>Name</b></td><td>Function</td><td rowspan="3">Reference page</td></tr>
<tr><td colspan="2">Syntax</td></tr>
<tr><td colspan="2">Parameters / options</td></tr>
</table>

- No. Sequence number.
- Command Name Name of command.
- Command Syntax Shows command syntax.
- Explanations of Parameters/Options Explains the parameters and options expressed in the command syntax.
- Reference Page The page to reference for an explanation in Chapter 3, “SID64K Command Details.”

Evaluation Chip Access Commands

1	**D**	Display the contents of the Evaluation Chip	3-4
	D [Δ *parm*..... Δ *parm*] ↵		
	parm	: *SFR_mnemonic, Register_mnemonic* PC (program counter), C (carry flag), BSR0, BSR1, BCF, BEF	
2	**C**	Change the contents of the Evaluation Chip	3-4
	C Δ *parm*[Δ *parm*..... Δ *parm*] ↵		
	parm	: *mnemonic* [=*data*]	
	mnemonic	: *SFR_mnemonic, Register_mnemonic* PC (program counter), C (carry flag), BSR0, BSR1, BCF, BEF	
3	**SDF**	Set Data Format	3-17
	SDF [Δ *parm*..... Δ *parm*] ↵		
	parm	: [˜] *SFR_mnemonic*	
4	**DPC**	Display Program Counter	3-18
	DPC ↵		
5	**CPC**	Change Program Counter	3-18
	CPC Δ *address* ↵		

Code Memory Commands			
1	**DCM**	Display Code Memory	3-20
	DCM Δ *parm* ↵ DCM Δ * ↵		
	parm	: *expression* Δ [*expression*.....Δ *expression*] *expression* : *address* : [*address* Δ *address*]	
2	**CCM**	Change Code Memory	3-22
	CCM Δ *parm* [Δ *parm* .....Δ *parm*] ↵ CCM Δ * [= *data*]↵		
	parm	: *address* [= *data*] : [*address* Δ *address*] = *data*	
3	**EXPAND**	Expand Code Memory	3-25
	EXPAND [Δ *mnemonic*] ↵		
	mnemonic	: ON, OFF	
4	**MCM**	Move Code Memory	3-27
	MCM Δ [*address* Δ *address*] Δ *address* ↵		
5	**LOD**	Load Disk file program into Code Memory	3-28
	LOD Δ *fname* [Δ *option* ..... Δ *option*] ↵		
	fname *option*	: [*Pathname*] *Filename* [*Extension*] : /S, /N, /B	
6	**SAV**	Save Code Memory into Disk file	3-32
	SAV Δ *fname* [[*address* Δ *address*]] [Δ option.....Δ option] ↵		
	fname *option*	: [*Pathname*] *Filename* [*Extension*] : /S, /N	

Code Memory Commands (continued)

7	**VER**	Verify Disk file with Code Memory	3-34
	VER Δ *fname* [[*address* Δ *address*]] ↵		
	fname	: [*Pathname*] *Filename* [*Extension*]	
8	**ASM**	Line Assembler Command This command provides assembler processing nearly fully compatible with the ASM64K assembler. The code it generates is stored in code memory.	3-37
	ASM Δ *address* ↵		
9	**DASM**	Disassembler Command Disassembles a specified address range of code memory.	3-41
	DASM [Δ *parm*] [Δ option ..... Δ option] ↵		
	parm *option*	: *address*, [*address* Δ *address*] : /NC, /NL	

Data Memory Commands

1	**DDM**	Display Data Memory	3-46
	DDM ∆ *parm* [∆ *parm* ..... ∆ *parm*] ↵ DDM ∆ * ↵		
	parm	: *address* : [*address* ∆ *address*]	
2	**CDM**	Change Data Memory	3-46
	CDM ∆ *parm* [∆ *parm* ..... ∆ *parm*] ↵		
	parm	: *address* [= *data*] : [*address* ∆ *address*] = *data*	
3	**MDM**	Move Data Memory	3-50
	MDM ∆ [*address* ∆ *address*] ∆ *address* ↵		

Emulation Commands			
1	**STP**	Step Execution	3-54
	STP [Δ *address*] [, *count*] ↵		
2	**SSF**	Set Step Format	3-56
	STP [Δ *parm* ..... Δ *parm*] ↵		
	parm	: [˜] *mnemonic*	
3	**G**	Real Time Emulation (Continuous Emulation)	3-59
	G [Δ *Emu_start_addr*] [, *Break_parm*] ↵		
	Emu_start_addr	: *starting address of realtime emulation*	
	Break_parm	: *address* [Δ *address* ..... Δ *address*] : [*address* Δ *address*] : *address* [*count*] : / *address* / *address* [/ *address*] : *mnem* [*&mask*] = *data* : *mnem* [*&mask*] = *data* [*count*] : *mnem* [*&mask*] = *data* [Δ /*address* [Δ *address* .....]] : *mnem* [*&mask*] = *data* [*count*] [Δ /*address* [Δ *address* .....]] : *mnem* [*&mask*] = *data* [/ [*address* Δ *address*]] : *mnem* [*&mask*] = *data* [*count*] [/ [*address* Δ *address*]]	
	mnem	: PRB, RAM [*ram_addr*]	
4	**ESC**	Forced Break of Emulation	3-65
	ESC ↵		

Emulation Commands (continued)

5	**DCT**	Display Cycle Counter Trigger	3-66
	DCT ↵		
6	**DTT**	Display Trace Trigger	3-67
	DTT ↵		
8	**D**	Display the Contents of the Evaluation Chip	3-68
	D [Δ *parm* ..... Δ *parm*] ↵		
	parm	: A, B, X, Y, H, L, PC, BSR0, BSR1, BEF, BCF, C	

Break Commands			
1	**SBC**	Set Break Condition Register	3-70
	SBC [Δ *parm* ..... Δ *parm*] ↵		
	parm	: [˜] *mnemonic*	
2	**DBC**	Display Break Condition Register	3-70
	DBC ↵		
3	**DBP**	Display Break Point Bits	3-73
	DBP Δ *parm* [Δ *parm* ..... Δ *parm*]↵ DBP Δ * ↵		
	parm	: *address* : [*address* Δ *address*]	
4	**CBP**	Change Break Point Bits	3-73
	CBP Δ *parm* [Δ *parm* ..... Δ *parm*] ↵ CBP Δ * [= *data*] ↵		
	parm	: *address* = *data* : [*address* Δ *address*] = *data* *data* : 0, 1	
5	**DBS**	Display Break Status	3-77
	DBS ↵		

Trace Commands

1	**DTM**	Display Trace Memory	3-80
	DTM Δ *parm* ↵ DTM Δ * ↵		
	parm	: – number$_{-step}$ Δ number$_{step}$: number$_{Tp}$ Δ number$_{step}$	
2	**STF**	Set Trace Format	3-86
	STF [Δ *parm* ..... Δ *parm*] ↵		
	parm	: [˜] *mnemonic*	
3	**CTO**	Change Trace Object	3-89
	CTO Δ *parm* [Δ *parm* [Δ *parm*]] ↵		
	parm	: *mnemonic*	
4	**DTO**	Display Trace Object	3-89
	DTO ↵		
5	**STT**	Set Trace Trigger	3-92
	STT Δ *mnemonic1* ↵ STT Δ *mnemonic2* [/ [*parm1*] / [*parm2*]] ↵ *parm1, parm2* : *address* : [*start_address* Δ *end_address*] : . STT Δ *mnemonic3* *trc_mnem* [*&mask*] = *data* ↵		
	mnemonic1 : ALL, TR, DIS *mnemonic2* : *SS mnemonic3* : AD, BD *parm1* : *starting address of trace, parm2* : *ending address of trace* *trc_mnem* : PRB (probe), RAM [Δ *ram_address*](data RAM)		
6	**DTT ↵**	Display Trace Trigger	3-96
	DTT ↵		

Trace Commands (continued)

7	**DTR**	Display Trace Enable bits	3-97
	DTR Δ *parm* [Δ *parm* ..... Δ *parm*]↵ DTR Δ * ↵		
	parm	: *address* : [*address* Δ *address*]	
8	**CTR**	Change Trace Enable bits	3-99
	CTR Δ *parm* [Δ *parm* ..... Δ *parm*]↵ CTR Δ * [= *data*]↵		
	parm	: *address* = *data* : [*address* Δ *address*] = *data* *data* : 0, 1	
9	**DTP**	Display Trace Pointer	3-101
	DTP ↵		
10	**RTP**	Reset Trace Pointer	3-101
	RTP ↵		
11	**S**	Search Trace Memory	3-103
	S [˜] *mnemonic data* [*parm*] ↵		
	mnemonic *data* *parm*	: : *search data (comparison data)* : [*count*], [*start_count* Δ *end_count*]	

Reset Commands			
1	**RST**	Reset the System	3-106
	RST ↵		
2	**RST E**	Reset the Evaluation chip	3-107
	RST Δ E ↵		
3	**URST**	Set User Reset Terminal	3-108
	URST [Δ *mnemonic*]↵		
	mnemonic : ON, OFF		

Performance/Coverage Commands

1	**DCC**	Display Cycle Counter	3-110
	DCC ↵		
2	**CCC**	Change Cycle Counter	3-111
	CCC Δ [–] *data* ↵		
3	**TIME**	Set Unit Time	3-112
	TIME [Δ *data*] ↵		
4	**SCT**	Set Cycle Counter Trigger	3-113
	SCT [Δ / [*parm1*] / [*parm2*]] ↵		
	parm1, parm2 : *address* : [*start_address* Δ *end_address*] : . *parm1 : start status, parm2 : stop status*		
5	**DCT**	Display Cycle Counter Trigger	3-116
	DCT ↵		
6	**RCT**	Reset Cycle Counter Trigger	3-116
	RCT ↵		

Performance/Coverage Commands (continued)

7	**DIE**	Display Instruct ion Executed bits	3-118
	DIE Δ *parm* [Δ *parm* ..... Δ *parm*]↵ DIE Δ * ↵		
	parm	: *address* : [*address* Δ *address*]	
8	**CIE**	Change Instruct ion Executed bits	3-119
	CIE Δ *parm* [Δ *parm* ..... Δ *parm*]↵ CIE Δ * [= *data*]↵		
	parm	: *address* = *data* : [*address* Δ *address*] = *data* *data* = 0, 1	
9	**DAP**	Display Address Pass Counter	3-121
	DAP [Δ *mnemonic* ..... Δ *mnemonic*] ↵		
	mnemonic	: C0, C1, C2, C3	
10	**CAP**	Change Address Pass Counter	3-122
	CAP Δ *mnemonic* [=*address*][Δ *count*] ↵		
	mnemonic	: C0, C1, C2, C3	

EPROM Programming Commands			
1	**TYPE**	Set EPROM Type	3-124
	TYPE [Δ *mnemonic*] ↵		
2	**PPR**	Program EPROM	3-126
	PPR Δ [*address* Δ *address*] [Δ *eprom_address*] ↵ PPR Δ * ↵		
3	**TPR**	Transfer EPROM into Program Memory	3-128
	TPR Δ [*address* Δ *address*] [Δ *CM_address*]↵ TPR Δ * ↵		
4	**VPR**	Verify EPROM with Program Memory	3-130
	VPR Δ [*address* Δ *address*] [Δ *eprom_address*]↵ VPR Δ * ↵		

Commands for Automatic Command Execution			
1	**BATCH**	Batch Processing	3-134
	BATCH ∆ *fname* ↵		
	fname : [*Pathname*] *Filename* [*Extension*]		
2	**PAUSE**	Pause Command Input	3-135
	PAUSE ↵		

Commands for Displaying/Changing/Removing Symbols			
1	**DSYM**	Display Symbol	3-138
	DSYM Δ *string* [Δ *string* .....Δ *string*] ↵ DSYM Δ * ↵		
2	**CSYM**	Change Symbol	3-140
	CSYM Δ *parm* [, *parm* ..... , *parm*] ↵		
	parm	: *string* [= *data*]	
3	**RSYM**	Remove Symbol	3-142
	RSYM Δ *string* [Δ *string* .....Δ *string*] ↵ RSYM Δ * ↵		

Other Commands

1	**LIST**	Listing. Redirect the Console output to Disk file	3-144
	LIST Δ *fname* ↵		
2	**NLST**	No Listing. Cancel the Console output Redirection	3-145
	NLST ↵		
3	**SH**	Call OS Shell	3-146
	SH ↵		
4	**RADIX**	Numeral Radix	3-148
	RADIX Δ *mnemonic* ↵		
	mnemonic : H, D, O, B		
5	**MAC**	Macro Command	3-149
	MAC [Δ [˜] *macro_command*] ↵		
6	**EXIT**	Terminate the Debugger and Exit to OS	3-153
	EXIT ↵		

2-3-2. Symbolic Input (Definition of Expressions)

As explained in Section 2-1-11, symbols and operators can be used for all numeric inputs of the SID64K debugger. These numeric inputs are called *expressions* in this manual.

Expressions are configured from symbols, numeric constants, and operators. Any number of spaces or tabs (shown as Δ below) can be included between these elements.

The input format of expressions is as follows.

expression	: =	***constant*** *or* ***symbol***
expression	: =	p [Δ] ***expression***
expression	: =	***expression*** [Δ] R [Δ] ***expression***
expression	: =	([Δ] ***expression*** [Δ])

Elements enclosed in brackets [] can be omitted. White space, indicated by Δ, follows the explanation of Section 2-3-1. The **: =** indicates that the left side is defined by the right side. The “p” indicates a preceding unary operator. The “R” indicates a relational operator.

Symbols, constant expressions, preceding unary operators, and relational operators are defined below.

❑ Symbols

A symbol is string of characters

- that starts with a character that has the letter attribute explained in Section 2-3-1-1, "Character Set,"
- with the remainder as characters with the letter attribute or digits,
- up to a delimiting character, an operator, or any other special character.

The maximum number of characters for a symbol depends on the number of other parameters in an input line. However, if a line consists of only one symbol, then its maximum is found as follows.

Input line buffer maximum characters - 1 = 71 characters

A symbol has its own value and segment attribute. These are defined at one of the following four times.

1. When symbol information is read from an ASM64K-generated file during execution of a **LOD** command.
2. When the symbol is defined as a label or defined with an **EQU**, **SET**, **CODE**, or **DATA** directive during an **ASM** command.
3. When reading from a DCL file after SID64K is invoked (☞1).
4. When changed with the **CSYM** (Change Symbol) command.

Symbol information is stored in a memory area managed by SID64K. This memory area is known as the symbol table.

When a symbol is input, the debugger searches the symbol table. If the given symbol is found, then its value will be taken as the value of the input symbol.

During a symbol search, segment attribute checks of symbols are not performed. In other words, if the name of an input symbol matches the name of a symbol in the symbol table, then the debugger will always return the value, even when the requested segment type does not match the segment type of the symbol in the table.

☞ 1 The DCL file contains system information for the target evaluation device (memory and SFR assignment information), as well as reserved keywords. If you need to see what reserved keywords are registered, then refer to the explanation about defining reserved keywords below.

Reserved keywords are defined with the **DEFDATA** statement as bit symbols with the DATA segment attribute. The format for defining them is as follows:

DEFDATA Δ *symbol, expression* ↵

❑ Constants

Constants include integer constants, character constants, and string constants. String constants can only be used with the **ASM** command, so they are not explained in this section. (Refer to Section 5-4-4 for details on string constants.)

Integer constants

Any string with the first character a digit 0 to 9 is handled as an integer constant. Integer constants can be expressed with radix 2 (binary), 8 (octal), 10 (decimal), or 16 (hexadecimal). In order to distinguish between these expression formats (radices), the number is appended with a radix operator, as shown in Table 2-2.

When the radix operator (H, D, O, Q, B) is omitted, the radix specified with the **RADIX** command will be followed. However, if a *number*, *bank*, or *count* is expressed in input, then it will be recognized as decimal regardless of the radix operator.

When the default radix specified with the **RADIX** command is hexadecimal, then binary expressions are not permitted. Conversely, if the default radix specified is binary, then hexadecimal expressions are not permitted.

If the first digit of a hexadecimal expression is a letter A-F, then it must be preceded with a 0 in order to distinguish it from a symbol.

Table 2-6. Format of Integer Constants

Radix	Usable Characters	Radix Operator	Examples
2 (binary)	0, 1	B	1010B 01101101B 1001_1001B
8 (octal)	0 to 7	O, Q	271O 514Q
10 (decimal)	0 to 9	D	30D 1263
16 (hexadecimal)	0 to 9, A to F	H	753H 0C6E7H

Character constants

A character constant is a constant enclosed in single quotation marks (‘) or an escape sequence. A character constant formed as a character other than backslash (\) enclosed in single quotation marks will take that character's ASCII code as its value. When a single quotation mark is followed by a backslash (\), then the value 00H–0FFH will be given in accordance with the following code. The backslash and the code that follows it is called an escape sequence. Escape sequences are only used with the **ASM** command, so they are not explained in detail here. Refer to Chapter 5, "Assembler Command Details."

❑ Operators

The **ASM** command permits all C language-compatible operators, but other commands only allow the binary operators '+' (addition) and '–' (substraction), and parentheses. A detailed explanation of operators is given in Chapter 5, so it is omitted here.

All calculations are performed as 32-bit unsigned format. If a calculation result is negative, then it will be expressed in 2's complement. Overflows will be ignored. A value of expression will be the calculated result of the operators acting on the values of symbols and integer constants.

Below are shown some examples of expressions using symbols and operators.

Example

Example 1

```
DCM  [LOOP1 + 10  LOOP2]
```

Displays the values of code memory from address LOOP1 + 10 to LOOP2.

Example 2

```
C   PC = DATA1   SP = MAX - 10
```

Changes the contents of the PSW to DATA1. Changes the contents of SP to MAX - 10.

Example 3

```
DATA1  EQU   200H
CAL    DATA1 & DATA2
```

Defines DATA1 as 200H within the assemble command. Incorporates the result of evaluating DATA1 & DATA2 (the logical AND of DATA1 and DATA2) as immediate data.

2-3-3. History Function

SID64K has a function for saving previous command line input (☞1). This function is known as the history function.

When using the debugger, occasionally you will want to input the same command as one several previous, or the same command except with different parameters. This is when the history function is especially powerful.

(1) Current line buffer and history buffer

SID64K has a current line buffer for editing the current command line input and a history buffer for saving command lines.

The command line buffer is a 72-character buffer for command line input. The history buffer is a 72-character by 20-line buffer for storing command line input in order.

There are two types of history buffers. One is for normal command line input, and one is for command line input during execution of the **ASM** command.

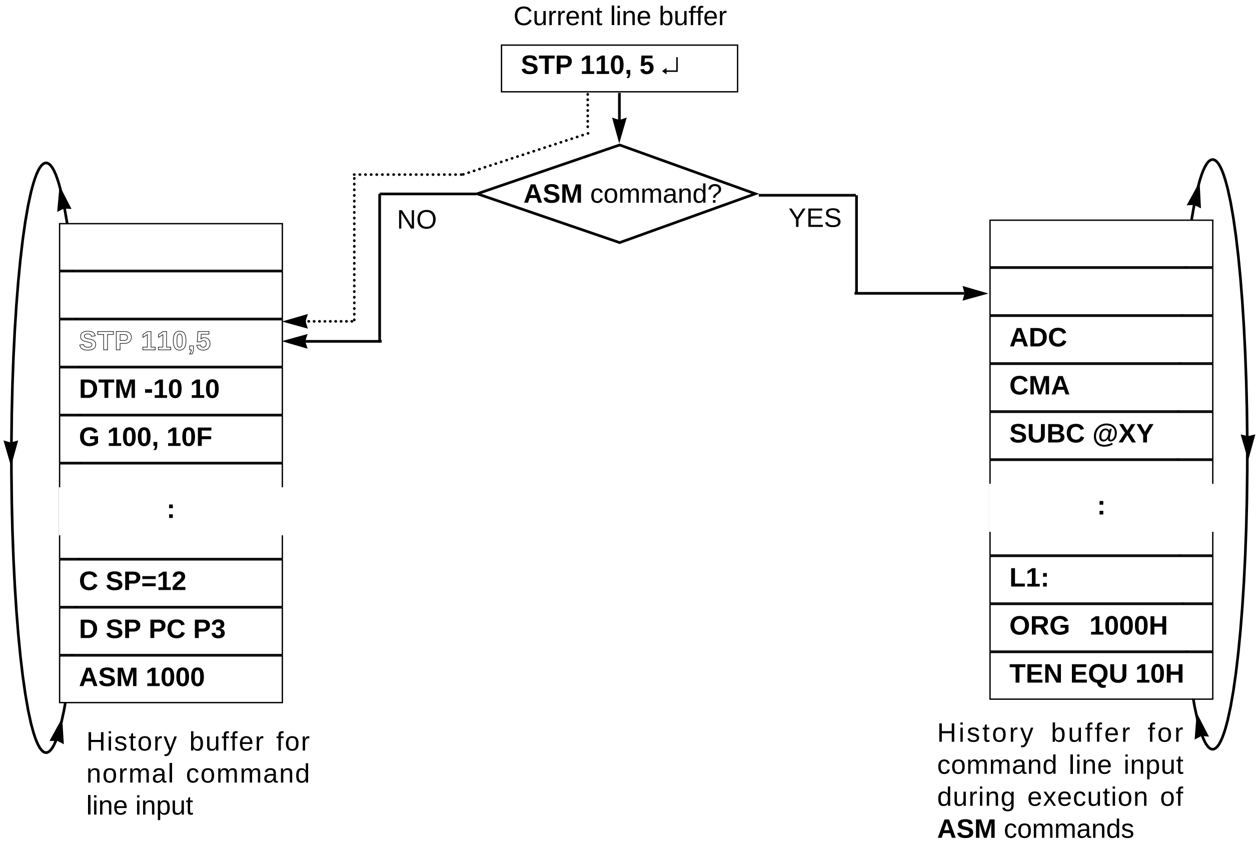


Figure 2-5. Current Line Buffer and History Buffer

A command line input by an operator is first stored in the current line buffer. Simultaneous to the operator pressing a carriage return, the contents of the current line buffer are stored in the history buffer. Each time a command line is input, its contents are stored in order in the history buffer.

The history buffer is configured as a ring. The oldest input line (the command line input 20 lines before the current command line input) is overwritten. As a result, the previous 10 lines of command line input will always be stored.

The operator can read the contents of the history buffer into the current line buffer at any time during command line input.

Note that input from a file called by the **BATCH** command will not be stored in the history buffer.

❏ Using history functions

This somewhat covers the same material as Section 2-3-4, "Special Keys For Raising Command Input Efficiency," but the history functions are utilized with the ↑ key (or **CTRL + K**) and the ↓ key (or **CTRL + J**).

Pressing the ↑ key will read the immediately previous command line input from the history buffer into the command line buffer and display it on the console. Then each time the ↑ key is pressed, the next previous command line input will be read and displayed.

Converse to the ↑ key, the ↓ key reads the command line input immediately afterward from the history buffer and displays it on the console.

After the operator has edited the displayed current line buffer contents with the special keys for command line editing, as explained in the next section, he can enter it as the new command line input by pressing the ↵ key. At this time, the current line buffer will be executed to its end as the command line input, regardless of the cursor position on the line.

Of course, the contents of the current line buffer can be executed if only the ↵ key is pressed without any editing.

☞ 1 SID64K command line input is input from the console after the SID64K output prompt "64153>" or "Go>>", and during **ASM** command execution.

2-3-4. Special Keys For Raising Command Input Efficiency

SID64K provides special editing keys, as mentioned in the previous section on the history function, for raising efficiency of current line buffer editing. There are a total of 12 special keys. They can effectively create new command line inputs. The special keys and their control functions are explained below.

(1) **CTRL+A** and **CTRL+Z**

CTRL+A moves the cursor to the first location of the current line buffer.

CTRL+Z moves the cursor to the last location of the current line buffer.

Example

Contents of current line buffer before editing

```
STP 1000,10█
```

CTRL + A pressed

Contents of current line buffer after editing

```
STP 1000,10
```

CTRL + Z pressed

Contents of current line buffer after editing

```
STP 1000,10█
```

(2) **CTRL+B** and **CTRL+F**

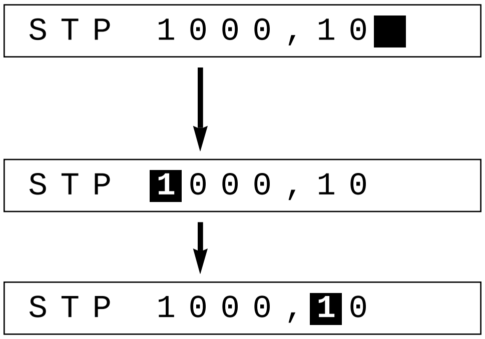
CTRL+B searches for a string *consisting of letters and digits only* from the current cursor location in the current line buffer toward the first location. In other words, it recognizes characters other than letters and digits as string delimiters.

If a string is detected, then the cursor will be moved to its first location. If no string could be detected, then the cursor will be moved to the first location of the current line buffer.

CTRL+F searches for a string *consisting of letters and digits only* from the current cursor location in the current line buffer toward the last location. In other words, it recognizes characters other than letters and digits as string delimiters.

If a string is detected, then the cursor will be moved to its first location. If no string could be detected, then the cursor will be moved to the last location of the current line buffer.

Example



Contents of current line buffer before editing

```
STP 1000,10█
```

CTRL + B pressed
CTRL + B pressed

Contents of current line buffer after editing

```
STP 1000,10
```

CTRL + F pressed

Contents of current line buffer after editing

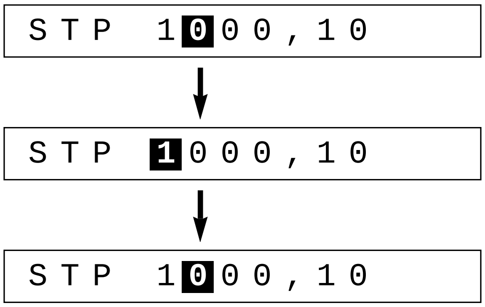
```
STP 1000,10
```

(3) **CTRL+H** (or ←) and **CTRL+L** (or →)

CTRL+H moves the cursor one location to the left of its current location in the current line buffer.

CTRL+L moves the cursor one location to the right of its current location in the current line buffer.

Example



Contents of current line buffer before editing

```
STP 1000,10
```

CTRL + H or ← pressed

Contents of current line buffer after editing

```
STP 1000,10
```

CTRL + L or → pressed

Contents of current line buffer after editing

```
STP 1000,10
```

(4) **CTRL+K** (or ↑) and **CTRL+J** (or ↓)

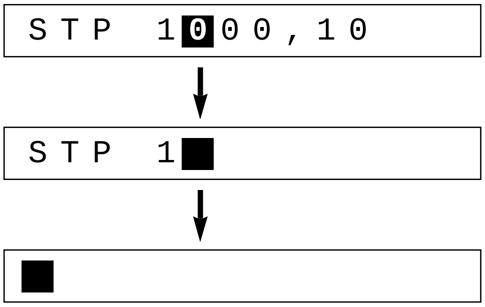
CTRL+K (or ↑) and **CTRL+J** (or ↓) read history buffer contents into the current line buffer, as explained in the previous section. For details, refer to the previous Section 2-3-3, “History Function.”

(5) **CTRL+D** and **CTRL+X**

CTRL+D deletes current line buffer contents from the current cursor position to the last location, and then moves the cursor to the end of the line.

CTRL+X deletes the current line buffer contents, and then moves the cursor to the start of the buffer.

Example



Contents of current line buffer before editing

```
STP 1000,10
```

CTRL + D pressed

Contents of current line buffer after editing

```
STP 1█
```

CTRL + X pressed

Contents of current line buffer after editing

```
█
```

(6) **CTRL+R** (or **INS**) and **DEL**

CTRL+R (or **INS**) inserts a single blank character at the current cursor position in the current line buffer.

DEL deletes a singles character at the current cursor position in the current line buffer. The cursor position does not change.

Example

Contents of current line buffer before editing

```
STP 1000,10
```

CTRL + R or **INS** pressed

↓

Contents of current line buffer after editing

```
STP 1█000,10
```

DEL pressed

↓

Contents of current line buffer after editing

```
STP 1000,10
```

If you will use SID64K with an IBM PC-AT, then add the appropriate ANSI escape sequence driver from your DOS system disk to CONFIG.SYS. If you forget to do so, then you will not be able to use the special editing keys.

Host Computer	ANSI Escape Sequence Driver Name
IBM PC-AT	ANSI.SYS

To use the ↑ , ↓ , ← , → , **INS** and **DEL** keys, set your host computer's key table to the same key code settings as in the table on the next page. If the settings do not match, then the danger exists that a special key function will operate differently. There is no need to set them for the IBM PC-AT, but for the NEC PC-9801 change the key table file to **KEY.TBL** using the MS-DOS utility program **KEY.EXE**.

The table below shows the special editing keys and how they affect the contents of the current line buffer. It also shows the SID64K internal processing code (in hexadecimal) for each key. Check the settings of your host computer's key table, and if they do not match these settings, then change them to match.

In the table, "line" means the current line buffer.

Editing Key	***Control Function***	***Code***
CTRL + A	Moves the cursor to the start of the current line buffer.	01H
CTRL + B	Searches for string of letters and digits only from the current cursor location to the first location, and moves the cursor to the start of the string.	02H
CTRL + D	Deletes all characters from the current cursor location to the last location.	04H
CTRL + F	Searches for string of letters and digits only from the current cursor location to the first location, and moves the cursor to the start of the string.	06H
CTRL + J or ↓	Reads the next command line input from the history buffer into the current line buffer and displays it.	0AH
CTRL + K or ↑	Reads the previous command line input from the history buffer into the current line buffer and displays it.	0BH
CTRL + H or ←	Moves the current cursor position one to the left.	08H
CTRL + L or →	Moves the current cursor position one to the right.	0CH
CTRL + X	Deletes the current line buffer, and moves the cursor to the first location.	18H
CTRL + Z	Moves the cursor to the end of the current line buffer.	1AH
CTRL + R or **INS**	Inserts a single blank at the current cursor location.	12H
DEL	Deletes a character at the current cursor location.	7FH

Chapter 3

SID64K Commands

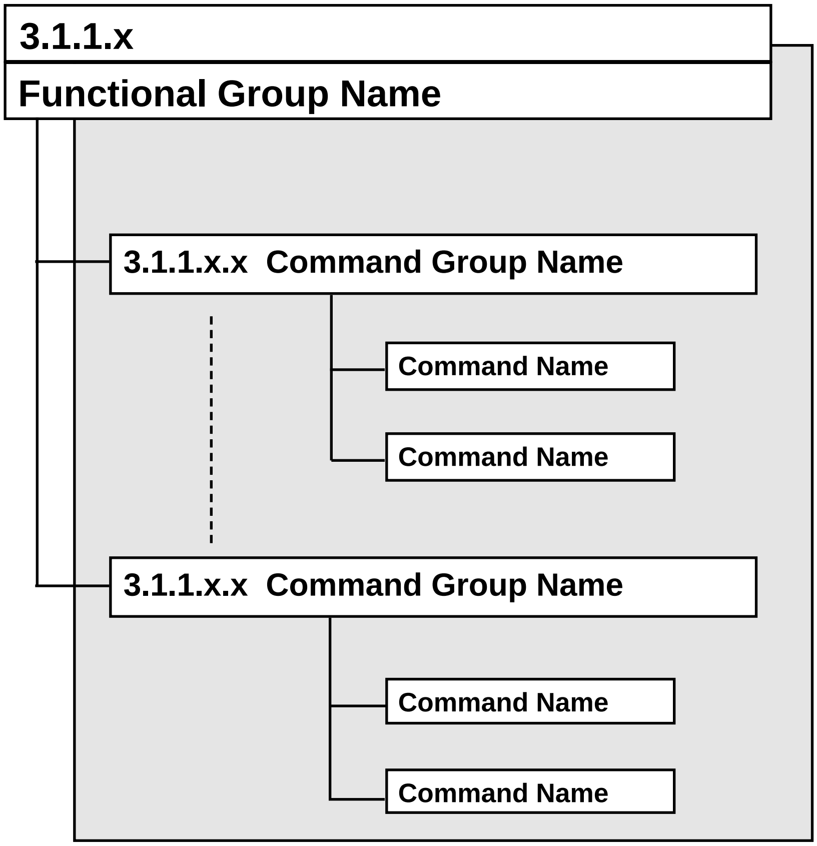
This chapter explains in detail how to use SID64K commands.

3-1. SID64K Commands

3-1-1. Command Details

This chapter explains the SID64K commands organized by function.

A list of contents like the one shown below is given at the start of each functional grouping. At the top is a two-line title box outlining the name of the functional group. Below it are the names of the command groups covered by the functional group, outlined in one-line title boxes. Under each command group are the names of the commands it covers.



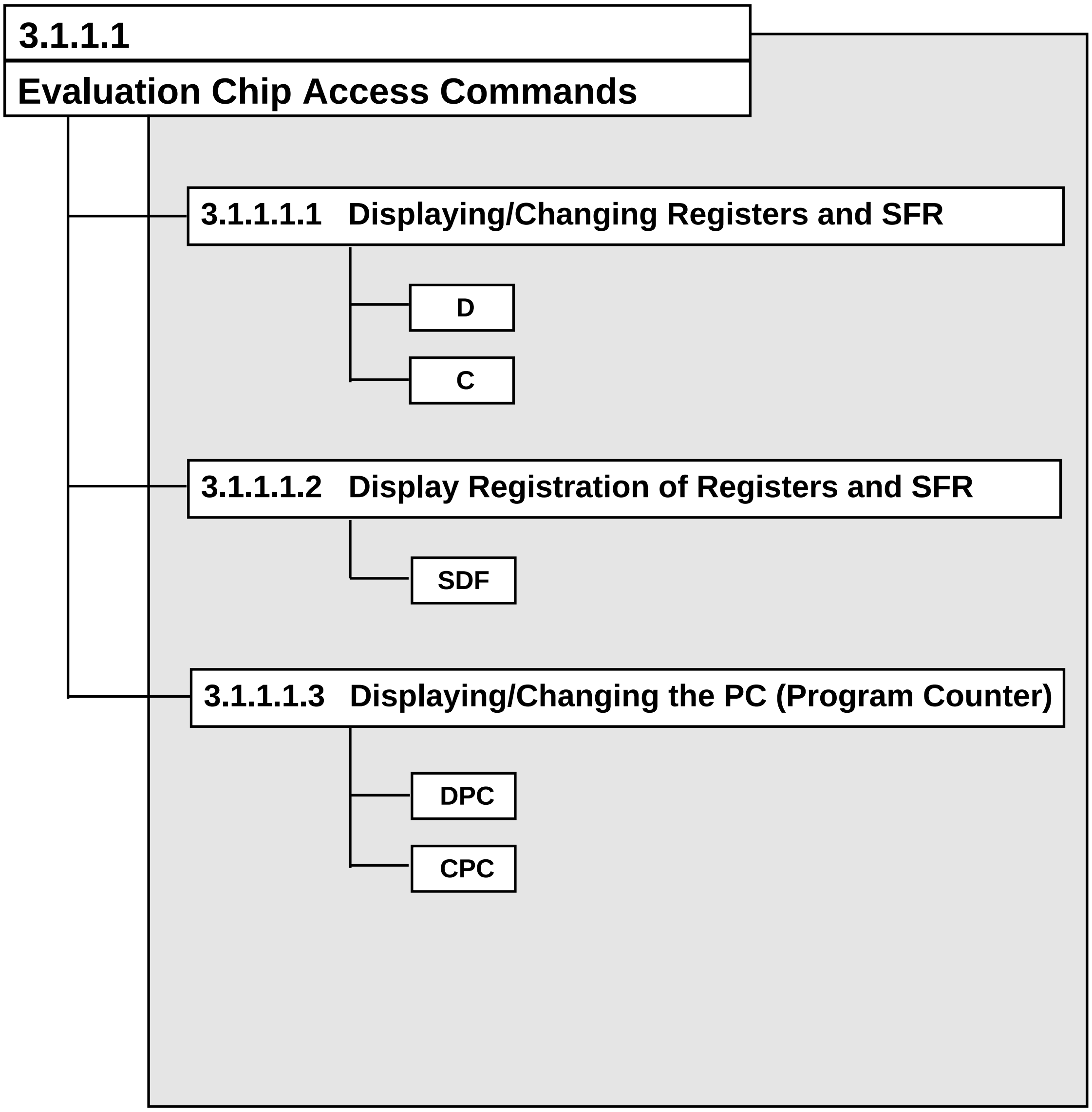
The header of each page shows the name of the command explained on that page in boldface and enclosed in a rectangle. This is provided for convenience when looking up command explanations.

Each command is explained in the order of input format, description, and execution example. These are given under the following respective title lines.

Input Format

Description

Execution Example


3.1.1.1
Evaluation Chip Access Commands
3.1.1.1.1   Displaying/Changing Registers and SFR
D
C
3.1.1.1.2   Display Registration of Registers and SFR
SDF
3.1.1.1.3   Displaying/Changing the PC (Program Counter)
DPC
CPC

3.1.1.1.1 Displaying/Changing Registers and SFR

D, C

Input Format

D [Δ *mnemonic* ..... Δ *mnemonic*] ↵ ◄·········· *Display command*

C Δ *parm* [Δ *parm* ..... Δ *parm*] ↵ ◄··············· *Change command*

parm : *mnemonic* [= *data*]
mnemonic : *SFR-mnemonic*
or
Register-mnemonic
or
PC (program counter), C (carry flag),
BSR0 (bank select register 0),
BSR1 (bank select register 1),
BCF (bank common flag), BEF (bank enable flag)

Description

The **D** command displays the contents of the the register or SFR specified by *mnemonic*. The *mnemonic* can be an *SFR-mnemonic* indicating an SFR register, a *Register-mnemonic* indicating a general-purpose register, or PC or C, or BSR0, or BSR1, or BEF, or BCF.

A *Register-mnemonic* is one of the following expressions.

Register-mnemonic	:	A	(A register)
	:	B	(B register)
	:	H	(H register)
	:	L	(L register)
	:	X	(X register)
	:	Y	(Y register)
	:	BA	(B register and A register)
	:	HL	(H register and L register)
	:	XY	(X register and Y register)

An *SFR-mnemonic* is one of the mnemonics shown in Table 3-1. Table 3-1 shows all the SFRs included in the MSM64E153 evaluation chip. For actual use, refer to the user's manual of the appropriate MSM64153 family microcontroller.

If no parameters are input, then the contents of the general-purpose registers, PC, C, and any SFR registered with the **SFR** command will be displayed.

The **C** command changes the contents of the the register or SFR specified by *SFR-mnemonic*, *Register-mnemonic*, PC, C, BSR0, BSR1, BEF, or BCF. The format of *Register-mnemonic* is the same as that of the **D** command.

An *SFR-mnemonic* is one of the mnemonics shown in Table 3-1. The *data* is an expression that must evaluate to a value in the range 0–0FFH for byte access, or 0–0FH for nibble access. If *data* is omitted, then the emulator outputs the following message and waits for data to be input.

```
        mnemonic:  old-data  -------►
```

Here *mnemonic* expresses the mnemonic of the SFR or register that is to have its current contents changed. The *old-data* will be the current contents. At this point the operator enters new data (*data*) and inputs a carriage return.

```
        mnemonic:   old-data  --------►  data ↲
        mnemonic:   old-data  --------►  ◄-------------  input data for next parameter
```

When the carriage return is input, processing moves to the next parameter. If there is no next parameter, then the **C** command terminates.

When the emulator is waiting for input data for a change, the following two key inputs are valid in addition to *data*.

Δ ↲ (space followed by carriage return)	------------►	Move processing to the next parameter without changing the current data. If there is no next parameter, then the **C** command terminates.
↲ (input carriage return only)	---------------------►	The **C** command terminates.

If bit symbols are defined for an SFR mnemonic input with the **D** command, then the the contents of each bit expressed by a bit symbol will be displayed simultaneously with the SFR contents.

If bit symbols are defined for an SFR mnemonic input with the **C** command, and if *data* is omitted when the **C** command is input, then input mode will be entered for each bit expressed by a bit symbol.

Table 3-2 shows the bit symbols defined for the SID64K. The table shows all SFR bit symbols included in the MSM64E153, so for actual use refer to the user's manual of the appropriate MSM64153 family microcontroller.

Table 3-2 shows the values assigned to each bit symbol. The values are given by the following arithmetic expression.

Bit symbol value = SFR address x 4 + bit position

Example: The value of P3F (bit 2) of P3CON1 (0DH)
P3F bit symbol value = 0DH x 4 + 2
36H

The values assigned to respective bit symbols are used by the SID64K command interpreter. Bit symbols can also be used during command input.

Example: “DCM P3F” is the same as “DCM 36H.”

☞ 1

‘**D** ↵’ will normally display the contents of **PC**, **C**, general-purpose registers, and the **BCF**, **BEF**, **BSR0**, and **BSR1**.

!

For several SFRs of the MSM64153 family, unallocated bits exist as reserved bits. For the EASE64158, these reserved bits are handled as shown below. Refer to the user’s manual of the appropriate MSM64153 family microcontroller regarding reserved bit contents.

D command: Reserved bits are displayed as “1.”
C command: Reserved bits will not change even if set to “0” or “1” data.

Table 3-1. List of SFR-mnemonics (☞1)

SFR-mnemonic	*Register Name*	*Address*
P0	Port 0 Register	00H
P1	Port 1 Register	01H
P2	Port 2 Register	02H
P3	Port 3 Register	03H
P4D	Port 4 Data Register	04H
P5D	Port 5 Data Register	05H
P6D	Port 6 Data Register	06H
P7D	Port 7 Data Register	07H
P2CON0	Port 2 Control Register 0	08H
P2CON1	Port 2 Control Register 1	09H
P2IE	Port 2 Interface Enable Register	0AH
P3CON0	Port 3 Control Register 0	0CH
P3CON1	Port 3 Control Register 1	0DH
P6CON	Port 6 Control Register	0EH
P7CON	Port 7 Control Register	0FH
BDCON	Buzzer Control Register	10H
TBCR	Time-Base Counter Register	11H
DSPCON0	Display Control Register 0	12H
DSPCON1	Display Control Register 1	13H
BUPCON	Backup Control Register	14H
BATCON	Battery Check Control Register	15H
CAPCON	Capture Control Register	17H
CAPR0	Capture Register 0	18H
CAPR1	Capture Register 1	19H
ECLR	Event Counter Low Register	1AH
ECHR	Event Counter High Register	1BH
ECCON	Event Counter Control Register	1CH
_100HzCON	100 Hz Control Register	1DH
_100HzC	100 Hz Count Register	1EH
_10HzC	10 Hz Count Register	1FH
ADCON0	A/D Converter Control Register 0	20H
ADCON1	A/D Converter Control Register 1	21H
CNTAL	A/D Converter CounterA Register Low	22H
CNTAM	A/D Converter CounterA Register Middle	23H
CNTAH	A/D Converter CounterA Register High	24H

ADCON2	A/D Converter Control Register 2	25H
CNTBL	A/D Converter Counter B Register Low	26H
CNTBM	A/D Converter Counter B Register Middle	27H
CNTBH	A/D Converter Counter B Register High	28H
MDCON0	Melody Control Register 0	2AH
TEMP0	Melody Tempo Register 0	2BH
MDR00	Melody Data Register 00	2CH
MDR01	Melody Data Register 01	2DH
MDR02	Melody Data Register 02	2EH
MDR03	Melody Data Register 03	2FH
MDCON1	Melody Control Register 1	30H
TEMP1	Melody Tempo Register 1	31H
MDR10	Melody Data Register 10	32H
MDR11	Melody Data Register 11	33H
MDR12	Melody Data Register 12	34H
MDR13	Melody Data Register 13	35H
IE0	Interrupt Enable Register 0	38H
IE1	Interrupt Enable Register 1	39H
IE2	Interrupt Enable Register 2	3AH
IE3	Interrupt Enable Register 3	3BH
IRQ0	Interrupt Request Register 0	3CH
IRQ1	Interrupt Request Register 1	3DH
IRQ2	Interrupt Request Register 2	3EH
IRQ3	Interrupt Request Register 3	3FH
DSPR0	Display Register 0	40H
DSPR1	Display Register 1	41H
DSPR2	Display Register 2	42H
DSPR3	Display Register 3	43H
DSPR4	Display Register 4	44H
DSPR5	Display Register 5	45H
DSPR6	Display Register 6	46H
DSPR7	Display Register 7	47H
DSPR8	Display Register 8	48H
DSPR9	Display Register 9	49H
DSPR10	Display Register 10	4AH
DSPR11	Display Register 11	4BH
DSPR12	Display Register 12	4CH
DSPR13	Display Register 13	4DH

DSPR14	Display Register 14	4EH
DSPR15	Display Register 15	4FH
DSPR16	Display Register 16	50H
DSPR17	Display Register 17	51H
DSPR18	Display Register 18	52H
DSPR19	Display Register 19	53H
DSPR20	Display Register 20	54H
DSPR21	Display Register 21	55H
DSPR22	Display Register 22	56H
DSPR23	Display Register 23	57H
DSPR24	Display Register 24	58H
DSPR25	Display Register 25	59H
DSPR26	Display Register 26	5AH
DSPR27	Display Register 27	5BH
DSPR28	Display Register 28	5CH
DSPR29	Display Register 29	5DH
DSPR30	Display Register 30	5EH
DSPR31	Display Register 31	5FH
DSPR32	Display Register 32	60H
DSPR33	Display Register 33	61H
DSPR34	Display Register 34	62H
DSPR35	Display Register 35	63H
DSPR36	Display Register 36	64H
DSPR37	Display Register 37	65H
DSPR38	Display Register 38	66H
DSPR39	Display Register 39	67H
DSPR40	Display Register 40	68H
DSPR41	Display Register 41	69H
DSPR42	Display Register 42	6AH
DSPR43	Display Register 43	6BH
DSPR44	Display Register 44	6CH
DSPR45	Display Register 45	6DH
DSPR46	Display Register 46	6EH
DSPR47	Display Register 47	6FH
DSPR48	Display Register 48	70H
DSPR49	Display Register 49	71H
DSPR50	Display Register 50	72H
DSPR51	Display Register 51	73H
DSPR52	Display Register 52	74H

DSPR53	Display Register 53	75H
DSPR54	Display Register 54	76H
DSPR55	Display Register 55	77H
DSPR56	Display Register 56	78H
DSPR57	Display Register 57	79H
DSPR58	Display Register 58	7AH
DSPR59	Display Register 59	7BH
MIEF	Master Interrupt Enable Register	7CH
HALT (☞ 2)	Halt Mode Register	7DH
SP (☞ 3)	Stack Pointer	7EH, 7DH

☞ 1 Table 3-2 shows all SFRs included in the MSM64E153. For actual use refer to the user's manual of the appropriate MSM64153 family microcontroller. The symbols shown for special function registers are their mnemonics. However, when the first character is a digit, prefix the digit with an underscore (_).

Example: 100HzCON ⟶ _100HzCON

☞ 2 In HALT mode, change commands are invalid (the HALT mode will be forcibly released when emulation is not executed).

☞ 3 The SP (stack pointer) can only be changed with the **C** command. It cannot be changed with the **CDM** command, described later.

Table 3-2. (a) Bit Symbols

SFR-mnemonic	Bit Symbols			
	bit 3	bit 2	bit 1	bit 0
P0	P03	P02	P01	P00
P1	P13	P12	P11	P10
P2	P23	P22	P21	P20
P3	*–	*–	P31	P30
P4D	P43	P42	P41	P40
P5D	P53	P52	P51	P50
P6D	P63	P62	P61	P60
P7D	P73	P72	P71	P70
P2CON0	P23MOD	P22MOD	P21MOD	P20MOD
P2CON1	*–	*–	*–	P2F
P2IE	P23IE	P22IE	P21IE	P20IE
P3CON0	*–	*–	P31MOD	P30MOD
P3CON1	*–	P3F	P3EXT1	P3EXT0
P6CON	*–	P6F	P6MOD	P6DIR
P7CON	*–	P7F	P7MOD	P7DIR
BDCON	SELF	EBD	BM1	BM0
TBCR	_1Hz	_2Hz	_4Hz	_8Hz
DSPCON0	DTRN1	DTRN0	ONOFF	DUTY
DSPCON1	*–	*–	HTRN	STRN
BUPCON	*–	*–	*–	BUPF
BATCON	*–	*–	EBAT	BATF
CAPCON	CRF1	CRF0	ECAP1	ECAP0
CAPR0	_32Hz0	_64Hz0	_128Hz0	_256Hz0
CAPR1	_32Hz1	_64Hz1	_128Hz1	_256Hz1
ECLR	EC3	EC2	EC1	EC0
ECHR	EC7	EC6	EC5	EC4
ECCON	*–	*–	ECOVF	EEC
_100HzCON	*–	*–	*–	ECNT
_100HzC	_100Hz3	_100Hz2	_100Hz1	_100Hz0
_10HzC	_10Hz3	_10Hz2	_10Hz1	_10Hz0

Note: Table entries marked *– are reserved bits, which are currently unassigned.

Table 3-2. (b) Bit Symbols

SFR-mnemonic	**Bit Symbols**			
	bit 3	**bit 2**	**bit 1**	**bit 0**
ADCON0	OM2	OM1	OM0	EADC
ADCON1	BSTP	ASTP	OVFB	OVFA
CNTAL	a3	a2	a1	a0
CNTAM	a7	a6	a5	a4
CNTAH	a11	a10	a9	a8
ADCON2	*–	*–	*–	OSCOUT
CNTBL	b3	b2	b1	b0
CNTBM	b7	b6	b5	b4
CNTBH	b11	b10	b9	b8
MDCON0	*–	*–	*–	MSA0
TEMP0	TP03	TP02	TP01	TP00
MDR00	L03	L02	L01	L00
MDR01	*–	END0	L05	L04
MDR02	N03	N02	N01	N00
MDR03	*–	N06	N05	N04
MDCON1	*–	*–	*–	MSA1
TEMP1	TP13	TP12	TP11	TP10
MDR10	L13	L12	L11	L10
MDR11	*–	END1	L15	L14
MDR12	N13	N12	N11	N10
MDR13	*–	N16	N15	N14
IE0	EP3	EMD1	EMD0	EEX1
IE1	EAD	EP7	EP6	EP2
IE2	E16Hz	E32Hz	E128Hz	E256Hz
IE3	EEX0	EP01	E1Hz	E4Hz
IRQ0	QP3	QMD1	QMD0	QEX1
IRQ1	QAD	QP7	QP6	QP2
IRQ2	Q16Hz	Q32Hz	Q128Hz	Q256Hz
IRQ3	QEX0	QP01	Q1Hz	Q4Hz
MIEF	*–	*–	*–	MI
HALT	*–	*–	*–	HLT
SP	SP3	SP2	SP1	*–
	*–	SP6	SP5	SP4

Note: Table entries marked *- are reserved bits, which are currently unassigned.

Table 3-3. (a) Values of Bit Symbols

Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value
P00	00H	P71	1DH	_8Hz	44H	EC2	6AH
P01	01H	P72	1EH	_4Hz	45H	EC3	6BH
P02	02H	P73	1FH	_2Hz	46H	EC4	6CH
P03	03H	P20MOD	20H	_1Hz	47H	EC5	6DH
P10	04H	P21MOD	21H	DUTY	48H	EC6	6EH
P11	05H	P22MOD	22H	ONOFF	49H	EC7	6FH
P12	06H	P23MOD	23H	DTRN0	4AH	EEC	70H
P13	07H	P2F	24H	DTRN1	4BH	ECOVF	71H
P20	08H	P20IE	28H	STRN	4CH	ECNT	74H
P21	09H	P21IE	29H	HTRN	4DH	_100Hz0	78H
P22	0AH	P22IE	2AH	BUPF	50H	_100Hz1	79H
P23	0BH	P23IE	2BH	BATF	54H	_100Hz2	7AH
P30	0CH	P30MOD	30H	EBAT	55H	_100Hz3	7BH
P31	0DH	P31MOD	31H	ECAP0	5CH	_10Hz0	7CH
P40	10H	P3EXT0	34H	ECAP1	5DH	_10Hz1	7DH
P41	11H	P3EXT1	35H	CRF0	5EH	_10Hz2	7EH
P42	12H	P3F	36H	CRF1	5FH	_10Hz3	7FH
P43	13H	P6DIR	38H	_256Hz0	60H	EADC	80H
P50	14H	P6MOD	39H	_128Hz0	61H	OM0	81H
P51	15H	P6F	3AH	_64Hz0	62H	OM1	82H
P52	16H	P7DIR	3CH	_32Hz0	63H	OM2	83H
P53	17H	P7MOD	3DH	_256Hz1	64H	OVFA	84H
P60	18H	P7F	3EH	_128Hz1	65H	OVFB	85H
P61	19H	BM0	40H	_64Hz1	66H	ASTP	86H
P62	1AH	BM1	41H	_32Hz1	67H	BSTP	87H
P63	1BH	EBD	42H	EC0	68H	a0	88H
P70	1CH	SELF	43H	EC1	69H	a1	89H

Table 3-3. (b) Values of Bit Symbols

Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value
a2	8AH	TP02	AEH	L15	CDH	QMD0	F1H
a3	8BH	TP03	AFH	END1	CEH	QMD1	F2H
a4	8CH	L00	B0H	N10	D0H	QP3	F3H
a5	8DH	L01	B1H	N11	D1H	QP2	F4H
a6	8EH	L02	B2H	N12	D2H	QP6	F5H
a7	8FH	L03	B3H	N13	D3H	QP7	F6H
a8	90H	L04	B4H	N14	D4H	QAD	F7H
a9	91H	L05	B5H	N15	D5H	Q256Hz	F8H
a10	92H	END0	B6H	N16	D6H	Q128Hz	F9H
a11	93H	N00	B8H	EEX1	E0H	Q32Hz	FAH
OSCOUT	94H	N01	B9H	EMD0	E1H	Q16Hz	FBH
b0	98H	N02	BAH	EMD1	E2H	Q4Hz	FCH
b1	99H	N03	BBH	EP3	E3H	Q1Hz	FDH
b2	9AH	N04	BCH	EP2	E4H	QP01	FEH
b3	9BH	N05	BDH	EP6	E5H	QEX0	FFH
b4	9CH	N06	BEH	EP7	E6H	MI	1F0H
b5	9DH	MSA1	C0H	EAD	E7H	HLT	1F4H
b6	9EH	TP10	C4H	E256Hz	E8H	SP1	1F9H
b7	9FH	TP11	C5H	E128Hz	E9H	SP2	1FAH
b8	A0H	TP12	C6H	E32Hz	EAH	SP3	1FBH
b9	A1H	TP13	C7H	E16Hz	EBH	SP4	1FCH
b10	A2H	L10	C8H	E4Hz	ECH	SP5	1FDH
b11	A3H	L11	C9H	E1Hz	EDH	SP6	1FEH
MSA0	A8H	L12	CAH	EP01	EEH	SP7	1FFH
TP00	ACH	L13	CBH	EEX0	EFH	–	–
TP01	ADH	L14	CCH	QEX1	F0H	–	–

Execution Example

```
64153> D P3CON1

    P3CON1 : 8
    ( P3F : 0  P3EXT1 : 0  P3EXT0 : 0 )
64153> C A
    A : 0 -----> 5 New
64153> D A
    A       : 5
64153> C A=1 B=2 H=4
64153> C P3CON1
    P3CON1 : 8
    ----- BIT -----
         P3EXT0  : 0 -----> 1 New
         P3EXT1  : 0 -----> 1 New
         P3F     : 0 -----> 1 New
```

Execution Example

```
64153> D
     A    : 1    B  : 2    H   : 4    L   : 0    X    : 0
     Y    : 0    PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
     BSR1 : 0    C  : 0
64153> SDF
       ** Not Set Display Format
64153> SDF P2 P6D P7D
     A    : 1     B  : 2    H   : 4    L   : 0    X    : 0
     Y    : 0     PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
     BSR1 : 0     C  : 0
     P6D  : 0    P7D : 0    P2  : 0
```

3.1.1.1.2 Display Registration of Registers and SFR

SDF

Input Format

SDF [Δ *parm* ..... Δ *parm*] ↵

parm : [˜] *SFR_mnemonic* (☞ 1)

Description

The **SDF** command registers which SFR mnemonics are displayed when the **D** command is input as “D↵”.

SFR-mnemonic is one of those shown in Table 3-1. If the mnemonic is prefixed by ‘~’ (tilde), then its registration will be cancelled.

If *parm* is omitted, then the currently set display format will be displayed.

“SDF↵” displays the registration contents.

Execution Example

```
64153> SDF
    P6D     P7D     P2
64153> SDF P2CON0 IE0 IE2 IE2 IE3 IRQ0 DSPR11
64153> D
    A      : 1    B   : 2    H      : 4    L    : 0    X   : 0
    Y      : 0    PC  : 0000 BCF    : 0    BEF  : 0
    BSR0   : 0    BSR1: 0    C      : 0
    P6D    : 0    P7D : 0    P2CON0 : 0    IRQ0 : 1    P2  : 0
    DSPR11 : 0    IE0 : 1    IE2    : 1    IE3  : D
64153> SDF~P2~P6D~P7D~P2CON0~IE0~IE1~IE2~IE3~IRQ0~DSPR11
64153> D
    A    : 1    B  : 2    H   : 4    L   : 0    X    : 0
    Y    : 0    PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
    BSR1 : 0    C  : 0
64153> SDF
       ** Not Set Display Format
64153>
```

3.1.1.1.3 Displaying/Changing the PC (Program Counter)

DPC, CPC

Input Format

DPC ↵

CPC Δ *address* ↵

Description

The **DPC** command displays the contents of the PC (program counter).

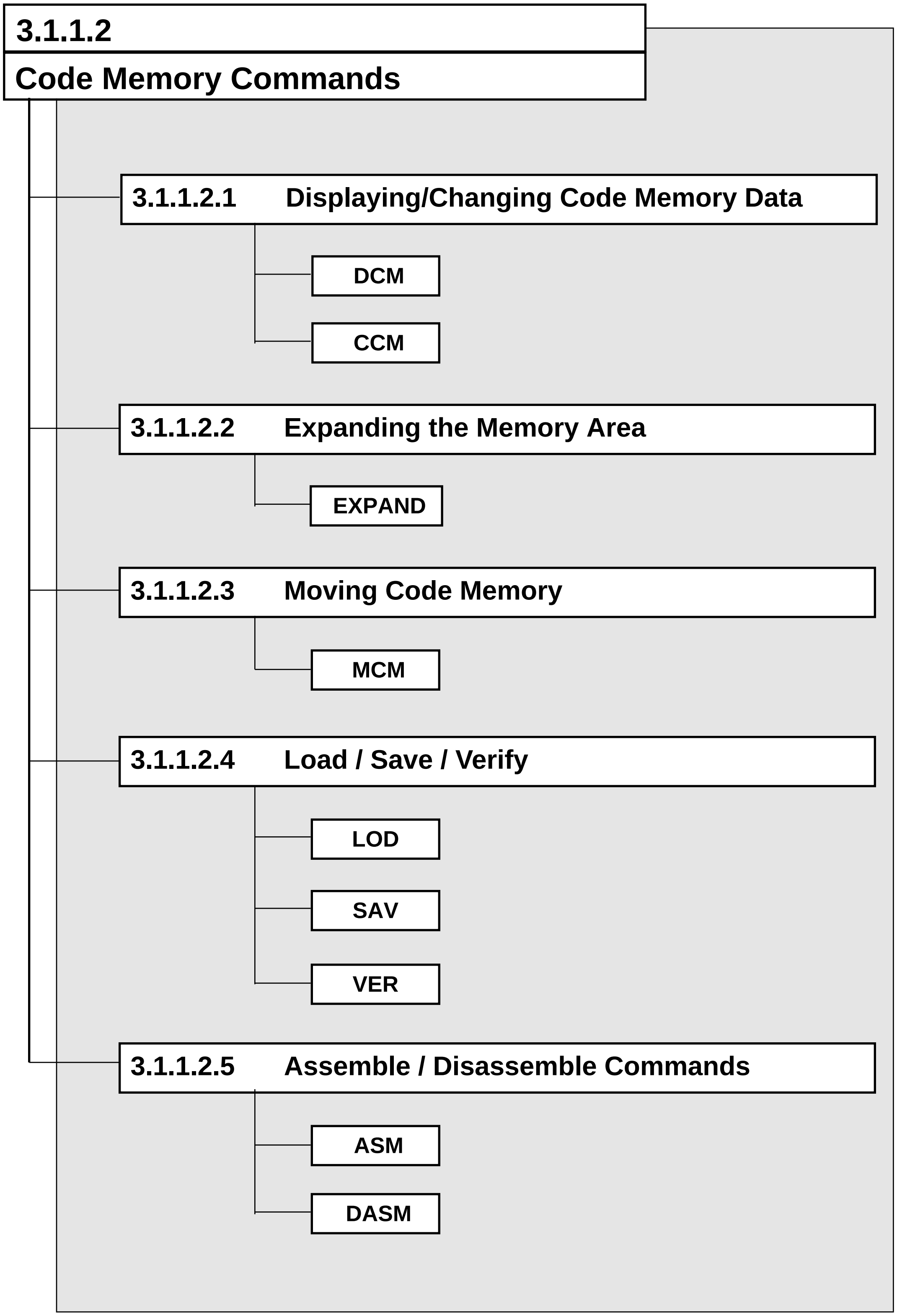
The **CPC** command changes the PC (program counter) to the value specified by *address*.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

! When the code memory is expanded using **EXPAND** command, expressions evaluated as 0–7FFFH will be acceptable for *address* input.

Execution Example

```
64153> DPC
    PC      : 0000
64153> CPC
    PC : 0000 -----> 248H  New
64153> DPC
    PC      : 248
64153> CPC
    PC : 0248 -----> 5 New
64153> DPC
    PC      : 0005
64153>
```



3.1.1.2

Code Memory Commands

3.1.1.2.1 Displaying/Changing Code Memory Data

- DCM
- CCM

3.1.1.2.2 Expanding the Memory Area

- EXPAND

3.1.1.2.3 Moving Code Memory

- MCM

3.1.1.2.4 Load / Save / Verify

- LOD
- SAV
- VER

3.1.1.2.5 Assemble / Disassemble Commands

- ASM
- DASM

3.1.1.2.1 Displaying/Changing Code Memory Data

DCM

Input Format

DCM Δ *parm* [Δ *parm* ..... Δ *parm*] ↵

DCM Δ * ↵

parm : *address*
        : [a*ddress* Δ *address*]

Description

The **DCM** command displays the contents of code memory.

The *address* is an expression that evaluates within code memory’s maximum address range. It indicates an address of code memory to be displayed (☞1).

Display contents are one of the following, depending on input format.

address	Displays the contents on one address.
[*address* Δ *address*]	Displays the range enclosed in [ ].
*	Displays the entire area of code memory.

When multiple parameters are specified, each will be displayed even if their address areas overlap.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

! Note that the emulator handles the test data area in the program area (code memory) of the MSM64153 family microcontrollers as an unusable area.

Execution Example

```
64153> DCM [0 1F]

               0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
              -----------------------------------------------
 LOC = 0000   00 00 D0 B0 00 00 70 D0 00 00 D0 F0 00 00 E0 60
 LOC = 0010   00 00 20 00 00 00 B0 F0 00 00 80 20 00 00 40 C0
64153> DCM 5
       LOC = 0005    00
64153> DCM [204 27A]

               0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
              -----------------------------------------------
 LOC = 0200   00 00 80 00 00 00 80 80 00 00 80 80 00 00 80 80
 LOC = 0210   00 00 80 80 00 00 80 00 00 00 80 00 00 00 80 00
 LOC = 0220   70 80 F0 F0 90 30 F0 F0 60 80 F0 F0 E0 40 F0 F0
 LOC = 0230   30 70 F0 F0 A0 B0 F0 F0 00 90 F0 F0 60 10 F0 F0
 LOC = 0240   00 00 40 00 00 00 C0 B0 00 00 00 00 00 00 40 10
 LOC = 0250   00 00 50 00 00 00 00 20 00 00 20 00 00 00 40 30
 LOC = 0260   50 00 F0 F0 00 00 F0 F0 20 20 F0 F0 00 00 F0 F0
 LOC = 0270   00 00 F0 F0 00 00 F0 F0 00 10 F0 F0 00 00 F0 F0
64153> DCM 123 456 789
 LOC = 0123   F0
 LOC = 0456   00
 LOC = 0789   00
64153>
```

CCM

Input Format

CCM Δ *parm* ↵

CCM Δ * [= *data*] ↵

parm : *address* [= *data*]
: [*address* Δ *address*] = *data*

Description

The **CCM** command changes the contents of code memory.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory (☞1). A '*' indicates the entire code memory area.

If '*' is input and *data* is omitted, then the entire area will be set to '0.'

The *data* is the value of the change data. Its range is 0H to 0FFH. Contents are changed in the order of the input parameters. The area changed is one of the following, depending on input format.

address	Changes the contents on one address.
[*address* Δ *address*]	Changes the range enclosed in [ ].
*	Changes the entire area of code memory.

When multiple parameters are specified, each will be changed even if their address areas overlap.

If *data* is omitted, then the following message will be output and the emulator will wait for data input.

```
LOC = adrs     old-data  --------►  _
```

Here *adrs* expresses the address of code memory whose current contents are to be changed. The *old-data* will be the current contents. At this point the operator enters the change data and inputs a carriage return. The emulator then automatically waits for change data input for the next input.

The **CCM** command will automatically end when *adrs* exceeds the maximum allowable value.

When the emulator is waiting for change data to be input, the following three editing keys are valid.

“∆” Do not change data, and wait for change data to be input at the next address.

“–” Do not change data, return to the address one previous, and wait for change data to be input.

“↵” Move processing to the address specified by the next parameter. If there is no parameter, the emulator will wait for command input.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

! Note that the emulator handles the test data area in the program area (code memory) of the MSM64153 family microcontrollers as an unusable area.

Execution Example

```
64153> CCM 40
 LOC = 0040  00 -----> 11  New
 LOC = 0041  00 -----> 22  New
 LOC = 0042  20 -----> 33  New
 LOC = 0043  00 -----> 44  New
 LOC = 0044  00 -----> 55  New
 LOC = 0045  00 ----->
64153> DCM [39 47]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0030  90 A0 F0 F0 80 D0 F0 F0 B0 F0 F0 F0 C0 D0 F0 F0
 LOC = 0040  11 22 33 44 55 00 00 00 00 00 00 00 00 00 00 20
64153> CCM 100=88 101=77 102=66
64153> DCM [100 10F]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0100  88 77 66 D0 00 00 80 D0 00 00 70 30 00 00 F0 00
64153>
```

CCM

Execution Example

```
64153> CCM 200

 LOC = 0200  00 ----->   Not change
 LOC = 0201  00 ----->   Not change
 LOC = 0202  80 -----> -
 LOC = 0201  00 -----> 12  New
 LOC = 0202  80 -----> -
 LOC = 0201  12 -----> 35  New
 LOC = 0202  80 ----->   Not change
 LOC = 0203  00 ----->
64153> DCM [200 204]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0200  00 35 80 00 00 00 80 80 00 00 80 80 00 00 80 80
64153> CCM *
64153> DCM [0B00 0B0F]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0B00  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
64153> CCM 200
 LOC = 0200  00 ----->   Not change
 LOC = 0201  00 ----->   Not change
 LOC = 0202  00 -----> -
 LOC = 0201  00 -----> 12  New
 LOC = 0202  00 -----> -
 LOC = 0201  12 -----> 35  New
 LOC = 0202  00 ----->   Not change
 LOC = 0203  00 ----->
64153>
```

3.1.1.2.2 Expanding the Memory Area

EXPAND

Input Format

EXPAND [∆ *mnemonic*] ↵

Description

The **EXPAND** command changes the area of the EASE64158 code memory, attribute memory, and instruction executed bit memory, regardless of chip mode.

One of the following is entered for *mnemonic*.

ON	:	Set the memory area 32K bytes (☞1).
OFF	:	Set the memory area to the maximum address of the appropriate MSM64153 family microcontroller (☞2).

The EASE64158 code memory, attribute memory, and instruction executed bit memory areas are set to the maximum address of the appropriate MSM64153 microcontroller during initialization.

If *mnemonic* is omitted, then the current setting will be displayed.

After this command is input, the EASE64158 resets the evaluation chip and clears to “0” all areas of code memory, attribute memory, and instruction executed bit memory.

☞ 1 By changing the memory area to 32K bytes, each memory address will be 0–7FFFH. Table 3-5 shows the relevant commands.

☞ 2 Refer to the appropriate user’s manual for the maximum address of the MSM64153 family microcontroller.

! When the setting is changed by the **EXPAND** command, the evaluation chip is reset.

EXPAND

When EXPAND mode is ON (memory expansion), the prompt is changed to the chip name appended by ‘S.’

Example: 64153> → 64153S>

Table 3-5. Commands That Change Input Parameter Maximum Addresses

Command Group Name	**Maximum Address**
Command Names	
Code Memory Commands	7FFFH
DCM, CCM, MCM, LOD, SAV, VER, ASM, DASM	
Emulation Commands	7FFFH
STP, G	
Break Commands	7FFFH
DBP, CBP	
Trace Commands	7FFFH
STT, DTR, CTR	
Performance / Coverage Commands	7FFFH
SCT, DIE, CIE, CAP	
EPROM Programmer Commands	7FFFH
PPR, TPR, VPR	

Execution Example

```
64153> EXPAND
    OFF MODE

64153> EXPAND ON
    CODE Memory has been expanded 32K byte.
    ***** EVA CHIP RESET *****
```

3.1.1.2.3 Moving Code Memory

MCM

Input Format

MCM ∆ [*address* ∆ *address*] ∆ *address* ↵

Description

The **MCM** command moves the contents of code memory in the specified range to follow the specified the address.

Each *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory.

[*address* ∆ *address*] indicates the area of code memory to be moved. The final *address* parameter indicates the starting address for the move.

Execution Example

```
64153S> CCM [103 10A]=55
64153S> DCM [100 12F]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0100  00 00 00 55 55 55 55 55 55 55 55 00 00 00 00 00
 LOC = 0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 LOC = 0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
64153S> MCM [100 10F] 120
   Code Memory Copy End.
   Last Code Memory Address = 010F
64153S> DCM [100 12F]

              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
             -----------------------------------------------
 LOC = 0100  00 00 00 55 55 55 55 55 55 55 55 00 00 00 00 00
 LOC = 0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 LOC = 0120  00 00 00 55 55 55 55 55 55 55 55 00 00 00 00 00
64153S>
```

! If the data specified by [*address* ∆ *address*] cannot be completely stored at the move destination address, then as much data as can be stored will be moved.

3.1.1.2.4 Load/Save/Verify

LOD

Input Format

LOD Δ *fname* [Δ *option* ..... Δ *option*] ↵

fname : [*Pathname*] *Filename* [*Extension*]
option : /S
: /N
: /B

Description

The **LOD** command loads the code information contents in an object file that has been output by ASM64K (extension of ".HEX") into code memory. Depending on the specified options, symbol information in the object may be loaded into the SID64K internal symbol table.

If the extension is omitted, then a ".HEX" file will be taken as the default.

The input filename can have a path specification. If the path is omitted, then the file in the current directory will be loaded. If the extension is omitted, then the default extension will be appended to the file.

To load a file that has no extension, append a '.' after the filename.

When the **LOD** command terminates, the following message will be output, and the emulator will wait for input (☞1).

Load Completed Address [X X X X – X X X X]

Minimum value of load addresses

Maximum value of load addresses

The file name and format to be loaded will depend on the presence of options, as shown below.

No options specified:

File format	Object file output by ASM64K
Code information	: Loaded
Default extension	: *fname*.HEX
Symbol information	: Not loaded
/S option	: Load symbol information.
/N option	: Do not load code information
/B option	: Set to 0 all breakpoint bits at addresses that are the same as the downloaded code memory addresses. (☞ 2)

When the /S option is input, the emulator will ask whether or not to clear the symbol table, as shown below.

```
Symbol table clear (Y/N)
```

The operator inputs **Y** or **N**.

Y	Load symbols after clearing previously user-defined symbols.
N	Load symbols without clearing previously user-defined symbols·

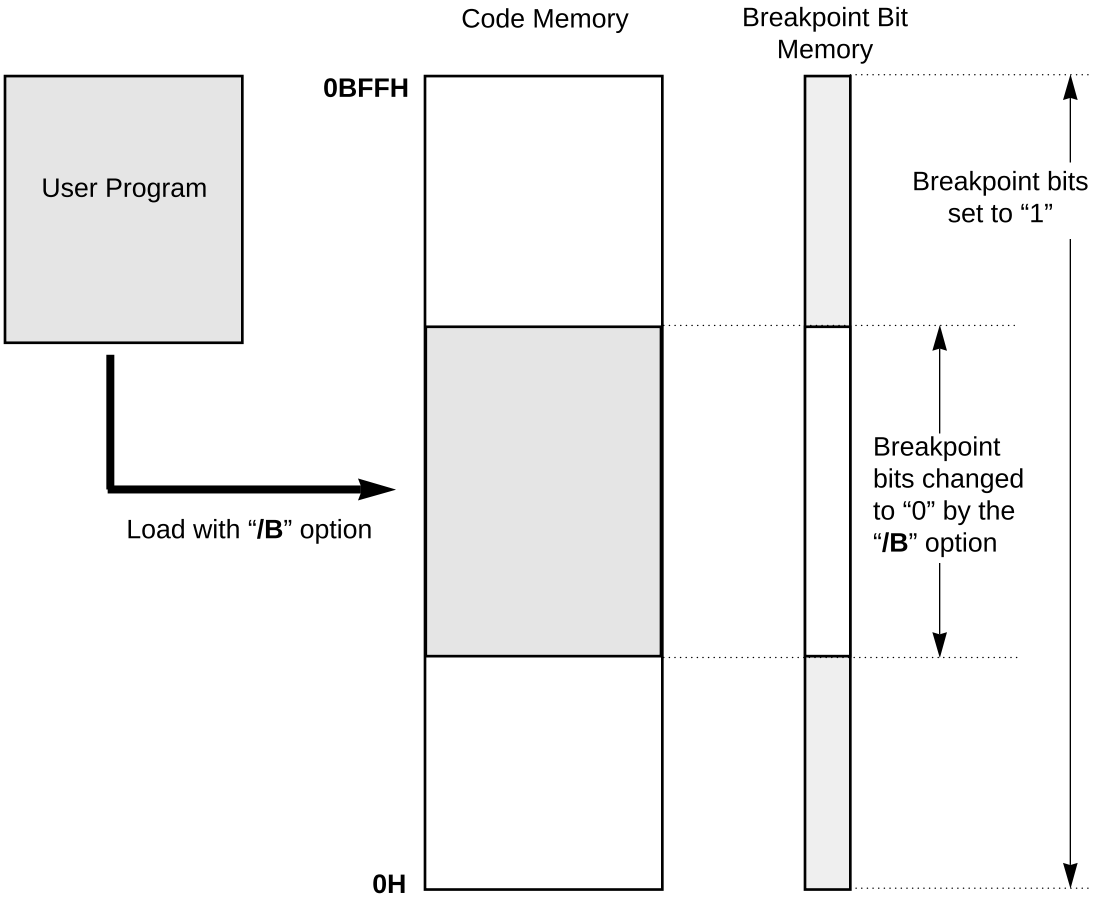
☞ 1

An object file output by the ASM64K cross-assembler includes code information translated from OLMS-64K instruction mnemonics and assembler directives in a source program file, as well as symbols defined in the source program file. The symbol information is generated by appending the “/S” option when assembling.

When SID64K loads an object file and the “/S” option is specified, first the symbol information is registered in the SID64K internal symbol table. Next the code information is loaded into EASE64158 code memory.

If there is an error in the loaded symbol information, then only the symbols in error will not be registered in the table. If there is an error in the loaded code information, then loading will be forcibly terminated. The contents loaded into the EASE64158 before termination cannot be guaranteed, so a new object file must be created and loaded again. For the various error messages output when loading does not complete normally, refer to Appendix 12, “Error Messages.”

☞ 2 Program runaway can be checked by presetting all breakpoint bits to "1" and then appending the "**/B**" option when loading.


Code Memory
Breakpoint Bit Memory
User Program
0BFFH
0H
Load with "/B" option
Breakpoint bits set to "1"
Breakpoint bits changed to "0" by the "/B" option

During emulation execution, a breakpoint bit break will be generated if the code memory areas corresponding to the cross-hatched areas of breakpoint bit memory are executed. Breakpoint bit breaks need to be enabled withe the **SBC** command for this to occur.

Execution Example

```
64153S> LOD INT2
    HEX File Loading...
    Load Completed   address [002C - 0288]

64153S> DASM 100H

LOC=0100  90            LAI      0H
LOC=0101  50 3E         LHLI     IRQ2       ;3EH
LOC=0103  6F            LMA
LOC=0104  50 3F         LHLI     IRQ3       ;3FH
LOC=0106  6F            LMA
LOC=0107  50 7C         LHLI     MIEF       ;7CH
LOC=0109  6F            LMA
LOC=010A  50 0E         LHLI     P6CON      ;EH
LOC=010C  91            LAI      1H
LOC=010D  6F            LMA
LOC=010E  50 0A         LHLI     P21E       ;AH
LOC=0110  93            LAI      P3         ;3H
LOC=0111  6F            LMA
LOC=0112  50 39         LHLI     IE1        ;39H
LOC=0114  90            LAI      0H

64153S> CBP [100 110] =1
64153S> LOD INT2 /B
    HEX File Loading...
    Load Completed   address [002C - 0288]
    Break Point Bit Cleared

64153S> DBP [100 110]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0100    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0110    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153S>
```

SAV

Input Format

SAV ∆ *fname* [[*address* ∆ *address*]] [∆ *option* ..... ∆ *option*] ↵

fname : [*Pathname*] *Filename* [*Extension*]
option : **/S**
: **/N**

Description

The **SAV** command saves the contents of the specified range of code memory to a disk file. The input filename can have a path specification. If the path is omitted, then a file in the current directory will be saved. If the extension is omitted, then the default extension (HEX) will be appended to the file. To load a file that has no extension, append a '.' after the filename.

[*address* ∆ *address*] indicates the area of code memory to be moved. If omitted, then the contents of the same address range of the file most recently loaded with the **LOD** command will be saved (☞1).

The file name and format to be saved will depend on the presence of options, as shown below.

No options specified:

File format	Object file output by ASM64K
Code information	: Saved
Default extension	: *fname*.HEX
Symbol information	: Not saved
/S option	: Save symbol information.
/N option	: Do not load code information

If the input file already exists, then the emulator will output the following message and wait for input.

```
        File exists: delete  (Y/N)
```

The operator inputs **Y** or **N**.

Y : File is modified.
N : File is not modified (save is not performed).

☞ **1** The format of the file saved is the same as object files output by ASM64K.

Saves are performed from low address to high address. To forcibly terminate a save, press the ESC key. By doing so, the file will not be updated. When saving symbol information, the save cannot be forcibly terminated.

If the specified file already exists and is write-protected, then the save will be forcibly terminated. In such cases the file will not be updated.

Execution Example

```
64153S> LOD INT2
    HEX File Loading...
    Load Completed   address [002C - 0288]

64153S> SAV TMP [0 300]
    File exist: delete (Y/N)SAV TMP [0 300] /S
    HEX File Saving...
    Save Completed   address [0000 - 0300]
```

VER

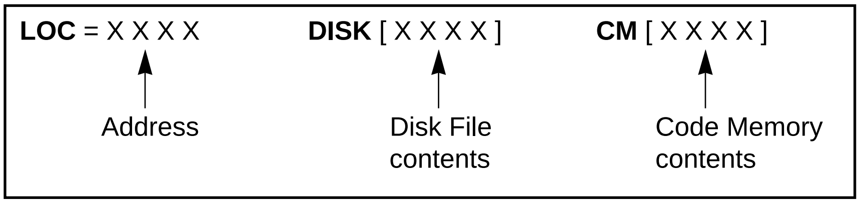
Input Format

VER Δ *fname* [[*address* Δ *address*]] ↵

fname : [*Pathname*] *Filename* [*Extension*]

Description

The **VER** command compares the contents of the specified disk file with the contents of code memory. When a difference is found, the address and the contents of the disk file and of code memory will be displayed as shown below. Symbol information is not compared.



LOC = X X X X **DISK** [X X X X] **CM** [X X X X]

Address

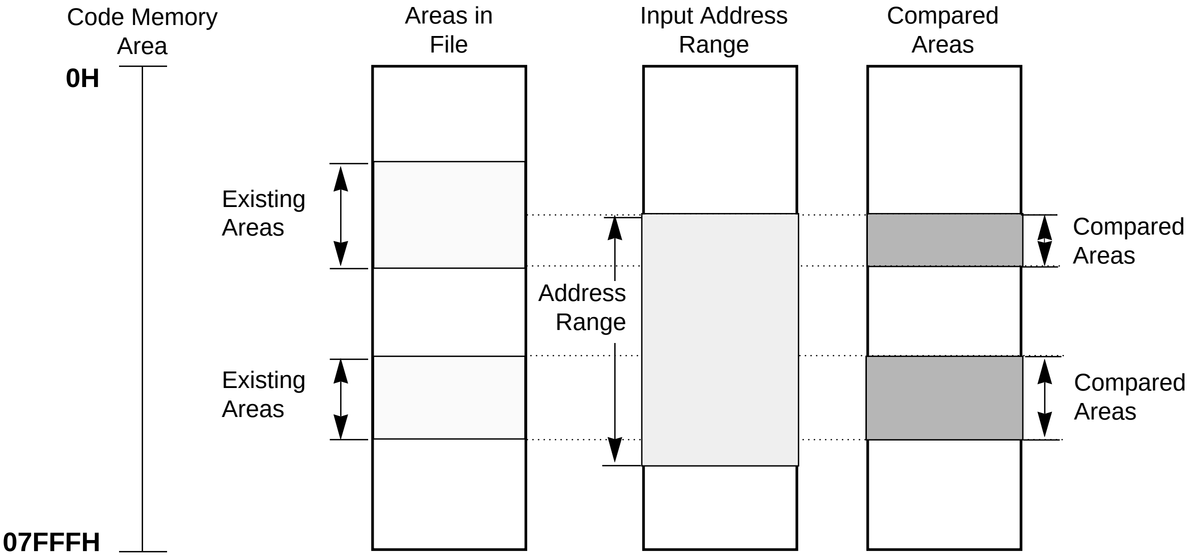
Disk File contents

Code Memory contents

The input filename can have a path specification. If the path is omitted, then a file in the current directory will be verified. If the extension is omitted, then the default extension (HEX) will be appended to the file. To load a file that has no extension, append a '.' after the filename (☞1).

[*address_address*] indicates the area of the disk file and of code memory to be verified. If omitted, then all addresses in the disk file will be compared.

☞ 1 Comparison between the disk file and code memory will be performed on the overlapping areas between the data that exists in the disk file and the [*address_address*] address range specified with with **VER** command. Comparison is performed from low address to high address.



Execution Example

```
64153S> LOD INT1
    HEX File Loading...
    Load Completed   address [002C - 0288]

64153S> SAV TMP
    File exist: delete (Y/N)Y
    HEX File Saving...
    Save Completed   address[002C - 0288]

64153S> CCM 100
       LOC = 0100    90 -----> VER TMP
       ** Error 102: Illegal data input.
       LOC = 0100    90 ----->
64153S>
```

3.1.1.2.5 Assemble/Disassemble Commands

ASM

Input Format

ASM Δ *exp* ↵

exp : *address*

```
line   Segment  Location    Source Statement
1      Code      adrs       SOURCE STATEMENT
```

SOURCE STATEMENT =

; comment line (☞1)
label : [OLMS-64K series mnemonic]
OLMS-64K series mnemonic
assembler directive (refer to description below)

Description

The **ASM** command converts OLMS-64K series instruction statements input from the console (directives, mnemonics, and operands) into object code using a 2-pass assembler based on ASM64K, and then stores that object code in code memory.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory (☞2).

When a carriage return is input, the emulator displays the following message and waits for input from the console.

```
line   Segment      Location     Source Statement
1      Code         adrs
```

At this point the operator can input code that follows the format below.

(1) The maximum number of characters that can be input on one line is 56.

(2) The **ASM** command terminates with an "END." When "END" is input, assembly is performed and the resulting object code is stored in the program memory area.

(3) The maximum number of lines that can be input is 100. When input of the 100th line ends, an “END” will be added automatically, performing assembly and storing the object code in code memory. To input more than 100 lines, use the **ASM** command more than once.

(4) Spaces or tabs can be used as delimiters.

(5) All OLMS-64K series mnemonics and operands can be used.

(6) Symbols can be used in operands and labels (for details, refer to Chapter 5, “Assemble Command”).

(7) Operators can be coded in operands (for details, refer to Chapter 5, “Assemble Command”).

(8) Character constants (such as ‘A’) and string constants (such as “ABC”) can be coded in operands.

(9) A semicolon “;” is used to code a comment.

(10) The default radix for immediate values used in operands is 10 (decimal values). To use a radix other than 10, input as shown in the following table.

Radix	***Syntax***	***Examples***
Binary (radix 2)	Append a 'B' after the number.	01010101B 0101_0101B
Octal (radix 8)	Append an 'O' or 'Q' after the number.	777O, 777Q
Decimal (radix 10)	Append a 'D' or nothing after the number.	10D, 10
Hexadecimal (radix 16)	Append a 'H' after the number.	0ABCDH

Append a '0' before a hexadecimal number which begins with a letter (A—F).

(11) The following assembler directives can be used (for details, refer to Chapter 5, “Assemble Command”).

Directive Type	***Directives Allowed***
Symbol definition	**EQU, SET, DATA, CODE**
Memory segment control	**CSEG, DSEG**
Location counter control	**ORG, DS, NSE**
Data definition	**DB, DW**

(12) The history function can be used. The **ASM** command has a 20-line buffer, separate from the debugger’s history buffer, for use as an assembler-only history function. This buffer’s constants are preserved even after the **ASM** command terminates, so when the **ASM** command is started again, the previously input 20 lines can easily be brought up for editing by using the arrow keys. For details on using the history function, refer to Section 2.3.3, “History Function.”

☞ 1 Comments input with the **ASM** command cannot be displayed with the **DASM** command.

☞ 2 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

! The ASM64K cross-assembler allows **B** (branch instruction) as an assembler directive, but the **ASM** command does not support this.

! Operators can be coded as an operand. Note that the result of division by zero and modulo operation will be handles as '0.'

ASM

Execution Example

```
 64153S> ASM 0
line  Segment  Location  Source Statement
   1  Code     0000      lai 0
   2  Code     0001      lba
   3  Code     0003      lhi 0
   4  Code     0005      lli 0
   5  Code     0006      lxi 0
   6  Code     0008      lyi 0
   7  Code     000A      lai 5
   8  Code     000B      lba
   9  Code     000D      lhi 5
  10  Code     000F      lli 5
  11  Code     0010      lxi 5
  12  Code     0012      lyi 5
  13  Code     0014      lhli 23
  14  Code     0016      lxyi 45
  15  Code     0018      lam
  16  Code     0019      jp 33h
  17  Code     001B      org 33h
  18  Code     0033      nop
  19  Code     0034      ;----- TEST -----
  20  Code     0034      smbd 0eh,0
  21  Code     0036      lhli 6h
  22  Code     0038      lai 0ah
  23  Code     0039      lma
  24  Code     003A      jp 0
  25  Code     003C      nop
  26  Code     003D      nop
  27  Code     003E      end
```

DASM

Input Format

DASM [Δ *exp*] ↵

exp : [*address*] [Δ *option* ..... Δ *option*]
: [[*address* Δ *address*]] [Δ *option* ..... Δ *option*]

option : **/NC**
: **/NL**

Description

The **DASM** command disassembles the contents of code memory and displays the results on the console.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory (☞1).

Note that if disassembly is set to begin on the second or third byte of a 2-byte or 3-byte instruction, then disassembly might not be performed correctly. If disassembly is set to end on the first byte of a 2-byte instruction, or the first or second byte of a 3-byte instruction, then disassembly will be forcibly performed to the end of that instruction.

The output to the console corresponds to the input parameters as explained below.

(1) No options specified

1. Address specification is omitted

Disassembles and displays the 15 lines following the last address disassembled by the previous **DASM** command. After the debugger is initialized, or after the emulator's reset switch is pressed, the 15 lines from address 0 will be displayed when this command is first input. If an address exceeds the maximum address of the appropriate MSM64153 family microcontroller, then disassembly will return to address 0.

2. An *address* is input

Disassembles and displays the 15 lines following the specified *address*. If an address exceeds the maximum address of the appropriate MSM64153 family microcontroller, then disassembly will return to address 0.

3. An [*address_address*] is input

Disassembles and displays from the first *address* to the second *address*.

(2) Options are specified

Specification of options can add the following functions to the input methods of (1) above.

1. /NC option

By specifying this option, object code will not be displayed.

2. /NL option

By specifying this option, addresses (LOC=xxxx) will not be displayed.

Example use of options:

Use the **LIST** command to send console output to a file, executed the **DASM** command with the **/NL** and **/NC** options appended, and then close the file with the **NLST** command. By editing this file with an editor, one can easily create a source file.

!

- Labels and statements cannot be displayed on the same line.

Example:

```
64153> DASM [LOOP LOOP+3] /NC /NL
       LOOP:      ——— Label displayed on one line
            LAI   0
            LMAD  0C
            SMBD  7C,0
```

- Comments coded with the **ASM** command cannot be output by the **DASM** command.

- Only the first 10 characters of symbols longer than 10 characters will be displayed.

- If multiple symbols with identical values exist, then expected symbols might not be displayed, but there is no problem with the contents of code memory.

- Symbol information is displayed as comments.

```
64153> DASM [200 . 205] /NC /NL
       LAMD P2    ; 02H      Symbol information
       LMA+                  displayed as comments
       LAMD P6D   ; 06H
       LMA+
```

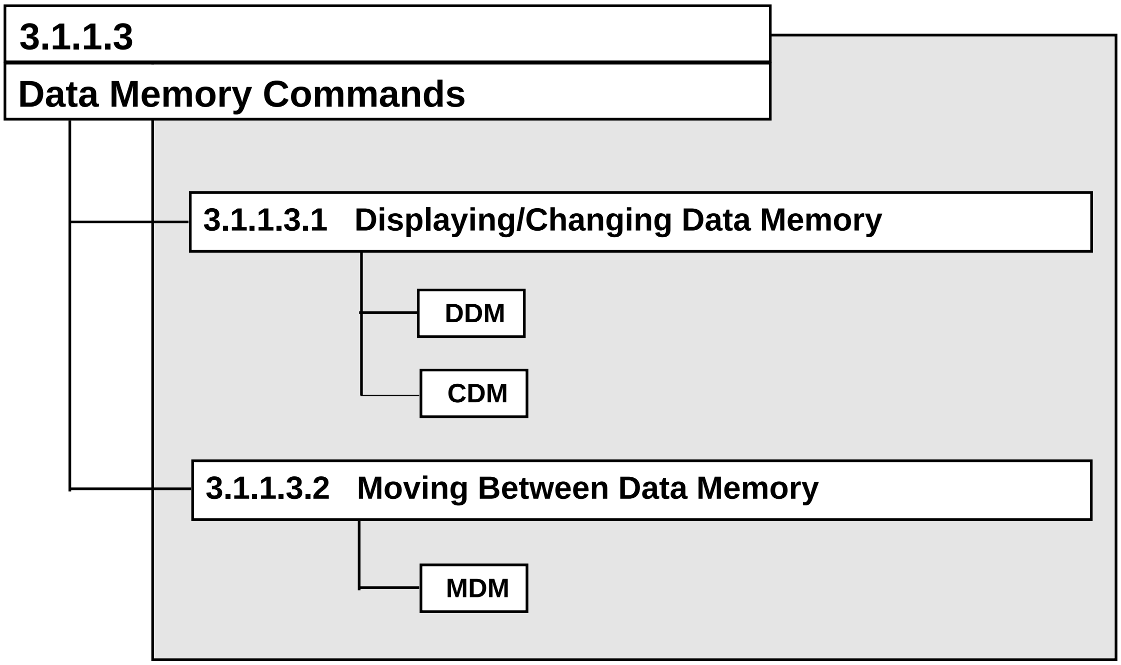
☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

Execution Example

```
 64153S> LOD INT2
     HEX File Loading...
     Load Completed   address [002C - 0288]

 64153S> DASM 100

LOC=0100  90             LAI      0H
LOC=0101  50 3E          LHLI     IRQ2      ;3EH
LOC=0103  6F             LMA
LOC=0104  50 3F          LHLI     IRQ3      ;3FH
LOC=0106  6F             LMA
LOC=0107  50 7C          LHLI     MIEF      ;7CH
LOC=0109  6F             LMA
LOC=010A  50 0E          LHLI     P6CON     ;EH
LOC=010C  91             LAI      1H
LOC=010D  6F             LMA
LOC=010E  50 0A          LHLI     P21E      ;AH
LOC=0110  93             LAI      P3        ;3H
LOC=0111  6F             LMA
LOC=0112  50 39          LHLI     IE1       ;39H
LOC=0114  90             LAI      0H

 64153S> DASM [103 111]

LOC=0103  6F             LMA
LOC=0104  50 3F          LHLI     IRQ3      ;3FH
LOC=0106  6F             LMA
LOC=0107  50 7C          LHLI     MIEF      ;7CH
LOC=0109  6F             LMA
LOC=010A  50 0E          LHLI     P6CON     ;EH
LOC=010C  91             LAI      1H
LOC=010D  6F             LMA
LOC=010E  50 0A          LHLI     P21E      ;AH
LOC=0110  93             LAI      P3        ;3H
LOC=0111  6F             LMA
```

Execution Example

```
 64153S> DASM [103 111] /NC

LOC=0103          LMA
LOC=0104          LHLI     IRQ3       ;3FH
LOC=0106          LMA
LOC=0107          LHLI     MIEF       ;7CH
LOC=0109          LMA
LOC=010A          LHLI     P6CON      ;EH
LOC=010C          LAI      1H
LOC=010D          LMA
LOC=010E          LHLI     P21E       ;AH
LOC=0110          LAI      P3         ;3H
LOC=0111          LMA

 64153S> DASM [103 111] /NL

6F             LMA
50 3F          LHLI     IRQ3       ;3FH
6F             LMA
50 7C          LHLI     MIEF       ;7CH
6F             LMA
50 0E          LHLI     P6CON      ;EH
91             LAI      1H
6F             LMA
50 0A          LHLI     P21E       ;AH
93             LAI      P3         ;3H
6F             LMA
```


3.1.1.3
Data Memory Commands
3.1.1.3.1   Displaying/Changing Data Memory
DDM
CDM
3.1.1.3.2   Moving Between Data Memory
MDM

3.1.1.3.1 Displaying/Changing Data Memory

DDM, CDM

Input Format

DDM ∆ *parm1* [∆ *parm1* ..... ∆ *parm1*] ↵
DDM ∆ * ↵
parm1 : *address*
[*address* ∆ *address*]

CDM ∆ *parm2* [∆ *parm2* ..... ∆ *parm2*] ↵
CDM ∆ * ↵
parm2 : *address* [= *data*]
[address ∆ address] = *data*

Description

The **DDM** command displays the contents of data memory as specified by *parm1*.

The *address* is an expression that evaluates within data memory's maximum address range. It indicates an address of data memory (☞1). An [*address* ∆ *address*] indicates a range between two addresses. A '*' indicates the entire data memory area. If multiple parameters are input, then display/change will be performed for each parameter, even if their address areas overlap.

The **CDM** command changes the contents of data memory as specified by *parm2*.

The *address* is an expression that evaluates within data memory's maximum address range. It indicates an address of data memory (☞1). An [*address* ∆ *address*] indicates a range between two addresses. A '*' indicates the entire data memory area, excluding the SFR area. If '*' is input and *data* is omitted, then the entire area will be set to '0.'

The *data* is an expression that must evaluate in the range 0H to 0FFH. If *data* is omitted, then the emulator outputs the following message and waits for data to be input.

```
LOC = address    old-data  ----------➤ _
```

Here *address* expresses the address in data memory that is to have its current contents changed. The *old-data* will be the current contents. At this point the operator enters new data (*data*) and inputs a carriage return.

```
LOC = address   old-data ---► data ◄-------- NEW
LOC = address   old-data ---► _  ◄────────── input data for next
                                             parameter
```

When the carriage return is input, processing moves to the next parameter. If there is no next parameter, then the **CDM** command terminates.

When the emulator is waiting for input data for a change, the following three key inputs are valid in addition to *data*.

Δ ↵ (space followed by carriage return) Move processing to the next parameter without changing the current data. If there is no next parameter, then the **C** command terminates.

- ↵ (minus followed by a carriage return) Move processing to the previous parameter without changing the current data.

↵ (input carriage return only) The **C** command terminates.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

! When the SFR area is displayed with the **DDM** command, bits in each SFR that do not exist will be displayed as “1” data. For example, after **RST E** is executed, the Halt Mode Register at address 7DH will be displayed with the value 0E.

! To change data memory at 0H-7DH (the SFR area) with the **CDM** command, input one address at a time. The SFR area cannot be changed if an address range is input. Addresses 7DH cannot be changed with **CDM** (HALT mode will be forcibly released when emulation is not executed), but must be changed as SP with the **C** command.

! The maximum number of individual addresses that can be input interactively is 200.

DDM, CDM

Execution Example

```
64153>
64153> CDM [760 7FF]=0
64153> DDM [760 7FF]

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0760    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0770    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0780    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0790    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07A0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07B0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07C0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07D0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 07E0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07F0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> CDM [765 789]=5
64153> DDM [760 7FF]

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0760    5 5 5 5 5 5 5 5 5 5 5 0 0 0 0 0
       LOC = 0770    5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
       LOC = 0780    0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5
       LOC = 0790    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07A0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07B0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07C0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07D0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 07E0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 07F0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Execution Example

```
64153> DDM 777
       LOC = 0777    5
64153> CDM 780
       LOC = 0780    5 -----> 0  New
       LOC = 0781    5 -----> 1  New
       LOC = 0782    5 -----> 2  New
       LOC = 0783    5 -----> 3  New
       LOC = 0784    5 -----> 4  New
       LOC = 0785    5 -----> 5  New
       LOC = 0786    5 ----->    Not change
       LOC = 0787    5 -----> -
       LOC = 0786    5 -----> -
       LOC = 0785    5 -----> 7  New
       LOC = 0786    5 -----> 6  New
       LOC = 0787    5 -----> 5  New
       LOC = 0788    5 -----> -
       LOC = 0787    5 ----->    Not change
       LOC = 0788    5 ----->
64153> DDM [780 788]

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0780    0 0 0 0 0 0 5 5 5 6 7 4 3 2 1 0
64153> CDM 788=9 790=2 7A0=4
64153> CDM 767 782 7AD
       LOC = 0767    5 -----> 0  New
       LOC = 0768    5 -----> 2  New
       LOC = 0769    5 ----->    Not change
       LOC = 076A    5 -----> 9  New
       LOC = 076B    5 -----> 3  New
       LOC = 076C    5 ----->    Not change
       LOC = 076D    5 ----->    Not change
       LOC = 076E    5 -----> 4  New
       LOC = 076F    5 -----> 3  New
       LOC = 0770    5 -----> -
       LOC = 076F    3 -----> 2  New
       LOC = 0770    5 ----->
       LOC = 0782    2 ----->
       LOC = 07AD    0 ----->
```

3.1.1.3.2 Moving Between Data Memory

MDM

Input Format

MDM Δ *parm* ↵

parm : [*address* Δ *address*] Δ *address*

Description

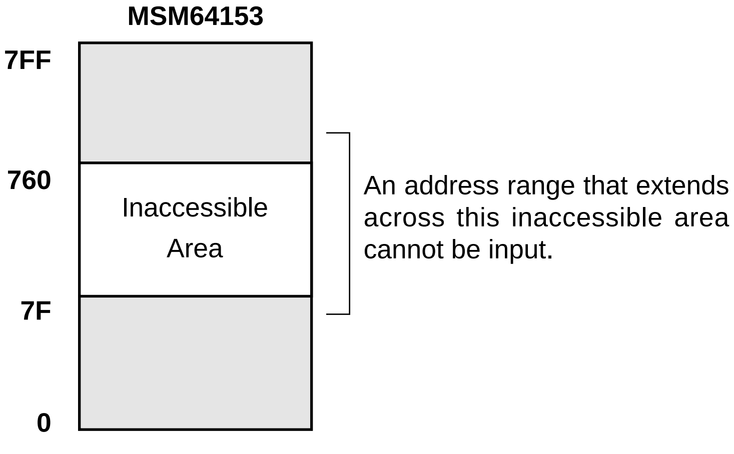
The **MDM** command moves the contents of data memory specified by [*address_address*] (excluding the SFR area) to the area following *address* (excluding the SFR area).

Each *address* is an expression that evaluates within data memory’s maximum address range. It indicates an address of data memory. However, it cannot be in the SFR area of data memory (☞1).

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

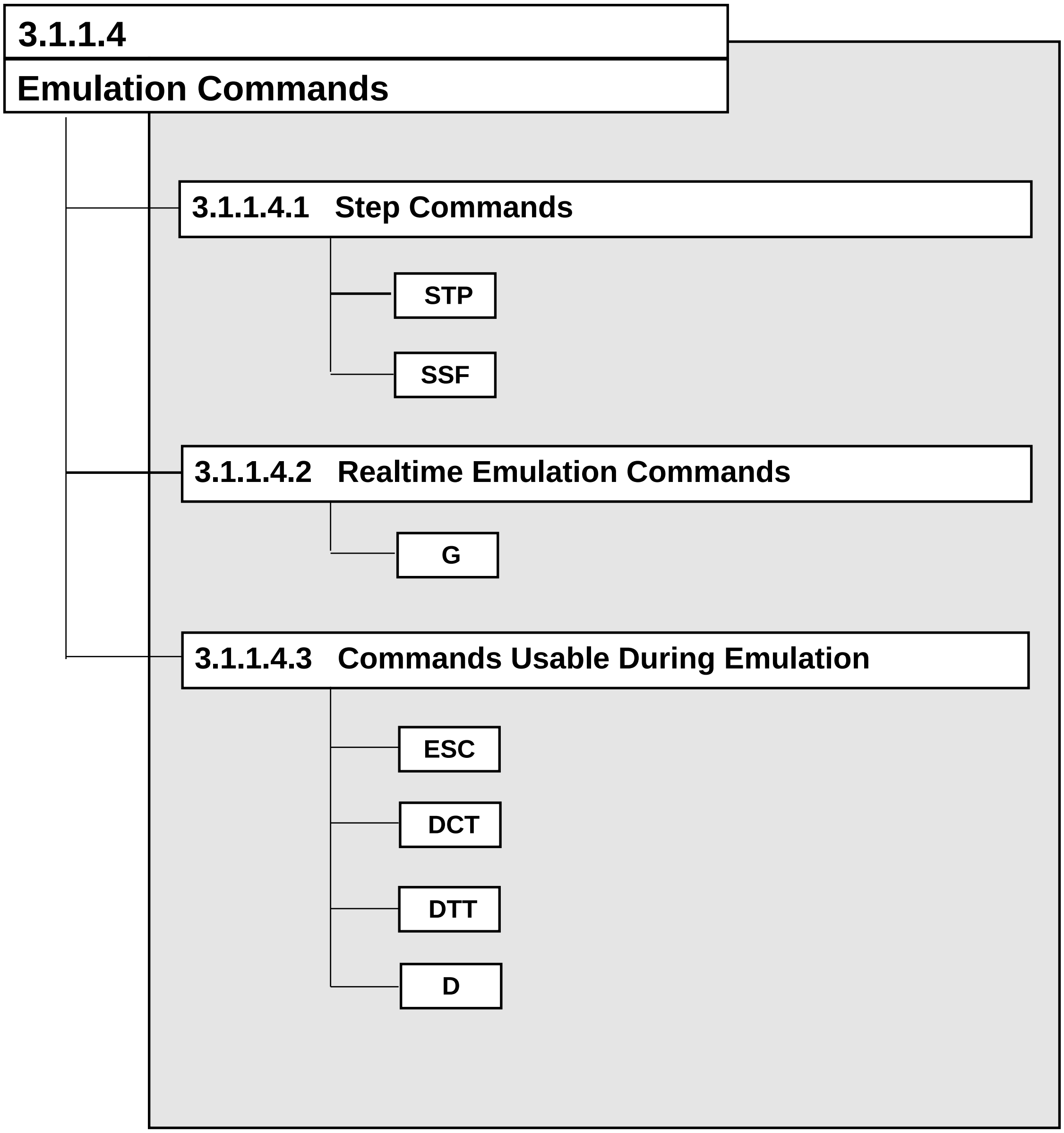
! Data cannot be transferred to or from the SFR area.

!


MSM64153
7FF
760
Inaccessible
Area
7F
0
An address range that extends across this inaccessible area cannot be input.

Execution Example

```
64153> CDM [760 7FF]=0FH
64153> CDM [780 788]=05H
64153> DDM [771 798]

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0770    F F F F F F F F F F F F F F F F
       LOC = 0780    F F F F F F F 5 5 5 5 5 5 5 5 5
       LOC = 0790    F F F F F F F F F F F F F F F F
64153> MDM [780 788] 794
   Data Memory Copy End.
   Last Data Memory Address = 0788
64153> DDM [771 798]

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0770    F F F F F F F F F F F F F F F F
       LOC = 0780    F F F F F F F 5 5 5 5 5 5 5 5 5
       LOC = 0790    F F F 5 5 5 5 5 5 5 5 5 F F F F
```



3.1.1.4

Emulation Commands

3.1.1.4.1   Step Commands

STP

SSF

3.1.1.4.2   Realtime Emulation Commands

G

3.1.1.4.3   Commands Usable During Emulation

ESC

DCT

DTT

D

3.1.1.4.1 Step Commands

STP

Input Format

STP [∆ *address*] [, *count*] ↵

The **STP** command executes a user program in code memory one instruction at a time.

The *address* is an expression that evaluates within data memory's maximum address range. It indicates the first address at which step execution is to begin (☞1). If *address* is omitted, then step execution will begin from the address indicated by the current program counter (PC).

The *count* is a decimal value from 1 to 65535. It indicates the number of steps to be executed. If *count* is omitted, then step execution will be performed for just one instruction and the command will terminate.

The **STP** command stops user program execution after each instruction. At each stop, it displays the address and mnemonic of the executed instruction, and then displays the states of the registers and ports after execution. The display format is specified with the **SSF** command.

The **STP** command does not display instructions that are skipped with a skip instruction. When the conditions for skipping an instruction are fulfilled (accumulate instruction, increment instruction, etc.) then the next instruction is skipped and one step ends.

! The **STP** command preserves the value of the time-base counters between each step. Although, operation of timers and counters that are synchronized to microcontroller internal clocks is guaranteed, operation synchronized to external clocks is not.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

Execution Example

```
64153> ASM 100
line  Segment  Location  Source Statement
   1  Code     0100      start:
   2  Code     0100      lbs0i 0
   3  Code     0102      smbd 0eh, 0
   4  Code     0104      lai 0
   5  Code     0105      sub:
   6  Code     0105      lba
   7  Code     0107      nop
   8  Code     0108      loop:
   9  Code     0108      ina
  10  Code     0109      nop
  11  Code     010A      jp loop
  12  Code     010C      end

64153> STP START, 3
 LOC=0100   LBS0I   0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



 LOC=0102   SMBD    EH,  0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



 LOC=0104   LAI     0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



64153> STP

 LOC=0105   LBA
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0
```

SSF

Input Format

SSF [∆ *parm 1*..... ∆ *parm 1*] ↵
SSF ∆ *parm 2* ↵

parm1 : [˜] *mnemonic1*
parm 2 : [˜] *mnemonic2*

Description

The **SSF** command determines display format during **STP** command execution through each *mnemonic*. The following can be entered for *mnemonic*.

mnemonic1:

INS	Instruction
INSC	Instruction code
SFR-mnemonic	Refer to Table 3-2
A	A register
B	B register
H	H register
L	L register
X	X register
Y	Y register
SP	Stack pointer
C	Carry flag
INT	Flag showing interrupt operation (☞2)
SKIP	Flag showing skip execution (☞2)

mnemonic2:

RAM ∆ *parm* [∆ *parm* ..... ∆ *parm*]
parm : *address* , [*address* ∆ *address*]
DEF Initial state, showing **LOC, INS, A, B, H, L**

“RAM” displays the contents of data memory specified by *parm*. Up to 10 *parm*s can be input at a time. Each *address* is an expression that evaluates within data memory’s maximum address range. It indicates an address in data memory (☞1).

If a ‘~’ (tilde) is input before *mnemonic*, then its setting can be canceled. To cancel the addresses set with RAM, input the same address that has been set with RAM, or the address range including addresses to be canceled for *mnemonic*. If *mnemonic* is omitted, then the currently set format will be displayed.

! ~DEF cannot be input.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding the range of input addresses.

☞ 2 Example:

```
  :
LAI      1   \
LAI      2    |
LAI      3    |
LAI      4    |
LMAD     7C  /
  :
```

In this program, if step execution of the instructions from “LAI 1” until “LMAD 7C” is performed, then the skipped “LAI 2,” “LAI 3,” and “LAI 4” will not be displayed during step execution, and “LMAD 7C” will be displayed after execution of “LAI 1” just like it has been executed after "LAI 1."

SSF

Execution Example

```
64153> SSF DEF
64153> STP START,3

 LOC=0100   LBS0I   0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



 LOC=0102   SMBD    EH,  0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



 LOC=0104   LAI     0H
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0



64153> SSF X Y C
64153> STP

 LOC=0105   LBA
             ------------------< Registers >------------------
               A:0  B:0  H:0  L:0  X:0  Y:0  C:0



64153> SSF ~H ~L
64153> STP

 LOC=0107   NOP
             ------------------< Registers >------------------
               A:0  B:0  X:0  Y:0  C:0
```

3.1.1.4.2 Realtime Emulation Commands

G

Input Format

G [Δ *Emu_start_addr*] [, *Break_parm*] ↵

Emu_start_addr : Start address for realtime emulation

Break_parm : *address* [Δ *address* ..... Δ *address*]
: [*address* Δ *address*]
: *address* [*count*]
: / *address* / *address* [/ *address*]
: *mnem* [*&mask*] = *data*
: *mnem* [*&mask*] = *data* [*count*]
: *mnem* [*&mask*] = *data* [Δ /*address* [Δ *address* ......]]
: *mnem* [*&mask*] = *data* [*count*] [Δ /*address* [Δ *address* ....]]
: *mnem* [*&mask*] = *data* [/ [*address* Δ *address*]]
: *mnem* [*&mask*] = *data* [*count*] [/ [*address* Δ *address*]]

Mnem : PRB (Probe)
: RAM [Δ *ram_addr*]

Description

The **G** command performs realtime emulation (continuous execution) of a user program within code memory.

The *Emu_start_addr* is an expression that evaluates within code memory's maximum address range. It indicates the address at which the user program is to begin realtime emulation (☞1). If *Emu_start_addr* is omitted, then realtime emulation will start as the address indicated by the current program counter (PC).

There are 10 possible break conditions. The condition that will break realtime emulation is entered in *Break_parm*. If *Break_parm* is omitted, then realtime emulation will continue to execute until a break on a break condition (☞2) or a break from an **ESC** command.

! The *mnem* within *Break_parm* is entered with a data match break on the probe pins or a RAM address. These are input as "PRB" and "RAM ram_addr" respectively ("ram_addr" can be omitted).

! To have a break condition based on the result of masking *mnem*, enter *mnem* & *mask*. The value entered for *mask* should be 0–0FH when *mnem* is "RAM," and should be 0–0FFH when *mnem* is "PRB." Set *mask* to 0 to invalidate the corresponding bit, 1 to validate it (if omitted, the value will be FH or FFH).

Example: If "RAM 100H & 1000B = 0FH" is specified, then a break will occur when RAM address 100H is 3H, 7H, BH, or FH.

! Each *address* is an expression that evaluates within code memory's maximum address range. However, be sure to input the address of the first byte of an instruction in the code memory area. No break will occur if any other addresses are entered.

! If a start address for realtime emulation is input, then the trace pointer will be cleared. If a start address is not input, then the trace pointer will increment from its previous value.

! When the **URST** command is set ON, reset input from the user cable’s USER•RESET pin is permitted. However, this reset input is only allowed during realtime emulation from the **G** command.

! If a break condition is satisfied during a skip, then the break will be held off until after the skip ends. In this case, the break condition will be "No Breakstatus."

Example:

```
  :
LAI     1
LAI     2
LAI     3
LAI     4
LMAD    7C
  :
```

In this program, if a breakpoint bit break is set at the location of the “LAI 3” instruction, and continuous execution is started from the “LAI 1” instruction, then the break will not occur at the “LAI 3” instruction, which comes during the skip, but instead will occur just before the “LMAD 7C” instruction.

! If a break condition is satisfied during an interrupt transferring cycle, then the break will occur under "No Breakstatus."

! If a break condition is satisfied when an interrupt is generated, then the break will be held off until after the interrupt operation ends. In this case, the break status will be "No Breakstatus."

! The values of time-base counters will be preserved after a break occurs until execution is started again. Although, operation of timers and counters that are synchronized to microcontroller internal clocks is guaranteed, operation synchronized to external clocks is not.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual regarding the range of input addresses.

☞ 2 Refer to Section 3-1-1-5, “Break Commands,” regarding break conditions.

Description of ***Break_parm***

(1) Address break (specified as individual addresses)

> *address* [∆ *address* ..... ∆ *address*]

A break will occur when an instruction at any of the addresses specified by *address* is executed. A maximum of 20 addresses can be entered at one time.

(2) Address break (specified as a range)

> [*address* ∆ *address*]

A break will occur when an instruction at any address within the specified range is executed.

(3) Address pass count break

> *address* [*count*]

A break will occur when the instruction at the address specified by *address* is executed *count* times. The *count* is a decimal value 1–65535.

(4) Address pass break

> / *address* / *address* [/ *address*]

When execution proceeds in the sequence of each slash-delimited (/) *address* from the left, then a break will occur after the instruction at the last specified address is executed.

(5) Data match break

> *mnem* [*&mask*] = *data*

A break will occur when the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked.

(6) Data match break with count

> *mnem* [*&mask*] = *data* [*count*]

A break will occur when the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked, for the number of times specified by *count*. The *count* is a decimal value 1–65535.

(7) Data match break at address

mnem [*&mask*] = *data* [Δ *address* [Δ / *address* •••]]

A break will occur when both the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked, and the PC is at a specified *address*. A maximum of 20 addresses can be entered at one time.

(8) Data match break at address with count.

mnem [*&mask*] = *data* [*count*] [Δ / *address* [Δ *address* •••]]

A break will occur when both the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked, and the PC is at a specified *address*, for the number of times specified by *count*. A maximum of 20 addresses can be entered at one time. The *count* is a decimal value 1–65535.

When the data match break at address with count is executed under the following condition, the break will occur at a different number of times that is specified by *count*.

```
64153 > ASM0
 1 Code  0000  LBS01 7
 2 Code  0002  LHLI 60
 3 Code  0004  LMA
 4 Code  0005  LMA
 5 Code  0006  LMA
 6 Code  0007  END

64153 >
```

For example as shown at the left, when writing instruction into data memory (LMA) is repeated, and if *address* is set to 5 or 6, then the break will occur at a different number of times specified by *count*.

(9) Data match break in address range

mnem [*&mask*] = *data* [/ [*address* , *address*]]

A break will occur when both the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked, and the PC is in the specified address range.

(10) Data match break in address range with count

mnem [*&mask*] = *data* [*count*] [/ [*address* , *address*]]

A break will occur when both the value of *data* matches the contents of *mnem*, or the contents of *mnem* masked, and the PC is in the specified address range, for the number of times specified by *count*. The *count* is a decimal value 1–65535.

If the trace trigger has been set (**STT** command) to trace after data match (AD) or trace before data match (BD), and the **G** command break condition is set to a PRB or RAM data match break, then the the trace trigger condition will be changed to free-run trace (ALL). In other words, the trace trigger condition will not be effective, while the break condition will be effective. Afterwards the trace trigger condition will remain as free-run trace (ALL) until it is set again with the **STT** command.

The timing of data match breaks using RAM addresses is such that a break will occur after execution of the next instruction following the instruction that satisfied the break condition.

When realtime emulation is started, the message "***** Emulation Go *****" will be displayed, and the prompt will change as follows.

```
      Go>>
```

When a break on some condition occurs during continuous execution, the following type of message will be displayed.

```
      ***** Break Status *****

      Break PC = [Break-address], Next PC = [Next-address], TP = [Trace-Pointer]
```

The *Break Status* is one of the break conditions.

SEE **DBS** command

The *Break-address* is the address of the user program where the realtime emulation break occurred. The *Next-address* is the first address of the instruction that is to be executed after the *Break-address*. The *Trace-Pointer* is the trace pointer value at the point the break occurred.

The *Break-address* and *Next-address* are an hexadecimal data that evaluate within code memory's maximum address range. The *Trace-Pointer* is decimal data.

Execution Example

```
64153> LOD INT1
    HEX File Loading...
    Load Completed   address [002C - 0288]

64153> G 100,114
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0114], Next PC =[0115], TP =[0015]
64153> G , 120

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0120], Next PC =[0121], TP =[0024]
64153> G 100,126[3]
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Pass Count Break *****
    Break PC =[0126], Next PC =[003B], TP =[0082]
64153> G 100,/106/114/126
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Pass Break *****
    Break PC =[0126], Next PC =[00FF], TP =[0029]
64153> G 100,[114 126]
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0114], Next PC =[0115], TP =[0015]
64153> G 100,109 107 10C 111
    Reset Trace Pointer

    ***** Emulation Go *****
Go>>

    ***** Address Break *****
    Break PC =[0107], Next PC =[0109], TP =[0006]
```

3.1.1.4.3 Commands Usable During Emulation

ESC

Input Format

ESC ↵

Description

The **ESC** command forcibly breaks realtime emulation. During realtime emulation the following prompt is displayed.

```
Go>>
```

If the **ESC** command is input while this prompt is displayed, then the following message will be output and realtime emulation will break.

```
Go>>

***** ESC Break *****

Break PC = [Break-address], Next PC = [Next-address], TP = [Trace-
Pointer]
```

The *Break-address* is the address of the user program where the realtime emulation break occurred. The *Next-address* is the first address of the instruction that is to be executed after the *Break-address*. The *Trace-Pointer* is the trace pointer value at the point the break occurred.

The *Break-address* and *Next-address* are an hexadecimal data that evaluate within code memory's maximum address range. The *Trace-Pointer* is decimal data.

Execution Example

```
64153> RST E

    ***** EVA CHIP RESET *****

64153> G 100
    Reset Trace Pointer

    ***** Emulation Go *****
Go >> ESC

Go >>

    ***** ESC Break *****
    Break PC =[0245], Next PC =[0247], TP =[5005]
```

DCT

Input Format

DCT ↵

Description

The EASE64158 can display the contents of its cycle counter trigger (start/stop addresses) during realtime emulation. For details on the **DCT** command, refer to “Execution Time Rules” of Section 3.1.1.8.1.

Execution Example

Refer to Section 3.1.1.8.1, “**DCT** command.”

DTT

Input Format

DTT ↵

Description

The EASE64158 can display the contents of its trace trigger setting during realtime emulation. For details on the **DTT** command, refer to "Setting/Displaying the Trace Trigger" of Section 3.1.1.6.3.

Execution Example

Refer to Section 3.1.1.6.3, "**DTT** command."

D

Input Format

D [∆ *mnemonic* ..... ∆ *mnemonic*]

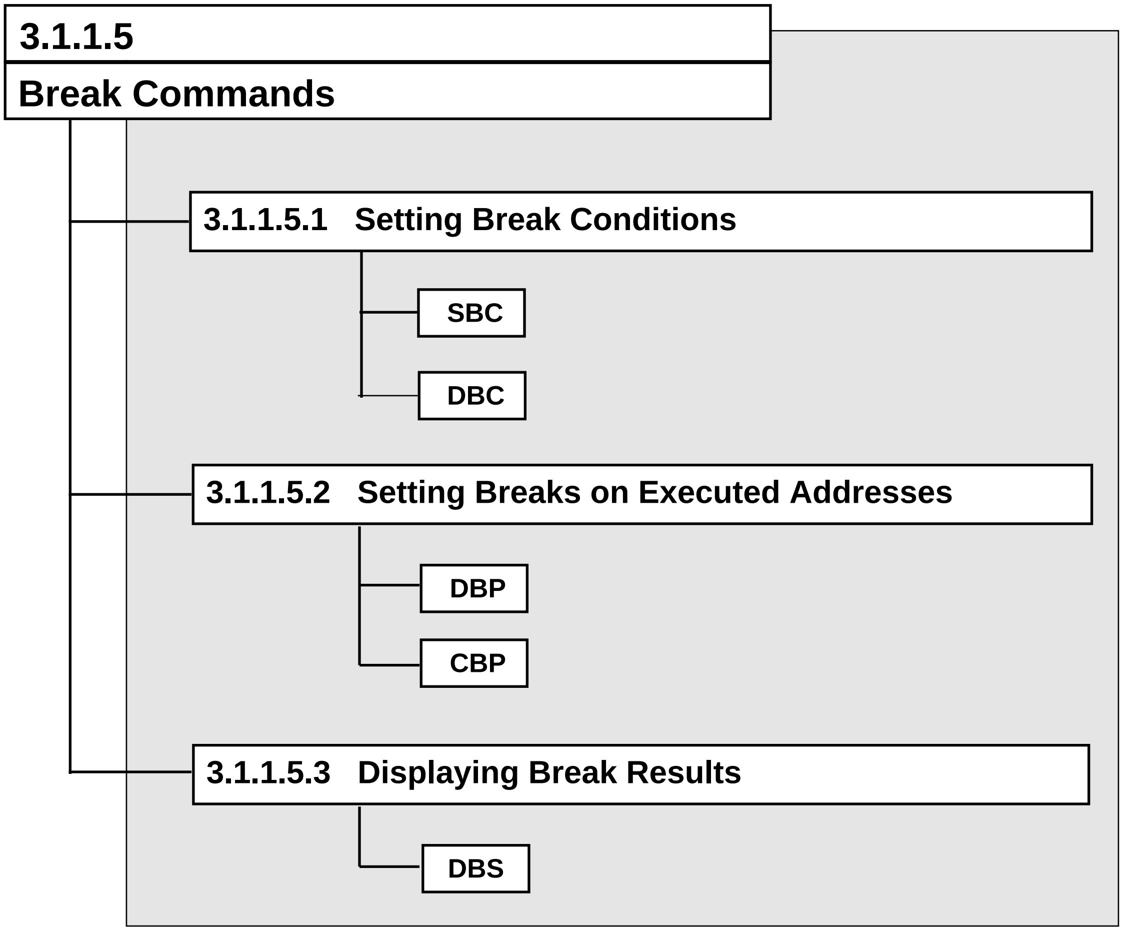
mnemonic : **A, B, X,Y, H, L, PC, BSR0, BSR1, BCF, BEF, C**

Description

The EASE64158 can display the contents of the registers specified by *mnemonic* during realtime emulation. However, the XY or HL register must be set with the **CTO** command.

Execution Example

Refer to Section 3.1.1.1.1, "**D** command."



3.1.1.5

Break Commands

3.1.1.5.1   Setting Break Conditions

SBC

DBC

3.1.1.5.2   Setting Breaks on Executed Addresses

DBP

CBP

3.1.1.5.3   Displaying Break Results

DBS

3.1.1.5.1 Setting Break Conditions

SBC, DBC

Input Format

SBC [∆ *parm* ..... ∆ *parm*] ↵

DBC ↵

parm : [~] *mnemonic*

Description

The **SBC** command sets the break conditions specified by *mnemonic*. These are separate from the break conditions specified by **G** command parameters.

If a *mnemonic* is prefixed by a '~' (tilde), then its setting will be cancelled.

One of the following can be entered for *mnemonic*.

BP	Breakpoint bit break
CC	Cycle counter overflow break
TF	Trace full break
AP	Address pass count overflow break
PD	Power down break
XP	External probe break

If *parm* is omitted, then the emulator will enter interactive input mode for each break condition.

```
mnemonic Condition SET? (Y/N) _
```

Here *mnemonic* indicates one of the above break conditions. The operator then sets or cancels each break condition by inputting at the underscore.

```
mnemonic Condition SET? (Y/N)  Y
mnemonic Condition SET? (Y/N)    ◄—— Input for next parameter
```

The following four key inputs are valid while the emulator is waiting for input.

Y ↵ .... Sets the break condition indicated by *mnemonic*.

N ↵ .... Cancels the break condition indicated by *mnemonic*.

Δ ↵ (space followed by carriage return) .... Without changing data, moves to process next *mnemonic*. If there is no next *mnemonic*, then the **SBC** command terminates.

↵ (carriage return only) .... Terminates the **SBC** command.

The **DBC** command displays the currently set break conditions.

```
Break Condition ───► mnemonic
```

The *mnemonic* indicates the set break condition.

SBC, DBC

Execution Example

```
64153> SBC
    BP Condition SET? (Y/N)    Not change
    CC Condition SET? (Y/N)    Not change
    TF Condition SET? (Y/N) Y
    AP Condition SET? (Y/N) N
    PD Condition SET? (Y/N) Y
    XP Condition SET? (Y/N) N
64153> DBC
    Break Condition -----> TF PD
64153> SBC ~TF ~PD
64153> DBC
    Break Condition -----> Not instituted
64153>
```

3.1.1.5.2 Setting Breaks on Executed Addresses

DBP, CBP

Input Format

DBP Δ *parm1* [Δ *parm1* ..... Δ *parm1*] ↵
DBP Δ * ↵
parm1 : *address*
: [*address* Δ *address*]

CBP Δ *parm2* [Δ *parm2* ..... Δ *parm2*] ↵
CBP Δ * [= *data*] ↵
parm2 : *address = data*
: [*address* Δ *address*] = *data*

data : **0,1**

Description

The **DBP** command displays the contents of the breakpoint bits (☞1).

Each *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of breakpoint bit memory (☞2).

Display contents are one of the following, depending on input format.

address	Displays the contents on one address.
[*address* Δ *address*]	Displays the range enclosed in [ ].
*	Displays the entire area of breakpoint bit memory.

When multiple parameters are specified, each will be displayed even if their address areas overlap.

Each address with its breakpoint bit set to "1" will have its breakpoint bit break enabled. Each one set to "0" will have its breakpoint bit break disabled.

DBP , CBP

The **CBP** command changes the contents of breakpoint bit memory.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of breakpoint bit memory (☞2).

The *data* is a value of "0" or "1," indicating the changed breakpoint bit value. Contents are changed in the order of the input parameters. If '*' is input and *data* is omitted, then the entire area will be set to '0.'

The area changed is one of the following, depending on input format.

address	Changes the contents on one address.
[*address* Δ *address*]	Changes the range enclosed in [ ].
*	Changes the entire area of breakpoint bit memory.

When multiple parameters are specified, each will be changed even if their address areas overlap.

☞ 1

Breakpoint bits correspond one-for-one with addresses in code memory. They are used to cause breaks at specified locations in a user program when executed with the **G** command.

A breakpoint bit break is enabled when the breakpoint bit is "1." However, the only breakpoint bits that can generate breaks are those corresponding to the address of the first byte of an instruction code in the user program.

Breakpoint bits are enabled as realtime emulation break conditions only when set as a break condition with **BP**.

When an object file is loaded by the **LOD** command with **/B** option, the breakpoint bits corresponding to the loaded addresses will all be set to "0."

☞ 2

Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

Execution Example

```
64153> DBP [100 190]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0100    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0110    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0120    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0130    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0140    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0150    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0160    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0170    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0180    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0190    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> DBP 300
       LOC = 0300    0
64153> DBP 120 203 340 450 678 924
       LOC = 0120    0
       LOC = 0203    0
       LOC = 0340    0
       LOC = 0450    0
       LOC = 0678    0
       LOC = 0924    0
```

DBP , CBP

Execution Example

```
64153> CBP [23 45]=1
64153> DBP [20 60]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0020    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0030    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0040    1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
       LOC = 0050    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0060    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> CBP 53=1 57=1 5A=1 5D=1 60=1 62=1 66=1 6F=1
64153> DBP [20 56]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0020    0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0030    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0040    1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
       LOC = 0050    0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0
64153> CBP *
```

3.1.1.5.3 Displaying Break Results

DBS

Input Format

DBS ↵

Description

The **DBS** command displays the break conditions from realtime emulation in the following format.

```
STATUS = Break-Condition
```

One of the following break conditions is entered for *Break-Condition*.

Address Break	Break on a **G** command break address.
Breakpoint Break	Break on a breakpoint bit.
Address Pass Break	Break on the parameters from the **G** command. (Address Pass Break)
Address Pass Count Break	Break on the parameters from the **G** command. (Address Pass Count Break)
RAM Data Match Break	Break on the parameters from the **G** command. (Data Match Break, ☞1)
Probe Data Match Break	Break on the parameters from the **G** command. (Data Match Break)
Cycle Counter Overflow Break	Break on cycle counter overflow.
Trace Full Break	Break on trace pointer overflow.
Step Break	Break on step execution.
ESC Break	Break on an **ESC** command.
Address Pass Counter Overflow Break	Break on address pass counter overflow.
Power down break	Break when the MSM64E153 evaluation chip enters halt mode.
External Break	Break on an external break signal (☞2).
RAM Address Break	Break on access to non-existent RAM.
N Area Break	Break on execution of non-existent code memory area.
NO Break Status	Indicates no break conditions.

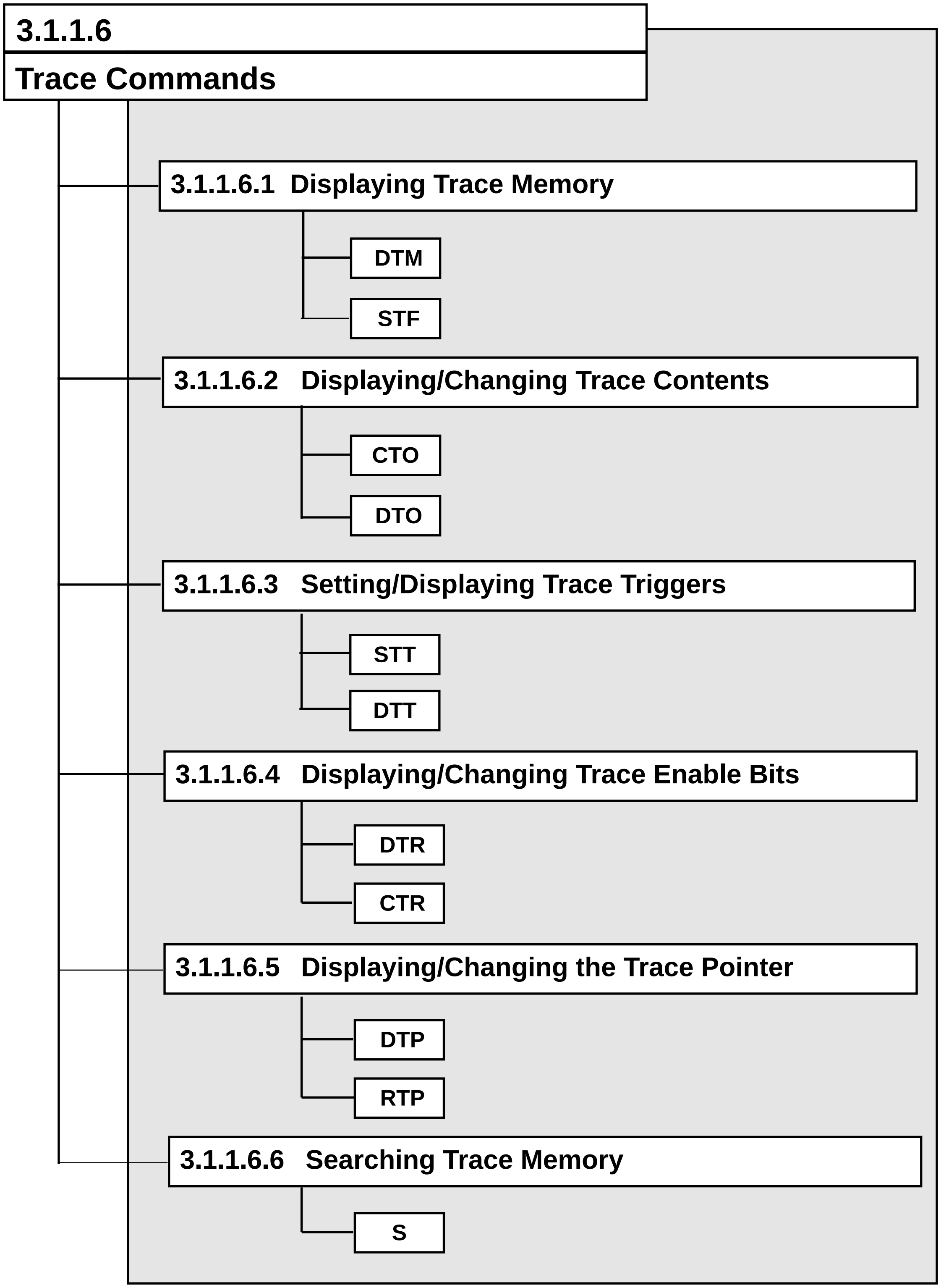
☞ 1 The timing of a RAM Data Match Break is such that the break will occur after execution of the next instruction following the instruction that satisfied the break condition.

☞ 2 A break will occur when the input signal on the probe cable's external break pin transitions from "L" to "H."

! The break status will be "No Breakstatus" when the emulator is turned on.

Execution Example

```
64153> CBP 3FH=1
64153> G 0,55
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0055], Next PC =[0056], TP=[0086]
64153> DBS

    ***** Address Break *****
64153> CBP [0 0BFFH]=0
       ** Error 103: Input data out of range.
64153> G 0,55
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0055], Next PC =[0056], TP=[0086]
64153> DBS

    ***** Address Break *****
```


3.1.1.6
Trace Commands
3.1.1.6.1 Displaying Trace Memory
DTM
STF
3.1.1.6.2 Displaying/Changing Trace Contents
CTO
DTO
3.1.1.6.3 Setting/Displaying Trace Triggers
STT
DTT
3.1.1.6.4 Displaying/Changing Trace Enable Bits
DTR
CTR
3.1.1.6.5 Displaying/Changing the Trace Pointer
DTP
RTP
3.1.1.6.6 Searching Trace Memory
S

3.1.1.6.1 Displaying Trace Memory

DTM

Input Format

DTM Δ *parm* ↵

parm : - $number_{-step}$ Δ $number_{step}$
: $number_{Tp}$ Δ $number_{step}$
: *

Description

The **DTM** command displays the contents of trace memory as specified by *parm*. Trace memory is an 8192 x 64-bit RAM area (☞1).

The $number_{-step}$ indicates a number of steps back from the current trace pointer value (called TP below). The $number_{step}$ indicates the number of steps to display as a decimal number 1–8192. The $number_{Tp}$ indicates the TP value at which to start the trace display as a decimal number 0–8191 (☞2). The * indicates that the contents of TP to TP-1 should be displayed if the trace pointer has overflowed, or the contents of 0 to TP-1 should be displayed if it has not.

Trace memory stores various information from realtime emulation. An operator can debug more efficiently by viewing this information.

As shown below, trace memory is configured as a ring, so during realtime emulation trace memory will be overwritten in order from the oldest contents first.

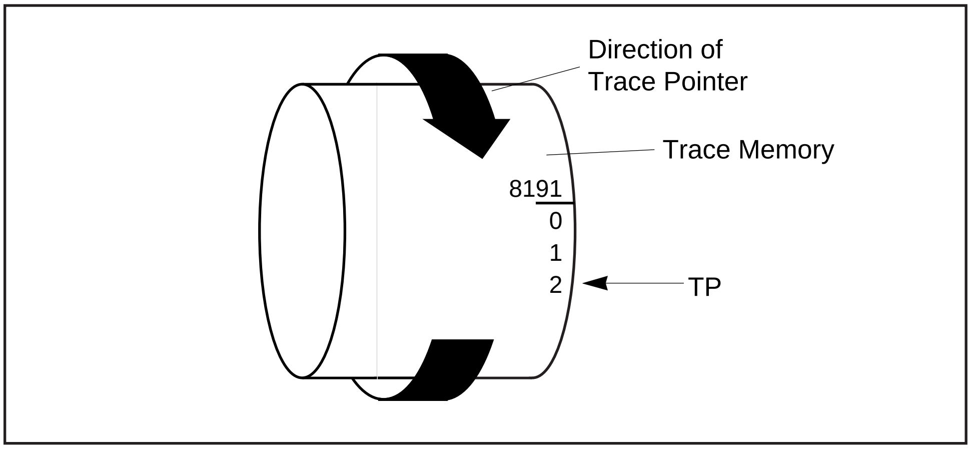
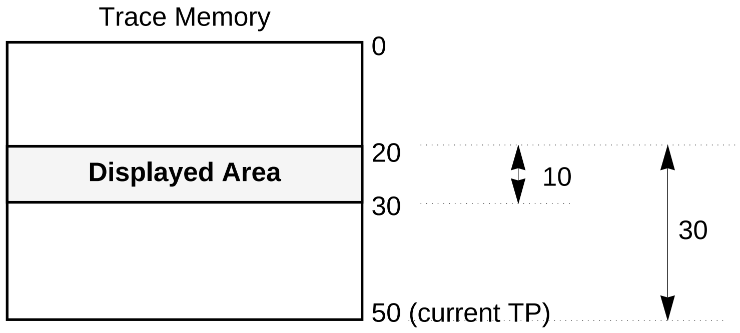


Figure 3-1. Trace Pointer Example

The following examples show the difference between input of $-number_{-step}$ $number_{step}$ and $number_{Tp}$ $number_{step}$. Assume that the current TP is 50.

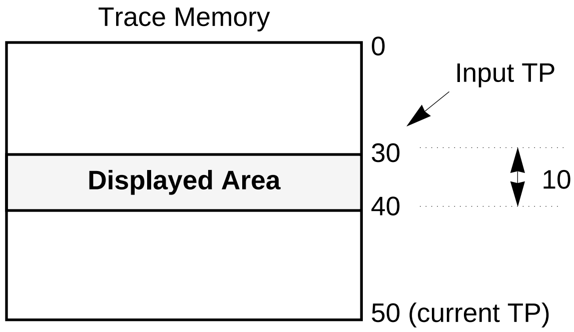
Example

```
DTM -30 10
```



Example

```
DTM 30 10
```



After the parameters are correctly input and a carriage return is pressed, a header in the format below will be displayed, followed by the trace memory contents for each trace pointer value.

```
LOC  MNEMONIC     SP  P2  P6D A B H L TP
```

The header is displayed every 8 steps. Trace data is shown as numbers only where it changes. It is displayed as '.' where it has not changed from the previous step. However, the trace data immediately after a header is always displayed as numbers.

The above header is the initial display state.

The trace contents displayed and the corresponding headers are shown below.

LOC	Program counter	
Code	Executed instruction code	
MNEMONIC	Executed instruction (☞ 3)	
RAM	Data memory address, data	
RAMA	Data memory address	
RAMD	Data memory data	
C	Carry flag	
MI	Master interrupt flag	
INT	Interrupt operation flag (☞ 6)	
SKIP	Skip execution flag (☞ 7)	
A	A register	
B	B register	
H	H register	(specified with **CTO** command)
L	L register	(specified with **CTO** command)
X	X register	(specified with **CTO** command)
Y	Y register	(specified with **CTO** command)
SP	Stack pointer	
P0, P1, P2,P3, **P4D, P5D, P6D, P7D**	Port data	(specified with **CTO** command) (☞ 4)
TP	Trace pointer (☞ 5)	

Which data in trace memory will be displayed is set by the **STF** command.

☞ 1 The EASE64158 control system's trace memory has a maximum area of 8192 x 64 bits, but the EASE64158 only uses 8192 x 63 bits of that.

☞ 2 Keep in mind the following points when displaying the contents of trace memory.

- If trace memory has not overflowed, then trace data will only be stored in trace memory from 0 to the current TP-1. Accordingly, if the input TP is greater than the current TP, then an error will result. If the number of back steps is greater than the current TP, then trace memory from 0 will be displayed.

- If trace memory has overflowed, then trace data will be stored in the entire trace memory (0–8191), regardless of the current TP. Accordingly, if the number of back steps is greater than the current TP, then data before a TP of 0 (8191, 8190, 8189, ...) will be displayed.

☞ 3 The following *mnemonics* set by the **STF** command correspond to the (header) trace contents displayed by the **DTM** command.

INS ◄——► **MNEMONIC**
INSC ◄——► **MNEMONIC, Code**

☞ 4 Available ports differ for each MSM64153 family microcontroller. For details, refer to each microcontroller's User's Manual.

☞ 5 The TP is always displayed. (It cannot be set with the **STF** command.)

! Trace data display will be delayed by one instruction except for INT and SKIP.

☞ 6 The INT flag indicates an interrupt operation, which will be set to "1" at the instruction in which the interrupt is executed.

☞ 7 The SKIP flag indicates skip execution of instructions, which will be set to "1" at the instruction in which skip execution is executed.

DTM

Execution Example

```
64153> LOD INT1
    HEX File Loading...
    Load Completed   address[002C - 0288]

64153> DASM 100

LOC=0100           START:
LOC=0100  90             LAI     0H
LOC=0101  50 3F          LHLI    IRQ2      ;3EH
LOC=0103  6F             LMA
LOC=0104  50 3F          LHLI    IRQ3      ;3FH
LOC=0106  6F             LMA
LOC=0107  50 7C          LHLI    MIEF      ;7CH
LOC=0109  6F             LMA
LOC=010A  50 0E          LHLI    P6CON     ;EH
LOC=010C  91             LAI     1H
LOC=010D  6F             LMA
LOC=010E  50 0F          LHLI    P7CON     ;FH
LOC=0110  91             LAI     1H
LOC=0111  6F             LMA
LOC=0112  50 0A          LHLI    P21E      ;AH
LOC=0114  93             LAI     P3        ;3H

64153> G 100,200
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0200], Next PC =[0202], TP=[0061]
```

Execution Example

```
64153> DTM 10 10

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=010E   LHLI    FH          FF  0  0  1 0 0 E 0010
LOC=0110   LAI     1H          ..  .  .  . . . F 0011
LOC=0111   LMA                 ..  .  .  . . . . 0012
LOC=0112   LHLI    AH          ..  .  .  . . . . 0013
LOC=0114   LAI     3H          ..  .  .  . . . A 0014
LOC=0115   LMA                 ..  .  .  3 . . . 0015
LOC=0116   LHLI    39H         ..  .  .  . . . . 0016
LOC=0118   LAI     1H          ..  .  .  . . 3 9 0017

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=0119   LMA                 FF  0  0  1 0 3 9 0018
LOC=011A   LHLI    3AH         ..  .  .  . . . . 0019

64153> DTM -30 10

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=0101   LHLI    3EH         F7  0  0  0 0 7 C 0031
LOC=0103   LMA                 ..  .  .  . . 3 E 0032
LOC=0104   LHLI    3FH         ..  .  .  . . . . 0033
LOC=0106   LMA                 ..  .  .  . . . F 0034
LOC=0107   LHLI    7CH         ..  .  .  . . . . 0035
LOC=0109   LMA                 ..  .  .  . . 7 C 0036
LOC=010A   LHLI    EH          ..  .  .  . . . . 0037
LOC=010C   LAI     1H          ..  .  .  . . 0 E 0038

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=010D   LMA                 F7  0  0  1 0 0 E 0039
LOC=010E   LHLI    FH          ..  .  .  . . . . 0040
```

STF

Input Format

STF [Δ *parm* ..... Δ *parm*] ↵
STF [~] ALL ↵
parm : [~] *mnemonic*

Description

The **STF** command changes the trace mnemonics displayed by the **DTM** command. One of the following is entered for *mnemonic*.

If a *mnemonic* is prefixed by a '~' (tilde), then its setting will be cancelled. If *parm* is omitted, then the currently set display format will be displayed.

mnemonic:

INS	Executed instruction
INSC	Executed instruction code
LOC	Executed address
RAM	Data memory address, data
RAMA	Data memory address
RAMD	Data memory data
C	Carry flag
MI	Master interrupt flag
INT	Interrupt operation flag
SKIP	Skip execution flag
A	A register
B	B register
H	H register (specified with **CTO** command) (☞ 1)
L	L register (specified with **CTO** command) (☞ 1)
X	X register (specified with **CTO** command) (☞ 1)
Y	Y register (specified with **CTO** command) (☞ 1)
SP	Stack pointer
P0, P1, P2, P3, **P4D, P5D,** **P6D, P7D**	Port data (specified with **CTO** command) (☞ 2)

ALL: Sets **LOC, INS, SP, P2, P6D, A, B, H, L** (☞ 3)

☞ 1

When a new port or a register is set with the **CTO** command, the ports or registers set with a previous **STF** command will be cancelled. Thus, trace ports will need to be set again with the **STF** command.

☞ 2

Available ports differ for each MSM64153 family microcontroller. For details, refer to each microcontroller's User's Manual.

☞ 3

If ~ALL is input, then only the executed address (**LOC**) and instruction (**INS**) will be set.

!

C, MI, INT, and SKIP cannot be set when **EXPAND** is ON.

Execution Example

```
64153> DTM 0 8

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=0100   LAI     0H          FF  0  0  0 0 0 0 0000
LOC=0101   LHLI    3EH         ..  .  .  . . . . 0001
LOC=0103   LMA                 ..  .  .  . . 3 E 0002
LOC=0104   LHLI    3FH         ..  .  .  . . . . 0003
LOC=0106   LMA                 ..  .  .  . . . F 0004
LOC=0107   LHLI    7CH         ..  .  .  . . . . 0005
LOC=0109   LMA                 ..  .  .  . . 7 C 0006
LOC=010A   LHLI    EH          ..  .  .  . . . . 0007

64153> STF
LOC        MNEMONIC            SP P2 P6D A B H L TP
64153> DTO
    Set Trace Object = P2 P6D HL
```

STF

Execution Example

```
64153> DTM 0 8

LOC        MNEMONIC            SP P2 P6D A B H L C INT TP
LOC=0100   LAI     0H          FF  0  0  0 0 0 0 0  0  0000
LOC=0101   LHLI    3EH         ..  .  .  . . . . .  .  0001
LOC=0103   LMA                 ..  .  .  . . 3 E .  .  0002
LOC=0104   LHLI    3FH         ..  .  .  . . . . .  .  0003
LOC=0106   LMA                 ..  .  .  . . . F .  .  0004
LOC=0107   LHLI    7CH         ..  .  .  . . . . .  .  0005
LOC=0109   LMA                 ..  .  .  . . 7 C .  .  0006
LOC=010A   LHLI    EH          ..  .  .  . . . . .  .  0007

64153> EXPAND ON

    CODE Memory has been expanded 32K byte.
    ***** EVA CHIP RESET *****

64153S> STF C
       ** Error 158: Illegal parameter.  (Can't use in EXPAND mode)
64153S> DTM 0 3

LOC        MNEMONIC            SP P2 P6D A B H L TP
LOC=0100   LAI     0H          FF  0  0  0 0 0 0 0000
LOC=0101   LHLI    3EH         ..  .  .  . . . . 0001
LOC=0103   LMA                 ..  .  .  . . 3 E 0002

64153S> EXPAND OFF

    Expanded CODE Memory allocation was released.
    ***** EVA CHIP RESET *****
```

3.1.1.6.2 Displaying/Changing Trace Contents

DTO, CTO

Input Format

DTO ↵

CTO Δ *parm* [Δ *parm* [Δ *parm*]] ↵
parm : *mnemonic*

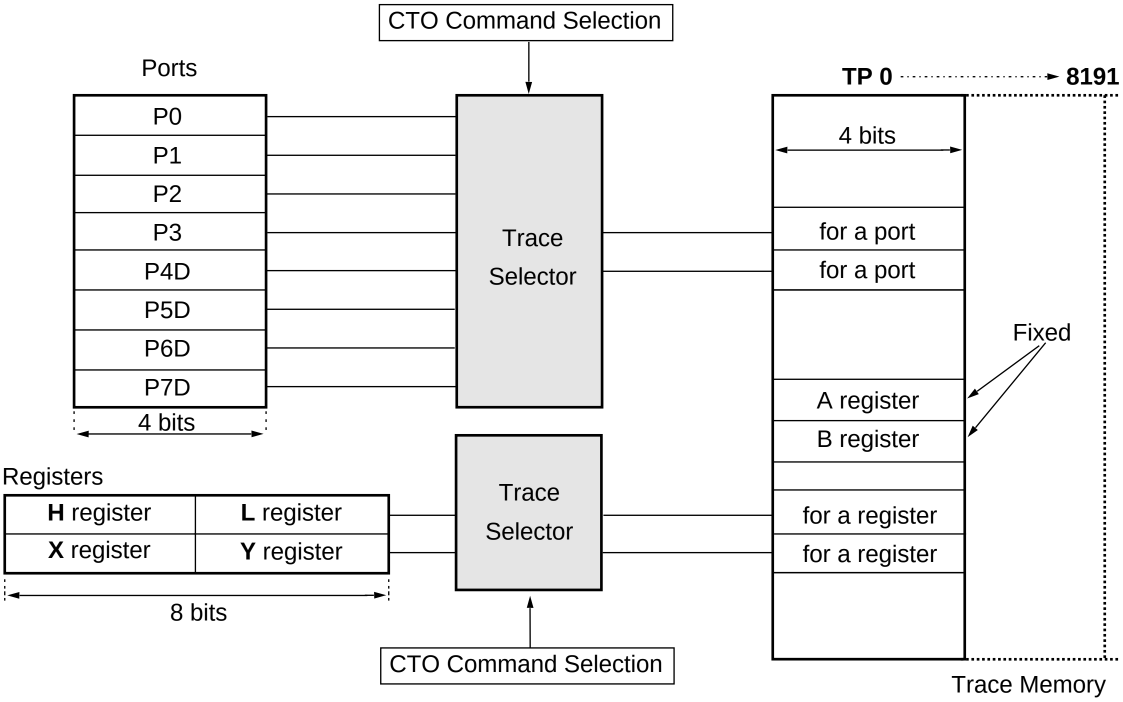
Description

EASE64158 trace memory has an area for storing port data trace results for two ports. The operator can select any two of the eight ports to trace with the **CTO** command.

Also, trace memory has an area for storing register trace results for four registers. Of these four, two are fixed for the A and B registers. The remaining two can be selected with the **CTO** command as either the **HL** registers or the XY registers.

Thus, up to two ports and one register can be set (☞ 1).

The **DTO** command displays the settings of the **CTO** command.



DTO, CTO

The following can be input for *mnemonic* (☞ 2).

P0	Port 0
P1	Port 1
P2	Port 2
P3	Port 3
P4D	Port 4
P5D	Port 5
P6D	Port 6
P7D	Port 7
HL	HL register
XY	XY register

When power is turned on, the default settings will be the HL register and ports corresponding to the appropriate MSM64153 family device (☞ 3).

When the traced ports are changed with the **CTO** command, the trace pointer is cleared to 0.

☞ 1

When a new port is set with the **CTO** command, the ports set with a previous **STF** command will be cancelled. Thus, to display ports with the **DTM** command, trace ports will need to be set again with the **STF** command.

☞ 2

The available ports differ depending on the particular MSM64153 family microcontroller. For details, refer to the user's manual of the appropriate microcontroller.

☞ 3

The default ports when power is turned on are as follows.

Chip Mode	Default Ports
MSM64152	P2, P6D
MSM64153	P2, P6D
MSM64155	P2, P6D
MSM64158	P2, P6D

Execution Example

```
64153> DTO
    Set Trace Object = P2 P6D HL
64153> STF
LOC        MNEMONIC            SP P2 P6D A B H L TP
64153> CTO P3 P7D XY
    Trace Pointer Cleared
64153> DTO
   Set Trace Object = P3 P7D XY
64153> STF
LOC        MNEMONIC            SP A B TP
64153> STF P3 P7D X Y
64153> STF
LOC        MNEMONIC            SP P3 P7D A B X Y TP
64153> DTM 0 3

LOC        MNEMONIC            SP P3 P7D A B X Y TP
LOC=0100   LAI     0H          FF  0  0  0 0 0 0 0000
LOC=0101   LHLI    3EH         ..  .  .  . . . . 0001
LOC=0103   LMA                 ..  .  .  . . 3 E 0002
```

3.1.1.6.3 Setting/Displaying the Trace Trigger

STT

Input Format

STT Δ *mnemonic1* ↵

STT Δ *mnemonic2* [/ [*parm1*] / [*parm2*]] ↵

```
parm1, parm2   : address
               : [ address Δ address ]
               : .
```

STT Δ *mnemonic3 trc_mnem* [*&mask*] = *data* ↵

```
trc_mnem       : PRB (Probe) ↵
               : RAM [ Δ ram_addr ]
```

Description

The **STT** command sets the conditions for tracing (trace trigger).

One of the following is input for *mnemonic*.

<*mnemonic1*>

ALL Trace all addresses in code memory during realtime emulation (free-running trace).

TR Trace only addresses with their trace enable bits set during realtime emulation (trace enable bit trace).

DIS Do not trace during realtime emulation (trace disable).

<*mnemonic2*>

SS Start tracing at the address specified by *parm1*, and stop tracing at the address specified by *parm2*.

The *parm1* indicates the trace start address. The start condition is one of the following, depending on input format.

address	Start tracing when the specified program address is executed.
[*address* Δ *address*]	Start tracing when any program address in the specified range is executed.
.	Start tracing when **G** command execution begins.
No input	Start tracing when the program address specified by the previous **STT** command is executed.

The *parm2* indicates the trace stop address. The stop condition is one of the following, depending on input format (☞1).

address Stop tracing when the specified program address is executed.

[*address* Δ *address*] Stop tracing when any program address in the specified range is executed.

. Trace continuously through **G** command execution (☞2).

No input Stop tracing when the program address specified by the previous **STT** command is executed.

If the parameters are omitted, then the emulator will display the following message and wait for input.

```
START -----> st-parm
```

Here the operator should input the trace start address for *st-parm* shown above.

The operator can also input one of the following keys instead of a start address.

. ↵ Start incrementing the trace trigger when G command execution begins.

- ↵ Re-enter the input.

Δ↵ Do not change the current setting.

↵ Do not change the current setting, and terminate the STT command.

STT, DTT

After *st-parm* has been input, the emulator will display the following message and wait for stop address input.

```
        STOP ----> stp-parm
```

Here the operator should input the trace stop address for *stp-parm* shown above.

The operator can also input one of the following keys instead of a stop address.

. ↵ Start incrementing the trace trigger when G command execution begins.

- ↵ Re-enter the input.

Δ↵ Do not change the current setting.

↵ Do not change the current setting, and terminate the STT command.

The debugger actually sets these two parameters when input is finished.

☞ 1 The trace pointer will not be incremented at the stop address specified by *parm2*.

☞ 2 If '.' is specified for *parm2*, then break addresses will also be traced.

<mnemonic3>

AD Start tracing when the value of *data* matches the contents of *trc_mnem*, or the masked contents of *trc_mnem* (trace after data match).

BD Stop tracing when the value of *data* matches the contents of *trc_mnem*, or the masked contents of *trc_mnem* (trace before data match).

The data match can be with either the probe pins or RAM for *trc_mnem*. These are specified by "PRB" or "RAM [*ram_addr*]. The *ram_addr* indicates address in the RAM, and can be omitted. The mask can have a value of 0–0FH for "RAM" and 0–0FFH for "PRB." The bits where the mask is "1" are ignored.

When EASE64158 power is turned on, the trace trigger is initialized to ALL.

SEE **DTR, CTR**

! If the trace trigger has been set to trace after data match (AD) or trace before data match (BD), and the **G** command break condition is set to a PRB or RAM data match break, then the trace trigger condition will be changed to free-run trace (ALL). In other words, the trace trigger condition will not be effective, while the break condition will be effective. Afterwards the trace trigger condition will remain as free-run trace (ALL) until it is set again with the **STT** command.

Execution Example Refer to the **DTT** command.

DTT

Input Format

DTT ↵

Description

The **DTT** command displays the current trace trigger set by the **STT** command.

Execution Example

```
64153> STT ALL
64153> DTT
    Current Trace Trigger : ALL
64153> STT SS
    Current Trace Trigger : ALL

    START ---> 10

    END   ---> 20


64153> DTT
    Current Trace Trigger : SS
     START ADDRESS : 0010
     STOP ADDRESS  : 0020
```

3.1.1.6.4 Displaying/Changing Trace Enable Bits

DTR

Input Format

DTR ∆ *parm* [∆ *parm* ..... ∆ *parm*] ↵

DTR ∆ * ↵

parm : *address*
: [*address* ∆ *address*]

Description

The **DTM** command displays the contents of trace enable bits.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory to be displayed (☞ 1).

Display contents are one of the following, depending on input format.

address	Displays the contents on one address.
[*address* ∆ *address*]	Displays the range enclosed in [ ].
*	Displays the entire area of trace enable bits.

When multiple parameters are specified, each will be displayed even if their address areas overlap.

Trace enable bits correspond one-for-one with the program memory area. The user can control trace execution by manipulating these bits.

When TR has been set with the **STT** command and the EASE64158 is executing a user program, the emulator examines the trace enable bit at the each address of each executed instruction code. If a trace enable bit is "1," then the trace information at that time will be written to trace memory. Thus, the user can write only the trace information he needs into trace memory by setting the appropriate trace enable bits to "1."

Only trace enable bits set at the first byte of an instruction code are effective.

Addresses where the displayed contents are "1" indicate addresses to be traced. Addresses where the displayed contents are "0" indicate addresses not to be traced.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

```
64153> DTR [20 80]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0020    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0030    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0040    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0050    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0060    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0070    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0080    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> DTR 0B00H 0A34H
       LOC = 0B00    0
       LOC = 0A34    0
```

CTR

Input Format

CTR Δ *parm* [Δ *parm* ..... Δ *parm*] ↵

CTR Δ * [=*data*] ↵

parm : *address* = *data*
: [*address* Δ *address*] = *data*

Description

The **CTR** command changes the contents of trace enable bits.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory (☞ 1).

If '*' is input and *data* is omitted, then the entire area will be set to '0.'

Contents are changed in the order of the input parameters. The area changed is one of the following, depending on input format.

address	Changes the contents on one address.
[*address* Δ *address*]	Changes the range enclosed in [ ].
*	Changes the entire area of code memory.

When multiple parameters are specified, each will be changed even if their address areas overlap.

The *data* is the value of the change data. Its value is 0 or 1. Set addresses to be traced to '1,' and addresses not to be traced to '0.'

Only trace enable bits set at the first byte of an instruction code are effective.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

CTR

Execution Example

```
64153> DTR [0 45]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0000    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0010    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0020    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0030    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0040    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> CTR [26 2B]=1
64153> DTR [0 45]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0000    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0010    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0020    0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
       LOC = 0030    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0040    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> STT TR
64153>
```

3.1.1.6.5 Displaying/Clearing the Trace Pointer

DTP, RTP

Input Format

DTP ↵ ← Display trace pointer

RTP ↵ ← Clear trace pointer

Description

The **DTP** command displays the current trace pointer value and its overflow state. The overflow state displays as '1' when the trace pointer has overflowed, and '0' when it has not.

The displayed values are decimal data

The **RTP** command clears the trace pointer value to '0.' The EASE64158 initializes the trace pointer to '0';

- when power is turned on,
- when the **CTO** command is executed, and
- when start address is input with the **G** command.

Execution Example

```
64153> DTP
    Trace Pointer -----> 0061  Overflow = 0
64153> RTP
    Reset Trace Pointer
64153> DTP
    Trace Pointer -----> 0000  Overflow = 0
```

DTP, RTP

Execution Example

```
64153> G 100,240
    Reset Trace Pointer

    ***** Emulation Go *****
Go >>

    ***** Address Break *****
    Break PC =[0240], Next PC =[0241], TP=[0000]
64153> DTP
    Trace Pointer -----> 0000  Overflow = 0
64153> RTP
    Reset Trace Pointer
64153> DTP
    Trace Pointer -----> 0000  Overflow = 0
64153>
```

3.1.1.6.6 Searching Trace Memory

S

Input Format

S [~] *mnemonic* = *data* [*parm*] ↵

parm : [*count*]
: [*start_count* Δ *end_count*]

mnemonic	: **LOC**	Program counter	
	: **RAMA**	Memory address	
	: **RAMD**	Memory data	
	: **C**	Carry flag	
	: **MI**	Master interrupt flag	
	: **INT**	Interrupt transfer flag	
	: **SKIP**	Skip execution flag	
	: **A**	A register	
	: **B**	B register	
	: **H**	H register	Registers specified by the **CTO** command.
	: **L**	L register	
	: **X**	X register	
	: **Y**	Y register	
	: **SP**	Stack pointer	
	: **P0, P1, P2, P3,** **P4D, P5D, P6D, P7D**	Port data (2 ports specified by the **CTO** command)	

data = *search data* (comparison data)
count = count for satisfying comparison criteria during searches
start_count = start count for satisfying comparison criteria during searches
end_count = end count for satisfying comparison criteria during searches

☞ 1 Available ports differ for each MSM64153 family microcontroller. For details, refer to each microcontroller's User's Manual.

Description

The **S** command searches trace data in trace memory. It searches for a match between the data of the trace mnemonic specified by *mnemonic* and the trace data specified by data, and then displays the trace information.

When a '~' (tilde) is input, the search is performed from the oldest trace data to the newest. When a '~' is not input, the search is performed from the newest data to the oldest.

The *parm* indicates a count for satisfying comparison criteria during searches.

If *count* = 3 — Displays the contents of trace memory at the TP when the comparison criteria is satisfied the third time.

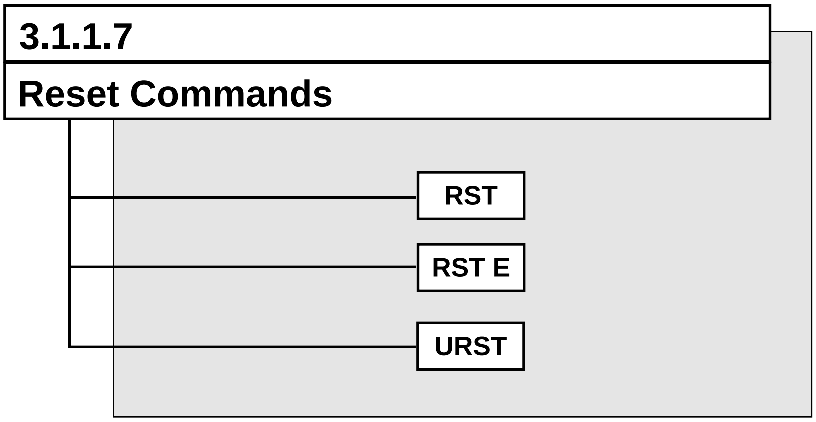
If *start_count* = 1 and *end_count* = 3 — Displays the contents of each trace memory at the TP when the comparison criteria is satisfied the first, second, and third times.

If *parm* is omitted, then the **S** command displays the contents of trace memory at the TP when the comparison criteria is satisfied the first time. The *count, start_count, and end_count* have decimal values of 1-8192.

The oldest and the newest trace data will be handled in the same manner as the **DTM** command. Refer to ☞2 in **DTM** command.

Execution Example

```
64153> S B=5


LOC         MNEMONIC            SP P2 P6D A B H L TP
LOC=0002    LBA                 FF  0  0  6 5 5 5 3502

64153> S H=1


LOC         MNEMONIC            SP P2 P6D A B H L TP
LOC=0005    INH                 FF  0  0  2 2 1 1 0704

64153>
```



3.1.1.7

Reset Commands

RST

Input Format

RST ↵

Description

The **RST** command resets the EASE64158 as follows.

Item	Reset State
MSM64E153 evaluation chip	Resets to same as when a reset is input to the appropriate MSM64153 family member.
Break status	Cleared to state of no breaks generated.
Cycle counter	Cleared to 0.
Address pass counters 0 - 3	Cleared to 0.

Execution Example

```
64153> RST

    ***** SYSTEM RESET *****

SID64K Symbolic Debugger Ver.1.00b Jun 1993
Copyright (c) 1993. OKI Electric Ind. Co.,Ltd.

64153>
```

SEE Refer to Table 2-6 of Chapter 2 for details on initialization by applying power or pressing the reset switch of the EASE64158.

RST E

Input Format

RST Δ E ↵

Description

The **RST E** command resets the EASE64158 as follows.

- Resets the MSM64E153 evaluation chip.

After this command is executed, the MSM64E153 evaluation chip will be reset to the same state as the appropriate MSM64153 family microcontroller (refer to the appropriate user's manual of the MSM64153 family microcontroller for details about its state after reset).

Execution Example

```
64153> RST E

   ***** EVA CHIP RESET *****

64153>
```

URST

Input Format

URST [Δ *mnemonic*]

Description

The **URST** command sets whether the USER RESET pin (pin 43) input of the user cable of user connector 2 is enabled or not.

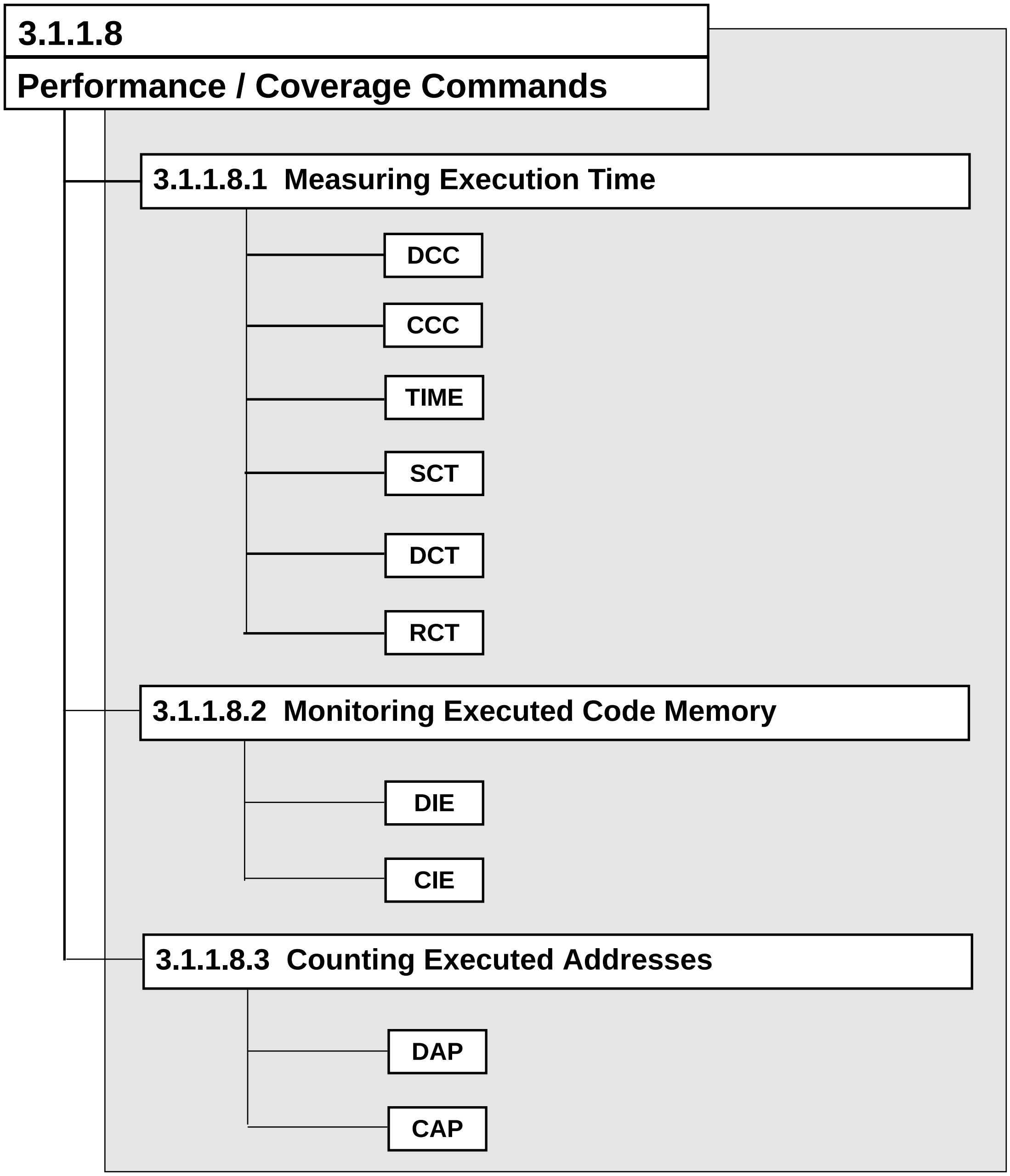
One of the following parameters is entered for *mnemonic*.

ON	:	Reset inputs during realtime emulation are enabled.
OFF	:	Inputs from the USER•RESET pin are prohibited.

If *mnemonic* is omitted, then the current setting will be displayed.

Execution Example

```
64153> URST

    User reset disable
64153> URST ON

64153> URST

    User reset enable
64153> URST OFF
```


3.1.1.8
Performance / Coverage Commands
3.1.1.8.1 Measuring Execution Time
DCC
CCC
TIME
SCT
DCT
RCT
3.1.1.8.2 Monitoring Executed Code Memory
DIE
CIE
3.1.1.8.3 Counting Executed Addresses
DAP
CAP

3.1.1.8.1 Measuring Execution Time

DCC

Input Format

DCC ↵

Description

The **DCC** command displays information about the cycle counter.

```
CURRENT STATUS  --------► value Time =time  Overflow =data
```

The *value* is the cycle counter value. The time is value converted to a time. Both are displayed as decimal numbers.

The *data* is '1' if the cycle counter has overflowed, or '0' if it has not.

The cycle counter is a 32-bit counter used for measuring program execution time. Also, cycle counter overflow can be used as a break condition.

! The cycle counter is incremented by 1 every single machine cycle.

Execution Example

```
64153> DCC
    CURRENT STATUS -----> 4294967172  Time = 858993.4344m (sec)
                          Overflow = 0
64153>
```

CCC

Input Format

CCC ∆ [-] *data* ↵

Description

The **CCC** command changes the contents of the cycle counter to the value specified by *data*. The *data* is a decimal number 0–4294967295. If *-data* is input, then the cycle counter will be changed to the value of 4294967295-*data*.

Below are cycle counter overflow examples where the cycle counter is set to *data* and *-data*.

Examples : CCC 4294967279 ········ Overflow will occur when 16 cycles have elapsed after the cycle counter is started.

CCC -100 ···················· The cycle counter is set to 4294967195. Overflow will occur when 101 cycles have elapsed after the cycle counter is started.

Execution Example

```
64153> DCC
  CURRENT STATUS -----> 4294967172 Time = 858993.4344m (sec)
                        Overflow = 0
64153> CCC 100
64153> DCC
  CURRENT STATUS -----> 100 Time = 20.0000u (sec) Overflow = 0
64153> CCC -123
64153> DCC
  CURRENT STATUS -----> 4294967172 Time = 858993.4344m (sec)
                        Overflow = 0
64153>
```

TIME

Input Format

TIME [Δ *exp*] ↵

exp : *data*

Description

The **TIME** command sets the time of a single machine cycle for the time display of the **DCC** command.

Input the time of a single machine cycle (µs) for *data*. Up to five places after the decimal point are valid, with the fifth position being rounded up or down. Values are input in microseconds (µs), but are displayed after a unit conversion in milliseconds (ms), microseconds (µs), or nanoseconds (ns).

If *data* is omitted, the current time setting will be displayed.

The default value sets one cycle's operating time as 91.0 µs, assuming that the MSM64153 family's operating frequency is 36.768 kHz.

! The value input with the **TIME** command only affects displays of execution time with the **DCC** command. It does not affect emulation execution time in any way.

Execution Example

```
64153> TIME
    Time = 0.2000u (sec)
64153> TIME 1000
64153> TIME
    Time = 1.0000m (sec)
64153> TIME 10.234
64153> TIME
    Time = 10.2340u (sec)
64153> TIME 0.2
64153> TIME
    Time = 0.2000u (sec)
64153>
```

SCT

Input Format

SCT [Δ / [*parm1*] / [*parm2*] ↵

parm1, parm2 : *address*
: [*start_address* Δ *end_address*]
: .

Description

The **SCT** command sets that starting and stopping addresses for incrementing the cycle counter. This command allows the cycle counter to be incremented during **G** command execution.

The *parm1* indicates the cycle counter increment start address. The start condition is one of the following, depending on input format.

address — Start incrementing when the specified program address is executed.

[*address* Δ *address*] — Start incrementing when any program address in the specified range is executed.

. — Start incrementing when **G** command execution begins.

No input — Start incrementing when the program address specified by the previous **SCT** command is executed.

The *parm2* indicates the cycle counter increment stop address. The stop condition is one of the following, depending on input format (☞1).

address — Stop incrementing when the specified program address is executed.

[*address* Δ *address*] — Stop incrementing when any program address in the specified range is executed.

. — Increment continuously through **G** command execution (☞2).

No input — Stop incrementing when the program address specified by the previous **SCT** command is executed.

If *parm1* is omitted, then the emulator will display the following message and wait for input.

```
        START   status ------------► st-parm
```

Here status indicates the current setting of the start address. The operator should input the cycle counter increment start address for *st-parm*. The operator can also input one of the following keys instead of a start address.

. ↵ Start incrementing the cycle counter when **G** command execution begins.

- ↵ Re-enter the input.

_ ↵ Do not change the current setting.

↵ Do not change the current setting, and terminate the **SCT** command.

If *parm2* is omitted, then the emulator will display the following message and wait for input.

```
STOP  status ------------► stp-parm
```

Here status indicates the current setting of the stop address. The operator should input the cycle counter increment stop address for *stp-parm*. The operator can also input one of the following keys instead of a stop address.

. ↵ Stop incrementing the cycle counter when **G** command execution ends.

- ↵ Re-enter the input from the start.

_ ↵ Do not change the current setting.

↵ Do not change the current setting, and terminate the **SCT** command.

The debugger actually sets these two parameters when input is finished.

☞ 1 The cycle counter will not be incremented at the increment stop address specified by *parm2*.

☞ 2 If *parm2* is '.', then the cycle counter will also be incremented at the address at which a break is occured.

Execution Example

```
64153> DCT
     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

64153> SCT
     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

    START ---> 100

    END   ---> 140


64153> DCT
     START ADDRESS : 0100
     STOP  ADDRESS : 0140

64153> SCT /./.
64153> DCT
     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

64153> SCT
     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

    START ---> 200

    END   ---> -


    START ---> 210

    END   ---> 300
```

DCT, RCT

Input Format

DCT ↵

RCT ↵

Description

The **DCT** command displays the currently set cycle counter triggers (start/stop addresses). The display format is as follows.

```
CycleCounter  START : st-status   STOP : stp-status
START ADDRESS : st-status
STOP  ADDRESS : stp-status
```

The current start and stop addresses are displayed for *st-status* and *stp-status*. Their display contents are as follows.

Hexadecimal address — Indicates the currently set address.

FREE START — Indicates that cycle counter incrementing will start along with **G** command execution.

STOP FREE — Indicates that cycle counter incrementing will stop along with **G** command execution.

TRIGGER RESET — Indicates that the cycle counter trigger has not been set. If this setting is shown for *st-status*, then the cycle counter will not start.

The **RCT** command clears the currently set cycle counter triggers. After the **RCT** command is executed, the **DCT** and **SCT** commands will display TRIGGER RESET.

Execution Example

```
64153> DCT
     START ADDRESS : 0210
     STOP  ADDRESS : 0300

64153> SCT
     START ADDRESS : 0210
     STOP  ADDRESS : 0300

    START ---> 2AD

    END   ---> .


64153> DCT
     START ADDRESS : 02AD
     STOP  ADDRESS : STOP FREE

64153> SCT
     START ADDRESS : 02AD
     STOP  ADDRESS : STOP FREE

    START ---> 0A00

    END   ---> 0B00


64153> DCT
     START ADDRESS : 0A00
     STOP  ADDRESS : 0B00

64153> RCT
64153> DCT
     START ADDRESS : TRIGGER RESET
     STOP  ADDRESS : TRIGGER RESET

64153>
```

3.1.1.8.2 Monitoring Executed Code Memory

DIE

Input Format

DIE Δ *parm* [Δ *parm* ..... Δ *parm*] ↵

DIE Δ * ↵

parm : *address*
: [*address* Δ *address*]

Description

The **DIE** command displays the contents of the instruction executed bit memory.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of instruction executed bit memory to be displayed (☞ 1).

Display contents are one of the following, depending on input format.

address	Displays the contents on one address.
[*address* Δ *address*]	Displays the range enclosed in [ ].
*	Displays the entire area of instruction executed bit memory.

When multiple parameters are specified, each will be displayed even if their address areas overlap.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

Execution Example

```
64153> DIE 100 110
       LOC = 0100     1
       LOC = 0110     0
64153>
```

CIE

Input Format

CIE Δ *parm* [Δ *parm* ..... Δ *parm*] ↵

CIE Δ * [=*data*]

parm : *address* = *data*
: [*address* Δ *address*] = *data*

Description

The **CIE** command changes the contents of instruction executed bit memory.

The *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of instruction executed bit memory (☞ 1).

The *data* is the value of the change data. Its value can be '0' or '1.' Contents are changed in the order of the input parameters. The area changed is one of the following, depending on input format. If '*' is input and *data* is omitted, then the entire area will be set to '0.'

address	Changes the contents on one address.
[*address* Δ *address*]	Changes the range enclosed in [ ].
*	Changes the entire area of instruction executed bit memory.

When multiple parameters are specified, each will be changed even if their address areas overlap.

The **CIE** and **DIE** commands allow program execution flow to be examined.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

Execution Example

```
64153> DIE 100 [200 220]
       LOC = 0100    1

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0200    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0210    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0220    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153> DIE 234 345 56 989
       LOC = 0234    0
       LOC = 0345    0
       LOC = 0056    0
       LOC = 0989    0
64153> CIE *
64153> G 100,103
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Pass Counter Overflow Break *****
    Break PC =[0100], Next PC =[0101], TP=[0000]
64153> DIE [100 113]

                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0100    1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0110    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64153>
```

3.1.1.8.3 Counting Executed Addresses

DAP

Input Format

DAP [*mnemonic* ..... *mnemonic*] ↵

mnemonic : **C0**, **C1**, **C2**, **C3**

Execution Example

The **DAP** command displays the contents of the address pass counters as set by the **CAP** command. The EASE64158 provides four address pass counters: **C0, C1, C2, and C3**.

The display for each address pass counter is the number of times the instruction at the address of that pass counter (set by the **CAP** command) was executed during emulation execution.

If *mnemonic* is not input, then the contents of all address pass counters will be displayed.

Execution Example

```
64153> CAP C0=128 200
64153> CAP C3=482
64153> CAP C2=100 0
64153> DAP
        AP  :   ADDRESS    COUNT   OVERFLOW
      ------+---------------------------------
        C0  :    0128        200      0
        C1  :    0200          0      0
        C2  :    0100          0      0
        C3  :    0482          0      0
```

CAP

Input Format

CAP *mnemonic* [= *address*] [Δ *count*]

mnemonic : **C0**, **C1**, **C2**, **C3**

Description

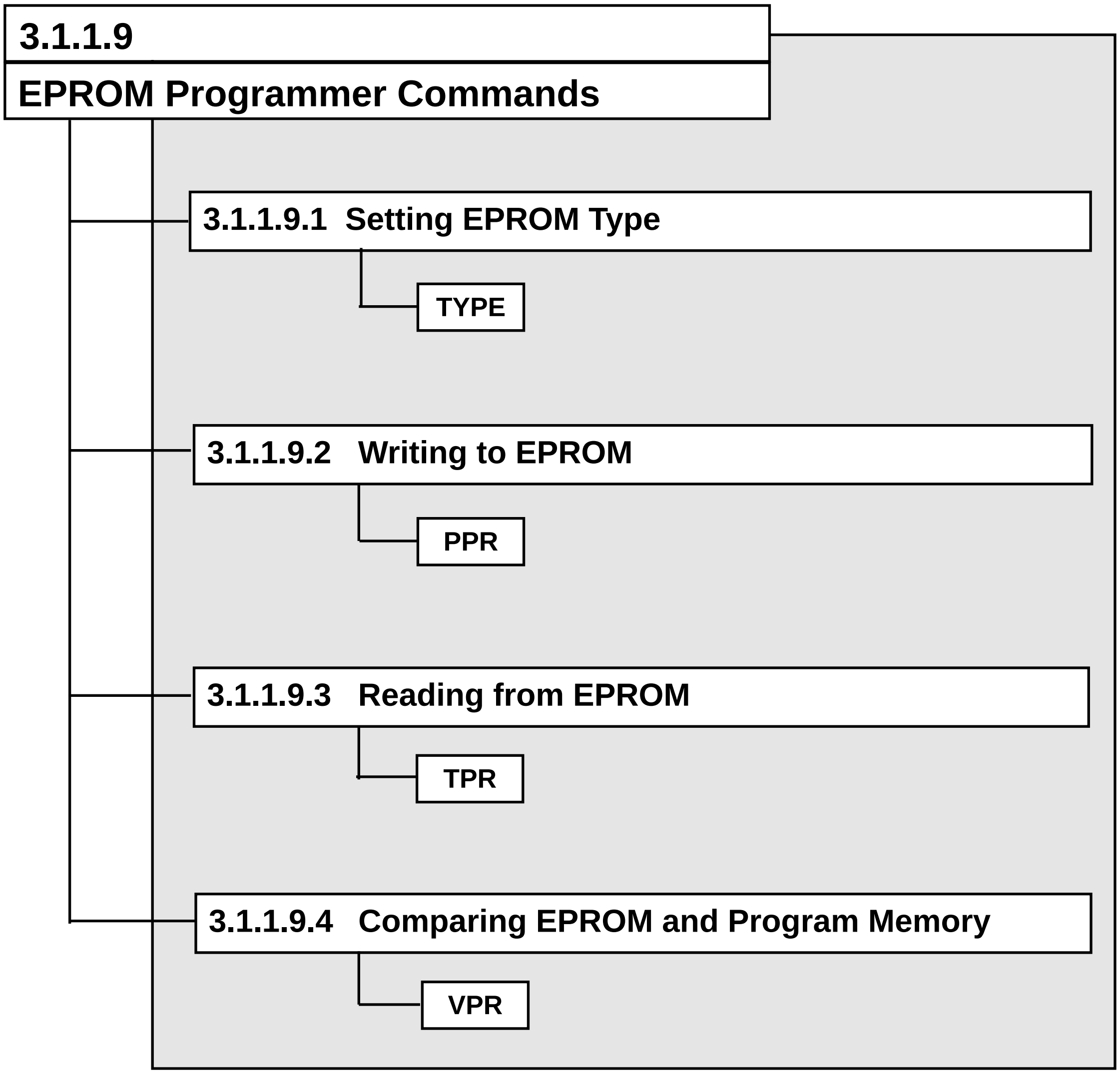
The **CAP** command is provided for monitoring how often an instruction at some address is executed. The EASE64158 provides four address pass counters: **C0**, **C1**, **C2**, and **C3**.

The *mnemonic* specifies one of the four address pass counters, and the *address* specifies the associated address for incrementing. If *address* is omitted, then the address set with the previous **CAP** command will be used. The *count* is in the range 0–65535. If *count* is omitted, then it will be set to 0.

The **C0** address pass counter has an overflow break function. If the **SBC** command has been set to AP (address pass counter overflow breaks), then emulation execution will break and terminate at the point the counter value exceeds 65535.

Execution Example

```
64153> SBC AP
64153> CAP C1=200
64153> CAP C0=100 65535
64153> DAP
        AP  :   ADDRESS    COUNT   OVERFLOW
      ------+------------------------------
        C0  :    0100      65535      0
        C1  :    0200          0      0
        C2  :    0100          0      0
        C3  :    0482          0      0
```


3.1.1.9
EPROM Programmer Commands
3.1.1.9.1 Setting EPROM Type
TYPE
3.1.1.9.2 Writing to EPROM
PPR
3.1.1.9.3 Reading from EPROM
TPR
3.1.1.9.4 Comparing EPROM and Program Memory
VPR

3.1.1.9.1 Setting EPROM Type

TYPE

Input Format

TYPE ∆ *mnemonic* ↵

The **TYPE** command specifies the type of EPROM that will be used in the EPROM programmer. The *mnemonic* indicates the EPROM type.

Usable EPROM types can be classified into the following two broad categories.

1. Intel products and other EPROMs that are written at high speed with the Intelligent Programming method.

2. Fujitsu products and other EPROMs that are written at high speed with the Fujitsu Programming method.

These two categories are distinguished by adding a prefix before the EPROM name when entering *mnemonic*. Prefix an 'I' for the first category, and 'F' for the second.

The following can be input for *mnemonic*.

EPROM Type	mnemonic	
	Intel	Fujitsu
2764	I2764	F2764
27C64	I27C64	F27C64
2764A	I2764A	–
27128	I27128	F27128

continued on next page

EPROM Type	mnemonic	
	Intel	Fujitsu
27C128	–	F27C128
27128A	I27128A	–
27C128A	I27C128A	–
27256	I27256	F27256
27C256	I27C256	F27C256
27C256A	–	F27C256A
27512	I27512	–
27C512	–	F27C512

Products with an entry marked by "—" do not exist, so no *mnemonic* is provided.

If *mnemonic* is omitted, then the currently set EPROM type will be displayed. The setting will be "I27512" after power is turned on.

Execution Example

```
64153> TYPE

   EPROM TYPE----->I27512
64153> TYPE F27C256A

64153> TYPE

   EPROM TYPE----->F27C256A
64153>
```

3.1.1.9.2 Writing to EPROM

PPR

Input Format

PPR ∆ [*address* ∆ *address*] [∆ *eprom-address*] ↵

PPR ∆ * ↵

Description

The **PPR** command writes the contents of the specified code memory area to the specified EPROM address.

An *address* is an expression that evaluates within code memory's maximum address range. It indicates an address of code memory (☞ 1). The [*address* ∆ *address*] specifies the range of code memory to be written. If '*' is input, then a range of code memory that corresponds to the EPROM type will be set (☞ 2).

The *eprom-address* is the EPROM's starting address for writing. If this address is omitted, then writing will start from EPROM address 0.

Input continues until a carriage return is entered. Then the following message will be output.

```
EPROM TYPE ---> type
PROGRAMMING VOLTAGE = voltage
PROGRAMMING METHOD = method
START PROGRAMMING [Y/N] ---> _
```

Here *type* indicates the currently set EPROM type. The *voltage* is the write voltage, while the *method* is the write method.

If the EPROM type displayed is the same as the EPROM type that the user wants to write, then enter "**Y**↵" at the underscore. If they are different, then input "**N**↵" and set the EPROM type again with the **TYPE** command.

When "**Y**↵" is input, the EASE64158 "RUN" LED will light, and the data write will start. If the data write completes normally, then the LED will go off, the **PPR** command will terminate, and the emulator will wait for another command input.

☞ 1 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

! Note that the test data area in the program area (code memory) of each MSM64153 family microcontroller cannot be used by the emulator.

☞ 2 The code memory range that will be written when '*" is input will be as follows.

EPROM Type	Address Range	EPROM Type	Address Range
2764	0 ~ 1FFFH	27256	0 ~ 7FFFH
27128	0 ~ 3FFFH	27512	0 ~ 7FFFH

However, the maximum write address will evaluate within the maximum address range of the code memory.

Execution Example

```
64153> TYPE F27C256A

64153> PPR [0 20]

   EPROM TYPE----->F27C256A
   PROGRAMMING VOLTAGE=12.5 V
   FUJITSU QUICK PROGRAMMING
   START PROGRAMMING [Y/N]----->Y
64153> PPR [100 135] 100

   EPROM TYPE----->F27C256A
   PROGRAMMING VOLTAGE=12.5 V
   FUJITSU QUICK PROGRAMMING
   START PROGRAMMING [Y/N]----->Y
64153> PPR [300 400] 500

   EPROM TYPE----->F27C256A
   PROGRAMMING VOLTAGE=12.5 V
   FUJITSU QUICK PROGRAMMING
   START PROGRAMMING [Y/N]----->N
64153>
```

3.1.1.9.3 Reading from EPROM

TPR

Input Format

TPR ∆ [*address* ∆ *address*] [∆ *CM-address*] ↵

TPR ∆ * ↵

Description

The **TPR** command reads the EPROM contents in the specified range and transfers them to the specified code memory area.

Each *address* is an EPROM address. The [*address* ∆ *address*] specifies the EPROM range to be read. If '*" is input, then the entire EPROM area corresponding to the EPROM type will be set.

The *CM-address* is the code memory starting address for transferring. If this *address* is omitted, then the transfer will start from code memory address 0.

Input continues until a carriage return is entered. Then the following message will be output.

```
EPROM TYPE ---> type
START READING [Y/N] ---> _
```

☞ 1 The code memory range that will be read when '*" is input will be as follows.

EPROM Type	Address Range	EPROM Type	Address Range
2764	0 ~ 1FFFH	27256	0 ~ 7FFFH
27128	0 ~ 3FFFH	27512	0 ~ 7FFFH

However, the maximum read address will evaluate within the maximum address range of the code memory.

☞ 2 Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

! Note that the emulator handles the test data area in the program area (code memory) of the MSM64153 family microcontrollers as an unusable area.

Here *type* indicates the currently set EPROM type.

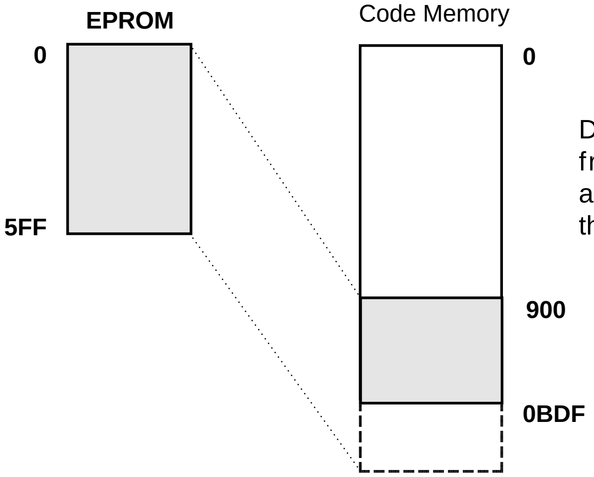
If the EPROM *type* displayed is the same as the EPROM type that the user wants to read, then enter "Y↵" at the underscore. If they are different, then input "N↵" and set the EPROM type again with the **TYPE** command.

When "Y↵" is input, the EASE64158 "RUN" LED will light, and the data transfer will start. If the data transfer completes normally, then the LED will go off, the TPR command will terminate, and the emulator will wait for another command input.

As shown in the following example, if the range of EPROM read, as specified by [*address* ∆ *address*], exceeds the maximum address of code memory, then the transfer will terminate at that point.

Example

TPR [0 5FF] 900



Data will be transferred from address 900 to address 0BDF, and then the transfer will terminate.

Execution Example

```
64153> TPR [100 132]

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->Y
64153> TPR [200 209] 100

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->Y
64153> TPR [323 523] 512

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->N
64153>
```

3.1.1.9.4 Comparing EPROM and Program Memory

VPR

Input Format

VPR Δ [*address* Δ *address*] [Δ *eprom-address*] ↵

VPR Δ * ↵

Description

The **VPR** command compares the contents of the specified range of code memory with the contents of the EPROM starting at the specified address, and displays any differences on the console.

An *address* is an address of code memory. The [*address* Δ *address*] specifies the range of code memory to be compared (☞ 1). If '*' is input, then a range of code memory that corresponds to the EPROM type will be set (☞ 2).

The *eprom-address* is the EPROM's starting address for comparison. If this *address* is omitted, then comparison will start from EPROM address 0.

Input continues until a carriage return is entered. Then the following message will be output.

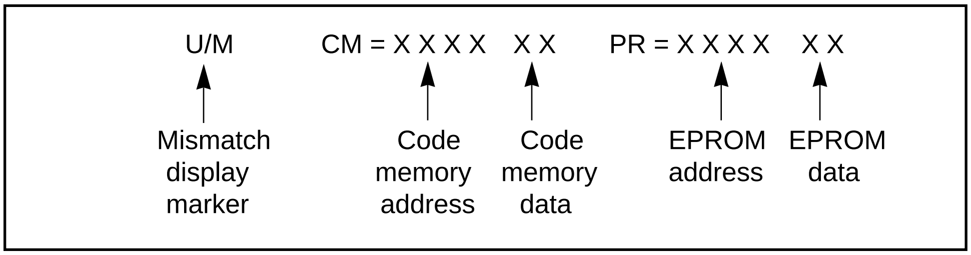
```
EPROM TYPE ---> type
START READING [Y/N] ---> _
```

Here *type* indicates the currently set EPROM type.

If the EPROM type displayed is the same as the EPROM type that the user wants to compare, then enter "Y↵" at the underscore. If they are different, then input "N↵" and set the EPROM type again with the **TYPE** command.

When "Y↵" is input, the EASE64158 "RUN" LED will light, and the data comparison will start. If the data comparison completes normally, then the LED will go off, the **VPR** command will terminate, and the emulator will wait for another command input.

Whenever a comparison error occurs, the information will be displayed on the console in the following format (☞ 3).



☞ **1** Refer to each MSM64153 family microcontroller's User's Manual, regarding address ranges.

☞ 2

The code memory range that will be compared when '*" is input will be as follows.

EPROM Type	Address Range	EPROM Type	Address Range
2764	0 ~ 1FFFH	27256	0 ~ 7FFFH
27128	0 ~ 3FFFH	27512	0 ~ 7FFFH

However, the maximum comparison address will evaluate within the maximum address range of the code memory.

☞ 3

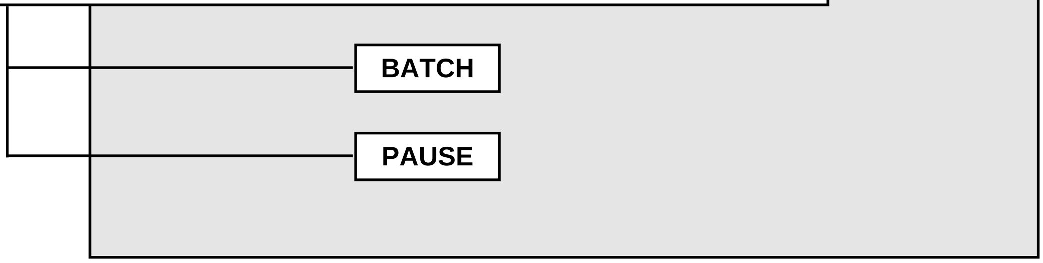
If the number of comparison error exceeds 100, the emulator automatically ends verifying to return to the prompt.

Execution Example

```
64153> CCM *=0FFH

64153> VPR [20 54]

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->Y
64153> VPR [733 766] 100

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->Y
64153>

64153>

64153> CCM 0

   LOC=0000  FF----->2   New
   LOC=0001  FF----->4   New
   LOC=0002  FF----->6   New
   LOC=0003  FF----->
64153>

64153> VPR [0 19]

   EPROM TYPE----->F27C256A
   START READING [Y/N]----->Y
  U/M   CM=0000 2   PR=0000 FF
  U/M   CM=0001 4   PR=0001 FF
  U/M   CM=0002 6   PR=0002 FF
64153>
```

3.1.1.10

Commands for Automatic Command Execution



BATCH

Input Format

BATCH Δ *fname* ↵

fname : [*Pathname*] *Filename* [*Extension*]

Description

The **BATCH** command automatically executes the file contents that is specified by *fname* as emulator commands.

The input file name can have a path specification. If the path is omitted, then the file will be taken in the current directory.

If the file extension is omitted, then a default extension (.CMD) will be appended. To specify a file without an extension, append a period '.' after the filename.

In addition to emulator commands, the batch file can also contain assembler mnemonics input within the **ASM** command.

Automatic execution is performed until the end of the file. If the **ESC** key is pressed during execution, then automatic execution will be suspended.

❗ Only one batch file can be open. Therefore, even if a **BATCH** command is included within a batch file, it will be ignored.

❗ If the reset key is pressed during **BATCH** command execution, the **BATCH** command will be terminated and the batch file will be closed.

Execution Example

```
64153> BATCH BAT.TP

     Batchfile : BAT.TP opened
64153> ASM 100
line  Segment  Location  Source Statement
   1  Code     0100      nop
   2  Code     0101      lai 0
   3  Code     0102      lhi 1
   4  Code     0104      ina
   5  Code     0105      nop
   6  Code     0106      end

64153> D
     A : 0    B : 0    H : 0    L : 0    X : 0
     Y : 0    PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
     BSR1 : 0    C : 0
64153>
     Batchfile: BAT.TP closed
64153>
```

PAUSE

Input Format

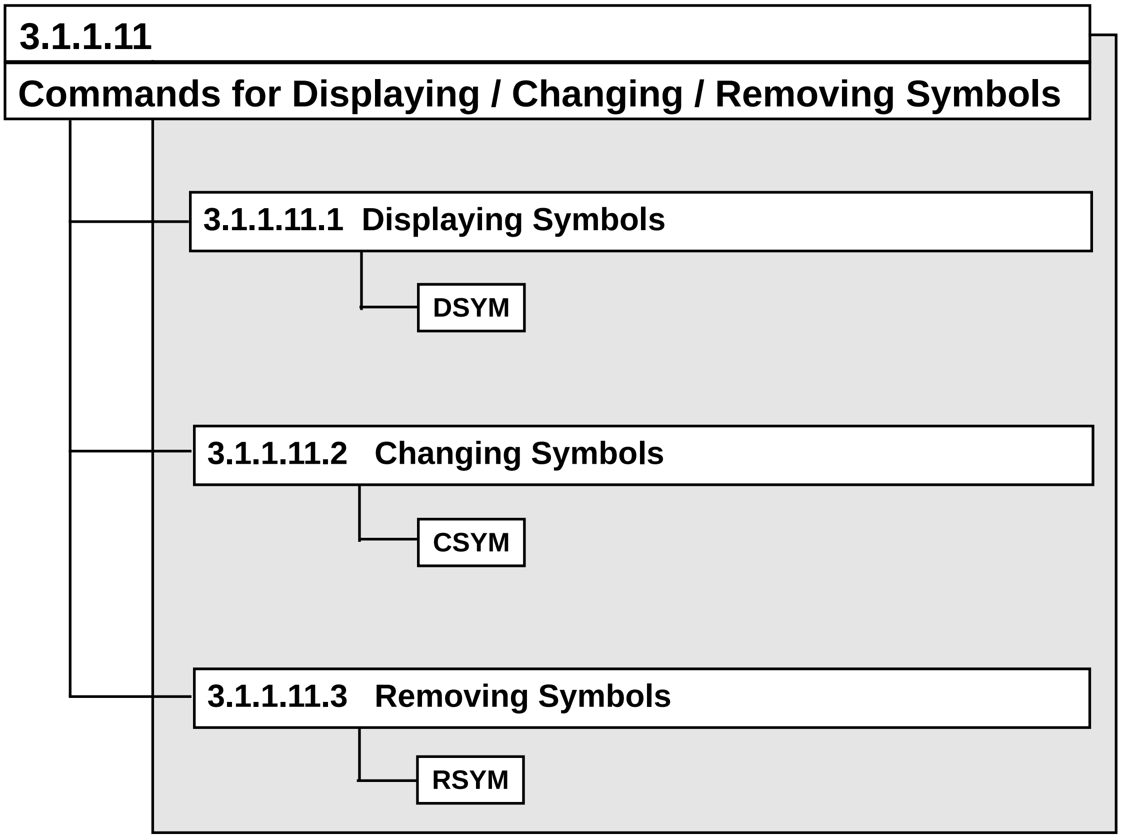
PAUSE ↲

Description

The **PAUSE** command waits for keyboard input when executed. By placing a **PAUSE** command in a batch file, automatic command execution can be temporarily suspended. The input wait state will be released upon input from the keyboard, or if the emulator reset switch is pressed.

Execution Example

```
64153> PAUSE

   *** Hit Any Key ***
64153>
```



3.1.1.11

Commands for Displaying / Changing / Removing Symbols

3.1.1.11.1 Displaying Symbols

DSYM

3.1.1.11.2 Changing Symbols

CSYM

3.1.1.11.3 Removing Symbols

RSYM

3.1.1.11.1 Displaying Symbols

DSYM

Input Format

DSYM Δ *string* [Δ *string* ..... Δ *string*] ↵

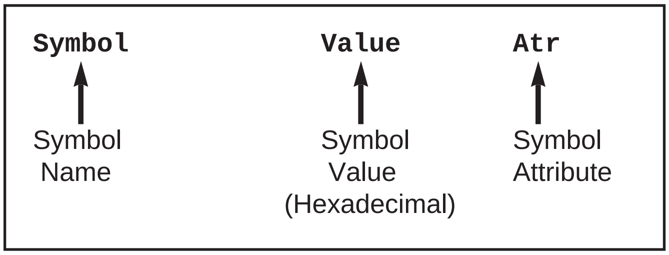
DSYM Δ * ↵

Description

The **DSYM** command displays information about user symbols loaded with the **LOD** command (with **/S** option) or defined by labels or assembler directives (**EQU, SET, CODE, DATA**) within the **ASM** command.

A symbol name is entered for *string*. If only a '*' is input, then all currently registered user symbols will be displayed. Input symbol names can use wild cards like '*' and '?' in the same manner as MS-DOS and PC-DOS.

The displayed information will be as follows.



The "**Atr**" will be one of the following.

CODE	Code address attribute
DATA	Data address attribute
NUMBER	Number attribute

Execution Example

```
64153> DSYM *
 Symbol                        Value     Atr
 AAA                           0103     CODE
 BBB                           0103     CODE
 ABC                           0201     CODE
 CCC                           0200     CODE
 ACD                           0201     CODE
 BCD                           0204     CODE
 START                         0100     CODE
64153> DSYM A*
 Symbol                        Value     Atr
 AAA                           0103     CODE
 ABC                           0201     CODE
 ACD                           0201     CODE
64153> DSYM B??
 Symbol                        Value     Atr
 BBB                           0103     CODE
 BCD                           0204     CODE
64153> DSYM START
 Symbol                        Value     Atr
 START                         0100     CODE
```

3.1.1.11.2 Changing Symbols

CSYM

Input Format

CSYM Δ *parm* [, *parm* ..... , *parm*] ↵

parm : *string* [= *data*]

Description

The **CSYM** command changes the values of user symbols loaded with the **LOD** command (with **/S** option) or defined by labels or assembler directives (**EQU, SET, CODE, DATA**) within the **ASM** command (☞ 1).

A symbol name is entered for *string*. The data to be changed is entered for *data*.

If *data* is omitted, then the it will be entered for each input symbol as follows.

```
     old [old-data]------► _
```

The operator inputs the new data at the underscore. The *old-data* will be the symbol's currently set value.

☞ 1 The symbol attribute (Atr) cannot be changed.

In addition to change data, the following input is also valid while the emulator is waiting for input.

"Δ↵" Without changing the data, proceed to input data for the next symbol. If there is no next symbol, then input terminates.

"↵" Terminates input.

A symbol name can contain wild cards, "*," or "?", as in MS-DOS or PC-DOS.

Execution Example

```
64153> DSYM *
 Symbol                        Value     Atr
 AAA                           0103     CODE
 BBB                           0103     CODE
 ABC                           0201     CODE
 CCC                           0200     CODE
 ACD                           0201     CODE
 BCD                           0204     CODE
 START                         0100     CODE
64153> CSYM B*
 BBB  old[0103] ----> 0AD  New
 BCD  old[0204] ----> 7  New
64153> DSYM *
 Symbol                        Value     Atr
 AAA                           0103     CODE
 BBB                           00AD     CODE
 ABC                           0201     CODE
 CCC                           0200     CODE
 ACD                           0201     CODE
 BCD                           0007     CODE
 START                         0100     CODE
64153> CSYM ??C
 ABC  old[0201] ----> 90A  New
 CCC  old[0200] ----> CSYM ST???        ** Error 102: Illegal data input.
 CCC  old[0200] ----> 0B00 New
64153> DSYM *
 Symbol                        Value     Atr
 AAA                           0103     CODE
 BBB                           00AD     CODE
 ABC                           090A     CODE
 CCC                           0B00     CODE
 ACD                           0201     CODE
 BCD                           0007     CODE
 START                         0100     CODE
```

3.1.1.11.3 Removing Symbols

RSYM

Input Format

RSYM Δ *string* [Δ *string* ..... Δ *string*] ↵

RSYM Δ * ↵

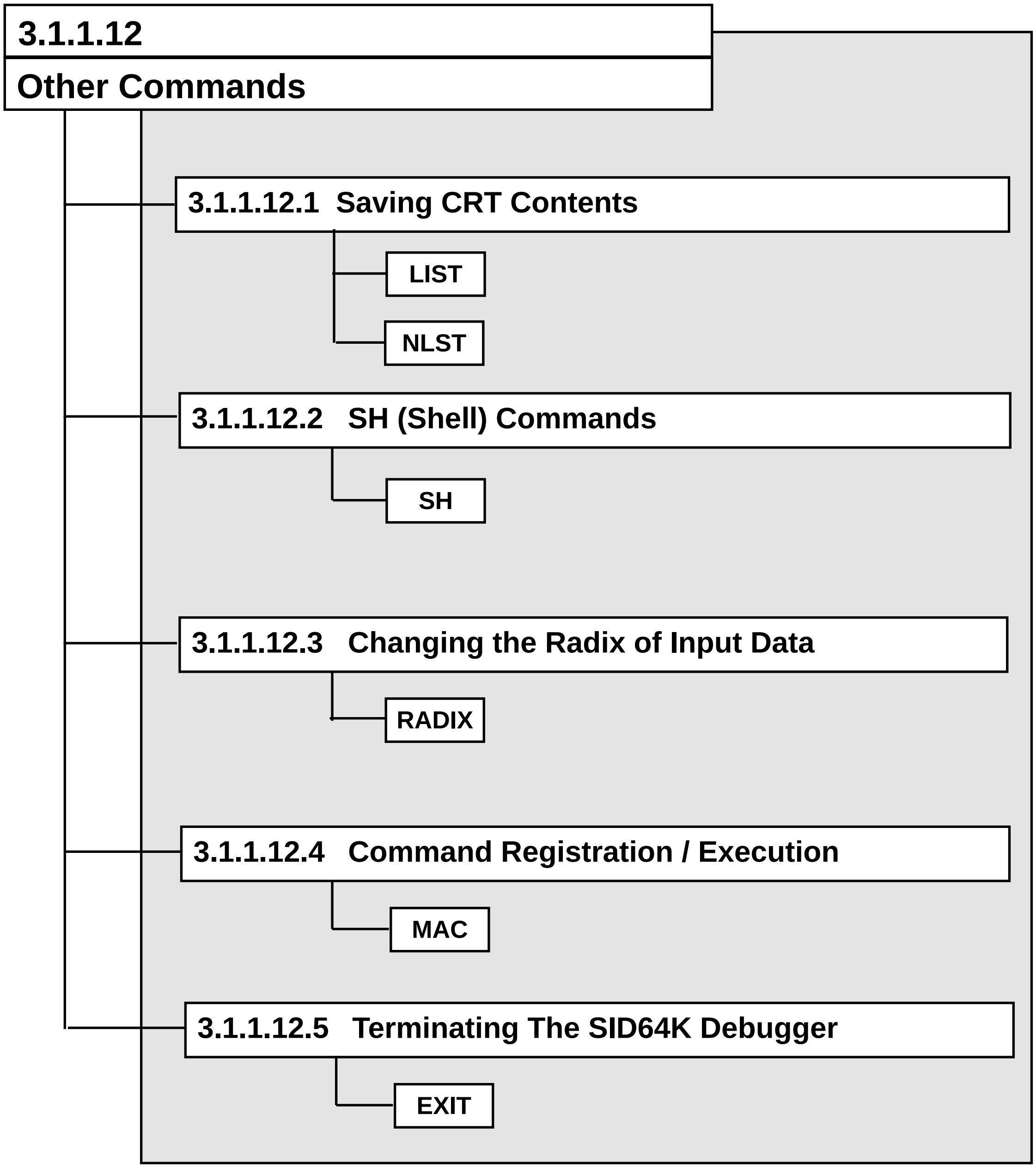
Description

The **RSYM** command removes user symbols loaded with the **LOD** command (with **/S** option) or defined by labels or assembler directives (**EQU, SET, CODE, DATA**) within the **ASM** command.

A symbol name is entered for *string*. If only a '*' is input, then all currently registered user symbols will be removed. Input symbol names can use wild cards like '*' and '?' in the same manner as MS-DOS and PC-DOS.

Execution Example

```
64153> DSYM *
 Symbol                        Value     Atr
 AAA                           0103     CODE
 BBB                           00AD     CODE
 ABC                           090A     CODE
 CCC                           0B00     CODE
 ACD                           0201     CODE
 BCD                           0007     CODE
 START                         0100     CODE
64153> RSYM AAA
64153> DSYM *
 Symbol                        Value     Atr
 BBB                           00AD     CODE
 ABC                           090A     CODE
 CCC                           0B00     CODE
 ACD                           0201     CODE
 BCD                           0007     CODE
 START                         0100     CODE
64153> RSYM B??
64153> DSYM *
 Symbol                        Value     Atr
 ABC                           090A     CODE
 CCC                           0B00     CODE
 ACD                           0201     CODE
 START                         0100     CODE
64153> DSYM BBB
 Symbol                        Value     Atr
 BBB
       ** Error 092: Symbol not found.
```


3.1.1.12
Other Commands
3.1.1.12.1 Saving CRT Contents
LIST
NLST
3.1.1.12.2 SH (Shell) Commands
SH
3.1.1.12.3 Changing the Radix of Input Data
RADIX
3.1.1.12.4 Command Registration / Execution
MAC
3.1.1.12.5 Terminating The SID64K Debugger
EXIT

3.1.1.12.1 Saving CRT Contents

LIST

Input Format

LIST ∆ *fname* ↵

fname : [*Pathname*] *Filename* [*Extension*]

Description

The **LIST** command stores the contents displayed to the console in the specified file.

The input file name can have a path specification. If the path is omitted, then the file will be taken in the current directory. If a file of the same name exists in the specified directory, then that file will be deleted and a new file will be created. If the specified file is write-protected, then the **LIST** command will be forcibly terminated.

If the file extension is omitted, then a default extension (.LST) will be appended.

While a file is being created by a **LIST** command, another **LIST** command cannot be used (only one list file can be open).

The **LIST** command becomes valid immediately after it has been input. When any of the following occurs, the **LIST** command becomes invalid and the list file is closed.

- An **NLST** command is input.
- The SID64K symbolic debugger terminates.
- The EASE64158 base unit's reset switch is pressed.

Execution Example

```
64153> LIST SAMP.LST
64153> D

    A    : 0    B  : 0    H   : 0    L   : 0    X    : 0
    Y    : 0    PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
    BSR1 : 0    C  : 0
64153> D PC

    PC : 0000
64153> NLST
```

NLST

Input Format

NLST ↵

Description

The **NLST** command terminates a previous **LIST** command. It will close the list file opened by the **LIST** command.

Contents are stored in the list file until the **NLST** command.

Execution Example

```
64153> LIST SAMP.LST
64153> D

    A : 0    B : 0    H : 0    L : 0    X : 0
    Y : 0    PC : 0000 BCF : 0    BEF : 0    BSR0 : 0
    BSR1 : 0    C : 0
64153> D PC

    PC : 0000
64153> NLST
```

3.1.1.12.2 SH (Shell) Command

SH

Input Format

SH ↵

Description

The **SH** command invokes the DOS shell COMMAND.COM (command interpreter) as a child process of the debugger. Thus, even if any environment variables (PATH, COMSPEC, etc.) are set after the **SH** command invokes COMMAND.COM, the settings will be lost when control is returned to the debugger by entering **EXIT**. Accordingly, the path of the invoked COMMAND.COM cannot be changed.

The procedure when the **SH** command invokes the child process (COMMAND.COM) is explained below.

(1) The current directory is searched for COMMAND.COM, and if found it is invoked. If not found, then the search moves to (2).

(2) The directories set in the **PATH** environment variable are searched in order.

For example,

```
PATH = a:\,a:\bin,a:\uty,a:\SID64K
```

The directories are searched in the order "a:\", "a:\bin", "a:\uty", and "a:\SID64K." The first COMMAND.COM found will be executed. If not found, then the search moves to (3).

(3) The child process is invoked using the path name set in the COMSPEC environment variable. Assuming COMMAND.COM exists in the root directory of the A: drive, set the following before using the debugger.

```
COMSPEC = A:\COMMAND.COM   (☞ 1)
```

☞ 1

When DOS terminates a child process (the debugger), it reloads COMMAND.COM referring to the COMSPEC environment variable. If the COMSPEC environment variable is set to something other than COMMAND.COM, then DOS will attempt to reload COMMAND.COM but will not be able to. The only way to release this state is to reset or turn off the PC, so it is recommended that you specify the full path name of the DOS shell (command interpreter) in the COMSPEC environment variable (the path name is specified by "path+filename+extension" and is distinct from the PATH environment variable).

In order to realize the shell function, the free area of the system being used must have sufficient space for invoked programs. The resident portion of SID64K.EXE consumes about 220K bytes. In addition, the symbol table consumes the following number of bytes.

[total characters of all registered symbols] + [number of registered symbols] x [33 bytes]

Thus, for a program to be invoked after the **SH** command has been executed, it must have fewer bytes than the original free area less the above byte count and less the size of COMMAND.COM.

Execution Example

```
64153> SH
Command version 3.10

A>CD
\USR\TEST

A>EXIT

64153> NLST
```

3.1.1.12.3 Changing the Radix of Input Data

RADIX

Input Format

RADIX Δ *mnemonic* ↵

Description

The **RADIX** command changes the radix for values input on SID64K debugger command lines. The *mnemonic* can be one of the following.

D	Input data will be recognized as radix 10 (decimal).
H	Input data will be recognized as radix 16 (hexadecimal).
B	Input data will be recognized as radix 2 (binary).
O	Input data will be recognized as radix 8 (octal).

The following values will always be recognized as decimal when input, regardless of the current radix setting.

- Delay count values
- Cycle count values
- Pass count values
- Trace pointer values
- Step counts

When EASE64158 power is turned on, the radix will be set to H (hexadecimal) by default.

! Values input in source statements of the **ASM** command are not affected by the **RADIX** command setting.

! When a hexadecimal number is input and it begins with A–F, it needs to be prefixed with a '0' (zero).

Execution Example

```
64153> ; RADIX
64153> ;
64153>
64153> RADIX D
64153> DCM 10
       LOC = 000A    00
64153> RADIX H
64153> DCM 10
       LOC = 0010    00
64153> RADIX B
64153> DCM 10
       LOC = 0002    00
64153> RADIX O
64153> DCM 10
       LOC = 0008    00
64153>
```

3.1.1.12.4 Registering/Executing Commands

MAC

Input Format

MAC [Δ [~] *macro_command*] ↵

Description

The **MAC** command allows many consecutive SID64K commands (except for **MAC**) to be replaced as a single macro command and automatically executed. When the same sequence of commands is often used during debug operations, defining it as a macro with the **MAC** command can improve operating efficiency.

Up to five macro commands can be registered.

- New registration of a macro command name and its command lines

Input the new command name for *macro_command*. Up to 8 characters can be input.

```
    64153>  MAC  DISP ↵
```

If this name is not the same as an SID64K command, then the emulator will output the following message and wait for input of one command line to be registered.

```
            1. ------►
```

Up to 10 command lines can be registered. Up to 61 characters can be input on one line. When a carriage return is input after a line, the emulator will wait for input for the next line. To stop registration, input "↵."

After input of the tenth line ends, an "↵" will be added automatically and registration will terminate.

- Verifying/adding/removing a previously registered macro command's command lines

Input the **MAC** command, followed by the registered command name and a carriage return. Then the registered command lines will be displayed as follows.

```
    64153> MAC DISP ↵
          1. -----> DCM [0 100]
          2. -----> CCM [10 20] = 5 50 = 0A
          3. -----> DCM [0 100]

          ADD(+) or DEL(-) ==>
```

To simply verify the contents, input a carriage return and the "64153>" prompt will return. To remove a command line, enter "- *number* ↵." The *number* is the number of the command line to be removed.

To add a command line, enter "+ ↵." If fewer than 10 lines are registered, then the emulator will wait for command line input. To add a command line in between already registered command lines, input "*+ number* ↵." The number is the sequence number to be given to the command line. The sequence numbers of all command lines after the added one will be incremented automatically.

When you are done registering, input "↵."

- Executing a registered macro command

To execute a registered macro command, input the command name followed by a carriage return.

```
64153> DISP ↵
```

- Verifying/removing a registered macro command

To verify the registered macro commands, input the following.

```
64153>  MAC ↵
  1.  DISP
```

To remove a registered macro command, input the following.

```
64153> MAC ~DISP ↵
```

Execution Example

```
64153> MAC DISP

       1. -----> DCM [0 23]
       2. -----> DCM 90
       3. -----> CCM 30=1
       4. -----> D

       ADD(+) OR DEL(-) ==> DCM [0 23]


64153> DISP
 MAC > DCM [0 23]

             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
            ------------------------------------------------
 LOC = 0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 LOC = 0010  00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00

 MAC > DCM 90
       LOC = 0090 00

 MAC > CCM 30=1

 MAC > D
    A : 0    B : 0    H : 0    L : 0    X : 0
    Y : 0    PC : 0044 BCF : 0    BEF : 0    BSR0 : 0
    BSR1 : 0    C : 0
```

Execution Example

```
64153> MAC DISP

       1. -----> DCM [0 23]
       2. -----> DCM 90
       3. -----> CCM 30=1
       4. -----> D

       ADD(+) OR DEL(-) ==> -5


64153> MAC TP

       1. -----> D
       2. -----> G 100,103
       3. -----> DTM *

       ADD(+) or DEL (-) ==> D

64153> TP
 MAC > D
    A : 0    B : 0    H : 0    L : 0    X : 0
    Y : 0    PC : 0044 BCF : 0    BEF : 0    BSR0 : 0
    BSR1 : 0    C : 0

 MAC > G 100,103
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0043], Next PC =[0044], TP=[0004]

 MAC > DTM *

LOC        MNEMONIC            SP P3 P7D A B X Y TP
LOC=0040   NOP                 FF  0  0  0 0 0 0 0000
LOC=0041   NOP                 ..  .  .  . . . . 0001
LOC=0042   NOP                 ..  .  .  . . . . 0002
LOC=0043   NOP                 ..  .  .  . . . . 0003

64153>
```

3.1.1.12.5 Terminating the SID64K Debugger

EXIT

Input Format

EXIT ↵

Description

The **EXIT** command terminates the SID64K debugger.

If a list file has been opened by the **LIST** command, then it will be closed before the debugger terminates.

Execution Example

```
64153> EXIT
A>
```

! When the SID64K symbolic debugger is restarted without turning off its power after it has been terminated with the **EXIT** command, then the message shown in (9) of Section 2.6, "Starting the EASE64158 Emulator," will be output. Afterwards, press the EASE64158 reset switch to start.

Chapter 4

Debugging Notes

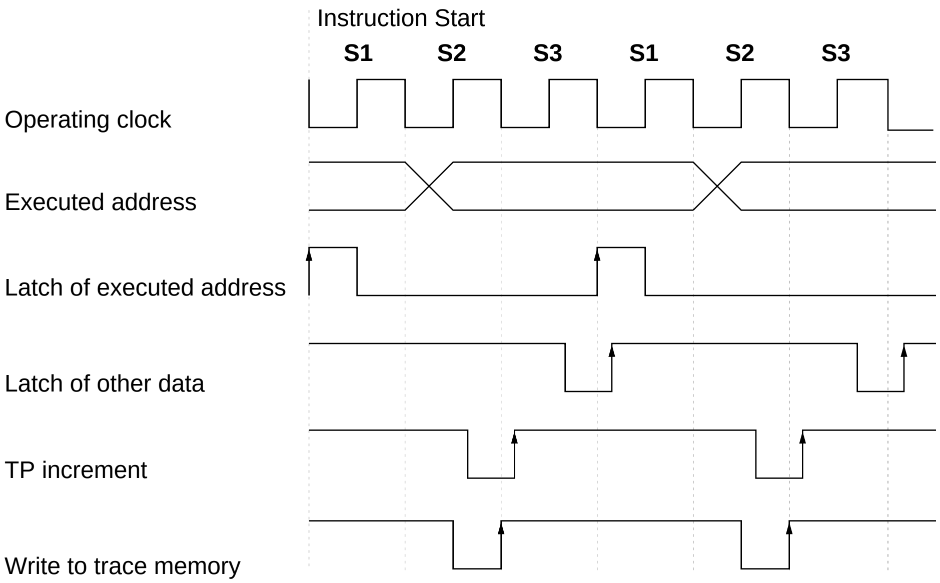
This chapter provides some notes about debugging with the EASE64158 system.

4-1. Debugging Notes

4-1-1. Tracing

The timing for writes to EASE64158 trace memory is as follows.

Executed address	M1S1 & SYSCLK/
Trace pointer (TP) count	$(\overline{\text{M1S2 \& SYSCLK}})$ + gate delay
Other (AR, BR, SP, etc.)	$(\overline{\text{S3 \& SYSCLK}})$ + gate delay



As can be seen from this timing, the trace data displayed when a **DTM** command is executed will lag the changes in trace data by one instruction except for SKIP flag and INT flag.

4-1-2. Resets

The USER•RESET pin (pin 43) of the user cables is effective only during POD64158 operation under realtime emulation from the **G** command, and during POD64158 standalone operation (POD mode). However, during realtime emulation this is further restricted to when the **URST** command setting is on.

Refer to Figure 4-1 to see how the USER•RESET pin is used.

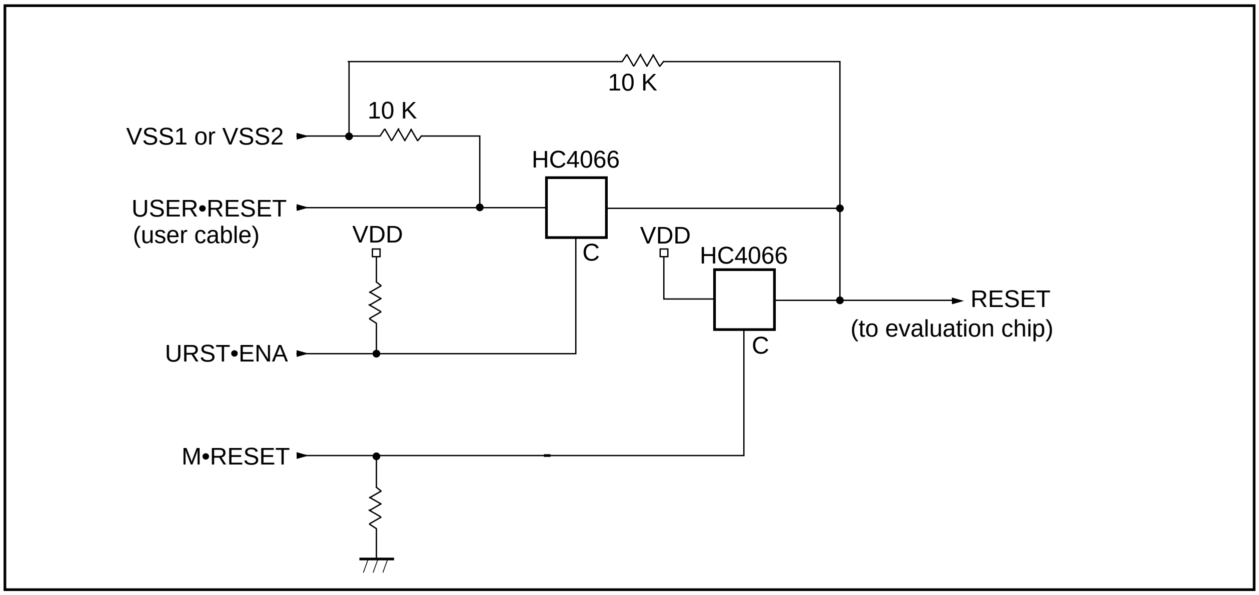


Figure 4-1. Reset Circuit

- USER•RESET indicates the USER•RESET pin (pin 43) of the user cable.
- URST.ENA is a signal that becomes “H” when the **URST ON** command is executed.
- M. RESET is the reset signal output from the EASE64158 internal controller.
- RESET is connected to the reset pin of the MSM64E153 evaluation chip.

4-1-3. User Cables

VDD is not supplied from the user cables connected to the POD64158 and the user application system. When connected to the user application system during debugging, supply VDD to the user application system from a separate power supply.

4-1-4. Cycle Counter Overflow Breaks

When program execution breaks due to a cycle counter overflow break, the break will occur after execution of the next instruction after the instruction at which the cycle counter overflowed. Accordingly, the cycle counter value at the break will not be 0, but will be the cycle count value of the instruction that generated the break.

4-1-5. EPROM Programmer

Always remove any EPROM in the EASE-LP2 EPROM programmer when the EASE64158 is started. Mount EPROMs when the emulator is waiting for command input.

SEE Appendix 9, “Mounting EASE-LP2 EPROMs.”

4-1-6. DASM Command

NOP	and	AIS 0	(both codes 0H)
INA	and	AIS 1	(both codes 1H)
LAM	and	LAMM 0	(both codes 70H)
XAM	and	XAMM 0	(both codes 74H)

The instructions in each of the above instruction pairs have identical instruction codes. When the SID64K disassembles using the **DASM** command, the mnemonics on the left side will be displayed.

4-1-7. Break

(1) When any break condition is satisfied during skip operation (stack instruction, etc.), the break will occur after the completion of the ongoing skip operation (similar for step command). The break condition for this case will be "No breakstatus." (In trace display, SKIP flag will be "1" during skip operation.)

Example:

```
.
.
.
LAI     1
LAI     2
LAI     3
LAI     4
LMAD    7C
.
.
.
```

If the sample program shown at the left is executed continuously from **LAI 1** with a break point bit break specified at the **LAI 3** instruction, the break will not occur at skip execution of **LAI 3**, but just before **LMAD 7C** instruction instead.

In step operation, execution of **LAI 1** makes skip operation continue to **LAI 4** one by one, and **LMAD 7C** will also be executed.

(2) When any break condition is satisfied during interrupt transferring cycle, the break will occur after the completion of interrupt transferring cycle execution. The interrupt vector address will be the break PC for this case, and "No breakstatus" will be the break condition.

An instruction whose INT flag is "1" on trace display is actually not executed (a dummy trace indicates that an interrupt transferring cycle is executed instead).

In step command, instruction itself will be executed as well as the interrupt transferring cycle execution, and the break will occur after thier completion.

(3) When an interrupt is occur at the same time as setting master interrupt enable (MI) flag to "1," the MI flag of trace display will not be "1."

4-1-8. Probe Cable

Probe cable can be used only when the MSM64E153 emulation chip operating voltage is 3.0 V. It cannot be used when the voltage is 1.5 V.

4-1-9. Operating Clock

However the MSM64153 family microcontrollers MSM64155 and MSM64158 can select the CR oscillation circuit as the clock generator by using mask option, the EASE64158 cannot select the CR oscillation circuit as a clock generator. If this hinders your program development, please contact Oki Electric's engineering department.

4-1-10. LCD Driver

Data transfer from display register to display F/F will forcibly be terminated when the LCD assignment definition data contents is 0FFH. So, set the unused area of the LCD assignment definition data other than 0FFH. For details, refer to each MSM64153 family microcontroller's User's Manual.

4-2. EASE64158 Timing

EASE64158 timing is shown on the next page. The entries on the timing chart are explained below.

SYS•CLK System clock.

M1•S1 Start of instruction.

S2 Start of second machine cycle.

PC MSM64E153 evaluation chip address.

Cycle Counter Up Count timing of cycle counter.

Trace Latch 1 Trace latch timing for executed address during **G** command continuous execution.

Trace Latch 2 Trace latch timing for everything but executed address (**AR**, **BR**, **SP**, **ports**, **RAM** data) during **G** command continuous execution.

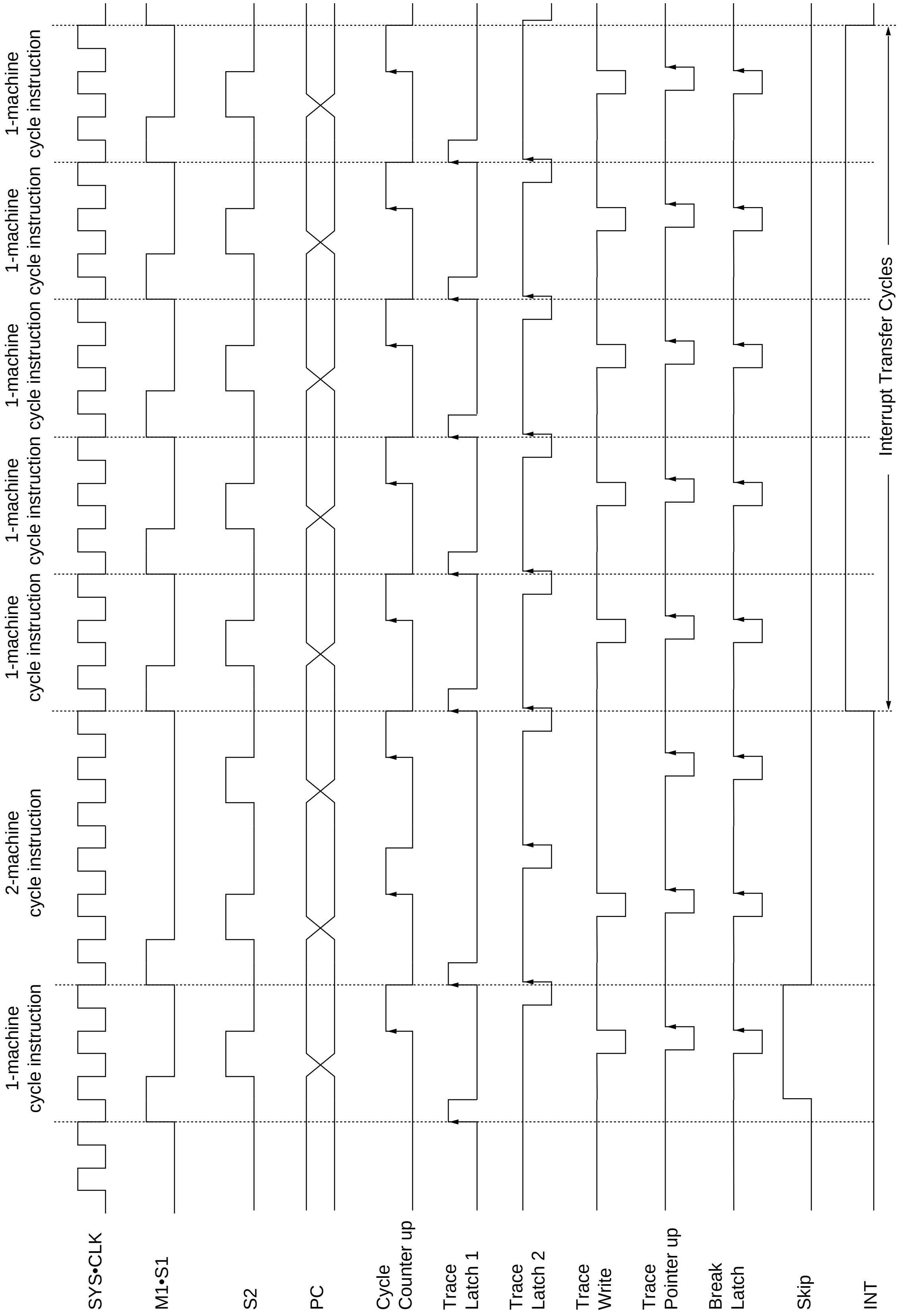
Trace Write Timing for writes of data latched with trace latch 1 or trace latch 2 timing to trace data memory.

Trace Pointer Up Count timing of trace pointer.

Break Latch Break timing for address breaks, breakpoint breaks, trace full breaks, and cycle counter overflow breaks.

SKIP Skip execution.

INT Interrupt transfer cycle flag.


1-machine cycle instruction
2-machine cycle instruction
1-machine cycle instruction
1-machine cycle instruction
1-machine cycle instruction
1-machine cycle instruction
1-machine cycle instruction
Interrupt Transfer Cycles
SYS•CLK
M1•S1
S2
PC
Cycle Counter up
Trace Latch 1
Trace Latch 2
Trace Write
Trace Pointer up
Break Latch
Skip
INT

Chapter 5

Assemble Command

This chapter describes the assemble command in detail.

The assemble command (**ASM** command) is provided to enhance program debugging effectiveness with the emulator. By using the assemble command, the user can rewrite code memory using OLMS-64K mnemonics.

The assemble command supplied with SID64K performs symbol processing with a complete 2-pass process. Thus it can make use of symbols loaded with the **LOD** command, as well as labels, including forward references. Symbols defined within the assemble command can also be referenced by other commands. In addition, the assemble command supports operators compatible with C language, enabling addressing with complex expressions.

Furthermore, the assemble command supports code segments and data segments as logical memory segments. This allows coding of memory allocations within data memory.

The explanations of this chapter assume the MSM64153 as an example. For other chips, refer that chip’s corresponding user’s manual.

5-1. Address Space

The OLMS-64K series has two physically independent memories, code memory and data memory. Each consists of contiguous addresses, and both are logically defined as independent logical address spaces:

❑ Code address space
❑ Data address space

Code address space corresponds one-for-one with code memory, with addresses allocated in 1-byte units. Data address space corresponds one-for-one with data memory, with addresses allocated in 4-bit units. In order to clearly separate these address spaces, a segment type attribute is assigned to each.

When a symbol is defined at an address in one of these address spaces, the symbol is assigned the value of that address value and the segment type of that address space.

The above explanation is summarized in Table 5-1.

Table 5-1. Address Spaces and Segment Types (example of MSM64153)

Address Space	*Corresponding Area*	*Segment Type*
Code address space	Code memory area (0~BFFH)	CODE
Data address space	Internal RAM data area in data space (0~760H)	DATA

5-2. Segments

The concept of segments is introduced with the **ASM** command. The **ASM** command allocates segments to program memory. A segment, defined as an area that has contiguous addresses, is the basic unit for constructing programs.

Segments are classified into the following two types, depending on which address spaces they are allocated to.

CODE segment	Code address space
DATA segment	Data address space

Each segment has its own location counter. A location counter points to a location within its segment. Location counters are managed by the **ASM** command. The range of locations for each segment is shown below (an example of MSM64153).

Segment	*Location Range*
CODE segment	0~0BFFH
DATA segment	0~0760H

Program coding within each segment reflects the features of the corresponding memories. CODE segments are coded with mnemonics that generate machine language code, and with **DB** and **DW** directives that perform memory initialization. DATA segments are coded with **DS** directives that reserve areas for storage. Non-CODE segments cannot be coded to initialize memory contents. For either segment, the location counter can be set to any value with the **ORG** directive.

The value of a segment’s location counter expresses a physical address. Segments are initialized by placing **CSEG** and **DSEG** directives within a program.

5-3. Symbol Table

The **ASM** command has a data table for managing symbols. Generally called a symbol table, it holds symbols expressed within a program and various information about them. The size of the symbol table depends on the size of usable memory.

If the size of memory becomes insufficient for the table, then at that point in time the **ASM** command will output an error message and terminate assembly.

5-4. Assembly Language Format

This section describes the rules of assembly language and the syntax of a source program.

5-4-1. Character Set

All 1-byte character codes can be used. Characters that require 2-byte codes (Japanese characters) cannot be used.

5-4-2. Statement Format

The input of the assemble command is defined as a block of statements. A statement is a character string of up to 56 characters, ending with a carriage return key input.

Statements are broadly divided into instruction statements and directive statements. Instruction statements are statements that will be translated into machine language code for OLMS-64K series microcomputers. Directive statements are statements for controlling the assemble command; they are not translated into machine language code.

Statements are constructed from four fields: label, instruction, operand, and comment. They are generally coded as follows.

```
LOOP1:     ADC        @XY      ;Comment
label    instruction  operand   comment
```

These four fields are not necessarily required to code statements of actual source programs. Only the needed fields have to be coded. As a special case, blank lines (lines with just a carriage return key input) are recognized as statements.

The order of the fields cannot be altered even if one or more is omitted in the statement. Between the instruction field and operand field one or more spaces or tabs are required. Other fields can be delimited by any number of spaces or tabs (including zero, where two fields are coded with no separation). The maximum number of characters in one statement is 56.

Each field is defined as follows.

(1) Label field

A label field comprises a symbol followed by a colon (:). The colon is handled as the termination code of the label field. Any number of blanks or tabs (including 0) can be placed between the symbol and the colon. The symbol of a label field is assigned the value of the location counter and the segment type of the segment that contains the label field's statement. The symbol of a label field can be referred to by any statement's operand field.

(2) Instruction field

For an instruction statement, the instruction field codes a reserved word that corresponds to machine language (these reserved words are referred to as "instruction mnemonics" or simply "mnemonics" below). For a directive statement, the instruction field codes a reserved word that corresponds to a directive.

(3) Operand field

An operand field codes the necessary number of operands for the instruction coded in the instruction field. Depending on the instruction type, there may be no operand field. Operands are delimited by commas (,). Any number of spaces or tabs can be placed before and after a comma.

(4) Comment field

A comment field starts with a semicolon (;) and ends with a carriage return key. The contents of a comment field are ignored during assembly processing, and have no effect on assembly.

5-4-3. Symbols

Symbols express numbers, addresses, registers, and flags. They can be broadly divided between reserved symbols and user-defined symbols. Reserved symbols, such as **SFR**, are symbols whose meanings and values are predefined. User-defined symbols are defined by the user within the program. By using these symbols effectively, programs can be input more efficiently.

5-4-3-1. Reserved Symbols

The **ASM** command contains basic instructions, directives, control statements, special assembler symbols, and operators as reserved words. There are also data address symbols, bit address symbols, and code address symbols defined for SFR addresses.

These reserved words can be used, but not defined, in a user program. They can only be used for their original purpose. In other words, reserved words are not permitted to be used as labels in a program or to be newly defined with symbol definition directives.

(1) Special assembler symbols

Special assembler symbols are symbols used for certain register types that are required as operands of certain instructions. The special assembler symbols and their corresponding registers are shown below.

Special Assembler Symbol	Register
BA	BA register pair
BSR	Bank specification register
HL	HL register pair
@XY	XY register pair (indirect addressing)

(2) Data address symbols

Data address symbols have as their values I/O data addresses allocated to SFR space (0H—07FH of data address space). Such I/O addresses could by programmed directly as numeric constants, but such programs are difficult to read. Thus, these addresses should be coded using the predefined reserved words.

Because the OLMS-64K series is premised on ASIC expansion for I/O, the names and addresses assigned to I/O will differ for each particular user. To handle this, the debugger reads data address symbol definition files (DCL files) for each device when it initializes.

(3) Code address symbols

Code address symbols have particular code addresses as their values. For example, the reset entry address, interrupt entry addresses, and other addresses fixed in advance will be assigned to symbols. Code address symbols may also differ for different devices, so similarly to data address symbols, this is handled by reading definition files.

5-4-3-2. User-Defined Symbols

Within a source program, symbols defined as labels and symbols defined with symbol definition directives (**EQU**, **CODE**, **DATA**, **SET**) are called user-defined symbols. A user-defined symbols is given a value and segment type in accordance with type of statement that defines the symbol and with the type of segment that includes the statement.

Symbols follow the rules below.

(1) Usable character set for symbols

A — Z a — z 0 — 9 ? _ $

The following characters can be used for symbols.

However, in order to distinguish symbols from numeric constants, the first character of a symbol must not be a numeric digit. Up to 50 characters may be used for a symbol. The assemble command does not distinguish between upper-case and lower-case letters. For example, “TELEX” and “telex” are handled as the same symbol. This feature enables long symbols to be given readable names. For instance, the symbol

WATCHDOGTIMER

is more difficult to read than

WatchDogTimer.

The second symbol can be comprehended immediately.

In general, a symbol can be defined only once within a single module. The symbol defined first will be valid. Definitions with the **SET** directive are an example of this.

5-4-3-3. Location Counter Symbol

The dollar sign ($) is allowed as a symbol indicating the value of the location counter. It indicates the address holding the instruction that uses it. If that instruction is a 2-word instruction, then the location counter value will be the address of the first word. Take the following instruction as an example.

```
JP  $-5  ;Jump to the fifth address before the current location
counter
```

“$” may also be used within user-defined symbols. The “$” is handled as the location counter symbol only when it is used alone. For example, $$ and A$ are handled as user-defined symbols.

5-4-4. Constants

5-4-4-1. Integer Constants

The assemble command handles strings that start with a digit 0 to 9 as integer constants.

Binary, octal, decimal, and hexadecimal numeric expressions are permitted as integer constants. In order to distinguish between these expression radices, a type suffix is appended after the number. For decimal constants only the type suffix “D” may be omitted. When a hexadecimal constant’s first character would normally be a letter (A—F), a zero needs to be inserted as the first character to distinguish it from a symbol.

Table 5-2. Integer Constant Expression Format

Number Type	***Characters Used***	***Type Suffix***	***Examples***
Binary (radix 2)	0, 1	B	1010B, 01101101B, 1001_1001B
Octal (radix 8)	0–7	O, Q	271O, 514Q
Decimal (radix 10)	0–9	D	30D, 1263
Hexadecimal (radix 16)	0–9, A–F	H	753H, 0C6E7H

5-4-4-2. Character Constants

Character constants are characters and escape sequences enclosed in single quotation marks (‘). If a character enclosed in single quotation marks is anything other than a backslash (\), then the character constant will have that character’s ASCII code as its value. If the character after a single quotation mark is a backslash (\), then the character constant will be given a value 00H—FFH in accordance with the code following. The backslash (\) and its following code are called an escape sequence.

❑ *Escape sequences*

\nnn ....Each ‘n’ is a digit 0—7. The ‘nnn’ is recognized as a three-digit octal number which will be taken as the value of the character constant.

\xnn or \Xnn .....Each ‘n’ is a hexadecimal digit (0—9, A—F). The ‘nn’ is recognized as a two-digit hexadecimal number which will be taken as the value of the character constant.

\a ....The ‘a’ can be any character other than ‘x’ or ‘X.’ The character constant is given the ASCII code of ‘a’ as its value. This escape sequence is used to code a single quotation mark or a backslash.

‘\’’ expresses a single quotation mark.
‘\\’ expresses the backslash.

!

1. Character constants are used to code byte values. They should evaluate to values within the range 0H—0FFH. Accordingly, characters with 2-byte codes (Japanese characters) cannot be used between single quotation marks. If an escape sequence evaluates to a value larger than 0FFH, then an error will occur.

! 2. The escape sequences described here are based on the escape sequences of C language. However, such special C character codes as \t (tab), \b (backspace), and \n (carriage return) are not permitted.

5-4-4-3. String Constants

String constants are strings of up to 50 characters enclosed in double quotation marks (“). They are used only as operands of **DB** and **DW** directives. A string constant is given the string’s ASCII codes as its value.

For example, the ASCII codes of ‘A,’ ‘B,’ and ‘C’ are 41H, 42H, and 43H respectively, so the code

```
DB “ABC”
```

will result in the same code as

```
DB 41H, 42H, 43H.
```

Furthermore, string constants can make use of the escape sequences described in section 5-4-4-2. For example,

```
DB “Hello world”, 0DH, 0AH
```

can be coded as

```
DB “Hello world\x0d\x0a”.
```

! 1. Assembly languages in general do not distinguish between character constants and string constants. Frequently both are expressed with single quotation marks. The reason for distinguishing them here is to match the C language specifications for operators and constant expressions in a unified manner.

! 2. (For programmers familiar with C language)
String constants of the assemble command are based on C language, but there is one big difference. In C, a null (‘\0’) is automatically appended after the string, but the assemble command does not add a null to string constants.

5-4-5. Expressions

5-4-5-1. General Format of Expressions

Expressions are coded in the operand field of instructions to provide values. Except for special assembler symbols, all operands of instructions are expressions. Expressions are coded by joining symbols (other than special assembler symbols), constants (except for string constants), and operators. Any number of spaces or tabs may be placed between the symbols, constants, and operators that comprise an expression.

Expressions are evaluated by applying the calculations indicated by the operators to the values of the symbols and constants. The evaluation of an expression has both a value and a type. The value is incorporated within the instruction code, while the type is matched against the type of the segment in which the instruction lies. Single symbols and constants are also recognized as expressions (probably most operands will be coded in this manner). Refer to section 5-4-5-2 regarding the operators used in expressions, and section 5-4-5-3 regarding type evaluation methods.

During evaluation of expressions all values are handled as unsigned 32-bit data. If a calculation result is negative, then it will become a 2’s complement expression. Overflows are ignored. An instruction’s operands have an appropriate range of values for that instruction. When an expression is coded as the operand of an instruction, overflows that occur during calculation are completely ignored, and only the final result will be evaluated for its appropriateness to the instruction. For example, the operand range of the jump instruction **LJP** is 0—0BFFH (the entire code segment area). However,

```
LJP 0FFFFFFFF + 1
```

will not result in an error. The value 0FFFFFFFFH clearly exceeds the operand range of the LJP instruction, but by evaluating the expression with unsigned 32-bit calculations and ignoring overflows, the result will be 0. The above instruction therefore does not become an error, but instead is translated into machine language as

```
LJP 0
```

5-4-5-2. Operators

This section describes the operators that can be used within expressions.

5-4-5-2-1. Arithmetic Operators

Operators	***Function***
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo operation (returns the remainder from dividing the left operand by the right operand)

5-4-5-2-2. Bitwise Logical Operators

Operator	***Function***
&	Bitwise logical AND of left and right operands.
|	Bitwise logical OR of left and right operands.
^	Bitwise exclusive OR of left and right operands.
<<	Bit shift left operand to the left by the value of the right operand.
>>	Bit shift right operand to the right by the value of the right operand.
~	Bitwise invert the right operand.

5-4-5-2-3. Relational Operators

The result of a calculation with a relational operator is boolean value, either TRUE or FALSE. Here FALSE equals 0 and TRUE equals 1.

Operator	***Function***
>	Returns TRUE if the left operand is greater than the right operand. Returns FALSE otherwise.
<	Returns TRUE if the left operand is less than the right operand. Returns FALSE otherwise.
==	Returns TRUE if the left operand and the right operand are equal. Returns FALSE otherwise.

5-4-5-3. Operator Precedence

Operators are not evaluated in their order of appearance, but rather are evaluated in accordance with some predetermined operator precedence. Table 5-3 shows the operator precedence. Operator precedence of 1 is the highest, and successive numbers indicate lower precedence. Operators shown on the same line have the same precedence.

Operators are evaluated in order of precedence, from high to low. Operators with the same precedence are evaluated in order of appearance, from left to right.

Precedence	***Operators***
1	( )
2	~
3	* / %
4	+ -
5	<< >>
6	< >
7	==
8	&
9	^
10	|

5-4-5-4. Segment Type Attributes For Expression Evaluation

The evaluated results for most expressions constructed using operators will have no segment type attribute. However, in several cases they do have a segment type attribute. The rules for segment types within expressions are given below.

(1) An expression that is only a symbol or constant that has no segment type will itself have no segment type.

(2) An expression that is only a symbol that has a segment type will itself have that segment type.

(3) The result of an expression evaluated with the operators +, -, and ( ) may or may not have a segment type. Table 5-4 shows the rules used to decide. In the table, the symbols ‘S’ and ‘N’ indicate whether or not the expression result has a segment type.

S	Value has a segment type.
N	Value has no segment type.

(4) The result of an expression evaluated with any operators other than +, -, or ( ) will not have a segment type.

Table 5-4.

Operand	***Operator***	***Operand***	***Result***
	( )	S	S
	+	S	S
	-	S	S
N	+	S	S
S	+	N	S
S	+	S	N
N	-	S	S
S	-	N	S
S	-	S	N

5-4-6. Addressing Modes

The **ASM** command has the following addressing modes.

1. HL indirect addressing mode
2. XY indirect addressing mode
3. Direct addressing mode
4. Stack pointer indirect addressing mode

For details on addressing modes, refer to the user's manual for the particular microcomputer of the MSM64153 series.

5-5. Basic Instructions

Basic instructions are instructions that correspond to OLMS-64K machine language. They are translated from assembler commands, and after being converted to machine language instructions, they are stored in code memory. For details, refer to the user's manual for the particular microcomputer of the MSM64153 series.

5-6. Directives

Directives are used to control the conditions of assembly, so except for the **DB** and **DW** directives, they do not generate any code.

In general, directives can be coded anywhere within a program, with the following exception: **DB** and **DW** directives cannot be coded within a code segment.

5-6-1. Symbol Definition Directives

Symbol definition directives allow the user to define symbols that express numbers and addresses. Defined symbols can be referenced from anywhere within a program.

5-6-1-1. EQU

Format

```
      symbol  EQU  constant expression
or    symbol  =    constant expression
```

Description

The value given by the expression is assigned to the symbol. Symbols defined with this directive are not given a segment type.

The expression must not include any forward references, and must evaluate to a value in the range 0—0FFFFH (unsigned 16-bit). Symbols defined with this directive are not allowed to be redefined at another location in the same module.

Example

```
ABC   EQU   0BH
ZZZ   =     ABC+2
:     :     :
      LAI   ABC
:     :     :
      LMI   ZZZ
```

5-6-1-2. SET

Format

symbol SET constant expression

Description

The value given by the expression is assigned to the symbol. Symbols defined with this directive are not given a segment type.

The expression must not include any forward references, and must evaluate to a value in the range 0—0FFFFH (unsigned 16-bit). Symbols defined with this directive may be redefined any number of times in the same program with additional **SET** directives.

Example

```
FLAG  SET   1
      :
      LAI   FLAG  ;Value of flag is 1
      :
FLAG  SET   2
      :
      LMI   FLAG  ;Value of flag is 2
:     :     :
```

5-6-1-3. CODE

Format

symbol CODE constant expression

Description

The value given by the expression is assigned to the symbol. Symbols defined with this directive are given the CSEG segment type.

The expression must not include any forward references, and must evaluate to a value in the code address range (0—0BFFH). Symbols defined with this directive are not allowed to be redefined at another location in the same module.

Example

```
CADR1 CODE  250H  ;Assign code address 250H to CADR1
CADR2 CODE  500H  ;Assign code address 500H to CADR2
```

5-6-1-4. DATA

Format

symbol DATA constant expression

Description

The value given by the expression is assigned to the symbol. Symbols defined with this directive are given the DSEG segment type.

The expression must not include any forward references, and must evaluate to a value in the data address range (0—760H). Symbols defined with this directive are not allowed to be redefined at another location in the same module.

Example

```
DADR  DATA  30H   ;Assign data address 30H to DADR
BUFF  DATA  DADR  ;Assign data address DADR to BUFF
```

5-6-2. Memory Segment Control Directives

Code and data definitions are placed in address spaces (segments) that should be defined. The assemble command selects an address space with memory segment directives.

Each segment has its own independent location counter. The location counters are managed by the assemble command itself. Location counter values correspond one-for-one with the addresses in each segment.

The code segment's location counter is initialized to a value given as an operand when the **ASM** command is invoked. The data segment's location counter is initialized to 0 when the assemble command is invoked.

One segment can be split up into numerous instances within a program. In such cases, the location counter of a newly selected segment will inherit the value held by the location counter of the same segment when last selected.

One address segment can be selected at one time. A selected segment is effective until either a new segment is selected or until an **END** directive is encountered. In other words, the termination of a segment is not explicitly coded.

The code segment (CSEG) is selected when the assemble command is invoked. At this time the location counter is initialized to 0.

5-6-2-1. CSEG

Format

CSEG

Description

This directive defines the start of the code segment. When CSEG is first defined the location counter will have a value of 0. The location counter of the code segment takes values in the range 0—0BFFH. The location counter is updated by **ORG**, **DS**, **DB**, and **DW** directives as well as instructions that translate to machine language.

When CSEG is defined two or more times, its location counter value will start from the last location counter value within the previous CSEG.

Example

```
ORG   100H
DS    10    ;100H
AIS   0FH   ;10AH
DSEG
:
CSEG
DCM         ;10BH
DW    123   ;10CH
```

5-6-2-2. DSEG

Format

DSEG

Description

This directive defines the start of the data segment. When DSEG is first defined the location counter will have a value of 0. The location counter of the code segment takes values in the range 0—0760H. The location counter is updated by **ORG**, and **DS** directives.

When DSEG is defined two or more times, its location counter value will start from the last location counter value within the previous DSEG.

Example

```
      DSEG
      ORG   10H
DX1:  DS    10    ;10H
      CSEG
      :
      DSEG
DX2:  DS    5     ;1AH
DX3:  DS    2     ;1FH
      :
```

5-6-3. Location Counter Directives

Location counter directives are used to change the location counter to any value.

5-6-3-1. ORG

Format

ORG constant expression

Description

This directive changes the location counter of the current segment to the value of the constant expression.

The constant expression must not include forward references. Its value cannot exceed the range for locations of the current segment. If the constant expression has a segment type, then it must be the same as the current segment type.

If this directive increases the location counter from its current value, then the addresses in the intervening space will reside in the currently selected segment.

Example

```
ORG   50H
LAM         ;50H
AND   @XY   ;51H
INM         ;53H
:
ORG   60H
LMI   5
XAM         ;60H
:           ;62H
```

5-6-3-2. DS

Format

[label:]   DS   constant expression

Description

This directive reserves an area with the number of bytes given by the expression and advances the location counter. The assemble command does not generate and code in this area.

The **DS** directive cannot be used within a bit segment.

The constant expression must not include forward references.

This directive updates the location counter of the current segment by the value of the expression, but it cannot exceed the range of that segment's location.

Example

```
ORG   20H
DS    10    ;Reserves code memory
            ;for 10 bytes
LMI   0FH
DSEG
DS    50    ;Reserves code memory
:           ;for 50 bytes
CSEG
NOP
:
```

5-6-3-3. NSE

Format

NSE

Description

This directive advances the location to a 16-byte boundary.

Example

```
      JPL   SUB_1 ;131H
      NSE
TBL:  DB    41H   ;140H
      DB    42H
      :
      :
```

5-6-4. Data Definition Directives

Data definition directives initialize code memory in 1-byte or 1-word units.

5-6-4-1. DB

Format

[label:]   DB   constant expression(s)

Description

This directive is used to initialize the contents of code memory in 1-byte units. Accordingly, it is used only within the code segment. Each expression must evaluate in the range 0—0FFH.

String constants can be used as the constant expression. They will be recognized as a string of data of the 1-byte ASCII codes of each character. When two or more expressions or string constants are coded, they must be separated by commas.

Each item of data is allocated to memory in order starting from the current code address.

String constants can contain a maximum of 50 characters.

If the location symbol ($) is specified, then it will be recognized as the code address value at the defined location.

Example

```
      DB    0
      DB    1, 2, 3
      DB    ‘A’
MSG:  DB    “string”
```

5-6-4-2. DW

Format

[label:]   DW   constant expression(s)

Description

This directive is used to initialize the contents of code memory in 1-word units. Accordingly, it is used only within the code segment. Each expression must evaluate in the range 0—0FFFFH.

When two or more expressions are coded, they must be separated by commas. Each item of data is allocated to memory in order starting from the current code address.

If the location symbol ($) is specified, then it will be recognized as the code address value at the defined location.

Note: Unlike the **DB** directive, the **DW** directive cannot take string constants for operands.

Example

```
DW     ‘A’    ;Allocate a 0
DW     1      ;to the upper byte
DW     12345
ORG    100H
DW     $      ;Allocate
DW     $-2    ;100H
```

5-6-5. Assembler Directives

Assembler directives add special checking functions during assembly and change the state of assembly.

5-6-5-1. END

Format

END

Description

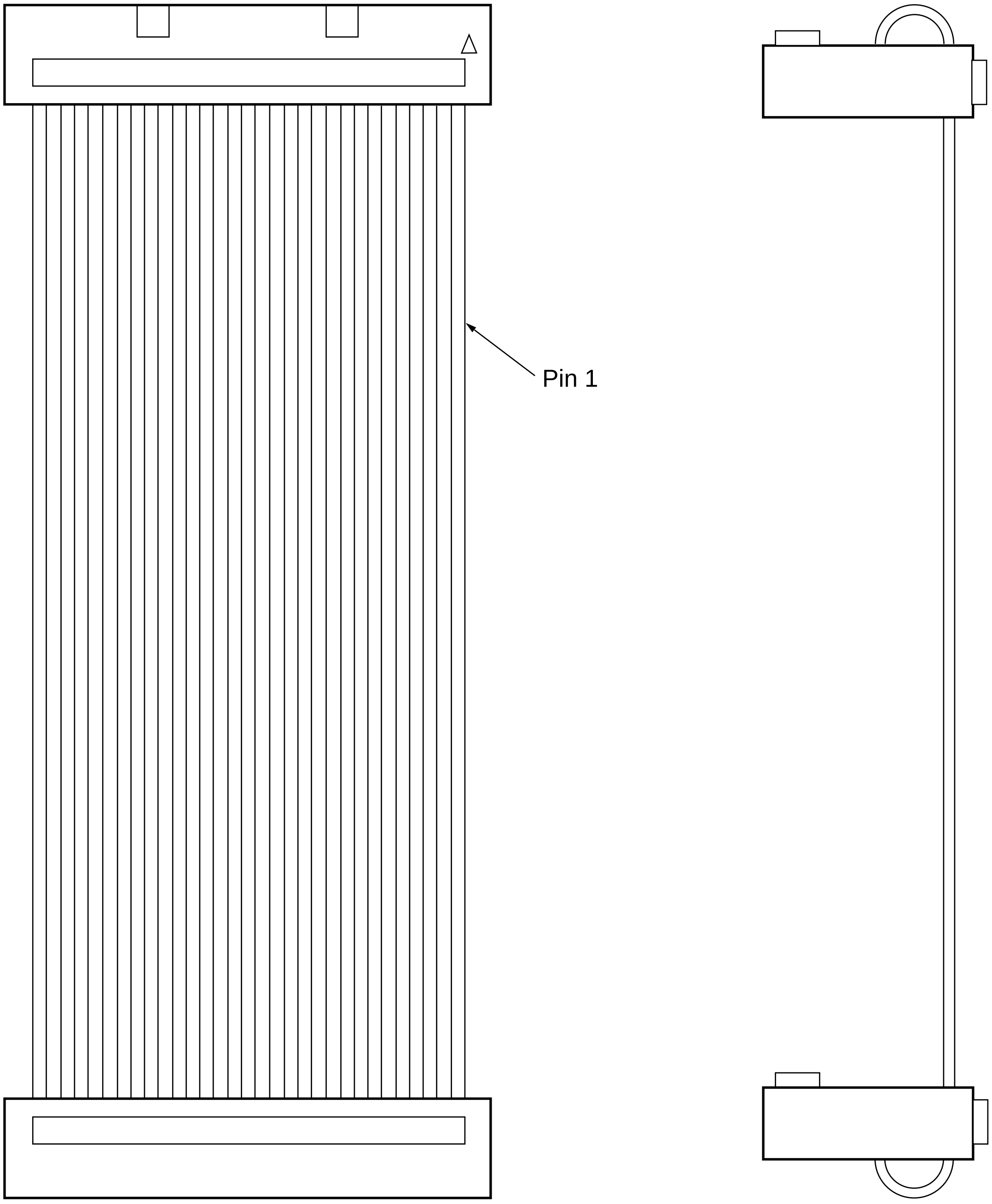
This directive indicates the end of a program. When the **ASM** command encounters an **END** directive, it completes pass 1 processing and immediately enters pass 2 processing.

Appendix

A-1. User Cable Configuration

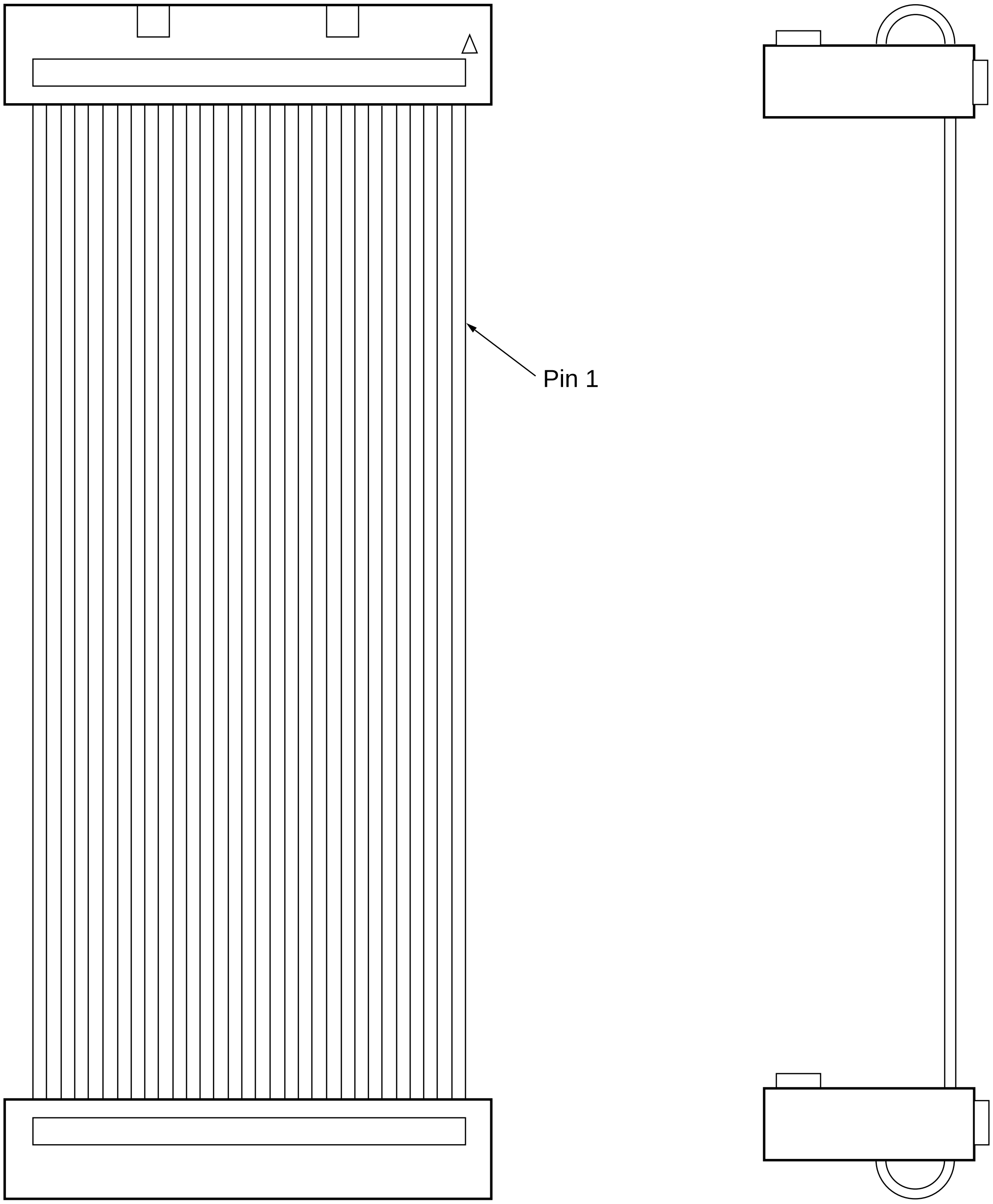
(1) User Cable 1 Configuration

The diagram below shows the configuration of the accessory user cable 1 (one 64-pin cable). The user cable 1 should be connected to USRCN1 connector.



(2) User Cable 2 Configuration

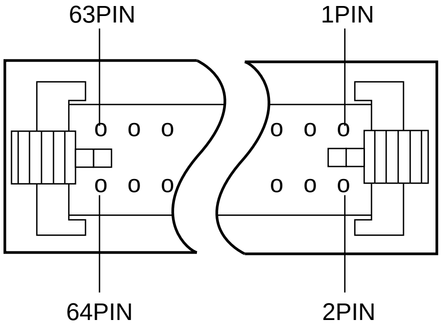
The diagram below shows the configuration of the accessory user cable 2 (one 60-pin cable). The user cable 2 should be connected to USRCN2 connector.



A-2. Pin Layout of User Cable Connectors

(1) Pin Layout of User Cable Connector 1 (USRCN1)

User Connector 1 (USRCN1)



- As shown at left, user connector 1 is a 64-pin connector with pin 1 at the upper right.

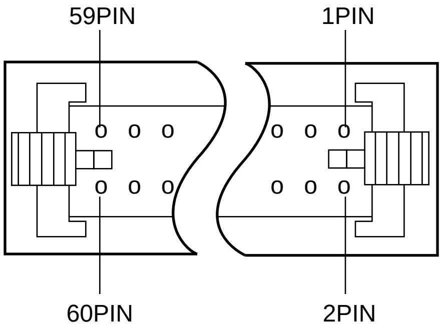
!

The user connector 1 (USRCN1) includes all segment pins available on the MSM64E153 evaluation chip. For details, refer to each MSM64153 family microcontroller's User's Manual.

Pin Number	Signal Name	Pin Number	Signal Name
1	COM 2	33	SEG 29
2	1	34	28
3	4	35	31
4	3	36	30
5	SEG 1	37	33
6	0	38	32
7	3	39	35
8	2	40	34
9	5	41	37
10	4	42	36
11	7	43	39
12	6	44	38
13	9	45	41
14	8	46	40
15	11	47	43
16	10	48	42
17	13	49	45
18	12	50	44
19	15	51	47
20	14	52	46
21	17	53	49
22	16	54	48
23	19	55	51
24	18	56	50
25	21	57	53
26	20	58	52
27	23	59	55
28	22	60	54
29	25	61	57
30	24	62	56
31	27	63	59
32	26	64	58

(1) Pin Layout of User Cable Connector 2 (USRCN2)

User Connector 2 (USRCN2)



- As shown at left, user connector 2 is a 60-pin connector with pin 1 at the upper right.

! The user connector 2 (USRCN2) includes all of the pins other than segment pins available on the MSM64E153 evaluation chip. For details, refer to each MSM64153 family microcontroller's User's Manual.

! The VIN pin and the VOUT pin can be used only with the MSM64153 family microcontrollers that are equipped with the battery check circuit. For detils, refer to each microcontroller's User's Manual.

! The CLK•OUT pin provides the monitor output of the internal clock generated in the POD64158 clock generator.

! 59th and 60th pins of the user connector 2 (USRCN2) will output Vss1 when the MSM64E153 evaluation chip's operating voltage is 1.5 V, and will output Vss2 when the voltage is 3.0 V.

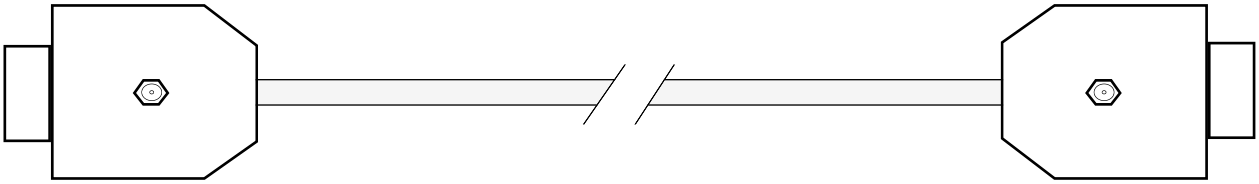
User Connector 2 Pin List

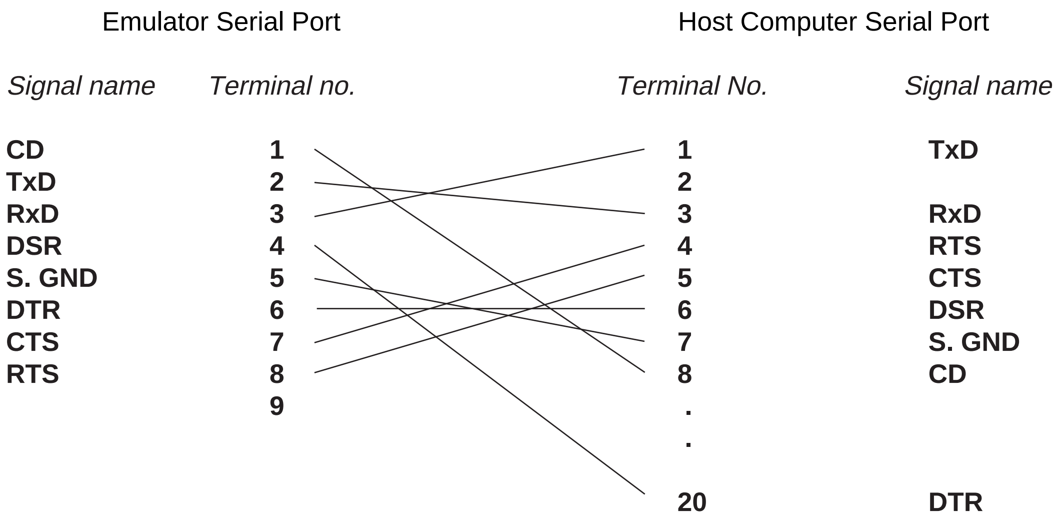
Pin Number	Signal Name	Pin Number	Signal Name
1	BD/	31	P5. 0
2	BD	32	P4. 3
3	MD0/	33	P5. 2
4	MD0	34	P5. 1
5	MD1/	35	P6. 0
6	MD1	36	P5. 3
7	N.C.	37	P6. 2
8	N.C.	38	P6. 1
9	N.C.	39	P7. 0
10	N.C.	40	P6. 3
11	VIN	41	P7. 2
12	N.C.	42	P7. 1
13	P0. 0	43	USER • RESET
14	VOUT	44	P7. 3
15	P0. 2	45	N • C
16	P0. 1	46	EXT • CLK
17	P1. 0	47	CLK • OUT
18	P0. 3	48	N • C
19	P1. 2	49	N • C
20	P1. 1	50	N • C
21	P2. 0	51	N • C
22	P1. 3	52	N • C
23	P2. 2	53	N • C
24	P2. 1	54	N • C
25	P3. 0	55	N • C
26	P2. 3	56	N • C
27	P4. 0	57	N • C
28	P3. 1	58	N • C
29	P4. 2	59	Vss1 or Vss2
30	P4. 1	60	Vss1 or Vss2

Note: NC indicates pin is not connected.

A-3. RS232C Cable Configuration

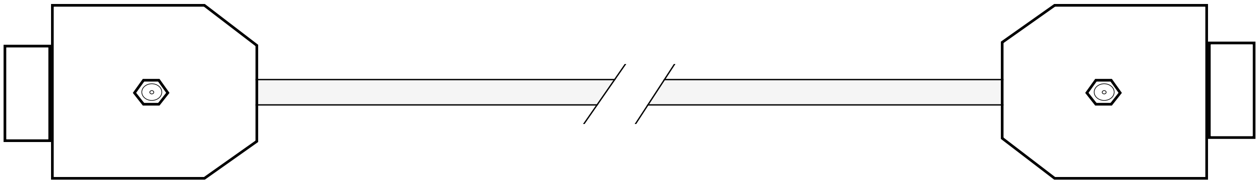
(1) For NEC PC9801 series computers

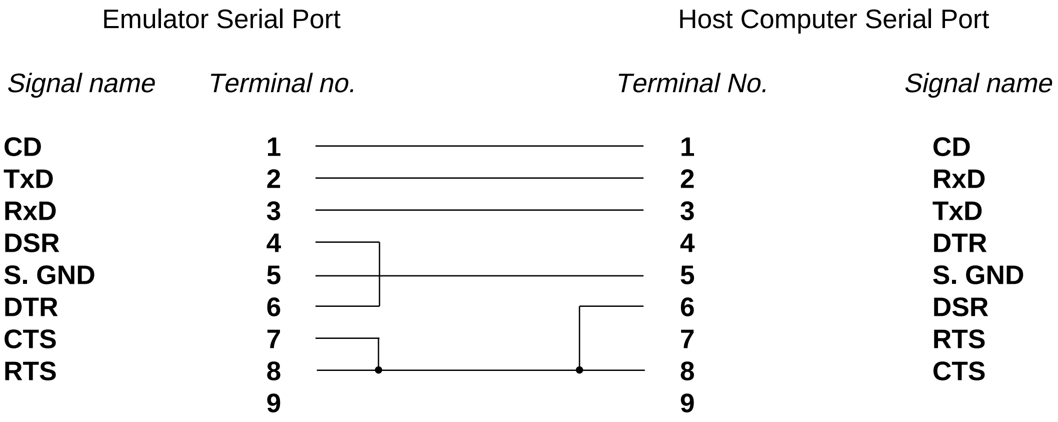




Emulator Serial Port			Host Computer Serial Port	
Signal name	*Terminal no.*		*Terminal No.*	*Signal name*
CD	**1**		**1**	**TxD**
TxD	**2**		**2**	
RxD	**3**		**3**	**RxD**
DSR	**4**		**4**	**RTS**
S. GND	**5**		**5**	**CTS**
DTR	**6**		**6**	**DSR**
CTS	**7**		**7**	**S. GND**
RTS	**8**		**8**	**CD**
	9		**.**	
			.	
			20	**DTR**

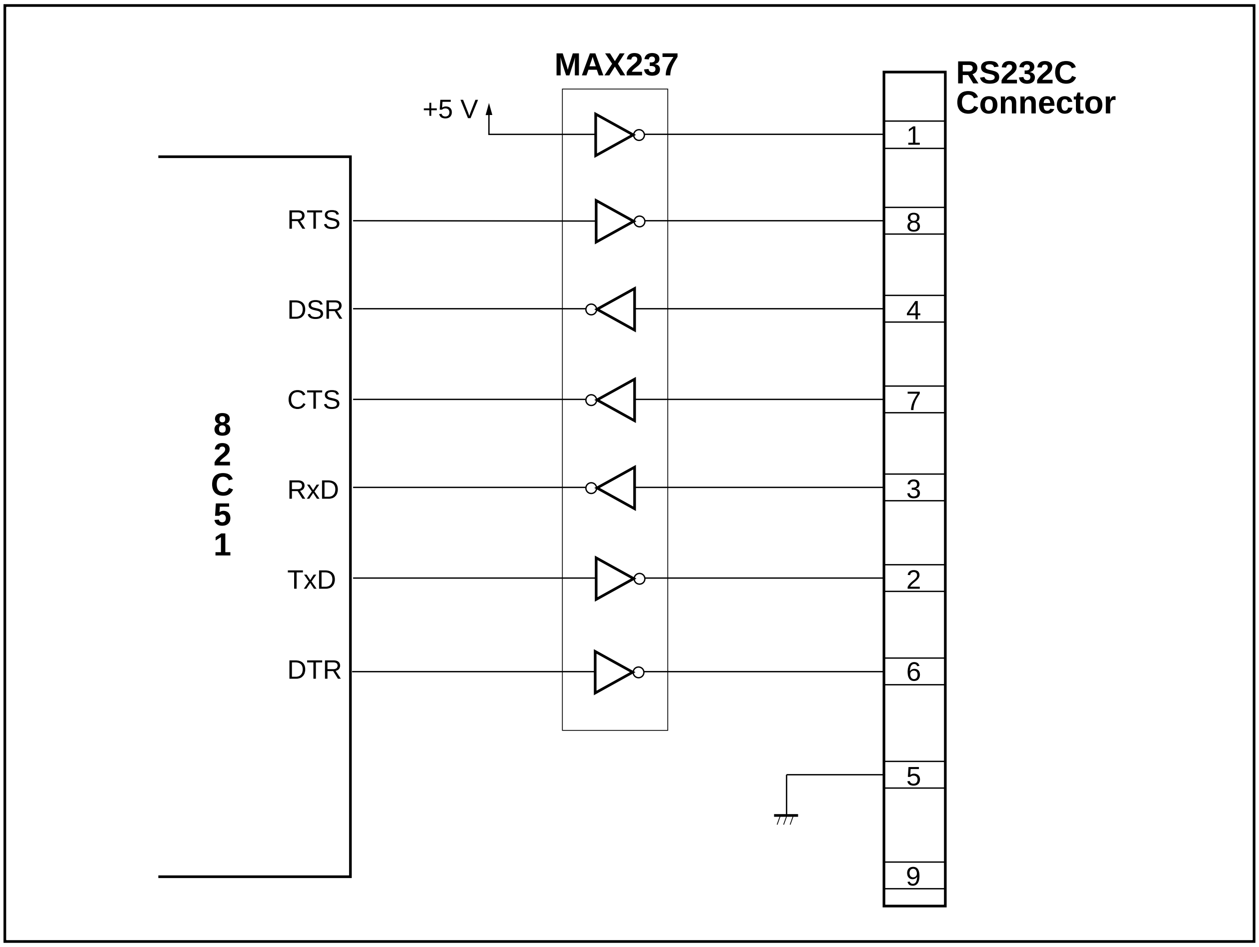
(2) For IBM PC/AT computers



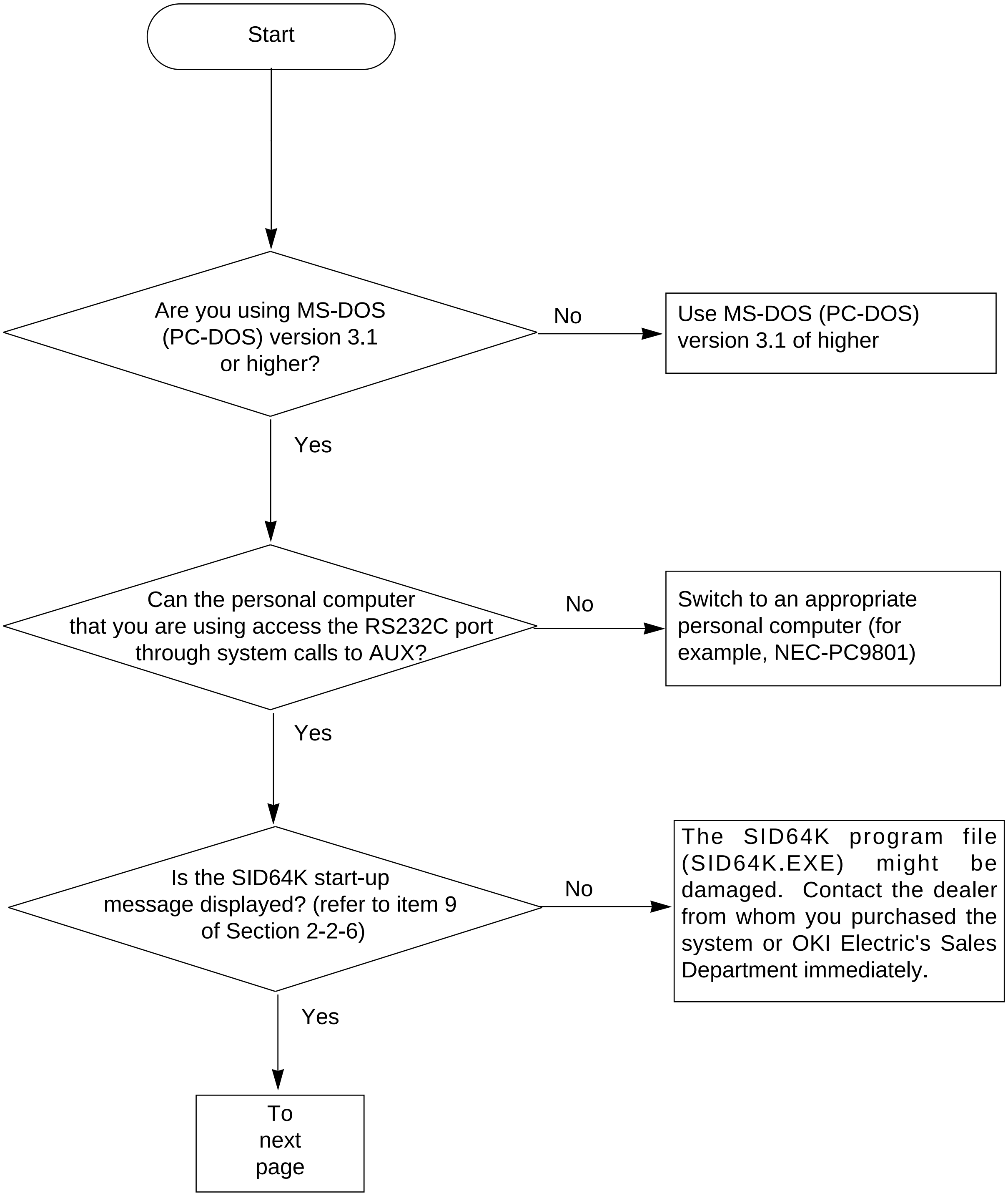


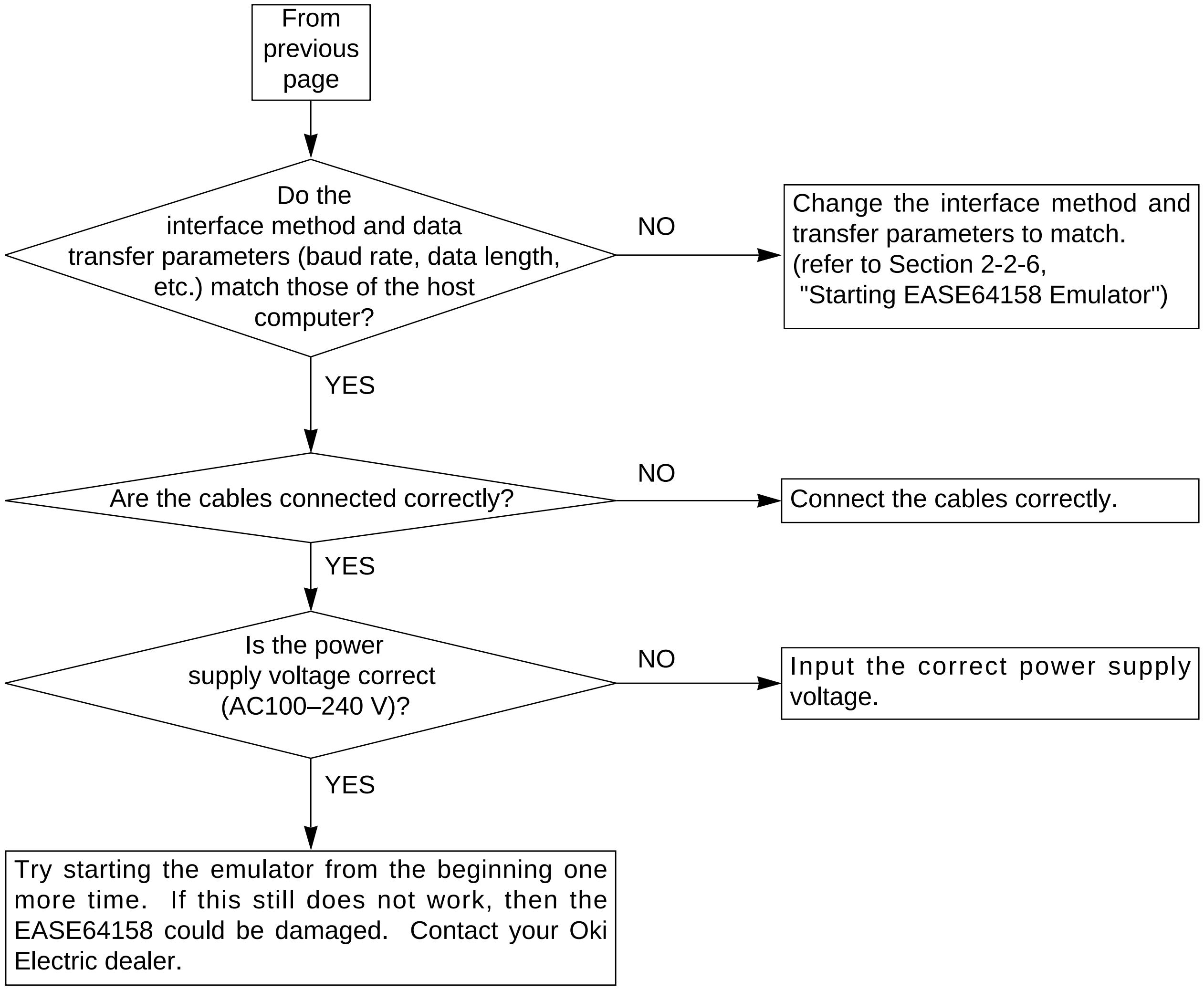
Emulator Serial Port			Host Computer Serial Port
Signal name	*Terminal no.*	*Terminal No.*	*Signal name*
CD	**1**	**1**	**CD**
TxD	**2**	**2**	**RxD**
RxD	**3**	**3**	**TxD**
DSR	**4**	**4**	**DTR**
S. GND	**5**	**5**	**S. GND**
DTR	**6**	**6**	**DSR**
CTS	**7**	**7**	**RTS**
RTS	**8**	**8**	**CTS**
	9	**9**	

A-4. Emulator RS232C Interface Circuit

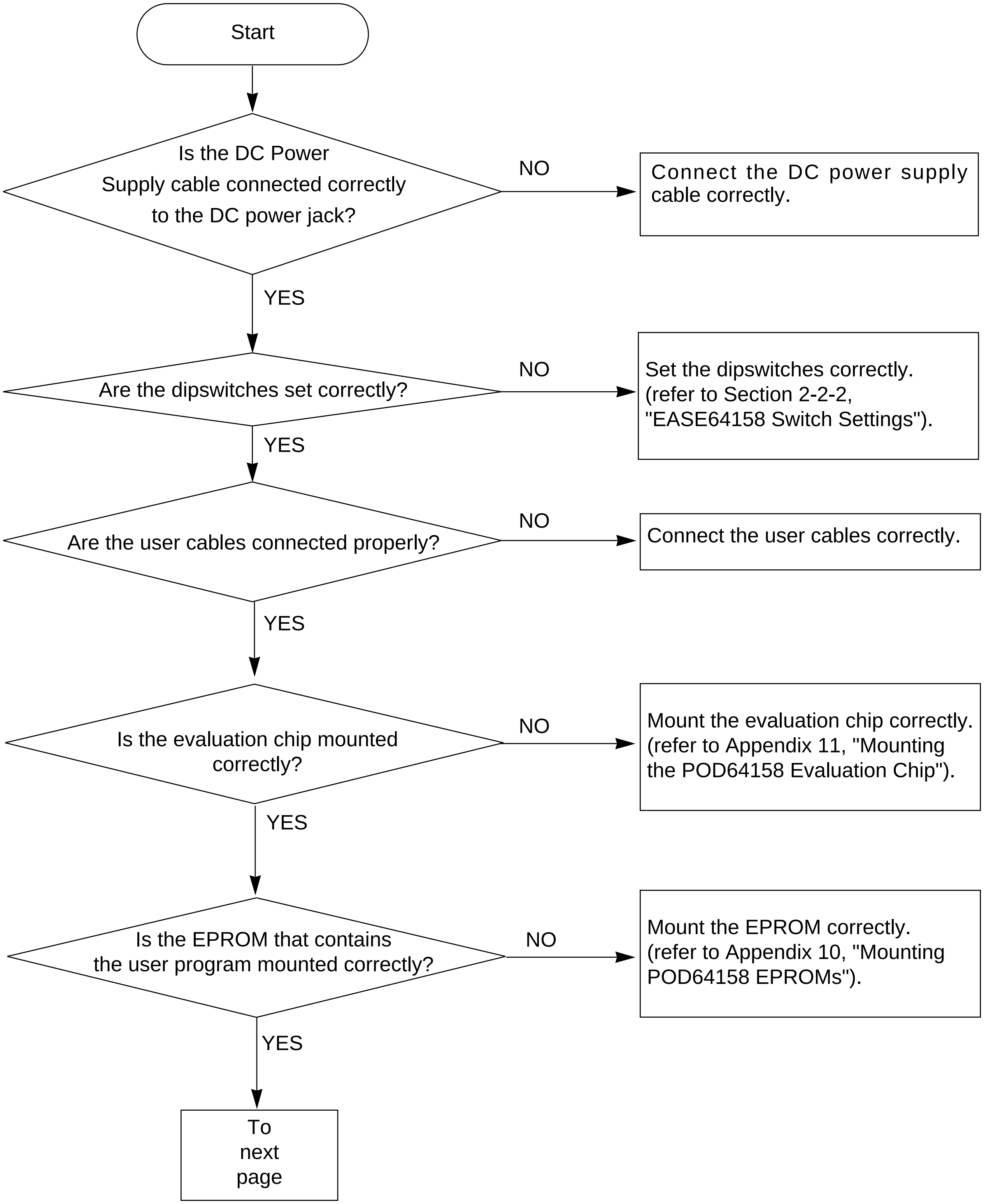


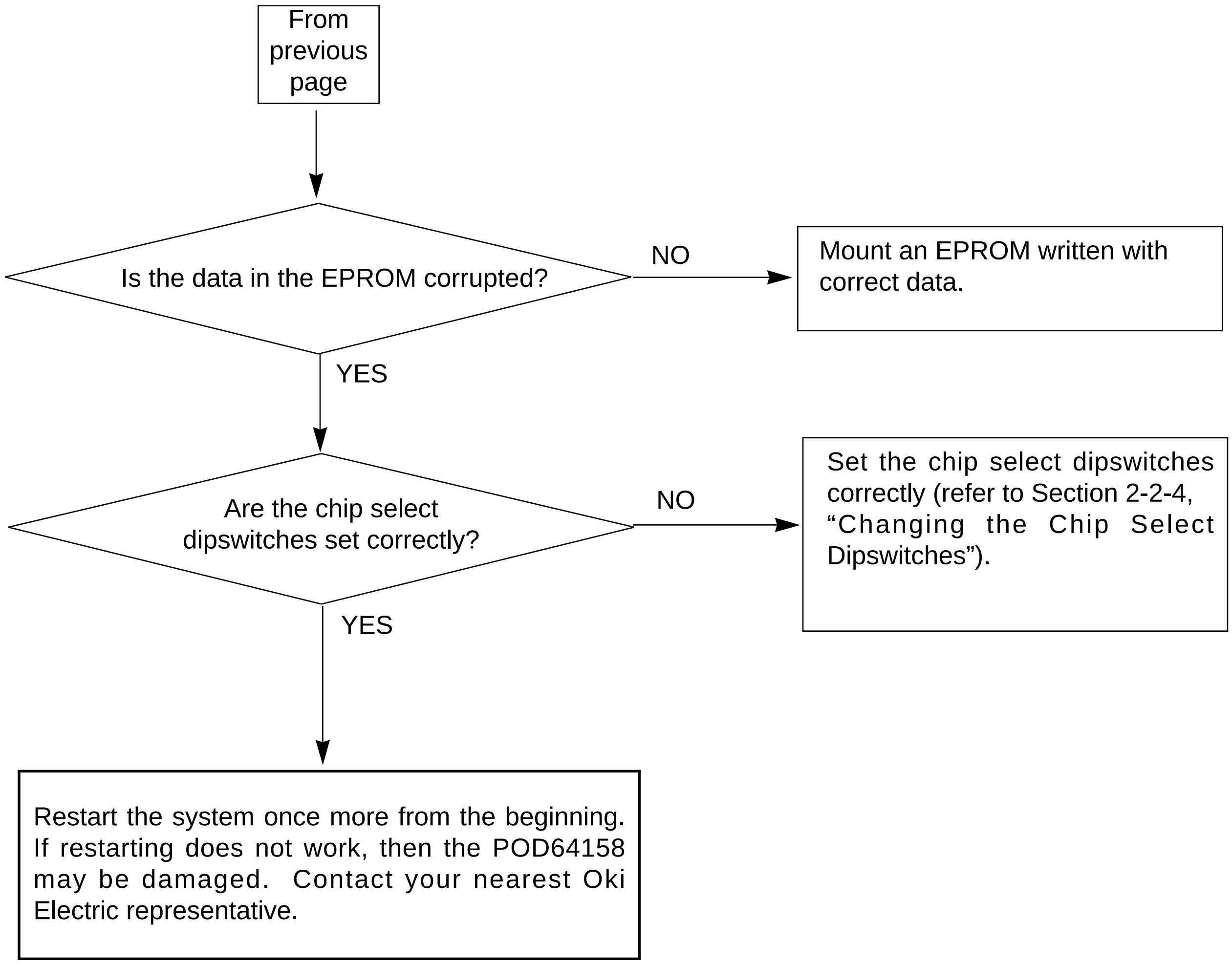
A-5. If EASE64158 Won’t Start


Start
Are you using MS-DOS (PC-DOS) version 3.1 or higher?
No
Use MS-DOS (PC-DOS) version 3.1 of higher
Yes
Can the personal computer that you are using access the RS232C port through system calls to AUX?
No
Switch to an appropriate personal computer (for example, NEC-PC9801)
Yes
Is the SID64K start-up message displayed? (refer to item 9 of Section 2-2-6)
No
The SID64K program file (SID64K.EXE) might be damaged. Contact the dealer from whom you purchased the system or OKI Electric's Sales Department immediately.
Yes
To next page


From previous page
Do the interface method and data transfer parameters (baud rate, data length, etc.) match those of the host computer?
NO
Change the interface method and transfer parameters to match. (refer to Section 2-2-6, "Starting EASE64158 Emulator")
YES
Are the cables connected correctly?
NO
Connect the cables correctly.
YES
Is the power supply voltage correct (AC100–240 V)?
NO
Input the correct power supply voltage.
YES
Try starting the emulator from the beginning one more time. If this still does not work, then the EASE64158 could be damaged. Contact your Oki Electric dealer.

A-6. If POD64158 Isn’t Operating Correctly


Start
Is the DC Power Supply cable connected correctly to the DC power jack?
NO
Connect the DC power supply cable correctly.
YES
Are the dipswitches set correctly?
NO
Set the dipswitches correctly. (refer to Section 2-2-2, "EASE64158 Switch Settings").
YES
Are the user cables connected properly?
NO
Connect the user cables correctly.
YES
Is the evaluation chip mounted correctly?
NO
Mount the evaluation chip correctly. (refer to Appendix 11, "Mounting the POD64158 Evaluation Chip").
YES
Is the EPROM that contains the user program mounted correctly?
NO
Mount the EPROM correctly. (refer to Appendix 10, "Mounting POD64158 EPROMs").
YES
To next page



From previous page

Is the data in the EPROM corrupted?

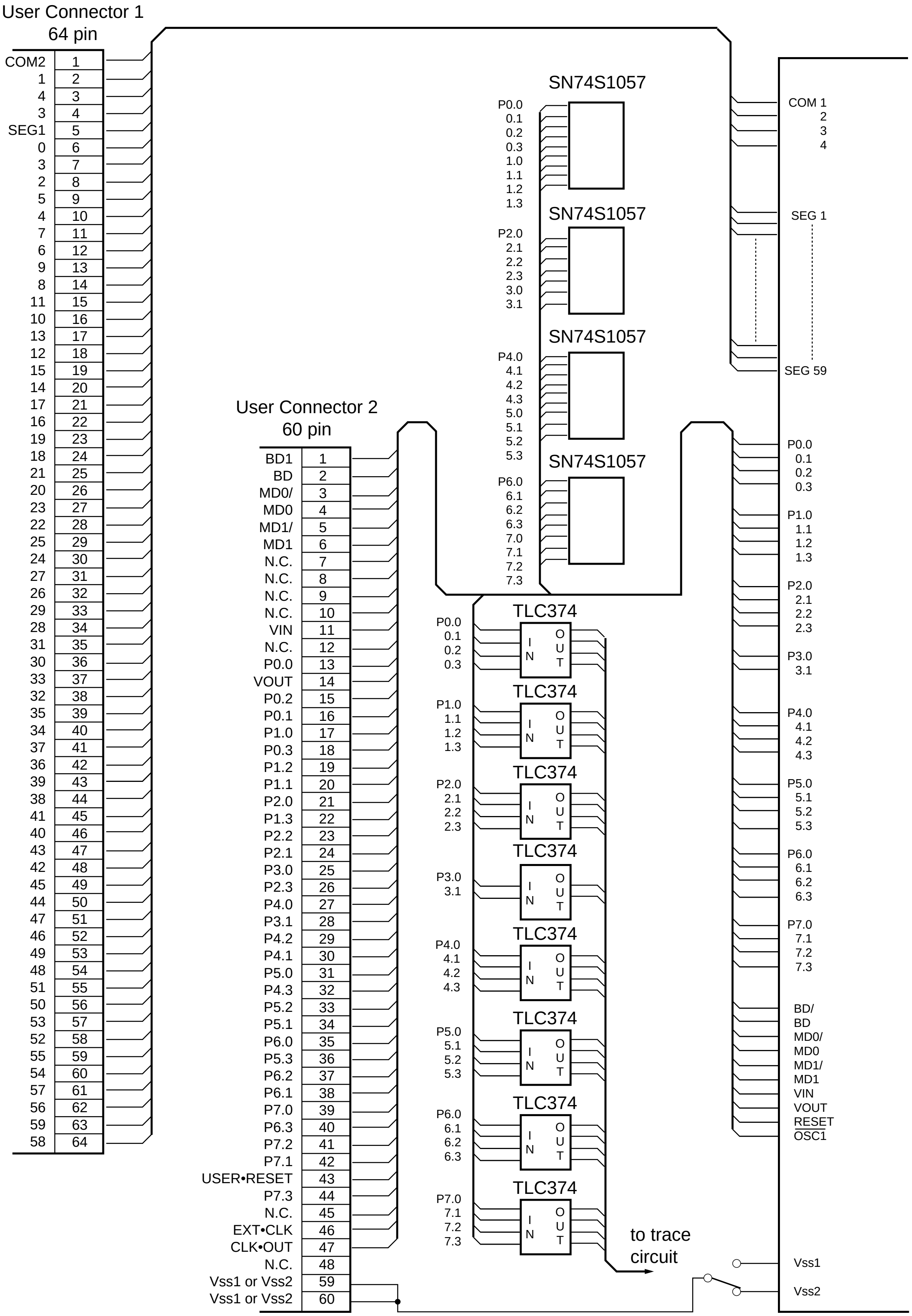
Mount an EPROM written with correct data.

Are the chip select dipswitches set correctly?

Set the chip select dipswitches correctly (refer to Section 2-2-4, "Changing the Chip Select Dipswitches").

Restart the system once more from the beginning. If restarting does not work, then the POD64158 may be damaged. Contact your nearest Oki Electric representative.

A-7. User Cable Peripheral Circuit



User Connector 1
64 pin

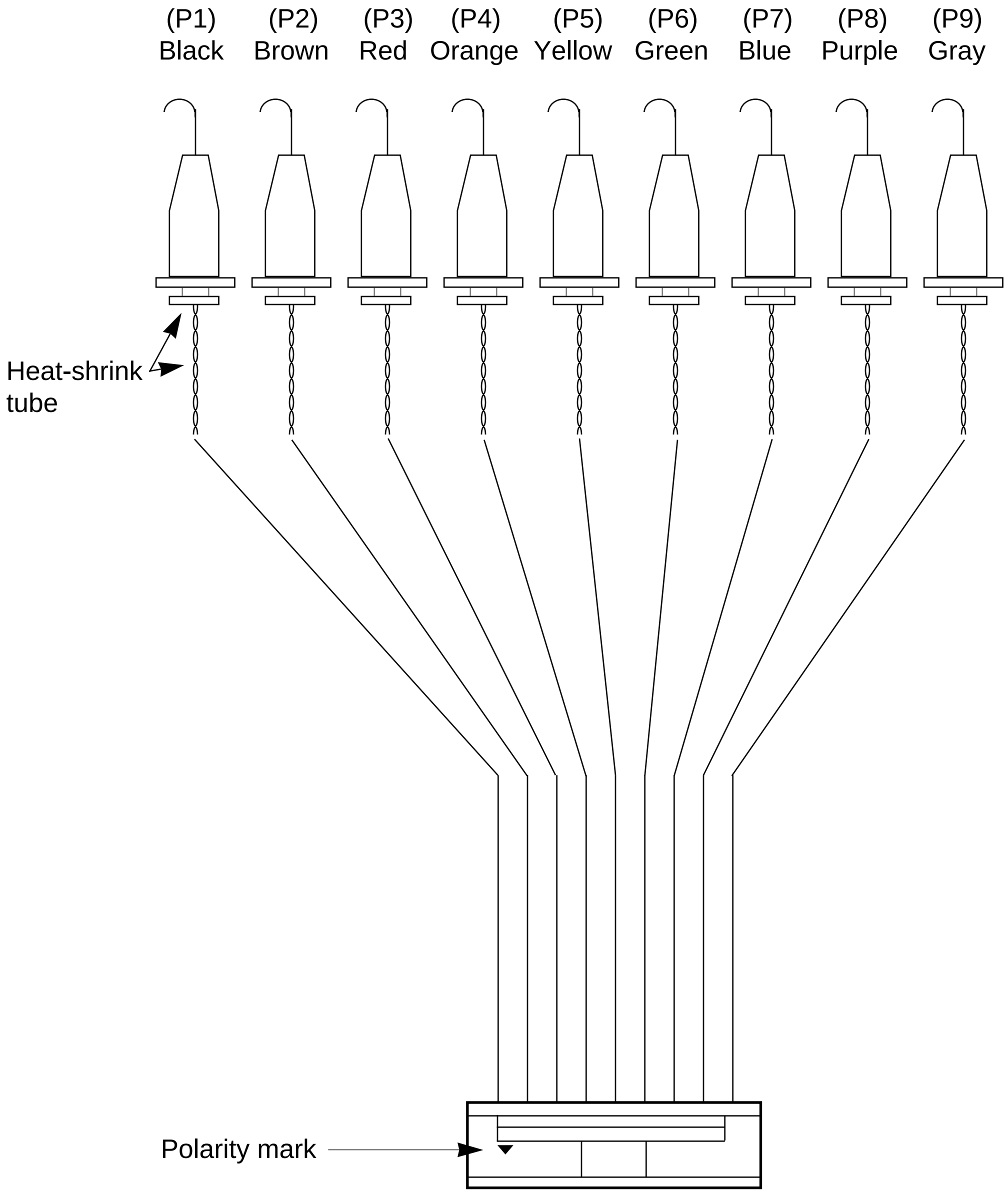
COM2	1
1	2
4	3
3	4
SEG1	5
0	6
3	7
2	8
5	9
4	10
7	11
6	12
9	13
8	14
11	15
10	16
13	17
12	18
15	19
14	20
17	21
16	22
19	23
18	24
21	25
20	26
23	27
22	28
25	29
24	30
27	31
26	32
29	33
28	34
31	35
30	36
33	37
32	38
35	39
34	40
37	41
36	42
39	43
38	44
41	45
40	46
43	47
42	48
45	49
44	50
47	51
46	52
49	53
48	54
51	55
50	56
53	57
52	58
55	59
54	60
57	61
56	62
59	63
58	64

User Connector 2
60 pin

BD1	1
BD	2
MD0/	3
MD0	4
MD1/	5
MD1	6
N.C.	7
N.C.	8
N.C.	9
N.C.	10
VIN	11
N.C.	12
P0.0	13
VOUT	14
P0.2	15
P0.1	16
P1.0	17
P0.3	18
P1.2	19
P1.1	20
P2.0	21
P1.3	22
P2.2	23
P2.1	24
P3.0	25
P2.3	26
P4.0	27
P3.1	28
P4.2	29
P4.1	30
P5.0	31
P4.3	32
P5.2	33
P5.1	34
P6.0	35
P5.3	36
P6.2	37
P6.1	38
P7.0	39
P6.3	40
P7.2	41
P7.1	42
USER•RESET	43
P7.3	44
N.C.	45
EXT•CLK	46
CLK•OUT	47
N.C.	48
Vss1 or Vss2	59
Vss1 or Vss2	60

A-8. Probe Cable Configuration

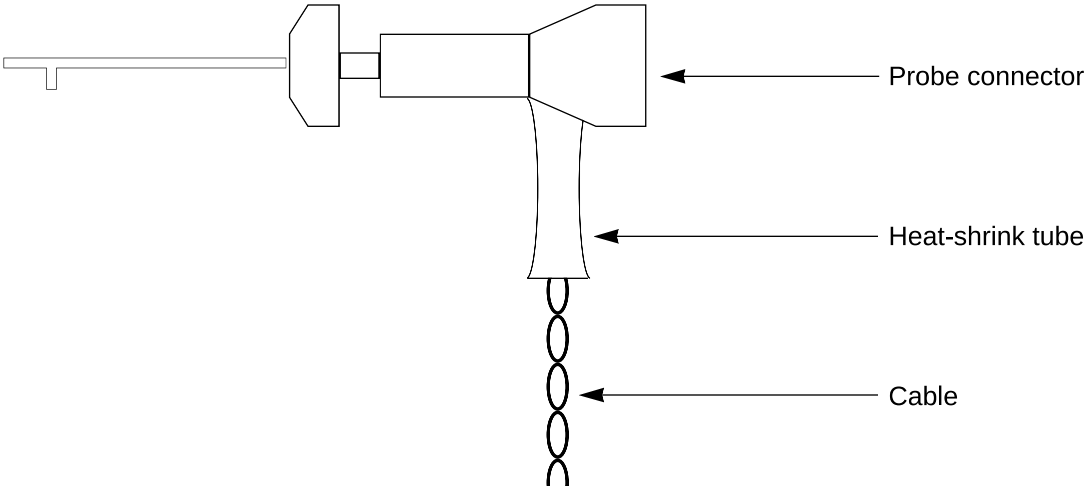
The connector on the right side of the emulation kit marked “PROBE” is for the probe cable. The probe cable configuration is shown below.



The probe cable pins are described next.

The table below shows the probe connector color, the heat shrink tube color, and the cable color for each pin.

Pin number	P-1	P-2	P-3	P-4	P-5	P-6	P-7	P-8	P-9
Probe connector color	Black	Brown	Red	Orange	Yellow	Green	Blue	Purple	Gray
Heat shrink tube color	Gray	Gray	Gray	Gray	Gray	Gray	Gray	Gray	Gray
Cable color	Gray, Black	Gray, Brown	Gray, Red	Gray, Orange	Gray, Yellow	Gray, Green	Gray, Blue	Gray, Purple	Gray, Pink



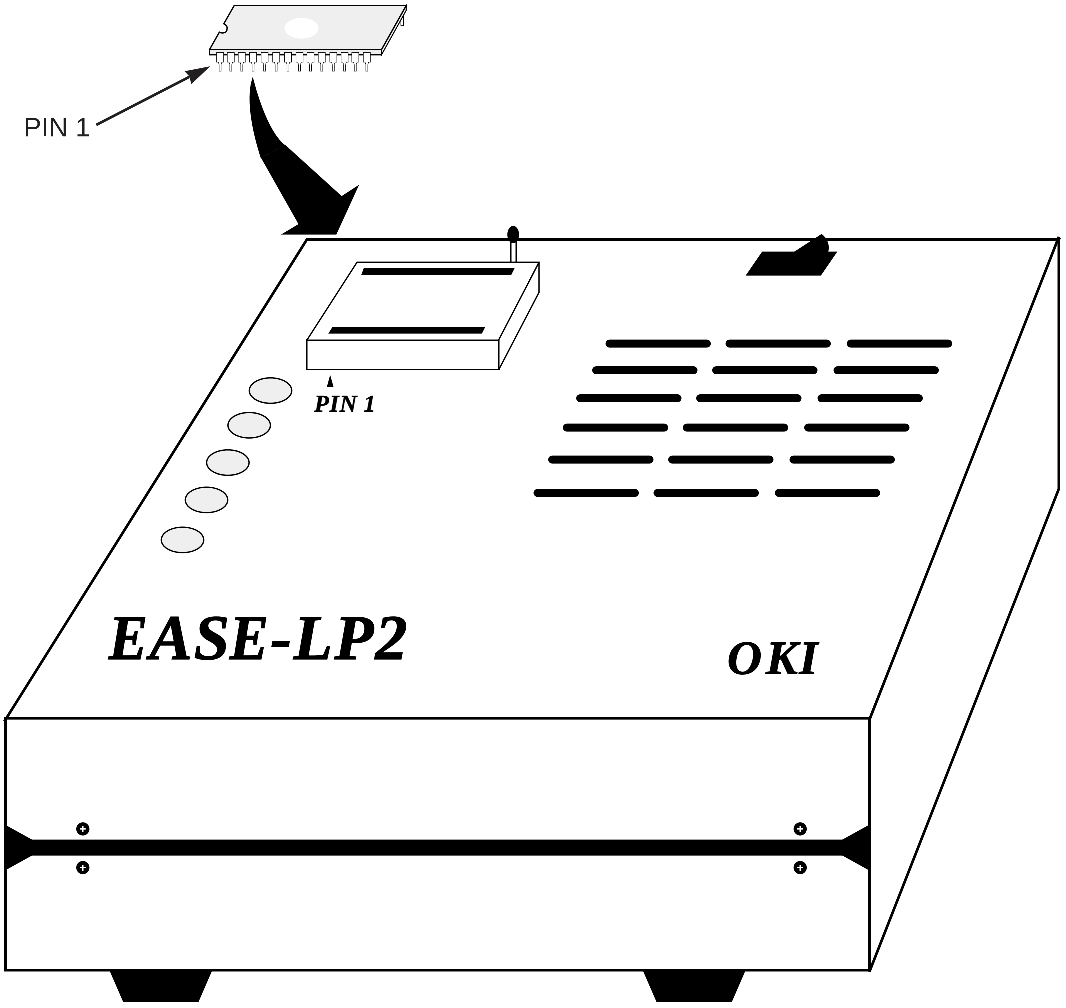
The function of each pin is shown below.

P-1	Probe input bit 0
P-2	Probe input bit 1
P-3	Probe input bit 2
P-4	Probe input bit 3
P-5	Probe input bit 4
P-6	Probe input bit 5
P-7	Probe input bit 6
P-8	Probe input bit 7
P-9	External break signal input

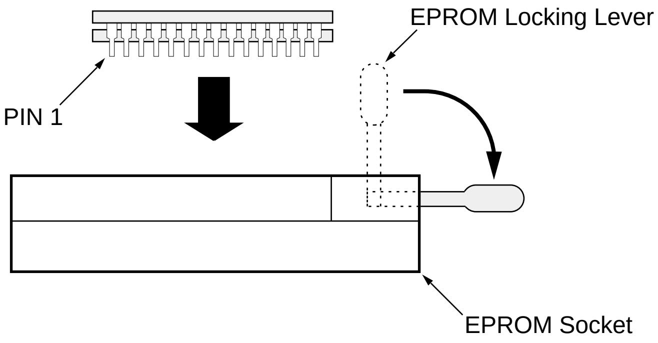
A-9. Mounting EASE-LP2 EPROMs

Follow the procedure below to insert an EPROM into the EASE-LP2's EPROM programmer.

(1) Release the EPROM locking lever on the top surface of the EASE-LP2, as shown below.



(2) Place the EPROM to be read or written in the EPROM socket, as shown below.



To set the EPROM, insert the EPROM in the EPROM socket while the EPROM locking lever is up, and then flip the EPROM locking lever to the horizontal position.

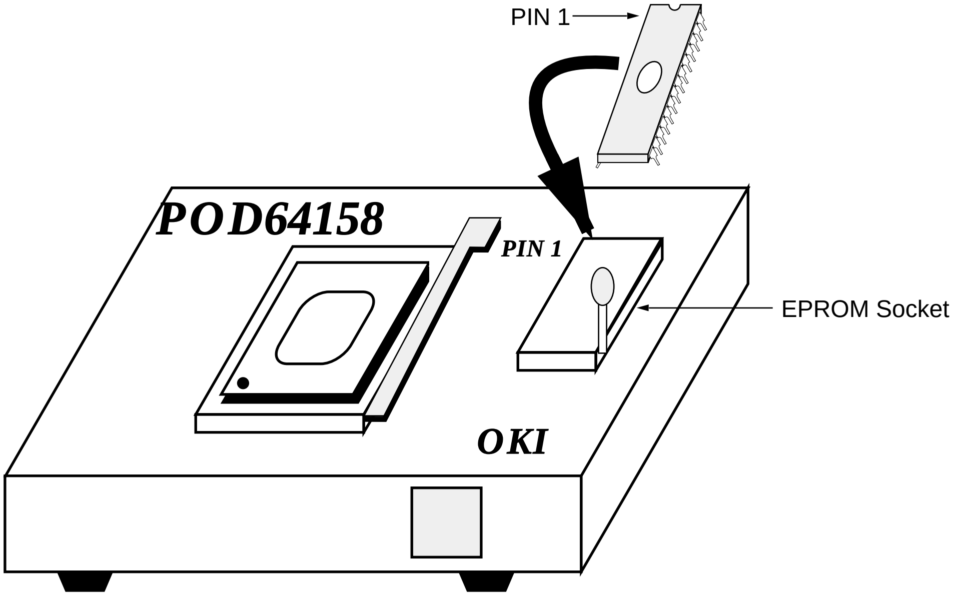
The following types of EPROMs can be written using the EPROM programmer:

2764, 27128, 27256, 27512, 27C64, 27C128, 27C256, 27C512

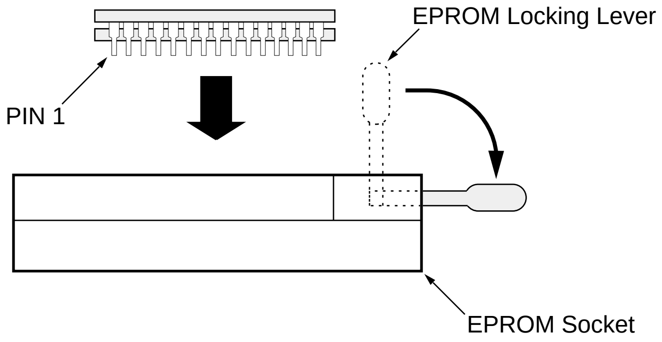
A-10. Mounting POD64158 EPROMs

Follow the procedure below to insert an EPROM into the POD64158's EPROM socket.

(1) Release the EPROM locking lever on the top surface of the POD64158, as shown below.



(2) Place the EPROM containing the user program in the EPROM socket, as shown below.



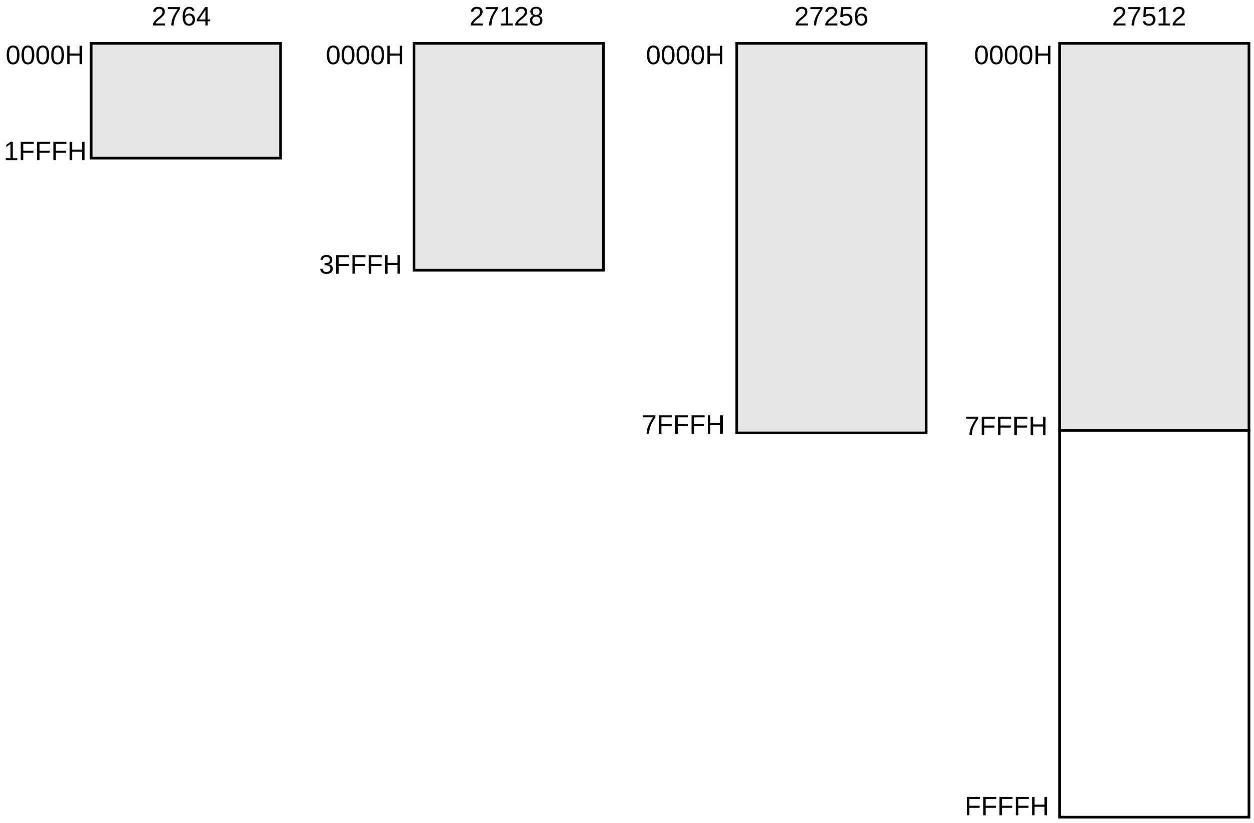
To set the EPROM, insert the EPROM in the EPROM socket while the EPROM locking lever is up, and then flip the EPROM locking lever to the horizontal position.

The following types of EPROMs can be be placed in the EPROM socket:

2764, 27128, 27256, 27512, 27C64, 27C128, 27C256, 27C512

The user program write area for each type is shown below.

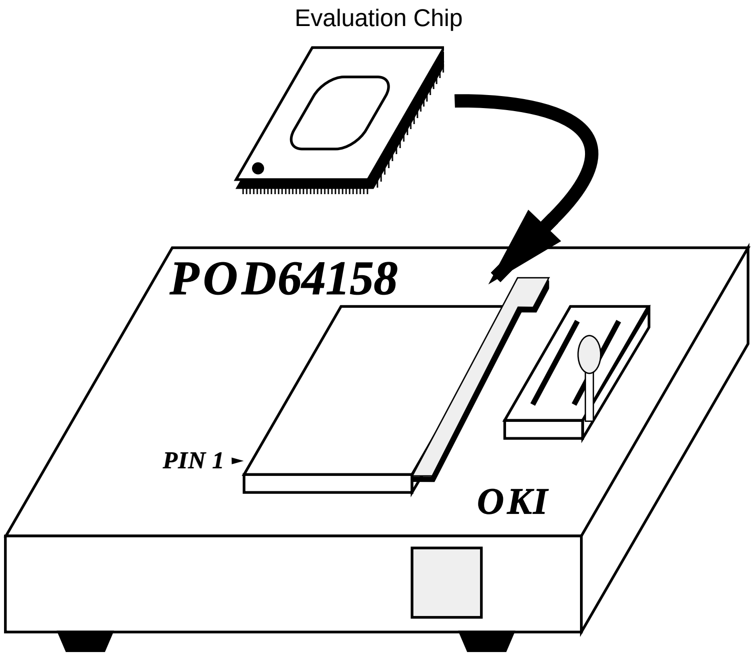
User Program Write Areas
User program is written into shaded portions.



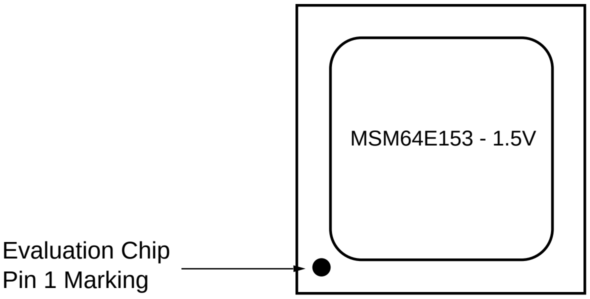
A-11. Mounting the POD64158 Evaluation Chip

Follow the procedure below to insert an evaluation chip into the POD64158's EVA socket.

(1) Release the EVA socket locking lever on the top surface of the POD64158, as shown below.

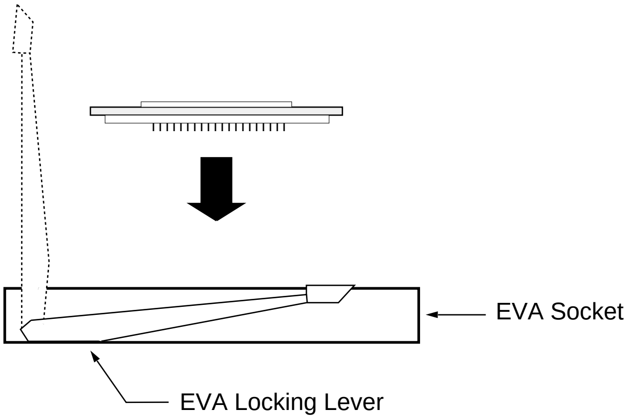


! Be sure that the positions of the pin 1 markings on the evaluation chip and the POD64158 match.



! The evaluation chip with an operating voltage of 1.5 V is marked **MSM64E153-1.5V**. The evaluation chip with an operating voltage of 3.0 V is marked **MSM64E153-3.0V**.

(2) Place the evaluation chip in the EVA socket and then set the locking lever, as shown below.



! ALWAYS TURN OFF THE POWER SUPPLY BEFORE INSERTING OR REMOVING THE EVALUATION CHIP.

! If the evaluation chip is not set in the EVA socket correctly, then it may not operate properly.

! When changing to an evaluation chip with a different operating voltage, the POD64158's dipswitch SW1-2 must be switched. For details, refer to Section 2-2-2, "EASE64158 Switch Settings."

A-12. Error Messages

`Error 002: Emulation busy.`

A command that cannot be executed during emulation was entered.

`Error 003: Data read error.`

Data could not be read from code memory, data memory, or attribute memory. The hardware might be damaged.

`Error 004: Data write error.`

Data could not be written into code memory, data memory or attribute memory. The hardware might be damaged.

`Error 006: Data verify error.`

An error occurred during data verify.

`Error 007: Data address error.`

The input address was not an allowable value.

`Error 011: Read only error.`

An attempt was made to write data to write-disabled SFR.

`Error 012: Write only error.`

An attempt was made to read data from read-disabled SFR.

`Error 013: No support command.`

This command cannot be used on the current version.

`Error 016: Data error.`

The input data value was not an allowable value.

`Error 017: Evachip powerdown.`

The evaluation chip is currently powered down. To release power-down mode, input a reset command or press the reset switch.

`Error 018: Cancel due to N area accessed.`

An unused code memory area was accessed.

`Error 019: Evachip may be faulty.`

An evaluation chip might operate abnormally. The hardware might be damaged. Contact with Oki Electric or sales agent as soon as possible.

`Error 022: Mnemonic error.`

Any error in the *mnemonic* specified for ICE.

`Error 023: Search data not found.`

The data searched for by a search command does not exist.

`Error 025: Function not ready.`

An attempt was made to use functions that are not supported in the current version. Contact with Oki Electric or sales agent as soon as possible.

`Error 028: Trace data not ready.`

An attempt was made to access trace memory that contains no traced data.

`Error 031: Machine trouble.`

Any trouble in ICE. Contact with Oki Electric or sales agent as soon as possible.

`Error 032: Resource number not defined.`

Specified resource number cannot be recognized by ICE.

`Error 035: Illegal parameter.`

Incorrect parameter is specified for ICE.

`Error 036: Trigger mode cancelled.`

Trigger mode setting has been cancelled.

`Error 051: Timeout Error.`

This error message is displayed when the communication port of the host computer remains busy for a fixed time, or when the emulator does not receive any reply from the host computer for a fixed time. ICE abandons the communication for the current block data.

`Error 052: Communication Error.`

This error message is displayed when the data sent from the host computer is out of the specified format, or when it includes an illegal code. The communication line might be in error.

`Error 053: Memory insufficient error.`

Any error caused by insufficient memory of the host computer.

Error 054: Fatal error.

A fatal error occurred in communication control. Contact with Oki Electric or sales agent as soon as possible.

Error 055: Communication buffer overflow.

This error message is displayed when the received data exceeds receive buffer capacity. Busy control setting for the emulator might differ for the host computer.

Error 056: RS232C Transmitter busy.

Data could not be sent to the host computer.

Error 057: SOH Received.

This error message is displayed when both of the emulator and the host computer sent data simultaneously. In this case, the host computer abandons data send and receives data from ICE.

Error 058: Illegal character.

An illegal code is included in received data.

Error 059: RS232C Transmitter empty.

Data could not be received from the host computer.

Error 080: Illegal character.

An illegal character is coded in a symbol.

Error 081: Item too long.

Input character string exceeds allowable number (130 characters).

Error 082: Illegal string constant.

Format of the character string is illegal.

Error 083: Missing terminater of string.

A terminator (“) is not found in a character string.

Error 084: Illegal character constant.

Format of character constant specification is illegal.

Error 085: Illegal hexadecimal character.

Any illegal hexadecimal expression.

Error 086: Illegal decimal character.

Any illegal decimal expression.

Error 087: Illegal octal character.

Any illegal octal expression.

Error 088: Illegal binary character.

Any illegal binary expression.

Error 089: Too many parameters.

Number of input parameter exceeds allowable number.

Error 090: Illegal syntax.

Command expression is incorrect.

Error 091: Operation stack over flow.

The operator stack overflowed during expression analysis.

Error 092: Symbol not found.

Input symbol is undefined.

Error 093: Illegal expression.

There is an error in an expression.

Error 094: Symbol multi-definition.

Specified symbol already defined.

Error 095: Illegal label.

Any illegal character in a label.

Error 096: Reserved symbol.

A reserved word was specified.

Error 097: Special reserved word found in expression.

A special assembler symbol was coded within an expression.

Error 098: Illegal Record.

Any abnormality in Intel HEX file record information.

Error 100: Command not found.

The command does not exist.

Error 101: Illegal address input.

The starting address is greater than the ending address.

Error 102: Illegal data input.

The input data value was not an allowable value.

Error 103: Input data out of range.

The input data value exceeded the allowable range.

Error 104: Illegal filename.

The path name or file name contains an error.

Error 105: File open error.

The specified file cannot be opened.
This error message is displayed when the specified file does not exist, or when the file is a write-only file.

Error 106: File read error.

The file could not be read correctly.

Error 107: File close Failure.

The file could not be closed correctly.

Error 108: File write error.

The file cannot be written correctly. The file might be a read-only file.

Error 109: List file already opened.

An attempt was made to open the already opened list file.

Error 110: List file not opened.

An attempt was made to close a list file that has not been opened.

Error 111: List file close failure.

The list file could not be closed correctly.

Error 112: Batch file already opened.

An attempt was made to open the already opened batch file.

Error 113: Batch file close failure.

The batch file cannot be closed correctly.

Error 114: Checksum error.

A checksum error found during file loading.

Error 115: Memory alloc insufficient.

The necessary memory area could not be reserved for continuing execution. This error message is also displayed when the necessary memory area cannot be reserved for symbol storing.

Error 116: Symbol defined more than once.

An attempt was made to redifine the already defined symbol.

Error 117: Illegal symbol name.

Specified symbol name contains error.

Error 120: Register read error.

A failure occurred in reading register contents.

Error 121: Option error.

Any illegal option specification in **LOD, SAV,** or **VER** command.

Error 122: Illegal filename.

Any illegal character found in the input filename.

Error 123: Target address range over.

The specified address exceeds the EPROM address range.

Error 124: DCL file not found.

The DCL file was not found.

Error 125: Macro Command name too long.

Input macro command name could not be defined, because the name is longer than 8 characters.

Error 126: Illegal macro name.

Illegal macro command name was input.

Error 127: Macro buffer overflow.

An attempt was made to define 10 lines or more of command as a macro.

Error 128: This command is not allowed in MAC.

An attempt was made to define unallowed command in a macro.

`Error 129: Maximum number of mnemonic is Port:2, Register:1.`

Number of trace object exceeds the allowable range. Two ports and one register in maximum can be specified as trace object.

`Error 132: Instruction error in DCL file.`

The #INSTRUCTION of the DCL file contains any assembler instruction that cannot be used for MSM64153 family.

`Error 133: Forwarding address out of range.`

The destination address for **MOV** command exceeds the allowable range.

`Error 134: Illegal TP input.`

Input TP for **DTM** command contains any error.

`Error 135: Trace object error.`

An undefined trace object was specified for a mnemonic of **STF** command or **S** command. Use **CTO** command to define it.

`Error 136: Trace trigger mnemonic error.`

The mnemonic of the specified trace trigger contains any error.

`Error 137: Too many number of trigger address.`

Number of specified trigger address exceeds allowable range.

`Error 151: Connected ICE cannot be used. (May be Custom ICE)`

Currently connected ICE cannot be used. It might be a custom ICE.

`Error 156: This command is not allowed in emulation.`

Any command that cannot be used in emulation mode was input.

`Error 157: COMMAND.COM could not execute.`

Child process (COMMAND.COM) could not be invoked in **SH** command execution.

`Error 158: Illegal parameter. (Can't use in EXPAND mode)`

Any parameter that cannot be used in EXPAND mode was input.