EASE64165/167

Program Development Support for the MSM64165/167 Family

User's Manual

Rev. 1.13

June 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 System Engineering Group
   Product Development Department
   Logic LSI Division
   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 EASE64165/167 in-circuit emulator for Oki Electric's MSM64165 and MSM64167 CMOS 4-bit microcontrollers. The EASE64165/167 is configured from the POD64165/167 evaluation module and the EASE-LP2 special-purpose control system.

The following are related manuals:

- MSM64165 User’s Manual
    - MSM64165 hardware description
    - MSM64165 instruction set description
    - Addressing description

- MSM64167 User’s Manual
    - MSM64167 hardware description
    - MSM64167 instruction set description
    - Addressing description

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

- MASK165 User's Manual
    - MASK165 (MSM64165 mask option generator) operation description

- MASK167 User's Manual
    - MASK167 (MSM64167 mask option generator) operation description

TABLE OF CONTENTS

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

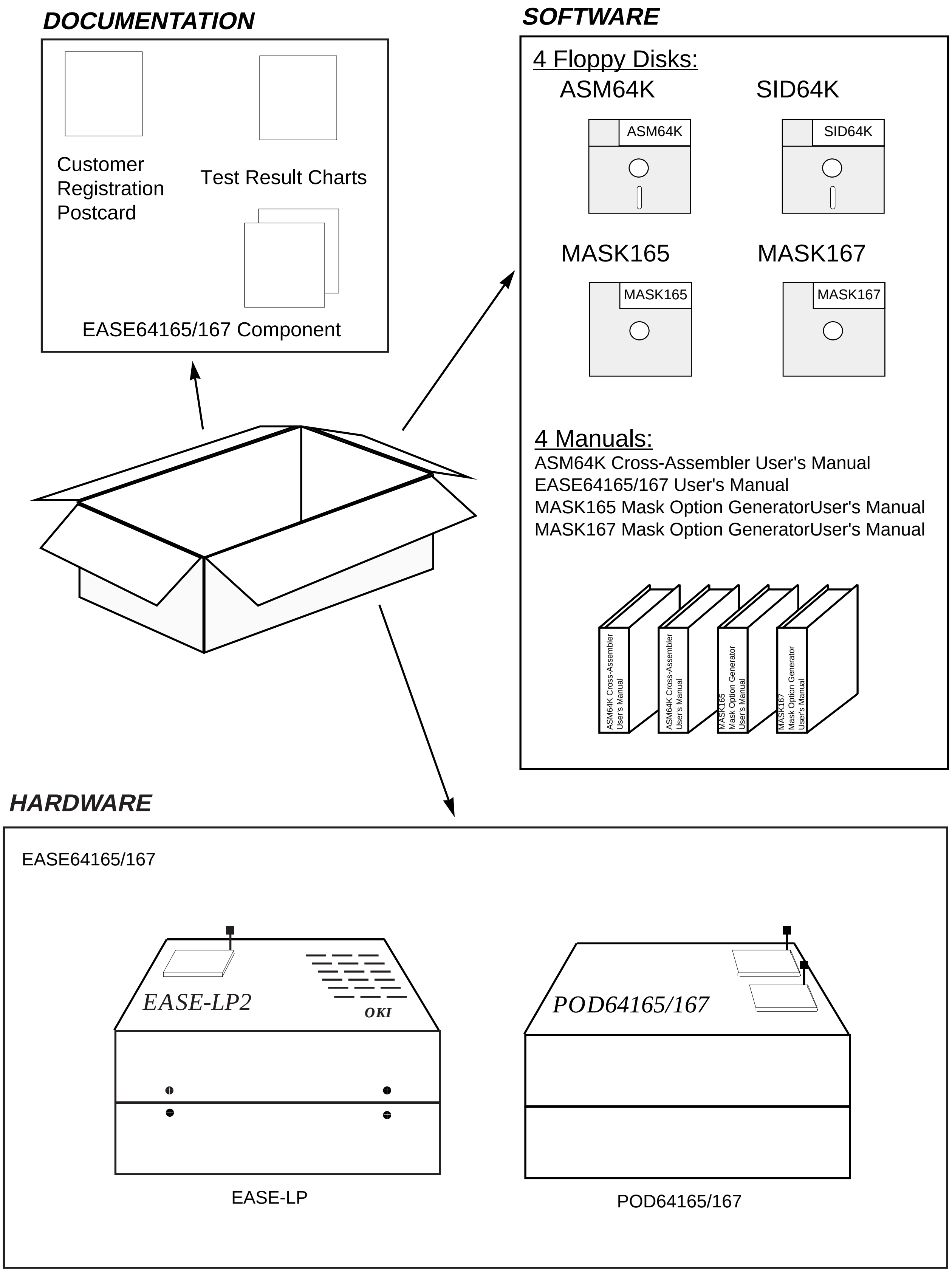
Before Starting

This chapter describes the first things you should do after taking delivery of an EASE64165/167 program development support system.

Thank you for buying Oki Electric’s EASE64165/167 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

EASE64165/167 Component

SOFTWARE

4 Floppy Disks:

ASM64K SID64K

MASK165 MASK167

ASM64K
SID64K
MASK165
MASK167

4 Manuals:

ASM64K Cross-Assembler User's Manual
EASE64165/167 User's Manual
MASK165 Mask Option GeneratorUser's Manual
MASK167 Mask Option GeneratorUser's Manual

ASM64K Cross-Assembler User's Manual
ASM64K Cross-Assembler User's Manual
MASK165 Mask Option Generator User's Manual
MASK167 Mask Option Generator User's Manual

HARDWARE

EASE64165/167

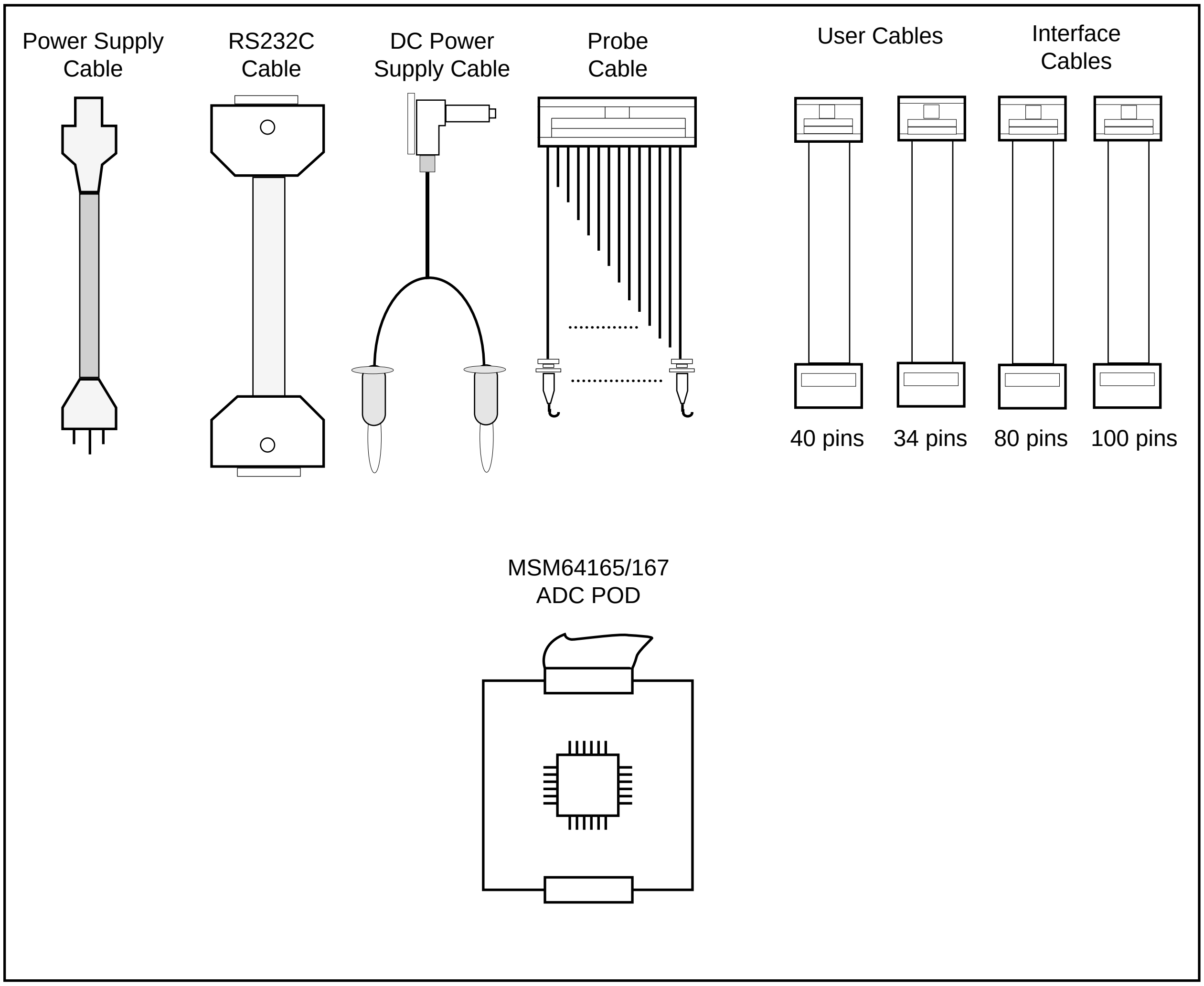
EASE-LP2
OKI
POD64165/167

EASE-LP

POD64165/167

0-2. Confirm Shipping Contents (2)

ACCESSORIES



Your purchase of the EASE64165/167 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.

EASE64165/167 Components List

This is a list of the items shipped.

Test Results Charts

This chart shows that the EASE64165/167 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.

POD64165/167

This is the POD64165/167 evaluation module. It emulates the operation of the MSM64165/167 family.

! The EASE-LP2 and POD64165/167 will be called “EASE64165/167” or “emulation kit” for short.

Software

☞ 1	**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.
☞ 1	**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.
☞ 1	**Floppy Disk: MASK165**	This disk contains the MASK165 executable files. It can be supplied in the formats described below. Floppy disk contents are explained in MASK165 Mask Option Generator User's Manual.
☞ 1	**Floppy Disk: MASK167**	This disk contains the MASK167 executable files. It can be supplied in the formats described below. Floppy disk contents are explained in MASK167 Mask Option Generator Uer's Manual.
	ASM64K Cross-Assembler User's Manual	This is the user's manual for the ASM64K cross-assembler.
	EASE64165/167 User's Manual	This is the user's manual (this manual) for the EASE64165/167.
	MASK165 Mask Option Generator User's Manual	This is the user's manual for MASK165, the mask option generator for MSM64165.
	MASK167 Mask Option Generator User's Manual	This is the user's manual for MASK167, the mask option generator for MSM64167.

☞ **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 personal computers)
- (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: one for NEC-PC9801 and Oki if800 series computers, and one 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 POD64165/167 to the user's application system. Two cables are supplied: a 40-pin flat cable and a 34-pin flat cable.

Interface Cables

These cables connect the EASE-LP2 and the POD64165/167. Two cables are supplied: a 100-pin flat cable and an 80-pin flat cable.

DC Power Supply Cable

This cable supplies Vcc to the POD64165/167 when the POD64165/167 is used stand-alone. It connects to the POD64165/167's DC power jack.

MSM64165/167 ADC POD (3.0V)

This is the ADC POD for MSM64165 and MSM64167. It connects to the POD64165/167.

☞ 2

Unless specified before the EASE64165/167 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 Cable has a 25-pin male connector and 9-pin male connector.

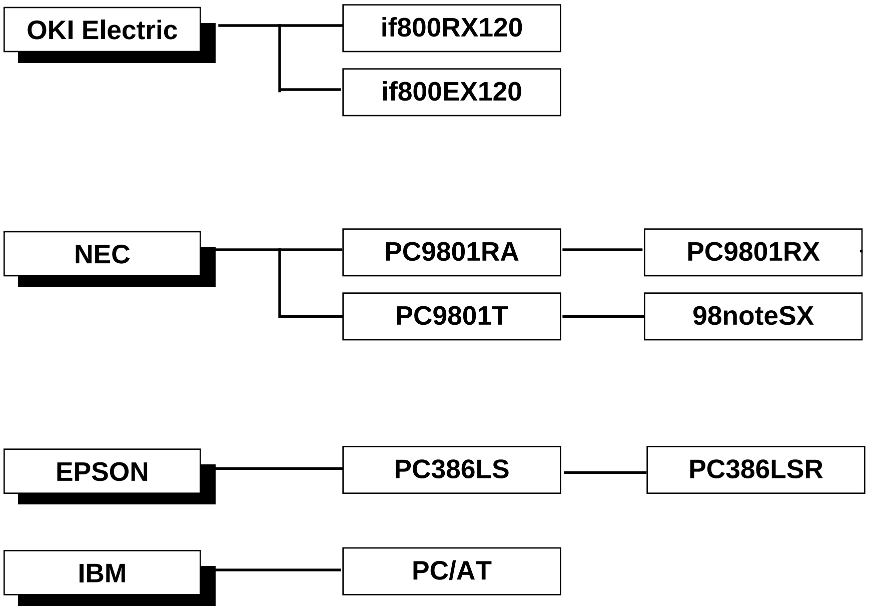
(2) IBM-PC/AT Cable has a 9-pin male connector and 9-pin female connector.

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.

0-2. Confirm Floppy Disk Contents

0-2-1. Host Computer

SID64K, the symbolic debugger for EASE64165/167, 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 EASE64165/167, 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

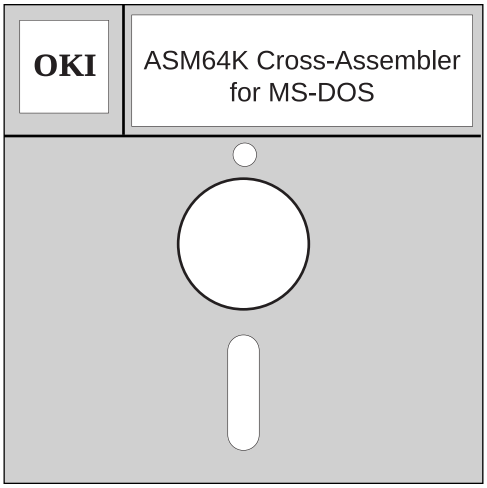
The operating system of computers other than IBM-PCs should be Japanese MS-DOS version 3.1 or later. For IBM-PCs, it should PC-DOS version 3.1 or higher.

0-2-3. Floppy Disk Contents

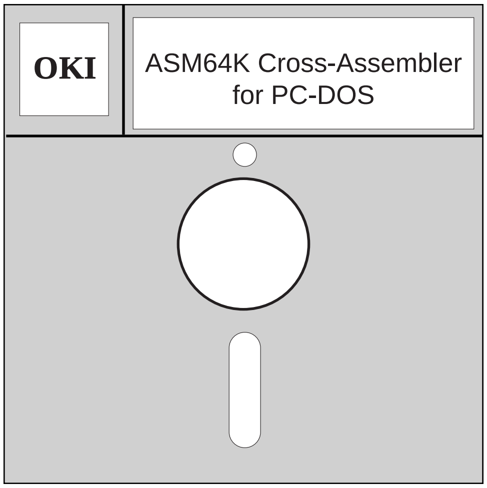
If the conditions described in Sections 0-2-1 and 0-2-2 are satisfied, then there will be no problem with your host computer model. Next, check the contents of the floppy disks.

(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.

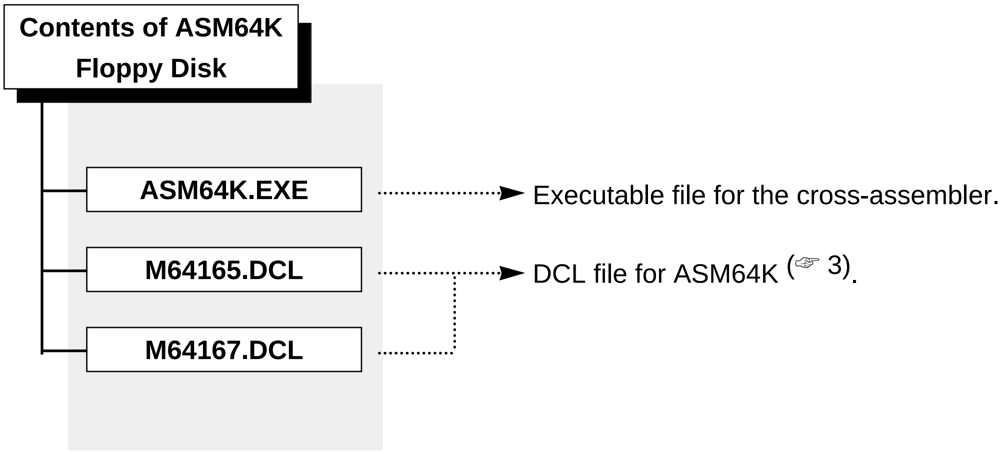


For PC9801/if800 Series



For IBM-PC

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.

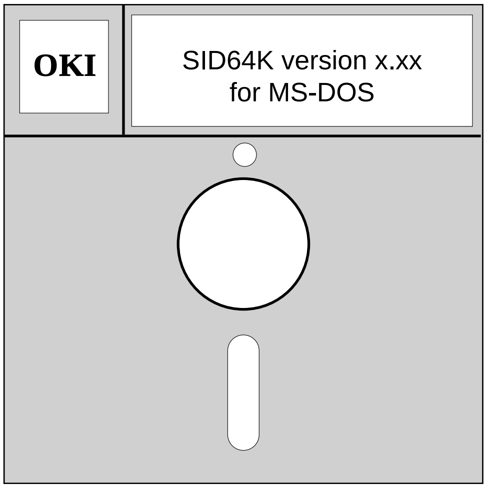


Executable file for the cross-assembler.

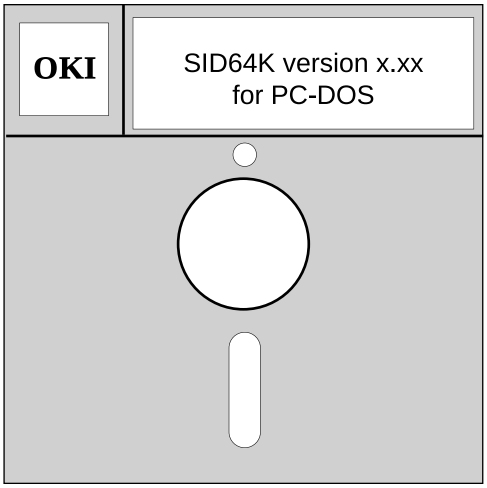
DCL file for ASM64K (☞ 3).

(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.

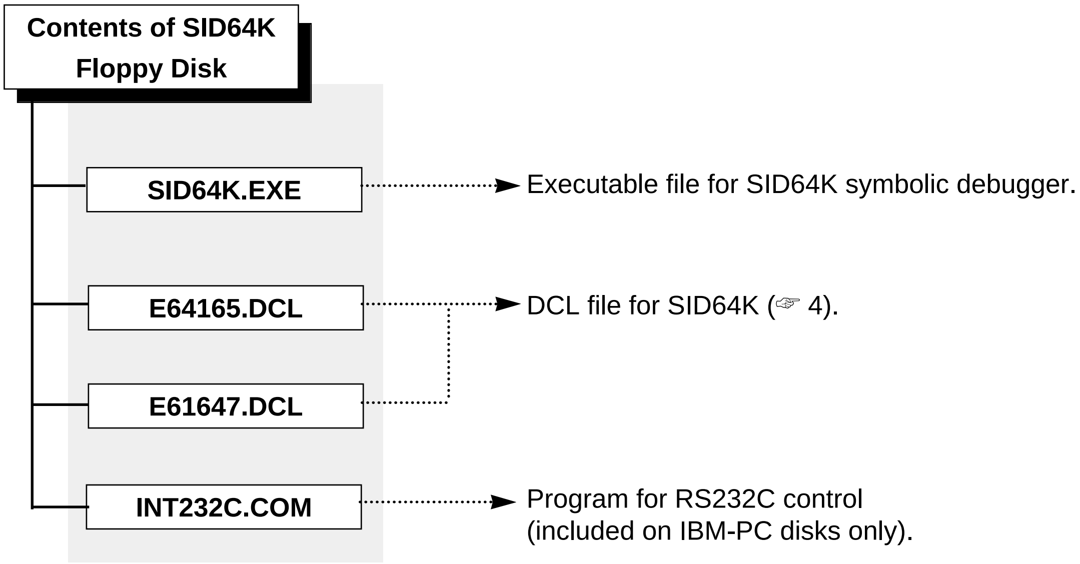


For PC9801/if800 Series



For IBM-PC

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.



Contents of SID64K Floppy Disk

SID64K.EXE → Executable file for SID64K symbolic debugger.

E64165.DCL, **E61647.DCL** → DCL file for SID64K (☞ 4).

INT232C.COM → Program for RS232C control (included on IBM-PC disks only).

☞ 3

The DCL file for ASM64K defines the following items to match operation with the appropriate member of MSM64165 and MSM64167.

(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 of MSM64165 and MSM64167. Note that the floppy disk contains DCl files for all devices supported by ASM64K.

MSM64165:	M64165.DCL
MSM64167:	M64167.DCL

☞ 4

The DCL file for SID64K defines the following items to match operation with the appropriate member of MSM64165 and MSM64167.

(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 of MSM64165 and MSM64167. Note that the floppy disk contains DCl files for all devices supported by SID64K.

MSM64165:	E64165.DCL
MSM64167:	E64167.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**64167.DCL (first character of the file name is "E")
DCL file for ASM64K:	**E**64167.DCL (first character of the file name is "M")

Chapter 1

Overview

This chapter provides an overview of EASE64165/167 program development support system configuration, describes the program development procedure with the EASE64165/167 system.

1-1. EASE64165/167 Emulator Configuration

The EASE64165/167 in-circuit emulator is configured from:

(1) Control system (EASE-LP2)
(2) POD64165/167 evaluation module
(3) ASM64K cross-assembler
(4) SID64K symbolic debugger
(5) Mask option generator

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

The EASE-LP2 is a general-purpose control system for in-circuit emulators for Oki Electric’s MSM64165/167 family of CMOS 4-bit microcontrollers. The EASE64165/167 in-circuit emulator is constructed by connecting the control system to a POD64165/167 evaluation module.

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

	• System controller	MC68HC000
☞1	• Code memory	64K x 8 bits
	• Trace memory	8K steps x 64 bits
	• Cycle counters	32-bit binary counter x 1
☞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 The maximum address of code memory, attribute memory, and instruction executed memory is 0FFFFH (64K bytes). However, in MSM64165 mode only addresses to 7DFH (2016 bytes) are valid, and in MS64167 mode only address to 0FDFH (4064 bytes) are valid. The valid addresses can be expanded to 7FFFH (32K bytes) in EXPAND mode.

! The emulator handles the test data area of the MSM64165/MSM64167 program as an unusable area.

1-1-2. POD64165/167 Evaluation Module

The EASE64165/167 in-circuit emulator for Oki Electric's MSM64165 and MSM64167 CMOS 4-bit microcontrollers is constructed by connecting the POD64165/167 evaluation module to an EASE-LP2. The POD64165/167 can also be used in evaluation mode without connecting it to a host computer (☞2).

The POD64165/167 employs a specially developed evaluation board for emulating (☞3).

☞ 2

The POD64165/167 can be used individually when an EPROM that contains the user program is inserted in the EPROM socket. For details, refer to Section 2-2-11, "Starting EASE64165/167 Emulator."

☞ 3

The evaluation board is a board internal to the POD64165/167 that emulates the functions of the MSM64165 and MSM64167. It is configured from a nx-4s core evaluation chip equivalent to the CPU core of the MSM64165 and MSM64167, input/output circuits equivalent to those of the MSM64165 and MSM64167 (other than the A/D converter), and an LCD driver equivalent to that of the MSM64165 and MSM64167.

The input/output circuits are constructed from ordinary discrete components, so the electrical characteristics of each port will differ from those of the MSM64165 and MSM64167. The LCD driver simulates the register assignments of the mask option, so the display timing will differ from the MSM64165 and MSM64167.

The MSM64165 and MSM64167 A/D converter is assigned to the accessory MSM64165/167 ADCPOD. An MSM64167 is mounted in the MSM64165/167 ADCPOD, so the A/D converter will have identical electrical characteristics to the MSM64167.

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 EASE64165/167 program 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 EASE64165/167. The EASE64165/167 operates through this software. SID64K also supports symbolic debugging.

SID64K is stored on a floppy disk that comes with the purchase of an EASE64165/167 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. MASK165/MASK167 Mask Option Generator

The MASK165 and MASK167 mask option generators are used by an operator to input the option settings shown below for the MSM64165 and MSM64167 respectively. The mask option generator converts the input data to an Intel HEX format mask option file.

Mask option settings:
LCD driver duty value
☞4 Assignment of each segment pin to port, common, or segment
Assignment of each segment pin to display register
Presence of X'tal oscillator capacitor

The mask option files created by MASK165 and MASK167 are used to generate the masks needed to manufacture MSM64165 and MSM64167.

If a mask option EPROM written from the mask option file is mounted in the POD64165/167's mask option EPROM socket, then EASE64165/167 will be unable to verify the following items.

Unverifiable items:

Assignment of segment pins to ports

Segment pins cannot be assigned to ports. To use segment pins as ports, use the user connector pins P30-P33 and P40-P43.

Presence of X'tal oscillator capacitor

The emulation kit cannot verify whether an X'tal oscillator capacitor is present or not.

☞ 4 MSM64165 has segment pins L0-L23. MSM64167 has segment pins L0-L30.

1-1-6. System Configuration

The EASE64165/167 system can be used in the following two modes.

* EASE-LP mode

In this mode, the system is used by connecting a host computer, EASE-LP2, and POD64165/167. This mode can be used for high-level debugging functions.

* EVA mode

In this mode, the POD64165/167 is used standalone. The user program mounted in the program EPROM socket will be will executed with continuous execution.

Figures 1-1 and 1-2 show the system configuration in each mode.

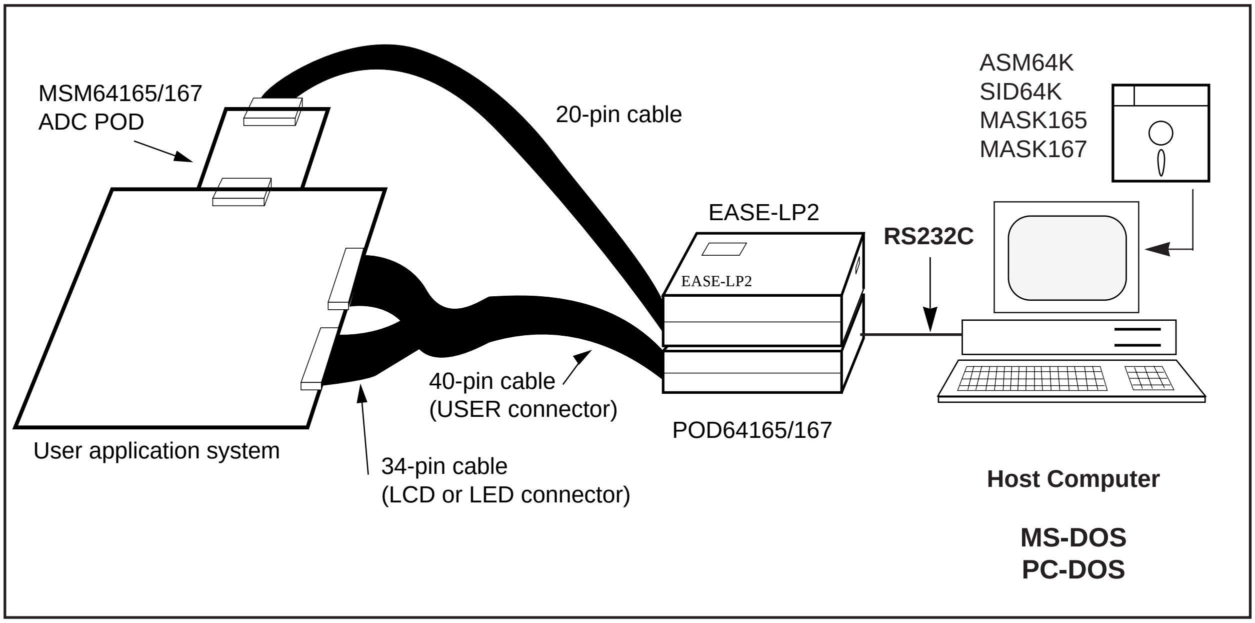


Figure 1-1. System Configuration Diagram In EASE-LP Mode

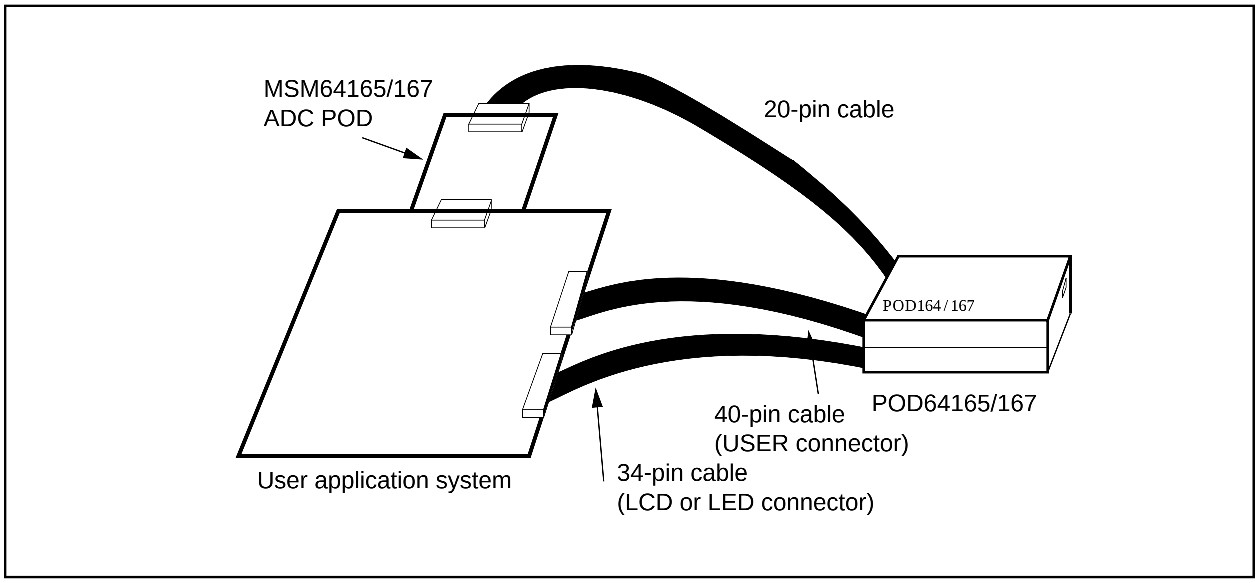


Figure 1-2. System Configuration Diagram In EVA Mode

1-2. EASE64165/167 Component Descriptions

This section provides basic explanations of the EASE64165/167 components.

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

(1) EPROM Programmer
The EPROM programmer is used to write code memory contents to EPROM, and to transfer EPROM contents to code memory.

(2) Indicators

POWER indicator	Lights when the power switch is ON.
RUN indicator	Lights during realtime emulation (continuous execution) and when the EPROM programmer is accessed.
ERROR indicator	Lights when the emulator is not operating correctly or when an error occurs during operation. Refer to Appendix 5.
POWER DOWN indicator	Lights when the emulator enters HALT mode during emulation (continuous or step execution).
POD indicator	Lights when the EASE-LP2 and POD64165/167 are connected correctly with the interface cable and power is applied.

(3) RS232C connector
The RS232C connector connects to the host computer with the accessory RS232C cable.

(4) DIP SW
These switches set the RS232C interface baud rate.

(5) Reset switch
This switch resets the EASE-LP2.

(6) Probe cable connector
The probe cable connector connects to the accessory probe cable for performing breaks on external signals.

(7) Interface cable connector
The interface cable connector connects to the POD64165/167 with the accessory interface cable (80-pin, 100-pin).

(8) Power supply connector
The power supply connector connects to the accessory power supply cable. Note especially that it is rated for AC 100-240 V.

(9) Power supply switch
This is the EASE64165/167 power supply switch.

1-2-2. POD64165/167 Evaluation Module

(1) Program EPROM socket
An EPROM written with a use program is mounted in the program EPROM socket.

(2) Mask option EPROM socket
An EPROM written with the contents of a mask option file is mounted in the mask option EPROM socket.

(3) Indicators
POWER indicator Lights when the power switch is ON.
RUN indicator Lights during realtime emulation (continuous execution).
ERROR indicator Lights when the emulator is not operating correctly or when an error occurs during operation. Refer to Appendix 5.
POWER DOWN indicator Lights when the emulator enters HALT mode during emulation (continuous or step execution).

(4) DIP SW2
These switches select between MSM64165 and MSM64167 mode, and switch the operating clock.

(5) MODE switch (1,2)
These switches select between EASE-LP mode and EVA mode.

(6) Reset switch
This switch resets the POD64165/167.

(7) CROSC board
The CROSC board generates MSM64165/167's CR oscillation (high-speed) clock.

(8) XT board
The XT board generates MSM64165/167's X'tal oscillation (low-speed) clock.

(9) Interface cable connector
The interface cable connector connects to the EASE-LP2 with the accessory interface cable (80-pin, 100-pin) when the system is used in EASE-LP mode.

(10) ADC connector
The ADC connector connects with MSM64165/167 ADC POD.

(11) USER connector
The USER connector connects to the user application system with the accessory user cable (40-pin).

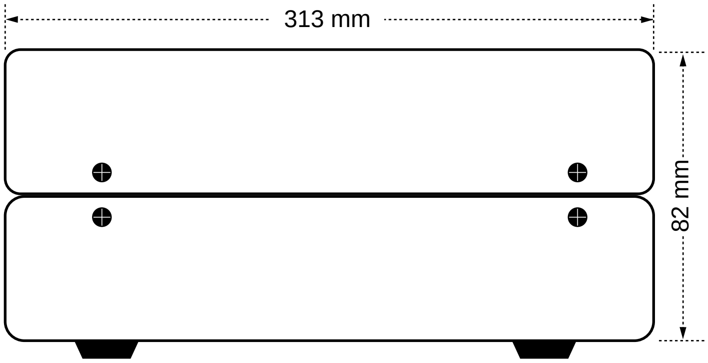
(12) VCC select switch
The VCC select switch selects whether the Vcc power supply for the USER connector interface is supplied from POD64165/167 internally or from the USER connector VCC pin.

(13) LCD connector
The LCD connector connects to the user application system with the accessory user cable (34-pin) when the user application system uses LCD.

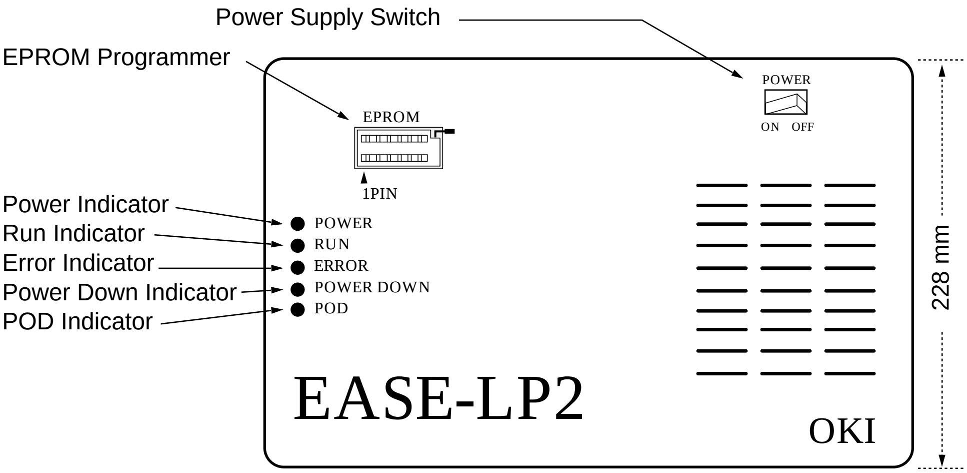
(14) LED connector
The LED connector connects to the user application system with the accessory user cable (34-pin) when the user application system uses LED.

(15) DC power jack
The DC power jack is a connector that supplies power to the POD64165/167 when the system is used in EVA mode. It supplies an external DC 5V (+/-5%) from the accessory DC power supply cable. Please take note of the polarity.

EASE-LP2 External Views (1)

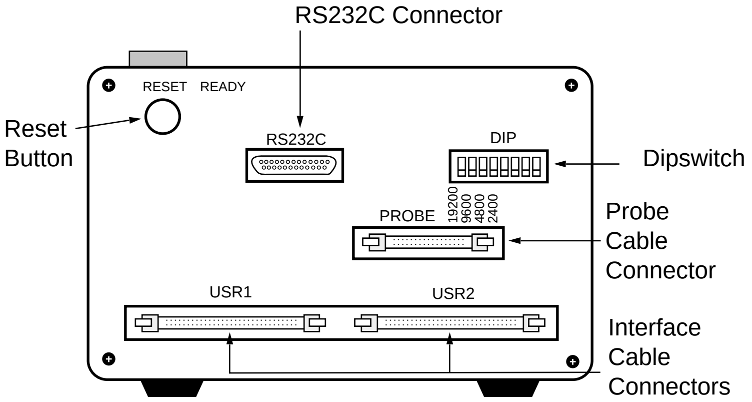


Front View

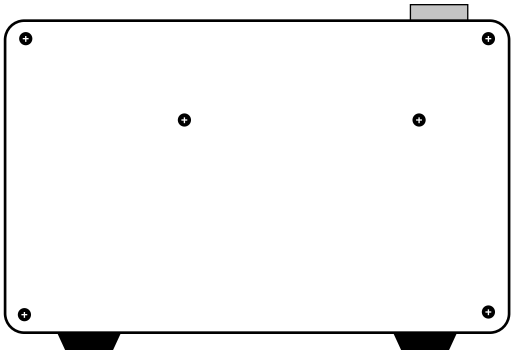


Top View

EASE-LP2 External Views (2)

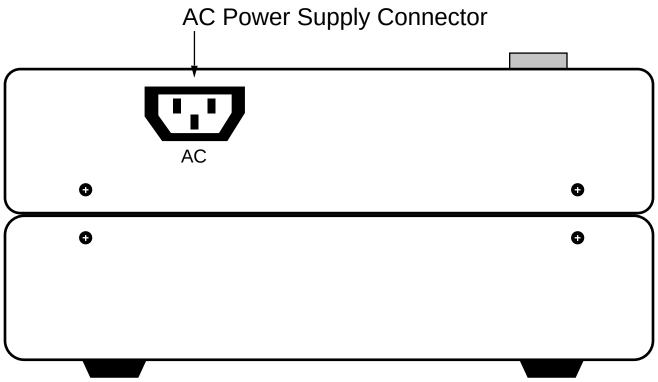


Left View



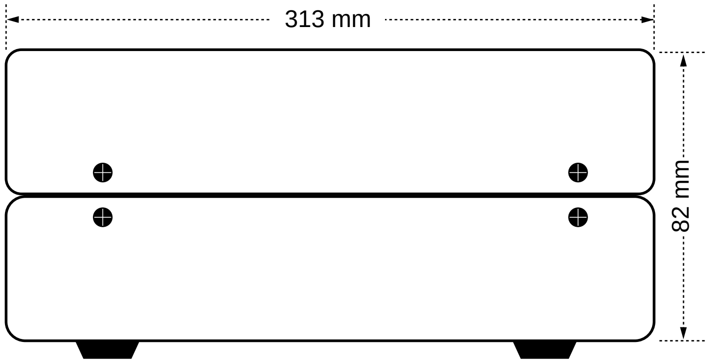
Right View

EASE-LP2 External Views (3)

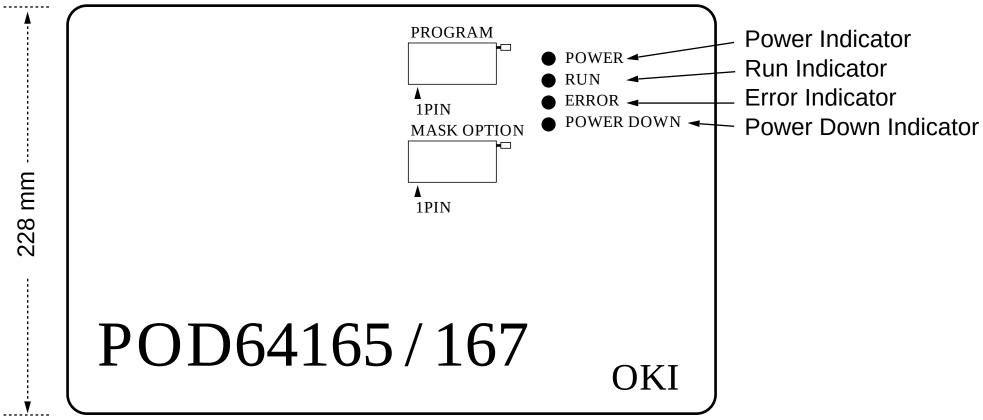


Rear View

POD64165/167 External Views (1)

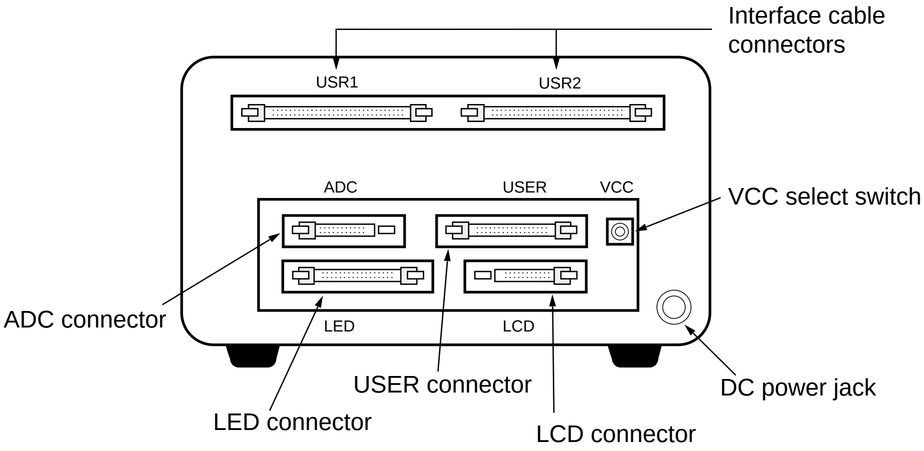


Front View

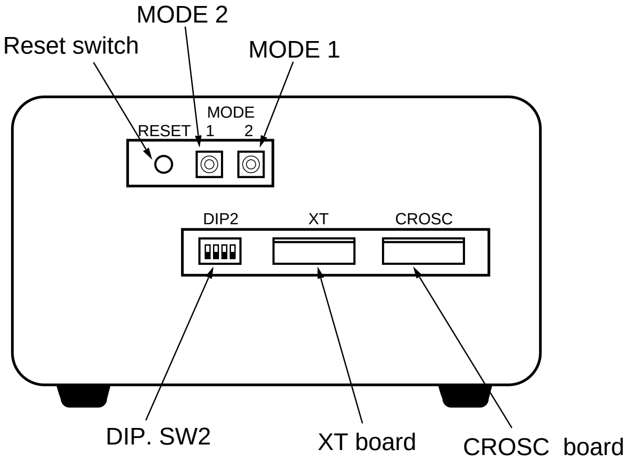


Top View

POD64165/167 External Views (2)

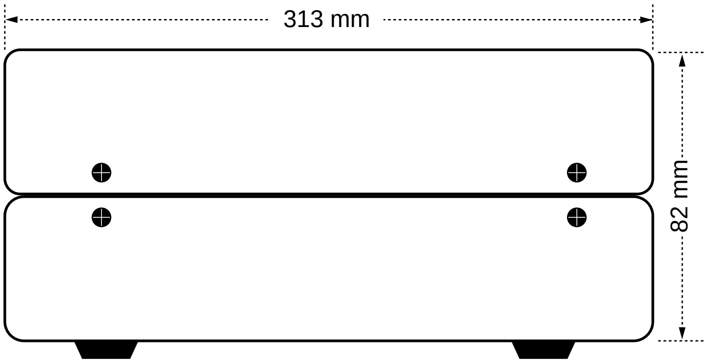


Left View



Right View

POD64165/167 External Views (3)



Rear View

1-3. Program Development With EASE64165/167

1-3-1. General Program Development and EASE64165/167

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. Generation of mask option files and mask option EPROMs with MASK165 or MASK167 is performed in this process as well.

Next is the *debug process*. This is the process for which the EASE64165/167 especially excels (☞2). An object file created in the program design process is downloaded to the EASE64165/167, and by using the various functions of the EASE64165/167 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.

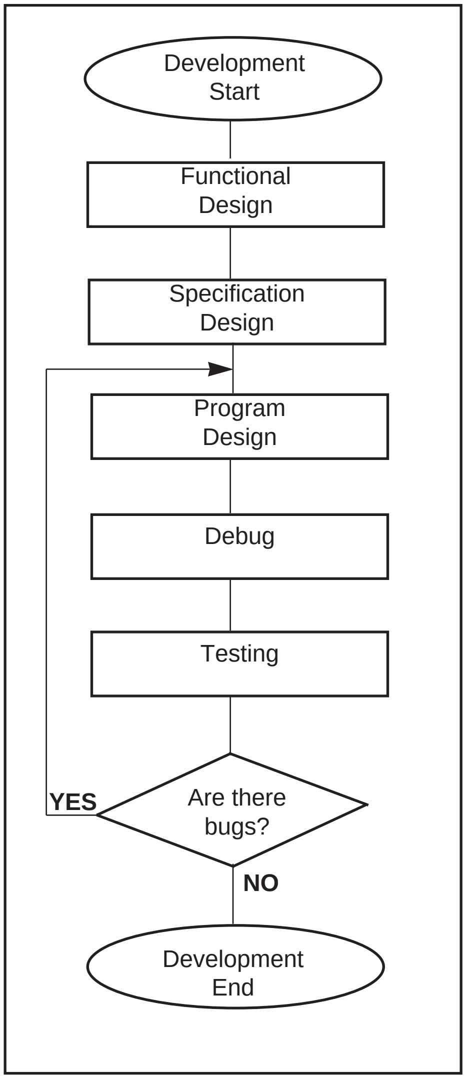


Figure 1-3. General Flow of Program Development

☞ 1	The general flow and terminology given here are typical, but other documents and manuals will have different expressions.
☞ 2	Refer to Chapter 2, “EASE64165/167 Emulator,” for details about the various function of the EASE64165/167 emulator.

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

In order to perform debugging with the EASE64165/167 emulator, an object file for downloaded to the EASE64165/167 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).

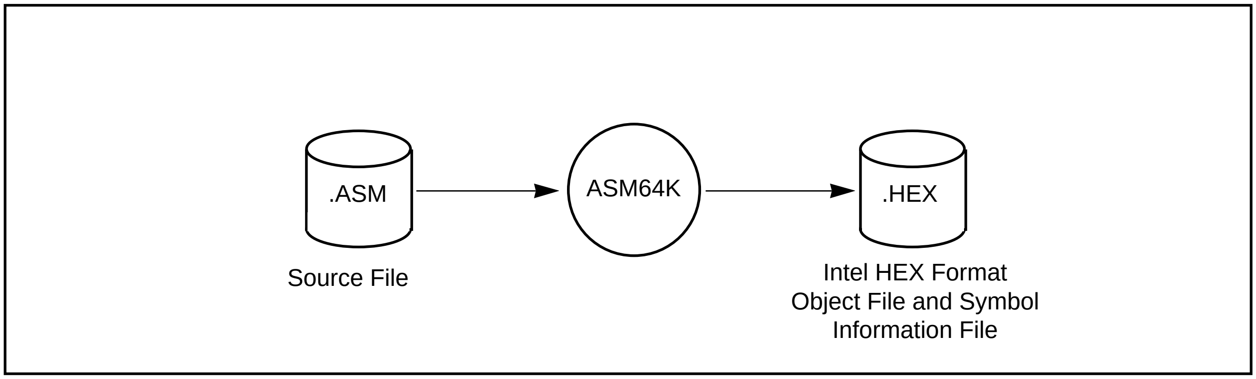


Figure 1-4. Process of Generating Object Files From Source Files

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

Object files that the EASE64165/167 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 EASE64165/167 code memory with the SID64K **LOD** command. Refer to Section 3-2-3, “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 EASE64165/167 emulator can handle.

1-3-3. Mask Option File and Mask Option EPROM Generation

In addition to the object files described above, an MSM64165 or MSM64167 mask option file and mask option EPROM must be created in order to perform debugging with the EASE64165/167 emulator. The mask option EPROM is then mounted in the POD64165/167's mask option EPROM socket.

Figure 1-5 shows the process for creating a mask option EPROM

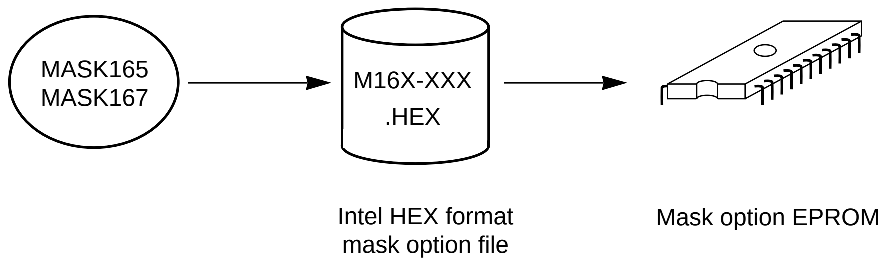


Figure 1-5. Creation Process for Mask Option EPROMs

The circle above indicates operation of the MASK165 or MASK167 program. The cylinder indicates the file generated by the program.

The generated mask option file is written to an EPROM to create a mask option EPROM. The following EPROM types can be used for a mask option EPROM:

27256, 27512, 27C256, 27C512

1-3-4. Files Usable With the EASE64165/167 Emulator

The files usable with the EASE64165/167 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 3-2-3, "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

EASE64165/167 Emulator

This chapter explains the actual use of the EASE64165/167 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 EASE64165/167 emulation kit and the SID64K symbolic debugger

Section 2-2 explains how to start the EASE64165/167. EASE64165/167 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 EASE64165/167.

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. EASE64165/167 Functions

2-1-1. Overview

Section 1-3 explained the program development process with the microcontrollers of the MSM64165/167 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 MSM64165 and MSM64167 family. This is known as *emulation*, as distinguished from program simulation with large computers. Here "execute" means to execute a program under the same electrical characteristics and at the same speed as the same volume-production MSM64165 or MSM64167 microcontroller.

This portion operates the same as an MSM64165 or MSM64167.

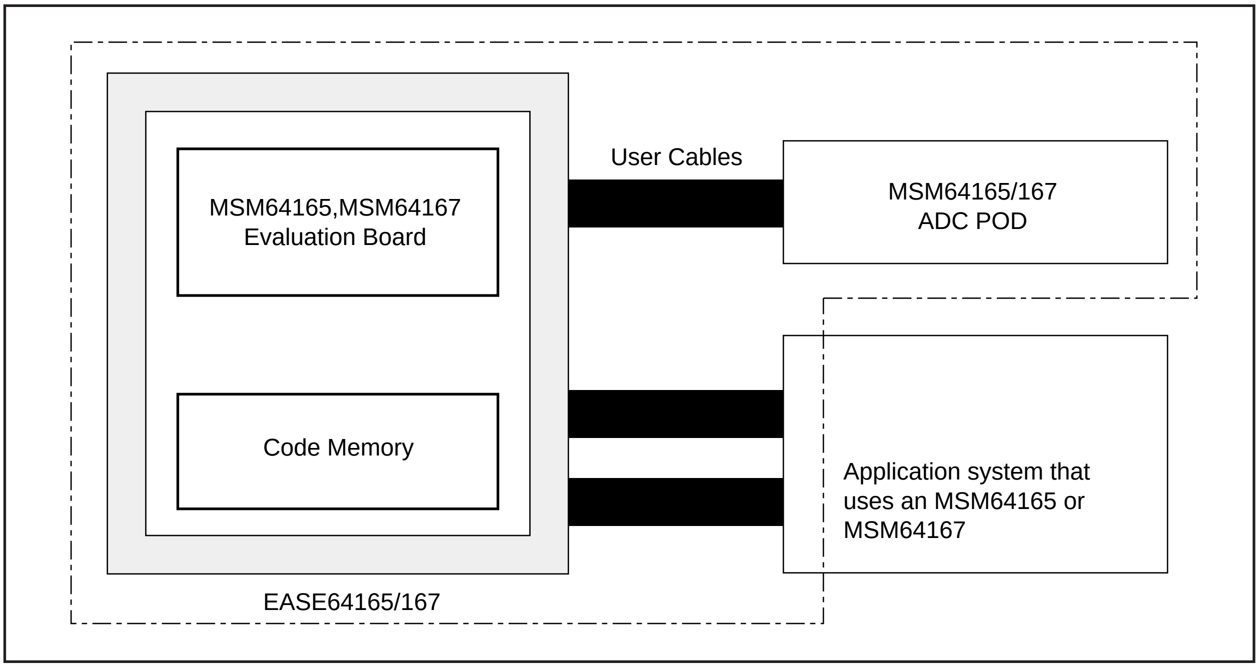


Figure 2-1

The volume-production MSM64165 and MSM64167 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 EASE64165/167 has in internal 32K x 8-bit program storage RAM. This RAM is called *code memory* (☞ 1). Refer to Figure 2-1 on the previous page.

EASE64165/167 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 MSM64165 or MSM64167 family microcontroller, but at the debug stage it is replaced with a connector in the user application system. This connector is attached to an EASE64165/167 user cable (Refer to Figure 2-1).

Within the EASE64165/167 (strictly speaking, within the POD64165/167) is an evaluation board that emulates MSM64165 and MSM64167 functions. This evaluation board has the same CPU circuit and the same external pins as the MSM64165 and MSM64167. It differs from the MSM64165 and MSM64167 in that it has no internal mask ROM, but it does have some special control circuitry and external control pins. In addition, the A/D converter is implemented in the MSM64165/167 ADC POD.

The particular features of the evaluation board are that it does not have an internal mask ROM, and it does have special control circuitry and external control pins. These additional circuits and pins are used to control execution of programs and reading of internal memory, registers, and flags. The EASE64165/167 can read and execute the contents of code memory instead of mask ROM.

The external pins of the evaluation board that are common with the volume-production MSM64165 and MSM64167 chips connect to the corresponding pins of the user application system through the user cables. The A/D converter pins come from the MSM64165/167 ADC POD.

As a result, the user application system sees the end of the user cable and the MSM64165/167 ADC POD as equivalent to the pins of an MSM64165 and MSM64167.

☞ 1 Refer to Table 2-1 regarding code memory addresses of the MSM64165 and MSM64167. Within code memory, up to 32K x 8 bits can be used as RAM for program storage.

☞ 2 The POD64165/167 has an EPROM socket for code memory. If the POD64165/167 is used standalone, then the EPROM in this socket will be allocated to the program area. However, if used as an EASE64165/167, then do not use the EPROM socket.

☞ 3

The evaluation board's CPU circuit is constructed from ordinary discrete components, so the electrical characteristics of each pin will differ from those of the MSM64165 and MSM64167. The MSM64165 and MSM64167 A/D converter is assigned to the accessory MSM64165/167 ADC POD. An MSM64167 is mounted in the MSM64165/167 ADC POD, so the A/D converter will have identical electrical characteristics to the MSM64167.

The evaluation board's LCD driver incorporates the mask option assignments of each register in order to simulate operation. Therefore, display timing will differ from the MSM64165 and MSM64167.

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, EASE64165/167 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 EASE64165/167 is an in-circuit emulator for the MSM64165 and MSM64167. The appropriate device is set with POD64165/167 dipswitches. SID64K will read the DCL file that corresponds to the setting of the dipswitches. The evaluation board will also switch to the appropriate device.

(a) POD64165/167 dipswitch settings

The number 1 position on dipswitch 2 on the right side of the POD64165/167 sets the device. If switched up, then MSM64165 mode will be selected; if switched down, then MSM64167 mode will be selected. The switch setting is read when SID64K is invoked. In EASE-LP mode, it is also read when the EASE-LP2 reset switch is pressed.

(b) DCL file

The DCL file defines symbol information needed to perform symbolic debugging, the code memory address range, and the data address range. The DCL file is read when SID64K is invoked. In EASE-LP mode, it is also read when the EASE-LP2 reset switch is pressed.

❗ As described above, the POD64165/167 dipswitch for device selection is used to select the DCL file read by SID64K and to switch the evaluation board. The DCL file is read when SID64K is invoked and when the EASE-LP2 reset switch is pressed. The evaluation board switch will become effective immediately after the device selection dipswitch is switched.

Therefore, whenever the device selection dipswitch is switched, the EASE-LP2 reset switch must be pressed.

❑ *Reading the DCL file*

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 the device selection dipswitch on the POD64165/167. When the file name 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.

❑ *EASE64165/167 Settings in MSM64165 and MSM64167 Mode*

EASE64165/167 will be set as follows, depending on the setting of POD64165/167's device selection dipswitch.

Table 2-1. Setting for Each Target Chip

	ITEM	*MSM64165 MODE*	*MSM64167 MODE*
(☞1)	**Code Memory Addresses**	000 ~ 7DFH	000 ~ 0FDFH
(☞1)	**Attribute Memory Addresses**	000 ~ 7DFH	000 ~ 0FDFH
(☞1)	**Instruction executed memory addresses**	000 ~ 7DFH	000 ~ 0FDFH
	Data memory	780 ~ 7FFH	700 ~ 7FFH
	LCD pins	L0 ~ L23	L0 ~ L30
	Timer	12-bit	16-bit
	Serial port	No	Yes
	Serial port interrupt	No	Yes

☞ **1** The size of code memory, attribute memory, and instruction executed memory can be expanded to 32K bytes (000-7FFFH) by setting EXPAND mode.

2-1-3. Emulation Functions

The EASE64165/167 has two modes for its emulation functions (program execution functions).☞2

(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 EASE64165/167 allows some of the debug commands to be input. For details, refer to Section 3-4-3, “Commands Usable During Emulation.”

SEE ▷ **G**

☞ 2 The emulation functions shown here are for EASE-LP mode and. In EVA mode, where POD64165/167 is operated standalone, only continuous execution from the user program EPROM mounted in the POD is possible.

❑ *Operating Clock*

The operating clock of the EASE64165/167 can be selected from either a clock supplied by an internal oscillation circuit or a clock input from the user cable. Operating clock selection is performed by switching a dipswitch on the POD64165/167.

When the EASE64165/167 is shipped, it is set to operate using the clock supplied by its internal oscillation circuit. The internal oscillation circuit's low-speed frequency is 32.768 kHz (typical). The high-speed frequency is approximately 546 kHz. To change the internal oscillation circuit's low-speed clock frequency, change the crystal on the POD64165/167's X'tal board. To change the internal oscillation circuit's high-speed clock frequency, change the CR oscillation resistor on the POD64165/167's CROSC board.

For details, refer to Section 2-2-3, "Setting Operating Clock Frequency."

!

- The EASE64165/167 can operate at frequencies 32.768 kHz to 700 kHz.
- The oscillation capacitor or resistor may have to be changed depending on the crystal's manufacturer and frequency.
- The CROSC board contains a MSM64167 and implements CR oscillation. Refer to the "MSM64167 User's Manual" regarding changes to the CR oscillation resistor.

2-1-4. Realtime Trace Functions

One EASE64165/167'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 EASE64165/167 has trace memory for 8K steps. It traces the following items. The EASE64165/167 has trace memory for 8K steps (instructions). 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
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, "EASE64165/167 Timing," for output timing of the INT flag and SKIP flag.

❑ *Controlling trace execution*

Realtime tracing is controlled in the following ways.

a. Free-running trace
Tracing is always performede during program execution.

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

c. Trace disable
Tracing is not performed during program execution.

d. Trigger-based trace start/stop
Tracing starts when the trace start address is executed, and stops when the trace stop address is executed.

e. Data match post-trace
Tracing starts when a probe or RAM value matches the specified value.

f. Data match pre-trace
Tracing ends when a probe or RAM value matches the specified value.

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 executed under the control conditions (refer to Figure 2-2).

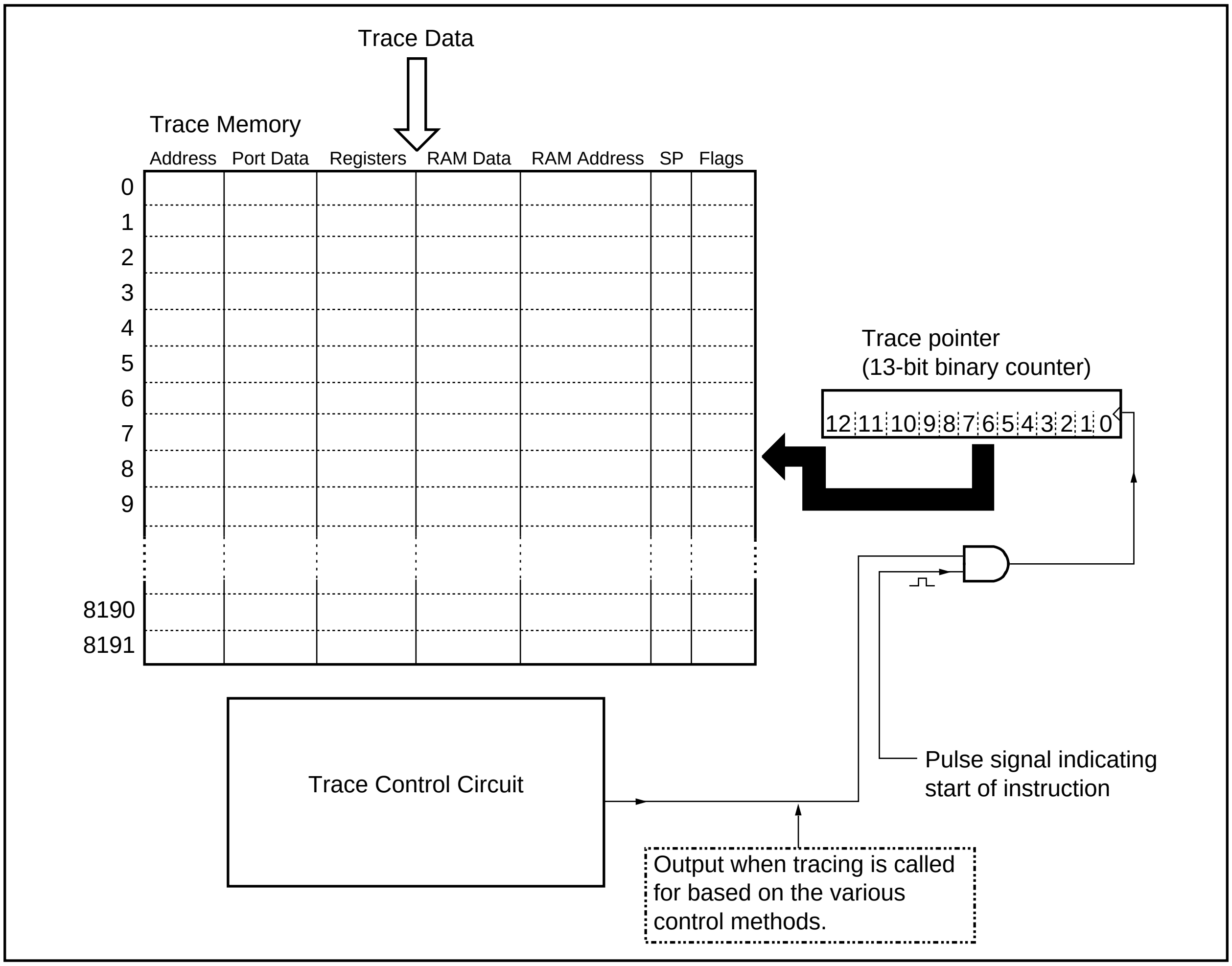


Figure 2-2. Trace Control Conceptual Diagram

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 as long as the previously described trace control methods are effective. 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.

2-1-5. Break Functions

The following methods for breaking program execution are available with the EASE64165/167.

(a) Breakpoint bit breaks

The EASE64165/167 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*.

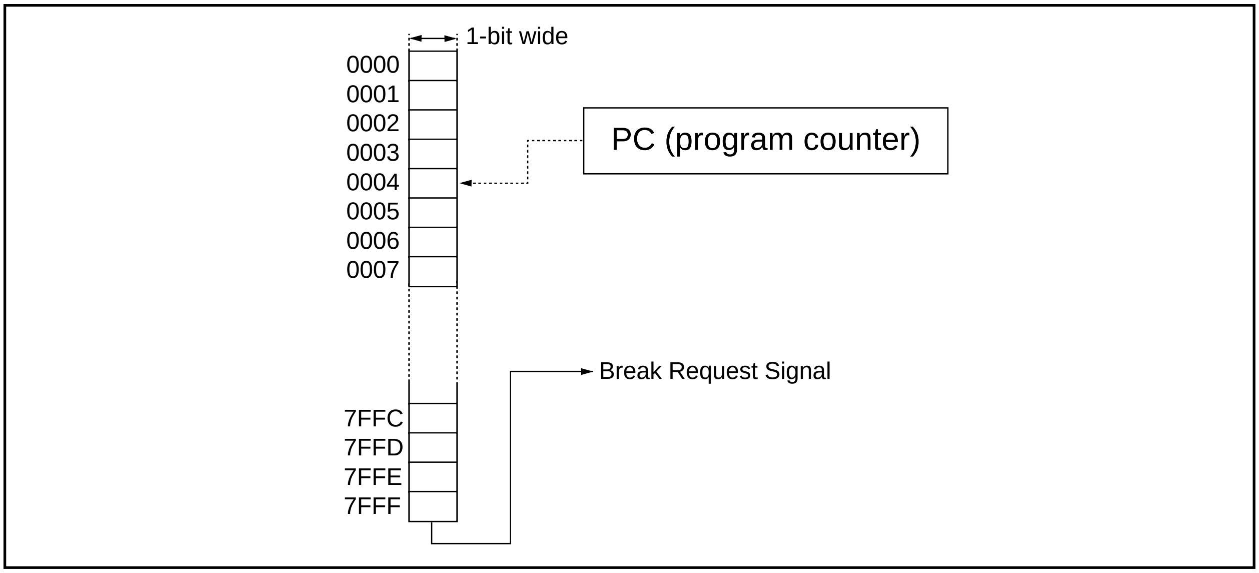


Figure 2-3. Breakpoint Bits Conceptual Diagram

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 EASE64165/167 can force a break using overflow of the trace pointer.

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

(c) Cycle counter overflow breaks

The EASE64165/167 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 Address pass counter overflow breaks

The EASE64165/167 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 MSM64165/167 family. In other words, it occurs when MSM64165 or MSM64167 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 EASE64165/167 will forcibly break when it accesses an area that exceeds the maximum address for its respective chip modes.

(i) External break

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 2-4, masking of break conditions is performed using a register called the *break condition register*.

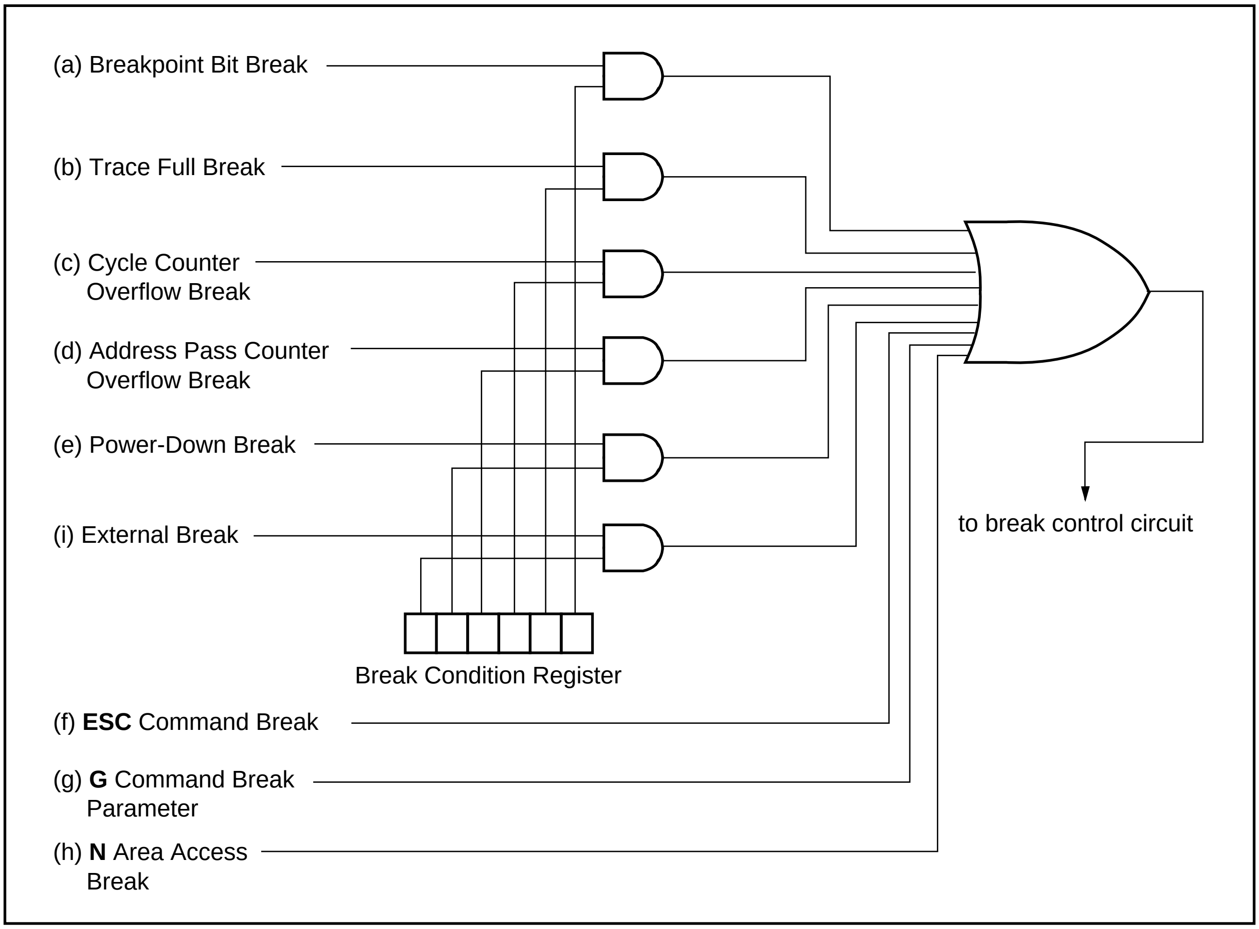

(a) Breakpoint Bit Break
(b) Trace Full Break
(c) Cycle Counter Overflow Break
(d) Address Pass Counter Overflow Break
(e) Power-Down Break
(i) External Break
Break Condition Register
(f) **ESC** Command Break
(g) **G** Command Break Parameter
(h) **N** Area Access Break
to break control circuit

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.

2-1-6. Performance/Coverage Functions

The EASE64165/167 has the following performance/coverage functions.

(a) Check for program areas not yet passed

The EASE64165/167 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 EASE64165/167 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 EASE64165/167's four 16-bit address pass counters (AP).

SEE ▷ **CAP, DAP**

2-1-7. Probe Cable Functions

The EASE64165/167 utilizes a probe cable with nine probe pins. The probe cable is connected to the EASE-LP2 probe connector. Refer to Appendix 7, "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 2-1-5, "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 2-1-4, “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 2-1-5, “Break Functions.”

2-1-8. EPROM Programmer

The EASE64165/167 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). However, in POD mode the EPROM programmer cannot be used.

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 8, “Mounting EASE-LP2 EPROMs,” for information about how to handle EPROMs.

2-1-9. 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.”

2-1-10. 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 **C** language compatible 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. EASE64165/167 Emulator Initialization

2-2-1. Setting Operating Mode

The EASE64165/167 can be used in the following two operating modes. This section explains how to set each operating mode.

- EASE-LP mode

In this mode, the system is used by connecting a host computer, EASE-LP2, and POD64165/167. This mode can be used for high-level debugging functions.

- EVA mode

In this mode, the POD64165/167 is used standalone. The user program mounted in the program EPROM socket will be executed with continuous execution.

The mode is set with the mode switches (1,2) on the right side of the POD64165/167. Figure 2-5 shows the mode switches, and Table 2-2 shows the mode switch settings.

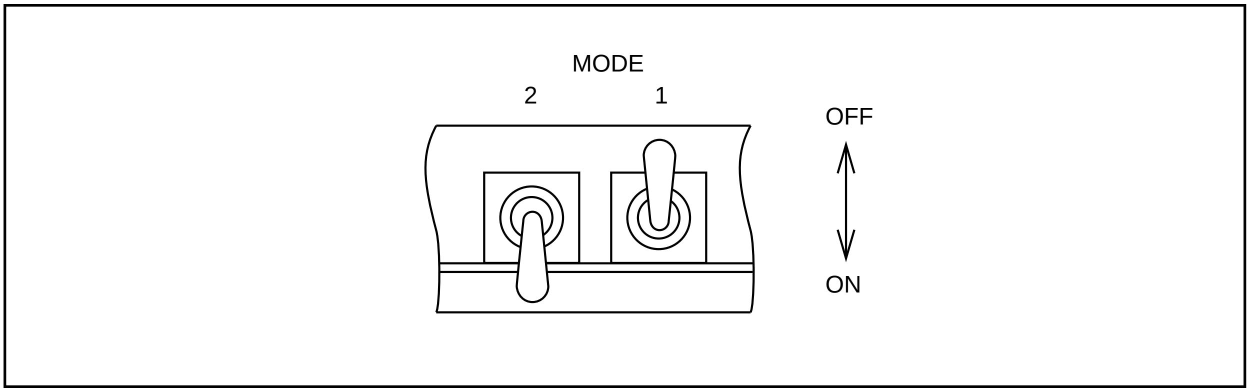


Figure 2-5. Mode Switches (settings when shipped)

Table 2-2. Mode Switch Settings

Operating Mode	**Mode 2**	**Mode 1**
EASE-LP mode	ON	OFF
EVA mode	ON	ON

2-2-2. Setting Microcontroller Type

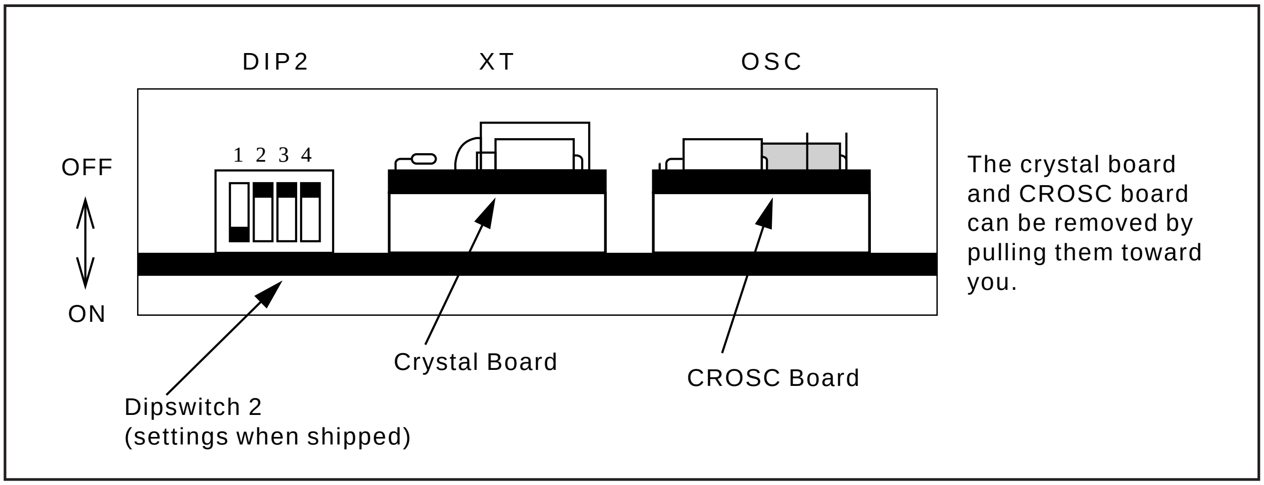
The EASE64165/167 allows program development for two types of microcontrollers, the MSM64165 and the MSM64167. The type is set with the number 1 switch of dipswitch 2 on the right side of the POD64165/167 (refer to Figure 2-6). Table 2-3 shows the microcontroller type switch settings, and Table 2-4 shows the differences between MSM64165 and MSM64167.

Table 2-3. Microcontroller Type Switch Settings

Dipswitch 2, No. 1	Microcontroller Type
Off (up)	MSM64165
On (down)	MSM64167

Table 2-4. MSM64165 and MSM64167 Differences

Item	MSM64167	MSM64165
Code memory	4064 x 8 bits	2016 x 8 bits
Data memory	256 x 4 bits	128 x 4 bits
LCD pins	L0-L30	L0-L23
Timer	16 bits	12 bits
Serial port	Yes	No
Serial port interrupt	Yes	No



The crystal board and CROSC board can be removed by pulling them toward you.

Figure 2-6. Crystal Board, CROSC Board, and Dipswitch 2

2-2-3. Setting Operating Frequency

As explained in Section 2-1-3, the EASE64165/167 operates with the low-speed 32.768-kHz (typical) clock and high-speed 546-kHz (approximate) clock supplied from the POD64165/167's internal oscillation circuit when it is shipped. Oki Electric normally recommends that the EASE64165/167 be used as it is with this setting.

The clock setting can be changed with the following two methods.

- Change the oscillation clock of the crystal board or CROSC board on the POD64165/167.
- Input a clock from the user connector XT pin or OSC pin.

Selection of the clock from the POD64165/167's internal clock or the user connector pins is performed with the number 3 and 4 switches of dipswitch 2 on the right side of the POD64165/167 (refer to Figure 2-6). Table 2-5 shows the high-speed clock switch settings, and Table 2-6 shows the low-speed clock switch settings.

Table 2-5. High-Speed Clock Switch Settings

Dipswitch 2, No. 3	High-Speed Clock Supply
Off (up)	POD64165/167 internal clock (CROSC board)
On (down)	User connector OSC pin

Table 2-6. Low-Speed Clock Switch Settings

Dipswitch 2, No. 4	High-Speed Clock Supply
Off (up)	POD64165/167 internal clock (crystal board)
On (down)	User connector XT pin

The POD64165/167 crystal board and CROSC board, and the user connector pins, are explained next.

(1) Crystal Board

The crystal board is mounted in the right side of the POD64165/167. It generates a 32.768-kHz low-speed clock.

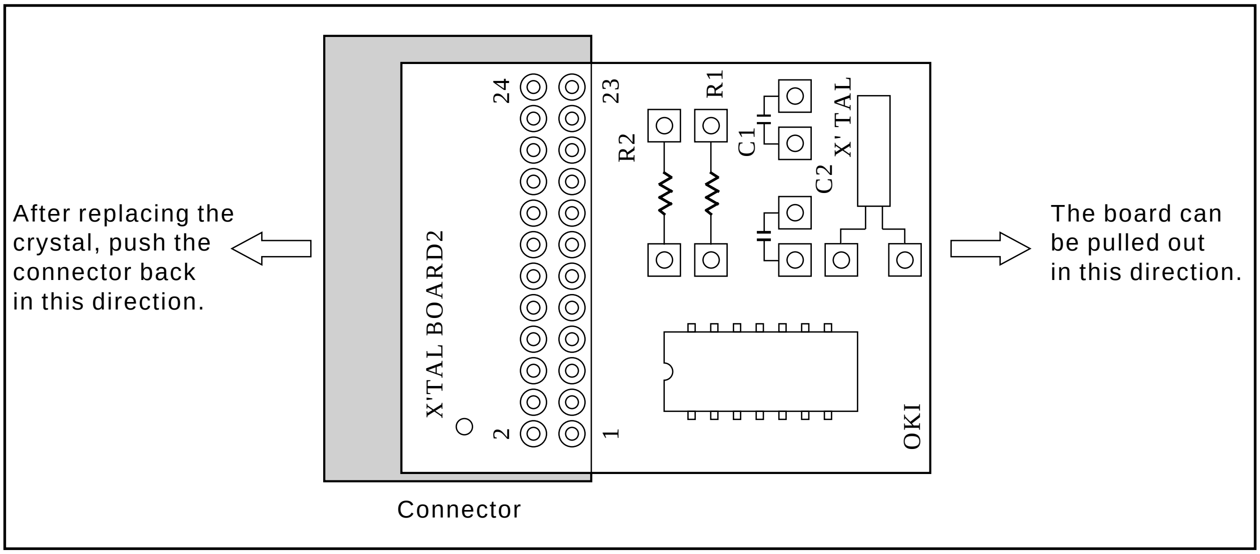
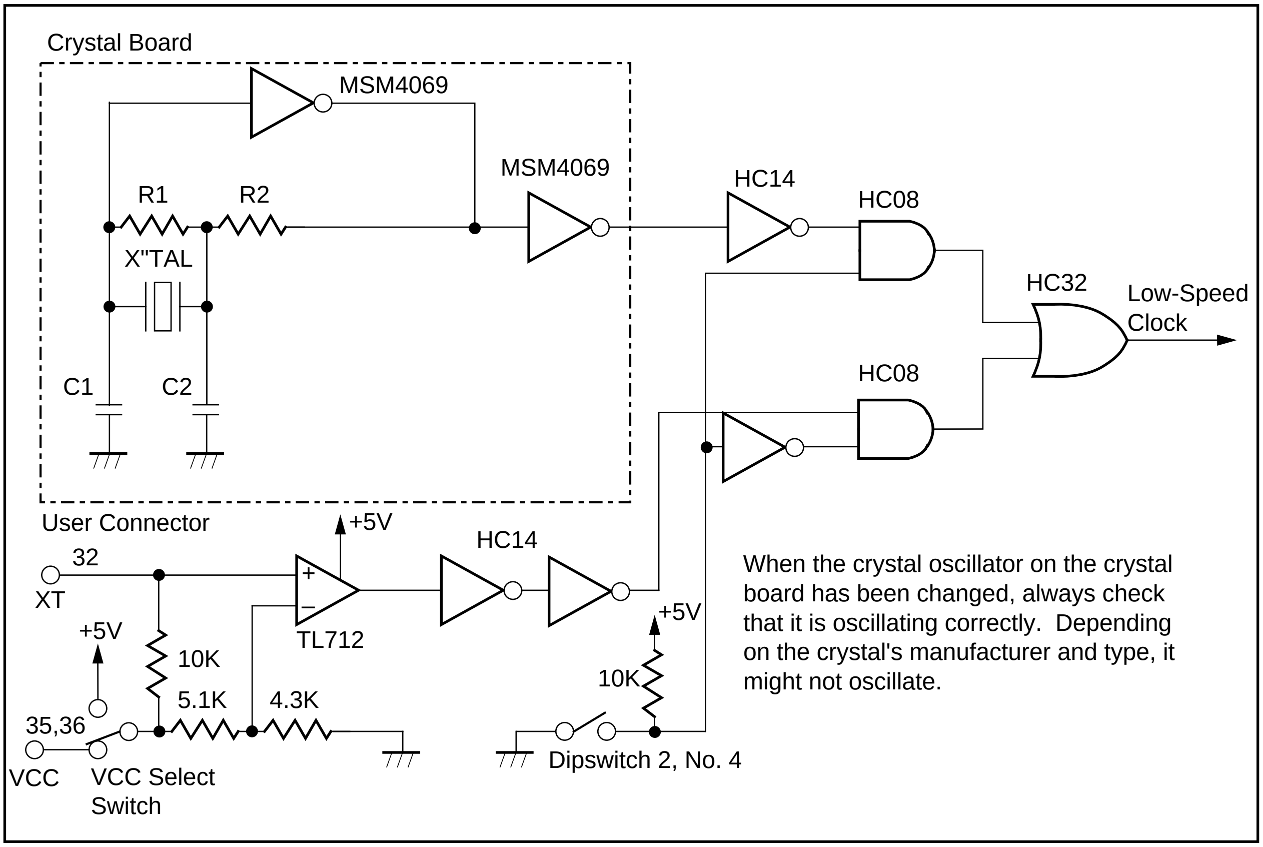

After replacing the crystal, push the connector back in this direction.
X'TAL BOARD2
24
2
23
1
R2
R1
C1
C2
X'TAL
OKI
The board can be pulled out in this direction.
Connector

Figure 2-7. 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 oscillate.

Figure 2-8. Low-Speed Clock Circuit

(2) CROSC Board

The CROSC board is mounted in the right side of the POD64165/167. It generates an approximately 546-kHz high-speed clock.

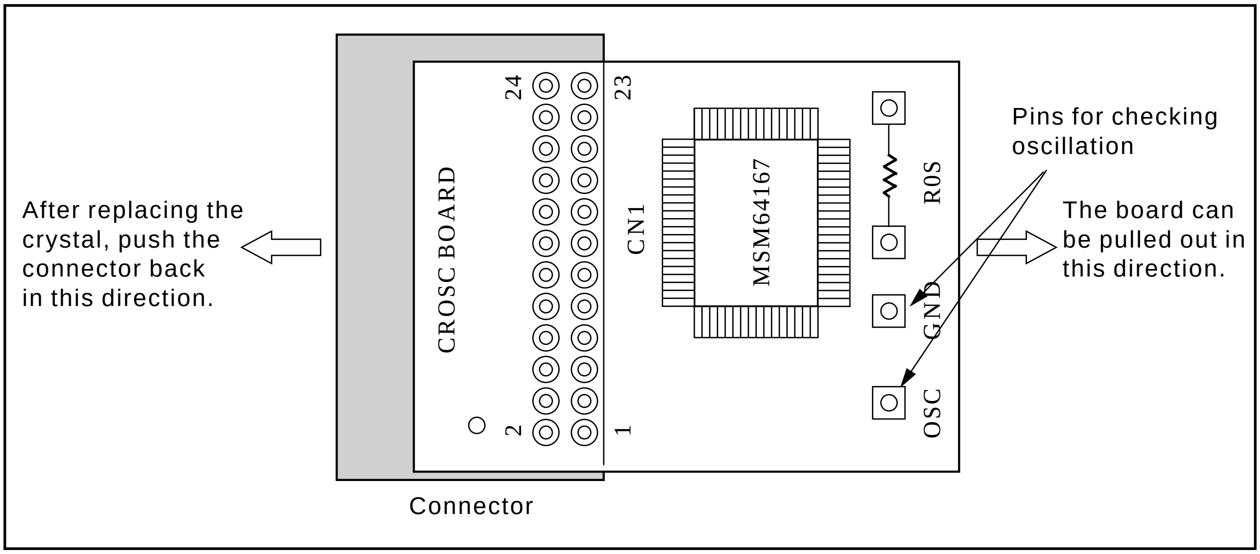

After replacing the crystal, push the connector back in this direction.
CROSC BOARD
24
23
2
1
CN1
MSM64167
R0S
GND
OSC
Pins for checking oscillation
The board can be pulled out in this direction.
Connector

Figure 2-9. CROSC Board

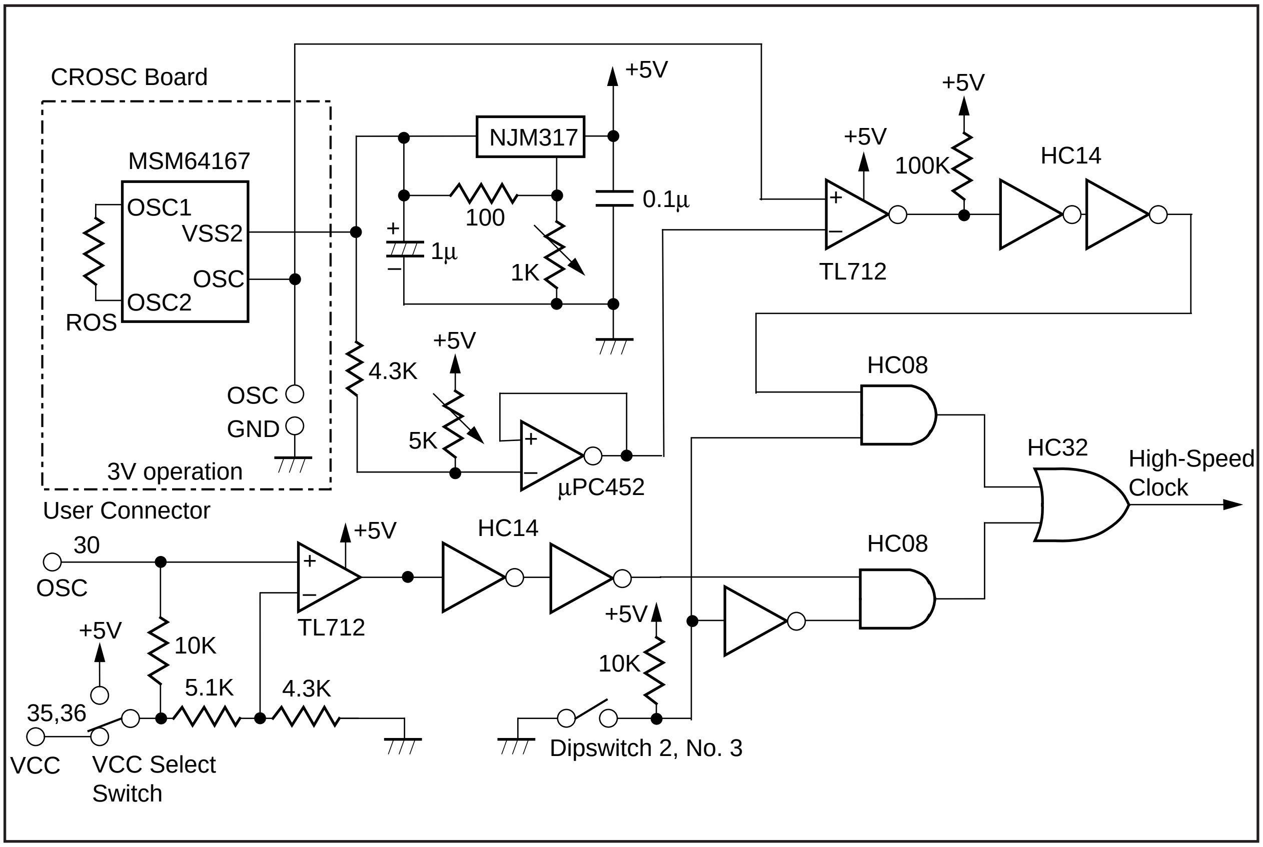


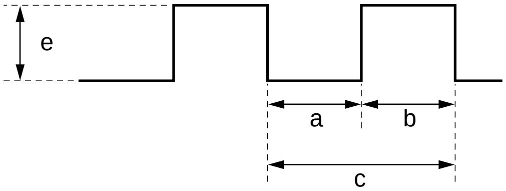
Figure 2-10. High-Speed Clock Circuit

The CROSC board has an MSM64167 mounted and performs CR oscillation. When the CROSC board's CR oscillation resistor (ROS) has been changed, always verify that it is oscillating correctly. Refer to the "MSM64167 User's Manual" for the value of the CR oscillation resistor.

(3) User Connector XT and OSC Pins

A low-speed clock can be input from the user connector XT pin (pin 32). A high-speed clock can be input from the user connector OSC pin (pin 30). (Refer to Figures 2-8 and 2-10.)

Use a signal like that shown below for inputting a clock on the XT pin or OSC pin.



Duty ratio a: b = 1:1
Frequency c = operating frequency
Voltage e= 3-5 V (☞ 1)

The input clock can utilize a pulse generator output clock or an oscillator circuit clock from the user application system. When utilizing a pulse generator output clock, the clock will be input to a TL712 as shown in Figures 2-8 and 2-10, so match its impedance to the TL712.

When using a clock from the user connector XT or OSC pin, always verify that it is oscillating correctly.

☞ 1

The voltage level of the clock supplied from the XT pin or OSC pin will differ depending on the VCC select switch setting.

- When VCC select switch is off, match the voltage given on the user connector Vcc pin (3 - 5V).
- When VCC select switch is on, match the emulator kit's internal voltage (5V).

2-2-4. Setting Reset Input

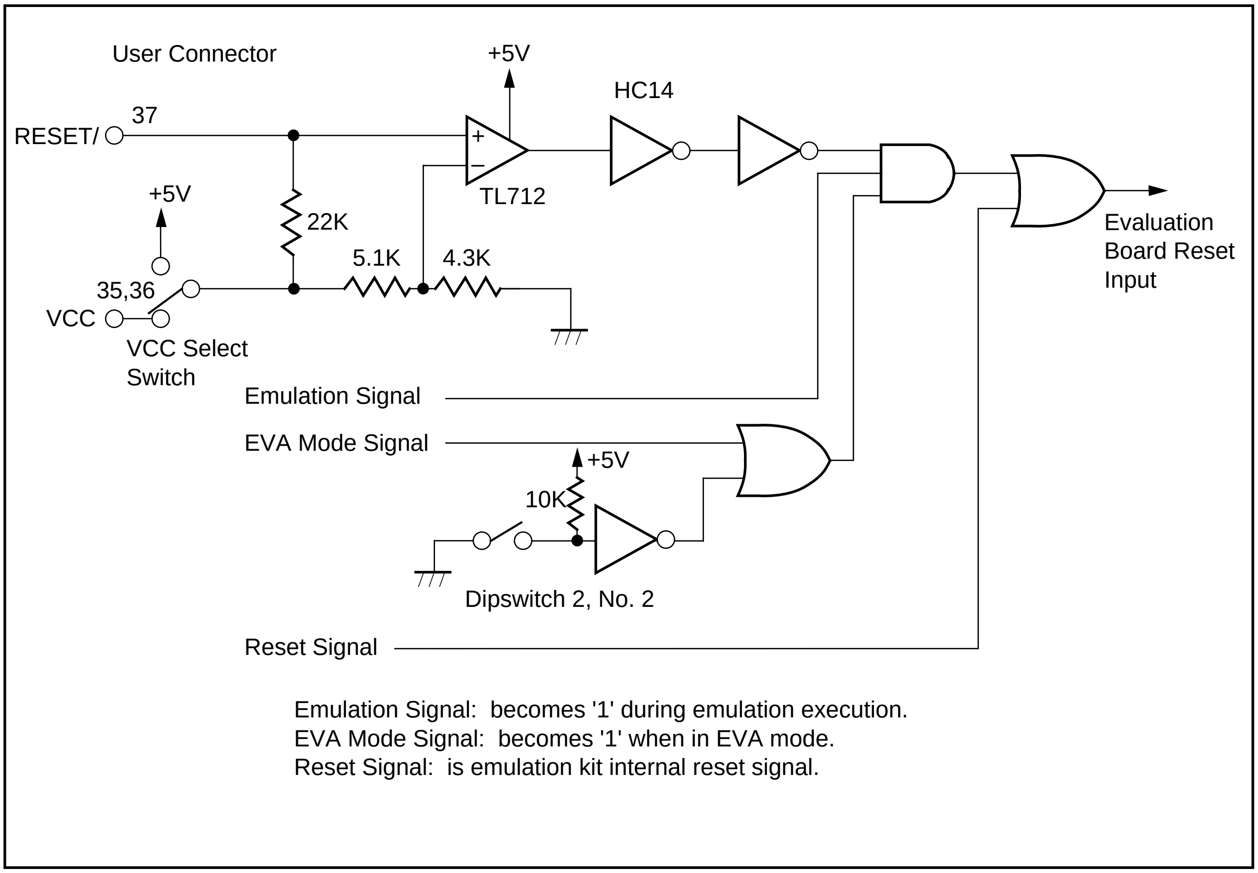
The reset input of the EASE64165/167's internal evaluation board is input from the emulation kit's internal reset signal when the system is shipped. By changing the number 2 switch of dipswitch 2 of the POD64165/167, the evaluation board's reset input can also be input from the user connector RESET/ pin.

In EASE-LP mode, a reset signal input from the user connector RESET/ pin will be valid only during a G command or STP command. In EVA mode, it will always be valid (refer to Figure 2-12).

Table 2-7 shows the reset input switch settings.

Table2-7. Reset Input Switch Settings

Dipswitch 2, No. 3	User Connector RESET/ Pin Reset Signal	
	EASE-LP mode	EVA mode
Off (up)	Not valid	Valid
On (down)	Valid	Valid



Emulation Signal: becomes '1' during emulation execution.
EVA Mode Signal: becomes '1' when in EVA mode.
Reset Signal: is emulation kit internal reset signal.

Figure 2-11. Reset Input Circuit

2-2-5. VCC Select Switch

The VCC select switch selects whether the interface power of the user connector pins is supplied internally from the POD64165/167 or supplied from the user connector VCC pins.

Table 2-8 shows the VCC select switch settings. Figure 2-12 shows a drawing of the VCC select switch, and Figure 2-13 shows the peripheral circuit diagram of the VCC select switch.

Table 2-8. VCC Select Switch Settings

VCC Select Switch	Interface Power Supply
On (down)	Supply (3-5V) from the user connector VCC pins (pin 35 and 36).
Off (up)	Supply internally from POD64165/167.

! The rated input voltage range for the user connector VCC pins is DC 3-5V. Input of any other voltage range will cause the device to malfunction.

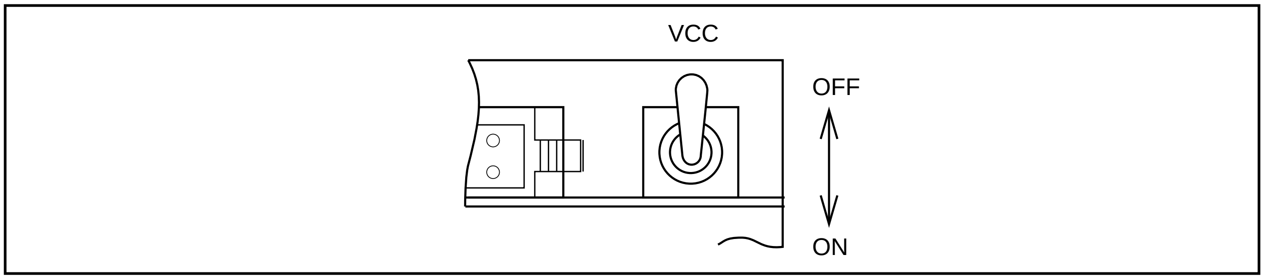


Figure 2-12. VCC Select Switch

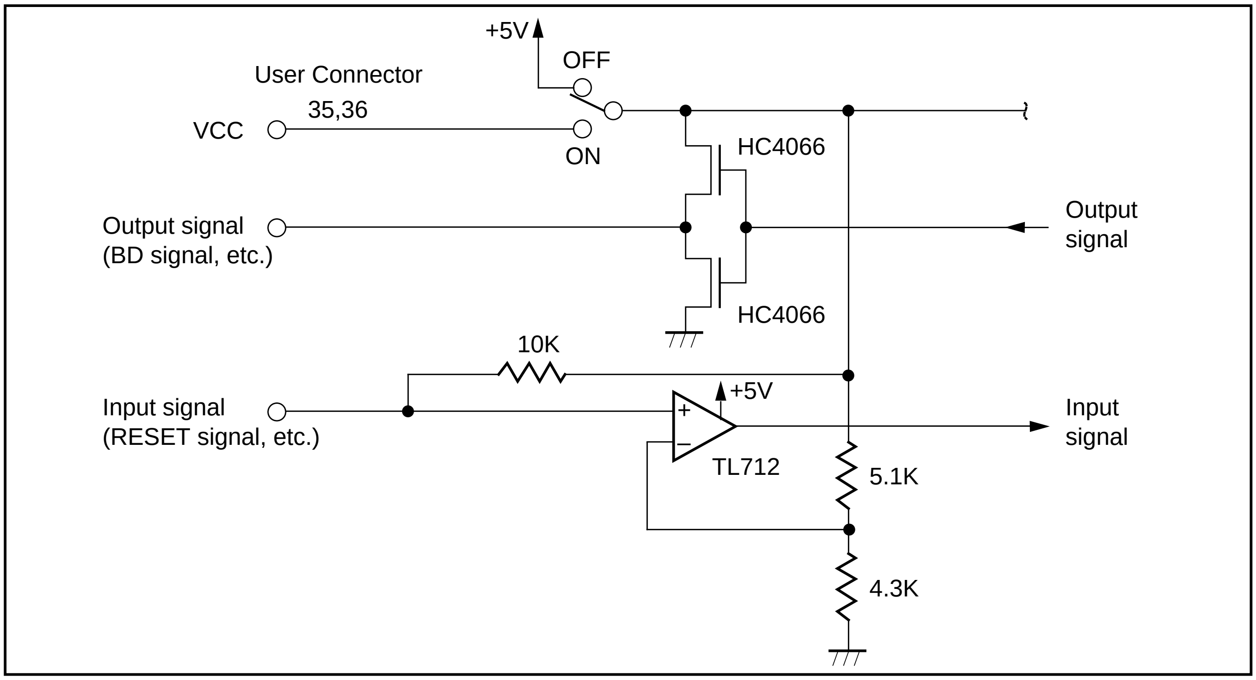


Figure 2-13. VCC Select Switch Peripheral Circuit

2-2-6. Mounting Mask Option EPROM

The EASE64165/167 can verify the mask options listed below by mounting in the POD64165/167 a mask option EPROM written with a mask option file created by the MASK165 or MASK167 mask option generator. A mask option EPROM needs to be mounted in all modes: EASE-LP mode and EVA mode.

Verifiable mask options
- LCD driver duty value
- Assignment of segment pins to common and segments
- Assignment of segment pins to display registers

Unverifiable mask options
- Presence of X'tal oscillator capacitor
- Assignment of segment pins to ports

Segment pins cannot be assigned to ports. To use segment pins as ports, use the user connector pins P30-P33 and P40-43.

The procedure for creating and mounting a mask option EPROM is given below.

(1) <u>Create a mask option file</u>

Start the MASK165 or MASK167 mask option generator, input the required items, and generate a mask option file. For details on the mask option generators, refer to the "MASK165 User's Manual" or the "MASK167 User's Manual."

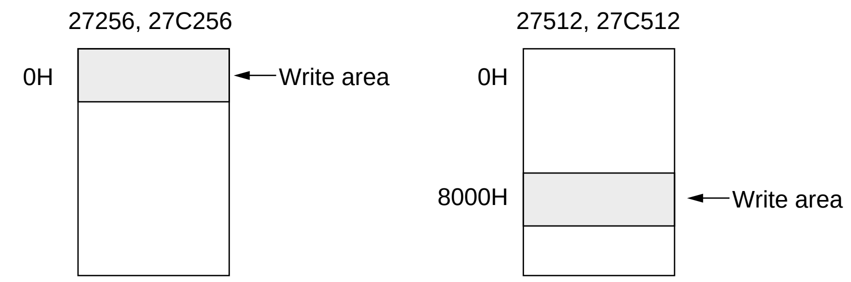
(2) <u>Create a mask option EPROM</u>

Write the generated mask option file to an EPROM to create the mask option EPROM.

Usable EPROM types:

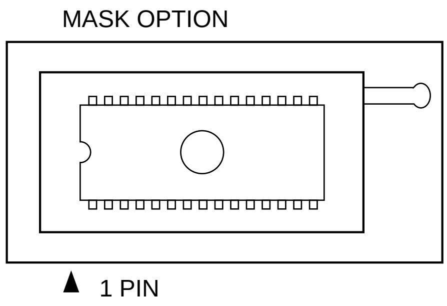
27256, 27512, 27C256, 27C512

Mask option file write areas:



(3) Mount the mask option EPROM

Mount the mask option EPROM in the POD64165/167 mask option EPROM socket. Make sure the power supply is off when mounting the EPROM.



Mount the EPROM when the power supply is off.

Figure 2-14. Mounting Mask Option EPROM

! When the mask option EPROM is not mounted or when mask options are not specified, changing display register (**C** command or **CDM** command) cannot be executed.

2-2-7. Mounting User Program EPROM

The EASE64165/167 can operate in EVA mode. EVA mode is continuous execution of the user program with the POD64165/167 standalone. This requires that an EPROM written with the user program be mounted in the POD64165/157 program EPROM socket.

The procedure for creating and mounting a user program EPROM is given below.

(1) Create an object file

Assemble a source program coded in assembly language with the ASM64K cross-assembler and generate an object file. For details on the ASM64K cross-assembler, refer to the "ASM64K Cross-Assembler User's Manual."

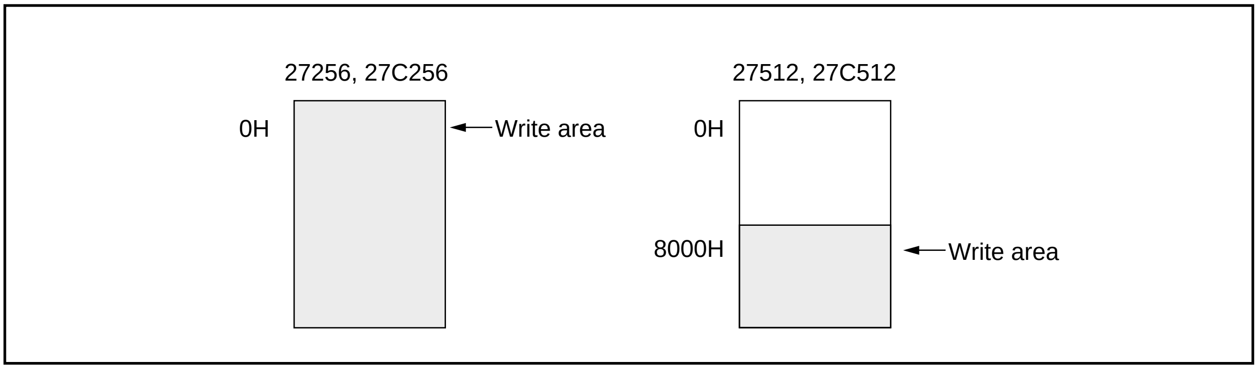
(2) Create a program EPROM

Write the generated object file to an EPROM to create the user program EPROM.

Usable EPROM types:

27256, 27512, 27C256, 27C512

Object file write areas:



(3) Mount the user program EPROM

Mount the user program EPROM in the POD64165/167 program EPROM socket. Make sure the power supply is off when mounting the EPROM.

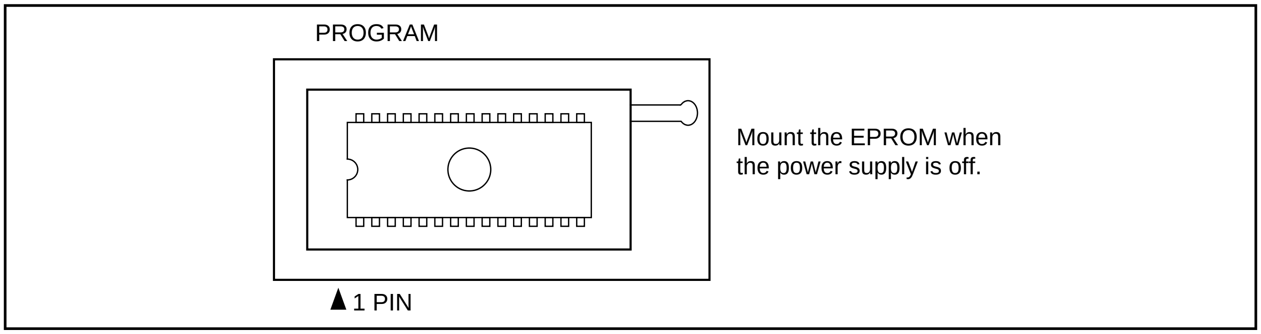


Figure 2-15. Mounting User Program EPROM

2-2-8. Setting Baud Rate

The EASE64165/167 uses a RS232C interface to communicate with the host computer. The baud rate switch sets the RS232C interface baud rate.

(1) Setting baud rate (dipswitches 1 to 4)

In EASE-LP mode, the baud rate is set with the dipswitch on the right side of the EASE-LP2. Table 2-9 shows the baud rate switch settings.

Table 2-9. EASE-LP2 Baud Rate Settings

Dipswitches		Baud Rate			
		19200	9600	4800	2400
1	19200	ON	OFF	OFF	OFF
2	9600	OFF	ON	OFF	OFF
3	4800	OFF	OFF	ON	OFF
4	2400	OFF	OFF	OFF	ON

(2) Setting flow control (dipswitch 5)

The dipswitch 5 is used to set the EASE-LP2 flow control to XON/XOFF or DTR/DSR as shown in Table 2-10.

Table 2-10. EASE-LP2 Flow Control Settings

Dipswitch		XON/XOFF	DTR/DSR
5	FLOW	OFF	ON

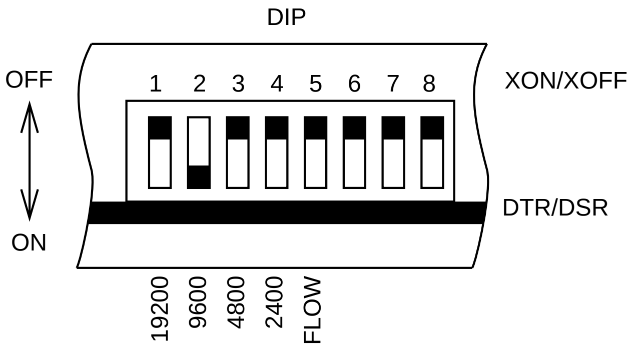


Figure 2-18. EASE-LP2 Dipswitch (settings when shipped)

RS232C parameters other than EASE-LP2 baud rate are set as follows.

RS232C parameter settings

- Transfer format	8 bits, 1 stop bit, no parity
- Others	asynchronous transmission, 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. (☞1)

☞ 1

For Oki if800 series computers, make these settings using the SWITCH command.
For PC9801 series computers, make these settings using the SPEED command.
For IBM PC computers, make these settings using the INT232C program (explained in Section 2-2-11).

For details, refer to the host computer's manuals.

!

The settings of the EASE-LP2 must always match those of the host computer that is connected through RS232C cable. If not, the EASE-LP2 cannot be invoked.

!

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.

!

The INT232C program does not support 19200 bps, so customers that will use IBM PCs should set the baud rate value to 2400-9600 bps.

2-2-9. MSM64165/167 ADC POD

The MSM64165/167 ADC POD serves as the MSM64165 or MSM64167 A/D converter.

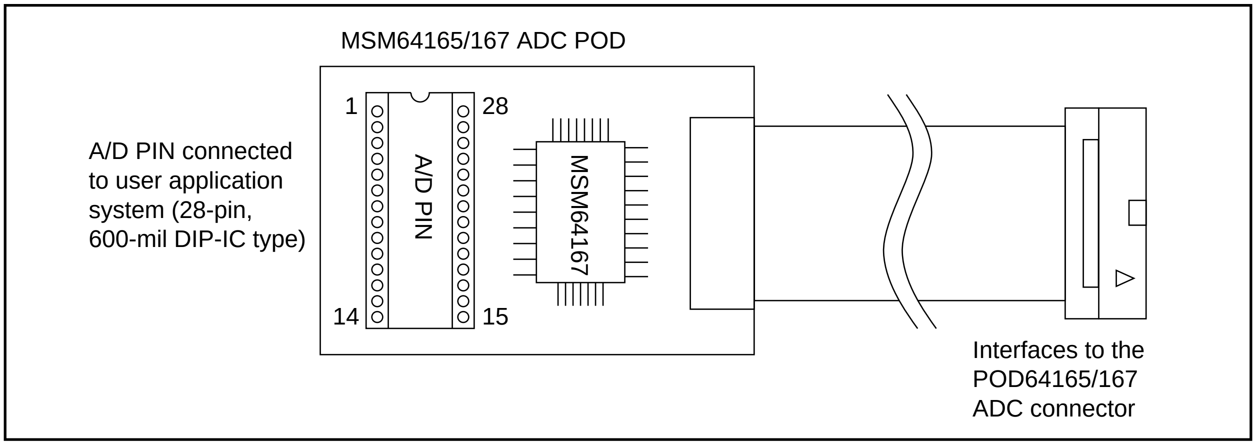

MSM64165/167 ADC POD
A/D PIN connected to user application system (28-pin, 600-mil DIP-IC type)
1
28
14
15
A/D PIN
MSM64167
Interfaces to the POD64165/167 ADC connector

Figure 2-17. MSM64165/167 ADC POD

An MSM64167 is mounted in the MSM64165/167 ADC POD, so it will perform A/D conversion with identical electrical characteristics to the MSM64167. Refer to the "MSM64165 User's Manual" or "MSM64167 User's Manual" for connection the A/D converter pins with the user application system.

The pin layout of the A/D PIN of the MSM64165/167 ADC POD is as shown below.

Pin no.	Signal Name	Pin no.	Signal Name
1	VOF	15	VG
2	VDDA	16	OPO0
3	VrA	17	OPN0
4	AIN0	18	OPP0
5	AIN1	19	OPO1
6	AIN2	20	OPN1
7	AIN3	21	OPP1
8	RA	22	N·C
9	RI	23	N·C
10	RCM	24	N·C
11	CZ1	25	N·C
12	CI	26	N·C
13	CZ2	27	N·C
14	GND	28	VDD

Note 1) NC pins are not connected.

Note 2) The VDD pin (pin 28) outputs +5V as the power supply for the user application system's A/D converter. However, do not use this power supply for port or buzzer circuits. Absolutely do not connect the VDD pin to the user connector VCC pins (pins 35 and 36).

! MISCONNECTION WILL DAMAGE THE POD64165/64167.

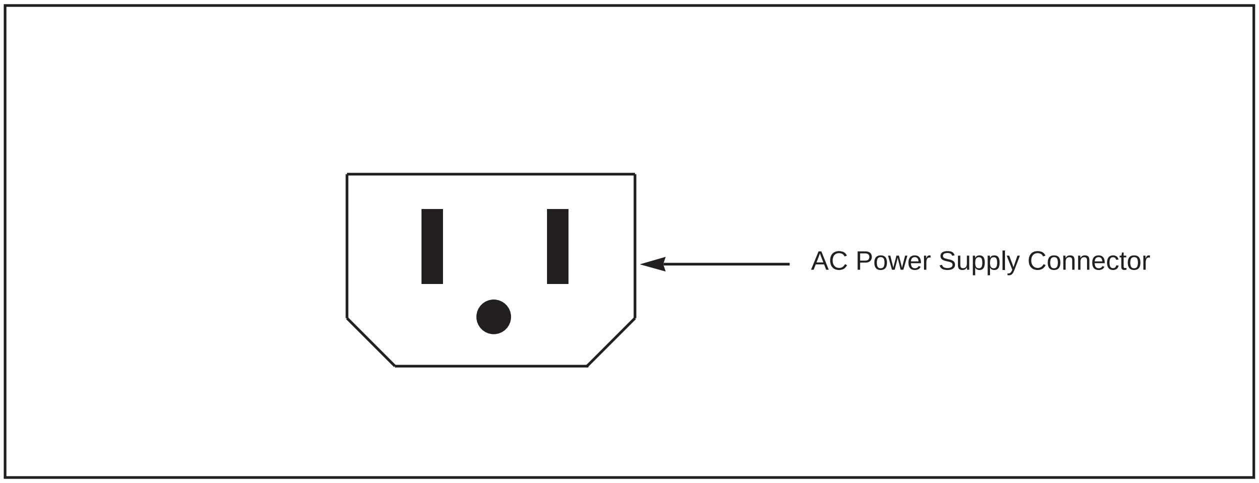
2-2-10. EASE64165/167 Power Supply

The EASE64165/167's power supply method differs for EASE-LP mode, when EASE-LP2 and POD64165/167 are both connected, and EVA mode, when EASE-LP2 is not used.

(1) EASE-LP mode

In EASE-LP mode, the EASE-LP2 and POD64165/167 operate using normal household power. This operation makes use of the EASE-LP2 internal switching regulator.

The power supply input voltage conversion ranges are AC90-132V and AC180-264V, but the rated voltages (applied safety standard values) are AC100-240V.



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

(2) EVA mode

In EVA mode, the POD64165/167 must be supplied from an external DC power supply using the accessory DC power supply cable. The DC power supply must conform to at least 5V +/- 5%, 3A. Connect the DC power supply cable red plug to the plus side and the black plug to the minus side.

! ABSOLUTELY DO NOT MIX UP THE POLARITY OF THE INPUT DC POWER SUPPLY. DOING SO WILL DAMAGE THE POD64165/64167.

2-2-11. Starting the EASE64165/167 Emulator

The procedure for starting the EASE64165/167 emulator differs depending on the operating mode. The procedures for each mode are given below.

(1) Starting in EASE-LP mode

1. Verify that the necessary cable types are connected to the emulation kit.

(a) Connect the AC power supply cable to the AC power supply connector.
- Be sure that the EASE-LP2 power supply switch is off.
- Note that the AC power supply rated voltage is AC 100-240 V.

(b) Connect the host computer to 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 POD64165/167.
- Connect the interface cables (80-pin, 100-pin) between the EASE-LP2 and POD64165/167 USR1 connectors and USR2 connectors.

(d) Connect the user cable.
- Connect the user cable when using the user application system.
- Connect the accessory 40-pin user cable between the POD64165/167 USER connector and the user application system.
- To debug using LCD, connect the accessory 34-pin cable between the POD64165/167 LCD connector and the user application system.
- To debug using LED, connect the accessory 34-pin cable between the POD64165/167 LED connector and the user application system.

(e) Connect the MSM64165/167 ADC POD.
- Connect the MSM64165/167 ADC POD to the POD64165/167's ADC connector. Also connect the ADC POD to the user application system.

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

(g) Connect the external power supply.
- Connect the external power supply to the user application system.
- Confirm that the external power supply's supply switch is off.
- If the POD64165/167's VCC select switch is off, then the user connector interface voltage will be 5V. If the VCC select switch is on, then the user connector interface voltage will depend on the external power supply voltage. Note that in this case, the DC voltage is rated for 3-5V.

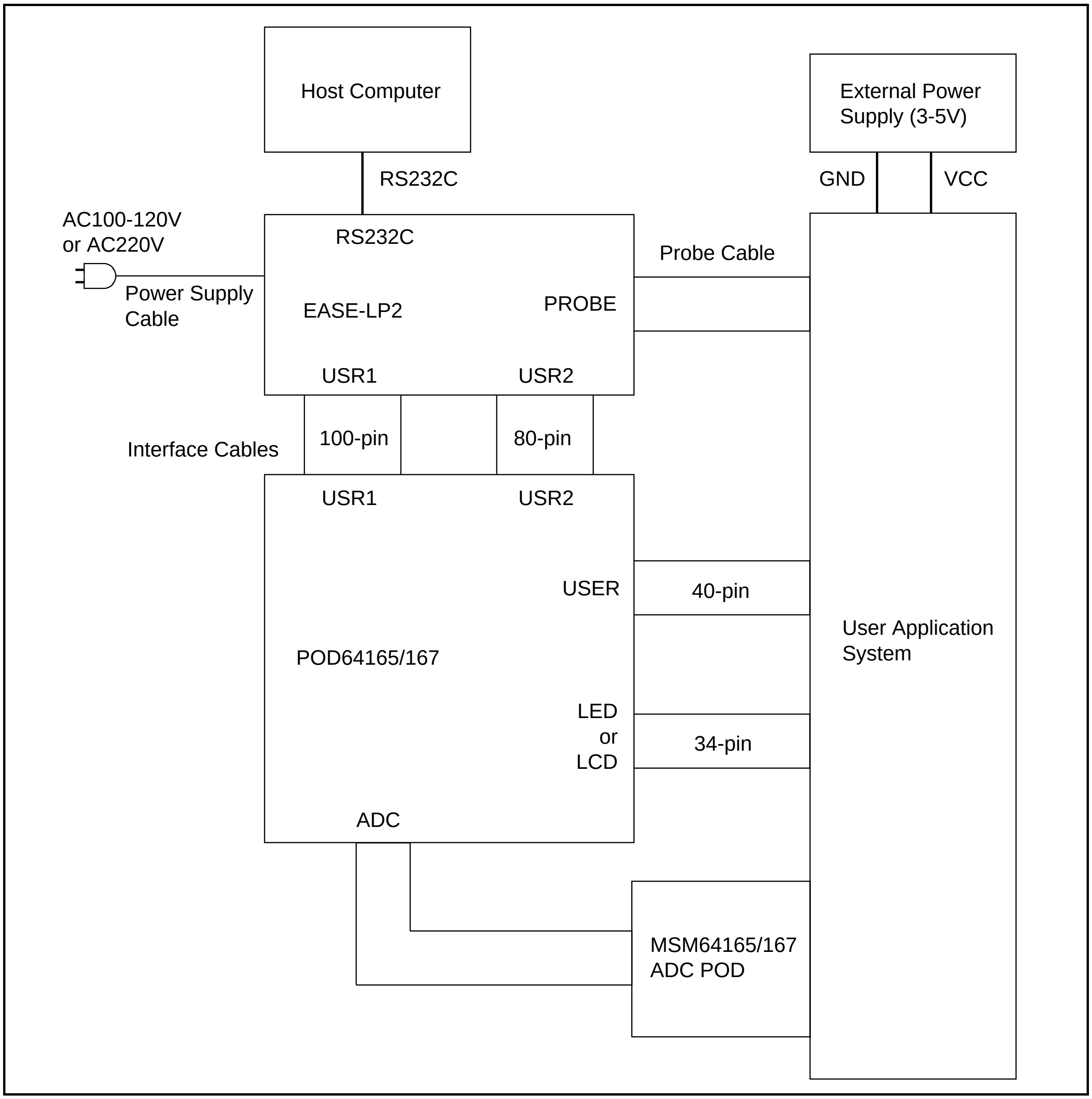


Figure 2-18. Cable Connection Diagram in EASE-LP Mode

! The system will start even if the user application system is not connected. In this case, do not connect the user cables.

! Vcc is not supplied to the user application system from the user cables (however, GND is connected to the user application system through the user cables). If Vcc must be supplied to the user application system, then supply it from a separate power supply.

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

EASE-LP2

(a) Baud rate setting
- Set the appropriate dipswitch 2400-19200 to match the baud rate to be used.

(b) Flow control setting
- Set the dipswitch FLOW to match the flow control to be used.

POD64165/167

(a) Operating mode setting
- Select EASE-LP mode by setting the mode 2 switch on and mode 1 switch off.

(b) Microcontroller type setting
- Select MSM64165 or MSM64167 with the number 1 switch of dipswitch 2.

(c) Operation frequency setting
- Select a low-speed or high-speed clock supply with the number 3 and 4 switches of dipswitch 2.

(d) Reset input setting
- Select whether the user connector RESET pin input is valid or not with the number 2 switch of dipswitch 2.

(e) VCC select switch setting
- Select an internal or external user connector interface power supply with the VCC select switch.

! Refer to Sections 2-2-1 to 2-2-9 regarding the various switch settings.

(3) Mount the mask option EPROM.

Mount the mask option EPROM generated with the mask option generator in the mask option EPROM socket.

(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 EASE64165/167 is shipped, its data transfer parameters are as follows. Except for baud rate, the EASE64165/167 parameters cannot be changed.

Baud rate	9600 bps
Transfer format	8 bits, 1 stop bit, no parity
Flow control	XON/XOFF control
Others	asynchronous transmission, baud rate factor x 16

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 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 computers use 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*

```
A> INT232C *;<baud>,N,8,1 ↵
```

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.

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 would 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 memory has been loaded, the message will be displayed.

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

!

To use the EASE64165/167 with a baud rate setting of 9600 bps, the following short form can be input.

```
A> INT232C * ↵
```

☞ 1

The valid baud rates for the EASE64165/167 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 or modem 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 XM;1200,E,7,2`** ↵

  This initializes the RS232C port to **XON/XOFF** control, modem 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. (☞2). 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 = pathname ↵
```

The pathname 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.

☞ 2

Currently the following **DCL** files are provided with the **EASE64165/167**.

Device	DCL File Name
MSM64165	E64165.DCL
MSM64167	E64167.DCL

(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>  pathname\SID64K ↵
```

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

(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. After about 20 seconds, 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 DCL file for MSM64165 or MSM64167 will be read, and memory mapping of the appropriate device will be performed.

```
Reading DCL file (E64XXX.DCL)...
```

E64XXX.DCL is the name of the **DCL** file read. Refer to the previous footnote regarding types of DCL files 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. The POWER indicators of the EASE-LP2 and POD64165/167 will light, and other indicators will go out.

```
64XXX>
```

The prompt will be the chip name of MSM64165 or MSM64167. For example, if E64167.DCL is read, then the prompt will be 64167>.

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

!

(1) When the EASE-LP2 and POD64165/167 are connected with the interface cable, the POD64165/167 will be supplied with power from the EASE-LP2, so do not connect the DC power supply cable to the POD64165/167 DC power jack.

(2) When the interface cable is correctly connected between the EASE-LP2 and POD64165/167, after power is applied the POD indicator on top of the EASE-LP2 will light.

Table 2-11 (a)-(b) shows the items initialized when power is applied to the EASE64165/167, when the reset switch is pressed, when an **RST** command is executed, or 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 that are opened by list command, etc. will be closed.

Table 2-11(a). Initialization In EASE-LP Mode

Item	Contents Initialized	Power Applied	Reset Switch Pressed	RST Command	RST E Command
Evaluation board	Initializes to same state as when a reset is input to a MSM64165 or MSM64167.	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", the state when no program has been executed.	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** command)	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 2-11(b). Initialization In EASE-LP Mode

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 (☞ 3).	–	–	–	–
Symbol Registration	Removes registrations (☞ 3).	–	–	–	–

☞ 3

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, they are not initialized by applying power or a reset to the EASE64165/167. However, if the SID64K terminates once, then all registered information will be lost.

(2) Starting in EVA mode

1. Verify that the necessary cable types are connected to the emulation kit.

(a) Connect the DC power supply cable to the DC power jack.

- Connect the accessory DC power supply cable to the POD64165/167 DC power jack.
- Connect the DC power supply red plug to the plus side of the of the external power supply, and the black plug to the minus side. Verify that the external power supply switch is off before connecting the cable.
- Note that the rated voltage of POD64165/167 is DC 5 V (+/- 5%).

(b) Connect the user cable.

- Connect the user cable when using the user application system.
- Connect the accessory 40-pin user cable between the POD64165/167 USER connector and the user application system.
- To debug using LCD, connect the accessory 34-pin cable between the POD64165/167 LCD connector and the user application system.
- To debug using LED, connect the accessory 34-pin cable between the POD64165/167 LED connector and the user application system.

(c) Connect the MSM64165/167 ADC POD.

- Connect the MSM64165/167 ADC POD to the POD64165/167's ADC connector. Also connect the ADC POD to the user application system.

(d) Connect the external power supply.

- Connect the external power supply to the user application system.
- Confirm that the external power supply's supply switch is off.
- If the POD64165/167's VCC select switch is off, then the user connector interface voltage will be 5V. If the VCC select switch is on, then the user connector interface voltage will depend on the external power supply voltage. Note that in this case, the DC voltage is rated for 3-5V.

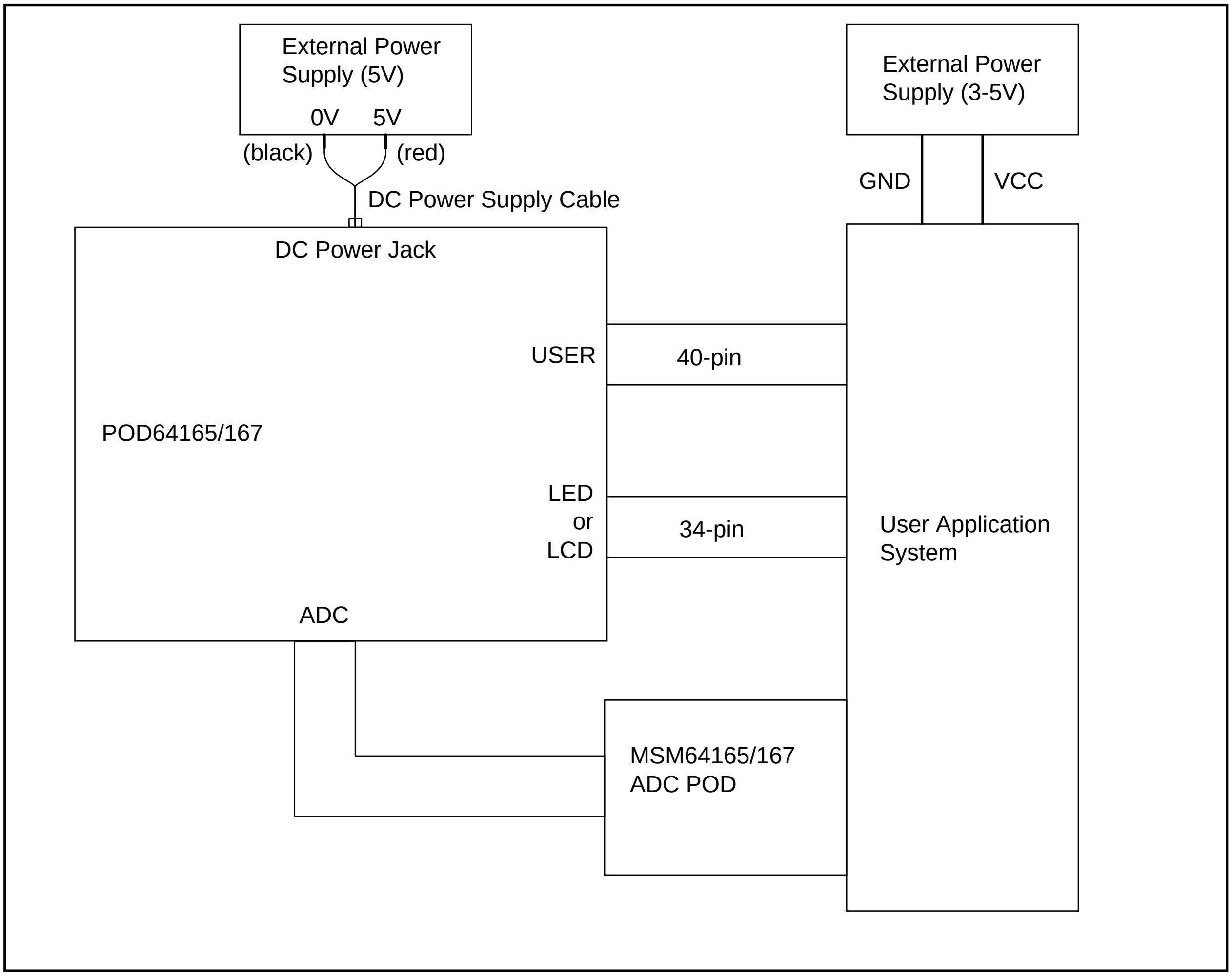


Figure 2-19. Cable Configuration Diagram in EVA Mode

Vcc is not supplied to the user application system from the user cables (however, GND is connected to the user application system through the user cables). If Vcc must be supplied to the user application system, then supply it from a separate power supply.

(2) Verify that the POD64165/167 switches are set correctly.

(a) Operating mode setting
- Select EVA mode by setting the mode 2 switch on and mode 1 switch on.

(b) Microcontroller type setting
- Select MSM64165 or MSM64167 with the number 1 switch of dipswitch 2.

(c) Operation frequency setting
- Select a low-speed or high-speed clock supply with the number 3 and 4 switches of dipswitch 2.

(d) VCC select switch setting
- Select an internal or external user connector interface power supply with the VCC select switch.

! Refer to Sections 2-2-1 to 2-2-9 regarding the various switch settings.

(3) Mount the mask option EPROM.

Mount the mask option EPROM generated with the mask option generator in the mask option EPROM socket.

(4) Mount the user program EPROM.

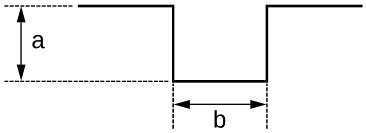
Mount the user program EPROM generated with the cross-assembler in the program EPROM socket.

(5) Turn on the POD64165/167 external power supply switch and the power supply of the user application system.

The POWER indicator and RUN indicator of the POD64165/167 will light, and other indicators will go out.

(6) Input a reset signal on the user connector RESET/ pin.

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



a: voltage 3-5 V (☞ 4)
b: 10 ms or longer

☞ 4 The voltage level of the reset signal input on the RESET/ pin will differ depending on the setting of the VCC select switch. If the VCC select switch is on, then match the voltage to the voltage provided by the user connector VCC pins (3-5V). If the VCC select switch is off, then match the voltage to the emulation kit's internal voltage (5V).

(7) At this point, all internal states are initialized, ad the user program is executed from address 00H.

!

(1) Do not handle the mask option EPROM or user program EPROM when power is on.

(2) The reset signal input from the RESET pin must be at a "L" level when power is applied until the POD64165/167's internal oscillator begins oscillating. If you change the oscillator mounted when the system was shipped, then you must input a reset signal appropriate for the oscillator to be used. When power has not been just applied, the reset operation can be performed if the reset input is at a "L" level for at least one clock width. For details on the reset operation, refer to the "MSM64165 User's Manual" or "MSM64167 User's Manual."

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 strings 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 value used within an explanation. The xxxx enclosed in [ ] means that it can be omitted.
[addressΔaddress]	This indicates an address range.
___ (underline)	When operator input and text displayed automatically by the debugger 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

SID64X 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

   + - * / % & | ^ ~ ! . < + > ( )
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 comment fields, described later.

2-3-1-2. Command Format

❏ *Debugger Command Format*

command_name ∆ *parm* ∆ *parm* . . . *parm* ∆ *option* ∆ *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 comment.

Example

```
64167> ;;;; This an example of a whole line comment ;;;;
64167> 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 any key other than the space key.

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

❏ *Command Name Format*

Command names are strings consisting of 1-7 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 evaluation board 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, BATCH, LIST, NLST, SH, EXIT

2-3-2. Command Summary

This section gives a summary table of all SID64K commands.

<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">Command Syntax</td></tr>
<tr><td colspan="2">Explanation of Parameters / options</td></tr>
</table>

Detailed explanations of each command are given in Section 3-3-4. 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.

• 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."

2-3-2-1. Command Summary in EASE-LP Mode

Evaluation Chip Access Commands			
1	**D**	(Display the contents of the Evaluation Chip)	3-5
	D [Δ *parm* ......Δ *parm*]↵		
	parm	: *SFR_mnemonic, Register_mnemonic*	
2	**C**	(Change the contents of the Evaluation Chip)	3-5
	C Δ *parm* [Δ *parm* ......Δ *parm*]↵		
	parm	: *mnemonic* [=*data*]	
	mnemonic	: *SFR_mnemonic, Register_mnemonic*	
		PC (program counter), C (carry flag)	
	BSR0 (bank select register 0), BSR1 (bank select register 1)		
3	**SDF**	(Set Data Format)	3-16
	SDF [Δ *parm* ......Δ *parm*]↵ SDF [~]ALL		
	parm	: [~] *SFR_mnemonic*	
4	**DPC**	(Display Program Counter)	3-17
	DPC↵		
5	**CPC**	(Change Program Counter)	3-17
	CPC Δ *address* ↵		

Code Memory Commands			
1	**DCM**	(Display Code Memory)	3-19
	DCM Δ *parm* ↵ DCM Δ* ↵		
	parm	: *expression* Δ [*expression* ......Δ *expression*] *expression* : *address* : [*address* Δ *address*]	
2	**CCM**	(Change Code Memory)	3-21
	CCM Δ *parm* [Δ *parm* ......Δ *parm*]↵ CCM Δ* = *data* ↵		
	parm	: *address* [=*data*] : [*address* Δ *address*]	
3	**MCM**	(Move Code Memory)	3-24
	MCM Δ [*address* Δ *address*] Δ *address*↵		
4	**LOD**	(Load Disk file program into Code Memory)	3-26
	LOD Δ *fname* [Δ *option* ...... Δ *option*]↵		
	fname *option*	: [*Pathname*] *Filename* [*Extension*] : /S, /N, /B	
5	**SAV**	(Save Code Memory into Disk file)	3-30
	SAV Δ *fname* [[*address* Δ *address*]][Δ *option* ...... Δ *option*]↵		
	fname *option*	: [*Pathname*] *Filename* [*Extension*] : /S, /N	

Code Memory Commands (continued)			
6	**VER**	(Verify Disk file with Code Memory)	3-32
	VER Δ *fname* [[*address* Δ *address*]]↵		
	fname	: [*Pathname*] *Filename* [*Extension*]	
7	**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.	3-35
	ASM Δ *address*↵		
8	**DASM**	Disassembler Command Disassembles a specified address range of code memory.	3-39
	DASM [Δ *parm*][Δ *option* ...... Δ *option*]↵		
	parm *option*	: [*address*] , [[*address* Δ *address*]] : /NC, /NL	
9	**EXPAND**	(Expand the code memory area)	3-42
	EXPAND [Δ *mnemonic*]↵		
	mnemonic	: ON, OFF	

Data Memory Commands			
1	**DDM**	(Display Data Memory)	3-45
	DDM Δ *parm* [Δ *parm* ......Δ *parm*]↵ DDM Δ* ↵		
	parm	: *address* : [*address* Δ *address*]	
2	**CDM**	(Change Data Memory)	3-45
	CDM Δ *parm* [Δ *parm* ......Δ *parm*]↵		
	parm	: *address* [=*data*] : [*address* Δ *address*] = *data*	
3	**MDM**	(Move Data Memory)	3-49
	MDM Δ [Δ*address* ...... Δ *address*] Δ *address*↵		

	Emulation Commands		
1	**STP**	Step Execution	3-52
	STP [Δ *address*] [Δ ,*count*]↵		
2	**SSF**	Set Step Format	3-54
	SSF [Δ *parm* ......Δ *parm*]↵		
	parm	: [~] *mnemonic*	
3	**G**	Real Time Emulation (Continuous Emulation)	3-57
	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*]	
4	**ESC**	Command usable during emulation Forced Break of the Emulation	3-64
	ESC↵		

Emulation Commands (continued)

5	**DCT**	Command usable during emulation Display Cycle Counter Trigger	3-66
	DCT↵		
6	**DTT**	Command usable during emulation Display Trace Trigger	3-67
	DTT↵		
7	**D**	Command usable during emulation Display the Contents of the Evaluation Chip	3-68
	D [∆ *parm* ......∆ *parm*]↵		
	parm	: A, B, X, Y, H, L, PC (however, the XY and HL registers depend on the **CTO** command)	

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* ↵		
	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	3-81
	DTM Δ *parm* ↵ DCM D* ↵		
	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	3-88
	CTO Δ *parm* Δ *parm* [Δ *parm* [Δ *parm*]]↵		
	parm	: *mnemonic*	
4	**DTO**	Display Trace Object	3-88
	DTO ↵		
5	**STT**	Set Trace Trigger	3-91
	STT Δ *mnemonic1* STT Δ *mnemonic2* [/ [*parm1*] / [*parm2*]]↵ *parm1, parm2* : *address* : [*address* Δ *address*] : . STT Δ *mnemonic3 trc_mnem* [*&mask*] = *data* ↵		
	mnemonic1 : ALL, TR, DIS *mnemonic3* : AD, BD *parm1* : starting address of trace *trc_mnem* : PRB (Probe), RAM [Δ *ram_address*] (data RAM)	*mnemonic2* : SS *parm2* : ending address of trace	
6	**DTT**	Display Trace Trigger	3-94
	DTT ↵		

Trace Commands (continued)

7	**DTR**	Display Trace Enable Bits	3-95
	DTR Δ *parm* [Δ *parm* ...... Δ *parm*]↵ DTR Δ* ↵		
	parm	: *address* : [*address* Δ *address*]	
8	**CTR**	Change Trace Enable Bits	3-97
	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	3-99
	DTP ↵		
10	**RTP**	Reset Trace Pointer	3-99
	RTP ↵		
11	**S**	Search Trace Memory	3-101
	S [~] *mnemonic data* [*parm*]↵		
	mnemonic *data* *parm*	: : search data (comparison data) : [*count*] , [*start_count* Δ *end_count*]	

Reset Commands

1	**RST**	Reset the System	3-104
	RST ↵		
2	**RST E**	Reset the Evaluation chip	3-105
	RST ∆ E ↵		

Performance/Coverage Commands			
1	**DCC**	Display Cycle Counter	3-107
	DCC ↵		
2	**CCC**	Change Cycle Counter	3-108
	CCC Δ [-] *data* ↵		
3	**TIME**	Set Unit Time	3-109
	TIME [Δ *data*]↵		
4	**SCT**	Set Cycle Counter Trigger	3-110
	SCT [Δ / [Δ *parm1*] / [Δ *parm2*]]↵		
	parm1, parm2	: *address* : [*address* Δ *address*] : . *parm1* : start status; *parm2* : stop status	
5	**DCT**	Display Cycle Counter Trigger	3-113
	DCT ↵		
6	**RCT**	Reset Cycle Counter Trigger	3-113
	RCT ↵		

Performance/Coverage Commands (continued)

7	**DIE**	Display Instruction Executed Bits	3-115
	DIE Δ *parm* [Δ *parm* ......Δ *parm*]↵ DIE Δ* ↵		
	parm	: *address* : [*address* Δ *address*]	
8	**CIE**	Change Instruction Executed Bits	3-116
	CIE Δ *parm* [Δ *parm* ......Δ *parm*]↵ CIE Δ* ↵		
	parm	: *address = data* : [*address* Δ *address*] = *data* *data* : 0, 1	
9	**DAP**	Display Address Pass Counter	3-118
	DAP [Δ *mnemonic* ......Δ *mnemonic*]↵		
	mnemonic	: C0, C1, C2, C3	
10	**CAP**	Change Address Pass Counter	3-119
	CAP Δ *mnemonic* [= *address*][Δ *count*]↵		
	mnemonic	: C0, C1, C2, C3	

EPROM Programming Commands			
1	**TYPE**	Set EPROM Type	3-121
	TYPE [Δ *mnemonic*]↵		
2	**PPR**	Program EPROM	3-123
	PPR Δ *parm* [*address* Δ *address*] [Δ *eprom_address*]↵ PPR Δ* ↵		
3	**TPR**	Transfer EPROM into Program Memory	3-125
	TPR Δ *parm* [*address* Δ *address*] [Δ *CM_address*]↵ TPR Δ* ↵		
4	**VPR**	Verify EPROM with Code Memory	3-127
	VPR Δ [*address* Δ *address*] [Δ *eprom_address*]↵ VPR Δ* ↵		

Commands for Automatic Command Execution			
1	**BATCH**	Batch Processing	3-131
	BATCH Δ *fname* ↵		
	fname	: [*Pathname*] *filename* [*Extension*]	
2	**PAUSE**	Pause Command Input	3-133
	PAUSE ↵		

Commands for Displaying/Changing/Removing Symbols			
1	**DSYM**	Display Symbol	3-135
	DSYM Δ *string* [Δ *string* ...... Δ *string*]↵ DSYM Δ* ↵		
2	**CSYM**	Change Symbol	3-137
	CSYM Δ *parm* [*,parm* ...... *,parm*]↵ *parm* : *string* [= *data*]		
3	**RSYM**	Remove Symbol	3-139
	RSYM Δ *string* [Δ *string* ...... Δ *string*]↵ RSYM Δ* ↵		

Other Commands			
1	**LIST**	Listing. Redirect the Console output to Disk file	3-141
	LIST Δ *fname* ↵		
2	**NLST**	No Listing. Cancel the Console output Redirection	3-142
	NLST ↵		
3	**SH**	Call OS Shell	3-143
	SH ↵		
4	**RADIX**	Numeral Radix	3-145
	RADIX Δ *mnemonic* ↵		
	mnemonic	: H, D, O, B	
5	**MAC**	Macro Command	3-147
	MAC [Δ [~] *macro_command*] ↵		
6	**EXIT**	Terminate the Debugger and Exit to OS	3-151
	EXIT ↵		

2-3-3. 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 below) can be included between these elements.

The input format of expression is as follows.

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

Elements enclosed in branckets [ ] 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** directives 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 ∆ *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 2-12.

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.

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 2-12. 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 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 after the backslash. 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 '-' (subtraction), and parentheses. A detailed explanation of operators is given in Chapter 5, so it is omitted here.

All calculations are perfoermed as 32-bit unsigned calculations. If a calculation result is negative, then it will be expressed in 2's complement. Overflows are ignored. An expression's value 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.

2-3-4. 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 current 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 (see Figure 2-20).

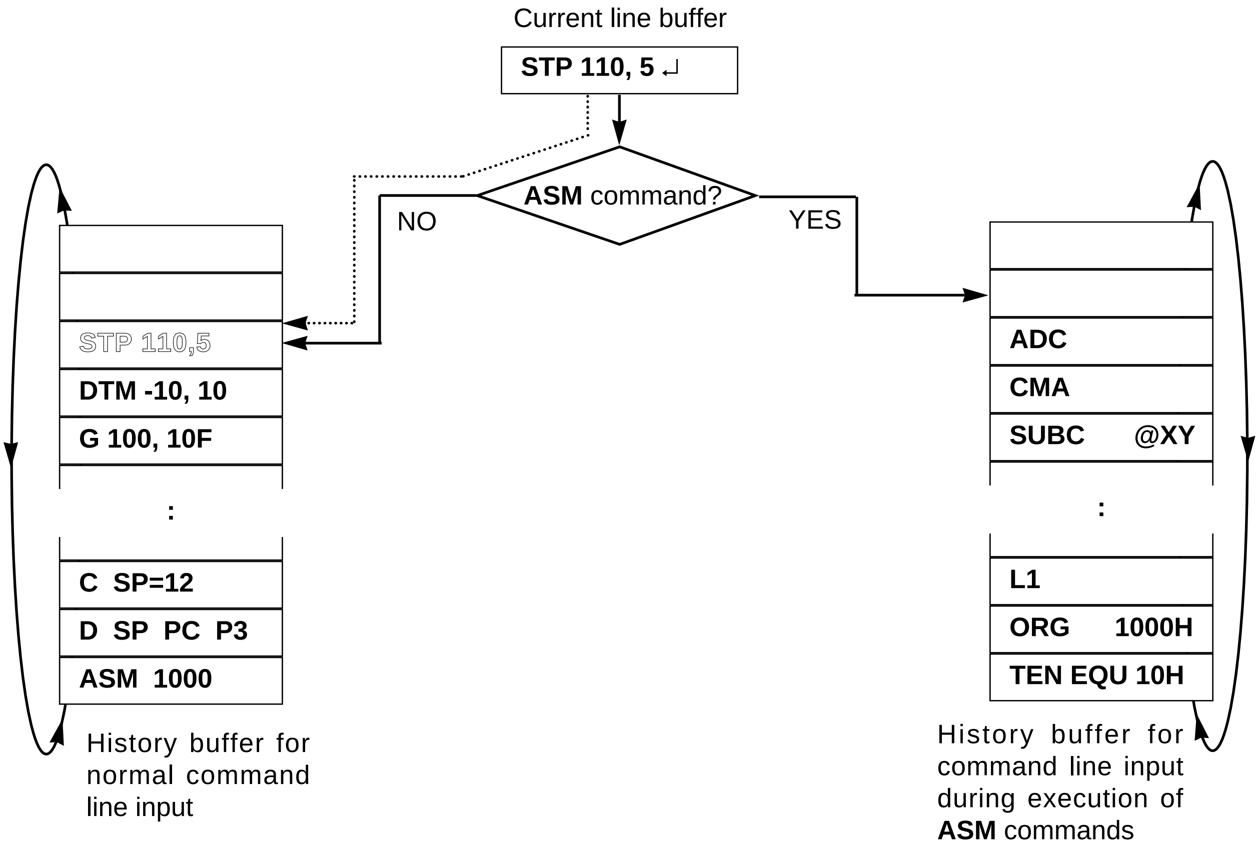


Figure 2-20. Current Line Buffer and History Buffer

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 20 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.

(2) Using history functions

This somewhat covers the same material as Section 3-3-3, "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 without any editing if only the ↵ key is pressed .

☞ 1 SID64K command line input is input from the console after the SID64K output prompt "64165>," "64167>," and "GO>>," and during **ASM** command execution.

2-3-5. 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.

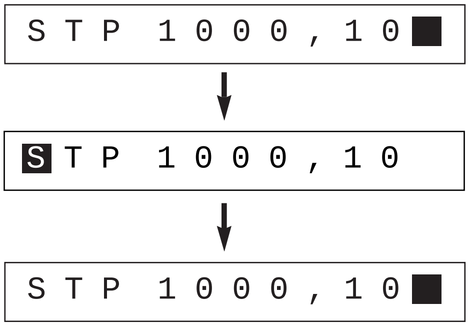
Example Contents of current line buffer before editing

CTRL + A pressed

Contents of current line buffer after editing

CTRL + Z pressed

Contents of current line buffer after editing



```
STP 1000,10█
```

```
STP 1000,10
```

```
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

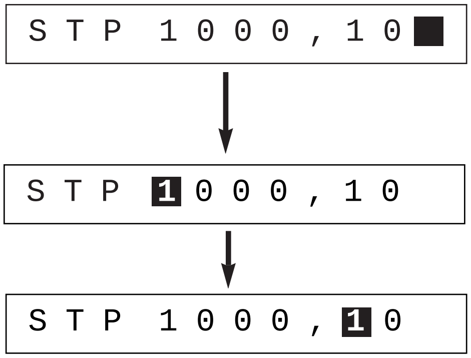
Contents of current line buffer before editing

CTRL + B pressed
CTRL + B pressed

Contents of current line buffer after editing

CTRL + F pressed

Contents of current line buffer after editing



(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

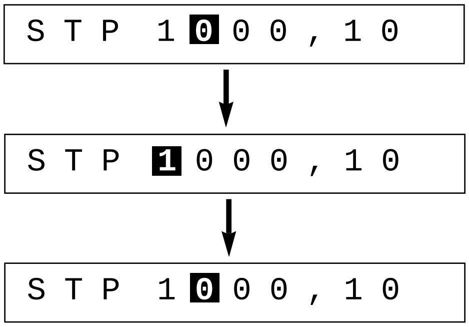
Contents of current line buffer before editing

CTRL + H or ← pressed

Contents of current line buffer after editing

CTRL + L or → pressed

Contents of current line buffer after editing



(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 2-3-4, "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

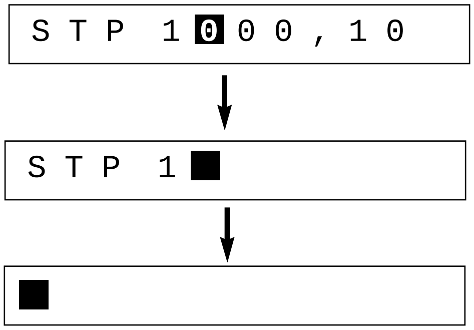
Contents of current line buffer before editing

CTRL + D pressed

Contents of current line buffer after editing

CTRL + X pressed

Contents of current line buffer after editing



```
STP 1000,10
```

```
STP 1■
```

```
■
```

(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. The cursor position does not change.

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

```
S T P 1 0 0 0 , 1 0
```

CTRL + R or **INS** pressed

↓

Contents of current line buffer after editing

```
S T P 1 █ 0 0 0 , 1 0
```

DEL pressed

↓

Contents of current line buffer after editing

```
S T P 1 0 0 0 , 1 0
```

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. For the NEC PC-9801 change the key table file **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 last 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.	0H8
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

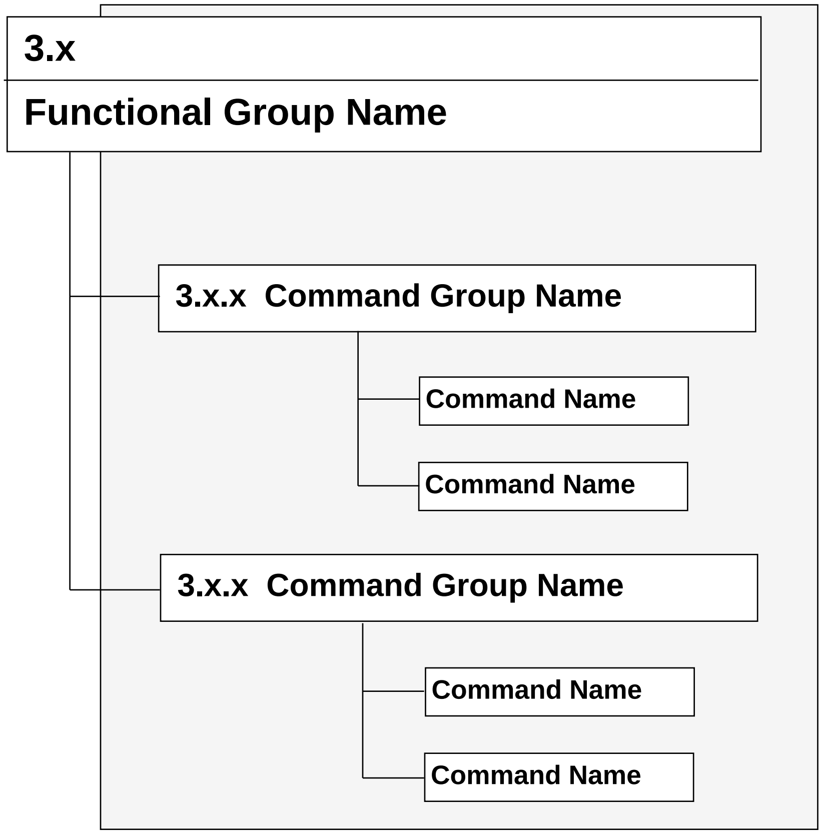
Chapter 3

SID64K Commands

This chapter explains in detail how to use SID64K commands.

This section explains the debugger 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

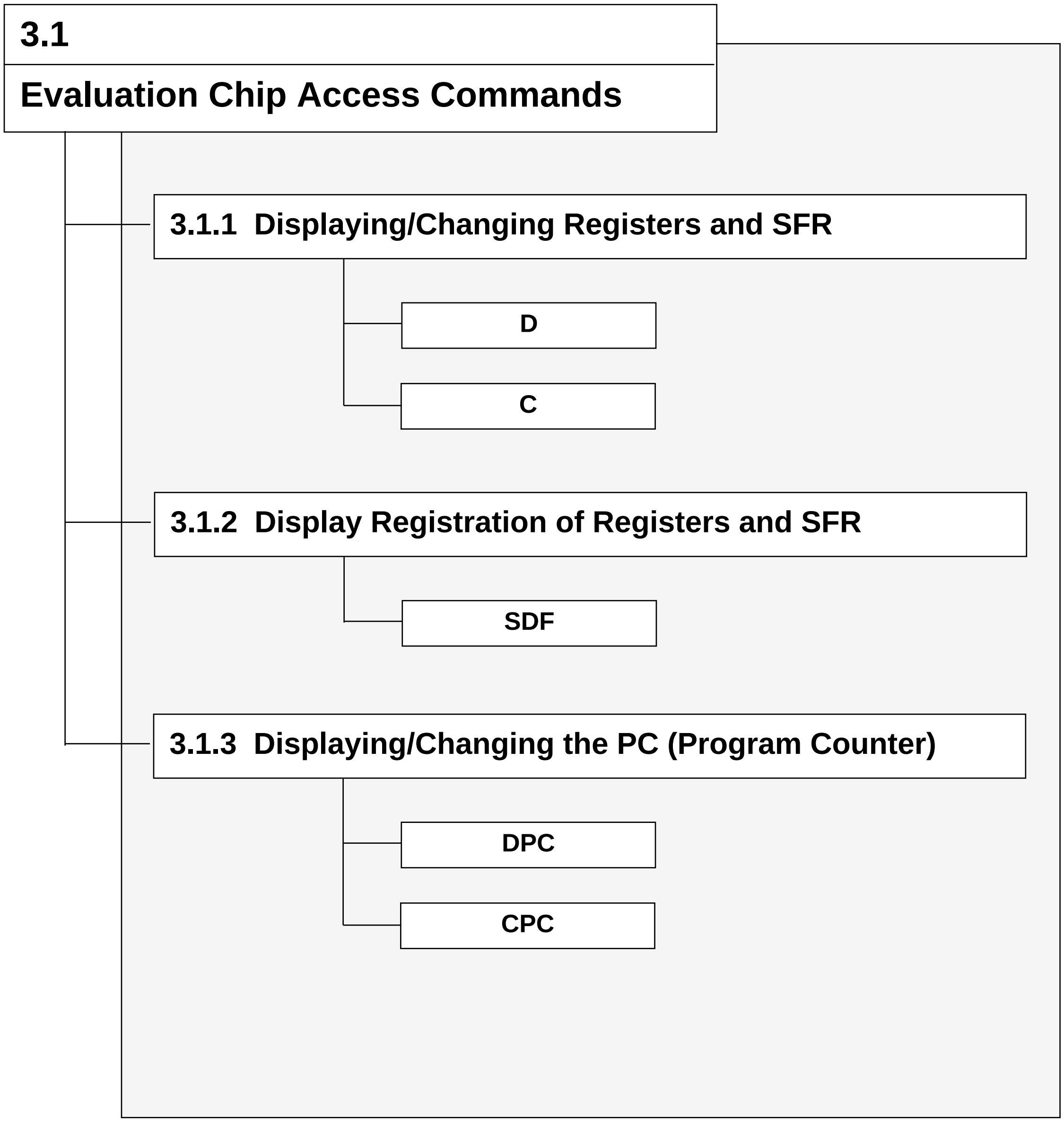
Description

Execution Example

If the **EXPAND** command has been set ON, then expand mode will be selected. Expand mode expands the addresses of code memory, attribute memory, and instruction executed memory to 0-7FFFH. Valid commands in expand mode are indicated by the following mark.

EXPAND Indicates addresses for the command will be 0-7FFFH in expand mode.

SEE **EXPAND**



3.1

Evaluation Chip Access Commands

3.1.1 Displaying/Changing Registers and SFR

D

C

3.1.2 Display Registration of Registers and SFR

SDF

3.1.3 Displaying/Changing the PC (Program Counter)

DPC

CPC

3.1.1 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)
or
BSR0 (bank select register0),
BSR1 (bank select register1),
BCF (bank select flag),
BEF (bank enable flag),

Description

The **D** command displays the contents of 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, BSR1, BCF or BEF.

A *Register-mnemonic* indicates a general-purpose register. It is expressed as follows.

Register-mnemonic	:	A	(A register)
	:	B	(B register)
	:	H	(H register)
	:	L	(L register)
	:	X	(X register)
	:	Y	(Y register)
	:	AB	(A register and B 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 (a-c).

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

The **C** command changes the contents of 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 (a-c). 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
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 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 (a)-(b) shows the bit symbols defined in the DCL files. Table 3-3 (a)-(b) 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 EBD (bit2) of BDCON (0AH)
EBD bit symbol value = 0AH x 4 + 2
= 2AH

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 EBD" is the same as "DCM 2AH"

☞ 1 '**D** ↵' will normally display the general-purpose registers (A, B, H, L, X, and Y), PC (program counter), C (carry flag), BSR0, BSR1, BCF, and BEF.

! For several SFRs of the MSM64165 and MSM64167, unallocated bits exist as reserved bits. For the EASE64165/167, these reserved bits are handled as shown below. Refer to the user's manual of the MSM64165 or MSM64167 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. (a) List of SFR-mnemonics

	SFR-mnemonic	Register Name	Address
	P0	Port 0 Register	00H
	P1	Port 1 Register	01H
	P2	Port 2 Register	02H
	VSSLCON	VSSL Control Register	08H
	FCON	Frequency Control Register	09H
	BDCON	Buzzer Driver Control Register	0AH
	BFCON	Buzzer Frequency Control Register	0BH
	TBCR	Time-Base Counter Register	0FH
(☞**2**)	P00CON	Port 00 Control Register	10H
(☞**2**)	P01CON	Port 01 Control Register	11H
(☞**2**)	P02CON	Port 02 Control Register	12H
(☞**2**)	P03CON	Port 03 Control Register	13H
(☞**2**)	P10CON	Port 10 Control Register	14H
(☞**2**)	P11CON	Port 11 Control Register	15H
(☞**2**)	P12CON	Port 12 Control Register	16H
(☞**2**)	P13CON	Port 13 Control Register	17H
(☞**2**)	P20CON	Port 20 Control Register	18H
(☞**2**)	P21CON	Port 21 Control Register	19H
(☞**2**)	P22CON	Port 22 Control Register	1AH
(☞**2**)	P23CON	Port 23 Control Register	1BH
(☞**2**)	PCON	Port Control Register	1CH
	DSPCON	Display Control Register	1EH
	TMD0	Timer Data Register 0	20H
	TMD1	Timer Data Register 1	21H
	TMD2	Timer Data Register 2	22H
(☞**4**)	TMD3	Timer Data Register 3	23H
	TMC0	Timer Counter Register 0	24H
	TMC1	Timer Counter Register 1	25H
	TNC2	Timer Counter Register 2	26H
(☞**4**)	TMC3	Timer Counter Register 3	27H
	TMCON0	Timer Control Register 0	28H
(☞**2**)	TMCON1	Timer Control Register 1	29H
(☞**3**)	TMSTAT	Timer Status Register	2AH

Table 3-1. (b) List of SFR-mnemonics

	SFR-mnemonic	Register Name	Address
	ADCON0	A/D Control Register 0	2CH
	ADCON1	A/D Control Register 1	2DH
	ADCON2	A/D Control Register 2	2EH
(☞**3**)	ADSTAT	A/D Status Register	2FH
	IE0	Interrupt Enable Register 0	30H
	IE1	Interrupt Enable Register 1	31H
	IRQ0	Interrupt Request Register 0	32H
	IRQ1	Interrupt Request Register 1	33H
	IE2	Interrupt Enable Register 2	34H
	IRQ2	Interrupt Request Register 2	36H
(☞**2**)	WDTCON	Watchdog Timer Control Register	38H
	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

Table 3-1. (c) List of SFR-mnemonics

	SFR-mnemonic	Register Name	Address
	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
	STCONL	Transmit Control Register (L)	60H
(☞**4**)	STCONH	Transmit Control Register (H)	61H
(☞**4**)	STBUFL	Transmit Buffer Register (L)	62H
(☞**4**)	STBUFH	Transmit Buffer Register (H)	63H
(☞**4**)	SRCONL	Receive Control Register (L)	64H
(☞**4**)	SRCONH	Receive Control Register (H)	65H
(☞**3, 4**)	SRBUFL	Receive Buffer Register (L)	66H
(☞**3, 4**)	SRBUFH	Receive Buffer Register (H)	67H
(☞**4**)	SRBRT	Receive Baud Rate Setting Register	68H
(☞**3, 4**)	SSTAT	Serial Port Status Register	69H
	MIEF	Master Interrupt Enable Register	7CH
(☞**5**)	HALT	Halt Mode Register	7DH
	SP	Stack Pointer	7EH, 7FH

☞ **2** Because P00CON, P01CON, P02CON, P03CON, P10CON, P11CON, P12CON, P13CON, P20CON, P21CON, P22CON, P23CON, PCON, TMCON1, and WDTCON are write-only SFRs, they are not valid with the display command.

☞ **3** Because TMSTAT, ADSTAT, SRBUFL, SRBUFH, and SSTAT are read-only SFRs, they are not valid with the change command.

☞ **4** TMD3, TMC3, STCONL, STCONH, STBUFL, STBUFH, SRCONL, SRCONH, SRBUFL, SRBUFH, SRBRT, and SSTAT are invalid in MSM64165 mode.

☞ **5** HALT is invalid with the change command (HALT mode will be forcibly released when emulation is not executed).

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
	VSSLCON	*-	*-	*-	*-
	FCON	*-	*-	*-	VSSLH
	BDCON	SELF	EBD	BMI	BM0
	BFCON	*-	*-	*-	*-
(☞**8**)	TBCR	_1HZ	_2HZ	_4HZ	_8HZ
	P00CON	P00IE	P00F	P00DIR	P00MOD
	P01CON	P01IE	P01F	P01DIR	P01MOD
	P02CON	P02IE	P02F	P02DIR	P02MOD
	P03CON	P03IE	P03F	P03DIR	P03MOD
	P10CON	P10IE	P10F	P10DIR	P10MOD
	P11CON	P11IE	P11F	P11DIR	P11MOD
	P12CON	P12IE	P12F	P12DIR	P12MOD
	P13CON	P13IE	P13F	P13DIR	P13MOD
	P20CON	P20IE	P20F	P20DIR	P20MOD
	P21CON	P21IE	P21F	P21DIR	P21MOD
	P22CON	P22IE	P22F	P22DIR	P22MOD
	P23CON	P23IE	P23F	P23DIR	P23MOD
	PCON	*-	*-	*-	PUD
	DSPCON	*-	LCDOFF	DUTY1	DUTY0
	TMD0	TD3	TD2	TD1	TD0
	TMD1	TD7	TD6	TD5	TD4
	TMD2	TD11	TD10	TD9	TD8
(☞**7**)	TMD3	TD15	TD14	TD13	TD12
	TMC0	TC3	TC2	TC1	TC0
	TMC1	TC7	TC6	TC5	TC4
	TMC2	TC11	TC10	TC9	TC8
(☞**7**)	TMC3	TC15	TC14	TC13	TC12
	TMCON0	*-	ECAP	FMEAS	TMRUN
	TMCON1	*-	*-	CL1	CL0
	TMSTAT	*-	*-	TMOVF	CAPF

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	ICH1	ICH0	CH1	CH0
ADCON1	ENOPA	ENADC	SOPP0	*-
ADCON2	*-	SMODE	SMCH1	SMCH0
ADSTAT	ADPOL	ADOVF	ADST1	ADST0
IE0	EAD	E32HZ	E16HZ	E1HZ
(☞**6**) IRQ0	QAD	Q32HZ	Q16HZ	Q1HZ
IE1	ESR	EST	EXI1	ETM
IRQ1	QSR	QST	QXI1	QTM
IE2	*-	*-	*-	EXI0
IRQ2	*-	*-	QWDT	QXI0
(☞**7**) STCONL	STSTB	STL0	STL1	STMOD
(☞**7**) STCONH	STLMB	STPOE	STPEN	STCLK
(☞**7**) STBUFL	TB3	TB2	TB1	TB0
(☞**7**) STBUFH	TB7	TB6	TB5	TB4
(☞**7**) SRCONL	SREN	SRL0	SRL1	SRMOD
(☞**7**) SRCONH	SRLMB	STPOE	SRPEN	SCCLK
(☞**7**) SRBUFL	RB3	RB2	RB1	RB0
(☞**7**) SRBUFH	RB7	RB6	RB5	RB4
(☞**7**) SRBRT	*_	*-	BRT0	BRT1
(☞**7**) SSTAT	BFULL	PERR	0ERR	FERR
MIEF	*-	*-	*-	MI
HALT	*-	*-	*-	HLT
SP	SP3	SP2	SP1	*-
	*-	SP6	SP5	SP4

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

☞ **6** The QSR and QST bits of the IRQ1 register are reserved bits when in MSM64165 mode.

☞ **7** The bit symbols of TMD3, TMC3, STCONH, STBUFL, STBUFH, SRCONL, SRCONH, SRBUFL, SRBUFH, SRBRT, and SSTAT are invalid in MSM64165 mode.

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

Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value
P00	00H	P02 MOD	48H	RXC	66H	TC0	90H
P01	01H	P02 DIR	49H	P21 IE	67H	TC1	91H
P02	02H	P02 F	4AH	P22 MOD	68H	TC2	92H
P03	03H	P02 IE	4BH	P22 DIR	69H	TC3	93H
P10	04H	P03 MOD	4CH	TXD	6AH	TC4	94H
P11	05H	P03 DIR	4DH	P22 IE	6BH	TC5	95H
P12	06H	P03 F	4EH	P23 MOD	6CH	TC6	96H
P13	07H	P03 IE	4FH	P23 DIR	6DH	TC7	97H
P20	08H	P10 MOD	50H	TMO	6EH	TC8	98H
P21	09H	P10 DIR	51H	P23 IE	6FH	TC9	99H
P22	0AH	P10 F	52H	PUD	70H	TC10	9AH
P23	0BH	P10 IE	53H	DUTY0	78H	TC11	9BH
VSSLH	20H	P11 MOD	54H	DUTY1	79H	TC12	9CH
CPUCLK	24H	P11 DIR	55H	LCDOFF	7AH	TC13	9DH
BM0	28H	P11 F	56H	TD0	80H	TC14	9EH
BM1	29H	P11 IE	57H	TD1	81H	TC15	9FH
EBD	2AH	P12 MOD	58H	TD2	82H	TMRUN	A0H
SELF	2BH	P12 DIR	59H	TD3	83H	FMEAS	A1H
(☞**8**) _8HZ	3CH	P12 F	5AH	TD4	84H	ECAP	A2H
(☞**8**) _4HZ	3DH	P12 IE	5BH	TD5	85H	CL0	A4H
(☞**8**) _2HZ	3EH	P13 MOD	5CH	TD6	86H	CL1	A5H
(☞**8**) _1HZ	3FH	P13 DIR	5DH	TD7	87H	CAPF	A7H
P00MOD	40H	P13 F	5EH	TD8	88H	TMOVF	A8H
P00DIR	41H	P13 IE	5FH	TD9	89H	CH0	B0H
P00F	42H	P20 MOD	60H	TD10	8AH	CH1	B1H
P00IE	43H	P20 DIR	61H	TD11	8BH	ICH0	B2H
P01MOD	44H	TXC	62H	TD12	8CH	ICH1	B3H
P01DIR	45H	P20 IE	63H	TD13	8DH	SOPP0	B5H
P01F	46H	P21 MOD	64H	TD14	8EH	ENADC	B6H
P01IE	47H	P21 DIR	65H	TD15	8FH	–	–

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

Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value	Bit Symbol	Value
ENOPA	B7H	QAD	CBH	TB3	18BH	RB6	19EH
SMCH0	B8H	QTM	CCH	TB4	18CH	RB7	19FH
SMCH1	B9H	QXI1	CDH	TB5	18DH	BRT1	1A0H
SMODE	BAH	QST	CEH	TB6	18EH	BRT0	1A1H
ADST0	BCH	QSR	CFH	TB7	18FH	FERR	1A4H
ADST1	BDH	EXI0	D0H	SRMOD	190H	OERR	1A5H
ADOVF	BEH	QXI0	D8H	SRL1	191H	PERR	1A6H
ADPOL	BFH	QWDT	D9H	SRL0	192H	BFULL	1A7H
E1HZ	C0H	STMOD	180H	SREN	193H	MI	1F0H
E16HZ	C1H	STL1	181H	SRCLK	194H	HLT	1F4H
E32HZ	C2H	STL0	182H	SRPEN	195H	SP1	1F9H
EAD	C3H	STSTB	183H	SRPOE	196H	SP2	1FAH
ETM	C4H	STCLK	184H	SRLMB	197H	SP3	1FBH
EXI1	C5H	STPEN	185H	RB0	198H	SP4	1FCH
EST	C6H	STPOE	186H	RB1	199H	SP5	1FDH
ESR	C7H	STLMB	187H	RB2	19AH	SP6	1FEH
Q1HZ	C8H	TB0	188H	RB3	19BH	–	–
Q16HZ	C9H	TB1	189H	RB4	19CH	–	–
Q32HZ	CAH	TB2	18AH	RB5	19DH	–	–

☞ 8 Note that the bit symbols of TBCR, 1HZ, 2HZ, 4HZ, and 8HZ are handled with "_" as a precedent character: _1HZ, _2HZ, _4HZ, and _8HZ.

! Bit symbols cannot be individually input for **D** command and **C** command. They are displayed by SID64K when SFR-mnemonics are input for **D/C** command.

Execution Example

```
64167> 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
64167> C A

    A : 0 -----> 5 New
64167> D A

   A       : 5
64167> C A=3 B=7 H=0D

64167> D A B H

   A       : 3    B       : 7    H       : D
64167> C IE0

    IE0 : 0
    ----- BIT -----
        E1HZ    : 0 -----> 1 New
        E16HZ   : 0 ----->  Not change
        E32HZ   : 0 -----> 0 New
        EAD     : 0 -----> 1 New

64167> D IE0


   IE0 : 9
    (EAD : 1  E32HZ : 0  E16HZ : 0  E1HZ : 1 )
64167>
```

3.1.2 Display Registration of Registers and SFR

SDF

Input Format

SDF [Δ *parm* ...... Δ *parm*] ↵

SDF [~]*ALL* ↵

parm : [~]*SFR_mnemonic*

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 (a)-(c). If the mnemonic is prefixed by '~' (tilde), then its registration will be cancelled.
If ALL is specified, then all displayable SFR mnemonics will be registered.

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

Execution Example

```
64167> SDF P2 IE0 STCONL

64167> SDF

   STCONL  P2      IE0
64167> D

   A       : 3    B       : 7    H       : D    L       : 0    X       : 0
   Y       : 0    PC      : 0000 BCF     : 0    BEF     : 0    BSR0    : 0
   BSR1    : 0    C       : 0
   STCONL  : 0    P2      : F    IE0     : 9
64167> SDF  ALL

64167> D

   A       : 3    B       : 7    H       : D    L       : 0    X       : 0
   Y       : 0    PC      : 0000 BCF     : 0    BEF     : 0    BSR0    : 0
   BSR1    : 0    C       : 0
64167> SDF

       ** Not Set Display Format
64167>
```

3.1.3 Displaying/Changing the PC (Program Counter)

DPC, CPC

Input Format

DPC ↲

DPC Δ *address* ↲

Description

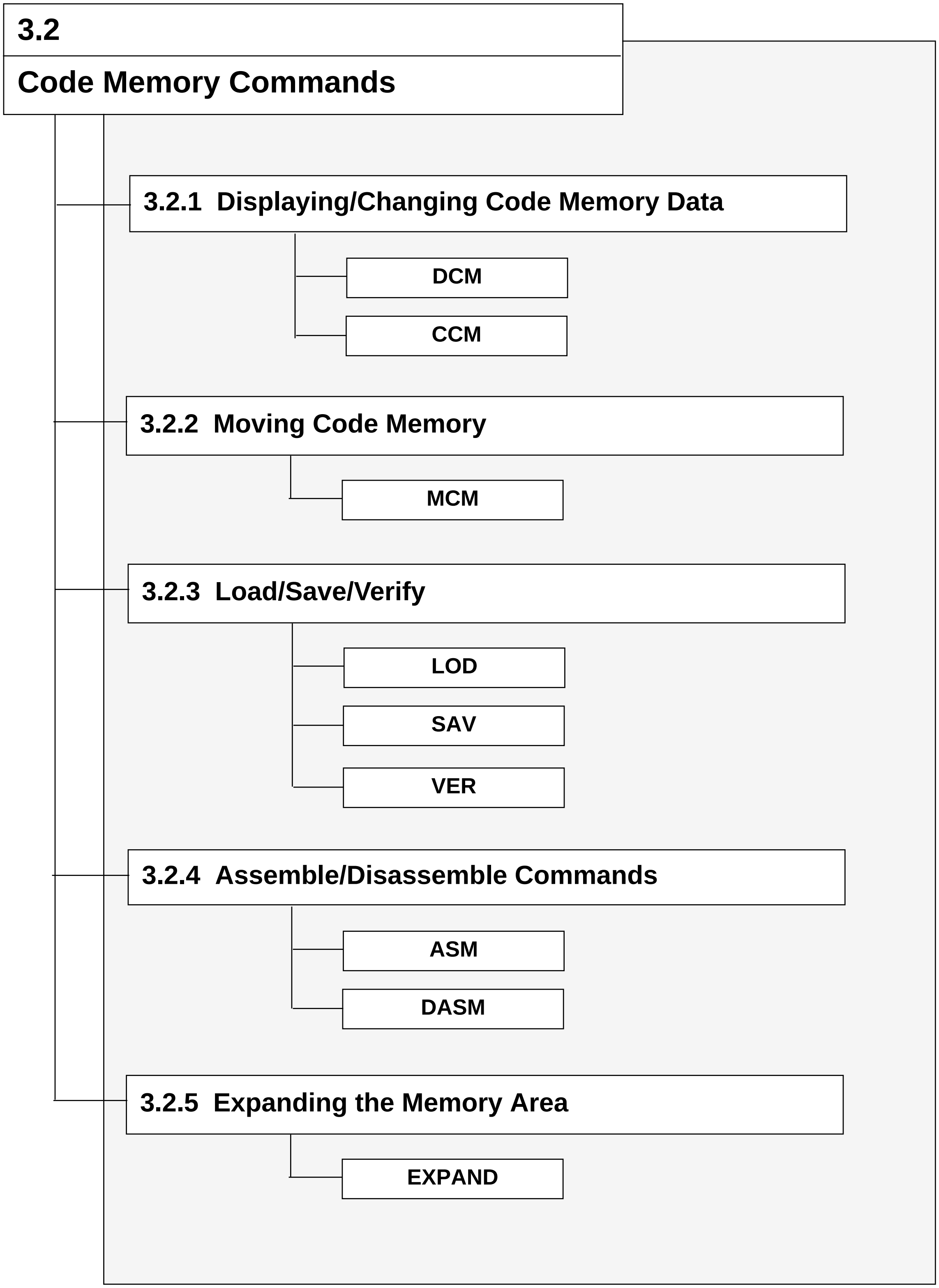
The **DPC** command displays the contents of the PC (program counter).

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

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

Execution Example

```
64167> DPC

       PC : 0000
64167> CPC 248

64167> DPC

       PC : 0248
64167> CPC 516

64167> D

   A       : 3    B       : 7    H       : D    L       : 0    X       : 0
   Y       : 0    PC      : 0516 BCF     : 0    BEF     : 0    BSR0    : 0
   BSR1    : 0    C       : 0
64167> RST E


    ***** EVA CHIP RESET *****

64167> DPC

       PC : 0000
64167>
```



3.2

Code Memory Commands

3.2.1 Displaying/Changing Code Memory Data

- DCM
- CCM

3.2.2 Moving Code Memory

- MCM

3.2.3 Load/Save/Verify

- LOD
- SAV
- VER

3.2.4 Assemble/Disassemble Commands

- ASM
- DASM

3.2.5 Expanding the Memory Area

- EXPAND

DCM EXPAND

3.2.1 Displaying/Changing Code Memory

Input Format

DCM ∆ *parm* [∆ *parm* ...... ∆ *parm*]↵

DCM ∆ * ↵

parm : *address*
: [*address* ∆ *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.

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

address	Displays the contents of one address.
[*address* ∆ *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.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

Execution Example

```
64167> DCM [100 11F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0100    90 2D 00 2D 01 2D 02 2D 30 2D 32 2D 34 BE A7 2D
       LOC = 0110    00 2D 01 2D 02 2D 03 2D 04 2D 05 2D 06 2D 07 2D
65167> DCM 125

       LOC = 125     2D
64167> DCM [200 27F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0200    28 34 28 7C 50 01 BE 31 BE A7 50 05 BE 61 AA 0A
       LOC = 0210    BE 30 50 06 BE 30 BE A0 50 01 BE 30 50 38 BE 35
       LOC = 0220    BE 3A 24 7C 24 30 24 34 90 2D 14 2D 15 2D 16 2D
       LOC = 0230    17 50 06 BE 61 AA 31 BE 30 BE A0 24 7C 2A 30 28
       LOC = 0240    7C 00 28 7D BE A7 50 02 BE 61 AA 46 BE 30 50 06
       LOC = 0250    BE 30 BE A0 24 7C 26 30 28 09 00 24 09 00 01 AA
       LOC = 0260    58 BE A7 50 06 BE 30 BE A0 90 2D 24 2D 25 2D 26
       LOC = 0270    2D 27 2D 29 2A 30 28 7C 95 2D 28 90 BE A7 50 02
64167>
```

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. 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.

However, only up to 200 items can be input consecutively.

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

" Δ ↲" (space followed by carriage return) Do not change data, and wait for change data to be input at the next address.

"– ↲" (minus followed by carriage return) Do not change data, return to the address one previous, and wait for change data to be input.

"↲" (carriage return only) Terminate the CCM command.

Operating Mode	**Address Range**
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

Execution Example

```
64167> CCM 40

       LOC = 0040    00 -----> 11  New
       LOC = 0041    00 -----> 22  New
       LOC = 0042    00 -----> 33  New
       LOC = 0043    00 -----> 44  New
       LOC = 0044    00 -----> 55  New
       LOC = 0045    00 -----> -
       LOC = 0044    55 -----> 66  New
       LOC = 0045    00 ----->     Not change
       LOC = 0046    00 ----->
64167> DCM [39 47]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0030    05 18 00 00 00 00 00 00 BC 05 1E BC 05 24 00 00
       LOC = 0040    11 22 33 44 66 00 00 00 00 00 00 00 00 00 00 00
64167> CCM 100=88 101=77 108=66

64167> DCM [100 10F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0100    88 77 00 2D 01 2D 02 2D 66 2D 32 2D 34 BE A7 2D
64167>
```

3.2.2 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 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.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

! If the data specified by [*address* ∆ *address*] cannot be completely stored at the move destination address, then no data will be moved.

Execution Example

```
64167> CCM *

64167> CCM [102 10D]=55

64167> DCM [100 10F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0100    00 00 55 55 55 55 55 55 55 55 55 55 55 55 00 00
64167> DCM [130 13F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0130    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
64167> MCM [104 109] 131

   Code Memory Copy End.
   Last Code Memory Address = 0109
64167> DCM [130 13F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0130    00 55 55 55 55 55 55 00 00 00 00 00 00 00 00 00
64167> DCM [100 10F]


                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     ------------------------------------------------
       LOC = 0100    00 00 55 55 55 55 55 55 55 55 55 55 55 55 00 00
64167>
```

3.2.3 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 (HEX) 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     Maximum
                           value of    value of
                           load        load
                           addresses   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 defined user-symbols.

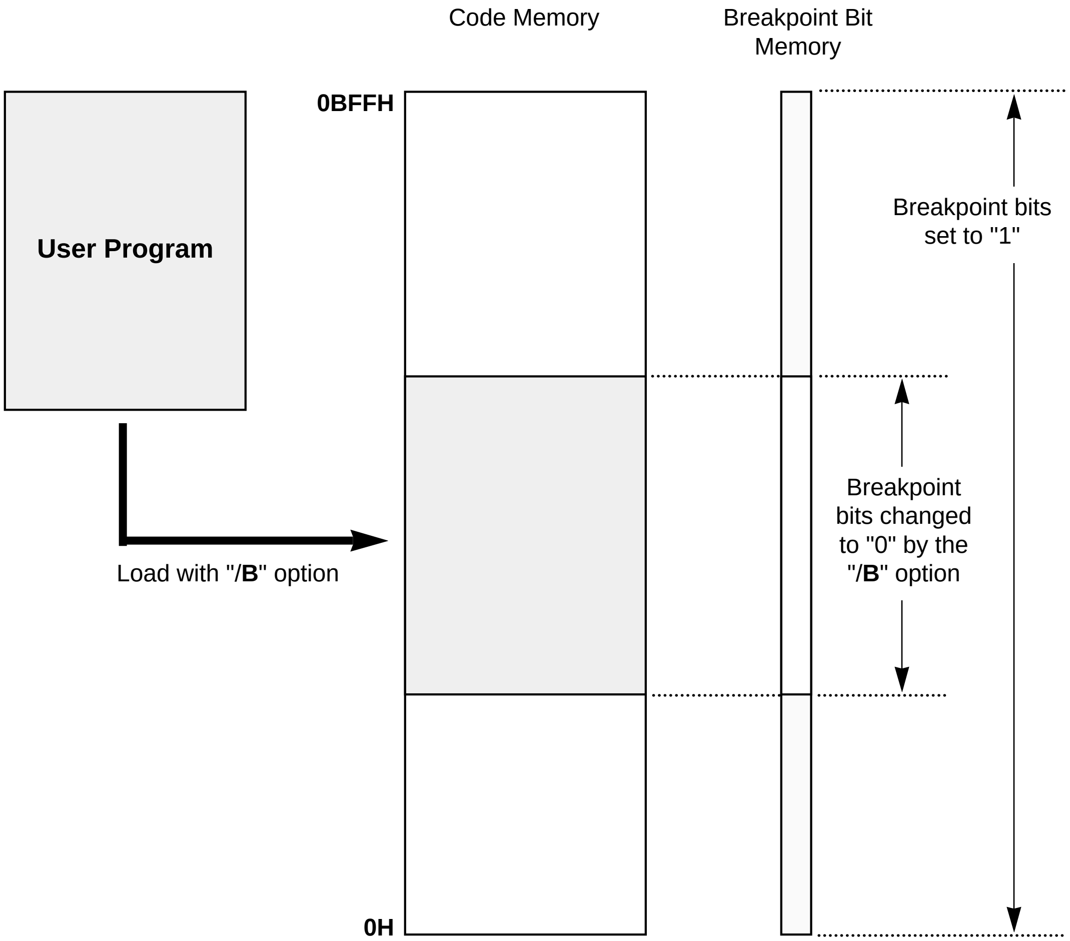
☞ **1**

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 EASE64165/167 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 EASE64165/167 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 10, "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 toned areas of breakpoint bit memory are executed. Breakpoint bit breaks need to be enabled with the **SBC** command for this to occur.

Execution Example

```
64167> LOD CHIPTP

    HEX File Loading...
    Load Completed   address[0000 - 0629]

64167> DASM 100


LOC=0100  90             LAI     P0         ;0H
LOC=0101  2D 00          LMAD    P0         ;0H
LOC=0103  2D 01          LMAD    P1         ;1H
LOC=0105  2D 02          LMAD    P2         ;2H
LOC=0107  2D 30          LMAD    IE0        ;30H
LOC=0109  2D 32          LMAD    IE1        ;32H
LOC=010B  2D 34          LMAD    IE2        ;34H
LOC=010D  BE A7          LBS0I   7H
LOC=010F  2D 00          LMAD    P0         ;0H
LOC=0111  2D 01          LMAD    P1         ;1H
LOC=0113  2D 02          LMAD    P2         ;2H
LOC=0115  2D 03          LMAD    3H
LOC=0117  2D 04          LMAD    4H
LOC=0119  2D 05          LMAD    5H
LOC=011B  2D 06          LMAD    6H

64167> CBP *=1

64167> LOD CHIPTP /B

    HEX File Loading...
    Load Completed   address[0000 - 0629]
    Break Point Bit Cleared

64167> DBP [0F0 120]


                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 00F0    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       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
64167>
```

SAV

EXPAND

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 saved. 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	: Loaded
Default extension	: *fname*.HEX
Symbol information	: Not saved
/**S** option	: Save symbol information.
/**N** option	: Do not save code information.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

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

```
64167> LOD CHIPTP

    HEX File Loading...
    Load Completed   address[0000 - 0629]

64167> SAV SAMP [0 700]

    HEX File Saving...
    Save Completed   address[0000 - 0700]

64167> SAV SAMP [0 700] /S

    File exist: delete (Y/N)Y
    Symbol Saving...
    HEX File Saving...
    Save Completed   address[0000 - 0700]

64167>
```

VER

VER

EXPAND

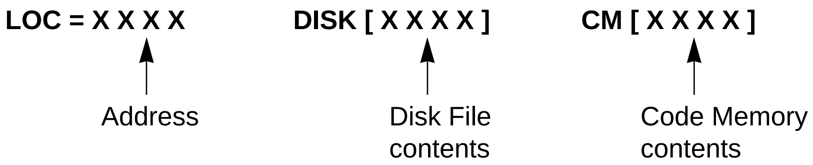
Input Format

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.



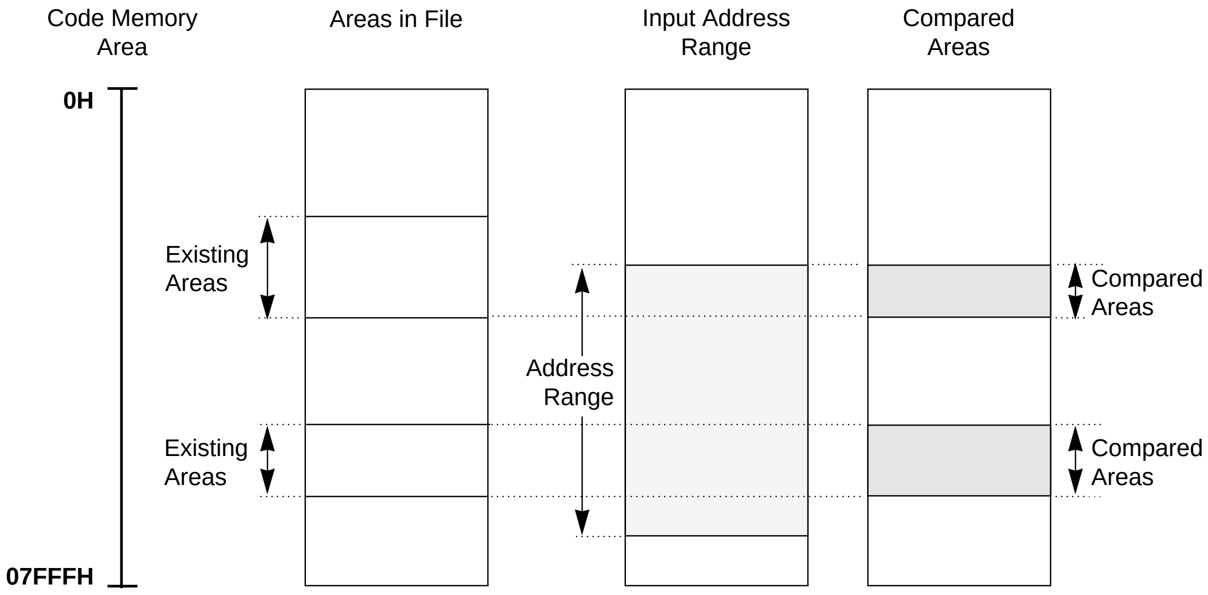
The input filename can have a path specification. If the path is omitted, then a file in the current directory will be verified. If the extension is omitted, then the default extension (HEX) will be appended to the file. To verify a file that has no extension, append a '.' after the filename (☞**1**).

[*address* Δ *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.

If the extension is omitted, then the default extension HEX will be assumed.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

☞ 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* Δ *address*] address range specified with **VER** command. Comparison is performed from low address to high address.



Execution Example

```
64167> LOD CHIPTP

    HEX File Loading...
    Load Completed   address[0000 - 0629]

64167> CCM 100

       LOC = 0100    90 -----> 1 New
       LOC = 0101    2D -----> 3 New
       LOC = 0102    00 -----> 5 New
       LOC = 0103    2D -----> 7 New
       LOC = 0104    01 ----->
64167> VER CHIPTP

    Verification Start
    LOC = 0100   Disk[0090]  CM[0001]
    LOC = 0101   Disk[002D]  CM[0003]
    LOC = 0102   Disk[0000]  CM[0005]
    LOC = 0103   Disk[002D]  CM[0007]
    Verification End

64167>
```

3.2.4 Assemble/Disassemble Commands

ASM

EXPAND

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 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.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

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.

Execution Example

(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, "Assembler Command").

(7) Operands can be coded in operands (for details, refer to Chapter 5, "Assembler 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 an 'H' after the number.	0ABCDH

When inputting a hexadecimal constant starting with a character (A - F), a '0' (zero) needs to be inserted as the first character to distinguish it from a symbol.

(11) The following assembler directives can be used (for details, refer to Chapter 5, "Assembler 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 2.3.4, "History Function."

☞ 1

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

!

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

!

Operators can be used for operand. However, the result of a division by 0 and a modulo operation will be regarded as '0.'

Execution Example

```
64167> ASM 100

line  Segment  Location  Source Statement
   1  Code     0100      LAI     1
   2  Code     0101      LAI     2
   3  Code     0102      LAI     3
   4  Code     0103      LAI     4
   5  Code     0104      LAI     5
   6  Code     0105      LHI     6
   7  Code     0107      LLI     7
   8  Code     0108      LXI     8
   9  Code     010A      LYI     9
  10  Code     010C      LHLI    23H
  11  Code     010E      LXYI    6DH
  12  Code     0110      LAM
  13  Code     0111      JP      130
  14  Code     0113      ORG     130
  15  Code     0082      ORG     130H
  16  Code     0130      ORG     111H
  17  Code     0111      JP      130H
  18  Code     0113      ORG     130H
  19  Code     0130      NOP
  20  Code     0131      ;---- TEST ----
  21  Code     0131      LHLI    20H
  22  Code     0133      LAI     0
  23  Code     0134      LMA
  24  Code     0135      JP      100H
  25  Code     0137      NOP
  26  Code     0138      NOP
  27  Code     0139      END

64167>
```

DASM

EXPAND

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.

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 the instruction.

Operating Mode	Address Range
MSM64165 mode	0 - 7DFH
MSM64167 mode	0 - 0FDFH
EXPAND mode	0 - 7FFFH

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 MSM64165/167 family microcontrollers, 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 MSM64165/167 family microcontrollers, 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 (instruction 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:

```
64167> 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.

```
64167> DASM [200 . 205] /NC /NL
       LAMD P2    ; 02H
       LMA+
       LAMD P6D   ; 06H
       LMA+
```

Symbol information displayed as comments

Execution Example

```
64167> LOD CHIPTP

    HEX File Loading...
    Load Completed   address[0000 - 0629]

64167> DASM 100


LOC=0100  90           LAI     P0          ;0H
LOC=0101  2D 00        LMAD    P0          ;0H
LOC=0103  2D 01        LMAD    P1          ;1H
LOC=0105  2D 02        LMAD    P2          ;2H
LOC=0107  2D 30        LMAD    IE0         ;30H
LOC=0109  2D 32        LMAD    IE1         ;32H
LOC=010B  2D 34        LMAD    IE2         ;34H
LOC=010D  BE A7        LBS0I   7H
LOC=010F  2D 00        LMAD    P0          ;0H
LOC=0111  2D 01        LMAD    P1          ;1H
LOC=0113  2D 02        LMAD    P2          ;2H
LOC=0115  2D 03        LMAD    3H
LOC=0117  2D 04        LMAD    4H
LOC=0119  2D 05        LMAD    5H
LOC=011B  2D 06        LMAD    6H

64167> DASM [103 111]


LOC=0103  2D 01        LMAD    P1          ;1H
LOC=0105  2D 02        LMAD    P2          ;2H
LOC=0107  2D 30        LMAD    IE0         ;30H
LOC=0109  2D 32        LMAD    IE1         ;32H
LOC=010B  2D 34        LMAD    IE2         ;34H
LOC=010D  BE A7        LBS0I   7H
LOC=010F  2D 00        LMAD    P0          ;0H
LOC=0111  2D 01        LMAD    P1          ;1H

64167>
```

3.2.5 Expanding the Memory Area

EXPAND

Input Format

EXPAND [∆ *mnemonic*] ↵

Description

The **EXPAND** command changes the area of the EASE64165/167 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 MSM64165/167 family microcontrollers (☞**2**)

The EASE64165/167 code memory, attribute memory, and instruction executed bit memory areas are set to the maximum address of the appropriate MSM64165 or MSM64167 microcontroller during initialization.

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

After this command is input, the EASE64165/167 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 3-4 shows the affected commands. The description of the affected commands will be indicated with the following mark.

EXPAND

☞ **2** Refer to the user's manuals for the maximum addresses of the MSM64165 and MSM64167. However, be aware that the emulator handles the test data area of the program area as an unusable area.

! When the setting is changed by the **EXPAND** command, the evaluation chip is reset.3

When ON (memory expansion) the prompt is changed to the chip name appended by 'S.'

Example:

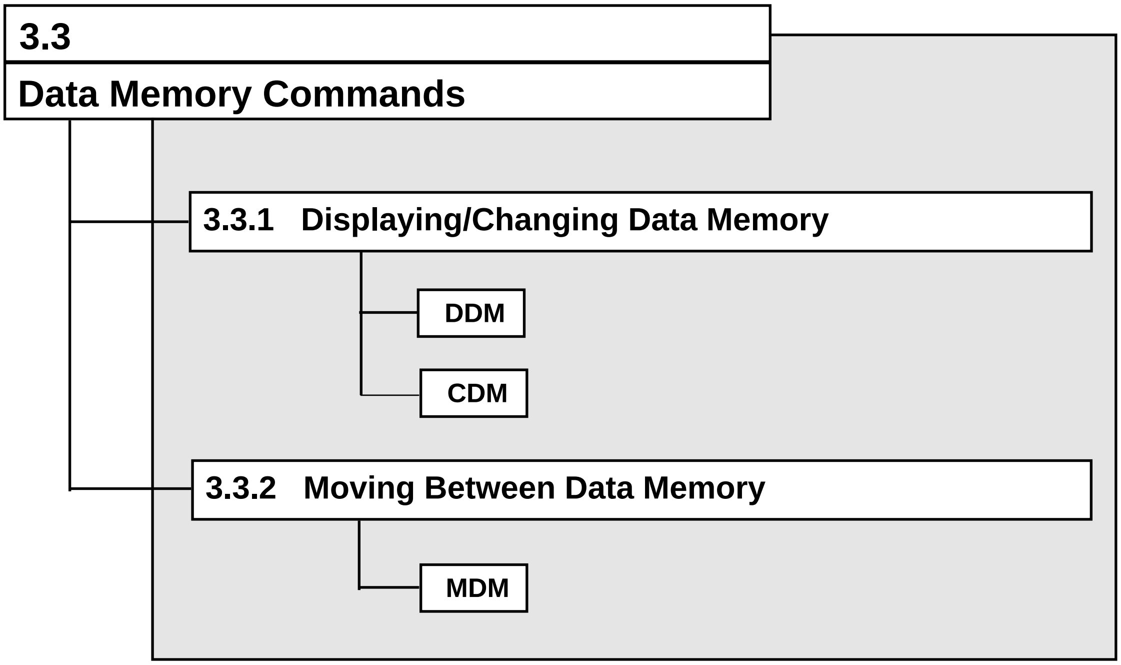
```
64167> ——► 64167S>
```

Table 3-4. 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	
Emulating Commands	7FFFH
STP, G	
Break Commands	7FFFH
DBC, CBP	
Trace Commands	7FFFH
STT, DTR, CTR	
Performance / Coverage Commands	7FFFH
SCT, DIE, CIE, CAP	
EPROM Programmer Commands	7FFFH
PPR, TPR, VPR	

Execution Example

```
64167> EXPAND ON


    CODE Memory has been expanded 32K byte.
    ***** EVA CHIP RESET *****

64167S> EXPAND

    ON MODE
64167S> EXPAND OFF


    Expanded CODE Memory allocation was released.
    ***** EVA CHIP RESET *****

64167> EXPAND

    OFF MODE
64167>
```



3.3 Data Memory Commands

3.3.1 Displaying/Changing Data Memory

- DDM
- CDM

3.3.2 Moving Between Data Memory

- MDM

3.3.1 Displaying/Changing Data Memory

DDM, CDM

Input Format

DDM Δ *parm1* [Δ *parm1* ..... Δ *parm1*] ↵

DDM Δ * ↵

parm1 : *address*

[*address* Δ *address*]

CDM Δ *parm2* [Δ *parm2* ..... Δ *parm2*] ↵

parm2 : *address* [= *data*]

[*address* Δ *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. An [*address_address*] indicates a range between two addresses. A '*' indicates the entire data memory area, excluding the SFR 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. An [*address* Δ*address*] indicates a range between two addresses.

The *data* is an expression that must evaluate in the range 0H to 0FH. If *data* is omitted, then the emulator outputs the following message and waits for data to be input.

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

DDM, CDM

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*.

_ ↵ (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.

Operating Mode	Address Range
MSM64165 mode	0 ~7FH (SFR area) 780~7FFH
MSM64167 mode	0 ~7FH (SFR area) 700~7FFH

! 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 7E and 7F cannot be changed with **CDM**, but must be changed as SP with the **C** command.
The **CDM** command is invalid for address 7DH (HALT mode is forcibly released when emulating is not executed).

! The maximum number of individual addresses that can be input interactively is 200.

Execution Example

```
64167> RST E

    ***** EVA CHIP RESET *****

64167> DDM *

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0000    1 F F F E 0 E E F F F F F F F F
       LOC = 0010    F 8 F F F F F F F F F F F F F F
       LOC = 0020    0 8 0 0 F C F 8 7 9 C A 5 E A 3
       LOC = 0030    F F F F F F F F F F C E 0 0 0 0
       LOC = 0040    F F F F F F F 7 F F F B F D F F
       LOC = 0050    F F F F F F F F F F F F F F F F
       LOC = 0060    F F F F F F 0 C 0 0 0 0 0 0 0 0
       LOC = 0070    F F E E F F F F F F F F F F F F

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0700    8 3 0 0 0 0 0 0 0 0 0 0 0 0 1 0
       LOC = 0710    1 6 0 0 2 0 0 4 0 0 0 0 2 0 4 0
       LOC = 0720    0 0 0 0 E 8 0 4 0 1 0 0 8 0 8 0
       LOC = 0730    0 0 2 1 0 8 0 A 0 1 0 0 0 1 4 1
       LOC = 0740    0 9 2 0 0 0 0 4 8 0 0 2 0 0 0 9
       LOC = 0750    0 4 4 0 C 0 0 0 8 0 1 8 9 2 8 1
       LOC = 0760    1 2 0 1 2 0 0 0 0 2 0 0 8 8 0 8
       LOC = 0770    0 0 1 0 0 0 0 2 2 4 8 0 0 0 0 0

                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0780    0 8 4 0 0 C 4 0 0 0 0 0 0 9 0 0
       LOC = 0790    3 1 4 0 0 8 0 0 4 C 2 2 0 8 0 0
       LOC = 07A0    0 0 0 0 0 4 0 0 1 2 0 0 8 9 2 4
       LOC = 07B0    1 2 2 4 0 F 0 0 0 1 0 0 0 8 0 0
       LOC = 07C0    0 0 2 2 0 6 8 0 4 5 0 0 2 0 0 9
       LOC = 07D0    4 0 0 0 1 8 0 0 0 0 0 0 0 0 0 1
       LOC = 07E0    2 4 0 0 8 0 8 1 1 0 0 3 0 8 0 0
       LOC = 07F0    0 2 6 5 0 0 0 6 0 8 0 0 5 8 4 0
64167> CDM [700 7FF]=0

64167> DDM [760 76F]

                     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
64167> CDM [765 77E]=5

64167> DDM [760 77F]

                     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    0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

Execution Example

```
64167> DDM 777

       LOC = 0777    5
64167> CDM 780

       LOC = 0780    0 -----> 1  New
       LOC = 0781    0 -----> 2  New
       LOC = 0782    0 -----> 3  New
       LOC = 0783    0 -----> 4  New
       LOC = 0784    0 -----> 5  New
       LOC = 0785    0 -----> 6  New
       LOC = 0786    0 ----->
64167> DDM [780 78F]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0780    0 0 0 0 0 0 0 0 0 0 6 5 4 3 2 1
64167> CDM 780

       LOC = 0780    1 ----->    Not change
       LOC = 0781    2 -----> 2  New
       LOC = 0782    3 -----> 5  New
       LOC = 0783    4 -----> 8  New
       LOC = 0784    5 -----> 0D New
       LOC = 0785    6 -----> -
       LOC = 0784    D -----> 7  New
       LOC = 0785    6 -----> 2  New
       LOC = 0786    0 -----> E
       ** Error 102: Illegal data input.
       LOC = 0786    0 ---->
64167> DDM [780 78F]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0780    0 0 0 0 0 0 0 0 0 0 2 7 8 5 2 1
64167> CDM 782=0D 785=1 78D=4

64167> DDM [780 78F]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0780    0 0 4 0 0 0 0 0 0 0 1 7 8 D 2 1
64167>
```

3.3.2 Moving Between Data Memory

MDM

Input Format

MDM Δ *parm* ↵

parm : [*address* Δ *address*]

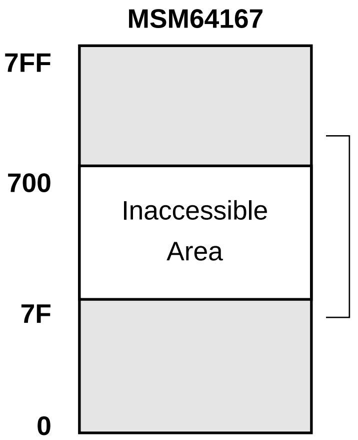
Description

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, excluding the SFR area.

Operating Mode	Address Range
MSM64165 mode	0 ~7FH (SFR area) 780~7FFH
MSM64167 mode	0 ~7FH 700~7FFH

Data cannot be transferred to or from the SFR area.

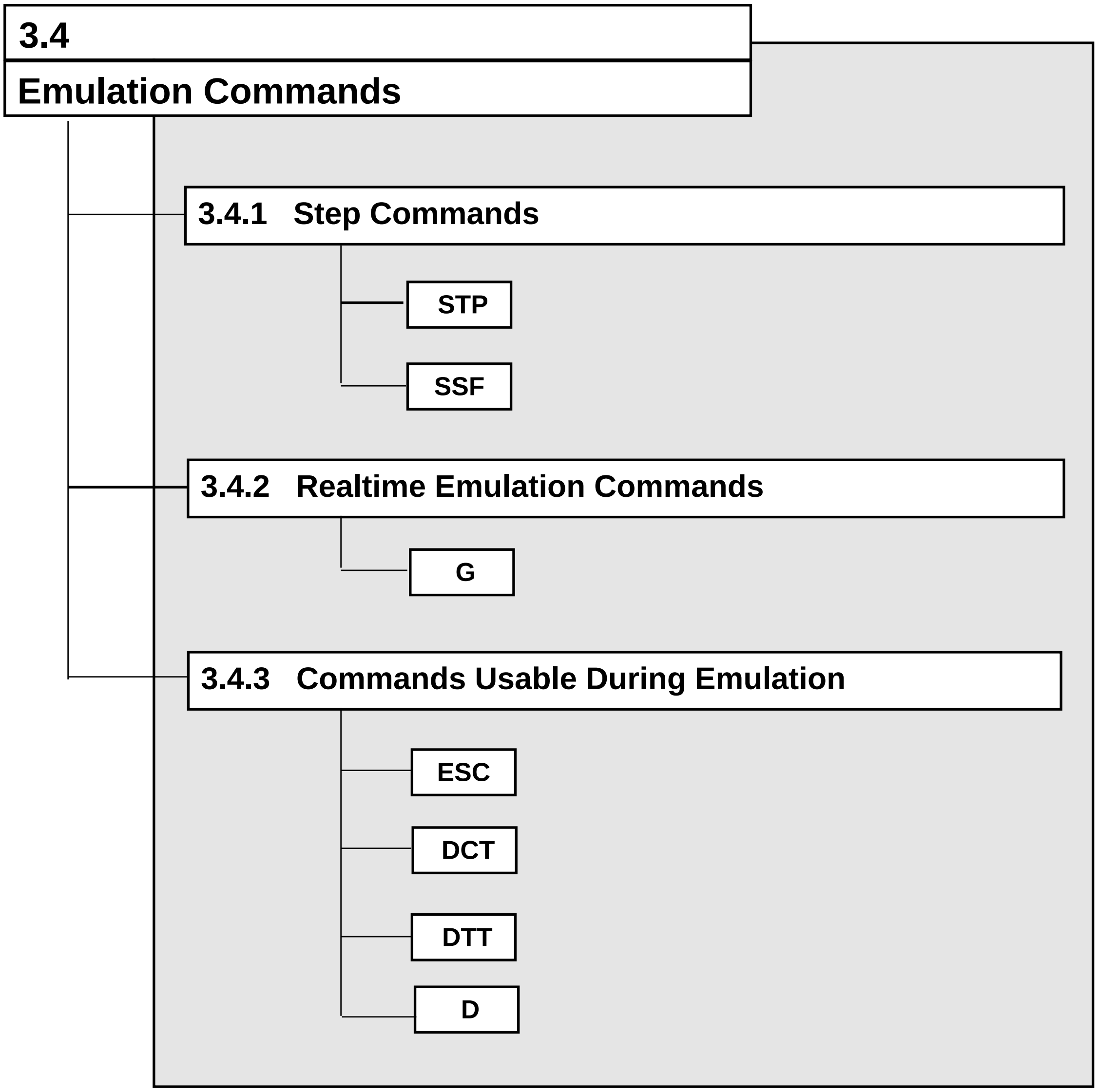


An address range that extends across this inaccessible area cannot be input.

MDM

Execution Example

```
64167> CDM [700 7FF]=0

64167> CDM [730 73F]=5

64167> DDM [720 740]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0720    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       LOC = 0730    5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
       LOC = 0740    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64167> MDM [733 73D] 751

   Data Memory Copy End.
   Last Data Memory Address = 073D
64167> DDM [750 75F]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0750    0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 0
64167> DDM [730 73F]


                     F E D C B A 9 8 7 6 5 4 3 2 1 0
                     -------------------------------
       LOC = 0730    5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
64167>
```



3.4

Emulation Commands

3.4.1 Step Commands

STP

SSF

3.4.2 Realtime Emulation Commands

G

3.4.3 Commands Usable During Emulation

ESC

DCT

DTT

D

3.4.1 Step Commands

STP

EXPAND

Input Format

STP [Δ 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. 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 (refer to ☞**2** in **SSF** command).

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

! 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.

Execution Example

```
64167> LOD CHIPTP

    HEX File Loading...
    Load Completed   address[0000 - 0629]

64167> STP 0,3


 LOC=0000  JP      100H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0



 LOC=0100  LAI     0H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0



 LOC=0101  LMAD    0H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0



64167> STP


 LOC=0103  LMAD    1H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0



64167>
```

SSF

Input Format

SSF [Δ parm ..... Δ parm] ↵

parm : [~] *mnemonic*

Description

The **SSF** command determines display format during **STP** command execution through each *mnemonic*. The following can be entered for *mnemonic*.

INS	Instruction
INSC	Instruction code
SFR-mnemonic	Refer to Table 3-2 (a), (b), (c)
A	A register
B	B register
H	H register
L	L register
X	X register
Y	Y register
C	Carry flag
INT	Flag showing interrupt operation (☞2)
SKIP	Flag showing skip execution (☞2)

RAM Δ *parm* [Δ *parm* ..... Δ *parm*]
parm : *address* , [*address* Δ *address*]

DEF	Initial state, showing **LOC, INS, A, B, H, L**

“RAM” displays the contents of data memory specified by *parm*. Up to 10 addresses or address ranges can be specified by "RAM" in all. 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 address specified by "RAM," input the specified address or address range that includes the specified address. If *mnemonic* is omitted, then the currently set format will be displayed.

! ~DEF cannot be input.

☞ 1

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH (SFR area) 780~7FFH
MSM64167 mode	0 ~0FDFH 700~7FFH

.

☞ 2 The SKIP flag indicates if skip execution of an instruction. This flag is set to “1” during skip execution (This flag will not be set to "1" in step commands).

Example:

```
   :
LAI      1
LAI      2
LAI      3
LAI      4
LMAD     7C
   :
```

In this program, if the instructions from “LAI 1” until “LMAD 7C” is step-executed, then the display of execution results for the “LAI 2,” "LAI 3," and "LAI 4" instruction will not show the SKIP flag as “1.” So, display will be just like execution of “LMAD 7C” after execution of “LAI 1.”

The INT flag indicates an interrupt operation. This flag is set to “1” during the cycle in which an interrupt transferring is executed (This flag will not be set to "1" in step commands).

```
       :
 LMAD  7C
< Interrupt operation >
 PUSH  BA
 PUSH  HL
       :
```

This program shows that an interrupt was generated during execution of “LMAD 7C”, and that execution transferred to the interrupt routine. In this case, step execution of the “LMAD 7C” will also complete the interrupt transferring routine.

SSF

Execution Example

```
64167> SSF IE0 IE1 IE2

64167> STP , 4


 LOC=0105  LMAD    2H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0

                             ------------------< SFR_mnemonic >---------------
                               IE0:0  IE1:0  IE2:E


 LOC=0107  LMAD    30H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0

                             ------------------< SFR_mnemonic >---------------
                               IE0:0  IE1:0  IE2:E


 LOC=0109  LMAD    32H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0

                             ------------------< SFR_mnemonic >---------------
                               IE0:0  IE1:0  IE2:E


 LOC=010B  LMAD    34H
                             ------------------< Registers >------------------
                               A:0  B:0  H:0  L:0

                             ------------------< SFR_mnemonic >---------------
                               IE0:0  IE1:0  IE2:E


64167>
```

3.4.2 Realtime Emulation Commands

G

EXPAND

Input Format

G [∆ *Emu_start_addr*] [, *Break_parm*] ↵

Emu_start_addr : Start address for realtime emulation

Break_parm : *address* [∆ *address* ..... ∆ *address*]
: [*address* ∆ *address*]
: *address* [*count*]
: / *address* / *address* [/ *address*]
: *mnem* [*&mask*] = *data*
: *mnem* [*&mask*] = *data* [*count*]
: *mnem* [*&mask*] = *data* [∆ / *address* [∆ *address* •••]]
: *mnem* [*&mask*] = *data* [*count*] [∆ / *address* [∆ *address* •••]]
: *mnem* [*&mask*] = *data* [/ [*address* ∆ *address*]]
: *mnem* [*&mask*] = *data* [*count*] [/ [*address* ∆ *address*]]

mnem : PRB (Probe)
: RAM [∆ *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. If *Emu_start_addr* is omitted, then realtime emulation will start as the address indicated by the current program counter (PC).

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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 (☞1) 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 its corresponding bit, and set *mask* to "1" to validate it (if omitted, it will be regarded as FH or FFH).

Example: If “RAM 100H & 3H = 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 switch 2 of dipswitch 2 on the POD64165/167 is on, reset input from the user cable RESET pin will be 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. "No Breakstatus" is the break condition for this case.

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 when an interrupt is generated, then the break will be held off until after the interrupt transferring routine ends. "No Breakstatus" is the break condition for this case.

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 Section 3-5, “Break Commands,” regarding break conditions.

Description of ***Break_parm***

(1) Address break (specified as individual addresses)

address [Δ *address* ..... Δ 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* Δ *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 by *mask*.

(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* [∆ / *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 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*] [∆ / *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 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.

Data match break will occur on a different number of times specified by *count* when it is executed under the following condition.

```
64167> ASM0
 1 Code  0000  LBS0I 7
 2 Code  0002  LHLI 60
 3 Code  0004  LMA
 4 Code  0005  LMA
 5 Code  0006  LMA
 6 Code  0007  END

64167>
```

In a program contains repeated writing instruction to data memory (LMA) like shown at the left, if the value 5 or 6 is specified by *address*, a break will occur on 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* 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 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

```
64167> LOD CHIPTP /S

    Symbol table clear (Y/N) Y
    Symbol Loading...
    HEX File Loading...
    Load Completed   address[0000 - 0648]

64167> DASM 100


LOC=0100           SET_1:
LOC=0100  90             LAI      P0        ;0H
LOC=0101  2D 00          LMAD     P0        ;0H
LOC=0103  2D 01          LMAD     P1        ;1H
LOC=0105  2D 02          LMAD     P2        ;2H
LOC=0107  2D 30          LMAD     IE0       ;30H
LOC=0109  2D 32          LMAD     IE1       ;32H
LOC=010B  2D 34          LMAD     IE2       ;34H
LOC=010D  BE A7          LBS0I    7H
LOC=010F  2D 00          LMAD     P0        ;0H
LOC=0111  2D 01          LMAD     P1        ;1H
LOC=0113  2D 02          LMAD     P2        ;2H
LOC=0115  2D 03          LMAD     3H
LOC=0117  2D 04          LMAD     4H
LOC=0119  2D 05          LMAD     5H
LOC=011B  2D 06          LMAD     6H

64167> G 0, 117

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0117], Next PC =[0119], TP=[0014]
```

Execution Example

```
64167> DASM


LOC=011D  2D 07          LMAD      7H
LOC=011F  2D 08          LMAD      VSSLCON   ;8H
LOC=0121  2D 09          LMAD      FCON      ;9H
LOC=0123  2D 10          LMAD      P00CON    ;10H
LOC=0125  2D 20          LMAD      TMD0      ;20H
LOC=0127  2D 30          LMAD      IE0       ;30H
LOC=0129  2D 31          LMAD      IRQ0      ;31H
LOC=012B  2D 32          LMAD      IE1       ;32H
LOC=012D  2D 33          LMAD      IRQ1      ;33H
LOC=012F  BE A0          LBS0I     P0        ;0H
LOC=0131  24 7C          RMBD      MIEF,  0H      ;7CH
LOC=0133           SET_1HZ:
LOC=0133  28 30          SMBD      IE0,  0H       ;30H
LOC=0135  28 7C          SMBD      MIEF,  0H      ;7CH
LOC=0137  BE A7          LBS0I     7H
LOC=0139           LOOP_1HZ:
LOC=0139  50 00          LHL1      P0        ;0H

64167> G, SET_1HZ


    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0133], Next PC =[0135], TP=[0028]
64167>
```

3.4.3 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.

```
***** 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

```
64167> G 0

    Reset Trace Pointer

    ***** Emulation Go *****
GO >> ESC

GO >>

    ***** ESC Break *****
    Break PC =[01E4], Next PC =[01E6], TP=[6689]
64167>
```

DCT

Input Format

DCT ↵

Description

The EASE64165/167 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 3.8.1.

Execution Example

Refer to Section 3.8.1, “**DCT** command.”

DTT

Input Format

DTT ↵

Description

The EASE64165/167 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 3.6.3.

Execution Example

Refer to Section 3.6.3, “**DTT** command.”

D

Input Format

D [Δ *mnemonic* ..... Δ *mnemonic*] ↵

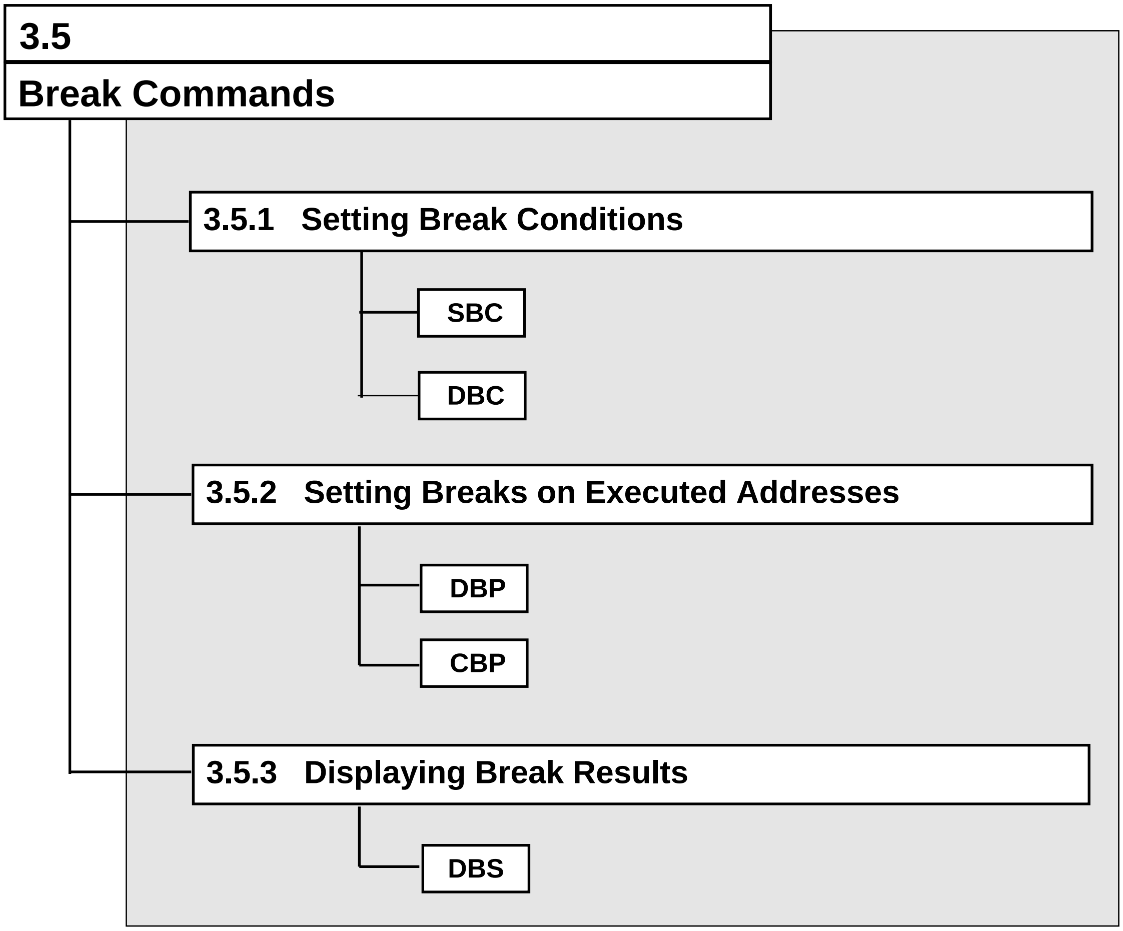
mnemonic : **A, B, X,Y, H, L, PC**

Description

The EASE64165/167 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, "**D** command."



3.5

Break Commands

3.5.1 Setting Break Conditions

SBC

DBC

3.5.2 Setting Breaks on Executed Addresses

DBP

CBP

3.5.3 Displaying Break Results

DBS

3.5.1 Setting Break Conditions

SBC, DBC

Input Format

SBC [Δ *parm* ..... Δ *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.

The break conditions BP and PD will be set when power is applied.

SBC, DBC

Execution Example

```
64167> 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
64167> DBC

    Break Condition -----> BP TF PD
64167> SBC ~TF ~PD CC

64167> DBC

    Break Condition -----> BP CC
64167>
```

3.5.2 Setting Breaks on Executed Addresses

DBP, CBP

EXPAND

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*
: [*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.

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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.

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* ∆ *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

Attribute memory 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."

Execution Example

```
64167> DBP [100 170]


                        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
64167> DBP 300

       LOC = 0300       0
64167> DBP 120 230 451

       LOC = 0120       0
       LOC = 0230       0
       LOC = 0451       0
64167> CBP [105 119]=1

64167> DBP [100 120]


                        0 1 2 3 4 5 6 7 8 9 A B C D E F
                        -------------------------------
       LOC = 0100       0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0110       1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
       LOC = 0120       0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64167> CBP 108=0 112=0 120=1 101=1


                        0 1 2 3 4 5 6 7 8 9 A B C D E F
                        -------------------------------
       LOC = 0100       0 1 0 0 0 1 1 1 0 1 1 1 1 1 1 1
       LOC = 0110       1 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0
       LOC = 0120       1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
64167> CBP *=1
```

DBP , CBP

Execution Example

```
64167> DBP [0F0 120]


                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 00F0    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0100    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0110    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       LOC = 0120    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
64167> LOD CHIPTP /B

    HEX File Loading...
    Load Completed   address[0000 - 0648]
    Break Point Bit Cleared

64167> DBP [0F0 120]


                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 00F0    1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
       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
64167> CBP *

64167> DBP [0F0 120]


                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 00F0    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       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
64167>
```

3.5.3 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 Count Break	Break on the parameters from the **G** command.
RAM Data Match Break	Break on the parameters from the **G** command (☞1).
Probe Data Match Break	Break on the parameters from the **G** command.
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 MSM64165/167 evaluation chip enters halt mode.
External Break	Break on an external break signal (☞2).
N Area Break	Break on execution of non-existent code memory area.
No Break Status	Indicates no break conditions.

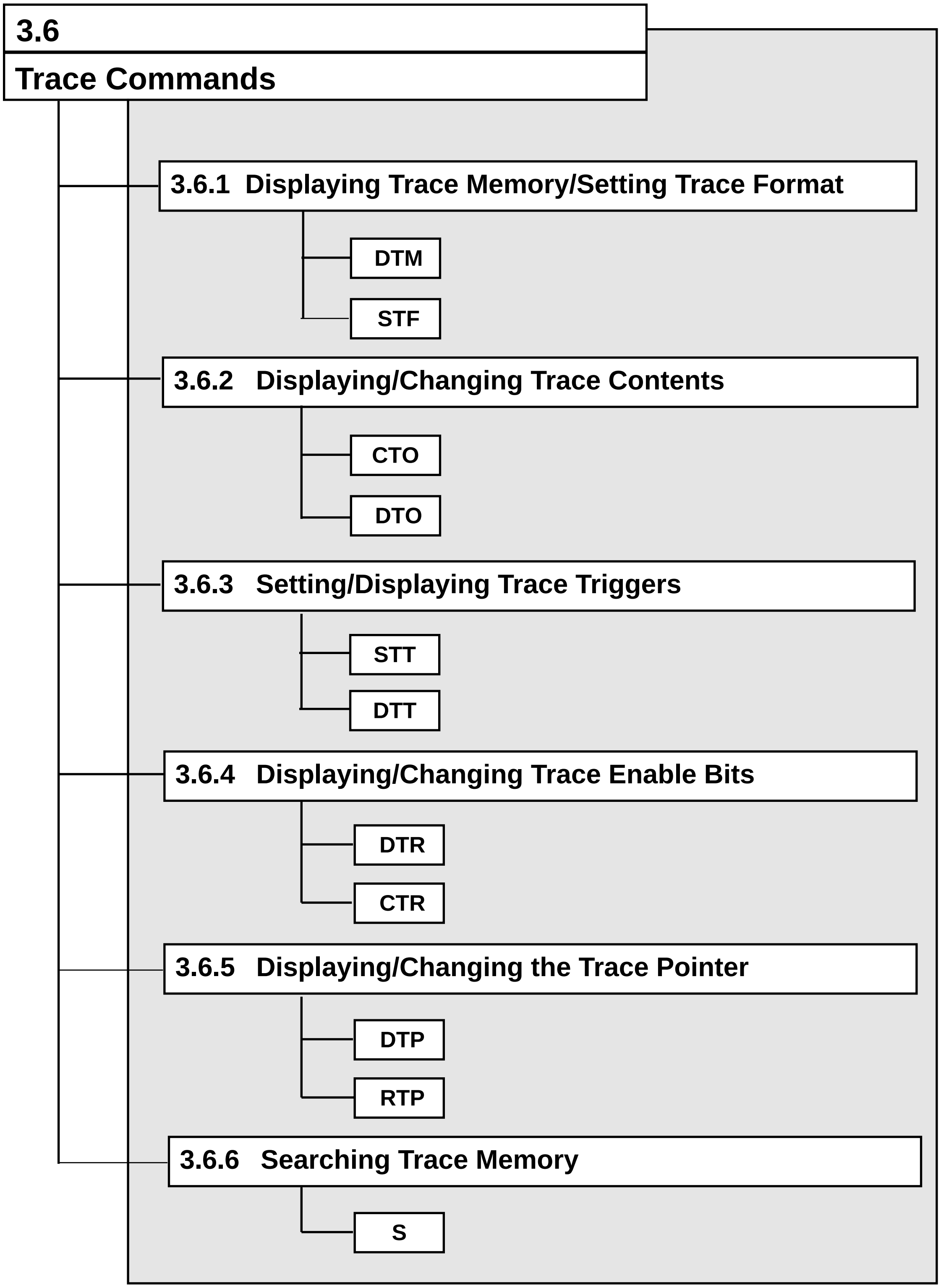
☞ 1 The timing of a RAM Data Match Break is such that the break will occur after execution of the next instruction following the instruction that satisfied the break condition.

☞ 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 "No Break Status" will be set when power is applied.

Execution Example

```
64167> G 0, LOOP_1HZ

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0139], Next PC =[013B], TP=[0031]
64167> DBS


    ***** Address Break *****
64167> G 0, LOOP_1HZ[3]

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Pass Count Break *****
    Break PC =[0139], Next PC =[013B], TP=[0037]
64167> DBS


    ***** Address Pass Count Break *****
64167> G


    ***** Emulation Go *****
GO >> ESC

GO >>

    ***** ESC Break *****
    Break PC =[01E6], Next PC =[01E2], TP=[0411]
64167> DBS


    ***** ESC Break *****
64167>
```



3.6 Trace Commands

3.6.1 Displaying Trace Memory/Setting Trace Format

DTM

STF

3.6.2 Displaying/Changing Trace Contents

CTO

DTO

3.6.3 Setting/Displaying Trace Triggers

STT

DTT

3.6.4 Displaying/Changing Trace Enable Bits

DTR

CTR

3.6.5 Displaying/Changing the Trace Pointer

DTP

RTP

3.6.6 Searching Trace Memory

S

3.6.1 Displaying Trace Memory/Setting Trace Format

DTM

Input Format

DTM ∆ *parm* ↵

parm : - *number*$_{\textit{-step}}$ ∆ *number*$_{\textit{step}}$
: *number*$_{\textit{Tp}}$ ∆ *number*$_{\textit{step}}$
: *

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*$_{\textit{-step}}$ indicates a number of steps back from the current trace pointer value (called TP below). The *number*$_{\textit{step}}$ indicates the number of steps to display as a decimal number 1–8192. The *number*$_{\textit{Tp}}$ 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.

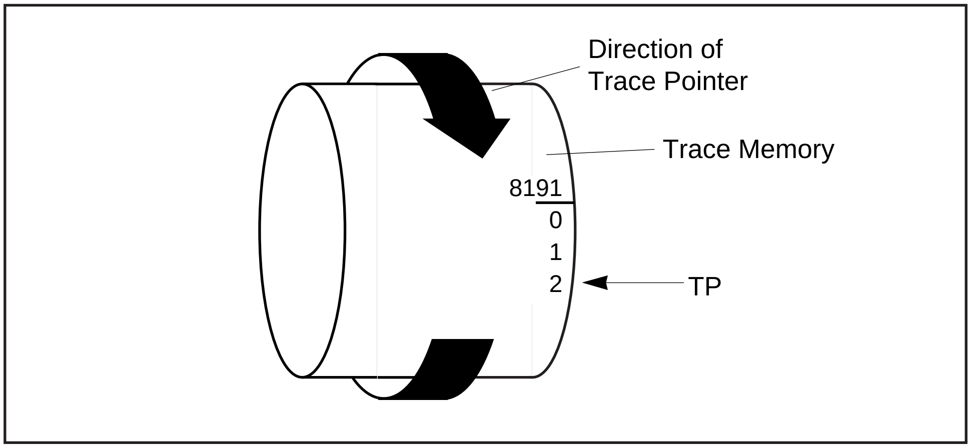
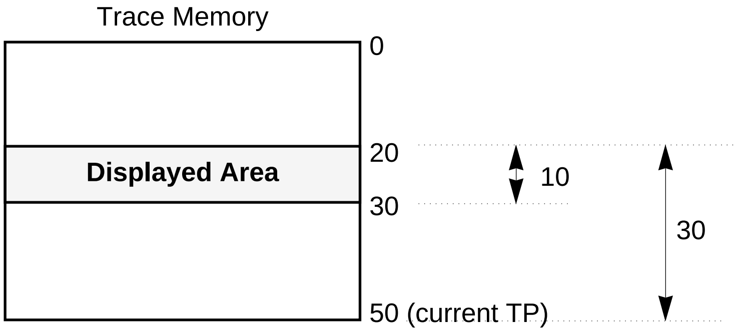


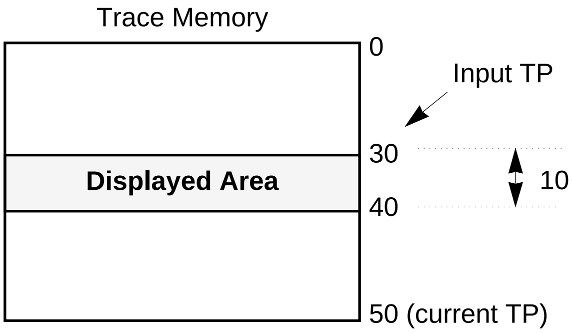
Figure 3-1. Trace Pointer Example

The following examples show the difference between input of *-number*$_{\textit{-step}}$ *number*$_{\textit{step}}$ and *number*$_{\textit{Tp}}$ *number*$_{\textit{step}}$. 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  RAMA   P0  P1 A B H L TP
```

The header is displayed every 9 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)	
ADR	Executed address	
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,DR0,DR1	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 EASE-LP2's trace memory has a maximum area of 8192 x 64 bits, but the EASE64165/167 only uses 8192 x 63 bits of it.

☞ 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 DR0 and DR1 are valid as trace data when the LCD driver segment outputs have been set as output ports by mask option. DR0 and DR1 can still be traced when the LCD driver segment outputs have not been set as output ports, but their contents are undefined.

☞ 5 The TP is always displayed. (It cannot be set with the **STF** command.)

☞ 6 The INT flag, which is set to "1" at interrupt transferring instruction, indicates an interrupt operation. So, the instruction in which the INT flag is set to "1" will actually be not executed.

☞ 7 The SKIP flag, which is set to "1" at skip execution, indicates a step execution of instruction. So, the instruction in which the SKIP flag is set to "1" will actually be not executed, but the skip operation instead.

! Except for INT and SKIP, the trace data display will delay by one instruction.

Execution Example

```
64167> LOD CHIPTP /S

    Symbol table clear (Y/N) Y
    Symbol Loading...
    HEX File Loading...
    Load Completed  address[0000 - 0648]

64167> DASM SET_HAL


LOC=0245           SET_HAL:
LOC=0245  2A 30          SMBD    IE0 ,  2H       ;30H
LOC=0247  28 7C          SMBD    MIEF ,  0H      ;7CH
LOC=0249  00             NOP
LOC=024A  28 7D          SMBD    HALT ,  0H      ;7DH
LOC=024C  BE A7          LBS0I   7H
LOC=024E           LOOP_HAL:
LOC=024E  50 02          LHLI    P2         ;2H
```

```
LOC=0250  BE 61          CMI     P1         ;1H
LOC=0252  AA 4E          JP      LOOP_HAL   ;24EH
LOC=0254  BE 30          LMI     P0         ;0H
LOC=0256  50 06          LHLI    6H
LOC=0258  BE 30          LMI     P0         ;0H
LOC=025A  BE A0          LBS0I   P0         ;0H
LOC=025C  24 7C          RMBD    MIEF ,  0H      ;7CH
LOC=025E  26 30          RMBD    IE0 ,  2H       ;30H
LOC=0260  24 7C    SET_SYS:
LOC=0260  28 09          SMBD    FCON ,  0H      ;9H

64167> G SET_HAL, SET_SYS

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0260], Next PC =[0262], TP=[0021]
64167> DTM -10 10


LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L INT SKIP TP
LOC=024E   LHLI    2H          FF 0706    2  F  F 0 0 0 6  0    0  0011
LOC=0250   CMI     1H          .. ....    .  .  . . . . 2  .    .  0012
LOC=0252   JP      24EH        .. 0702    1  .  . . . . .  .    1  0013
LOC=0254   LMI     0H          .. ....    .  .  . . . . .  .    0  0014
LOC=0256   LHLI    6H          .. ....    0  .  . . . . .  .    .  0015
LOC=0258   LMI     0H          .. ....    .  .  . . . . 6  .    .  0016
LOC=025A   LBS0I   0H          .. 0706    .  .  . . . . .  .    .  0017
LOC=025C   RMBD    7CH ,  0H   .. ....    .  .  . . . . .  .    .  0018

LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L INT SKIP TP
LOC=025E   RMBD    30H ,  2H   FF 007C    E  F  F 0 0 0 6  0    0  0019
LOC=0260   SMBD    9H ,  0H    .. 0030    0  .  . . . . .  .    .  0020

64167> DTM 0 10


LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L INT SKIP TP
LOC=0245   SMBD    30H ,  2H   FF 0006    F  F  F 0 0 0 6  0    0  0000
LOC=0247   SMBD    7CH ,  0H   .. 0030    4  .  . . . . .  .    .  0001
LOC=0249   NOP                 .. 007C    F  .  . . . . .  .    .  0002
LOC=024A   SMBD    7DH ,  0H   .. 0006    .  .  . . . . .  .    .  0003
LOC=024C   LBS0I   7H          .. ....    .  .  . . . . .  1    .  0004
LOC=0026   LJP     50CH        F7 07F9    6  .  . . . . .  0    .  0005
LOC=050C   LBS0I   7H          .. 0006    .  .  . . . . .  .    .  0006
LOC=050E   SMBD    2H ,  0H    .. ....    .  .  . . . . .  .    .  0007

LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L INT SKIP TP
LOC=0510   NOP                 F7 0702    1  F  F 0 0 0 6  0    0  0008
LOC=0511   RTI                 .. 0706    .  .  . . . . .  .    .  0009

64167>
```

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	
RAMA	Data memory address	
RAMD	Data memory data	
RAM	Data memory address and 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)
L	L register	(specified with **CTO** command) (☞ 1)
X	X register	(specified with **CTO** command)
Y	Y register	(specified with **CTO** command)
SP	Stack pointer	
P0,P1,2,DR0,DR1	Port data	(specified with **CTO** command) (☞ 1)

ALL: Sets **LOC, INS, SP, P0,P1, A, B, H, L** (☞ 2)

☞ 1 When a new port is set with the **CTO** command, the ports set with a previous **STF** command will be cancelled. Thus, trace ports will need to be set again with the **STF** command.

☞ 2 If ~ALL is input, then only the executed address (**LOC**) and instruction (**INS**) will be set.

! If EXPAND mode is set in EASE-LP mode, then C, MI, INT, and SKIP cannot be set.

Execution Example

```
64167> DTM -10 10

LOC        MNEMONIC             SP P0 P1 A B H L TP
LOC=024E   LHLI    2H           FF  F  F 0 0 0 6 0011
LOC=0250   CMI     1H           ..  .  . . . . 2 0012
LOC=0252   JP      24EH         ..  .  . . . . . 0013
LOC=0254   LMI     0H           ..  .  . . . . . 0014
LOC=0256   LHLI    6H           ..  .  . . . . . 0015
LOC=0258   LMI     0H           ..  .  . . . . 6 0016
LOC=025A   LBS0I   0H           ..  .  . . . . . 0017
LOC=025C   RMBD    7CH ,  0H    ..  .  . . . . . 0018

LOC        MNEMONIC             SP P0 P1 A B H L TP
LOC=025E   RMBD    30H ,  2H    FF  F  F 0 0 0 6 0019
LOC=0260   SMBD    9H ,  0H     ..  .  . . . . . 0020

64167> STF

LOC        MNEMONIC             SP P0 P1 A B H L TP
64167> DTO

    Set Trace Object = P0 P1 HL
64167> DTM -10 10

LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=024E   LHLI    2H           FF 0706    2  F  F 0 0 0 6 1   0    0  0011
LOC=0250   CMI     1H           .. ....    .  .  . . . . 2 .   .    .  0012
LOC=0252   JP      24EH         .. 0702    1  .  . . . . . .   .    1  0013
LOC=0254   LMI     0H           .. ....    .  .  . . . . . .   .    0  0014
LOC=0256   LHLI    6H           .. ....    0  .  . . . . . .   .    .  0015
LOC=0258   LMI     0H           .. ....    .  .  . . . . 6 .   .    .  0016
LOC=025A   LBS0I   0H           .. 0706    .  .  . . . . . .   .    .  0017
LOC=025C   RMBD    7CH ,  0H    .. ....    .  .  . . . . . .   .    .  0018

LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=025E   RMBD    30H ,  2H    FF 007C    E  F  F 0 0 0 6 0   0    0  0019
LOC=0260   SMBD    9H ,  0H     .. 0030    0  .  . . . . . .   .    .  0020

64167> DTM 3 8

LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=024A   SMBD    7DH ,  0H    FF 0006    F  F  F 0 0 0 6 1   0    0  0003
LOC=024C   LBS0I   7H           .. ....    .  .  . . . . . 0   1    .  0004
LOC=0026   LJP     50CH         F7 07F9    6  .  . . . . . .   0    .  0005
LOC=050C   LBS0I   7H           .. 0006    .  .  . . . . . .   .    .  0006
LOC=050E   SMBD    2H ,  0H     .. ....    .  .  . . . . . .   .    .  0007
LOC=0510   NOP                  .. 0702    1  .  . . . . . .   .    .  0008
LOC=0511   RTI                  .. 0706    .  .  . . . . . .   .    .  0009
LOC=024C   LBS0I   7H           FF 07FF    2  .  . . . . . 1   .    .  0010

64167>
```

3.6.2 Displaying/Changing Trace Contents

DTO, CTO

Input Format

DTO ↵

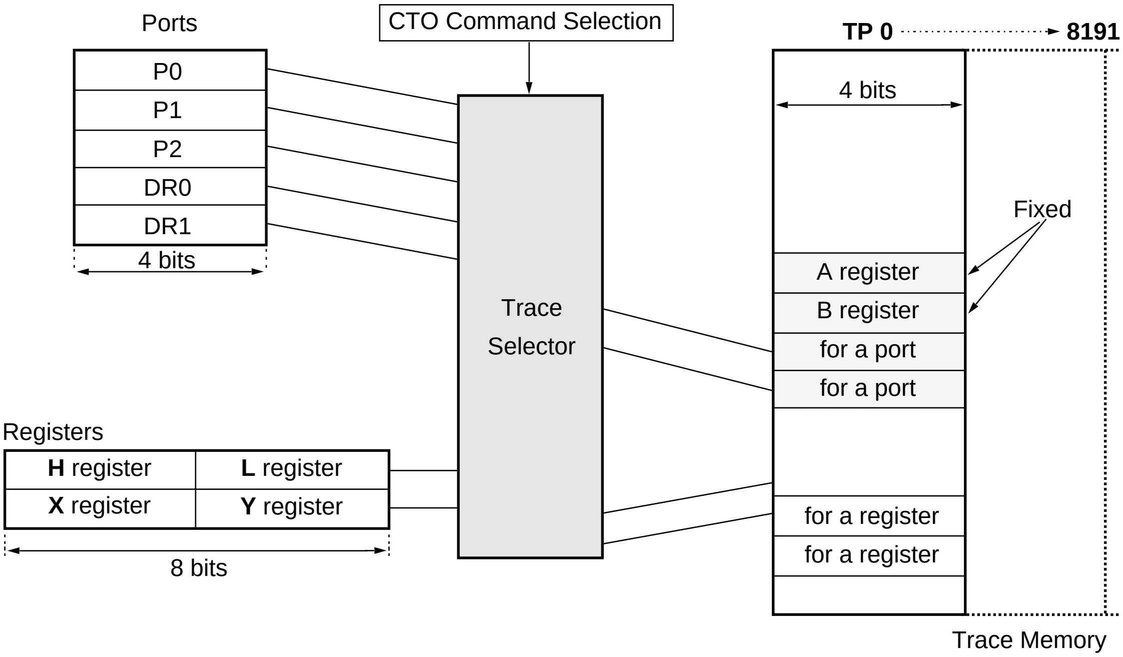
CTO Δ *parm* [Δ *parm* [Δ *parm*]] ↵
parm : *mnemonic*

Description

EASE64165/167 trace memory has an area for storing port data trace results for two ports. The operator can select any two of the five ports to trace with the **CTO** command. The **DTO** command displays the settings of 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 following can be input for *mnemonic* (☞ 2).

P0	Port 0
P1	Port 1
P2	Port 2
DR0	Display register 0 (☞ 2)
DR1	Display register 1 (☞ 2)
HL	HL register
XY	XY register

When power is turned on, the HL register and P0 and P1 ports will be set by default.

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 DR0 and DR1 are valid as trace data when the LCD driver segment outputs have been set as output ports by mask option. DR0 and DR1 can still be traced when the LCD driver segment outputs have not been set as output ports, but their contents are undefined.

DTO, CTO

Execution Example

```
64167> DTO

    Set Trace Object = P0 P1 HL
64167> STF

LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
64167> CTO P2 DR0 XY

    Trace Pointer Cleared
64167> DTO

    Set Trace Object = P2 DR0 XY
64167> STF

LOC        MNEMONIC            SP RAMA RAMD A B MI INT SKIP TP
64167> STF X Y P2 DR0

64167> STF

LOC        MNEMONIC            SP RAMA RAMD P2 DR0 A B X Y MI INT SKIP TP
64167> DTM 0 10


       ** Error 028: Trace data not ready.
64167>
```

3.6.3 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. 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.
_ ↵	Do not change the current setting.
↵	Do not change the current setting, and terminate the STT command.

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. The operator can also input one of the following keys instead of a stop address.

. <-	Stop incrementing the trace trigger when G command execution ends.
- <-	Re-enter the input.
_ <-	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* 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 "0" are ignored.

When EASE64165/167 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 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. Hereafter 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 **DT**T command displays the current trace trigger set by the **STT** command.

Execution Example

```
64167> STT ALL

64167> DTT

    Current Trace Trigger : ALL
64167> STT SS

    Current Trace Trigger : ALL

    START ---> 10

    END   ---> 20


64167> DTT

    Current Trace Trigger : SS
    START ADDRESS : 0010
    STOP  ADDRESS : 0020
64167>
```

3.6.4 Displaying/Changing Trace Enable Bits

DTR

EXPAND

Input Format

DTR Δ *parm* [Δ *parm* ..... Δ *parm*] ↵

DTR Δ * ↵

parm : *address*
: [*address* Δ *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.

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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 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 in EASE-LP mode, the EASE64165/167 executes a user program to examine the trace enable bit at the 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.

Execution Example

```
64167> 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
64167> DTR 120 0B00 0A35

       LOC = 0120    0
       LOC = 0B00    0
       LOC = 0A35    0
64167>
```

CTR

EXPAND

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 attribute memory trace enable bits.

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

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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.

Execution Example

```
64167> 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
64167> CTR [26 2B]=1

64167> 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
64167> STT TR

64167> DTT

    Current Trace Trigger : TR
64167>
```

3.6.5 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 EASE64165/167 trace pointer will be initialized to "0" in the following circumstances.

- when power is applied
- when a CTO command is executed
- when a start address is input with the G command.

Execution Example

```
64167> G 100,110

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0110], Next PC =[0111], TP=[0017]
64167> DTP

    Trace Pointer -----> 0017  Overflow = 0
64167> RTP

    Reset Trace Pointer
64167> DTP

    Trace Pointer -----> 0000  Overflow = 0
```

```
64167> G


    ***** Emulation Go *****
GO >>

    ***** Trace full Break *****
    Break PC =[0100], Next PC =[0101], TP=[0000]
64167> DTP

    Trace Pointer -----> 0000  Overflow = 1
64167> RTP

    Reset Trace Pointer
64167> DTP

    Trace Pointer -----> 0000  Overflow = 0
64167>
```

3.6.6 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, DR0, DR1**	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

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.

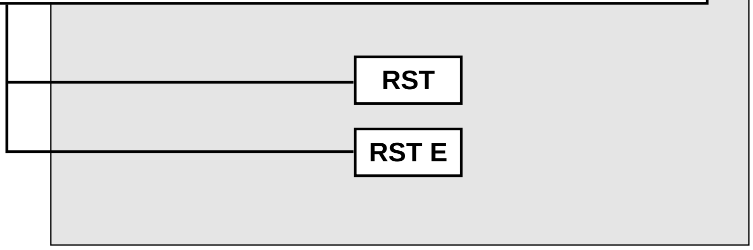
If *parm* is omitted, then the **S** command displays the contents of trace memory at the TP when the comparison criteria is satisfied the first time. The *count, start_count, and end_count* have decimal values of 1-8192.

! Both of the oldest trace data and the newest trace data will be handled in the same manner as the **DTM** command (refer to ☞**2** in **DTM** command).

Execution Example

```
64167> S SKIP=1


LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=0252   JP      24EH         FF 0702    1  F  F 0 0 0 2 1   0    1  0013

64167> S MI=1  [3 8]


LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=0258   LMI     0H           FF 0702    0  F  F 0 0 0 6 1   0    0  0016
LOC=0256   LHLI    6H           .. ....    .  .  . . . . 2 .   .    .  0015
LOC=0254   LMI     0H           .. ....    1  .  . . . . . .   .    .  0014
LOC=0252   JP      24EH         .. ....    .  .  . . . . . .   .    1  0013
LOC=0250   CMI     1H           .. 0700    2  .  . . . . . .   .    0  0012
LOC=024E   LHLI    2H           .. ....    .  .  . . . . 0 .   .    .  0011

64167> S ~MI=1  [3 8]


LOC        MNEMONIC             SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=024C   LBS0I   7H           FF 07FF    2  F  F 0 0 0 0 1   0    0  0010
LOC=024E   LHLI    2H           .. 0700    .  .  . . . . . .   .    .  0011
LOC=0250   CMI     1H           .. ....    .  .  . . . . 2 .   .    .  0012
LOC=0252   JP      24EH         .. 0702    1  .  . . . . . .   .    1  0013
LOC=0254   LMI     0H           .. ....    .  .  . . . . . .   .    0  0014
LOC=0256   LHLI    6H           .. ....    0  .  . . . . . .   .    .  0015

64167>
```

3.7
Reset Commands



RST

RST E

RST

Input Format

RST ↵

Description

The **RST** command resets the EASE64165/167 as follows.

Item	Reset State
Evaluation board	Resets to same as when a reset is input to a MSM64165 or MSM64167.
Break status	Cleared to state of no breaks generated (No Break Status).
Cycle counter	Cleared to 0.
Address pass counters 0-3	Cleared to 0.

Execution Example

```
64167> RST


    *****SYSTEM RESET*****
SID64K Symbolic Debugger Ver.1.05 Sep 1993
Copyright (C) 1993. Oki Electric Ind. Co.,Ltd.


64167>
```

SEE Refer to Table 2-11 (a)-(b) in chapter 2 regarding initialization states when power is applied or the EASE64165/167 reset switch is pressed.

RST E

Input Format

RST Δ E ↵

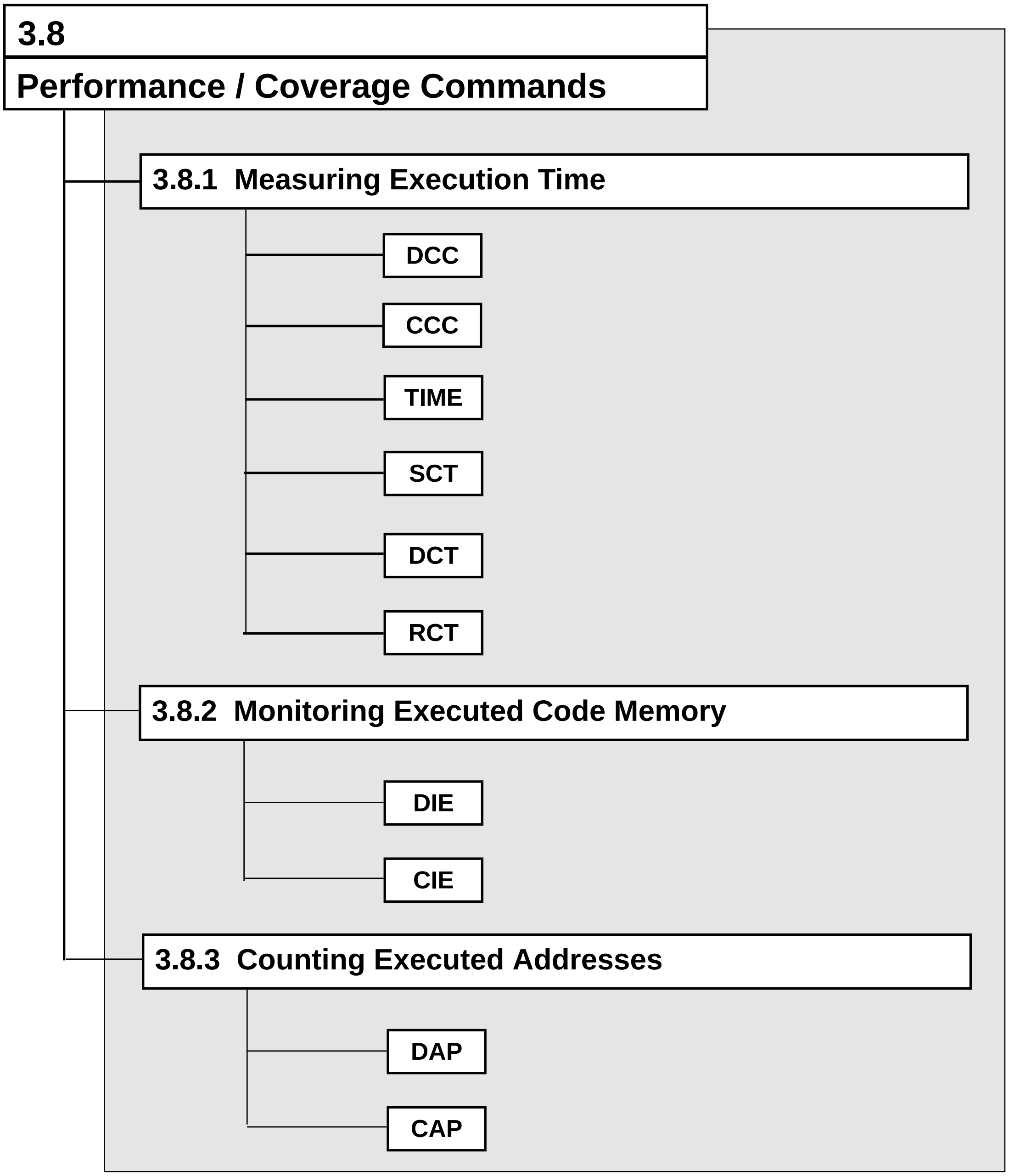
Description

The RST E command resets the evaluation board.

After this command is executed, the evaluation board will be reset to the same state as the MSM64165 or MSM64167 (refer to the MSM64165 or MSM64167 user's manual for details about its state after reset).

Execution Example

```
64167> RST E


    ***** EVA CHIP RESET *****

64167>
```


3.8
Performance / Coverage Commands
3.8.1 Measuring Execution Time
DCC
CCC
TIME
SCT
DCT
RCT
3.8.2 Monitoring Executed Code Memory
DIE
CIE
3.8.3 Counting Executed Addresses
DAP
CAP

3.8.1 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 increments by one for each machine cycle.

Execution Example

```
64167> DCC

    CURRENT STATUS -----> 0  Time = 0.0000u (sec) Overflow = 0
64167>
```

CCC

Input Format

CCC Δ [-] *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 4294967295 ······ Overflow will occur when 16 cycles have elapsed after the cycle counter is started.

CCC -100 ····················· The cycle counter is set to 4294967295. Overflow will occur when 101 cycles have elapsed after the cycle counter is started.

Execution Example

```
64167> DCC

    CURRENT STATUS -----> 0  Time = 0.0000u (sec) Overflow = 0
64167> CCC 100

64167> DCC

    CURRENT STATUS -----> 100  Time = 9.1000m (sec) Overflow = 0
64167> CCC -123

64167> DCC

    CURRENT STATUS -----> 4294967172  Time = 390842012.6520m (sec) Overflow = 0
64167>
```

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 the operating time of one cycle as 91.0 us, assuming that the microcontroller'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

```
64167> TIME

    Time = 91.0000u (sec)
64167> TIME 1000

64167> TIME

    Time = 1.0000m (sec)
64167> TIME 10.234

64167> TIME
    Time = 10.2340u (sec)
64167> TIME 0.2

64167> TIME

    Time = 0.2000u (sec)
64167>
```

SCT

EXPAND

Input Format

SCT [∆ / [*parm1*] / [*parm2*] ↵

parm1, parm2 : *address*
: [*start_address* ∆ *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.

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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* ∆ *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).

(☞ 2)

address	Stop incrementing when the specified program address is executed.
[*address* ∆ *address*]	Stop incrementing when any program address in the specified range is executed.
.	Increment continuously through **G** command execution.
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 (☞ 2). 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 (☞ 2). 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 address specified by parm2.

☞ 2 If '.' is specified for parm2, then the cycle counter will also be incremented at break addresses.

Execution Example

```
64167> DCT

     START ADDRESS : TRIGGER RESET
     STOP  ADDRESS : TRIGGER RESET

64167> SCT

     START ADDRESS : TRIGGER RESET
     STOP  ADDRESS : TRIGGER RESET

    START ---> 100

    END   ---> 140


64167> DCT

     START ADDRESS : 0100
     STOP  ADDRESS : 0140

64167> SCT /./.

64167> DCT

     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

64167> SCT

     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

    START --->
  Not Change
    END   ---> -


    START ---> 120

    END   ---> 210


64167>
```

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.

```
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.

FREE STOP Indicates that cycle counter incrementing will stop along with **G** command execution.

TRG 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 TRG RESET.

DCT, RCT

Execution Example

```
64167> DCT

     START ADDRESS : 0120
     STOP  ADDRESS : 0210

64167> SCT

     START ADDRESS : 0120
     STOP  ADDRESS : 0210

    START ---> 2A5

    END   ---> .


64167> DCT

     START ADDRESS : 02A5
     STOP  ADDRESS : STOP FREE

64167> RCT

64167> DCT

     START ADDRESS : TRIGGER RESET
     STOP  ADDRESS : TRIGGER RESET

64167> SCT /./.

64167> DCT

     START ADDRESS : FREE START
     STOP  ADDRESS : STOP FREE

64167>
```

3.8.2 Monitoring Executed Code Memory

DIE

EXPAND

Input Format

DIE ∆ *parm* [∆ *parm* ..... ∆ *parm*] ↵

DIE ∆ * ↵

parm : *address*
: [*address* ∆ *address*]

Description

The DIE command displays 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 to be displayed.

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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 instruction executed bit memory.

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

CIE

EXPAND

Input Format

CIE ∆ *parm* [∆ *parm* ..... ∆ *parm*] ↵

CIE ∆ * [=*data*]
parm : *address* = *data*
: [*address* ∆ *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.

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

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 ∆ *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.

Execution Example

```
64167> DIE [100 120]


                     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
64167> DIE 210 [0 10]

       LOC = 0210    0


                     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
64167> G 0,LOOP_1HZ

    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0139], Next PC =[013B], TP=[0031]
64167> DIE [100 120]


                     0 1 2 3 4 5 6 7 8 9 A B C D E F
                     -------------------------------
       LOC = 0100    1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
       LOC = 0110    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
       LOC = 0120    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
64167> CIE [100 110]=0

64167> DIE


                     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 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
       LOC = 0120    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
64167>
```

3.8.3 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 EASE64165/167 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

```
64167> CAP C0=128 200

64167> CAP C3=482

64167> CAP C2=100 0

64167> DAP

          AP  :   ADDRESS    COUNT   OVERFLOW
      ------+-------------------------------
          C0  :   0128         200      0
          C1  :   0000           1      0
          C2  :   0100           0      0
          C3  :   0482           0      0

64167>
```

CAP

EXPAND

Input Format

CAP *mnemonic* [= *address*] [Δ *count*]

mnemonic : **C0**, **C1**, **C2**, **C3**

The **CAP** command is provided for monitoring how often an instruction at some address is executed. The EASE64165/167 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.

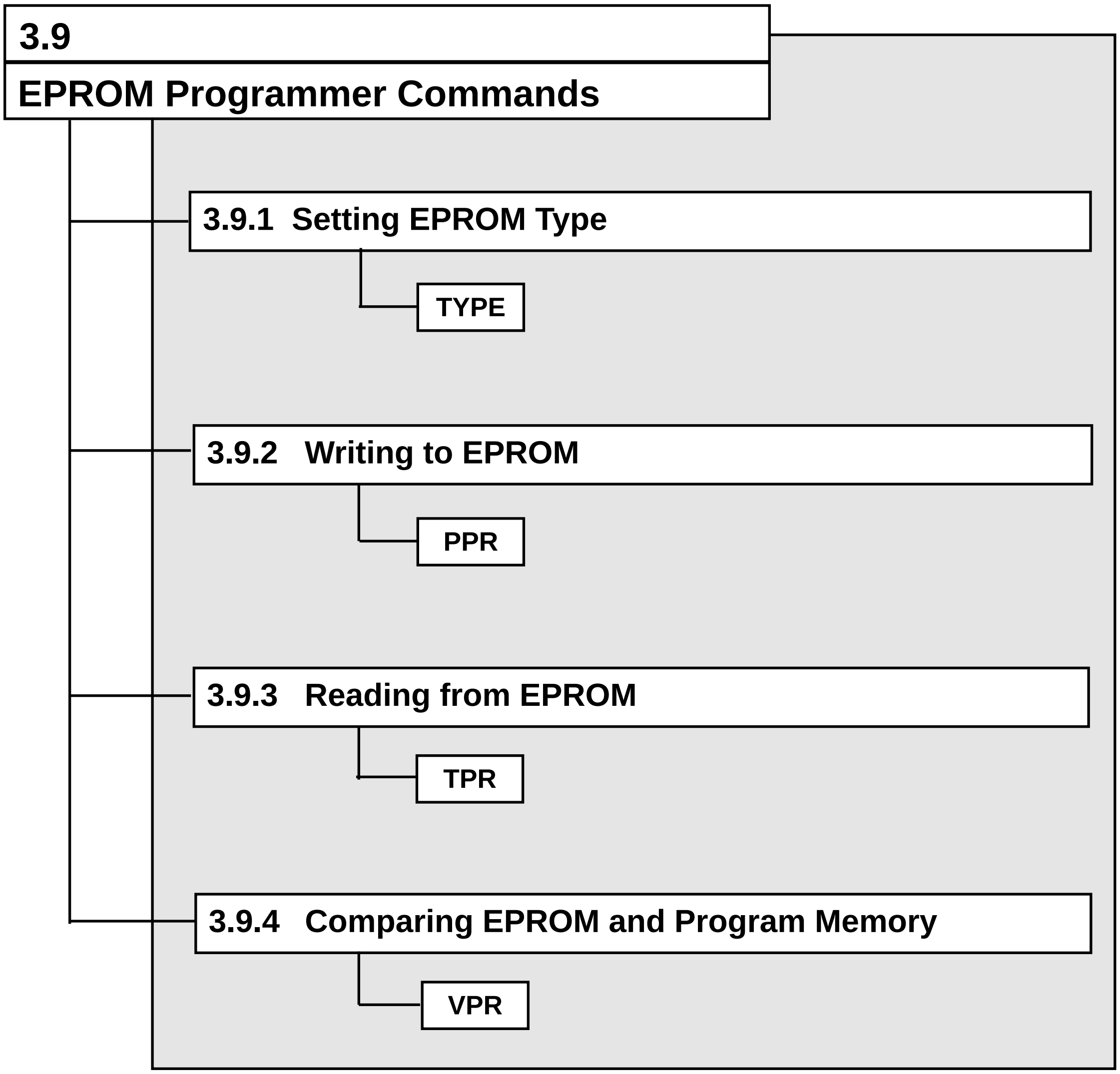
The **C0** address pass counter has an overflow break function. If the **SBC** command has been set to AP (address pass counter overflow breaks), then emulation execution will break and terminate at the point the counter value exceeds 65535.

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

Execution Example

```
64167> SBC AP

64167> CAP C1=200

64167> CAP C0=100 65525

64167>
```

3.9 EPROM Programmer Commands



3.9.1 Setting EPROM Type

TYPE

Input Format

TYPE ∆ *mnemonic* ↵

Description

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
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.

TYPE

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

```
64167> TYPE

    EPROM TYPE -----> I27512
64167> TYPE F27C128

64167> TYPE

    EPROM TYPE -----> F27C128
64167>
```

3.9.2 Writing to EPROM

PPR

EXPAND

Input Format

PPR Δ [*address* Δ *address*] [Δ *eprom-address*] ↵

PPR Δ * ↵

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.

Operating Mode	**Address Range**
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

The [*address* Δ *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.

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 EASE-LP2 "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.

PPR

☞ 1 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

```
64167> TYPE F27C512

64167> PPR [0 20]

    EPROM TYPE -----> F27C512
    PROGRAMMING VOLTAGE = 12.5 V
    FUJITSU QUICK PROGRAMMING
    START PROGRAMMING [Y/N] -----> Y

 EPROM Program End. Next EPROM Address = 0021
64167> PPR [100 135] 200

    EPROM TYPE -----> F27C512
    PROGRAMMING VOLTAGE = 12.5 V
    FUJITSU QUICK PROGRAMMING
    START PROGRAMMING [Y/N] -----> Y

 EPROM Program End. Next EPROM Address = 0236
64167>
```

3.9.3 Reading from EPROM

TPR

EXPAND

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 highest address of the transfer range must be an expression that evaluates within the maximum address range of code memory.

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

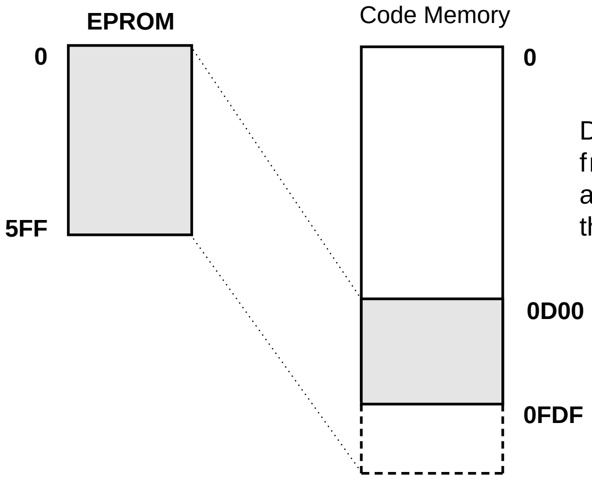
Here *type* indicates the currently set EPROM type.

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 EASE-LP2 "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] 0D00



Data will be transferred from address 0D00 to address 0FDF, and then the transfer will terminate.

Execution Example

```
64167> TPR [100 132]

    EPROM TYPE -----> F27C512
    START READING [Y/N] -----> Y

 EPROM Transfer End. Next CM Address = 0033
64167> TPR [200 254] 100

    EPROM TYPE -----> F27C512
    START READING [Y/N] -----> Y

 EPROM Transfer End. Next CM Address = 0155
64167>
```

3.9.4 Comparing EPROM and Program Memory

VPR

EXPAND

Input Format

VPR ∆ [*address* ∆ *address*] [∆ *eprom-address*] ↵

VPR ∆ * ↵

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* ∆ *address*] specifies the range of code memory to be compared.

Operating Mode	Address Range
MSM64165 mode	0 ~7DFH
MSM64167 mode	0 ~0FDFH
EXPAND mode	0 ~7FFFH

If '*' is input, then a range of code memory that corresponds to the EPROM type will be set (☞ 1).

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.

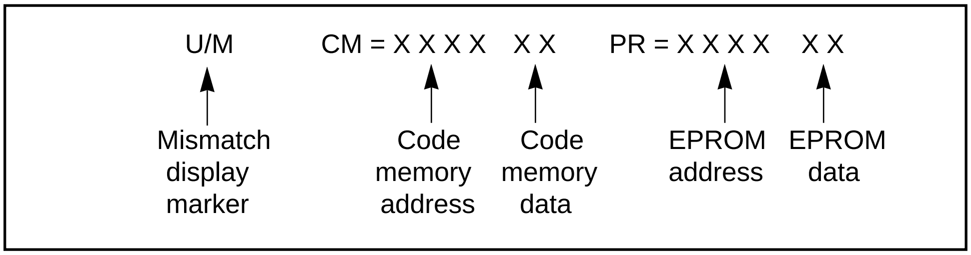
```
EPROM TYPE ---> type
START READING [Y/N] ---> _
```

Here *type* indicates the currently set EPROM type. 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 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 EASE64165/167 "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 (☞ 2).



```
U/M     CM = X X X X   X X     PR = X X X X   X X
```

☞ 1 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.

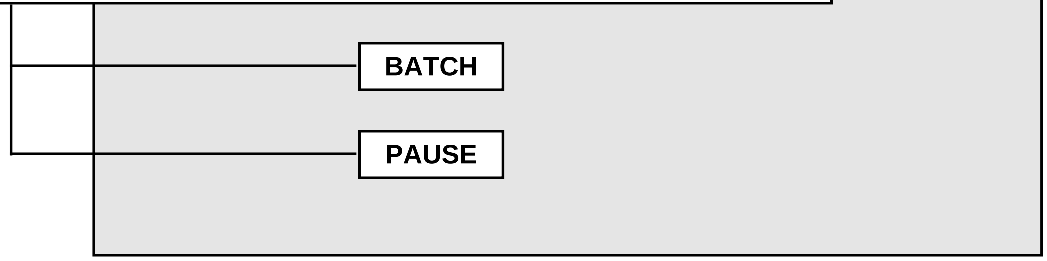
☞ 2 If the number of comparison errors exceeds 100, then verification will automatically stop, and the emulator will return to the prompt.

Execution Example

```
64167> VPR [0 10]

    EPROM TYPE -----> F27C512
    START READING [Y/N] -----> Y
  U/M     CM = 0000  A9    PR = 0000  FF
  U/M     CM = 0001  00    PR = 0001  FF

 EPROM Verify End. Next EPROM Address = 0011
64167> VPR [30 50] 100

    EPROM TYPE -----> F27C512
    START READING [Y/N] -----> Y
  U/M     CM = 0030  05    PR = 0100  FF
  U/M     CM = 0031  18    PR = 0101  FF
  U/M     CM = 0032  BC    PR = 0102  FF
  U/M     CM = 0033  05    PR = 0103  FF
  U/M     CM = 0034  2F    PR = 0104  FF
  U/M     CM = 0035  BC    PR = 0105  FF
  U/M     CM = 0036  05    PR = 0106  FF
  U/M     CM = 0037  35    PR = 0107  FF

  U/M     CM = 0038  BC    PR = 0108  FF
  U/M     CM = 0039  05    PR = 0109  FF
  U/M     CM = 003A  1E    PR = 010A  FF
  U/M     CM = 003B  BC    PR = 010B  FF
  U/M     CM = 003C  05    PR = 010C  FF
  U/M     CM = 003D  24    PR = 010D  FF

 EPROM Verify End. Next EPROM Address = 0121
64167> VPR [0E0 0FF] 0E0

    EPROM TYPE -----> F27C512
    START READING [Y/N] -----> Y

 EPROM Verify End. Next EPROM Address = 0100
64167>
```

3.10

Commands for Automatic Command Execution



BATCH

PAUSE

BATCH

Input Format

BATCH Δ *fname* ↵

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

Description

The **BATCH** command automatically executes the contents of the specified *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 switch is pressed during **BATCH** command execution, the command execution will be ended and the batch file will be closed.

BATCH

Execution Example

```
64167> BATCH BAT

     Batchfile: BAT.CMD opened
64167> D
   A       : 0    B       : 0    H       : 0    L       : 0    X       : 0
   Y       : 0    PC      : 013B BCF     : 0    BEF     : 0    BSR0    : 7
   BSR1    : 0    C       : 0
64167> RST E

    ***** EVA CHIP RESET *****

64167> 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
64167> G 0,LOOP_1HZ
    Reset Trace Pointer

    ***** Emulation Go *****
GO >>

    ***** Address Break *****
    Break PC =[0139], Next PC =[013B], TP=[0031]
64167> DTM -3 3

LOC        MNEMONIC            SP RAMA RAMD P0 P1 A B H L MI INT SKIP TP
LOC=0135   SMBD    7CH ,  0H   FF 0030    1  F  F 0 0 0 0 0   0    0  0028
LOC=0137   LBS0I   7H          .. 007C    F  .  . . . . . 1   .    .  0029
LOC=0139   LHL1    0H          .. 0000    .  .  . . . . . .   .    .  0030

      Batchfile: BAT.CMD closed
64167>
```

PAUSE

Input Format

PAUSE ↵

Description

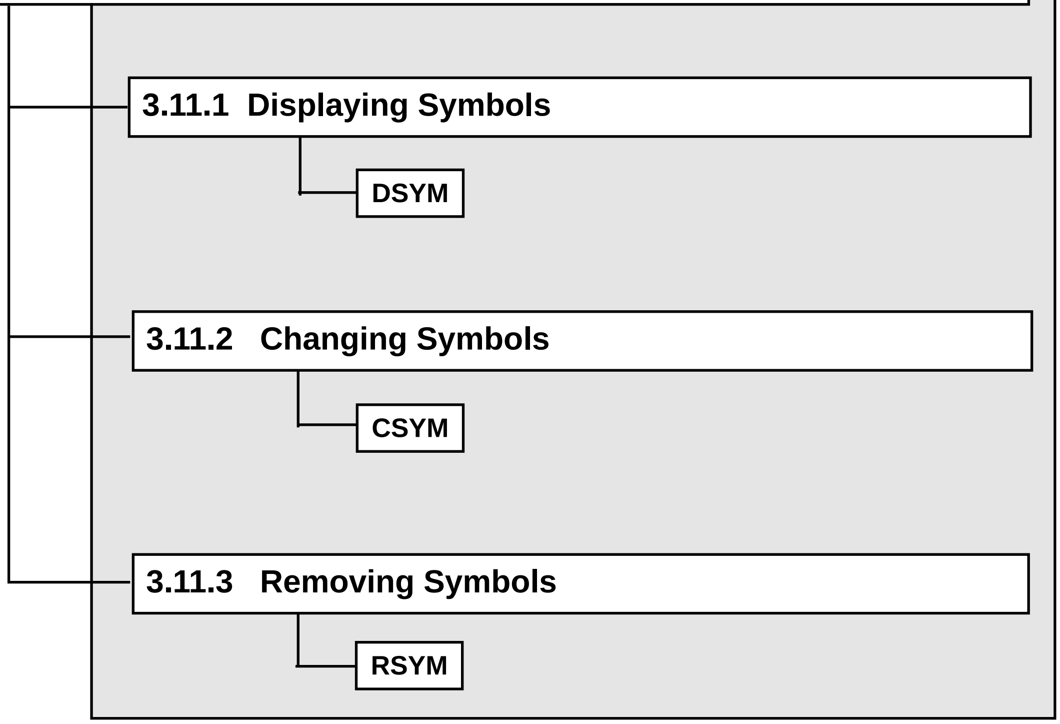
The **PAUSE** command waits for keyboard input when executed. By placing a **PAUSE** command in a batch file, automatic command execution can be temporarily suspended. The input wait state will be released upon input from the keyboard, or if the emulator reset switch is pressed.

Execution Example

```
64167> PAUSE

   *** Hit Any Key ***
64167>
```

3.11

Commands for Displaying / Changing / Removing Symbols



3.11.1 Displaying Symbols

DSYM

Input Format

DSYM Δ *string* [Δ *string* ..... Δ *string*] ↵

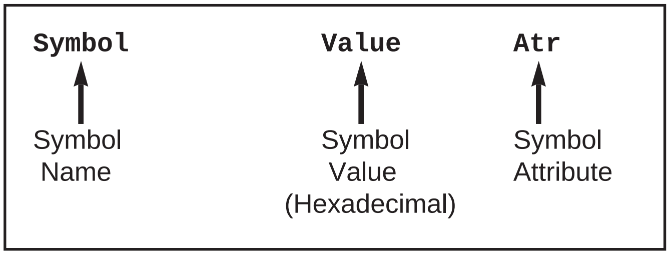
DSYM Δ * ↵

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

```
64167> LOD CHIPTP /S

    Symbol table clear (Y/N) Y
    Symbol Loading...
    HEX File Loading...
    Load Completed   address[0000 - 0648]

64167> DSYM *

 Symbol                        Value     Atr
 LOOP_WDT                      0239     CODE
 LOOP_X0                       0212     CODE
 LOOP_X1                       01E2     CODE
 SET_1                         0100     CODE
 SET_HAL                       0245     CODE
 TMINT                         0512     CODE
 SRINT                         0535     CODE
 STINT                         052F     CODE
 INT32HZ                       050C     CODE
 INT16HZ                       0506     CODE
 SET_ITM                       019B     CODE
 SET_1HZ                       0133     CODE
 LOOP_TM                       0288     CODE
 SET_X0                        0200     CODE
 SET_X1                        01C7     CODE
 LOOP_BZ                       053B     CODE
 LOOP_SR                       063E     CODE
 LOOP_HAL                      024E     CODE
 LOOP_ST                       062A     CODE
 SET_SYS                       0260     CODE
 XI0INT                        051E     CODE
 XI1INT                        0518     CODE
 LOOP_32HZ                     017F     CODE
 INT1HZ                        0500     CODE
 WDTINT                        0524     CODE
 SET_TM                        0273     CODE
 LOOP_16HZ                     015B     CODE
 SET_32HZ                      0179     CODE
 SET_16HZ                      0155     CODE
 LOOP_ITM                      01AF     CODE
 ERR_LOOP                      0600     CODE
 LOOP_1HZ                      0139     CODE
64167>
```

3.11.2 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.

"_↵"	Without changing the data, proceed to input data for the next symbol. If there is no next symbol, then input terminates.
"↵"	Terminates input.

Execution Example

```
64167> DSYM *

 Symbol                        Value     Atr
 TMINT                         0512     CODE
 SRINT                         0535     CODE
 STINT                         052F     CODE
 WDTINT                        0524     CODE
64167> CSYM S*

 SRINT  old[0535] ----> 0AD  New
 STINT  old[052F] ----> 7    New
64167> DSYM *

 Symbol                        Value     Atr
 TMINT                         0512     CODE
 SRINT                         00AD     CODE
 STINT                         0007     CODE
 WDTINT                        0524     CODE
64167> CSYM ??INT

 TMINT  old[0512] ----> 3    New
 SRINT  old[00AD] ----> 123  New
 STINT  old[0007] ----> 0BD3 New
64167> DSYM *

 Symbol                        Value     Atr
 TMINT                         0003     CODE
 SRINT                         0123     CODE
 STINT                         0BD3     CODE
 WDTINT                        0524     CODE
64167>
```

3.11.3 Removing Symbols

RSYM

Input Format

RSYM Δ *string* [Δ *string* ..... Δ *string*] ↵

RSYM Δ * ↵

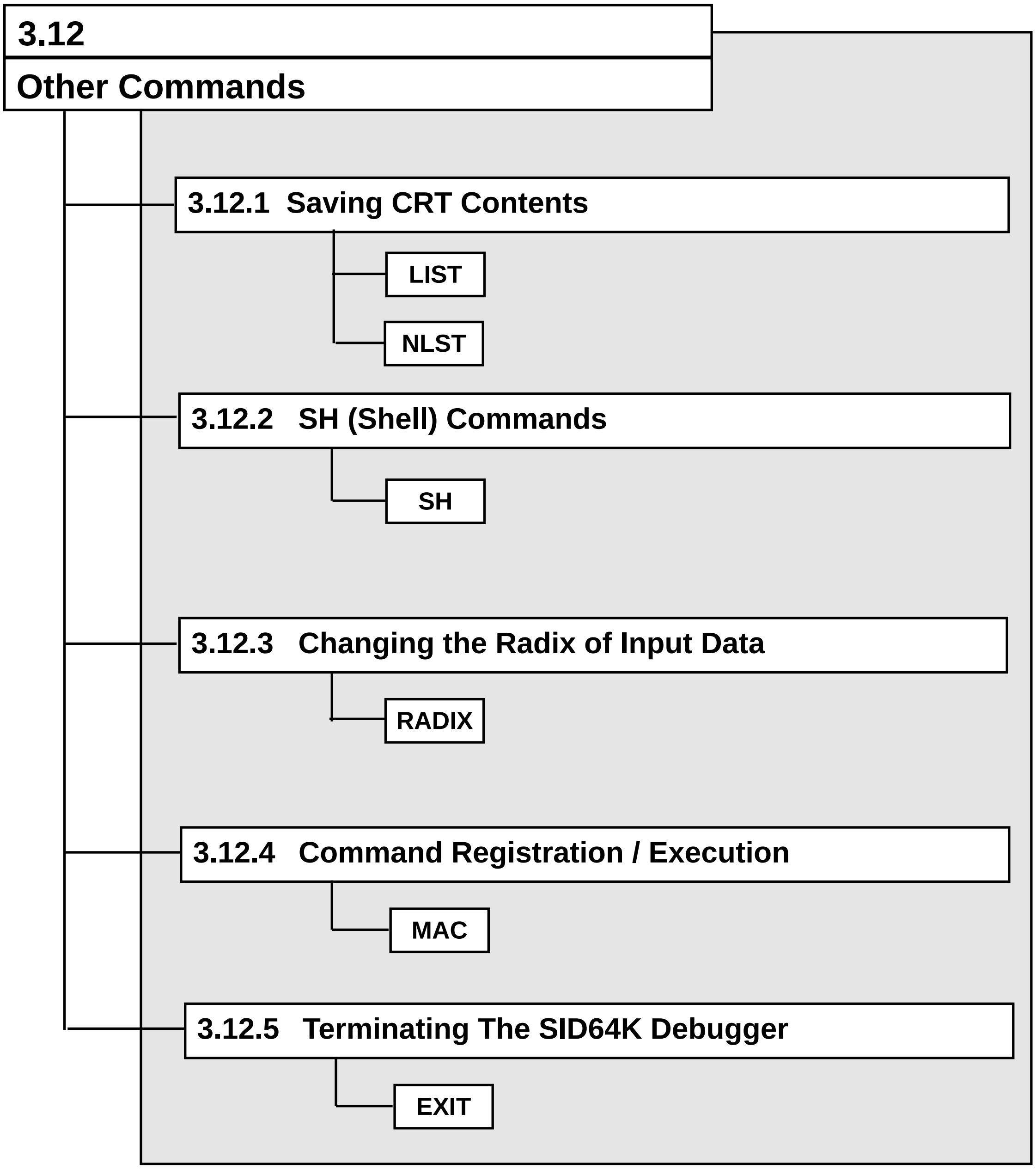
Description

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

```
64167> DSYM *

 Symbol                        Value     Atr
 TMINT                         0003     CODE
 SRINT                         0123     CODE
 STINT                         0BD3     CODE
 WDTINT                        0524     CODE
64167> RSYM S*

64167> DSYM *

 Symbol                        Value     Atr
 TMINT                         0003     CODE
 WDTINT                        0524     CODE
64167> DSYM S*

 Symbol                        Value     Atr
64167>
```



3.12 Other Commands

3.12.1 Saving CRT Contents

LIST

NLST

3.12.2 SH (Shell) Commands

SH

3.12.3 Changing the Radix of Input Data

RADIX

3.12.4 Command Registration / Execution

MAC

3.12.5 Terminating The SID64K Debugger

EXIT

3.12.1 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.
- In EASE-LP mode, the EASE-LP2 reset switch is pressed.

Execution Example

```
64167> LIST SAMP
     Listfile: SAMP.LST opened
64167> D

   A       : 0    B       : 0    H       : 0    L       : 0    X       : 0
   Y       : 0    PC      : 013B BCF     : 0    BEF     : 0    BSR0    : 7
   BSR1    : 0    C       : 0
64167>
```

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

```
64167> LIST SAMP
     Listfile: SAMP.LST opened
64167> NLST
```

3.12.2 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).

SH

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

```
64167> SH

64167>
```

3.12.3 Changing the Radix of Input Data

RADIX

Input Format

RADIX ∆ *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 EASE64165/167 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).

RADIX

Execution Example

```
64167> RADIX D

64167> DCM 10

       LOC = 000A    FF
64167> RADIX H

64167> DCM 10

       LOC = 0010    FF
64167> RADIX B

64167> DCM 10

       LOC = 0002    FF
64167> RADIX O

64167> DCM 10

       LOC = 0008    FF
64167>
```

3.12.4 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, registering it 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.

```
64167>  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, registration will automatically be terminated.

- 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.

```
64167> 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 "64167>" 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.

```
64167> DISP ↵
```

- Verifying/removing a registered macro command

To verify the registered macro commands, input the following.

```
64167>  MAC ↵
  1.  DISP
```

To remove a registered macro command, input the following.

```
64167> MAC ~DISP ↵
```

Execution Example

```
64167> MAC DISP


       1. -----> DCM [0 23]

       2. -----> DCM 90

       3. -----> CCM 30=1

       4. -----> D

       5. ----->

64167> DISP

 MAC > DCM [0 23]

                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     -----------------------------------------------
       LOC = 0000    A9 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
       LOC = 0010    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
       LOC = 0020    BC 05 00 BC 05 06 BC 05 0C FF FF FF BC 05 12 BC

 MAC > DCM 90
       LOC = 0090    FF

 MAC > CCM 30=1

 MAC > D
   A       : 0    B       : 0    H       : 0    L       : 0    X       : 0
   Y       : 0    PC      : 013B BCF     : 0    BEF     : 0    BSR0    : 7
   BSR1    : 0    C       : 0
```

```
64167> MAC DISP


       1. -----> DCM [0 23]
       2. -----> DCM 90
       3. -----> CCM 30=1
       4. -----> D

       ADD(+) or DEL(-) ==> +2

       2. -----> DDM 790



       1. -----> DCM [0 23]
       2. -----> DDM 790
       3. -----> DCM 90
       4. -----> CCM 30=1
       5. -----> D

       ADD(+) or DEL(-) ==> -3

       1. -----> DCM [0 23]
       2. -----> DDM 790
       3. -----> CCM 30=1
       4. -----> D

       ADD(+) or DEL(-) ==>


64167> DISP

 MAC > DCM [0 23]

                     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
                     -----------------------------------------------
       LOC = 0000    A9 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
       LOC = 0010    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
       LOC = 0020    BC 05 00 BC 05 06 BC 05 0C FF FF FF BC 05 12 BC

 MAC > DDM 790
       LOC = 0790    6

 MAC > CCM 30=1

 MAC > D
   A       : 0    B       : 0    H       : 0    L       : 0    X       : 0
   Y       : 0    PC      : 013B BCF     : 0    BEF     : 0    BSR0    : 7
   BSR1    : 0    C       : 0

64167>
```

3.12.5 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

```
64167> EXIT
```

! When the SID64K symbolic debugger is restarted without turning off its power after it has been terminated with the EXIT command, please do the following:

(1) Restarting in EASE-LP mode

After SID64K displays its start message, press the EASE-LP2 reset switch.

Chapter 4

Debugging Notes

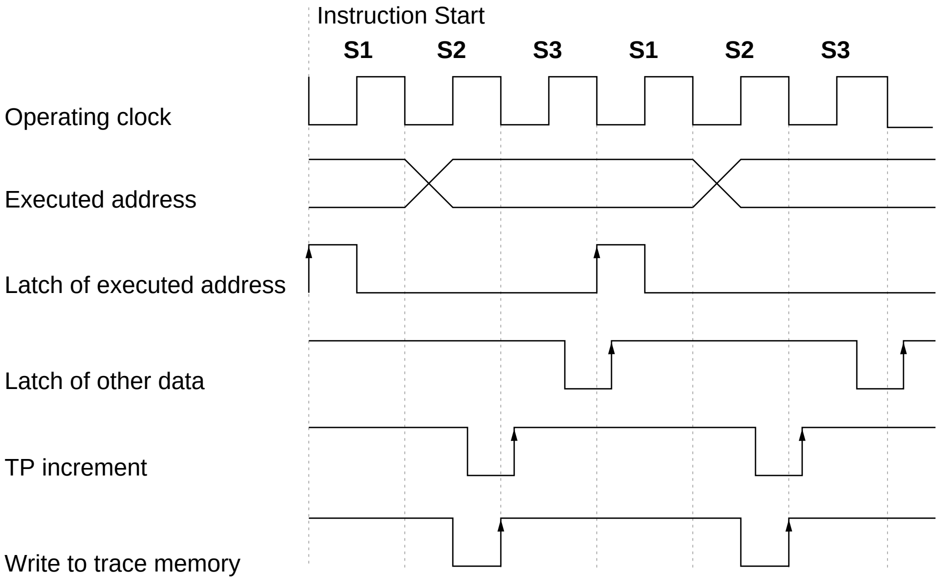
This chapter provides some notes about debugging with the EASE64165/167 system.

4-1. Debugging Notes

4-1-1. Tracing

The timing for writes to EASE64165/167 trace memory is as follows.

Latch of executed address	S1 & $\overline{\text{SYS.CLK/}}$
Other (AR, BR, SP, etc.)	$\overline{\text{(S2 \& SYS.CLK/)}}$ + gate delay
Trace pointer (TP) count	$\overline{\text{(M1S2 \& SYS.CLK/)}}$ + gate delay
Write to trace memory	$\overline{\text{M1S2 \& SYS.CLK/}}$



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.

4-1-2. Cycle Counter Overflow Breaks

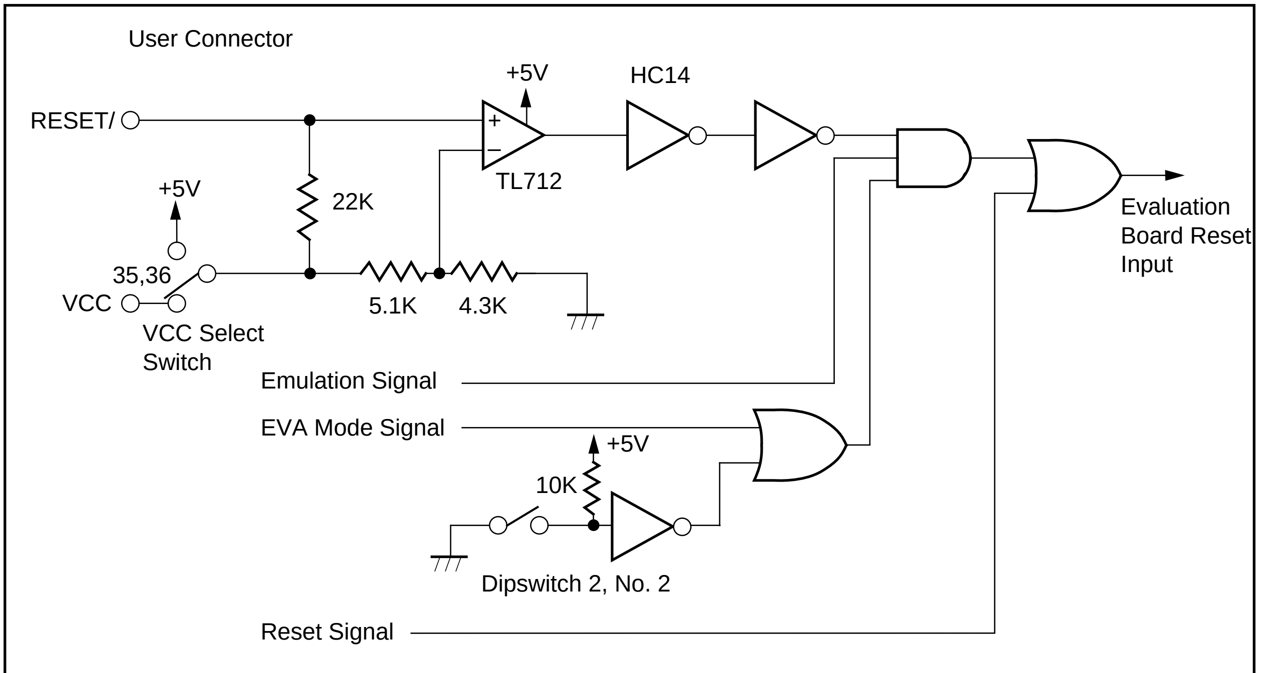
When program execution breaks due to a cycle counter overflow break, the break will occur after execution of the next instruction after the instruction at which the cycle counter overflowed. Accordingly, the cycle counter value at the break will not be 0, but will be the cycle count value of the instruction that generated the break.

4-1-3. User Cables

Power is not supplied from the user cables connected to the POD64165/167 and the user application system. When connected to the user application system during debugging, supply Vcc to the user application system from a separate power supply.

4-1-4. Reset

The user cable RESET/ pin is effective only in EVA mode, where the POD64165/167 is operated standalone, and under realtime emulation from the G command. However, during realtime emulation this is restricted to when switch 3 of dipswitch 2 of the POD64165/167 is on.



Emulation Signal: becomes '1' during emulation execution.
EVA Mode Signal: becomes '1' when in EVA mode.
Reset Signal: is emulation kit internal reset signal.

Figure 4-1. Reset Circuit

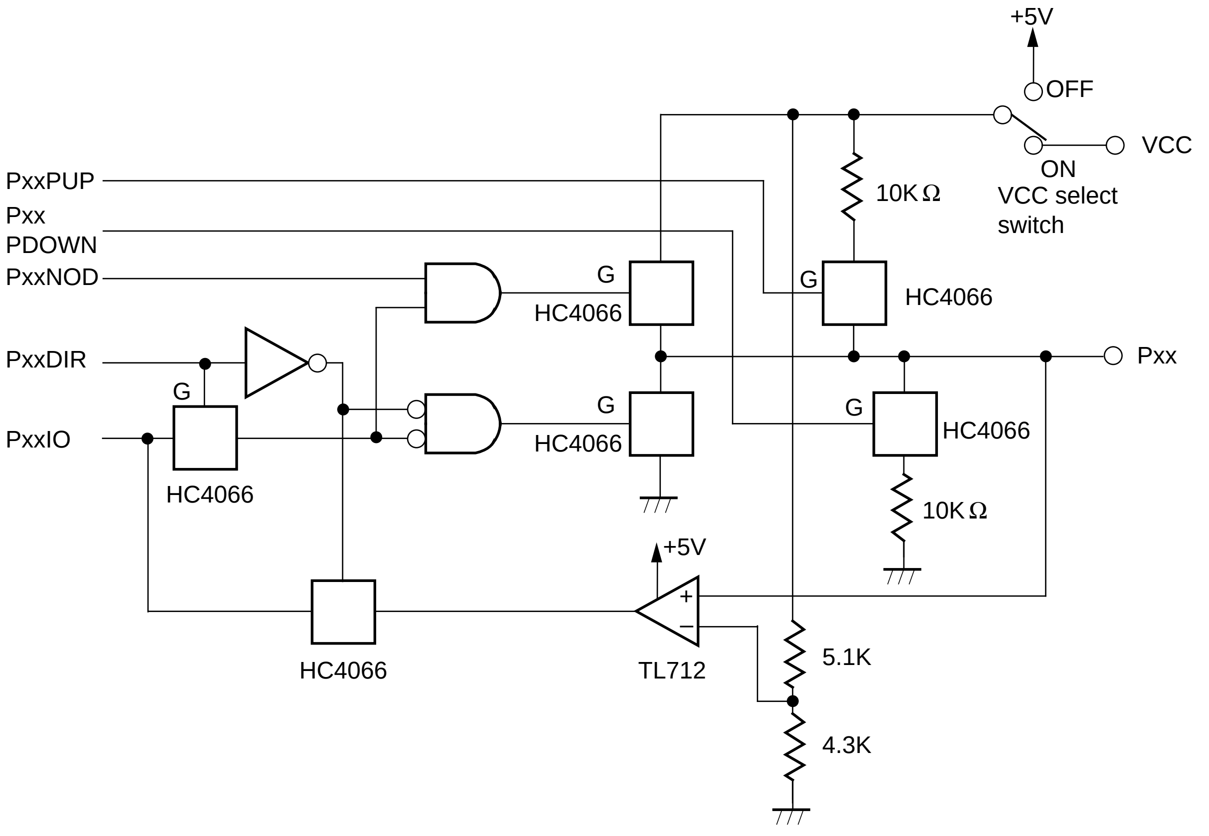
4-1-5. Ports

The circuit configuration of each pin of the user connector is shown below. Please be aware that the input/output characteristics of the MSM64165 and MSM64167 will be different.

Because the EASE64165/167 internal circuits operate at 5V, all user connector pins have an internal level conversion circuit. The pins' voltage level will be 5V when the VCC select switch is off, and it will depend on the voltage applied to the user connector VCC pin when the switch is on.

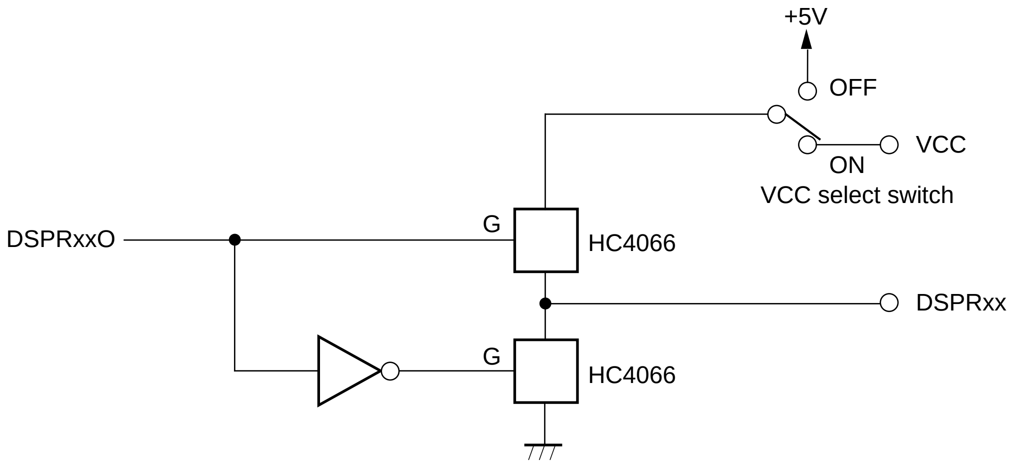
The VCC pin input voltage range is 3-5V.

a. P00-P03, P10-P13, P20-P23



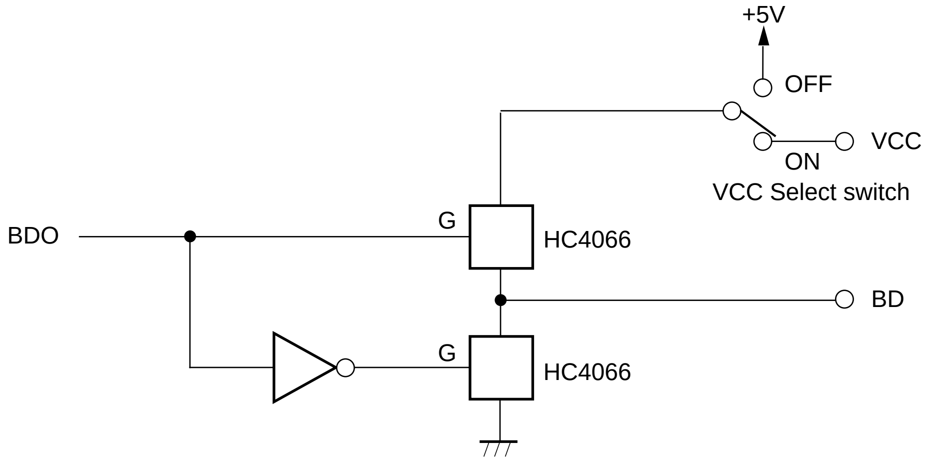
- Pxx indicates user connector pins P00-P03, P10-P13, and P20-P23.
- VCC indicates the user connector VCC pin.
- PxxIO indicates the internal input/output signals of P00-P03, P10-P13, and P20-P23.
- PxxDIR is the input/output mode select signals of P00-P03, P10-P13, and P20-P23.
- PxxNOD is the N-channel open-drain output select signals of P00-P03, P10-P13, and P20-P23.
- PxxPUP is the pull-up select signals of P00-P03, P10-P13, and P20-P23.
- PxxPDOWN is the pull-down select signals of P00-P03, P10-P13, and P20-P23.

b. DSPR00-DSPR03, DSPR10-DSPR13.



- DSPRxx indicates user connector pins DSPR00-DSPR03 and DSPR10-DSPR13. These pins become valid when the LCD driver pins are set to output ports by mask option.
- VCC indicates the user connector VCC pin.
- DSPRxxO indicates the internal output signals of DSPR00-DSPR03 and DSPR10-DSPR13.

c. BD

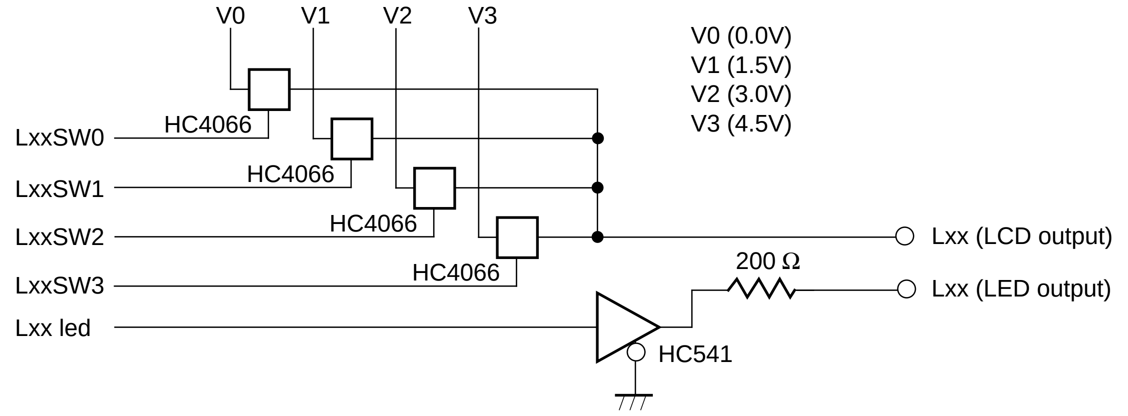


- BD indicates the user connector BD pin.
- VCC indicates the user connector VCC pin.
- BDO is the internal input/output signal of BD.

4-1-6. LCD Drivers

(1) Output Circuit

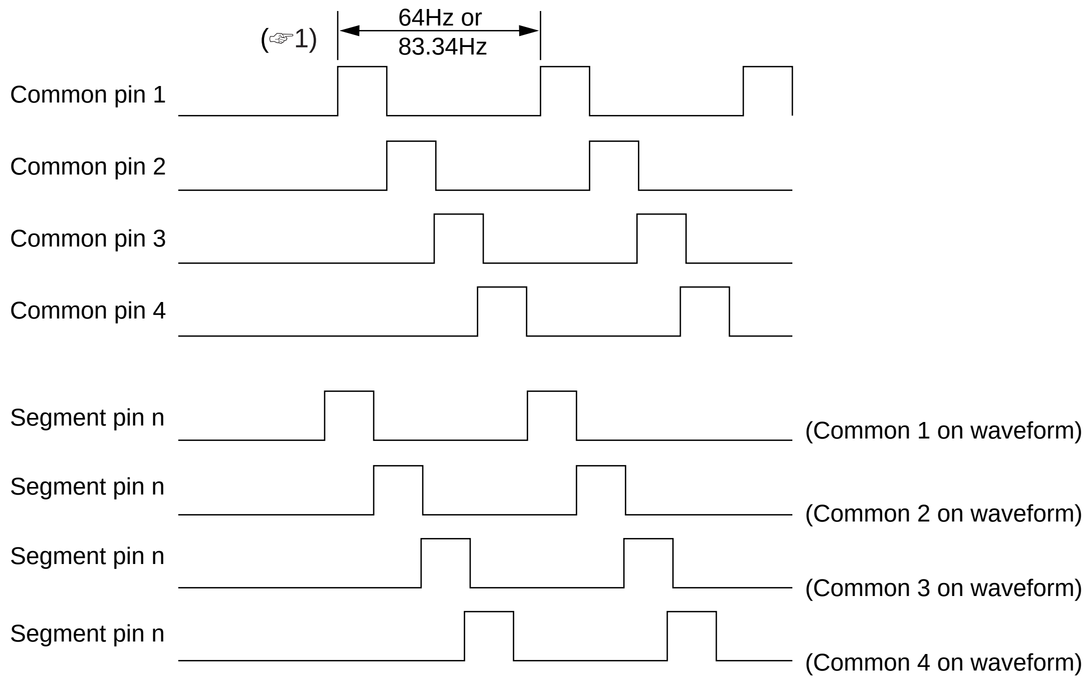
LCD driver outputs are constructed as shown below. The output characteristics of the MSM64165 and MSM64167 will be different.



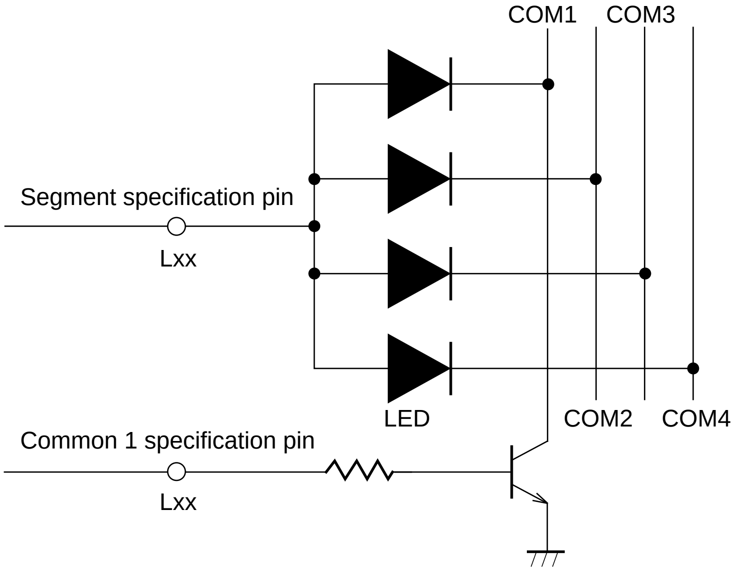
- Lxx (LCD output) indicates LCD connector pins L0-L30. They are used for LCDs in program debug.
- Lxx (LED output) indicates LED connector pins L0-L30. They are used for LEDs (light-emitting diodes) in program debug.

(2) LED Connector Output Signals

LED outputs are pins for using LEDs (light-emitting diodes) to evaluation the LCD portion of an application. They output the following signals.



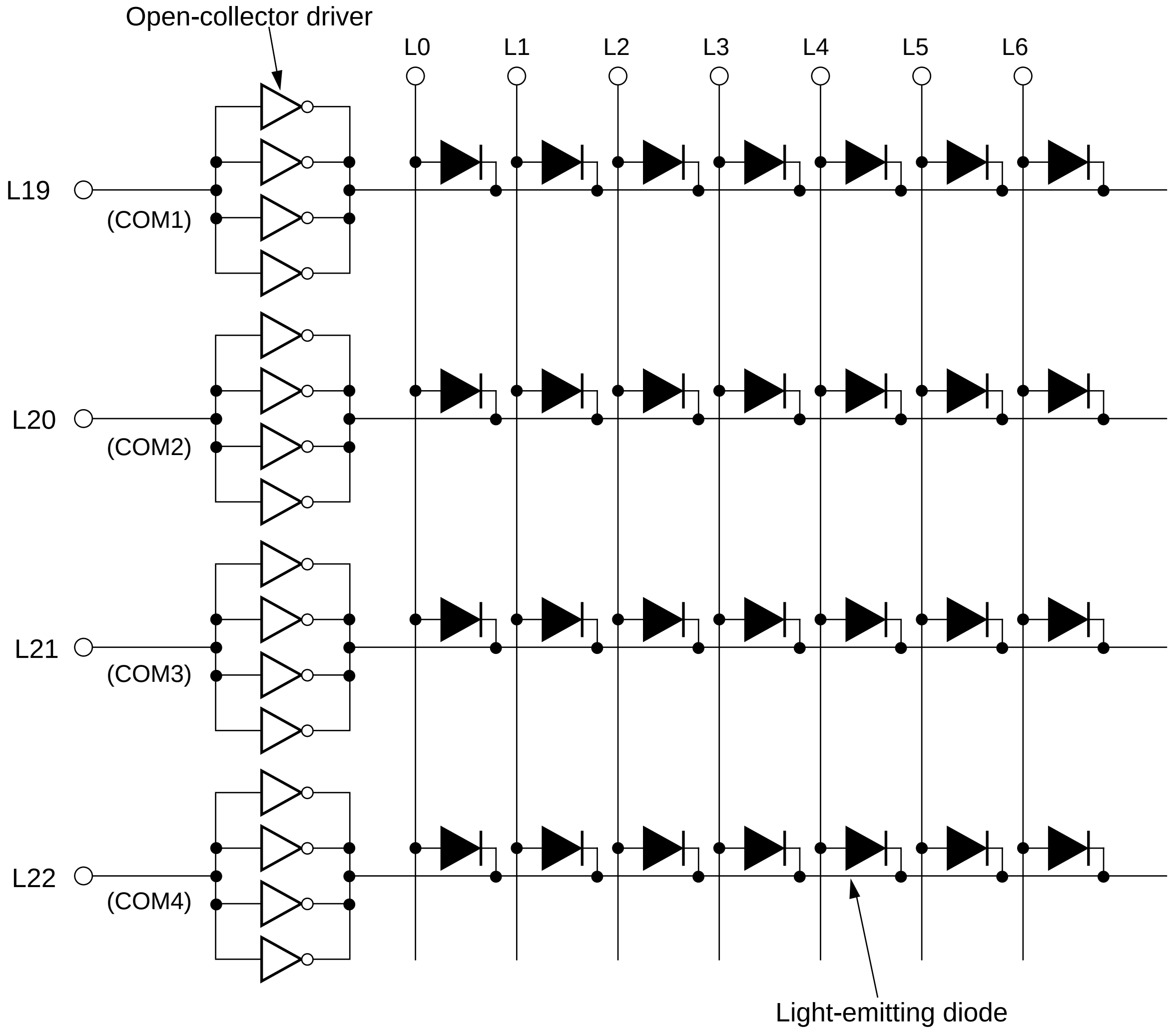
To perform dynamic evaluation with LEDs, construct a circuit like the following.



Connection Example for LED Dynamic Evaluation

In this circuit, if the current flowing in one LED is assumed to be 1.25 mA, then the common pin transistor's collector current will be up to 37.5 mA (for 1/4 duty), so a high-current drive transistor will be needed.

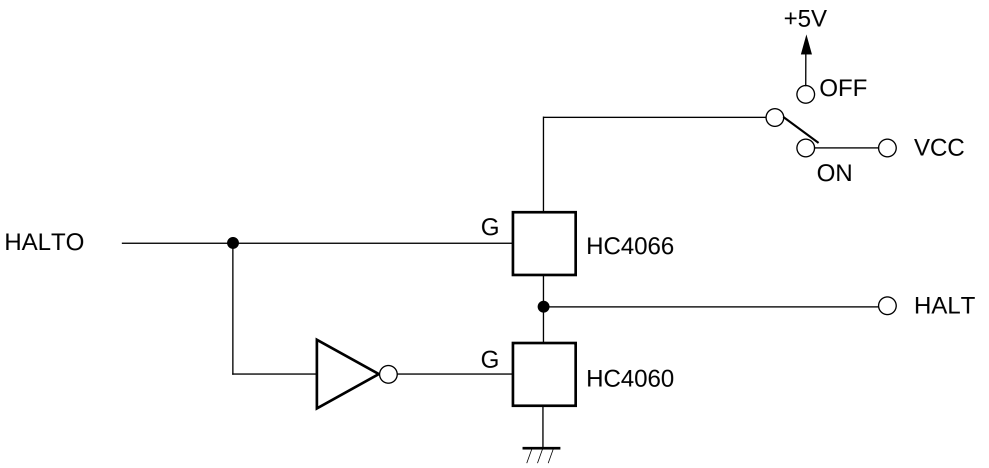
☞ 1 Frequency will be 64 Hz when 1/4 or 1/2 duty is selected, and 83.34 Hz when 1/3 duty is selected.



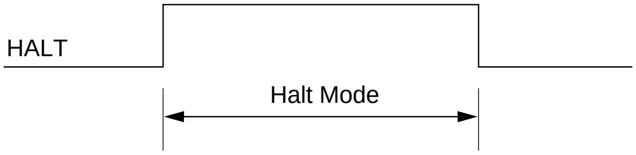
LED Dynamic Evaluation Circuit Example

4-1-7. HALT pin

The user connector HALT pin is a fixed emulation kit signal that outputs a "H" level when in halt mode.



- HALT indicates the user connector HALT pin.
- VCC indicates the user connector VCC pin.
- HALTO is the internal output signal of HALT.
- The HALT pin outputs a "H" level when in halt mode.



4-1-8. EPROM Programmer

Always remove any EPROM in the EASE-LP2's EPROM programmer when the EASE64165/167 is started. Mount EPROMs when the emulator is waiting for command input.

SEE Appendix 8, “Mounting EASE-LP2 EPROMs.”

4-1-9. Mask Option EPROM

Always mount a mask option EPROM before starting the EASE64165/167, whether in EASE-LP mode or EVA mode. Mount the mask option EPROM while the power supply is off.

4-1-10. Program EPROM

Always mount a program EPROM before starting the EASE64165/167 in EVA mode. Mount the program EPROM while the power supply is off.

4-1-11. 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.

4-1-12. 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."

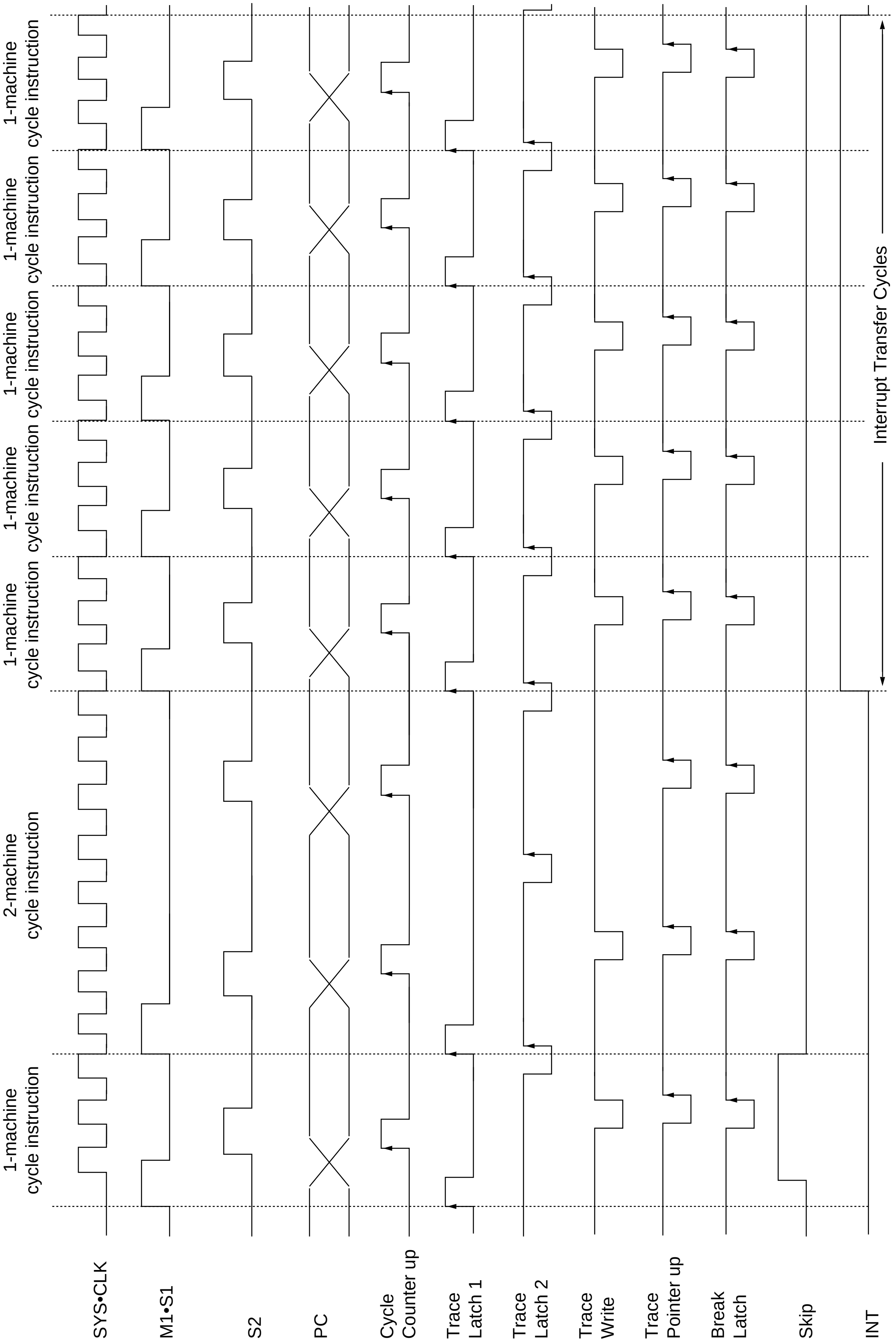
4-1-13. High-Speed Clock

In MSM64165 and MSM64167, high-speed clock is not based on the low-speed clock. Therefore, interrupt timing in break (emulation) operation or in step execution might differ when EASE64165/167 is operated under high-speed clock (number of instruction execution before interrupt occurence will be unfixed).

4-2. EASE64165/167 Timing

EASE64165/167 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	MSM64E165/167 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

Address Space	***Corresponding Area***		***Segment Type***
Code address space	Code memory area (0~BFFH)		CODE
	MSM64165 Mode	MSM64167 Mode	
	0 ~7DFH	0 ~ FDFH	
Data address space	Internal RAM data area in data space (0~760H)		DATA
	MSM64165 Mode	MSM64167 Mode	
	780 ~7FFH	700 ~ 7FFH	

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.

Segment	***Location Range***	
	MSM64165 Mode	MSM64167 Mode
Code segment	0 ~7DFH	0 ~ FDFH
Data segment	780 ~7FFH	700 ~ 7FFH

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
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.

3. The backslash (\) will normally be a yen mark (Y) on Japanese keyboards. If you are using a Japanese keyboard, then replace the backslash with a yen mark in the above explanation.

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 MSM64165 or MSM64167 user's manual.

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 MSM64165 or MSM64167 user's manual.

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—7DFH or 0—FDFH). 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 780—07FFH or 700—7FFH. 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 Control 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 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

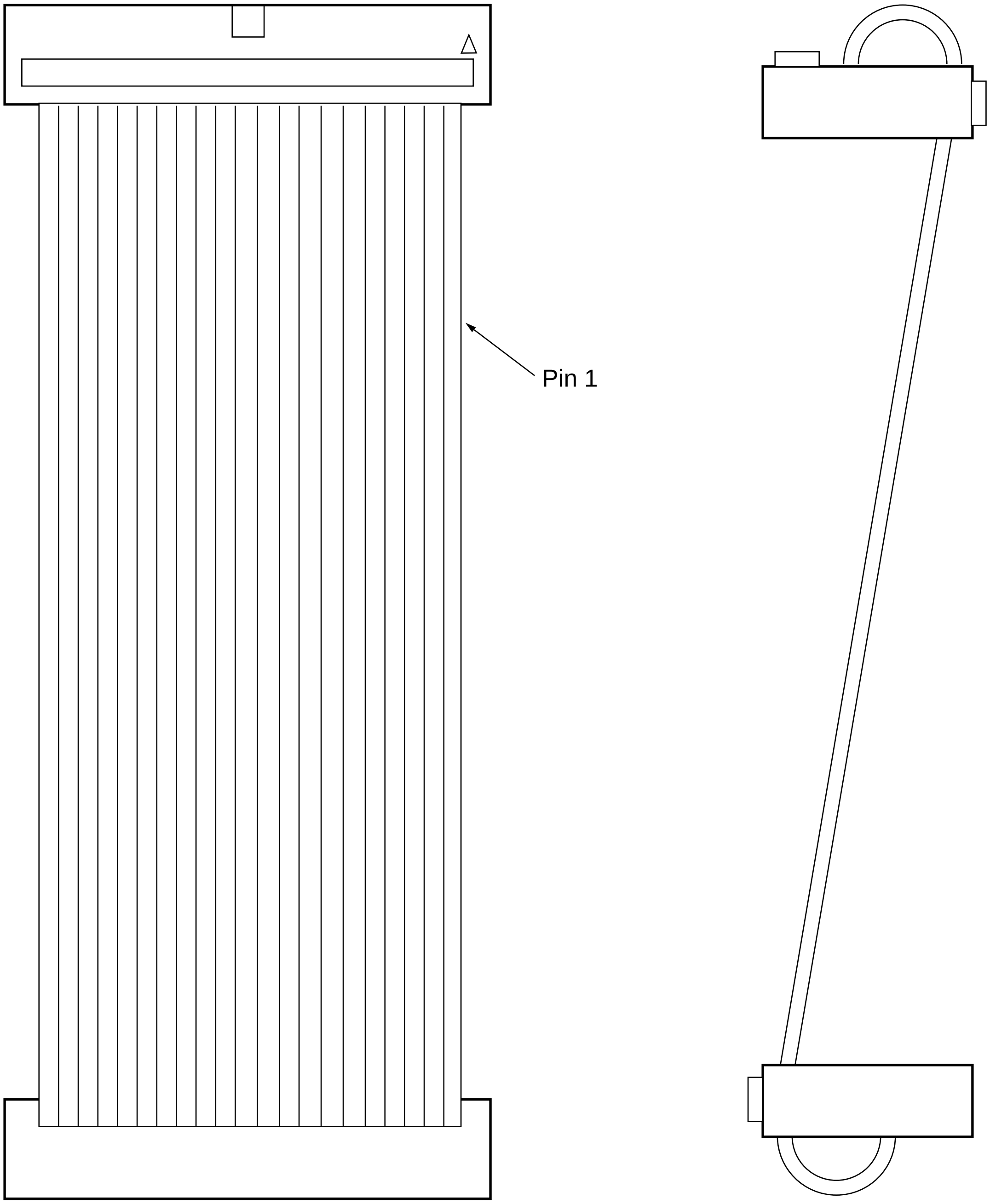
This directive indicates the end of a program. When the **ASM** command encounters an **END** directive, it completes pass 1 processing and immediately enters pass 2 processing.

Appendix

A-1. User Cable Configuration

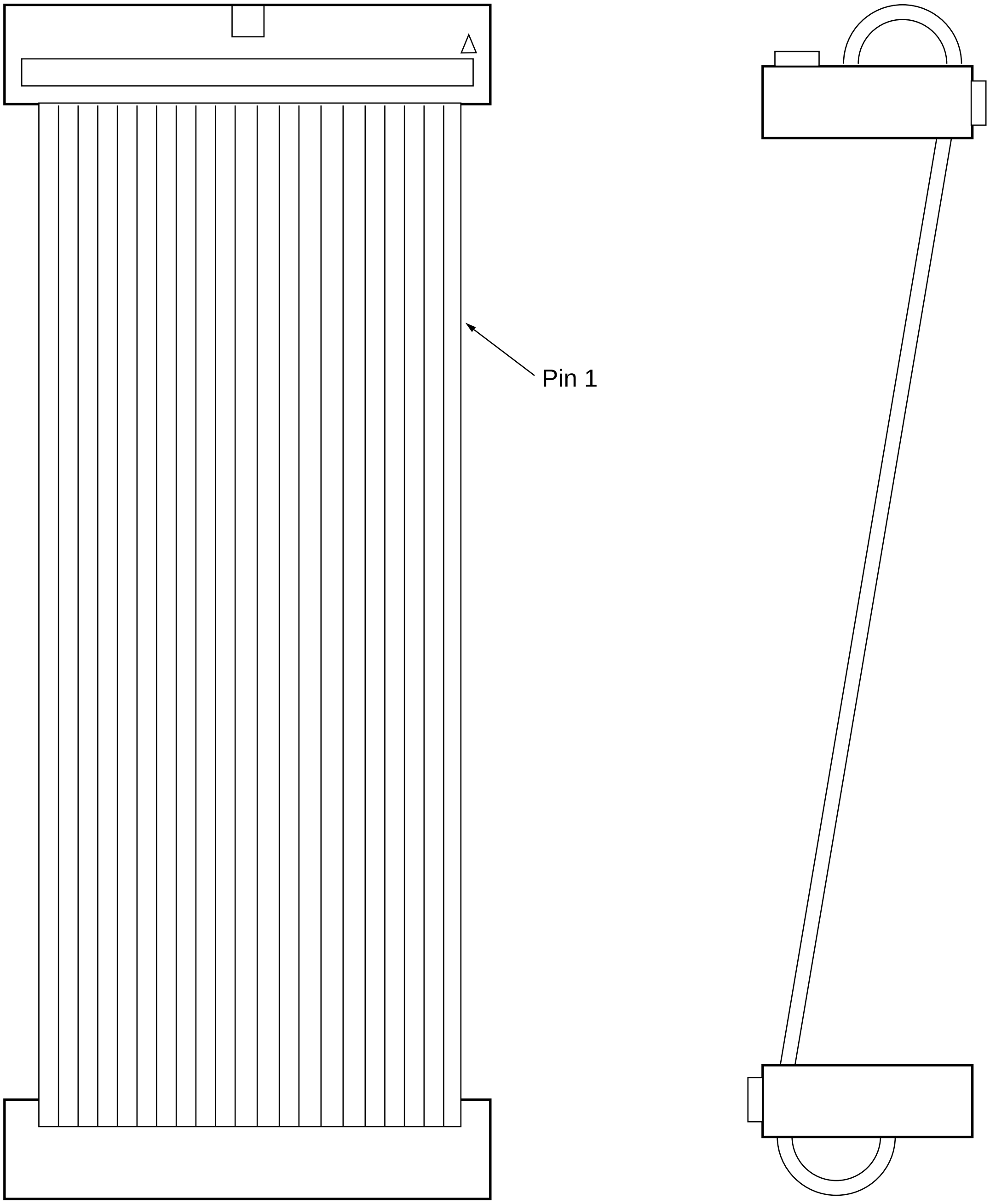
(1) User Cable 1 Configuration

The diagram below shows the configuration of the accessory user cable 1 (one 40-pin cable). User cable 1 connects to the user connector.



(2) User Cable 2 Configuration

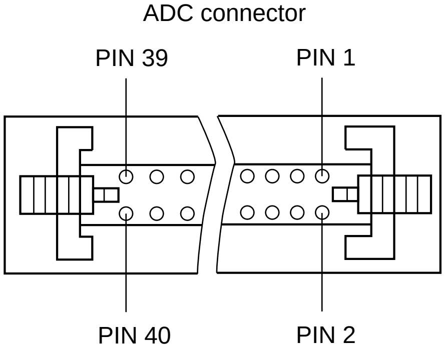
The diagram below shows the configuration of the accessory user cable 2 (one 34-pin cable). User cable 2 connects to the LCD or LED connector.



A-2. Pin Layout of User Cable Connectors

(1) Pin Layout of User Cable Connector

User Connector

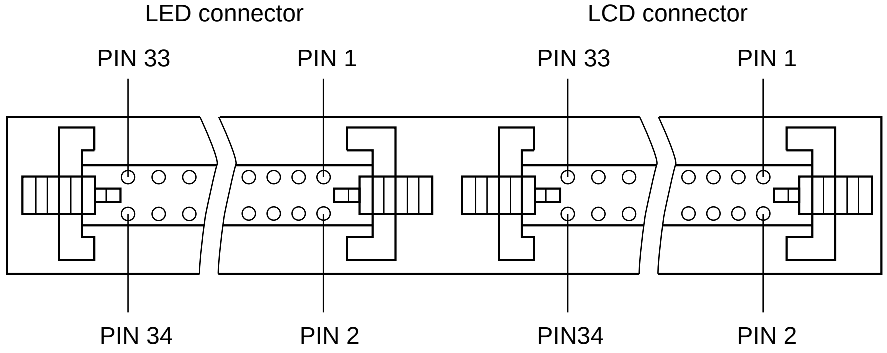


- As shown at left, user connector 1 is a 40-pin connector with pin 1 at the upper right.

Pin Number	Signal Name	Pin Number	Signal Name
1	BD	21	DSPR10
2	P00	22	DSPR11
3	P01	23	DSPR12
4	P02	24	DSPR13
5	P03	25	GND
6	P10	26	GND
7	P11	27	N.C.
8	P12	28	HALT
9	P13	29	GND
10	P20	30	OSC
11	P21	31	GND
12	P22	32	XT
13	P23	33	N.C.
14	N.C.	34	N.C.
15	GND	35	VCC
16	GND	36	VCC
17	DSPR00	37	RESET/
18	DSPR01	38	N.C.
19	DSPR02	39	GND
20	DSPR03	40	GND

Note 1: NC indicates pin is not connected.
Note 2: When the VCC select switch is on and power is to be supplied externally, a 3-5V external power supply must be connected to the VCC pins (pins 35 and 36).
Note 3: The user connector HALT pin is a fixed emulation kit signal that outputs a "H" level when in halt mode.

(1) Pin Layout of User Cable Connector 2



- As shown above, the LED connector and LCD connector are 34-pin connectors, with pin 1 at the upper right.
- Use the LCD connector to perform program debugging with LCD, or use the LEC connector to perform program debugging with LEDs.

Pin Number	Signal Name	Pin Number	Signal Name
1	L0	18	L17
2	L1	19	L18
3	L2	20	L19
4	L3	21	L20
5	L4	22	L21
6	L5	23	L22
7	L6	24	L23
8	L7	25	L24
9	L8	26	L25
10	L9	27	L26
11	L10	28	L27
12	L11	29	L28
13	L12	30	L29
14	L13	31	L30
15	L14	32	L31
16	L15	33	GND
17	L16	34	GND

LED Connector Pin List

Pin Number	Signal Name	Pin Number	Signal Name
1	L0	18	L17
2	L1	19	L18
3	L2	20	L19
4	L3	21	L20
5	L4	22	L21
6	L5	23	L22
7	L6	24	L23
8	L7	25	L24
9	L8	26	L25
10	L9	27	L26
11	L10	28	L27
12	L11	29	L28
13	L12	30	L29
14	L13	31	L30
15	L14	32	L31
16	L15	33	GND
17	L16	34	GND

LCD Connector Pin List

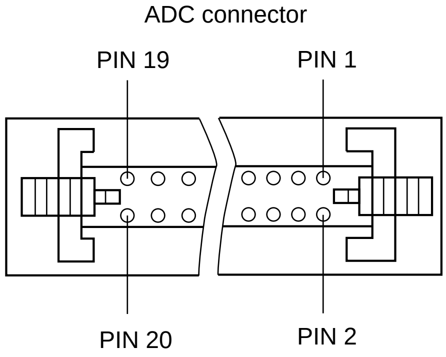
Note 1: NC indicates pin is not connected.

Note 2: L31 is a reserved pin not provided by MSM64165 or MSM64167.

User Connector 2 Pin List

(3) ADC connector pin layout

ADC Connector



- As shown at left, the ADC connector is a 20-pin connector with pin 1 at the upper right.
- The ADC connector connects to the MSM64165/167 ADC POD.

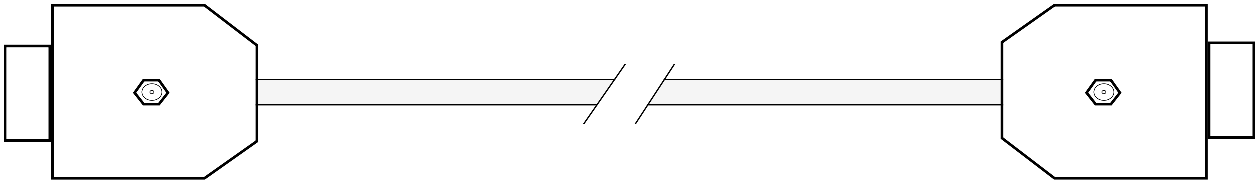
ADC Connector Pin List

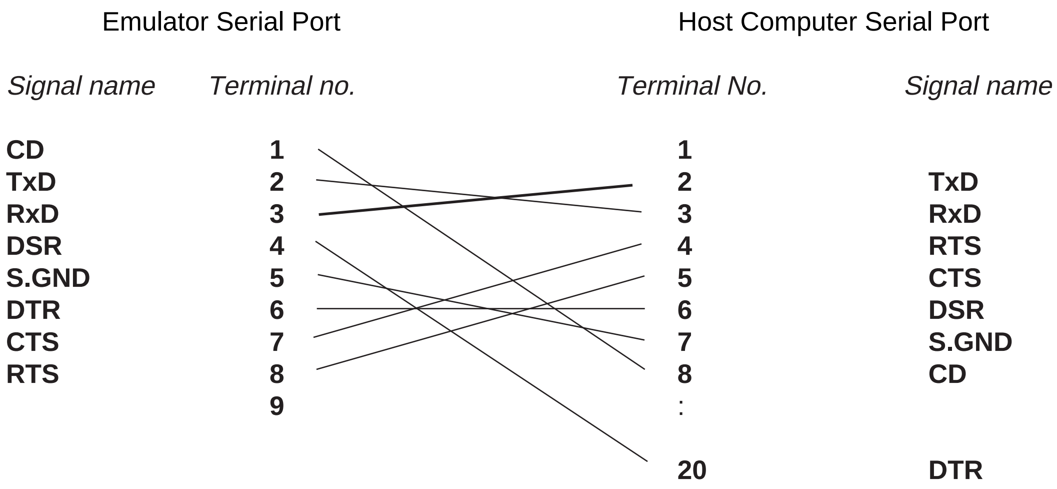
Pin Number	Signal Name	Pin Number	Signal Name
1	CH0	11	SVRP
2	CH1	12	SVRN
3	ICH0	13	COMP
4	ICH1	14	VSS2
5	+5V	15	N.C
6	SOPP0	16	+5V
7	ENADC	17	+5V
8	ENOP	18	RESET/
9	SVG	19	GND
10	SVIN	20	GND

Note 1: NC indicates pin is not connected.
Note 2: These signals control the MSM64165/167 ADC POD. They are not normally used by the customer.
Note 3: Refer to section 2-2-9, "MSM64165/167 ADC POD."

A-3. RS232C Cable Configuration

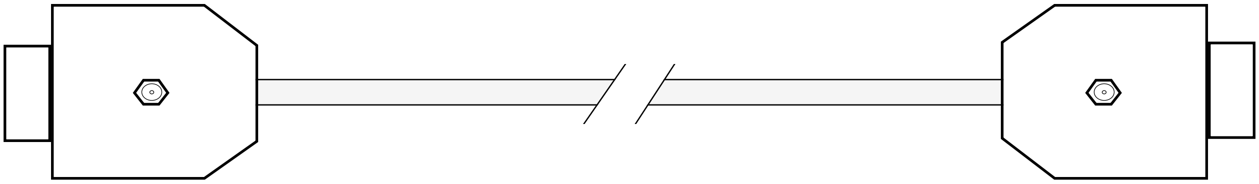
(1) For NEC PC9801 series computers



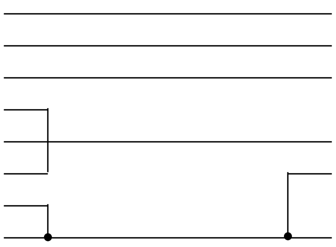


Emulator Serial Port		Host Computer Serial Port	
Signal name	*Terminal no.*	*Terminal No.*	*Signal name*
CD	**1**	**1**	
TxD	**2**	**2**	**TxD**
RxD	**3**	**3**	**RxD**
DSR	**4**	**4**	**RTS**
S.GND	**5**	**5**	**CTS**
DTR	**6**	**6**	**DSR**
CTS	**7**	**7**	**S.GND**
RTS	**8**	**8**	**CD**
	9	:	
		20	**DTR**

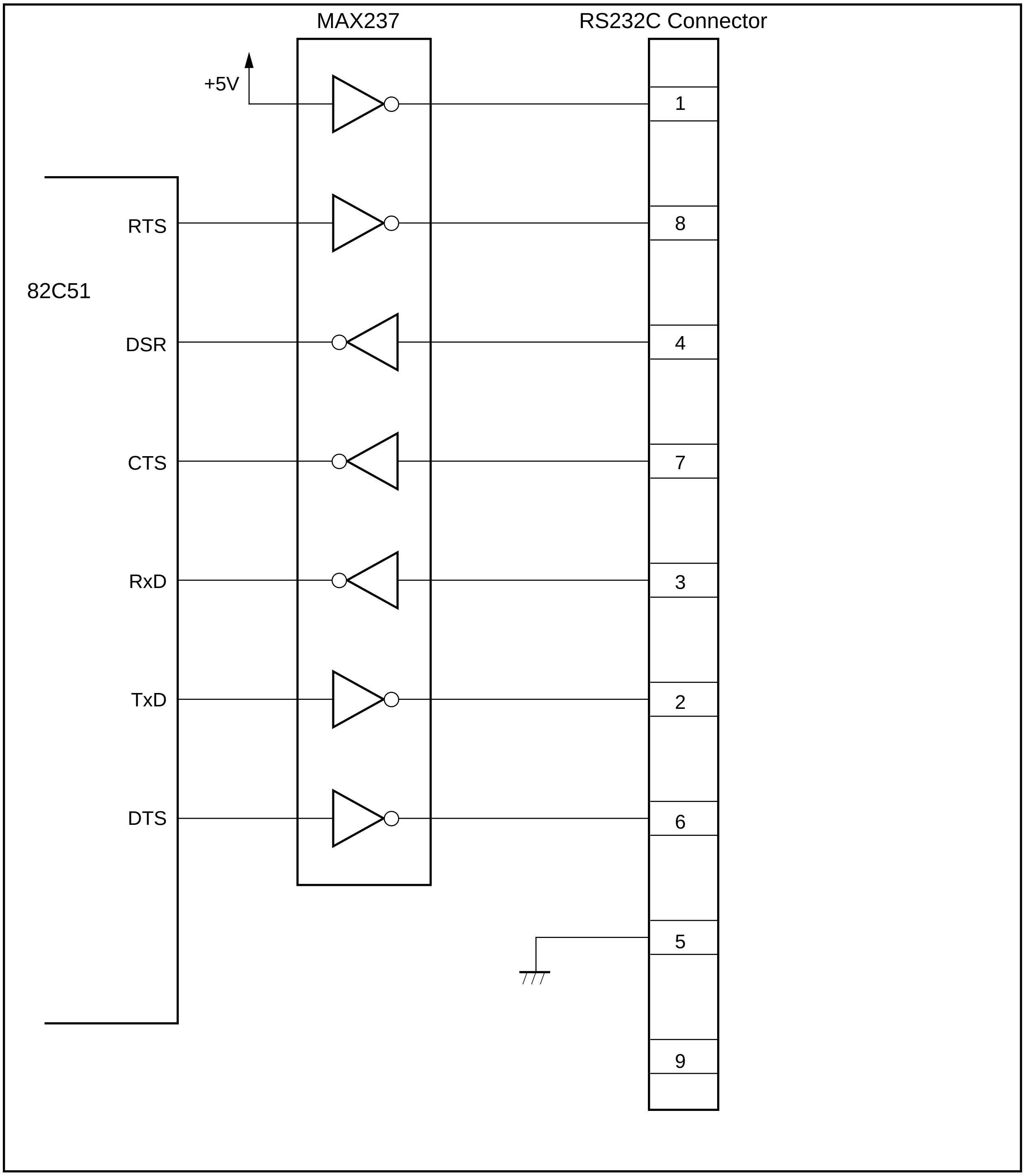
(2) For IBM PC/AT computers



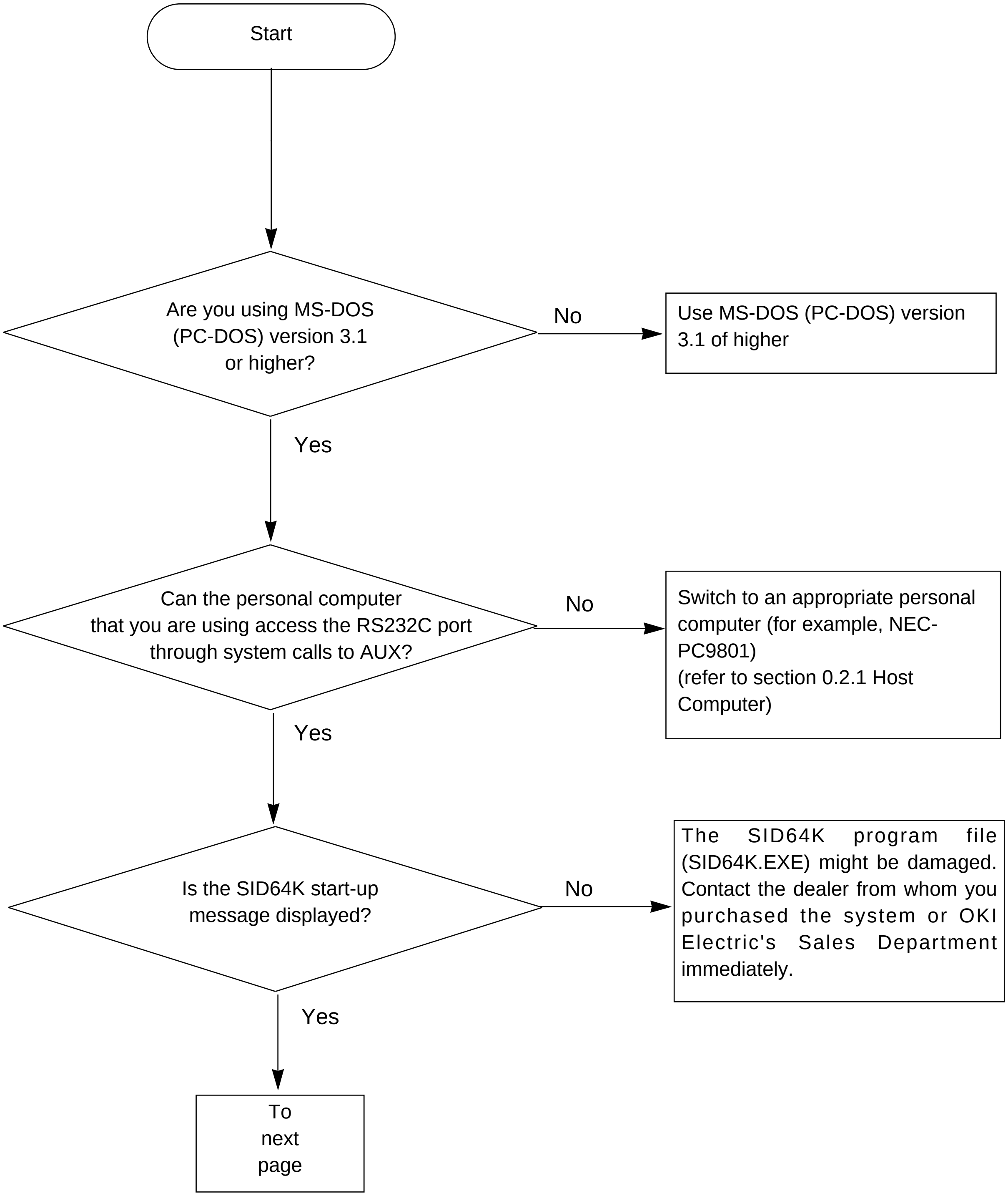
Emulator Serial Port		Host Computer Serial Port	
Signal name	*Terminal no.*	*Terminal No.*	*Signal name*
CD	**1**	**1**	**CD**
TxD	**2**	**2**	**RxD**
CxD	**3**	**3**	**TxD**
DSR	**4**	**4**	**DTR**
S.GND	**5**	**5**	**S.GND**
DTR	**6**	**6**	**DSR**
CTS	**7**	**7**	**RTS**
RTS	**8**	**8**	**CTS**
	9	**9**	



A-4. Emulator RS232C Interface Circuit



A-5. If EASE-LP Mode Won’t Start



Are you using MS-DOS (PC-DOS) version 3.1 or higher?

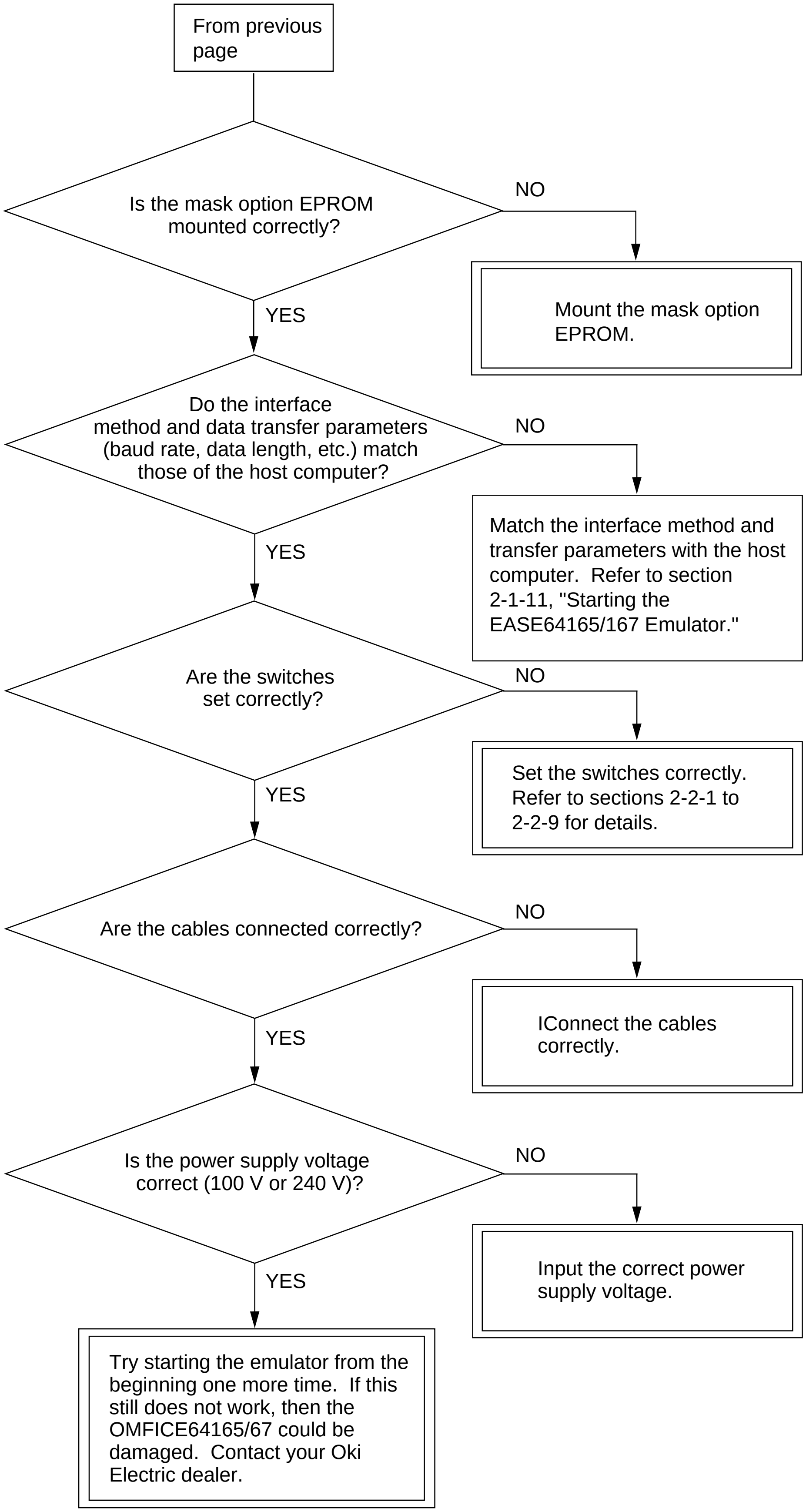
Use MS-DOS (PC-DOS) version 3.1 of higher

Can the personal computer that you are using access the RS232C port through system calls to AUX?

Switch to an appropriate personal computer (for example, NEC-PC9801)
(refer to section 0.2.1 Host Computer)

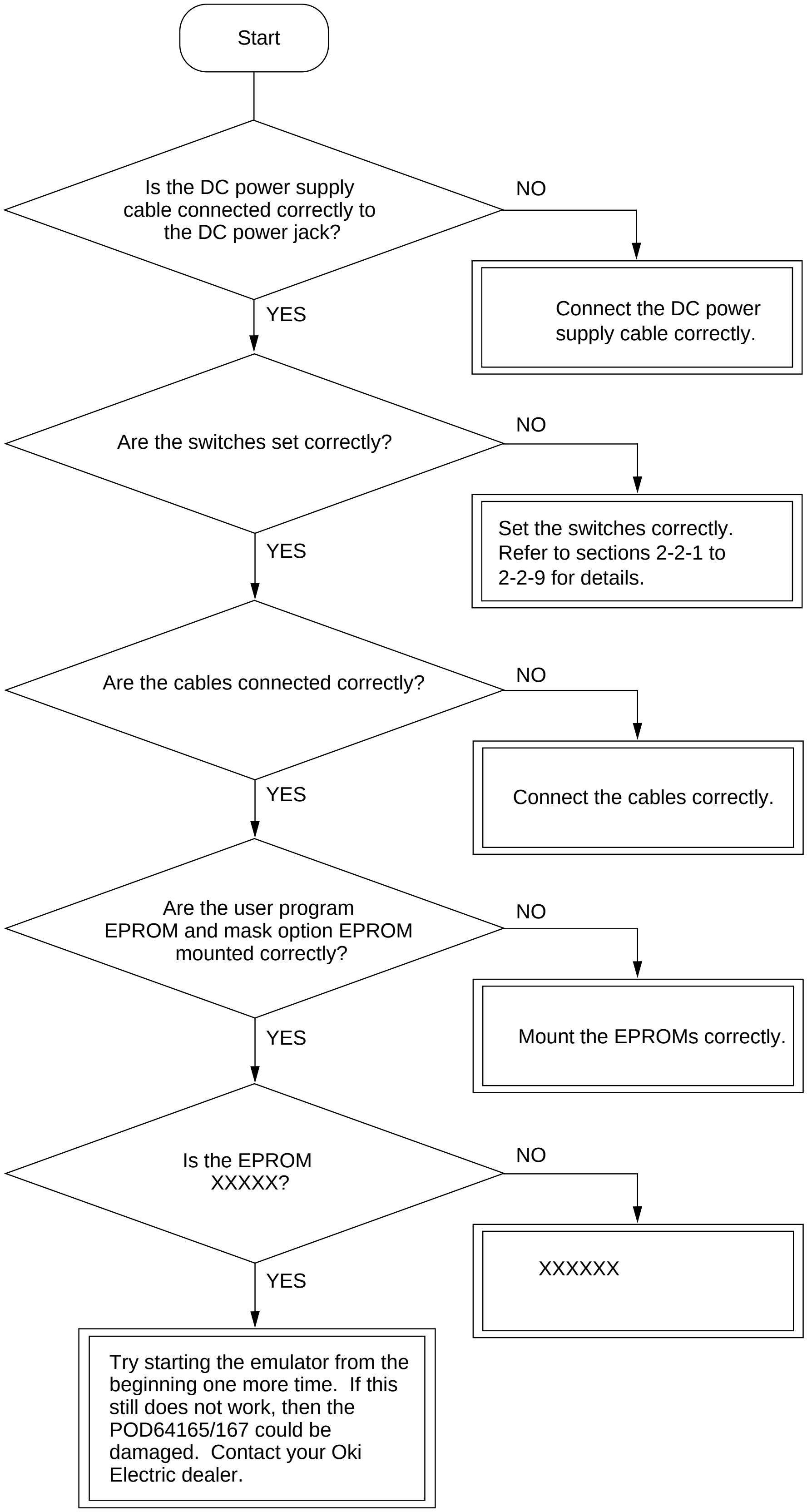
Is the SID64K start-up message displayed?

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.



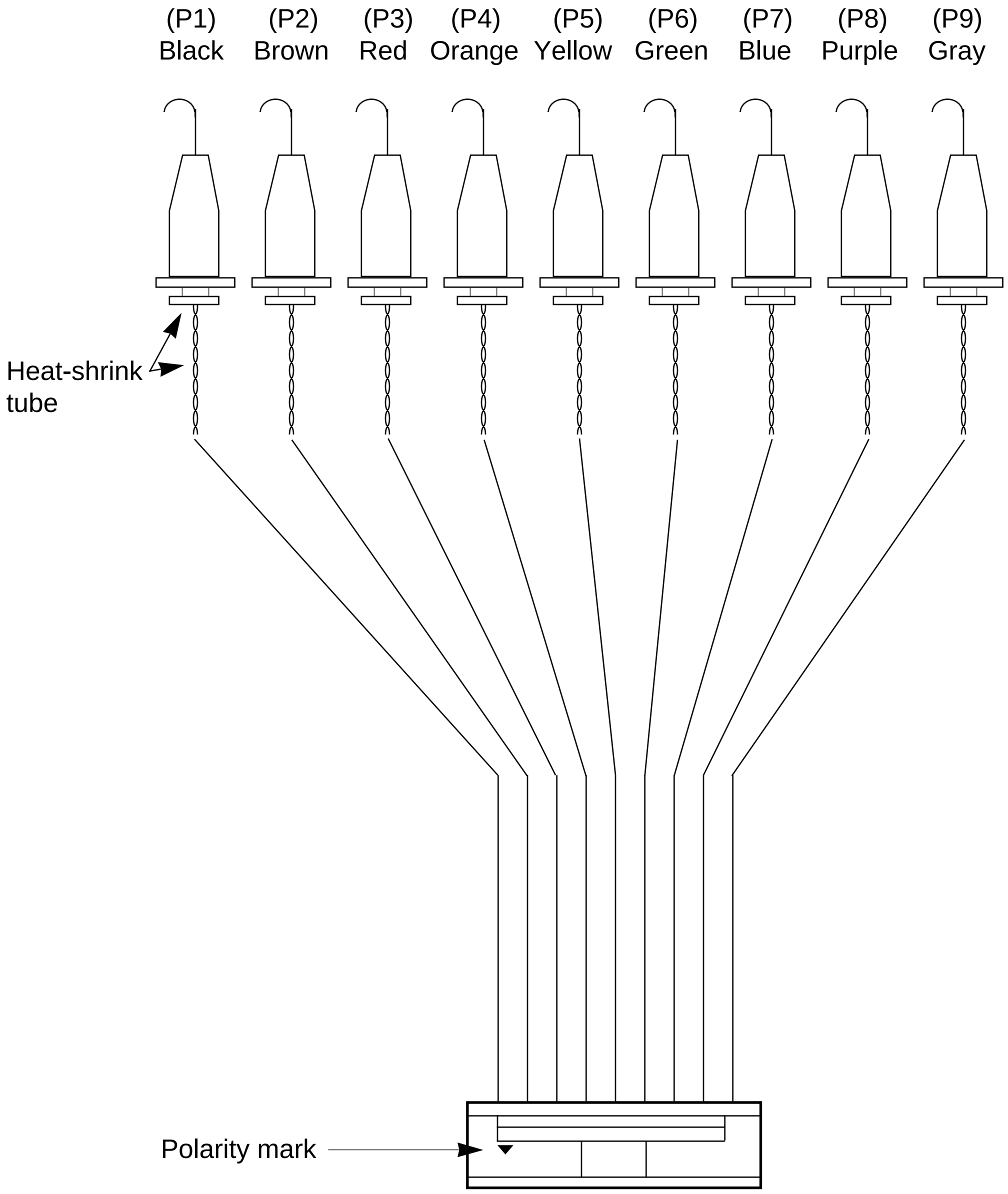
From previous page
Is the mask option EPROM mounted correctly?
NO
Mount the mask option EPROM.
YES
Do the interface method and data transfer parameters (baud rate, data length, etc.) match those of the host computer?
NO
Match the interface method and transfer parameters with the host computer. Refer to section 2-1-11, "Starting the EASE64165/167 Emulator."
YES
Are the switches set correctly?
NO
Set the switches correctly. Refer to sections 2-2-1 to 2-2-9 for details.
YES
Are the cables connected correctly?
NO
IConnect the cables correctly.
YES
Is the power supply voltage correct (100 V or 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 OMFICE64165/67 could be damaged. Contact your Oki Electric dealer.

A-6. If EVA Mode 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 switches set correctly?
NO
Set the switches correctly. Refer to sections 2-2-1 to 2-2-9 for details.
YES
Are the cables connected correctly?
NO
Connect the cables correctly.
YES
Are the user program EPROM and mask option EPROM mounted correctly?
NO
Mount the EPROMs correctly.
YES
Is the EPROM XXXXX?
NO
XXXXXX
YES
Try starting the emulator from the beginning one more time. If this still does not work, then the POD64165/167 could be damaged. Contact your Oki Electric dealer.

A-7. Probe Cable Configuration

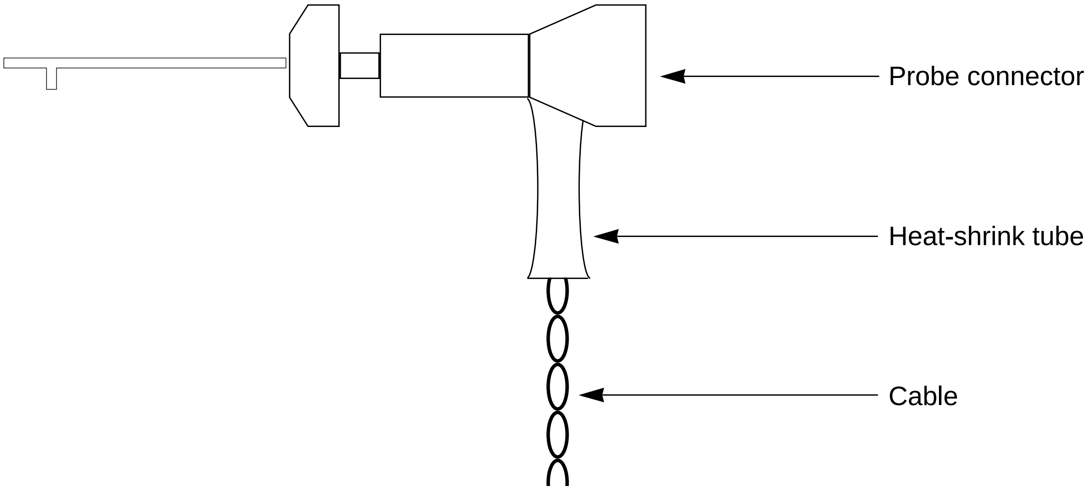
The connector on the right side of the emulation kit marked “PROBE” is for the probe cable. The probe cable configuration is shown below.



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, Peach



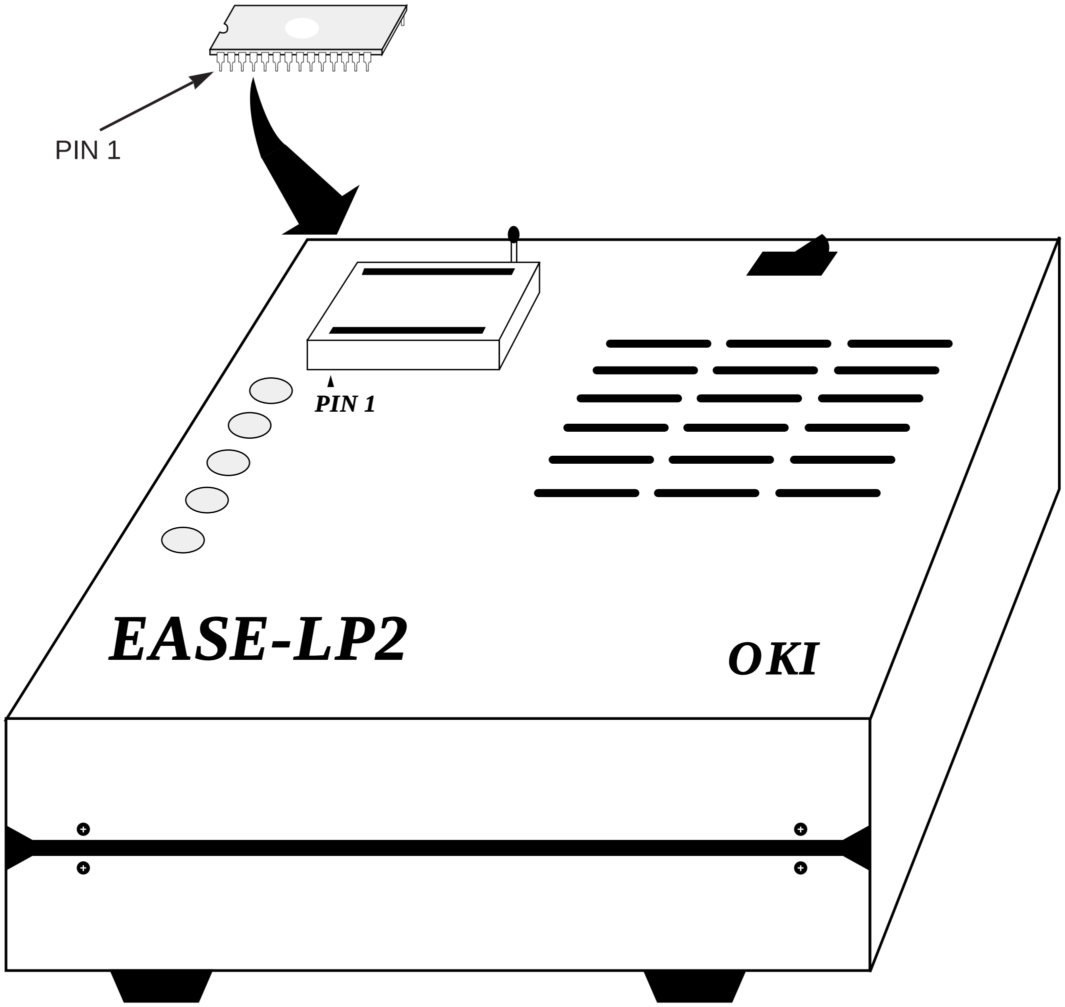
The function of each pin is shown below.

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

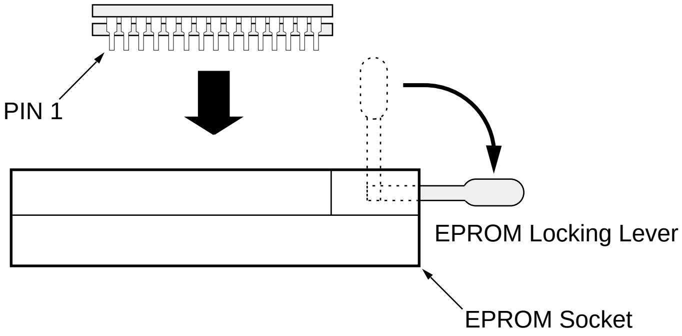
A-8. Mounting EASE-LP2 EPROMs

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.

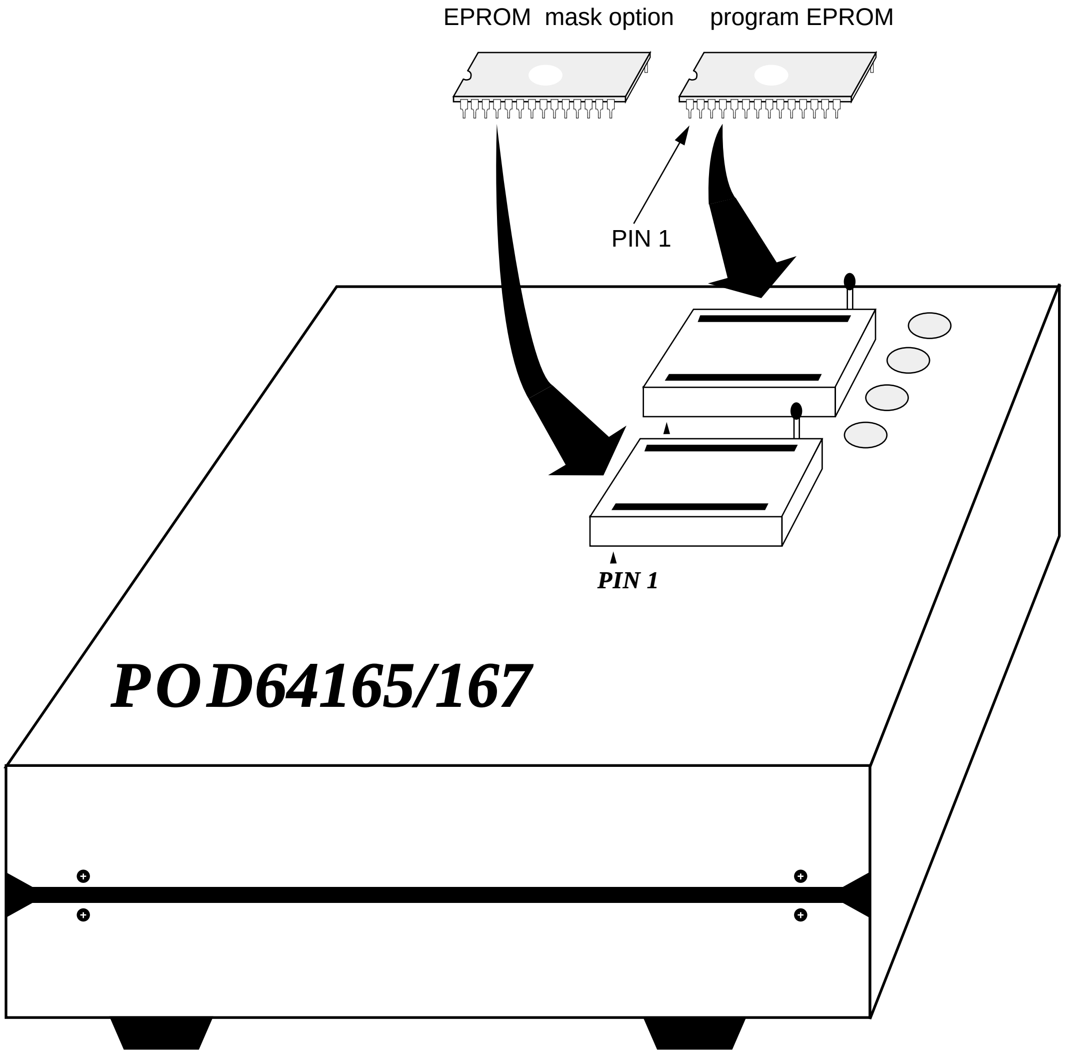
The following types of EPROMs can be written using the EPROM programmer:

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

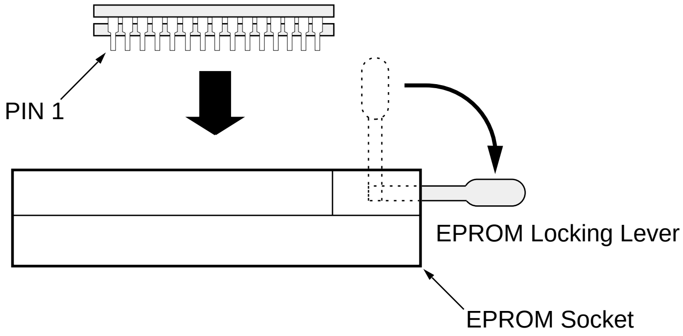
A-9. Mounting POD64165/167 EPROMs

Follow the procedure below to insert a mask option EPROM and program EPROM into the POD64165/167's EPROM sockets.

(1) Release the EPROM locking lever on the top surface of the POD64165/167, as shown below.



(2) Place the program EPROM or mask option EPROM into 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.

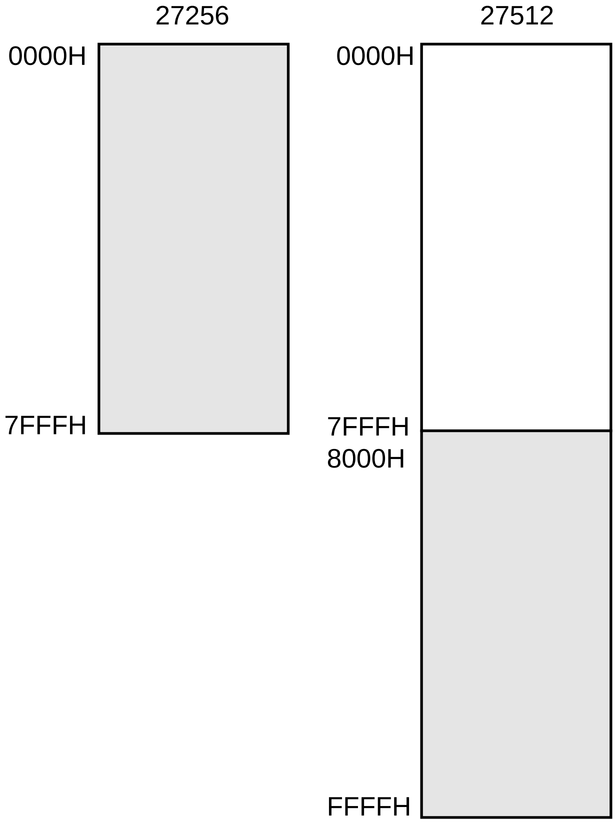
The EPROM write areas are shown below.

(1) Program EPROM

Allowed EPROMs: 27256, 27512, 27C256, 27C512

User Program Write Areas

User program is written into shaded portions.

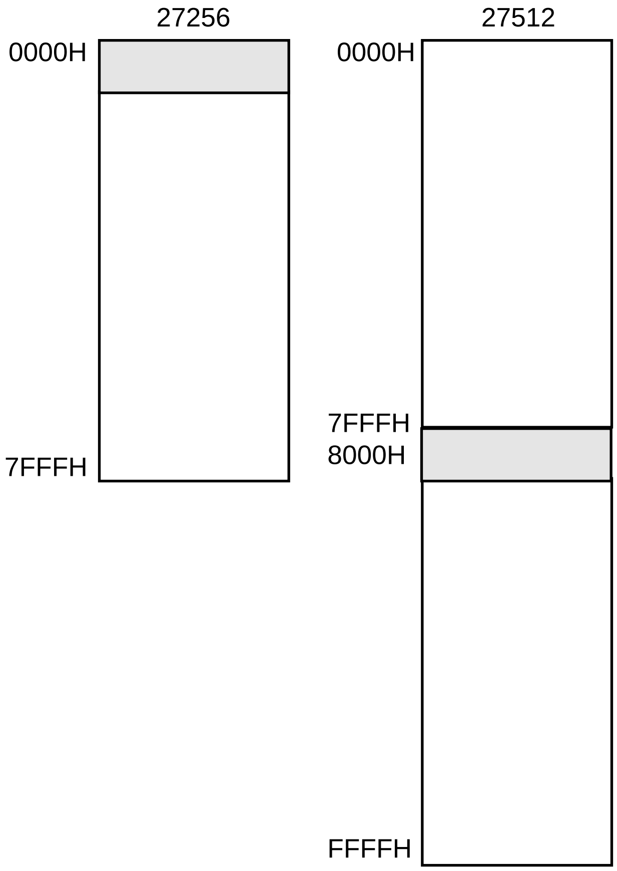


(2) Mask option EPROM

Allowed EPROMs: 27256, 27512, 27C256, 27C512

Mask Option Write Areas

Mask option is written into shaded portions.



A-10. 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 19: 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 terminator 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 MSM64165/167.

`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 150: Connected POD cannot be used.`

Currently connected POD cannot be used.

`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 executed during realtime emulation was input.

`Error 157: COMMAND.COM could not be execute.`

Child process could not be executed in **SH** command.

`Error 158: Illegal parameter. (Can't use in EXPAND mode)`

Any parameter that cannot be used in **EXPAND** mode was input.