
Visual Memory Tutorial Revision 1.00



Dreamcast™

Introduction

Thank you for cooperating in the development of applications for Sega hardware.

This manual explains how to write source code, combine tools, transfer programs to Visual Memory, and interface with the Dreamcast startup screen.

Interfacing with Dreamcast is accomplished by using a data structure called an "information fork." This manual also describes how to start up and operate the transfer utility.

Contents of This Manual

Application Development Procedure

This chapter describes important points that should be observed when starting to write source code, and also how to use, in combination, the various tools that are needed, all of the way up to the creation of the binary files that are transferred to Visual Memory in the end.

Basically, this chapter gives a broad overview of the flow of development work.

Interfacing between Visual Memory and Dreamcast

This chapter explains what must be done in order to establish the proper interface with the Dreamcast startup screen. This chapter also describes the startup screen, the means for implementing the startup screen, file structures, etc.

Memory Card Utility

This chapter explains how to transfer programs from a PC to Visual Memory through the Dev.Box, and how to start up and operate tools.

Appendices

The appendices include a list of label icons that are stored in the Dreamcast boot ROM, and a sample program that is included in the Visual Memory SDK.

Note:

The contents of this manual are subject to change without notice.

Sega Enterprises cannot bear any responsibility for any harm resulting from the use of Visual Memory or this manual.

Trademarks

"Sega," "Dreamcast," and "Visual Memory" are trademarks of Sega Enterprises.

"Microsoft," "Windows," and the Windows CE logo are registered trademarks of Microsoft Corporation in the U.S. and other countries.

Other product names that appear in this manual are trademarks or registered trademarks of their respective owners.

The ™ and ® symbols are not used in this text.

Revision History

November 30, 1998 First Edition

Copyright © 1998 Sega Enterprises

Copyright © ASCII Corporation

Edited and Published by Ascii AAP Publications Editing Department

Relationship To Other Manuals

Visual Memory Hardware Manual

This is a technical reference for the Visual Memory hardware and the system BIOS specifications. The Overview at the beginning of the manual includes a simple summary of the visual memory specifications.

Game designers should read the Overview, and programmers should read the entire manual.

Visual Memory Programmer's Guide

This manual explains how to install and use the Assembler for the Sanyo Electric LC86700, which is installed in the Visual Memory, the Linker, the Library Manager, and "E2H86K.EXE", which creates execution files for visual memory. This manual also explains the LC86700 instruction set and the Assembler syntax.

Refer to this manual for the installation procedure for the Visual Memory SDK, beginning with the Visual Memory Simulator.

Visual Memory Simulator Guide

This manual explains the Visual Memory Simulator, which emulates the operation of Visual Memory through software.

Sega Library Reference Vol. 2

Refer to this manual for details on how to transfer data and programs from Dreamcast to Visual Memory, or how to display images on the LCD of a Visual Memory device that is connected to a Dreamcast controller.

Refer to the backup functions in the Shinobi library for details on reading or writing Dreamcast game data in Visual Memory.

Refer to the LCD functions in the Shinobi library for details on drawing in the LCD.

Restrictions

Other Software and Hardware That Must Be Obtained Separately

The Memory Card Utility requires an RS-232C reverse cable and communications software that will run under Windows. These are not provided in the SDK.

Note that the Memory Card Utility only runs with Dev.Box with Set 5.2X or later. It will not run on a Dev.Box with Set 5.1X or earlier.

In order to execute programs, CodeScape and GD Workshop are both required. These are both included in the Dreamcast SDK, so be sure to install them ahead of time.

Technical Support

If, in the course of your development work, you have any technical questions, encounter hardware difficulties, need information that is not included in this manual, or experience a hardware malfunction, contact technical support at:

Sega Enterprises
Technical Support Center
1-2-12, Haneda, Ota-ku, Tokyo 144-8531
Japan
Tel: 03-5736-7355
Fax: 03-5736-5357
E-mail: katana@sft.sega.co.jp

contents

Introduction	2
Relationship To Other Manuals	4
Restrictions	5
Technical Support	6

Chapter 1

Application Development Procedure	11
.....	
1.1 Writing Source Code	11
1.2 Correcting GHEAD.ASM	12
1.3 Assembly Without Using MAKE	12
1.3.1 Assembly	13
1.3.2 Linking	14
1.3.3 Converting an EVA File Into a HEX File	15
1.3.4 Converting a HEX File to a Binary File	16
1.3.5 Creating a MAKE File	16
1.4 Creating the Information Fork	18
1.5 Transferring the Program to Visual Memory	18

Chapter 2

Interfacing between Visual Memory and Dreamcast

19

2.1 Names of Elements in the Startup Screen	20
2.1.1 Memory Selection Screen	20
2.1.2 File Management Screen	22
2.2 Creating a Volume Icon	25
2.3 Creating an Animated Icon	27
2.3.1 Three File Structures	27
2.3.2 Information Fork	28
2.3.3 Visual Comment Data Structure	33
2.3.4 Game Name Sorting Rules	34

Chapter 3

Memory Card Utility

39

3.1 Memory Card Utility Preparation and Startup	39
3.1.1 Requirements for Transfer	39
3.1.2 Software Preparation	42
3.1.3 Memory Card Utility Startup	45
3.2 Memory Card Utility Operation	50
3.2.1 Main Menu	50
3.2.2 Memory Selection Menu	51
3.2.3 Command Selection Menu	52
3.2.4 File Operations Menu	59
3.3 Initializing Visual Memory	64
3.4 Transferring Files from a PC to Visual Memory	67

Contents

Appendix A	
Little Endian Format	73

Appendix B	
List of Label Icons	75

Appendix C	
Sample Program Listings	77

C.1 LCD Pattern Display	77
C.2 LCD Character Pattern Display	80
C.3 Counter That Uses Base Timer Interrupts	84
C.4 Button Press Detection	90
C.5 Using the PWM Sound Source	95
C.6 Interrupt Using Timer 0	97
C.7 Serial Communications (Sending Side)	101
C.8 Serial Communications (Receiving Side)	107
C.9 General-purpose Serial Driver	113
C.10 Reading and Writing Flash Memory	123
C.11 Low Battery Detection and Saving Data	129

Chapter 1

Application Development Procedure

Reference

For details on indirect address registers, refer to the “Visual Memory Hardware Manual.”

Caution

The Assembler and Linker use EMS. Before starting, display the MS-DOS prompt properties and enable EMS memory usage in the “EMS Memory” group under the “Memory” tab.

EMS memory cannot be used when EMM386.EXE is embedded in CONFIG.SYS and the NOEMS option is specified, so the NOEMS option must be removed.

Note

An EVA-format file is a file that uses special debugging hardware. Because the Visual Memory Simulator is used in the development of applications for Visual Memory, think of the EVA file as a temporary file.

Extension	Description
H00	This file can be loaded into the Visual Memory Simulator. The loading time is reduced because only the code itself is saved.
HEX	The 64K-byte image of bank 0 in flash memory is stored in this file. No matter how small the program is, a 64K file is created. Whatever portion that is not filled by the program is filled with “00H”. This file can be loaded into the Visual Memory Simulator, but it will not function properly.

This chapter explains the application development procedure, from coding the program to checking the program on an actual machine. This section assumes that the application specifications have already been established.

1.1 Writing Source Code

The following declarations must be made at the start of the program:

```
chip LC868700
World external
Public main
Extern _game_end
```

Because all Visual Memory applications will be stored in flash memory, “external” must be declared in the “world” statement.

When an application is called from system BIOS, address 0000H in flash memory is called. Because “jump main” is written in 0000H by GHEAD.ASM, the application provides the label “main” for entry into game mode. Because “main” is referenced from GHEAD.ASM, the “public” declaration is used.

Conversely, when an application ends, it jumps to “_game_end” in GHEAD.ASM, so the “extern” declaration is used to indicate that this label is external to the application.

When an application calls a flash memory-related BIOS or a clock-related BIOS, the “extern” declaration is used to indicate that “fm_wrt_ex”, “fm_vrf_ex”, “fm_prd_ex”, etc., are external programs.

Next, the structure of the indirect address register for the data segment (DSEG) is defined. The entire data segment is expanded in RAM. Because addresses 0000 to 000FH in RAM are indirect address registers, 16 bytes of RAM should be allocated for these registers, whether or not the application will use the indirect address registers. The area in RAM that can be used by an application starts from address 0010H.

Reference

For details on indirect address registers, refer to the “Visual Memory Hardware Manual.”

Start the code segment (CSEG) from an address higher than 0280H (org 280H). GHEAD.ASM uses 0000H to 01FFH, and the information fork uses 0200H to 027FH (minimum).

GHEAD.ASM contains interrupt vector definitions, and BIOS cal and return destinations. The information fork data such as the application name and icon. The size of the information fork is variable because of choices that the designer can make:

the icon may or may not be animated, or one application's icon may be larger than another's.

If the information fork image is pre-determined, add the size of the information fork image to 0200H and start the program from that address.

Reference

For details on the information fork, refer to [Chapter 2](#), “[Interfacing between Visual Memory and Dreamcast](#).”

1.2 Correcting GHEAD.ASM

Once you have written the application source code, it is necessary to correct GHEAD.ASM.

If the application uses interrupts, describe the vector table for the interrupts that are to be used in GHEAD.ASM.

Even if interrupts are not to be used, it is still necessary to define an interrupt vector table. Also write the interrupt handler so that it does nothing except execute “RETI”.

Because the program jumps to the start (0000H) of GHEAD.ASM if the user selects game mode, a jump instruction to the main routine of the game should be written at the start of GHEAD.ASM.

The processing that is to be performed for BIOS calls is described starting from address 100H. Do not change this processing. Because the BIOS in ROM specifies addresses directly and then returns control to flash memory, BIOS calls will not be made correctly if addresses change by even one byte.

When writing to flash memory in particular, it is necessary to use 1/6 RC for the system clock. Make this change within the application program by calling fm_wrt_ex, fm_vrf_ex, fm_prd_ex, and then changing over to crystal oscillation when control returns to the main program.

Note

When loading from flash memory, it does not matter if 1/6 or 1/12 RC is used.

Also be careful not to change the “org” instruction that specifies each BIOS start address.

1.3 Assembly Without Using MAKE

This section explains how to assemble and link the source code, and then build a file in a format that can be actually executed in Visual Memory.

Caution

The Assembler and Linker use EMS. Before starting, display the MS-DOS prompt properties and enable EMS memory usage in the “EMS Memory” group under the “Memory” tab.

EMS memory cannot be used when EMM386.EXE is embedded in CONFIG.SYS and the NOEMS option is specified, so the NOEMS option must be removed.

1.3.1 Assembly

Execute the “M86K” command from the MS-DOS command prompt to assemble the source code.

For example, if the source code file name is “TEST.ASM”, set the current drive and the current directory to the directory where “TEST.ASM” resides, and then perform the assembly process by executing the following command:

```
C>M86K TEST.ASM
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights
reserved.
```

```
Pass 1 .....
Source file:      TEST
Chip name:        LC868700
ROM size:         60K bytes
RAM size:         512 bytes
XRAM size:        196 bytes
Pass 2 .....
```

When assembly is completed, an object file with the extension “.OBJ” is created.

Assembling GHEAD.ASM in the same manner creates GHEAD.OBJ.

If an error message similar to the following appears during the assembly process, a problem exists in the line indicated by the line number in the message.

```
Pass 1 .....
TEST.ASM(93):                move            #080h,b
** Error, syntax error near #
0 warning(s) and 1 error(s) were detected. Further execution aborted.
```

Correct the source code so that no warnings or errors are generated.

Reference

For details on the M86K assembler's warning messages and error messages, refer to “Visual Memory Programmer's Manual.”

Caution

Except when incorporating source code equivalent to GHEAD.ASM into your own source code, GHEAD.OBJ and the user program object file are both required. Note that interrupt vectors, interrupt service routines, BIOS call programs, etc., are described in GHEAD.ASM, and are placed in addresses below the user program by the Linker that is executed next.

1.3.2 Linking

After preparing an object file (created by the Assembler) and a GDUMMY.OBJ file that indicates the addresses in internal ROM where BIOS is written, use the Linker to create an EVA-format file.

Note

An EVA-format file is a file that uses special debugging hardware. Because the Visual Memory Simulator is used in the development of applications for Visual Memory, think of the EVA file as a temporary file.

Before executing the linker, make a note of the GDUMMY.OBJ path. Then input the following command line to link each of the object files.

```
D>L86K GHEAD.OBJ IFORK.OBJ TEST.OBJ -C=200
C:\VM_SDK\LC86K\OBJ\GDUMMY.OBJ,
TEST.EVA,,,
```

```
SANYO (R) LC86K series Linkage Loader Version 6.00c
Copyright (c) SANYO Electric Co., Ltd. 1989-1997. All right reserved.
```

```
Pass 1 ...
Pass 2 ...
Pass 3 ...
```

```
Link process complete !!
TEST.EVA created
```

The option “-C=200” specifies the address in flash memory where the user program that is specified immediately afterwards is to be placed.

If an error message is displayed, review GHEAD.ASM and the user program. Check the labels that are used for BIOS calls in particular.

1.3.3 Converting an EVA File Into a HEX File

The Linker combines all of the object files into a single file with the “.EVA” extension. The next step is to convert this file into a file that can be loaded into Visual Memory or the Visual Memory Simulator. Input the following command line.

```
D>E2H86K TEST.EVA
SANYO LC86000 Series EVA-file to HEX-file generator  V1.21A
Copyright (C) SANYO Electric Co.,Ltd. 1992-1997

EVA file name:  TEST.EVA
ROM data packed: FF(hex)
Chip name:      LC868716

All ROM(64KB)  block records: 03875
All ROM(64KB)  block records: 04096
Module name: GHEAD      External CSEG(In)    0000 - 0002 records: 00001
Module name:           External CSEG(In)    0003 - 0004 records: 00001
Module name:           External CSEG(In)    000B - 000C records: 00001
Module name:           External CSEG(In)    0013 - 0014 records: 00001
Module name:           External CSEG(In)    001B - 001C records: 00001
Module name:           External CSEG(In)    0023 - 0024 records: 00001
Module name:           External CSEG(In)    002B - 002C records: 00001
Module name:           External CSEG(In)    0033 - 0034 records: 00001
Module name:           External CSEG(In)    003B - 003C records: 00001
Module name:           External CSEG(In)    0043 - 0044 records: 00001
Module name:           External CSEG(In)    004B - 0057 records: 00002
Module name:           External CSEG(In)    0100 - 0105 records: 00001
Module name:           External CSEG(In)    0110 - 0115 records: 00001
Module name:           External CSEG(In)    0120 - 0125 records: 00001
Module name:           External CSEG(In)    0130 - 013B records: 00001
Module name:           External CSEG(In)    01F0 - 01F4 records: 00001
Module name: IFORK      External CSEG(In)    01F5 - 0474 records: 00041
Module name: TEST       External CSEG(In)    0475 - 051C records: 00011
```

There are no option switches.

Executing this command results in the creation of a file with the “.H00” extension and a file with the “.HEX” extension.

Extension	Description
H00	This file can be loaded into the Visual Memory Simulator. The loading time is reduced because only the code itself is saved.
HEX	The 64K-byte image of bank 0 in flash memory is stored in this file. No matter how small the program is, a 64K file is created. Whatever portion that is not filled by the program is filled with “00H”. This file can be loaded into the Visual Memory Simulator, but it will not function properly.

Caution

Normally, only the H00 file is used.

Once the H00 file has been created, an operation check is performed in the Visual Memory Simulator. However, because the Visual Memory Simulator does not have the same clock as the actual machine, a timing check is not appropriate.

Divide the debugging phase so that the program logic is checked in the simulator and the timing and speed are checked on the actual machine.

Reference

For details on the Visual Memory Simulator, refer to the “Visual Memory Simulator Guide.”

1.3.4 Converting a HEX File to a Binary File

This procedure uses H2BIN.EXE to convert an H00 file that was created by the E2H86K into a binary file (extension “.BIN”).

Input the following command line.

```
H2BIN TEST.H00 TEST.BIN
```

There are no option switches. The second parameter “TEST.BIN” may be omitted. If it is omitted, the extension “.BIN” is automatically used.

This procedure creates a file that can be loaded into Visual Memory on the actual machine.

1.3.5 Creating a MAKE File

If a MAKE file is created for the MAKE command, it is possible to perform the assembly, linking, and file format conversion processes through batch processing.

If the dependence information, such as which files to insert in which commands and which files are output, is described in the MAKE file and the MAKE command is executed, the command compares the time stamps of the files that are to be inserted and are to be output, and then assembles and links only those files that have been updated.

For details on the MAKE command, refer to the “Visual Memory Programmer's Manual.”

Caution

Because the MAKE command is provided for a variety of development environments, when the computer that you are using has multiple development environments installed, either change the command retrieval path (the environment variable “PATH”) or change the file name of the MAKE command.

The following file is an example of the type of MAKE file to create in order to MAKE the series of procedures described up to this point. For this example, we will assume that file name is "TEST.MAK".

```
TARGET = test
OBJECTS = ifork.obj test.obj
HEADOBJ = ghead.obj
SYSOBJ = $(TOOL86)\obj\gdummy.obj

.asm.obj:
    m86k $*

$(TARGET).eva: $(HEADOBJ) $(OBJECTS)
    186k $(HEADOBJ) $(SYSOBJ) $(OBJECTS), $(TARGET).eva,,,

$(TARGET).h00: $(TARGET).eva
    e2h86k $(TARGET)

$(TARGET).hex: $(TARGET).h00
    h2bin $(TARGET).h00
```

This MAKE file is built by specifying MAKE as shown below. This assembles and builds the source files that have been updated.

Caution

Specify the "/F" option when executing the MAKE file. Input the command line in this format: "MAKE /F <MAKE file name>".

```
D>MAKE /F TEST.MAK
```

```
SANYO LC86000 Series MAKE Utility Version 1.00A
Copyright (C) SANYO Electric Co.,Ltd. 1993-1994 All rights reserved.
```

```
m86k GHEAD
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights
reserved.
```

```
Pass 1 .....
Source file:    GHEAD
Chip name:     LC868700
ROM size:      60K bytes
RAM size:      512 bytes
XRAM size:     196 bytes
Pass 2 .....
```

```
m86k IFORK
```

```
SANYO (R) LC86K series Macro Assembler Version 4.0K
Copyright (c) SANYO Electric Co., Ltd. 1989-1995. All rights
reserved.
```

(Subsequently, the Linker, E2H86K, and then H2BIN are executed and a binary file is created.)

1.4 Creating the Information Fork

In the sample in the Visual Memory SDK, IFORK.ASM is created and then a binary file is created. It is also possible to use a binary editor, etc., to fill the information fork of the binary file that was produced.

The information fork is filled with the icons and application names that are displayed on the Dreamcast file management screen, the comments that are displayed in Visual Memory file mode, the game names (sort keys), and comments that use larger icons.

Of these, the required items are VM comment data, GUI comment data, game names, the number of icons, visual types, and icon information (for a minimum of one icon).

Refer to chapter 2, “Interfacing between Visual Memory and Dreamcast,” while editing the information fork.

1.5 Transferring the Program to Visual Memory

Once editing of the information fork is complete, transfer the file to Visual Memory. The “Memory Card Utility” that is provided in the Visual Memory SDK is used to transfer the program.

Caution

The Memory Card Utility is provided in ELF file format as a Dev.Box application. It is not a program for general-purpose personal computers.

The following hardware and software is needed in order to transfer a program to Visual Memory:

- 1) An RS-232C cross cable
- 2) A communications program that runs under Windows
- 3) A debugger, such as CodeScape
- 4) GD Workshop
- 5) Dev.Box (Set 5.2X or later)

Note that items 1) and 2) are not provided in our SDK, and must be obtained separately.

Reference

For details on the transfer method, refer to chapter 3, “Memory Card Utility.”

Chapter 2

Interfacing between Visual Memory and Dreamcast

This chapter describes the interface between the Dreamcast startup screen with Visual Memory. This interface is installed in boot ROM, and is deeply inter-related with the menu screen that is displayed when Dreamcast is started up.

Following the explanation of the menu screen is a description of the data structure of the file that implements that menu screen.



Caution

The descriptions in this manual are based on boot ROM version 1.001. Some functions or names may be different in future upgrades.

Note that the version number is not displayed in the upper right corner on the actual machine.

2.1 Names of Elements in the Startup Screen

After the opening animation that is displayed when the Dreamcast power is turned on, the screen shown on the previous page is displayed. This screen is called the “Main Menu.” The Main Menu is controlled and administered by the boot ROM in Dreamcast.

2.1.1 Memory Selection Screen

If a file for which Visual Memory is displayed is selected from the Main Menu, the following screen appears.



This screen is called the “Memory Selection Screen.” As of November 30, 1998, the term “memory” refers to “Visual Memory,” but other storage media may be available in the future.

When there is more than one controller or memory module connected to Dreamcast or a controller, a list of these devices is displayed.

Caution

Devices other than storage media, such as a voice recognition device, are not displayed.

When you look at the list, you will notice some icons that have an image of a monster in them, some that have an image of an animal, and others that are empty. Unique icons can be assigned to each memory module. (One icon per module.)

There are two types of icons: those that the end user uniquely assigns, and those that are displayed by writing a special file in Visual Memory. The icon in the lower center indicates memory that has not been initialized.

Label Icons

Those icons that the end user uniquely assigns are called “label icons.” when memory has been initialized, the user can freely assign any of the 124 icons stored in boot ROM. In the case of Visual Memory, the same icon that is displayed on the screen is also displayed on the LCD on the Visual Memory unit.

Note that when both a label icon and a volume icon (explained later) are assigned to one memory module, the volume icon takes precedence and is displayed.

Reference

The pattern data for the icons in boot ROM can be read by using the boot ROM font function. For details on the boot ROM font function, refer to the “Sega Library Manual Vol. 2.” For a list of labels, refer to the Appendix, “List of Label Icons.”

Volume Icons

The monster icons are called “volume icons.” Volume icons can be implemented by storing a file called “ICONDATA_VMS” in Visual Memory. Accordingly, the end user cannot assign volume icons.

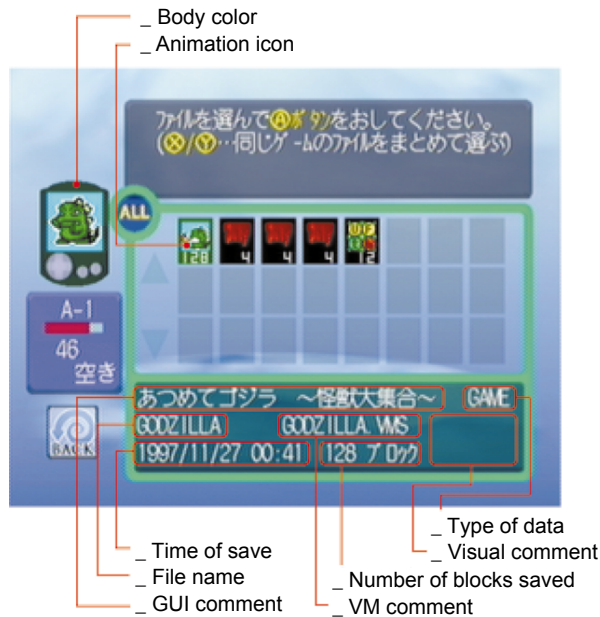
These icons are 32×32 graphic images that use 16 colors out of a possible 65,536 (ARGB4444). In the case of Visual Memory, when the unit is connected to a Dreamcast controller, the same graphic as the volume icon that is displayed on the screen is displayed on the LCD of the Visual Memory unit in monochrome. In order to display volume icons, the file “ICONDATA_VMS” must be prepared and the Memory Card Utility must be used to transfer that file into memory.

Reference

For details on how to transfer the volume icon file, refer to [Chapter 3](#), “[Memory Card Utility](#).”

2.1.2 File Management Screen

Once a memory module is selected from the Memory Selection screen, the following screen is displayed. This screen is called the “File Management Screen.”



This screen displays a list of the applications that are stored in the memory module that was selected, and a list of the Dreamcast game save data.

When a file is selected, detailed information on that file is displayed on the bottom portion of the screen.

① Body Color

A single color can be assigned to a single memory module. This color can be specified when initializing the memory. The end user can also change the color. Note that because the color information is stored within Visual Memory, it cannot be changed through Dreamcast in real time.

② Animated Icon

The 32×32 icons can be displayed with a graphic that uses 16 colors out of a possible 65,536. Data for up to three patterns can be used to display animation.

One icon represents one file. When an icon is selected, the border flashes yellow and detailed information on the selected file is displayed on the bottom portion of the screen. Application files are displayed with green borders and data files are displayed with black borders.

Note

When multiple files are selected by using the X button and the Y button, the GUI comments and the total number of blocks for all of the selected files are displayed.

③ GUI Comments

GUI comments can be displayed using normal-width letters numbers, symbols, and kana, and double-width characters. The character string length is 32 bytes. The normal-width characters that can be displayed are ASCII codes 20H to 7EH and 0A1H to 0DFH. The double-width characters are the Shift JIS codes.

Note

JIS Level 2 characters are also supported. JIS X 0208-1983 is supported, so musical notes and other symbols can also be displayed.

④ File Names

A list of the files that are stored in that memory module is displayed. The characters that are not shaded in the following table can be used in file names.

		Lower 4 bits															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Upper 4 bits	0			Space	0	@	P	'	p				一	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			”	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			・	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
	8			(8	H	X	h	x			イ	ク	ネ	リ		
	9)	9	I	Y	I	y			ウ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			エ	コ	ハ	レ		
	B			+	:	K	[k	{			オ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ヤ	シ	フ	ワ		
	D			—	=	M]	m	}			ユ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ヨ	セ	ホ	”		
	F			/	?	O	_	o				ッ	ソ	マ	°		

Characters That Can Be Used in File Names

Note

Lower-case letters may not be used.

Although a “-” can be input in the Memory Card Utility, do not use this character in file names. Such an application will not be in conformance with the software creation standards.

⑤ VM Comments

These are comments that are displayed in Visual Memory file mode. In addition to the characters that can be used in file names, lower-case letters can also be used in VM commands.

⑥ Save Time

The date and time at which a file was saved are displayed. This data cannot be changed from within an application.

⑦ Number of Blocks Used

The file size is shown in blocks. Since one block is 512 bytes, in the case of Visual Memory (HKT-7000) a maximum of 200 blocks can be used for data storage, and a maximum of 128 blocks can be used for the game. This data cannot be changed from within an application.

⑧ Data Type

This indicates whether the file in question is an application or Dreamcast save data. This data cannot be changed from within an application.

Caution

Changes are possible only when the Memory Card Utility was used.

⑨ Visual Comments

A 72×56 graphic that uses up to 65,536 colors (ARGB4444) can be displayed. It is also possible to not display visual comments.

2.2 Creating a Volume Icon

In order to display a volume icon on the Memory Selection Screen, create a file named ICONDATA.VMS with the file specifications described below.

Reference

In the Visual Memory SDK, samples are contained in the “Volumeicon” folder. The assembly source code is also provided for reference.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	VM comment data															
0010	Monochrome icon data start address				Color icon data start address						Reserved					
0020	Monochrome icon pattern data (32 × 32 dots)															
0030																
0040																
0050																
0060																
0070																
0080																
0090																
00A0	Color icon palette data (16 colors)															
00B0	Color icon pattern data (32 × 32 dots)															
00C0																
00D0																
00E0																
00F0																
0100																
0110																
0120																
0130																
0140																
0150																
0160																
0170																
0180																
0190																
01A0																
01B0																
01C0																
01D0																
01E0																
01F0																
0200																
0210																
0220																
0230																
0240																
0250																
0260																
0270																
0280																
0290																
02A0																
02B0																

VM Comment Data

This is filled with a 16-byte comment. This is displayed when ICONDATA.VMS is selected on the Visual Memory File Management Screen or the Dreamcast File Management Screen.

Comments are displayed in Visual Memory file mode. Refer to section 2.1.2, “File Management Screen,” for the characters that can be used in a VMS comment. Fill any unused bytes with the space character (20H).

Monochrome Icon Data Start Address

This specifies the starting address of pattern data for a monochrome icon as an offset address from the start of the file.

Normally, this data is 00000020H. (“20 00 00 00” in a memory dump.)

Caution

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Color Icon Data Start Address

This specifies the starting address of palette data for a color icon as an offset address from the start of the file.

Normally, this data is 000000A0H. (“A0 00 00 00” in a memory dump.)

Caution

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Note that this offset address points to the starting address of the palette data for color icons, not the pattern data.

Reserved Area

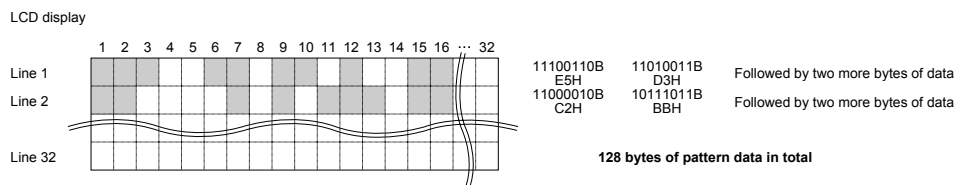
This area is reserved for future expansion. Fill this eight-byte area with “00”.

Monochrome Icon Pattern Data

This data specifies the 32×32 -dot monochrome volume icon that is displayed on the LCD of the Visual Memory unit while the Memory Selection Screen or the File Management Screen is displayed.

This data is pattern data, starting from the upper right of the LCD and heading towards the lower left. One byte contains the pattern data for eight dots. The MSB of the data is the left-hand bit, and the LSB is the right-hand bit. Setting a bit to “1” causes the corresponding dot to be displayed as black (blue) on the LCD.

One line (32 dots) requires four bytes of data, so the $32\text{-dot} \times 32\text{-line}$ pattern requires 128 bytes of pattern data.



Color Icon Palette Data

Color icons can be displayed with a 16-color graphic. Write the 16-color palette in this area with ARGB4444 palette data. Each palette data entry consists of two bytes.

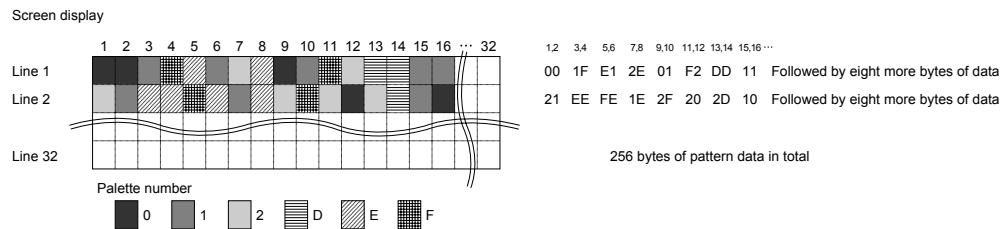
A value from “0” to “0FH” can be specified for each of A, R, G, and B. Note that a value of “0” for A makes the color transparent, and a value of “0FH” makes the color opaque.

Color Icon Pattern Data

This is the pattern data for the icon that is displayed on the Memory Selection Screen or the File Management Screen.

This data is specified with palette numbers, starting from the upper right and heading towards the lower left. Four bits of data specify the palette number for one dot. The upper four bits are the palette number for the right-hand dot, and the lower four bits are the palette number for the left-hand dot.

One line consists of 32 dots, which requires 16 bytes of data, so the 32-dot × 32-line pattern requires 512 bytes of pattern data.



2.3 Creating an Animated Icon

The data for animated icons and GUI comments that are displayed on the File Management Screen must be included in the files themselves. Because there are three different file structures, this section explains the file types and the file structures.

2.3.1 Three File Structures

There are three files stored in memory, and each has its own file structure.

ICONDATA_VMS Format

This is the structure that was described in the previous section. This file does not have an information fork.

Note

For details, refer to section 2.2, “Creating a Volume Icon.”

Visual Memory Application Format

Files that can be executed in Visual Memory game mode must have the following file structure:

Address	Contents
0000	Visual Memory header (equivalent to GHEAD.ASM)
0200	Information fork
xxxx	Application code

Data File Format

A file that stores game data for a Dreamcast application must have the following file structure:

Address	Contents
0000	Information fork
0200	Game data

2.3.2 Information Fork

Files other than the “ICONDATA_VMS” file have a section called an “information fork.” This information is the colored portion of a file dump displayed by the Memory Card Utility. All detailed file information is contained in this information fork.

Caution

The data area for an animated icon is not displayed in color.

Note

Refer to the Visual Memory SDK sample “total”, since it includes an information fork for an animated icon and visual comments.

The structure of the information fork is described below. Note that the addresses in the table are given as offset addresses from the start of the information fork.

Interfacing between Visual Memory and Dreamcast

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	VM comment data															
0010	GUI comment data															
0020																
0030	Game name (sort key)															
0040	Number of icons	Animation speed	Visual type	CRC				Save data size				Reserved				
0050	Reserved															
0060	Icon palette data (16 colors)															
0070																
0080	Icon #1 pattern data (32 × 32 dots)															
0090																
I																
0260																
0270																
0280	Icon #2 pattern data (32 × 32 dots)															
0290																
I																
0460																
0470																
0480	Icon #3 pattern data (32 × 32 dots)															
0490																
I																
0660																
0670																
0680	Visual comment (palette and pattern data)															
0690																
XXXX																

VM Comment Data

This contains a 16-byte comment. This comment is displayed when the file is selected on the Visual Memory File Management Screen or the Dreamcast File Management Screen.

The comment is displayed in Visual Memory File Mode. The characters that are not shaded in the following table can be used in VM comments. Note that any bytes that are not used should be filled with space characters (20H).

		Lower 4 bits															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Upper 4 bits	0			Space	0	@	P	'	p				一	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			”	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			・	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
	8			(8	H	X	h	x			ィ	ク	ネ	リ		
	9)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			エ	コ	ハ	レ		
	B			+	:	K	[k	{			オ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ャ	シ	フ	ワ		
	D			—	=	M]	m	}			ュ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ヨ	セ	ホ	”		
	F			/	?	O	_	o				ッ	ソ	マ	°		

Characters That Can Be Used in VM Comments

GUI Comment Data

This contains a 32-byte comment that is displayed on the File Management Screen. The comment is displayed when the file is selected on the Dreamcast File Management Screen.

The normal-width characters that can be used are those that are not shaded in the following table. The double-width characters are the Shift JIS codes; JIS Level 2 characters are also supported. JIS X 0208-1983 is supported, so musical notes and other symbols can also be displayed.

Note that any bytes that are not used should be filled with space characters (20H).

		Lower 4 bits															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Upper 4 bits	0			Space	0	@	P	‘	p				一	タ	ミ		
	1			!	1	A	Q	a	q			。	ア	チ	ム		
	2			”	2	B	R	b	r			「	イ	ツ	メ		
	3			#	3	C	S	c	s			」	ウ	テ	モ		
	4			\$	4	D	T	d	t			、	エ	ト	ヤ		
	5			%	5	E	U	e	u			■	オ	ナ	ユ		
	6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
	7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
	8			(8	H	X	h	x			イ	ク	ネ	リ		
	9)	9	I	Y	I	y			ウ	ケ	ノ	ル		
	A			*	:	J	Z	j	z			エ	コ	ハ	レ		
	B			+	;	K	[k	{			オ	サ	ヒ	ロ		
	C			,	<	L	¥	l				ヤ	シ	フ	ワ		
	D			—	=	M]	m	}			ユ	ス	ヘ	ン		
	E			.	>	N	^	n	~			ヨ	セ	ホ	”		
	F			/	?	O	_	o				ツ	ソ	マ	°		

Normal-width Characters That Can Be Used in GUI Comments

Game Name (Sort Key)

File names are sorted when they are listed on the Dreamcast File Management Screen. This area is used for the sort key. 16 bytes of data are specified for this area; fill this area with unique character code display data.

Reference	For details on assigning game names and the unique code table, refer to section 2.3.4, “Game Name Sorting Rules.”
-----------	---

Number of Icons

For an animated icon, specify either “2” or “3” in this field. Specify “1” for a still-image (normal) icon.

The range of values that can be specified in this field is “1” to “3,” so an animation pattern can consist of a maximum of three patterns.

Caution

Do not specify a value outside the range of “1” to “3”. Operation is not guaranteed if “0” or a value of “4” or more is specified.

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Animation Speed

This specifies the speed at which the icons are switched when using an animated icon. The range of values that can be specified is from “1” to “65,535.” If the specified value is “n,” the animation patterns are switched every $n/30$ seconds.

For example, if “1” is specified, the animation patterns are switched every $1/30$ of a second. If “30” is specified, the animation patterns are switched every second. If “65,535” is specified, the animation patterns are switched roughly every 36 minutes.

When there are three animation patterns, they are displayed in the sequence $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow \dots$. If there are two animation patterns, they are displayed alternately. If there is only one icon, this value is meaningless.

Caution

Do not specify a value of “0.” Operation is not guaranteed if “0” is specified.

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Visual Type

This specifies the type of the visual comment that is displayed in the lower right corner of the Dreamcast File Management Screen. A 72×56 -dot graphic can be displayed for a visual comment.

Caution

Animation cannot be used for the visual comment. Also note that using a graphic with a lot of colors will consume more memory.

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

The specifiable values and the corresponding visual comment types, number of bytes required, and number of blocks used are listed in the following table.

Specified value	Visual comment type	Number of bytes required	For data	For palette	Number of blocks used
0	None	0	0	0	0
1	Direct color (type A)	8064	8064	0	16
2	256-color graphic (type B)	4544	4032	512	9
3	16-color graphic (type C)	2048	2016	32	4

Reference

For details on visual comments, refer to section 2.3.3, “Visual Comment Data Structure.”

CRC

Write the CRC (error checking/correction code) in this field.

When a file is saved by using the backup utility function “buMakeBackupFileImage()” from the Sega Library, the CRC is calculated automatically, and that value is written in this field. Note that the CRC applies only to the data portion, and not to the information fork.

Caution

When a Visual Memory application is transferred by using the Memory Card Utility, there is no need to write the CRC value or to perform a CRC check. In this case, fill this field with “00 00.”

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Save Data Size

Specify the size of the data area (not including the information fork) in bytes.

Caution

When a Visual Memory application is transferred by using the Memory Card Utility, there is no need to write the data size. In this case, fill this field with “00 00.”

Specify the data in Little Endian format. For details on Little Endian format, refer to the Appendix, “Little Endian Format.”

Reserved Area

This area is reserved for future expansion. Fill this 20-byte area with “00.”

Icon Palette Data

This specifies the 16 colors of the palette that is used for the icon pattern that follows.

Specify the palette data in ARGB4444 format. Specify two bytes for each color.

A value from “0” to “0FH” can be specified for each of A, R, G, and B. Note that a value of “0” for A makes the color transparent, and a value of “0FH” makes the color opaque.

This palette also is used for icons #2 and #3. Note that it is not possible to change the palette for each icon.

This is the pattern data for a 32×32 -dot, 16-color icon.

One line consists of 32 dots, which requires 16 bytes of data, so the 32-dot \times 32-line pattern requires 512 bytes of pattern data.

Screen display

Line 1

Line 2

Line 32

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ... 32

12 34 56 78 910 1112 1314 1516...

01 F1 E1 2E 01 F2 DD 11 Followed by eight more bytes of data

21 EE FE 1E 2F 20 2D 10 Followed by eight more bytes of data

256 bytes of pattern data in total

Palette number

0 1 2 D E F

When using a visual comment (when a value other than “0” was specified for the visual type), specify the data in this field. The visual comment data is the data for a 72 × 56-dot graphic.

Direct Color (Type A)

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	ARGB4444 data for (0,0)			ARGB4444 data for (1,0)			ARGB4444 data for (2,0)			ARGB4444 data for (7,0)	
0010	ARGB4444 data for (0,8)		
1F70	ARGB4444 data for (7,15)	

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	Palette 0		Palette 1			Palette 7	
01F0		Palette 255	
0200	Palette number for (0,0)	Palette number for (1,0)	Palette number for (2,0)	Palette number for (15,0)
0010	Palette number for (17,0)
11B0	Palette number for (11, 5)

16-color Graphic (Type C)

This has palette data for 16 colors. The palette colors are specified in ARGB4444.

The pattern data is specified with four bits for each dot. One byte contains data for two dots; the upper four bits specify the palette number for the left-hand dot, and the lower four bits specify the palette number for the right-hand dot.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	Palette 0		Palette 1			Palette 7	
0010		Palette 15	
0020	Palette number for (0,0)(1,0)	Palette number for (2,0)(3,0)	Palette number for (4,0)(5,0)	Palette number for (30,0)(31,0)
0010	Palette number for (32,0)(33,0)
<hr/>																
07F0	Palette number for (70,55)(71,55)

2.3.4 Game Name Sorting Rules

The game name (sort key) in the information fork is used to determine the order in which icons are displayed on the Dreamcast File Management Screen. Therefore, it is necessary to create a game naming scheme that will cause titles of games that are a series of sequels to be displayed in order.

Using the game name character code table, set 16-byte game names. This section will explain the rules for assigning and sorting game names with a variety of examples.

Only the first 16 characters of an application name are read.

Replace the hyphen-like symbol that indicates a drawn-out vowel in katakana with a “small” vowel-sound katakana.

Use [KATAKANA “HA”] for [HIRAGANA “HA”] even when it is read as “wa”.

Use [KATAKANA “KA”] in [USE THE JAPANESE AS IS].

____10____

Use katakana for English titles.

Star

Symbols that indicate a space, such as “[DOT]” or “-”, are ignored.

Fill this field with 16 bytes of “00” for applications that are intended for use outside of Japan.

If the official name exceeds 16 characters and an abbreviation is used in the GUI comment, use that abbreviation.

When the official title is _____1__ and the GUI comment is _____1_:

_____1_

When English letters are used in an abbreviation, use the typical reading.

_____V.R._

Use numbers at the very end.

_____3_

_____3_

The following type of use of numbers is prohibited.

_3_____

_3_____

If the number of a title in a series comes at the beginning of the title, put the number at the end.

_1_____V.R._

_2_____V.R. _____

_____1_

_____2_

If an earlier title in a series is abbreviated, use the same abbreviation in later titles.

If an earlier title was abbreviated as _J_____ (official name: J_____), use _J_____ for subsequent titles in that series.

Use _____ and _____2_.

If a year number comes at the end of a name, consider what will happen if the year “2000” is used:

Incorrect example: The following titles will be sorted out of order as “2000”, “98”, and then “99”.

_____98_
_____99_
_____2000_

Correct example: The following titles will be sorted correctly.

_____1_
_____2_
_____3_

If the number of titles in a series is likely to exceed “10,” suppress “0”.

**** SHOULD THIS BE “DO NOT SUPPRESS “0”? ****

_____3_
_____12_

_____03_
_____12_

When using different secondary titles after the main game name to distinguish titles in a series of sequels, we recommend using numbers after the main game name in this field so that the titles will be properly sorted in order.

_____1_
_____2_

Once game names have been determined by following these rules, use the following Game Name Character Code Table to create the 16-byte data for the game name (sort key) field.

Character	Code		Character	Code		Character	Code		Character	Code	
	Hexadecimal	Decimal		Hexadecimal	Decimal		Hexadecimal	Decimal		Hexadecimal	Decimal
NULL	00	0	シ	18	24	ハ	30	48	ヨ	48	72
ア	01	1	ジ	19	25	バ	31	49	ヨ	49	73
ァ	02	2	ス	1A	26	パ	32	50	ラ	4A	74
イ	03	3	ズ	1B	27	ヒ	33	51	リ	4B	75
ィ	04	4	セ	1C	28	ピ	34	52	ル	4C	76
ウ	05	5	ゼ	1D	29	ピ	35	53	レ	4D	77
ヴ	06	6	ソ	1E	30	フ	36	54	ロ	4E	78
ゥ	07	7	ゾ	1F	31	ブ	37	55	ワ	4F	79
エ	08	8	タ	20	32	プ	38	56	ヰ	50	80
ェ	09	9	ダ	21	33	ヘ	39	57	ヱ	51	81
オ	0A	10	チ	22	34	ベ	3A	58	ヲ	52	82
ォ	0B	11	ヂ	23	35	ベ	3B	59	ン	53	83
カ	0C	12	ツ	24	36	ホ	3C	60	0	54	84
ガ	0D	13	ヅ	25	37	ボ	3D	61	1	55	85
キ	0E	14	ッ	26	38	ボ	3E	62	2	56	86
ギ	0F	15	テ	27	39	マ	3F	63	3	57	87
ク	10	16	デ	28	40	ミ	40	64	4	58	88
グ	11	17	ト	29	41	ム	41	65	5	59	89
ケ	12	18	ド	2A	42	メ	42	66	6	5A	90
ゲ	13	19	ナ	2B	43	モ	43	67	7	5B	91
コ	14	20	ニ	2C	44	ヤ	44	68	8	5C	92
ゴ	15	21	ヌ	2D	45	ヤ	45	69	9	5D	93
サ	16	22	ネ	2E	46	ユ	46	70			
ザ	17	23	ノ	2F	47	ユ	47	71			

Game Name Character Code Table

This chapter explains how to use the Memory Card Utility program, which is used to transfer files between a PC and the Dev.Box, and between the Dev.Box and Visual Memory.

3.1 Memory Card Utility Preparation and Startup

The Memory Card Utility is a Dreamcast program. A file called "mem_util.elf" is provided in the "utility" folder in the folder where the Visual Memory SDK was installed. Executing this program requires a program that is included in the Dreamcast SDK, such as CodeScape or GD Workshop.

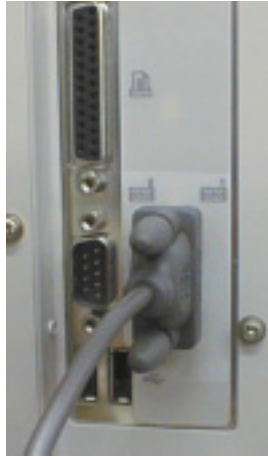
3.1.1 Requirements for Transfer

The Memory Card Utility exchanges files with a PC through an RS-232C serial interface. Therefore, the following items are required in addition to the SDK provided by Sega:

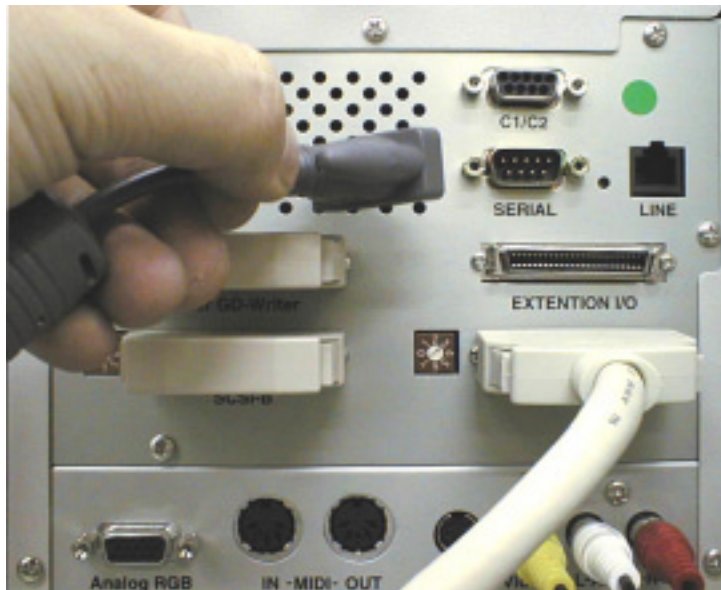
- 1) An RS-232C reverse cable
- 2) A communications program that runs under Windows

Item 1) can be purchased at most computer stores. The cable should be connected between the RS-232C interface on the PC and the connector labeled "SERIAL" on the back of the Dev.Box.

RS-232C Serial Interface Connector on the Computer



SERIAL Connector on the Dev.Box



Caution

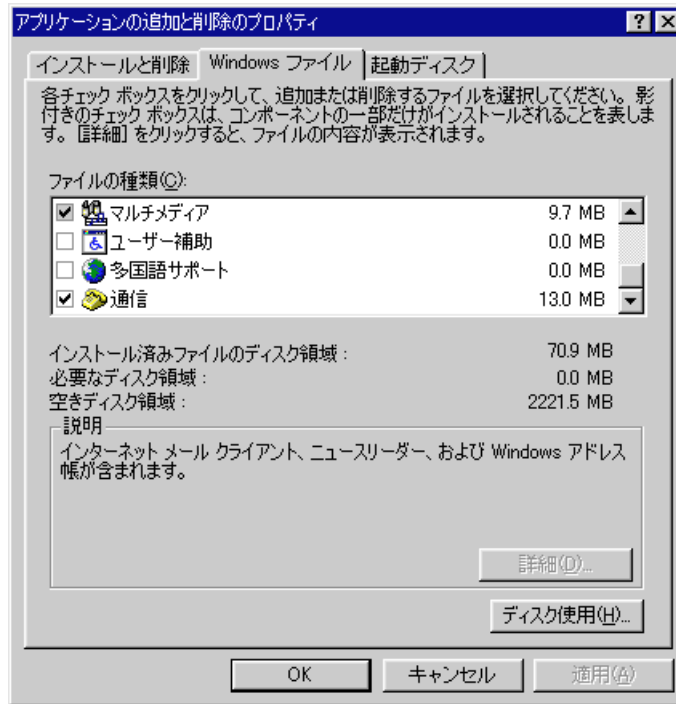
Be certain to use a reverse cable. A straight cable will not work.

"HyperTerminal," which is provided with Windows, suffices for item 2). If you use other communications software, it must support file transfers using the Xmodem protocol.

If you do not have HyperTerminal in your Windows environment, you can install it by following this procedure:

1. In the "Control Panel" window, double-click on the "Add/Remove Programs" icon.
2. Click the "Windows Setup" tab.

3. Double click "Communications".



4. Check the box next to "HyperTerminal."

HyperTerminal will now be installed in your system.

Note

The version of HyperTerminal that is provided with Windows 98 may not be able to recognize path and file names correctly if they contain Kanji characters, or a file with corrupted characters may be stored in the next higher folder. In this case, we recommend either not using file names that contain Kanji characters, or else using another communications program.

Also prepare the following items, which are included in Sega's SDK:

- 3) Dev.Box (Set 5.2X or higher)
- 4) CodeScape
- 5) GD Workshop
- 6) Visual Memory
- 7) Dreamcast controller

For details on connections and setup, refer to the "Setup Guide." The explanations in this manual will assume that setup has been completed properly.

Caution

The Memory Card Utility will not run on a Set 5.1X or earlier Dev.Box.

3.1.2 Software Preparation

Once the PC has been connected to the Dev.Box through the RS-232C interface and software setup has been completed, it is necessary to set the properties, etc.

Communications Protocol Setup

This setting specifies which communications profile is to be used for transferring data between the PC and the Dev.Box.

Make this setting in the application that you will be using. Refer to the manual for that application for details on making that setting.

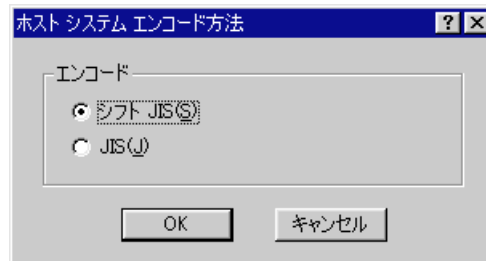
Caution

When the Memory Card Utility is started up, a startup message is displayed on the communications screen. If this message is not displayed correctly, recheck the communications protocol and make sure that the interface to which the cross cable is connected is selected.

Setting Item	Setting
Communications speed	38,400bps
Data length	8 bits
Stop bit length	1 bit
Parity checking	None
Flow control	None
Kanji codes	Shift JIS codes

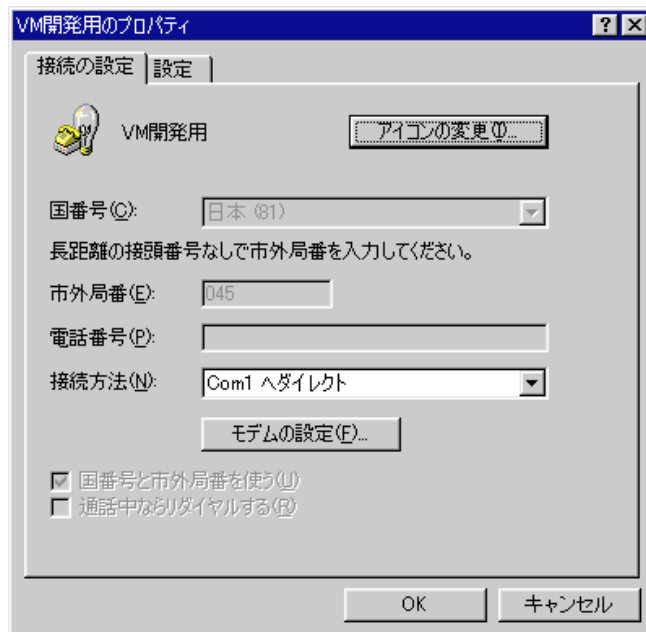
When using HyperTerminal, set the dialog boxes as shown below.



**Caution**

Flow control must be set to "None." Do not specify "Xon/off" or hardware control.

If there are multiple RS-232C interfaces, select the interface to which the cross cable is connected.



GD Workshop Setup

At startup, the Memory Card Utility detects the door on the GD-ROM drive being closed (i.e., that a GD-ROM is mounted). Therefore, it is necessary to create a dummy GD-ROM to emulate the door being closed.

Reference

For details on the operation of GD Workshop, refer to the "GD Workshop Manual."

Create a dummy GD-ROM according to the procedure described below.

1. Start up GD Workshop.
2. Create a new project. For this example, we will create a project named "DoorClose."
3. Drag files suitable for three tracks.
 Since emulation is not possible if the file is too small, copy an execution file for a large application, etc.
 For example, drag the warning sound file that is supplied with the Dreamcast SDK to an audio track.
4. Save the project.

Confirm that the dummy GD-ROM was created properly.

5. Start up DA Checker and restart the Dev.Box in OS mode.
6. Start up GD Workshop, and load the project named "DoorClose."
7. Select "Sound" from the Dreamcast main menu.
8. In GD Workshop, change the GD-ROM to emulation mode.
9. Press the "Door Close/Open" button to close the GD-ROM door and mount the GD-ROM.

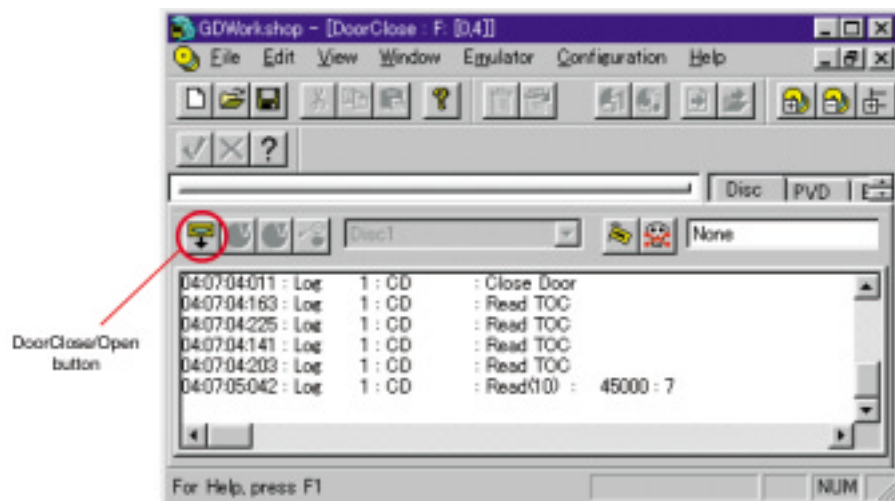
If the dummy GD-ROM has been created properly, the CD-ROM graphic will appear in the Sound menu. If the "CD-ROM" is played back, the warning sound should be heard.

In the future, dummy GD-ROM emulation can be initiated simply by loading the project.

3.1.3 Memory Card Utility Startup

This section describes how to execute the Memory Card Utility using CodeScope.

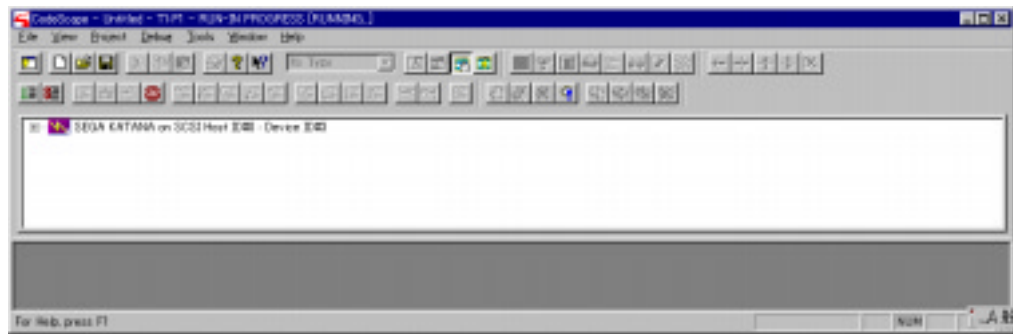
1. Before starting up the Memory Card Utility, start up GD Workshop.
Once GD Workshop has been started up, load the "DoorClose" project and start initialization. Press the "Door Close/Open" button to mount the GD-ROM.



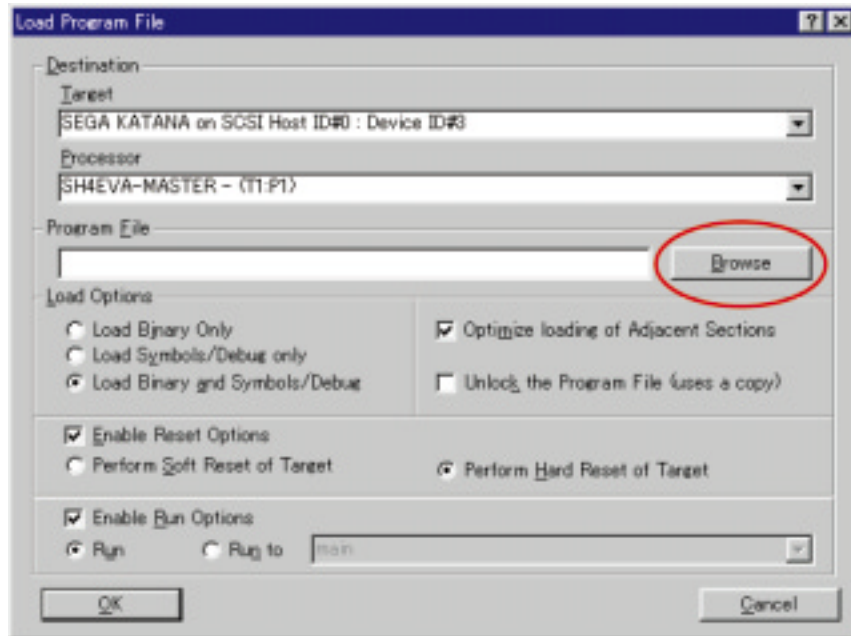
2. Start up the communications software. Confirm the settings for the communications protocol, the interface number, etc.



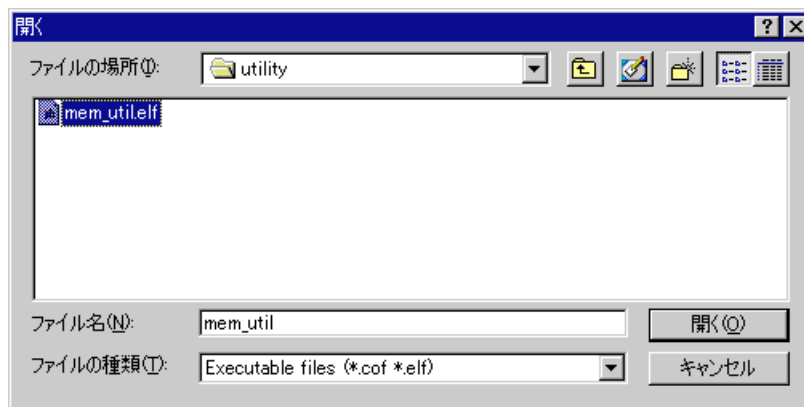
3. Start up CodeScope.



4. Select "Load Program File..." from the "File" menu.

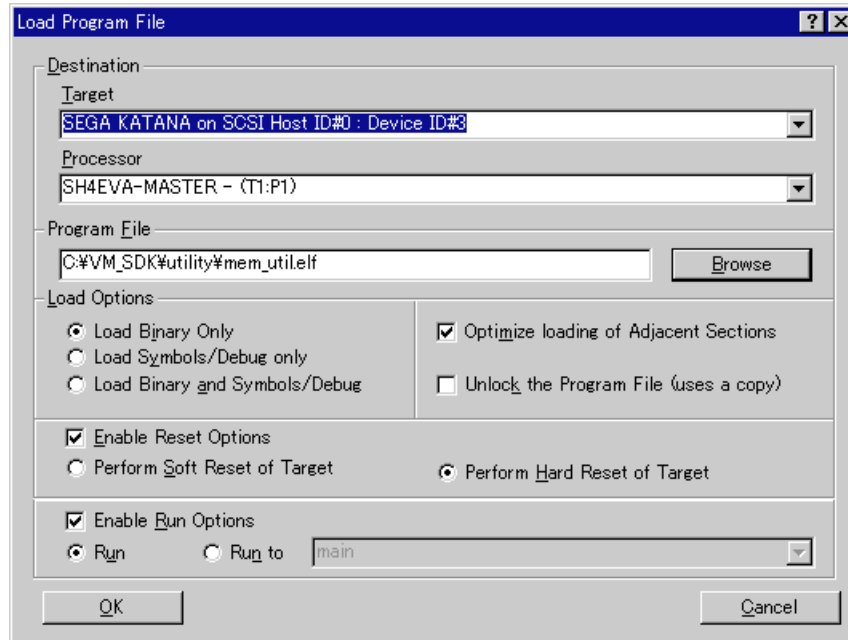


5. Click the [Browse] button and select "mem.util.elf".
The file "mem.util.elf" in the "utility" folder in the folder where the Visual Memory SDK was installed.

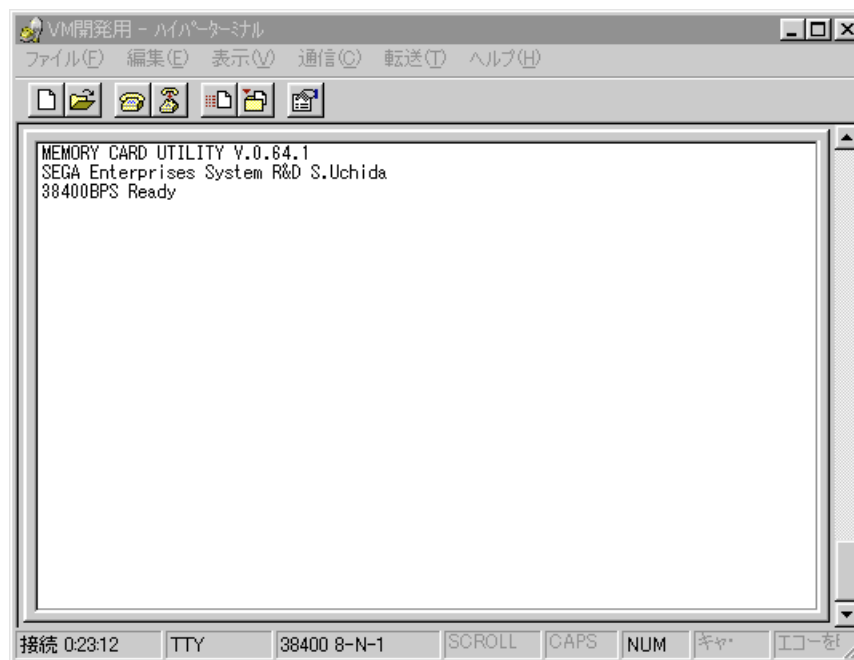


6. Select the "Load Binary Only" option.
7. Check the "Enable Reset Options" box and select the "Perform Hard Reset of Target" option.
8. Check the "Enable Run Options" box and select the "Run" option.

- Once the dialog box settings have been made, click the [OK] button.



- A graph bar is displayed; when it reaches 100%, the Memory Card Utility starts up. At this point, the following startup message is displayed in the communications software window:



If the message does not appear or if it is corrupted, recheck the communications protocol settings.

The main menu of the Memory Card Utility appears on the Dev.Box display (an NTSC or VGA monitor).

**Note**

If "Session Save" is selected from the "File" menu at this point, the Memory Card utility can be started up next time simply by opening the session.

11. Except for file transfers between a PC and the Dev.Box, subsequent Memory Card Utilities are performed by the Dreamcast controller.

3.2 Memory Card Utility Operation

This section explains how to operate the Memory Card Utility. Note that the manual is based on version 0.64.1.

Operations are performed by using the direction button to select a menu item and then pressing the A button to enter that selection. Pressing the B button cancels the operation and returns you to the previous menu. On some menus, the functions of the buttons sometimes differ, or other buttons are used. In those cases, the functions of the buttons will be explained for each menu.

3.2.1 Main Menu

The following screen appears when the Memory Card Utility is started up. This is called the "Main Menu."



Menu items that are grayed out are items that have not been implemented and cannot be selected.

BACKUP UTILITY

If this menu item is selected, the Memory Selection Menu is displayed.

This menu is normally selected when transferring a file to Visual Memory, or to work with a file that has already been saved.

SYSTEM CONFIG

This menu item is used to set Visual Memory's internal clock according to the Dev.Box's internal clock.

If this menu item is selected, the System Configuration Menu appears. If "MEMORY CARD TIME" is selected, a confirmation message appears and the clocks in all of the Visual Memory units that are connected are set according to the Dev.Box clock.

Note

Only "MEMORY CARD TIME" can be selected in the System Configuration Menu.

EXIT

This menu item terminates the Memory Card Utility and passes control to boot ROM. The Dreamcast startup menu is displayed.

3.2.2 Memory Selection Menu

If "BACKUP UTILITY" is selected from the Main Menu, the Memory Selection Menu is displayed.

Caution

Visual Memory can be inserted or removed until this screen. When inserting or removing Visual Memory, set the operation mode to a mode other than game mode. If Visual Memory is connected while it is in game mode, the connection status will not be recognized correctly.



This menu is used to select the memory unit that is to be the object of subsequent operations.

All of the memory units that are currently connected are displayed on this screen. "A," "B," "C," and "D" represent the controller ports. Upper and lower tiers are displayed when multiple memory units are connected to one controller.

Note that in the case of a Visual Memory unit, a number that corresponds to the screen is displayed on the LCD.

After selecting the desired memory unit, press the A button. The Command Selection Screen now appears.

USED

Displays only the number of blocks already in use in the data storage area.

FREE

Displays the number of free blocks in the data storage area.

GAME

Indicates the usage status of the Visual Memory application area, in the format:

«number of blocks in use»/«maximum number of blocks that can be used by an application»

For example, a display of "128/128" indicates that an application has already been written in the Visual Memory unit. A display of "0/128" indicates that no application has been written in the Visual Memory unit.

3.2.3 Command Selection Menu

This menu is used to transfer data to Visual Memory and to manipulate files that have been written in memory.

The right side of the screen shows a list of files written in Visual Memory. If the entire list cannot be displayed on one screen, make the file list active (move the ▲ mark to the file list) and then press the right-hand direction button to display the rest of the list.



INFO

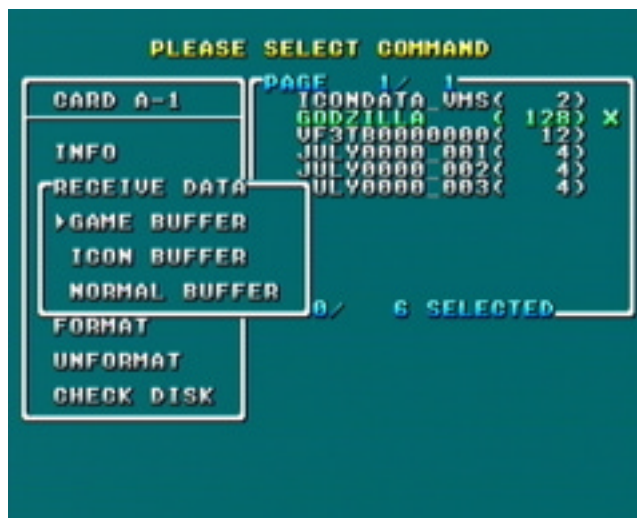
If this menu item is selected, the following screen appears:



This screen displays information concerning the capacity of Visual Memory, body color information, volume icon information, and information on the date and time of initialization.

RECV DATA

This menu item is used to transfer applications, volume icon files, and data files from a PC to a Dev.Box. If "RECV DATA" is selected, the following screen appears:



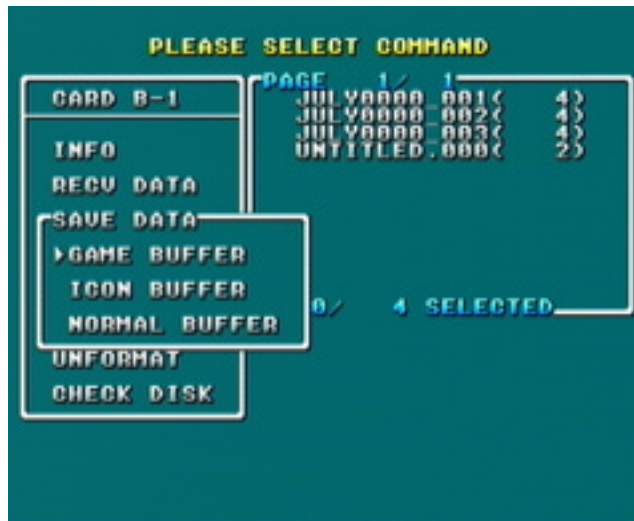
On this screen, specify the type of file that is to be received from the PC. when receiving a Visual Memory application, select "GAME BUFFER;" when receiving volume icon data, select "ICON BUFFER." When receiving a data file, select "NORMAL BUFFER."

Reference

For details on data transfers from a PC to Visual Memory, refer to section 3.4, "File Transfers from a PC to Visual Memory."

SAVE DATA

This menu writes data that was previously received in the Dev.Box buffer into Visual Memory. If SAVE DATA is selected, the following screen appears.



On this screen, specify which file (buffer data) to write in Visual Memory.

Caution

"GAME BUFFER" cannot be selected for Visual Memory in which an application has already been written. Delete the application file first.

To write a Visual Memory application, select "GAME BUFFER;" to write volume icon data, select "ICON BUFFER." To write a data file, select "NORMAL BUFFER."

Reference

For details on data transfers from a PC to Visual Memory, refer to section 3.4, "File Transfers from a PC to Visual Memory."

DUPLICATE

This menu item makes an exact copy of the contents of the currently selected Visual Memory unit in another Visual Memory unit. This function is equivalent to the disk copy functions in Windows and MS-DOS.

If this menu item is selected, the following screen appears:



Once the destination Visual Memory unit has been selected, a confirmation message is displayed.

Caution

The DUPLICATE function copies the contents of flash memory exactly. All data previously saved in the destination Visual Memory unit will be lost.

When you select "OK," the contents of memory are copied exactly.

DEFRAG

Selecting this menu item eliminates fragments (defrags) that develop when files are repeatedly saved and deleted.

Caution

This menu item cannot be selected for a Visual Memory unit in which an application has been written.

Because the FAT system is used for file management in Visual Memory, the data storage area becomes fragmented as files of different sizes are written and deleted over the course of time. This function reorganizes the data so that the fragments are eliminated.

Note

Because applications have to be allocated in a continuous area in memory, execute the DEFRAG function before transferring an application into Visual Memory.

If an application cannot be stored even though there is sufficient space, it is likely that fragmentation is the culprit. Execute the DEFRAG function to create continuous free space.

If "DEFRAG" is selected, a confirmation message appears and then the defragmentation processing is performed.

FORMAT

If this menu item is selected, the following screen appears and the selected Visual Memory unit can be initialized.



This screen is used to specify the body color and label icon, to set the date and time of initialization, etc. After making all of the necessary settings, a confirmation message appears; selecting "OK" causes the Visual Memory unit to be initialized. All files stored in the Visual Memory unit that is being initialized will be lost.

Reference

For details on the initialization procedure, refer to section 3.3, "Initializing Visual Memory."

UNFORMAT

This menu item can be selected in order to create an uninitialized visual memory unit. The system BIOS determines whether a Visual Memory unit has been initialized or not by checking a management area (an upper address in bank 1 in flash memory), such as the FAT.

Just as data cannot be written on a floppy disk that has not been initialized, data and applications cannot be stored in a Visual Memory unit that has not been initialized.

Caution

Visual Memory that is purchased commercially is shipped in an initialized state. Rarely, a lot might be shipped in an uninitialized state.

If "UNFORMAT" is selected, the following screen appears:



If "COMPLETE" is executed, 00H is written to every address in the flash memory to put it into the uninitialized state. If "QUICK" is selected, just the management area is cleared.

Caution

The "QUICK" function is not implemented in Memory Card Utility Version 0.64.1 and therefore cannot be selected.

Whether "COMPLETE" or "QUICK" is used, all data stored in Visual Memory is deleted. Even if "QUICK" is selected, there is no means for salvaging the data after initialization.

CHECK DISK

If this menu item is executed, the following screen appears and a check is made of FAT conformance, missing data bits, etc. This function is equivalent to the scan disk function in Windows and the CHKDSK function in MS-DOS.

However, this function does not have an error correction capability.

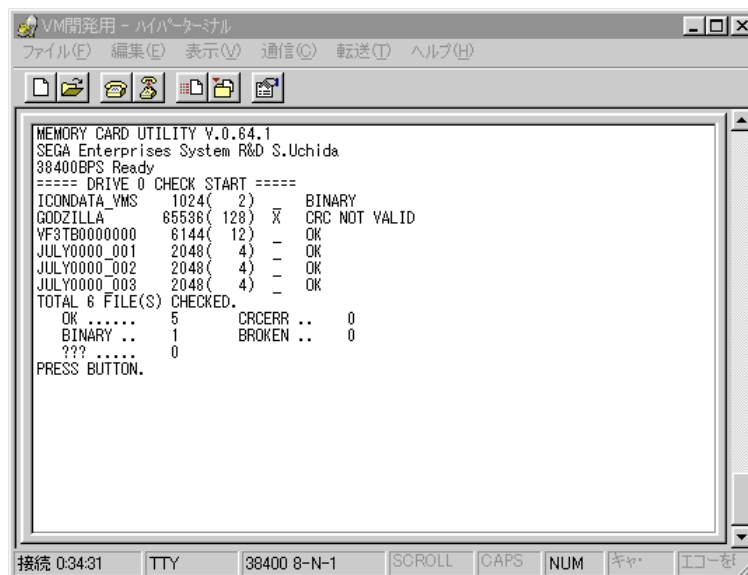
This function also conducts a CRC check of the information fork of each file, and checks for missing bits in files.

Note

If the CRC value was omitted from an information fork, this check returns an error for that information fork, but this error can be ignored.

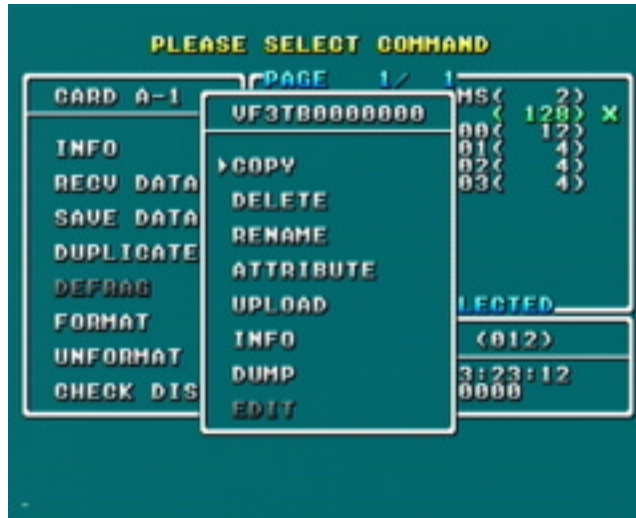


The results of these checks are also displayed in the communications software window as shown below.



3.2.4 File Operations Menu

If the ▲ mark is moved on the file list, a file is selected, and the A button is pressed, the File Operations Menu is displayed.



Multiple files can be selected by pressing the X button after selecting each file. If the A button is pressed while multiple files are selected, the menu operation that is performed is performed on all selected files.

If the Y button is pressed, all files that are currently selected are deselected, and all files that are currently not selected are selected.

COPY

If "COPY" is selected, the following screen appears, and the selected file is copied to a different Visual Memory unit.



DELETE

This deletes the selected file. A confirmation message is displayed when "DELETE" is selected.

RENAME

This changes a file name. This item cannot be selected when multiple files are selected. If "RENAME" is selected, the following screen appears; the file name can now be changed.



Caution

The Memory Card Utility is designed to permit the "-" character to be used in file names, but do not use the character.

Although it poses no problem for the debugger, do not use the "-" character in file names in final products. Such an application will not be in conformance with the software creation standards.

Use "←" and "→" to move the cursor. Select "OK" to change the file name to the new file name that was input. "CANCEL" is equivalent to the B button. If "RESET" is selected, the file name that was being input is cleared and the original file name is restored.

On this screen, the R trigger and the L trigger can be used to move the cursor, and pressing the Start button has the same effect as selecting "OK."

ATTRIBUTE

If "ATTRIBUTE" is selected, the following screen appears and the file attribute (copying prohibited/permitted) can be changed.



If the COPY FLAG is set to "FF," copying that file is prohibited. A file for which copying is prohibited can not be copied through the Dreamcast File Management Screen. If COPY FLAG is set to any other value (00 to FE), copying that file is permitted. Because any value other than FF can be used, a program can be created that uses this field to determine what generation of copying a given file is.

Caution

Note that when a file for which the COPY FLAG is set to any value from 00 to FE is copied through the Dreamcast File Management Screen, the COPY FLAG in the newly copied file is set to "00."

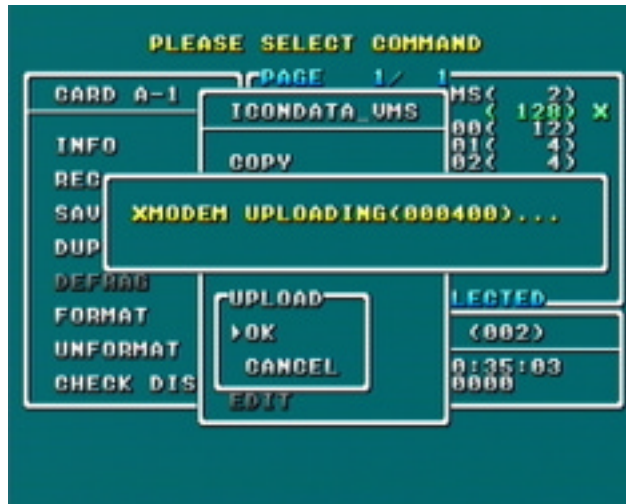
HEADER OFFSET cannot be changed.

UPLOAD

This can be used to transfer a selected file to a PC.

Before selecting "UPLOAD," execute an Xmodem download using the file transfer function of the communications software. As long as the Xmodem protocol is used, it does not matter if it is CRC or 1024. When the file name input screen appears, enter an appropriate file name. The file name that is input does not have to be identical to the file name that is used in Visual Memory.

When "UPLOAD" is selected, the message "NOW LOADING..." appears and the file transfer to the PC begins.



Caution

"UPLOAD" cannot be aborted. To abort, first complete the file transfer that is in progress.

"UPLOAD" cannot be selected while multiple files are selected.

INFO

If "INFO" is selected, the message "NOW LOADING..." is displayed, and then the following screen appears.



Because the game name (sort key) is also displayed, this screen can be used to check the game name after it has been input.

If "DUMP" is selected, the message "NOW LOADING..." is displayed, and then the file dump screen appears.



Caution	DUMP cannot be selected when multiple files are selected. Kanji cannot be displayed in the character dump.
----------------	---

Up button	Scrolls towards the beginning of the file.
Down button	Scrolls towards the end of the file.
Left button	Scrolls rapidly towards the beginning of the file.
Right button	Scrolls rapidly towards the end of the file.
L trigger	Displays the beginning of the file.
R trigger	Displays the end of the file.
X button	Each time this button is pressed, the display delimiting unit switches between BYTE, WORD (2 bytes), and DWORD (4 bytes).
Start button	Halts the file dump.

This is for future expansion. This menu item cannot be selected because the editing function is not implemented in Ver. 0.64.1.

3.3 Initializing Visual Memory

This section describes the procedure for initializing Visual Memory.

1. Select the Visual Memory unit that is to be initialized, and then display the Command Selection Menu.



2. Select "FORMAT," and the following screen appears.



3. In the "ICON NO." field, specify the label icon number. Specifying "00" specifies the default Visual Memory icon. Specify a value from 000 to 123. Do not set a value of 124 or higher.

Reference

For a list of the label icon designs and numbers, refer to the Appendix, "List of Label Icons."

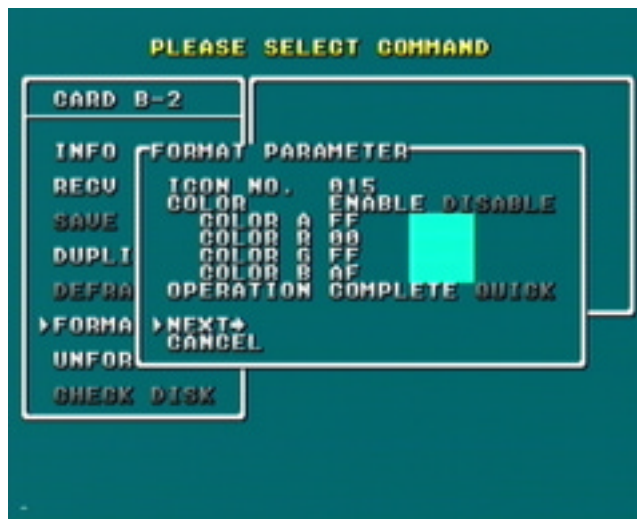
4. Set whether the body color information (which is set next) is valid or not.
If the body color information is valid, select "ENABLE." If the body color information is invalid, select "DISABLE." If "DISABLE" is selected, the body color is white and the color information setting becomes unavailable.
5. Set the color information. "COLOR A" specifies the transparency. A value of "FF" is completely opaque, and a value of "00" is completely transparent.

COLOR R, G, and B specify the intensity of the red, green, and blue components. A value of "FF" is the maximum intensity, and a value of "00" is the minimum intensity.

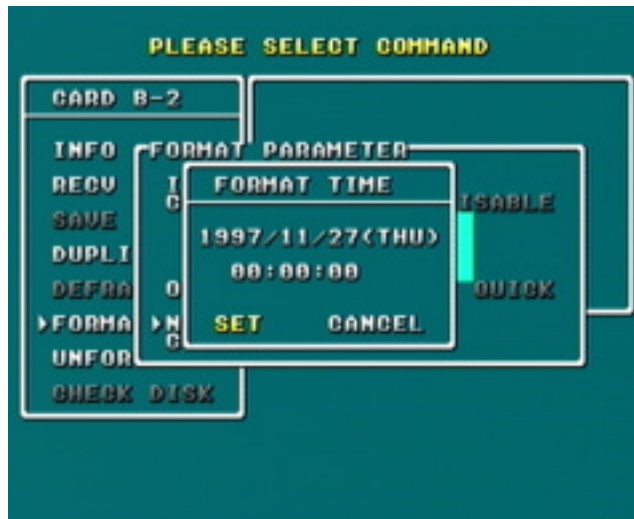
Caution

Note that if a value of "00" is set for COLOR A, the other colors will be transparent.

6. "OPERATION" specifies whether to initialize just the FAT, or to initialize all of memory.
Select "QUICK" to initialize just the FAT, and select "COMPLETE" to initialize all of memory.
7. Select "NEXT" and the following screen appears.



8. Set the date and time of initialization. Use the left and right buttons to set the date and time, and then use the up and down buttons to change the value.



9. Lastly, select "SET" and a confirmation message appears. Select "OK" to begin the initialization process.

Caution

If a Visual Memory unit is initialized, all files that were written in that unit are deleted. Even if "QUICK" is selected for "OPERATION," there is no means for salvaging the data after initialization.

3.4 Transferring Files from a PC to Visual Memory

This section explains the procedure for transferring files from a PC to Visual Memory. In this example, HyperTerminal, which is provided with Windows, will be used as the communications software.

1. Select the Visual Memory unit to which the file is to be transferred, and then display the Command Selection Menu.



2. Select "RECV DATA," and then select the buffer for the file that is to be transferred.

When transferring an application, select "GAME BUFFER."

When transferring a volume icon ("ICONDATA_VMS"), select "ICON BUFFER."

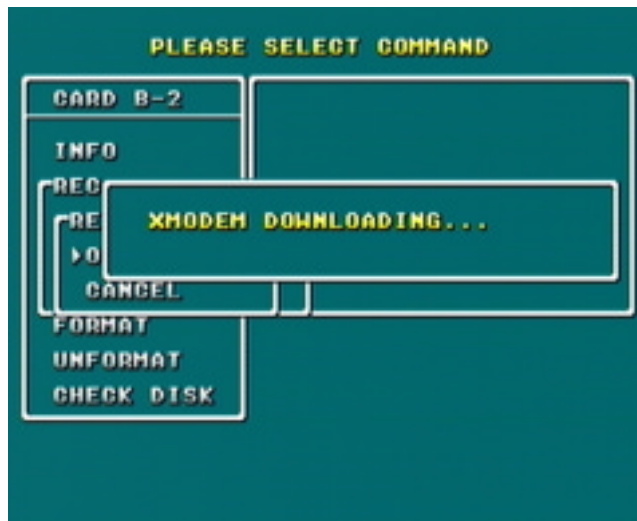
When transferring a data file, select "NORMAL BUFFER."



3. When the confirmation message is displayed, select "OK."



In the Dev.Box screen



7. When the file transfer is completed, the following screen appears on the Dev.Box side. Press the A button.

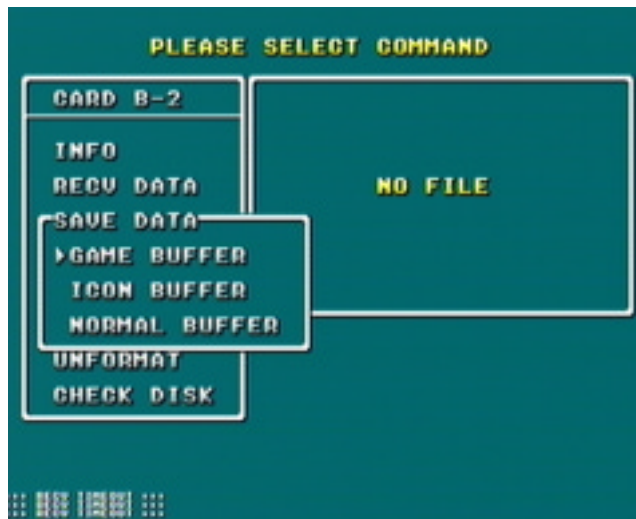


8. Select "SAVE DATA" and specify which buffer's contents to write in Visual Memory.

When writing an application, select "GAME BUFFER."

When writing a volume icon ("ICONDATA_VMS"), select "ICON BUFFER."

When writing a data file, select "NORMAL BUFFER."



Caution

If an application has already been written in the Visual Memory unit, "GAME BUFFER" cannot be selected. Delete the old file and then select "SAVE DATA" again.

9. If a buffer other than "ICON BUFFER" was selected, the following screen appears.



Input the file name and select "OK." The file name does not have to be in "8.3" format (an eight-character file name, a period, and a three-character extension).

If a file with the same name already exists, the program asks whether or not to overwrite the old file.

The PC-to-Visual Memory file transfer process is now complete.

If an application was transferred, return to the Memory Selection Screen, disconnect the Visual Memory unit, set the Visual Memory unit to game mode, and then execute the application.

Appendix A

Little Endian Format

When storing multiple bytes of data in memory, some CPUs use a format that starts from the high-order byte and stores it in the high-order byte in memory, while other CPUs use a format that starts from the high-order byte and stores it in the low-order byte in memory.

For example, when storing the data "00 FE 2E EF" in memory as an "unsigned long int" value (an unsigned 32-bit integer), the following two methods could be used:

Big Endian Format

	+00	+01	+02	+03
0000	00	FE	2E	EF

Little Endian Format

	+00	+01	+02	+03
0000	EF	2E	FE	00

This format, in which the upper and lower bytes of data are stored in reverse order is called "Little Endian Format." The format in which data is stored in its normal order is called "Big Endian Format."

Because the SH4 CPU that is inside the Dreamcast uses Little Endian format, all data other than byte data must be stored in memory in Little Endian format.

For example, because the value "00 00 00 20" must be specified for the "monochrome icon data starting address" in ICONDATA_VMS", the value is stored in memory in the order "20 00 00 00". In addition, when storing unsigned 16-bit data ("00 FF") in memory, it should be stored in the order "FF 00"

Appendix B

List of Label Icons

The label icons that are built into the Dreamcast boot ROM are listed below.

Icon	Icon number	
	Hexadecimal	Decimal
	00	0
	01	1
	02	2
	03	3
	04	4
	05	5
	06	6
	07	7
	08	8
	09	9
	0A	10
	0B	11
	0C	12
	0D	13
	0E	14
	0F	15
	10	16
	11	17
	12	18
	13	19
	14	20
	15	21
	16	22
	17	23
	18	24
	19	25
	1A	26
	1B	27
	1C	28
	1D	29
	1E	30

Icon	Icon number	
	Hexadecimal	Decimal
	1F	31
	20	32
	21	33
	22	34
	23	35
	24	36
	25	37
	26	38
	27	39
	28	40
	29	41
	2A	42
	2B	43
	2C	44
	2D	45
	2E	46
	2F	47
	30	48
	31	49
	32	50
	33	51
	34	52
	35	53
	36	54
	37	55
	38	56
	39	57
	3A	58
	3B	59
	3C	60
	3D	61

Icon	Icon number	
	Hexadecimal	Decimal
	3E	62
	3F	63
	40	64
	41	65
	42	66
	43	67
	44	68
	45	69
	46	70
	47	71
	48	72
	49	73
	4A	74
	4B	75
	4C	76
	4D	77
	4E	78
	4F	79
	50	80
	51	81
	52	82
	53	83
	54	84
	55	85
	56	86
	57	87
	58	88
	59	89
	5A	90
	5B	91
	5C	92

Icon	Icon number	
	Hexadecimal	Decimal
	5D	93
	5E	94
	5F	95
	60	96
	61	97
	62	98
	63	99
	64	100
	65	101
	66	102
	67	103
	68	104
	69	105
	6A	106
	6B	107
	6C	108
	6D	109
	6E	110
	6F	111
	70	112
	71	113
	72	114
	73	115
	74	116
	75	117
	76	118
	77	119
	78	120
	79	121
	7A	122
	7B	123

Appendix C

Sample Program Listings

This section includes listings of the sample programs that are included with the Visual Memory SDK.

Explanations of the information from source file “IFORK.ASM” and details of “GHEAD.ASM” are not included.

Caution

When viewing the sample programs, set the tab (09H) to "4" (byte).

C.1 LCD Pattern Display

This sample program displays a simple pattern on the LCD (XRAM).

Lines 51, 55, 59, 63, and 67 specify the pattern, and the “matrix” routine draws this pattern on the LCD.

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** LCD display processing sample 1 **
005 ;
006 ; Transfers data to display RAM and displays a simple pattern on the display
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      Lc868700          ; Specifies the chip type for the assembler
012 world     external         ; External memory program
013
014 public    main              ; Symbol referenced from ghead.asm
015
016 extern    _game_end         ; Application end
017
018
019 ; **** Definition of System Constants ****
020
021                                ; OCR (Oscillation Control Register) settings
022 osc_rc      equ 081h         ; Specifies internal RC oscillation for the System clock
023 osc_xt      equ 082h         ; Specifies crystal oscillation for the system Clock
024
025
026 ; *** Data Segment ****
027
028          dseg                ; Data segment start
029
030 r0:       ds      1          ; Indirect addressing register r0
031 r1:       ds      1          ; Indirect addressing register r1
032 r2:       ds      1          ; Indirect addressing register r2
033 r3:       ds      1          ; Indirect addressing register r3
034          ds      12          ; Other registers reserved for the system
035
```

```
036
037 ; *** Code Segment *****
038
039         cseg                ; Code segment start
040
041 ; *-----*
042 ; * User program
043 ; *-----*
044 main:
045     mov     #0f0h,c         ; Display data
046     call    matrix         ; Display pattern on the LCD
047     set1    PCON,0         ; Enters HALT mode and waits for an interrupt.
048                                     ; HALT mode is cancelled and processing continues
049                                     ; when a base timer interrupt is generated.
050
051     mov     #00fh,c         ; The following lines display different patterns in the same manner
052     call    matrix
053     set1    PCON,0
054
055     mov     #0cch,c
056     call    matrix
057     set1    PCON,0
058
059     mov     #033h,c
060     call    matrix
061     set1    PCON,0
062
063     mov     #055h,c
064     call    matrix
065     set1    PCON,0
066
067     mov     #0aah,c
068     call    matrix
069     set1    PCON,0
070
071                                     ; ** [M] (mode) Button Check **
072     ld      P3
073     bn      acc,6,finish    ; If the [M] button is pressed, the application ends
074
075     jmp     main            ; Repeat
076
077 finish:
078     jmp     _game_end      ; ** Application End Processing **
079                                     ; Application end
080
081 ; *-----*
082 ; * Displays pattern on entire LCD
083 ; * Input c: Basic display pattern
084 ; *-----*
085 matrix:
086                                     ; **** Draws one LCD screen ****
087     push    acc             ; Pushes each register onto the stack
088     push    b
089     push    c
090     push    XBNK
091
092     xb0_a:  mov     #000h,XBNK    ; Specifies the display RAM bank address (BANK0)
093             mov     #080h,b
094
095     la1:    ld      c             ; c: Display data
096             call    line2         ; 2-line display
097             ld      b             ; Advances address two lines ahead
098             add     #010h         ;
099             st      b             ;
100             bnz     la1          ; Repeats until end of bank is reached
101
102     xb1_a:  mov     #001h,XBNK    ; Specifies the display RAM bank address (BANK1)
103             mov     #080h,b
104
105     la2:    ld      c             ; c: Display data
106             call    line2         ; 2-line display
107             ld      b             ; Advances address two lines ahead
```

```
108      add    #010h    ;
109      st     b         ;
110      bnz    1a2       ; Repeats until end of bank is reached
111
112      pop    XBNK      ; Pops the registers from the stack
113      pop    c         ;
114      pop    b         ;
115      pop    acc       ;
116
117      ret              ; Matrix end
118
119
120 line2:                      ; **** LCD 2-line display ****
121
122      push   acc       ; Pushes each register onto the stack
123      push   b         ;
124      push   c         ;
125      push   PSW       ;
126      push   OCR       ;
127      mov    #osc_rc,OCR ; Specifies the system clock
128      setl   PSW,1     ; Selects data RAM bank 1
129      st     c         ; Stores display data in c
130      ld     b         ; Sets the display RAM address in r2
131      st     r2        ;
132
133 lp1:                      ; **** First line display processing ****
134      ld     c         ; Transfers the display data to display RAM
135      st     @r2       ;
136      inc    r2        ; Advances the address to the next display position
137      ld     r2        ;
138      and    #00fh     ; If the display position is not at the right end of the first line...
139      xor    #006h     ;
140      bnz    lp1       ; ...repeat
141
142      ld     c         ; Inverts the bit pattern in the c register
143      xor    #0ffh     ;
144      st     c         ;
145
146 lp2:                      ; **** Second line display processing ****
147      ld     c         ; Transfers the display data to display RAM
148      st     @r2       ;
149      inc    r2        ; Advances the address to the next display position
150      ld     r2        ;
151      and    #00fh     ; If the display position is not at the right end of the second line...
152      xor    #00ch     ;
153      bnz    lp2       ; ...repeat
154
155      pop    OCR       ; Pops registers off of the stack
156      pop    PSW       ;
157      pop    c         ;
158      pop    b         ;
159      pop    acc       ;
160
161      ret              ; line2 end
162
163      end
```

C.2 LCD Character Pattern Display

This sample program displays the text “SEGA 1998” on the LCD (XRAM).

Because the built-in fonts cannot be used from an application, the font pattern data must be prepared beforehand.

This program calls “putch”, which writes the specified font pattern at the coordinates specified by registers B and C in the main routine. The font information (8 × 8 dot data) starts in line 222.

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** LCD display processing sample 2 **
005 ;
006 ; ·Clears the display image by filling display RAM with zeroes
007 ; ·Displays character pattern in a specified position
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip      Lc868700          ; Specifies the chip type for the assembler
013 world     external         ; External memory program
014
015 public    main              ; Symbol referenced from ghead.asm
016
017 extern    _game_end         ; Application end
018
019
020 ; **** Definition of System Constants ****
021
022                                ; OCR (Oscillation Control Register) settings
023 osc_rc     equ 081h          ; Specifies internal RC oscillation for the system clock
024 osc_xt     equ 082h          ; Specifies crystal oscillation for the system clock
025
026
027 ; *** Data Segment ****
028
029          dseg                ; Data segment start
030
031 r0:       ds      1          ; Indirect addressing register r0
032 r1:       ds      1          ; Indirect addressing register r1
033 r2:       ds      1          ; Indirect addressing register r2
034 r3:       ds      1          ; Indirect addressing register r3
035          ds      12          ; Other registers reserved for the system
036
037
038 ; *** Code Segment ****
039
040          cseg                ; Code segment start
041
042 ; *-----*
043 ; * User program
044 ; *-----*
045 main:
046          call     cls         ; Clears the LCD display image
047
048          mov      #1,c        ; Horizontal coordinate
049          mov      #1,b        ; Vertical coordinate
050          mov      #0ah,acc     ; Character code 'S'
051          call     putch       ; Single character display
052
053          mov      #2,c
054          mov      #1,b
055          mov      #0bh,acc     ; 'E'
056          call     putch
```



```

057
058     mov     #3,c
059     mov     #1,b
060     mov     #0ch,acc    ; 'G'
061     call    putchar
062
063     mov     #4,c
064     mov     #1,b
065     mov     #0dh,acc    ; 'A'
066     call    putchar
067
068     mov     #1,c
069     mov     #2,b
070     mov     #1,acc      ; 'I'
071     call    putchar
072
073     mov     #2,c
074     mov     #2,b
075     mov     #9,acc      ; '9'
076     call    putchar
077
078     mov     #3,c
079     mov     #2,b
080     mov     #9,acc      ; '9'
081     call    putchar
082
083     mov     #4,c
084     mov     #2,b
085     mov     #8,acc      ; '8'
086     call    putchar
087
088 loop0:                                ; ** [M] (mode) Button Check **
089     ld      P3
090     bn      acc,6,finish ; If the [M] button is pressed, the application ends
091
092     jmp     loop0        ; Repeat
093
094 finish:                                ; ** Application End Processing **
095     jmp     _game_end    ; Application end
096
097
098 ; *-----*
099 ; * Clearing the LCD Display Image *
100 ; *-----*
101 cls:
102     push    OCR          ; Pushes the OCR value onto the stack
103     mov     #osc_rc,OCR   ; Specifies the system clock
104
105     mov     #0,XBNK      ; Specifies the display RAM bank address (BANK0)
106     call    cls_s        ; Clears the data in that bank
107
108     mov     #1,XBNK      ; Specifies the display RAM bank address (BANK1)
109     call    cls_s        ; Clears the data in that bank
110     pop     OCR          ; Pops the OCR value off of the stack
111
112     ret                     ; cls end
113
114 cls_s:                                ; *** Clearing One Bank of Display RAM ***
115     mov     #80h,r2      ; Points the indirect addressing register at the start of display RAM
116     mov     #80h,b       ; Sets the number of loops in loop counter b
117 loop3:
118     mov     #0,@r2       ; Writes "0" while incrementing the address
119     inc     r2            ;
120     dbnz    b,loop3      ; Repeats until b is "0"
121
122     ret                     ; cls_s end
123
124
125 ; *-----*
126 ; * Displaying One Character in a Specified Position *
127 ; * Inputs: acc:Character code *
128 ; *           c:   Horizontal position of character *
129 ; *           b:   Vertical position of character *

```

130 ; *-----*

```

131 putch:
132     push    XBNK
133     push    acc
134     call    locate    ; Calculates display RAM address according to coordinates
135     pop     acc
136     call    put_chara  ; Displays one character
137     pop     XBNK
138
139     ret                      ; putch end
140
141
142 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification ****
143     ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
144     ; ** Outputs: r2: RAM address XBNK: Display RAM bank
145
146     ; *** Determining the Display RAM Bank Address ***
147     ld      b              ; Jump to next1 when b >= 2
148     sub     #2              ;
149     bn      PSW,7,next1    ;
150
151     mov     #00h,XBNK      ; Specifies the display RAM bank address (BANK0)
152     br      next2
153 next1:
154     st      b
155     mov     #01h,XBNK      ; Specifies the display RAM bank address (BANK1)
156 next2:
157
158     ; *** Calculating the RAM Address for a Specified Position on the
Display ***
159     ld      b              ; b * 40h + c + 80h
160     rol
161     rol
162     rol
163     rol
164     rol
165     rol
166     add     c
167     add     #80h
168     st      r2              ; Stores the RAM address in r2
169
170     ret                      ; locate end
171
172
173 put_chara:
174     push    PSW            ; Pushes the PSW value onto the stack
175     set1    PSW,1          ; Selects data RAM bank 1
176
177     ; *** Calculating the Character Data Address ***
178     ; (TRH,TRL) = acc*8 + fontdata
179     rol
180     rol
181     add     #low(fontdata) ;
182     st      TRL            ;
183     mov     #0,acc          ;
184     addc    #high(fontdata) ;
185     st      TRH            ;
186
187     push    OCR             ; Pushes the OCR value onto the stack
188     mov     #osc_rc,OCR     ; Specifies the system clock
189
190     mov     #0,b            ; Offset value for loading the character data
191     mov     #4,c            ; Loop counter
192 loop1:
193     ld      b              ; Loads the display data for the first line
194     ldc
195     inc     b              ; Increments the load data offset by 1
196     st      @r2            ; Transfers the display data to display RAM
197     ld      r2              ; Adds 6 to the display RAM address
198     add     #6
199     st      r2
200

```

Visual Memory Tutrial Revision 1.00

```
201      ld      b          ; Loads the display data for the second line
202      ldc          ;
203      inc      b          ; Increments the load data offset by 1
204      st      @r2        ; Transfers the display data to display RAM
205      ld      r2         ; Adds 10 to the display RAM address
206      add      #10       ;
207      st      r2         ;
208
209      dec      c          ; Decrements the loop counter
210      ld      c          ;
211      bnz      loop1     ; Repeats for 8 lines (four times)
212
213      pop      OCR        ; Pops the OCR value off of the stack
214      pop      PSW        ; Pops the PSW value off of the stack
215
216      ret              ; put_chara end
217
218
219 ; *-----*
220 ; * Character Bit Image Data
221 *
222 ; *-----*
222 fontdata:
223      db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h      ; '0' 00
224      db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h      ; '1' 01
225      db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h      ; '2' 02
226      db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h      ; '3' 03
227      db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h      ; '4' 04
228      db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h      ; '5' 05
229      db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h      ; '6' 06
230      db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h      ; '7' 07
231      db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h      ; '8' 08
232      db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h      ; '9' 09
233
234      db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h      ; 'S' 0a
235      db 0feh, 0c0h, 0c0h, 0f8h, 0c0h, 0c0h, 0feh, 000h      ; 'E' 0b
236      db 07ch, 0e6h, 0c0h, 0dch, 0c6h, 0e6h, 07ch, 000h      ; 'G' 0c
237      db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h      ; 'A' 0d
```

C.3 Counter That Uses Base Timer Interrupts

This sample program detects and counts interrupts that are generated by the base timer every 0.5 seconds.

When an interrupt is generated, the program jumps to line 35 of “GHEAD.ASM”, and then jumps from there to the label “INT_1B”. Base timer interrupt processing starts in line 108, but here internal clock processing is performed. After performing the clock processing in ROM once up to line 112, control jumps to the label “int_BaseTimer” in “B_TIMER1.ASM”.

In “B_TIMER1.ASM”, the label “int_BaseTimer”, which is referenced by an external program, is declared with a PUBLIC declaration (line 16). The user's base timer interrupt handler starts from line 242. The counter is incremented within this interrupt handler.

Control returns to “GHEAD.ASM”, the contents of the IE register are returned to the value that it had when “int_1b” was called, and then control returns from the interrupt (IRET).

The counter value is always displayed on the LCD by the main routine.

• GHEAD.ASM

```

001 chip      Lc868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *                               05/20-'98    *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main      ; Symbol in the user program
014 extern  int_BaseTimer  ; Symbol in the user program
015
016 ; *-----*
017 ; * Vector table(?)                      *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023      jmpf    main      ; main program jump
024 org 0003h
025 ;int_03:
026      jmp int_03
027 org 000bh
028 ;int_0b:
029      jmp int_0b
030 org 0013h
031 ;int_13:
032      jmp int_13
033 org 001bh
034 ;int_1b:
035      jmp int_1b
036 org 0023h
037 ;int_23:
038      jmp int_23
039 org 002bh
040 ;int_2b:
041      jmp int_2b
042 org 0033h

```

```
043 ;int_33:
044     jmp int_33
045 org 003bh
046 ;int_3b:
047     jmp int_3b
048 org 0043h
049 ;int_43:
050     jmp int_43
051 org 004bh
052 ;int_4b:
053     jmp int_4b
054 ; *-----*
055 ; * interrupt programs          *
056 ; *-----*
057 int_03:
058     reti
059 int_0b:
060     reti
061 int_13:
062     reti
063 int_23:
064     reti
065 int_2b:
066     reti
067 int_33:
068     reti
069 int_3b:
070     reti
071 ; *-----*
072 int_43:
073     reti
074 int_4b:
075     clr1    p3int,1    ; interrupt flag clear
076     reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program      *
081 ; *-----*
082 fm_wrt_ex:
083     change fm_wrt_in
084     fm_wrt_ex_exit:
085     ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program      *
089 ; *-----*
090 fm_vrf_ex:
091     change fm_vrf_in
092     fm_vrf_ex_exit:
093     ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program      *
098 ; *-----*
099 fm_prd_ex:
100     change fm_prd_in
101     fm_prd_ex_exit:
102     ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*
108 int_1b:
109 timer_ex:
110     push    ie
111     clr1    ie,7    ; interrupt prohibition
112     change timer_in
113     timer_ex_exit:
114     call    int_BaseTimer    ; (User base timer interrupt processing)
115     pop ie
116     reti
117
```

```

118 org 01f0h
119 _game_end:
120     change game_end
121 end

```

• B_TIMER1.ASM

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** Base Timer Interrupt Usage Sample 1 **
005 ;
006 ; ·Counts base timer interrupts (every 0.5 seconds)
007 ; ·Displays the counter value as a two digit decimal number on the LCD
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip    LC868700          ; Specifies the chip type for the assembler
013 world   external         ; External memory program
014
015 public  main              ; Symbol referenced from ghead.asm
016 public  int_BaseTimer     ; Symbol referenced from ghead.asm
017
018 extern  _game_end          ; Symbol reference to ghead.asm
019
020
021 ; **** Definition of System Constants ****
022
023                                ; OCR (Oscillation Control Register) settings
024 osc_rc    equ 081h          ; Specifies internal RC oscillation for the system clock
025 osc_xt    equ 082h          ; Specifies crystal oscillation for the system clock
026
027
028 ; *** Data Segment ****
029
030         dsezz              ; Data segment start
031
032 r0:      ds      1          ; Indirect addressing register r0
033 r1:      ds      1          ; Indirect addressing register r1
034 r2:      ds      1          ; Indirect addressing register r2
035 r3:      ds      1          ; Indirect addressing register r3
036         ds      12         ; Other registers reserved for the system
037
038 counter: ds      1          ; Base timer interrupt counter
039 work1:   ds      1          ; For work (put2digit)
040
041
042 ; *** Code Segment ****
043
044         cseg              ; Code segment start
045
046 ; *-----*
047 ; * User program
048 ; *-----*
049 main:
050     mov     #0,counter      ; Resets the counter value
051
052     call    cls            ; Clears the LCD display image
053
054 loop0:
055     mov     #2,c           ; Display position (horizontal)
056     mov     #1,b           ; Display position (vertical)
057     ld      counter        ; Moves the counter value to acc
058     call    put2digit       ; Displays the acc value (two digits)
059
060     setl    pcon,0         ; Waits in HALT mode until the next interrupt
061
062                                ; ** [M] (mode) Button Check **
063     ld      P3
064     bn      acc,6,finish    ; If the [M] button is pressed, the application ends

```

```
065
066      br      loop0      ; Repeat
067
068 finish:                ; ** Application End Processing **
069      jmp      _game_end  ; Application end
070
071
072 ; *-----*
073 ; * Displaying a Two-digit Value
074 ; *
075 ; * Inputs: acc:      Numeric value
076 ; *
077 ; *          c:Horizontal position of character
078 ; *          b:Vertical position of character
079 ; *-----*
080 put2digit:
081      push     b          ; Pushes the coordinate data onto the stack
082      push     c          ;
083      st       c          ; Calculates the tens digit and the ones digit
084      xor      a          ; ( acc = acc/10, work1 = acc mod 10 )
085      mov      #10,b      ;
086      div      ;
087      ld       b          ;
088      st       work1      ; Stores the ones digit in work1
089      ld       c          ;
090      pop      c          ; Pops the coordinate values into (c, b)
091      pop      b          ;
092      push     b          ; Pushes the coordinates onto the stack again
093      push     c          ;
094      call     putch      ; Displays the tens digit
095      ld       work1      ; Loads the ones digit
096      pop      c          ; Pops the coordinate values into (c, b)
097      pop      b          ;
098      inc      c          ; Moves the display coordinates to the right
099      call     putch      ; Displays the ones digit
100      ret              ; put2digit end
101
102 ; *-----*
103 ; * Clearing the LCD Display Image
104 ; *-----*
105 cls:
106      push     OCR        ; Pushes the OCR value onto the stack
107      mov      #osc_rc,OCR ; Specifies the system clock
108
109      mov      #0,XBNK    ; Specifies the display RAM bank address (BANK0)
110      call     cls_s      ; Clears the data in that bank
111
112      mov      #1,XBNK    ; Specifies the display RAM bank address (BANK1)
113      call     cls_s      ; Clears the data in that bank
114      pop      OCR        ; Pops the OCR value off of the stack
115
116      ret              ; cls end
117
118 cls_s:
119      mov      #80h,r2     ; Clearing One Bank of Display RAM
120      mov      #80h,b      ; Points the indirect addressing register at the start of display RAM
121      ; Sets the number of loops in loop counter b
122 loop3:
123      mov      #0,@r2      ; Writes "0" while incrementing the address
124      inc      r2          ;
125      dbnz     b,loop3     ; Repeats until b is "0"
126      ret              ; cls_s end
127
128
129 ; *-----*
130 ; * Displaying One Character in a Specified Position
131 ; *
132 ; * Inputs: acc:      Character code
133 ; *
134 ; *          c:Horizontal position of character
135 ; *          b:Vertical position of character
136 ; *-----*
```



```
134 ; *-----*
135 putch:
136     push    XBNK
```

Visual Memory Tutrial Revision 1.00

```
137     push    acc
138     call    locate    ; Calculates display RAM address according to coordinates
139     pop     acc
140     call    put_chara ; Displays one character
141     pop     XBNK
142
143     ret     ; putch end
144
145
146 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification ****
147     ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
148     ; ** Outputs: r2: RAM address XBNK: Display RAM bank
149
150     ; *** Determining the Display RAM Bank Address ***
151     ld      b        ; Jump to next1 when b >= 2
152     sub     #2        ;
153     bn      PSW,7,next1 ;
154
155     mov     #00h,XBNK ; Specifies the display RAM bank address (BANK0)
156     br      next2
157 next1:
158     st      b
159     mov     #01h,XBNK ; Specifies the display RAM bank address (BANK1)
160 next2:
161
162     ; *** Calculating the RAM Address for a Specified Position on the
Display ***
163     ld      b        ; b * 40h + c + 80h
164     rol
165     rol
166     rol
167     rol
168     rol
169     rol
170     add     c
171     add     #80h
172     st      r2        ; Stores the RAM address in r2
173
174     ret     ; locate end
175
176
177 put_chara:
178     push    PSW        ; Pushes the PSW value onto the stack
179     set1    PSW,1      ; Selects data RAM bank 1
180
181     ; *** Calculating the Character Data Address ***
182     rol
183     rol
184     rol
185     add     #low(fontdata) ;
186     st      TRL        ;
187     mov     #0,acc      ;
188     addc    #high(fontdata) ;
189     st      TRH        ;
190
191     push    OCR        ; Pushes the OCR value onto the stack
192     mov     #osc_rc,OCR ; Specifies the system clock
193
194     mov     #0,b        ; Offset value for loading the character data
195     mov     #4,c        ; Loop counter
196 loop1:
197     ld      b        ; Loads the display data for the first line
198     ldc
199     inc     b        ; Increments the load data offset by 1
200     st     @r2        ; Transfers the display data to display RAM
201     ld      r2        ; Adds 6 to the display RAM address
202     add     #6
203     st      r2
204
205     ld      b        ; Loads the display data for the second line
206     ldc
207     inc     b        ; Increments the load data offset by 1
208     st     @r2        ; Transfers the display data to display RAM
```

```

209      ld      r2          ; Adds 10 to the display RAM address
210      add     #10         ;
211      st      r2          ;
212
213      dec     c            ; Decrements the loop counter
214      ld      c            ;
215      bnz     loop1        ; Repeats for 8 lines (four times)
216
217      pop     OCR          ; Pops the OCR value off of the stack
218      pop     PSW          ; Pops the PSW value off of the stack
219
220      ret              ; put_chara end
221
222
223 ; *-----*
224 ; * Character Bit Image Data
225 ; *
226 ; *-----*
227 fontdata:
228      db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h      ; '0' 00
229      db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h      ; '1' 01
230      db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h      ; '2' 02
231      db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h      ; '3' 03
232      db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h      ; '4' 04
233      db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h      ; '5' 05
234      db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h      ; '6' 06
235      db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h      ; '7' 07
236      db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h      ; '8' 08
237      db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h      ; '9' 09
238
239 ; *-----*
240 ; * Base Timer Interrupt Handler
241 ; *
242 ; *-----*
242 int_BaseTimer:
243      push    acc          ; Pushes the register that was used onto the stack
244      inc     counter       ; Increments the counter
245      ld      counter       ; If the counter reaches 100...
246      bne     #100,next3    ;
247      mov     #0,counter    ; Resets the counter
248 next3:
249      pop     acc          ; Pops the register back off of the stack
250
251      ret              ; (User) interrupt processing end

```

C.4 Button Press Detection

This sample program checks the status (pressed or not pressed) of the Visual Memory buttons (except for the reset button and the mode button), and displays on the LCD any button that was pressed.

Line 46 writes 0FFH to port 3, pulling up all bits.

The button press status is loaded in line 51 by loading the status of port 3 into the ACC. If there is a button that is being pressed at this point, the corresponding bit is reset to “0”.

In line 52, the bits are checked to see if they are set (i.e., the corresponding button is not being pressed), and then control proceeds to the next button check processing. If a button is being pressed, the condition on line 52 becomes false, and the processing that is indicated for that button is performed.

Note

The port 3 interrupt is enabled immediately after this program is called from the system BIOS. Furthermore, because the port 3 interrupt is a level interrupt, the interrupt remains in effect while a button is being pressed.

Disabling the port 3 interrupt improves the overall performance of applications.

Note that in an application that cancels HALT mode in response to a port 3 interrupt, the port 3 interrupt must be enabled beforehand.

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** Button Status Detection Sample 1 **
005 ;
006 ; Reads the button statuses and displays the button that is being pressed on
    the LCD
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; Specifies the chip type for the assembler
012 world     external         ; External memory program
013
014 public    main              ; Symbol referenced from ghead.asm
015
016 extern    _game_end         ; Symbol reference to ghead.asm
017
018
019 ; **** Definition of System Constants ****
020
021
022 osc_rc     equ 04dh          ; OCR (Oscillation Control Register) settings
                                ; Specifies internal RC oscillation for the system clock
023 osc_xt     equ 0efh          ; Specifies crystal oscillation for the system clock
024
025
026 ; *** Data Segment ****
027
028          dseg                ; Data segment start
029
030 r0         ds      1          ; Indirect addressing register r0
031 r1         ds      1          ; Indirect addressing register r1
032 r2         ds      1          ; Indirect addressing register r2
033 r3         ds      1          ; Indirect addressing register r3
034           ds      12          ; Other registers reserved for the system
035
036 ; *** Code Segment ****
037
038          cseg                ; Code segment start
```

```

039
040 ; *-----*
041 ; * User program
    *
042 ; *-----*
043 main:
044     call    cls          ; Clears the LCD display image
045
046     mov     #0f0h,P3     ; P3 initialization (pull-up setting)
047
048 loop0:
049     ; ** [A] Button Check **
050     mov     #0,b
051     ld      P3           ; Loads the status of P3
052     bp      acc,4,next3  ; next3 if [A] button is being pressed
053     mov     #1,b        ; Display character code 'A'
054 next3:
055     ld      b
056     mov     #4,c        ; Display coordinate (horizontal)
057     mov     #3,b        ; Display coordinate (vertical)
058     call    putch       ; Displays single character
059
060     ; ** [B] Button Check **
061     mov     #0,b
062     ld      P3
063     bp      acc,5,next4  ; next4 if [B] button is being pressed
064     mov     #2,b        ; Display character code 'B'
065 next4:
066     ld      b
067     mov     #5,c
068     mov     #2,b
069     call    putch
070
071     ; ** [↑] Button Check **
072     mov     #0,b
073     ld      P3
074     bp      acc,0,next5  ; next5 if [↑] button is being pressed
075     mov     #3,b        ; Display character code '↑'
076 next5:
077     ld      b
078     mov     #1,c
079     mov     #1,b
080     call    putch
081
082     ; ** [→] Button Check **
083     mov     #0,b
084     ld      P3
085     bp      acc,3,next6  ; next6 if [→] button is being pressed
086     mov     #4,b        ; Display character code '→'
087 next6:
088     ld      b
089     mov     #2,c
090     mov     #2,b
091     call    putch
092
093     ; ** [↓] Button Check **
094     mov     #0,b
095     ld      P3
096     bp      acc,1,next7  ; next7 if [↓] button is being pressed
097     mov     #5,b        ; Display character code '↓'
098 next7:
099     ld      b
100     mov     #1,c
101     mov     #3,b
102     call    putch
103
104     ; ** [←] Button Check **
105     mov     #0,b
106     ld      P3
107     bp      acc,2,next8  ; next8 if [←] button is being pressed
108     mov     #6,b        ; Display character code '←'
109 next8:
110     ld      b
111     mov     #0,c
112     mov     #2,b

```

Visual Memory Tutrial Revision 1.00

```
113      call    putch
114
115                      ; ** [S] Button Check **
116      mov     #0,b
117      ld      P3
118      bp      acc,7,next9 ; next9 if [S] button is being pressed
119      mov     #8,b      ; Display character code 'S'
120 next9:
121      ld      b
122      mov     #4,c
123      mov     #1,b
124      call    putch
125
126                      ; ** [M] Button Check **
127      ld      P3
128      bn      acc,6,finish ; If the [M] button is pressed, the application ends
129
130      brf     loop0      ; Repeat
131
132 finish:
133      jmp     _game_end  ; Application end
134
135
136 ; *-----*
137 ; * Clearing the LCD Display Image
138 ; *-----*
139 cls:
140      push    OCR        ; Pushes the OCR value onto the stack
141      mov     #osc_rc,OCR ; Specifies the system clock
142
143      mov     #0,XBNK    ; Specifies the display RAM bank address (BANK0)
144      call    cls_s      ; Clears the data in that bank
145
146      mov     #1,XBNK    ; Specifies the display RAM bank address (BANK1)
147      call    cls_s      ; Clears the data in that bank
148      pop     OCR        ; Pops the OCR value off of the stack
149
150      ret
151
152 cls_s:
153      mov     #80h,r2    ; *** Clearing One Bank of Display RAM ***
154      mov     #80h,b     ; Points the indirect addressing register at the start of display RAM
155      ; Sets the number of loops in loop counter b
156 loop3:
157      mov     #0,@r2     ; Writes "0" while incrementing the address
158      inc     r2
159      dbnz    b,loop3    ; Repeats until b is "0"
160      ret
161
162
163 ; *-----*
164 ; * Displaying One Character in a Specified Position
165 ; *
166 ; * Inputs: acc: Character code
167 ; *
168 ; * c:Horizontal position of character
169 ; *
170 ; * b:Vertical position of character
171 ; *-----*
172 putch:
173      push    XBNK
174      push    acc
175      call    locate     ; Calculates display RAM address according to coordinates
176      pop     acc
177      call    put_chara  ; Displays one character
178      pop     XBNK
179
180      ret
181
182
183 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification ****
184      ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
185      ; ** Outputs: r2: RAM address XBNK: Display RAM bank
186
187      ; **** Determining the Display RAM Bank Address ****
```


Visual Memory Tutrial Revision 1.00

```

185      ld      b          ; Jump to next1 when b >= 2
186      sub     #2         ;
187      bn     PSW,7,next1 ;
188
189      mov     #00h,XBNK   ; Specifies the display RAM bank address (BANK0)
190      br     next2
191 next1:
192      st      b
193      mov     #01h,XBNK   ; Specifies the display RAM bank address (BANK1)
194 next2:
195
196      ; *** Calculating the RAM Address for a Specified Position on the
Display ***
197      ld      b          ; b * 40h + c + 80h
198      rol     ;
199      rol     ;
200      rol     ;
201      rol     ;
202      rol     ;
203      rol     ;
204      add     c          ;
205      add     #80h       ;
206      st      r2         ; Stores the RAM address in r2
207
208      ret          ; locate end
209
210
211 put_chara:
212      push    PSW        ; Pushes the PSW value onto the stack
213      setl    PSW,1      ; Selects data RAM bank 1
214
215      ; *** Calculating the Character Data Address ***
216      rol     ; (TRH,TRL) = acc*8 + fontdata
217      rol     ;
218      rol     ;
219      add     #low(fontdata) ;
220      st      TRL        ;
221      mov     #0,acc      ;
222      addc    #high(fontdata) ;
223      st      TRH        ;
224
225      push    OCR         ; Pushes the OCR value onto the stack
226      mov     #osc_rc,OCR ; Specifies the system clock
227
228      mov     #0,b        ; Offset value for loading the character data
229      mov     #4,c        ; Loop counter
230 loop1:
231      ld      b          ; Loads the display data for the first line
232      ldc     ;
233      inc     b          ; Increments the load data offset by 1
234      st      @r2        ; Transfers the display data to display RAM
235      ld      r2         ; Adds 6 to the display RAM address
236      add     #6         ;
237      st      r2         ;
238
239      ld      b          ; Loads the display data for the second line
240      ldc     ;
241      inc     b          ; Increments the load data offset by 1
242      st      @r2        ; Transfers the display data to display RAM
243      ld      r2         ; Adds 10 to the display RAM address
244      add     #10        ;
245      st      r2         ;
246
247      dec     c          ; Decrements the loop counter
248      ld      c          ;
249      bnz     loop1      ; Repeats for 8 lines (four times)
250
251      pop     OCR         ; Pops the OCR value off of the stack
252      pop     PSW        ; Pops the PSW value off of the stack
253
254      ret          ; put_chara end
255
256

```



```
257 ; *-----*
258 ; * Character Bit Image Data
    *
259 ; *-----*
260 fontdata:
261         db 000h, 000h, 038h, 038h, 038h, 000h, 000h, 000h      ; '•' 00
262         db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h      ; 'A' 01
263         db 0fch, 066h, 066h, 07ch, 066h, 066h, 0fch, 000h      ; 'B' 02
264
265         db 010h, 038h, 07ch, 0feh, 038h, 038h, 038h, 000h      ; '↑' 03
266         db 010h, 018h, 0fch, 0feh, 0fch, 018h, 010h, 000h      ; '→' 04
267         db 038h, 038h, 038h, 0feh, 07ch, 038h, 010h, 000h      ; '↓' 05
268         db 010h, 030h, 07eh, 0feh, 07eh, 030h, 010h, 000h      ; '←' 06
269
270         db 0c6h, 0eeh, 0feh, 0d6h, 0c6h, 0c6h, 0c6h, 000h      ; 'M' 07
271         db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h      ; 'S' 08
```

C.5 Using the PWM Sound Source

This sample program alternately generates a high tone (781Hz) and a low tone (342Hz).

The important portion of this sample is the subroutine that starts from line 72.

“SndInit” readies the program to use PWM. “Snd1(2)Start” sets the frequency that is generated by timer 1, and begins the counting operation. “SndStop” stops the counting operation and stops the audio output.

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** Sound Usage Sample 1 **
005 ;
006 ; Intermittently outputs two tones (high/low)
007 ; (Low tone for 0.5 seconds - Silence for 0.5 seconds - High tone for 0.5
    seconds - Silence for 0.5 seconds...)
008 ;-----
009 ; 1.00 981208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip      LC868700          ; Specifies the chip type for the assembler
013 world     external         ; External memory program
014
015 public    main              ; Symbol referenced from ghead.asm
016
017 extern    _game_end         ; Application end
018
019
020 ; **** Definition of System Constants ****
021
022                                ; OCR (Oscillation Control Register) settings
023 osc_rc     equ 081h          ; Specifies internal RC oscillation for the system clock
024 osc_xt     equ 082h          ; Specifies crystal oscillation for the system clock
025
026
027 ; *** Data Segment ****
028
029          dseg                ; Data segment start
030
031 r0:       ds      1          ; Indirect addressing register r0
032 r1:       ds      1          ; Indirect addressing register r1
033 r2:       ds      1          ; Indirect addressing register r2
034 r3:       ds      1          ; Indirect addressing register r3
035          ds      12          ; Other registers reserved for the system
036
037
038 ; *** Code Segment ****
039
040          cseg                ; Code segment start
041
042 ; *-----*
043 ; * User program
    *
044 ; *-----*
045 main:
046          call     SndInit     ; Sound initialization
047
048 loop0:
049          call     Snd1Start   ; Starts generating tone at approximately 342Hz
050          set1     PCON,0      ; HALT mode until base timer interrupt (0.5 seconds)
051
052          call     SndStop     ; Buzzer sound off
053          set1     PCON,0      ; HALT mode until base timer interrupt (0.5 seconds)
054
055          call     Snd2Start   ; Starts generating tone at approximately 781
056          set1     PCON,0      ; HALT mode until base timer interrupt (0.5 seconds)
057
```

```

058      call    SndStop      ; Buzzer sound off
059      set1    PCON,0       ; HALT mode until base timer interrupt (0.5 seconds)
060
061
062                                ; ** [M] (mode) Button Check **
063      ld      P3
064      bn      acc,6,finish  ; If the [M] button is pressed, the application ends
065
066      br      loop0        ; Repeat
067
068 finish:                                ; ** Application End Processing **
069      jmp     _game_end    ; Application end
070
071
072 ; *-----*
073 ; * Sound Output-related Routines *
074 ; *-----*
075 SndInit:                                ; *** Sound Output Hardware Initialization ***
076      clr1    P1,7         ; Sets the sound output port
077
078      ret
079
080 Snd1Start:                                ; *** Start of 342Hz Tone ***
081      mov     #0f0h,T1LR   ; Cycles = 100h - 0f0h = 16
082      mov     #0f8h,T1LC   ; L level width = 100h - 0f8h = 8
083      mov     #0D0h,T1CNT  ; Sound output start
084
085      ret
086
087 Snd2Start:                                ; *** Start of 781Hz Tone ***
088      mov     #0f9h,T1LR   ; Cycles = 100h - 0f9h = 7
089      mov     #0fch,T1LC   ; L level width = 100h - 0fch = 4
090      mov     #0D0h,T1CNT  ; Sound output start
091
092      ret
093
094 SndStop:                                ; *** Sound Stop ***
095      mov     #0,T1CNT     ; Stops sound output
096
097      ret

```

C.6 Interrupt Using Timer 0

This sample program generates an interrupt once every second. The program generates a sound when the interrupt is generated.

The important portion of this program is the routine “T0Mode2Init” in lines 78 through 93. This program uses Timer 0 in mode 2, as a 16-bit counter with prescaler. Because a 32kHz signal is input to the timer, the program sets up the timer so that the signal causes an overflow in the prescaler and counter approximately every second.

Because Timer 0 generates an interrupt in response to the overflow, the program provides a handler for that interrupt. This handler increments the count, resets the interrupt source flag to “0”, and then terminates interrupt processing.

The main routine determines whether the counter value is an even or odd number, and uses this information to output a high tone and a low tone in alternation. In line 61, the CPU is put into HALT mode, and operation stops until an interrupt is generated. If a timer 0 interrupt or a port 3 interrupt is generated, processing resumes from the next line.

• GHEAD.ASM

```
001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *              05/20-'98          *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main          ; Symbol in the user program
014 extern  int_T0H        ; Symbol in the user program
015
016 ; *-----*
017 ; * Vector table(?)          *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023         jmpf      main      ; main program jump
024 org 0003h
025 ;int_03:
026         jmp int_03
027 org 000bh
028 ;int_0b:
029         jmp int_0b
030 org 0013h
031 ;int_13:
032         jmp int_13
033 org 001bh
034 ;int_1b:
035         jmp int_1b
036 org 0023h
037 ;int_23:
038         jmp int_23
039 org 002bh
040 ;int_2b:
041         jmp int_2b
042 org 0033h
043 ;int_33:
044         jmp int_33
```

```

045 org 003bh
046 ;int_3b:
047     jmp int_3b
048 org 0043h
049 ;int_43:
050     jmp int_43
051 org 004bh
052 ;int_4b:
053     jmp int_4b
054 ; *-----*
055 ; * interrupt programs                *
056 ; *-----*
057 int_03:
058     reti
059 int_0b:
060     reti
061 int_13:
062     reti
063 int_23:
064     jmp     int_T0H      ; (To user interrupt processing)
065 int_2b:
066     reti
067 int_33:
068     reti
069 int_3b:
070     reti
071 ; *-----*
072 int_43:
073     reti
074 int_4b:
075     clr1    p3int,1      ; interrupt flag clear
076     reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program *
081 ; *-----*
082 fm_wrt_ex:
083     change  fm_wrt_in
084     fm_wrt_ex_exit:
085     ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program *
089 ; *-----*
090 fm_vrf_ex:
091     change  fm_vrf_in
092     fm_vrf_ex_exit:
093     ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program *
098 ; *-----*
099 fm_prd_ex:
100     change  fm_prd_in
101     fm_prd_ex_exit:
102     ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*
108 int_1b:
109 timer_ex:
110     push    ie
111     clr1    ie,7        ; interrupt prohibition
112     change  timer_in
113     timer_ex_exit:
114     pop     ie
115     reti
116
117 org 01f0h
118 _game_end:

```

```
119         change   game_end
120     end
```

• TIMER1.ASM

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** Timer/Counter T0 Interrupt Usage Sample 1 **
005 ;
006 ; ·Intermittently sounds the buzzer (every two seconds)
007 ;-----
008 ; 1.00 981208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; Specifies the chip type for the assembler
012 world     external         ; External memory program
013
014 public    main              ; Symbol referenced from ghead.asm
015 public    int_T0H           ; Symbol referenced from ghead.asm
016
017 extern    _game_end         ; Application end
018
019
020 ; **** Definition of System Constants ****
021
022                                ; OCR (Oscillation Control Register) settings
023 osc_rc     equ 04dh          ; Specifies internal RC oscillation for the system clock
024 osc_xt     equ 0efh          ; Specifies crystal oscillation for the system clock
025
026
027 ; *** Data Segment ****
028
029         dseg                ; Data segment start
030
031 r0:      ds      1          ; Indirect addressing register r0
032 r1:      ds      1          ; Indirect addressing register r1
033 r2:      ds      1          ; Indirect addressing register r2
034 r3:      ds      1          ; Indirect addressing register r3
035          ds      12         ; Other registers reserved for the system
036
037 counter: ds      1          ; Timer interrupt counter
038
039
040 ; *** Code Segment ****
041
042         cseg                ; Code segment start
043
044 ; *-----*
045 ; * User program
046 ; *-----*
047 main:
048     call    SndInit         ; Sound output initialization
049     call    T0Mode2Init     ; Timer T0 initialization
050     mov     #0,counter      ; Clears counter
051
052 loop0:
053     ld      counter         ; Loads the counter value
054     bp      acc,1,next1     ; next1 if bit 0 of the counter is "1"
055
056     call    Snd2Start       ; Starts sound
057     br      next2
058 next1:
059     call    SndStop         ; Stops sound
060 next2:
061     set1    PCON,0          ; HALT mode until next interrupt
062
063
064                                ; ** [M] (mode) Button Check **
065     ld      P3
```

```

066      bn      acc,6,finish ; If the [M] button is pressed, the application ends
067
068      br      loop0        ; Repeat
069
070 finish:
071      jmp      _game_end    ; ** Application End Processing **
072
073
074 ; *-----*
075 ; * Timer/Counter T0 Initialization *
076 ; *   Applied an interrupt about once per second in mode 2 (16-bit counter) *
077 ; *-----*
078 T0Mode2Init:
079      mov      #255,T0PRR    ; Sets the prescaler value
080                      ; Since this is an 8-bit prescaler:
081                      ; Cycle = (256-255) * 0.000183 = 0.000183 (sec.)
082      mov      #high(65536-5464),T0HR ; Sets preset value (H)
083      mov      #low(65536-5464),T0LR  ; Sets preset value (L)
084                      ; As a set with the prescaler:
085                      ; 0.000183 * 5464 = 0.999912 (=1sec)
086                      ; An overflow occurs about once per second
087      mov      #0ffh,T0L        ; Sets up an immediate initial overflow
088      mov      #0ffh,T0H        ;
089      mov      #0e4h,T0CNT      ; Mode 2 (16-bit counter)
090                      ; Generates an interrupt according to the T0H overflow
091                      ; T0 operation start
092
093      ret                      ; T0Mode2Init end
094
095
096 T0HStop:
097                      ; *** T0H timer stop ***
098      clr1     T0CNT,7        ; T0H count operation stop
099      ret
100
101
102 ; *-----*
103 ; * Timer T0H Interrupt Handler *
104 ; *-----*
105 int_T0H:
106      inc      counter        ; *** T0H Interrupt Handler ***
107
108      clr1     T0CNT,3        ; Clears the timer T0H interrupt source
109      reti
110
111
112 ; *-----*
113 ; * Sound Output-related Routines *
114 ; *-----*
115 SndInit:
116      clr1     P1,7          ; *** Sound Output Hardware Initialization ***
117                      ; Sets the sound output port
118      ret
119
120 Snd1Start:
121                      ; *** Start of 342Hz Tone ***
122      mov      #0f0h,T1LR    ; Cycles = 100h - 0f0h = 16
123      mov      #0f8h,T1LC    ; L level width = 100h - 0f8h = 8
124      mov      #0D4h,T1CNT   ; Sound output start
125      ret
126
127 Snd2Start:
128                      ; *** Start of 781Hz Tone ***
129      mov      #0f9h,T1LR    ; Cycles = 100h - 0f9h = 7
130      mov      #0fch,T1LC    ; L level width = 100h - 0fch = 4
131      mov      #0D4h,T1CNT   ; Sound output start
132      ret
133
134 SndStop:
135                      ; *** Sound Stop ***
136      mov      #0,T1CNT      ; Stops sound output
137      ret

```

C.7 Serial Communications (Sending Side)

This sample program uses the serial interface to send data values from 0 to 99.

Caution

Conduct serial communications with crystal oscillation.

Perform reception with the “Serial Communications (Receiving Side)” that is described on the following page.

Line 53 disables automatic low battery detection. The actual routines are “BattChkOn” and “BattChkOff” in line 158 and beyond.

Line 56 initializes the serial interface. The actual initialization routine starts in line 95. This routine is very detailed, so we recommend simply copying this code and using it as is.

After initialization, a counter is incremented by the base timer interrupt, which is generated every 0.5 seconds. The value of this counter is sent through the serial interface.

If this program is halted by pressing the MODE button, the standard value is written for the serial interface again (in the “SioEnd” routine that starts from line 126), automatic low battery detection is enabled again, and the program ends.

• GHEAD.ASM

```
001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *              05/20-'98          *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main          ; Symbol in the user program
014 extern  int_BaseTimer ; Symbol in the user program
015
016 ; *-----*
017 ; * Vector table(?)          *
018 ; *-----*
019 cseg
020 org 0000h
021 _game_start:
022 ;reset:
023     jmpf    main      ; main program jump
024 org 0003h
025 ;int_03:
026     jmp int_03
027 org 000bh
028 ;int_0b:
029     jmp int_0b
030 org 0013h
031 ;int_13:
032     jmp int_13
033 org 001bh
034 ;int_1b:
035     jmp int_1b
```



```

036 org 0023h
037 ;int_23:
038     jmp int_23
039 org 002bh
040 ;int_2b:
041     jmp int_2b
042 org 0033h
043 ;int_33:
044     jmp int_33
045 org 003bh
046 ;int_3b:
047     jmp int_3b
048 org 0043h
049 ;int_43:
050     jmp int_43
051 org 004bh
052 ;int_4b:
053     jmp int_4b
054 ; *-----*
055 ; * interrupt programs          *
056 ; *-----*
057 int_03:
058     reti
059 int_0b:
060     reti
061 int_13:
062     reti
063 int_23:
064     reti
065 int_2b:
066     reti
067 int_33:
068     reti
069 int_3b:
070     reti
071 ; *-----*
072 int_43:
073     reti
074 int_4b:
075     clr1    p3int,1    ; interrupt flag clear
076     reti
077
078 org 0100h
079 ; *-----*
080 ; * flash memory write external program      *
081 ; *-----*
082 fm_wrt_ex:
083     change  fm_wrt_in
084     fm_wrt_ex_exit:
085     ret
086 org 0110h
087 ; *-----*
088 ; * flash memory verify external program      *
089 ; *-----*
090 fm_vrf_ex:
091     change  fm_vrf_in
092     fm_vrf_ex_exit:
093     ret
094
095 org 0120h
096 ; *-----*
097 ; * flash memory page read external program      *
098 ; *-----*
099 fm_prd_ex:
100     change  fm_prd_in
101     fm_prd_ex_exit:
102     ret
103
104 org 0130h
105 ; *-----*
106 ; * flash memory => timer call external program *
107 ; *-----*

```

```
108 int_1b:
109 timer_ex:
110     push    ie
111     clr1    ie,7          ; interrupt prohibition
112     change  timer_in
113     timer_ex_exit:
114     call    int_BaseTimer ; User interrupt processing
115     pop ie
116     reti
117
118 org 01f0h
119 _game_end:
120     change  game_end
121 end
```

• TIMER1.ASM

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** Serial Communications Sample 1 (Data Transmission) **
005 ;
006 ; .Sends simple data through the serial communications port on a regular cycle
007 ;-----
008 ; 1.01 990208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip    LC868700          ; Specifies the chip type for the assembler
012 world   external         ; External memory program
013
014 public  main              ; Symbol referenced from ghead.asm
015 public  int_BaseTimer     ; Symbol referenced from ghead.asm
016
017 extern  _game_end         ; Application end
018
019
020 ; **** Definition of System Constants ****
021
022                                ; OCR (Oscillation Control Register) settings
023 osc_rc    equ 04dh          ; Specifies internal RC oscillation for the system clock
024 osc_xt    equ 0efh          ; Specifies crystal oscillation for the system clock
025
026 LowBattChk equ 06eh         ; Low battery detection flag (RAM bank 0)
027
028
029 ; *** Data Segment ****
030
031     dseg                    ; Data segment start
032
033 r0:      ds      1          ; Indirect addressing register r0
034 r1:      ds      1          ; Indirect addressing register r1
035 r2:      ds      1          ; Indirect addressing register r2
036 r3:      ds      1          ; Indirect addressing register r3
037         ds      12          ; Other registers
038
039 counter: ds      1          ; Counter
040
041
042 ; *** Code Segment ****
043
044     cseg                    ; Code segment start
045
046 ; *-----*
047 ; * User program
048 ; *-----*
049 main:
050     push    PSW            ; Pushes the PSW value onto the stack
051     set1    PSW,1          ; Selects data RAM bank 1
052
053     call    BattChkOff     ; Turns off the low battery automatic detection function
```

```

054
055 cwait:
056     call    SioInit      ; Serial communications initialization
057     bz      start       ; Starts if VM is connected
058
059     ld      P3           ; [M] button check
060     bn      acc,6,finish ; If the [M] button is pressed, the application ends
061
062     jmp     cwait        ; Waits until VM is connected
063 start:
064
065     setl    pcon,0       ; Waits in HALT mode until next interrupt (0.5 seconds)
066
067     mov     #0,counter   ; Resets the counter value to "0"
068 loop0:
069     ld      counter      ; Loads the counter value
070
071     call    SioSend1     ; Sends one byte
072
073     setl    pcon,0       ; Waits in HALT mode until next interrupt (0.5 seconds)
074
075                                ; [M] (mode) Button Check
076     ld      P3
077     bn      acc,6,finish ; If the [M] button is pressed, the application ends
078
079     jmp     loop0        ; Repeat
080
081 finish:
082     call    SioEnd       ; ** Application End Processing **
083     call    BattChkOn    ; Turns on the low battery automatic detection function
084     pop     PSW          ; Pops the PSW value off of the stack
085     jmp     _game_end    ; Application end
086
087 ; *-----*
088 ; * Serial Communications Initialization
089 ; *
090 ; * Outputs:    acc = 0    : Normal end
091 ; *
092 ; *          acc = 0ffh: VM not connected
093 ; *-----*
094 ; Serial communications initialization
095 ; This sample assumes that the system clock is in crystal mode.
096
097 SioInit:
098     ; **** VM Connection Check ****
099     ld      P7           ; Checks the connection status
100     and     #00001101    ; Checks P70, P72, P73
101     sub     #00001000    ; P70 = 0, P72 = 0, P73 = 1
102     bz      next2       ; To next2 if connected
103
104     mov     #0ffh,acc     ; If not connected, abnormal end with acc = 0ffh
105     ret
106
107 next2:
108     ; **** Serial Communications Initialization ****
109     mov     #0,SCON0     ; Specifies output as 'LSB first'
110     mov     #0,SCON1     ; Specifies input as 'LSB first'
111     mov     #0ddh,SBR    ; Sets the transfer rate
112     clr1    P1,0         ; Clears the P10 latch (P10/S00)
113     clr1    P1,2         ; Clears the P12 latch (P12/SCK0)
114     clr1    P1,3         ; Clears the P13 latch (P13/S01)
115
116     mov     #00000101,P1FCR ; Sets the pin functions
117     mov     #00000101,P1DDR ; Sets the pin functions
118
119     mov     #0,SBUF0     ; Clears the transfer buffer
120     mov     #0,SBUF1     ; Clears the transfer buffer
121
122     ret
123 ; *-----*
124 ; * Serial Communications End
125 ; *-----*

```

```
126 SioEnd:                ; **** Serial Communications End Processing ****
127
128     mov     #0,SCON0     ; SCON0 = 0
129     mov     #0,SCON1     ; SCON1 = 0
130     mov     #0bfh,P1FCR  ; P1FCR = 0bfh
131     mov     #0a4h,P1DDR  ; P1DDR = 0a4h
132
133     ret                     ; SioEnd end
134
135
136 ; *-----*
137 ; * Sending 1 Byte from a Serial Port
138 ; *
139 ; *   Inputs: acc: Transmission data
140 ; *
141 ; *-----*
142 SioSend1:                ; **** Sending 1 Byte ****
143
144     push    acc           ; Pushes the transmission data onto the stack
145
146     sslp1: ld     SCON0     ; Waits, if the previous data is still being sent
147            bp     acc,3,sslp1 ;
148
149     pop     acc           ; Pops the transmission data off of the stack
150
151     st      SBUF0         ; Sets the data to be transferred
152     set1    SCON0,3       ; Starts sending
153
154     ret                     ; SioSend1 end
155
156 ; *-----*
157 ; * Low Battery Automatic Detection Function ON
158 ; *
159 ; *-----*
160 BattChkOn:
161
162     push    PSW           ; Pushes the PSW value onto the stack
163
164     clr1    PSW,1         ; Selects data RAM bank 0
165     mov     #0,acc        ; Detects low battery (0)
166     st      LowBattChk    ; Low battery automatic detection flag (RAM bank 0)
167
168     pop     PSW           ; Pops the PSW value off of the stack
169     ret                     ; BattChkOn end
170
171 ; *-----*
172 ; * Low Battery Automatic Detection Function OFF
173 ; *
174 ; *-----*
175 BattChkOff:
176
177     push    PSW           ; Pushes the PSW value onto the stack
178
179     clr1    PSW,1         ; Selects data RAM bank 0
180     mov     #0ffh,acc     ; Does not detect low battery (0ffh)
181     st      LowBattChk    ; Low battery automatic detection flag (RAM bank 0)
182
183     pop     PSW           ; Pops the PSW value off of the stack
184     ret                     ; BattChkOff end
185
186 ; *-----*
187 ; * Base Timer Interrupt Handler
188 ; *
189 ; *-----*
190 int_BaseTimer:
191
192     push    PSW           ; Pushes the PSW value onto the stack
193     push    acc
194
195     set1    PSW,1         ; Selects data RAM bank 1
196
197     inc     counter       ; Increments the counter
198
199     ld      counter       ; If the counter value is:
200     bne     #100,next1    ; not 100, then next1
201     mov     #0,counter    ; 100, then reset to '0'
```

197 next1:

C.8 Serial Communications (Receiving Side)

This sample program uses the serial interface to receive data.

Because this program is intended to primarily explain serial communications, it does not use SIO interrupts. For details on serial communications in actual practice, refer to the “General-purpose Serial Driver,” described on the next page.

The program stops automatic low battery detection and initializes the serial interface.

Line 64 checks whether there is a byte of data in the serial interface. If there is, the received data is converted to a decimal value by the “put2digit” routine and is displayed on the LCD.

If this program is halted by pressing the MODE button, the standard value is written for the serial interface again (in the “SioEnd” routine that starts from line 121), automatic low battery detection is enabled again, and the program ends.

Caution

If data is sent before reception processing is completed, a data overflow occurs. This is not a problem in this sample program because the “Serial Communications (Sending Side)” sample program sends data every 0.5 seconds.

When receiving data consecutively, use the SIO interrupts.

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** Serial Communications Sample 2 (Data Reception) **
005 ;
006 ; -Displays a numeric value that was received from the serial communications
    port on the LCD
007 ;-----
008 ; 1.01 990208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      LC868700          ; Specifies the chip type for the assembler
012 world     external         ; External memory program
013
014 public    main              ; Symbol referenced from ghead.asm
015
016 extern    _game_end         ; Application end
017
018
019 ; **** Definition of System Constants ****
020
021                                     ; OCR (Oscillation Control Register) settings
022 osc_rc     equ 04dh          ; Specifies internal RC oscillation for the system clock
023 osc_xt     equ 0efh          ; Specifies crystal oscillation for the system clock
024
025 LowBattChk equ 06eh          ; Low battery detection flag (RAM bank 0)
026
027
028 ; *** Data Segment ****
029
030          dseg                ; Data segment start
031
032 r0:      ds      1           ; Indirect addressing register r0
033 r1:      ds      1           ; Indirect addressing register r1
034 r2:      ds      1           ; Indirect addressing register r2
035 r3:      ds      1           ; Indirect addressing register r3
036          ds      12          ; Other registers
037
038 counter: ds      1           ; Counter
039 work1:   ds      1           ; Work (used in put2digit)
040
041

```

```
042 ; *** Code Segment *****
043
044         cseg                ; Code segment start
045
046 ; *-----*
047 ; * User program
048 ; *-----*
049 main:
050         call    cls          ; Clears the LCD display
051         call    BattChkOff   ; Turns off the low battery automatic detection function
052
053 cwait:
054         call    SioInit      ; Serial communications initialization
055         bz      start        ; Starts if VM is connected
056
057         ld      P3           ; [M] button check
058         bn      acc,6,finish ; If the [M] button is pressed, the application ends
059
060         jmp     cwait        ; Waits until VM is connected
061 start:
062
063 loop0:
064         call    SioRecv1     ; Receives one byte
065         bnz     next4        ; If there is no received data, then goes to next4
066
067         ld      b            ; Loads the received data into acc
068         mov     #2,c         ; Display coordinates (horizontal)
069         mov     #1,b         ; Display coordinates (vertical)
070         call    put2digit    ; Displays the two-digit value on the LCD
071
072 next4:
073         ld      P3           ; ** [M] (mode) Button Check **
074         bn      acc,6,finish ; If the [M] button is pressed, the application ends
075
076         jmp     loop0        ; Repeat
077
078 finish:
079         call    SioEnd       ; ** Application End Processing **
080         call    BattChkOn    ; Turns on the low battery automatic detection function
081         jmp     _game_end    ; Application end
082
083 ; *-----*
084 ; * Serial Communications Initialization
085 ; *
086 ; * Outputs:  acc = 0      : Normal end
087 ; *
088 ; *          acc = 0ffh: VM not connected
089 ; *-----*
090 ; Serial communications initialization
091 ; This sample assumes that the system clock is in crystal mode.
092 SioInit:
093         ; **** VM Connection Check ****
094         ld      P7           ; Checks the connection status
095         and     #%00001101   ; Checks P70, P72, P73
096         sub     #%00001000   ; P70 = 0, P72 = 0, P73 = 1
097         bz      next3        ; To next3 if connected
098
099         mov     #0ffh,acc     ; If not connected, abnormal end with acc = 0ffh
100         ret              ; SioInit end
101
102         ; **** Serial Communications Initialization ****
103         mov     #0,SCON0     ; Specifies output as 'LSB first'
104         mov     #0,SCON1     ; Specifies input as 'LSB first'
105         mov     #088h,SBR    ; Sets the transfer rate
106         clr1    P1,0         ; Clears the P10 latch (P10/S00)
107         clr1    P1,2         ; Clears the P12 latch (P12/SCK0)
108         clr1    P1,3         ; Clears the P13 latch (P13/S01)
109
110         mov     #%00000101,P1FCR ; Sets the pin functions
111         mov     #%00000101,P1DDR ; Sets the pin functions
112
113         mov     #0,SBUF0     ; Clears the transfer buffer
114         mov     #0,SBUF1     ; Clears the transfer buffer
```


Visual Memory Tutrial Revision 1.00

```

115         ret                ; SioInit end
116
117
118 ; *-----*
119 ; * Serial Communications End
120 ; *-----*
121 SioEnd:                ; **** Serial Communications End Processing ****
122
123     mov     #0,SCON0      ; SCON0 = 0
124     mov     #0, SCON1     ; SCON1 = 0
125     mov     #0bfh,P1FCR   ; P1FCR = 0bfh
126     mov     #0a4h,P1DDR   ; P1DDR = 0a4h
127
128     ret                ; SioEnd end
129
130
131 ; *-----*
132 ; * Receiving 1 Byte from a Serial port
133 ; * Outputs: b: Received data
134 ; *      acc = 0      : Received data found
135 ; *
136 ; *      acc = 0ffh: Received data not found
137 ; *-----*
138 SioRecv1:                ; **** Receiving 1 Byte ****
139     ld      SCON1
140     bp      acc,1,next5   ; If received data is found, then go to next5
141     bp      acc,3,next6   ; If transfer is currently in progress, then go to next6
142
143     set1    SCON1,3       ; Starts transfer
144 next6:     mov     #0ffh,acc ; Returns with acc = 0ffh (received data not found)
145     ret                ; SioRecv1 end
146 next5:
147
148     ld      SBUF1         ; Loads the received data
149     st      b             ; Copies the data into b
150
151     clr1    SCON1,1       ; Resets the transfer end flag
152
153     mov     #0,acc        ; Returns with acc = 0 (received data found)
154     ret                ; SioRecv1 end
155
156
157 ; *-----*
158 ; * Displaying a two-digit value
159 ; *
160 ; *      Inputs: acc: Numeric value
161 ; *      c: Horizontal position of character
162 ; *      b: Vertical position of character
163 ; *-----*
164 put2digit:
165     push    b             ; Pushes the coordinate data onto the stack
166     push    c             ;
167     st      c             ; Calculates the tens digit and the ones digit
168     xor     a             ; ( acc = acc/10, work1 = acc mod 10 )
169     mov     #10,b         ;
170     div     ;
171     ld      b             ;
172     st      work1         ; Stores the ones digit in work1
173     ld      c             ;
174     pop     c             ; Pops the coordinate values into (c, b)
175     pop     b             ;
176     push    b             ; Pushes the coordinates onto the stack again
177     push    c             ;
178     call    putch         ; Displays the tens digit
179     ld      work1         ; Loads the ones digit
180     pop     c             ; Pops the coordinate values into (c, b)
181     pop     b             ;
182     inc     c             ; Moves the display coordinates to the right
183     call    putch         ; Displays the ones digit
184
185     ret                ; put2digit end
186

```



```
187 ; *-----*
188 ; * Clearing the LCD Display Image *
189 ; *-----*
190 cls:
191     push    OCR           ; Pushes the OCR value onto the stack
192     mov     #osc_rc,OCR   ; Specifies the system clock
193
194     mov     #0,XBNK       ; Specifies the display RAM bank address (BANK0)
195     call    cls_s         ; Clears the data in that bank
196
197     mov     #1,XBNK       ; Specifies the display RAM bank address (BANK1)
198     call    cls_s         ; Clears the data in that bank
199     pop     OCR           ; Pops the OCR value off of the stack
200
201     ret                     ; cls end
202
203 cls_s:
204     ; *** Clearing One Bank of Display RAM ***
205     mov     #80h,r2       ; Points the indirect addressing register at the start of display RAM
206     mov     #80h,b        ; Sets the number of loops in loop counter b
207 loop3:
208     mov     #0,@r2        ; Writes "0" while incrementing the address
209     inc     r2            ;
210     dbnz    b,loop3       ; Repeats until b is "0"
211     ret                     ; cls_s end
212
213
214 ; *-----*
215 ; * Displaying One Character in a Specified Position *
216 ; *
217 ; * Inputs: acc: Character code *
218 ; *           c: Horizontal position of character *
219 ; *           b: Vertical position of character *
220 ; *-----*
221 putch:
222     push    XBNK
223     push    acc
224     call    locate        ; Calculates display RAM address according to coordinates
225     pop     acc
226     call    put_chara     ; Displays one character
227     pop     XBNK
228     ret                     ; putch end
229
230
231 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification ****
232 ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
233 ; ** Outputs: r2: RAM address XBNK: Display RAM bank
234
235     ; *** Determining the Display RAM Bank Address ***
236     ld      b             ; Jump to next1 when b >= 2
237     sub     #2            ;
238     bn      PSW,7,next1   ;
239
240     mov     #00h,XBNK     ; Specifies the display RAM bank address (BANK0)
241     br      next2
242 next1:
243     st      b             ;
244     mov     #01h,XBNK     ; Specifies the display RAM bank address (BANK1)
245 next2:
246
247     ; *** Calculating the RAM Address for a Specified Position on the
248     Display ***
249     ld      b             ; b * 40h + c + 80h
250     rol     ;
251     rol     ;
252     rol     ;
253     rol     ;
254     rol     ;
255     add     c             ;
256     add     #80h         ;
257     st      r2           ; Stores the RAM address in r2
258
```


Visual Memory Tutrial Revision 1.00

```
259         ret                ; locate end
260
261
262 put_chara:
263     push    PSW             ; Pushes the PSW value onto the stack
264     setl    PSW,1          ; Selects data RAM bank 1
265
266     ; *** Calculating the Character Data Address ***
267     rol     ; (TRH,TRL) = acc*8 + fontdata
268     rol     ;
269     rol     ;
270     add     #low(fontdata) ;
271     st      TRL             ;
272     mov     #0,acc          ;
273     addc    #high(fontdata) ;
274     st      TRH             ;
275
276     push    OCR             ; Pushes the OCR value onto the stack
277     mov     #osc_rc,OCR     ; Specifies the system clock
278
279     mov     #0,b            ; Offset value for loading the character data
280     mov     #4,c            ; Loop counter
281 loop1:
282     ld      b               ; Loads the display data for the first line
283     ldc     ;
284     inc     b               ; Increments the load data offset by 1
285     st      @r2             ; Transfers the display data to display RAM
286     ld      r2              ; Adds 6 to the display RAM address
287     add     #6              ;
288     st      r2              ;
289
290     ld      b               ; Loads the display data for the second line
291     ldc     ;
292     inc     b               ; Increments the load data offset by 1
293     st      @r2             ; Transfers the display data to display RAM
294     ld      r2              ; Adds 10 to the display RAM address
295     add     #10             ;
296     st      r2              ;
297
298     dec     c               ; Decrements the loop counter
299     ld      c               ;
300     bnz     loop1          ; Repeats for 8 lines (four times)
301
302     pop     OCR             ; Pops the OCR value off of the stack
303     pop     PSW             ; Pops the PSW value off of the stack
304
305     ret                ; put_chara end
306
307
308 ; *-----*
309 ; * Character Bit Image Data
310 ; *-----*
311 fontdata:
312     db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h    ; '0' 00
313     db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h    ; '1' 01
314     db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h    ; '2' 02
315     db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h    ; '3' 03
316     db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h    ; '4' 04
317     db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h    ; '5' 05
318     db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h    ; '6' 06
319     db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h    ; '7' 07
320     db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h    ; '8' 08
321     db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h    ; '9' 09
322
323
324 ; *-----*
325 ; * Low Battery Automatic Detection Function ON
326 ; *-----*
327 BattChkOn:
328     push    PSW             ; Pushes the PSW value onto the stack
329
330     clr1    PSW,1          ; Selects data RAM bank 0
```

```

331      mov    #0,acc      ; Detects low battery (0)
332      st     LowBattChk  ; Low battery automatic detection flag (RAM bank 0)
333
334      pop     PSW         ; Pops the PSW value off of the stack
335      ret      ; BattChkOn end
336
337
338 ; *-----*
339 ; * Low Battery Automatic Detection Function OFF
340 ; *-----*
341 BattChkOff:
342      push    PSW         ; Pushes the PSW value onto the stack
343
344      clr1    PSW,1       ; Selects data RAM bank 0
345      mov     #0ffh,acc   ; Does not detect low battery (0ffh)
346      st     LowBattChk  ; Low battery automatic detection flag (RAM bank 0)
347
348      pop     PSW         ; Pops the PSW value off of the stack
349      ret      ; BattChkOff end

```

C.9 General-purpose Serial Driver

This is a serial transmission/reception program that uses a general-purpose serial driver with a buffer that uses the port 3 interrupt.

If this program is executed on two Visual Memory units, it can be used to send data back and forth between the units and to display the data on their LCDs.

The main routine checks the reception buffer and, if data is found, it gives the highest priority to displaying the received data on the LCD. If the buffer is empty, the program outputs the data to be sent (“0” to “99”). The data that is to be sent is incremented once every 0.5 seconds in response to the base timer interrupt.

The “SioInit” routine in lines 128 to 161 confirm that Visual Memory is connected, initialize the interface, initialize the buffer (RAM), and enable the SIO interrupts. The “SioGet1” routine in lines 203 to 245 get one byte of data that is waiting in the reception buffer.

The SIO reception handler, which operates when an SIO interrupt is received, is the “int_SioRx” routine in lines 278 to 317. The received data is stored in the buffer.

Caution

When performing communications using this sample program on both the receiving side and the sending side, no data overflow occurs, but when transferring data consecutively, a wait for a fixed time period should be inserted after each send.

If this sample program is used with the previous “Serial Communications (Receiving Side)” sample program, data overflows will occur and smooth communications will not be possible.

• GHEAD.ASM

```
001 chip      LC868700
002 world     external
003 ; *-----*
004 ; * External header program Ver 1.00          *
005 ; *                      05/20-'98          *
006 ; *-----*
007
008 public  fm_wrt_ex_exit, fm_vrf_ex_exit
009 public  fm_prd_ex_exit, timer_ex_exit, _game_start, _game_end
010 other_side_symbol  fm_wrt_in, fm_vrf_in
011 other_side_symbol  fm_prd_in, timer_in, game_end
012
013 extern  main                ; Symbol in the user program
014 extern  int_BaseTimer       ; Symbol in the user program
015 extern  int_SioRx           ; Symbol in the user program
016
017 ; *-----*
018 ; * Vector table(?)          *
019 ; *-----*
020 cseg
021 org 0000h
022 _game_start:
023 ;reset:
024         jmpf    main        ; main program jump
025 org 0003h
026 ;int_03:
027         jmp int_03
028 org 000bh
029 ;int_0b:
030         jmp int_0b
031 org 0013h
```



```

032 ;int_13:
033     jmp int_13
034 org 001bh
035 ;int_1b:
036     jmp int_1b
037 org 0023h
038 ;int_23:
039     jmp int_23
040 org 002bh
041 ;int_2b:
042     jmp int_2b
043 org 0033h
044 ;int_33:
045     jmp int_33
046 org 003bh
047 ;int_3b:
048     jmp int_3b
049 org 0043h
050 ;int_43:
051     jmp int_43
052 org 004bh
053 ;int_4b:
054     jmp int_4b
055 ; *-----*
056 ; * interrupt programs                *
057 ; *-----*
058 int_03:
059     reti
060 int_0b:
061     reti
062 int_13:
063     reti
064 int_23:
065     reti
066 int_2b:
067     reti
068 int_33:
069     reti
070 int_3b:
071     jmp     int_SioRx    ; SIO reception interrupt handler
072
073 ; *-----*
074 int_43:
075     reti
076 int_4b:
077     clr1    p3int,1      ; interrupt flag clear
078     reti
079
080 org 0100h
081 ; *-----*
082 ; * flash memory write external program    *
083 ; *-----*
084 fm_wrt_ex:
085     change  fm_wrt_in
086     fm_wrt_ex_exit:
087     ret
088 org 0110h
089 ; *-----*
090 ; * flash memory verify external program    *
091 ; *-----*
092 fm_vrf_ex:
093     change  fm_vrf_in
094     fm_vrf_ex_exit:
095     ret
096
097 org 0120h
098 ; *-----*
099 ; * flash memory page read external program    *
100 ; *-----*
101 fm_prd_ex:
102     change  fm_prd_in
103     fm_prd_ex_exit:
104     ret
105
106 org 0130h

```

```
107 ; *-----*
108 ; * flash memory => timer call external program *
109 ; *-----*
110 int_lb:
111 timer_ex:
112     push    ie
113     clr1    ie,7        ; interrupt prohibition
114     change  timer_in
115     timer_ex_exit:
116     call    int_BaseTimer ; (User interrupt processing)
117     pop     ie
118     reti
119
120 org 01f0h
121 _game_end:
122     change  game_end
123 end
```

• TIMER1.ASM

```
001 ; Tab width = 4
002
003 ;-----
004 ; ** Serial Communications Sample 3 (Interrupt-Driven Serial Driver with
005 ;   Reception Buffer) **
006 ;
007 ; ·Demonstrates the usage of a serial communications driver with a 16-byte
008 ;   reception buffer
009 ; ·Displays the received data values
010 ; ·Sends simple data on a regular cycle
011 ;-----
012 ; 1.01 990208 SEGA Enterprises,LTD.
013 ;-----
013 chip    LC868700        ; Specifies the chip type for the assembler
014 world    external       ; External memory program
015
016 public   main           ; Symbol referenced from ghead.asm
017 public   int_BaseTimer  ; Symbol referenced from ghead.asm
018 public   int_SioRx      ; Symbol referenced from ghead.asm
019
020 extern   _game_end      ; Application end
021
022
023 ; **** Definition of System Constants ****
024
025                                     ; OCR (Oscillation Control Register) settings
026 osc_rc      equ 04dh        ; Specifies internal RC oscillation for the system clock
027 osc_xt      equ 0efh        ; Specifies crystal oscillation for the system clock
028
029 LowBattChk   equ 06eh        ; Low battery detection flag (RAM bank 0)
030
031 SioRxCueSize equ 16         ; Serial communications buffer size
032
033
034 ; *** Data Segment ****
035
036     dseg                    ; Data segment start
037
038 r0:    ds    1             ; Indirect addressing register r0
039 r1:    ds    1             ; Indirect addressing register r1
040 r2:    ds    1             ; Indirect addressing register r2
041 r3:    ds    1             ; Indirect addressing register r3
042        ds    12            ; Other registers
043
044                                     ; ** For Serial Driver **
045 SioRxCueBehind: ds 1        ; Amount of received data waiting
046 SioRxCueRPnt:  ds 1        ; Reception buffer reading point
047 SioRxCueWPnt:  ds 1        ; Reception buffer writing point
048 SioRxCue:      ds SioRxCueSize ; Reception buffer
049 SioOverRun:    ds 1        ; Reception overrun flag
050
051                                     ; ** Work Areas for Usage Sample **
052 bcount: ds    1            ; Base clock counter
```

```

053 work1:  ds      1          ; Work 1
054 work2:  ds      1          ; Work 2
055
056 work0:  ds      1          ; Work (put2digit)
057
058
059 ; *** Code Segment *****
060
061         cseg                ; Code segment start
062
063 ; *-----*
064 ; * Serial Communications Driver Usage Sample *
065 ; * Sends simple data at a regular interval *
066 ; * Displays the received data values on the LCD *
067 ; *-----*
068 main:
069     mov     #0,bcount
070     mov     #0,work1        ; Initial value of transmission data
071     clr1    P3INT,0         ; Masks P3 interrupts
072     call    cls             ; Clears the LCD
073     call    BattChkOff      ; Turns off the low battery automatic detection function
074     call    SioInit         ; Serial communications initialization
075     bnz     finish         ; Ends if VM is not connected
076
077 stlp1:
078     ; *** Displaying If Data Has Been Received ***
079     call    SioGet1         ; 1-byte reception
080     be      #0ffh, stnx1    ; Skip if no data has been received
081     bz      stnx3           ; If normal received data is found, go to stnx3
082 error:  br      finish      ; Forcibly terminate if an error is detected
083 stnx3:
084     ld      b              ; Load received data from b -> acc
085     mov     #0,c           ; Display coordinate (horizontal)
086     mov     #0,b           ; Display coordinate (vertical)
087     call    put2digit       ; Displays numeric value on the LCD
088     br      stlp1          ; Continues to repeat as long as there is received data
089 stnx1:
090
091     set1    pcon,0         ; Waits until next interrupt
092
093     ; *** Sending Simple Data at a Regular Interval ***
094     ld      bcount         ; Base timer counter value
095     be      work2, stnx4    ; Does not send if unchanged
096     st      work2          ; Updates work2
097
098     ld      work1          ; Loads the transmission data
099     call    SioPut1         ; Sends
100
101     inc     work1           ; Updates the transmission data
102     ld      work1          ; (Sends values form 0 to 99, in sequence)
103     bne     #100, stnx2     ;
104     mov     #0,work1       ;
105 stnx2:
106 stnx4:
107
108     ; ** [M] (mode) Button Check **
109     ld      P3
110     bn      acc,6, finish   ; If the [M] button is pressed, the application ends
111
112     jmp     stlp1          ; Repeat
113
114 finish:
115     ; ** Application End Processing **
116     call    SioEnd         ; Serial communications end processing
117     call    BattChkOn      ; Turns on the low battery automatic detection function
118     jmp     _game_end      ; Application end
119 ; *****
120 ; ***** Simple Serial Communications Driver *****
121 ; *****
122
123 ; *-----*
124 ; * Serial communications initialization *
125 ; *

```

```
126 ; * This sample assumes that the system clock is in crystal mode.
127 ; *-----*
128 SioInit:
129 ; **** VM Connection Check ****
130     ld      P7                ; Checks the connection status
131     and     #%00001101        ; Checks P70, P72, P73
132     be      #%00001000,next3  ; P70 = 0, P72 = 0, P73 = 1
133 ; To next3 if connected
134     mov     #0ffh,acc         ; If not connected, abnormal end with acc = 0ffh
135     ret                                ; SioInit end
136 next3:
137
138 ; **** Serial Communications Initialization ****
139     mov     #0,SCON0          ; Specifies output as 'LSB first'
140     mov     #0,SCON1          ; Specifies input as 'LSB first'
141     mov     #0ddh,SBR         ; Sets the transfer rate
142     clr1    P1,0              ; Clears the P10 latch (P10/S00)
143     clr1    P1,2              ; Clears the P12 latch (P12/SCK0)
144     clr1    P1,3              ; Clears the P13 latch (P13/S01)
145
146     mov     #%00000101,P1FCR  ; Sets the pin functions
147     mov     #%00000101,P1DDR  ; Sets the pin functions
148
149     mov     #0,SBUF0          ; Clears the transfer buffer
150     mov     #0,SBUF1          ; Clears the transfer buffer
151
152     mov     #0,acc            ; Resets amount of received data waiting
153     st      SioRxCueBehind     ; Reception buffer reading point
154     st      SioRxCueRPnt       ; Reception buffer writing point
155     st      SioRxCueWPnt       ; Reception buffer writing point
156     st      SioOverRun        ; Resets reception overrun flag
157
158     set1    SCON1,0            ; Receiving side transfer end interrupt enable
159     set1    SCON1,3            ; Receiving standby
160
161     ret                                ; SioInit end
162
163
164 ; *-----*
165 ; * Serial Communications End
166 ; *-----*
167 SioEnd:
168 ; **** Serial Communications End Processing ****
169     mov     #0,SCON0          ; SCON0 = 0
170     mov     #0,SCON1          ; SCON1 = 0
171     mov     #0bfh,P1FCR       ; P1FCR = 0bfh
172     mov     #0a4h,P1DDR       ; P1DDR = 0a4h
173
174     ret                                ; SioEnd end
175
176
177 ; *-----*
178 ; * Sending 1 Byte
179 ; *
180 ; * Inputs: acc: Transmission data
181 ; *-----*
182 SioPut1:
183     push    acc                ; Pushes the transmission data onto the stack
184 splp1:    ld      SCON0        ; Waits until any previous transfer is completed
185     bp      acc,3,splp1        ;
186     pop     acc                ; Pops the transmission data off of the stack
187
188     st      SBUF0              ; Sets the data to be transferred
189     set1    SCON0,3            ; Starts sending
190
191     ret                                ; SioPut1 end
192
193
194 ; *-----*
195 ; * Reading 1 Byte from the Reception Buffer (Asynchronous Reception)
196 ; *
197 ; * Outputs: acc: 0 = Normal end
198 ; *-----*
```

```

198 ; *          0ffh = No received data
199 ; *          0feh = Buffer overflow
200 ; *          0fdh = Overrun error
201 ; *          b: Received data (Valid only in the case of normal end.)
202 ; *-----*
203 SioGet1:
204          ; ** Waiting Data Amount check **
205          ld      SioRxCueBehind    ; Waiting amount of data
206          bnz     sgnx1             ; When waiting amount != 0
207          mov     #0ffh,acc         ; When waiting amount == 0
208          ret                      ; Returns when acc = 0ffh (no received data)
209 sgnx1:
210          ; ** Buffer Overflow Detection **
211          ; SioRxCueBehind - SioRxCueSize
212          be      #SioRxCueSize,sgnx3 ; SioRxCueBehind == SioRxCueSize
213          bp      PSW,7,sgnx3       ; SioRxCueBehind < SioRxCueSize
214          ; SioRxCueBehind > SioRxCueSize
215          mov     #0feh,acc         ; When the buffer capacity has been exceeded
216          ret                      ; Return when acc = 0feh (buffer overflow)
217 sgnx3:
218          ; ** Overrun Error Detection **
219          ld      SioOverRun        ; Overrun flag
220          bz      sgnx4             ; Not detected
221          mov     #0fdh,acc         ; Detected
222          ret                      ; Return when acc = 0fdh (overrun error)
223 sgnx4:
224          dec     SioRxCueBehind    ; dec waiting amount
225          ; ** Calculating the received data reading point
226          ; r0 = SioRxCue + SioRxCueRPnt
227          ld      SioRxCueRPnt
228          add     #SioRxCue ;
229          st      r0
230          ;
231          inc     SioRxCueRPnt      ; inc data reading point
232          ; ** If reading point = buffer size,
233          ; ** then reading point is reset to 0
234          ld      SioRxCueRPnt
235          bne     #SioRxCueSize,sgnx2 ; When SioRxCueRPnt != SioRxCueSize
236          mov     #0,SioRxCueRPnt   ; When SioRxCueRPnt == SioRxCueSize
237          ;
238          ;
239 sgnx2:
240          ld      @r0               ; Loads the input data into acc
241          st      b                 ; Stores the value in b
242          mov     #0,acc            ; acc = acc = 0 (normal end, data exists)
243          ;
244          ret                      ; SioGet1 end
245          ;
246          ;
247          ;
248 ; *-----*
249 ; * Reading 1 Byte from the Reception Buffer
250 ; * (If there is no received data, this routine waits until data is received)
251 ; *
252 ; * Outputs: acc: 0 = Normal end
253 ; *          0feh = Buffer overflow
254 ; *          0fdh = Overrun error
255 ; *          b: Received data (Valid only in the case of normal end.)
256 ; *-----*
257 SioGet1W:
258          call    SioGet1           ; Asynchronous reception
259          be      #0ffh,SioGet1W    ; Waits until data is received
260          ret                      ; SioGet1W end
261          ;
262          ;
263          ;
264 ; *-----*
265 ; * Getting the Amount of Data Waiting in the Reception Buffer
266 ; *
267 ; * Output: acc: Amount of data (bytes)

```

268 ; *-----*

```

269 SioGetRxLen:
270     ld      SioRxCueBehind      ; Amount waiting
271
272     ret                                ; SioGetRxLen end
273
274
275 ; *-----*
276 ; * SIO Reception Interrupt Handler *
277 ; *-----*
278 int_SioRx:
279     push    acc                    ; Pushes the register to be used onto the stack
280     push    PSW                    ;
281     setl    PSW,1                  ; Selects data RAM bank 1
282     push    r0                    ; Pushes the register onto the stack
283
284                                     ; ** Calculating the Writing Point **
285     ld      SioRxCueWPnt          ; r0 = SioRxCue + SioRxCueWPnt
286     add     #SioRxCue             ;
287     st      r0                    ;
288
289     ld      SBUF1                  ; Loads the received data
290     st      @r0                    ; Writes the data to the buffer
291
292     inc     SioRxCueWPnt          ; Writing point ++
293
294                                     ; ** Resets the writing point once it
295                                     ; ** reaches the buffer size
296     ld      SioRxCueWPnt          ;
297     bne     #SioRxCueSize,isnx1   ;
298     mov     #0,SioRxCueWPnt       ;
299 isnx1:
300
301     inc     SioRxCueBehind        ; Data Waiting Amount ++
302
303     clr1    SCON1,1               ; Resets the transfer end flag
304
305                                     ; ** Checking the Overrun Error **
306     bn      SCON1,6,isnx2         ; If an overrun has not occurred, then isnx2
307     mov     #1,SioOverRun         ; If an overrun has occurred -> Sets flag
308     clr1    SCON1,6               ; Resets overrun flag
309 isnx2:
310
311     setl    SCON1,3               ; Starts the next transfer
312
313     pop     r0                    ; Pops the register to be used off of the stack
314     pop     PSW                    ;
315     pop     acc                    ;
316
317     reti                                ; int_SioRx end
318
319
320 ; *-----*
321 ; * Displaying a Two-digit Value *
322 ; *
323 ; * Inputs: acc: Numeric value *
324 ; *           c: Horizontal position of character *
325 ; *           b: Vertical position of character *
326 ; *-----*
326 put2digit:
327     push    b                      ; Pushes the coordinate data onto the stack
328     push    c                      ;
329     st      c                      ; Calculates the tens digit and the ones digit
330     xor     a                      ; ( acc = acc/10, work0 = acc mod 10 )
331     mov     #10,b                  ;
332     div     ;                      ;
333     ld      b                      ;
334     st      work0                  ; Stores the ones digit in work0
335     ld      c                      ;
336     pop     c                      ; Pops the coordinate values into (c, b)
337     pop     b                      ;
338     push    b                      ; Pushes the coordinates onto the stack again
339     push    c                      ;
340     call    putchar                ; Displays the tens digit

```

Visual Memory Tutrial Revision 1.00

```

341      ld      work0      ; Loads the ones digit
342      pop     c          ; Pops the coordinate values into (c, b)
343      pop     b          ;
344      inc     c          ; Moves the display coordinates to the right
345      call    putch      ; Displays the ones digit
346
347      ret              ; put2digit end
348
349
350 ; *-----*
351 ; * Clearing the LCD Display Image *
352 ; *-----*
353 cls:
354      push    OCR        ; Pushes the OCR value onto the stack
355      mov     #osc_rc,OCR ; Specifies the system clock
356
357      mov     #0,XBNK    ; Specifies the display RAM bank address (BANK0)
358      call    cls_s      ; Clears the data in that bank
359
360      mov     #1,XBNK    ; Specifies the display RAM bank address (BANK1)
361      call    cls_s      ; Clears the data in that bank
362      pop     OCR        ; Pops the OCR value off of the stack
363
364      ret              ; cls end
365
366 cls_s:
367      mov     #80h,r2     ; *** Clearing One Bank of Display RAM ***
368      mov     #80h,b      ; Points the indirect addressing register at the start of display RAM
369      loop3:
370      mov     #0,@r2      ; Writes "0" while incrementing the address
371      inc     r2          ;
372      dbnz    b,loop3     ; Repeats until b is "0"
373
374      ret              ; cls_s end
375
376
377 ; *-----*
378 ; * Displaying One Character in a Specified Position *
379 ; * Inputs: acc: Character code *
380 ; *          c: Horizontal position of character *
381 ; *          b: Vertical position of character *
382 ; *-----*
383 putch:
384      push    XBNK
385      push    acc
386      call    locate      ; Calculates display RAM address according to coordinates
387      pop     acc
388      call    put_chara   ; Displays one character
389      pop     XBNK
390
391      ret              ; putch end
392
393
394 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification ****
395      ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
396      ; ** Outputs: r2: RAM address XBNK: Display RAM bank
397
398      ; *** Determining the Display RAM Bank Address ***
399      ld      b          ; Jump to next1 when b >= 2
400      sub     #2
401      bn      PSW,7,next1 ;
402
403      mov     #00h,XBNK   ; Specifies the display RAM bank address (BANK0)
404      br      next2
405 next1:
406      st      b
407      mov     #01h,XBNK   ; Specifies the display RAM bank address (BANK1)
408 next2:
409
410      ; *** Calculating the RAM Address for a Specified Position on the
411      Display ***
412      ld      b          ; b * 40h + c + 80h
413      rol     ;

```



```

413         rol                ;
414         rol                ;
415         rol                ;
416         rol                ;
417         rol                ;
418         add     c           ;
419         add     #80h        ;
420         st      r2          ; Stores the RAM address in r2
421
422         ret                ; locate end
423
424
425 put_chara:
426     push     PSW           ; Pushes the PSW value onto the stack
427     setl     PSW,1         ; Selects data RAM bank 1
428
429                                     ; *** Calculating the Character Data Address ***
430         rol                ; (TRH,TRL) = acc*8 + fontdata
431         rol                ;
432         rol                ;
433         add     #low(fontdata) ;
434         st      TRL         ;
435         mov     #0,acc       ;
436         addc    #high(fontdata) ;
437         st      TRH         ;
438
439     push     OCR           ; Pushes the OCR value onto the stack
440     mov     #osc_rc,OCR    ; Specifies the system clock
441
442     mov     #0,b           ; Offset value for loading the character data
443     mov     #4,c           ; Loop counter
444 loop1:
445     ld      b              ; Loads the display data for the first line
446     ldc                ;
447     inc     b              ; Increments the load data offset by 1
448     st      @r2            ; Transfers the display data to display RAM
449     ld      r2             ; Adds 6 to the display RAM address
450     add     #6             ;
451     st      r2             ;
452
453     ld      b              ; Loads the display data for the second line
454     ldc                ;
455     inc     b              ; Increments the load data offset by 1
456     st      @r2            ; Transfers the display data to display RAM
457     ld      r2             ; Adds 10 to the display RAM address
458     add     #10            ;
459     st      r2             ;
460
461     dec     c              ; Decrements the loop counter
462     ld      c              ;
463     bnz     loop1          ; Repeats for 8 lines (four times)
464
465     pop     OCR            ; Pops the OCR value off of the stack
466     pop     PSW            ; Pops the PSW value off of the stack
467
468     ret                ; put_chara end
469
470
471 ; *-----*
472 ; * Character Bit Image Data
473 ; *
474 ; *-----*
475 fontdata:
476     db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; 0
477     db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; 1
478     db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; 2
479     db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; 3
480     db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; 4
481     db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; 5
482     db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; 6
483     db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; 7
484     db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; 8
485     db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; 9

```

```

485
486
487 ; *-----*
488 ; * Low Battery Automatic Detection Function ON *
489 ; *-----*
490 BattChkOn:
491     push    PSW            ; Pushes the PSW value onto the stack
492     cmlr    PSW,1          ; Selects data RAM bank 0
493
494     mov     #0,LowBattChk   ; Detects low battery (0)
495
496     pop     PSW            ; Pops the PSW value off of the stack
497     ret                     ; BattChkOn end
498
499
500 ; *-----*
501 ; * Low Battery Automatic Detection Function OFF *
502 ; *-----*
503 BattChkOff:
504     push    PSW            ; Pushes the PSW value onto the stack
505     cmlr    PSW,1          ; Selects data RAM bank 0
506
507     mov     #0ffh,LowBattChk ; Does not detect low battery (0ffh)
508
509     pop     PSW            ; Pops the PSW value off of the stack
510     ret                     ; BattChkOff end
511
512
513 ; *-----*
514 ; * Base Timer Interrupt Handler *
515 ; *-----*
516 int_BaseTimer:
517     cmlr    btcr,1          ; Clears the base timer interrupt source
518     inc     bcount          ; Counter ++
519     ret                     ; User interrupt processing end

```

C.10 Reading and Writing Flash Memory

This sample program writes, reads, and verifies flash memory, and displays the characters “SEGA” one at a time upon the completion of each phase.

Lines 60 to 78 prepare, in RAM, the data that will be written in flash memory. The data values range from 0 to 128, and are set in addresses 10H through 8FH in bank 1 of RAM, using the indirect address register. Once the data preparation phase is completed, the program displays an “S” on the LCD.

Lines 91 to 102 set the parameters for calling system BIOS, and disable automatic low battery detection.

Lines 104 to 115 switch the system clock to 1/6 RC before calling the system BIOS. The system clock is switched back to the original clock (crystal oscillation) after the system BIOS has been called.

After switching the clock, the program enables automatic low battery detection and then displays an “E”.

Caution

Disable all interrupts, including the base timer, while flash memory is being accessed. Because the built-in clock function is used by the base timer, keep the length of time that interrupts are disabled as short as possible.

When writing to flash memory, set the system clock to 1/6 RC. When loading from flash memory, 1/12 RC is also permissible.

Lines 128 to 146 uses the system BIOS' verify function to compare the data that was written into flash memory with the data in RAM. If the data matches exactly, the program displays a “G” on the LCD. If the data does not match and the system BIOS returned an error, the program does not display a “G” but does execute the next phase.

Lines 159 to 173 load into RAM the data that was written in flash memory. The data that is loaded is verified by the program's own compare routine starting in line 172. If the data matches completely, the program displays an “A” on the LCD and then terminates. If the data does not match, the program terminates without displaying an “A”.

If the “G” or “A” is not displayed, it indicates that the data was corrupted by an earlier access to flash memory.

The data in line 357 and beyond is where the data is to be written (flash memory).

Caution

When writing to flash memory, be certain to provide an area within the application itself where the data can be written. Writing to flash memory outside of the application is prohibited.

Because flash memory accesses are always conducted in units of 128 bytes, “ORG” in line 364 aligns the data with a 128-byte boundary.

Caution

The “DS” command, an assembler pseudo-instruction, cannot be used to allocate an area in flash memory. The “DS” command can only be used for RAM areas.

```
003 ;-----
004 ; ** Flash Memory Usage Sample 1 **
005 ;
006 ; This sample writes and verifies data in flash memory, and then reads and
    verifies the data.
007 ; If all operations are performed correctly, the characters "SEGA" appear on
    the
        LCD.
008 ;-----
009 ; 1.01 990208 SEGA Enterprises,LTD.
010 ;-----
011
012 chip      Lc868700          ; Specifies the chip type for the assembler
013 world     external         ; External memory program
014
015 public    main              ; Symbol referenced from ghead.asm
016
017 extern    _game_end         ; Symbol reference to ghead.asm
018 extern    fm_wrt_ex, fm_vrf_ex, fm_prd_ex ; Symbol reference to ghead.asm
019
020
021 ; **** Definition of System Constants ****
022
023          ; OCR (Oscillation Control Register) settings
024 osc_rc     equ 04dh          ; Specifies internal RC oscillation for the system clock (1/12)
025 osc_rcfw   equ 0cdh          ; Specifies internal RC oscillation for the system clock (1/6)
026 osc_xt     equ 0efh          ; Specifies crystal oscillation for the system clock
027
028 LowBattChk equ 06eh          ; Low battery detection flag (RAM bank 0)
029
030 fmflag     equ 07ch          ; Flash memory write end detection method
031 fmbank     equ 07dh          ; Flash memory bank switching
032 fmadd_h    equ 07eh          ; Flash memory upper address
033 fmadd_l    equ 07fh          ; Flash memory lower address
034
035 fmbuff     equ 080h          ; Start of buffer for flash memory reading/writing
036
037 ; *** Data Segment ****
038
039          dseg                ; Data segment start
040
041 r0:        ds      1          ; Indirect addressing register r0
042 r1:        ds      1          ; Indirect addressing register r1
043 r2:        ds      1          ; Indirect addressing register r2
044 r3:        ds      1          ; Indirect addressing register r3
045          ds      12          ; Other registers reserved for the system
046
047 ; *** Code Segment ****
048
049          cseg                ; Code segment start
050
051 ; *-----*
052 ; * User program
    *
053 ; *-----*
054 main:
055          call      cls          ; Clears the LCD display image
056
057
058          ; Preparing Data for the Test Write
059          ; Prepares 128 bytes of data from 10h to 8fh in fmbuff
060
061          push     PSW          ; Pushes the PSW value onto the stack
062          setl     PSW,1        ; Selects data RAM bank 1
063
064          mov      #fmbuff,r0   ; Moves the read/write buffer address to r0
065          mov      #128,c        ; Loop counter (128 times)
066          mov      #010h,b      ; Initial value of data to be written
067 loop4:
068          ld       b            ; Places the data in the buffer
069          st       @r0          ;
070
071          inc      b            ; Changes the writing test data
072
073          inc      r0           ; Increments the buffer address
```

```

074
075      dec     c           ; Decrements the loop counter
076      ld      c
077      bnz     loop4       ; Repeats 128 times
078
079      pop      PSW        ; Pops the PSW value off of the stack
080
081
082                        ; Displaying "S"
083
084      mov      #1,c        ; Horizontal coordinate
085      mov      #1,b        ; Vertical coordinate
086      mov      #0ah,acc    ; Character code 'S'
087      call     putch       ; Displays a single character
088
089
090                        ; **** Writing to Flash Memory ****
091
092      push     PSW        ; Pushes the PSW value onto the stack
093      setl     PSW,1       ; Selects data RAM bank 1
094
095      mov      #0,fmbank   ; Flash memory bank specification = 0
096      mov      #high(fmarea),fmadd_h ; Writing destination address (upper)
097      mov      #low(fmarea),fmadd_l ; Writing destination address (lower)
098
099      clrl     PSW,1       ; Selects data RAM bank 0
100      mov      #0ffh,acc   ; Does not detect low battery (0ffh)
101      st       LowBattChk  ; Low battery automatic detection flag (RAM bank 0)
102
103      pop      PSW        ; Pops the PSW value off of the stack
104
105      push     OCR         ; Pushes the OCR value onto the stack
106      mov      #osc_rc,OCR ; Specifies the system clock (RC)
107      call     fm_wrt_ex   ; BIOS "Writing to flash memory"
108      pop      OCR         ; Pops the OCR value off of the stack
109
110      push     PSW        ; Pushes the PSW value onto the stack
111      clrl     PSW,1       ; Selects data RAM bank 0
112      mov      #0,acc      ; Detects low battery (0)
113      st       LowBattChk  ; Low battery automatic detection flag (RAM bank 0)
114      pop      PSW        ; Pops the PSW value off of the stack
115
116
117                        ; **** Displaying "E" ****
118
119      mov      #2,c        ; Horizontal coordinate
120      mov      #1,b        ; Vertical coordinate
121      mov      #0bh,acc    ; Character code 'E'
122      call     putch       ; Displays a single character
123
124
125                        ; **** Verifying Flash Memory ****
126
127      push     PSW        ; Pushes the PSW value onto the stack
128      setl     PSW,1       ; Selects data RAM bank 1
129
130      mov      #0,fmbank   ; Flash memory bank specification = 0
131      mov      #high(fmarea),fmadd_h ; Address (upper)
132      mov      #low(fmarea),fmadd_l ; Address (lower)
133
134
135      push     OCR         ; Pushes the OCR value onto the stack
136      mov      #osc_rc,OCR ; Specifies the system clock (RC)
137      call     fm_vrf_ex   ; BIOS "Verifying flash memory"
138      pop      OCR         ; Pops the OCR value off of the stack
139
140      pop      PSW        ; Pops the PSW value off of the stack
141
142      bnz      vrt_bad     ; Branches when write failed
143                        ; Displays "G" only when successful
144
145                        ; **** Displaying "G" ****
146
147      mov      #3,c        ; Horizontal coordinate
148      mov      #1,b        ; Vertical coordinate

```

```
149     mov     #0ch,acc    ; Character code 'G'
150     call    putchar     ; Displays a single character
151 vrt_bad:
152
153
154                                     ; **** Reading Page Data form Flash Memory ****
155
156     push    PSW         ; Pushes the PSW value onto the stack
157     setl    PSW,1       ; Selects data RAM bank 1
158
159     mov     #0,fmbank    ; Flash memory bank specification = 0
160     mov     #high(fmarea),fmadd_h ; Address (upper)
161     mov     #low(fmarea),fmadd_l ; Address (lower)
162
163     push    OCR         ; Pushes the OCR value onto the stack
164     mov     #osc_rc,OCR  ; Specifies the system clock (RC)
165     call    fm_prd_ex    ; BIOS "Reading page data from flash memory"
166     pop     OCR         ; Pops the OCR value off of the stack
167
168     pop     PSW         ; Pops the PSW value off of the stack
169
170
171                                     ; **** Verifying the data that was read ****
172
173     push    PSW         ; Pushes the PSW value onto the stack
174     setl    PSW,1       ; Selects data RAM bank 1
175
176     mov     #fmbuff,r0   ; Moves the read/write buffer address into r0
177     mov     #128,c       ; Loop counter (128 times)
178     mov     #010h,b      ; Initial value for comparison data
179 loop5:
180     ld      b            ; Places the data in the buffer
181     sub     @r0          ; Compares the data
182     bnz     read_bad     ; If a compare error is found, ends without displaying 'A'
183
184     inc     b            ; Changes the data for the write test
185
186     inc     r0           ; Increments the buffer address
187
188     dec     c            ; Decrements the loop counter
189     ld      c
190     bnz     loop5        ; Repeats 128 times
191
192     pop     PSW         ; Pops the PSW value off of the stack
193
194
195                                     ; **** Displaying "A" ****
196
197     mov     #4,c         ; Horizontal coordinate
198     mov     #1,b         ; Vertical coordinate
199     mov     #0dh,acc     ; Character code 'A'
200     call    putchar     ; Displays a single character
201
202
203 read_bad:
204 loop6:                                     ; ** [M] (mode) Button Check **
205     ld      P3
206     bn      acc,6,finish ; If the [M] button is pressed, the application ends
207
208     br      loop6        ; Repeat
209
210 finish:                                     ; ** Application End Processing **
211     jmp     _game_end    ; Application end
212
213
214 ; -----*
215 ; * Clearing the LCD Display Image *
216 ; -----*
217 cls:
218     push    OCR         ; Pushes the OCR value onto the stack
219     mov     #osc_rc,OCR  ; Specifies the system clock
220
221     mov     #0,XBNK     ; Specifies the display RAM bank address (BANK0)
222     call    cls_s       ; Clears the data in that bank
223
```

```

224      mov     #1,XBNK      ; Specifies the display RAM bank address (BANK1)
225      call   cls_s        ; Clears the data in that bank
226      pop     OCR         ; Pops the OCR value off of the stack
227
228      ret                      ; cls end
229
230 cls_s:                      ; *** Clearing One Bank of Display RAM ***
231      mov     #80h,r2      ; Points the indirect addressing register at the start of display RAM
232      mov     #80h,b       ; Sets the number of loops in loop counter b
233 loop3:
234      mov     #0,@r2       ; Writes "0" while incrementing the address
235      inc     r2           ;
236      dbnz    b,loop3      ; Repeats until b is "0"
237
238      ret                      ; cls_s end
239
240
241 ; *-----*
242 ; * Displaying One Character in a Specified Position
243 ; *
244 ; * Inputs: acc: Character code
245 ; *          c: Horizontal position of character
246 ; *          b: Vertical position of character
247 ; *-----*
248      push    XBNK
249      push    acc
250      call   locate        ; Calculates display RAM address according to coordinates
251      pop     acc
252      call   put_chara     ; Displays one character
253      pop     XBNK
254
255      ret                      ; putch end
256
257
258 locate: ; **** Calculating the Display RAM Address According to the Display Position Specification
259 ****
260 ; ** Inputs: c: Horizontal position (0 to 5) b: Vertical position (0 to 3)
261 ; ** Outputs: r2: RAM address XBNK: Display RAM bank
262
263      ld      b             ; *** Determining the Display RAM Bank Address ***
264      sub     #2            ; Jump to next1 when b >= 2
265      bn      PSW,7,next1   ;
266
267      mov     #00h,XBNK     ; Specifies the display RAM bank address (BANK0)
268      br      next2
269 next1:
270      st      b
271      mov     #01h,XBNK     ; Specifies the display RAM bank address (BANK1)
272 next2:
273
274      ; *** Calculating the RAM Address for a Specified Position on the
275      Display ***
276      ld      b             ; b * 40h + c + 80h
277      rol
278      rol
279      rol
280      rol
281      rol
282      add     c             ;
283      add     #80h          ;
284      st      r2            ; Stores the RAM address in r2
285
286      ret                      ; locate end
287
288
289      push    PSW           ; Pushes the PSW value onto the stack
290      set1    PSW,1         ; Selects data RAM bank 1
291
292      ; *** Calculating the Character Data Address ***
293      ; (TRH,TRL) = acc*8 + fontdata
294      rol

```

```
295     rol                ;
296     rol                ;
297     add    #low(fontdata) ;
298     st      TRL        ;
299     mov     #0,acc      ;
300     addc    #high(fontdata) ;
301     st      TRH        ;
302
303     push    OCR         ; Pushes the OCR value onto the stack
304     mov     #osc_rc,OCR ; Specifies the system clock
305
306     mov     #0,b        ; Offset value for loading the character data
307     mov     #4,c        ; Loop counter
308 loop1:
309     ld      b           ; Loads the display data for the first line
310     ldc                ;
311     inc     b           ; Increments the load data offset by 1
312     st      @r2         ; Transfers the display data to display RAM
313     ld      r2          ; Adds 6 to the display RAM address
314     add     #6          ;
315     st      r2          ;
316
317     ld      b           ; Loads the display data for the second line
318     ldc                ;
319     inc     b           ; Increments the load data offset by 1
320     st      @r2         ; Transfers the display data to display RAM
321     ld      r2          ; Adds 10 to the display RAM address
322     add     #10         ;
323     st      r2          ;
324
325     dec     c           ; Decrements the loop counter
326     ld      c           ;
327     bnz     loop1       ; Repeats for 8 lines (four times)
328
329     pop     OCR         ; Pops the OCR value off of the stack
330     pop     PSW         ; Pops the PSW value off of the stack
331
332     ret                ; put_chara end
333
334 ; *-----*
335 ; * Character Bit Image Data
336 ; *
337 ; *-----*
338 fontdata:
339     db 07ch, 0e6h, 0c6h, 0c6h, 0c6h, 0ceh, 07ch, 000h ; '0' 00
340     db 018h, 038h, 018h, 018h, 018h, 018h, 03ch, 000h ; '1' 01
341     db 07ch, 0c6h, 0c6h, 00ch, 038h, 060h, 0feh, 000h ; '2' 02
342     db 07ch, 0e6h, 006h, 01ch, 006h, 0e6h, 07ch, 000h ; '3' 03
343     db 00ch, 01ch, 03ch, 06ch, 0cch, 0feh, 00ch, 000h ; '4' 04
344     db 0feh, 0c0h, 0fch, 006h, 006h, 0c6h, 07ch, 000h ; '5' 05
345     db 01ch, 030h, 060h, 0fch, 0c6h, 0c6h, 07ch, 000h ; '6' 06
346     db 0feh, 0c6h, 004h, 00ch, 018h, 018h, 038h, 000h ; '7' 07
347     db 07ch, 0c6h, 0c6h, 07ch, 0c6h, 0c6h, 07ch, 000h ; '8' 08
348     db 07ch, 0c6h, 0c6h, 07eh, 006h, 00ch, 078h, 000h ; '9' 09
349
350     db 07ch, 0e6h, 076h, 038h, 0dch, 0ceh, 07ch, 000h ; 'S' 0a
351     db 0feh, 0c0h, 0c0h, 0f8h, 0c0h, 0c0h, 0feh, 000h ; 'E' 0b
352     db 07ch, 0e6h, 0c0h, 0dch, 0c6h, 0e6h, 07ch, 000h ; 'G' 0c
353     db 01eh, 036h, 066h, 0c6h, 0c6h, 0feh, 0c6h, 000h ; 'A' 0d
354
355 ; *-----*
356 ; * Flash Memory Area for Saving Data
357 ; *
358 ; *-----*
359     org ((*-1) land 0ff80h) + 80h ; Aligns with 128-byte boundary
360     fmarea:
361     ; Allocates a 128-byte flash memory area
362     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
363     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
364     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
365     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
366     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
367     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```



```

368          db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
369          db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
370
371      end

```

C.11 Low Battery Detection and Saving Data

Visual Memory has a built-in function that automatically detects the low battery condition, displays a message to that effect, and then puts the unit into sleep mode. In this sample program, the application detects the low battery condition on its own, and then saves the data in RAM to flash memory.

The important portion of this program is the low battery detection routine in lines 115 to 125. This routine checks the port 7 low battery flag.

Although the port 7 interrupt could be used, the interrupt processing routine should be designed so that system BIOS is not called.

```

001 ; Tab width = 4
002
003 ;-----
004 ; ** Low Battery Detection and Data Save Sample 1 **
005 ;
006 ; Detects the low battery condition and saves essential data in flash memory
007 ;-----
008 ; 1.01 990208 SEGA Enterprises,LTD.
009 ;-----
010
011 chip      Lc868700          ; Specifies the chip type for the assembler
012 world     external         ; External memory program
013
014 public    main              ; Symbol referenced from ghead.asm
015
016 extern    _game_end         ; Symbol reference to ghead.asm
017 extern    fm_wrt_ex, fm_vrf_ex, fm_prd_ex ; Symbol reference to ghead.asm
018
019
020 ; **** Definition of System Constants ****
021
022          ; OCR (Oscillation Control Register) settings
023 osc_rc     equ 04dh          ; Specifies internal RC oscillation for the system clock (1/12)
024 osc_rcfw   equ 0cdh          ; Specifies internal RC oscillation for the system clock (1/6)
025 osc_xt     equ 0efh          ; Specifies crystal oscillation for the system clock
026
027 LowBattChk equ 06eh          ; Low battery detection flag (RAM bank 0)
028
029 fmflag     equ 07ch          ; Flash memory write end detection method
030 fmbank     equ 07dh          ; Flash memory bank switching
031 fmadd_h    equ 07eh          ; Flash memory upper address
032 fmadd_l    equ 07fh          ; Flash memory lower address
033
034 fmbuff     equ 080h          ; Start of buffer for flash memory reading/writing
035
036 ; *** Data Segment ****
037
038          dseg                ; Data segment start
039
040 r0:        ds 1              ; Indirect addressing register r0
041 r1:        ds 1              ; Indirect addressing register r1
042 r2:        ds 1              ; Indirect addressing register r2
043 r3:        ds 1              ; Indirect addressing register r3
044          ds 12              ; Other registers reserved for the system
045
046
047 ; *** Code Segment ****

```

```
048
049      cseg                      ; Code segment start
050
051 ; *-----*
052 ; * User program
053 ; *-----*
054 main:
055      call    cls                ; Clears the LCD display image
056
057 loop0:                      ; Start of test main loop
058
059      ; Application Main Processing
060
061      ; ** [M] (mode) Button Check **
062      ld      P3
063      bn      acc,6,finish      ; If the [M] button is pressed, the application ends
064
065      ; ** Battery Status Check **
066      call    ChkBatt           ; Checks the battery status
067      bz      loop0             ; If acc = 0 then battery normal; loops
068
069      ; ** Low Battery Processing **
070      call    prepare           ; Prepares data for test save
071      ; In an actual application, this routine would gather the data
072      ; that is to be saved and then place the data
073      ; in the flash ROM write buffer.
074
075      call    WriteData         ; Writes the data that was prepared in the buffer (to be saved)
076      ; to flash memory
077
078 finish:                      ; ** Application End Processing **
079      jmp     _game_end         ; Application end
080
081
082 ; *-----*
083
084 prepare:                      ; **** Preparing Data for Test Save ****
085      ; Prepares 128 bytes of data from 10h to 8fh in fmbuff
086
087      push    PSW               ; Pushes the PSW value onto the stack
088      setl    PSW,1             ; Selects data RAM bank 1
089
090      mov     #fmbuff,r0        ; Moves the read/write buffer address to r0
091      mov     #128,c             ; Loop counter (128 times)
092      mov     #010h,b           ; Initial value of data to be written
093 loop4:
094      ld      b                 ; Places the data in the buffer
095      st      @r0               ;
096
097      inc     b                 ; Changes the writing test data
098
099      inc     r0                ; Increments the buffer address
100
101      dec     c                 ; Decrements the loop counter
102      ld      c
103      bnz     loop4             ; Repeats 128 times
104
105      pop     PSW               ; Pops the PSW value off of the stack
106
107      ret                      ; prepare end
108
109
110 ; *-----*
111 ; * Detecting Low Battery Status
112 ; *
113 ; * Outputs:      acc = 0      : Battery status normal
114 ; *
115 ; *              acc = 0ffh: Low battery
116 ; *-----*
117 ChkBatt:
118      ld      P7                ; Checks the status of P71
119      bn      acc,1,next3       ; Branches if there is no battery
120
121      ; ** Battery Exists **
```

```
120      mov    #0,acc      ; acc = 0
121      ret                      ; ChkBatt end. acc = 0 is returned if battery exists
122
123 next3:                      ; ** No battery **
124      mov    #0ffh,acc    ; acc = 0ffh
125      ret                      ; ChkBatt end. acc = 0ffh is returned if battery exists
126
```

```
127
128 ; *-----*
129 ; * Writing Buffer Data to Flash Memory *
130 ; *-----*
131 WriteData: ; **** Writing to Flash Memory ****
132
133     push    PSW      ; Pushes the PSW value onto the stack
134     setl    PSW,1     ; Selects data RAM bank 1
135
136     mov     #0,fmbank ; Flash memory bank specification = 0
137     mov     #high(fmarea),fmadd_h ; Writing destination address (upper)
138     mov     #low(fmarea),fmadd_l ; Writing destination address (lower)
139     mov     #0,fmflag ; Detects end by toggle bit method
140
141     clrl    PSW,1     ; Selects data RAM bank 0
142     mov     #0ffh,acc ; Does not detect low battery (0ffh)
143     st      LowBattChk ; Low battery automatic detection flag (RAM bank 0)
144
145     pop     PSW      ; Pops the PSW value off of the stack
146
147     push    OCR      ; Pushes the OCR value onto the stack
148     mov     #osc_rc,OCR ; Specifies the system clock (RC)
149     call    fm_wrt_ex ; BIOS "Writing to flash memory"
150     pop     OCR      ; Pops the OCR value off of the stack
151
152     push    PSW      ; Pushes the PSW value onto the stack
153     clrl    PSW,1     ; Selects data RAM bank 0
154     mov     #0,acc    ; Detects low battery (0)
155     st      LowBattChk ; Low battery automatic detection flag (RAM bank 0)
156     pop     PSW      ; Pops the PSW value off of the stack
157
158     ret      ; WriteData end
159
160
161 ; *-----*
162 ; * Clearing the LCD Display Image *
163 ; *-----*
164 cls:
165     push    OCR      ; Pushes the OCR value onto the stack
166     mov     #osc_rc,OCR ; Specifies the system clock *
167
168     mov     #0,XBNK  ; Specifies the display RAM bank address (BANK0)
169     call    cls_s     ; Clears the data in that bank
170
171     mov     #1,XBNK  ; Specifies the display RAM bank address (BANK1)
172     call    cls_s     ; Clears the data in that bank
173     pop     OCR      ; Pops the OCR value off of the stack
174
175     ret      ; cls end
176
177 cls_s: ; *** Clearing One Bank of Display RAM ***
178     mov     #80h,r2  ; Points the indirect addressing register at the start of display RAM
179     mov     #80h,b   ; Sets the number of loops in loop counter b
180 loop3:
181     mov     #0,@r2   ; Writes "0" while incrementing the address
182     inc     r2       ;
183     dbnz    b,loop3  ; Repeats until b is "0"
184
185     ret      ; cls_s end
186
187
188 ; *-----*
189 ; * Flash Memory Area for Saving Data *
190 ; *-----*
191     org ((*-1) land 0ff80h) + 80h ; Aligns with 128-byte boundary
192 fmarea:
193     ; Allocates a 128-byte flash memory area
194     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
195     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
196     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
197     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
198     db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

```
199      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
200      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
201      db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
202
203 end
```

index

—	
Animation icon	22, 27
Animation pattern	31
Animation speed	31
Assemble	13
ATTRIBUTE	61

—	
BACKUP UTILITY	51
Big Endian.....	73
BIOS calls	12
Body color	22, 65
Boot ROM	19, 20
Built-in icons.....	21

—	
Changing a file name	60
CHECK DISK.....	57
chip	11
Clock	12
Code segment	11
CodeScape	39, 46
COLOR A	65
Color icon data start address	26
Color icon palette data	27
Color icon pattern data	27
Command Selection Menu.....	52
Communications protocol settings.....	42
Communications software file transfer	68
Communications speed.....	42
COMPLETE	57
Conversion to a binary file.....	16
Conversion to HEX file	15
COPY	59
Copying prohibited/permitted	61
CRC.....	32
cseg.....	11

—	
Data length.....	42
Data segment	11
Data type.....	24
Date and time of initialization	66

DEFRAG	55
DELETE	60
Deleting a file	60
Door closed	45
Dreamcast startup screen.....	19
dseg	11
Dummy GD-ROM	44
DUMP	63
DUPLICATE.....	55

—	
E2H86K	15
EVA-format file.....	14
EXIT	51
external	11

—	
File Management Screen.....	22
File name	23
File operation menu	59
File Structure	27
File transfer to Visual Memory	67
Flow control	42
FORMAT	56
Fragments.....	55
FREE	52
Free space in data storage area.....	52
Free space used.....	52

—	
GAME	52
GAME BUFFER	53, 54
Game name	30, 34, 63
Game name character code table	37
GD Workshop.....	39
GD Workshop Setup	44
GDUMMY.OBJ	14
GHEAD.ASM	12
GUI Comment Data.....	30
GUI Comments.....	23

—	
H00.....	15
H2BIN	16

HEADER OFFSET	61
HEX.....	15
HyperTerminal	40

—

ICON BUFFER	53, 54
ICON NO.	64
Icon palette data	32
Icon pattern data	33
ICONDATA.VMS	25
ICONDATA_VMS.....	21
If HyperTerminal cannot be found	40
Indirect address register	11
INFO.....	53, 62
Information fork.....	18, 28, 63
Information fork structure	28
Interrupt vector definition	12

—

Jump to main routine.....	12
---------------------------	----

—

Kanji code	42
------------------	----

—

L86K.....	14
Label icon	21, 75
LC868700	11
Link.....	14
Little Endian	73
Little Endian format	73

—

M86K.....	13
Main Menu	20, 50
MAKE.....	16
mem_util.elf	39
Memory Card Utility.....	39
Memory Card Utility Operation	50
Memory Selection Menu	51
Memory Selection Screen.....	20
Monochrome icon data start address	26
Monochrome icon pattern data	26
Multiple file selection	59

—

Names of Elements of the Startup Screen	20
---	----

NORMAL BUFFER	53, 54
Number of blocks used	24
Number of Icons.....	31

—

Object file.....	13
one block	24
OPERATION	65
org.....	12

—

Parity check.....	42
-------------------	----

—

QUICK	57
-------------	----

—

RC oscillation	12
Reading/writing flash memory	12
RECV DATA	53
RENAME	60
RS-232C reverse cable	39

—

SAVE DATA.....	54
Save time.....	24
Sort key	30, 34
Stop bit length	42
SYSTEM CONFIG	51

—

Transfer to Visual Memory	18
Transferring a file to a PC.....	62
Type A	31, 33
Type B	31, 33
Type C	31, 34

—

UNFORMAT	56
Unformatted memory	20
Unformatted Visual Memory	56
UPLOAD.....	62
Usage of the application area.....	52
USED	52

—

Visual comment.....	24
Visual comment data	33

Visual comment data structure33

Visual comment type31

Visual Memory20

Visual Memory formatting56, 64

Visual Memory Simulator16

Visual type31

VM Comment Data25, 29

VM Comments24

Voice recognition device20

Volume icon21

world.....11

Xmodem.....68