

## *Table of Contents*

### *BOOK I*

<b>SUBJECT</b>	<b>PAGE</b>
<b>PREFACE .....</b>	1
<b>SECTION 1 - APPROVAL PROCESS .....</b>	<b>1-1-1</b>
NOA Licensed Software Approval Process.....	1-1-1
Super NES Software Submission Requirements .....	1-2-1
<b>SECTION 2 - SUPER NES SOFTWARE .....</b>	<b>2-1-1</b>
Introduction.....	2-1-1
Object (OBJ) .....	2-2-1
Background (BG).....	2-3-1
Mosaic.....	2-4-1
Rotation/Enlargement/Reduction.....	2-5-1
Window (Window Mask) .....	2-6-1
Main/Sub Screen .....	2-7-1
CG Direct Select .....	2-8-1
H-Pseudo 512.....	2-9-1
Complementary Multiplication (Signed Multiplication) .....	2-10-1
H/V Counter Latch.....	2-11-1
Offset Change .....	2-12-1
Standard Controller .....	2-13-1
Programmable I/O Port .....	2-14-1
Absolute Multiplication/Division .....	2-15-1
H/V Count Timer .....	2-16-1
Direct Memory Access (DMA) .....	2-17-1
Interlace.....	2-18-1
H-512 Mode (BG Mode 5 & 6) .....	2-19-1
OBJ 33's Lines Over & Priority Order .....	2-20-1
CPU Clock and Address Map .....	2-21-1
Super NES Functional Operation.....	2-22-1
System Flowchart.....	2-23-1
Programming Cautions .....	2-24-1
Documented Problems .....	2-25-1
Register Clear (Initial Settings) .....	2-26-1
PPU Registers .....	2-27-1
CPU Registers .....	2-28-1

**TABLE of CONTENTS**

*Table of Contents (Continued)*

<b>SUBJECT</b>	<b>PAGE</b>
<b>SECTION 3 - SUPER NES SOUND .....</b>	3-1-1
SNES Sound Source Outline .....	3-1-1
BRR (Bit Rate Reduction) .....	3-2-1
I/O Ports .....	3-3-1
Control Register .....	3-4-1
Timers .....	3-5-1
DSP Interface Register.....	3-6-1
Register Used .....	3-7-1
CPU Organization .....	3-8-1
Sound Programming Cautions .....	3-9-1
<b>SECTION 4 - SUPER NES CPU DATA .....</b>	4-1-1
Outline.....	4-1-1
Explanation of CPU Terminal Functions.....	4-2-1
Explanation of Functions .....	4-3-1
Addressing Mode .....	4-4-1
Command Set (Alphabetical Order) .....	4-5-1
Command Set (Matrix Display).....	4-6-1
Cycles and Bytes of Addressing Modes .....	4-7-1
Differences Among 65C816, 65C02, and 6502.....	4-8-1
Restrictions Upon Use & Application Information .....	4-9-1
Details of Command Functions .....	4-10-1
Description of Commands .....	4-11-1
AC Characteristics .....	4-12-1
<b>TABLES OF APPENDIX</b>	
Appendix A - PPU Registers .....	A-1
Appendix B - CPU Registers .....	B-1
Appendix C - SPC700 Commands .....	C-1
Appendix D - Data Transfer Procedure .....	D-1

**INDEX**

**TABLE of CONTENTS**

*Table of Contents* (Continued)

***BOOK II***

<b>SUBJECT</b>	<b>PAGE</b>
<b>SECTION 1 - SUPER ACCELERATOR (SA-1).....</b>	<b>1-1-1</b>
Super Accelerator System Functions .....	1-1-1
Configuration of SA-1 .....	1-2-1
Super Accelerator Memory Map .....	1-3-1
SA-1 Internal Register Configuration.....	1-4-1
Multi-Processor Processing .....	1-5-1
Character Conversion.....	1-6-1
Arithmetic Function .....	1-7-1
Variable-Length Bit Processing.....	1-8-1
DMA .....	1-9-1
SA-1 Timer .....	1-10-1
<b>SECTION 2 - SUPER FX® .....</b>	<b>2-1-1</b>
Introduction to Super FX .....	2-1-1
GSU Functional Operation .....	2-2-1
Memory Mapping .....	2-3-1
GSU Internal Register Configuration .....	2-4-1
GSU Program Execution.....	2-5-1
Instruction Execution .....	2-6-1
Data Access.....	2-7-1
GSU Special Functions .....	2-8-1
Description of Instructions.....	2-9-1
<b>SECTION 3 - DSP/DSP1.....</b>	<b>3-1-1</b>
Introduction to DSP1 .....	3-1-1
Command Summary .....	3-2-1
Parameter Data Type.....	3-3-1
Use of DSP1.....	3-4-1
Description of DSP1 Commands.....	3-5-1
Math Functions and Equations.....	3-6-1

**TABLE of CONTENTS**

*Table of Contents* (Continued)

<b>SUBJECT</b>	<b>PAGE</b>
<b>SECTION 4 - ACCESSORIES</b> .....	4-1-1
The Super NES Super Scope® System .....	4-1-1
Principles of the Super NES Super Scope .....	4-2-1
Super NES Super Scope Functional Operation .....	4-3-1
Super NES Super Scope Receiver Functions.....	4-4-1
Graphics .....	4-5-1
Super NES Mouse Specifications .....	4-6-1
Using the Standard BIOS.....	4-7-1
Programming Cautions .....	4-8-1
MultiPlayer 5 Specifications.....	4-9-1
MultiPlayer 5 Supplied BIOS .....	4-10-1

**SUPPLEMENTAL INFORMATION**

Super NES Parts List .....	1
Game Content Guidelines .....	3
Guidelines Concerning Commercialism and Promotion of Licensee	
Products or Services in Nintendo Licensed Games .....	5
Super NES Video Timing Information .....	10

**INDEX**

**BULLETINS**

## *List of Figures*

### *BOOK I*

<u><b>TITLE</b></u>	<u><b>FIGURE NUMBER</b></u>	<u><b>PAGE</b></u>
Software Approval Process.....	1-1-1 .....	1-1-1
Picture Image .....	2-1-1 .....	2-1-1
Scanning Pattern for Interlace.....	2-1-2 .....	2-1-2
Super NES Functional Block Diagram.....	2-22-1 .....	2-22-1
System Block Diagram .....	3-1-1 .....	3-1-2
Memory Map .....	3-1-2 .....	3-1-3
Sound Signal Flow .....	3-1-3 .....	3-1-4
BRR Data String .....	3-2-1 .....	3-2-1
BRR Range Data.....	3-2-2 .....	3-2-2
Example Data When Filter = 0 .....	3-2-3 .....	3-2-3
I/O Diagram .....	3-3-1 .....	3-3-2
Port Clear .....	3-4-1 .....	3-4-1
Clear Timing .....	3-4-2 .....	3-4-1
Timer Control.....	3-4-3 .....	3-4-2
Timer Section.....	3-5-1 .....	3-5-1
4-Bit Counter .....	3-5-2 .....	3-5-2
Timing.....	3-5-3 .....	3-5-2
Timer Related Registers.....	3-5-4 .....	3-5-3
Interface Register .....	3-6-1 .....	3-6-1
Interface Register Flow.....	3-6-2 .....	3-6-1
Bent Line Mode .....	3-7-1 .....	3-7-4
CPU Registers.....	3-8-1 .....	3-8-3
Boolean Bit Operation Commands .....	3-8-2 .....	3-8-8
Memory Access Addressing Effective Address .....	3-8-3 .....	3-8-9
Wave-form Overflow.....	3-9-1 .....	3-9-1
SNES CPU Block Diagram .....	4-1-1 .....	4-1-2
SNES CPU Terminal Interface Diagram .....	4-1-2 .....	4-1-3
SNES CPU Timing Chart .....	4-12-1 .....	4-12-2
S-PPU Main/Sub Screen Window .....	A-1 .....	A-23

*List of Figures* (Continued)***BOOK II***

<b>TITLE</b>	<b>FIGURE NUMBER</b>	<b>PAGE</b>
Super Accelerator System Configuration .....	1-1-1 .....	1-1-3
SAS Bus Image .....	1-1-2 .....	1-1-4
SA-1 Block Diagram.....	1-2-1 .....	1-2-1
Bitmap Register Files 0~7 .....	1-4-1 .....	1-4-24
Bitmap Register Files 8~F .....	1-4-2 .....	1-4-25
Accelerator Mode.....	1-5-1 .....	1-5-6
Parallel Processing Mode.....	1-5-2 .....	1-5-7
Mixed Processing Mode .....	1-5-3 .....	1-5-8
Character Conversion 1.....	1-6-1 .....	1-6-1
Character Conversion 2.....	1-6-2 .....	1-6-2
Compressed Bitmap Data .....	1-6-3 .....	1-6-3
Bitmap Image Projection .....	1-6-4 .....	1-6-3
Bitmap Data Expansion .....	1-6-5 .....	1-6-5
Memory Addresses for the Bitmap Area .....	1-6-6 .....	1-6-6
Character Conversion Buffers.....	1-6-7 .....	1-6-7
Fixed Mode Process Flow Diagram.....	1-8-1 .....	1-8-2
Auto-increment Mode Process Flow Diagram .....	1-8-2 .....	1-8-3
Barrel Shift Process.....	1-8-3 .....	1-8-5
Normal DMA .....	1-9-1 .....	1-9-1
Character Conversion DMA .....	1-9-2 .....	1-9-1
Super FX System Configuration.....	2-1-1 .....	2-1-3
Game Pak ROM/RAM Bus Diagram .....	2-1-2 .....	2-1-4
GSU Functional Block Diagram.....	2-2-1 .....	2-2-1
Super NES CPU Memory Map.....	2-3-1 .....	2-3-2
Super FX Memory Map .....	2-3-2 .....	2-3-4
Example of General Register .....	2-4-1 .....	2-4-2
128 Dot High BG Character Array .....	2-8-1 .....	2-8-2
160 Dot High BG Character Array .....	2-8-2 .....	2-8-2
192 Dot High BG Character Array .....	2-8-3 .....	2-8-2
OBJ Character Array.....	2-8-4 .....	2-8-3
Plot Operations Assigned by CMODE .....	2-8-5 .....	2-8-13
System Block Diagram (DSP1) .....	3-1-1 .....	3-1-2
Super NES CPU and DSP1 Communications .....	3-1-2 .....	3-1-3
DSP1 Command Execution .....	3-1-3 .....	3-1-3
Mode 20/DSP Memory Map.....	3-1-4 .....	3-1-4
Mode 21/DSP Memory Map.....	3-1-5 .....	3-1-5
Super NES/DSP1 Memory Mapping (Mode 21).....	3-4-1 .....	3-4-1
DSP1 Status Register Configuration.....	3-4-2 .....	3-4-2

*List of Figures* (Continued)

<b>TITLE</b>	<b>FIGURE NUMBER</b>	<b>PAGE</b>
DSP1 Operations Flow Diagram .....	3-4-3	3-4-3
Super NES CPU/DSP1 Operational Timing.....	3-4-4	3-4-4
Trigonometric Calculation.....	3-5-1	3-5-3
Vector Calculation .....	3-5-2	3-5-4
Vector Size Comparison .....	3-5-3	3-5-6
Vector Absolute Value Calculation .....	3-5-4	3-5-7
Two-Dimensional Coordinate Rotation.....	3-5-5	3-5-8
Examples of Three-Dimensional Rotation.....	3-5-6	3-5-11
Assignment of Projection Parameter .....	3-5-7	3-5-13
Relationship of Sight and Projected Plane.....	3-5-8	3-5-13
Calculation of Raster Data.....	3-5-9	3-5-16
BG Screen and Displayed Area .....	3-5-10	3-5-16
Calculation of Projected Position of Object.....	3-5-11	3-5-18
Projection Image of Object .....	3-5-12	3-5-19
Calculation of Coordinates for the Indicated Point on the Screen.....	3-5-13	3-5-20
Attack Point and Position Indicated on Screen (Side View) .....	3-5-14	3-5-21
Attitude Computation .....	3-5-15	3-5-23
Object Coordinate Rotated on Y Axis .....	3-5-16	3-5-23
Object Coordinate Rotated on X Axis .....	3-5-17	3-5-23
Object Coordinate Rotated on Z Axis.....	3-5-18	3-5-23
Conversion of Global to Objective Coordinates .....	3-5-19	3-5-26
Conversion of Object to Global Coordinates.....	3-5-20	3-5-28
Calculation of Inner Product with Forward Attitude .....	3-5-21	3-5-29
Position of Aircraft and Vector Code .....	3-5-22	3-5-30
Calculation of Rotation Angle After Attitude Change .....	3-5-23	3-5-32
Signal Flow .....	4-1-1	4-1-1
Optical Alignment.....	4-1-2	4-1-2
Virtual Screen Alignment .....	4-1-3	4-1-2
Address and Bit Assignments .....	4-1-4	4-1-5
Picture Tube .....	4-2-1	4-2-1
Scanning.....	4-2-2	4-2-2
Area Seen by Super NES Super Scope .....	4-2-3	4-2-3
Vertical Positioning .....	4-2-4	4-2-4
Horizontal Positioning .....	4-2-5	4-2-5
Horizontal/Vertical Counter.....	4-2-6	4-2-6
Super NES Super Scope Block Diagram.....	4-3-1	4-3-2
Super NES Super Scope Flow Diagram .....	4-3-2	4-3-3
Raster Signal .....	4-3-3	4-3-4
Definition of One Bit .....	4-3-4	4-3-5
Output Signal Code.....	4-3-5	4-3-5
Definitions of Codes .....	4-3-6	4-3-6

*List of Figures* (Continued)

<b>TITLE</b>	<b>FIGURE NUMBER</b>	<b>PAGE</b>
Raster Signal Transmission Timing .....	4-3-7 .....	4-3-7
Receiver Block Diagram .....	4-4-1 .....	4-4-1
Operation Flow Diagram .....	4-4-2 .....	4-4-2
Receiver/Transmitter Interface Schematic .....	4-4-3 .....	4-4-3
One Bit Code Detection .....	4-4-4 .....	4-4-4
Cursor Mode Raster Detection Cycle .....	4-4-5 .....	4-4-6
Trigger Mode, Single Shot .....	4-4-6 .....	4-4-7
Trigger Mode, Multiple Shots .....	4-4-7 .....	4-4-8
Noise Flag .....	4-4-8 .....	4-4-9
Null Bit .....	4-4-9 .....	4-4-9
Pause Bit .....	4-4-10 .....	4-4-10
Trigger, Single Shot .....	4-4-11 .....	4-4-11
Trigger, Multiple Shots .....	4-4-12 .....	4-4-12
Optical Color Sensitivity Chart .....	4-5-1 .....	4-5-2
Valid Hyper Mouse Data String .....	4-6-1 .....	4-6-2
Serial Data Read Timing .....	4-6-2 .....	4-6-3
Explanation of Data Strings 2 Bits or Longer .....	4-6-3 .....	4-6-6
Super NES Hyper Mouse Dimensions .....	4-6-4 .....	4-6-7
Standard BIOS, Output Register .....	4-7-1 .....	4-7-3
Examples of Speed Switching Program Subroutine Call .....	4-7-2 .....	4-7-4
MultiPlayer 5 Device Hardware Connections .....	4-9-1 .....	4-9-2
MultiPlayer 5 Read Timing Chart, 5P Mode .....	4-9-2 .....	4-9-5
Data Read Timing for Dissimilar Devices .....	4-9-3 .....	4-9-8
Valid Controller Data String .....	4-9-4 .....	4-9-12
Sample Program Display Format .....	4-10-1 .....	4-10-2

## *List of Tables*

### *BOOK I*

#### TABLE

<u>TITLE</u>	<u>NUMBER</u>	<u>PAGE</u>
B-Bus Address Changes .....	2-17-1 .....	2-17-2
A-Bus Addressing .....	2-17-2 .....	2-17-4
BRR Filter Values.....	3-2-1 .....	3-2-2
Peripherals.....	3-3-1 .....	3-3-1
Port0 - Port3 Registers .....	3-3-2 .....	3-3-2
Timer Function.....	3-5-1 .....	3-5-1
DSP Register Map.....	3-7-1 .....	3-7-1
ADSR Parameters .....	3-7-2 .....	3-7-3
Gain Parameters .....	3-7-3 .....	3-7-5
Noise Generator Clock.....	3-7-4 .....	3-7-8
Source Directory .....	3-7-5 .....	3-7-11
Source Data Block Format .....	3-7-6 .....	3-7-12
Opcode Matrix .....	4-6-1 .....	4-6-2
Address Modes.....	4-7-1 .....	4-7-1
Table of Operations.....	4-7-2 .....	4-7-2
AC Characteristics .....	4-12-1 .....	4-12-1
Command Operand Symbols and Meaning .....	C-1 .....	C-1
Symbols and Meaning for Operational Description .....	C-2 .....	C-2
Explanation of Symbols in the Status Flag Column .....	C-3 .....	C-2
Data Transmission Commands, Group 1 .....	C-4 .....	C-3
Data Transmission Commands, Group 2 .....	C-5 .....	C-3
Data Transmission Commands, Group 3 .....	C-6 .....	C-4
Arithmetic Operation Commands .....	C-7 .....	C-5
8-Bit Logic Operation Commands.....	C-8 .....	C-6
Addition and Subtraction Commands .....	C-9 .....	C-7
Shift Rotation Commands .....	C-10 .....	C-7
16-Bit Data Transmission Commands .....	C-11 .....	C-8
16-Bit Operation Commands .....	C-12 .....	C-8
Multiplication and Division Commands .....	C-13 .....	C-8
Decimal Compensation Commands.....	C-14 .....	C-8
Branching Commands.....	C-15 .....	C-9
Subroutine Call, Return Commands .....	C-16 .....	C-9
Stack Operation Commands .....	C-17 .....	C-9
Bit Operation Commands .....	C-18 .....	C-10
Program Status Flag Operation Commands .....	C-19 .....	C-10
Other Commands .....	C-20 .....	C-10

*List of Tables* (Continued)***BOOK II***

<b>TITLE</b>	<b>TABLE NUMBER</b>	<b>PAGE</b>
Types of Interrupts.....	1-5-1	1-5-2
Interrupt Identification and Clear.....	1-5-2	1-5-2
Interrupt Mask.....	1-5-3	1-5-3
Sending and Receiving a Message.....	1-5-4	1-5-3
Situation Dependant Vectors .....	1-5-5	1-5-4
Operating Modes and Processing Speeds .....	1-5-6	1-5-9
Horizontal Size of VRAM (CDMA Register) .....	1-6-1	1-6-6
Number of Zero Bits in BW-RAM .....	1-6-2	1-6-8
Character Conversion and Data Format.....	1-6-3	1-6-10
Arithmetic Operations Settings and Cycles .....	1-7-1	1-7-1
Amount of Barrel Shift .....	1-8-1	1-8-4
Source Device Settings .....	1-9-1	1-9-3
Destination Device Settings.....	1-9-2	1-9-3
DMA Transmission Speed.....	1-9-3	1-9-4
Timer Modes and Their Ranges.....	1-10-1	1-10-1
Timer Interrupts .....	1-10-2	1-10-2
Registers Listed by Functional Group .....	2-2-1	2-2-3
Instruction Set .....	2-2-2	2-2-6
GSU General Registers .....	2-4-1	2-4-1
GSU Status Register Flags.....	2-4-2	2-4-4
Screen Height.....	2-4-3	2-4-8
Color Gradient .....	2-4-4	2-4-8
Dummy Interrupt Vector Addresses .....	2-5-1	2-5-4
Dummy Data .....	2-5-2	2-5-5
Functions of CMODE.....	2-8-1	2-8-9
DSP1 Command Summary .....	3-2-1	3-2-1
Parameter Data Type.....	3-3-1	3-3-1
Signal Bit Definitions .....	4-1-1	4-1-6
MultiPlayer 5 Switch Function .....	4-9-1	4-9-3
MultiPlayer 5 Data Format .....	4-9-2	4-9-6

[Nintendo Licensee Letterhead{

[Insert Date]

Re: Confidentiality Agreement

Dear \_\_\_\_\_:

\_\_\_\_\_ (the "Company") is a licensee of certain confidential, proprietary, and trade secret information belonging to Nintendo of America Inc. ("Nintendo"). Such information of Nintendo may be provided to you by the company or directly by Nintendo in reliance on your relationship to the Company and your agreement expressed herein. All references in this letter to confidential, proprietary, and trade secret information will be deemed to refer solely to confidential, proprietary, and trade secret information of Nintendo and its affiliated corporations.

Your obligations in connection with confidential, proprietary, and trade secret information of the Company which are reflected in other agreements between you and the Company, remain unchanged, and are in full force and effect.

In consideration of the disclosure of confidential, proprietary, and trade secret information to you, you agree that, except as required by your services to the Company, you will not, at any time during the term of your association with the Company, or at any time thereafter, directly or indirectly use, communicate, disclose, disseminate, discuss, lecture upon, or publish articles concerning such confidential, proprietary, and trade secret information without the prior written consent of the Company.

"Confidential, proprietary, and trade secret information" as used herein means any and all information concerning: (i) copyrights, patents, and/or patent applications owned by Nintendo which are applicable to the Nintendo Entertainment System ("NES"), Game Boy hand held video system ("Game Boy"), Super Nintendo Entertainment System (Super NES), or other hardware, accessory, or software products of Nintendo, (ii) the design, and operation of the NES, Game Boy, Super NES, or other Nintendo products, including without limitation the security system of such products, and (iii) new products, marketing plans, know-how, techniques, and methods relating to the development of software for the NES, Game Boy, Super NES, or other Nintendo hardware, accessory, or software products disclosed to you as a consequence of, or during your association with the Company. Such confidential, proprietary, and trade secret information will not include information which is: (i) a part of the public domain; or (ii) obtained by you from someone otherwise authorized to disclose such information.

All documents, tapes, computer records, notebooks, work papers, notes and memoranda containing confidential, proprietary, and trade secret information, made or compiled by you at any time, or made available to you during the term of your association with the Company, including all copies thereof, will be the property of the Company, and will be held by you in trust and solely for the benefit of the Company, and will be promptly delivered to the Company upon termination of your association with the Company or at any time upon request by the Company.

In the event of any material breach by you of your obligations under this agreement, then the Company will be entitled to such relief, including injunctive relief and damages, including attorney's fees, as may be awarded by a court of competent jurisdiction, in addition to all other relief available to the Company. In the event the Company fails to take action against you for such a breach, Nintendo will have a direct right of action against you, without the necessity of naming the Company.

## *Preface*

### **TECHNICAL QUESTIONS**

If you need technical assistance with your Nintendo Licensee product, our Licensee Support Group Engineers are available between 9:00 a.m. and 6:00 p.m. Pacific Standard Time.

**Telephone:** 1-206-861-2715

**Fax:** 1-206-882-3585

**Written Inquiries:** Nintendo of America Inc.  
Engineering Department  
Licensee Support Group  
4820 150th Ave. N.E.  
Redmond, Wa. 98052

### **CONFIDENTIALITY**

Pursuant to the terms of each Nintendo product license and/or confidentiality agreement, Nintendo licensees and developers are required to secure the confidential treatment of information received or derived from Nintendo from all employees, agents, or contractors.

In response to the request of several licensees, we have prepared a supplemental confidentiality agreement which is intended to cover only Nintendo derived information that may be used in your business in addition to confidentiality agreements which you will sign with your employees, agents, and contractors for your own benefit. A sample agreement is included at the end of the Preface for your information.

This supplemental agreement is a suggested format only and is not a required form, as laws in your state or jurisdiction may vary. You may wish to consult with your own legal counsel regarding recommended formats for your state/country. In many cases, your existing confidentiality agreements will protect both Nintendo and you fully. However, we urge each of you to review agreements that you have in place and consider this supplemental agreement, or other supplements, as may be appropriate or necessary to protect the rights of Nintendo.

If you do not presently have a confidentiality agreement in place for your own employees, agents, or contractors, including those who have access to confidential information of Nintendo, we suggest you contact your legal counsel for advice on proper agreements to protect your valuable confidential information and to insure that you are fully in compliance with your Nintendo license/confidentiality agreement.

Please contact our Legal or Licensing Departments at 1-206-882-2040 between 9:00 a.m. and 6:00 p.m. Pacific Standard Time, with any questions you may have concerning this matter.

The obligations set forth in this letter regarding treatment of confidential, proprietary, and trade secret information are continuing obligations which will continue regardless of your continuing association with the company.

Please acknowledge your understanding and acceptance of the foregoing by signing and returning two (2) copies of this letter to the Company for the benefit of the Company and Nintendo.

Yours sincerely,  
[Insert Nintendo Licensee/Developer Name]  
By: \_\_\_\_\_

**ACKNOWLEDGED AND ACCEPTED**

[Insert Contractor or Employee Name]

By: \_\_\_\_\_

Date: \_\_\_\_\_

## Chapter 1. NOA Licensed Software Approval Process

This chapter describes the process adopted by Nintendo of America Inc. (NOA) which affords interested parties to become Nintendo Authorized Software Developers and/or Nintendo Authorized Software Licensees. The normal process is summarized below.

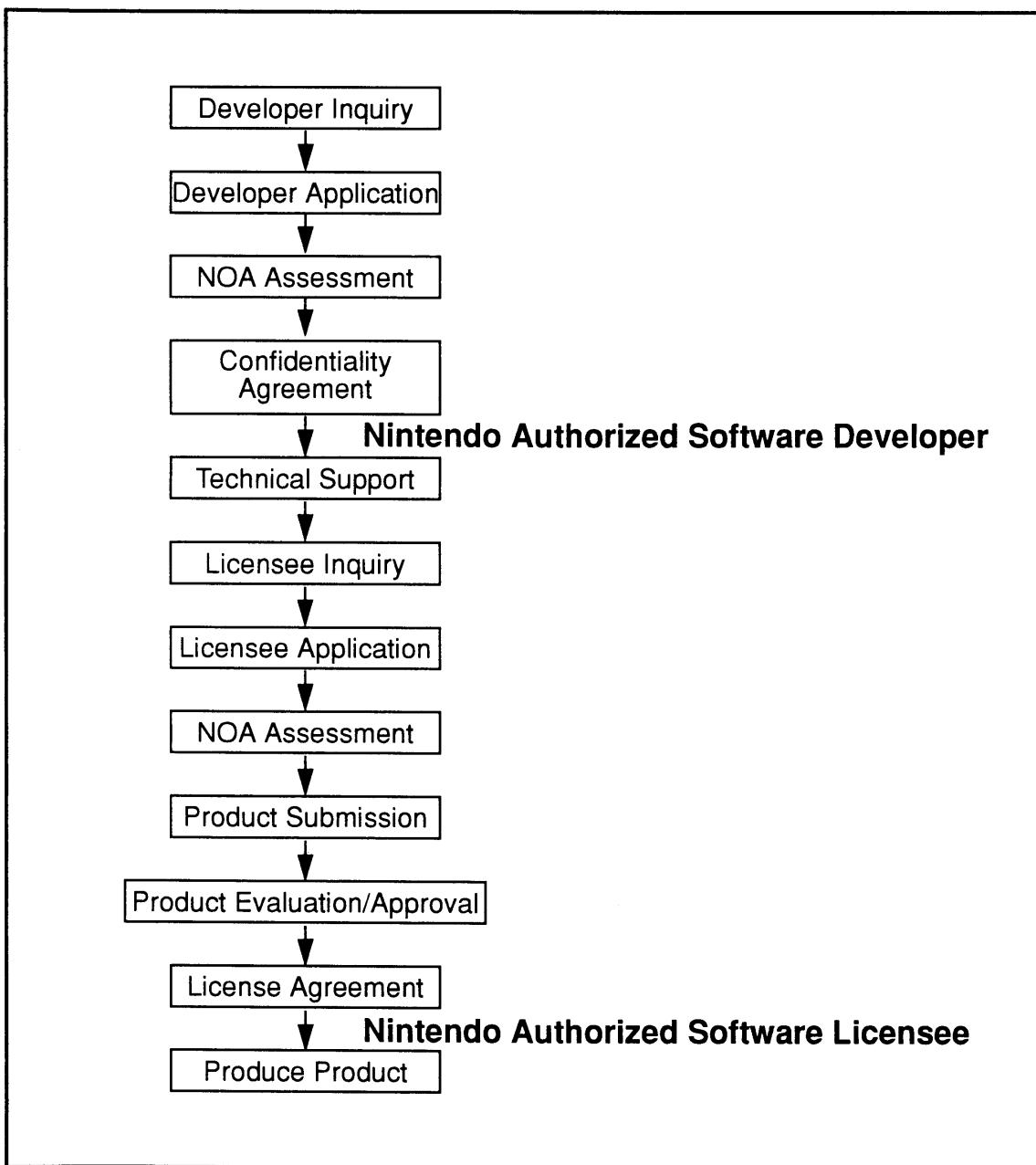


Figure 1-1-1 Software Approval Process

General requirements for the steps listed above are covered in the following paragraphs. Specific questions not answered within this manual should be addressed to NOA's Licensing Department.

## **1.2 AUTHORIZED SOFTWARE DEVELOPER REQUIREMENTS**

Parties interested in becoming a Nintendo Authorized Software Developer may contact the NOA Licensing Department via telephone, FAX, or in writing.

Written Inquiries: Nintendo of America Inc.  
Licensing Department  
4820-150th Avenue N.E.  
Redmond, WA 98052 USA

Telephone: (206) 882-2040

FAX: (206) 882-3585

In response, the NOA Licensing Department will send a letter which describes specific requirements for becoming a licensed developer. These requirements are described, in general, below.

### **1.2.1 LETTER OF APPLICATION**

A prospective developer's letter of application should include the following items.

- 1) A detailed description of the submitting individual or company, including a summary of software development or related experience, financial stability, and market leadership. This information should be in the form of a prospectus, business plan, or summary statement.
  - 2) A detailed introduction to key personnel and developers setting forth any technical, managerial, or development experience which may be relevant and identifying any particular software for any system for which they have contributed.
  - 3) A description of any relationships or work undertaken for Nintendo third party licensees.
  - 4) A description of business facilities and equipment.
  - 5) A copy of any confidentiality/non-disclosure agreement which the company's employees/agents are required to sign.
  - 6) A complete listing and at least three samples of software previously developed, especially those which incorporate elements important to successful NOA software or similar entertainment software.

### **1.2.2 NOA ASSESSMENT**

NOA will review the material submitted and make a preliminary determination of whether the prospective developer's qualifications will support designation as an authorized software developer for Nintendo. Because authorized developers are provided with highly proprietary information belonging to Nintendo, and because many of Nintendo's licensees rely on recommendations and referrals to authorized developers, Nintendo exercises a very high level of care in approving only a select number of authorized developers. The Licensing Department will contact the prospective developer with the results of NOA's assessment.

### **1.2.3 CONFIDENTIALITY AGREEMENT**

If the prospective developer's qualifications support designation as a licensed developer for Nintendo, NOA will prepare a formal confidentiality agreement for review by the prospective developer. Once this agreement has been formalized and processed by NOA's Licensing Department, the party described within the agreement becomes a Nintendo Authorized Software Developer for the specified product line(s).

### **1.2.4 TECHNICAL SUPPORT**

All technical documentation which is available for the licensed product line(s) will be forwarded to the licensed developer upon formalization of the confidentiality agreement. In addition, access is afforded to NOA's Engineering Department Licensee Support Group and NOA's Product Development and Analysis Department. These two support groups will assist the licensed developer with any situational requirements or specifications which are subject to special product development.

## **1.3 AUTHORIZED Software LICENSEE REQUIREMENTS**

To license a software product once it has been developed, the interested party must market the product through an existing Nintendo Authorized Software Licensee or become a Nintendo Authorized Software Licensee. NOA would prefer that interested parties contact Nintendo early in the development phase of a product. Therefore, the interested party will already be a licensed developer when they apply to become an software licensee. Exceptions will be made, however, for those parties which have already developed a software product and wish to license it with Nintendo. In such cases, the interested party will be processed and approved as a Nintendo Authorized Software Developer first, then processed as a Nintendo Authorized Software Licensee. In either case, the requirements in the following paragraphs will apply.

Parties interested in becoming a Nintendo Authorized Software Licensee should contact the NOA Licensing Department via telephone, FAX, or in writing. In response, the NOA Licensing Department will send a letter which describes specific requirements for becoming an software licensee. These requirements are described, in general, beginning on the following page.

### **1.3.1 LETTER OF APPLICATION**

A prospective licensee's letter of application should include the following items.

- 1) A detailed description of the company, including a summary of relevant industry experience, financial resources and stability, and industry leadership or market share. This information should be in the form of a prospectus, business plan, or summary statement.
- 2) A detailed introduction to key personnel and developers setting forth any technical, managerial, or development experience which may be relevant.
- 3) A marketing plan for the proposed product(s), including wholesale/retail price points, targeted distribution channels, advertising commitments, consumer service systems, and merchandising.
- 4) Any market study information on consumer demand for the proposed product(s) which the company may be relying upon.
- 5) A written description (in general terms) of the proposed product.
- 6) A complete listing and at least three samples of software previously developed, especially those which incorporate elements important to successful NOA software or similar entertainment software.

### **1.3.2 NOA ASSESSMENT**

NOA will make a preliminary determination if the:

- a) Product would compliment our current line of video game products.
- b) Company is capable of the distribution and customer service necessary to support a successful product.
- c) Product has any special technical requirements.

NOA's Licensing Department will inform the company of the decision made.

### **1.3.3 TECHNICAL SUPPORT**

If NOA decides to proceed, the prospective licensee will be provided with any technical considerations, suggestions, and any specific technical information required. Technical support will be provided throughout the development of the product, as needed. With respect to those parties previously licensed as developers, this will mean continued support; while those parties contacting Nintendo for the first time will receive a set of technical documentation which is related to the proposed product. A formal confidentiality agreement must be formalized prior to the release of support materials, if one is not already on file.

### **1.3.4 PRODUCT SUBMISSION AND TESTING**

Once the proposed product has been developed and tested by the prospective licensee, it should be submitted in accordance with the applicable software submission requirements, "Super NES Software Submission Requirements" are presented in the following chapter. The samples provided will be tested and results forwarded to the prospective licensee. In cases where failure conditions are detected during testing, NOA will require that the area(s) be corrected and the product be resubmitted for testing.

### **1.3.5 FORMAL LICENSE AGREEMENT**

Once the proposed product is approved, NOA will prepare a formal license agreement for the prospective licensee's review and signature. When formalized, this agreement authorizes the licensee to go into production with the specified licensed product.

## ***Chapter 2. Super NES Software Submission Requirements***

All software submissions to Nintendo of America Inc. must be forwarded to the attention of NOA Product Testing Supervisor. Otherwise, the submission's placement into the testing queue may be delayed. To help reduce a submission's turn-around time, it is suggested that licensees assign a primary contact person for each software submission. All communications with NOA concerning a submission's testing status should be forwarded through this individual. The contact person should also be responsible for notifying any other interested parties.

When a submission is not approved, NOA may send a videotaped copy of the programming problem(s) which prevent(s) the submission from being approved. This is intended to assist the licensee in analyzing the cause of the software problem. It is the licensee's responsibility to send a copy of this tape to any developer(s) of the software. NOA strongly encourages that copies be sent to developer(s) of the software as quickly as possible.

### **2.1 SPECIFICATION SHEET AND CHECK LIST**

The appropriate Software Specification sheet and the Software Submission checklist must be filled out completely and must be correct for the particular program version.

### **2.2 PROGRAM ROMs**

One (1) set of the game ROM(s) must be submitted for approval. ROM data submitted must be written on the same size ROM(s) which are intended to be used in production. If a submission ROM is not available in the size to be used, the next size smaller should be used (i.e., a 3M program should be submitted on two 2M ROMs). All ROMs submitted must be of the same manufacturer, size, and part number. A label should be attached to each master ROM which lists game code, ROM version, and ROM number. A copy of the game ROMs submitted should be retained by the licensee for reference, as NOA cannot return originals or copies of submitted ROMs.

## 2.3 EP-ROMs

Submit only the following EP-ROM types for approval.

- 4M: TOSHIBA TC574000D, SGS THOMPSON M27C4001,  
HITACHI HN27C4001, MITSUBISHI M5M27C401K,  
NEC D27C4001, Texas Instruments TMS27C040-JL,  
TMS27C040-JL4, TMS27C040-JE, TMS27C040-JE4,  
ATMEL AT27C040-12C, MACRONIX MX27C4000DC-12  
8M: ATMEL AT27C080-10DC, AT27C080-12DC, NEC D27C8001  
SGS THOMPSON M27C801-120F1

**Note: All ROMs must be 200ns or faster for Normal Speed.**

**All ROMs must be 120ns or faster for High Speed.**

## 2.4 ROM DATA

In addition to the EP-ROMs, a copy of the ROM data must be submitted in binary format on MS-DOS® 3.5 inch disk(s). The size of the file must be equal to the size of the EP-ROM (i.e., one 4 Meg EP-ROM = one 4 Meg file).

## 2.5 GAME PLAY VIDEO TAPE/RATING CERTIFICATE

A video tape containing complete game play is required unless the product has been rated by the Entertainment Software Ratings Board (ESRB). If the product has been rated by the ESRB, then a copy of the rating certificate must accompany the submission and no video tape is needed.

## 2.6 SCREEN TEXT

A printed copy of the complete screen text must be submitted.

## 2.7 INSTRUCTION MANUAL

Complete game play instructions must be submitted.

**NOTE:** If any of these items are not satisfied, the program will be rejected and will not be submitted into the approval process until all criteria are met.

## 2.8 SOFTWARE VERIFICATION

The following verification process will significantly improve the probability of approval of your software.

1. The licensing screen on all submissions should state "LICENSED BY NINTENDO".
2. Confirm the Licensing Screen information is correct.
3. Check the spelling on the Licensing Screen and Title Screen, as well as the spelling and grammar in the screen text.
4. Confirm the use of a <sup>TM</sup>, circle R (®), or circle C (©) where applicable.
5. Run a "Bypass" Test to assure that, when the game is powered up, the Licensing Screen is visible for at least one second, even if any combination of controller buttons are pressed repeatedly. Also "Power-up" the software repeatedly to assure it does so without programming failures.
6. Game characters should be moved in all possible directions or positions, regardless of whether it is required to play the game properly. For instance, if the game does not require going to a particular area to complete the game, go there anyway to assure there are no programming problems in going to that location.
7. The software should be paused many times during the test, as this often causes programming problems to surface.
8. All testing should be recorded onto a videotape, making it easier to review programming problems.
9. The entire attract mode (demo) should be viewed to assure there are no programming problems.
10. Routines designed to assist the programmer or developer in "debugging" the software should be removed from the game prior to submission. This includes routines to determine hardware type.
11. All references to the Super Famicom, Super Famicom logos, or Super Famicom controllers (with multi-colored buttons) should be removed or revised to represent the Super NES.
12. All games for use with the Super NES Super Scope are required to include a calibration mode.
13. All games are required to have a pause function activated by the "Start" key.

**2.8.1 LICENSEE GAME PLAY VIDEO TAPE PASS/FAIL GUIDELINES**

1. The licensee game play video tape must be recorded on a VHS tape, Standard Play speed (SP) for clarity.
2. No editing of the tape is allowed.
3. If more than one tape is needed to show the entire piece of software, then when a second tape begins it must show that the player is in the exact same place as when the first tape left off.
4. No codes or "built-up" characters are allowed.
5. All levels or areas must be completed, in succession.
6. Screen text must have correct grammar and spelling.
7. No deviations from NOA Software Standards Policy may be present.
8. The entire ending credits (if any) must be shown.

**2.8.2 LICENSING SCREEN INFORMATION PASS/FAIL GUIDELINES**

The following Licensing information should be included for all software.  
This can be displayed on one (1) or two (2) screens.

1. Licensee's software title.
2. Licensee's trademark and copyright notice  
(© 19\_\_\_\_ Licensee's name or copyright owner)
3. LICENSED BY NINTENDO
  - |     **EXAMPLE:**
  - Tom's Golf™ or®
  - © 1992 ABC Corporation
  - LICENSED BY NINTENDO

If a blank screen appears for more than two seconds when powered up, Nintendo suggests placing a message or graphic on the screen so that consumers do not think their game is inoperable (e.g., --"Please Wait"--). If a blank screen appears for more than five seconds during game play, a message or graphic should also be placed on the screen.

**2.8.3 COMMON PROBLEMS**

Some possible problems that may prevent approval of a piece of software include, but are not limited to the following:

1. Lock up of the software.
2. Scrambled blocks or characters appear on the screen.
3. The software won't pause.
4. Your character can get stuck somewhere with no possible way to get out.
5. Scrambled graphics at the edges of the screen when the screen scrolls in any direction.
6. Vowels in the passwords or password entry-system.
7. Colored lines at the top or bottom of the screen.
8. Shifting of the screen in any direction (other than normal scrolling).
9. Inconsistent scoring methods.

10. Flashes on screen.
11. Small flickering lines on the screen.
12. Hit or be hit by an enemy but no damage is incurred.
13. Three (3) or four (4) player game can be started without using a four player adapter.
14. Incorrect Licensing Screen; "Licensed by Nintendo" should appear for all formats.
15. Violation of any Programming Cautions in the product Development Manual.
16. Use of the Nintendo logo or representations of Nintendo products in software without license agreement.
17. The use of the term Super Nintendo or Nintendo when the Super Nintendo Entertainment System or Nintendo Entertainment System is the intended reference, respectively.
18. Character actions are inconsistent (for instance, a character that cannot fly, being able to walk off the edge of a platform and stand in midair).
19. Referring to the Nintendo control pad by an unacceptable term, such as; "joypad", "directional control", etc.
20. Referring to the Nintendo Controller by an unacceptable term, such as; "joystick", etc.
21. Referring to the Nintendo game pak by an unacceptable term, such as; "Game Cassette", etc.
22. Note: If Licensor approval is required, please assure that this has been finalized before the software submission has been made.
23. Display of Super Famicom symbols or controllers in Super NES games.

#### **2.8.4 A NOTE ON OBJECTIONABLE MATERIAL**

A copy of the Nintendo "Game Content Guidelines" is included in book 2 of this manual. If you are unsure of whether an item of text or element of a game is within Nintendo Software Standards, you may contact our Product Analysis Department early in the development process and they will go over questionable items over the phone. In cases concerning an extensive amount of text, please send it to the attention of NOA Product Testing Supervisor, using the address listed in the Preface of this manual, with the questionable items highlighted. The material will be evaluated and you will be contacted within a week to ten days.

## SOFTWARE SUBMISSION CHECK LIST

**MACHINE TYPE**       SNS       VUE       DMG

**GAME NAME** \_\_\_\_\_

**COMPANY** \_\_\_\_\_

**GAME CODE**      SNS \_\_\_\_\_ VUE \_\_\_\_\_ DMG \_\_\_\_\_

**VERSION**       Evaluation

Approval      Ver. \_\_\_\_\_

Specification Sheet

1 Set of ROMs  
(These must be specific EP-ROM type.  
See Submission Requirements.)

MS-DOS® 3 1/2 Disk(s) (Files must be in binary  
format. See Submission Requirements for  
specific information.)

1 copy of Custom DSP IC if applicable  
(Super NES Submissions Only)

1 Copy of VHS Tapes or ESRB Rating  
Certificate

Screen Text

Instruction Manual or Game Play Instructions

**REMARKS**

---



---



---



---

**NOTE:** This check list must be included with the software submission. If any of these items are not satisfied, the program will be promptly returned and will not be submitted into the approval process until all criteria are met.

**THIS PAGE INTENTIONALLY  
LEFT BLANK**

## Super NES Software Specification

<b>Game Title</b>						
<b>Product Code</b>		SNS - _____				
<b>Accessories</b>		<input type="checkbox"/> None	<input type="checkbox"/> Super Scope	<input type="checkbox"/> Super NES Mouse		
			<input type="checkbox"/> MultiPlayer 5	<input type="checkbox"/> Other ( _____ )		
<b>Overseas Version</b>		<input type="checkbox"/> No	<input type="checkbox"/> Yes	Game Title: _____ Country: _____ Release Date: _____		
<b>Company</b>						
<b>Department</b>						
<b>Contact Name</b>						
<b>Address</b>						
		Tel: _____		Fax No.: _____		
<b>Submission Date</b>		<u>  /  /  </u> <u>M  D  Y</u>	<b>Method of Submission:</b> <input type="checkbox"/> Mail <input type="checkbox"/> By Hand			

### ROM Registration Data

Data Name	Address	Data	Data Name	Address	Data
Maker Code	FFB0H	____H (' ____')	Game Title Registration	FFC0H~FFD4H	[REDACTED]
	FFB1H	____H (' ____')	Map Mode	FFD5H	____H
Game Code	FFB2H	____H (' ____')	Cartridge Type	FFD6H	____H
	FFB3H	____H (' ____')	ROM Size	FFD7H	____H
	FFB4H	____H (' ____')	RAM Size	FFD8H	____H
	FFB5H	____H (' ____')	Destination Code	FFD9H	____H
Fixed Value	FFC0H~FFBCH	00 H	Fixed Value	FFDAH	33 H
Expansion RAM Size	FFBDH	____ H	Mask ROM Ver.	FFDBH	____ H
Special Version	FFBEH	____ H	Complement Check	L FFDDH	____ H
Cartridge Type (Sub-number)	FFBFH	____ H	Check Sum	L FFDEH H FFDFH	____ H

\* Write equivalent letter in parenthesis "( )".

### Game Title Registration

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FFC0	Game Name															
	Code (ASCII)	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
FFD0	Game Name															
	Code (ASCII)	.	.	.	.	.	.	.	.	.						

\* Use code 20 (H) to fill space and unused area.

(Continued on reverse side)

**ROM Version**

Mask ROM	<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> __			
EP-ROM	<input type="checkbox"/> 0	<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> __	<input type="checkbox"/> E (Interim)

**Memory Configuration**

ROM	Size: _____ MBit	High Speed Required?	<input type="checkbox"/> Yes	<input type="checkbox"/> No
RAM	<input type="checkbox"/> No	Size:	Bit	
	<input type="checkbox"/> Yes	Battery Back Up?	<input type="checkbox"/> No	<input type="checkbox"/> Yes
External Co-processor	<input type="checkbox"/> None	<input type="checkbox"/> DSP (DSP____ )		
		<input type="checkbox"/> Super FX (Expansion RAM _____ Bit)		
		Data back-up:	<input type="checkbox"/> Yes	<input type="checkbox"/> No
		<input type="checkbox"/> Super FX2 (Expansion RAM _____ Bit)		
		Data back-up:	<input type="checkbox"/> Yes	<input type="checkbox"/> No
		<input type="checkbox"/> SA-1 Internal RAM Data Back-up:	<input type="checkbox"/> Yes	<input type="checkbox"/> No
	<input type="checkbox"/> Other: _____			

**Check Sums**

EP-ROM Configuration	_____ MBits x _____ Pcs x 1 Set	Manufacturer: _____	Model No.: _____
ROM 0	_____ H	ROM 4	_____ H
ROM 1	_____ H	ROM 5	_____ H
ROM 2	_____ H	ROM 6	_____ H
ROM 3	_____ H	ROM 7	_____ H
Total	_____ H		

Affix a label to master ROM which contains product Game Code, ROM Version, and ROM Number. Total check sum must be written even though disk media is used.

**File Names**

Floppy Disk Configuration	3.5"	<input type="checkbox"/> DSHD	<input type="checkbox"/> HD	_____ Pcs x 1 Set
	File Name	HEX Code	File Name	HEX Code
FILE 0	_____	_____ H	FILE 1	_____ H
FILE 2	_____	_____ H	FILE 3	_____ H
FILE 4	_____	_____ H	FILE 5	_____ H
FILE 6	_____	_____ H	FILE 7	_____ H

**Special Programming**

Special Programming?	<input type="checkbox"/> Yes ( _____ )	<input type="checkbox"/> No
----------------------	--	-----------------------------

**Remarks:**


---



---



---

***Instructions for Super NES Software Specification***

**1. Game Title, Product Code, Scheduled Release Date, Accessories, Overseas Version, Company, Contact, Address, Telephone No., Fax No., submission date and method.**

- Product Code (4 digits) will be determined by Nintendo.
- Scheduled release date should be entered.
- Game Title includes sub-title if any.
- Indicate accessories other than standard controller which can be used.
- If the product has been sold, or is to be sold in another country; write the game title, country, and the scheduled release date in that country.
- Company, Contact, Address, Telephone No., Fax No. must be completed.
- Submission date and method of submission should be entered.

**2. ROM Registration Data**

- Write the contents registered in the indicated addresses of the master ROM. Refer to "Description of ROM Registration Data Specification" for details. Enter ASCII characters in areas marked with parenthesis "( )".

**3. Game Title Registration**

- Enter the game title registered in the master ROM using ASCII characters and their ASCII codes. Refer to "Character Code List for Game Title Registration".

#### 4. ROM Version

- Mask ROM Version  
The Mask ROM Version number starts from 0 and increases for each revised version sent for changes after starting production.
- EP-ROM Version  
The EP-ROM Version number starts from 0 and increases for each revised version sent for approval.
- Example

#### 5. Memory Configuration

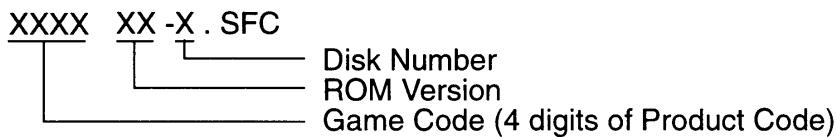
- Enter the memory configuration of the product.
- Enter ROM size and whether or not High Speed Mode (3.58MHz operation) is required.
- RAM  
If RAM is used, enter memory size and indicate whether or not Battery Back-up is used.
- External Co-processor  
If an external co-processor is used (i.e., DSP1, Super FX), select the configuration used.

## 6. Check Sums

- Enter the check sum of each ROM submitted. To calculate the check sum, add each byte in the ROM data. The lower 2 bytes of the resulting value is the check sum. Enter the check sum for each ROM submitted for the master program and the total of their individual check sums. The total is calculated by simply adding the individual check sums. This method of calculation is different from the check sum on the ROM Registration Specification.

## 7. File Names

- Write the file name of each disk using the following conventions.



For example,

If the Game Code is AAAE, ROM version is 0.1, and ROM size is 8M; the first disk (Disk 1 of 1) should be named: "AAAE01-0.SFC" (8M file).

If, on the other hand, the Game Code is MW, ROM version is 1.0, and ROM size is 20M;

- 1st Disk (1 of 3) = "MW\_E10-0.SFC" (8M file)
- 2nd Disk (2 of 3) = "MW\_E10-1.SFC" (8M file)
- 3rd Disk (3 of 3) = "MW\_E10-2.SFC" (4M file)

Note that when the Game Code only uses 2 digits, a bar "\_" is inserted in the 3rd digit's place and the destination code is inserted in the 4th digit's place.

## 8. Special Programming

- If special programming is implemented, such as for the purposes of copyright protection, it should be indicated. Also, the contents of the special programming must be explained in writing.

Note: When more than one ROM is required for the game program, all ROMs submitted as a set should be the same part number.

## 9. Remarks

- If a special configuration of game pak is used, please note the special configuration here. Write the name of the evaluation board which was used for debugging the game. Please write the full name as printed on the board. For example,  
SHVC-4PV5B-10
- If several boards were used for debugging the game, all boards must be listed.

***Character Code List for Game Title Registration***

	00	10	20	30	40	50	60	70	80	~	F0
0			SP	0	@	P	'	p			
1			!	1	A	Q	a	q			
2			"	2	B	R	b	r			
3			#	3	C	S	c	s			
4			\$	4	D	T	d	t			
5			%	5	E	U	e	u			
6			&	6	F	V	f	v			
7			,	7	G	W	g	w			
8			(	8	H	X	h	x			
9			)	9	I	Y	i	y			
A			*	:	J	Z	j	z			
B			+	;	K	[	k	{			
C			,	<	L	¥	l	l			
D			-	=	M	]	m	}			
E			.	>	N	^	n	~			
F			/	?	O	_	o				

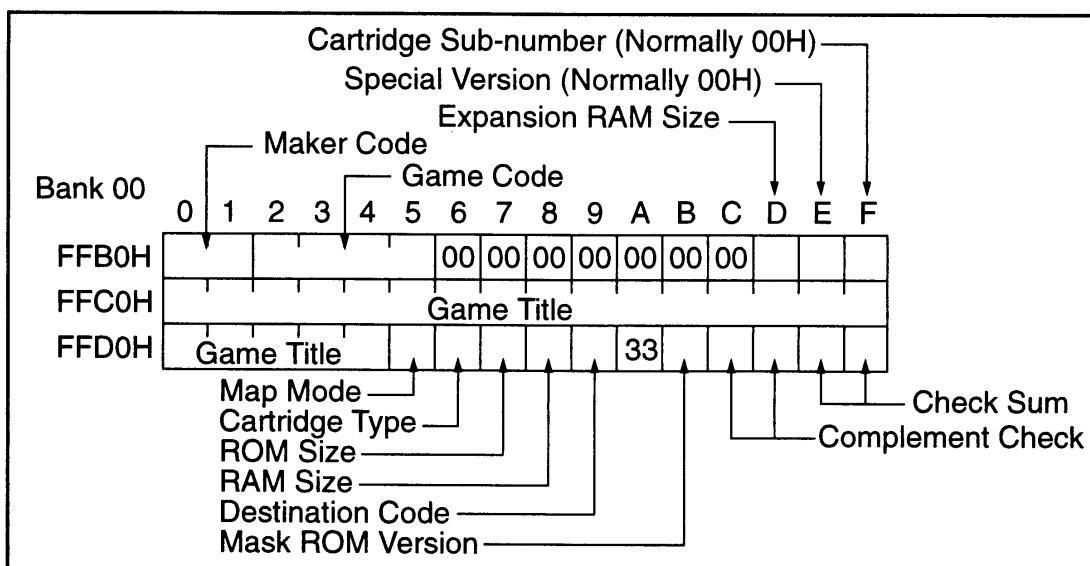
Note 1: Do not use characters in shaded areas.

Note 2: "SP" means space.

Example: If ASCII character is A, ASCII code is 41.

### ***ROM Registration Data Specification***

1. Insert the game title and Super NES game specification at the specified addresses in the ROM.
2. The ROM Registration Data area is 48 bytes from address 00:FFB0H ~ 00:FFDFH in Super NES Memory.
3. The address in ROM for registration data, using Map Mode 20, is 007FB0H ~ 007FDFH.
4. The address in ROM for registration data, using Map Mode 21, is 00FFB0H ~ 00FFDFH.
5. The address in ROM for registration data, using Map Mode 23 (SA-1), is 007FB0H ~ 007FDFH.
6. The address in ROM for registration data, using Map Mode 25, is 40FFB0H ~ 40FFDFH.
7. ROM registration data should be stored using the format below.



8. The following data will be stored in Super NES Memory for every Super NES game.

00:FFB6H ~ 00:FFBCH = 00H  
00:FFDAH = 33H

### **Description of ROM Registration Data Specification**

#### **1. Maker Code (FFB0H, FFB1H)**

Enter the 2-digit ASCII code assigned by Nintendo. Refer to the Nintendo/Licensee contract, if in doubt. All letters must be in upper case.

For example;

If Maker Code is 01, the ASCII code for 0 (30H) is stored at FFB0H and the ASCII code for 1 (31H) is stored at FFB1H.

If Maker Code is FF, the ASCII code for F (46H) is stored at FFB0H and FFB1H.

#### **2. Game Code (FFB2H ~ FFB5H)**

Enter the 4-digit Game Code assigned by Nintendo in ASCII. All letters must be in upper case.

For Example;

If Game Code is "SMWJ", the following ASCII codes will be entered at the indicated addresses.

53H (S) ⇒ FFB2H

4DH (M) ⇒ FFB3H

57H (W) ⇒ FFB4H

4AH (J) ⇒ FFB5H

If a game program which was previously assigned a 2-digit Game Code is to be manufactured again, the original 2-digit code will be entered followed by 2 "Space" codes. The ROM submission sheet should be completed in the same manner.

For example;

If Game Code is "MW", the following ASCII codes will be entered at the indicated addresses.

4DH (M) ⇒ FFB2H

57H (W) ⇒ FFB3H

20H (space) ⇒ FFB4H

20H (space) ⇒ FFB5H

#### **3. Fixed Value (FFB6H ~ FFBCH)**

Store fixed value 00H at addresses FFB6H ~ FFBCH.

#### 4. Expansion RAM Size (FFBDH)

Enter the size of the expansion RAM installed in the game pak using the table below. If the size used is not listed below, choose the next larger size which is listed.

For example, enter the size of the RAM used for Super FX co-processor. If no expansion RAM is installed, enter 00H at address FFBDH.

For game paks which use the SA-1, enter 00H at address FFBDH. Enter the size of the RAM used as BW-RAM at address FFD8H.

FFBDH	Size of Expansion RAM
00H	None
01H	16 KBit
03H	64 KBit
05H	256 KBit
06H	512 KBit
07H	1 MBit

#### 5. Special Version (FFBEH)

This is only used under special circumstances, such as for a promotional event. The code 00H should be entered under normal circumstances.

#### 6. Cartridge Type Sub-Number (FFBFH)

This is only assigned when it is necessary to distinguish between games which use the same cartridge type. The code 00H is normally assigned.

#### 7. Game Title (FFC0H ~ FFD4H)

Enter the game title using ASCII code (JIS 8 bit). Refer to "Character Code List for Game Title Registration" for characters which may be used. The code "20H" should be used for a space and for all unused areas. The game title registered should be close to the title under which the game will be marketed, not a temporary name used for development purposes.

## 8. Map Mode (FFD5H)

This location is used to store the map mode and the speed of operation for the Super NES CPU. Select the appropriate code from the table below.

FFD5H	Map Mode	Super NES CPU Clock
20H	Mode 20	2.68 MHz (normal speed)
21H	Mode 21	2.68 MHz (normal speed)
22H	Reserved-Future Use	-----
23H	Mode 23 (SA-1)	2.68 MHz (normal speed)
25H	Mode 25	2.68 MHz (normal speed)
30H	Mode 20	3.58 MHz (high speed)
31H	Mode 21	3.58 MHz (high speed)
35H	Mode 25	3.58 MHz (high speed)

**9. Cartridge Type (FFD6H)**

Indicate the game pak (cartridge) configuration. Use one of the tables below, depending upon whether or not a co-processor is used.

Without Co-processor

<b>FFD6H</b>	<b>Game Pak (Cartridge) Configuration</b>
00H	ROM Only
01H	ROM + RAM
02H	ROM + RAM + Battery

With Co-processor

<b>FFD6H</b>		<b>Game Pak (Cartridge) Configuration</b>
<b>Upper</b>	<b>Lower</b>	
0*H	-	Co-processor = DSP
1*H	-	Co-processor = Super FX
2*H	-	Co-processor = OBC1
3*H	-	Co-processor = SA-1
E*H	-	Co-processor = Other
F*H	-	Co-processor = Custom Chip
-	*3H	ROM + Co-processor
-	*4H	ROM + Co-processor + RAM
-	*5H	ROM + Co-processor + RAM + Battery
-	*6H	ROM + Co-processor + Battery

For example;

If a game pak uses the Super FX as its co-processor and contains a 256K Expansion RAM as game pak RAM for battery backup, store 15H at address FFD6H. In this case 05H would be stored at address FFBDH and 00H would be stored at address FFD8H.

If a game pak uses a DSP as its co-processor and no RAM, store 03H at address FFD6H. In this case 00H would be stored at addresses FFBDH and FFD8H.

If a game pak uses the SA-1 as its co-processor with 64K SRAM and battery, store 35H at address FFD6H. In this case, 00H would be stored at address FFBDH and 03H at address FFD8H.

#### 10. ROM Size (FFD7H)

The program ROM size is stored at this address. Select the appropriate code from the table below.

<b>FFD7H</b>	<b>ROM Size</b>
09H	3 ~ 4M Bit
0AH	5 ~ 8M Bit
0BH	9 ~ 16 M Bit
0CH	17 ~ 32M Bit
0DH	33 ~ 64M Bit

#### 11. RAM Size (FFD8H)

The CPU RAM size is stored at this address. Select the appropriate code from the table below. If CPU RAM is not installed in a game pak, store 00H at address FFD8H. If only expansion RAM (game pak RAM) is installed, such as the one used with the Super FX co-processor, 00H is also stored at address FFD8H. The BW-RAM size for an SA-1 game pak should be stored at this address.

<b>FFD8H</b>	<b>RAM Size</b>
00H	No RAM
01H	16K Bit
03H	64K Bit
05H	256K Bit
06H	512K Bit
07H	1M Bit

For example;

If a game pak does not contain a co-processor and uses a 64K RAM for battery backup, store 03H at address FFD8H. In this case 00H is stored at address FFBDH and 02H is stored at address FFD6H.

If a game pak uses the Super FX as its co-processor and contains a 256K Expansion RAM as game pak RAM for battery backup, store 00H at address FFD8H. In this case 05H is stored at address FFBDH and 15H is stored at address FFD6H.

#### 12. Destination Code (FFD9H)

Store the code, from the table below, which best describes where the product will be sold.

<b>FFD9H</b>	<b>Destination (Language)</b>	<b>ROM Recognition Code (Fourth digit of Game Code)</b>
00H	Japan	J
01H	North America (USA and Canada)	E
02H	All of Europe	P
03H	Scandinavia	W
06H	Europe (French only)	F
07H	Dutch	H
08H	Spanish	S
09H	German	D
0AH	Italian	I
0BH	Chinese	C
0DH	Korean	K
0EH	Common	A
0FH	Canada	N
10H	Brazil	B
	Nintendo Gateway System	G
11H	Australia	U
12H	Other Variation	X
13H	Other Variation	Y
14H	Other Variation	Z

## 13. Fixed Value (FFDAH)

Store fixed value 33H at address FFDAH.

## 14. Mask ROM Version (FFDBH)

Store the version number of the mask ROM released to the market as a product. The number begins with 0 at production and increases with each revised version.

## 15. Complement Check (FFDCH, FFDDH)

Store the 1's complement of the lower 2 bytes of the program check sum in the order of; FFDCH, lower and FFDDH, upper. Refer to "Check Sum", below, for calculation of the check sum.

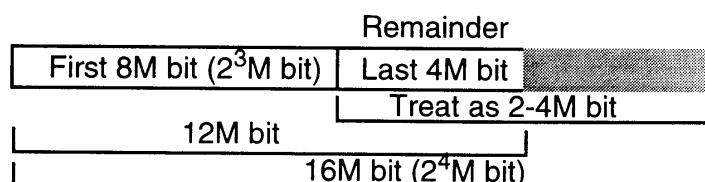
$$\text{Check Sum} \quad \text{Complement Check}$$

$$(FFDEH, FFDFH) + (FFDCH, FFDDH) = FFFFH$$

## 16. Check Sum (FFDEH, FFDFH)

First, store 0FFH into the complement check area (FFDCH, FFDDH) and 00H into the check sum area (FFDEH, FFDFH). Then add each byte in the ROM data. If ROM size cannot be expressed evenly in  $2^n$ M bit, such as 10M or 20M bit, add the remainder until a total of  $2^n$ M bit is reached.

For example, If the program contains 12M bit, perform the calculation as if it were 16M bit as shown below.



$$(\text{Total of first 8M bit}) + [(\text{Total of last 4M bit}) \times 2] = \text{Check Sum}$$

For 10M bit, perform the calculation as if it were 16M bit.

$$(\text{Total of first 8M bit}) + [(\text{Total of last 2M bit}) \times 4] = \text{Check Sum}$$

For 20M bit, perform the calculation as if it were 32M bit.

$$(\text{Total of first 16M bit}) + [(\text{Total of last 4M bit}) \times 4] = \text{Check Sum}$$

For 24M bit, perform the calculation as if it were 32M bit.

$$(\text{Total of first 16M bit}) + [(\text{Total of last 8M bit}) \times 2] = \text{Check Sum}$$

Next, store the lower 2 bytes of the check sum value into the check sum area (FFDEH, FFDFH). FFDEH will contain the lower byte and FFDFH will contain the upper byte.

Then, store the lower 2 bytes of the complement check in registers FFDCH and FFDDH.

***Data Storage on Floppy Disk***

1. Use 3.5" DSHD or HD diskettes in MS-DOS IBM format.
2. File data must be in ROM image binary format and not compressed. The maximum data size on a disk is 8M bit. If the program being submitted is larger than 8M bit, the program should be divided and recorded on multiple disks. The last disk must be written to use the full 8M bit.
3. The file name for the disk is determined as follows;



for example, "AAAJ01-0.SFC".

4. A seal must be affixed to each disk to specify company name, game title, game code, ROM version, date, and disk number.
5. For SA-1 games, don't split data by even and odd addresses.

## Super NES Cartridge PCB List

### Production PCB List<sup>\*1</sup>

Part Number	Production PCB	ROM	RAM	Other
22536	SHVC-1A0N	1M/2M/4M/8M	None	
22537	SHVC-1A1B	1M/2M/4M/8M	16K	Batt.
22538	SHVC-1A3B	1M/2M/4M/8M	64K	Batt.
22539	SHVC-1A5B	1M/2M/4M/8M	256K	Batt.
22540	SHVC-1B0N	1M/2M/4M/8M	None	DSP1
24468	SHVC-1B5B	1M/2M/4M/8M	256K	DSP1, Batt

### Evaluation PCB List<sup>\*2</sup>

Part Number	Evaluation PCB	ROM	RAM	
22427	SHVC-2P3B	1M/2M/4M/8M	None/64K	Battery & 64K SRAM
21945	SHVC-1P0N	1M/2M/4M	None	
24470	SHVC-2Q5B	1M/2M/4M/8M	None/64K/256K	Battery <sup>*4,5</sup>
25474	SHVC-4PV5B	4M/8M/12M/16M	None/16K/64K/256K	Battery <sup>*5</sup>
33366	SHVC-4PV7B	4M/8M/12M/16M/24M <sup>*7</sup>	None/512K/1M	Battery & 1M SRAM
28626	SHVC-8PV5B	4M ~ 32M or 4M ~ 64M	None/16K/64K/256K	Battery <sup>*5</sup>
26011	SHVC-2QW5B	4M/8M/12M/16M	None/64K/256K	Battery <sup>*4,5</sup>
28625	SHVC-1RA3B6S	4M or 8M	64K or 512K <sup>*6</sup>	Battery & GSU1
28760	SHVC-4QW5B	1M ~ 32M	None/64K/256K	Battery <sup>*4,5</sup>
22410 <sup>*3</sup>	SHVC-Multi Checker	1M/2M/4M/8M/16M	None/256K/1M	Battery & 256K SRAM
32321	SHVC-8X7B	4M ~ 32M	None/512K/1M	Battery & 1M SRAM

#### Notes:

- 1) Mask-ROM should be used on a Production PCB. Production PCBs listed above are bare boards.
- 2) EP-ROM should be used on an Evaluation PCB. Evaluation PCBs listed above are assemblies.
- 3) SHVC Multi Checker must only be used with SHVC (Japanese Super NES) in order to evaluate SNS software.
- 4) DSP1 must be purchased separately.
- 5) Static RAM(S-RAM) must be purchased separately.
- 6) The 512K SRAM used with GSU may be configured for battery back-up RAM.
- 7) 24M requires change of PLD.

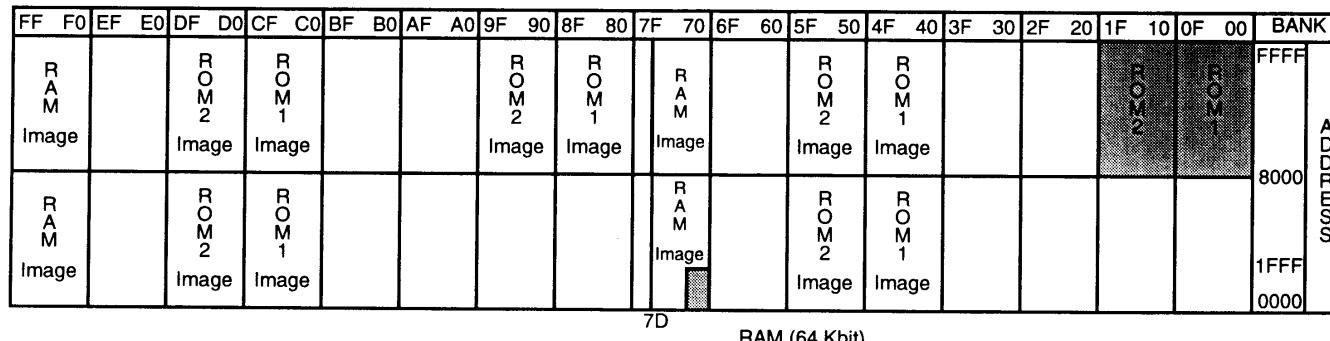
## **21.3 GAME PAK PCB MEMORY MAPPING**

The following memory maps are provided for reference. Only the most commonly used memory maps have been included. For information regarding memory maps which are not shown here, contact NOA's Licensee Support Group at (206) 861-2715.

Figure 2-21-1 SHVC-2P3B PCB MEMORY MAP

Mode 20 (4M x 2 pcs)

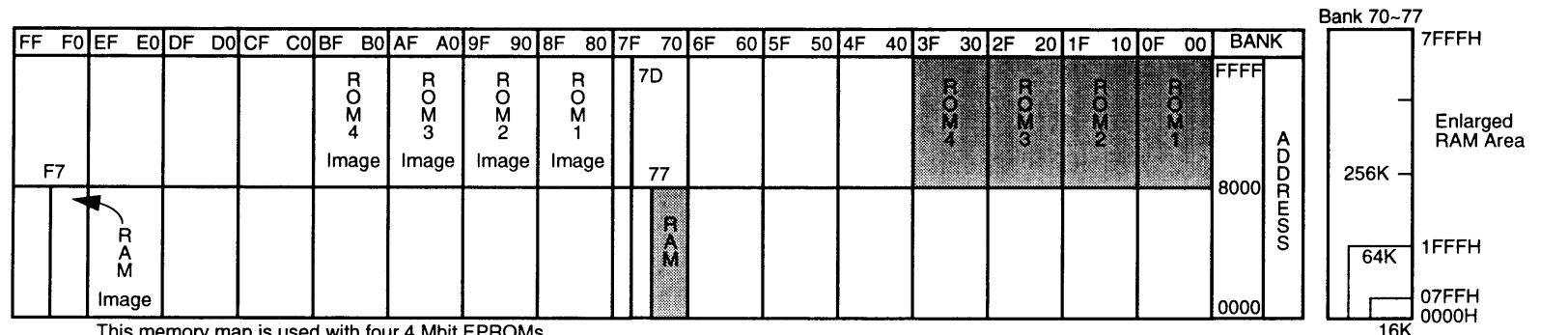
PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM
	20	1/2/4M	1M ~ 8M	64K



Note 1: Since the area of ROM2 is always started from bank 10, ROM1 and ROM2 will be discontinuous if 1 Mbit or 2 Mbit EPROM is used for ROM1.

Figure 2-21-2 SHVC-4PV5B PCB MEMORY MAP Mode 20 (4M x 4 pcs), Mode 21 (4M x 4 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM
	20 or 21	4M/8M <sup>*1</sup>	4/8/12/16M(8/16/24M <sup>*1</sup> )	None/16K/64K/256K



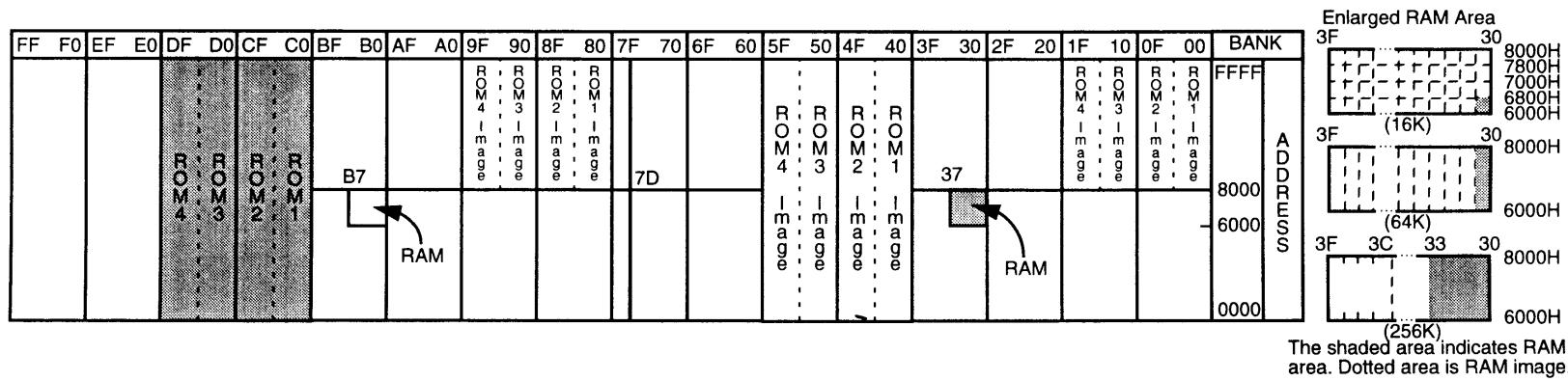
This memory map is used with four 4 Mbit EEPROMs.

Note: The RAM image is different from the production PCB.

If you buy PLD, you can use 8 Mbit EEPROMs. Maximum ROM size will be 24 Mbits (8 Mbit x 3).

The image of bank 70 is generated in bank 71~7D and F0~FF.

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM
	20 or 21	4M/8M <sup>*1</sup>	4/8/12/16M(8/16/24M <sup>*1</sup> )	None/16K/64K/256K

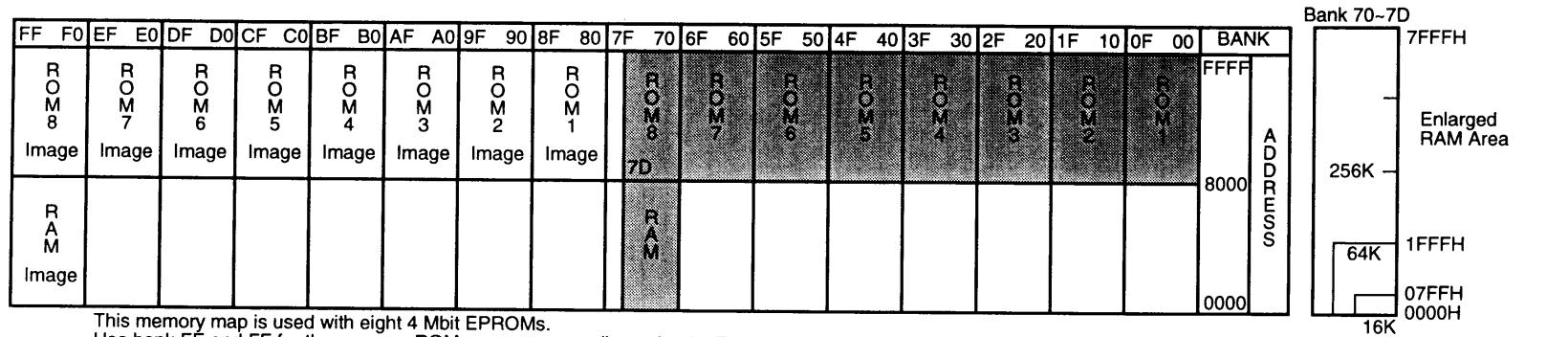


The shaded area indicates RAM area. Dotted area is RAM image.

Figure 2-21-3 SHVC-8PV5B PCB MEMORY MAP

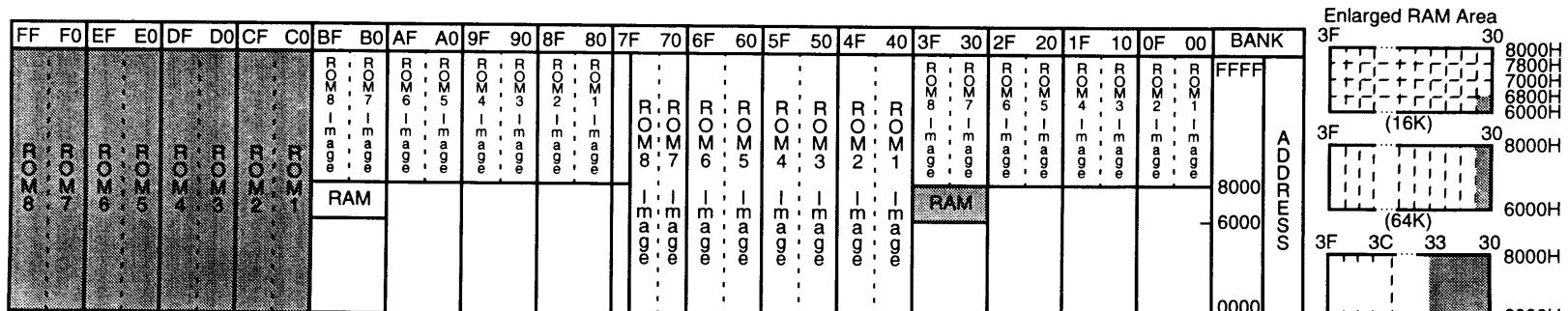
Mode 20 (4M x 8 pcs), Mode 21 (4M x 8 pcs)

PCB Configuration		Mapping		Usable EPROMs		ROM Size		Usable RAM	
		20 or 21	4M/8M	4M/8M	4M/8M	4/8/12/16/20/24/28/32M	None/16K/64K/256K		



1-2-27

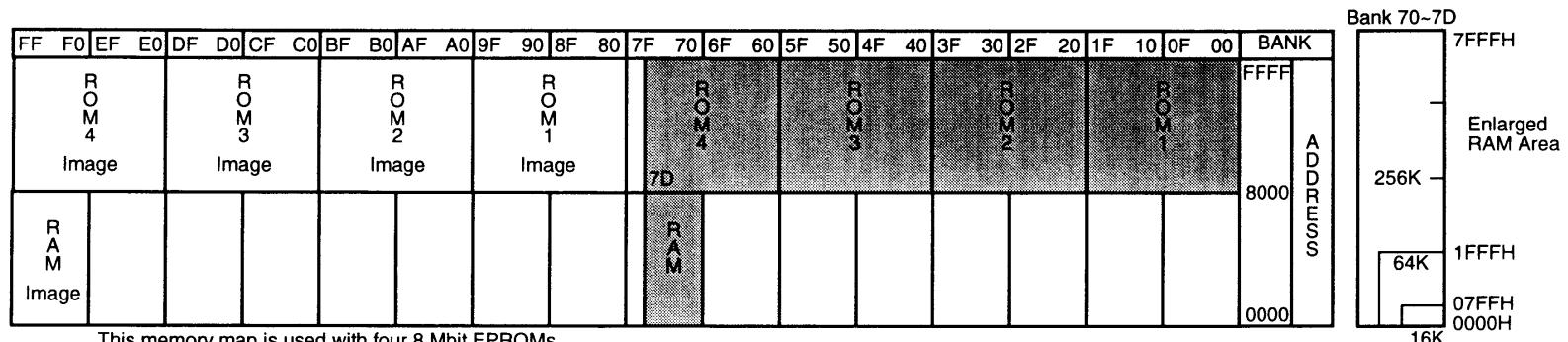
PCB Configuration		Mapping		Usable EPROMs		ROM Size		Usable RAM	
		20 or 21	4M/8M	4M/8M	4M/8M	4/8/12/16/20/24/28/32M	None/16K/64K/256K		



The ROM image in address 8000H ~ FFFFH of bank C0H ~ FFH is generated in bank 00H ~ 3FH and 80H ~ BFH.  
This memory map is used with eight 4 Mbit EPROMs.

Figure 2-21-4 SHVC-8PV5B PCB MEMORY MAP Mode 20 (8M x 4 pcs), Mode 21 (8M x 4 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM
	20 or 21	4M/8M	4/8/12/16/20/24/28/32M	None/16K/64K/256K



This memory map is used with four 8 Mbit EPROMs.

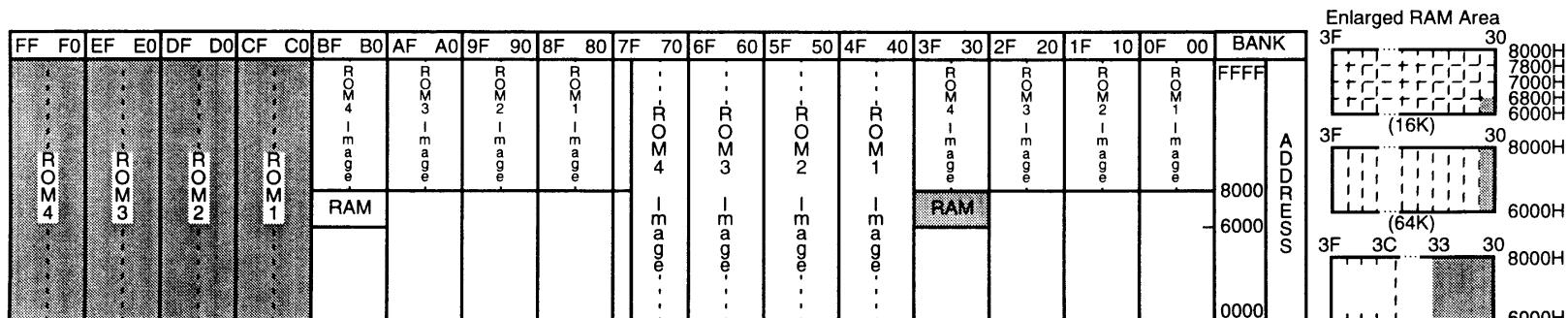
Use bank FE and FF for the program ROM area corresponding to bank 7E and 7F.

The earlier 8PV5B PCB version can use up to 64M ROM, but do not exceed 32M on this PCB (DSW1 pin 7 off).

The image of bank 70 is generated in bank 71~7D and F0-FF.

1-2-28

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM
	20 or 21	4M/8M	4/8/12/16/20/24/28/32M	None/16K/64K/256K

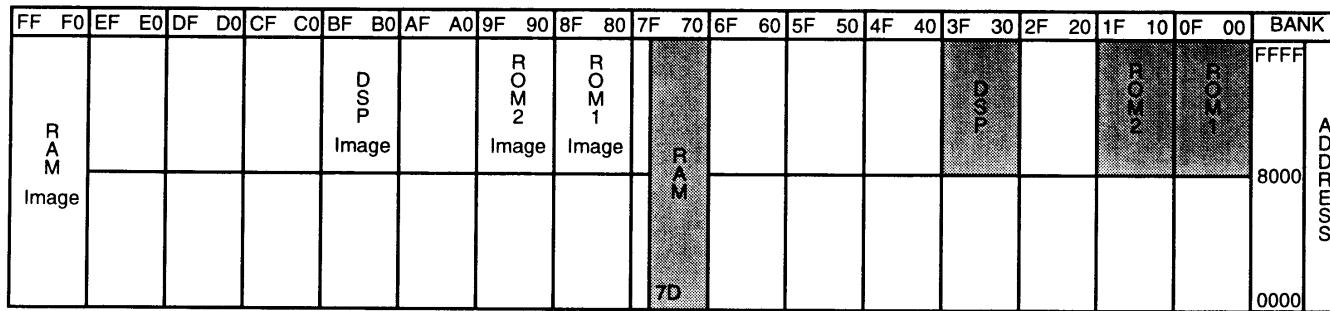


The ROM image in address 8000H ~ FFFFH of bank C0H ~ FFH is generated in bank 00H ~ 3FH and 80H ~ BFH.  
The earlier 8PV5B PCB version can use up to 64M ROM, but do not exceed 32M on this PCB (DSW1 pin 7 off).

The shaded area indicates RAM area. Dotted area is RAM image.

Figure 2-21-5 SHVC-2Q5B PCB MEMORY MAP Mode 20 (4M x 1 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20	1M/2M/4M	1/2/4/5/6/8M	None/64K/256K	DSP



The ROM2 area always starts from bank 10. If 1 Mbit (00H~03H) or 2 Mbit (00H~07H) EPROM is used for ROM1, and ROM2 is used, ROM 1 and ROM2 are not continuous.  
The DSP image appears in bank B0-BF, but do not use this area in the high speed mode.  
The RAM image appears in bank F0~FF, but do not use this area in the high speed mode.

1-2-29

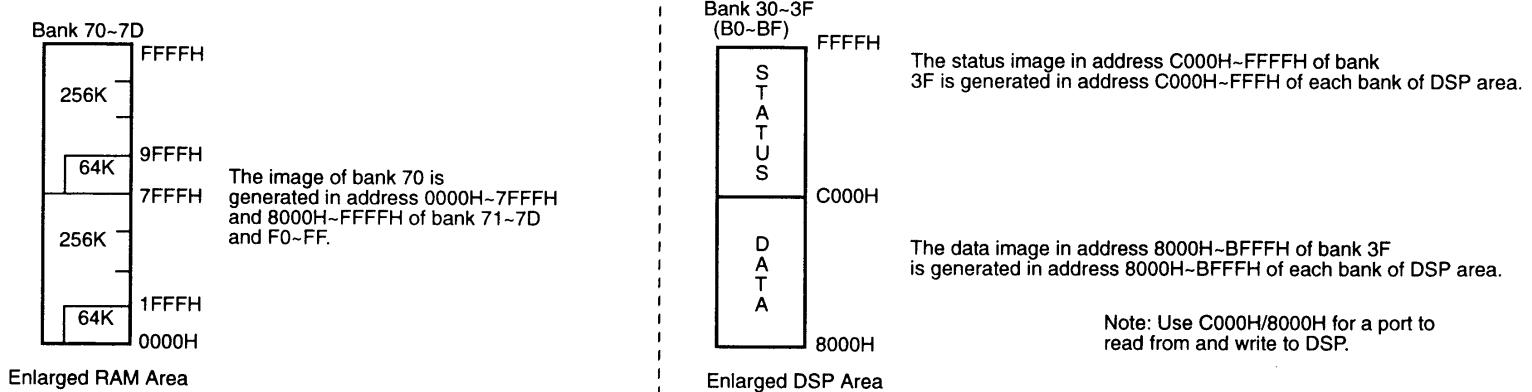
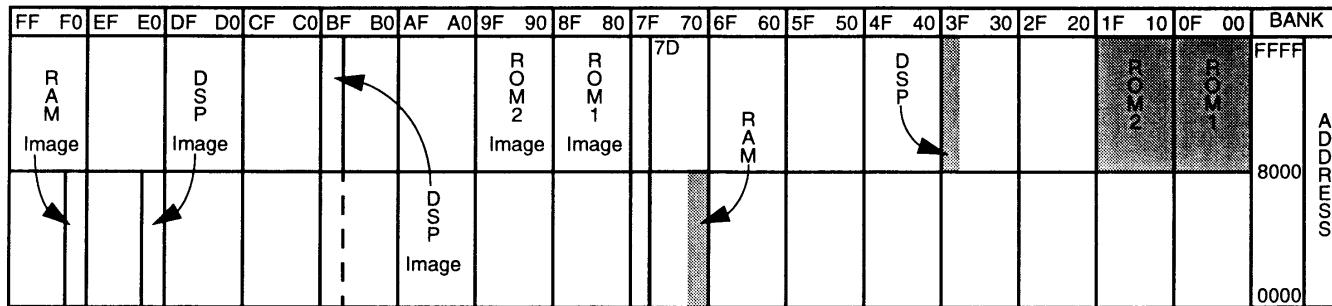


Figure 2-21-6 SHVC-2QW5B PCB MEMORY MAP Mode 20 (4M x 2 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	4M/8M	4/8/16M	None/16K/64K/256K	DSP



The DSP image appears in bank BF and E0, but do not use this area in the high speed mode.

The RAM image appears in bank F0~FF, but do not use this area in the high speed mode.

Be aware that the DSP area will change, depending upon the ROM size greater or less than 8M, during Mode 20.

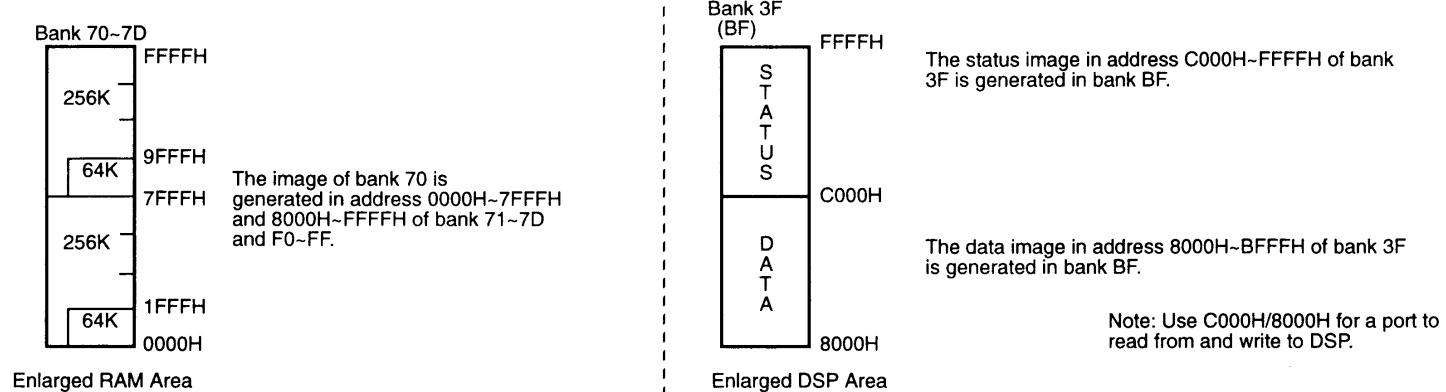
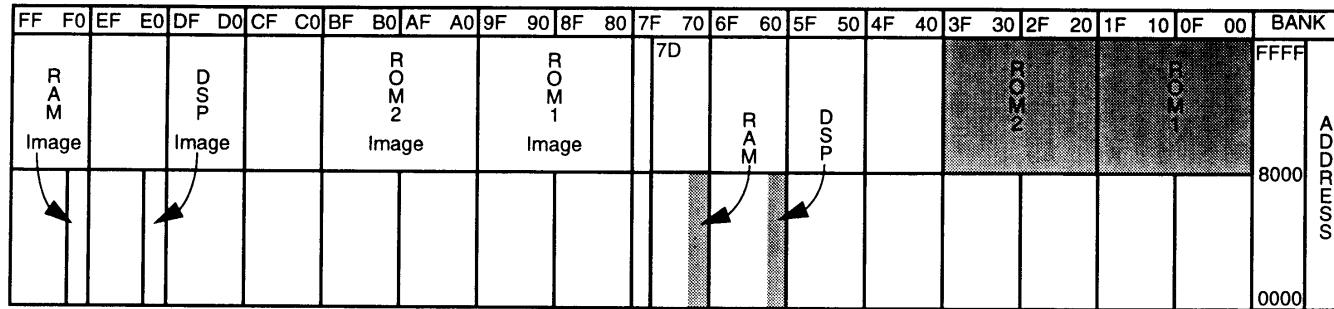


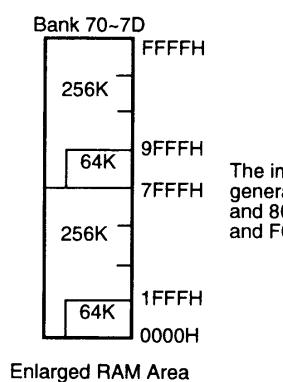
Figure 2-21-7 SHVC-2QW5B PCB MEMORY MAP Mode 20 (8M x 2 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	4M/8M	4/8/16M	None/16K/64K/256K	DSP



The DSP image appears in bank E0, but do not use this area in the high speed mode.  
 The RAM image appears in bank F0~FF, but do not use this area in the high speed mode.  
 Be aware that the DSP area will change, depending upon the ROM size greater or less than 8M, during Mode 20.

1-2-31



The image of bank 70 is generated in address 0000H~7FFFH and 8000H~FFFFH of bank 71~7D and F0~FF.

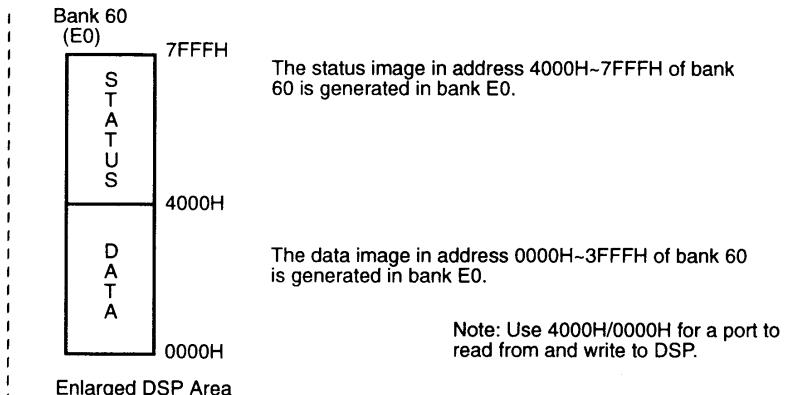
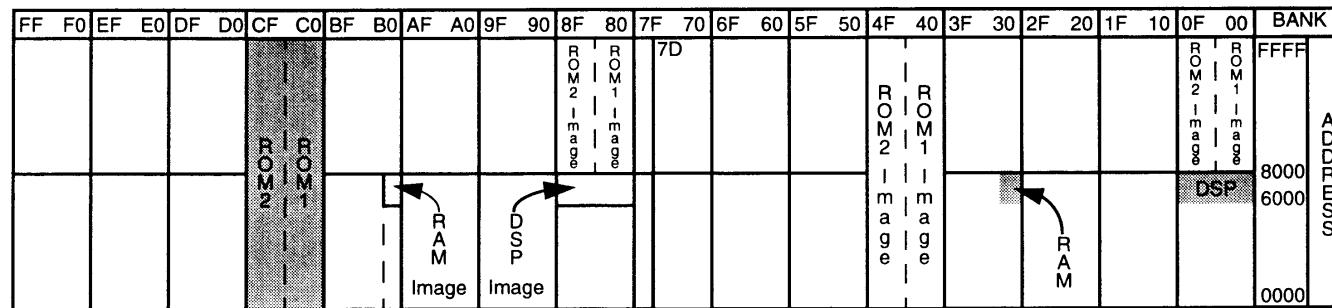


Figure 2-21-8 SHVC-2QW5B PCB MEMORY MAP Mode 21 (4M x 2 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	4M/8M	4/8/16M	None/16K/64K/256K	DSP



The ROM image in address 8000~FFFF of bank C0-CF appears in bank 00~0F and 80~8F.  
During Mode 21, the DSP area does not change even if the ROM size changes.

1-2-32

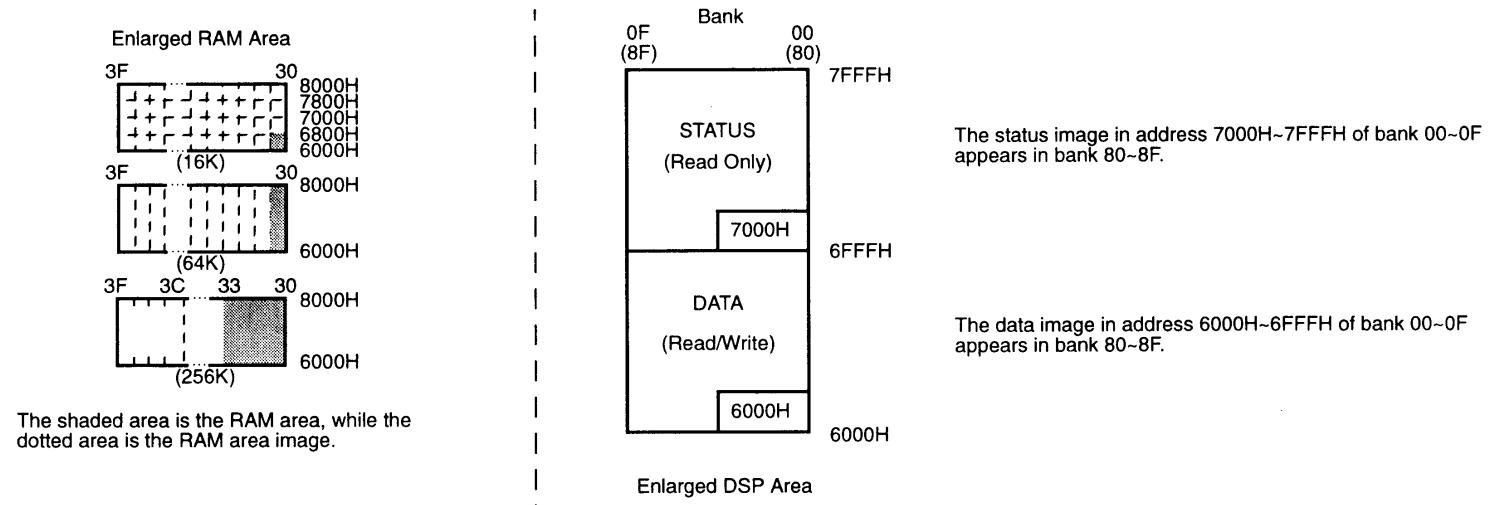
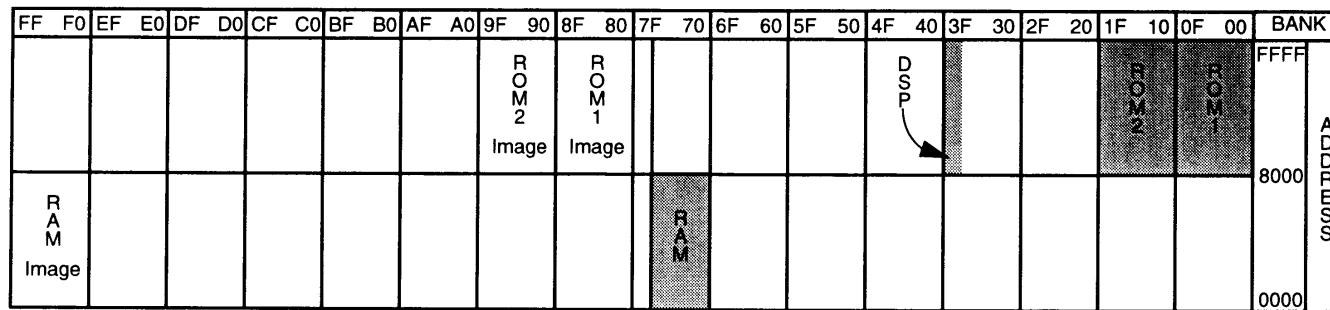


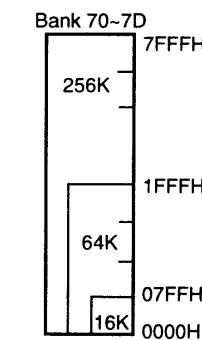
Figure 2-21-9 SHVC-4QW5B PCB MEMORY MAP Mode 20 (4M x 2 pcs) ROM Size is 8M or Less

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	1M/2M/4M/8M	1/2/4/6/8/12/ 16/24/32M	None/16K/64K/256K	DSP

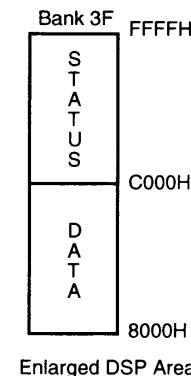


The RAM image appears in bank F0-FF, but do not use this area in the high speed mode.  
Ensure that the DSP area is switched to bank 3F when the ROM size is 8M or less during Mode 20 operation. (DSW1 [8] should be On.)

1-2-33



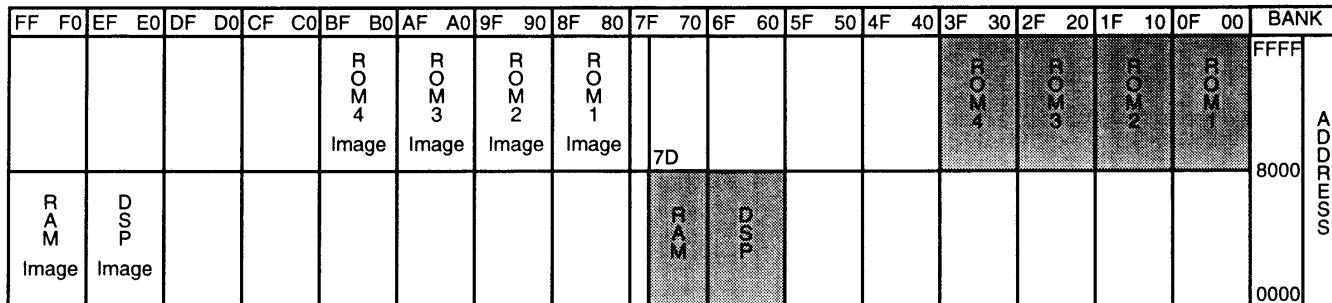
The image of bank 70 is appears in address 0000H~7FFFH of bank 71~7D and F0~FF.



Note: Use C000H/8000H of bank 3F for a port to read from and write to DSP when the ROM size is 8M or less.

Figure 2-21-10 SHVC-4QW5B PCB MEMORY MAP Mode 20 (4M x 4 pcs) ROM Size is 12M or Greater

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	1M/2M/4M/8M	1/2/4/6/8/12/ 16/24/32M	None/16K/64K/256K	DSP



The RAM image appears in bank F0~FF, but do not use this area in the high speed mode.

The DSP image appears in bank E0~EF, but do not use this area in the high speed mode.

Ensure that the DSP area is switched to bank 60 when the ROM size is 12M or greater during Mode 20 operation. (DSW1 [8] should be Off.)

1-2-34

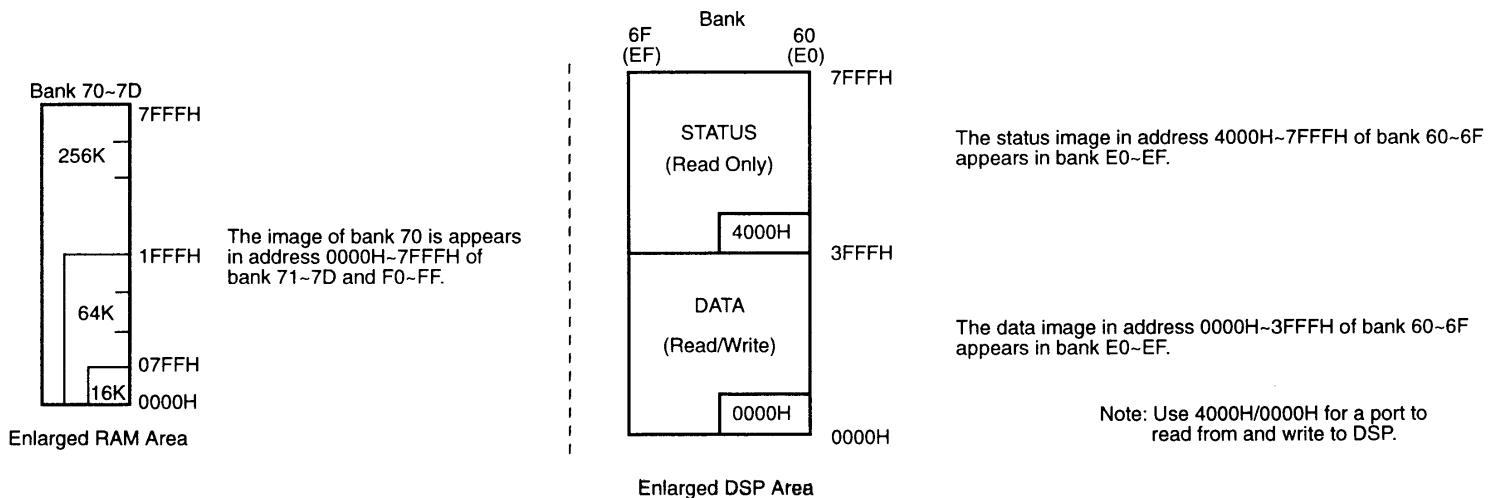
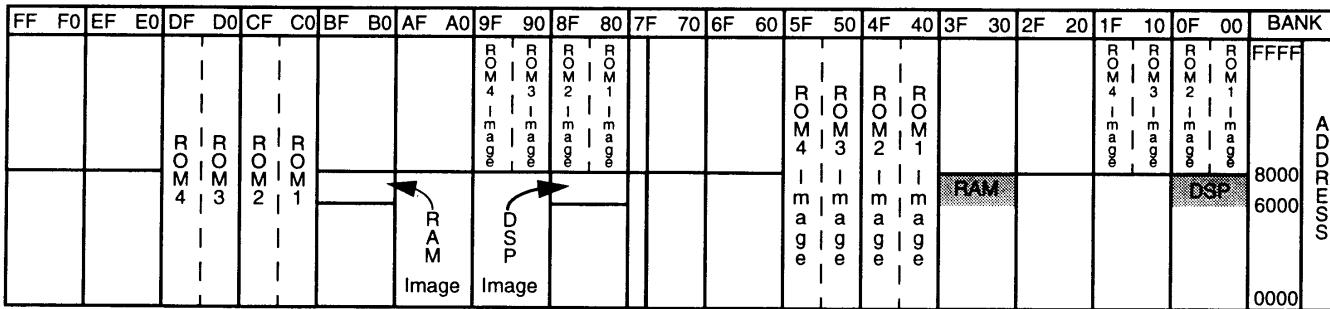


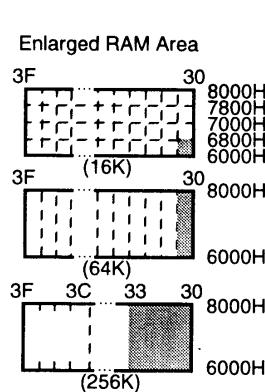
Figure 2-21-11 SHVC-4QW5B PCB MEMORY MAP Mode 21 (4M x 4 pcs)

PCB Configuration	Mapping	Usable EPROMs	ROM Size	Usable RAM	Auxillary Device
	20 or 21	1M/2M/4M/8M	1/2/4/6/8/12/ 16/24/32M	None/16K/64K/256K	DSP

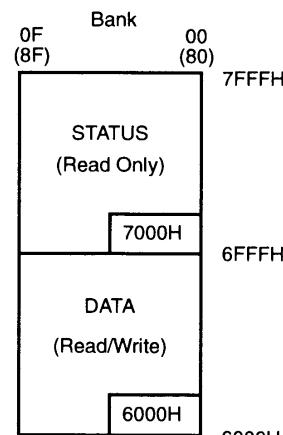


The ROM image in address 8000-FFFF of bank C0-DF appears in bank 00~1F and 80~9F. During Mode 21, the RAM AREA and DSP area do not change even if the ROM size changes.

1-2-35



The shaded area is the RAM area, while the dotted area is the RAM area image.



The status image in address 7000H~7FFFH of bank 00~0F is generated in bank 80~8F.

The data image in address 6000H~6FFFFH of bank 00~0F is generated in bank 80~8F.

Note: Use 7000H/6000H for a port to read from and write to DSP.

## Super NES EPROM Selection Tables

ROM SIZE		1M~8M	4M~16M	4M~16M	1M~32M	4M~32M	8M~64M	1M~8M	4/8/16M	1M~32M	4M~32M	COMMENT FIELD
MODE(S)		20	20/21	20/21	20/21	20/21	20/21/25	20/DSP	20/21/DSP	20/21/DSP	23/SA-1	
STATIC RAM SIZE	None/64K	①	—	—	—	—	—	—	—	—	—	w/back-up
	None/16/64/256K	—	②	—	⑧	④	⑩	⑤	⑥	⑦	—	
	None/512K/1M	—	—	③	—	—	—	—	—	—	⑨	

PCB ASSY		EPROM USED*	REMARKS
①	SHVC-2P3B ASSY	27C1001/27C2001/27C4001	64K SRAM Installed
	Cartridge Evaluation Kit (SHVC-2P3B)	Same as above	25 Units per Kit
②	SHVC-4PV5B ASSY	27C4001/27C8001	Up to 24M by changing PLD
	Cartridge Evaluation Kit (SHVC-4PV5B)	Same as above	25 Units per Kit
③	SHVC-4PV7B ASSY	27C4001/27C8001	Up to 24M by changing PLD 1M SRAM Installed
④	SHVC-8PV5B ASSY	27C4001/27C8001	
⑤	SHVC-2Q5B ASSY	27C1001/27C2001/27C4001	
⑥	SHVC-2QW5B ASSY	27C4001/27C8001	
⑦	SHVC-4QW5B ASSY	27C1001/27C2001/27C4001/27C8001	
⑧	SHVC Multi-Checker 2021	27C1001/27C2001/27C4001/27C8001	256K SRAM Installed
⑨	SHVC-8X7B.ASSY	27C4001	1M SRAM Installed
⑩	SHVC-8PV5B.ASSY-64M	27C8001	

\*Note: Use EPROM listed above or one with the same pin locations.

## SHVC Cartridge List (20 Map, Production Type)

ROM size	2M	4M	8M	10M	12M	16M	20M	24M	32M	specification
SRAM size	No SRAM	○	○	○	○	○	○	○	○	
	16K	○	○	○	○	○	○	△	△	
	64K	○	○	○	○	○	○	○	○	
	256K	○	○	○	○	○	○	△	△	
	512K	○	○	○	△	△	○	△	△	
	1M	○	○	○	△	△	○	△	△	

[for DSP (77C25)]

ROM size	2M	4M	8M	10M	12M	16M	20M	24M	32M	specification
SRAM size	No SRAM	○	○	○	△	△	△	△	△	
	16K	△	△	△	△	△	△	△	△	
	64K	○	○	○	○	○	○	△	△	
	256K	○	○	○	△	△	△	△	△	

○ : Now available.

○ : In development. Please submit "Price Quote Request for Super NES Cartridge" to the Licensing Department of NOA.

△ : No plan for development at this time. If necessary, please submit "Price Quote Request for Super NES Cartridge" to the Licensing Department of NOA five months prior to the release date.

Note: Back-up RAM sizes of 512K bit and 1M bit are under development. If required, contact NOA Licensing Department.

## SHVC Cartridge List (21 Map, Production Type)

ROM size	2M	4M	8M	10M	12M	16M	20M	24M	32M	specification
SRAM size	No SRAM	○	○	○	○	○	○	○	○	
	16K	○	○	○	△	○	○	△	△	
	64K	○	○	○	○	○	○	○	○	
	256K	○	○	○	○	○	○	△	△	
	512K	△	△	△	△	△	△	△	△	
	1M	△	△	△	△	△	△	△	△	

[for DSP (77C25)]

ROM size	2M	4M	8M	10M	12M	16M	20M	24M	32M	specification
SRAM Size	No SRAM	○	○	○	△	△	○	△	○	
	16K	○	○	○	○	○	○	△	△	
	64K	△	△	△	○	○	○	△	△	
	256K	△	△	△	△	△	△	△	△	

○ : Now available.

○ : In development. Please submit "Price Quote Request for Super NES Cartridge" to the Licensing Department of NOA.

△ : No plan for development at this time. If necessary, please submit "Price Quote Request for Super NES Cartridge" to the Licensing Department of NOA five months prior to the release date.

Note: Back-up RAM sizes of 512K bit and 1M bit are under development. If required, contact NOA Licensing Department.

**THIS PAGE INTENTIONALLY  
LEFT BLANK**

## Price Quote Request for Super NES Cartridge

Please send this form to Nintendo of America Inc. Attn.: Juana Tingdale, Licensing Department by Fax at (206) 861-2173.

Date(M/D/Y)	/ /	Licensee	
Release Date(M/D/Y)	/ /	Game Title	
Quantity		Contact	
<b>Specification</b>		Telephone No.	
<p>&lt;Map Mode&gt; 20 map / 21 map / to be determined (please circle)</p> <p>&lt;ROM size&gt; _____ M Bit</p> <p>&lt;RAM Specification&gt; _____ Bit / without RAM</p> <p>&lt;Backup&gt; Yes / No</p> <p>&lt;Co-processor&gt; DSP1 / μPD77C25 (original program) / other DSP ( ) OBC1 Super FX</p> <p style="text-align: center;">(please circle)</p>			
Others: Please specify if you are inquiring other than standard specification.			

<input type="checkbox"/> 任天堂記入欄		<b>FOR NCL USE ONLY</b>			業務部
⑨ 業務課受領者		業務課受領者	技術課担当者	部長	
⑩ 技術課担当者					
⑪ 設計担当部署					
<備考欄>					
					設計担当部
<コメント>		⑫ 予想開発期間 平成 年 月 日 頃			
		受領署	設計担当	検印	検印
※ E P R O M 基板の開発予定等 ..					

## **Chapter 1. Introduction**

The following is a brief discussion of basic concepts used to display game characters on the home television set. Even if you have developed software for the Nintendo Entertainment System (NES), please review this information.

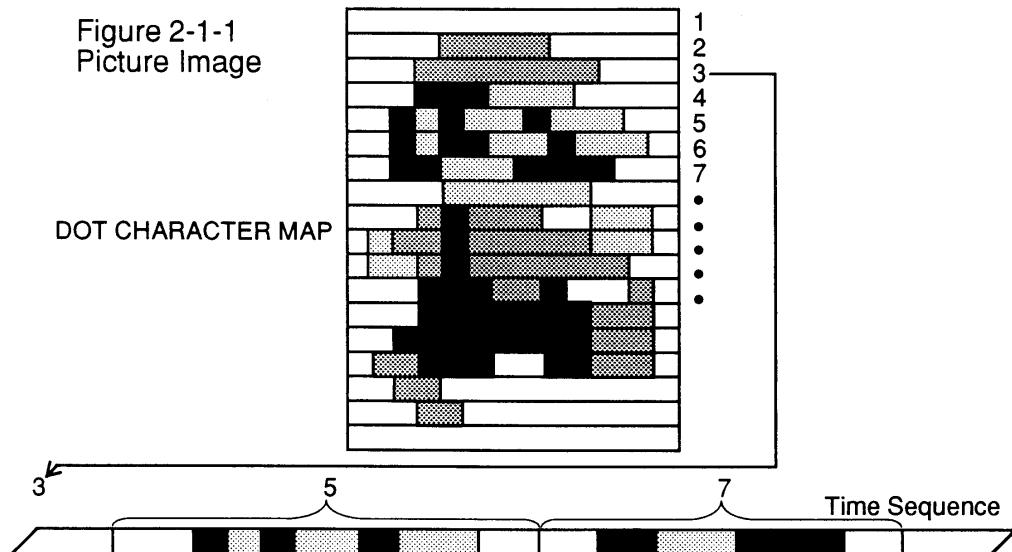
### **1.1 PICTURE IMAGE GENERATION**

The picture on a color television set consists of 525 horizontal lines with each line having color information. The broadcasting station breaks the picture into lines as shown in the figure below.

The odd numbered lines are converted to electronic signals from the top to the bottom of the screen. The remaining even numbered lines are converted from the top to the bottom in the same way.

This method, in which a trace is generated and displayed for every other line, is called the "INTERLACE" method. The electronic signal which has been transmitted is converted to a light signal and will create traces on the television screen in the same order generated.

The act of tracing light on the screen is called "scanning". The period while scanning the odd numbered lines is called the "1st field". The period while scanning the even numbered lines is called the "2nd field." A scan period on the screen is called "one frame". During the period of one frame, the first and second fields are displayed in sequence. Because 1/60 of a second is required to produce one field, 1/30 of a second is required to produce one frame. Therefore, a certain point on the screen is radiated only every 1/30th of a second. Due to the afterimage seen by the human eye and the luminescence of the CRT, the picture does not normally appear to flicker.



(NCL PG 2)

## 1.2 SUPER NES DISPLAY

The picture display on the Super Nintendo Entertainment System (Super NES) has two modes. One is an interlace mode, based on the television system. The other is a non-interlace mode, in which one frame takes 1/60th of a second. In the non-interlace mode the same position is scanned every field. Each frame consists of only 262 lines, half that of the interlace mode. There appears to be no flickering compared to the interlace mode, since each point on the screen is radiated every 1/60th of a second.

## 1.3 BLANKING

The screen is scanned from left to the right and from top to bottom (see Figure 1-1-2). After scanning the screen from left to right, horizontal blanking occurs to prevent the electron beam from being seen as it returns to the left side of the screen. When the beam reaches the bottom right hand side of the screen, vertical blanking occurs to allow the beam to reposition at the top left of the screen without being seen. The NES and the Super NES use this blanking efficiently to display the various movements of characters.

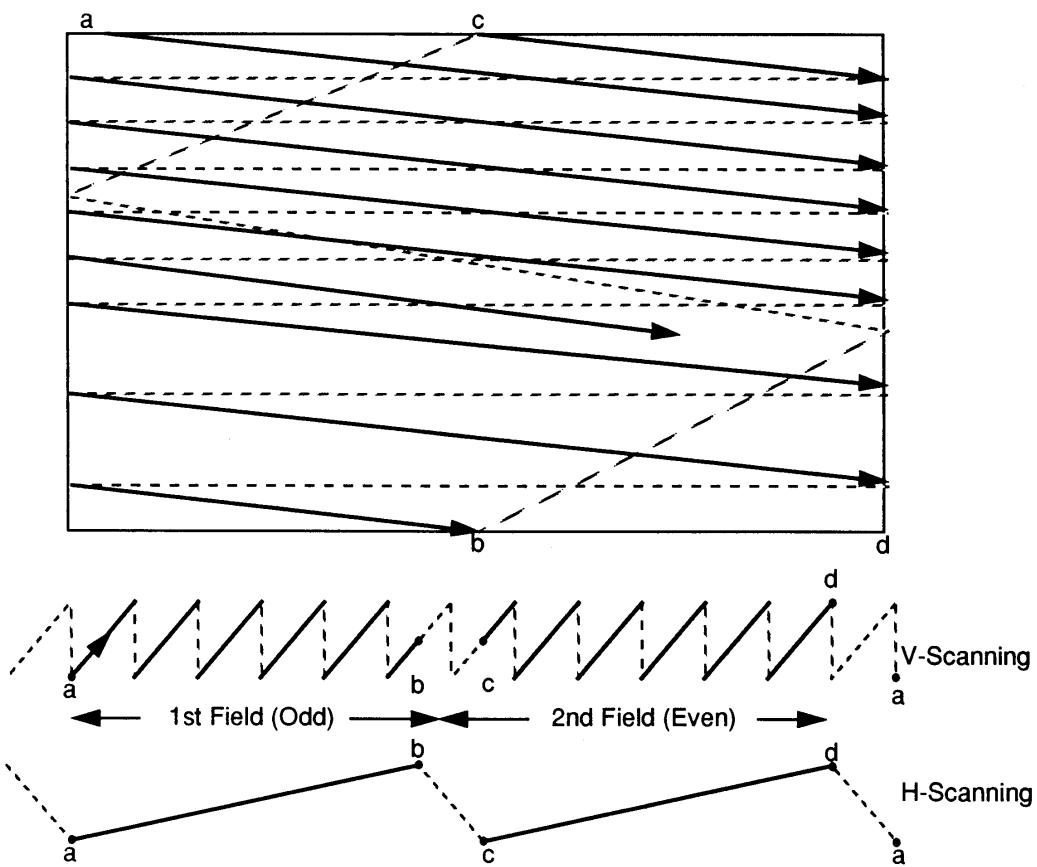


Figure 2-1-2 - Scanning Pattern for Interlace

(NCL PG 3)

## ***Chapter 2. Object (OBJ)***

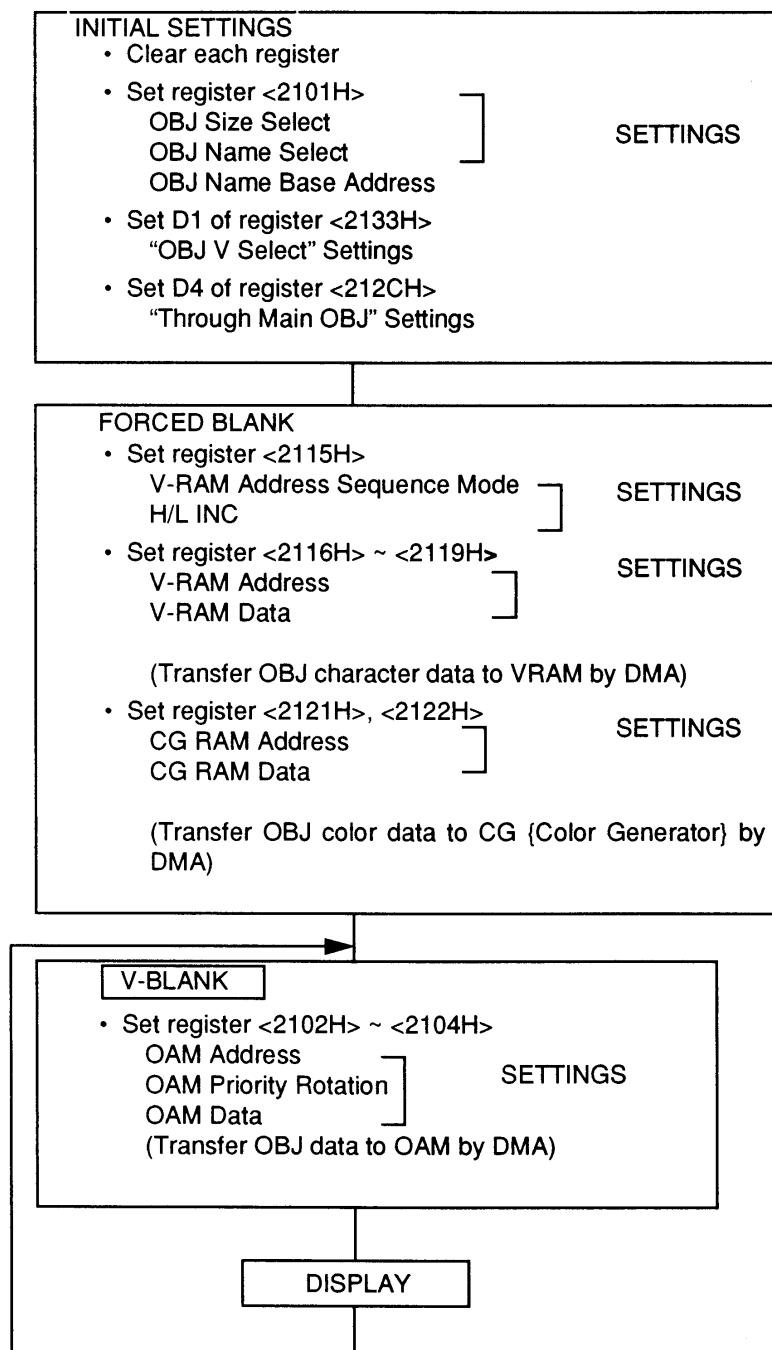
### **2.1 OUTLINE**

This function can display an object in a certain position on the screen. The characters, such as the UFO or the missile of a space game, look like they are moving. If the character's picture is replaced at the same time the point is moved, animation effects can occur (such as "Mario" character looking like he is walking).

### **2.2 FUNCTION**

The maximum number of OBJ's that can be displayed on the screen is 128 and there are four sizes. Two sizes can be selected in one frame and one size can be selected for each OBJ. There are 8 color pallets for OBJ's and one pallet can be selected for each OBJ. One color pallet has 16 color codes out of 32,768 colors. Therefore, each OBJ in the picture is drawn by 16 colors. Each of the 128 objects that may be displayed on the screen at one time has its own priority order, which will decide the display priority if 2 or more OBJ's are overlapped. In addition, there is the Flip function of "up-down," and "left-right," "BG Priority Order" and the "Priority Order" shifting function.

## 2.3 SETTING EXAMPLE



**CAUTION:** It is prohibited to write "100H" to the "OAM H-position (9-bit)  
(Refer to page A-4)

## ***Chapter 3. Background (BG)***

### **3.1 OUTLINE**

The background for OBJ, such as Mario, can be displayed on the screen and scrolled up, down, left or right. This helps the game effect.

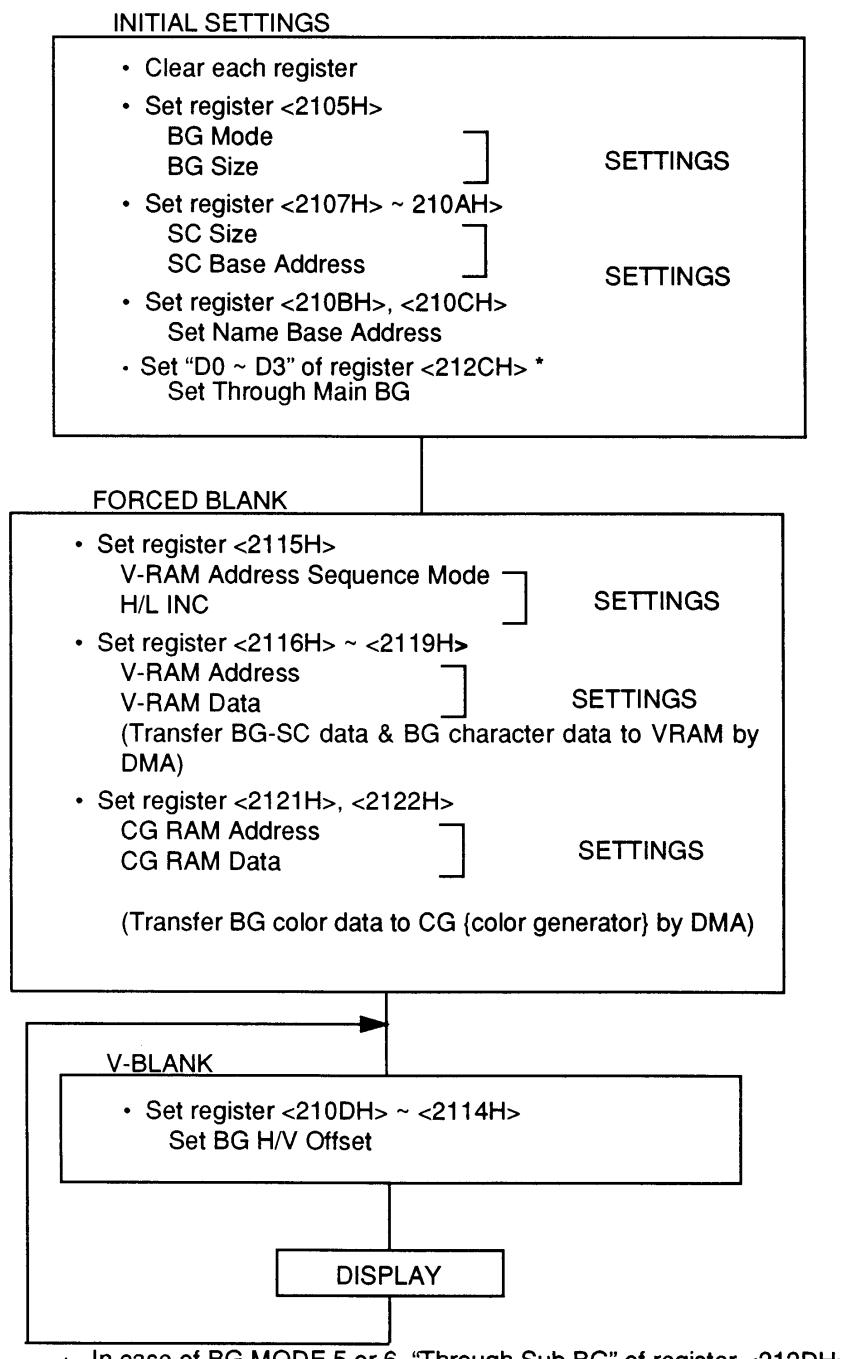
### **3.2 FUNCTION**

There are 8 kinds of BG mode. In BG mode 0 thru 6, there is a difference depending on the combination of numbers of screens, the numbers of the cell color, the resolution and the offset function. There are 4 screens provided and the number of the cell colors are 4 to 256. There are 3 kinds of the resolution selected from 256-dot x 224-dot, 512-dot x 224-dot, or 512-dot x 448-dot. The character size can be set "8-dot x 8-dot" or "16-dot x 16-dot" on each screen.

The offset value (scroll coordinate) can be set on each BG screen and the offset value can be changed every horizontal character unit, depending on the mode, so that the vertical partial scroll can be made. Eight pallets can be used per character, and H-Flip or V-Flip is available per character. Also, the priority order of BG and OBJ can be changed per character. (Refer to page A-19)

Mode-7 is a screen, which can rotate, enlarge or reduce. There are other functions for BG, such as mosaic, window, fixed color addition/subtraction, screen addition/subtraction, and H-Pseudo 512.

### 3.3 SETTING EXAMPLE



\* In case of BG MODE 5 or 6, "Through Sub BG" of register <212DH> should also be set

## Chapter 4. Mosaic

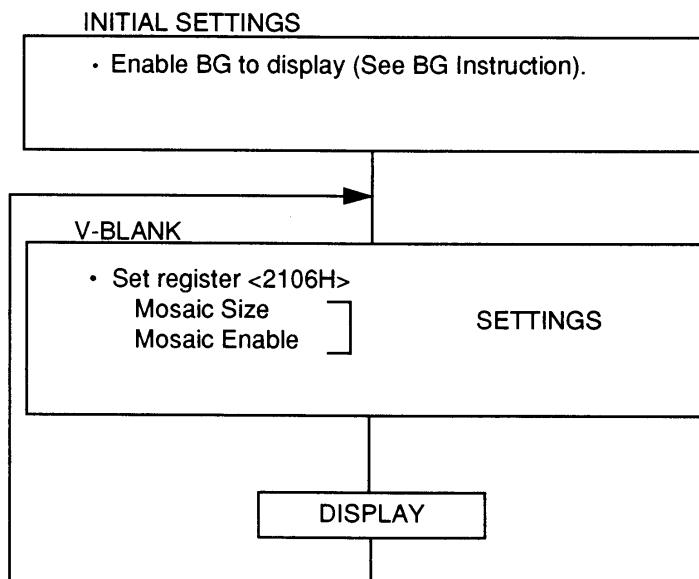
### 4.1 OUTLINE

The purpose of this function is to change BG screen to mosaic design and shade off a picture (refer to page A-7).

### 4.2 FUNCTION

A picture element of mosaic design can be changed to 15 sizes and a mosaic design can be selected for BG screen.

### 4.3 SETTING EXAMPLE



(NCL PG 8)

## *Chapter 5. Rotation/Enlargement/Reduction*

### **5.1 OUTLINE**

In the BG Mode-7 function, more animation effects, are available to the screen through rotation, enlargement or reduction, and a scroll function.

### **5.2 FUNCTION**

#### **5.2.1 TYPE I**

There are 256 character numbers (8-dot x 8-dot size). Each dot can be one of the 256 colors, from a selection of 32,768 colors. In EXTBG mode, each dot can be one of 128 colors from a selection of 32,768 colors and each dot can have priority order. In this function, it is possible to scroll up, down, to the left or right. The center coordinate of rotation, enlargement and reduction can be set at a point either outside or inside of the display area. The rotation angle and vertical or horizontal magnification values are changeable. Also, horizontal flip and vertical flip on the display area are possible. In case the display area goes beyond the screen area, one of three choices can be selected in order to display the excess portion:

- 1) the back drop color
- 2) a single character (CHR# 0)
- 3) repetition (wrap) of the screen area

#### **5.2.2 EXTBG MODE(TYPE II)**

EXTBG mode is originally provided as a function for the purpose of the LSI BG expand. For the Super NES, this function is used for rotation, enlargement and reduction in 128 colors with priority order..

## 5.3 SETTING EXAMPLE

### INITIAL SETTINGS

- Clear each register
  - Set register <2105H><sup>1</sup>  
BG Mode - 7 Settings
  - Set register <212CH><sup>2</sup>  
Through main BG settings
  - Set register <211AH>
    - Screen Flip
    - Screen Over
- SETTINGS
1. On EXTBG mode, EXT input of register <2133H> needs to be set.
  2. Normally, BG1 should be set, but BG 2 should be set on EXTBG mode.

### FORCED BLANK

- Set register <2115H>
    - V-RAM Address Sequence Mode
    - H/L INC
  - Set register <2116H> ~ <2119H>
    - V-RAM Address
    - V-RAM Data

(Transfer BG-SC data to lower address of V-RAM and character data to upper address of V-RAM by DMA)
  - Set register <2121H>, <2122H>
    - CG RAM Address
    - CG RAM Data

(Transfer BG color data to CG {color generator} by DMA)
- SETTINGS
- SETTINGS
- SETTINGS

### V-BLANK

- Set register <210DH>, <210EH>  
“BG 1 H/V Offset” Settings
- Set register <211BH> ~ <211EH>  
“Matrix Parameter” Settings
- Set register <211FH>, <2120H>  
“Center Position” Settings

### DISPLAY

(NCL PG 10)

## Chapter 6. Window (Window Mask)

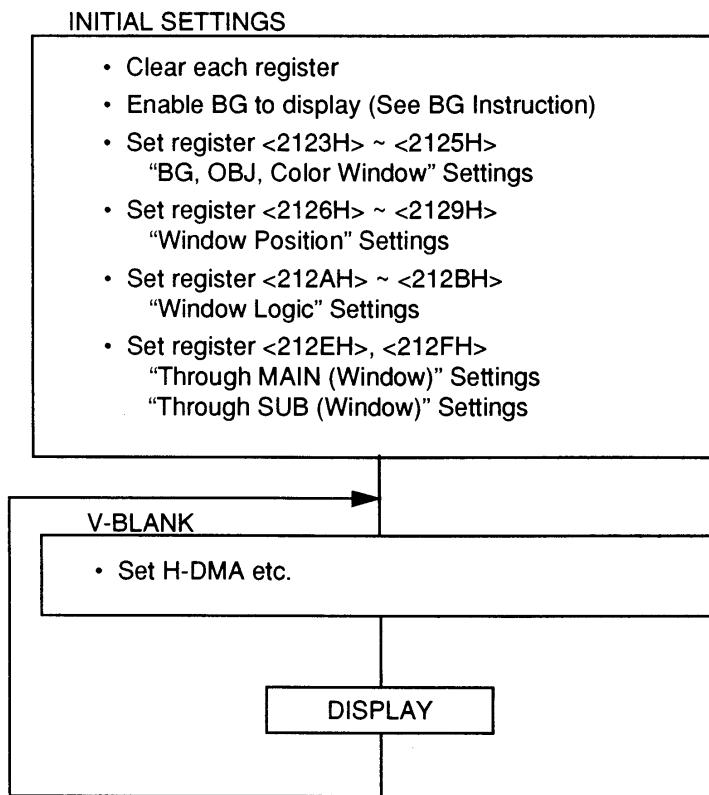
### 6.1 OUTLINE

This function limits the display area on the TV screen for BG and OBJ. This window can be set on the TV screen. BG and OBJ can be displayed inside or outside of this area.

### 6.2 FUNCTION

There are 2 windows. Each window can affect either the BG screen or OBJ and can be either internal or external masked. Four types of window mask logic (OR, AND, XOR and NXOR) can be selected for each BG and OBJ, using 2 kinds of windows simultaneously (refer to "Mask Logic Settings for Window 1 & 2" under "PPU Registers"). If this function is combined with the function of H-DMA, various shapes of the window will be formed, such as; a round shape, heart shape, or star shape. It is also possible to use this function combined with the screen addition/ subtraction and fixed color addition functions.

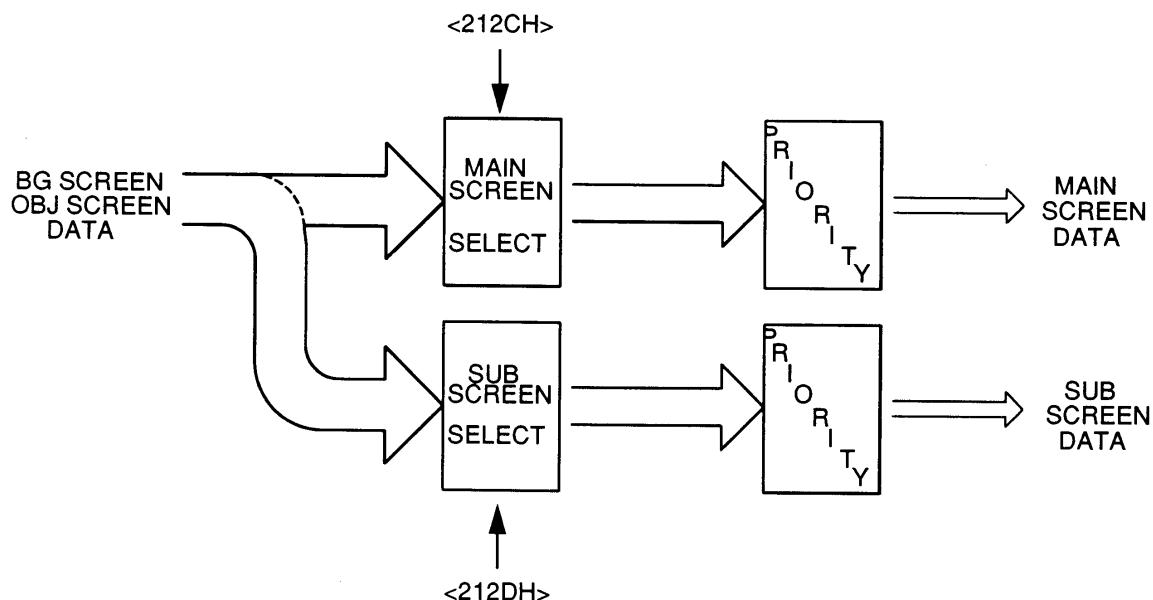
### 6.3 SETTING EXAMPLE



(NCL PG 11)

## Chapter 7. Main/Sub Screen

When displaying several BG and OBJ screens, the picture to be displayed in the overlapped portion is decided by two paths. One of them is called the main screen and the other is called the sub screen. The screen to be used for the main and sub screens can be selected by registers <212CH> and <212DH>. Furthermore, the data for the main and sub screens to be displayed is made according to the priority order. Unless the addition/subtraction screen is done as follows, the "Main SW" of the "Color Window" in register <2130H> is normally on, and the "Sub SW" is normally off so that only the main screen is displayed (see page A-23).



## 7.1 SCREEN ADDITION/SUBTRACTION

### 7.1.1 OUTLINE

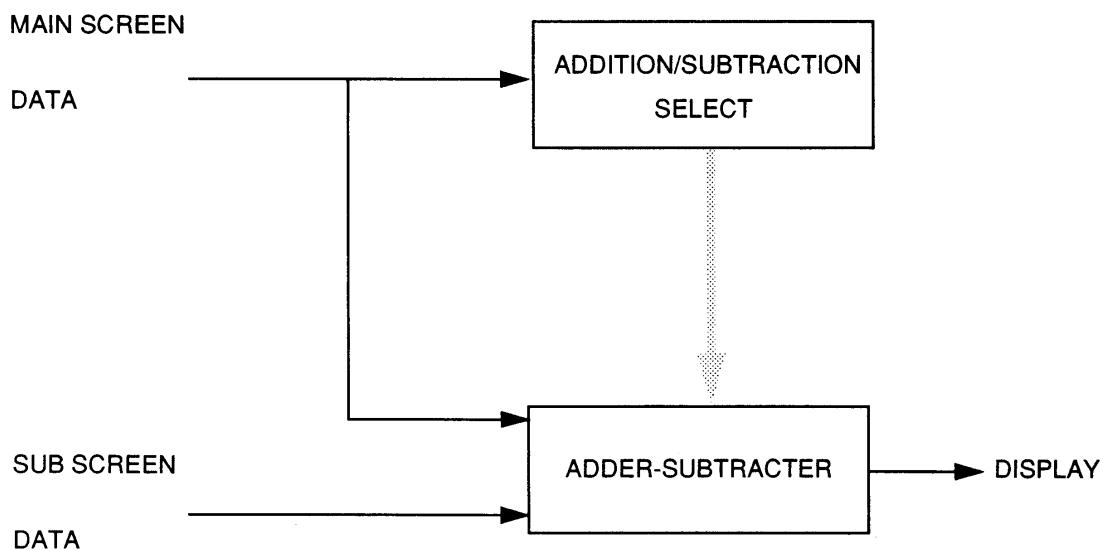
This function is the addition (Overlapping Light) or the subtraction (Lens Filter) for the main screen and the sub screen in order to have the effect of transparency.

### 7.1.2 FUNCTION

This function displays the result after the addition or subtraction of RGB data on the main screen and sub screen. This function can also select BG screen or OBJ data on the main screen to be added to or subtracted from the sub screen, similar to the figure below. However, when there is no screen data on the sub screen (screen is clear), the color constant explained on page 1-7-4 will be added or subtracted.

When the result of addition or subtraction exceeds 31, the value becomes 31. When the result of addition or subtraction is less than 0, the value becomes 0.

Please do not use this function on BG mode 5 or 6.



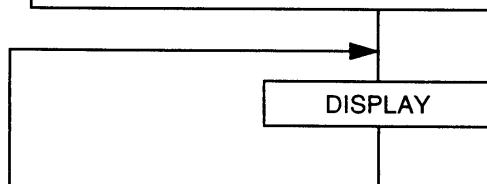
(NCL PG 13)

### 7.1.3 SETTING EXAMPLE

#### INITIAL SETTINGS

- Clear each register
- Enable BG to display (see BG instruction)
- Enable OBJ to display (see OBJ instruction)
- Set D1 of register <2130H>  
“CC ADD Enable” Settings
- Set register <2131H>  
ADD or SUB Enable  
1/2 Enable  
ADD/SUB
- Set register <212CH>  
“Through Main” Settings
- Set register <212DH>  
“Through Sub” Settings

SETTINGS



**NOTE:** When the main screen data is the OBJ, it will be added to or subtracted from the sub screen data only for the OBJ of the pallet code (4 to 7).

**NOTE:** When “1/2 Enable” of register <2131H> is enabled, the addition/subtraction result of each RGB becomes 1/2.

## 7.2 COLOR CONSTANT ADDITION/SUBTRACTION

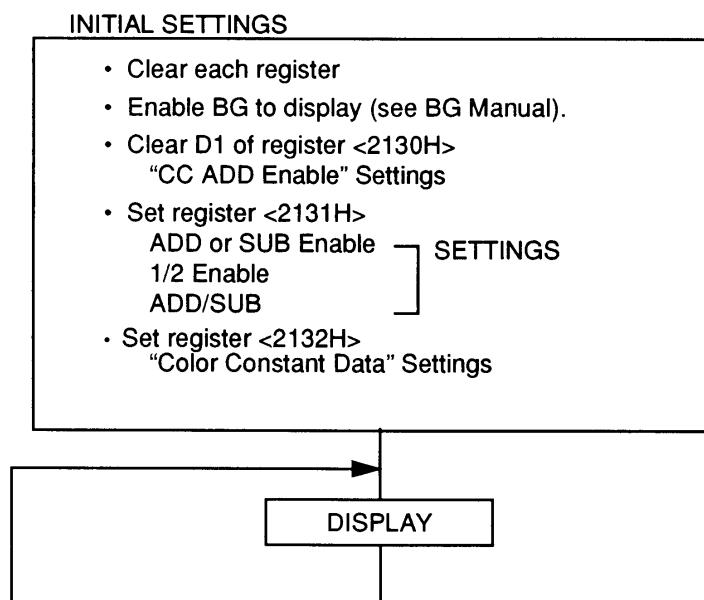
### 7.2.1 OUTLINE

This function can perform addition (overlapped light) or subtraction (lens filter) with RGB value (color constant) set by the main screen and register <2132H>. This will change the color on the display area.

### 7.2.2 FUNCTION

This function can perform addition/subtraction by using the RGB value (color constant) which is set by register <2132H> instead of the sub screen of the addition/subtraction screen described previously.

### 7.2.3 SETTING EXAMPLE



(NCL PG 15)

## 7.3 COLOR WINDOW (Combination of Window & Addition/Subtraction)

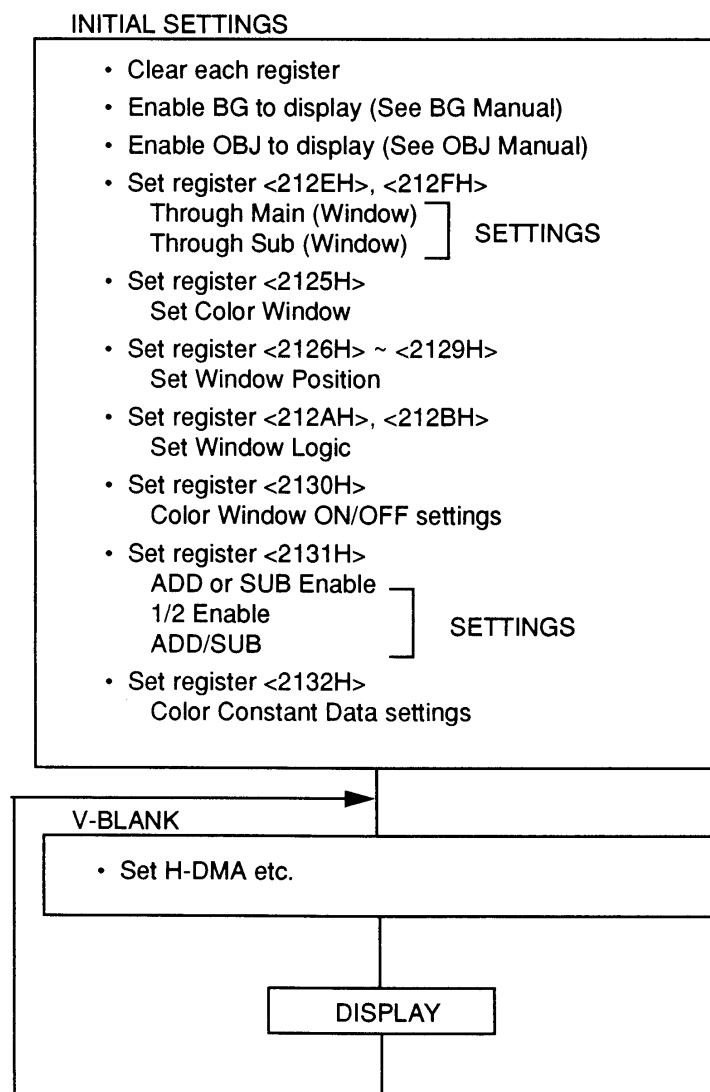
### 7.3.1 OUTLINE

The Screen Addition/Subtraction or the Color Constant Addition/Subtraction can be performed inside or outside the window (only one or the other).

### 7.3.2 FUNCTION

This function can select what portion of the window should be displayed and added or subtracted on each main screen and sub screen. The following is the function of window, the screen addition/subtraction and the color constant addition/subtraction.

### 7.3.3 SETTING EXAMPLE



(NCL PG 16)

## ***Chapter 8. CG Direct Select***

### **8.1 OUTLINE**

On BG-1 in Mode 3, 4 and 7, the character data can be used as the color data without using CG-RAM color data. BG-1 can be displayed using 2048 colors on Mode 3 and 4, and 256 fixed colors on Mode 7. BG-2 and OBJ can use the CG-RAM color data without being limited to the color data on BG-1.

### **8.2 FUNCTION**

When BG-1 on Mode 3, 4 and 7 is displayed on the TV screen, this function will display 8-bit color data per character dot without using the CG-RAM. The CG-RAM data is used for the objects and other background screens.

### **8.3 SETTING EXAMPLE**

- Enable BG to display (See BG Instruction)
- Set "D0" of register <2130H>  
"Direct Select" Settings

NOTE: See page A-17 for color data.

## Chapter 9. H-Pseudo 512

### 9.1 OUTLINE

In modes other than 5 and 6, this function provides gradation between 2 dots which are next to each other horizontally, which changes the color smoothly.

### 9.2 FUNCTION

This function utilizes screen addition/subtraction. The color constant addition/subtraction can not be done at the same time that this function is performed.

### 9.3 SETTING EXAMPLE

- Enable BG to display (see BG instruction)
- Set "D3" of register <2133H>  
"Pseudo 512" settings
- Set register <212CH>, <212DH>  
Through Main       SETTINGS  
Through Sub
- Set D1 of register <2130H>  
"CC ADD Enable" settings
- Set register <2131H>  
ADD or SUB Enable  
1/2 Enable       SETTINGS  
ADD/SUB

## ***Chapter 10. Complementary Multiplication (Signed Multiplication)***

### **10.1 OUTLINE**

The 2's complement multiplication will be performed with high speed. For example, to calculate the rotation parameter in mode 7, it will lighten the burden of the CPU processing.

### **10.2 FUNCTION**

The high speed multiplication of 16-bit (2's complement) and 8-bit (2's complement) will be performed with "no-wait," and the result becomes 24-bit (2's complement).

### **10.3 SETTING EXAMPLE**

- Set BG other than MODE-7 (or V-Blank/Forced Blank)  
(Except during V-Blank or Forced Blank period)
- Write lower 8-Bit (Multiplicand) to register <211BH>: (Input)
- Write higher 8-Bit (Multiplicand) to register <211BH>: (Input)
- Write register 8-Bit (Multiplier) to register <211CH>: (Input)
- Read register <2134H> ~ <2136H>: (Result)

## ***Chapter 11. H/V Counter Latch***

### **11.1 OUTLINE**

This function is used for synchronization of process timing by tracking the scanning beam on the screen.

### **11.2 FUNCTION**

This function sets the vertical and horizontal counter value (when register <2137H> is read) and tracks the raster beam on the screen by reading the register value. (The scanning is synchronized with an internal vertical and horizontal counter.)

### **11.3 SETTING EXAMPLE**

- Read register <2137H>: (counter latch)
- Read register <213FH>  
(Initialize register <213CH>, <213DH> in the order of Low and High)
- Read register <213CH>, <213DH>

## ***Chapter 12. Offset Change***

### **12.1 OUTLINE**

The horizontal and vertical scroll (offset) value can be performed every horizontal 8-dot (character unit) in mode 2, 4, and 6. The other part of the screen can be brought into the middle of the frame in order to have the effect of a window. A partial vertical scroll can also be made.

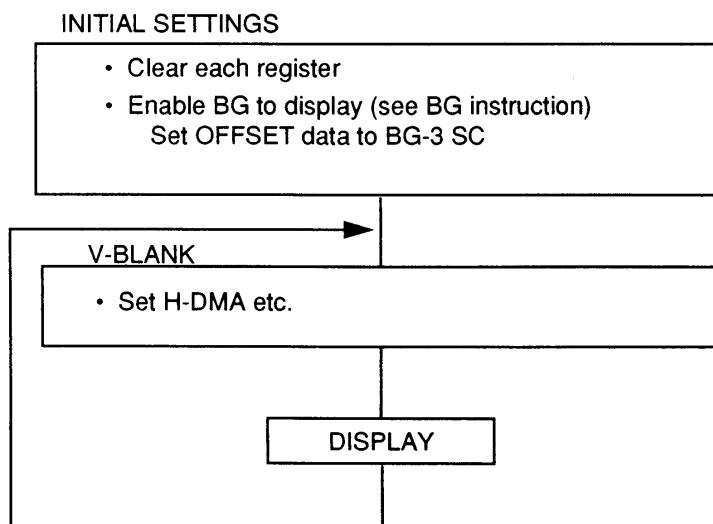
### **12.2 FUNCTION**

This function can be used in any of the three ways, listed below.

- Affect BG-1 only
- Affect BG-2 only
- Affect both BG-1 and BG-2

The offset for both H and V can be changed at every character unit on mode 2 and 6, but the offset for either H or V (only one or the other) can be changed on mode 4. The same offset will be performed on each line once the offset data for a horizontal line (32 characters) is set. To change the setting of the other offset value, depending on the scanning line, change "BG-3 SC Offset Address" or "BG-3 SC Base Address" during the H-DMA period.

### **12.3 SETTING EXAMPLE**



(NCL PG 21)

## ***Chapter 13. Standard Controller***

### **13.1 OUTLINE**

The switch status of the standard controller can be read automatically in serial order and will be converted to parallel data.

### **13.2 FUNCTION**

Two standard controllers can be connected to the Super NES. Four standard controllers may be connected by using an expanded connector, such as MultiPlayer 5 (refer to "Accessories"). Single bit data is assigned to each switch. Up to 16 bits can be read automatically for one standard controller. The expanded bit data can be read 1 bit at a time by the software, as for the NES. The hardware reads the data for about 215  $\mu$ s after the V Blank flag is set or NMI is applied. During this data read period, the standard controller register cannot be read properly.

- 215 (214.55)  $\mu$ s is equivalent to 3.4 (3.38) scanning lines; a period of 580 (576) bytes to be transferred by DMA. (If the CPU clock is 2.68 MHz, it is equivalent to 580 machine cycles.) As soon as V-Blank starts, normal flow is to perform general purpose DMA. Therefore, it is convenient if the total number of bytes to be transferred by general purpose DMA is used for read timing. (Please refer to the System Flowchart.)
- The standard controller data (register) should be read after confirming that "JOY-C Enable" of register <4212H> is not set during the V-Blank period, so that valid data can be read.
- After the 18  $\mu$ s (48 machine cycles with 2.68 MHz) from the beginning of V-Blank, the hardware will start to read. "Standard CNTRL Enable" of register <4212H> cannot be set during this period.

### 13.3 SETTING EXAMPLE

#### INITIAL SETTINGS

- Set "1" to "D0" of register <4200H>  
"Standard CNTRL Enable" Settings
- Set "0" to "D0" of register <4016H>

V-BLANK

- Process for DMA (215 µs or more)
- Read register <4218H> ~ <421FH>
- Read register <4016H> & <4017H>  
(If expanded bits exist)

DISPLAY

(NCL PG 23)

## ***Chapter 14. Programmable I/O Port***

### **14.1 OUTLINE**

An 8 bit programmable I/O port is provided for interface to peripheral devices, such as; a keyboard, the 3D glass, etc.

### **14.2 HOW TO USE**

A "1" should be written to register <4201H> for the bit to be used as the in-port. The selected bit will become the in-port, which can be read by register <4213H>. Output data should be written to the bit of register <4201H> to be used as the Out-port. This data can be output directly.

★Only 2 of the 8 bits can be used at the connector for the controller (Refer to page 1-28-1).

## ***Chapter 15. Absolute Multiplication/Division***

### **15.1 OUTLINE**

Absolute multiplication (8 bit by 8 bit) and absolute division (16 bit by 8 bit) can be done using this function. It is also convenient for processing arrays of tables and can improve the processing speed for multiplication and division.

### **15.2 FUNCTION**

The multiplication calculation between the multiplicand of an 8 bit absolute value (0 ~ 255) and the multiplier of an 8 bit absolute value (0 ~ 255) can be performed and can provide the result of a 16 bit product (0 ~ 65025). The division calculation between the dividend of a 16 bit absolute value (0 ~ 65535) and the divisor of an 8 bit absolute value (0 ~ 255) can be performed and can provide the result of a 16 bit quotient (0 ~ 65535) and a 16 bit remainder.

If the divisor is "0" in the division calculation, the quotient value becomes 65535 (0FFFFH) and the remainder becomes the dividend value. Therefore, caution is required.

It takes about 8 machine cycles for the multiplication calculation and about 16 machine cycles for the division calculation. The register value for multiplicand and dividend will not be destroyed even after the operation.

### **15.3 SETTING EXAMPLE**

- In case of Multiplication
  - Set register <4202H>  
"Multiplicand-A" Settings
  - Set register <4203H>  
"Multiplier-B" Settings
  - Wait for 8 Machine Cycles
  - Read register <4216H>, <4217H>  
Read Product-C
- In case of Division
  - Set register <4204H>, <4205H>  
"Dividend-C" Settings
  - Set register <4206H>  
"Divisor-B" Settings
  - Wait for 16 Machine Cycles
  - Read register <4214H>, <4215H>  
Read Quotient-A
  - Read register <4216H>, <4217H>  
Read Remainder

## ***Chapter 16. H/V Count Timer***

### **16.1 OUTLINE**

The Super NES has a timer synchronizing with the display on the TV screen, which is used for adjusting the synchronization of the scanning process on the screen and software execution.

### **16.2 FUNCTION**

This function can generate the interrupt at either a V or H position of the scanning lines. It can also generate the interrupt at any position of the scanning line.

### **16.3 SETTING EXAMPLE**

#### INITIAL SETTINGS

- Disable IRQ
- “Set D4 and D5” of register <4200H>  
“Timer Enable” Settings
- Set register <4207H> ~ <420AH>
 

H Count Time	<input type="checkbox"/>	SETTINGS
V Count Time	<input type="checkbox"/>	
- Enable IRQ

#### IRQ PROCESS

- Read “D7” of register <4211H>  
Confirm “Timer IRQ”
- Clear “D4” and “D5” of register <4200H>
- Process for Target Task

DISPLAY

(NCL PG 26)

## ***Chapter 17. Direct Memory Access (DMA)***

The DMA is the method to transfer the data in the same manner as the data transfer which is done by the CPU. However, the DMA can transfer the data at high speeds by using the hardware instead of the CPU. The SNES has the exclusive DMA, since the picture data has to be transferred rapidly.

The DMA for the SNES is to transfer the data between "A-Bus Address" in the CPU (0000000 ~ 0FFFFFF) and "B-Bus Address" in the S-PPU (0002100 ~ 00021FF), which has 8 channels total. There are two kinds of DMA: general purpose DMA and H-DMA. Either can be set at each channel. The data can be transferred between the same DMA's in the order of lower channel numbers (0 ~ 7). The H-DMA can interrupt even during the transfer by the general purpose DMA, which means that the H-DMA has higher priority than the general purpose DMA. Furthermore, the CPU process stops automatically during the DMA period, and will start again after the DMA is completed. It is not necessary to observe the DMA completion by the CPU.

### **17.1 GENERAL PURPOSE DMA**

#### **17.1.1 OUTLINE**

This function can transfer the data rapidly between 2 types of memory devices: memory which can be accessed directly by the CPU, such as a ROM on the game cartridge, and memory which has to be accessed through the S-PPU, such as the V-RAM.

#### **17.1.2 FUNCTION**

The maximum area of the A-Bus address which can be used in one channel is limited in one bank (65,536 Byte). Therefore, in case of spreading over more than 2 banks, it is necessary to use more than 2 channels or transfer twice. One A-Bus address basically is increased every time 1 byte of data is transferred. However, it can be decreased or fixed depending on the settings ("d3" and "d4" of register <43X0H>).

The following table shows four types of B-Bus address changes:

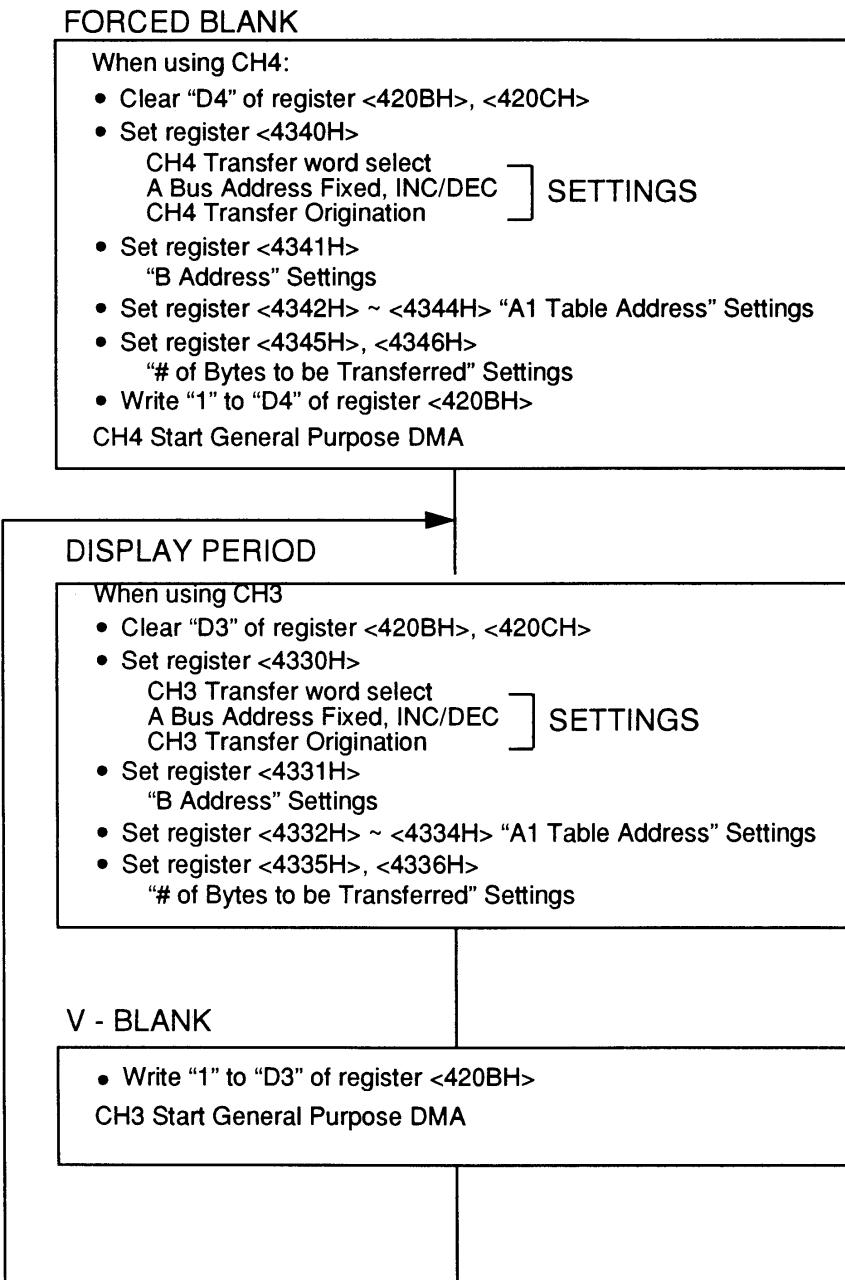
Table 2-17-1 B-Bus Address Changes

Transfer Word Select <43X0H>	D2 ~ D0 000 or 010	D2 ~ D0 001	D2 ~ D0 011	D2 ~ D0 100
# of Transfer (# of Byte)				
0	B	B	B	B
1	B	B + 1	B	B + 1
2	B	B	B + 1	B + 2
3	B	B + 1	B + 1	B + 3
4	B	B	B	B
5	B	B + 1	B	B + 1
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

- In case of 224 lines, general purpose DMA can transfer 6K byte data maximum during V-Blank period.

NOTE: B means the data of register <43X1H>

### 17.1.3 SETTING EXAMPLE



(NCL PG 29)

## 17.2 H-DMA

### 17.2.1 OUTLINE

This is a special DMA which can transfer data automatically, synchronizing with the H-Blank. The S-PPU settings can be varied by each horizontal scan line and special effects can be added to the picture.

### 17.2.2 FUNCTION

This function transfers the data from the A-Bus memory (CPU memory) to the S-PPU register. There are two kinds of addressing modes on the A-Bus side; absolute and indirect addressing. Either type of addressing can be set by each channel. There are two kinds of data transfer. One is to transfer a set of data during each horizontal blanking period. The other is to transfer a set of data every certain number of horizontal blanks.

Table 2-17-2 B-Bus Address Change

Transfer Word Select <43X0H>		D2 ~ D0 000	D2 ~ D0 001	D2 ~ D0 010	D2 ~ D0 011	D2 ~ D0 100
# of Line to be transferred						
1	B	B	B		B	B
		B + 1	B		B + 1	B + 2
					B + 1	B + 3
					B	B
2	B	B	B		B	B + 1
		B + 1	B		B + 1	B + 2
					B + 1	B + 3
•	•	•	•	•	•	•
•	•	•	•	•	•	•
•	•	•	•	•	•	•

NOTE: B means the data of register <43X1H>.

## 17.3 SETTING EXAMPLE

### FORCED BLANK

When using Indirect Addressing (Type 1) with CH0

- Clear "D0" of register <420CH>
  - Set register <4300H>
    - CH0 Transfer word select
    - CH0 TYPE = "1"
    - CH0 Transfer Origination
  - Set register <4301H>
    - "B Address" Settings
  - Set register <4302H> ~ <4304H> "A1 Table Address" Settings
  - Set register <4307H>
    - "CH0 Data Bank" Settings
  - Write "1" to "D0" of register <420CH>
- CH0 Start H - DMA

SETTINGS

### DISPLAY PERIOD

When using Absolute Addressing (Type 0) with CH1

- Clear "D1" of register <420CH>
- Set register <4310H>
  - CH1 Transfer word select
  - CH1 TYPE = "0"
  - CH1 Transfer Origination
- Set register <4311H>
  - "B Address" Settings
- Set register <4312H> ~ <4314H> "A1 Table Address" Settings

SETTINGS

### V - BLANK

- Write "1" to "D1" of register <420CH>
- CH1 Start H - DMA

## ***Chapter 18. Interlace***

### **18.1 BG MODE 0 ~ 4 & 7**

When "1" is written to D0 of register <2133H>, the picture signal output from the Super NES will be the interlace signal. In the case of BG modes 0 through 4 and 7, the same picture will be displayed unless the picture data is changed between the 1st field and the 2nd field. (Refer to BG Screen in Appendix A.)

### **18.2 BG MODE 5 & 6**

When using interlace on BG mode 5 and 6, the vertical resolution will be doubled in appearance. The picture is displayed using a one frame combination of the 1st field and 2nd field. (Refer to BG Screen in Appendix A.)

### **18.3 OBJ**

When "1" is written to "D1" of register <2133H>, the vertical resolution will be doubled as in the case of BG Mode 5 and 6, because a picture is generated using one frame. The range of the V-position for OBJ is 0 through 255 and this range will not be doubled.

## ***Chapter 19. H-512 Mode (BG Mode 5 & 6)***

### **19.1 MAIN SCREEN & SUB SCREEN SETTINGS**

The screen addition/subtraction function should not be used, because a part of both main screen and sub screen functions are used in this mode. With the exception of color constant addition/subtraction, “1” should be written to D4 and D5 of register <2130H> and the sub-switch should be off. The same data should be written to registers <212CH>, <212DH>, <212EH>, and <212FH>. “Through” should be the same for both the main and sub-screens.

### **19.2 FIXED COLOR ADDITION/SUBTRACTION**

D0 ~ D5 of register <2131H> is a flag which can select the main screen for addition/subtraction. Because a part of both main screen and sub screen functions are used, this selection cannot be performed. It is necessary to write “1” to 6 flags (D0 ~ D5) when color constant addition/subtraction is performed. The remaining settings are the same as the normal Color Constant Addition/Subtraction. There will be addition/subtraction every 2 dots, horizontally, in the color window function, because the window has only 256 positions horizontally.

### **19.3 DISPLAY WITH OBJ**

The name H-512 indicates a horizontal resolution of 512 for BG. The horizontal resolution for the OBJ is only 256-dot, regardless of the BG mode. The priority order for BG is determined by every dot.

### **19.4 OTHERS**

See “BG Screen” in the Tables of Appendix for details.

## ***Chapter 20. OBJ 33's Lines Over & Priority Order***

### **20.1 33'S RANGE OVER**

The number of OBJS which can be displayed in a horizontal line is limited. One of these limitations is called the "33's Range Over." This limits the number of OBJS which can be displayed in a horizontal line, regardless of the OBJ size. If "33's Range Over" has occurred in one field (at least one line), "D6" of register <213EH> will be set. For the line in which this "33's Range Over" occurs, only 32 OBJS can be displayed out of 33 or more OBJS present. The 32 OBJS displayed are selected using the priority order (selected from smaller OBJ number).

**NOTE:** "The number of displayed OBJS" counts OBJS hidden by BG window or other OBJS.

**NOTE:** If H-position is minus, and the OBJ is not displayed on the screen area (located on the left of the screen to be displayed), "the number of displayed OBJS" does not count them.

### **20.2 35'S TIME OVER**

The other limitation on the horizontal line is called "35's time over." This limits the number of OBJS (converted to character size 8-dot x 8-dot) that can be displayed. If the "35's Time Over" has occurred in one field (at least one line), "D7" of the register <213EH> will be set. In the line in which this "35's Time Over" has occurred, only 32 of the total OBJS available can be displayed according to the priority order (selected from larger OBJ number). This limit is due to a conversion limit of less than 35 OBJS (8 x 8) displayed per horizontal line. "These 32 OBJS must satisfy the display condition explained in "33's Range Over", above.

NOTE: There are characters (8-dot x 8-dot) which are not displayed on the display area depending on OBJ size and position. But they are not included in this limitation (34 or less).

## 20.3 PRIORITY ORDER SHIFTING

As mentioned above, limited numbers of OBJS can be displayed in a line and are related to the priority order. It is desirable to develop a game within this limitation. However, sometimes OBJS need to be displayed beyond this limitation. This can be accomplished using virtual OBJS. One method is to change the priority order every frame. Another method changes the OBJ data order through programming. The Super NES also contains a function which rotates the priority order of 128 OBJS. When using these methods, consider that the OBJ will flash every frame unit and the priority order among OBJS will change. The method for assignment is as follows:

- Step 1. Display the OBJ.
- Step 2. Write "1" to "D7" of register <2103H>.
- Step 3. Write the highest priority OBJ number (0 ~ 127) to "D1 ~ D7" of register <2102H> during V-Blank period every frame.
- Step 4. Repeat step 3.  
When OBJ number stored in step 3 is "n".

OBJ NUMBER	PRIORITY ORDER
O B J 0	1 2 9 - n
•	•
•	•
•	•
O B J (n - 1)	1 2 8
O B J (n)	1
O B J (n + 1)	2
•	•
•	•
•	•
O B J 1 2 7	1 2 8 - n

(NCL PG 35)

## ***Chapter 21. CPU Clock and Memory Mapping***

### **21.1 CPU CLOCK**

The CPU clock can be switched automatically, depending on the address to be accessed by the CPU. Three clock speeds are available: 3.58MHz, 2.68MHz, and 1.79 MHz. The device speed (ROM, RAM, LSI, etc.) will determine the speed to be used. If a medium speed ROM and RAM (access time less than 200ns) are used in the cartridge, it will be mapped to the address area for 2.68MHz. If high speed (access time less than 120ns) are used, it will be mapped to the address area for 3.58MHz. Please refer to "Frequency & Address Mapping" for the relation between the address and the clock. Two clocks (2.68MHz & 3.58MHz) can be selected by setting D0 of register <420DH> for the range of memory "②" shown in the illustration on the next page. The default setting is 2.68MHz. The CPU is operated internally with a 3.58MHz clock speed. (Regardless of the address, DMA will be performed with 2.68MHz clock speed).

### **21.2 CPU MEMORY MAP**

Please refer to "Frequency & Address Map" on the next page. The WRAM (8K-Byte) is mapped to address (0000 ~ 1FFF) of banks (00 ~ 3F), (80 ~ BF) and 7E. This is the WRAM used as common bank. This 8K-Bytes can be accessed from any bank described above. The WRAM (120K-Byte) is mapped to address (2000 ~ FFFF) of bank 7E and (0000 ~ FFFF) of bank 7F. Therefore, the WRAM (128K-Byte total) is included in the Super NES unit. This 128k-Byte (RAM ①, RAM ②) is one consecutive memory and can be accessed from the B - Bus address. The address "2000 ~ 5FFF" of bank "00 ~ 3F" and "80 ~ BF" are reserved as a register area of the S-PPU, DMA, etc. Because this basically is reserved as a common bank area, the S-PPU and DMA register can be accessed from any bank above.

Figure 2-21-1 Super NES CPU Memory Map

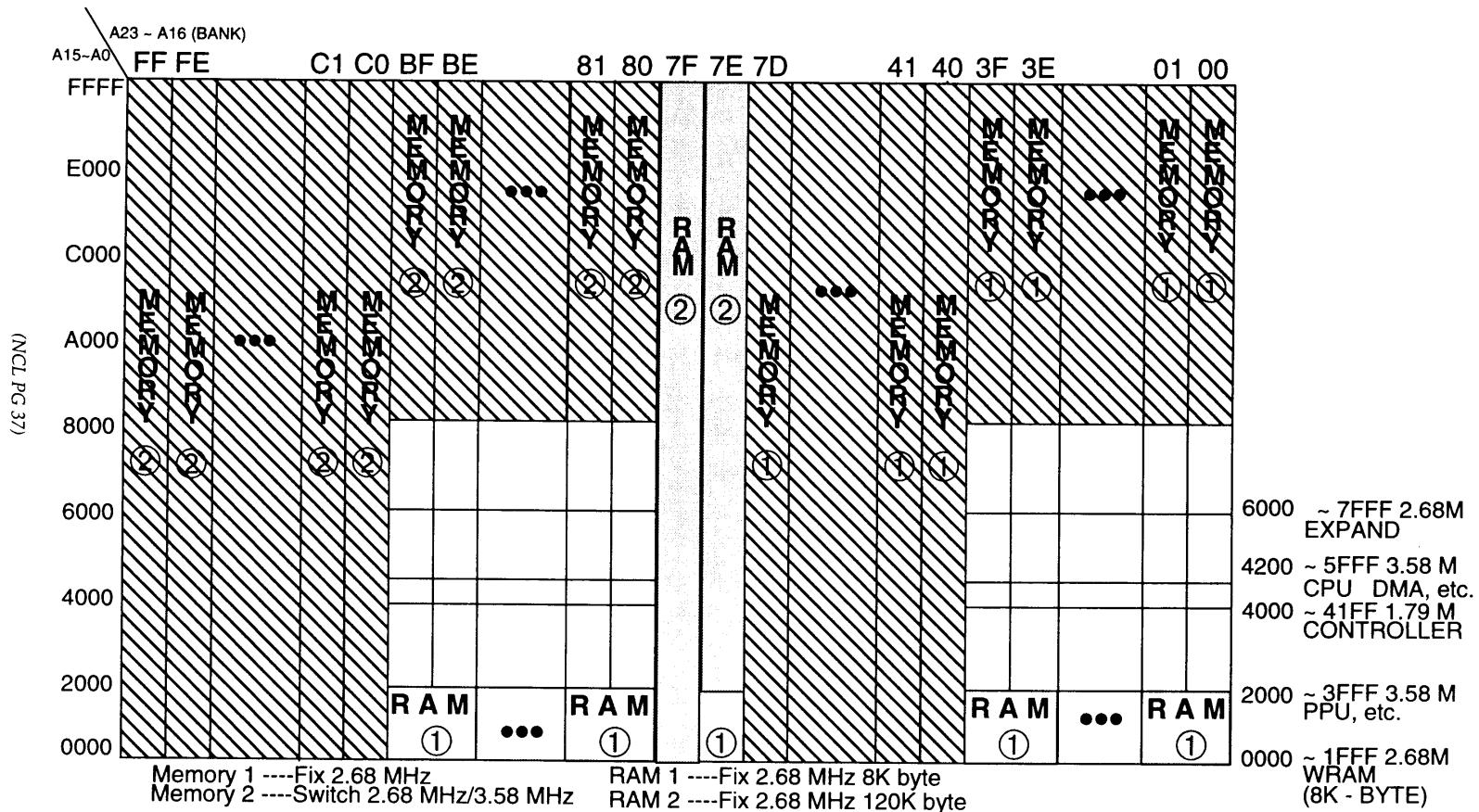
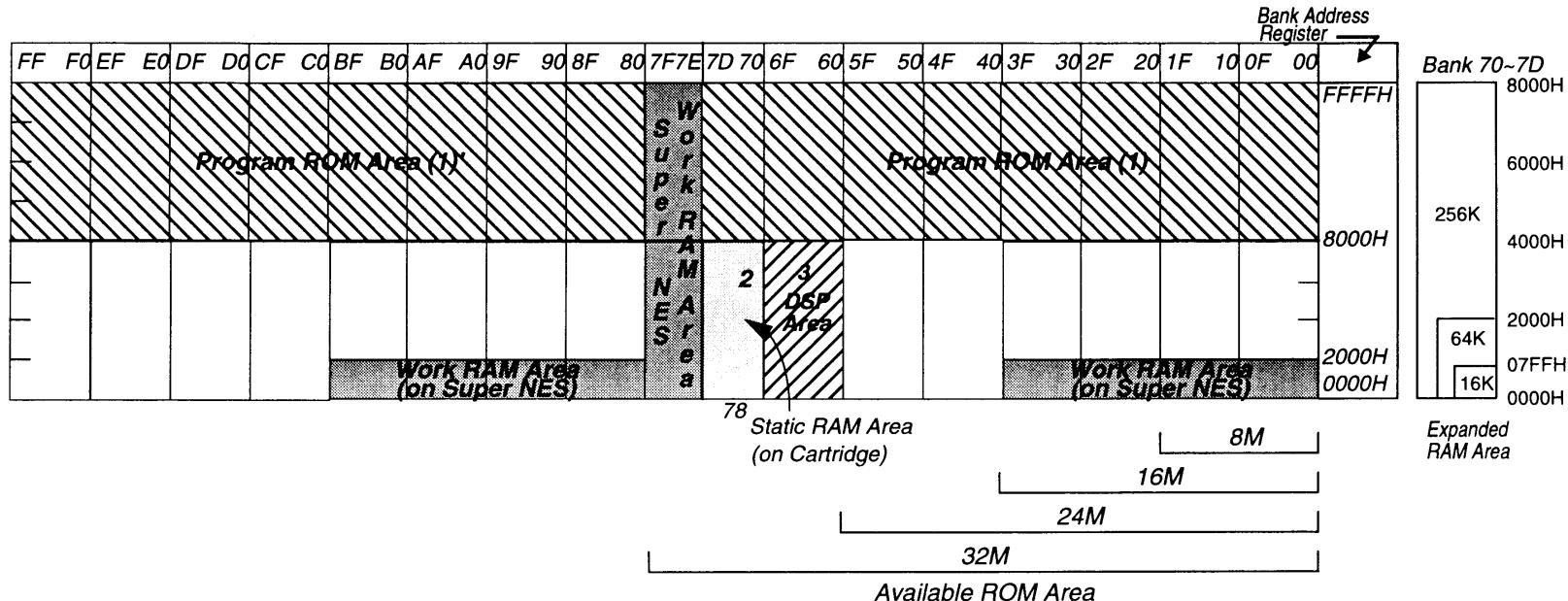
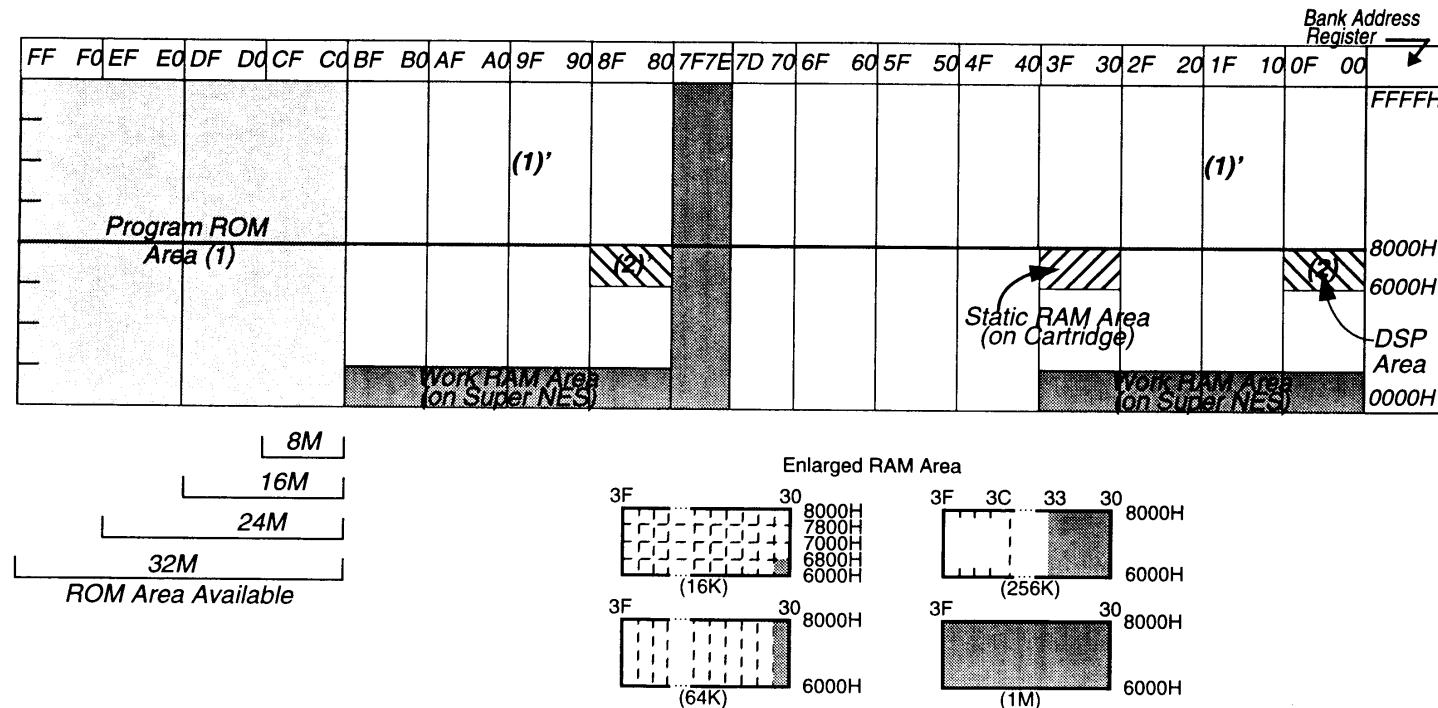


Figure 2-21-2 Super NES Memory Map (Mode 20)



2-21-3

Figure 2-21-3 Super NES Memory Map (Mode 21)

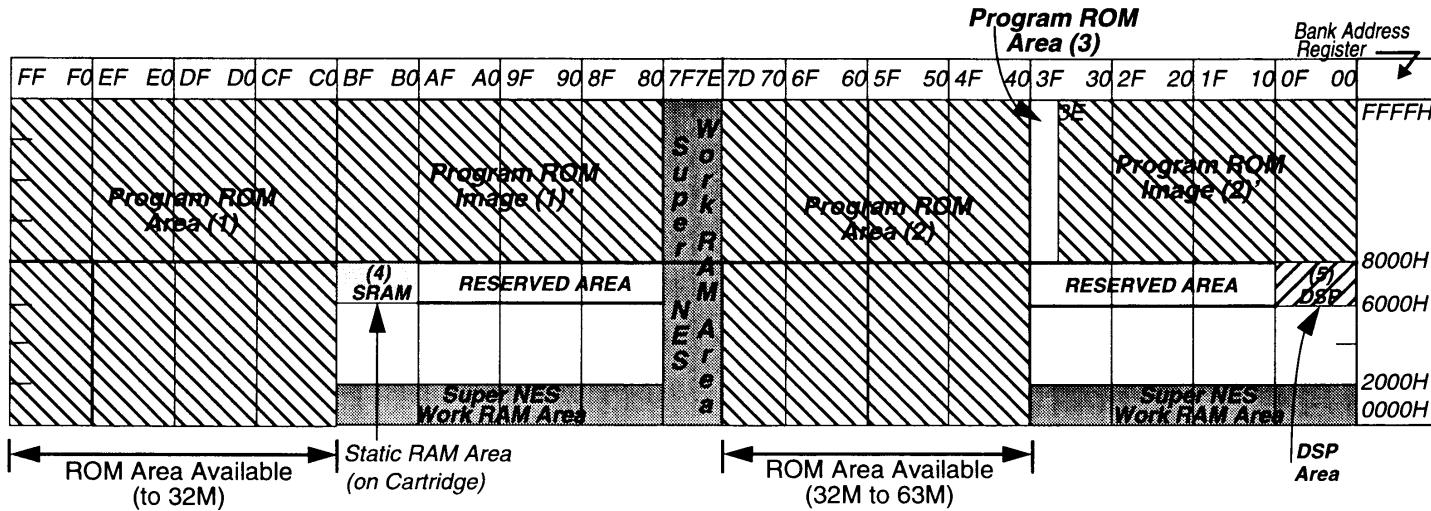


Note 1: In memory Mode 21, memory can be added from bank C0H to bank FFH (maximum is 32M bit). The ROM image of "8000H - FFFFH of bank C0H - FFH" will appear on "bank 00H - 3FH". Set vectors (i.e., Reset Vector) in the vector area of bank C0H".

Note 2: The ROM image from address 8000H ~ FFFFH of bank C0H ~ FFH is generated in bank 00H ~ 3FH and bank 80H ~ BFH.

Note 3: Programs located in the area of bank 80H ~ FFH can be executed in the high speed mode. Specify the need for the high speed mode in the submission form.

Figure 2-21-4 Super NES Memory Map (Mode 25, ROM Size Greater than 32 Mbits only)

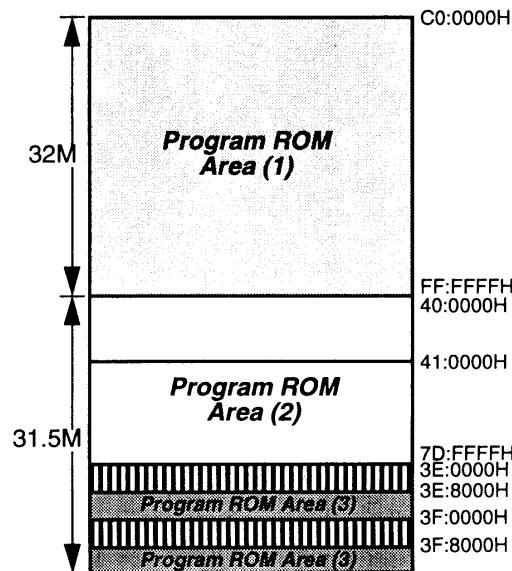


Note 1: The ROM image of 8000H~FFFFH of bank 40H~7DH will appear at bank 00H~3DH.  
Set vectors and registration data in FFB0H of bank 40H (ROM address 40:FFB0H).

Note 2: Programs located in the area of bank 80H~FFH can be executed in the high speed mode (3.58 MHz).

Note 3: Don't access null area.

Note 4: Use the area of bank 3E and 3F as the program ROM area of bank 7E and 7F.



## ***Chapter 22. Super NES Functional Operation***

This chapter provides the user with a basic understanding of the functional purpose for each of the major components of the Super NES control deck. Refer to the Super NES Functional Block Diagram (opposite page) while reading the following paragraphs.

### **22.1 SUPER NES CPU**

This is the Central Processing Unit for the Super NES. It coordinates all functions of the Super NES control deck and peripheral devices which are attached to the Super NES.

### **22.2 SUPER NES PPU1 AND PPU2**

These 2 units work together as the Picture Processing Unit for the Super NES. Pictures are generated for display based upon control inputs from the Super NES CPU. In general, PPU1 is used to generate background character data, rotation, and scaling; while PPU2 performs special effects like windows, mosaic, and fades.

### **22.3 SUPER NES WRAM**

The work RAM (WRAM) is a custom 128K x 8 bit RAM used by the Super NES CPU for data storage. Direct Memory Addressing (DMA) can be used by the Super NES CPU for rapid bulk transfer of data.

### **22.4 VRAM**

The VRAM is composed of 2 - 32K x 8 bit S-RAMs. This unit is used by PPU1 to store background character data until needed for display.

### **22.5 AUDIO PROCESSING UNIT (APU)**

The Audio Processing Unit performs all sound functions for the Super NES and is composed of the following units.

#### **22.5.1 SOUND CPU**

The Sound CPU is the central processing unit for the Super NES Audio Processing Unit. It controls sound functions much in the same way that the Super NES CPU controls functions of the Super NES.

#### **22.5.2 SOUND DSP**

The DSP has 8 channels of pulse code modulated (PCM) sound, a noise generator, echo, sweep, envelope, and other circuits to reproduce tone qualities from RAM data.

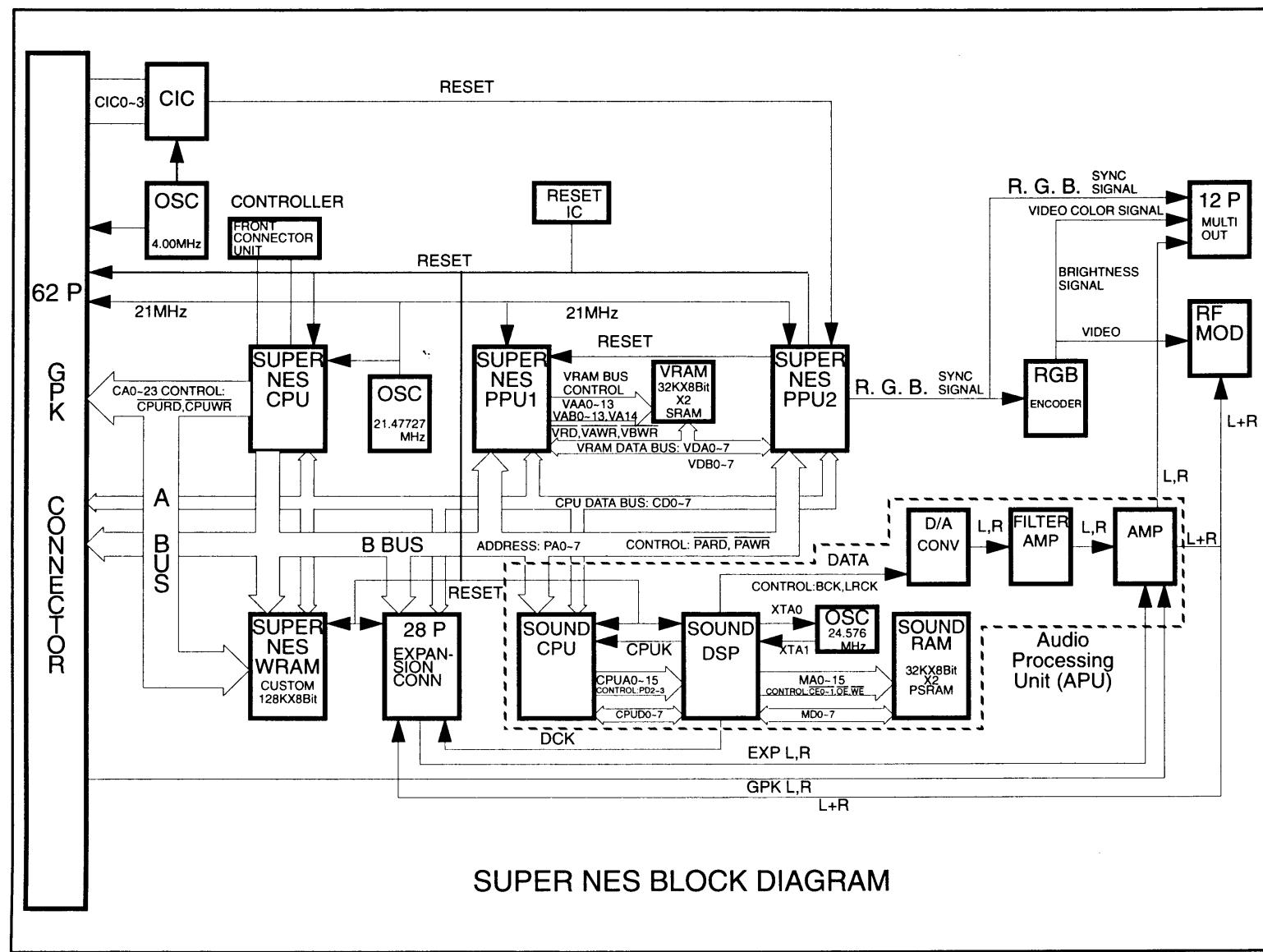


Figure 2-22-1 Super NES Functional Block Diagram

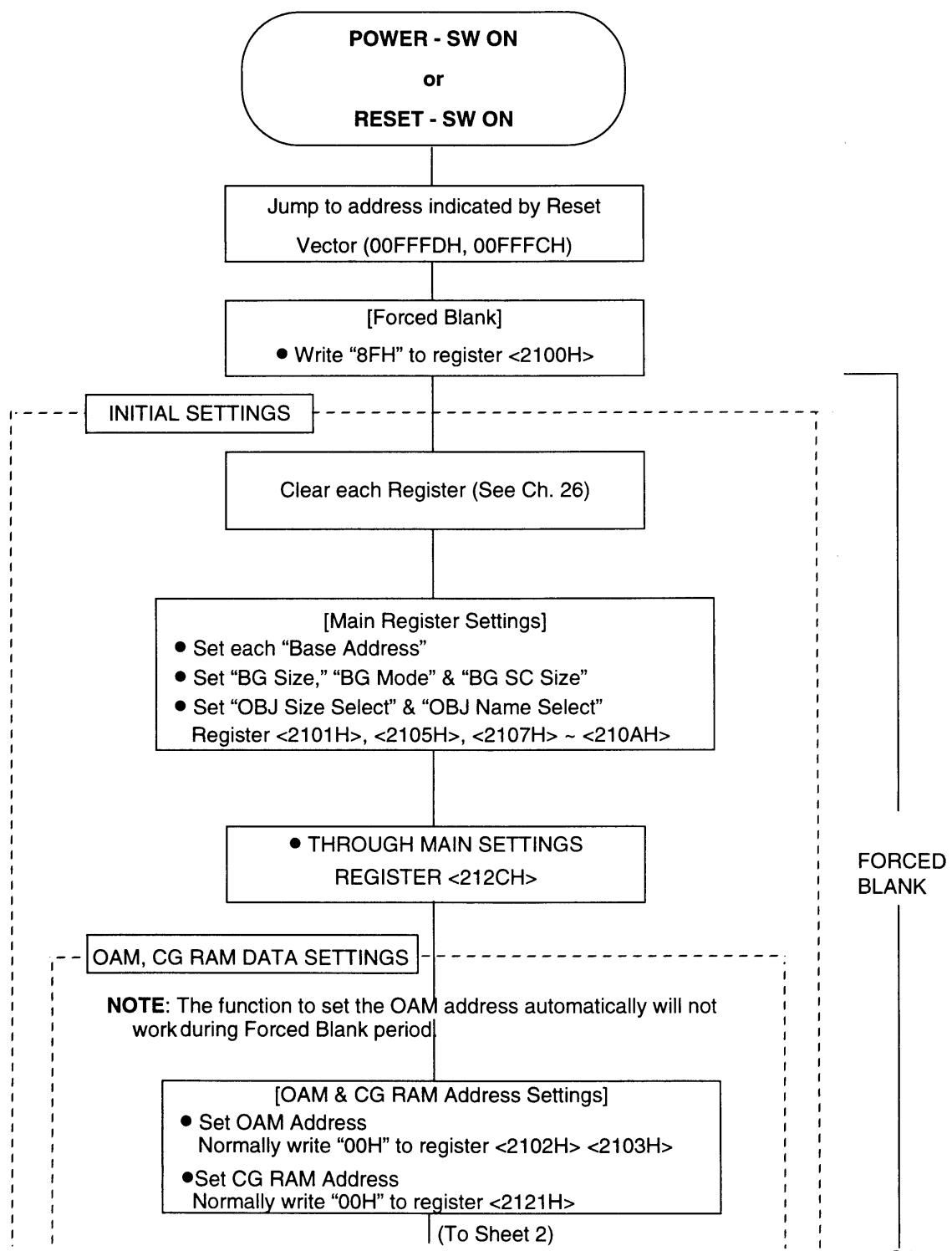
### **22.5.3 SOUND RAM**

The Sound RAM is composed of 2-32Kx8 bit SRAMs. Program and tone data are loaded from the game pak to the sound RAM by the Sound CPU. The RAM is time shared by the Sound CPU and DSP.

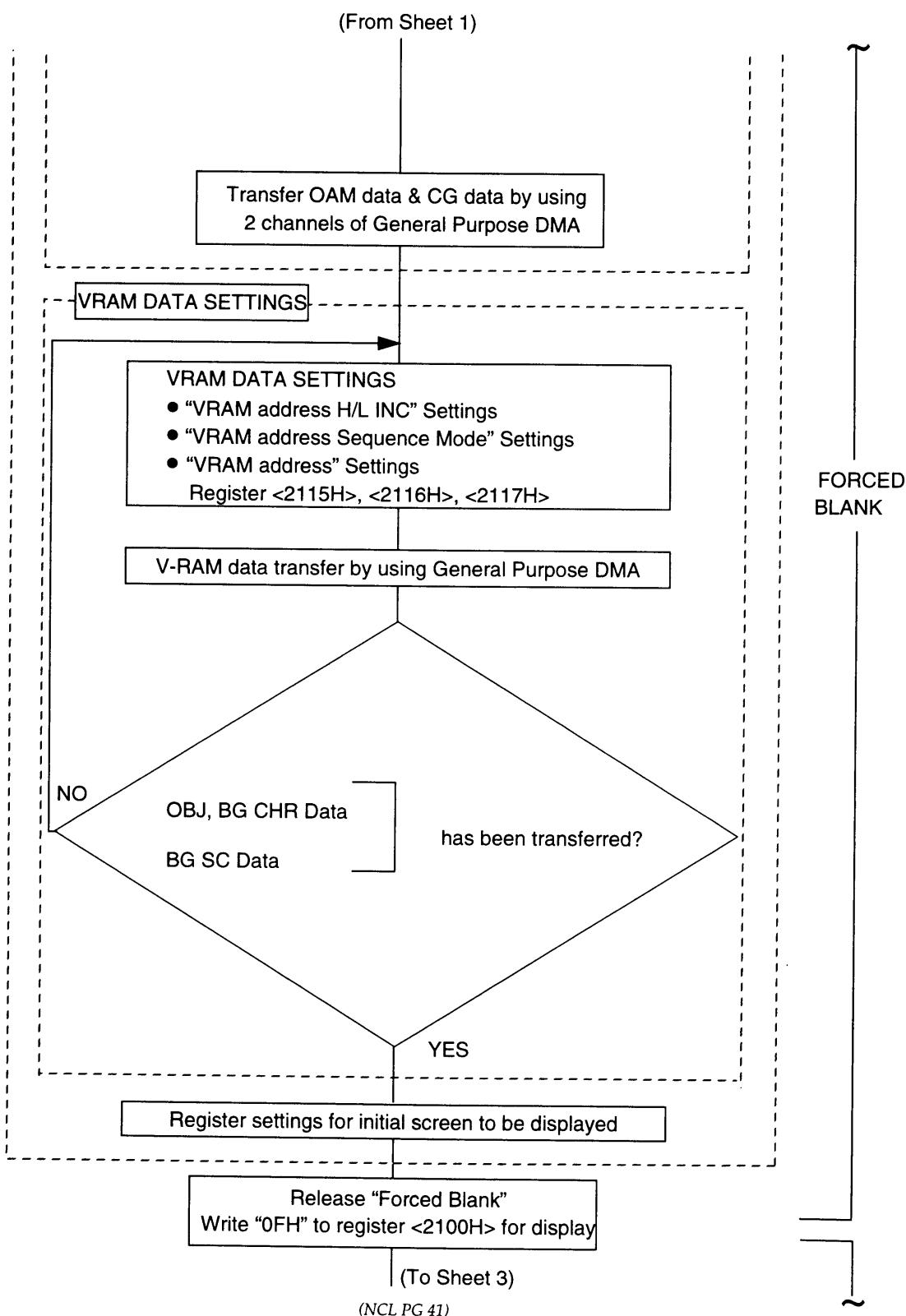
### **22.5.4 D/A CONVERTER**

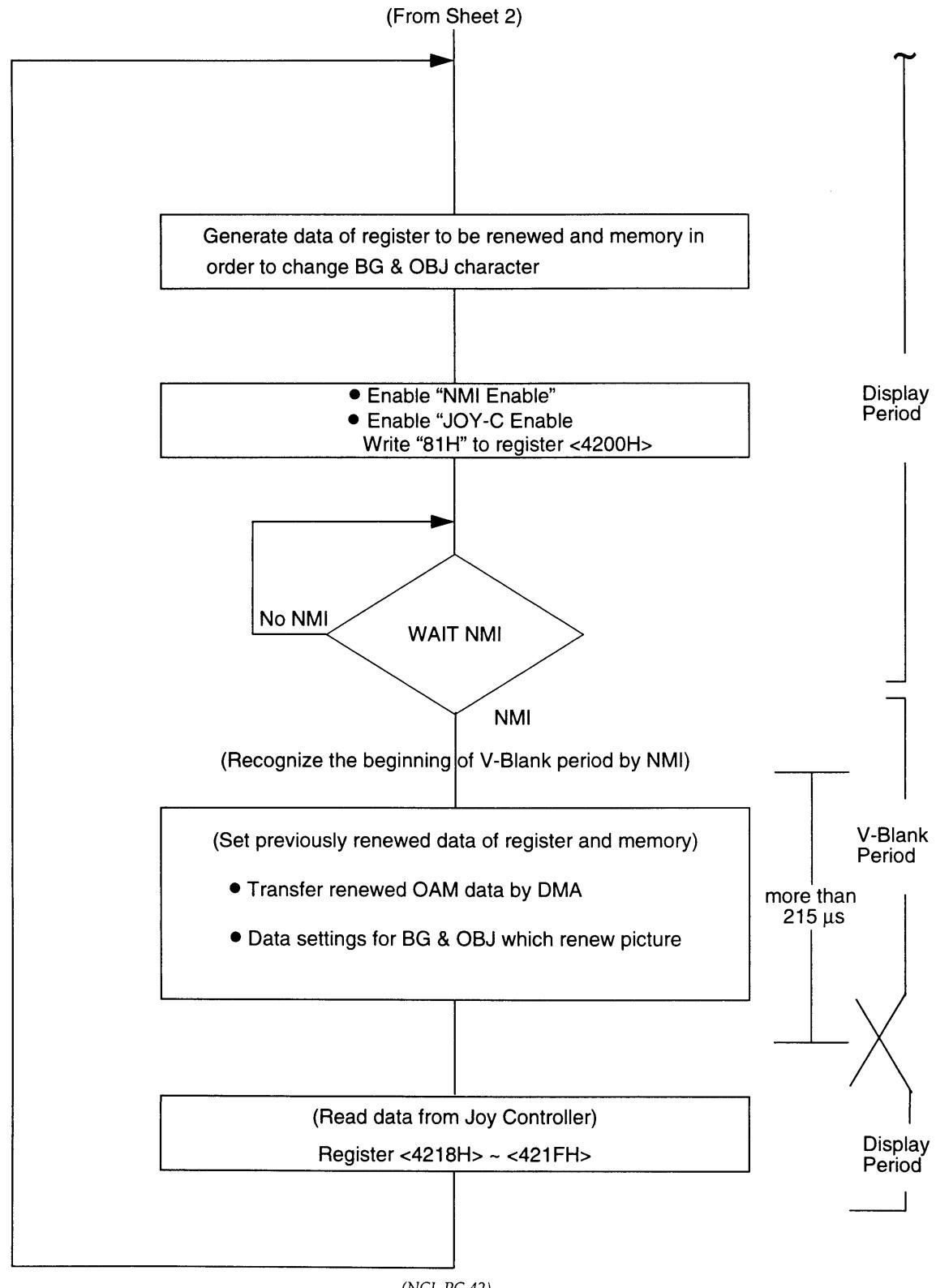
Converts the digitized sound to an analog signal which is filtered and amplified to produce the L+R (mono) output through the RF Modulator and L,R (stereo) outputs through the multi-out connector.

## Chapter 23. System Flowchart



(NCL PG 40)





(NCL PG 42)

## **Chapter 24. Programming Cautions**

### **24.1 CAUTION #1**

Registers <210DH> ~ <2114H> and <211BH> ~ <2120H> must be accessed in the order of Low and High twice (Read Twice or Write Twice). If it is not known whether the next access should be low or high, initialize as follows:

OAM, CGRAM, VRAM Other Registers (Write)	Set the address again. The lower data should be written more than one time, and the higher data should be written.
H/V Counter Read	The H/V counter will be initialized when the status register <213FH> is read. The data should be read in the order of Low and High.

### **24.2 CAUTION #2**

The period which can be accessed for the register is as follows:

V-RAM, OAM CG-RAM	Forced Blank or V-Blank period only. Forced Blank, V-Blank or H-Blank period only.
Other Register (Write)	All period (however, when writing the data, the picture may not be displayed properly).
Other Register (Read)	All period (However, the data which may be changed during display period may not be read properly).

### **24.3 CAUTION #3**

The address space for the V-RAM is 64K-word (1 word = 16-bit) maximum. 32K-word memory is installed in the Super NES unit.

### **24.4 CAUTION #4**

When the V-RAM is accessed from the CPU, the address counter will be increased automatically. For the V-RAM increment mode, please use the register mode designated by the instruction.

**24.5 CAUTION #5**

When the V-RAM is read continuously, the first address will not be incremented once the V-RAM data has been stored. The first address should be read as dummy data on subsequent passes.

**24.6 CAUTION #6**

The top color data of each CG color data palette is transparent. Because transparent is a color which is not displayed, any color can be set. The color data of CG address (00H) is normally black (background).

**24.7 CAUTION #7**

Even though 9-bits are provided as the OAM H-position, the value (100H) must not be used.

**24.8 CAUTION #8**

Before processing the controller keys, verify which devices are currently connected to the controller ports. The valid identification codes are:

- Standard Controller      0000B
- Super NES Mouse      0001B
- Super Scope      1111B

These codes may be found in bits D3 ~ D0 of registers <4218H> and <421AH>. If the standard controller is used for the game, inputs should be ignored whenever the ID code is not 0000B.

**24.9 CAUTION #9**

The initial value of the work RAM in the main computer is not set when power is applied to the computer. Programming should be done in such a way that no errors occur when the data is indeterministic. The initial value is different depending upon the computer used. Initialize the entire RAM area when, for example, it has been programmed under the misconception that the data is a fixed value, 00•FF.

**24.10 CAUTION #10**

When using the battery back-up SRAM, avoid program errors due to data loss. The CPU may crash if the user hits the control deck when the game pak is in use, if the game pak is not inserted properly, or if the game pak connector is dirty. Data loss may be unavoidable in some cases. Before reusing SRAM data, determine if the data is recoverable. One method of detection is to save the data in several areas of the SRAM and calculate the check sums of each area. Before utilizing any data in the SRAM, the program must compare each of the check sums. If the check sums are not equal, the data is corrupted.

## 24.11 CAUTION #11

In addition to using a check code to check a hot/cold start, determine if the content of the work RAM used is correct after the reset. Data in work RAM is lost gradually after the power is turned off. The speed at which data is lost differs according to the area. If the device is turned on immediately after it has been turned off, the area that is checked for hot/cold start code may contain the original data. This does not mean that the entire data have been recovered. Guidelines for prevention of data loss are the same as those for the previous caution.

## 24.12 CAUTION #12

When executing critical commands using the controller keys, such as; modify, erase data, or software reset, use all 16 bits of data including the input device's signature. Corrupt data may be sent by the controller if the controller is unplugged during a game. When the computer is reset using start, select, L, and R controller data simultaneously, verify that:

- The start, select, L, and R are pressed,
- No other keys are pressed, and
- The signature data is 0000.

In other words, check that the key data is 3030H.

## 24.13 CAUTION #13

Do not place critical game characters within two characters of the perimeter of the display screen area. This area of the television varies from one brand or model to the next. The Super NES may not be able to display characters in some areas if programmed too close to the edge of the screen. Critical game characters include score data and various parameters.

## 24.14 CAUTION #14

Ensure that the program clears the emulation bit on reset or start-up before executing 65816 instructions (i.e., JMP \$808007). This is demonstrated in the programming example, below:

Example:

RESET	;Reset vector
SEI	;Disable interrupt
CLC	;Clear carry
XCE	;Exchange carry with E bit, now in 65816 mode
JMP \$808009	;Example 65816 instruction

## 24.15 CAUTION #15

When utilizing the high speed mode (3.58MHz), perform a dummy jump at the start of every vector to change the Program Bank Register to the upper banks (\$80 or above). Refer to the following program example.

Example:

```

        ORG    $808000
RESET
        SEI
        CLC
        XCE
        JMP    ~RESETFAST ;Dummy jump to change PBR
RESETFAST           ;RESETFAST belongs to bank $80
...
...
NMI
        JMP    ~NMIFAST
NMIFAST            ...
...

```

## 24.16 CAUTION #16

When restarting controller read after it has been temporarily disabled, the user program should confirm that the buttons have been released before accepting the button inputs.

Some licensed controllers latch the last data which was received after disabling controller read. This data is held for about 3 fields (50 msec) into the next controller read sequence. This performance as compared to Nintendo's standard controller performance is demonstrated in the table below.

User Operation	No Operation				"B" Button		No Operation		"A" Button			
Nintendo Controller Output					B	B	N/A				A	A
Output of Some Licensed Controllers					B	B	N/A				B	B
Controller Read	Enslle				Disable				Enable			

Note: 1 field (16.6 msec)

For instance, if the software is programmed as follows;

1. Enter the room when “B” button is pressed.
2. Disable controller read while changing screen data.
3. The room appears and enable controller read.
4. Exit the room when “B” button is pressed.

the player will immediately exit the room.

This problem can be resolved in 2 different ways, as described below.

#### 24.16.1 EDGE DETECTION

If “edge detection” is used for processing controller data instead of “level detection”, the above problem can be avoided. The following sample program illustrates edge detection. The difference between controller (Cont) and trigger (Trig) data in the sample program is shown in the table below.

Cont	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	0	0
Trig	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Note: 0 = Off  
1 = On

## (SAMPLE PROGRAM)

```

;-----;
;- RAM Definition
;-----;
Cont1L      ds   1           ; Controller #1 data low byte
Cont1H      ds   1           ; Controller #1 data high byte
Cont2L      ds   1           ; Controller #2 data low byte
Cont2H      ds   1           ; Controller #2 data high byte
Trig1L     ds   1           ; Trigger data of controller #1
Trig1H     ds   1           ;
Trig2L     ds   1           ;Trigger data of controller #2
Trig2H     ds   1           ;
;-----;
;- Read Controller
;-----;
RdCont;
    push
    a8          ; Accumulator 8-bit
RdCont_Wait1
    LDA  HVBJoy      ; <4212>
    AND  #%00000001  ; Wait JOY-C Enable : D0=0
    BEQ  RdCont_Wait1
RdCont_Wait2
    LDA  HVBJoy
    AND  #%00000001
    BNE  RdCont_Wait2
    a16
    i16          ; Accumulator 16-bit
                  ; Index 16-bit
RdCont_Cont1
    LDY  Cont1L      ; Keep last data in "IY"
    LDA  Joy1L       ; <4218> (Cont1-L)
    STA  Cont1L      ; Store new controller data
    TYA
    EOR  Cont1L
    AND  Cont1L
    STA  Trig1L      ; Store trigger data
RdCont_Cont2
    LDY  Cont2L      ; Keep last data in "IY"
    LDA  Joy2L       ; <421AH> (Cont2-L)
    STA  Cont2L      ; Store new controller data
    TYA
    EOR  Cont2L
    AND  Cont2L
    STA  Trig2L      ; Store trigger data
    pop
    RTS

```

### 24.16.2 ALTERNATE METHOD

The problem may be avoided by ignoring controller data for about 3 fields, after restarting controller read. Since programming becomes very complicated, increasing the risk of program bugs, this method is not recommended.

If controller read is disabled for 1~2 fields, the consumer cannot press a button quickly enough to cause a problem. This configuration is illustrated in the table below.

User Operation	No Operation				“B” Button				No Operation				“A” Button				
Nintendo Controller Output					B	B	N/A	B	B					A	A	A	A
Output of Some Licensed Controllers					B	B	N/A	B	B					A	A	A	A
Controller Read	Enslbe				Disable				Enable								
Note: 1 field (16.6 msec)																	

*PROGRAMMING CAUTIONS*

THIS PAGE INTENTIONALLY  
LEFT BLANK

## Chapter 25. Documented Problems

The following paragraphs describe system problems which have been identified and provides solutions for the problems listed.

### 25.1 PROBLEM 1

#### 25.1.1 SYMPTOM

If H-DMA starts at about the same time that General Purpose DMA finishes, sometimes the CPU will cease to operate properly or H-DMA will not be correctly implemented (S-CPU ver. 1).

This could happen if General Purpose DMA finishes during the first 2.24  $\mu$ s of the H - Blank period on lines 0 - 224 (239), while H-DMA is being used. It can also happen at the beginning of line 0, as well. \*

\* The real time trace function of the ICE can be utilized to confirm the timing.

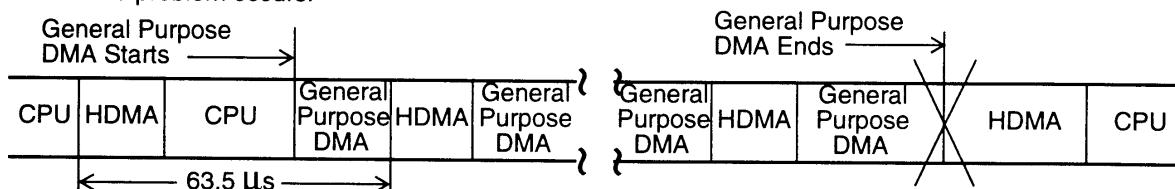
#### 25.1.2 SOLUTION

This problem will not happen if General Purpose DMA is used only during V - Blank or if H-DMA starts in the middle of data transfer of General Purpose DMA. It does not happen if H-DMA is not being used.

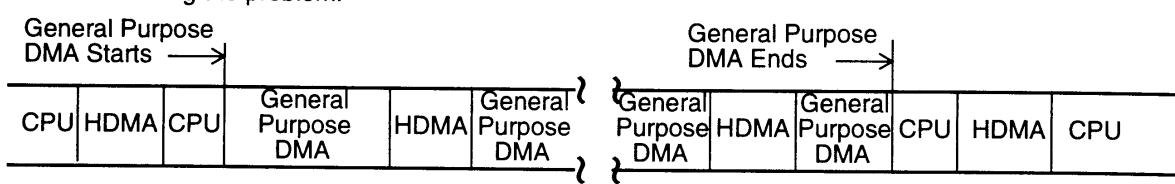
This problem can also be avoided by adjusting the time at which General Purpose DMA begins and/or decreasing the number of bytes transferred. The H and V count timers can be utilized to determine the start time of General Purpose DMA. One line takes 63.5  $\mu$ s and the value of one count on the H count timer is equivalent to 0.186  $\mu$ s.

The end timing of the General Purpose DMA changes depending on the amount of transferred data of H-DMA which happens in the middle of data transfer of the General Purpose DMA.

When the problem occurs:



When avoiding the problem:



## 25.2 PROBLEM 2

### 25.2.1 SYMPTOM

When the size of OBJ 0 is 16 x 16, 32 x 32, or 64 x 64, and its horizontal position is 0 through 255, and there are other objects present with negative horizontal positions (they are not displayed on the screen), the Time Over Flag will become 1 (S-PPU1 ver. 1).

### 25.2.2 SOLUTION

The cause is being examined.

## Chapter 26. Register Clear (Initial Settings)

(This is a recommended setting for beginners. It is not necessary to perform register clear exactly this way. However, the register status is not stable when power is turned on and initial settings must be performed).

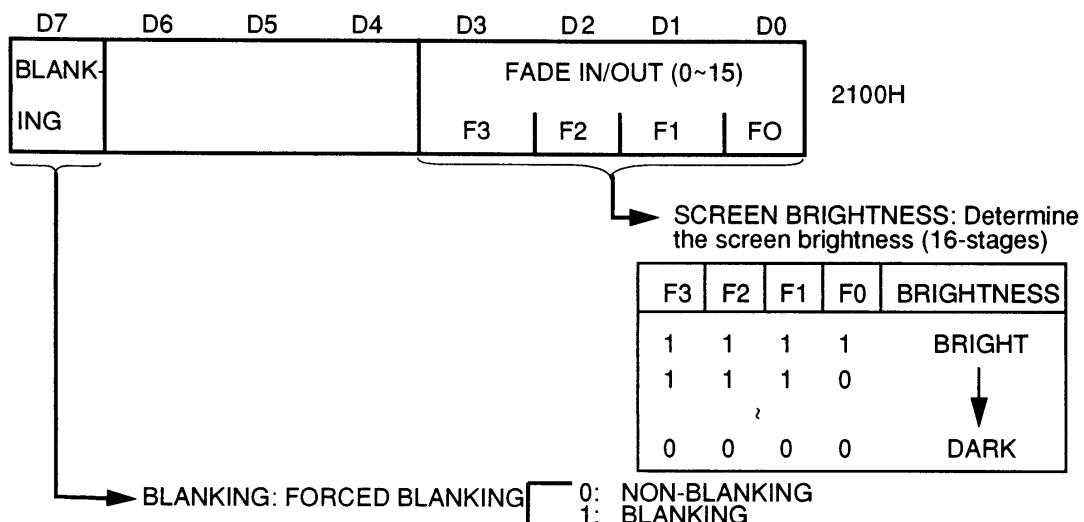
ADDRESS (HEX)	DATA (HEX)	ADDRESS (HEX)	DATA (HEX)
<2100>	8 F (Forced Blank)	<2120>	0 0 0 0
<2101>	0 0	<2121>	0 0
<2102>	0 0	<2122>	(CG Data)
<2103>	0 0	<2123>	0 0
<2104>	(OAM Data)	<2124>	0 0
<2105>	0 0	<2125>	0 0
<2106>	0 0	<2126>	0 0
<2107>	0 0	<2127>	0 0
<2108>	0 0	<2128>	0 0
<2109>	0 0	<2129>	0 0
<210A>	0 0	<212A>	0 0
<210B>	0 0	<212B>	0 0
<210C>	0 0 (Low) (High)	<212C>	0 0
<210D>	0 0 0 0	<212D>	0 0
<210E>	0 0 0 0	<212E>	0 0
<210F>	0 0 0 0	<2130>	3 0
<2110>	0 0 0 0	<2131>	0 0
<2111>	0 0 0 0	<2132>	E 0
<2112>	0 0 0 0	<2133>	0 0
<2113>	0 0 0 0	<4200>	0 0
<2114>	0 0 0 0	<4201>	F F
<2115>	8 0	<4202>	0 0
<2116>	0 0	<4203>	0 0
<2117>	0 0	<4204>	0 0
<2118>	(VRAM Data)	<4205>	0 0
<2119>	(VRAM Data)	<4206>	0 0
<211A>	0 0	<4207>	0 0
<211B>	0 0 0 1	<4208>	0 0
<211C>	0 0 0 0	<4209>	0 0
<211D>	0 0 0 0	<420A>	0 0
<211E>	0 0 0 1	<420B>	0 0
<211F>	0 0 0 0	<420C>	0 0
		<420D>	0 0

## Chapter 27. PPU Registers

ADDRESS: 2100H

NAME: INIDISP

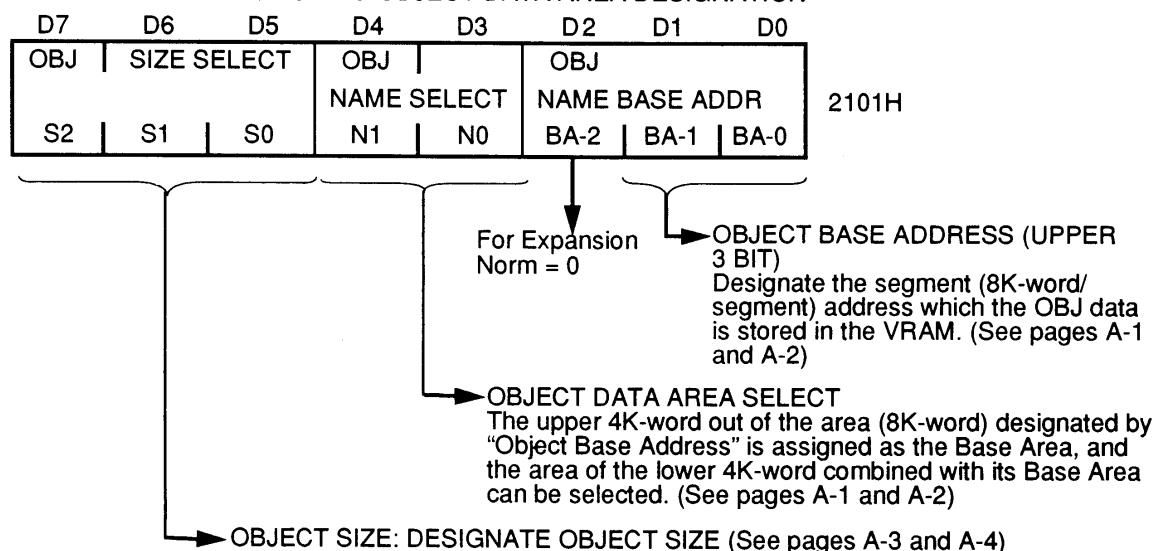
CONTENTS: INITIAL SETTINGS FOR SCREEN



ADDRESS: 2101H

NAME: OBJSEL

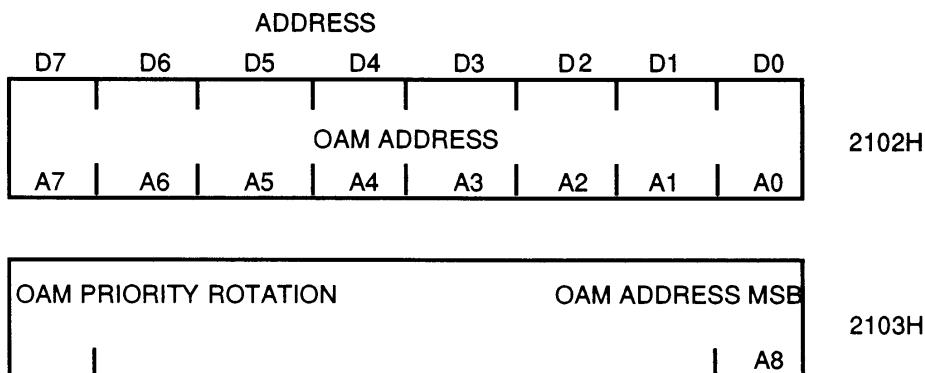
CONTENTS: OBJECT SIZE &amp; OBJECT DATA AREA DESIGNATION



S2	S1	S0	OBJ SIZE	
			0 (SM)	1 (LG)
0	0	0	8 DOT	16 DOT
0	0	1	8 DOT	32 DOT
0	1	0	8 DOT	64 DOT
0	1	1	16 DOT	32 DOT
1	0	0	16 DOT	64 DOT
1	0	1	32 DOT	64 DOT

(NCL PG 46)

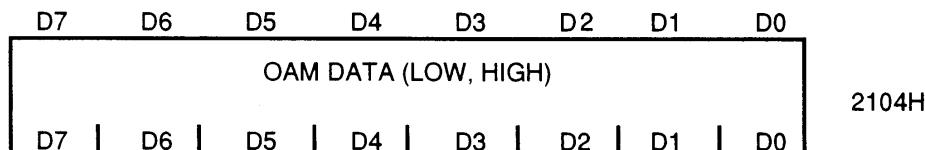
ADDRESS: 2102H /2103H  
 NAME: OAMADDL / OAMADDH  
 CONTENTS: ADDRESS FOR ACCESSING OAM (OBJECT ATTRIBUTE MEMORY)



- This is the INITIAL ADDRESS to be set in advance when reading from or writing to the OAM.
- To set the OBJ priority order, write "1" to D7 (OAM Priority Rotation) of register <2103H> and set the highest priority OBJ number (0 ~ 127) to D1 ~ D7 of register <2102H> (refer to "Priority Order Shifting").
- The address which has been set just before every field (beginning with V-BLANK) will be set again to registers <2102H> <2103H> automatically. However, the address cannot be set automatically during Forced Blank period.

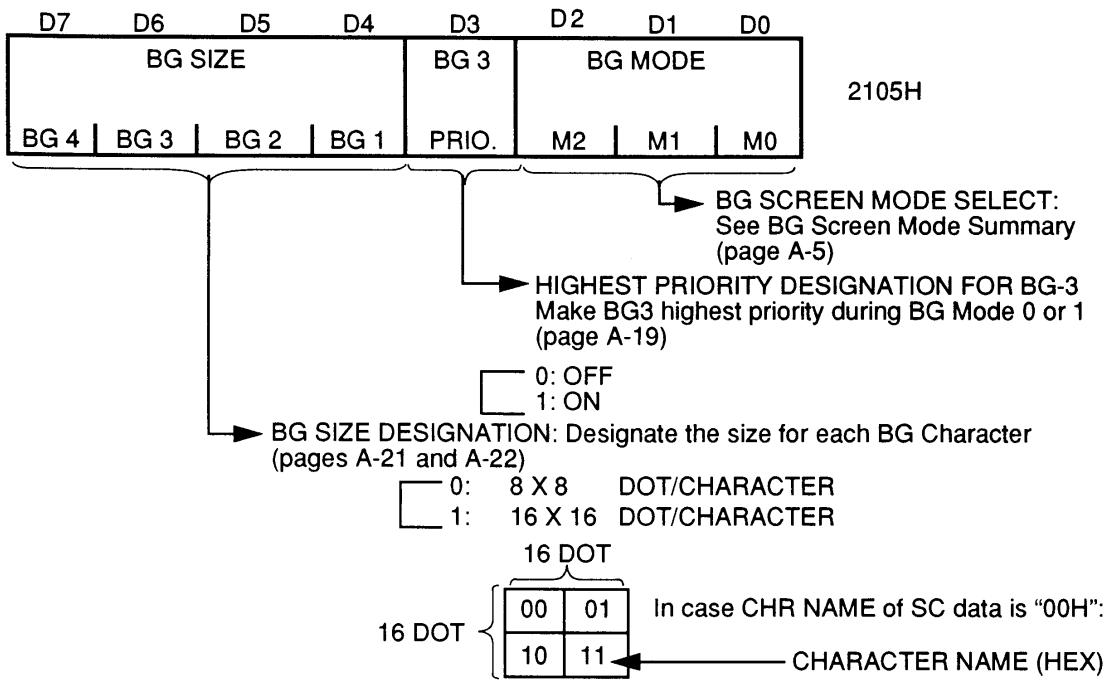
---

ADDRESS: 2104H  
 NAME: OAM DATA  
 CONTENTS: DATA FOR OAM WRITE

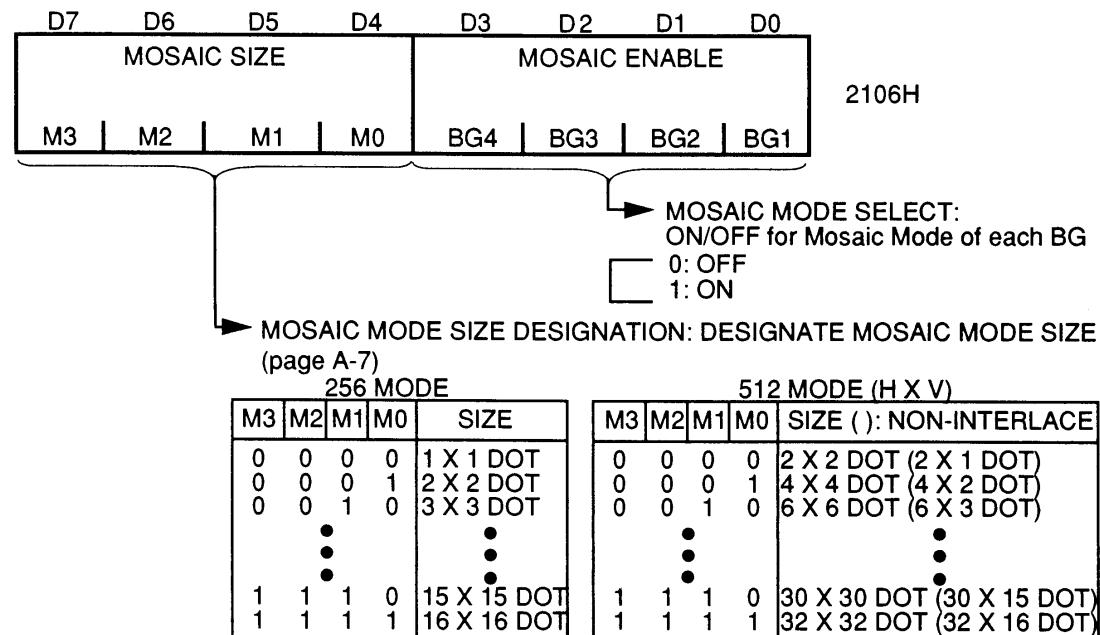


- This is the OAM data to be written to any address of the OAM (refer to page A-3).
- After register <2102H> or <2103H> is accessed, the data must be written in the order of Lower 8-bit and Upper 8-bit of register <2104H>. The OAM address will be increased automatically when the OAM data is written in the order of LOW to HIGH.
- The data can be written only during a V-BLANK or FORCED BLANK period.

ADDRESS: 2105H  
 NAME: BG MODE  
 CONTENTS: BG MODE & CHARACTER SIZE SETTINGS

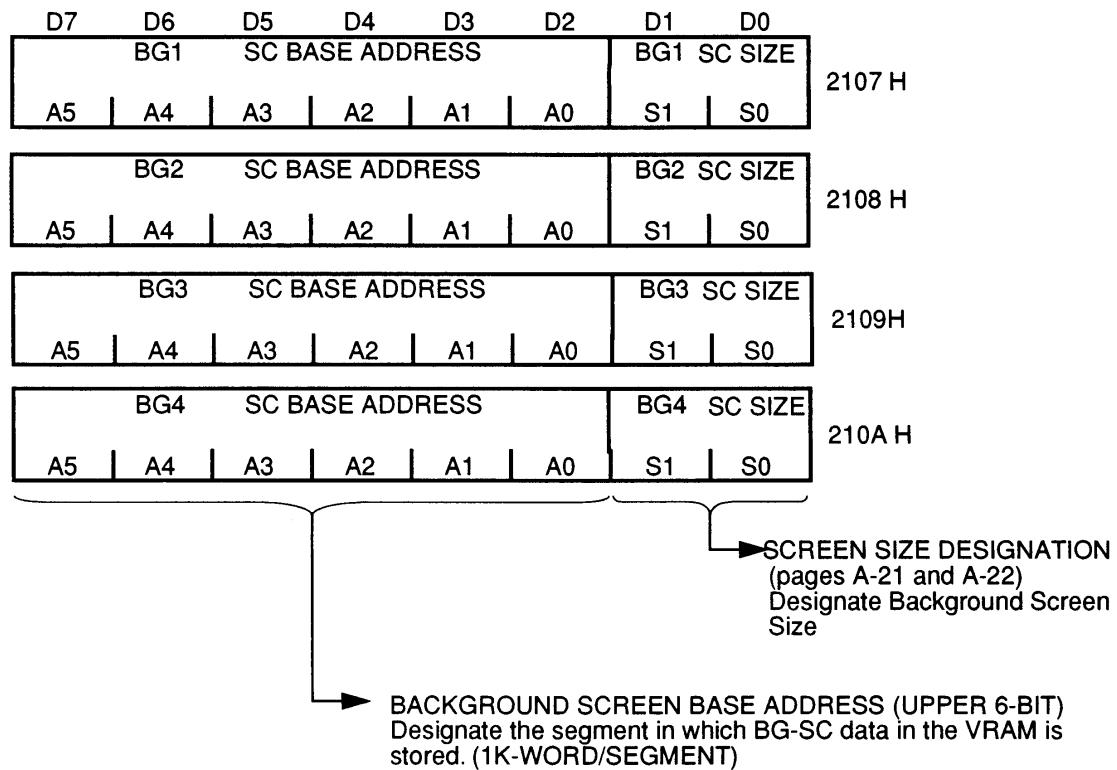


ADDRESS: 2106H  
 NAME: MOSAIC  
 CONTENTS: SIZE & SCREEN DESIGNATION FOR MOSAIC DISPLAY



(NCL PG 48)

ADDRESS: 2107H / 2108H / 2109H / 210AH  
 NAME: BG1SC / BG2SC / BG3SC / BG4SC  
 CONTENTS: ADDRESS FOR STORING SC-DATA OF EACH BG & SC SIZE DESIGNATION  
 (MODE 0 ~ 6)

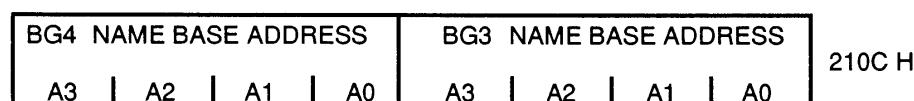
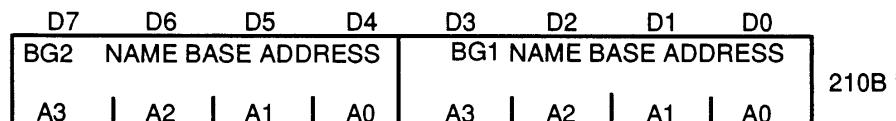


SCREEN SIZE &amp; SCREEN REPETITION

SCREEN SIZE		SCREEN SIZE	
0	0		
0	1		
1	0		
1	1		

(NCL PG 49)

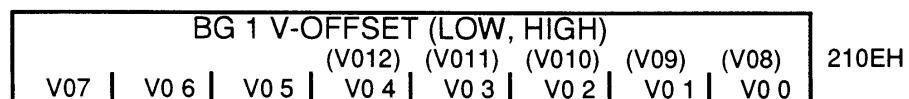
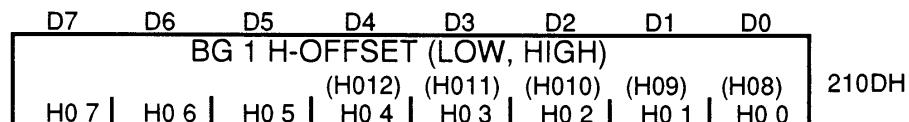
ADDRESS: 210BH / 210CH  
 NAME: BG12NBA / BG34NBA  
 CONTENTS: BG CHARACTER DATA AREA DESIGNATION



→ BACKGROUND NAME BASE ADDRESS (UPPER 4-BIT):  
 Designate the segment address in the VRAM in which BG character data  
 is stored. (4K-WORD/SEGMENT)

---

ADDRESS: 210DH / 210EH  
 NAME: BG1H0FS / BG1V0FS  
 CONTENTS: H/V SCROLL VALUE DESIGNATION FOR BG-1



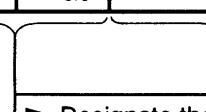
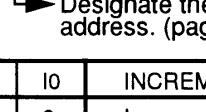
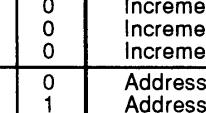
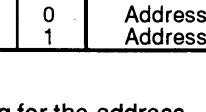
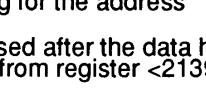
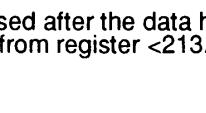
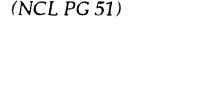
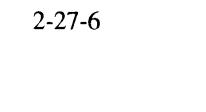
- 10-Bit maximum (0 ~ 1023) can be designated for H/V scroll value. (The size of 13-Bit maximum {-4096 ~ 4095} can be designated in MODE -7). (pages A-10 and A-11)
- By writing to the register twice, the data can be set in the order of Low and High.

ADDRESS: 210FH / 2110H / 2111H / 2112H / 2113H / 2114H  
 NAME: BG2H0FS / BG2V0FS / BG3H0FS / BG3V0FS / BG4H0FS / BG4V0FS  
 CONTENTS: H/V SCROLL VALUE DESIGNATION FOR BG-2, 3, 4

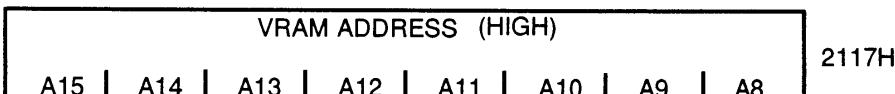
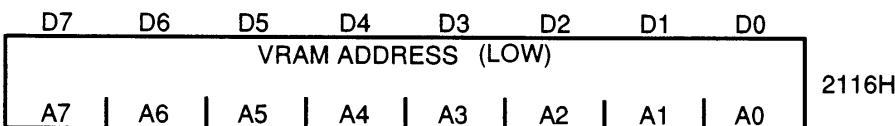
D7	D6	D5	D4	D3	D2	D1	D0	
BG H-OFFSET (LOW, HIGH)								
H0 7	H0 6	H0 5	H0 4	H0 3	H0 2	(H0 9)	(H0 8)	210FH 2111H 2113H
BG V-OFFSET (LOW, HIGH)								
V0 7	V0 6	V0 5	V0 4	V0 3	V0 2	(V0 9)	(V0 8)	2110H 2112H 2114H

- 10-Bit maximum (0 ~ 1023) of the H/V scroll value can be designated (page A-10)
- By writing to the register twice, the data can be set in the order of Low-and High.

ADDRESS: 2115H  
 NAME: VMAINC  
 CONTENTS: VRAM ADDRESS INCREMENT VALUE DESIGNATION

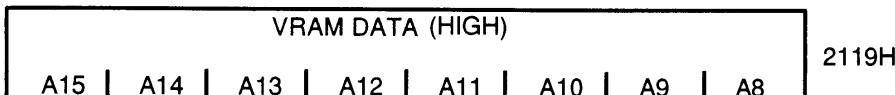
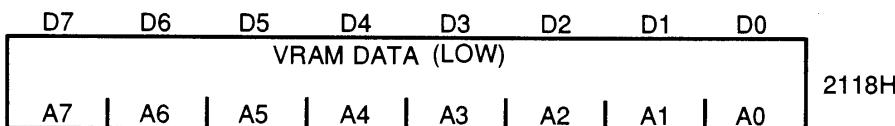
D7	D6	D5	D4	D3	D2	D1	D0	
H/L	INC			V-RAM ADDRESS FULL GRAPHIC	SEQUENCE MODE SC INCREMENT	I1	I0	2115H
								
								
								
								
								
								
								
								
								
								

ADDRESS: 2116H / 2117H  
 NAME: VMADDL / VMADDH  
 CONTENTS: ADDRESS FOR VRAM READ AND WRITE



- This is the initial address for reading from the VRAM or writing to the VRAM.
- The data is read or written by the address set initially, and every time the data is read or written, the address will be increased automatically.
- The value to be increased is determined by "SC INCREMENT" of register <2115H> and the setting value of the "FULL GRAPHIC."

ADDRESS: 2118H / 2119H  
 NAME: VMDATAL / VMDATAH  
 CONTENTS: DATA FOR VRAM WRITE

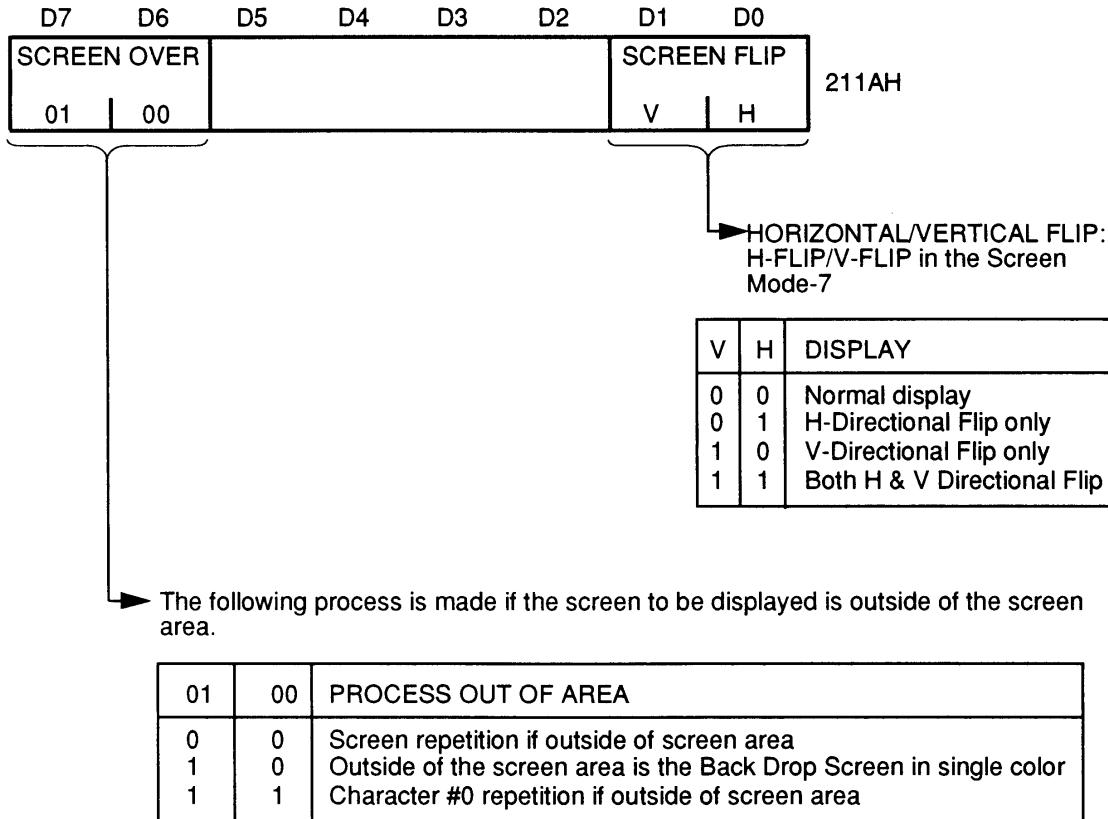


- This is the screen data and character data (BG & OBJ), which can be written to any address in the VRAM.
- According to the settings of register <2115H> "H/L INC," the data can be written to the VRAM as follows:

H/L INC	WRITE TO REGISTER	OPERATION
0	Write to <2118H> only	The data is written to lower 8-bit of the VRAM and the address will be increased automatically.
1	Write to <2119H> only	The data is written to upper 8-bit of the VRAM and the address will be increased automatically.
0	Write in the order of <2119H> and <2118H>	When the data is set in the order of upper and lower, the address will be increased.
1	Write in the order of <2118H> and <2119H>	When the data is set in the order of lower and upper, the address will be increased

NOTE: The data can be written only during V-BLANK or FORCED BLANK period.  
 (NCL PG 52)

ADDRESS: 211AH  
 NAME: M7SEL  
 CONTENTS: INITIAL SETTING IN SCREEN MODE-7



(NCL PG 53)

ADDRESS: 211B H / 211C H / 211D H / 211E H / 211F H / 2120 H  
 NAME: M7A / M7B / M7C / M7D / M7X / M7Y  
 CONTENTS: ROTATION/ENLARGEMENT/REDUCTION IN MODE-7, CENTER COORDINATE SETTINGS & MULTIPLICAND/MULTIPLIER SETTINGS OF COMPLEMENTARY MULTIPLICATION

D7	D6	D5	D4	D3	D2	D1	D0	
MATRIX PARAMETER A (LOW, HIGH)								
(MP15)	(MP14)	(MP13)	(MP12)	(MP11)	(MP10)	(MP9)	(MP8)	211 BH
MP 7	MP 6	MP 5	MP 4	MP 3	MP 2	MP 1	MP 0	
MATRIX PARAMETER B (LOW, HIGH)								
(MP15)	(MP14)	(MP13)	(MP12)	(MP11)	(MP10)	(MP9)	(MP8)	211 CH
MP 7	MP 6	MP 5	MP 4	MP 3	MP 2	MP 1	MP 0	
MATRIX PARAMETER C (LOW, HIGH)								
(MP15)	(MP14)	(MP13)	(MP12)	(MP11)	(MP10)	(MP9)	(MP8)	211 DH
MP 7	MP 6	MP 5	MP 4	MP 3	MP 2	MP 1	MP 0	
MATRIX PARAMETER D (LOW, HIGH)								
(MP15)	(MP14)	(MP13)	(MP12)	(MP11)	(MP10)	(MP9)	(MP8)	211 EH
MP 7	MP 6	MP 5	MP 4	MP 3	MP 2	MP 1	MP 0	

- The 8-bit data should be written twice in the order of lower and upper. Then, the parameter of rotation, enlargement and reduction should be set by its 16-bit data.
- The value down to a decimal point should be set to the lower 8-bit. The most significant bit of the upper 8-bit is for the signed bit. (MP15 is the signed bit. There is a decimal point between M7 & M8.)
- FORMULA FOR ROTATION/ENLARGEMENT/REDUCTION (Refer to Rotation/Enlargement/Reduction in Appendix A.).

$$\begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} X_1 - X_0 \\ Y_1 - Y_0 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}$$

$$A = \cos \gamma x (1 / \alpha), B = \sin \gamma x (1 / \alpha), C = -\sin \gamma x (1 / \beta), D = \cos \gamma x (1 / \beta)$$

$\gamma$ : Rotation angle  $\alpha$ : Reduction Rates for X (H)  $\beta$ : Reduction Rates for Y (v)

$X_0 \bullet Y_0$ : Center Coordinate,  $X_1 \bullet Y_1$ : Display Coordinate,

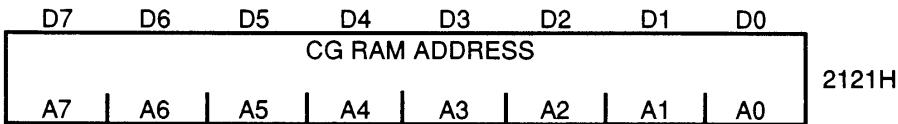
$X_2 \bullet Y_2$ : Coordinate Before Calculation

- Set the value of "A" to the register <211BH>. In the same way, set "B~D" to the register <211CH> ~ <211EH>.
- The complementary multiplication (16-bit x 8-bit) can be done by using registers <211BH> <211CH>. When setting 16-bit data to register <211BH> (must be written twice) and 8-bit data to register <211CH> (must be written only once), the multiplication result can be indicated rapidly by reading registers <2134H> ~ <2136H>.

D7	D6	D5	D4	D3	D2	D1	D 0	
CENTER POSITION X <sub>0</sub> (LOW, HIGH)								
X 7	X 6	X 5	X 4	X 3	X 2	X 1	X 0	211 FH
CENTER POSITION Y <sub>0</sub> (LOW, HIGH)								
Y 7	Y 6	Y 5	Y 4	Y 3	Y 2	Y 1	Y 0	212 0H

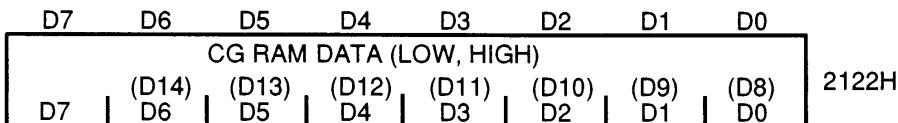
- The center coordinate (X<sub>0</sub> Y<sub>0</sub>) for Rotation/Enlargement/Reduction can be designated by this register.
- The coordinate value of X<sub>0</sub> & Y<sub>0</sub> can be designated by 13-bit (complement of 2).
- This register requires that the lower 8-bit set first and the upper 5-bit is set. Therefore, 13-bit data in total can be set.

ADDRESS: 2121H  
 NAME: CGADD  
 CONTENTS: ADDRESS FOR CG-RAM READ AND WRITE



- This is the initial address for reading from the CG-RAM or writing to the CG-RAM.
- The data is read by address set initially, and every time the data is read or written, the address will be increased automatically.

ADDRESS: 2122H  
 NAME: CGDATA  
 CONTENTS: DATA FOR CG-RAM WRITE

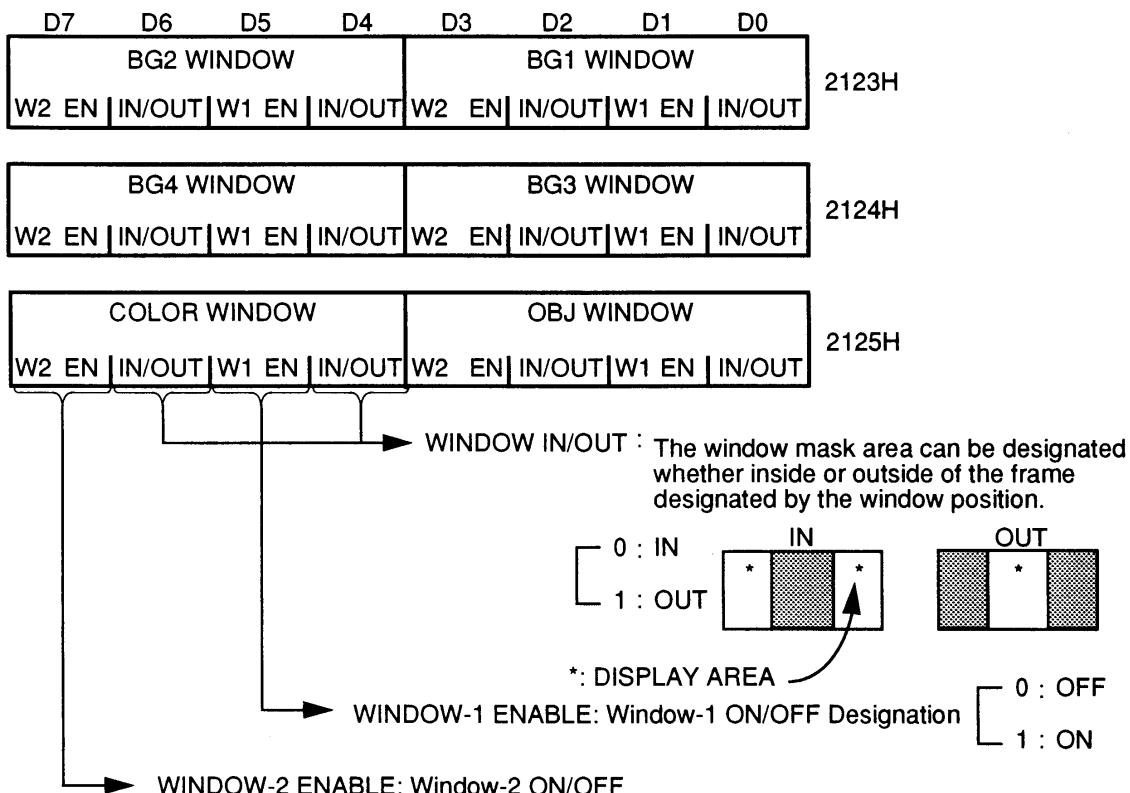


- This is the color generator data to be written at any address of the CG-RAM.
- The mapping of BG1 ~ BG 4 and OBJ data in the CG-RAM will be determined, which is performed by every mode selected by "BG MODE" of register <2105H>. (See page A-17)
- There are the color data of 8 palettes for each screen of BG1 ~ BG4. The palette selection is determined by 3-bit of the SC data "COLOR." (Refer to page A-10)
- Because the CG-RAM data is 15-bit/word, it is necessary to set lower 8-bit first to this register and then upper 7-bit should be set. When both lower and upper are set, the address will be increased by 1 automatically.

NOTE: After the address is set, the data should be written in the order of low, then high. This is similar to the OAM Data register.

NOTE: The data can be written only during H/V BLANK or FORCED BLANK period.

ADDRESS: 2123H / 2124H / 2125H  
 NAME: W12SEL/ W34SEL/ WOBJSEL  
 CONTENTS: WINDOW MASK SETTINGS (BG1~BG4, OBJ, COLOR)



The COLOR WINDOW is a window for main and sub screen. (It is related to the register <2130H>).

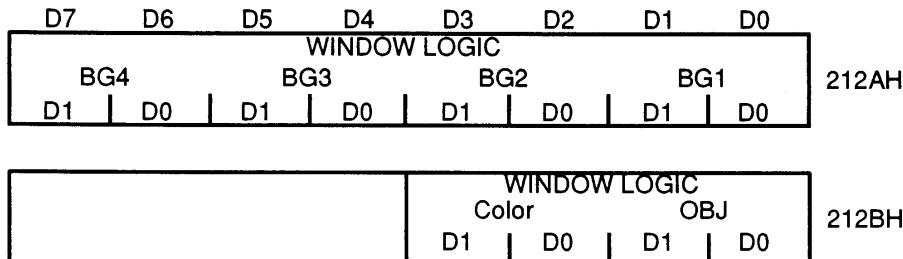
ADDRESS: 2126H/ 2127H/ 2128H/ 2129H  
 NAME: WH0/ WH1/ WH2/ WH3  
 CONTENTS: WINDOW POSITION DESIGNATION (Refer to page A-18)

D7	D6	D5	D4	D3	D2	D1	D0	2126H	WINDOW-1 LEFT POSITION DESIGNATION	2127H	POSITION DESIGNATION
WINDOW H0/H1/H2/H3 POSITION											
P7	P6	P5	P4	P3	P2	P1	P0	2128H		2129H	

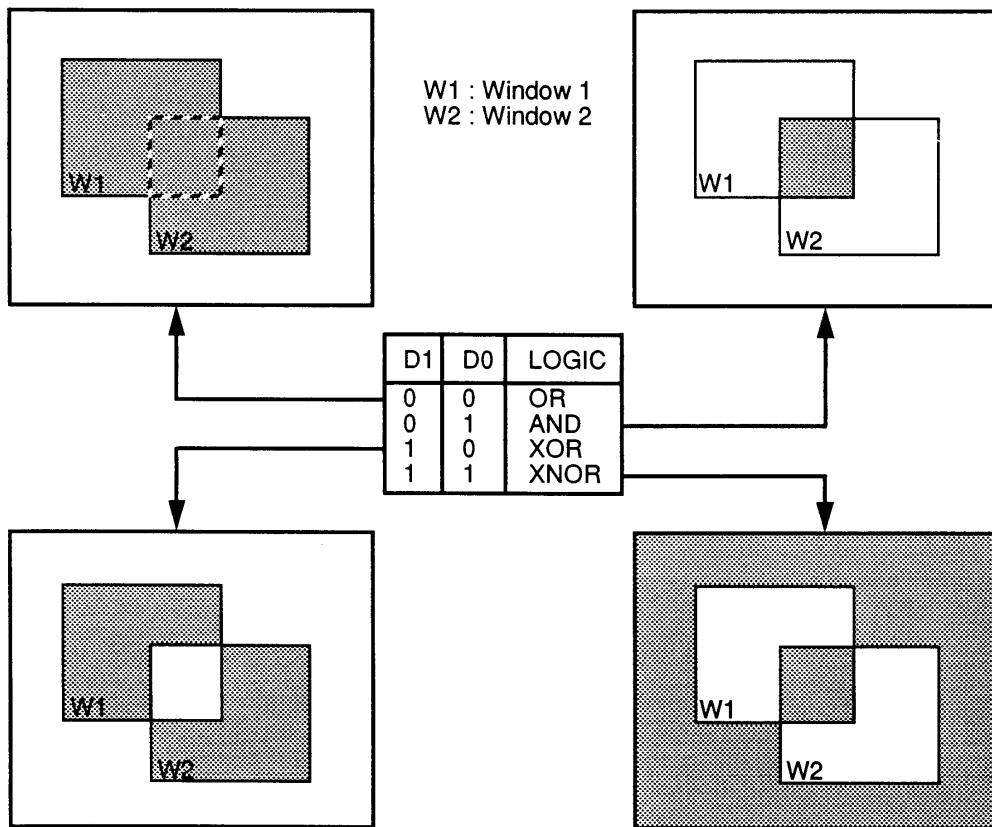
WINDOW H0 POSITION <2126H>:WINDOW-1 Left Position Designation. It can be set in range 0 ~255  
 WINDOW H1 POSITION <2127H>:WINDOW-1 Right Position Designation. It can be set in range 0 ~255  
 WINDOW H2 POSITION <2128H>:WINDOW-2 Left Position Designation. It can be set in range 0 ~255  
 WINDOW H3 POSITION <2129H>:WINDOW-2 Right Position Designation. It can be set in range 0 ~255

NOTE: If 'LEFT POSITION SETTING VALUE> RIGHT POSITION VALUE" is assumed, there will be no range of the window.

ADDRESS: 212AH/ 212BH  
 NAME: WBGLOG/ WOBJLOG  
 CONTENTS: MASK LOGIC SETTINGS FOR WINDOW-1 & 2 ON EACH SCREEN

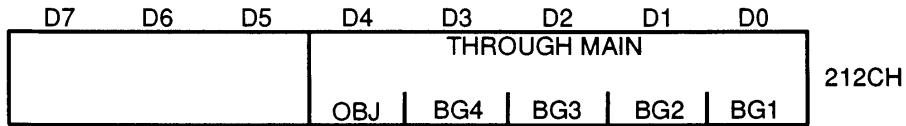


WINDOW LOGIC: SET MASK LOGIC FOR WINDOW-1 & 2  
 When both Window-1 and Window-2 are "IN," the shaded portion will be masked as follows:



NOTE: "IN/OUT" of registers <2123H> <2124H> <2125H> becomes the "NOT logic" for each Window-1 and Window-2.

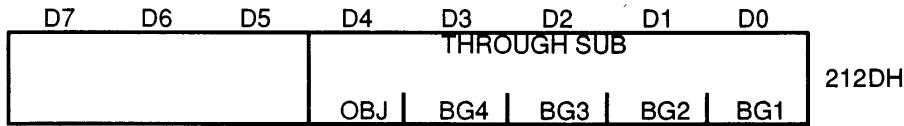
ADDRESS: 212CH  
NAME: TM  
CONTENTS: MAIN SCREEN DESIGNATION



MAIN SCREEN DESIGNATION:  
Designate the screen (BG1 ~ BG4, OBJ)  
to be displayed as the Main Screen.  
Designate the screen to be added for the  
screen addition/subtraction .

- 0 : DISABLE
- 1 : ENABLE

ADDRESS: 212DH  
NAME: TS  
CONTENTS: SUB SCREEN DESIGNATION

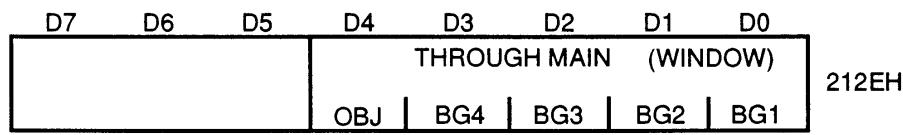


SUB SCREEN DESIGNATION:  
Designate the screen (BG1 ~ BG4, OBJ)  
to be displayed as SUB-Screen.  
Designate the screen to be added for the  
screen addition/subtraction

- 0 : DISABLE
- 1 : ENABLE

NOTE: When the screen addition/subtraction is functioning, the SUB screen is a screen  
to be added or subtracted against the MAIN screen.

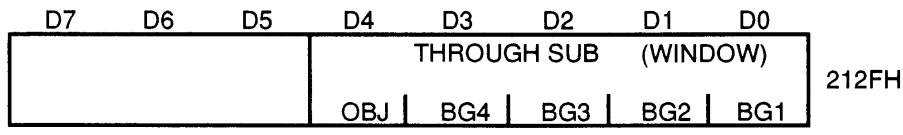
ADDRESS: 212EH  
 NAME: TMW  
 CONTENTS: WINDOW MASK DESIGNATION FOR MAIN SCREEN



WINDOW MASK DESIGNATION FOR MAIN SCREEN:  
 In the window area designated by register <2123H> ~ <2129H>, the screen to be displayed can be designated, which is selected among the Main screen designated by register <212CH>.

- 0 : DISABLE
- 1 : ENABLE

ADDRESS: 212FH  
 NAME: TSW  
 CONTENTS: WINDOW MASK DESIGNATION FOR SUB SCREEN

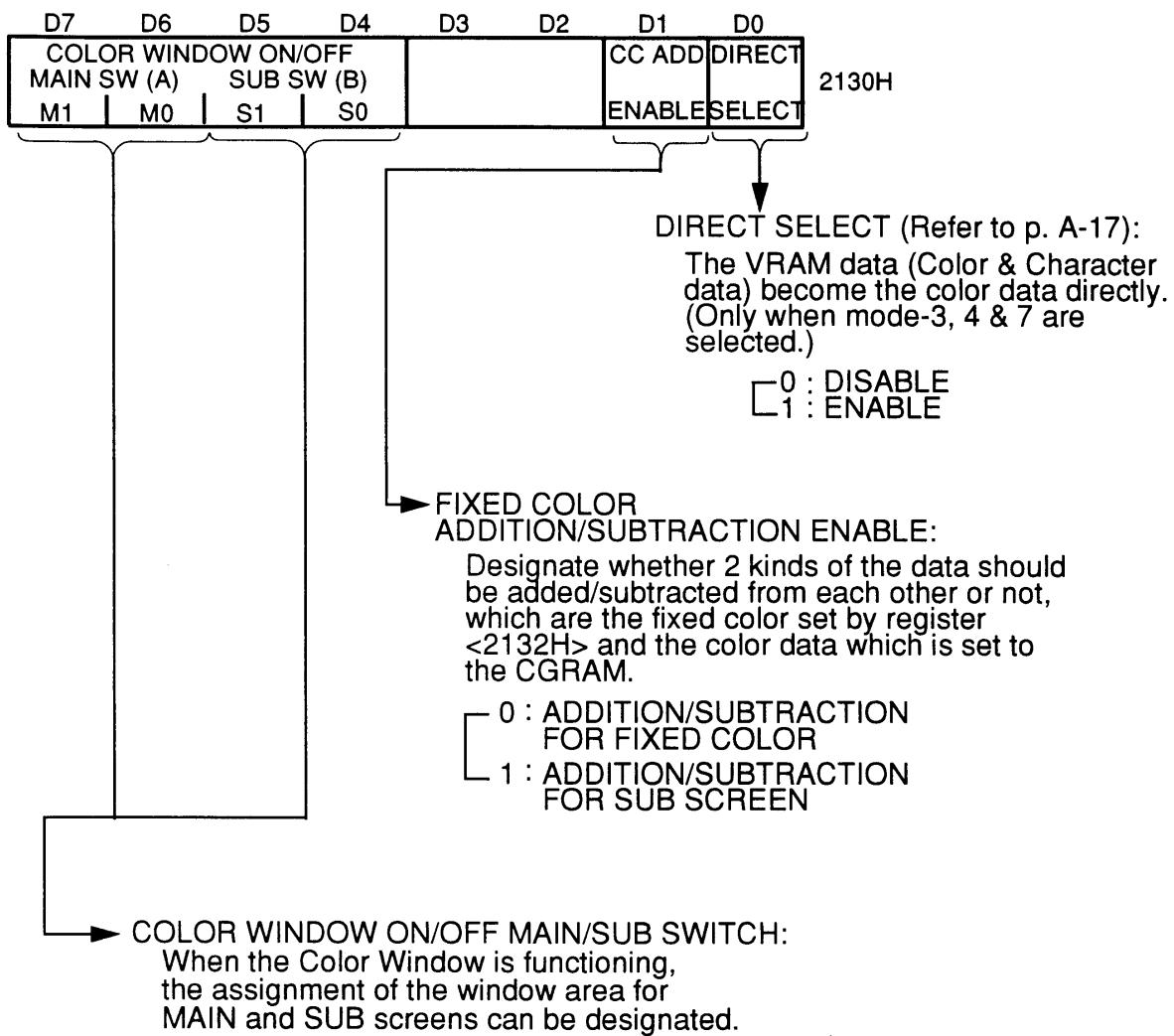


WINDOW MASK DESIGNATION FOR SUB SCREEN:  
 In the window area designated by register <2123H> ~ <2129H>, the screen to be displayed can be designated, which is selected among the Sub screen designated by register <212DH>.

- 0 : DISABLE
- 1 : ENABLE

NOTE: When the screen addition/subtraction is functioning, the SUB screen is a screen to be added or subtracted against the MAIN screen.

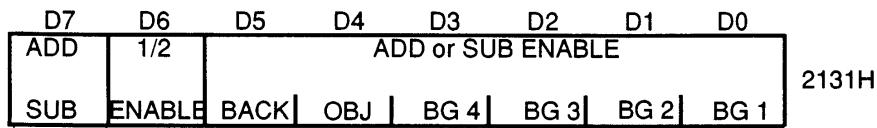
ADDRESS: 2130H  
 NAME: CGSWSEL  
 CONTENTS: INITIAL SETTINGS FOR FIXED COLOR ADDITION OR SCREEN ADDITION



(NCL PG 60)

ADDRESS: 2131H

NAME: CGADSUB

CONTENTS: ADDITION/SUBTRACTION & SUBTRACTION DESIGNATION FOR EACH BG SCREEN  
OBJ & BACKGROUND COLOR**COLOR DATA ADDITION/SUBTRACTION ENABLE:**

Designate the color data of BG1 ~ BG4, OBJ, or Back in the main screen for addition/subtraction of the Sub screen color data (or fixed color data.)

- 0:DISABLE
- 1:ENABLE (Addition/Subtraction function: ON)

Note: When OBJ is designated, the Addition/Subtraction function is available only when the OBJ color palette is 4 through 7.

**"1/2 OF COLOR DATA" DESIGNATION:**

When color constant addition/subtraction or screen addition/subtraction is performed, designate whether the RBG result in the addition/subtraction area should be "1/2" or not. The back (color constant) area on the Sub-screen, will not become "1/2"

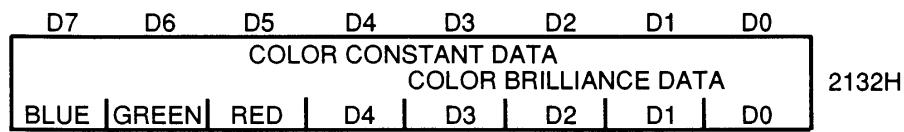
- 0 : DISABLE
- 1 : ENABLE (1/2 function: ON)

**COLOR DATA ADDITION/SUBTRACTION SELECT:**

In the case of executing screen addition/subtraction, designate either addition or subtraction mode.

- 0 : ADDITION MODE SELECT
- 1 : SUBTRACTION MODE SELECT

ADDRESS: 2132H  
 NAME: COLDATA  
 CONTENTS: FIXED COLOR DATA FOR FIXED COLOR ADDITION/SUBTRACTION

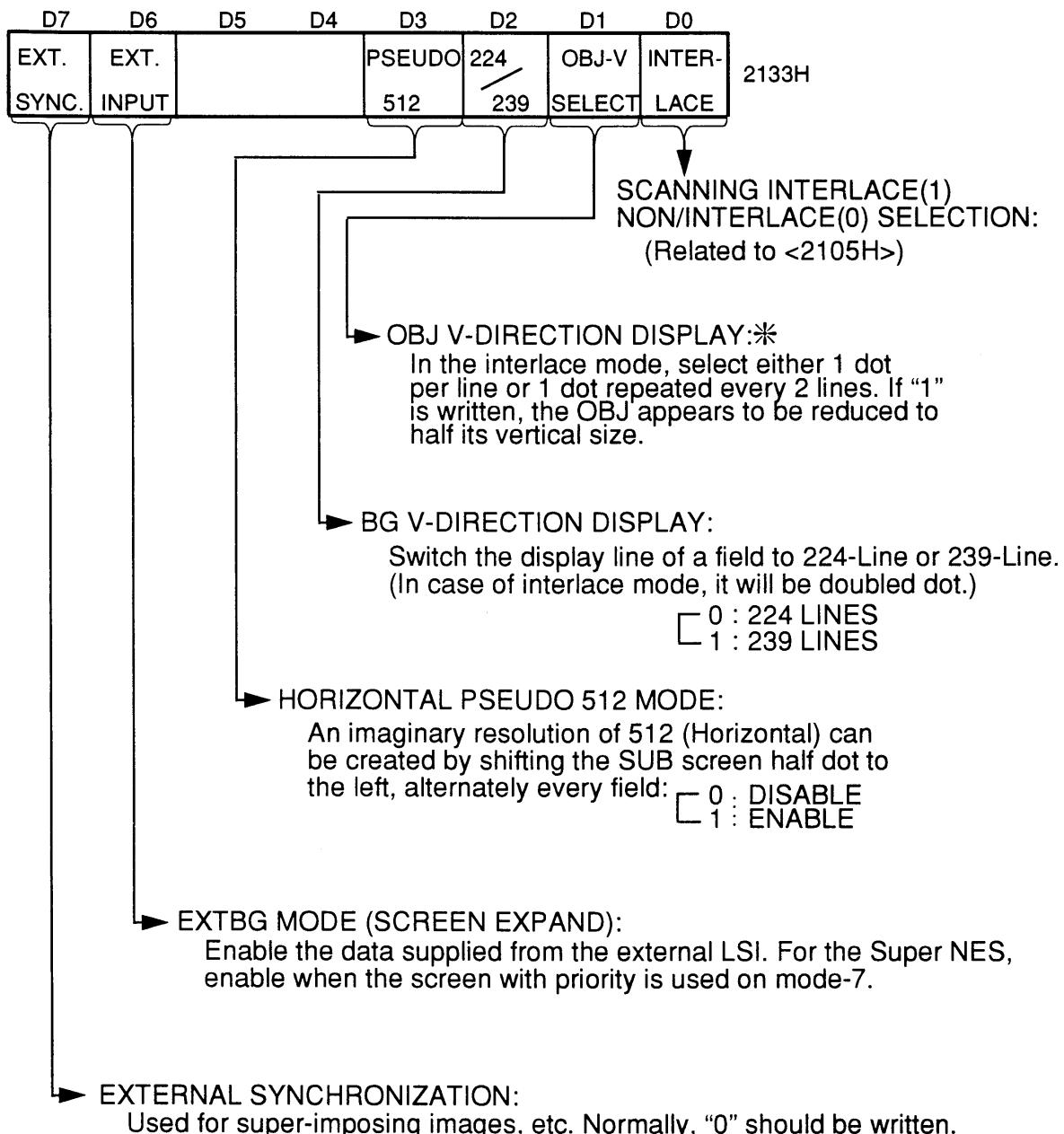


COLOR CONSTANT DATA:  
 Set the color constant data for color constant addition/subtraction.

► COLOR DESIGNATION: Bit for Selecting Desired Color  
 R/G/B brightness should be set using 5-bit data. Example:

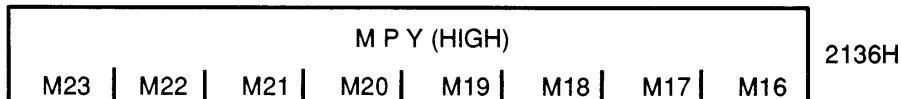
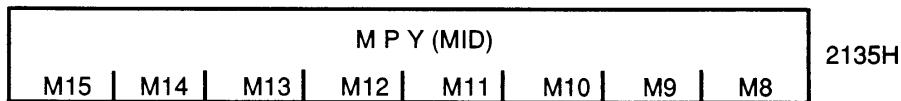
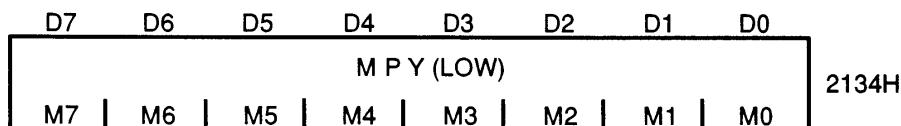
RED	:	C0H, 3FH (B=00H, G=00H, R=1FH)
GREEN	:	A0H, 5FH (B=00H, G=1FH, R=00H)
BLUE	:	60H, 9FH (B=1FH, G=00H, R=00H)
WHITE	:	FFH
BLACK	:	E0H

ADDRESS: 2133H  
 NAME: SETINI  
 CONTENTS: SCREEN INITIAL SETTING



\* If "D1" is set in non-interlace mode, even and odd numbered lines of the OBJ will be displayed alternately every field.

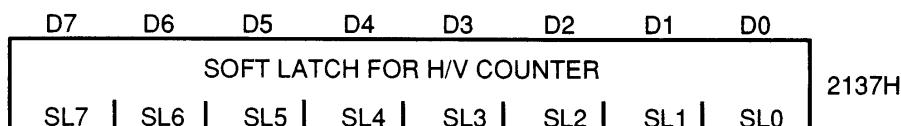
ADDRESS: 2134H / 2135H / 2136H  
NAME: \*MPYL / \*MPYM / \*MPYH  
CONTENTS: MULTIPLICATION RESULT



- This is a Multiplication result (complement of 2) and can be read by setting 16-bit to register <211BH> and setting 8-Bit data to register <211CH>

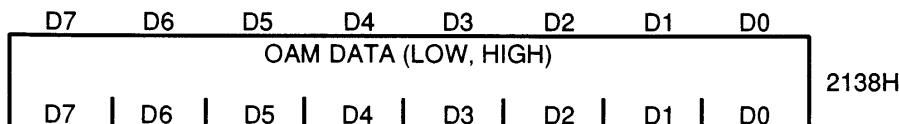
---

ADDRESS: 2137H  
NAME: \*SLHV  
CONTENTS: SOFTWARE LATCH FOR H/V COUNTER



- This is a register, which generates the pulse for latching the H/V counter value.
- The H/V counter value at the point when register <2137H> is read can be latched. The data which was read is meaningless data.
- The H/V counter value latched can be referred by registers <213CH> and <213DH>.

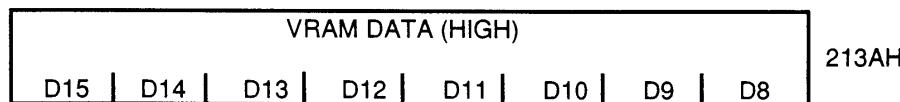
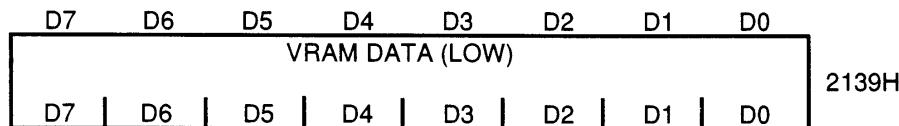
ADDRESS: 2138H  
 NAME: \*OAMDATA  
 CONTENTS: READ DATA FROM OAM



- This is a register, which can read the data at any address of the OAM.
- When the address is set to register <2102H> <2103H> and register <2138H> is also accessed, the data can be read in the order of Low 8-Bit/High 8-Bit. Afterward, the address will be increased automatically, and the data of the next address can be read.

NOTE: The data can be read only during H/V BLANK or FORCED BLANK period.

ADDRESS: 2139H / 213AH  
 NAME: \*VMDATAL / \*VMDATAH  
 CONTENTS: READ DATA FROM VRAM

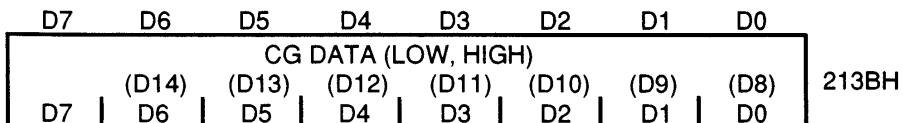


- This is a register, which can read the data at any address of the VRAM.
- The initial address should be set by registers <2116> and <2117H>. The data can be read by the address which has been set initially.
- When reading the data continuously, the first data for the address increment should be read as "dummy" data after the address has been set.
- Quantity to be increased will be determined by "SC INCREMENT" of register <2115H> and the setting value of the "FULL GRAPHIC."

NOTE: The data can be read only during H/V BLANK or FORCED BLANK period.

(NCL PG 63)

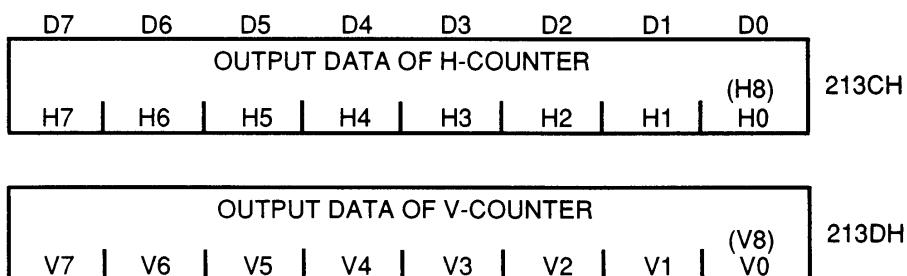
ADDRESS: 213BH  
 NAME: \*CGDATA  
 CONTENTS: READ DATA FROM CG-RAM



- This is a register, which can read the data at any address of the CG-RAM.
- The initial address can be set by register <2121H>. The lower 8-Bit is read first, and then the upper 7-Bit will be read by accessing the register. The current address will be increased to the next address at the same time the upper 7-Bit is read.

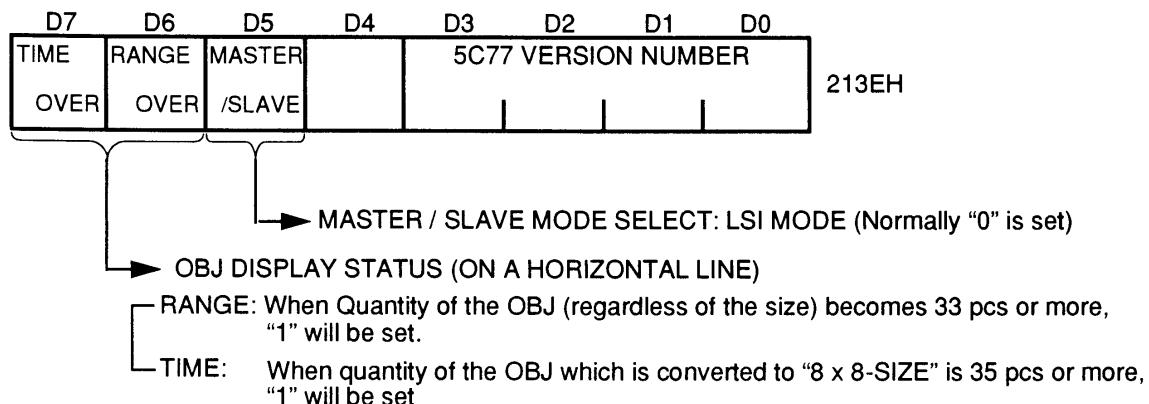
Note: The data can be read only during H/V blank or forced blank period.

ADDRESS: 213CH / 213DH  
 NAME: \*OPHCT / \*OPVCT  
 CONTENTS: H/V COUNTER DATA BY EXTERNAL OR SOFTWARE LATCH



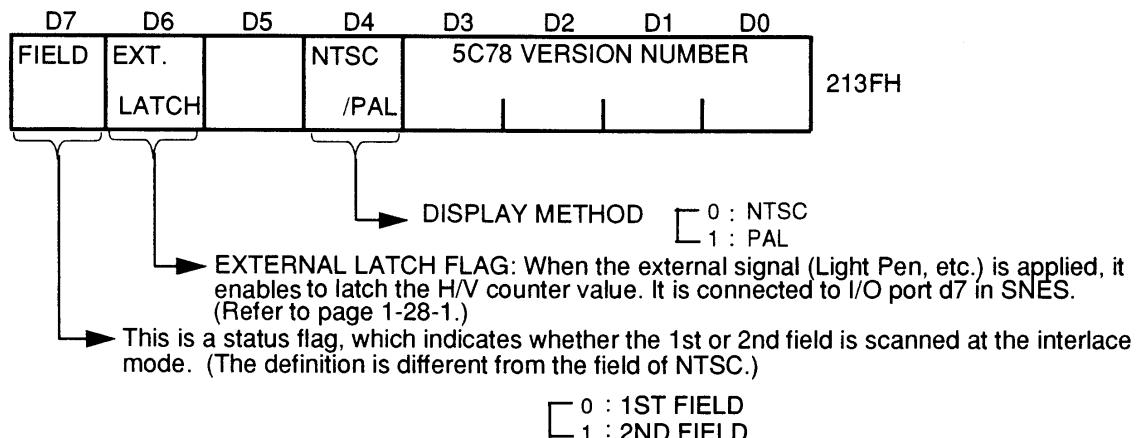
- The H/V counter is latched by reading register <2137H>, and its H/V counter value can be read by this register.
- The H/V counter is also latched by the external latch, and its value can be read by this register.
- If register <213CH> or <213DH> is read after register <213FH> has been read, the lower 8-Bit data will be read first, and then the upper 1-Bit will be read by reading the register.

ADDRESS: 213EH  
 NAME: \*STAT77  
 CONTENTS: PPU STATUS FLAG & VERSION NUMBER



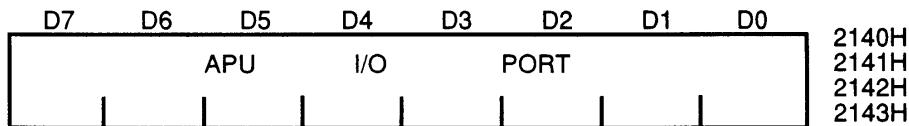
NOTE: The flag will be reset at the end of the V-BLANK period.

ADDRESS: 213FH  
 NAME: \*STAT78  
 CONTENTS: PPU STATUS FLAG & VERSION NUMBER



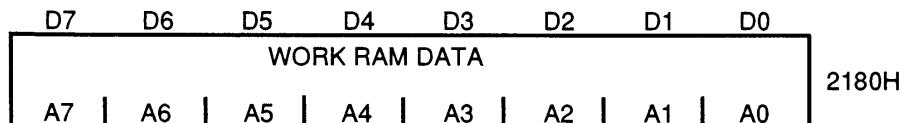
NOTE: When this register is read, registers <213CH> <213DH> will be initialized individually in the order of Low and High.

ADDRESS: 2140H / 2141H / 2142H / 2143H  
NAME: APUIO0 / APUIO1 / APUIO2 / APUIO3  
CONTENTS: COMMUNICATION PORT WITH APU



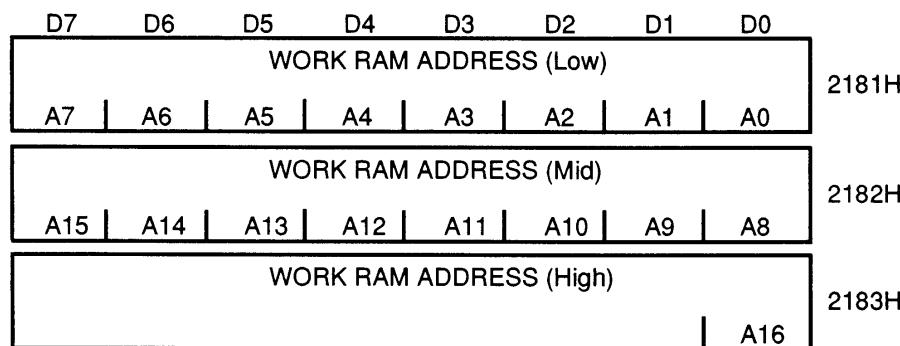
- The port provides more registers for the purpose of IN/OUT, which are 8 registers in total in the APU. Therefore, the different register will be accessed, whether reading or writing for the same address.
- Refer to Part 2 of this manual for the details of the communication method.

ADDRESS: 2180H  
 NAME: WMDATA  
 CONTENTS: DATA to consecutively read from and write to WRAM



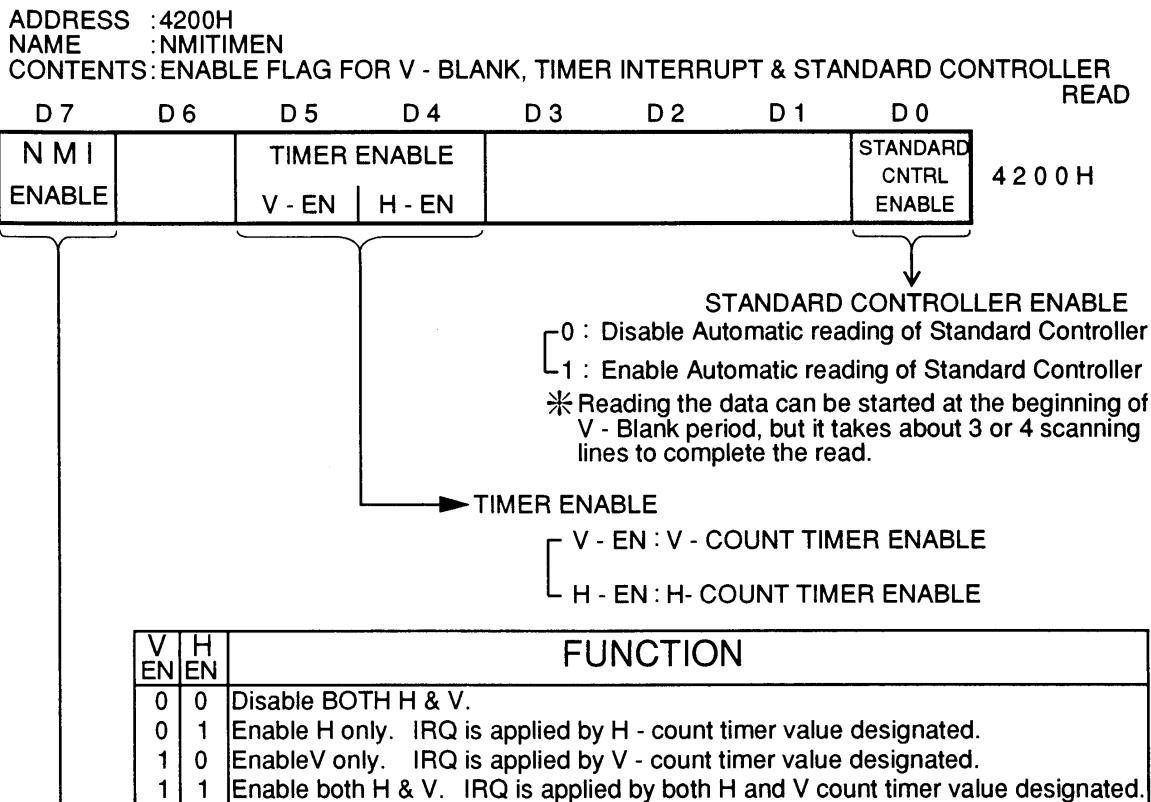
- Data to consecutively read and write at any address of WRAM
- Data is read and written at address set by register <2181H> ~ <2183H>, and address automatically increases each time data is read or written.

ADDRESS: 2181H/2182H/2183H  
 NAME: WMADDL/WMADDM/WMADDH  
 CONTENTS: Address to consecutively read and write WRAM



- Address to be set before WRAM is consecutively read or written.
- A0 through A16 at register <2181H> ~ <2183H> is lower 17 bit address to show address 7E0000 ~ 7FFFFF Memory.

## Chapter 28. CPU Registers



NMI ENABLE : Enable NMI at the point when V - Blank begins.  
 (When power is turned on or the reset signal is applied, it will be "0".)

- 0 : NMI DISABLE
- 1 : NMI ENABLE

---

ADDRESS : 4201H  
 NAME : WRIO  
 CONTENTS: PROGRAMMABLE I/O PORT (OUT - PORT)

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
I/O PORT								4 2 0 1 H
D7	D6	D5	D4	D3	D2	D1	D0	

- This is a Programmable I/O port (OUT - PORT). The written data will be output directly from the OUT - PORT.
- When this is used as a IMPORT, "1" should be written to the particular bit which will be used as a IN - PORT. The input data can be read by register <4213H>.
- Only D6 and D7 can be used by the Super NES. Standard Controller I and III (connector 1) has signal at D6 and Standard Controller II and IV (connector 2) has signal at D7. Signal at D7 is also an external latch input signal (Refer to page 1-27-23).

(NCL PG 88)

ADDRESS : 4202H / 4203H  
 NAME : WRMPYA / WRMPYB  
 CONTENTS: MULTIPLIER & MULTPLICAND BY MULTIPLICATION

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
MULTPLICAND - A								
A7	A6	A5	A4	A3	A2	A1	A0	4202 H
MULTIPLIER - B								
B7	B6	B5	B4	B3	B2	B1	B0	4203 H

- This is a register, which can set as multiplicand (A) and a multiplier (B) for Absolute Multiplication of "A (8-bit) X B (8-bit) = C (16-bit)".
- A PRODUCT (C) can be read by registers <4216H><4217H>.
- Set in the order of (A) and (B). The operation will start as soon as (B) has been set, and it will be completed right after an 8-machine cycle period.
- Once the data of the A-REGISTER is set, it will not be destroyed until new data is set.

ADDRESS : 4204H / 4205H / 4206H  
 NAME : WRDIVL / WRDIVH / WRDIVB  
 CONTENTS: DIVISOR & DIVIDEND BY DIVIDE

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
DIVIDEND- C (LOW)								
C7	C6	C5	C4	C3	C2	C1	C0	4204 H
DIVIDEND- C (HIGH)								
C15	C14	C13	C12	C11	C10	C9	C8	4205 H
DIVISOR - B								
B7	B6	B5	B4	B3	B2	B1	B0	4206 H

- This is a register, which can be set as Dividend (C) and a Divisor (B) for Absolute Divide of "C (16-bit) ÷ B (8-bit) = A (16-bit)".
- The Quotient (A) can be read by registers <4214H><4215H>. And the remainder can be read by registers <4216H><4217H>.
- Set in the order of (C) and (B). The operation will start as soon as (B) has been set, and it will be completed right after a 16-machine cycle period.
- Once the data of the C-REGISTER is set, it will not be destroyed until new data is set.

ADDRESS :4207H / 4208H  
 NAME :HTIMEL / HTIMEH  
 CONTENTS:H-COUNT TIMER SETTINGS

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
H - COUNT TIMER								
H7	H6	H5	H4	H3	H2	H1	H0	4 2 07 H
								H MSB H8 4 2 08 H

- This is a register, which can set the H-COUNT TIMER value.
- The stored value should be from 0 through 339, which is counted from the far left on the screen.
- When the coordinate counter becomes the count value set, the IRQ will be applied.  
And at the same time, "1" will be written to "timer IRQ" of register <4211H> (READ RESET).  
Enable/Disable of the interrupt will be determined by setting register <4200H>.
- ※ This continuous counter is reset every scanning line, therefore once the count value is stored, it is possible to apply the IRQ every time the scanning line comes to the same horizontal position on the screen.

ADDRESS :4209H / 420AH  
 NAME :VTIMEL / VTIMEH  
 CONTENTS:V-COUNT TIMER SETTINGS

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
V - COUNT TIMER								
V7	V6	V5	V4	V3	V2	V1	V0	4 2 09 H
								V MSB V8 4 2 0A H

- This is a register, which can set the V-COUNT TIMER value.
- The stored value should be from 0 through 261 (262), which is counted from top of the screen. (This line number described is different from the actual line number on the screen.)
- When the coordinate counter becomes the count value set, the IRQ will be applied.  
At the same time, "1" will be written to "timer IRQ" of register <4211H> (READ RESET).  
Enable/Disable of the interrupt will be determined by setting register <4200H>.
- ※ This is a continuous counter like the H-counter and will reset every time 262 lines are scanned. Once the count value is stored, it is possible to apply the IRQ every time the scanning line comes to the same vertical position on the screen.

ADDRESS :420BH  
 NAME :MDMAEN  
 CONTENTS:CHANNEL DESIGNATION FOR GENERAL PURPOSE DMA & TRIGGER (START)

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
GENERAL PURPOSE DMA ENABLE FLAG								
CH7 EN	CH6 EN	CH5 EN	CH4 EN	CH3 EN	CH2 EN	CH1 EN	CH0 EN	420B H

- General purpose DMA consists of 8 channels total (CH0 ~ CH7).
- This is used to designate 1 of the 8 channels (8 channels maximum).
- The channel to be used can be designated by writing “1” to the bit of this channel. As soon as “1” is written to the bit (after a few cycles have passed), the general purpose DMA transfer will begin.
- When general purpose DMA of the designated channel is completed, the flag will be cleared.

NOTE: Because the data area (register<4300H> ~) of each channel is held in common with the data of each H-DMA channel, the channel designated by the H-DMA channel designation register <420CH> can not be used. (It is prohibited to write “1” to the bit of the channel). Therefore 8 channels (CH0 ~ CH7) should be assigned by the H-DMA and the general purpose DMA.

NOTE: If the H-Blank comes during the operation of the general purpose DMA and the H-DMA is started, the general purpose DMA will be discontinued in the middle and resumed right after the H-DMA is complete.

NOTE: If 2 or more channels are designated, the DMA transfer will be performed continuously according to the priority order described on page B-1. The CPU will also stop operation until all the general purpose DMAs are completed.

ADDRESS :420CH  
 NAME :HDMAEN  
 CONTENTS:CHANNEL DESIGNATION FOR H-DMA

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
H - DMA ENABLE FLAG								
CH7 EN	CH6 EN	CH5 EN	CH4 EN	CH3 EN	CH2 EN	CH1 EN	CH0 EN	420CH

- The H-DMA consists of 8 channels total (CH0 ~ CH7).
- The register is used to designate the channel out of 8 channels (8 channels maximum).
- The channel which should be used can be designated by writing "1" to the bit of this register. As soon as H-Blank begins (after a few cycles have passed), the H-DMA transfer will begin.

NOTE: Once this flag is set, it will not be cleared until new data is set.  
 The initial settings are done automatically for every field and the same transfer pattern will be repeated. The flag is also set out of V-Blank period, so the DMA transfer will be performed properly for the next screen frame.

ADDRESS :420DH  
 NAME :MEMSEL  
 CONTENTS:ACCESS CYCLE DESIGNATION IN MEMORY ② AREA (Refer to "Memory Map")

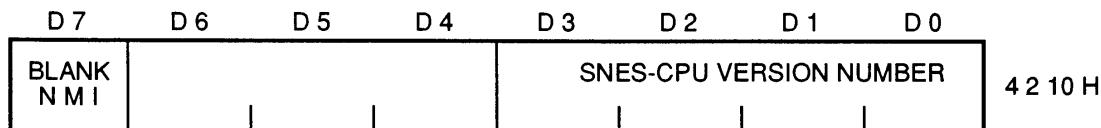
D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
								2.68 / 3.58 420DH

#### ACCESS CYCLE DESIGNATION IN MEMORY ② AREA

- 0 : 2.68MHz access cycle
- 1 : 3.58MHz access cycle (Only when the high speed memory is used)

- MEMORY ② shows the address (8000H ~ FFFFH) of the bank (80H ~ BFH) and all the addresses of the bank (C0H ~ FFH).
- When power is turned on or the reset signal is applied, it becomes "0".

ADDRESS :4210H  
 NAME :★ RDNMI  
 CONTENTS:NMI FLAG BY V - BLANK & VERSION NUMBER



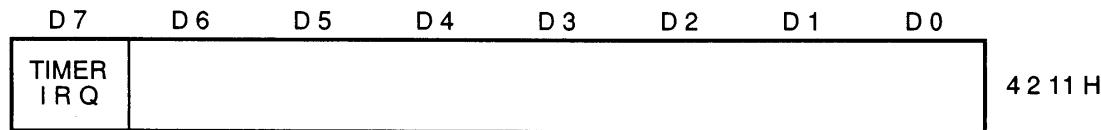
► NMI FLAG BY V-BLANK : When "1" is written to "NMI ENABLE" of register <4200H>, this flag will show NMI status.

- 0 : NMI has not occurred
- 1 : NMI has occurred

\* A "1" is written to this flag at the beginning of V-Blank and a "0" is written at the end of V-Blank.  
 It can also be reset by reading this register.

NOTE : It is necessary to reset by reading this flag during NMI processing. (Refer to page B-3.)

ADDRESS :4211H  
 NAME :★ TIMEUP  
 CONTENTS:IRQ FLAG BY H/V COUNT TIMER



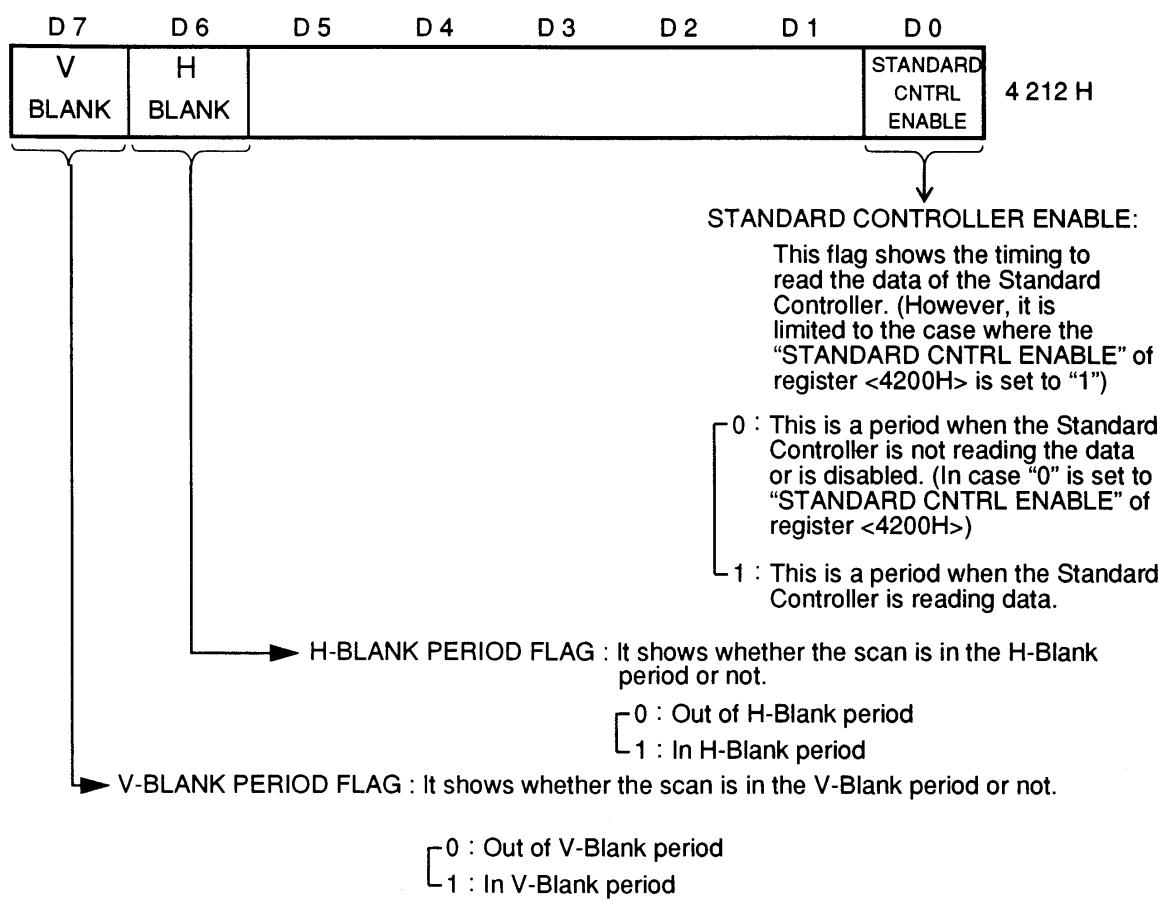
► IRQ FLAG BY H/V COUNT TIMER :

This flag is "READ RESET". (If Timer Enable is set by "Timer Enable" of register <4200H>, IRQ will be applied and the flag will be set as soon as H/V count timer reaches the value stored.

\* Even if V-EN ="0" and H-EN ="0" are set by "Timer Enable" of register <4200H>, this flag will be reset.

- 0 : Either H/V count timer is active or disabled
- 1 : Status of H/V count timer is Time-Up

ADDRESS : 4212H  
 NAME : \* HVBJOY  
 CONTENTS: H/V BLANK FLAG & STANDARD CONTROLLER ENABLE FLAG



ADDRESS : 4213H  
 NAME : \* RDIO  
 CONTENTS: PROGRAMMABLE I/O PORT (IN - PORT)

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
I/O PORT							4 213 H
D7	D6	D5	D4	D3	D2	D1	D0

- This is a Programmable I/O port (IN - PORT). The data which is set to the IN-PORT should be read directly.
- The bit in which "1" is written by register <4201H> is used as the IN-PORT.
- Only D6 and D7 can be used by the Super NES. Standard Controller I and III (connector 1) has signal at D6 and Standard Controller II and IV (connector 2) has signal at D7. Signal at D7 is also an external latch input signal (Refer to "PPU Status Flag and Version Number" in "PPU Registers").

(NCL PG 93)

ADDRESS :4214H / 4215H  
 NAME :\* RDDIVL / \* RDDIVH  
 CONTENTS: QUOTIENT OF DIVIDE RESULT

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
QUOTIENT - A (LOW)								
A7	A6	A5	A4	A3	A2	A1	A0	4214 H
QUOTIENT - A (HIGH)								
A15	A14	A13	A12	A11	A10	A9	A8	4215 H

- This is Quotient (A), which is a result of absolute division of "C (16 BIT) ÷ B (8 BIT) = A (16 BIT)".
- Dividend (C) and divisor (B) are set by registers <4204H>, <4205H>, and <4206H>.

ADDRESS :4216H / 4217H  
 NAME :\* RDMPYL / \* RDMPYH  
 CONTENTS: PRODUCT OF MULTIPLICATION RESULT OR REMAINDER OF DIVIDE RESULT

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
PRODUCT-C [MULTIPLICATION] / REMAINDER [DIVIDE] (LOW)								
C7	C6	C5	C4	C3	C2	C1	C0	4216 H
PRODUCT-C [MULTIPLICATION] / REMAINDER [DIVIDE] (HIGH)								
C15	C14	C13	C12	C11	C10	C9	C8	4217H

#### ① WHEN USED FOR MULTIPLICATION

- This is a Product (C), which is a result of Absolute Multiplication of "A (8 BIT) X B (8 BIT) = C (16 BIT)".
- Multiplicand (A) and Multiplier (B) are set by registers <4202H> and <4203H>.

#### ② WHEN USED FOR DIVISION

- This is a Remainder, which is a result of Absolute Division of "C (16 BIT) ÷ B (8 BIT) = A (16 BIT) • • • REMAINDER (8 or 16 bit)".
- Dividend (C) and divisor (B) are set by registers <4204H>, <4205H>, and <4206H>.

ADDRESS : 4218H/4219H/421AH/421BH/421CH/421DH/421EH/421FH  
 NAME : STD CNTRL1L / 1H / 2L/2H/3L/3H/4L/4H  
 CONTENTS: DATA FOR STANDARD CONTROLLER I, II, III, & IV

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
A BUTTON	X BUTTON	L BUTTON	R BUTTON	STANDARD CONTROLLER -I (LOW)				4 218 H
B BUTTON	Y BUTTON	SELECT BUTTON	START BUTTON	UP	DOWN	LEFT	RIGHT	STANDARD CONTROLLER -I (HIGH)
A BUTTON	X BUTTON	L BUTTON	R BUTTON	STANDARD CONTROLLER -II (LOW)				4 21AH
B BUTTON	Y BUTTON	SELECT BUTTON	START BUTTON	UP	DOWN	LEFT	RIGHT	STANDARD CONTROLLER -II (HIGH)
A BUTTON	X BUTTON	L BUTTON	R BUTTON					STANDARD CONTROLLER -III (LOW)
B BUTTON	Y BUTTON	SELECT BUTTON	START BUTTON	UP	DOWN	LEFT	RIGHT	STANDARD CONTROLLER -III (HIGH)
A BUTTON	X BUTTON	L BUTTON	R BUTTON					STANDARD CONTROLLER -IV (LOW)
B BUTTON	Y BUTTON	SELECT BUTTON	START BUTTON	UP	DOWN	LEFT	RIGHT	STANDARD CONTROLLER -IV (HIGH)

For controller expansion ←

- Registers <4016H> <4017H> can be used the same as the NES.

	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	PORT
4016H RD									4016H D0 : Data for Controller I 4016H D1 : Data for Controller III
4016H WR									OUT0, OUT1, OUT2 (OUT1 and OUT2 are not output from SNES)
4017H RD									4017H D0 : Data for Controller II 4017H D1 : Data for Controller IV

NOTE : Whether the standard controllers are connected to Super NES unit or not can be determined by reading the 17th bit of 4016H and 4017H. (Refer to "Standard Controller".)

[ 1 : connected  
 0 : not connected ]

ADDRESS : 43X0H (X : CHANNEL NUMBER <0 ~ 7>)

**NAME**

## CONTENTS: PARAMETER FOR DMA TRANSFER

D 7      D 6      D 5      D 4      D 3      D 2      D 1      D 0

CH *	CH TYPE		A BUS ADDRESS INC/DEC	FIXED	CH TRANSFER D2	WORD SELECT D1	D0
---------	------------	--	--------------------------	-------	-------------------	-------------------	----

43X0H

\* Transfer Origination

DMA TRANSFER WORD SELECT

GENERAL PURPOSE DMA : B-ADDRESS CHANGE  
METHOD DESIGNATION  
PER CHANNEL

D2	D1	D0	ADDRESS TO BE WRITTEN
0	0	0	1-ADDRESS
0	0	1	2-ADDRESS (VRAM etc.) L,H
0	1	0	1-ADDRESS (WRITE TWICE)
0	1	1	2-ADDRESS (WRITE TWICE) L,L,H,H
1	0	0	4-ADDRESS L,H,L,H

H-DMA : The number of bytes to be transferred per line and write method designation.

D2	D1	D0	# OF BYTE TO BE TRANS-FERRED	ADDRESS TO BE WRITTEN
0	0	0	1 BYTE	1-ADDRESS
0	0	1	2 BYTE	2-ADDRESS (VRAM etc.) L,H
0	1	0	2 BYTE	1-ADDRESS (WRITE TWICE) L,L
0	1	1	4 BYTE	2-ADDRESS (WRITE TWICE) L,L,H,H
1	0	0	4 BYTE	4-ADDRESS L,H,L,H

→ FIXED ADDRESS FOR A-BUS & AUTOMATIC INCREMENT /DECREMENT SELECT [IN CASE OF GENERAL PURPOSE DMA]

D3      0 : Automatic address increment/decrement  
        1 : Fixed address (To be used when clearing VRAM etc.)

D4      0 : Automatic increment (In case "0" is written to D3)  
        1 : Automatic decrement

→ TYPE DESIGNATION [H-DMA ONLY] : Addressing mode designation when accessing the data (Refer to page B-2).  
 0 : ABSOLUTE ADDRESSING  
 1 : INDIRECT ADDRESSING

→ TRANSFER ORIGINATION DESIGNATION : Transfer direction A Bus → B Bus  
B Bus → A Bus Designation (Refer to page B-1)  
 0 : A-BUS → B-BUS (CPU MEMORY → PPU)  
 1 : B-BUS → A-BUS (PPU → CPU MEMORY)

\* For example, in case the DMA transfer is performed from CPU memory to PPU, "0" should be written.

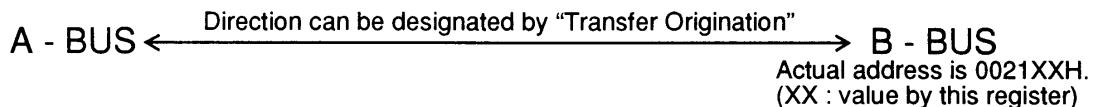
ADDRESS : 43X1H (X : CHANNEL NUMBER &lt;0 ~ 7&gt;)

NAME :

CONTENTS: B-BUS ADDRESS FOR DMA

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
B - ADDRESS								
BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	43X1H

- This is a register which can set the address of B-bus.
- Whether this is the address of the "Transfer Destination" or the address of the "Transfer Origination" can be determined by D7 (Transfer Origination) of register <43X0H>.



\* When H-DMA is performed, it will be the address of the "Transfer Destination".

ADDRESS : 43X2H / 43X3H / 43X4H (X : CHANNEL NUMBER &lt;0 ~ 7&gt;)

NAME :

CONTENTS: TABLE ADDRESS OF A-BUS FOR DMA &lt;A1 TABLE ADDRESS&gt;

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
A1 TABLE ADDRESS (LOW)								
A7	A6	A5	A4	A3	A2	A1	A0	43X2H
A1 TABLE ADDRESS (HIGH)								
A15	A14	A13	A12	A11	A10	A9	A8	43X3H
A-TABLE BANK								
A23	A22	A21	A20	A19	A18	A17	A16	43X4H

- This is a register, which can set the address of A-bus.
- Whether this is the address of the "Transfer Destination" or the address of the "Transfer Origination" can be determined by D7 (Transfer Origination) of register <43X0H>. A "0" should be written to D7 except in special cases.
- In the H-DMA mode, the address of the transfer origination is designated except it is a special case. Therefore, for the CPU area designated by this address, the data (page B-2) must be set by the absolute addressing mode or the indirect addressing mode.
- This address becomes the basic address on the A-Bus during DMA transfer period and the address will be increased or decreased based on this address. (When the general purpose DMA is performed, it will be decreased.)

ADDRESS : 43X5H / 43X6H / 43X7H (X : CHANNEL NUMBER &lt;0 ~ 7&gt;)

NAME :

CONTENTS: DATA ADDRESS STORE BY H-DMA

&amp; NUMBER OF BYTE TO BE TRANSFERRED SETTINGS BY GENERAL PURPOSE DMA

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
<b>DATA ADDRESS (LOW)</b>								
DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	FOR H-DMA
NUMBER OF BYTES TO BE TRANSFERRED (LOW)								
B7	B6	B5	B4	B3	B2	B1	B0	43X5H GENERAL PURPOSE DMA
<b>DATA ADDRESS (HIGH)</b>								
DA15	DA14	DA13	DA12	DA11	DA10	DA9	DA8	FOR H-DMA
NUMBER OF BYTES TO BE TRANSFERRED (HIGH)								
B15	B14	B13	B12	B11	B10	B9	B8	43X6H GENERAL PURPOSE DMA
<b>DATA BANK</b>								
DA23	DA22	DA21	DA20	DA19	DA18	DA17	DA16	FOR H-DMA
								43X7H

- IN CASE OF H-DMA

This is a register in which the indirect address will be stored automatically in the Indirect addressing mode. The indirect address means the data described on page B-2. It is not necessary to read or write directly by the CPU except in special cases.

- IN CASE OF GENERAL PURPOSE DMA

This is the register which can set the number of bytes to be transferred. However, the number of Byte (0000H) means 10000H.

ADDRESS : 43X8H / 43X9H (X : CHANNEL NUMBER <0 ~ 7>  
 NAME :  
 CONTENTS: TABLE ADDRESS OF A-BUS BY DMA <A2 TABLE ADDRESS>

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
A2 TABLE ADDRESS (LOW)								
A7	A6	A5	A4	A3	A2	A1	A0	43X8H
A2 TABLE ADDRESS (HIGH)								
A15	A14	A13	A12	A11	A10	A9	A8	43X9H

- This is the address which is used to access the CPU and RAM. It will be increased automatically. (See page B-2.)
  - The data of this register is used as the basic address which is the address set by the "A1 Table Address". Afterwards, because it will be increased (or decreased) automatically, it is not necessary to set the address into this register by the CPU directly.
- { However, if the data which is transferred needs to be changed by force, it can be done by setting the CPU memory address to this register. In such case, the address of the CPU which is accessed currently will be changed by reading this register. } H-DMA ONLY

ADDRESS : 43XAH (X : CHANNEL NUMBER <0 ~ 7>  
 NAME :  
 CONTENTS: THE NUMBER OF LINES TO BE TRANSFERRED BY H-DMA

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0	
NUMBER OF LINES								
Continue	L6	L5	L4	L3	L2	L1	L0	43XAH

- This is the register which shows the number of lines for H-DMA transfer. (Refer to page B-2.)
- The number of lines written to the CPU memory will be the basic number of lines. It is not necessary to write the data into this register by the CPU directly.

## ***Chapter 1. SNES Sound Source Outline***

### **1.1 OUTLINE**

The SNES sound source is composed of a Sound-CPU-IC, a single chip in which are integrated an 8-bit CPU, IPL ROM, I/O ports, a DSP-IC, and peripheral apparatus.

## **CHARACTERISTICS**

- CPU core : Sony SPC700 series CMOS 8-bit CPU
- Minimum Command Execution Time :  $1.953\mu s/2.48MHz$  when active
- Internal ROM : 64 byte (IPL ROM)
- Memory Space : 64K byte
- Peripheral Functions
- I/O Ports : SNES CPU Interface I/O Ports 8 bit x 4  
Universal I/O Ports 8 bit x 2
- Timers : (8 bit timer + 4 bit counter) x 3 sets
- Output Sound Production : 4-bit ADPCM sampling sound x 8 tones  
(simultaneous production)

## 1.2 SYSTEM OUTLINE

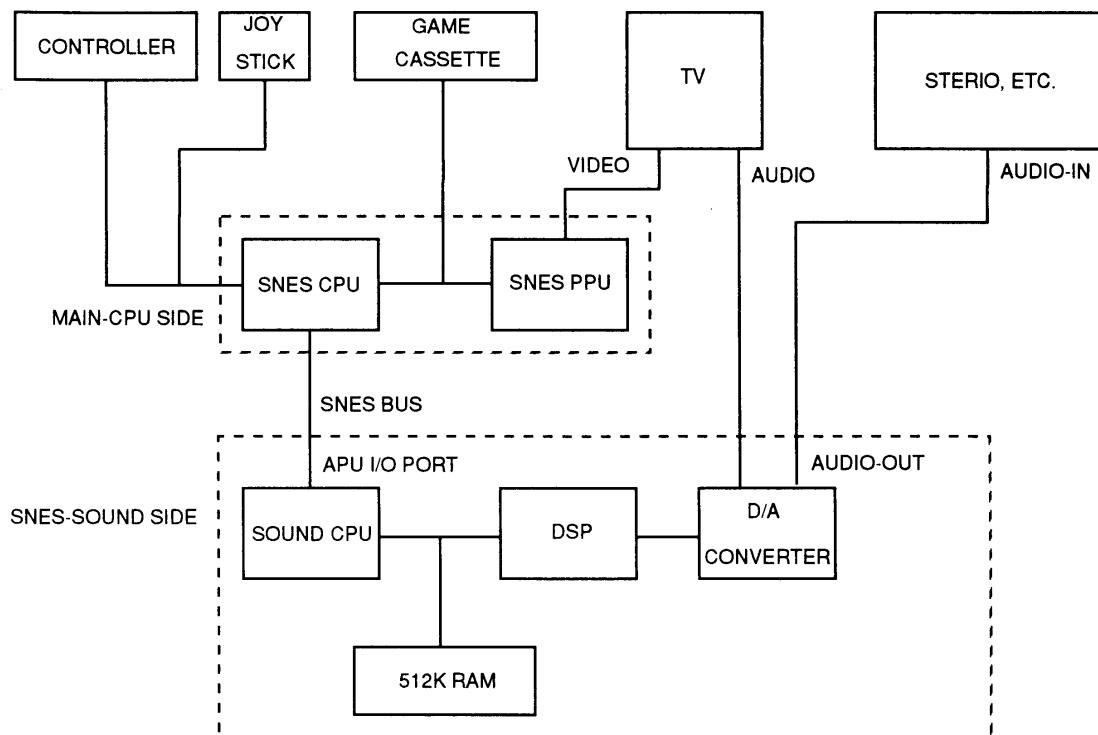


Figure 3-1-1 System Block Diagram

## 1.3 Designation and Role of Each Section:

### 1.3.1 Sound-CPU:

SNES sound source CPU. Program and tone color data are read into RAM from the game cassette through the SNES CPU, Consequently controlling the game music.

In addition, the Sound- CPU is provided with an internal IPL-ROM which is activated upon reset. The IPL-ROM provides for transmission of data through the SNES CPU, initial settings of the SNES sound source, etc.

### 1.3.2 DSP:

Digital Signal Processor. Reproduces tone quality data in RAM. Carries out various functions for the purpose of musical expression.

### 1.3.3 512K RAM:

Shared on a time basis by the Sound-CPU and the DSP.

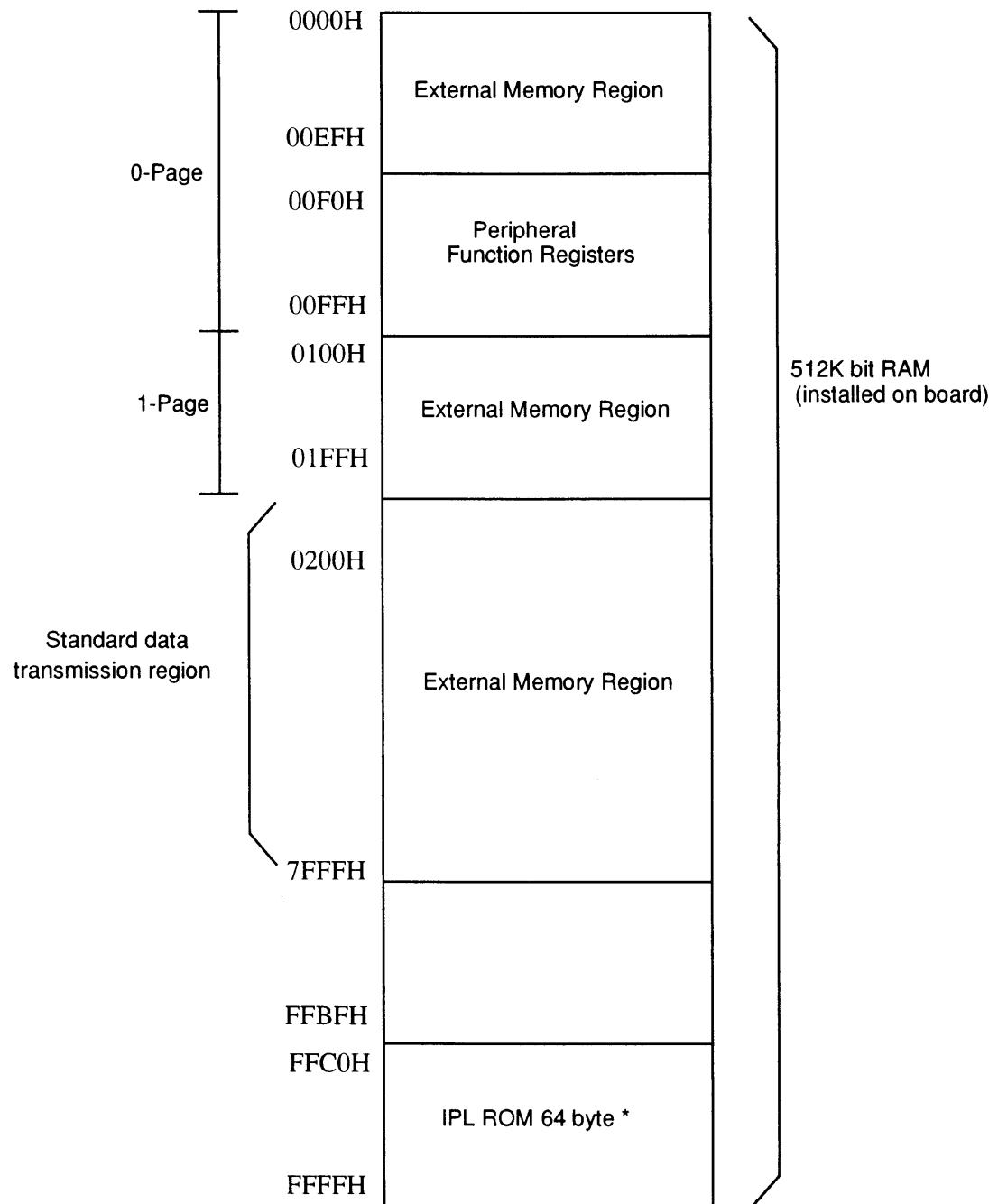
### 1.3.4 SNES CPU:

CPU for SNES use. Carries out progression of the game in conformity with the game cassette format.

### 1.3.5 SPPU:

PPU for SNES use. Creates imaging through CPU control.

## 1.4 MEMORY MAPPING

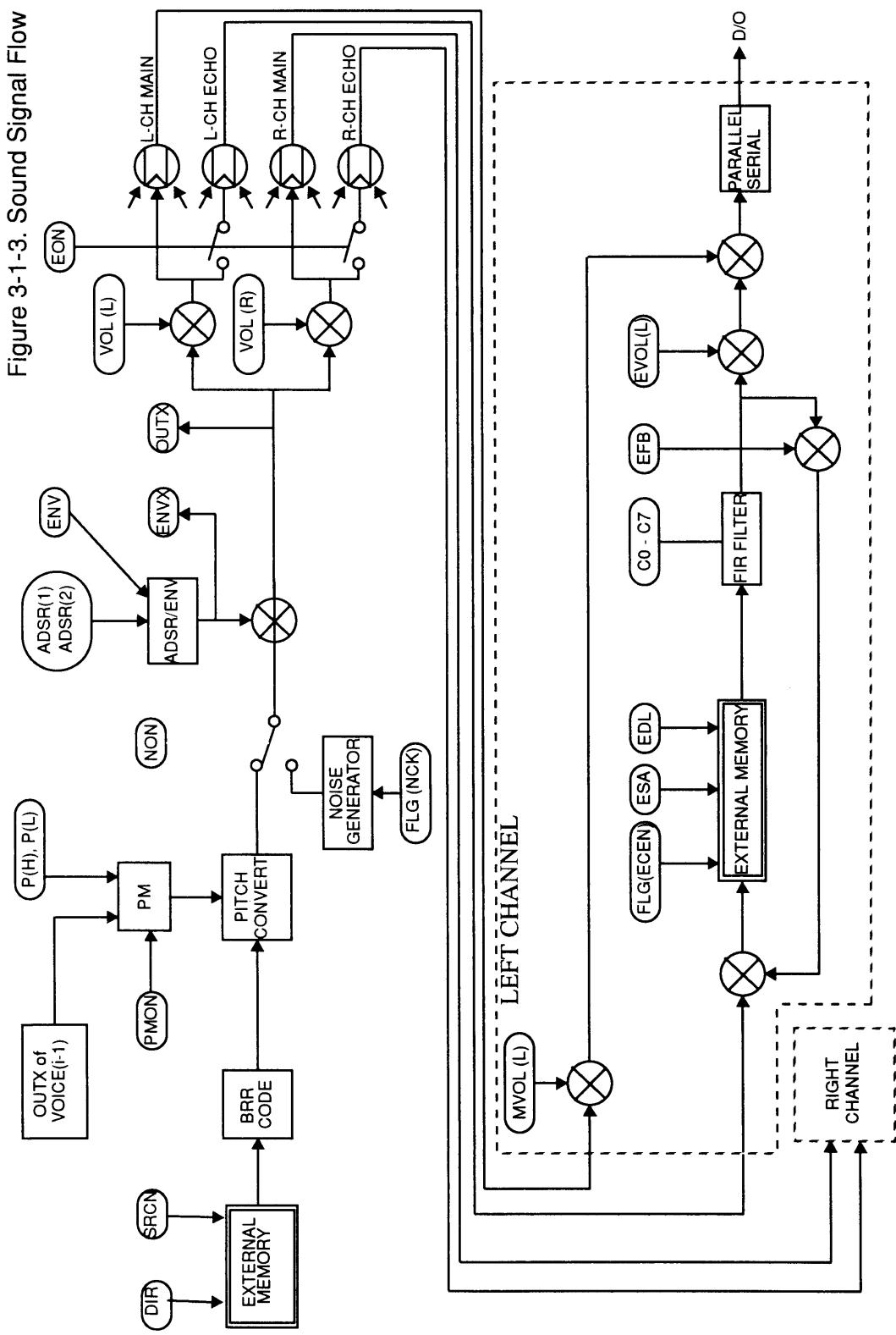


\* The initial hardware setting program is installed in the IPL ROM

Figure 3-1-2 Memory Map

(NCL PG 4)

## 1.5 SIGNAL FLOW



(NCL PG 5)

## Chapter 2. BRR (Bit Rate Reduction)

Sound data for the Super NES is recorded on a game data cassette in 4-bit ADPCM format. Creating data in this format requires the use of a technique called "BRR" for some sounds.

Knowledge of this operation is not necessary to correctly create data with the Sony NEWS system, which has been the standard tool used by Nintendo to date. When the same data is created using a different tool, however, one must understand BRR.

Complications arise during the creation of sound data from data in BRR format when creating the position of the program which selects the filter number described below. Also, Nintendo cannot currently support this programming effort.

One block of wave form data is comprised of a one-byte header and eight-byte wave form (4 bit x 16 samples). This is the minimum unit the sound IC can handle.

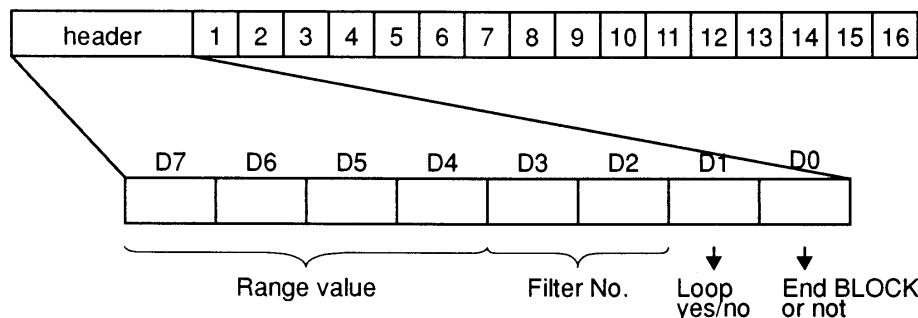


Figure 3-2-1 BRR Data String

From this, the range and filter contain the BRR information (how the data string in 1 - 16 will be regenerated). Refer to ¶ 7.2.2.5 for information on d1 and d0 bits.

The "filter number" can have a value from 0 ~ 3. These indicate, respectively, 0: constant, 1: first-order filter, 2: second-order filter, 3: third-order filter. Of these values, 1 ~ 3 require previous data values for data calculation. (Filter 1 requires one previous sample, and filters 2 and 3 require two previous samples.)

The "range" indicates the number of bits shifted. One sample data (4-bit) is shifted to the left for the number of bits and re-created as 16-bit data. The maximum value for a range is 12(1100). (See the next page.)

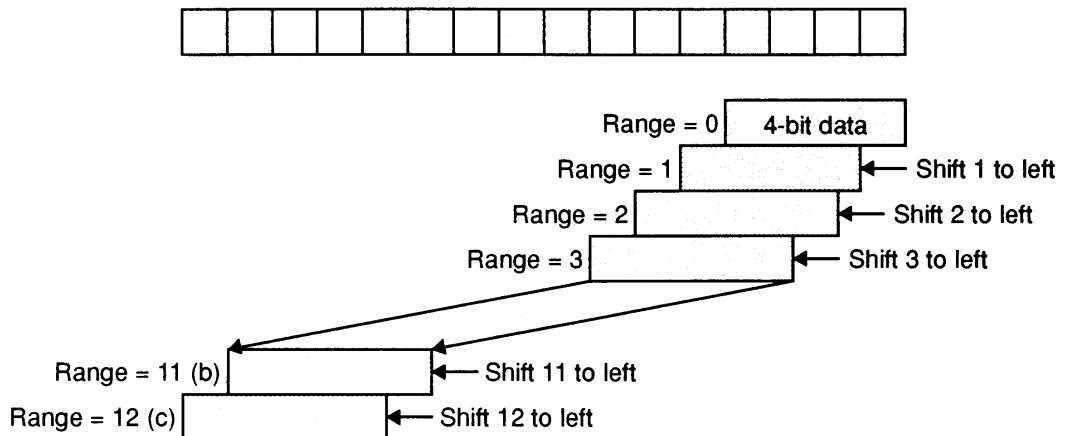


Figure 3-2-2 BRR Range Data

An equation for decoding a sample value,  $x$  from the previous sample  $x_{-1}$  and second previous sample  $x_{-2}$ , is given below.

$$x = R + ax_{-1} + bx_{-2}$$

$R$  is the value obtained by shifting the 4-bit data,  $d$ , by the range value,  $r$ .

$$R = [d] 2^{r-15}$$

([d] is a decimal presentation of  $d$ , which is in two's compliment form, -7 ~+8)

The values of  $a$  and  $b$  for each filter are as follows:

Filter No.	a	b
0	0	0
1	0.9375	0
2	1.90625	-0.9375
3	1.796875	-0.8125

Table 3-2-1 BRR Filter Values

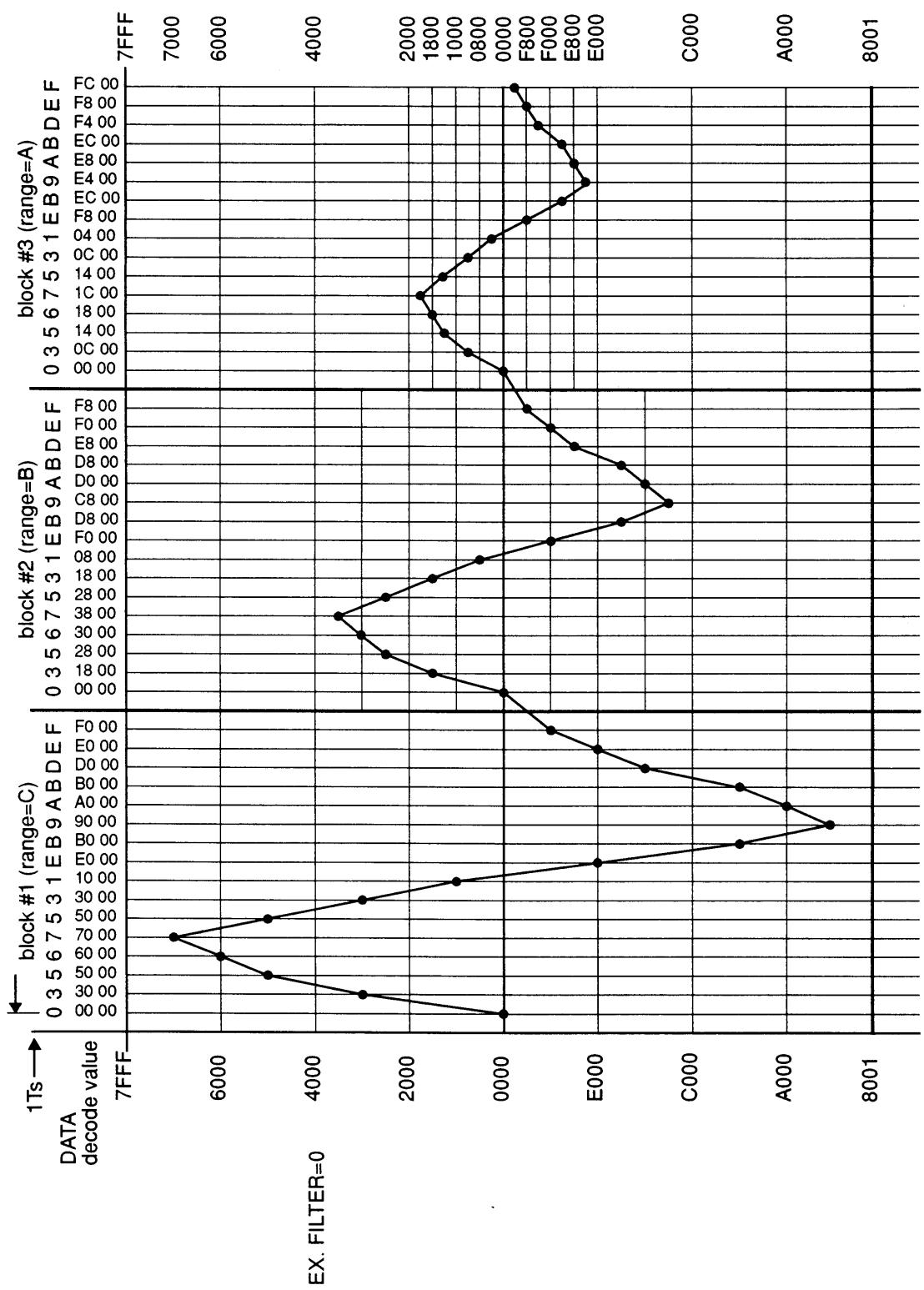


Figure 3-2-3 Example Data when Filter = 0.

## Chapter 3. I/O Ports

### 3.1 PERIPHERAL FUNCTIONS REGISTERS

Peripheral Function Registers

Table 3-3-1. Peripherals

Address	Function Register	R/W	When Reset	Remarks
00FOH	(test)	---	-----	Installed in Sound-CPU
00F1H	Control	W	Control = "--00-000"	
00F2H	Register Add.	R/W	Indeterminate	Installed in DSP
00F3H	Register Data	R/W	Indeterminate	Installed in DSP
00F4H	Port-0	R/W	Port 0r = "00" Port 0w = "00"	Installed in Sound-CPU
00F5H	Port-1	R/W	Port 1r = "00" Port 1w = "00"	Installed in Sound-CPU
00F6H	Port-2	R/W	Port 2r = "00" Port 2w = "00"	Installed in Sound-CPU
00F7H	Port-3	R/W	Port 3r = "00" Port 3w = "00"	Installed in Sound-CPU
00F8H	-----	---	-----	-----
00F9H	-----	---	-----	-----
00FAH	Timer-0	W	Indeterminate	Installed in Sound-CPU
00FBH	Timer-1	W	Indeterminate	Installed in Sound-CPU
00FCH	Timer-3	W	Indeterminate	Installed in Sound-CPU
00FDH	Counter-0	R	Indeterminate	Installed in Sound-CPU
00FEH	Counter-1	R	Indeterminate	Installed in Sound-CPU
00FFH	Counter-3	R	Indeterminate	Installed in Sound-CPU

(NCL PG 6)

### 3.2 APU I/O PORTS

Ports 0-3 are ports which carry out data transmission to the SCPU through the SNES bus and are composed of four 8-bit input registers and four 8-bit output registers. Port  $n\ r$  registers can only write from the SCPU section and can only read from the Sound-CPU section. The opposite is true of the port  $n\ w$  registers. Since the composition of each of these ports is identical, only an explanation of Port 0r and Port 0w is provided.

1. Data is input into Port 0r when the SCPU writes data into 2140H. Then, the contents of Ports 0r are read when the Sound-CPU reads the data in 00F4H (this is also true of Ports 1r - 3r).
2. Data is written into Port 0w when the Sound-CPU writes data into the APU I/O port (00F4H). Then the contents of Port 0w are read when the SCPU reads 2140H (this is also true of Ports 1w - 3w).
3. When reset is applied, the contents of Port  $n\ r$  registers and Port  $n\ w$  registers become "00" ( $n=0-3$ ).

Table 3-3-2 Port0 - Port3 Registers

Address Seen From Sound-CPU	Address Seen From SCPU	Register Name	W/R	Function Seen From Sound-CPU Section
00F4H	2140H	Port0r Port0w	R W	Read content of Port0r register. Write to Port0w register.
00F5H	2141H	Port1r Port1w	R W	Read content of Port1r register. Write to Port1w register.
00F6H	2142H	Port2r Port2w	R W	Read content of Port2r register. Write to Port2w register.
00F7H	2143H	Port3r Port3w	R W	Read content of Port3r register. Write to Port3w register.

Figure 3-3-1 I/O Diagram



## Chapter 4. Control Register

### 4.1 THE PORT CLEAR FUNCTION BY MEANS OF THE CONTROL REGISTER.

The ports are cleared to "00" when "1" is written into the control register port clear control bits PC32 and PC10. When "0" is written in, they are not cleared.

When "1" is written into the port clear control bit PC10, both the port 0r register and the port 1r register are cleared to "00". In the same manner, when "1" is written into PC32, both the port 2r register and the port 3r register are cleared to "00".

CONTROL REGISTER

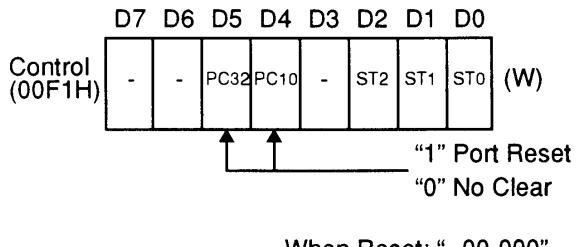


Figure 3-4-1 Port Clear

#### Note: Clear Timing

Port clear is executed during the machine cycle following that in which "1" is written into the port clear control bit.

When port clear timing conflicts with write timing to the port in question from the SNES bus, there are cases in which the contents of the register in question become indeterminate.

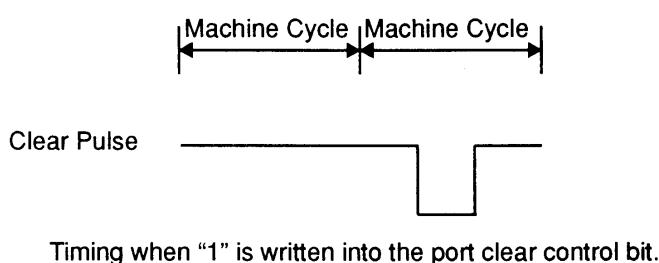
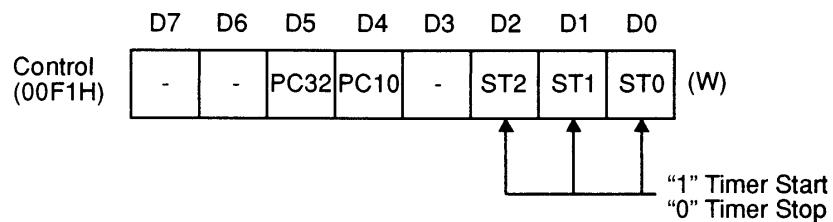


Figure 3-4-2 Clear Timing

(NCL PG 8)

## 4.2 TIMER CONTROL BY MEANS OF THE CONTROL REGISTER

CONTROL REGISTER



When Reset: "--00-000"

Figure 3-4-3 Timer Control

ST0 is the Timer T0 start/stop control bit; the timer stops with "0" and starts with "1". At this timer, it is necessary to input "1" into ST0 once it has been changed to "0".

ST1 and ST2 are respectively the start/stop control bits of timers T1 and T2. Their function is identical to that of ST0.

NOTE: In regard to the functional operation of timers, please refer to the next page.

## Chapter 5. Timers

### 5.1 FUNCTION OF TIMERS T0, T1, AND T2

The SNES sound source is provided with three timers; T0, T1, and T2.

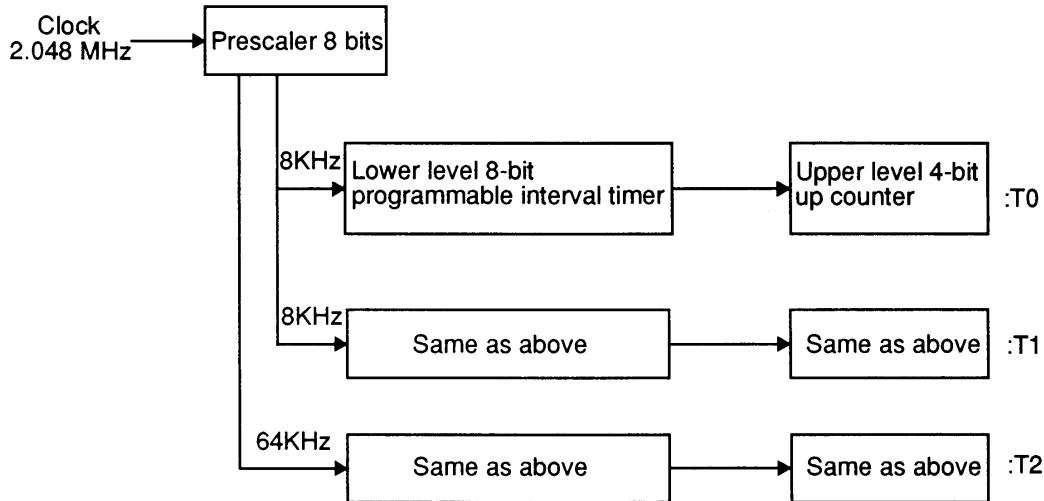


Figure 3-5-1 Timer Section

The timers T0, T1, and T2 are each composed of a lower level 8-bit programmable interval timer connected to an upper level 4-bit up counter.

The 8-bit timer is made up of an 8-bit binary up counter, comparator, timer register, and control circuit. Each of the timers; T0, T1, and T2; is independently programmable.

The clock input to timers T0 and T1 from the prescaler is 8KHz (125  $\mu$ s) and the clock input to timer T2 from the prescaler is 64KHz (15.6  $\mu$ s).

	8-bit Timer		4-bit Up Counter
	Resolution	Max. Count Value	Max. Count Value
Timer T0, T1	125 $\mu$ sec.	32 msec.	512 msec.
Timer T2	15.6 $\mu$ sec.	4 msec.	64 msec.

Table 3-5-1 Timer Function

(NCL PG 10)

## 5.2 TIMER ACTION

Since timers T0, T1, and T2 are alike in structure, an explanation of only timer T0 is provided.

The lower level 8-bit timer of timer T0 is composed principally of a binary up counter, which is incremented at each count of the clock input. When its value corresponds to the contents of the timer register, it is cleared to 00H. Simultaneously a pulse is generated to the 4-bit up counter.

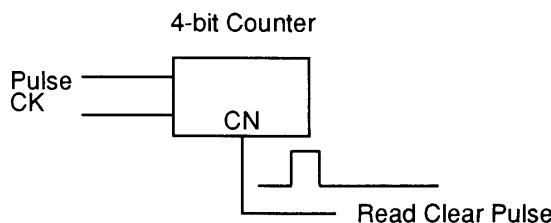
The 4-bit up counter is composed principally of a binary up counter, which increments at each input of a lower level pulse.

The action of the counter of timer T0 is controlled by the 0 bit of the control register. When bit ST0 is "0" count up is suspended. Count up commences when both upper and lower level counters are cleared by "1". Consequently, in order to clear the counters, it is necessary to set bit ST0 to "1" after having set it to "0".

Writing to the timer register is carried out while the counter is stopped. At this time the minimum write value is 00H and the maximum value is 01H. Though it is not possible to read the value of the timer register, it is possible to read the 4-bit value CN0 at any time. When the value of CN0 is read, only the 4-bit up counter section is cleared to "00".

Upper Level 4-bit Counter Timing

Figure 3-5-2 4-bit Counter



Action of timer T0 is stopped by means of the reset input (POR="L"). At the time of reset, ST0 of the control register is "0" and; CN0 and TM0 of the timer register are indeterminate.

When CN is read, the 4-bit up counter alone is cleared through IC internal timing. But the read clear pulse and the pulse to the 4-bit up counter do not conflict with each other.

Consequently, when the pulse is input to the 4-bit up counter, the value of CN will necessarily be incremented; or when the value of CN is read, CN will be cleared and become "0".

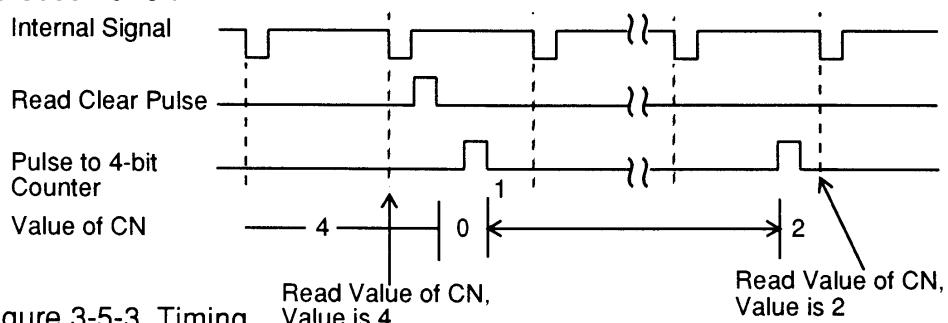
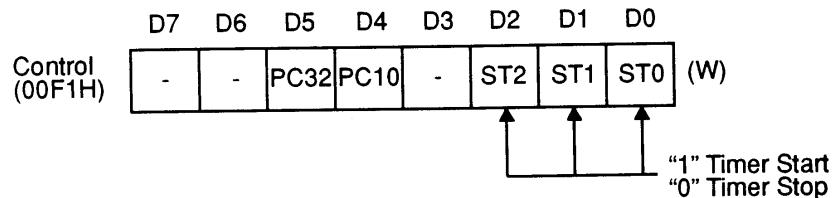


Figure 3-5-3. Timing

(NCL PG 11)

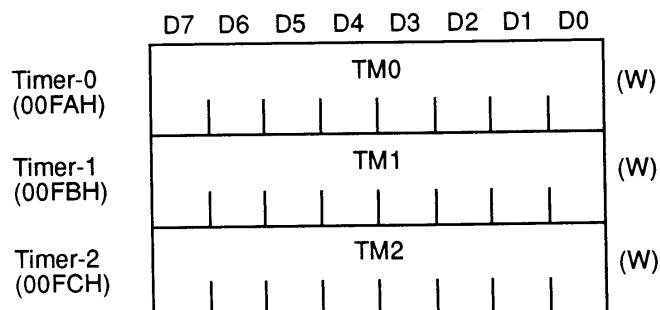
### 5.3 TIMER RELATED REGISTERS

#### Control Register

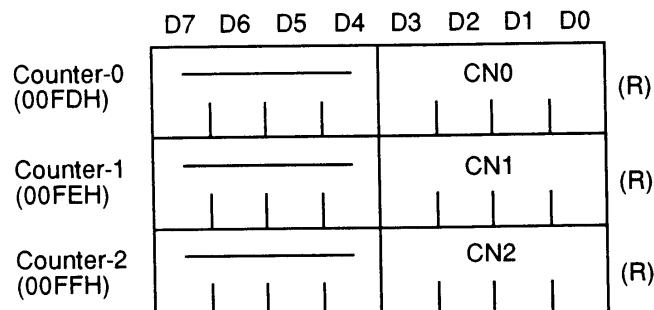


When Reset: "--00-000"

#### Timer Register



#### Counter Register



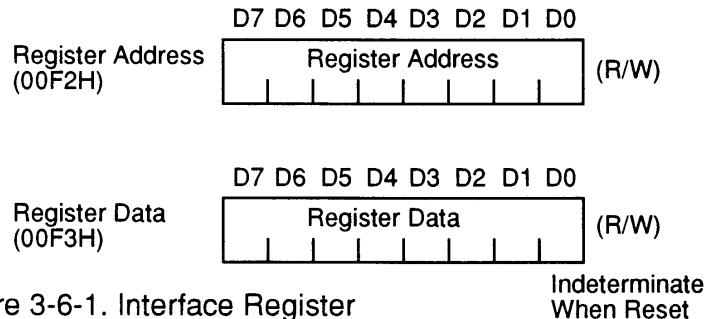
Indeterminate  
When Reset

Figure 3-5-4 Timer Related Registers

(NCL PG 12)

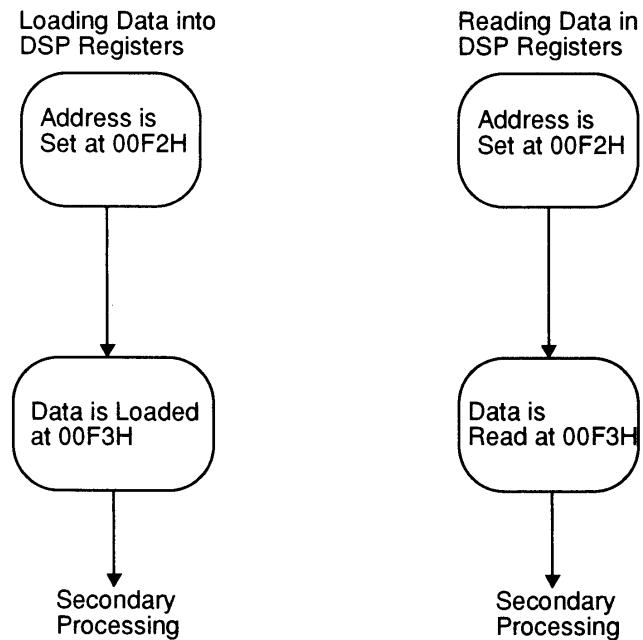
## Chapter 6. DSP Interface Register

### 6.1 Interface Register



This is the register which loads data into the registers within DSP. Values are loaded into the designated register in accordance with the path of the flow-chart below. The DSP address is written to 00F2H and data is written to 00F3H (refer to Flow A). When the contents of the register data is read, it conforms to Flow B. The address to be read is loaded into 00F2H and the contents of 00F3H are read.

Figure 3-6-2. Interface Register Flow



## Chapter 7. Register Used

### 7.1 DSP REGISTER MAP

Address	Register	Explanation of Function
00	VOL (L)	Left Channel Volume
01	VOL (R)	Right Channel Volume
02	P (L)	The total 14 bits of P(H) and P(L) express
03	P (H)	Pitch Height
04	SRCN	Designates source number from 0-255
05	ADSR (1)	Address is designated by D7=1 of ADSR(1); when
06	ADSR (2)	D7=0, Gain is operative
07	GAIN	Envelope can be freely designated by the program
08	.: ENVX	Present value of envelope which DSP rewrites at each Ts
09	.: OUTX	Value after envelope multiplication & before VOL multiplication (present wave height value)
10 ~ 19	Voice 1	
20 ~ 29	Voice 2	
30 ~ 39	Voice 3	
40 ~ 49	Voice 4	
50 ~ 59	Voice 5	
60 ~ 69	Voice 6	
70 ~ 79	Voice 7	
0C	MVOL (L)	Main Volume (L)
1C	MVOL (R)	Main Volume (R)
2C	EVOL (L)	Echo Volume (L)
3C	EVOL (R)	Echo Volume (R)
4C	KON	Key On, D0-D7 correspond to Voice0-Voice7
5C	KOF	Key Off
6C	FLG	Designated on/off of reset, mute, echo, and noise clock
7C	.: ENDX	Indicates source end block
0D	EFB	Echo Feedback
1D	---	Not Used
2D	PMON	Pitch modulation of Voice i with OUTX of Voice (i-1) as modulated wave
3D	NON	Noise on/off, D0-D7 correspond to Voice0-Voice7
4D	EON	Echo On/Off
5D	DIR	Off-set address of source directory
6D	ESA	Off-set address of echo region, Echo Start Address
7D	EDL	Echo Delay. Only lower level 4 bits active.
0F	Filter Coefficients	C0
1F		C1
2F		C2
3F		C3
4F		C4
5F		C5
6F		C6
7F		C7

.: Register written to by DSP during conditions of activity.

Table 3-7-1 DSP Register Map

(NCL PG 14)

## 7.2 REGISTER FUNCTION

### 7.2.1 Register of each voice (Addresses indicated are those of Voice 0).

#### 7.2.1.1 VOL (L), VOL (R)

	D7	D6	D5	D4	D3	D2	D1	D0
VOL (L) (00H)	sign							
VOL (R) (01H)	sign							

Each is a volume level multiplied by Lch and Rch, which is in a 2's complement form making D7 the sign bit. When a negative value is entered, phases reverse.

#### 7.2.1.2 P(L), P(H)

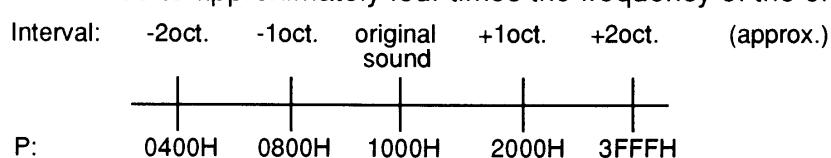
	D7	D6	D5	D4	D3	D2	D1	D0
P(H) (03H)	(0)	(0)						
P(L) (02H)								

Pitch is expressed by the total 14 bits combining six lower level bits of P(H) and eight bits of P(L). At the current time, two upper level bits of P(H) are not used. (Considered to be "0" at all times.) With  $f$  as the frequency of the reproduced sound,  $f_0$  as the frequency of the original sound (sound at the time of recording), and P as the value expressed by the lower level fourteen bits of P(H) and P(L), the following formula is performed:

$$f = f_0 \cdot \frac{P}{2^{12}}$$

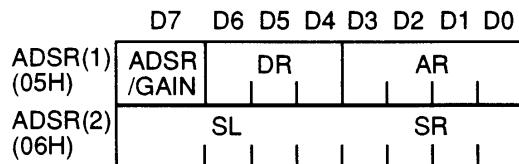
The diagram below illustrates the relationship between P and the octaval ratio of the reproduced sound and the original sound.

There are theoretically no limitations in the practical range so long as the original sound is converted **lower**. The upper range is limited to approximately four times the frequency of the original sound.



In terms of tone quality, the lower level 4 bits of P(L) should be set at "0" when possible in cases where pitch aberrations are not of concern.

### 7.2.1.3 ADSR(1), ADSR(2)

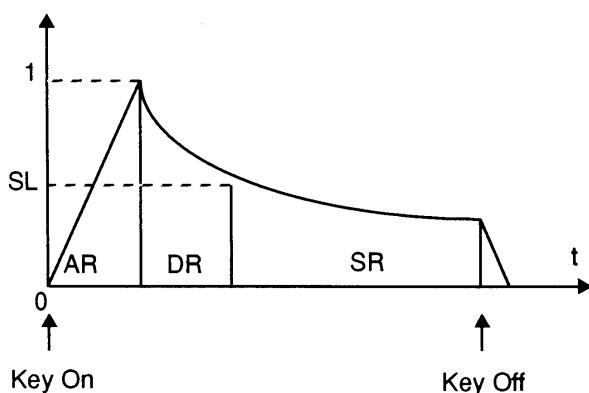


When D7 of ADSR(1) = "1", these two bytes become operable.  
(ADSR mode)

AR is added to the fixed value "1/64" and DR, SR by the fixed value "1-1/256". When in the state of "Key Off", the "click" sound is prevented by the addition of the fixed value "1/256". (GAIN mode is identical.)

Table 3-7-2 Adsr Parameters

AR	Time from 0 to 1	DR	Time from 1 to SL	SL	Ratio	SR	Time from 0 to 1
0	4.1 sec	0	1.2 sec	0	1/8	0	Infinite
1	2.6 sec	1	740 msec	1	2/8	1	38 sec
2	1.5 sec	2	440 msec	2	3/8	2	28 sec
3	1.0 sec	3	290 msec	3	4/8	3	24 sec
4	640 msec	4	180 msec	4	5/8	4	19 sec
5	380 msec	5	110 msec	5	6/8	5	14 sec
6	260 msec	6	74 msec	6	7/8	6	12 sec
7	160 msec	7	37 msec	7	1	7	9.4 sec
8	96 msec					8	7.1 sec
9	64 msec					9	5.9 sec
A	40 msec					A	4.7 sec
B	24 msec					B	3.5 sec
C	16 msec					C	2.9 sec
D	10 msec					D	2.4 sec
E	6 msec					E	1.8 sec
F	0 msec					F	1.5 sec
						10	1.2 sec
						11	880 msec
						12	740 msec
						13	590 msec
						14	440 msec
						15	370 msec
						16	290 msec
						17	220 msec
						18	180 msec
						19	150 msec
						1A	110 msec
						1B	92 msec
						1C	74 msec
						1D	55 msec
						1E	37 msec
						1F	18 msec



(NCL PG 16)

#### 7.2.1.4 GAIN

This becomes operable when D7 of ADSR(1) = 0. The following five modes are available.

	D7	D6	D5	D4	D3	D2	D1	D0
Direct Designation (07H)	0							
Increase Mode (Linear) (07H)	1	1	0					
Increase Mode (Bent Line) (07H)	1	1	1					
Decrease Mode (Linear) (07H)	1	0	0					
Decrease Mode (Exponential) (07H)	1	0	1					

∴ Direct Designation: The value of GAIN is set directly by the values of D0 ~ D6.

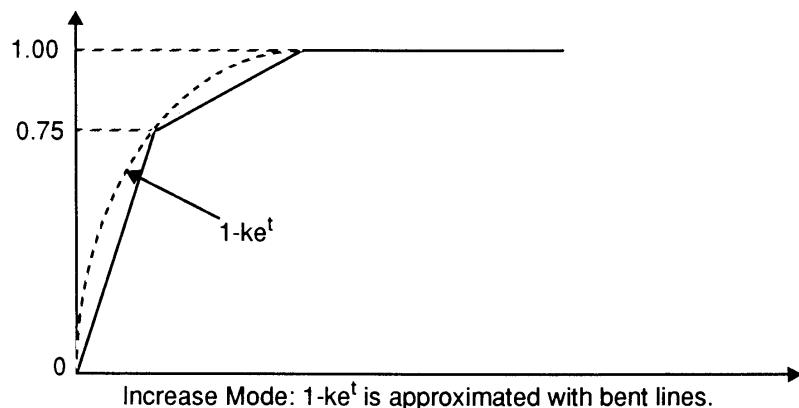
∴ Increase (Linear): Addition of the fixed value 1/64.

∴ Increase (Bent Line): Addition of the constant 1/64 up to 0.75, addition of the constant 1/256 from 0.75 to 1.

∴ Decrease (Linear): Subtraction of the fixed value 1/64.

∴ Decrease (Exponential): Multiplication by the fixed value 1-1/256.

In all cases, present envelope values (indicated by ENVX) are utilized for initial values.



Increase Mode:  $1 - e^{-t}$  is approximated with bent lines.

Figure 3-7-1 Bent Line Mode

The various parameter values are indicated on the next page.

## GAIN PARAMETERS

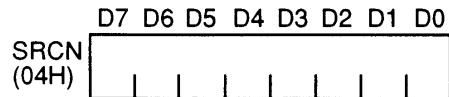
Parameter Values	Increase Mode Linear (0 → 1)	Increase Mode Bentline(0 → 1)	Decrease Mode Linear(1 → 0)	Decrease Mode Exponential (0 → 1/10)
00	Infinite	Infinite	Infinite	Infinite
01	4.1 sec	7.2 sec	4.1 sec	38 sec
02	3.1 sec	5.4 sec	3.1 sec	28 sec
03	2.6 sec	4.6 sec	2.6 sec	24 sec
04	2.0 sec	3.5 sec	2.0 sec	19 sec
05	1.5 sec	2.6 sec	1.5 sec	14 sec
06	1.3 sec	2.3 sec	1.3 sec	12 sec
07	1.0 sec	1.8 sec	1.0 sec	9.4 sec
08	770 msec	1.3 sec	770 msec	7.1 sec
09	640 msec	1.1 sec	640 msec	5.9 sec
0A	510 msec	900 msec	510 msec	4.7 sec
0B	380 msec	670 msec	380 msec	3.5 sec
0C	320 msec	560 msec	320 msec	2.9 sec
0D	260 msec	450 msec	260 msec	2.4 sec
0E	190 msec	340 msec	190 msec	1.8 sec
0F	160 msec	280 msec	160 msec	1.5 sec
10	130 msec	220 msec	130 msec	1.2 sec
11	96 msec	170 msec	96 msec	880 msec
12	80 msec	140 msec	80 msec	740 msec
13	64 msec	110 msec	64 msec	590 msec
14	48 msec	84 msec	48 msec	440 msec
15	40 msec	70 msec	40 msec	370 msec
16	32 msec	56 msec	32 msec	290 msec
17	24 msec	42 msec	24 msec	220 msec
18	20 msec	35 msec	20 msec	180 msec
19	16 msec	28 msec	16 msec	150 msec
1A	12 msec	21 msec	12 msec	110 msec
1B	10 msec	18 msec	10 msec	92 msec
1C	8 msec	14 msec	8 msec	74 msec
1D	6 msec	11 msec	6 msec	55 msec
1E	4 msec	7 msec	4 msec	37 msec
1F	2 msec	3.5 msec	2 msec	18 msec

Table 3-7-3Gain Parameters

(NCL PG 18)

### 7.2.1.5 SRCN

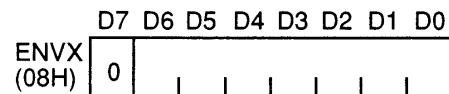
Refers to source number. It is the sequence of tone color within the hexa-file of tones produced by means of a separate tool. (0 ~ 255)



### 7.2.1.6 ENVX

The present value of the ADSR/GAIN envelope constant. The DSP section rewrites this at each Ts (31.25 µsec).

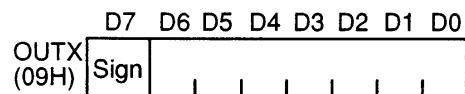
Seven bits without a sign bit. (D7 is always 0).



### 7.2.1.7 OUTX

The present value of the wave height after envelope multiplication and prior to VOL multiplication. The DSP section rewrites this at each Ts. (31.25 µsec). Its value is utilized as the modulated wave of pitch modulation.

Eight bits with a sign bit.



## 7.2.2 COMPLETE VOICE REGISTERS

### 7.2.2.1 KON, KOF

"Key on" and "Key off". D0 ~ D7 correspond to Voice 0 ~ 7. When a "1", key on or key off are active; when "0" neither is active. These two registers need not be reset. With KOF, in regard to any Voice in which a "1" is written and whether in the ADSR mode or GAIN mode, 1 to 0 decreases at the rate of 8 msec by means of the addition of the fixed value 1/256. In writing in a succession of KON and KOF, two Ts (62.5  $\mu$ sec) or more should be released. (In writing a succession of various data in less than 2 Ts, the data written may not be usable later.)

	D7	D6	D5	D4	D3	D2	D1	D0
KON (4CH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	Voice 0
KOF (5CH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	Voice 0

### 7.2.2.2 PMON

Pitch modulation is imposed on Voice *n* with OUTX of Voice(*n*-1) (*n*=1-7) as a modulated wave. When D*n*=1, it becomes modulation ON. (For example, when D1=1, a modulated tone is generated from Voice 1.) However modulation does not affect Voice 0. Therefore, the bit D0 is not active. In regard to the method of pitch modulation, when  $y_0$  is the wave height value of the modulated wave and P is the value of P(H) and P(L), then:

$$P' = P (1+y_0)$$

The value of P'', as above, is substituted for P and used as the pitch at that time.

	D7	D6	D5	D4	D3	D2	D1	D0
PMON (2DH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	X

### 7.2.2.3 NON

Noise on/off. D0 ~ 7 correspond to Voice 0 ~ 7. When on, noise is issued instead of sound source data. At this time, if sound source data of formants only is designated through the previous SRCN, then noise is generated only for the duration of the sound source data. When reproduction for random lengths of time is desired, sound source data incorporating a loop must be designated through the SRCN. In addition, even though two or more Voices may be on, the source of noise is the same.

Note: Modulation can not be imposed on this noise.

	D7	D6	D5	D4	D3	D2	D1	D0
NON (3DH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	Voice 0

(NCL PG 20)

### 7.2.2.4 EON

Echo on/off. Active “1”. D0 ~ 7 correspond to Voice 0 ~ 7.

	D7	D6	D5	D4	D3	D2	D1	D0
EON (4DH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	Voice 0

(5) FLG

	D7	D6	D5	D4	D3	D2	D1	D0
FLG (6CH)	RES	MUTE	ECEN			N C K		

RES: Soft reset is turned on when D7=1. At this time, all Voices are in a state of “Key On” suspension and Mute is turned on. It becomes a “1” with power on.

MUTE: Mute is turned on in all Voices when D6=1. This always occurs when power is first applied.

ECEN: Allows the possibility to write into external memory through Echo, when D5=0. (Echo Enable). After power on, read out data is indeterminate until initial data is written in by the CPU.

NCK: Designates the clock of the noise generator.

Table 3-7-4. Noise Generator Clock

NCK	Freq.	NCK	Freq.	NCK	Freq.	NCK	Freq.
00	0 Hz	08	83 Hz	10	500 Hz	18	3.2 KHz
01	16 Hz	09	100 Hz	11	667 Hz	19	4.0 KHz
02	21 Hz	0A	125 Hz	12	800 Hz	1A	5.3 KHz
03	25 Hz	0B	167 Hz	13	1.0 KHz	1B	6.4 KHz
04	31 Hz	0C	200 Hz	14	1.3 KHz	1C	8.0 KHz
05	42 Hz	0D	250 Hz	15	1.6 KHz	1D	10.7 KHz
06	50 Hz	0E	333 Hz	16	2.0 KHz	1E	16 KHz
07	63 Hz	0F	400 Hz	17	2.7 KHz	1F	32 KHz

It is only possible to write into these registers from the CPU section.

### 7.2.2.5 ENDX

When BRR decode of the block having the Source End flag is completed, the DSP section sets up a "1". D0 ~ 7 correspond to Voice 0 ~ 7. If there is a voice which has been keyed on, the bit corresponding to this voice is reset. In addition, when the CPU section writes into this register, all bits are reset.

	D7	D6	D5	D4	D3	D2	D1	D0
ENDX (7CH)	Voice 7	Voice 6	Voice 5	Voice 4	Voice 3	Voice 2	Voice 1	Voice 0

### 7.2.2.6 MVOL(L), MVOL(R), EVOL(L), and EVOL(R)

Refer to Main Volume (Lch, Rch) and Echo Volume (Lch, Rch). The output of this register is the sum of main volume and echo volume with a sign bit.

MVOL(Lch, Rch)	D7	D6	D5	D4	D3	D2	D1	D0
EVOL(Lch, Rch) (0CH) (1CH)	Sign							

### 7.2.2.7 ESA

Echo Start Address. Issues the off-set address of the Echo region. (ESA) x 100H becomes the lead-off address of the Echo region.

### 7.2.2.8 EDL

Echo Delay. Only the lower level four bits are used. Delay time  $\alpha$  is an interval of 16 msec. and is variable within a range of 0 ~ 240 msec. If this time is considered to be  $t$ , the necessary external memory region is  $(2t)$  Kbytes, with a maximum allowable of 30 Kbytes. However, when EDL=0, the four byte memory region of ESA - ESA+3 becomes necessary.

EDL (7DH)	D7	D6	D5	D4	D3	D2	D1	D0
	X	X	X	X				

### 7.2.2.9 EFB

Refers to Echo Feed-Back. This word consists of eight bits including a sign bit.

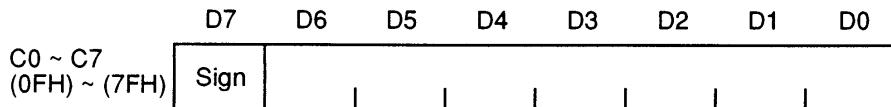
EFB (0DH)	D7	D6	D5	D4	D3	D2	D1	D0
	Sign							

### 7.2.2.10 DIR

Issues the off-set address of the source directory. (DIR) x 100Hs is the beginning address of the directory.

## 7.2.2.11 C0 ~ C7

Issues the filter coefficient. It is composed of eight bits, including a sign bit and makes up an eight tap FIR filter (identical with that of Lch and Rch).



Filter Setting Example 1: When a low pass filter is imposed on the echo sound.

Register	Numerical Value
C0	FF
C1	08
C2	17
C3	24
C4	24
C5	17
C6	08
C7	FF

Filter Setting Example 2: When the echo sound is given the same tone color as the original sound.

Register	Numerical Value
C0	7F
C1	00
C2	00
C3	00
C4	00
C5	00
C6	00
C7	00

## 7.3 SOUND SOURCE DATA (SOURCE) SPECIFICATIONS \*

Sound source data is produced according to the following specifications by means of specialized tools.

### 7.3.1 Source Directory

#### 7.3.1.1 SA(H), SA(L)

The source start address. This 16 bit address is the lead-off address of the lead-off block.

#### 7.3.1.2 LSA(H), LSA(L)

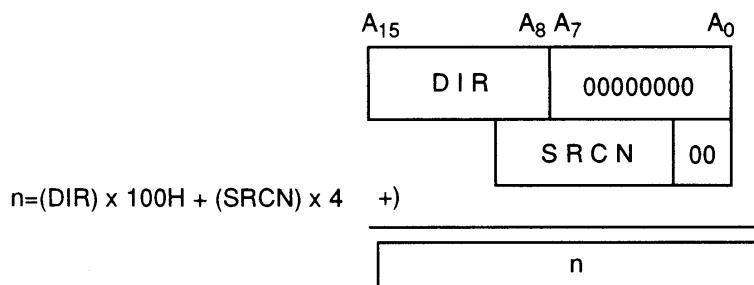
Source loop start address. This 16 bit address is the lead-off address of the loop start block.

Table 3-7-5. Source Directory

Memory Address	Directory
n+0	SA(L)
n+1	SA(H)
n+2	LSA(L)
n+3	LSA(H)

SA: Source Start Address

LSA: Source Loop Start Address



\* Sound source data used in the SNES is called "Source".

### 7.3.2 SOURCE DATA

#### 7.3.2.1 BLOCK FORMAT

The sound, sampled at 32KHz, undergoes BRR (bit rate reduction) processing and the data is condensed from 16 bits to 4 bits. The four-bit data is arranged into sixteen portions and, together with the RF register, is formed into one block of nine bytes.

	D7	D6	D5	D4	D3	D2	D1	D0
RF	BRR Data					Loop on/off	END	
D <sub>A0</sub>	D <sub>A0H</sub>					D <sub>A0L</sub>		
D <sub>B0</sub>	D <sub>B0H</sub>					D <sub>B0L</sub>		
D <sub>A1</sub>	D <sub>A1H</sub>					D <sub>A1L</sub>		
D <sub>B1</sub>	D <sub>B1H</sub>					D <sub>B1L</sub>		
D <sub>A2</sub>	D <sub>A2H</sub>					D <sub>A2L</sub>		
D <sub>B2</sub>	D <sub>B2H</sub>					D <sub>B2L</sub>		
D <sub>A3</sub>	D <sub>A3H</sub>					D <sub>A3L</sub>		
D <sub>B3</sub>	D <sub>B3H</sub>					D <sub>B3L</sub>		

Table 3-7-6 Source Data Block Format

#### 7.3.2.2 RF

Bits D7 ~ D2 is composed of data relating to BRR. When D1=1, it indicated that it is a source having a loop and when D1=0, it indicates that the block is the block with the final data.

## ***Chapter 8. CPU Organization***

A Sony SPC700 series is used in the CPU core of the SNES Sound Source. It is possible to access and address space of 64 Kbytes in the SPC series CPU. Address classification of the memory space is made according to purpose; addresses  $0000_H \sim 00FF_H$  are called page 0 and addresses  $0100_H \sim 01FF_H$  are called page 1. In regard to the data in this region; when direct page designation is carried out by the direct page flag (P) within the program status word, it is possible to carry out data processing in wide-ranging addressing modes with a small number of machine cycles.

Within the CPU there are the universal registers A, X, and Y, program status word (PSW) of the various flags, program counter (PC), and stack pointer (SP).

The A register is operable by the greatest number of commands and becomes an 8-bit operation accumulator. When 16-bit operations are carried out, it becomes paired with the Y register and becomes the lower level 8-bit register of the 16-bit accumulator. The X and Y registers, in addition to their function as universal registers, are used in various operations. These include; functions as index register of various index addressing modes, as dual address command source, destination address register, etc.

In the command set there are single address commands which carry out arithmetic and logical operations centered in the A register and dual address commands which can designate random addresses within the direct page as source addresses and destination addresses.

In regard to bit processing diversified by control purpose, Boolean bit operation commands are applicable to the 8 Kbyte wide range of data of addresses  $0000_H \sim 1FFF_H$ . Moreover, in regard to the bits within the direct page, set, reset and bit conditional relative jump can be utilized. In regard to the data within the total space of the 64 Kbytes; commands of multiple bit test and set, test, and reset are provided. For the purpose of data which must be systematized or in order to carry out data processing rapidly, it is possible to operate 16-bit data with a single command. Addition, subtraction, comparison, and transference are possible between two bytes of continuous 16-bit data within the direct page and the paired Y register and A register. In addition, increment and decrement of continuous 16-bit data within the direct page are possible.

There are multiplication and division commands for the purpose of rapid data processing and processing of data in a variety of forms. Multiplication is 8-bits x 8-bits with no sign and is carried out with the multiplicand stored in the Y register and the multiplier stored in the A register; the result is entered into the (Y,A) 16-bit accumulator. Division is 16 bits/8 bits with no sign and is carried out with the dividend stored in the (Y,A) 16 bit accumulator and the divisor stored in the X register. The resulting quotient is entered into the A register and the remainder into the Y register.

When processing decimal data, there are decimal addition/subtraction correcting commands in regard to the results of both addition and subtraction.

In regard to branched commands, there are relative branched commands according to the conditions of the various status flags, according to the conditions of set or reset of random bits within the direct page, etc. In addition, in regard to looped branched commands, there are comparison branched commands and subtraction branched commands. For these there are two types of addressing modes.

In regard to subroutine call commands, there are subroutine address direct designation, Three-byte call commands within the 64 Kbytes, Two-byte call commands for calling specific areas, and One-byte call commands using call tables. It is possible to improve byte efficiency through proper usage in response to the frequency of subroutine use.

## 8.1 CPU REGISTERS

Within the CPU are the registers necessary for the execution of various commands. These are the A register (also functions as an 8-bit accumulator), X register, Y register (8-bit universal register which can also be used as an index register), PSW (program status word), SP (stack pointer), etc. These are all 8-bit registers, but the PC (program counter) is made up of 16 bits.

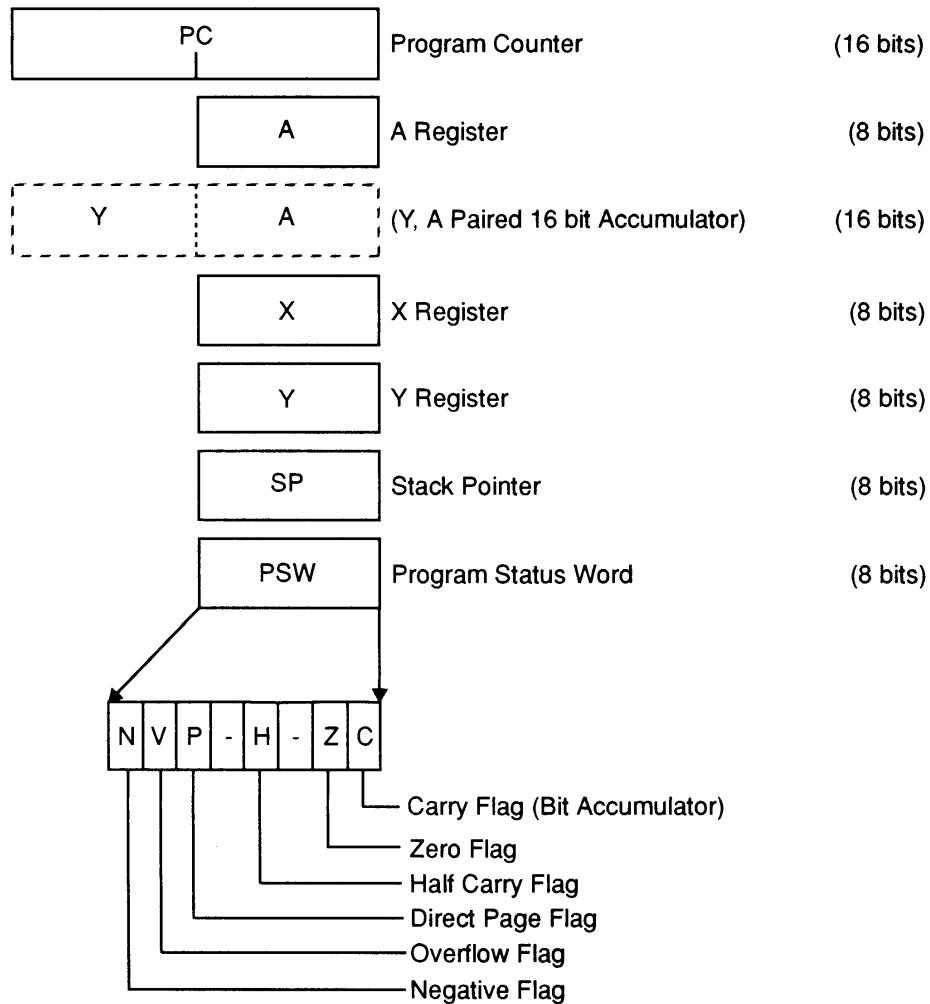


Figure 3-8-1 CPU Registers

### 8.1.1 A REGISTER

This register is used as an 8-bit accumulator. At times of 16-bit operation commands, it becomes the register which contains low byte data in the 16-bit accumulator, made up of this paired with the Y register. When operation commands are issued, it becomes the multiplier register and low byte data of the product is entered. When division commands are issued, paired with the Y register, it formulates the dividend and the resulting quotient is entered.

### 8.1.2 X REGISTER

In addition to its role as a universal data register, it also functions as an index register when index addressing is being carried out. In addition, it is used as a two-address command destination address register and X register indirect address register. In division commands, it becomes the divisor register.

### 8.1.3 Y REGISTER

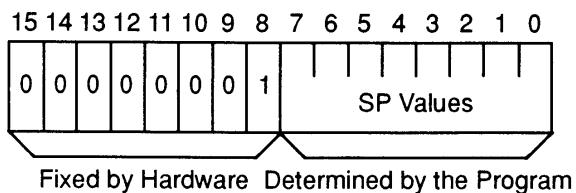
In addition to its role as a universal register, it functions as an index register when index addressing is being carried out. In addition, it is used as a two address command source address register. When carrying out 16-bit operation commands, it becomes the register which contains the high byte data of the 16-bit accumulator, which is made up of the pairing of this register with the A register. When multiplication commands are being carried out, it becomes the dividend register and the product high byte data is entered. When carrying out division commands, paired with the A register it formulates the dividend and the resulting remainder is entered.

### 8.1.4 PROGRAM COUNTER

The program counter is made up of 16 bits and has an address region of 64 Kbytes. The upper level 8 bits are called PCH and the lower level 8 bits are called PCL. Normally, it will contain the address to be executed during the next machine cycle and will be incremented by only the number of bytes necessary for the command to be fetched. When there is a branching command in the midst of the program, the address of the branch destination will be stored in the program counter. When there is a reset (POR) input, reset vectors which are in addresses  $\text{FFFF}_H$  and  $\text{FFFE}_H$  enter respectively PCH and PCL for branching to take place.

### 8.1.5 STACK POINTER

The stack pointer is used to send data to the RAM or to recover data from the RAM when the subroutine call commands push (PUSH), pop (POP), or return (RET) are to be carried out. The address region indicated by the stack pointer is within page 1 (addresses  $0100_H \sim 01FF_H$ ).



When sending data to the RAM, the stack pointer decreases by one after sending data (post decrement) and increases by one prior to restoring data (pre-increment). The diversified activities of the stack pointer are summarized below.

\*SUB-ROUTINE CALLS

Stack Address	Activity	SP Value After Sending
SP	Sending to PCH	SP-1
SP-1	Sending to PCL	SP-2

\*RESTORING FROM SUB-ROUTINE

Stack Address	Activity	SP Value After Sending
SP	Restore to PCH	SP+1
SP+1	Restore to PCL	SP+2

To send the contents of the A register, X register, Y register, or PSW (program status word) to and from the stack, the commands PUSH and POP can be used.

\*PUSH A (X, Y, PSW)

Stack Address	Activity	SP Value After Sending
SP	Sending of A (X, Y, PSW)	SP-1

\*POP A (X, Y, PSW)

Stack Address	Activity	SP Value After Sending
SP	Restore A (X, Y, PSW)	SP+1

### 8.1.6 PROGRAM STATUS WORD (PSW)

The program status word is made up of the various flags which are set and reset according to the results of the execution of 8-bit register commands and the various flags which determine the activities of the CPU. When reset it becomes "000-0-00".

7	6	5	4	3	2	1	0
N	V	P	-	H	-	Z	C

◊ Carry Flag (C)

After operation execution, this flag is set when there has been a carry from the uppermost bit of the arithmetic logic unit (ALU) or when there has been no borrow. It is also altered with shift or rotate commands. It acts as bit accumulator for Boolean bit operation commands. It is set with the SETC command and reset with the CLRC command. The carry flag inverts with the NOTC command.

◊ Zero Flag (Z)

After operation execution, this flag is set when the result is zero and reset when the result is not zero. Even with 16-bit operation commands, zero detection is carried out. It is possible to carry out tests with conditional branching commands.

◊ Half Carry Flag (H)

After operation execution, this flag is set when there has been a carry from bit 3 of the ALU to bit 4 or when there has not been any borrow. There is no command to set the half carry flag however, it is reset by means of the CLRV command. Whenever the half carry flag is set, the overflow flag is also set.

◊ Direct Page Flag (P)

This is the flag which designates the direct page to which many addressing modes are applicable, such as direct page addressing, etc. When "0", the direct page becomes the addresses of the region  $0000_H \sim 00FF_H$  and when "1", it becomes the addresses of the region  $0100_H \sim 01FF_H$ . It is set by the SET P command and reset by the CLR P command.

◊ Overflow Flag (V)

After arithmetic operation execution, this flag is set when overflow or underflow has been produced. When this occurs the H flag is also set. It is possible to carry out tests with conditional branching commands.

◊ Negative Flag (N)

After operation execution, this flag is set when the value of the result of MSB is "1" and reset when its value is "0". It is possible to carry out tests with conditional branching commands.

## 8.2 MEMORY SPACE

It is possible for the Sound-CPU to address 64 Kbytes of memory. Memory space is divided up according to purpose. From address 0000H, 512 bytes are divided into two pages of 256 byte units called page zero and page one. It is possible to access data within these regions by means of numerous addressing modes, such as direct page addressing, etc. Page one is taken up by the stack.

### 8.2.1 Direct Pages (Page Zero, Page One)

By means of setting or resetting the Direct Page flag (P) within the program status word, it is possible to designate whether page zero or page one is to be made the direct page. It is set up such that the data within this page can be treated with fewer bytes, at a higher speed, and with more numerous types of commands and addressing modes.

#### 8.2.1.1 Stack Area

The stack region is established in the RAM region within page one. The uppermost byte of the stack address is fixed at 01. The lowermost byte of the stack address must be given its initial setting by the program.

### 8.2.2 Uppermost Page (Internal ROM Region)

A mask ROM is installed within the Sound-CPU from FFC0H - FFFFF. There is a program in it which transmits data from the ROM cassette to the 512 Kbit RAM through the SNES CPU. This region is used by means of reset.

### 8.2.3 Area of Applicable Bit Operation Commands

#### 8.2.3.1 SET1, CLR1

The commands SET1 (set memory bit) and CLR1 (clear memory bit) are applicable to one-bit data with the direct page.

#### 8.2.3.2 TSET1, TCLR1

The commands TSET1 (test and set bit) and TCLR1 (test and clear bit) are applicable to the total 64 Kbyte region.

#### 8.2.3.3 Boolean Operation Commands

The Boolean operation commands (AND1, OR1, EOR1, MOV1, NOT1) are applicable to the 8 Kbyte region of  $0000_H \sim 1FFF_H$ .

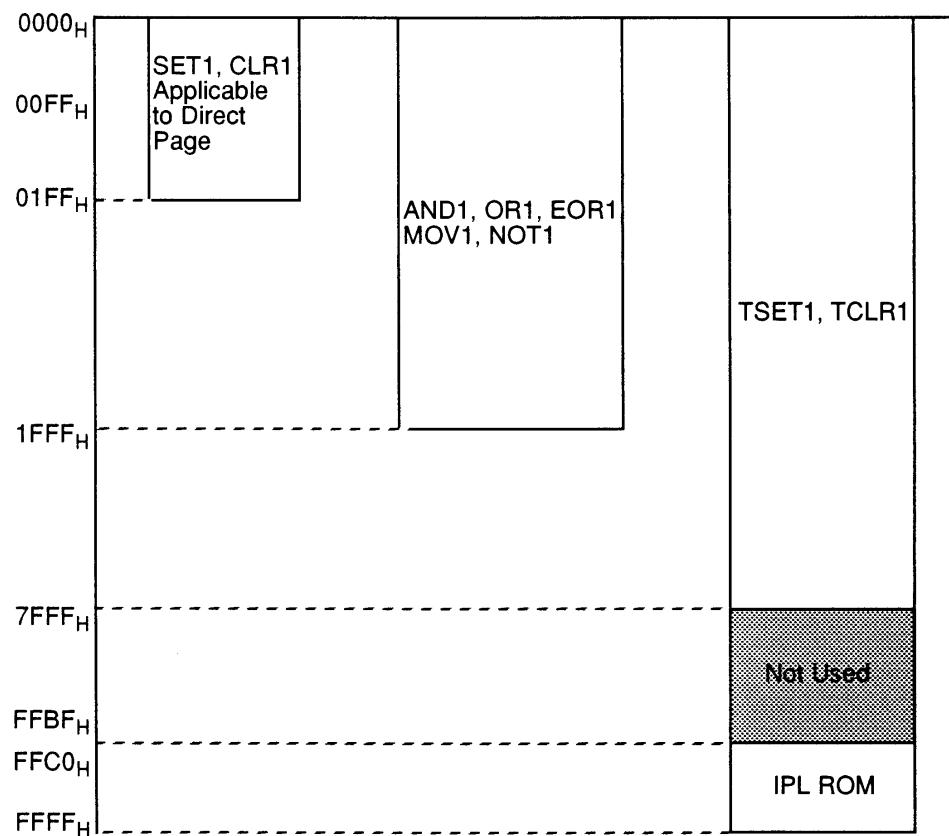


Figure 3-8-2 Boolean Bit Operation Commands

### 8.2.4 Direct Page Addressing

Since all of the addressing modes indicated in Figure 2-7-3 are applicable to the data of the direct page (P=0: addresses  $0000_H \sim 00FF_H$ , P=1: addresses  $0100_H \sim 01FF_H$ ) designated by the direct page (P) flag, it is possible to manipulate the data in various ways. In addition, byte efficiency improves due to the fact that direct address designation is possible by one-byte data within the command words. Since effective command cycles also decrease, data can be accessed more rapidly.

Symbol	Addressing	# of Bytes	Effective Address Region		
			$0000_H \sim 01FF_H$	$\sim 1FFF_H$	$\sim 1FFF_H$
dp	Direct Page	2			
dp+X	X-Indexed Direct Page	2			
dp+Y	Y-Indexed Direct Page	2			
(X)	Indirect	1			
(X)+	Indirect Auto-Increment	1			
dp. dp	Direct Page to D.P.	3			
(X),(Y)	Indirect Page to I.P.	1			
dp,#imm	Immediate Data to D.P.	3			
dp.bit	Direct Page Bit	2			
dp.bit,rel	Direct Page Bit Relative	3			
mem.bit	Absolute Boolean Bit	3			
!abs	Absolute	3			
!abs+X	X-Indexed Absolute	3			
!abs+y	Y-Indexed Absolute	3			
[DP+X]	X-Indexed Indirect	2			
[DP+Y]	Indirect Y-Indexed Indirect	2			

Figure 3-8-3 Memory Access Addressing Effective Address

(NCL PG 33)

## Chapter 9. Sound Programming Cautions

### 9.1 CAUTION #1

When layering sound on several tracks (for example, when layering sound effects on back-ground music), make sure an overflow does not occur due to the additional output. Eight-track sound is ultimately transmitted as one signal, which is limited by the maximum value for the DAC. Distortion noise is created when the signal exceeds this limit.

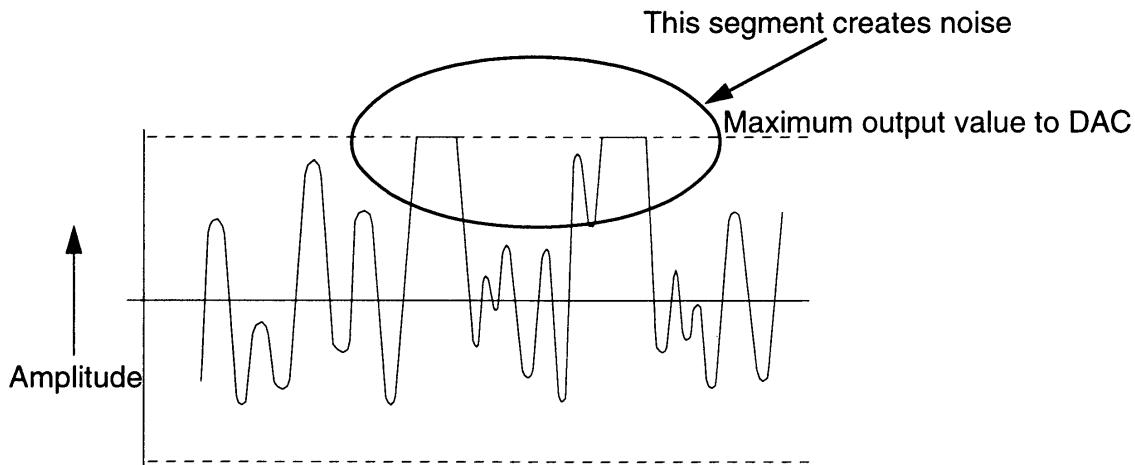


Figure 3-9-1 Wave-form Overflow

### 9.2 CAUTION #2

The following precautions should be observed when making the initial selections for a sound driver echo function.

1. The FLG's ECEN should not be turned "on" immediately after the EDL and ESA registers have been assigned a number. Otherwise, the RAM area used by the program or other area could be damaged. Either of the following guidelines can be used to determine the appropriate wait period after setting the EDL and ESA registers.
  - a) Wait 240 ms.
  - b) Read the EDL value ( $\alpha$ ) before writing to it, and calculate the wait period based on the following formula.

$$\alpha \times 16 \text{ (ms)}$$

In addition, the EVOL should be set high only after (the EDL value)  $\times$  16 ms or greater. (The read data is undefined until the DSP begins writing data, and could generate noise.)

2. Turn both the ECEN and EVOL "off" when the echo function is not in use. Data will be read and output unless the EVOL is 0.

### 9.3 CAUTION #3 (ECHO OPERATIONS)

This caution describes the procedure to be followed when writing echo data to the appropriate RAM area.

- ESA (Echo Start Address - 6DH): Initial address for the echo start area.
- EDL (Echo Delay - 7DH): Determines the number of addresses in the echo area begining from the initial address.

#### 9.3.1 PROCEDURE

An internal counter exists for the echo data, which is written sequentially. This counter is called the "echo counter". The EDL determines the maximum value of the echo counter. When the echo counter reaches (the ESA value x 80H), the echo counter is set to 00. Echo data is written two bytes at a time (4 bytes for the left and right) every 31.25 s. A delay of 16 ms occurs using a RAM address area of 80H.

The echo counter is 15 bits, from 000H ~ 7FFH. The following formula is used to determine the RAM address to which the echo data is written.

$$(ESA \text{ value} \times 100H) + (\text{echo counter value}) = (\text{echo write address})$$

D15	D8 D7	D0
ESA	0 0 0 0 0 0 0 0	
+ Echo counter value		
Echo write conditions		

However, changes in the echo counter value do not immediately follow changes in the ESA value using the above formula. Therefore, unanticipated problems, such as data loss, could occur. The relationship between the echo counter and EDL value can be explained as follows.

It was mentioned above that the echo counter is set to 00 when it reaches (the ESA value x 80H). However, the ESA value mentioned here is not the value of ESA at that time, but at the time when the echo counter was previously set to 00. Even when the ESA value is changed, the counter continues counting until it reaches the previously set ESA value, wherein it is set to 00. Then, the last-specified ESA value and echo counter value are compared. (This is to prevent the echo counter from incrementing until the maximum value is reached, when a small value is assigned to ESA). For these reasons, the echo write address will be within the specified range if the programmer waits for the period of time specified below when re-writing the EDL value.

wait time = (the EDL value prior to rewrite) x 16 ms

To insure that the echo data is written to the echo area, wait for a period equal to the last-specified EDL value x 16 ms.

#### **9.4 CAUTION #4**

Always select appropriate values for the echo parameters. Inappropriate values could lead to loss of data in critical RAM areas or noise generation. Reverberation may occur when the echo feedback value is too large.

#### **9.5 CAUTION #5**

It is extremely important to follow the recommended procedure (Caution #3, above) when setting the initial echo values and modifying the echo parameters. The RAM used as the echo buffer is also used for the program, wave form data, and sound driver. If an echo is started before the echo parameter initialization is established, critical data may be overwritten in the RAM area.

#### **9.6 CAUTION #6**

Do not use an excessive sound data compression ratio. An excessive compression ratio results in distorted sound output.

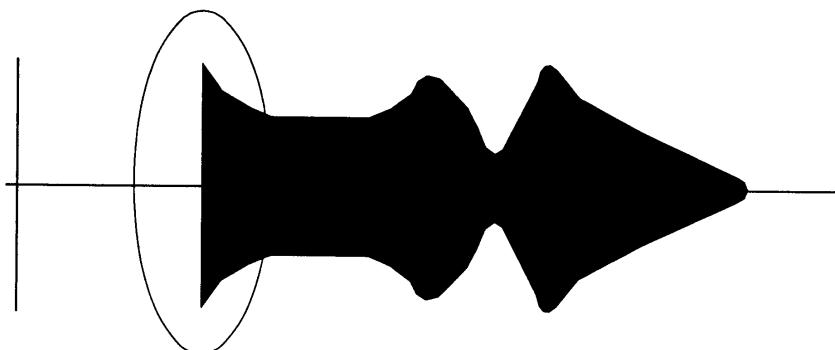
#### **9.7 CAUTION #7**

When performing sound checks, the monaural sound output should also be checked. Sound data created for stereo output may not be produced as desired when played on a monaural output device. Super NES monaural sound is generated by adding stereo sound output in the circuit. When, for example, a phase effect is created in stereo by setting negative values in the volume register, the sound volume may be altered when the sound is combined to generate monaural output.

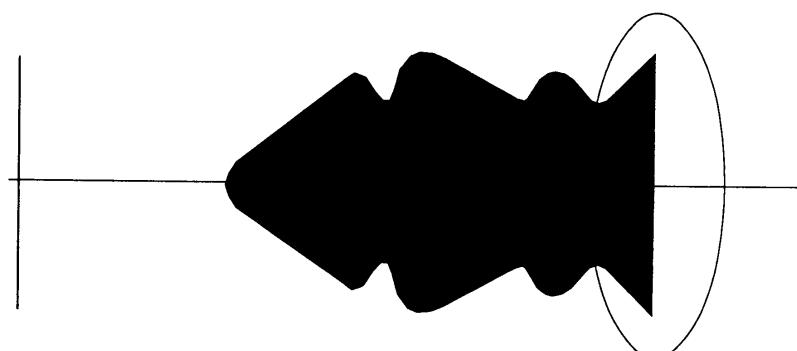
## 9.8 CAUTION #8

Sampled data should not have any discontinuity. A crackling noise is produced by discontinuous samples. The following are examples of discontinuity in the sampled data.

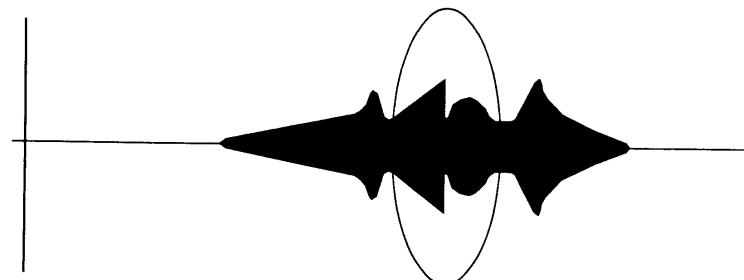
- The sampled data does not begin at 0.



- The sampled data does not end at 0.



- A discontinuity occurs in the middle of the sampled data.



## 9.9 CAUTION #9

When transferring data between the Super NES CPU and the APU using the IPL loader, a hang-up can occur if the program is interrupted.

When the Super NES CPU sends the termination code, the Sound CPU sends a code to the Super NES CPU to indicate that it has received data. The Sound CPU erases this code after 300-400  $\mu$ sec.

If an interrupt occurs after sending the termination code, for a period which is greater than 300-400 micro-seconds, the status code from the Sound CPU will be erased before it can be read by the Super NES CPU. The hang-up will occur because the Super NES CPU will wait indefinitely for the Sound CPU to indicate that it has received data.

Two possible options are available to prevent this from occurring.

- Modify the transfer routine run on the Super NES CPU side.
- Inhibit interrupts during transfer.

These options are demonstrated below.

### 9.7.1 MODIFIED TRANSFER ROUTINE

Add two lines as shown to the routine.

```

.
.
.
adc  #07fh          ; original code
pla
sta  !APU_port 0   ; original code
cpx  #1             ; solution #1
bcc  boot_ret       ; solution #1
boot_wait3          cmp  !APU_port0    ; original code
                    bne  boot_wait3  ; original code
                    bvs
boot_ret            plp
                    rts
                    end

```

### 9.7.2 INHIBITING INTERRUPTS

Inhibit any interrupt from the time the termination code is sent until the Sound CPU sends an acknowledgement. This is demonstrated by the highlighted code below.

```

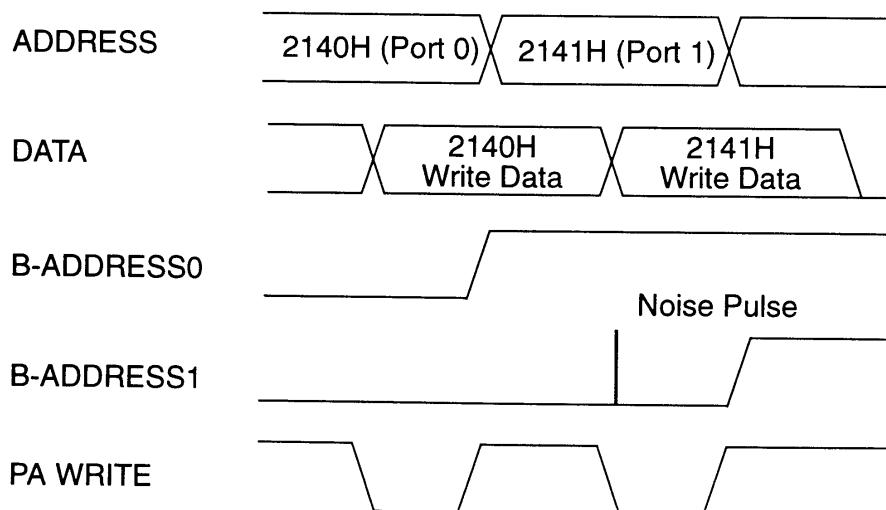
.
.
.
boot_wait3    adc #07fh          ; original code
               pla              ; original code
               sta !APU_port 0      ; no interrupt
               cmp !APU_port0       ; no interrupt
               bne boot_wait3       ; no interrupt
               bvs
               pop
               rts
               end

```

## 9.10 CAUTION #10 - DATA TRANSFER

When data is written to Port 0 <2140H> and Port 1 <2141H> in the 16 bit mode, during data transfer from the Super NES APU, the value of Port 3 <2143H> may, inadvertently, be changed. Therefore, the 8 bit mode should be used when writing data to these ports.

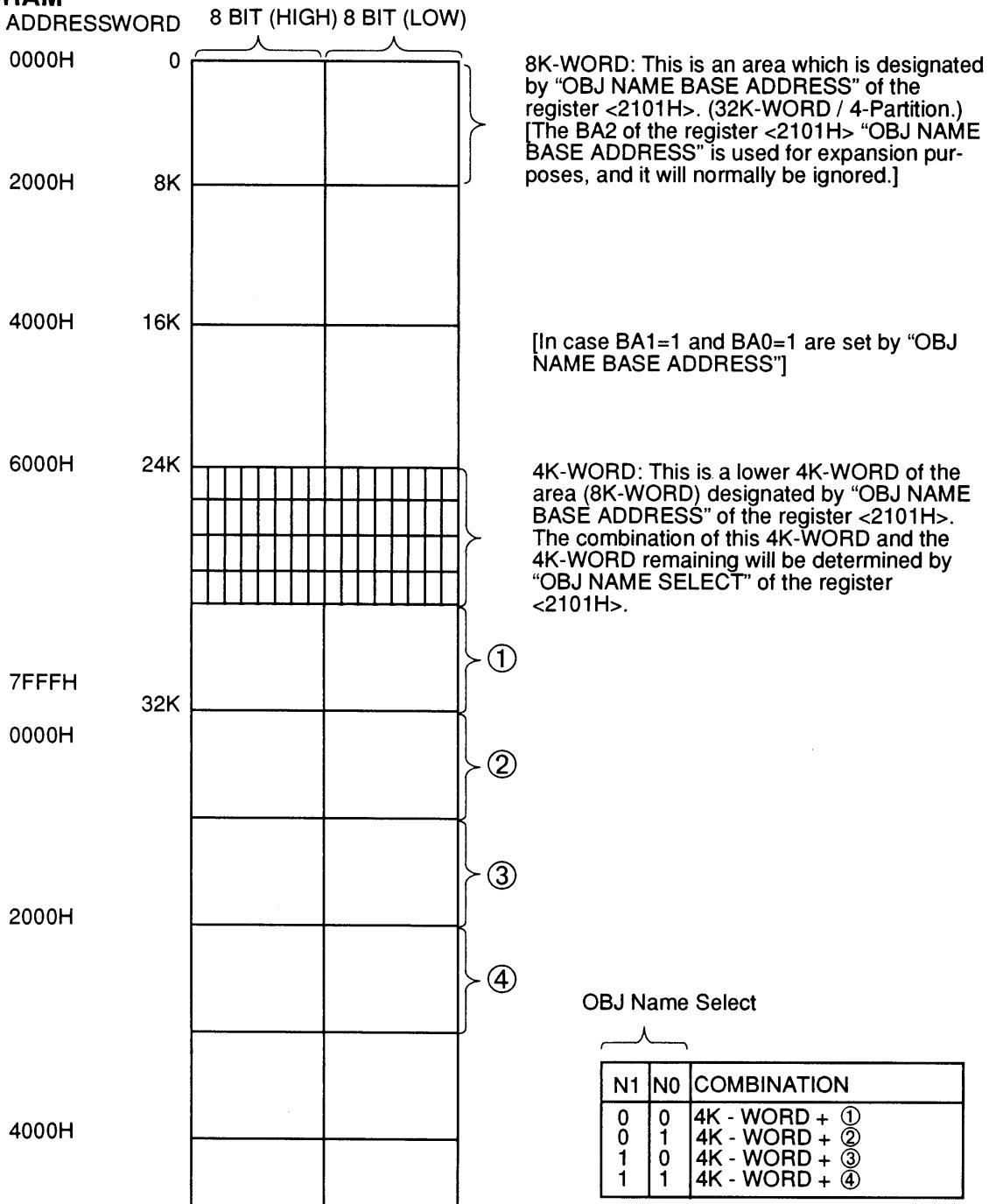
This occurs because multiple ROMs installed on the game pak PCB can increase load capacity of the data bus and, when combined with a drastic fluctuation of CPU data output, cause noise in the data being written. An example is provided below.



In the example on the previous page, if data is written to 2140H (Port 0) and 2141H (Port 1) in the 16 bit mode, noise pulses may occur at B-Address 1 due to noise which occurs when all CPU data simultaneously changes from high to low. This depends upon the type of CPU data. When data is written to 2141H (Port 1), B-Address1 becomes "1". In other words, the same data is written to 2143H (Port 3) due to this pulse.

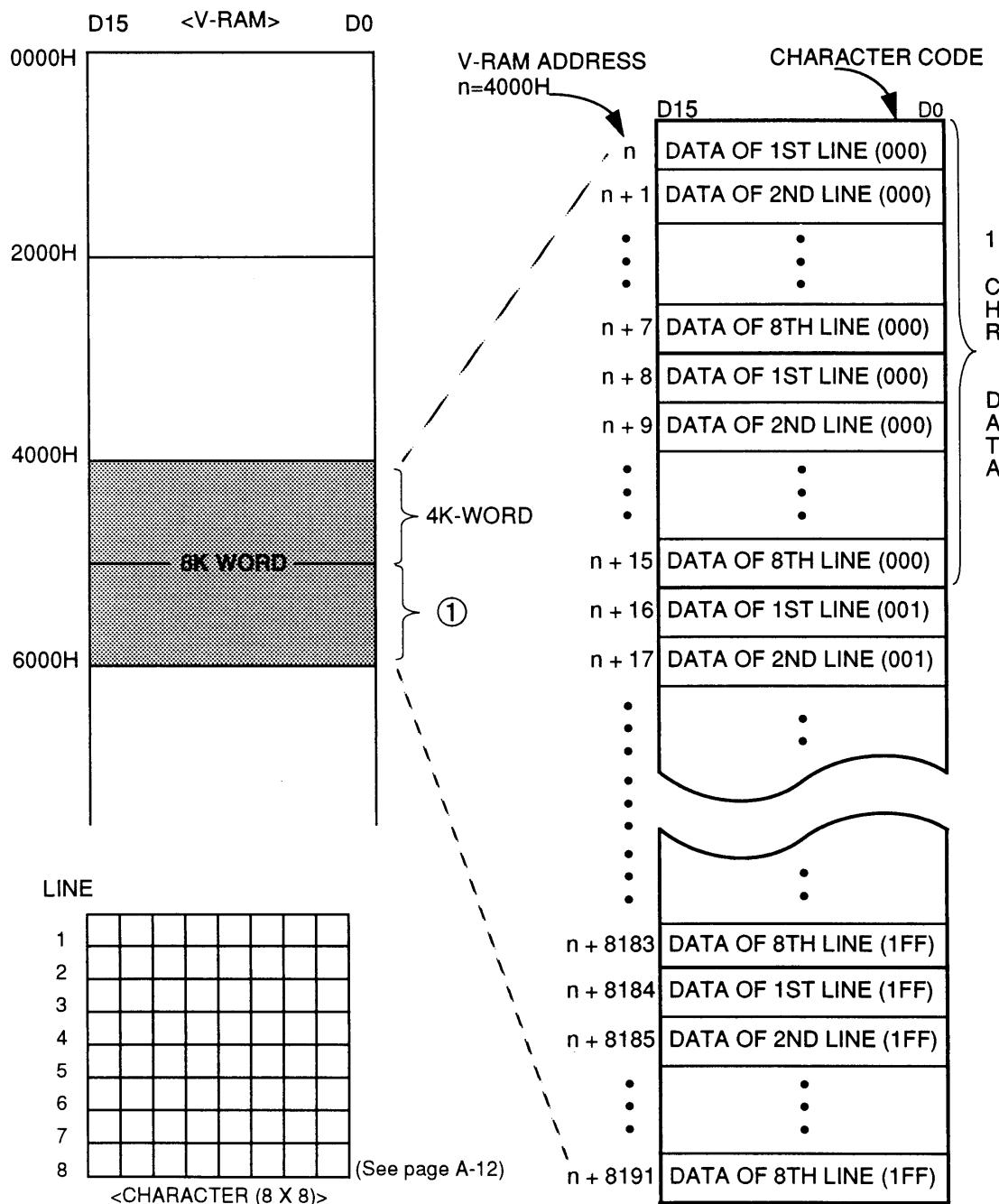
## Appendix A. PPU Registers

### V-RAM

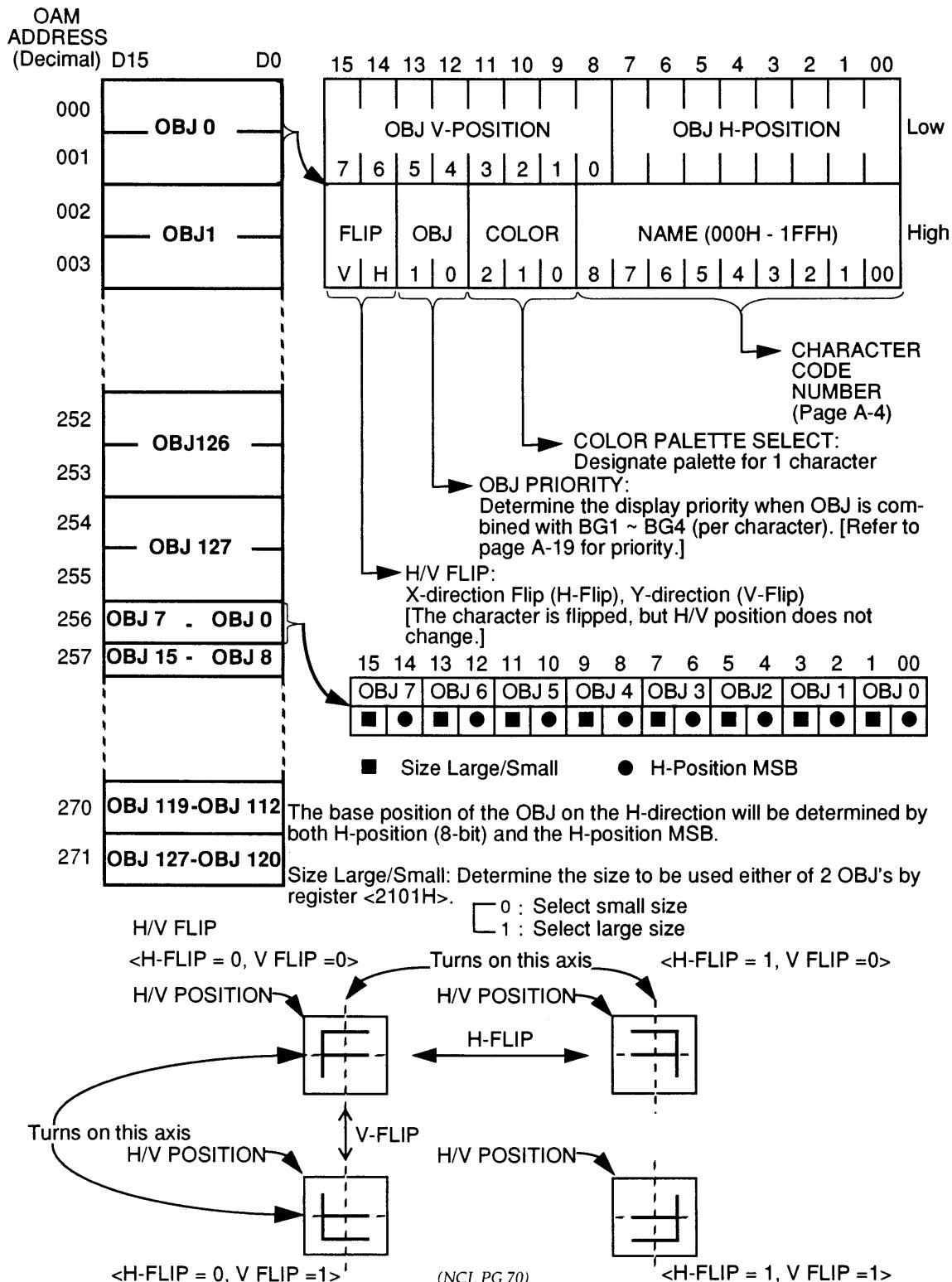


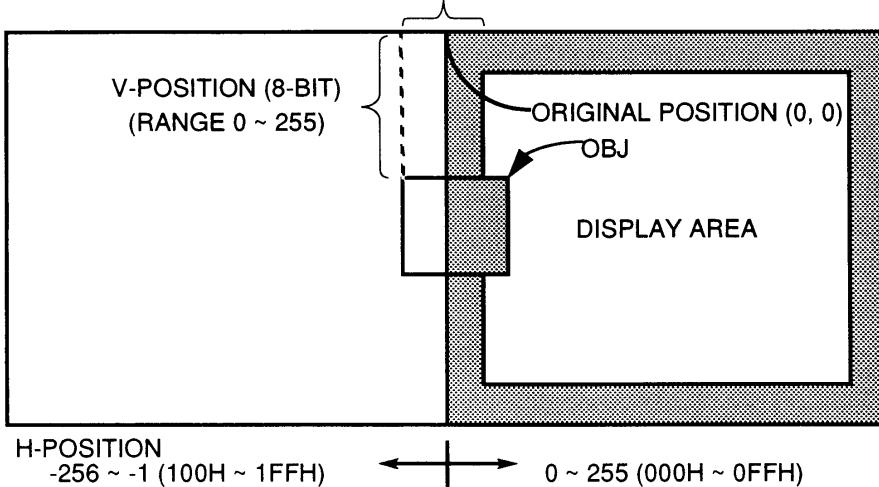
## OBJECT DATA TO BE STORED

4 BIT CONSTRUCTION [8 x 8 x 4 Bit (16 WORD) / CHARACTER] (Refer to page A-12)  
 8 x 8 (Character Size) x 4 (Bit Construction) x 512 (Number of character)  $\longrightarrow$  16K-BYTE  
 [In case BA1=1 and BA0=0 are set by "OBJ NAME BASE ADDRESS" and also N1=0 and N0=0 are set by "OBJ NAME SELECT"]

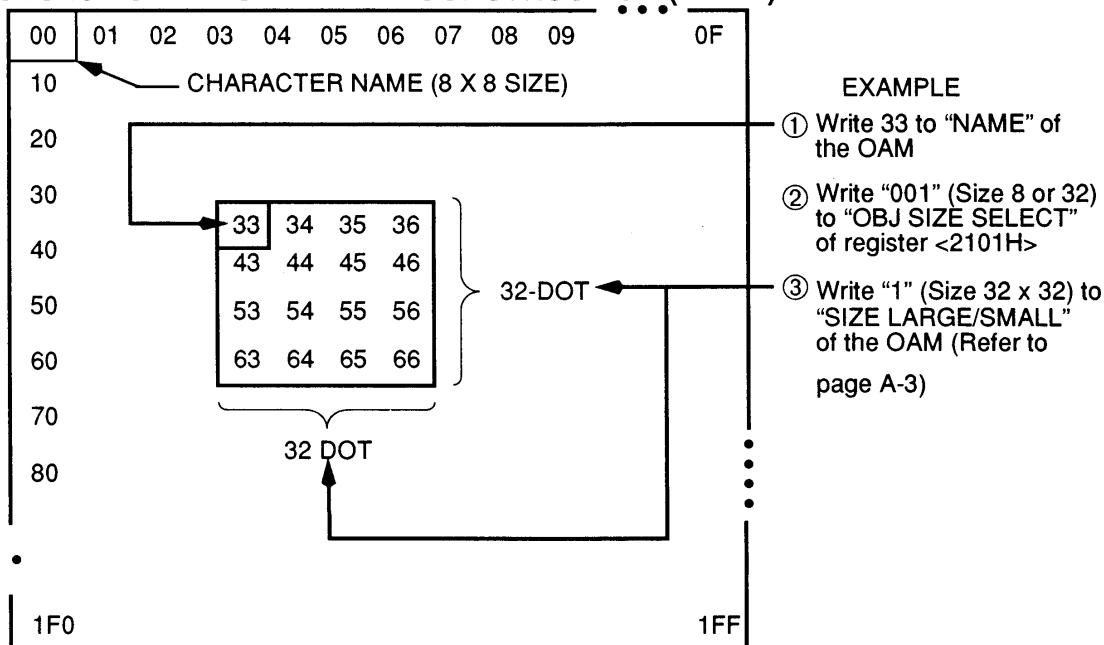


(NCL PG 69)

**OBJECT DATA**

**OBJECT DISPLAY**      H-POSITION (9-BIT) (RANGE -256 ~ 255)

- (NOTE-1) The H-position is a complementary expression of 2 (9-bit).
- (NOTE-2) The coordinate of the OBJ displayed is shifted down compared to the coordinate of the BG displayed. [Interlace: 2-dot / Non-Interlace: 1-dot] (See page A-10.)
- (NOTE-3) "100H" is basically prohibited to use for 9-bit of the H-Position. (If it is used, it must be counted as OBJ quantity displayed even if it is not displayed on the screen.)

**OBJECT CHARACTER DATA CONSTRUCTION (VRAM)**

In case the character code is 000 through 0FF, the V-RAM address per character data (16-word) will be " $n$  (Name Base Address) +  $N$  (Name)  $\times$  16 ~  $n$  +  $N \times 16 + 15$ ." If the character code is 100 through 1FF, it will be " $n + Ns$  (Name Select)  $\times$  4K +  $N \times 16 ~ n + Ns \times 4K + N \times 16 + 15$ ."

(NCL PG 71)

**OBJECT**

# OF CELLS DISPLAYED		1 2 8			
CELL SIZE		8X8	16X16	32X32	64X64
# OF LINES DISPLAYED		32-pcs (converted to 8x8 size)			
# OF CELL-COLOR		1 6			
# OF PALETTE		8			
# OF COLOR ON SCREEN		1 2 8			
ATTRIBUTE		H-FLIP, V-FLIP FUNCTION DISPLAY PRIORITY (Select priority against BG)			

**BG**

MODE	# OF SCREENS DISPLAYED	SCREEN	# OF CELL DOT	# OF CELL COLOR	# OF PALETTES	# OF COLORS PER SCREEN	FUNCTION		⑩	⑪							
							①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪
0	MAX 4	BG1	8 X 8	4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	
		BG2	OR	4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	
		BG3		4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	
		BG4	16 X 16	4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	
1	MAX 3	BG1		16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	
		BG2		16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	
		BG3		4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	
2	MAX 2	BG1		16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	⑪
		BG2		16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	⑪
3	MAX 2	BG1		256	1	256	①	②	③		⑤	⑥	⑦	⑧	⑨	⑩	
		BG2		16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	
4	MAX 2	BG1		256	1	256	①	②	③		⑤	⑥	⑦	⑧	⑨	⑩	⑪
		BG2		4	8	32	①	②	③		⑤	⑥	⑦	⑧		⑩	⑪
5	MAX 2	BG1	↓	16	8	128	①	②	③		⑤	⑥	⑦	⑧			⑫
		BG2		4	8	32	①	②	③		⑤	⑥	⑦	⑧			⑫
6	1	BG1	16X8	16	8	128	①	②	③		⑤	⑥	⑦	⑧		⑩	⑫
7	1	BG1	8 X 8	256	1	256	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	
EXT BG	1	BG2	8 X 8	128	1	128	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	

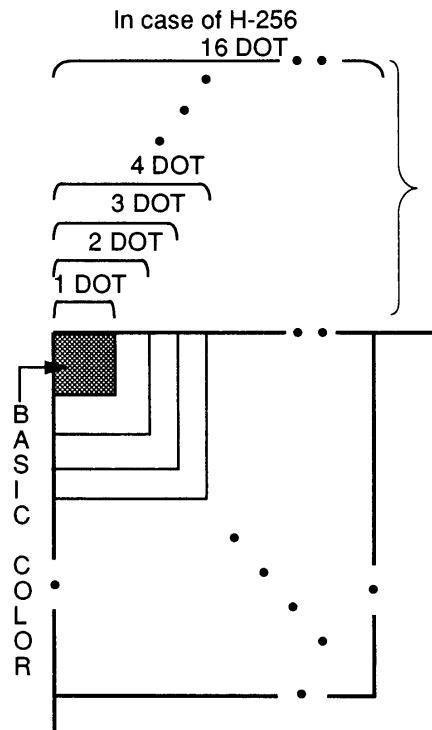
(NCL PG 72)

[Main Function of BG]

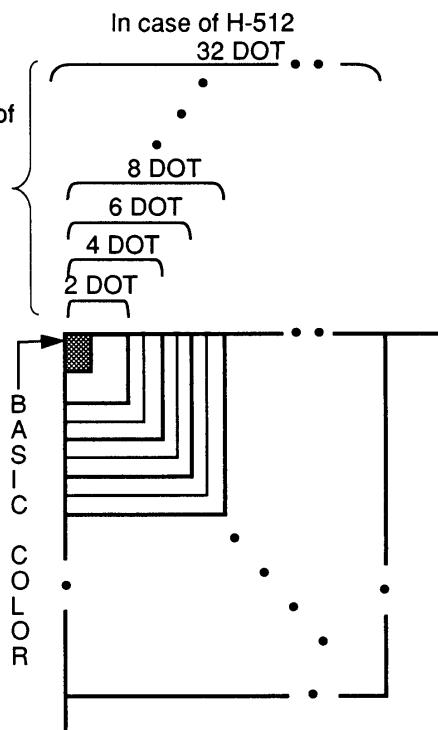
- |   |                       |
|---|-----------------------|
| 1. HV Scroll (each screen)              |                       |
| 2. HV Flip (each character)             |                       |
| 3. Mosaic                               | (Refer to Chapter 4)  |
| 4. Rotate, Enlarge, Reduce              | (Refer to Chapter 5)  |
| 5. Window Mask                          | (Refer to Chapter 6)  |
| 6. Screen Addition and Subtraction      | (Refer to ¶7.1)       |
| 7. Fixed Color Addition and Subtraction | (Refer to ¶7.2)       |
| 8. Color Window                         | (Refer to ¶7.2)       |
| 9. CG Direct Select                     | (Refer to Chapter 8)  |
| 10. Horizontal Pseudo 512               | (Refer to Chapter 9)  |
| 11. Offset Change                       | (Refer to Chapter 12) |
| 12. Horizontal 512 Mode                 | (Refer to Chapter 19) |

[Other Function]

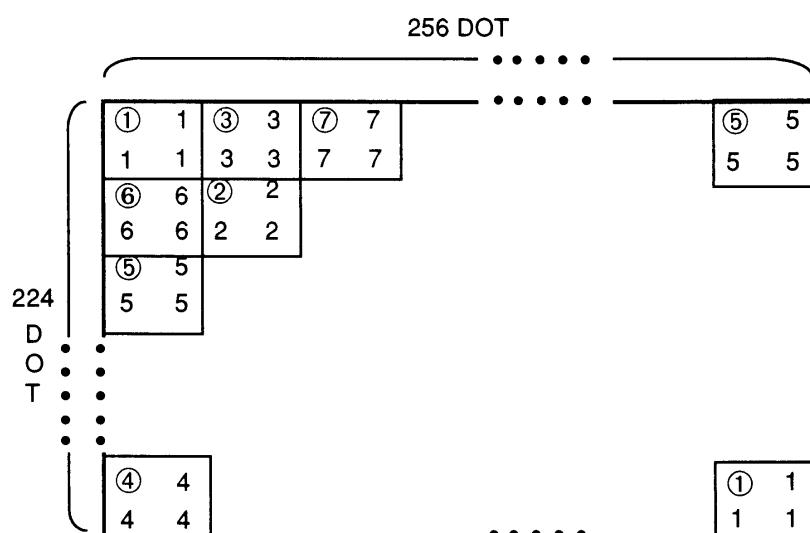
- Priority (each character/mode 0 ~ 6)
- Screen HV Rotate (mode 7)

**MOSAIC SCREEN**

\*All dots of the size designated become this color.



In case of the H-Pseudo 512 mode,  
2 x 2-dot mosaic can be made in size-0.  
(Refer to page A-3)

**MOSAIC SCREEN DISPLAY EXAMPLE (BG SCREEN)**  
 (When the mosaic size is 2 x 2-dot in the 256-mode)


⑪ is the basic color data

(NCL PG 73)

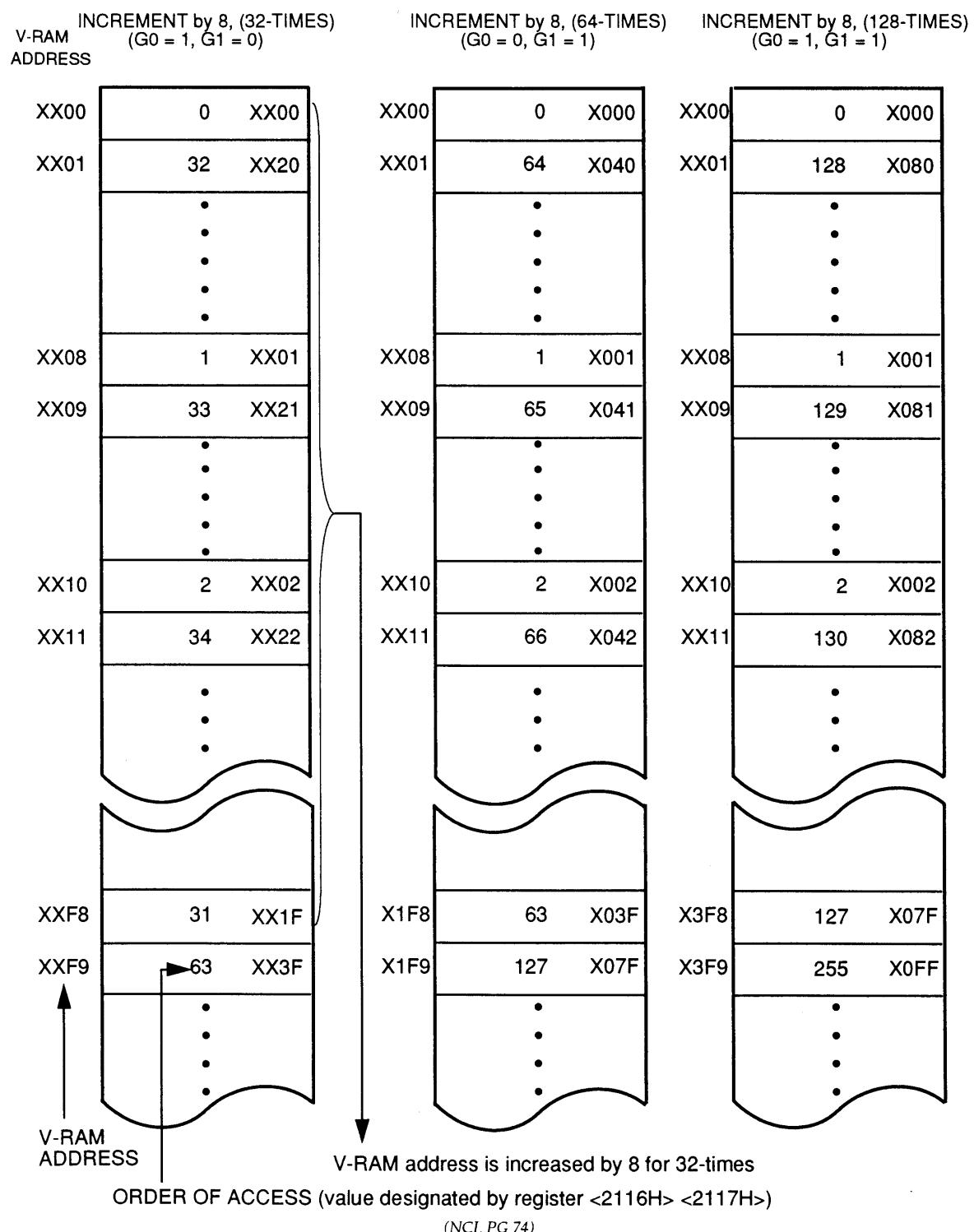
**ADDRESS INCREMENT ORDER**

Diagram illustrating memory mapping:

000H	001H	002H	003H
020H	021H	022H	
040H	041H		
340H			
360H	361H		

	01DH	01EH	01FH
	03EH	03FH	
		05FH	
			35FH
	37DH	37EH	37FH

When SC data (Name) is set during BG Mode 0 ~ 6 as demonstrated in the table above, Character data accesses horizontally by 8 dots in Full Graphic (G0, G1) of register <2115H>.

2 Bit/Dot (G0 = 1, G1 = 0)

Access Order →

0	1	2		...	29	30	31
32							63

4 Bit/Dot (G0 = 0, G1 = 1)

(1 Frame is 8 - bit x 2)

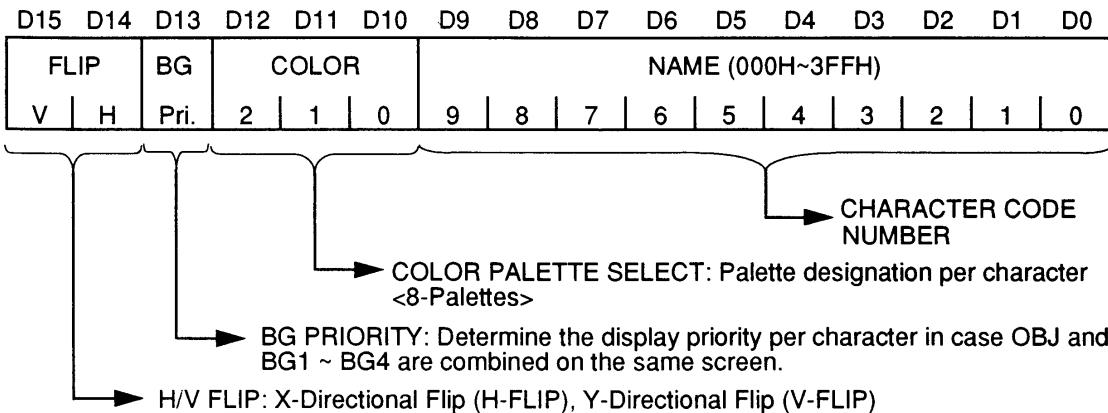
0,1	2,3	4,5		...	58,59	60,61	62,63
64,65							126,127

8 Bit/Dot (G0 = 1, G1 = 1)

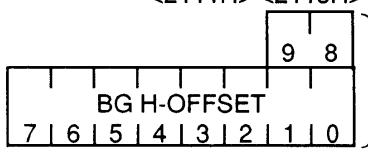
(1 Frame is 8 - bit x 4)

0~3	4~7	8~11		...	116~119	120~123	124~127
128~131							252~255

(NCL PG 74a)

**BG SC DATA (MODE 0 ~ 6)**

## BG SCREEN H/V SCROLL

REGISTER <210DH> <210FH>  
<2111H> <2113H>

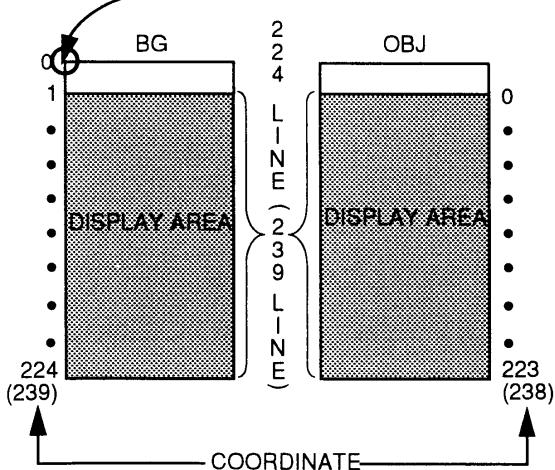
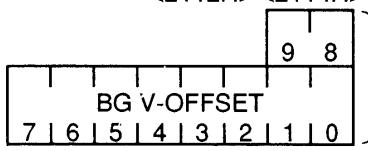
RANGE OF H-SCROLL

0 ~ 1023 DOT

RANGE OF V-SCROLL

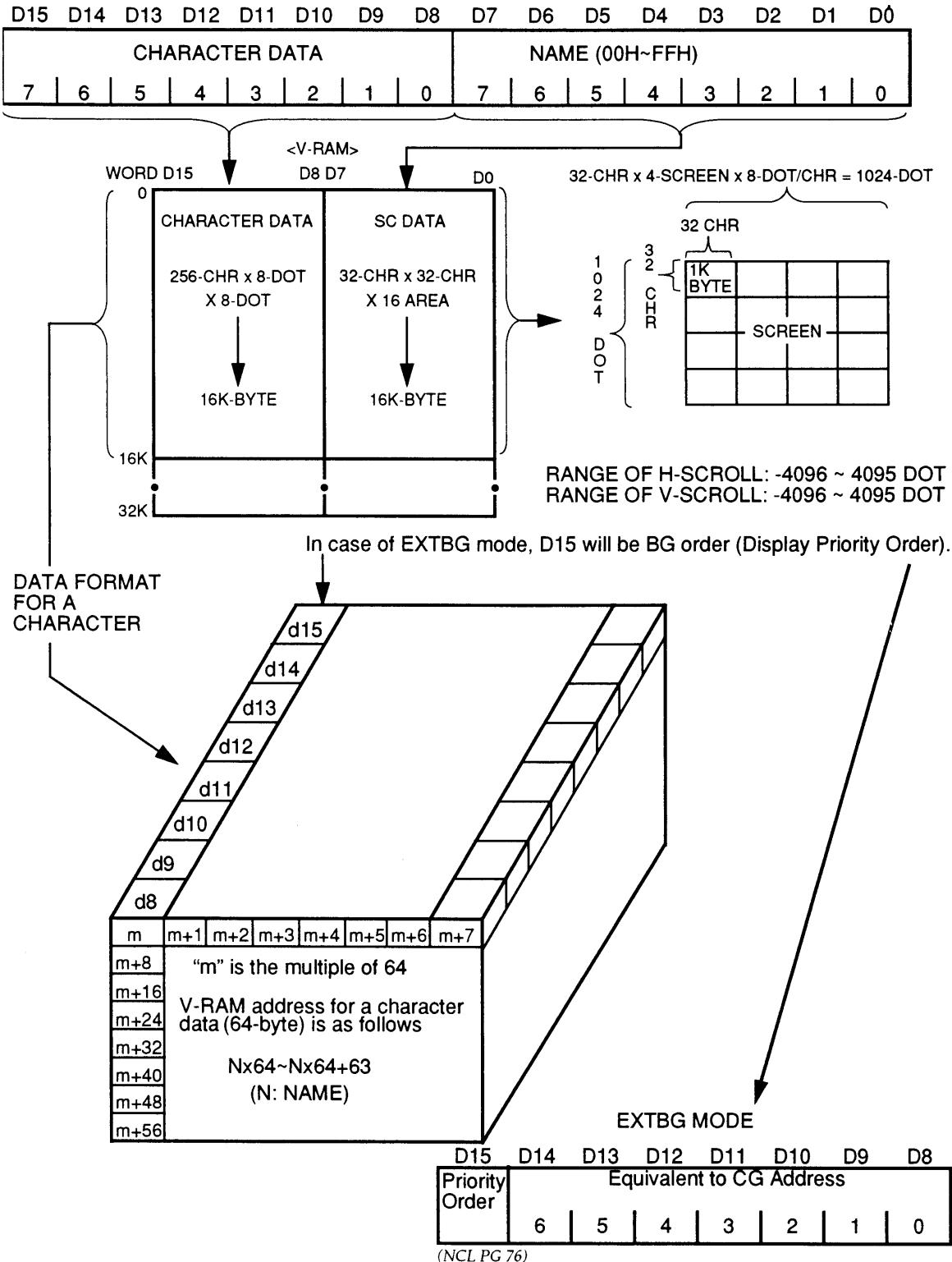
0 ~ 1023 DOT

H/Vscroll range may be changed and 2-dot scroll may be possible, depending on the combination of the modes (512, 16-size, interlace, etc.). Also, SC size may be changed against the screen. (Page A-21 and A-22)

REGISTER <210EH> <2110H>  
<2112H> <2114H>

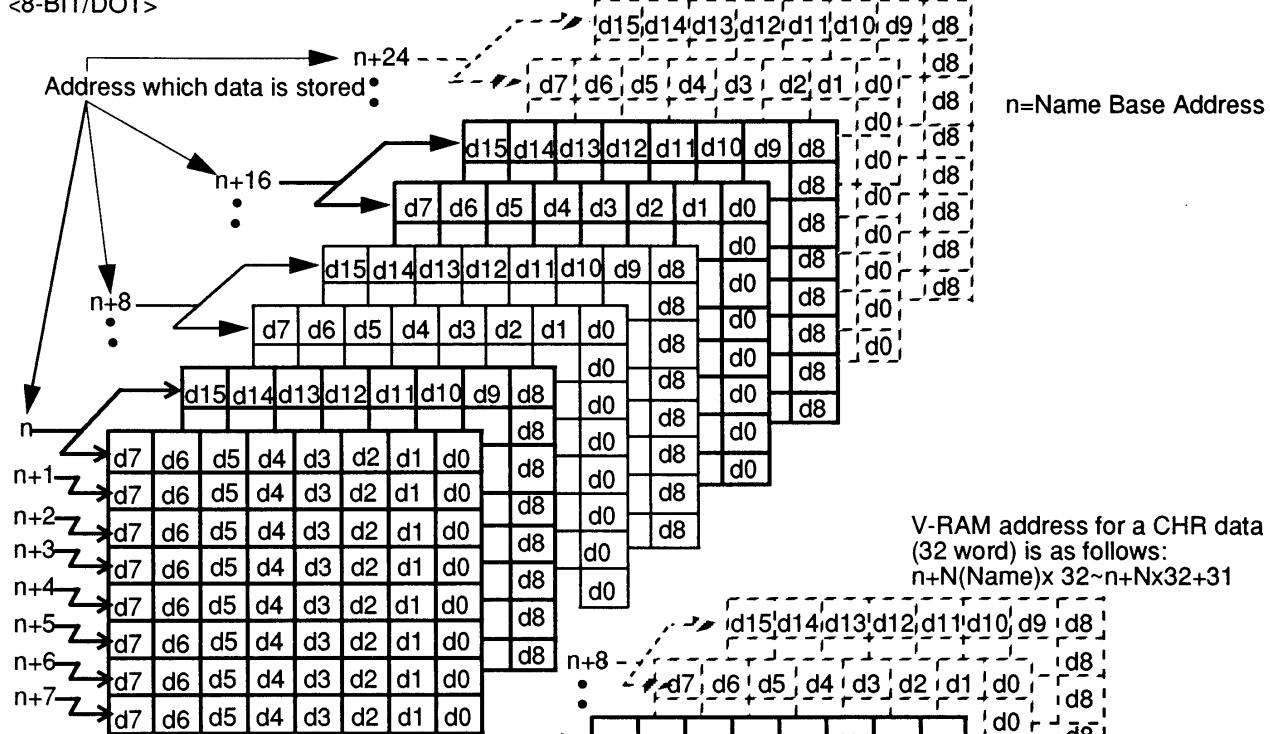
Y coordinate of BG and OBJ has one - line gap.

(NCL PG 75)

**BG SC DATA (MODE 7)**

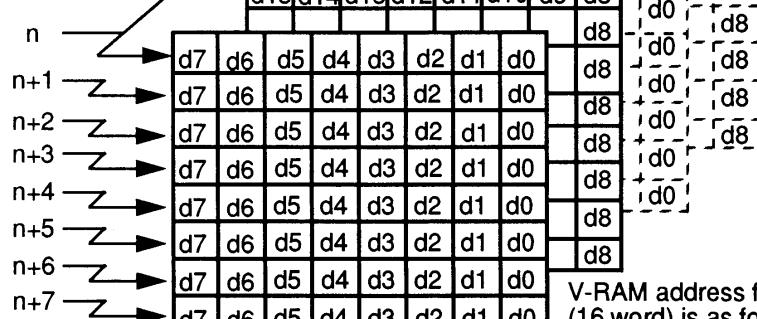
## CHR DATA CONSTRUCTION

&lt;8-BIT/DOT&gt;



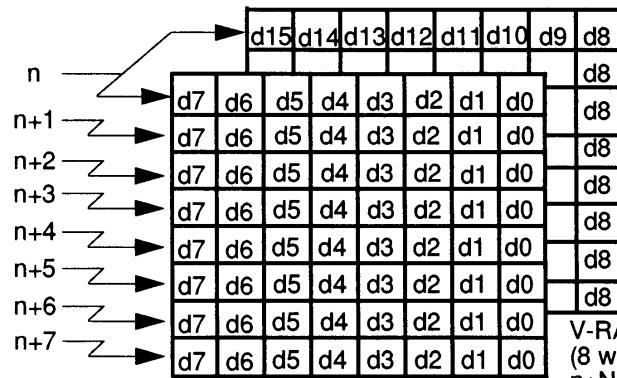
V-RAM address for a CHR data  
(32 word) is as follows:  
 $n+N(\text{Name}) \times 32 \sim n+N \times 32 + 31$

&lt;4-BIT/DOT&gt;



V-RAM address for a CHR data  
(16 word) is as follows:  
 $n+N(\text{Name}) \times 16 \sim n+N \times 16 + 15$

&lt;2-BIT/DOT&gt;

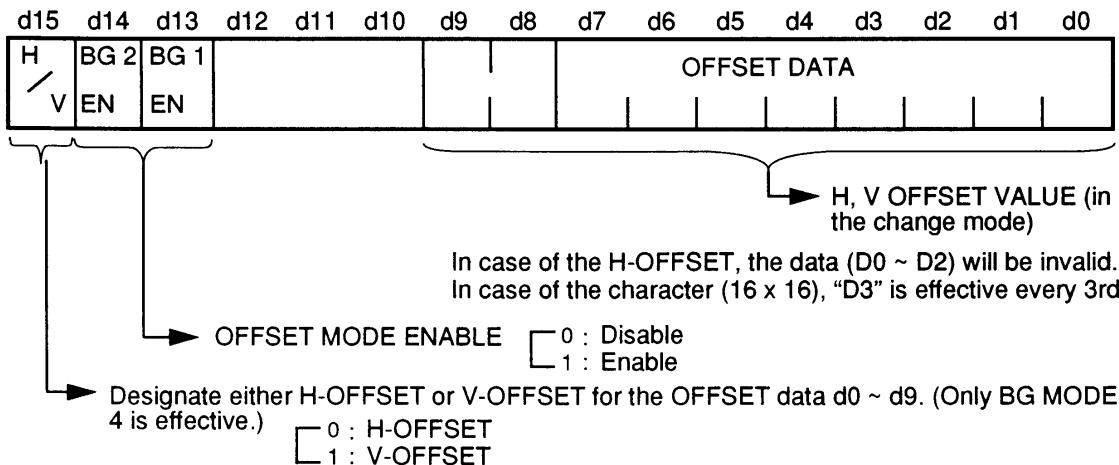


V-RAM address for a CHR data  
(8 word) is as follows:  
 $n+N(\text{Name}) \times 8 \sim n+N \times 8 + 7$

(NCL PG 77)

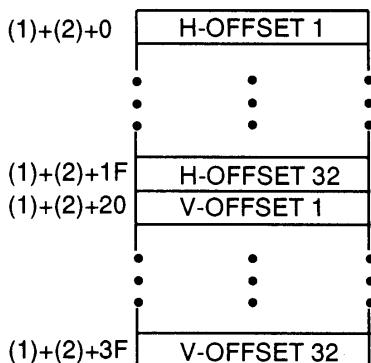
## OFFSET CHANGE MODE

The offset change mode can be used in the BG mode 2, 4 and 6, and the following data is required in this mode.

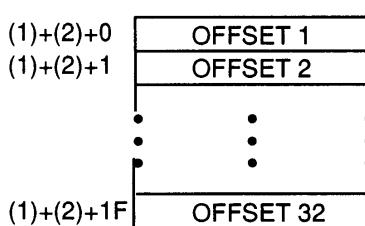


Write this data to VRAM of address designated at (1) and (2), using the BG Mode.  
(See below.)

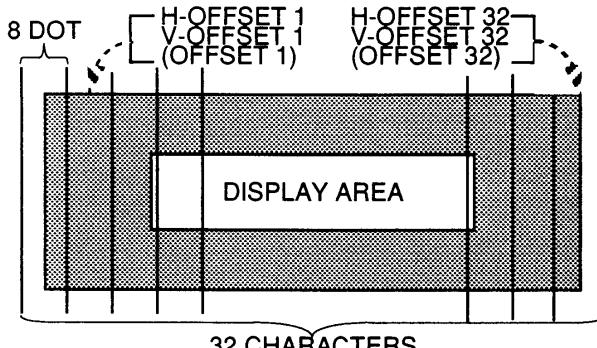
## &lt;MODE 2,6&gt;



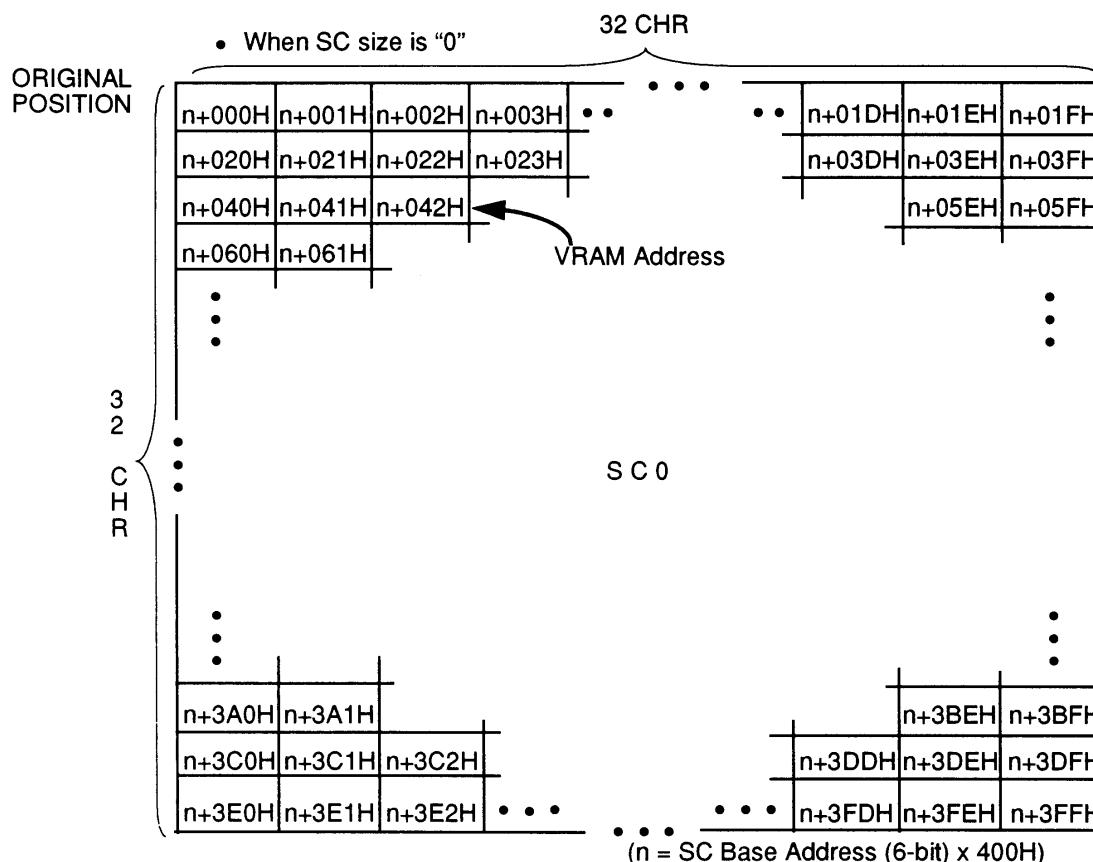
## &lt;MODE 4&gt;



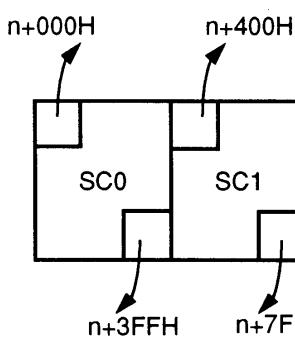
In case BG3 SC size is S1=0, S0=0  
(1): BG3 SC Base Address ([value set by "d2" ~ "d7" of <2109H>]x1024)  
(2): BG3 SC Offset Address ([value set by "d3" ~ "d7" of <2112H>]x32) + ([value set by "d3" ~ "d7" of <2111H>])



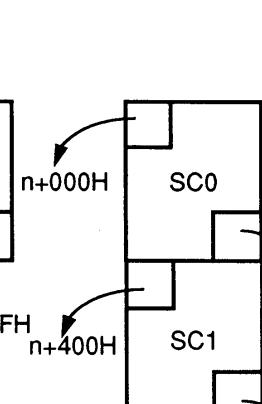
The offset value can be changed by each column (character unit).  
(Up to 3rd character can be seen horizontally on the screen by setting the offset value of the entire screen, but the offset can not be changed for 1st character (0 character).  
(NCL PG 78)

**BG SCREEN** (BG Mode 0 ~ 6)

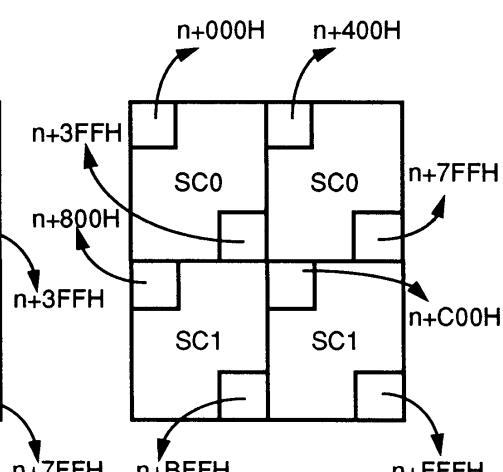
• When SC size is "1"



• When SC size is "2"

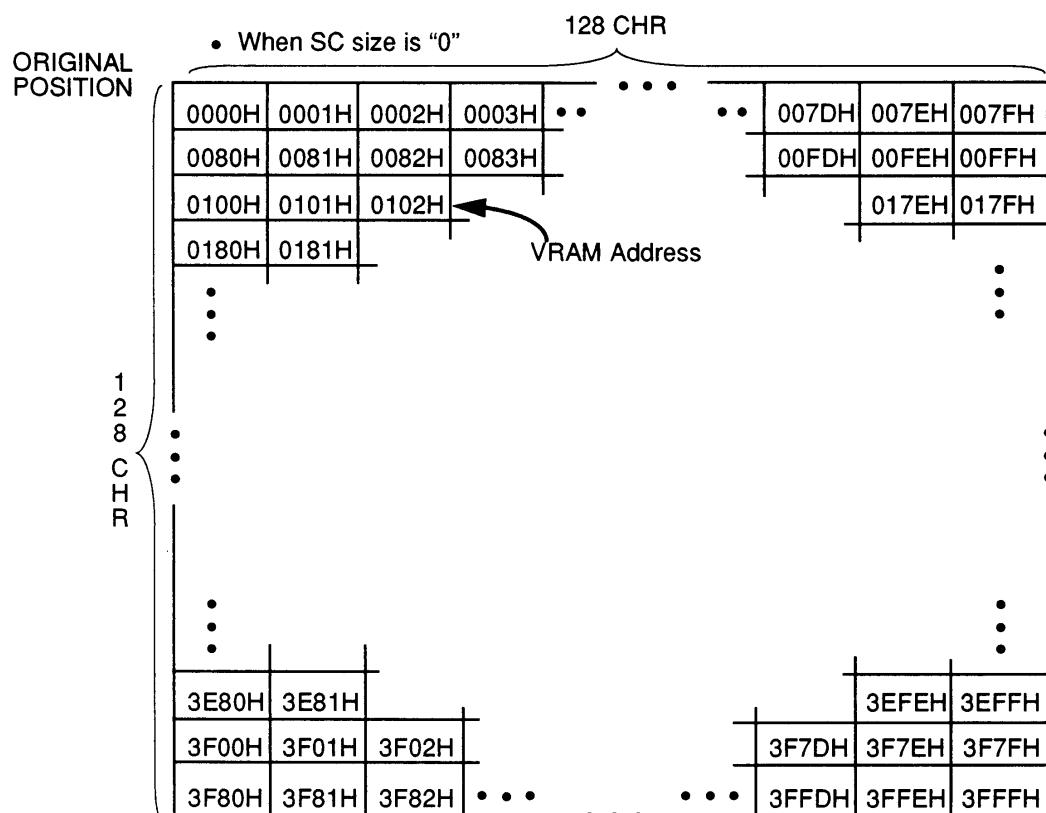


• When SC size is "3"



(NCL PG 79)

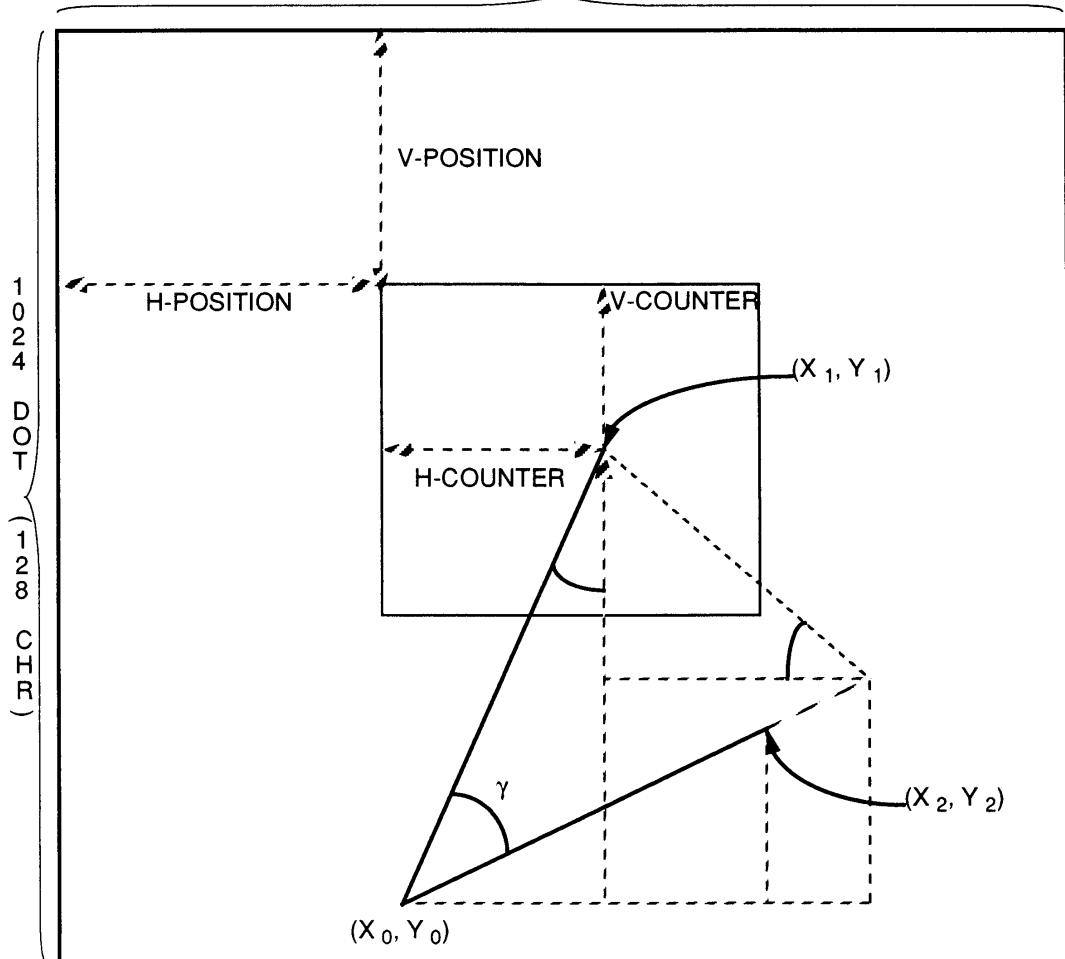
**BG Screen (BG Mode 7)**  
Screen Size and Area are Fixed



(NCL PG 79a)

**OPERATION (ROTATION/ENLARGEMENT/REDUCTION)**

1024 DOT (128 CHR)

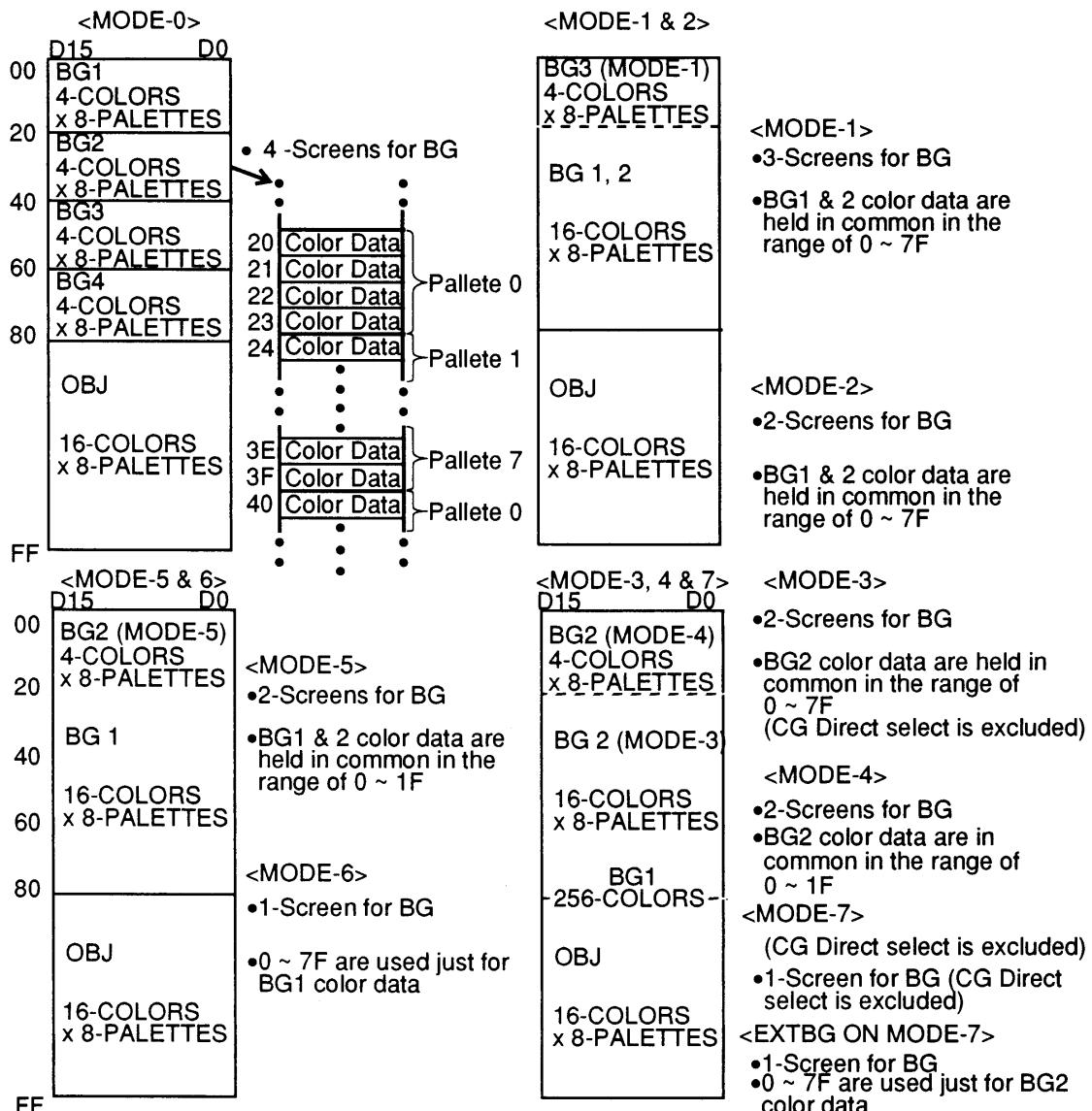
**ROTATIONAL TRANSFORM FORMULA**

$$\begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} \cos\gamma & \sin\gamma \\ -\sin\gamma & \cos\gamma \end{bmatrix} \begin{bmatrix} X_1 - X_0 \\ Y_1 - Y_0 \end{bmatrix} + \begin{bmatrix} X_0 \\ Y_0 \end{bmatrix}$$

 $X_0 \bullet Y_0$ : Center Coordinate $X_1 \bullet Y_1$ : Display Coordinate $X_2 \bullet Y_2$ : Coordinate before calculation of display coordinate

If the reduction rates for X-dir ( $a$ ) and the reduction rates for Y-dir ( $\beta$ ) are considered, the formula described above will be as follows:

$$\begin{aligned} A &= \cos\gamma \times (1/a), & B &= \sin\gamma \times (1/a), \\ C &= -\sin\gamma \times (1/\beta), & D &= \cos\gamma \times (1/\beta), \\ && (NCL PG 80) \end{aligned}$$

**CG-RAM****CG-RAM COLOR DATA**

BLUE	GREEN	RED
d14   d13   d12   d11   d10	d9   d8   d7   d6   d5	d4   d3   d2   d1   d0

**DIRECT SELECT COLOR DATA**

BLUE	GREEN	RED
DA7   DA6   CL2   0   0	DA5   DA4   DA3   CL1   0	DA2   DA1   DA0   CL0   0

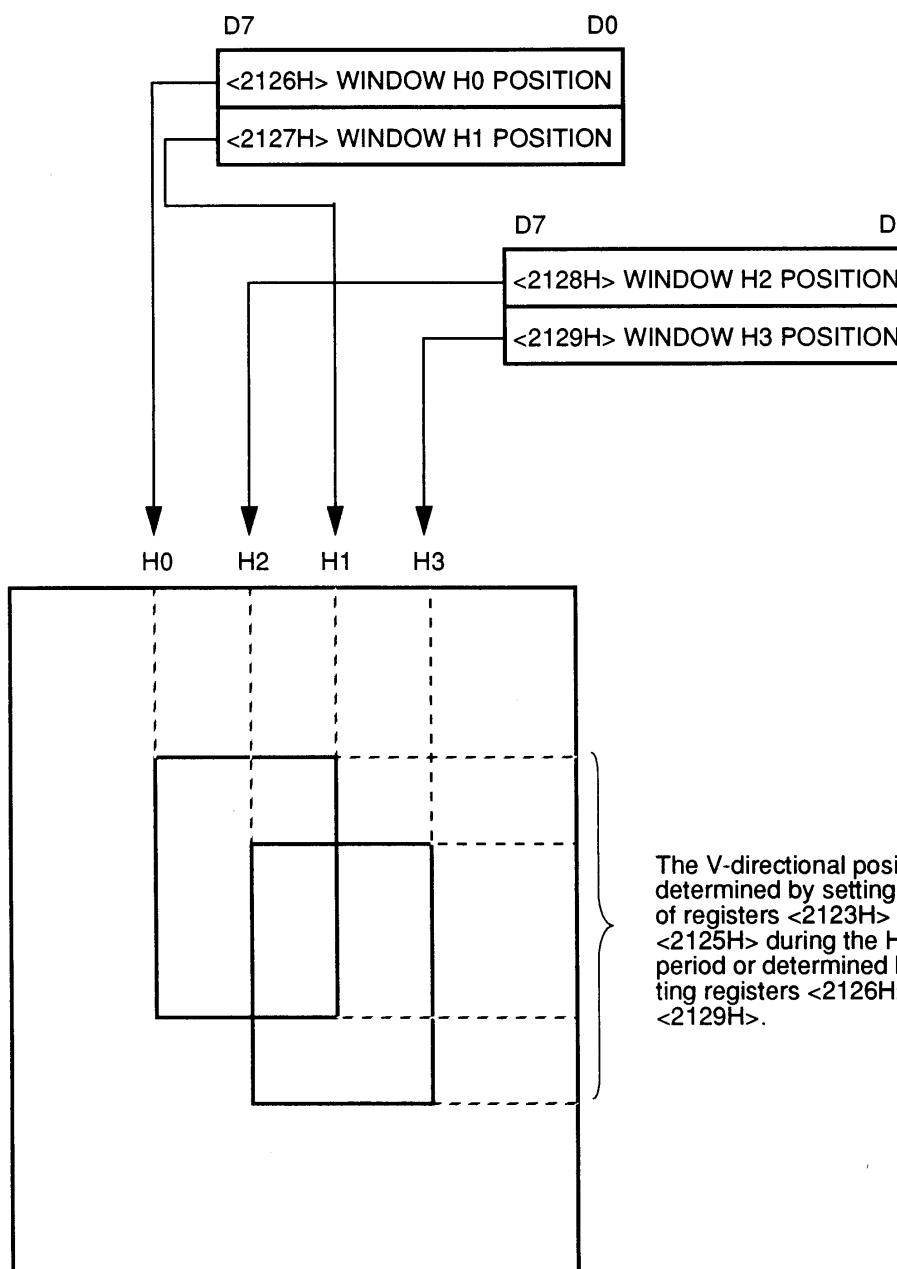
NOTE: DA0 ~ DA7 are used for the character dot data. CL0 ~ CL2 are used for the BG-SC data of the color. (However, in case of Mode-7, CL0 ~ CL2 should be "0")

NOTE: If they are "0," it becomes transparent. The color of CG-RAM address (00H) will be background.

(NCL PG 81)

**WINDOW**

REGISTER &lt;2126H&gt; ~ &lt;2129H&gt;

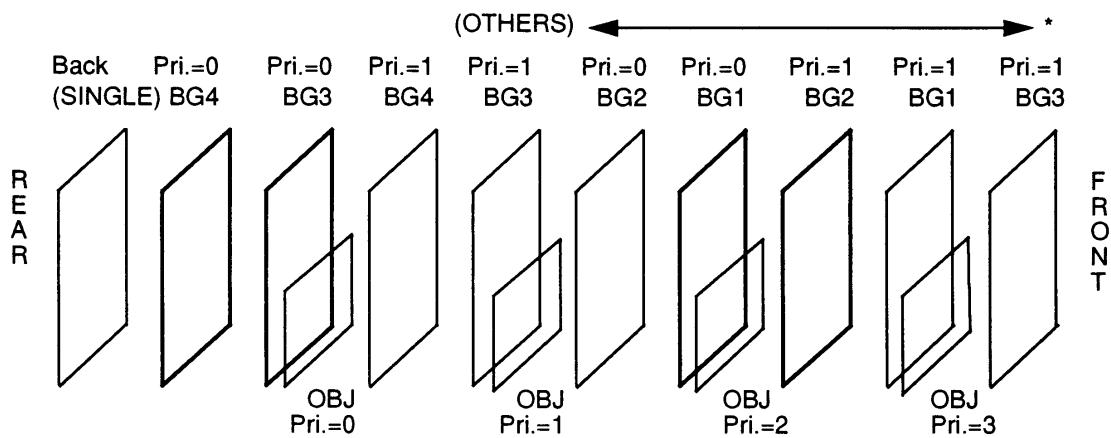


(NCL PG 82)

**BG & OBJ PRIORITY**

4-SCREEN/3-SCREEN MODE (In case Mode 0 and 1 are selected by register &lt;2105H&gt;)

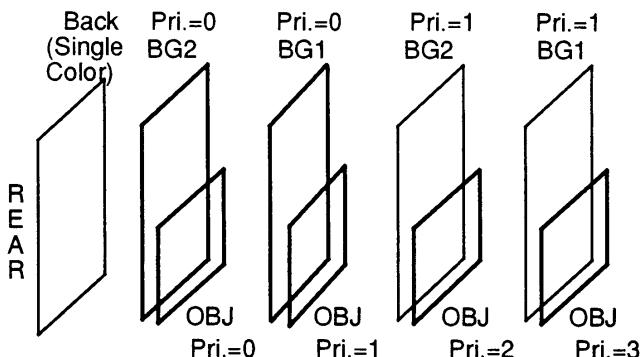
\*In case "D3=1" is selected by register &lt;2105H&gt; in the mode-1



&lt;Example of Display Priority (in case of mode 0)&gt;



2-SCREEN/1-SCREEN MODE (in case Mode 2 ~ 7 is selected by register &lt;2105H&gt;)



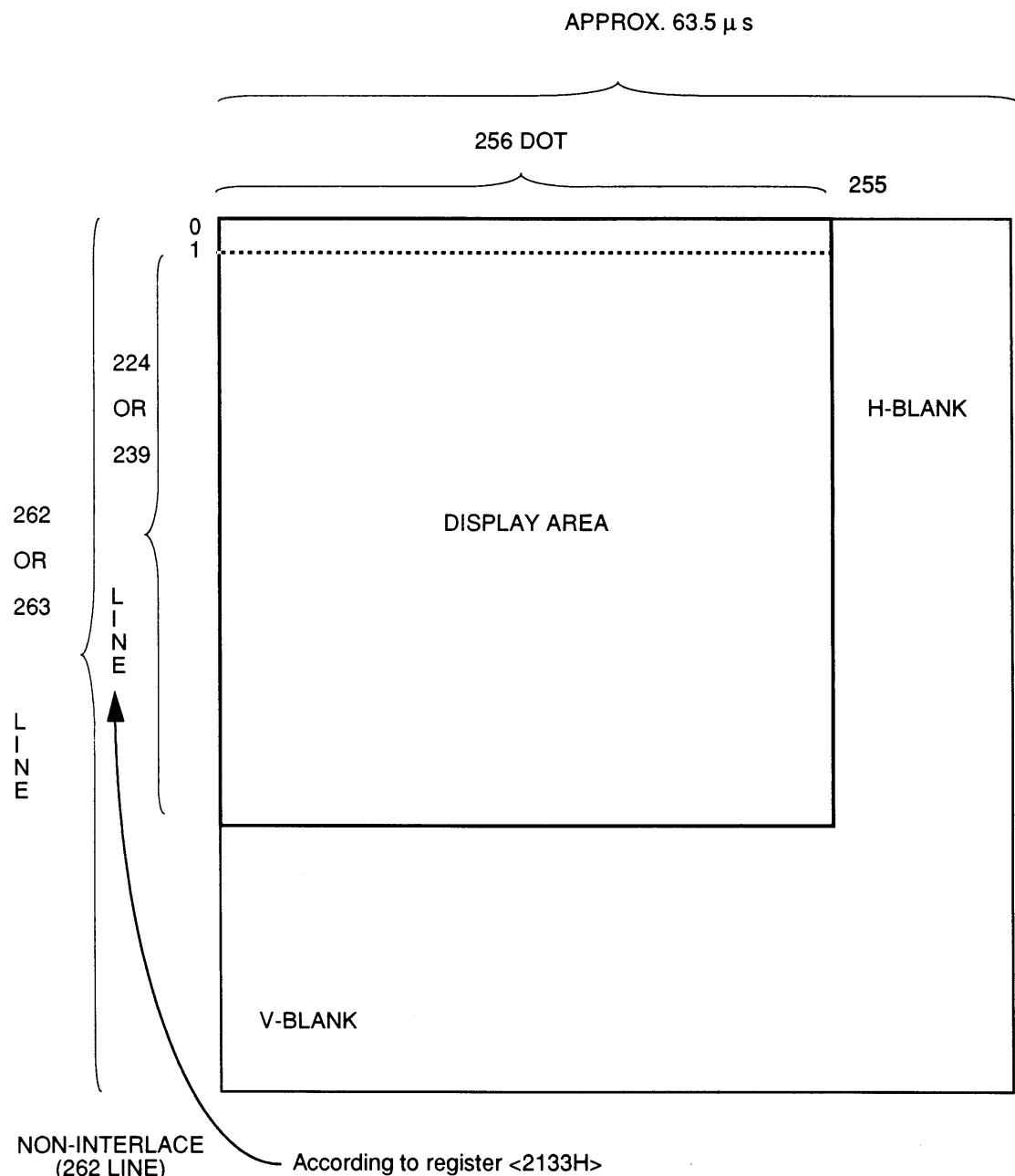
NOTE: In case of the display priority between the OBJ's, normally the lower numbered OBJ will be displayed as higher priority. (See page 1-20-2 for exception.)

This display priority will be determined before the priority between OBJ and BG is determined.

NOTE: In case of Mode 7, the priority is 0 at BG1.

(NCL PG 83)

## SCREEN



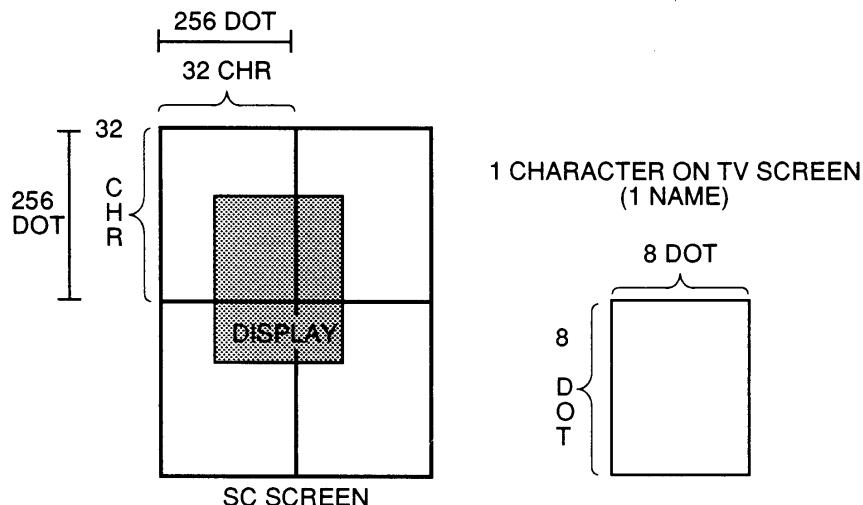
(NCL PG 84)

**BG SCREEN**

H/V SCROLL ① (Scroll range by the combination of modes and SC size against screen)  
 <Example: in case SC size is "3" - refer to register 2107H ~ 210AH>

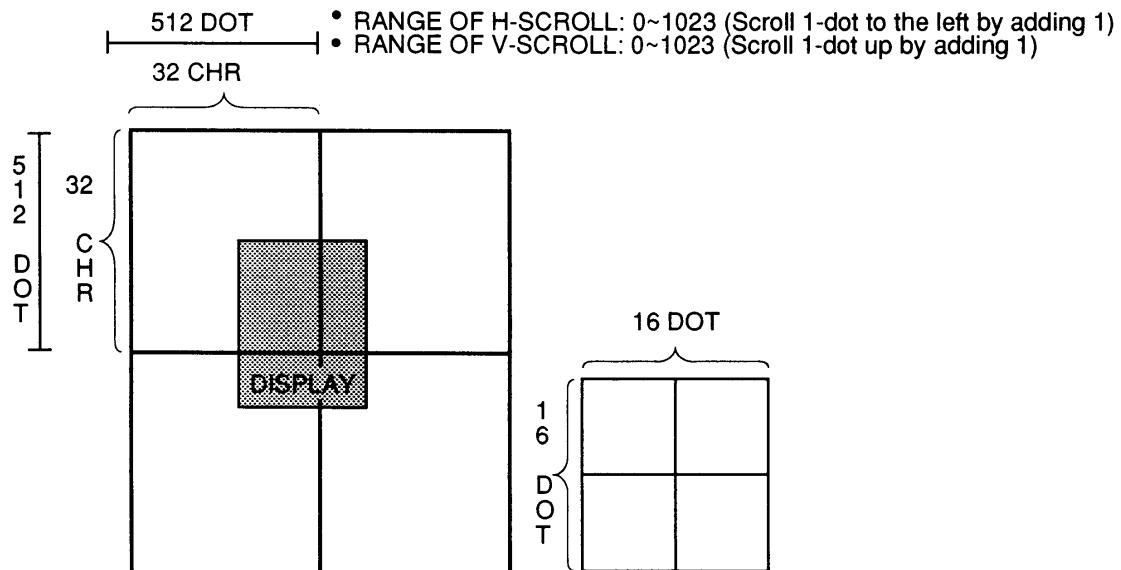
\*In case of mode 0, 1, 2, 3, & 4

- BG SIZE (8 x 8)



- RANGE OF H-SCROLL: 0~511 (Scroll 1-dot to the left by adding 1)
- RANGE OF V-SCROLL: 0~511 (Scroll 1-dot up by adding 1)

- BG SIZE (16 x 16)

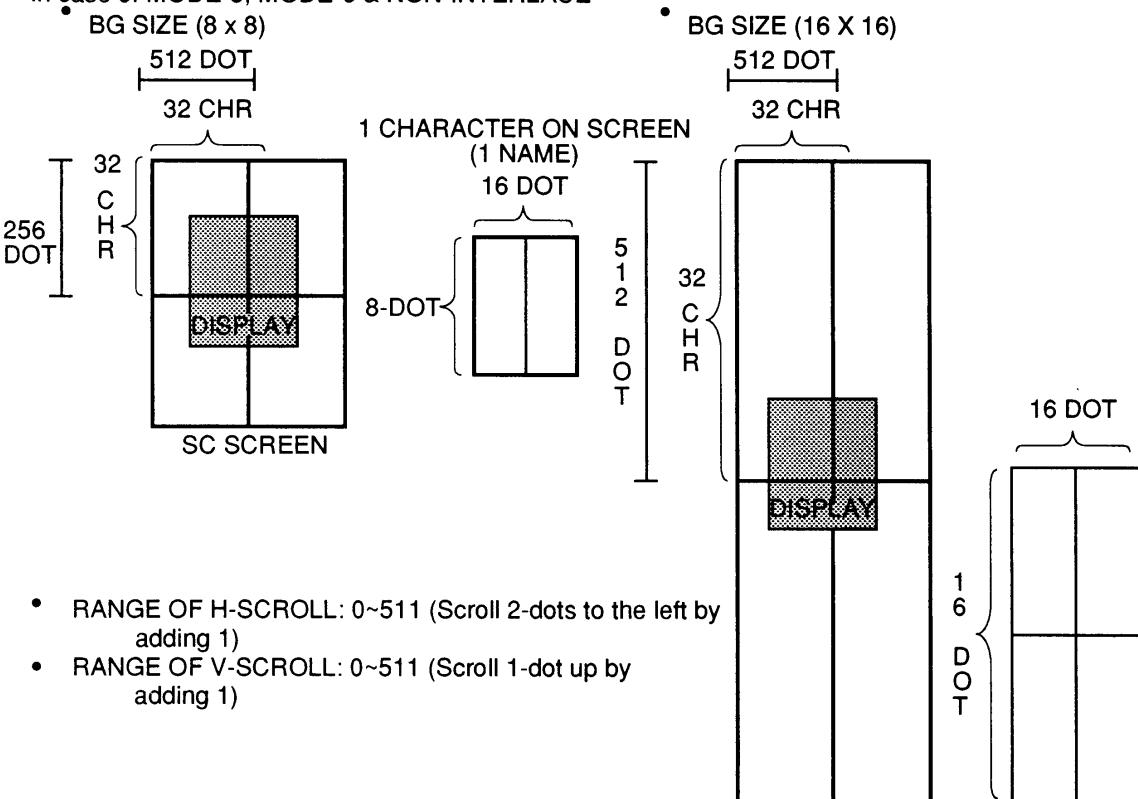


(NCL PG 85)

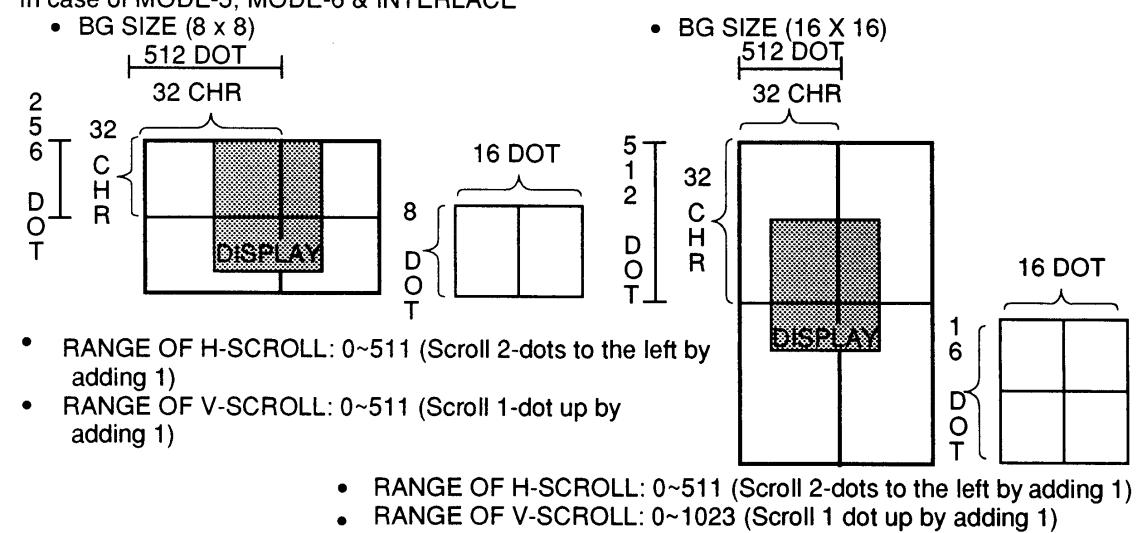
**BG SCREEN**

H/V SCROLL ② (Scroll range by the combination of modes and SC size against screen)  
 <Example: in case SC size is "3" - refer to register 2107H ~ 210AH>

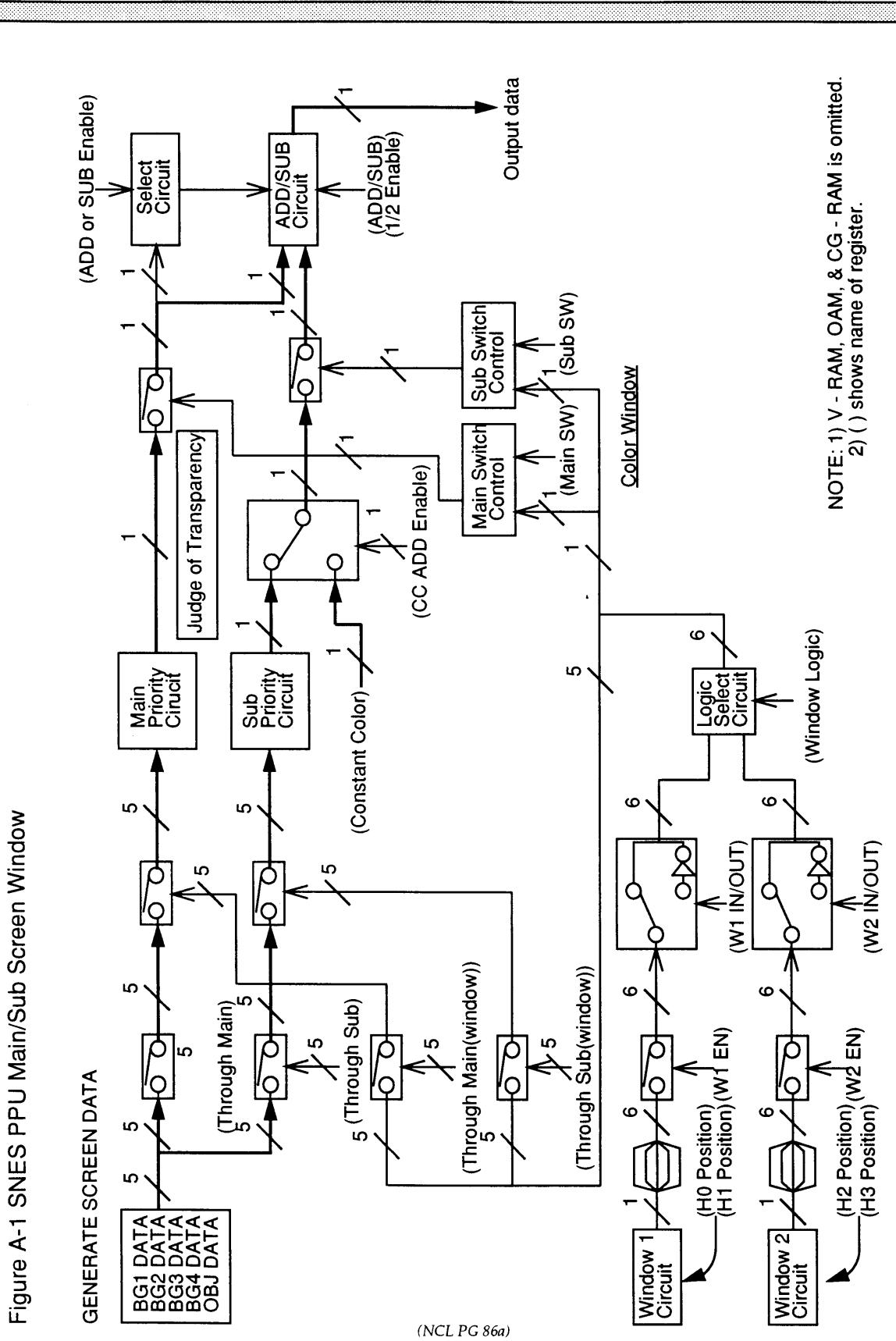
\*In case of MODE-5, MODE-6 & NON-INTERLACE



\*In case of MODE-5, MODE-6 & INTERLACE



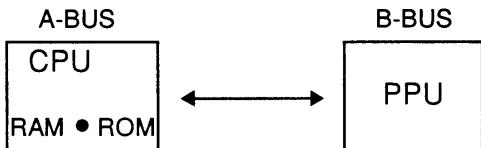
(NCL PG 86)



## *Appendix B. CPU Registers*

## GENERAL PURPOSE DMA

### i) Transfer Method

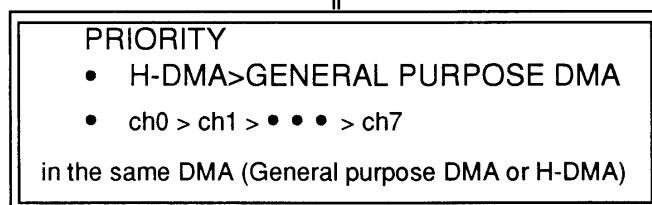


## ii) Data Format

## Data Format

### iii) Trigger (Start)

## Trigger (Start) General Purpose DMA Enable Flag

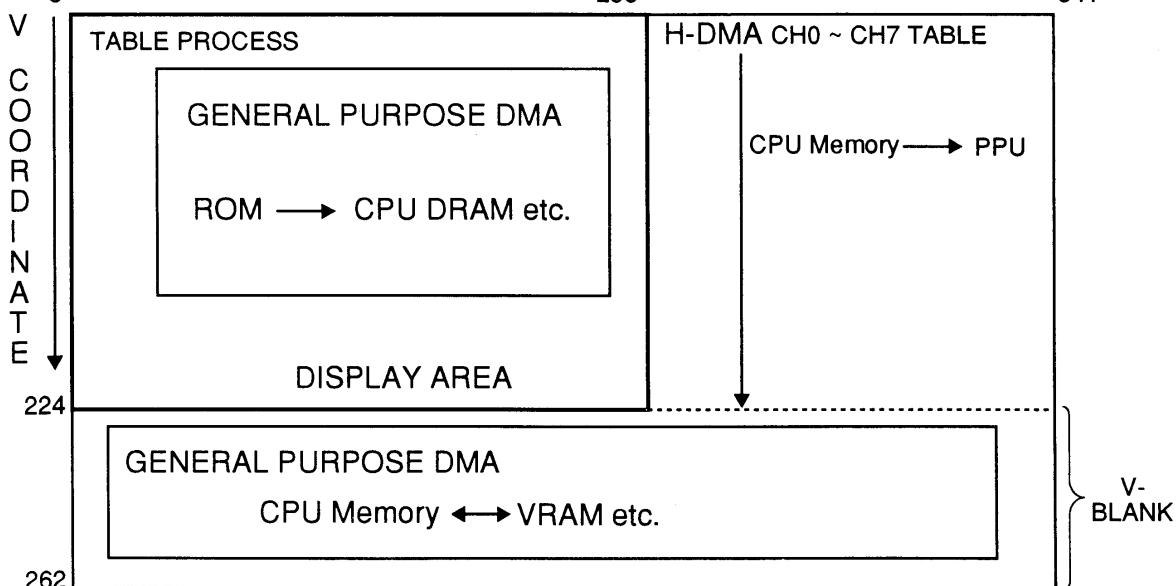


## H COORDINATE

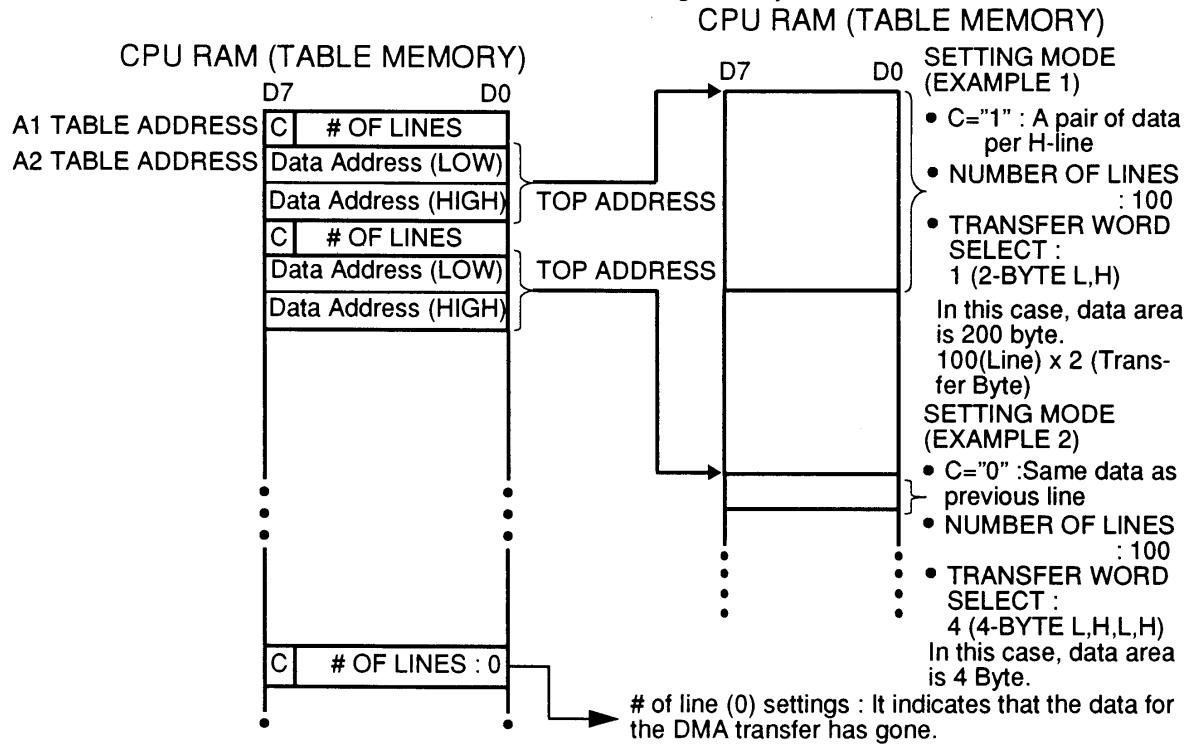
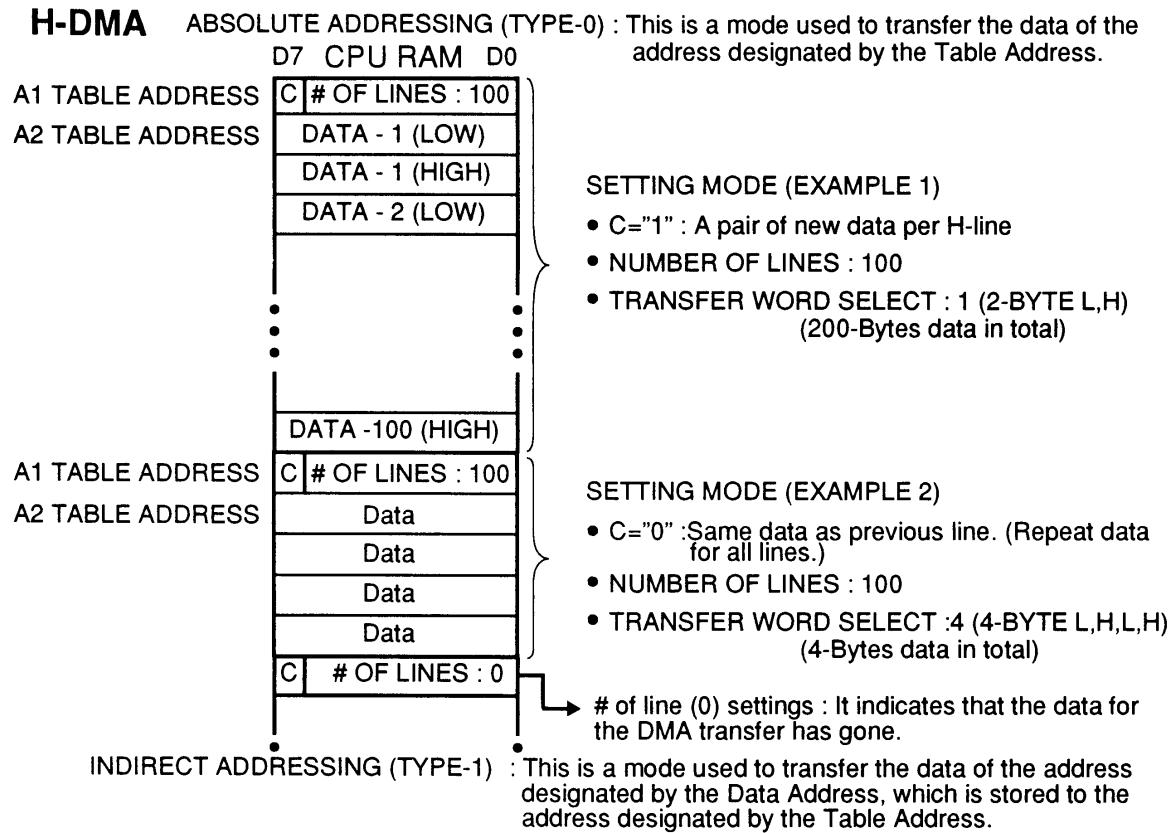
2

H-BLANK

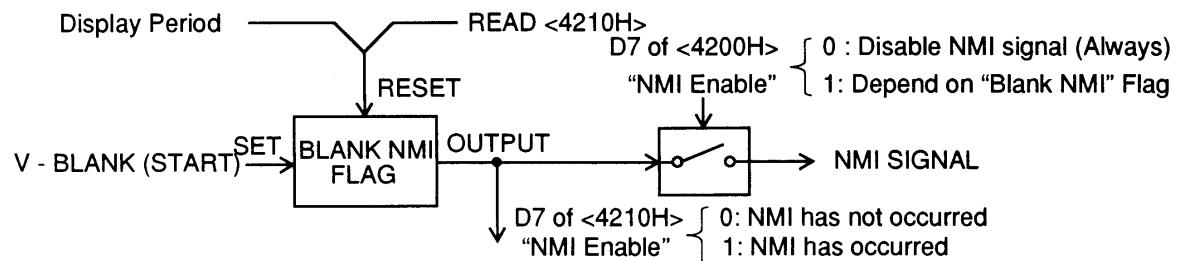
341



(NCL PG 101)



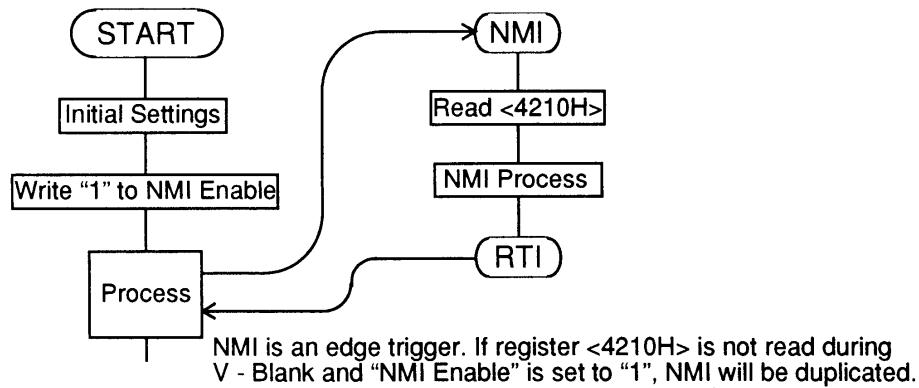
## DETECT BEGINNING OF V - BLANK



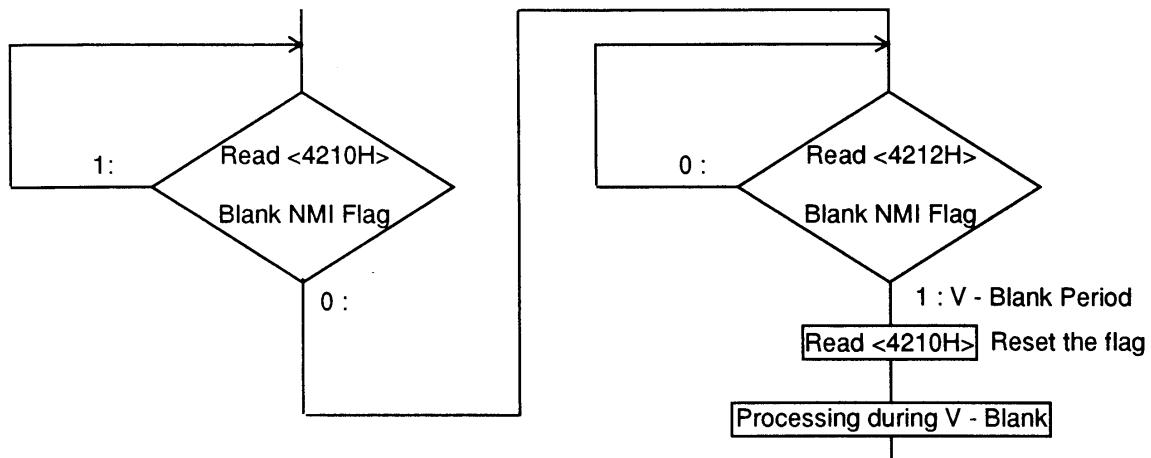
The "Blank NMI" flag of register <4210H> will be set at the beginning of V - Blank and will reset at the end of V - Blank. It may also be reset by reading register <4210H>.

### <EXAMPLE>

1. In case of detecting the beginning of V - Blank by NMI :



2. In case of detecting the beginning of V - Blank by the flag :



**SUMMARY OF REGISTERS**

REGISTERS (WRITE ) S - PPU

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0									
2100H	Blanking				Fade IN/OUT (0 ~ 15)												
2101H	OBJ Size Select			OBJ Name Select		OBJ Name Base Address											
2102H	OAM Address																
2103H	OAM Priority Rotation						OAM Address MSB										
2104H	OAM Data (Low, High)																
2105H	BG4	BG3	BG2	BG1	BG 3 Priority	BG Mode (0 ~ 7)											
2106H	Mosaic Size				Mosaic Enable												
2107H	BG1 SC Base Address						BG1 SC Size										
2108H	BG2 SC Base Address						BG2 SC Size										
2109H	BG3 SC Base Address						BG3 SC Size										
210AH	BG4 SC Base Address						BG4 SC Size										
210BH	BG2 Name Base Address				BG1 Name Base Address												
210CH	BG4 Name Base Address				BG3 Name Base Address												
210DH	BG1 H - Offset (Low, High)																
210EH	BG1V - Offset (Low, High)																
210FH	BG2 H - Offset (Low, High)																
2110H	BG2 V - Offset (Low, High)																
2111H	BG3 H - Offset (Low, High)																
2112H	BG3 V - Offset (Low, High)																
2113H	BG4 H - Offset (Low, High)																
2114H	BG4 V - Offset (Low, High)																
2115H	H/L Inc				V - RAM Address Sequence Mode Full Graphic   SC Increment												
2116H	V - RAM Address (Low)																
2117H	V - RAM Address (High)																
2118H	V - RAM Data (Low)																
2119H	V - RAM Data (High)																
211AH	Screen Over					Screen Flip V   H											

(NCL PG 104)

## REGISTERS (WRITE ) S - PPU

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
211BH								Matrix Parameter A (Low, High)
211CH								Matrix Parameter B (Low, High)
211DH								Matrix Parameter C (Low, High)
211EH								Matrix Parameter D (Low, High)
211FH								Matrix Parameter X (Low, High)
2120H								Matrix Parameter Y (Low, High)
2121H								CG - RAM Address
2122H								CG - RAM Data (Low, High)
2123H	BG2 Window W2 EN   IN/OUT   W1 EN   IN/OUT				BG1 Window W2 EN   IN/OUT   W1 EN   IN/OUT			
2124H	BG4 Window W2 EN   IN/OUT   W1 EN   IN/OUT				BG3 Window W2 EN   IN/OUT   W1 EN   IN/OUT			
2125H	Color Window W2 EN   IN/OUT   W1 EN   IN/OUT				OBJ Window W2 EN   IN/OUT   W1 EN   IN/OUT			
2126H					Window H0 Position (0 ~ 255)			
2127H					Window H1 Position (0 ~ 255)			
2128H					Window H2 Position (0 ~ 255)			
2129H					Window H3 Position (0 ~ 255)			
212AH	BG4		BG3		BG2		BG1	Window Logic
212BH					Color			Window Logic OBJ
212CH				OBJ	BG4	BG3	BG2	Through Main BG1
212DH				OBJ	BG4	BG3	BG2	Through Sub BG1
212EH				OBJ	BG4	BG3	BG2	Through Main (Window) BG1
212FH				OBJ	BG4	BG3	BG2	Through Sub (Window) BG1
2130H	Main SW (A)	Window ON/OFF Sub SW (B)					CG ADD Enable	Direct Select
2131H	ADD SUB	1/2 Enable	BACK	OBJ	ADD or SUB Enable BG4	BG3	BG2	BG1
2132H	Blue	Green	Red		Color Constant Data			Color Brilliance Data
2133H	EXT. Sync.	EXT. Input		Pseudo 512	224/239	OBJ - V Select		Inter- lace

(NCL PG 104)

## S - PPU READ REGISTER

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
2134H					M P Y (Low)			
2135H					M P Y (Mid)			
2136H					M P Y (High)			
2137H				Soft Latch for H/V Counter				
2138H				OAM Data (Low, High)				
2139H				V - RAM Data (Low)				
213AH				V - RAM Data (High)				
213BH				CG Data (Low, High)				
213CH				Output Data of H - Counter (Low, High)				
213DH				Output Data of V - Counter (Low, High)				
213EH	Time Over	Range Over	Master /Slave		S - PPU1 Version Number			
213FH	Field	EXT. Latch		NTSC /PAL	S - PPU2 Version Number			

## APU READ/WRITE REGISTER

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
2140H					APU I/O Port			
2141H					APU I/O Port			
2142H					APU I/O Port			
2143H					APU I/O Port			

## WORK RAM READ/WRITE REGISTER

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
2180H					WORK RAM Data			

## WORK RAM WRITE REGISTER

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
2181H					WORK RAM Address (Low)			
2182H					WORK RAM Address (Mid)			
2183H					WORK RAM Address (High)			

## REGISTERS (WRITE ) S - CPU

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
4200H	NMI Enable		Timer V - EN	Enable H - EN				Joy - C Enable
4201H					I/O Port			
4202H					Multiplicand - A			
4203H					Multiplier - B			
4204H					Dividend - C (Low)			
4205H					Dividend - C (High)			
4206H					Divisor - B			
4207H					H - Counter Timer			
4208H							H - MSB	
4209H					V - Counter Timer			
420AH							V - MSB	
420BH					General Purpose DMA (Enable Flag) CH7 EN   CH6 EN   CH5 EN   CH4 EN   CH3 EN   CH2 EN   CH1 EN   CH0 EN			
420CH					H-DMA (Enable Flag) CH7 EN   CH6 EN   CH5 EN   CH4 EN   CH3 EN   CH2 EN   CH1 EN   CH0 EN			
420DH							2.68 /3.58	

(NCL PG 106)

## REGISTERS (READ) S - CPU

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
4210H	Blank NMI							SNES - CPU Version Number
4211H	Timer IRQ							
4212H	V -Blank	H -Blank					Joy - C Enable	
4213H				I/O Port				
4214H				Quotient - A (Low)				
4215H				Quotient - A (High)				
4216H				Product - C / Remainder (Low)				
4217H				Product - C / Remainder (High)				
4218H				Joy Controller I (Low)				
4219H				Joy Controller I (High)				
421AH				Joy Controller II (Low)				
421BH				Joy Controller II (High)				
421CH				Joy Controller III (Low)				
421DH				Joy Controller III (High)				
421EH				Joy Controller IV (Low)				
421FH				Joy Controller IV (High)				

## REGISTERS (WRITE ) S - CPU

ADDRESS	D7	D6	D5	D4	D3	D2	D1	D0
43X0H	CHX ★T-Org	CHX Type		A - Bus Address INC/DEC   Fixed		CHX Transfer Word Select		
43X1H				CHX B - Address				
43X2H				CHX A1 Table Address (Low)				
43X3H				CHX A1 Table Address (High)				
43X4H				CHX A Table Bank				
43X5H	CHX Data Address (H-DMA) / Number of Bytes to be Transferred (General Purpose DMA)							(Low)
43X6H	CHX Data Address (H-DMA) / Number of Bytes to be Transferred (General Purpose DMA)							(High)
43X7H				CHX Data Bank (H - DMA)				
43X8H				CHX A2 Table Address (Low)				
43X9H				CHX A2 Table Address (High)				
43XAH	Continue			Number of Lines				

\* T - Org means the "Transfer Orientation".

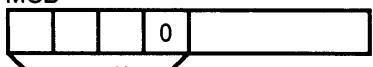
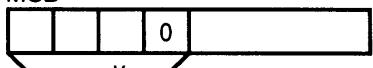
(NCL PG 106)

## Appendix C SPC700 Commands

### C.1 SUMMARY OF SPC700 COMMANDS

An SPC700 series is used for the SNES sound source CPU. However, standby and sleep modes cannot be used. The command set operand notation and explanation of command activity are indicated in the table below. The upper portion of the table contains symbols necessary to operand description. These are symbols necessary for assembler description. In the lower portion of the table, the values of the various operands are expressed as symbols. Assembler descriptions are given as numerical values or labels.

Table C-1 Command Operand Symbols and Meaning

Symbol	Meaning
A	A Register
X	X Register
Y	Y Register
PSW	Program Status Word
YA	Y, A paired 16-bit register
PC	Program Counter
SP	Stack Pointer
( )	Indirect Expression
( )+	Indirect Auto-increment Expression
#	Immediate Data
!	Absolute Address
/	Bit Reversal
.	Bit Position Indicator
[ ]	Indexed Indirect Expression
H	Hexadecimal Notation
imm	8-bit Immediate Data
dp	Offset Address within Direct Page
abs	16-bit Absolute Address
rel	Relative Offset 2's Complement
mem	Boolean Bit Operation Address
bit	Bit Location
x	MSB 
y	MSB 
upage	Offset Within U Page
n	Vector Call Number

(NCL PG 35)

The following symbols are used, in addition to those on the previous page, for the purpose of explaining operational functions.

Table C-2 Symbols and Meaning for Operational Description

Symbol	Meaning
N	Negative Flag
V	Overflow Flag
P	Direct Page Flag
B	Break Flag
H	Half Carry Flag
I	Indirect Master Enable Flag
Z	Zero Flag
C	Carry Flag
+	Addition
-	Subtraction
:	Comparison
AND	Logic Product
OR	Logic Sum
EOR	Exclusive Logic Sum
*	Multiplication
/	Division
Q	Division Quotient
R	Division Remainder
<d>	Destination
<S>	Source
→	Direction of Data Transmission
--	Data Decrement
++	Data Increment
<<	1 Bit Shift Left
>>	1 Bit Shift Right

Note: The number of cycles of conditional branching commands are appropriate to cases when there is no branching to the left side and there is branching to the right side.

Table C-3 Explaination of Symbols in the Status Flag Column

Symbol	Meaning
.	No Change
0	Cleared to "0"
1	Set to "1"
Flag Name	Set or Cleared Depending on Result

(NCL PG 36)

Table C-4 8-bit Data Transmission Commands, Group 1

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHI <sub>Z</sub> C
MOV	A, #imm	E8	2	2	A $\leftarrow$ imm	N.....Z
MOV	A, (X)	E6	1	3	A $\leftarrow$ (X)	N.....Z
MOV	A, (X)+	BF	1	4	A $\leftarrow$ (X) with auto increment	N.....Z
MOV	A, dp	E4	2	3	A $\leftarrow$ (dp)	N.....Z
MOV	A, dp+X	F4	2	4	A $\leftarrow$ (dp+X)	N.....Z
MOV	A, !abs	E5	3	4	A $\leftarrow$ (abs)	N.....Z
MOV	A, !abs+X	F5	3	5	A $\leftarrow$ (abs+X)	N.....Z
MOV	A, !abs+Y	F6	3	5	A $\leftarrow$ (abs+Y)	N.....Z
MOV	A, [dp+X]	E7	2	6	A $\leftarrow$ ((dp+X+1)(dp+X))	N.....Z
MOV	A, [dp]+Y	F7	2	6	A $\leftarrow$ ((dp+1)(dp)+Y)	N.....Z
MOV	X, #imm	CD	2	2	X $\leftarrow$ imm	N.....Z
MOV	X, dp	F8	2	3	X $\leftarrow$ (dp)	N.....Z
MOV	X, dp+Y	F9	2	4	X $\leftarrow$ (dp+Y)	N.....Z
MOV	X, !abs	E9	3	4	X $\leftarrow$ (abs)	N.....Z
MOV	Y, #imm	8D	2	2	Y $\leftarrow$ imm	N.....Z
MOV	Y, dp	EB	2	3	Y $\leftarrow$ (dp)	N.....Z
MOV	Y, dp+X	FB	2	4	Y $\leftarrow$ (dp+X)	N.....Z
MOV	Y, !abs	EC	3	4	Y $\leftarrow$ (abs)	N.....Z

Table C-5 8-bit Data Transmission Commands, Group 2

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHI <sub>Z</sub> C
MOV	(X), A	C6	1	4	A $\rightarrow$ (X)	.....
MOV	(X)+, A	AF	1	4	A $\rightarrow$ (X) with auto increment	.....
MOV	dp, A	C4	2	4	A $\rightarrow$ (dp)	.....
MOV	dp+X, A	D4	2	5	A $\rightarrow$ (dp+X)	.....
MOV	!abs, A	C5	3	5	A $\rightarrow$ (abs)	.....
MOV	!abs+X, A	D5	3	6	A $\rightarrow$ (abs+X)	.....
MOV	!abs+Y, A	D6	3	6	A $\rightarrow$ (abs+Y)	.....
MOV	[dp+X], A	C7	2	7	A $\rightarrow$ ((dp+X+1)(dp+X))	.....
MOV	[dp]+Y, A	D7	2	7	A $\rightarrow$ ((dp+1)(dp)+Y)	.....
MOV	dp, X	D8	2	4	X $\rightarrow$ (dp)	.....
MOV	dp+Y, X	D9	2	5	X $\rightarrow$ (dp+Y)	.....
MOV	!abs, X	C9	3	5	X $\rightarrow$ (abs)	.....
MOV	dp, Y	CB	2	4	Y $\rightarrow$ (dp)	.....
MOV	dp+X, Y	DB	2	5	Y $\rightarrow$ (dp+X)	.....
MOV	!abs, Y	CC	3	5	Y $\rightarrow$ (abs)	.....

Table C-6 8-bit Data Transmission Commands, Group 3

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC
MOV	A, X	7D	1	2	A ← X	N.....Z
MOV	A, Y	DD	1	2	A ← Y	N.....Z
MOV	X, A	5D	1	2	X ← A	N.....Z
MOV	Y, A	FD	1	2	Y ← A	N.....Z
MOV	X, SP	9D	1	2	X ← SP	N.....Z
MOV	SP, X	BD	1	2	SP ← X	.....
MOV	dp<d>, dp<s>	FA	3	5	(dp<d>) ← (dp<s>)	.....
MOV	dp, #imm	8F	3	5	(dp) ← imm	.....

(NCL PG 38)

Table C-7 8-bit Arithmetic Operation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHI <sub>Z</sub> C
ADC	A, #imm	88	2	2	A $\leftarrow$ A + imm + C	NV..H.ZC
ADC	A, (X)	86	1	3	A $\leftarrow$ A + (X) + C	NV..H.ZC
ADC	A, dp	84	2	3	A $\leftarrow$ A + (dp) + C	NV..H.ZC
ADC	A, dp+X	94	2	4	A $\leftarrow$ A + (dp+X) + C	NV..H.ZC
ADC	A, !abs	85	3	4	A $\leftarrow$ A + (abs) + C	NV..H.ZC
ADC	A, !abs+X	95	3	5	A $\leftarrow$ A + (abs + X) + C	NV..H.ZC
ADC	A, !abs+Y	96	3	5	A $\leftarrow$ A + (abs + Y) + C	NV..H.ZC
ADC	A, [dp+X]	87	2	6	A $\leftarrow$ A + (dp+X+1)(dp+X) + C	NV..H.ZC
ADC	A, [dp]+Y	97	2	6	A $\leftarrow$ A + ((dp+1)(dp)+Y) + C	NV..H.ZC
ADC	(X), (Y)	99	1	5	(X) $\leftarrow$ (X) + (Y) + C	NV..H.ZC
ADC	dp<d>, dp<s>	89	3	6	(dp<d>) $\leftarrow$ (dp<d>) + (dp<s>) + C	NV..H.ZC
ADC	dp, #imm	98	3	5	(dp) $\leftarrow$ (dp) + imm + C	NV..H.ZC
SBC	A, #imm	A8	2	2	A $\leftarrow$ A - imm - C	NV..H.ZC
SBC	A, (X)	A6	1	3	A $\leftarrow$ A - (X) - C	NV..H.ZC
SBC	A, dp	A4	2	3	A $\leftarrow$ A - (dp) - C	NV..H.ZC
SBC	A, dp+X	B4	2	4	A $\leftarrow$ A - (dp+X) - C	NV..H.ZC
SBC	A, !abs	A5	3	4	A $\leftarrow$ A - (abs) - C	NV..H.ZC
SBC	A, !abs+X	B5	3	5	A $\leftarrow$ A - (abs + X) - C	NV..H.ZC
SBC	A, !abs+Y	B6	3	5	A $\leftarrow$ A - (abs + Y) - C	NV..H.ZC
SBC	A, [dp+X]	A7	2	6	A $\leftarrow$ A - (dp+X+1)(dp+X) - C	NV..H.ZC
SBC	A, [dp]+Y	B7	2	6	A $\leftarrow$ A - ((dp+1)(dp)+Y) - C	NV..H.ZC
SBC	(X), (Y)	B9	1	5	(X) $\leftarrow$ (X) - (Y) - C	NV..H.ZC
SBC	dp<d>, dp<s>	A9	3	6	(dp<d>) $\leftarrow$ (dp<d>) - (dp<s>) - C	NV..H.ZC
SBC	dp, #imm	B8	3	5	(dp) $\leftarrow$ (dp) - imm - C	NV..H.ZC
CMP	A, #imm	68	2	2	A - imm	N.....ZC
CMP	A, (X)	66	1	3	A - (X)	N.....ZC
CMP	A, dp	64	2	3	A - (dp)	N.....ZC
CMP	A, dp+X	74	2	4	A - (dp+X)	N.....ZC
CMP	A, !abs	65	3	4	A - (abs)	N.....ZC
CMP	A, !abs+X	75	3	5	A - (abs+X)	N.....ZC
CMP	A, !abs+Y	76	3	5	A - (abs+Y)	N.....ZC
CMP	A, [dp+X]	67	2	6	A - ((dp+X+1)(dp+X))	N.....ZC
CMP	A, [dp]+Y	77	2	6	A - ((dp+1)(dp)+Y)	N.....ZC
CMP	(X), (Y)	79	1	5	(X) - (Y)	N.....ZC
CMP	dp<d>, dp<s>	69	3	6	(dp<d>) - (dp<s>)	N.....ZC
CMP	dp, #imm	78	3	5	(dp) - imm	N.....ZC
CMP	X, #imm	C8	2	2	X - imm	N.....ZC
CMP	X, dp	3E	2	3	X - (dp)	N.....ZC
CMP	X, !abs	1E	3	4	X - (abs)	N.....ZC
CMP	Y, #imm	AD	2	2	Y - imm	N.....ZC
CMP	Y, dp	7E	2	3	Y - (dp)	N.....ZC
CMP	Y, !abs	5E	3	4	Y - (abs)	N.....ZC

(NCL PG 39)

Table C-8 8-bit Logic Operation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC
AND	A, #imm	28	2	2	A $\leftarrow$ A AND imm	N.....Z.
AND	A, (X)	26	1	3	A $\leftarrow$ A AND (X)	N.....Z.
AND	A, dp	24	2	3	A $\leftarrow$ A AND (dp)	N.....Z.
AND	A, dp+X	34	2	4	A $\leftarrow$ A AND (dp+X)	N.....Z.
AND	A, !abs	25	3	4	A $\leftarrow$ A AND (abs)	N.....Z.
AND	A, !abs+X	35	3	5	A $\leftarrow$ A AND (abs+X)	N.....Z.
AND	A, !abs+Y	36	3	5	A $\leftarrow$ A AND (abs+Y)	N.....Z.
AND	A, [dp+X]	27	2	6	A $\leftarrow$ A AND ((dp+X+1)(dp+X))	N.....Z.
AND	A, [dp]+Y	37	2	6	A $\leftarrow$ A AND ((dp+1)(dp)+Y)	N.....Z.
AND	(X), (Y)	39	1	5	(X) $\leftarrow$ (X) AND (Y)	N.....Z.
AND	dp<d>, dp<s>	29	3	6	(dp<d>) $\leftarrow$ (dp<d>) AND (dp<s>)	N.....Z.
AND	dp, #imm	38	3	5	(dp) $\leftarrow$ (dp) AND imm	N.....Z.
OR	A, #imm	08	2	2	A $\leftarrow$ A OR imm	N.....Z.
OR	A, (X)	06	1	3	A $\leftarrow$ A OR (X)	N.....Z.
OR	A, dp	04	2	3	A $\leftarrow$ A OR (dp)	N.....Z.
OR	A, dp+X	14	2	4	A $\leftarrow$ A OR (dp+X)	N.....Z.
OR	A, !abs	05	3	4	A $\leftarrow$ A OR (abs)	N.....Z.
OR	A, !abs+X	15	3	5	A $\leftarrow$ A OR (abs+X)	N.....Z.
OR	A, !abs+Y	16	3	5	A $\leftarrow$ A OR (abs+Y)	N.....Z.
OR	A, [dp+X]	07	2	6	A $\leftarrow$ A OR ((dp+X+1)(dp+X))	N.....Z.
OR	A, [dp]+Y	17	2	6	A $\leftarrow$ A OR ((dp+1)(dp)+Y)	N.....Z.
OR	(X), (Y)	19	1	5	(X) $\leftarrow$ (X) OR (Y)	N.....Z.
OR	dp<d>, dp<s>	09	3	6	(dp<d>) $\leftarrow$ (dp<d>) OR (dp<s>)	N.....Z.
OR	dp, #imm	18	3	5	(dp) $\leftarrow$ (dp) OR imm	N.....Z.
EOR	A, #imm	48	2	2	A $\leftarrow$ A EOR imm	N.....Z.
EOR	A, (X)	46	1	3	A $\leftarrow$ A EOR (X)	N.....Z.
EOR	A, dp	44	2	3	A $\leftarrow$ A EOR (dp)	N.....Z.
EOR	A, dp+X	54	2	4	A $\leftarrow$ A EOR (dp+X)	N.....Z.
EOR	A, !abs	45	3	4	A $\leftarrow$ A EOR (abs)	N.....Z.
EOR	A, !abs+X	55	3	5	A $\leftarrow$ A EOR (abs+X)	N.....Z.
EOR	A, !abs+Y	56	3	5	A $\leftarrow$ A EOR (abs+Y)	N.....Z.
EOR	A, [dp+X]	47	2	6	A $\leftarrow$ A EOR ((dp+X+1)(dp+X))	N.....Z.
EOR	A, [dp]+Y	57	2	6	A $\leftarrow$ A EOR ((dp+1)(dp)+Y)	N.....Z.
EOR	(X), (Y)	59	1	5	(X) $\leftarrow$ (X) EOR (Y)	N.....Z.
EOR	dp<d>, dp<s>	49	3	6	(dp<d>) $\leftarrow$ (dp<d>) EOR(dp<s>)	N.....Z.
EOR	dp, #imm	58	3	5	(dp) $\leftarrow$ (dp) EOR imm	N.....Z.

(NCL PG 40)

Table C-9 Addition and Subtraction Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC
INC	A	BC	1	2	++ A	N.....Z.
INC	dp	AB	2	4	++ (dp)	N.....Z.
INC	dp+X	BB	2	5	++ (dp+X)	N.....Z.
INC	!abs	AC	3	5	++ (abs)	N.....Z.
INC	X	3D	1	2	++ X	N.....Z.
INC	Y	FC	1	2	++ Y	N.....Z.
DEC	A	9C	1	2	-- A	N.....Z.
DEC	dp	8B	2	4	-- (dp)	N.....Z.
DEC	dp+X	9B	2	5	-- (dp+X)	N.....Z.
DEC	!abs	8C	3	5	-- (abs)	N.....Z.
DEC	X	1D	1	2	-- X	N.....Z.
DEC	Y	DC	1	2	-- Y	N.....Z.

Table C-10 Shift Rotation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC	
ASL	A	1C	1	2	C << A	<<0	N.....ZC
ASL	dp	0B	2	4	C << (dp)	<<0	N.....ZC
ASL	dp+X	1B	2	5	C << (dp+X)	<<0	N.....ZC
ASL	!abs	0C	3	5	C << (abs)	<<0	N.....ZC
LSR	A	5C	1	2	C << A	<<C	N.....ZC
LSR	dp	4B	2	4	C << (dp)	<<C	N.....ZC
LSR	dp+X	5B	2	5	C << (dp+X)	<<C	N.....ZC
LSR	!abs	4C	3	5	C << (abs)	<<C	N.....ZC
ROL	A	3C	1	2	C << A	<<C	N.....ZC
ROL	dp	2B	2	4	C << (dp)	<<C	N.....ZC
ROL	dp+X	3B	2	5	C << (dp+X)	<<C	N.....ZC
ROL	!abs	2C	3	5	C << (abs)	<<C	N.....ZC
ROR	A	7C	1	2	C << A	<<C	N.....ZC
ROR	dp	6B	2	4	C << (dp)	<<C	N.....ZC
ROR	dp+X	7B	2	5	C << (dp+X)	<<C	N.....ZC
ROR	!abs	6C	3	5	C << (abs)	<<C	N.....ZC
XCN	A	9F	1	5	A (7 ~ 4)↔ A (3 ~ 0)	N.....Z.	

Table C-11 16-bit Data Transmission Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
MOVW	YA,dp	BA	2	5	YA $\leftarrow$ (dp+1)(dp)	N.....Z.
MOVW	dp, YA	DA	2	4	(dp+1)(dp) $\leftarrow$ YA	.....

Table C-12 16-bit Operation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
INCW	dp	3A	2	6	increment dp memory pair	N.....Z.
DECW	dp	1A	2	6	decrement dp memory pair	N.....Z.
ADDW	YA, dp	7A	2	5	YA $\leftarrow$ YA+ (dp+1)(dp)	NV..H.ZC
SUBW	YA, dp	9A	2	5	YA $\leftarrow$ YA- (dp+1)(dp)	NV..H.ZC
CMPW	YA, dp	5A	2	4	YA- (dp+1)(dp)	N.....ZC

Table C-13 Multiplication and Division Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
MUL	YA	CF	1	9	YA(16bits) $\leftarrow$ Y * A	N.....Z.
DIV	YA, X	9E	1	12	Q:A R:Y $\leftarrow$ YA / X	NV..H.Z.

Table C-14 Decimal Compensation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
DAA	A	DF	1	3	decimal adjust for addition	N.....ZC
DAS	A	BE	1	3	decimal adjust for subtraction	N.....ZC

Table C-15 Branching Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
BRA	rel	2F	2	4	branch always	.....
BEQ	rel	F0	2	2/4	branch on Z=1	.....
BNE	rel	D0	2	2/4	branch on Z=0	.....
BCS	rel	B0	2	2/4	branch on C=1	.....
BCC	rel	90	2	2/4	branch on C=0	.....
BVS	rel	70	2	2/4	branch on V=1	.....
BVC	rel	50	2	2/4	branch on V=0	.....
BMI	rel	30	2	2/4	branch on N=1	.....
BPL	rel	10	2	2/4	branch on N=0	.....
BBS	dp,bit, rel	x3	3	5/7	branch on dp, bit=1	.....
BBC	dp,bit, rel	y3	3	5/7	branch on dp, bit=0	.....
CBNE	dp,rel	2E	3	5/7	compare A with (dp) then BNE	.....
CBNE	dp+X, rel	DE	3	6/8	compare A with (dp+X) then BNE	.....
DBNZ	dp,rel	6E	3	5/7	decrement memory (dp) then JNZ	.....
DBNZ	Y,rel	FE	2	4/6	decrement Y then JNZ	.....
JMP	!abs	5F	3	3	jump to new location	.....
JMP	[!abs+X]	1F	3	6	PC $\leftarrow$ (abs+X+1)(abs+X)	.....

(NCL PG 42)

Table C-16 Subroutine Call, Return Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
CALL	!abs	3F	3	8	subroutine call	.....
PCALL	upage	4F	2	6	upage call	.....
TCALL	n	n1	1	8	table call	.....
BRK		0F	1	8	software interrupt	..1.0..
RET		6F	1	5	return from subroutine	.....
RETI		7F	1	6	return from interrupt	(Restored)

Table C-17 Stack Operation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
PUSH	A	2D	1	4	push A to stack	.....
PUSH	X	4D	1	4	push X to stack	.....
PUSH	Y	6D	1	4	push Y to stack	.....
PUSH	PSW	0D	1	4	push PSW to stack	.....
POP	A	AE	1	4	pop A from stack	.....
POP	X	CE	1	4	pop X from stack	.....
POP	Y	EE	1	4	pop Y from stack	.....
POP	PSW	8E	1	4	pop PSW from stack	(Restored)

Table C-18 Bit Operation Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHIZC
SET1	dip. bit	x2	2	4	set direct page bit	.....
CLR1	dip. bit	y2	2	4	clear direct page bit	.....
TSET1	!abs	0E	3	6	test and set bits with A	N.....Z.
TCLR1	!abs	4E	3	6	test and clear bits with A	N.....Z.
AND1	C, mem. bit	4A	3	4	C ← C AND (mem. bit)	.....C
AND1	C, /mem. bit	6A	3	4	C ← C AND (mem. bit)	.....C
OR1	C, mem. bit	0A	3	5	C ← C OR (mem. bit)	.....C
OR1	C, /mem. bit	2A	3	5	C ← C OR (mem. bit)	.....C
EOR1	C, mem. bit	8A	3	5	C ← C EOR (mem. bit)	.....C
NOT1	mem. bit	EA	3	5	complement (mem. bit)	.....
MOV1	C, mem. bit	AA	3	4	C ← (mem. bit)	.....C
MOV1	mem. bit, C	CA	3	6	C → (mem. bit)	.....

Table C-19 Program Status Flag Operation Commands

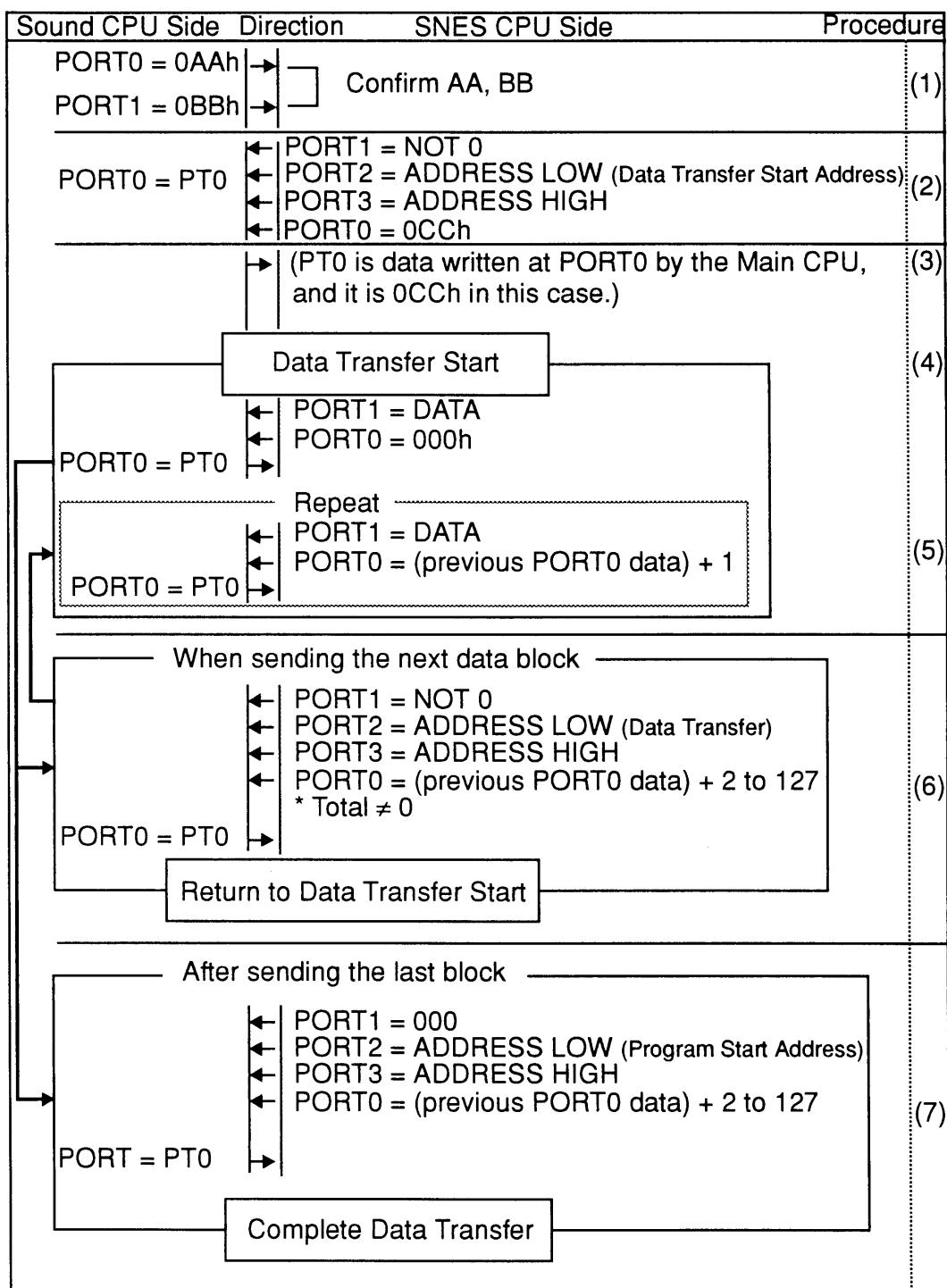
Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC
CLRC		60	1	2	clear carry flag	.....0
SETC		80	1	2	set carry flag	.....1
NOTC		ED	1	3	complement carry flag	.....C
CLRV		E0	1	2	clear V and II	.0..0...
CLRP		20	1	2	clear direct page flag	..0....
SETP		40	1	2	set direct page flag	..1....
EI		A0	1	3	set interrupt enable flag	.....1.
DI		C0	1	3	clear interrupt enable flag	.....0.

Table C-20 Other Commands

Mnemonic	Operand	Code	Bytes	Cycles	Operation	NVPBHZC
NOP		00	1	2	no operation	.....
SLEEP		EF	1	3	standby SLEEP mode	.....
STOP		FF	1	3	standby STOP mode	.....

## Appendix D. Data Transfer Procedure

### D.1 Data Transfer Procedure



(NCL PG 45)

## D.2 Data Transfer Instruction

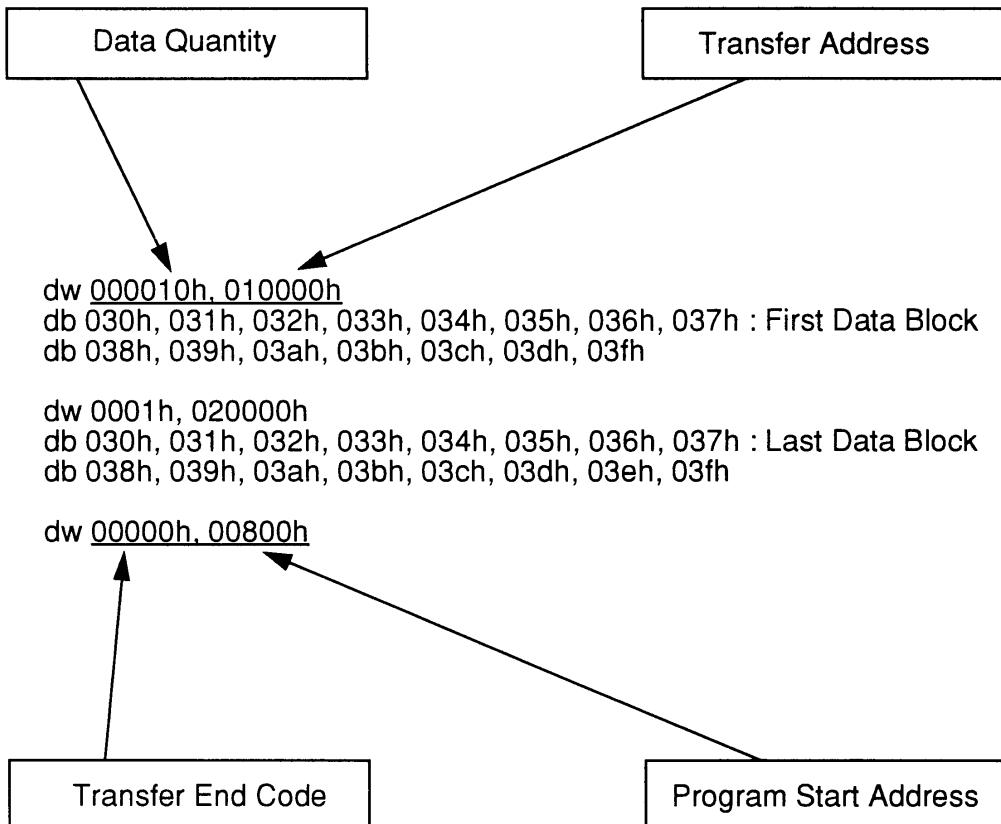
The transfer program on the Sound CPU is stored in the internal ROM called IPL ROM. This ROM functions after reset. The program ROM functions using the Main CPU and PORT 0 through 3.

- (5) The sound CPU writes AAh to PORT 1. The Main CPU reads and confirms data at PORT 0 and 1.
- (6) The Main CPU writes Start Address to PORT 2 and 3. After storing Port 2 and 3, store any number except 0 to PORT 1 and store CCh to PORT 0.
- (7) The sound CPU checks PORT 0 for CCh and writes CCh to PORT 0.
- (8) Start data transfer. The Main CPU writes first data to PORT 1 and writes 00h to PORT 0. The Sound CPU reads data from PORT 1 and writes 00h to PORT 0.
- (9) The Main CPU checks PORT 0, writes next data to PORT 1, and increments of PORT 0. This is the data transfer procedure. The data block contains the quantity of data to be transferred.
- (10) When PORT 0 stops incrementing, proceed to the next step. The value that the SNES CPU writes to PORT 0 must not be 00h. Write any value but 00h to PORT 1. The Sound CPU writes the same value to PORT 0 and then returns to step (4).
- (11) After sending all data blocks using steps (4) through (6), the data transfer is completed. Program Start Address is stored to PORT 2 and 3, Write 00h to PORT 1.

### D.3 Data Block Organization

Data is divided into several blocks having consecutive addresses. The quantity of data (2 byte) and address (2 byte) are stored in front of data.

Data Block Example:



## D.4 Sound Boot Loader V1.1

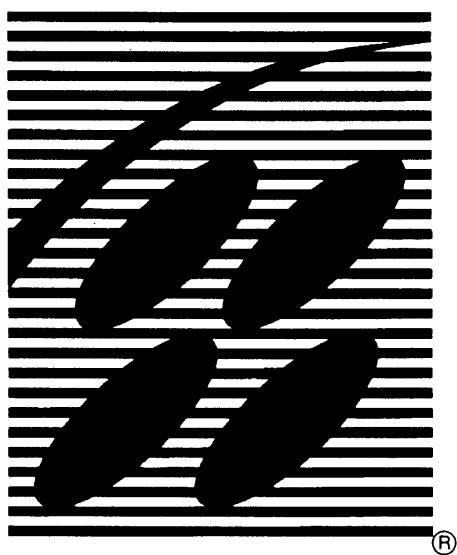
	glb	Boot_APU	
APU_port	0	equ 02140h	
APU_port	1	equ 02141h	
APU_port	2	equ 02142h	
APU_port	3	equ 02143h	
address		equ 00000h	Input Sound ROM Start Address (3 byte) in 0 page and call
:			"Boot_APU" from main routine.
;			
Boot_APU		code	
		php	
		rep #00110000b	
		idx16	:sony news
		mem16	:sony news
		on16i	:SNES Emulator
		on16a	:SNES Emulator
		1dy #0	
boot_initial		1da #0bbaah	
		cmp !APU_PORT0	;m16
		bne boot_initial	
		sep #00100000b	
		mem8	:sony news
		off16a	:SNES Emulator
		1da #0cch	
boot_repeat		bra boot_entry1	
		1da [address],y	
		iny	
		xba	
		1da #0	
boot_loop		bra boot_entry2	
		xba	
		1da [address],y	
		iny	
		xba	
boot_wait1		1da cmp !APU_PORT0	
		bne boot_wait1	
		inc a	
boot_entry2		rep #00100000b	
		sta !APU_PORT0	;m16
		sep #00100000b	
		dex	
boot_wait2		bne boot_loop	
		cmp !APU_PORT0	
		bne boot_wait2	
boot_zero		adc #3	
boot_entry1		beq boot_zero	
		pha #00100000b	>
		rep [address],y	;m16
		1da iny	
		iny tax	
		1da [address],y	;m16
		iny	
		iny sta !APU_PORT2	
		sep #00100000b	;m16
		cpx #1	
		1da #0	
		rol a	
		sta !APU_PORT1	
		adc #07fh	
		pla	
		sta !APU_PORT0	
boot_wait3		cmp !APU_PORT0	
		bne boot_wait3	
		bvs boot_repeat	
		plp	
		rts	
		end	

(NCL PG 48)



***DEVELOPMENT  
MANUAL***

***BOOK II***



**"Confidential"**

This document contains confidential and proprietary information of Nintendo and is also protected under the copyright laws of the United States and foreign countries. No part of this document may be released, distributed, transmitted or reproduced in any form or by any electronic or mechanical means, including information storage and retrieval systems, without permission in writing from Nintendo.

© 1993, 1994, 1995 Nintendo

The terms Sony and Sony NEWS are registered trademarks of Sony Corporation. ® and ™ are registered trademarks of Nintendo.

## ***Table of Contents***

### ***BOOK II***

SUBJECT	PAGE
<b>SECTION 1 - SUPER ACCELERATOR (SA-1).....</b>	<b>1-1-1</b>
Super Accelerator System Functions .....	1-1-1
Configuration of SA-1 .....	1-2-1
Super Accelerator Memory Map .....	1-3-1
SA-1 Internal Register Configuration.....	1-4-1
Multi-Processor Processing .....	1-5-1
Character Conversion.....	1-6-1
Arithmetic Function .....	1-7-1
Variable-Length Bit Processing.....	1-8-1
DMA .....	1-9-1
SA-1 Timer .....	1-10-1
<b>SECTION 2 - SUPER FX® .....</b>	<b>2-1-1</b>
Introduction to Super FX .....	2-1-1
GSU Functional Operation .....	2-2-1
Memory Mapping .....	2-3-1
GSU Internal Register Configuration .....	2-4-1
GSU Program Execution.....	2-5-1
Instruction Execution .....	2-6-1
Data Access.....	2-7-1
GSU Special Functions .....	2-8-1
Description of Instructions.....	2-9-1
<b>SECTION 3 - DSP/DSP1.....</b>	<b>3-1-1</b>
Introduction to DSP1 .....	3-1-1
Command Summary .....	3-2-1
Parameter Data Type.....	3-3-1
Use of DSP1.....	3-4-1
Description of DSP1 Commands .....	3-5-1
Math Functions and Equations.....	3-6-1

**TABLE of CONTENTS**

*Table of Contents (Continued)*

<b>SUBJECT</b>	<b>PAGE</b>
<b>SECTION 4 - ACCESSORIES .....</b>	4-1-1
The Super NES Super Scope® System.....	4-1-1
Principles of the Super NES Super Scope .....	4-2-1
Super NES Super Scope Functional Operation .....	4-3-1
Super NES Super Scope Receiver Functions.....	4-4-1
Graphics .....	4-5-1
Super NES Mouse Specifications .....	4-6-1
Using the Standard BIOS.....	4-7-1
Programming Cautions .....	4-8-1
MultiPlayer 5 Specifications.....	4-9-1
MultiPlayer 5 Supplied BIOS .....	4-10-1

**SUPPLEMENTAL INFORMATION**

Super NES Parts List .....	1
Game Content Guidelines .....	3
Guidelines Concerning Commercialism and Promotion of Licensee	
Products or Services in Nintendo Licensed Games .....	5
Super NES Video Timing Information .....	10

**INDEX**

**BULLETINS**

## *List of Figures*

### *BOOK II*

<u>TITLE</u>	<u>FIGURE NUMBER</u>	<u>PAGE</u>
Super Accelerator System Configuration .....	1-1-1	1-1-3
SAS Bus Image .....	1-1-2	1-1-4
SA-1 Block Diagram.....	1-2-1	1-2-1
Bitmap Register Files 0~7 .....	1-4-1	1-4-24
Bitmap Register Files 8~F .....	1-4-2	1-4-25
Accelerator Mode.....	1-5-1	1-5-6
Parallel Processing Mode.....	1-5-2	1-5-7
Mixed Processing Mode .....	1-5-3	1-5-8
Character Conversion 1.....	1-6-1	1-6-1
Character Conversion 2.....	1-6-2	1-6-2
Compressed Bitmap Data .....	1-6-3	1-6-3
Bitmap Image Projection .....	1-6-4	1-6-3
Bitmap Data Expansion .....	1-6-5	1-6-5
Memory Addresses for the Bitmap Area .....	1-6-6	1-6-6
Character Conversion Buffers.....	1-6-7	1-6-7
Fixed Mode Process Flow Diagram.....	1-8-1	1-8-2
Auto-increment Mode Process Flow Diagram .....	1-8-2	1-8-3
Barrel Shift Process.....	1-8-3	1-8-5
Normal DMA .....	1-9-1	1-9-1
Character Conversion DMA .....	1-9-2	1-9-1
Super FX System Configuration.....	2-1-1	2-1-3
Game Pak ROM/RAM Bus Diagram .....	2-1-2	2-1-4
GSU Functional Block Diagram.....	2-2-1	2-2-1
Super NES CPU Memory Map.....	2-3-1	2-3-2
Super FX Memory Map.....	2-3-2	2-3-4
Example of General Register .....	2-4-1	2-4-2
128 Dot High BG Character Array .....	2-8-1	2-8-2
160 Dot High BG Character Array .....	2-8-2	2-8-2
192 Dot High BG Character Array .....	2-8-3	2-8-2
OBJ Character Array.....	2-8-4	2-8-3
Plot Operations Assigned by CMODE .....	2-8-5	2-8-13
System Block Diagram (DSP1) .....	3-1-1	3-1-2
Super NES CPU and DSP1 Communications .....	3-1-2	3-1-3
DSP1 Command Execution .....	3-1-3	3-1-3
Mode 20/DSP Memory Map.....	3-1-4	3-1-4
Mode 21/DSP Memory Map.....	3-1-5	3-1-5
Super NES/DSP1 Memory Mapping (Mode 21).....	3-4-1	3-4-1
DSP1 Status Register Configuration.....	3-4-2	3-4-2

*List of Figures (Continued)*

<b>TITLE</b>	<b>FIGURE NUMBER</b>	<b>PAGE</b>
DSP1 Operations Flow Diagram .....	3-4-3 .....	3-4-3
Super NES CPU/DSP1 Operational Timing.....	3-4-4 .....	3-4-4
Trigonometric Calculation.....	3-5-1 .....	3-5-3
Vector Calculation .....	3-5-2 .....	3-5-4
Vector Size Comparison .....	3-5-3 .....	3-5-6
Vector Absolute Value Calculation .....	3-5-4 .....	3-5-7
Two-Dimensional Coordinate Rotation.....	3-5-5 .....	3-5-8
Examples of Three-Dimensional Rotation.....	3-5-6 .....	3-5-11
Assignment of Projection Parameter .....	3-5-7 .....	3-5-13
Relationship of Sight and Projected Plane.....	3-5-8 .....	3-5-13
Calculation of Raster Data.....	3-5-9 .....	3-5-16
BG Screen and Displayed Area .....	3-5-10 .....	3-5-16
Calculation of Projected Position of Object.....	3-5-11 .....	3-5-18
Projection Image of Object .....	3-5-12 .....	3-5-19
Calculation of Coordinates for the Indicated Point on the Screen.....	3-5-13 .....	3-5-20
Attack Point and Position Indicated on Screen (Side View) .....	3-5-14 .....	3-5-21
Attitude Computation .....	3-5-15 .....	3-5-23
Object Coordinate Rotated on Y Axis .....	3-5-16 .....	3-5-23
Object Coordinate Rotated on X Axis .....	3-5-17 .....	3-5-23
Object Coordinate Rotated on Z Axis.....	3-5-18 .....	3-5-23
Conversion of Global to Objective Coordinates.....	3-5-19 .....	3-5-26
Conversion of Object to Global Coordinates.....	3-5-20 .....	3-5-28
Calculation of Inner Product with Forward Attitude .....	3-5-21 .....	3-5-29
Position of Aircraft and Vector Code .....	3-5-22 .....	3-5-30
Calculation of Rotation Angle After Attitude Change .....	3-5-23 .....	3-5-32
Signal Flow .....	4-1-1 .....	4-1-1
Optical Alignment.....	4-1-2 .....	4-1-2
Virtual Screen Alignment .....	4-1-3 .....	4-1-2
Address and Bit Assignments .....	4-1-4 .....	4-1-5
Picture Tube .....	4-2-1 .....	4-2-1
Scanning.....	4-2-2 .....	4-2-2
Area Seen by Super NES Super Scope .....	4-2-3 .....	4-2-3
Vertical Positioning .....	4-2-4 .....	4-2-4
Horizontal Positioning .....	4-2-5 .....	4-2-5
Horizontal/Vertical Counter.....	4-2-6 .....	4-2-6
Super NES Super Scope Block Diagram .....	4-3-1 .....	4-3-2
Super NES Super Scope Flow Diagram .....	4-3-2 .....	4-3-3
Raster Signal .....	4-3-3 .....	4-3-4
Definition of One Bit .....	4-3-4 .....	4-3-5
Output Signal Code.....	4-3-5 .....	4-3-5
Definitions of Codes .....	4-3-6 .....	4-3-6

*List of Figures* (Continued)

<b>TITLE</b>	<b>FIGURE NUMBER</b>	<b>PAGE</b>
Raster Signal Transmission Timing .....	4-3-7 .....	4-3-7
Receiver Block Diagram.....	4-4-1 .....	4-4-1
Operation Flow Diagram .....	4-4-2 .....	4-4-2
Receiver/Transmitter Interface Schematic.....	4-4-3 .....	4-4-3
One Bit Code Detection .....	4-4-4 .....	4-4-4
Cursor Mode Raster Detection Cycle .....	4-4-5 .....	4-4-6
Trigger Mode, Single Shot.....	4-4-6 .....	4-4-7
Trigger Mode, Multiple Shots.....	4-4-7 .....	4-4-8
Noise Flag .....	4-4-8 .....	4-4-9
Null Bit .....	4-4-9 .....	4-4-9
Pause Bit .....	4-4-10 .....	4-4-10
Trigger, Single Shot.....	4-4-11 .....	4-4-11
Trigger, Multiple Shots.....	4-4-12 .....	4-4-12
Optical Color Sensitivity Chart.....	4-5-1 .....	4-5-2
Valid Hyper Mouse Data String .....	4-6-1 .....	4-6-2
Serial Data Read Timing.....	4-6-2 .....	4-6-3
Explanation of Data Strings 2 Bits or Longer.....	4-6-3 .....	4-6-6
Super NES Hyper Mouse Dimensions.....	4-6-4 .....	4-6-7
Standard BIOS, Output Register.....	4-7-1 .....	4-7-3
Examples of Speed Switching Program Subroutine Call .....	4-7-2 .....	4-7-4
MultiPlayer 5 Device Hardware Connections .....	4-9-1 .....	4-9-2
MultiPlayer 5 Read Timing Chart, 5P Mode .....	4-9-2 .....	4-9-5
Data Read Timing for Dissimilar Devices.....	4-9-3 .....	4-9-8
Valid Controller Data String .....	4-9-4 .....	4-9-12
Sample Program Display Format.....	4-10-1 .....	4-10-2

## *List of Tables*

### *BOOK II*

<u><b>TITLE</b></u>	<b>TABLE</b>	<b>NUMBER</b>	<b>PAGE</b>
Types of Interrupts.....	1-5-1 .....	1-5-2	
Interrupt Identification and Clear.....	1-5-2 .....	1-5-2	
Interrupt Mask.....	1-5-3 .....	1-5-3	
Sending and Receiving a Message.....	1-5-4 .....	1-5-3	
Situation Dependant Vectors .....	1-5-5 .....	1-5-4	
Operating Modes and Processing Speeds .....	1-5-6 .....	1-5-9	
Horizontal Size of VRAM (CDMA Register) .....	1-6-1 .....	1-6-6	
Number of Zero Bits in BW-RAM .....	1-6-2 .....	1-6-8	
Character Conversion and Data Format.....	1-6-3 .....	1-6-10	
Arithmetic Operations Settings and Cycles .....	1-7-1 .....	1-7-1	
Amount of Barrel Shift .....	1-8-1 .....	1-8-4	
Source Device Settings .....	1-9-1 .....	1-9-3	
Destination Device Settings.....	1-9-2 .....	1-9-3	
DMA Transmission Speed.....	1-9-3 .....	1-9-4	
Timer Modes and Their Ranges.....	1-10-1 .....	1-10-1	
Timer Interrupts .....	1-10-2 .....	1-10-2	
Registers Listed by Functional Group .....	2-2-1 .....	2-2-3	
Instruction Set .....	2-2-2 .....	2-2-6	
GSU General Registers.....	2-4-1 .....	2-4-1	
GSU Status Register Flags.....	2-4-2 .....	2-4-4	
Screen Height.....	2-4-3 .....	2-4-8	
Color Gradient .....	2-4-4 .....	2-4-8	
Dummy Interrupt Vector Addresses .....	2-5-1 .....	2-5-4	
Dummy Data.....	2-5-2 .....	2-5-5	
Functions of CMODE.....	2-8-1 .....	2-8-9	
DSP1 Command Summary .....	3-2-1 .....	3-2-1	
Parameter Data Type.....	3-3-1 .....	3-3-1	
Signal Bit Definitions .....	4-1-1 .....	4-1-6	
MultiPlayer 5 Switch Function .....	4-9-1 .....	4-9-3	
MultiPlayer 5 Data Format .....	4-9-2 .....	4-9-6	

## ***Chapter 1      Super Accelerator System Functions***

The co-processor installed on the Super Accelerator System (SA-1) is an LSI developed to work with the Super NES CPU and enhance its processing speed, graphics, and arithmetic functions.

### **1.1 SA-1 FEATURES**

#### **1.1.1 CPU CORE**

The SA-1 uses a 16-bit 65C816 processor for its CPU core (SA-1 CPU). It can process the same commands as the Super NES CPU. No new architecture needs be learned and existing programs can be used without modification.

Because the 65C816 is a 16-bit CPU, it efficiently processes 16-bit operations such as X and Y character coordinates.

Due to the commonality of the core CPUs, evaluation of the coprocessor in the middle of game development is quite simple and program modifications are kept to a minimum.

#### **1.1.2 CPU SPEED**

The SA-1 CPU operates at 10.74 MHz, which is four times faster than the normal operating speed of the Super NES CPU.

The SA-1 CPU and the Super NES CPU operate simultaneously, which results in five times greater performance of the Super Accelerator System (SAS) over the current Super NES.

#### **1.1.3 INTERNAL RAM**

The SA-1 has a 2 Kbyte internal work RAM (SA-1 I-RAM). This RAM can be used as the SA-1 CPU's page-zero stack, or as protected memory with a backup battery, when connected to an external battery.

#### **1.1.4 COMMON MEMORY MAPPING**

The Super NES CPU and SA-1 CPU use the same memory mapping. SA-1 programs can be developed with the Super NES Emulator-SE.

Subroutines can be shared by both CPUs, resulting in efficient use of memory.

#### **1.1.5 LARGE-CAPACITY MEMORY**

The SAS has a total capacity of 64 Mbits of ROM and 2 Mbytes of RAM. SRAM is used for I-RAM and back-up/work RAM (BW-RAM), and can be protected with a backup battery.

### 1.1.6 ARITHMETIC HARDWARE

The SA-1 has hardware for high-speed execution of multiplication (16 bits x 16 bits), division (16 bits x 16 bits), and cumulative arithmetic ( $\Sigma$ (16 bits x 16 bits)) operations. This results in high-speed calculation of matrix and 3D arithmetic operations.

### 1.1.7 BIT-MAP DATA OPERATIONS

The SAS allows virtual bitmap VRAM to be set up in the SA-1 CPU's RAM area. The bitmap data in virtual VRAM can be converted to Super NES PPU character format via hardware using DMA functions.

### 1.1.8 VARIABLE-LENGTH BIT DATA OPERATIONS

The SA-1 has a function to read ROM data as 1~16 bit variable-length data, treating ROM data as strings of one-bit data. This allows for high-speed expansion of compressed data.

### 1.1.9 CUSTOM DMA CIRCUIT

The SA-1 has a custom DMA circuit in addition to the Super NES CPU's multi-purpose H-DMA. The DMA circuit performs data transfer between ROM, RAM and SA-1 BW-RAM. During DMA transfer, bitmap-to-character conversion, and sequential operations with the Super NES CPU multi-purpose DMA can be performed.

### 1.1.10 TIMER FUNCTION

The SA-1 has an HV timer synchronized to the Super NES PPU. The HV timer can be used to reference the scan line position on the screen by the SA-1 CPU or to generate HV interrupts. The timer can also be used as a linear timer.

### 1.1.11 INCREASED LEVEL OF SECURITY

The SA-1 is connected between the Super NES CPU and memory (ROM, RAM). The SA-1 ROM is also different from the standard Super NES game pak ROM. This guards against unlicensed products and FD copies.

## 1.2 SYSTEM CONFIGURATION

The following diagram depicts the SAS system configuration.

The SA-1 and memory (game pak ROM and BW-RAM) are installed in the game pak. When desired, data can be protected by connecting a backup battery to BW-RAM or SA-1 I-RAM.

When external RAM is not required, the system can also be configured without BW-RAM.

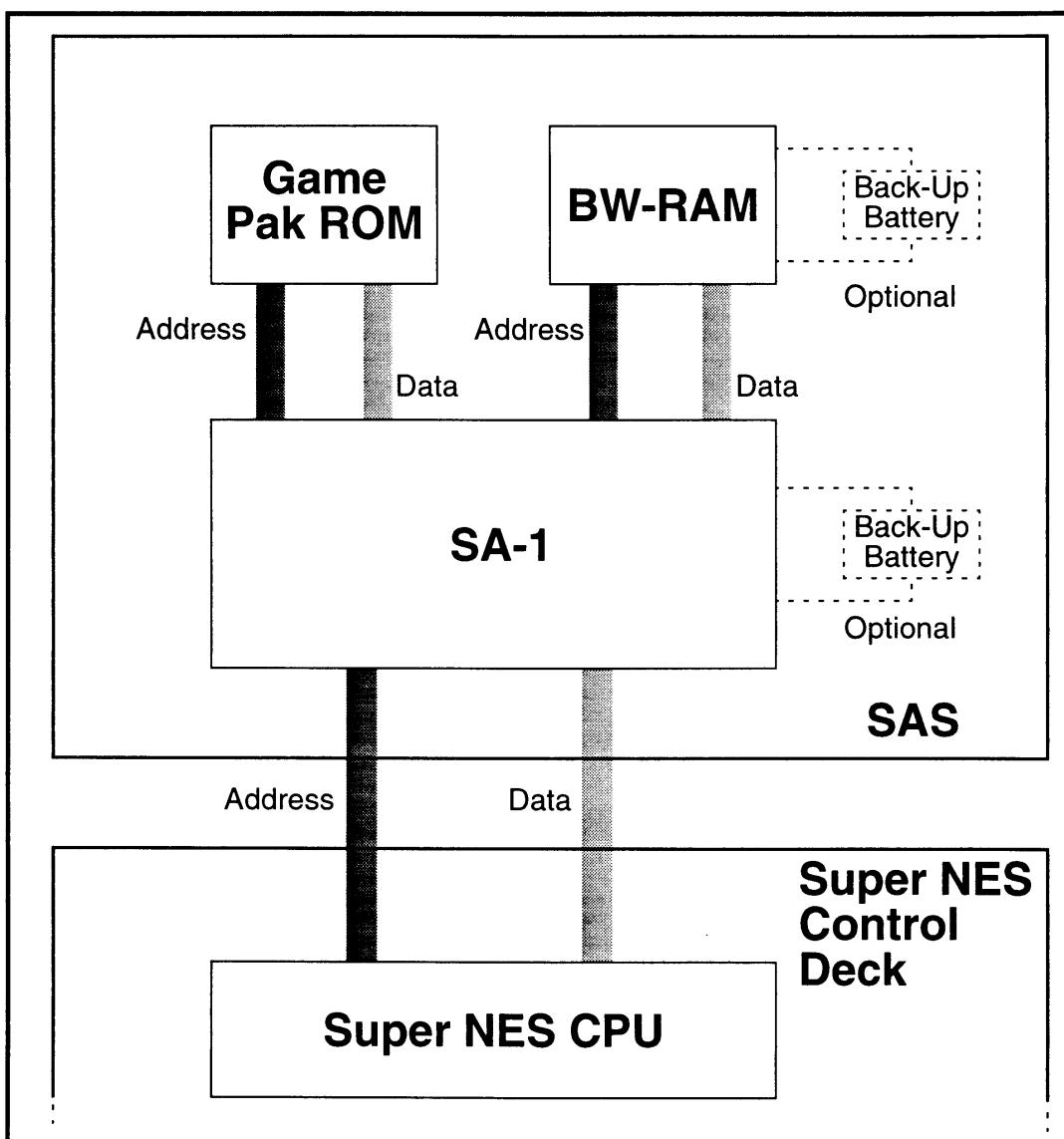


Figure 1-1-1 Super Accelerator System Configuration

### 1.3 BUS IMAGE DIAGRAM

The bus image as seen by the SAS software is depicted below. The SA-1 CPU can access game pak ROM, BW-RAM and I-RAM.

The Super NES CPU can access game pak ROM, BW-RAM, I-RAM, Super NES PPU, Super NES WRAM and Super NES APU.

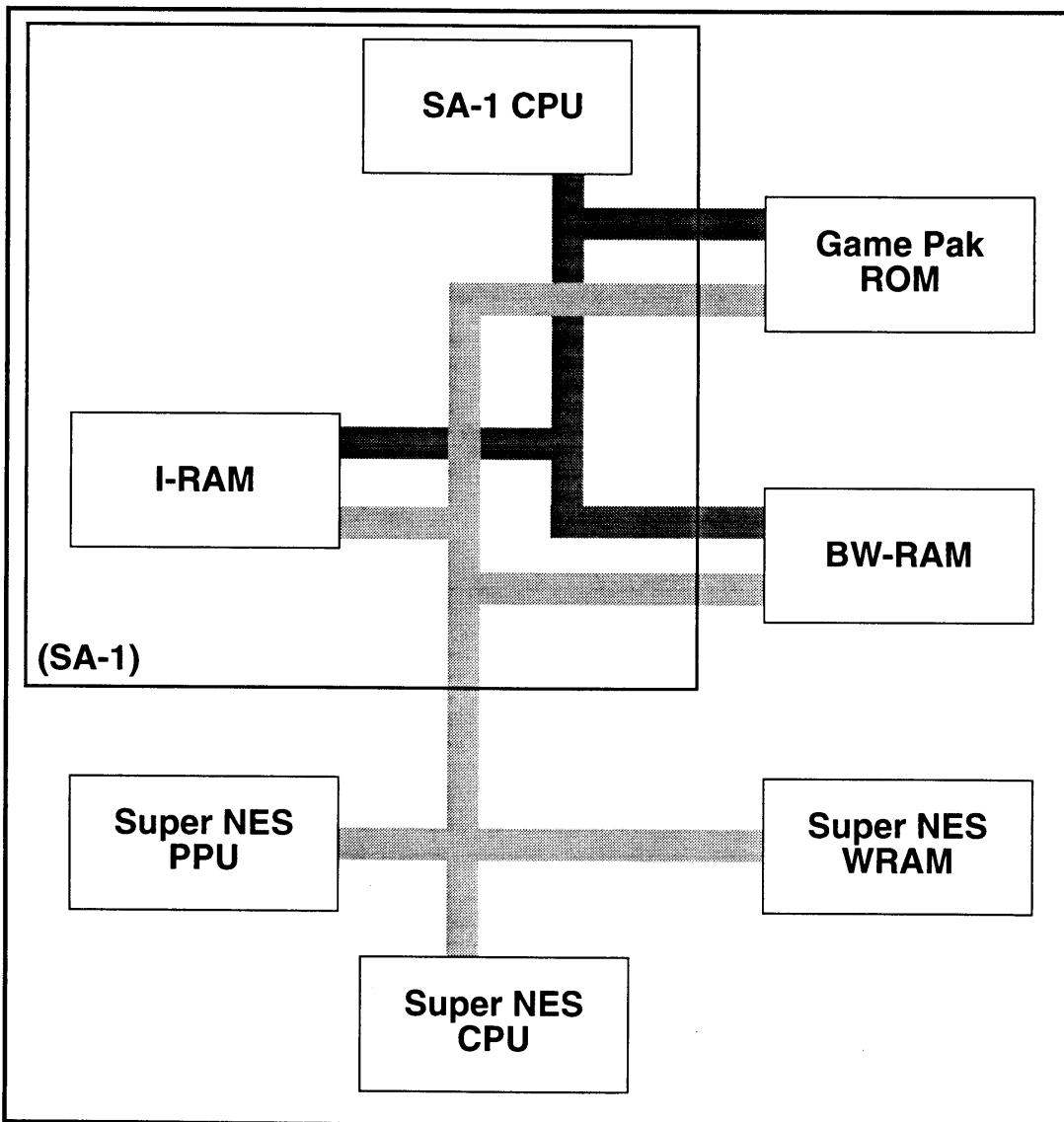


Figure 1-1-2 SAS Bus Image

The two MPUs (Super NES CPU and SA-1 CPU) can freely access memory (game pak ROM, BW-RAM and I-RAM). If the two MPUs try to access the same memory at the same time, one of the MPUs is automatically excluded, and any conflict is averted.

## Chapter 2 Configuration of SA-1

### 2.1 SA-1 FUNCTIONAL DESCRIPTION

The SA-1 is internally comprised of nine components. A block diagram is illustrated below.

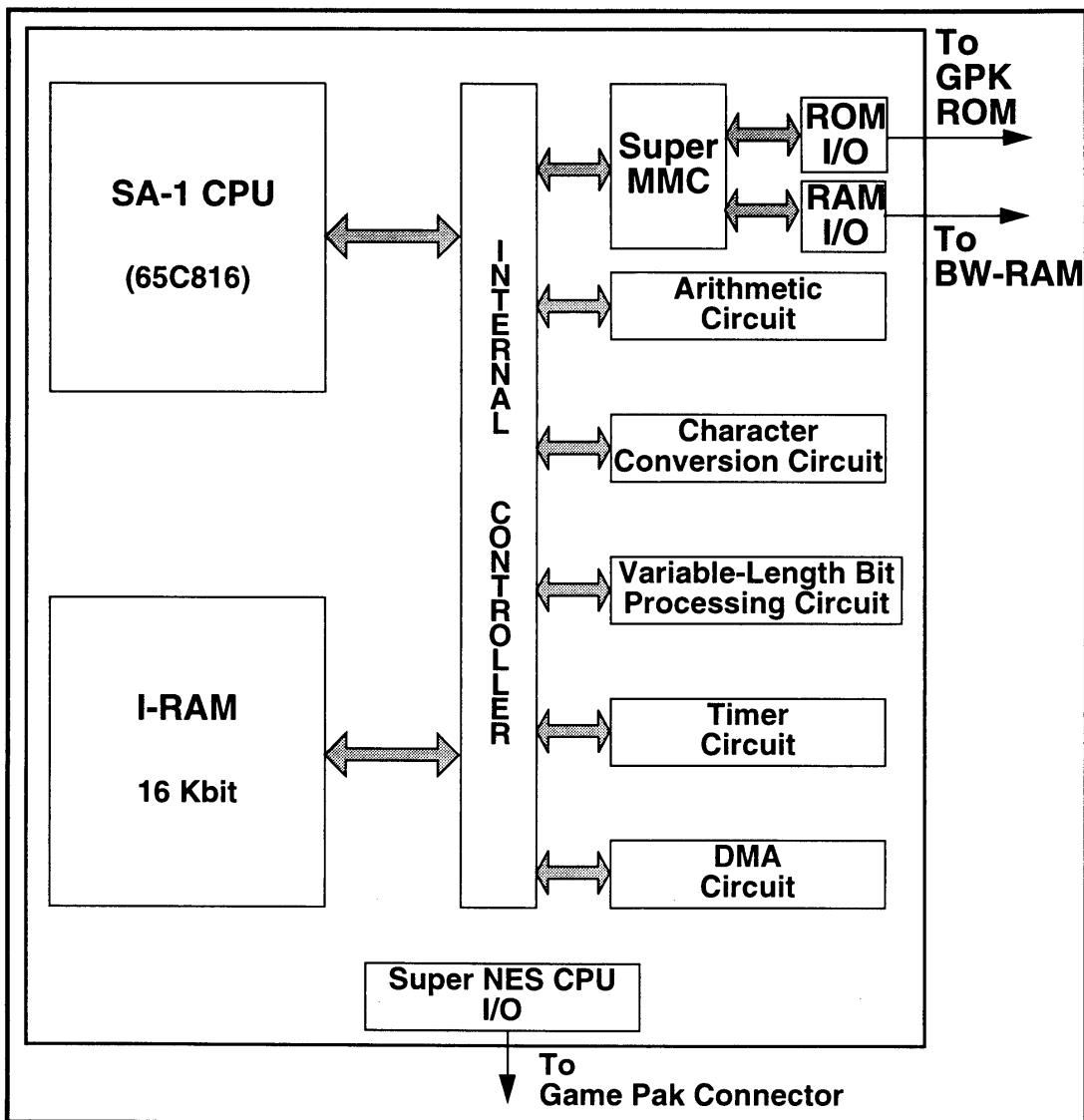


Figure 1-2-1 SA-1 Block Diagram

**2.1.1 SA-1 CPU**

The 65C816 serves as the CPU core. It operates at 10.74 MHz.

**2.1.2 I-RAM**

The I-RAM consists of a 16 Kbit RAM. The SA-1 CPU can access the I-RAM at 10.74 MHz in a no-wait state.

The I-RAM data can be protected by connecting RAM to an external battery.

**2.1.3 SUPER MMC**

The Super MMC performs memory control in a map mode where the ROM capacity exceeds 32 Mbits (Map Mode 22).

The SA-1 has a Super MMC chip emulation circuit.

The Super MMC includes a backup data protection function.

**2.1.4 INTERNAL CONTROLLER**

This controls bus access within the SA-1. It performs collision control functions between Super NES CPU and SA-1 CPU.

**2.1.5 ARITHMETIC CIRCUIT**

The arithmetic circuit hardware performs multiplication, division, and cumulative arithmetic operations.

**2.1.6 CHARACTER CONVERSION CIRCUIT**

The character conversion circuit hardware converts bitmap data to character data format.

**2.1.7 VARIABLE-LENGTH BIT PROCESSING CIRCUIT**

The variable-length bit processing circuit hardware processes data in the game pak ROM as 1~16 bit variable-length data.

**2.1.8 TIMER CIRCUIT**

The SA-1 has a HV timer which is equivalent to the Super NES PPU timer. The timer can also be used as an 18-bit linear timer.

**2.1.9 DMA CIRCUIT**

The DMA circuit transfers data between game pak ROM, BW-RAM and I-RAM.

## 2.2 MEMORY ACCESS

### 2.2.1 GAME PAK ROM ACCESS

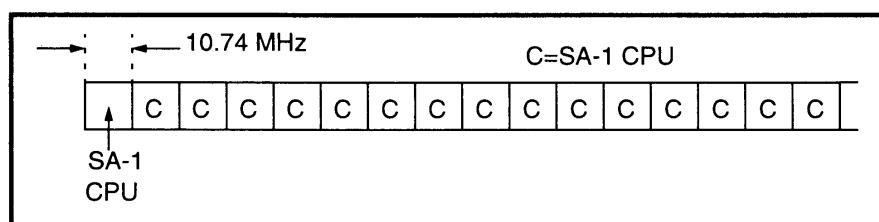
The Super NES CPU and SA-1 CPU share the entire game pak ROM area and can both freely access it. This is known as 2-phase access.

When only the SA-1 CPU uses game pak ROM, the SA-1 CPU operates at 10.74 MHz. During this period the Super NES CPU executes its program on Super NES WRAM.

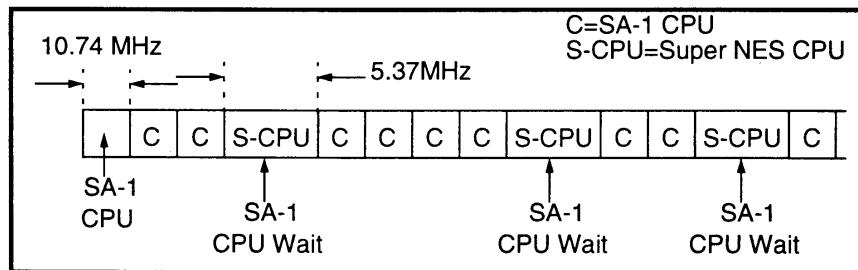
When both the Super NES CPU and SA-1 CPU execute a program on game pak ROM, the SA-1 CPU runs at 5.37 MHz and the Super NES CPU runs at 2.68 MHz.

The SAS cannot utilize the Super NES CPU's high-speed mode (3.58 MHz). It operates at a fixed speed of 2.68 MHz even when only the Super NES CPU uses game pak ROM. This timing is illustrated below for each of these conditions.

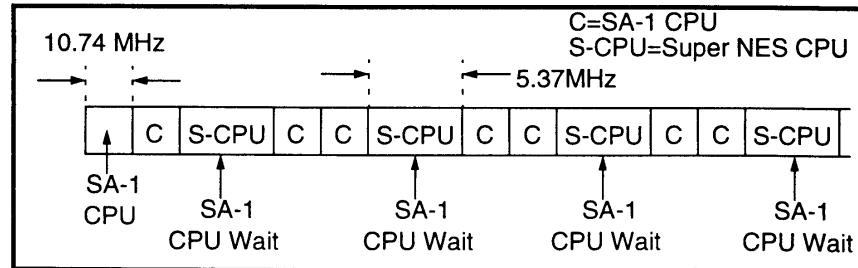
#### 2.2.1.1 ONLY SA-1 CPU USES ROM



#### 2.2.1.2 SUPER NES CPU ACCESSES ROM DURING SA-1 CPU OPERATIONS



#### 2.2.1.3 BOTH PROCESSORS ACCESS ROM (2-PHASE ACCESS)

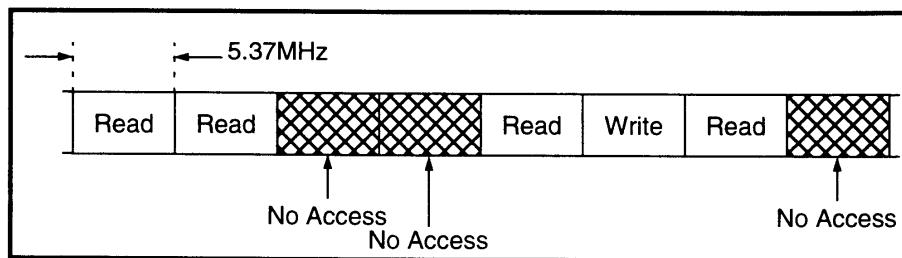


## 2.2.2 BW-RAM ACCESS

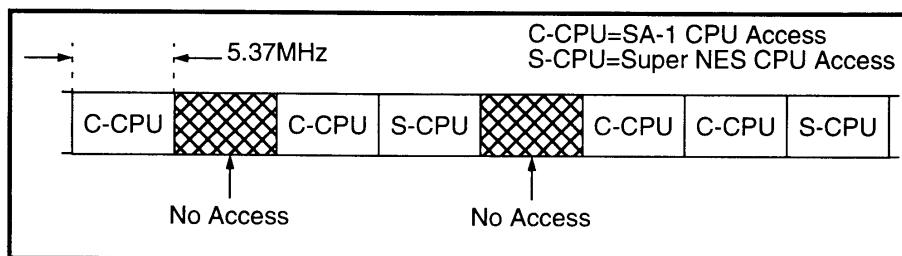
The Super NES CPU and SA-1 CPU share all areas of BW-RAM and can freely access it (two-phase access).

The SA-1 CPU accesses BW-RAM at 5.37 MHz and the Super NES CPU accesses BW-RAM at 2.68 MHz.

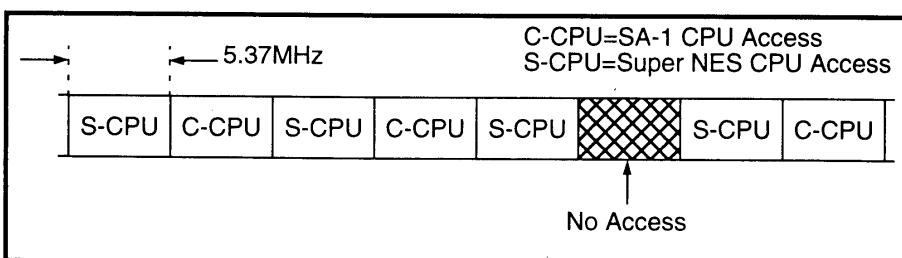
### 2.2.2.1 ONLY SA-1 CPU USES BW-RAM



### 2.2.2.2 SUPER NES CPU ACCESSES BW-RAM DURING SA-1 CPU OPERATIONS



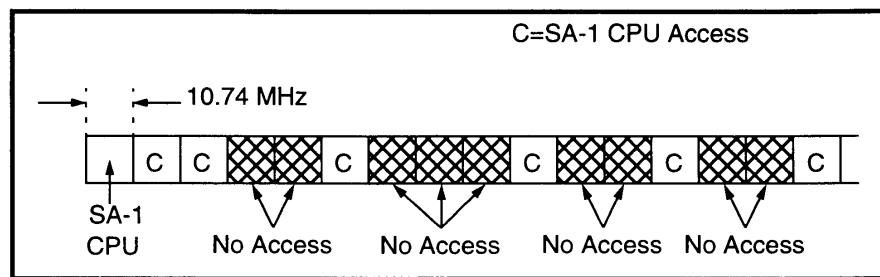
### 2.2.2.3 BOTH PROCESSORS ACCESS BW-RAM (2-PHASE ACCESS)



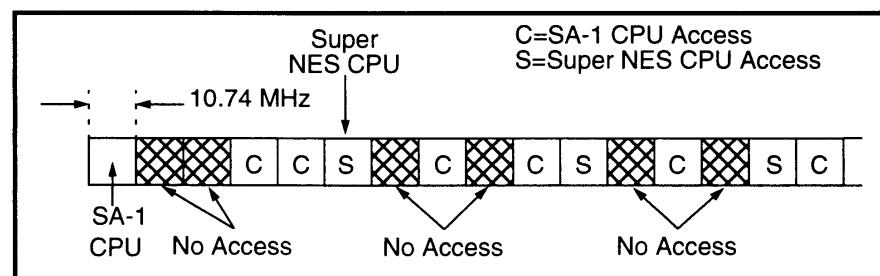
### 2.2.3 SA-1 I-RAM ACCESS

The Super NES CPU and SA-1 CPU can both access all areas of SA-1 I-RAM at any time.

#### 2.2.3.1 ONLY THE SA-1 CPU ACCESSES I-RAM

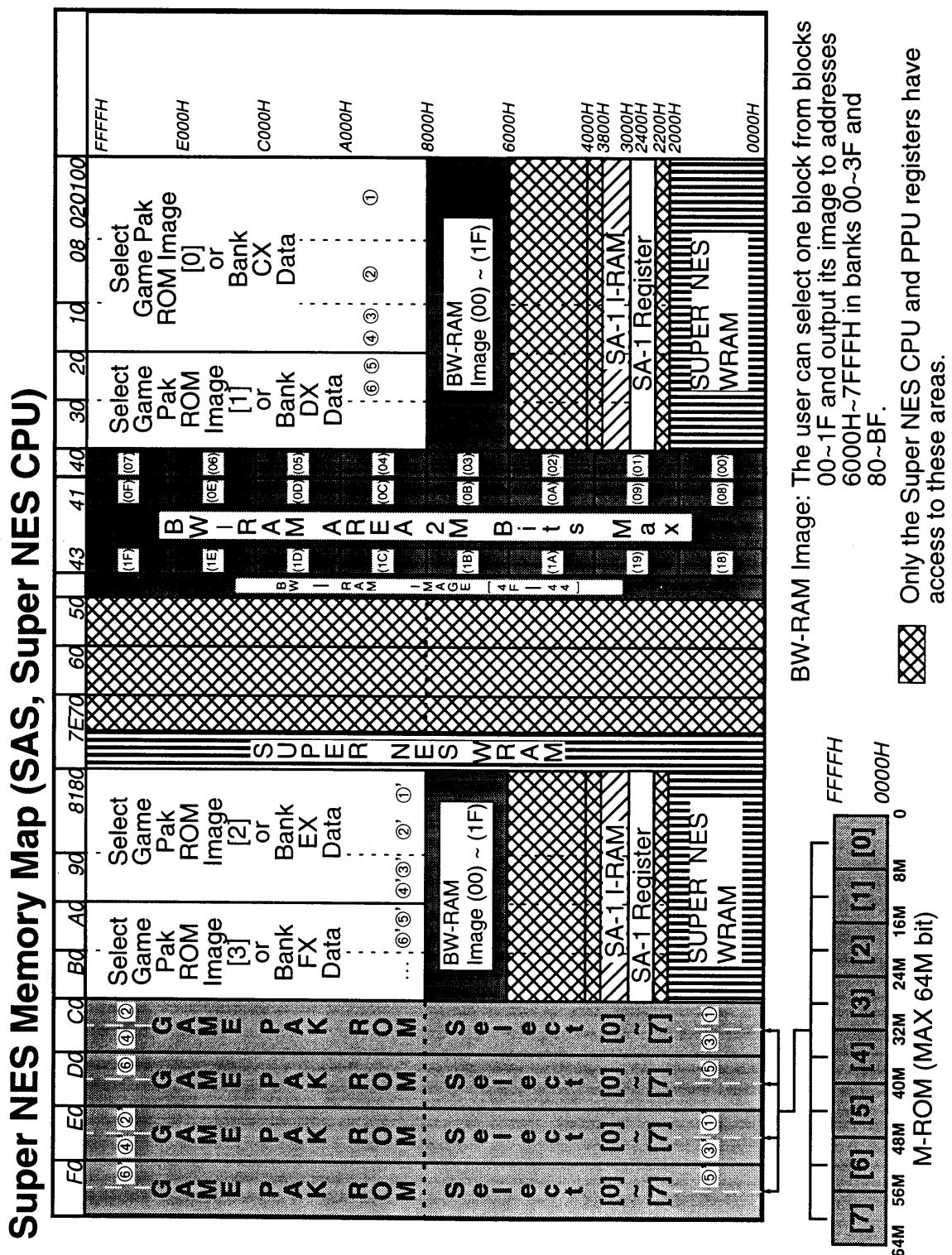


#### 2.2.3.2 BOTH SA-1 CPU AND SUPER NES CPU ACCESS I-RAM

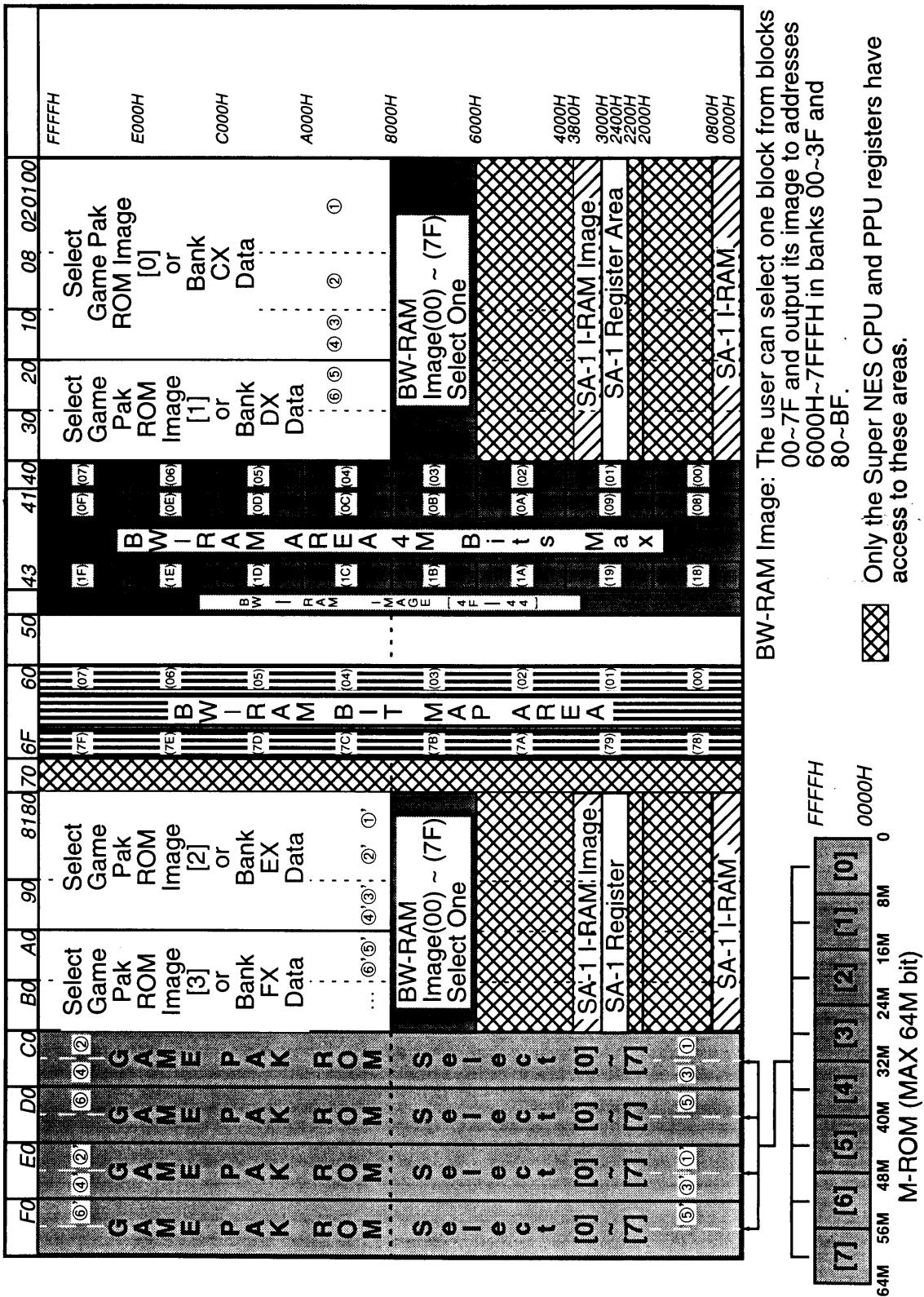


## **Chapter 3      Super Accelerator Memory Map**

### 3.1 MEMORY MAP FROM SUPER NES CPU PERSPECTIVE



## Super NES Memory Map (SAS, SA-1 CPU)



### 3.3 SUPER MMC

The Super MMC is a Super NES memory controller which can support a ROM capacity in excess of 32 Mbits. The memory map used by the Super MMC is called Map Mode 22. The SA-1 contains the Super MMC memory control function. Map Mode 22 features are described below.

#### 3.3.1 ROM BANK SWITCHING

The entire mask ROM is divided into 8 Mbit blocks, which can be projected onto the 8 Mbit areas, 0000H~FFFFH in banks C0H~CFH, D0H~DFH, E0H~EFH, and F0H~FFH. The same 8 Mbit data can be projected onto multiple areas.

#### 3.3.2 ROM IMAGE PROJECTION

The ROM data in banks CXH, DXH, EXH, and FXH, described above, is image projected onto, respectively, the 8 Mbit area 8000H~FFFFH, in banks 00H~1FH, 20H~3FH, 80H~9FH, and A0H~BFH.

The image projection method used is different from that used in Map Mode 21 in that the ROM data is projected in successive order, as demonstrated below.

```

C0:0000H~C0:7FFFH → 00:8000H~00:FFFFH
C0:8000H~C0:FFFFH → 01:8000H~01:FFFFH
C1:0000H~C1:7FFFH → 02:8000H~02:FFFFH
    •
    •
    •
CF:8000H~CF:7FFFH → 1F:8000H~1F:FFFFH

```

It is also possible to project the first 8 Mbits of data in the mask ROM (00:0000H~0F:FFFFH) onto bank 00H~1FH, regardless of the settings for banks CXH, DXH, EXH, and FXH. In a similar manner, data in 10:0000H~1F:FFFFH, 20:0000H~2F:FFFFH, and 30:0000H~3F:FFFFH can be projected onto banks 20H~3FH, 80H~9FH, and A0H~BFH, respectively.

#### 3.3.3 BACKUP RAM

Backup RAM is assigned to areas in bank 40H, justified to 0000H, as illustrated below.

16K RAM:	40:0000H~40:07FFFH
64K RAM:	40:0000H~40:1FFFFH
256K RAM:	40:0000H~40:7FFFFH
1M RAM:	40:0000H~41:FFFFH

Backup RAM is image projected to the 64 Kbit areas in 6000H~7FFFH of banks 00H~3FH and 80H~BFH. The backup area can be divided into 64 Kbit blocks. Any of these blocks can be projected as images. The data is identical in banks 00H~3FH and 80H~BFH.

### 3.3.4 PROTECTION OF BACKUP DATA

A write-protect setting is available to prevent data in the backup data area (banks 40H~7DH from being damaged. This setting protects data even in case of a CPU crash.

### 3.3.5 CONTROL REGISTERS

The Super MMC control registers are assigned to 2200H~23FFH of banks 00H~3FH and 80H~BFH.

### 3.3.6 CAUTIONS

Note that when the SA-1 Super MMC emulation function is used, the following specifications for the Super MMC do not apply.

#### 3.3.6.1 HIGH SPEED MODE

The SAS cannot use the Super NES CPU high-speed mode (3.58 MHz).

#### 3.3.6.2 ROM AND BACKUP RAM AREA

The maximum mask ROM area is 64 Mbits. The maximum backup RAM area is 2 Mbits.

#### 3.3.6.3 SHARED ROM MEMORY MAP

The Super NES CPU and SA-1 CPU share a common ROM memory map.

The ROM data in banks CXH, DXH, EXH, and FXH is identical (the same data is projected) for the Super NES CPU and SA-1 CPU. However, the program can be executed in different banks for each processor.

#### 3.3.6.4 BACKUP RAM PROTECTION

The image projected to Backup RAM is specified separately.

The RAM data which is projected to the backup RAM image area in 00H~3FH and 80H~BFH can be specified separately for the Super NES CPU and SA-1 CPU.

### 3.3.6.5 SA-1 I-RAM PRE-ASSIGNED

SA-1 internal RAM (I-RAM) is assigned according to memory mapping.

The I-RAM is assigned to 3000H~37FFH in banks 00H~3FH and 80H~BFH during Super NES CPU access and to 3000H~37FFH and 0000H~07FFH in banks 00H~3FH and 80H~BFH during SA-1 CPU access.

## 3.4 VECTORS AND ROM-REGISTERED DATA

Set the address for the Super NES CPU vectors and ROM-registered data to 00:7FB0H~00:7FFFH. When set to this area, they are assigned to FFB0H~FFFFH in bank 00H at Super NES start-up.

## **Chapter 4      SA-1 Internal Register Configuration**

The SA-1 internal registers are assigned to addresses 2200H~23FFH in the Super NES CPU and SA-1 CPU banks 00H~3FH and 80H~BFH. Registers with addresses 22\*\*H are write registers and those with addresses 23\*\*H are read registers.

### **4.1 EXPLANATION OF REGISTERS**

#### **4.1.1 SA-1 CPU CONTROL (CCNT)**

Access: Super NES CPU Write

Address: \*\*2200H

Size: 8 bits

Initial value: 20H

D7	D6	D5	D4	D3	D2	D1	D0	
SA-1 CPU IRQ	SA-1 CPU RDY B	SA-1 CPU RESB	SA-1 CPU NMI	SMEG3	SMEG2	SMEG1	SMEG0	2200H

SA-1 CPU IRQ:     SA-1 CPU IRQ (from Super NES CPU)  
                   0: No Interrupt  
                   1: Interrupt

SA-1 CPU RDY B: Ready  
                   0: Ready  
                   1: Wait

SA-1 CPU RESB:    SA-1 CPU reset  
                   0: Cancel  
                   1: Reset

SA-1 CPU NMI:     SA-1 CPU NMI (from Super NES CPU)  
                   0: No Interrupt  
                   1: Interrupt

SMEG0~SMEG3:    Message from Super NES CPU to SA-1 CPU

#### 4.1.2 SUPER NES CPU INT ENABLE (SIE)

Access: Super NES CPU Write

Address: \*\*2201H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SA-1 CPU IRQEN	0	CHDMA IRQEN	0	0	0	0	0	2201H

SA-1 CPU IRQEN: IRQ enable/disable from the SA-1 CPU

0: Disable

1: Enable

CHDMA IRQEN: Character conversion DMA IRQ enable/disable

0: Disable

1: Enable

#### 4.1.3 SUPER NES CPU INT CLEAR (SIC)

Access: Super NES CPU Write

Address: \*\*2202H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SA-1 CPU IRQCL	0	CHDMA IRQCL	0	0	0	0	0	2202H

SA-1 CPU IRQCL: IRQ clear from the SA-1 CPU

0: No change

1: Clear

CHDMA IRQCL: Character conversion DMA IRQ clear

0: No change

1: Clear

#### 4.1.4 SA-1 CPU RESET VECTOR (CRV)

Access: Super NES CPU Write

Address: \*\*2203H, \*\*2204H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
CRV7	CRV6	CRV5	CRV4	CRV3	CRV2	CRV1	CRV0	2203H
CRV15	CRV14	CRV13	CRV12	CRV11	CRV10	CRV9	CRV8	2204H

#### 4.1.5 SA-1 CPU NMI VECTOR (CNV)

Access: Super NES CPU Write

Address: \*\*2205H, \*\*2206H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
CNV7	CNV6	CNV5	CNV4	CNV3	CNV2	CNV1	CNV0	2205H
CNV15	CNV14	CNV13	CNV12	CNV11	CNV10	CNV9	CNV8	2206H

#### 4.1.6 SA-1 CPU IRQ VECTOR (CIV)

Access: Super NES CPU Write

Address: \*\*2207H, \*\*2208H

Size: 16 bits

Initial value: Unspecified

D7	D6	D5	D4	D3	D2	D1	D0	
CIV7	CIV6	CIV5	CIV4	CIV3	CIV2	CIV1	CIV0	2207H
CIV15	CIV14	CIV13	CIV12	CIV11	CIV10	CIV9	CIV8	2208H

#### 4.1.7 SUPER NES CPU CONTROL (SCNT)

Access: SA-1 CPU Write

Address: \*\*2209H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SNES CPU IRQ	SNES CPU IVSW	0	SNES CPU NVSW	CMEG3	CMEG2	CMEG1	CMEG0	2209H

Super NES  
CPU IRQ: IRQ from SA-1 CPU to Super NES CPU  
0: No IRQ interrupt  
1: IRQ interrupt

Super NES  
CPU IVSW: Super NES CPU IRQ vector selection  
0: Game pak ROM  
1: Super NES CPU IRQ vector register

Super NES  
CPU NVSW: Super NES CPU NMI vector selection  
0: Game pak ROM  
1: Super NES CPU NMI vector register

CMEG0~CMEG3: Message from SA-1 CPU to Super NES CPU

#### 4.1.8 SA-1 CPU INT ENABLE (CIE)

Access: SA-1 CPU Write

Address: \*\*220AH

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SNES CPU IRQEN	Timer IRQEN	DMA IRQEN	SNES CPU NMIEN	0	0	0	0	220AH

Super NES  
CPU IRQEN: IRQ control from Super NES CPU to SA-1 CPU

0: Disable

1: Enable

Timer IRQEN: IRQ control from timer to SA-1 CPU

0: Disable

1: Enable

DMA IRQEN: IRQ control to SA-1 CPU at end of SA-1 DMA

0: Disable

1: Enable

Super NES  
CPU NMIEN: NMI control from Super NES CPU to SA-1 CPU

0: Disable

1: Enable

#### 4.1.9 SA-1 CPU INT CLEAR (CIC)

Access: SA-1 CPU Write

Address: \*\*220BH

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SNES CPU IRQCL	Timer IRQCL	DMA IRQCL	SNES CPU NMICL	0	0	0	0	220BH

Super NES

CPU IRQCL: IRQ clear from Super NES CPU to SA-1 CPU

0: No change

1: Clear

Timer IRQCL: IRQ clear from timer to SA-1 CPU

0: No change

1: Clear

DMA IRQCL: IRQ clear to SA-1 CPU at end of SA-1 DMA

0: No change

1: Clear

Super NES

CPU NMICL: NMI clear from Super NES CPU to SA-1 CPU

0: No change

1: Clear

#### 4.1.10 SUPER NES CPU NMI VECTOR (SNV)

Access: SA-1 CPU Write

Address: \*\*220CH, \*\*220DH

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
Super NES CPU NMI Vector (Low)								
SNV7	SNV6	SNV5	SNV4	SNV3	SNV2	SNV1	SNV0	220CH
Super NES CPU NMI Vector (High)								
SNV15	SNV14	SNV13	SNV12	SNV11	SNV10	SNV9	SNV8	220DH

#### 4.1.11 SUPER NES CPU IRQ VECTOR (SIV)

Access: SA-1 CPU Write

Address: \*\*220EH, \*\*220FH

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
Super NES CPU IRQ Vector (Low)								
SIV7	SIV6	SIV5	SIV4	SIV3	SIV2	SIV1	SIV0	220EH
Super NES CPU IRQ Vector (High)								
SIV15	SIV14	SIV13	SIV12	SIV11	SIV10	SIV9	SIV8	220FH

#### 4.1.12 H/V TIMER CONTROL (TMC)

Access: SA-1 CPU Write

Address: \*\*2210H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
HVSELB	0	0	0	0	0	VEN	HEN	2210H

HVSELB: Select HV timer

0: HV Timer

1: Linear Timer

VEN, HEN: V count enable, H count enable

00: Disable both H and V

01: Enable H only: IRQ at H timer value

10: Enable V only: IRQ at V timer value

11: Enable both H and V: IRQ at H/V timer values

#### 4.1.13 SA-1 CPU TIMER RESTART (CTR)

Access: SA-1 CPU Write

Address: \*\*2211H

Size: 8 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
--	--	--	--	--	--	--	--	2211H

Writing any value to this register restarts the timer at 0.

#### 4.1.14 SET H-COUNT (HCNT)

Access: SA-1 CPU Write

Address: \*\*2212H, \*\*2213H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
H7	H6	H5	H4	H-Count (Low) H3	H2	H1	H0	2212H
0	0	0	0	H-Count (High)	0	0	H8	2213H

HV timer: Timer IRQ H count value (0~340)

Linear timer: Lower 9 bits of the timer IRQ linear counter (0~511)

#### 4.1.15 SET V COUNT (VCNT)

Access: SA-1 CPU Write

Address: \*\*2214H, \*\*2215H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
V7	V6	V5	V4	V-Count (Low) V3	V2	V1	V0	2214H
0	0	0	0	V-Count (High)	0	0	V8	2215H

HV timer: Timer IRQ V count value

NTSC, 0~261

PAL, 0~311

Linear timer: Upper 9 bits of the timer IRQ linear counter (0~511)

#### 4.1.16 SET SUPER MMC BANK C (CXB)

Access: Super NES CPU Write

Address: \*\*2220H

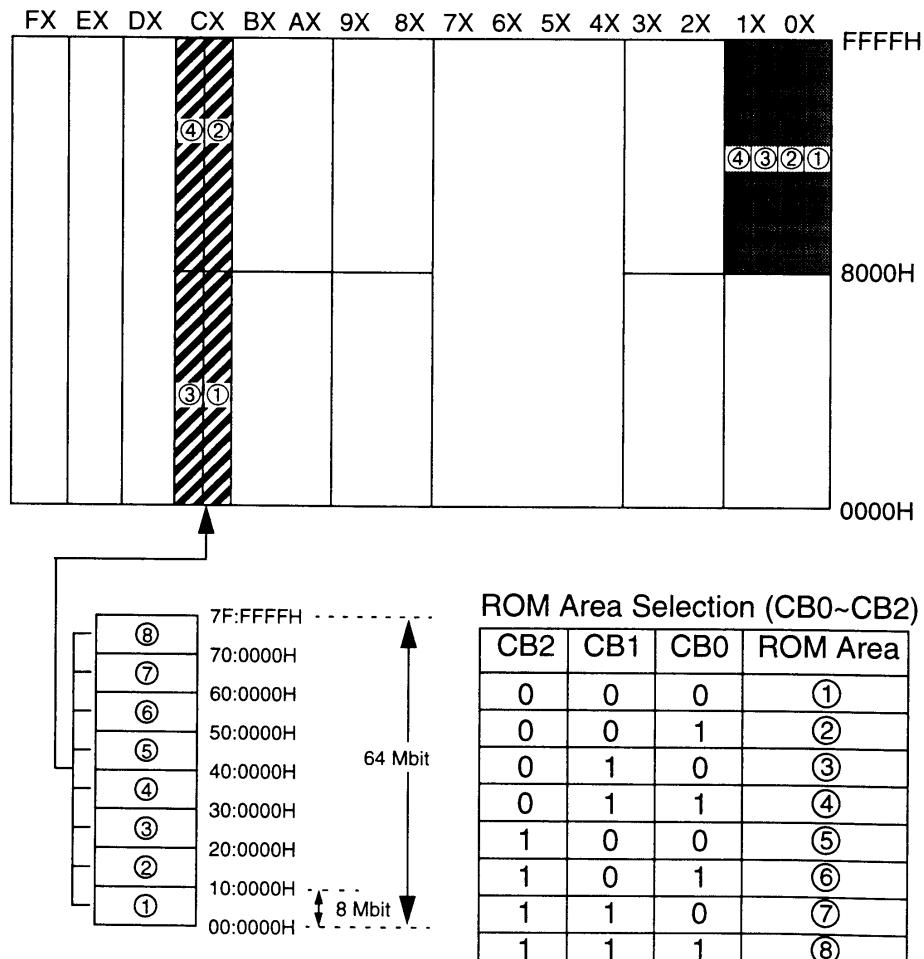
Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
CBM	0	0	0	0	CB2	CB1	CB0	2220H

CBM: CXH Bank Image Projection

- 1: CXH bank data is copied into addresses 8000H~FFFFH of banks 0XH~1XH (shaded).
- 0: The game pak ROM area ① is copied to addresses 8000H~FFFFH of banks 0XH~1XH.



#### 4.1.17 SET SUPER MMC BANK D (DXB)

Access: Super NES CPU Write

Address: \*\*2221H

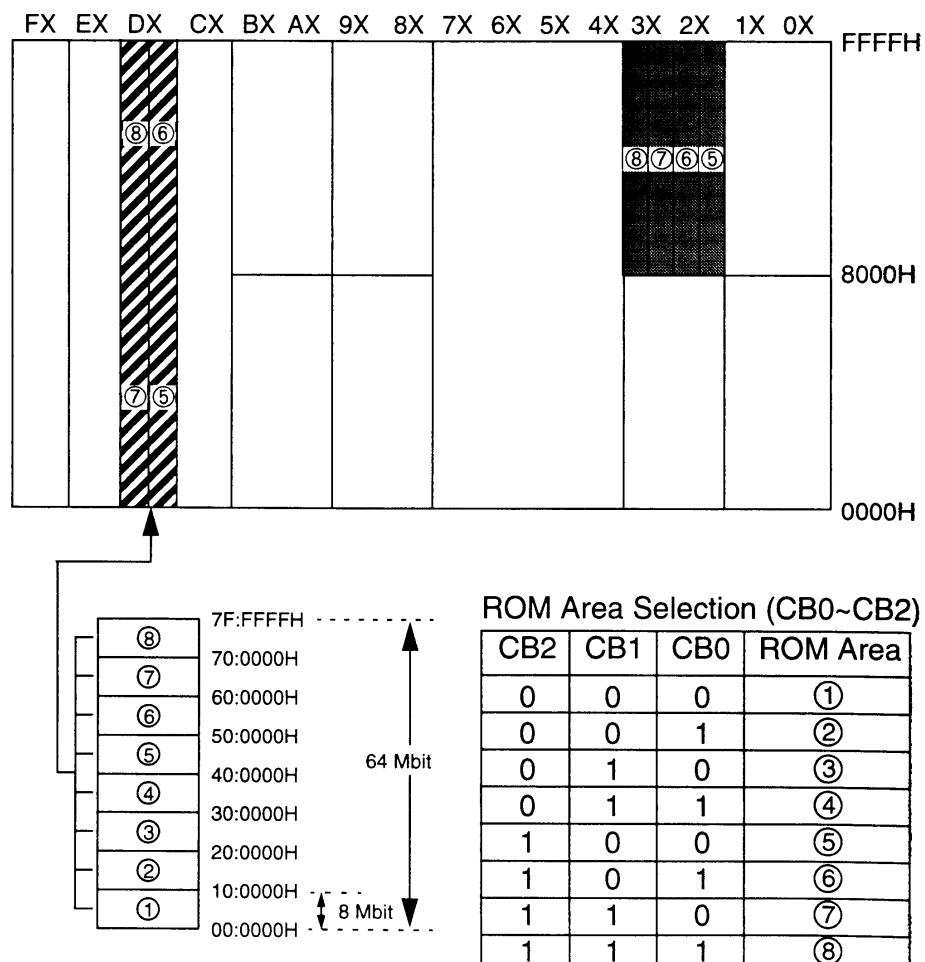
Size: 8 bits

Initial value: 01H

D7	D6	D5	D4	D3	D2	D1	D0	
DBM	0	0	0	0	CB2	CB1	CB0	2221H

DBM: DXH Bank Image Projection

- 1: DXH bank data is copied into addresses 8000H~FFFFH of banks 2XH~3XH (shaded).
- 0: The game pak ROM area ② is copied to addresses 8000H~FFFFH of banks 2XH~3XH.



#### 4.1.18 SET SUPER MMC BANK E (EXB)

Access: Super NES CPU Write

Address: \*\*2222H

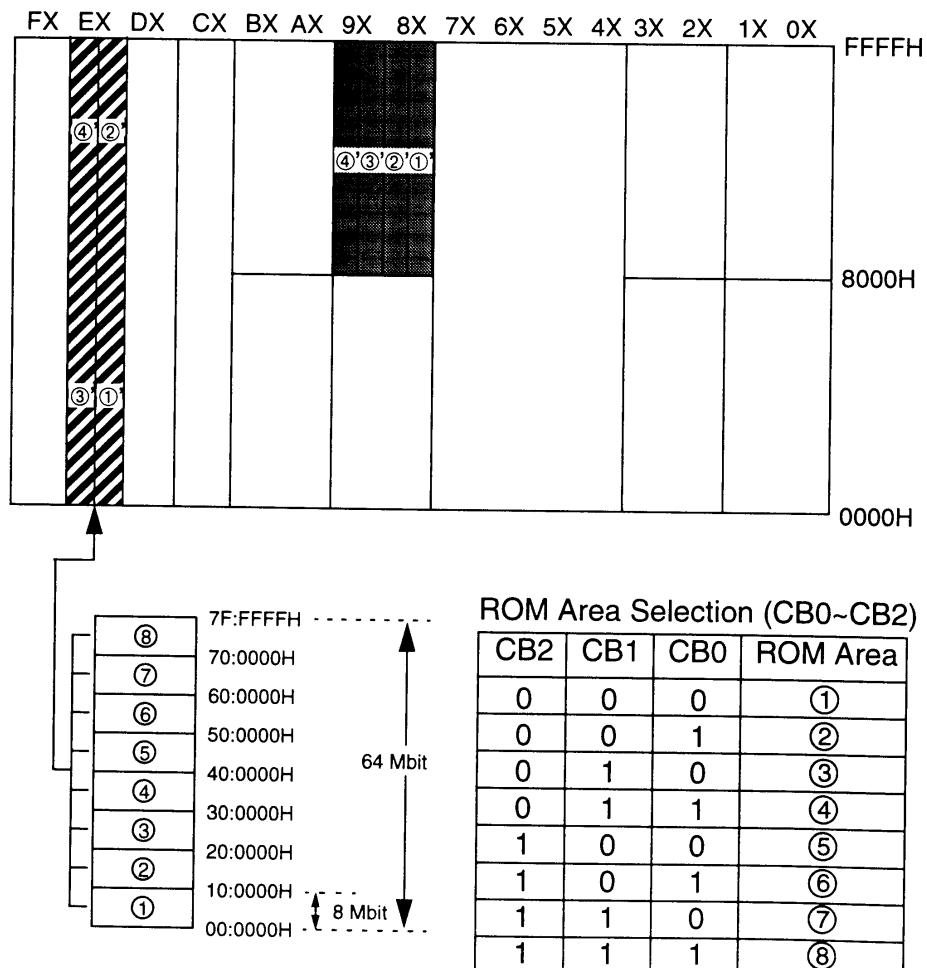
Size: 8 bits

Initial value: 02H

	D7	D6	D5	D4	D3	D2	D1	D0	
EBM	0	0	0	0	CB2	CB1	CB0		2222H

EBM: EXH Bank Image Projection

- 1: EXH bank data is copied into addresses 8000H~FFFFH of banks 8XH~9XH (shaded).
- 0: The game pak ROM area ③ is copied to addresses 8000H~FFFFH of banks 8XH~9XH.



#### 4.1.19 SET SUPER MMC BANK F (FXB)

Access: Super NES CPU Write

Address: \*\*2223H

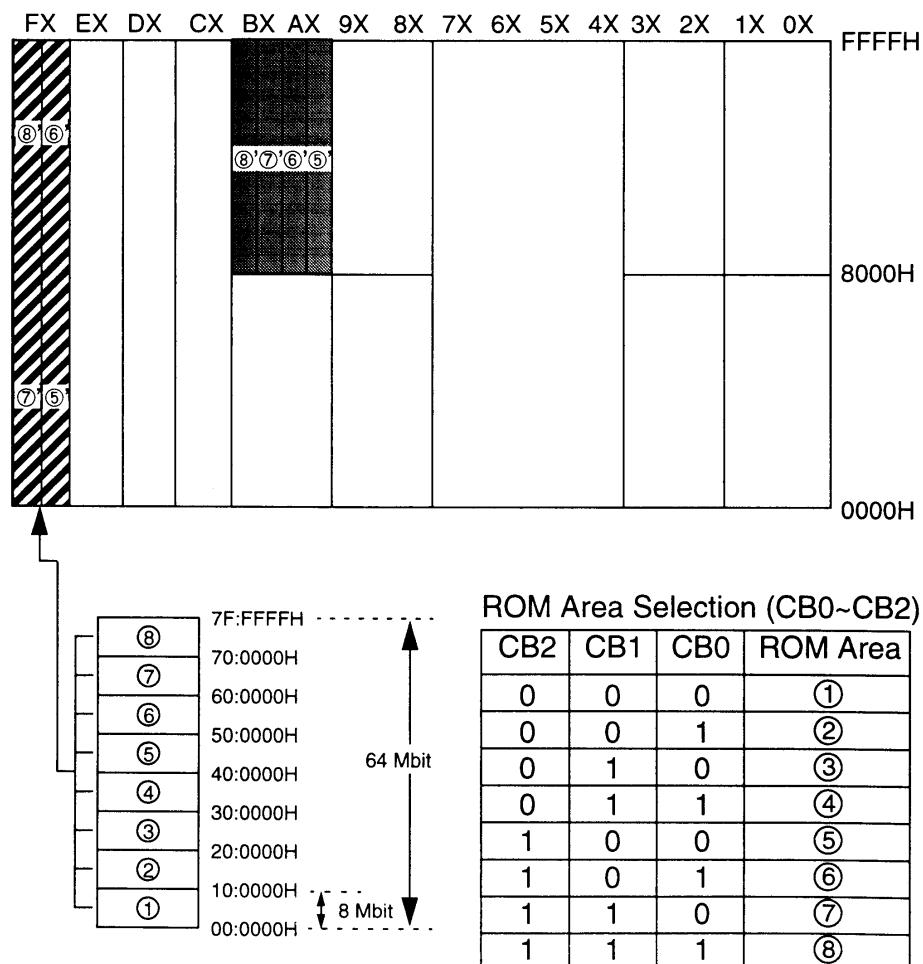
Size: 8 bits

Initial value: 03H

D7	D6	D5	D4	D3	D2	D1	D0	
FBM	0	0	0	0	CB2	CB1	CB0	2223H

FBM: FXH Bank Image Projection

- 1: FXH bank data is copied into addresses 8000H~FFFFH of banks AXH~BXH (shaded).
- 0: The game pak ROM area ④ is copied to addresses 8000H~FFFFH of banks AXH~BXH.



#### 4.1.20 SUPER NES CPU BW-RAM ADDRESS MAPPING (BMAPS)

Access: Super NES CPU Write

Address: \*\*2224H

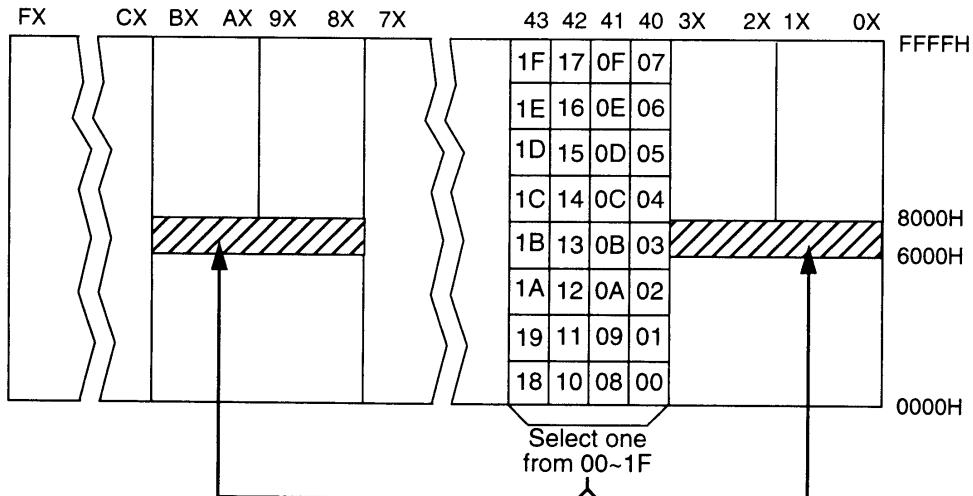
Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	SBM4	SBM3	SBM2	SBM1	SBM0	2224H

SBM0~4: BW-RAM Address Image Mapping for Super NES CPU

The BW-RAM image to be mapped to addresses 6000H~7FFFH of banks 00H~3FH and 80H~BFH is user selectable from 00~1F.



Note: The same image is mapped to all areas, (i.e., 00:6000H~00:7FFFH, 01:6000H~01:7FFFH .... BF:6000H~BF:7FFFH).

#### 4.1.21 SA-1 CPU BW-RAM ADDRESS MAPPING (BMAP)

Access: SA-1 CPU Write

Address: \*\*2225H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SW46	CBM6	CBM5	CBM4	CBM3	CBM2	CBM1	CBM0	2225H

CBM0~CBM6: BW-RAM Address Image Mapping for SA-1 CPU

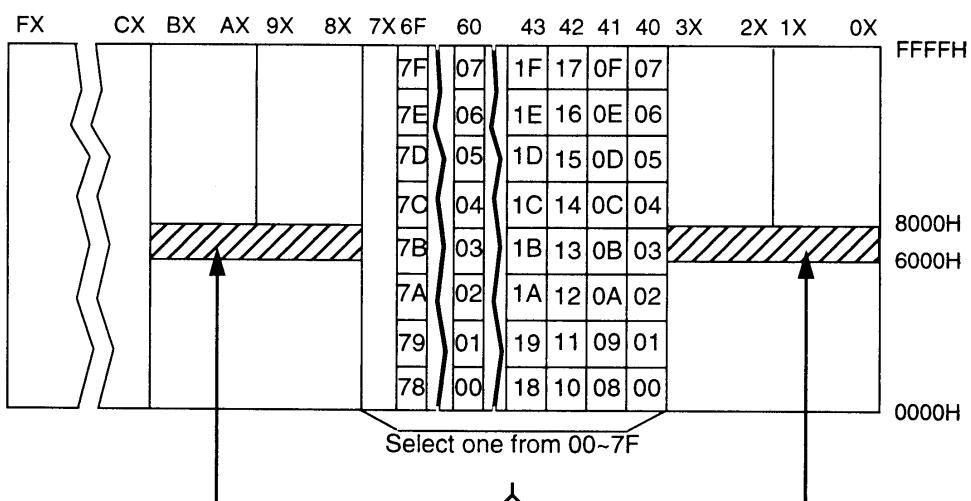
This selects the BW-RAM image to be mapped to the SA-1 CPU at addresses 6000H~7FFFH of banks 00H~3FH and 80H~BFH.

SW46:

Specifies the BW-RAM source to be projected

0: Banks 40H~43H are displayed in 32 blocks using CBM0~CBM4.

1: Banks 60H~6FH are displayed in 128 blocks using CBM0~CBM6.



Note: The same image is mapped to all areas, (i.e., 00:6000H~00:7FFFH, 01:6000H~01:7FFFH .... BF:6000H~BF:7FFFH).

#### 4.1.22 SUPER NES CPU BW-RAM WRITE ENABLE (SBWE)

Access: Super NES CPU Write

Address: \*\*2226H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SWEN	0	0	0	0	0	0	0	2226H

SWEN: Cancels BW-RAM write protection from Super NES CPU

0: Protect

1: Write enable

#### 4.1.23 SA-1 CPU BW-RAM WRITE ENABLE (CBWE)

Access: SA-1 CPU Write

Address: \*\*2227H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
CWEN	0	0	0	0	0	0	0	2227H

CWEN: Cancels BW-RAM write protection from SA-1 CPU

0: Protect

1: Write enable

#### 4.1.24 BW-RAM WRITE-PROTECTED AREA (BWPA)

Access: Super NES CPU Write

Address: \*\*2228H

Size: 8 bits

Initial value: FFH

D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	BWP3	BWP2	BWP1	BWP0	2228H

BWP0~3: BW-RAM Write Protected Area Setting

BWP3	BWP2	BWP1	BWP0	BW-RAM Write Protected Area	
				Area	Size (bits)
0	0	0	0	400000 - 4000FF	2K
0	0	0	1	400000 - 4001FF	4K
0	0	1	0	400000 - 4003FF	8K
0	0	1	1	400000 - 4007FF	16K
0	1	0	0	400000 - 400FFF	32K
0	1	0	1	400000 - 401FFF	64K
0	1	1	0	400000 - 403FFF	128K
0	1	1	1	400000 - 407FFF	256K
1	0	0	0	400000 - 40FFFF	512K
1	0	0	1	400000 - 41FFFF	1M
1	0	1	0	400000 - 43FFFF	2M

At start-up, all areas are write-protected.

#### 4.1.25 SA-1 I-RAM WRITE PROTECTION (SIWP)

Access: Super NES CPU Write

Address: \*\*2229H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SIWP7	SIWP6	SIWP5	SIWP4	SIWP3	SIWP2	SIWP1	SIWP0	2229H

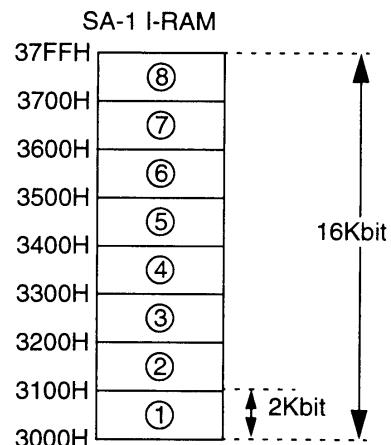
SIWP0~7:

SA-1 I-RAM Write Protection Setting

0: Write disable

1: Write enable

- SIWP0: Sets 3000H ~ 30FFH
- SIWP1: Sets 3100H ~ 31FFH
- SIWP2: Sets 3200H ~ 32FFH
- SIWP3: Sets 3300H ~ 33FFH
- SIWP4: Sets 3400H ~ 34FFH
- SIWP5: Sets 3500H ~ 35FFH
- SIWP6: Sets 3600H ~ 36FFH
- SIWP7: Sets 3700H ~ 37FFH



#### 4.1.26 SA-1 I-RAM WRITE PROTECTION (CIWP)

Access: SA-1 CPU Write

Address: \*\*222AH

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
CIWP7	CIWP6	CIWP5	CIWP4	CIWP3	CIWP2	CIWP1	CIWP0	222AH

CIWP0~CIWP7: SA-1 I-RAM write protection setting

0: Write disable

1: Write enable

CIWP0: Sets 3000H ~ 30FFH  
0000H ~ 00FFH

CIWP1: Sets 3100H ~ 31FFH  
0100H ~ 01FFH

CIWP2: Sets 3200H ~ 32FFH  
0200H ~ 02FFH

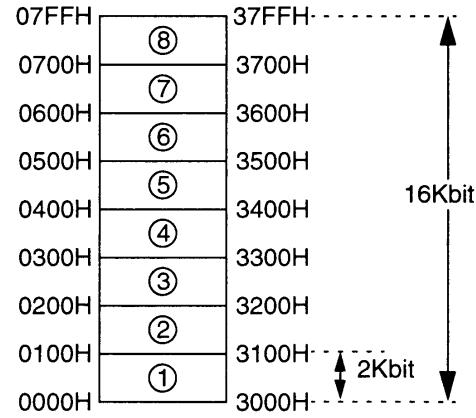
CIWP3: Sets 3300H ~ 33FFH  
0300H ~ 03FFH

CIWP4: Sets 3400H ~ 34FFH  
0400H ~ 04FFH

CIWP5: Sets 3500H ~ 35FFH  
0500H ~ 05FFH

CIWP6: Sets 3600H ~ 36FFH  
0600H ~ 06FFH

CIWP7: Sets 3700H ~ 37FFH  
0700H ~ 07FFH



#### 4.1.27 DMA CONTROL (DCNT)

Access: SA-1 CPU Write

Address: \*\*2230H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
DMAEN	DPrio	CDEN	CDSEL	0	DD	SD1	SD0	2230H

DMAEN: DMA Enable Control

- 0: DMA disable
- 1: DMA enable

DPrio: Processing priority between SA-1 CPU and DMA

- 0: SA-1 CPU priority
- 1: DMA priority

DD: Destination device

- 0: SA-1 I-RAM
- 1: BW-RAM

SD0, SD1: Source Device

SD1	SD0	Device
0	0	Game Pak ROM
0	1	BW-RAM
1	0	SA-1 I-RAM

CDEN: DMA mode selection

- 0: Normal DMA
- 1: Character conversion DMA

CDSEL: Character conversion DMA type

- 0: SA-1 CPU → SA-1 I-RAM write (CHR conv 2)
- 1: BW-RAM → SA-1 I-RAM transfer (CHR conv 1)

#### 4.1.28 CHARACTER CONVERSION DMA PARAMETERS (CDMA)

Access: SA-1 CPU/Super NES CPU Write

Address: \*\*2231H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
CHDEND	0	0	SIZE2	SIZE1	SIZE0	CB1	CB0	2231H

CB0 and CB1: Character conversion DMA color mode

CB1	CB0	Character Format
0	0	8 Bit/Dot
0	1	4 Bit/Dot
1	0	2 Bit/Dot
1	1	-----

SIZE 0~2: Number of virtual VRAM horizontal characters

SIZE2	SIZE1	SIZE0	Number of Characters
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32

CHDEND: End character conversion 1

When character conversion 1 is completed,  
CHDEND is set to "1" by the Super NES CPU.

#### 4.1.29 DMA SOURCE DEVICE START ADDRESS (SDA)

Access: Super NES CPU/SA-1 CPU Write

Address: \*\*2232H ~ \*\*2234H

Size: 24 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
DSA7	DSA6	DSA5	DSA4	DSA3	DSA2	DSA-1	DSA0	2232H
DSA-15	DSA-14	DSA-13	DSA-12	DSA-11	DSA-10	DSA9	DSA8	2233H
DSA23	DSA22	DSA21	DSA20	DSA-19	DSA-18	DSA-17	DSA-16	2234H

DSA0-DSA23: DMA source device A start address

Data should be stored to the SDA registers in the order of Low → Middle → High.

#### 4.1.30 DMA DESTINATION START ADDRESS (DDA)

Access: Super NES CPU/SA-1 CPU Write

Address: \*\*2235H ~ \*\*2237H

Size: 24 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	2235H
DDA15	DDA14	DDA13	DDA12	DDA11	DDA10	DDA9	DDA8	2236H
DDA23	DDA22	DDA21	DDA20	DDA19	DDA18	DDA17	DDA16	2237H

DDA0-DDA23: DMA destination device start address

When transmitting to SA-1 I-RAM, DMA transfer is initiated by the write to register 2236H.

When transmitting to BW-RAM, DMA transfer is initiated by the write to register 2237H.

Data should be stored to the DDA registers in the order of Low → Middle → High.

#### 4.1.31 DMA TERMINAL COUNTER (DTC)

Access: SA-1 CPU Write

Address: \*\*2238H, \*\*2239H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
T7	T6	DMA Terminal Counter (Low)				T1	T0	2238H
T15	T14	DMA Terminal Counter (High)				T9	T8	2239H

T0-T15: Number of bytes (1 ~ 65535) for DMA transmission

#### 4.1.32 BW-RAM BIT MAP FORMAT (BBF)

Access: SA-1 CPU Write

Address: \*\*223FH

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
SEL42	--	--	--	--	--	--	--	223FH

SEL42: BW-RAM bitmap logical space format setting from

the perspective of the SA-1 CPU

0: 16 color mode (4 bits/dot)

1: 4 color mode (2 bits/dot)

#### 4.1.33 BIT MAP REGISTER FILE (BRF)

Access: SA-1 CPU Write

Address: \*\*2240H ~ \*\*224FH

Size: 16 bytes

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
BM07	BM06	BM05	BM04	BM03	BM02	BM01	BM00	2240H
BM17	BM16	BM15	BM14	BM13	BM12	BM11	BM10	2241H
BM27	BM26	BM25	BM24	BM23	BM22	BM21	BM20	2242H
BM37	BM36	BM35	BM34	BM33	BM32	BM31	BM30	2243H
BM47	BM46	BM45	BM44	BM43	BM42	BM41	BM40	2244H
BM57	BM56	BM55	BM54	BM53	BM52	BM51	BM50	2245H
BM67	BM66	BM65	BM64	BM63	BM62	BM61	BM60	2246H
BM77	BM76	BM75	BM74	BM73	BM72	BM71	BM70	2247H

Figure 1-4-4 Bitmap Register Files 0 ~ 7

BM87	BM86	BM85	BM84	BM83	BM82	BM81	BM80	2248H
BM97	BM96	BM95	BM94	BM93	BM92	BM91	BM90	2249H
BMA7	BMA6	BMA5	BMA4	BMA3	BMA2	BMA1	BMA0	224AH
BMB7	BMB6	BMB5	BMB4	BMB3	BMB2	BMB1	BMB0	224BH
BMC7	BMC6	BMC5	BMC4	BMC3	BMC2	BMC1	BMC0	224CH
BMD7	BMD6	BMD5	BMD4	BMD3	BMD2	BMD1	BMD0	224DH
BME7	BME6	BME5	BME4	BME3	BME2	BME1	BME0	224EH
BMF7	BMF6	BMF5	BMF4	BMF3	BMF2	BMF1	BMF0	224FH

Figure 1-4-5 Bitmap Register Files 8 ~ FF

BRF0 ~ BRF7: Buffer 1

BRF8 ~ BRFF: Buffer 2

#### 4.1.34 ARITHMETIC CONTROL (MCNT)

Access: SA-1 CPU Write

Address: \*\*2250H

Size: 8 bits

Initial value: 00H

D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	ACM	M/D	2250H

Types of M/D and ACM arithmetic operations

ACM	M/D	TYPE OF OPERATION
0	0	Multiplication
0	1	Division
1	0	Cumulative Sum

NOTE: Store a "1" in ACM to clear the result register during cumulative sum operations.

#### 4.1.35 ARITHMETIC PARAMETERS: MULTIPLICAND/DIVIDEND (MA)

Access: SA-1 CPU Write

Address: \*\*2251H, \*\*2252H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
Arithmetic Parameters: Multiplicand/Dividend (Low)								2251H
MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0	
Arithmetic Parameters: Multiplicand/Dividend (High)								2252H
MA15	MA14	MA13	MA12	MA11	MA10	MA9	MA8	

MA0-MA15: Multiplicand/Dividend setting (signed 16-bit data)

The data contained in MA0~MA15 is saved even after it is acted upon.

The register does not need to be reset, when used for multiplication.

When used for division, however, the register must be reset each time.

#### 4.1.36 ARITHMETIC PARAMETERS: MULTIPLIER/DIVISOR (MB)

Access: SA-1 CPU Write

Address: \*\*2253H, \*\*2254H

Size: 16 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
Arithmetic Parameters: Multiplier/Divisor (Low)								2253H
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0	
Arithmetic Parameters: Multiplier/Divisor (High)								2254H
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8	

MB0-MB15: Multiplier/divisor setting

- Signed data when used for multiplication
- Unsigned data when used for division

The arithmetic operation is executed following a write to register 2254H.

The multiplier/divisor must be reset each time an operation is performed.

#### 4.1.37 VARIABLE-LENGTH BIT PROCESSING (VBD)

Access: SA-1 CPU Write

Address: \*\*2258H

Size: 8 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
HL	0	0	0	VB3	VB2	VB1	VB0	2258H

HL: Variable-length data read mode

1: Auto-increment mode

0: Fixed mode

VB0-VB3: Significant bit length of data previously stored

VB3	VB2	VB1	VB0	Data Length (bits)
0	0	0	0	16
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

#### 4.1.38 VARIABLE-LENGTH BIT GAME PAK ROM START ADDRESS (VDA)

Access: SA-1 CPU Write

Address: \*\*2259H-\*\*225BH

Size: 24 bits

Initial value: Nonspecific

D7	D6	D5	D4	D3	D2	D1	D0	
Variable-Length Bit Game Pak ROM Start Address (Low)								
VA7	VA6	VA5	VA4	VA3	VA2	VA1	VA0	2259H
Variable-Length Bit Game Pak ROM Start Address (Middle)								
VA15	VA14	VA13	VA12	VA11	VA10	VA9	VA8	225AH
Variable-Length Bit Game Pak ROM Start Address (High)								
VA23	VA22	VA21	VA20	VA19	VA18	VA17	VA16	225BH

VA0-VA23: Game Pak ROM variable-length bit area start address setting.

Variable-length bit execution begins with a write to register 225BH.

#### 4.1.39 SUPER NES CPU FLAG READ (SFR)

Access: Super NES CPU Read

Address: \*\*2300H

Size: 8 bits

D7	D6	D5	D4	D3	D2	D1	D0	
SA-1 CPU IRQ	IVSW	CHDMA IRQ	NVSW	CMEG3	CMEG2	CMEG1	CMEG0	2300H

SA-1 CPU IRQ: IRQ flag from SA-1 CPU

0: No IRQ

1: IRQ

IVSW: Super NES CPU IRQ vector setting

0: Game pak ROM data

1: SIV register data

CHDMA IRQ: Character conversion DMA IRQ flag

0: No IRQ

1: IRQ (character conversion 1 stand-by)

NVSW: Super NES CPU NMI vector setting

0: Game pak ROM data

1: SNV register data

CMEG0-CMEG3: Message port from SA-1 CPU: 0~15

NOTE: Reading this register does not clear its contents.

#### 4.1.40 SA-1 CPU FLAG READ (CFR)

Access: SA-1 CPU Read

Address: \*\*2301H

Size: 8 bits

D7	D6	D5	D4	D3	D2	D1	D0	
SNES CPU IRQ	Timer IRQ	DMA IRQ	SNES CPU NMI	SMEG3	SMEG2	SMEG1	SMEG0	2301H

Super NES

CPU IRQ: IRQ flag from Super NES CPU

0: No IRQ

1: IRQ

Timer IRQ:

IRQ flag from timer.

0: No IRQ

1: IRQ

DMA IRQ:

IRQ flag at the end of DMA

0: No IRQ

1: IRQ (end of DMA)

Super NES

CPU NMI: NMI flag from Super NES CPU

0: No NMI

1: NMI

SMEG0-SMEG3: Message port from Super NES CPU: 0~15

NOTE: Reading this register does not clear its contents.

#### 4.1.41 H-COUNT READ (HCR)

Access: SA-1 CPU Read

Address: \*\*2302H, \*\*2303H

Size: 16 bits

D7	D6	D5	D4	D3	D2	D1	D0	
H7	H6	H5	H4	H3	H2	H1	H0	2302H
--	--	--	--	--	--	--	H8	2303H

H0-H8:

HV timer:H-count (dots,0~340) read

Linear timer: Lower 9-bit count (0~511) read

All HV counter values are latched when register 2302H is read.

#### 4.1.42 V-COUNT READ (VCR)

Access: SA-1 CPU Read

Address: \*\*2304H, \*\*2305H

Size: 16 bits

D7	D6	D5	D4	D3	D2	D1	D0	
V7	V6	V5	V4	V3	V2	V1	V0	2304H
--	--	--	--	--	--	--	V8	2305H

V0-V8:

HV timer:V-count (lines) read

NTSC, 0~261

PAL, 0~311

Linear timer: Upper 9-bit counter value (0~511) read

#### 4.1.43 ARITHMETIC RESULT [PRODUCT/QUOTIENT/ACCUMULATIVE SUM] (MR)

Access: SA-1 CPU Read

Address: \*\*2306H ~ \*\*230AH

Size: 40 bits

D7	D6	D5	D4	D3	D2	D1	D0	
Read Arithmetic Result (product/quotient/cumulative sum) W0								
D7	D6	D5	D4	D3	D2	D1	D0	2306H
Read Arithmetic Result (product/quotient/cumulative sum) W1								
D15	D14	D13	D12	D11	D10	D9	D8	2307H
Read Arithmetic Result (product/remainder/cumulative sum) W2								
D23	D22	D21	D20	D19	D18	D17	D16	2308H
Read Arithmetic Result (product/remainder/cumulative sum) W3								
D31	D30	D29	D28	D27	D26	D25	D24	2309H
Read Arithmetic Result (cumulative sum) W4								
D39	D38	D37	D36	D35	D34	D33	D32	230AH

D0-D39:

Arithmetic result

Multiplication: 16 (S) x 16 (S) = 32 (S)...D0-D31

Division: 16 (S) / 16 (U) = 16 (S) ...D0-D15

Remainder: 16 (U)

...D16-D31

Cumulative Sum:  $\sum(16 (S) \times 16 (S)) = 40 (S)$

...D0-D39

#### 4.1.44 ARITHMETIC OVERFLOW FLAG (OF)

Access: SA-1 CPU Read

Address: \*\*230BH

Size: 8 bits

D7	D6	D5	D4	D3	D2	D1	D0	
OF	--	--	--	--	--	--	--	230BH

OF: Overflow flag  
 1: Overflow  
 0: No overflow

#### 4.1.45 VARIABLE-LENGTH DATA READ PORT (VDP)

Access: SA-1 CPU Read

Address: \*\*230CH, \*\*230DH

Size: 16 bits

D7	D6	D5	D4	D3	D2	D1	D0	
Variable-Length Data Read Port (Low)								
VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	230CH
Variable-Length Data Read Port (High)								
VD15	VD14	VD13	VD12	VD11	VD10	VD9	VD8	230DH

VD0-VD15: The 16-bit data resulting from barrel-shifting the values stored in the VBD register (\*\*2258H).

#### 4.1.46 VERSION CODE REGISTER (VC)

Access: Super NES CPU Read

Address: \*\*230EH

Size: 8 bits

D7	D6	D5	D4	D3	D2	D1	D0	
VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	230EH

VC0 ~ VC7: SA-1 Device Version

## ***Chapter 5 Multi-Processor Processing***

### **5.1 MULTI-PROCESSOR SYSTEM**

The Super Accelerator System (SAS) is a multi-processor system in which two MPUs (the Super NES CPU and the SA-1 CPU) operate in parallel. The Super NES CPU performs as the main processor, controlling execution of the SA-1 CPU. The SA-1 CPU cannot control Super NES CPU operations. This main/sub relationship is a hardware arrangement. Software can be used to manipulate flags and interrupts to use the faster SA-1 CPU as the main processor.

### **5.2 STARTING AND STOPPING THE SA-1 CPU**

When power is applied to the Super NES control deck or its reset button is pressed, the SA-1 CPU is placed in its “stop” state. The Super NES CPU manipulates SA-1 internal registers to start and stop the SA-1 CPU as directed by software.

#### **5.2.1 STARTING THE SA-1 CPU**

The Super NES CPU sets the SA-1 CPU program start address into the RV register (2203H, 2204H) and resets the SA-1 CPU RES bit of the CCNT register (2200H) to “0” to initiate SA-1 CPU processing from the address set in the RV register.

#### **5.2.2 STOPPING THE SA-1 CPU**

When the Super NES CPU sets the SA-1 CPU RES bit of the CCNT register (2200H) to “1”, the SA-1 CPU stops processing and is placed in stop status.

## 5.3 MPU HANDSHAKES

Because the Super NES CPU and SA-1 CPU collaborate in processing programs, the SAS defines the following handshakes between the two MPUs.

### 5.3.1 INTERRUPTS

The Super NES CPU and SA-1 CPU can each transmit interrupts such as IRQ and NMI to each other, as listed in the following table.

Interrupt type	Direction	Register Set
IRQ	S → C	CCNT (2200H), SA-1 CPU IRQ bit = 1
NMI	S → C	CCNT (2200H), SA-1 CPU NMI bit = 1
IRQ	C → S	SCNT (2209H), Super NES CPU IRQ bit = 1
NMI	C → S	Not possible

Table 1-5-1 Types of Interrupts

An NMI interrupt cannot be sent from the SA-1 CPU to the Super NES CPU.

The MPU being interrupted identifies the source of the interrupt and clears the interrupt when the source is the other MPU.

Interrupt type	Direction	Interrupt Identification	Clear Register
IRQ	S → C	CFR (2301H) Super NES CPU IRQ bit	CIC (220BH) Super NES CPU IRQCL bit=1
NMI	S → C	CFR (2301H) Super NES CPU NMI bit	CIC (220BH) Super NES CPU NMI CL bit =1
IRQ	C → S	SFR (2300H) SA-1 CPU IRQ bit	SIC (2202H) SA-1 CPU IRQCL bit = 1

Table 1-5-2 Interrupt Identification and Clear

To temporarily block interrupts, they can be masked in an MPU.

Interrupt Type	Direction	Mask Register
IRQ	S → C	CIE (220AH), Super NES CPU IRQEN bit = 0
NMI	S → C	CIE (220AH), Super NES CPU NMIEN bit = 0
IRQ	C → S	SIE (2202H), SA-1 CPU IRQEN bit = 0

Table 1-5-3      Interrupt Mask

A masked interrupt becomes active after the mask is cancelled. To prevent this interrupt when the mask is cancelled, the programmer may use the interrupt identification registers, described in the table on the previous page, to identify an interrupt, then clear that interrupt before cancelling the mask.

### 5.3.2 MESSAGE

A four-bit message can be sent along with an interrupt signal between the MPUs, as described in the table below.

Interrupt Type	Direction	Register Sending the Message	Register Receiving the Message
IRQ	S → C	CCNT (2200H), SMEG0~3	SFR (2300H) CMEG0~3
NMI	S → C	CCNT (2200H) SMEG0~3	SFR (2300H) CMEG0~3
IRQ	C → S	SCNT (2209H) CMEG0~3	CFR (2301H) SMEG0~3

Table 1-5-4      Sending and Receiving a Message

## 5.4 SHARED MEMORY

Since SA-1 I-RAM can be accessed by both MPUs, a section of the SA-1 I-RAM can be used as a command exchange window. This window can be used in lieu of an interrupt to perform a handshake between the two MPUs. It also allows more command information to be sent than is possible with a "message", described previously. The size of shared memory in SA-1 I-RAM can be assigned by each program.

The SA-1 has a collision-control circuit for memory access, so that simultaneous read/write access by both MPUs does not cause any problems. If simultaneous access does occur, the Super NES CPU has priority access and the SA-1 CPU is put on hold.

The BW-RAM also has an area assigned to joint access and can be used as shared memory as well. However, it is generally best to use SA-1 I-RAM due to the RAM access speed (operating speed) and because BW-RAM cannot be used during character conversion DMA.

## 5.5 VECTOR SWITCHING

Parts of the Super NES CPU and SA-1 CPU vectors are registers in the SAS. This permits situation dependant multiple routines to be used. For example, interrupt processing can be expedited by preparing multiple IRQ routines in advance and setting the IRQ interrupt destination address in response to game situations.

Vectors which can be specified in registers include the following.

Vector Type	Destination Setting	Valid/Invalid Selection Bits
Super NES CPU NMI	SNV (220CH, 220DH)	SCNT Super NES CPU NVSW bit
Super NES CPU IRQ	SIV (220EH, 220FH)	SCNT Super NES CPU IVSW bit
SA-1 CPU reset	CRV (2203H, 2204H)	Always valid
SA-1 CPU NMI	CNV (2205H, 2206H)	Always valid
SA-1 CPU IRQ	CIV (2207H, 2208H)	Always valid

Table 1-5-5      Situation Dependant Vectors

When the Super NES CPU register setting vector is set to invalid, the program jumps to the address indicated in ROM.

## 5.6 SA-1 CPU CORE

The SA-1 core CPU is the same 16-bit CPU (65C816) used in the Super NES CPU and can execute all the Super NES instructions. The differences between the SA-1 CPU and Super NES CPU cores are as follows:

### 5.6.1 VECTORS

The reset, NMI, IRQ and other vectors registered in the M-ROM are for the Super NES CPU. The SA-1 CPU vectors must be set separately. The SA-1 CPU vectors should be set in the following registers using the Super NES CPU.

Reset vector:	RV (2203H, 2204H)
NMI vector:	CNV (2205H, 2206H)
IRQ vector:	CIV (2207H, 2208H)
Other vectors:	Same as the Super NES CPU (M-ROM data)

### 5.6.2 SA-1 CPU WAIT

The SA-1 CPU operates at 10.74 MHz, but a wait cycle may be introduced when some commands and functions are executed, or when it is accessed by the Super NES CPU. This happens when:

1. the following instructions are executed:

RTS, RTI, RTL, JMP (a), JML (a), JMP a, JMP al, JMP (a,x), JSR (a,x), JSR a, JSL al, BRA cop

2. the destination address of the following commands is odd:

BPL, BMI, BVC, BVS, BRA, BCC, BCS, BNE, BEQ, BRL

3. data is read from Game Pak ROM or BW-RAM.

4. the SA-1 CPU, Super NES CPU or the Super NES CPU's DMA access the same device (Game Pak ROM, BW-RAM, or SA-1 I-RAM) simultaneously.

5. the BW-RAM write buffer is full when writing to BW-RAM.

6. the source of the SA-1 DMA transmission is Game Pak ROM

## 5.7 OPERATION MODES

The SA-1 does not have special registers for setting the operation mode. The Super NES CPU is always in program execution state and controls the SA-1 CPU operations (start and stop).

The remainder of this chapter introduces representative relationships between the Super NES CPU and SA-1 CPU operations. They are examples and do not represent the entire SAS operation modes.

### 5.7.1 ACCELERATOR MODE

In the accelerator mode, the SA-1 CPU is used only to handle the high-load part of the program as subroutines. While the SA-1 CPU is processing, the Super NES CPU waits, in a loop, for the end of this processing. When the SA-1 CPU finishes processing, it informs the Super NES CPU by an interrupt, as illustrated below.

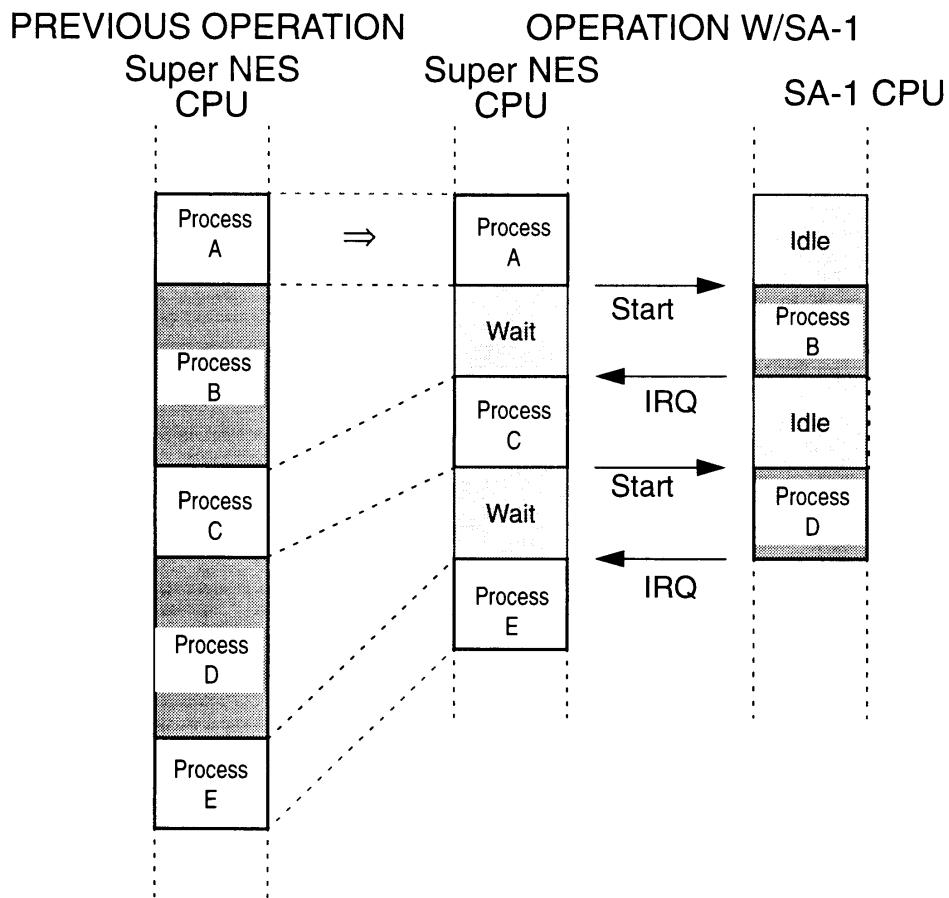


Figure 1-5-1 Accelerator Mode

In the accelerator mode, the process flow is like a single-thread and it is easy to avoid programming errors. This mode is suitable for utilizing the speed of SA-1 without much complexity. On the other hand, it is not very efficient due to MPU stop and loop time.

### 5.7.2 PARALLEL PROCESSING MODE

The parallel processing mode is a multi-processing mode in which both MPUs are operating simultaneously and are synchronized by handshakes. Both MPUs can freely access memory thanks to the SA-1's automatic collision control.

The handshake between MPUs is achieved by using interrupt signals and shared memory.

The SA-1 CPU can process the program while the Super NES CPU is processing the multi-use DMA, as demonstrated below.

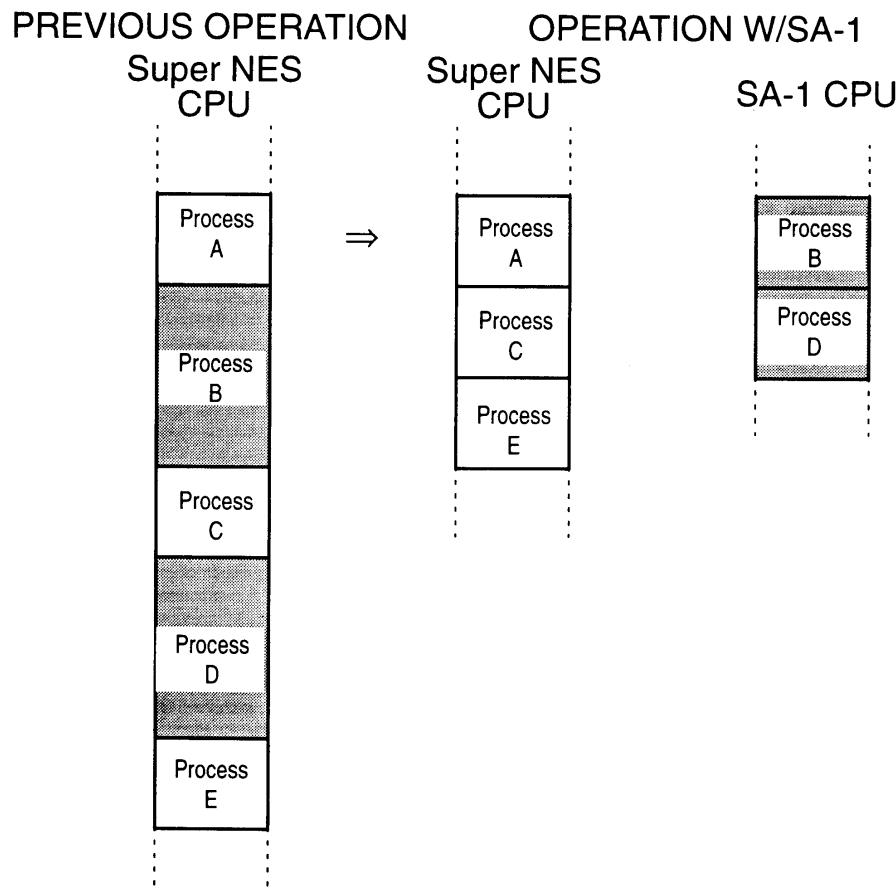


Figure 1-5-2 Parallel Processing Mode

In the parallel processing mode the highest processing efficiency can be achieved, as both MPUs operate without waiting for one another. However, the process flow is complicated and more care must be taken to avoid programming errors, unsuccessful handshakes, and crashes.

### 5.7.3 MIXED PROCESSING MODE

In the mixed processing mode, the SA-1 CPU can be used as a Super NES CPU accelerator during parallel processing in the parallel processing mode. In the SA-1, an operation mode is nothing more than program architecture, therefore, this type of processing is possible.

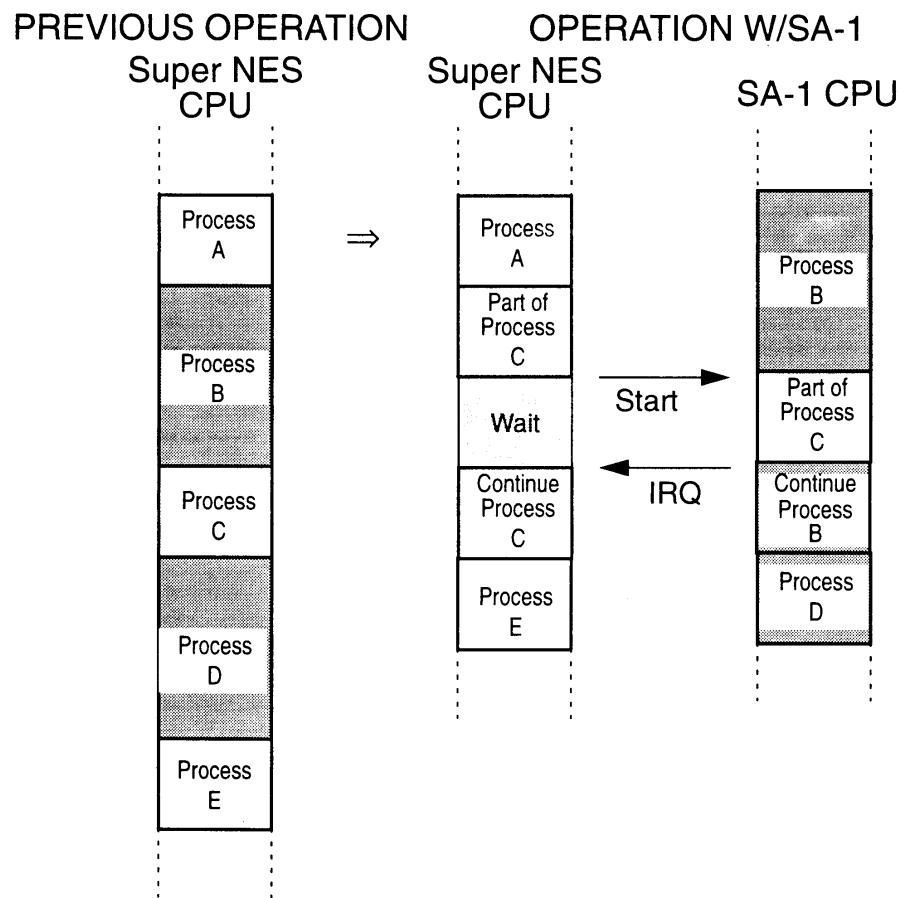


Figure 1-5-3 Mixed Processing Mode

## 5.8 OPERATING MODES AND PROCESSING SPEEDS

The operating speed of the SA-1 CPU in each of the SA-1 operating modes is as follows.

SA-1 Operation Mode	SA-1 CPU Operating Speed	Memory Used by SA-1 CPU	Super NES CPU Operations	Memory Used by Super NES CPU
Accelerator	10.74MHz	Game Pak ROM SA-1 I-RAM	Loop program	WRAM
Parallel Processing	10.74MHz	Game Pak ROM	Multi-purpose DMA	Other than Game Pak RAM
	10.74MHz	SA-1 I-RAM	Multi-purpose DMA	Other than SA-1 I-RAM
	5.37MHz	Game Pak ROM	Multi-purpose DMA	Game Pak ROM
	5.37MHz	Game Pak ROM	Normal operations	Game Pak ROM
	10.74MHz	SA-1 I-RAM	Normal operations	Game Pak ROM
	10.74MHz	Game Pak ROM SA-1 I-RAM	Normal operations	WRAM

Table 1-5-6      Operating Modes and Processing Speeds

## Chapter 6     *Character Conversion*

### 6.1 INTRODUCTION TO CHARACTER CONVERSION

The SA-1 contains a function for converting VRAM data stored in virtual bitmap format on BW-RAM and SA-1 I-RAM to Super NES PPU character format VRAM data.

Rotation, enlargement, and reduction of screen data and 3-D displays, such as polygons, are performed readily when the data is stored in bitmap format. Data compression can also be done more efficiently when the data to be compressed is stored in bitmap format.

#### 6.1.1 BITMAP FORMAT

“Bitmap format” refers to a data format where one address is assigned to each pixel (dot) on the screen. The SA-1 uses byte-long addresses. The effective data length is 2 bits in the 4 color mode and 4 bits in the 16 color mode. The remaining bits in the byte are ignored.

The Super NES PPU is incapable of directly processing bitmap data. The SA-1 includes a function which converts bitmap data to Super NES PPU character formatted data using DMA.

### 6.2 CHARACTER CONVERSION FUNCTIONS

The SA-1 has two character conversion functions for converting bitmap data to character data (Character Conversion 1 and Character Conversion 2).

#### 6.2.1 CHARACTER CONVERSION 1

Character Conversion 1 sends bitmapped data contained on BW-RAM to the VRAM of the Super NES PPU and displays it on the screen by simultaneously performing the DMA function in the SA-1 and Super NES general purpose DMA, as demonstrated below.

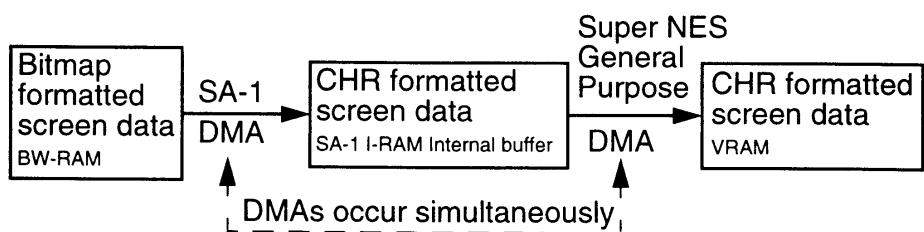


Figure 1-6-1     Character Conversion 1

Character conversion 1 uses the buffer area in SA-1 I-RAM to convert and transmit data to the VRAM of the Super NES PPU. The buffer can be a maximum of 128 bytes (256 color mode) or 32 bytes minimum (4 color mode).

### 6.2.2 CHARACTER CONVERSION 2

Character conversion 2 is used when the bitmap data is in SA-1 I-RAM or game pak ROM, or when the game pak is configured without BW-RAM,

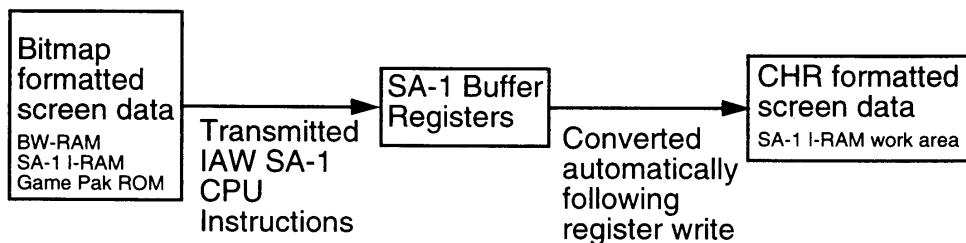


Figure 1-6-2 Character Conversion 2

### 6.3 BITMAP ACCESS

The bitmap data storage area (virtual VRAM) is normally assigned to BW-RAM. Bitmap data is compressed (packed) and stored in BW-RAM as illustrated below.

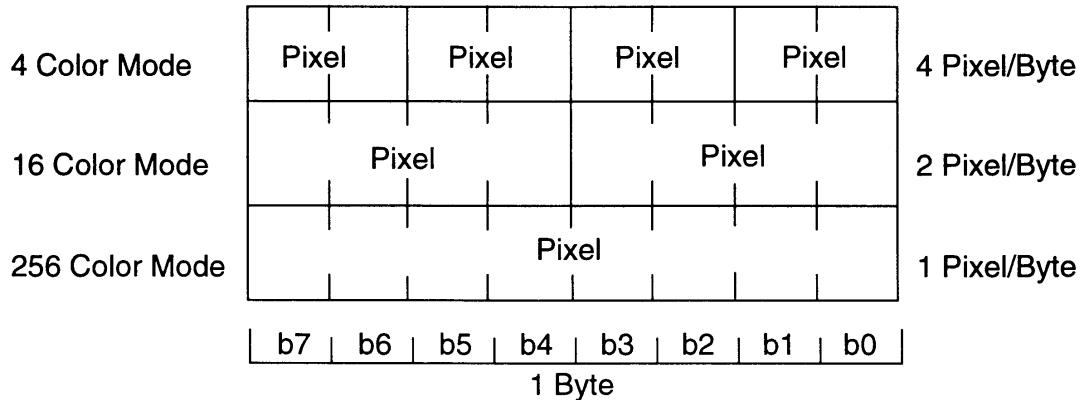


Figure 1-6-3 Compressed Bitmap Data

#### 6.3.1 BW-RAM IMAGE PROJECTION

Within the SA-1, the BW-RAM image is projected into  $6 \times H$  banks in the SA-1 CPU's memory map. When BW-RAM is accessed in these  $6 \times H$  banks, it can be accessed at one pixel per byte in either the 4 color or 16 color modes.

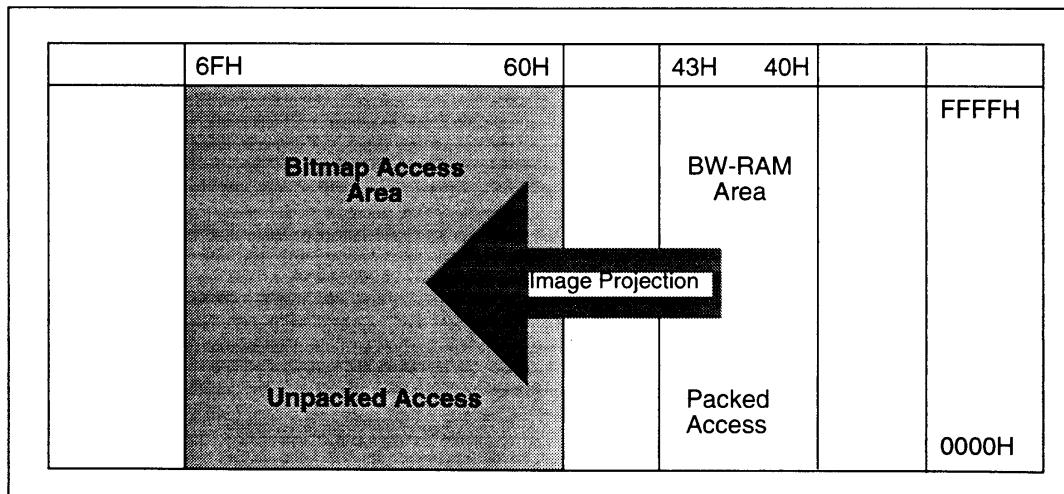


Figure 1-6-4 Bitmap Image Projection

**For 64 Kbit BW-RAM:**

BW-RAM                    Bitmap (16 color) / Bitmap (4 color)  
40:0000H~40:1FFFFH ⇒ 60:0000H~60:3FFFFH / 60:0000H~60:7FFFFH

**For 256 Kbit BW-RAM:**

BW-RAM                    Bitmap (16 color) / Bitmap (4 color)  
40:0000H~40:7FFFFH ⇒ 60:0000H~60:FFFFFH / 60:0000H~61:FFFFFH

**For 2 Mbit BW-RAM:**

BW-RAM                    Bitmap (16 color) / Bitmap (4 color)  
40:0000H~43:FFFFFH ⇒ 60:0000H~67:FFFFFH / 60:0000H~6F:FFFFFH

In the 256 color mode, the bitmap data is copied directly on the BW-RAM area.

### 6.3.2 BW-RAM DATA EXPANSION

The compressed BW-RAM data is expanded sequentially and assigned from address 60:0000H. This is demonstrated in the figure below. All areas of BW-RAM are expanded during this operation and no special register is provided for designating the expanded area. Therefore, when only a partial area of BW-RAM is used for virtual VRAM, the bitmap area corresponding to the area assigned as virtual VRAM must be accessed.

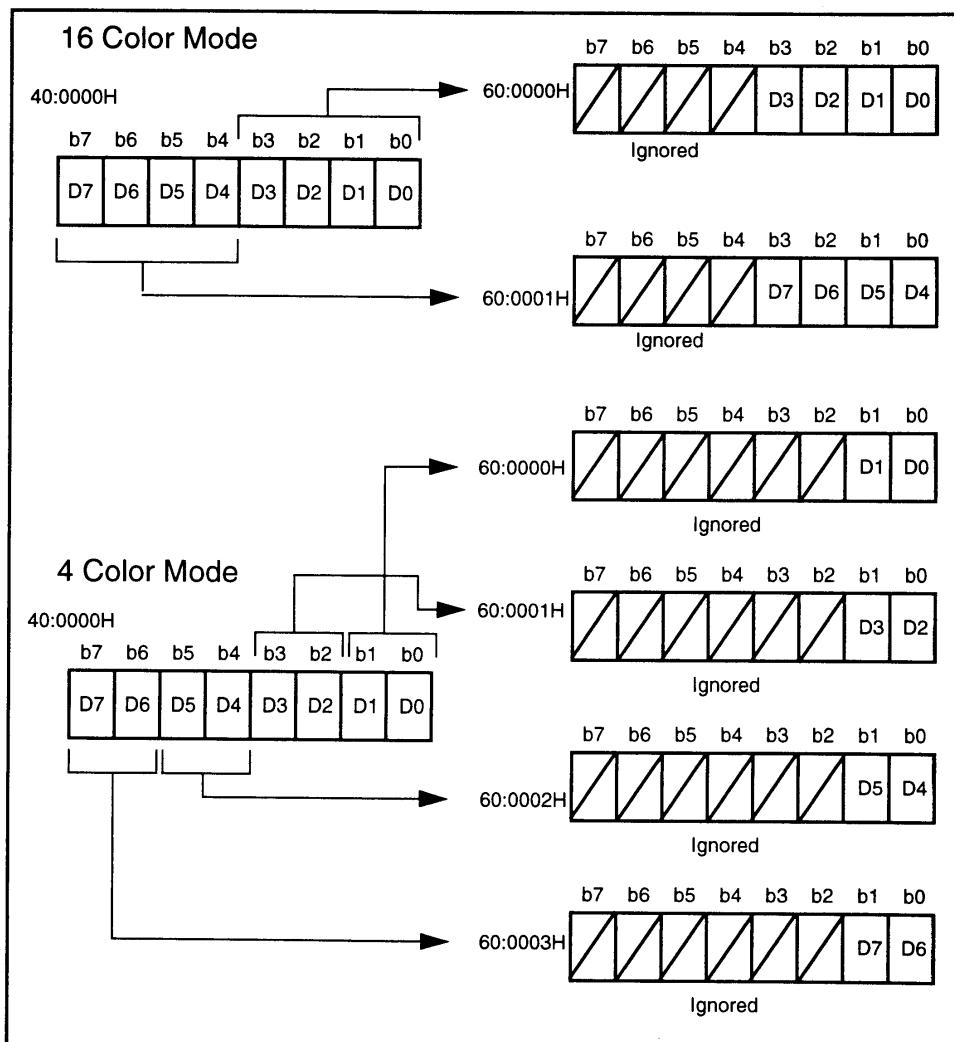


Figure 1-6-5    Bitmap Data Expansion

The color mode of the bitmap access area is set in bit SEL42 of the BBF register (223FH).

SEL42 = 0: 16 color mode

SEL42 = 1: 4 color mode

The bitmap area is configured as follows.

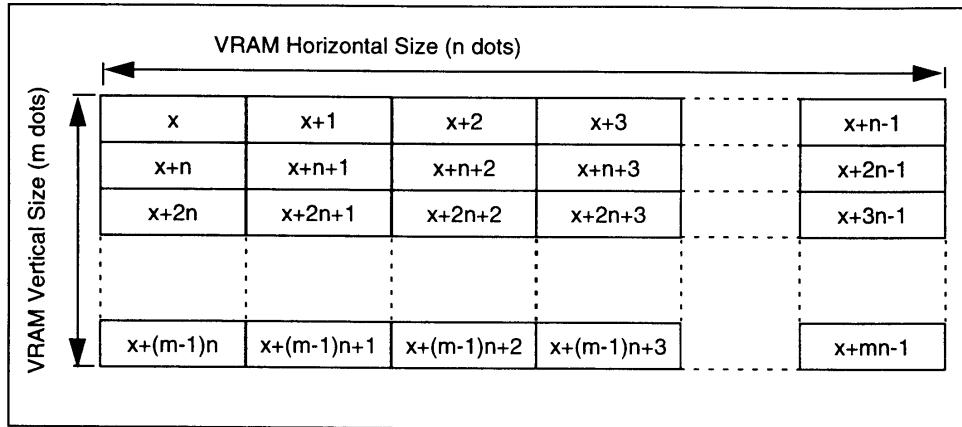


Figure 1-6-6 Memory Addresses for the Bitmap Area

The variable "x" indicates the start address of the bitmap area in virtual VRAM. The variable "n" is the horizontal size (dots) of VRAM and "m" is the vertical size (dots) of VRAM. Variable "n" can be specified in bits SIZE0~2 of the CDMA register (2231H), as demonstrated below. No register is provided for specifying vertical size "m". Vertical size can be set within the limits of BW-RAM size as a function of internal program logic. Variable "m" is processed in character units and must be a multiple of eight.

SIZE0	SIZE1	SIZE2	Horizontal Character Number
0	0	0	1 (8 dots)
0	0	1	2 (16 dots)
0	1	0	4 (32 dots)
0	1	1	8 (64 dots)
1	0	0	16 (128 dots)
1	0	1	32 (256 dots)

Table 1-6-1 Horizontal Size of VRAM (CDMA Register)

## 6.4 CHARACTER CONVERSION 1, DETAILED DESCRIPTION

Character conversion 1 is used to convert the bitmap screen data in BW-RAM to Super NES PPU character formatted VRAM data, with SA-1 DMA and Super NES general purpose DMA working in parallel. A larger volume of data can be converted at one time with character conversion 1 than with character conversion 2, due to efficient usage of both DMAs.

Character conversion 1 requires two characters of memory space in SA-1 I-RAM for use as buffers (work space). The required I-RAM size is 32 bytes in the 4 color mode, 64 bytes in the 16 color mode, and 128 bytes in the 256 color mode. Any I-RAM address can be specified by the user.

Character conversion 1 uses these two buffers to read the data from BW-RAM to VRAM in parallel. Since the processing speed is determined by the speed of the Super NES CPU's general purpose DMA, the same amount of characters can be converted as with the Super NES, alone.

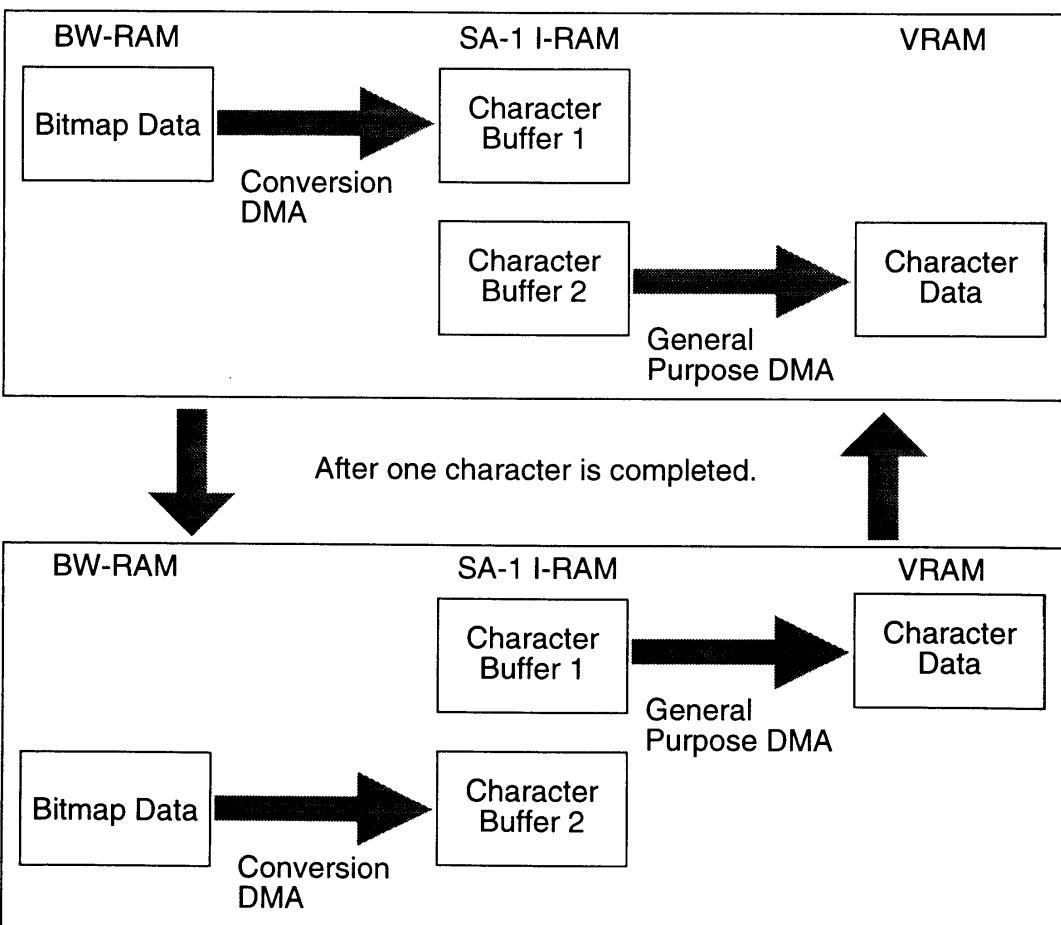


Figure 1-6-7 Character Conversion Buffers

## 6.5 CHARACTER CONVERSION 1 PROGRAMMING PROCEDURE

When character conversion 1 is used, the user must carefully coordinate register settings in the Super NES CPU and SA-1 CPU. The following procedure is provided to aid the user in coordinating these settings.

**STEP 1.** Set DCNT (2230H) using the SA-1 CPU.

CDEN bit = 1 (character conversion enable)

CDSEL = 1 (BW-RAM to SA-1 I-RAM transmission)

**NOTE:** The registers indicated in the following steps are set using the Super NES CPU.

**STEP 2.** Specify the SA-1 DMA transmission source address using the Super NES CPU.

Store the transmission source address (BW-RAM) in SDA (2232H~2234H).

A specific number of low bit of the address must be set to "0", as a function of the color mode and the number of horizontal characters set in SIZE0~2 of CDMA (2231H). The specific number of "0" bits can be determined from the table below.

Color Mode	4	4	4	4	4	4	16	16	16	16	16	256	256	256	256	256	
Number of Horizontal Characters	1	2	4	8	16	32	1	2	4	8	16	32	1	2	4	8	16
Zero Bits	4	5	6	7	8	9	5	6	7	8	9	10	6	7	8	9	10

Table 1-6-2 Number of Zero Bits in BW-RAM

**STEP 3.** Set CDMA (2231H) using the Super NES CPU.

Store the color mode (4, 16, or 256) in CB0 and CB1.

Store the number of virtual VRAM horizontal characters in SIZE0~2.

**STEP 4.** Specify the SA-1 I-RAM address for the buffers as the transmission destination.

Store the buffer address in DDA (2235H and 2236H).

**NOTE:** It is not necessary to set 2237H because I-RAM is specified.

The lowest 5 bits of the I-RAM address must all be "0" for 4 color mode. The lowest 6 bits of the I-RAM address must be "0" for 16 color mode. And, the lowest 7 bits of the I-RAM address must be "0" for 256 color mode.

- STEP 5. Wait for the IRQ (CHRIIRQ) generated from SA-1 to the Super NES CPU.

The Super NES CPU waits for the IRQ and verifies that the CHRDMA IRQ bit of the SFR register (2300H) = 1 (character conversion 1 DMA standby). IRQ is generated for some other reason when CHRDMA IRQ = 0.

- STEP 6. Transmit the character data in SA-1 I-RAM to VRAM.

Character data which has been converted by the Super NES CPU's general purpose DMA is transmitted to VRAM. Set the general purpose DMA source address to the start address of the virtual VRAM in BW-RAM.

- STEP 7. Use the Super NES CPU to notify the SA-1 that the conversion is complete.

Set bit CHDEND of the CDMA register (2231H) to "1" to indicate that one cycle of character conversion 1 has been completed and return control of register access to the SA-1 CPU.

When necessary, use an IRQ or SA-1 I-RAM to notify the SA-1 CPU of the end of character conversion.

Using the above procedure, the SA-1 internal character conversion circuit converts characters in order based upon the request from the Super NES CPU's DMA.

The SA-1 CPU can return to program processing after STEP 1 has been performed, however, it must wait during any simultaneous access to BW-RAM or SA1 I-RAM as DMA has priority.

Although CDMA, DDA, and SDA are SA-1 CPU registers, they are set by the Super NES CPU when using character conversion 1. The user should not access BW-RAM from the Super NES CPU during these operations. SA-1 I-RAM can be accessed by the user through the Super NES CPU, so flags can be changed within the SA-1 CPU.

## 6.6 CHARACTER CONVERSION 2, DETAILED DESCRIPTION

Character conversion 2 performs character conversion by writing bitmap data to the SA-1 registers according to the SA-1 CPU's program. Because the transmission is controlled by the SA-1 CPU's program, the memory for bitmap data expansion can be set up more freely than when using character conversion 1. Also, when the game pak configuration does not include BW-RAM, character conversion 2 is the only means of character conversion.

The bitmap data when using character conversion 2 is one pixel/byte (unpacked). As previously described, packed data cannot be converted. Therefore, bits b7 ~ b2 of the data are invalid in the 4 color mode. Similarly, b7 ~ b4 are invalid in the 16 color mode. All bits are valid in the 256 color mode.

The table below shows the actual data in memory. When the bitmap access function is used with character conversion 1, one pixel/byte access is possible.

	Bitmap Data Format	
	4 Color Mode	16 Color Mode
Character Conversion 1	4 Pixel/Byte	2 Pixel/Byte
Character Conversion 2	1 Pixel/Byte	1 Pixel/Byte

Table 1-6-3      Character Conversion and Data Format

Character conversion 2 also requires buffers for two characters in SA-1 I-RAM, similar to character conversion 1. The bitmap data written to the SA-1 registers by the SA-1 CPU is converted as written and generated as character data in the buffer area in SA-1 I-RAM. Character conversion is performed using the two SA-1 buffers alternately. When the conversion of data contained in buffer 1 is completed, conversion begins on the data contained in buffer 2. When this conversion is completed, new data contained in buffer 1 is converted. The Super NES CPU reads the data from the buffer in the SA-1 I-RAM at the end of each conversion using its general purpose DMA.

## 6.7 CHARACTER CONVERSION 2 PROGRAMMING PROCEDURE

The following procedure is provided to aid the user in executing character conversion 2.

**STEP 1.** Set DCNT (2230H) using the SA-1 CPU.

DMAEN = 1 (DMA enable)

CDEN = 1 (character conversion DMA)

CDSEL = 0 (SA-1 CPU to SA-1 I-RAM write)

No other bits need to be set.

**STEP 2.** Store the color mode in CDMA (2231H) using the SA-1 CPU.

The color mode is set using bits CB0 and CB1 (4, 16, or 256 color modes). Bits SIZE0~2 need not be set.

**STEP 3.** Specify the SA-1 I-RAM transmission destination address using the SA-1 CPU.

Store the I-RAM buffer address in DDA (2235H and 2236H).

The lowest 5 bits of the I-RAM address must be set to all zeros for 4 color mode. The lowest 6 bits must be zero for 16 color mode. The lowest 7 bits must be zero for 256 color mode.

**STEP 4.** Write the bitmap data in the conversion register using the SA-1 CPU.

The data must be written 4 times in succession (64 pixels = 1 character of data) to BRF (2240H~224FH).

The 4 write operations should be performed in the following order.

BRF0→1→2→... F→0→1→... →F

Character conversion DMA will begin automatically, following each 8 pixel write operation and generate the characters in I-RAM.

**STEP 5.** Notify the Super NES CPU that character conversion is complete.

Notify the Super NES CPU using an interrupt or SA-1 I-RAM when a character has been completed.

The Super NES CPU transmits the character data to VRAM or WRAM using general purpose DMA or a program.

**STEP 6.** Repeat STEP 4 and 5 to continue character conversion.

To continue to convert characters, write 64 pixels in succession. The character data is created using DMA transmission in the other SA-1 I-RAM buffer.

**STEP 7.** Indicate when character conversion is over.

Reset bit DMAEN of the DCNT register (2230H) to "0". This ends one cycle of character conversion 2.

During these operations, other SA-1 DMA functions cannot be performed. The Super NES general purpose DMA may be used for other functions.

## **Chapter 7      Arithmetic Function**

### **7.1 TYPES OF ARITHMETIC OPERATIONS**

The SA-1 has an arithmetic circuit for high speed processing of arithmetic operations. This is in addition to the arithmetic circuit installed in the Super NES PPU. The SA-1 arithmetic circuit runs faster and can run concurrently with the Super NES CPU. The SA-1 arithmetic circuit performs the following three types of arithmetic functions.

#### **1. MULTIPLICATION**

Multiplicand	Multiplier	Result
16 bits (S)	$\times$	16 bits (S) = 32 bits (S)

#### **2. DIVISION**

Dividend	Divisor	Result
16 bits (S)	$\div$	16 bits (U) = 16 bits (S) 16 bits (U) Remainder

#### **3. CUMULATIVE SUM**

Multiplicand	Multiplier	Result
$\Sigma(16 \text{ bits (S)}$	$\times$	16 bits (S)) = 40 bits (S)

Note: (S) indicates signed data and (U) indicates unsigned data.

The type of arithmetic operation is specified in the arithmetic operation control register (\*\*2250H) using the SA-1 CPU. The user should choose between ACM (d1) for cumulative sum operations and M/D (d0) for multiplication or division operations. The required number of cycles for each operation are shown below.

Arithmetic Operation	ACM	M/D	Number of Cycles
Multiplication	0	0	5
Division	0	1	5
Accumulative	1	-	6

Table 1-7-1      Arithmetic Operations Settings and Cycles

The number of cycles is calculated based upon 10.74 MHz per cycle.

## 7.2 MULTIPLICATION

Multiplication operations are carried out as follows.

1. Set MCNT (2250H)  
ACM=0, M/D=0
2. Set the arithmetic parameters.  
Store the multiplicand in MA (2251H and 2252H).  
Store the multiplier in MB (2253H and 2254H).
3. Read the result after 5 cycles.

The arithmetic result is stored in W0~W3 of MR (2306H~2309H).

W0 is the lowest byte and W3 the highest.

The multiplicand is saved in memory following the operation, while the multiplier is not.

## 7.3 DIVISION

Division operations are carried out as follows.

1. Set MCNT (2250H)  
ACM=0, M/D=1
2. Set the arithmetic parameters.  
Store the dividend in MA (2251H and 2252H).  
Store the divisor in MB (2253H and 2254H).
3. Read the result after 5 cycles.  
The arithmetic result is stored in W0 and W1 of MR (2306H and 2307H).  
The remainder is stored in W2 and W3 of MR (2306H and 2307H).  
W0 and W2 are the low bytes, while W1 and W3 are the high bytes.

Neither the dividend nor the divisor is saved in memory.

The SA-1 does not detect “divide by zero” errors. The product and remainder for division by zero will be “0”. Special attention is required to the sign of the remainder in division when using negative numbers.

## 7.4 CUMULATIVE SUM

Cumulative sum operations are carried out as follows.

1. Set MCNT (2250H)

ACM=1

When the ACM bit is set (1) the cumulative result is cleared to "0".

2. Set the arithmetic parameters.

Store the multiplicand in MA (2251H and 2252H).

Store the multiplier in MB (2253H and 2254H).

3. Reset the parameters after 6 cycles.

Repeat this step until the operation is completed.

4. Read the cumulative result.

The arithmetic result is stored in W0~W3 of MR (2306H~2309H).

W0 is the lowest byte and W3 the highest.

The multiplicand is saved in memory following the operation, while the multiplier is not.

The OF bit in the OF register (230BH) is set to "1" when the cumulative result exceeds 40 bits.

## ***Chapter 8      Variable-Length Bit Processing***

### **8.1 READING VARIABLE-LENGTH DATA**

The SA-1 variable-length bit processing function consists of a barrel shift circuit which treats the entire game pak ROM as a stream (string) of bits which are sequentially read in 1 ~ 16 bit lengths. This allows the SA-1 to process data of variable lengths without having to shift the data to byte boundaries, resulting in higher processing speed.

The SA-1 variable-length bit processing function consists only of a barrel shift function. The function supports, but does not perform data compression or expansion. These processes must be performed as a part of each program.

The function is configured in this way to allow the programmer to select the best compression algorithm for each piece of software, in order to achieve the optimal processing speed-compression rate combination.

The SA-1 variable-length bit processing function includes two data read modes, the Fixed Mode and the Auto-increment Mode.

The data read mode is specified in the HL bit of the VBD register (2258H).

HL=0:      Fixed Mode

HL=1:      Auto-increment Mode

## 8.2 FIXED MODE

In the Fixed Mode, the data stored in the variable-length data port will be read over and over until the number of bits to be barrel shifted is reached. The shift is carried out when the amount of the shift is written to the VBD register (2258H). The Fixed Mode is used to read data which is formatted so that the valid bit length is known only after the data is read. Variable-length data is processed as follows in the Fixed Mode.

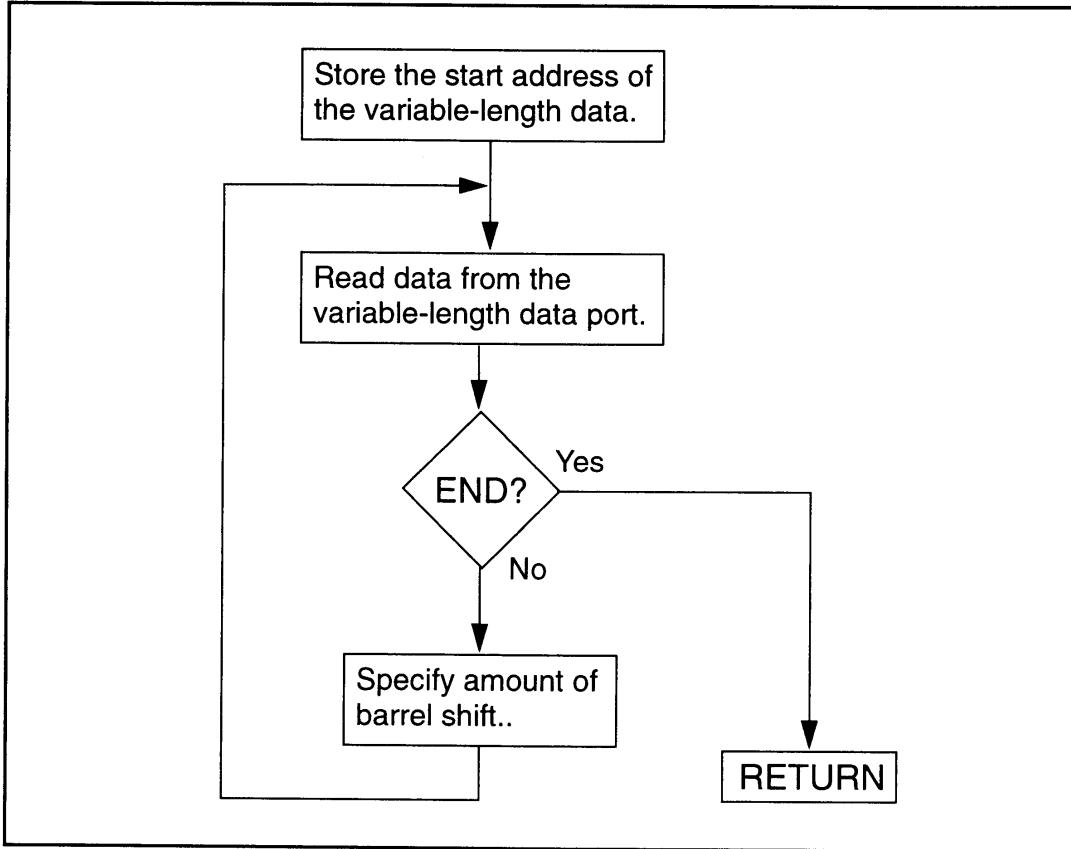


Figure 1-8-1 Fixed Mode Process Flow Diagram

### 8.3 AUTO-INCREMENT MODE

In the Auto-increment Mode, the amount of the barrel shift is specified in advance. Data is shifted automatically following the data read and the next data is placed on standby.

The Auto-increment Mode is used when the valid bit length of data is known in advance or when data of the same length is to be repeated. Variable-length data is processed as follows in the Auto-increment Mode.

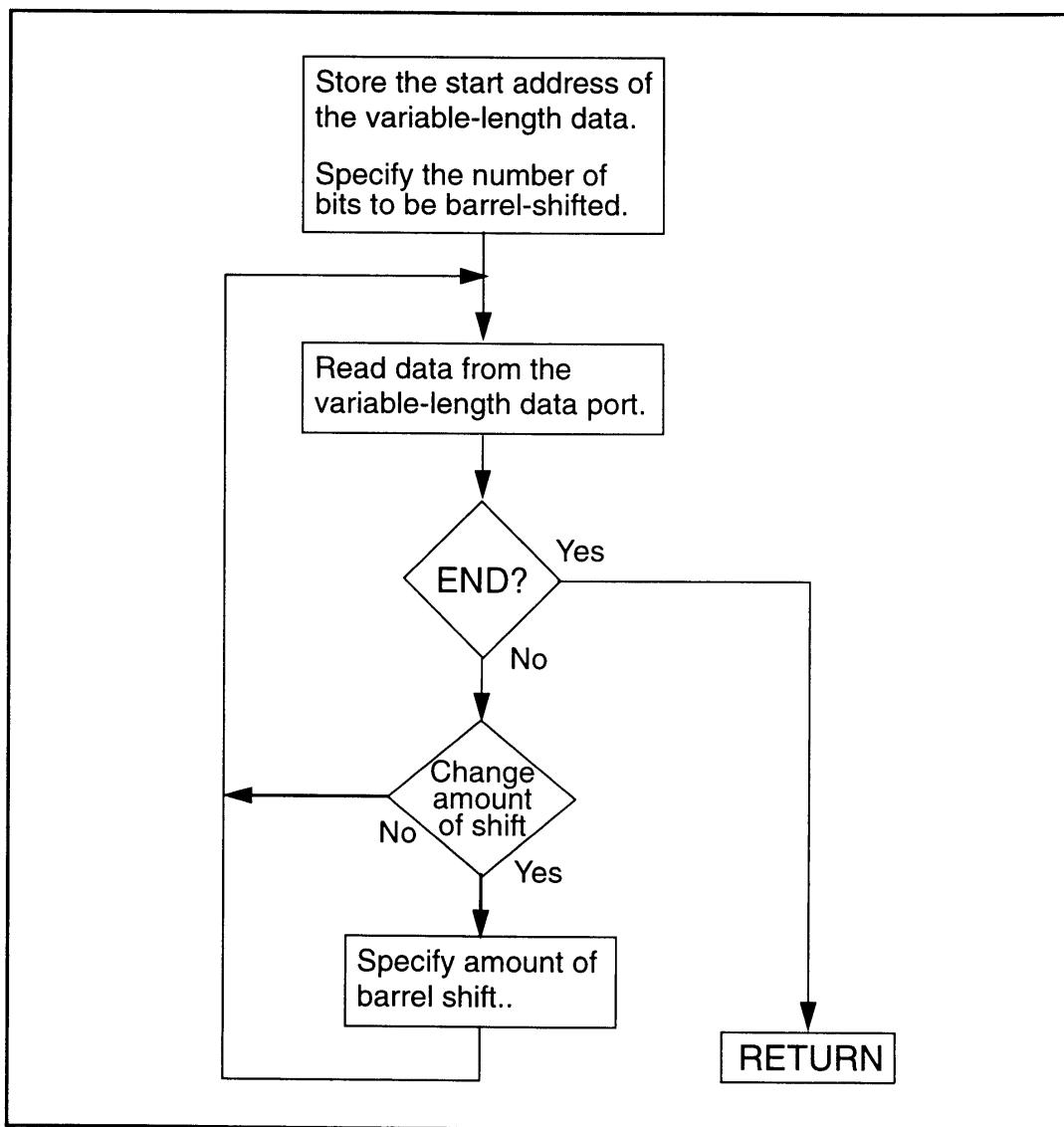


Figure 1-8-2 Auto-increment Mode Process Flow Diagram

## 8.4 VARIABLE-LENGTH DATA PROCESSING SETTINGS

Specify the number of bits to be shifted and parameters for the SA-1 variable-length data read in the following registers.

**STEP 1.** Set variable-length data start address.

Store the start address of the variable-length bit stream in the VDA register (2259H~225BH).

**STEP 2.** Perform variable-length data read.

Read variable-length data from the VDP register (230CH and 230DH).

An LSB-justified 16 bit block of data is read from the start of the remaining bit stream.

**STEP 3.** Set the amount of the barrel shift.

Store the amount of the barrel shift in bits VB0~VB3 of the VBD register (2258H).

VB3	VB2	VB1	VB0	Significant Bit Length
0	0	0	0	16
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Table 1-8-1 Amount of Barrel Shift

The barrel shift is carried out from MSB to LSB and the next data is read into the vacant MSB. This flow is demonstrated in the following illustration.

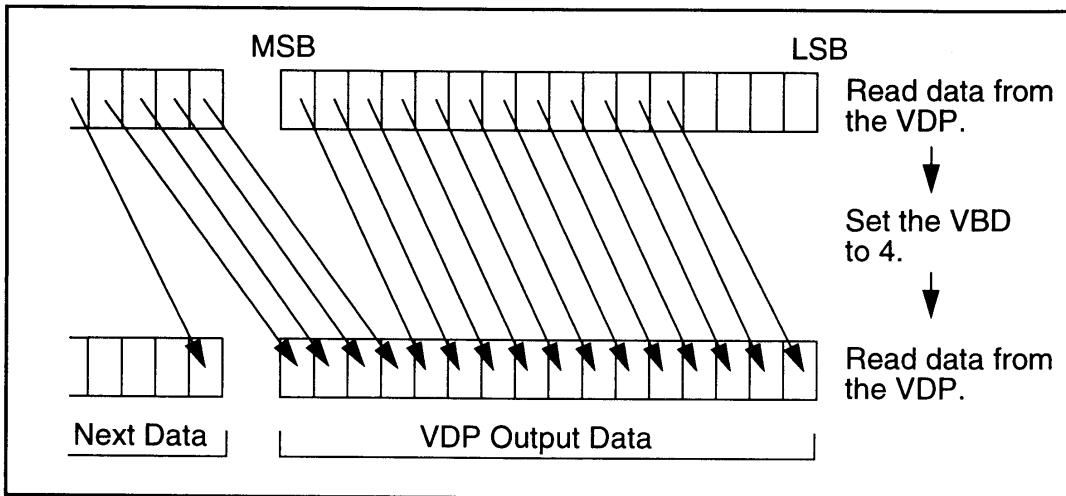


Figure 1-8-3 Barrel Shift Process

When specifying the amount of barrel shift, the number of bits from the word boundary is specified. For example, when 2-bit blocks of data are used;

- set VB3~0 to 0010 (2) for the first shift,
- set VB3~0 to 0100 (4) for the second shift, and
- set VB3~0 to 0110 (6) for the third shift.

Note that the data set in the VB bits is not the number of bits to be discarded, but rather the number of unnecessary bits counting from the word boundary.

## Chapter 9 DMA

### 9.1 TYPES OF DMA

The SA-1 internal DMA function transfers data between game pak ROM, BW-RAM, and SA-1 I-RAM. SA-1 internal DMA can be operated independent of the Super NES CPU's general purpose DMA and H-DMA. Even when both DMAs access the same memory at the same time, no problems arise because memory access is exclusive.

SA-1 internal DMA has two basic operation modes. The Normal DMA Mode is used to transfer data between memories, while the Character Conversion DMA Mode is used to transmit data while converting from bitmap format to character format. This chapter describes the Normal DMA Mode. Refer to the previous chapter, "Character Conversion", for details concerning the Character Conversion DMA Mode.

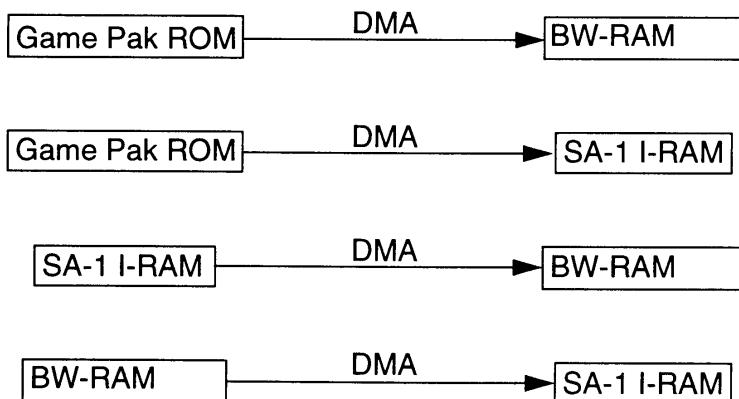


Figure 1-9-1 Normal DMA

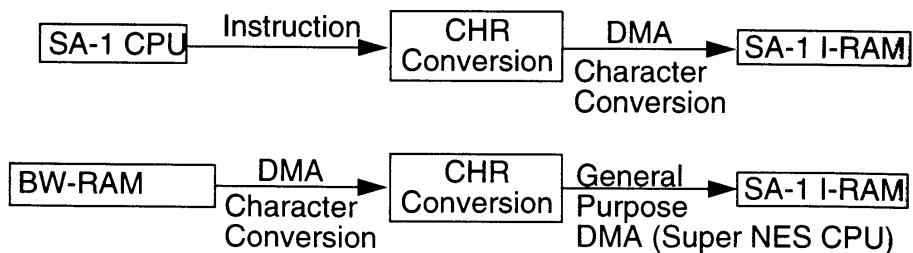


Figure 1-9-2 Character Conversion DMA

## 9.2 NORMAL DMA OPERATION

All Normal DMA is started from the SA-1 CPU. The DMA-related registers (2230H~2239H) are used to start DMA, as described in the following procedure.

**STEP 1.** Set the DCNT register (\*\*2230H).

Store the transmission source device in bits SD0 and SD1.

Store the transmission destination device in bit DD.

**NOTE:** The same device cannot be used for source and destination.

Source Device

Destination Device

SD1	SD0	Device
0	0	Game Pak ROM
0	1	BW-RAM
1	0	SA-1 I-RAM

DD	Device
0	SA-1 I-RAM
1	BW-RAM

Store the transmission mode in bit CDEN.

CDEN=0: Normal DMA

CDEN=1: Character Conversion DMA

Set DPRIO (d6) to assign priority between SA-1 CPU and DMA.

DPrO=0: SA-1 CPU priority (Instructions can be executed during transmission)

DPrO=1: DMA priority (SA-1 CPU waits during DMA)

**NOTE:** The DPrO setting is only valid during Normal DMA between BW-RAM and SA-1 I-RAM.

Set DMAEN to enable or disable DMA.

DMAEN=0: DMA disable (DMA is not used)

DMAEN=1: DMA enable (Use DMA, clear parameters)

When setting the DMA parameters, first set DMAEN=1 from the SA-1 CPU and then set the other parameters. Set DMAEN=0 after the DMA has been completed.

**STEP 2. Specify the start address of the transmission source.**

Store the transmission source start address in the SDA register (2232H~2234H). The bit length varies according to the source device.

Source Device	Bit Number Setting	Register
Game Pak ROM	24 bits	**2232H, 2233H, 2234H
BW-RAM	18 bits	**2232H, 2233H, 2234H
SA-1 I-RAM	11 bits	**2232H, 2233H

Table 1-9-1      Source Device Settings

When transmitting from game pak ROM, start from the even address. When transmitting from BW-RAM, transmit from bank 40H~43H. No transmissions can be sent from a bitmap access area.

**STEP 3. Set the number of bytes for transmission.**

Store the number of bytes for transmission in the DTC register (2238H and 2239H). The value set in DTC is transferred to the internal counter in the DMA circuit (terminal counter). The DTC range is from 1~65535 bytes.

**STEP 4. Specify the transmission destination start address.**

Store the transmission destination start address in the DDA register (2235H~2237H). The bit length varies according to the destination device.

Destination Device	Bit Number Setting	Register	Start Trigger
BW-RAM	18 bits	**2235H, 2236H, 2237H	**2237H
SA-1 I-RAM	11 bits	**2235H, 2236H	**2236H

Table 1-9-2      Destination Device Settings

When transferring data to BW-RAM, send the data to banks 40H~43H. Data cannot be sent to the bitmap access area. The DMA circuit begins the transmission after the trigger address has been written.

Normal DMA transmission ends when the internal terminal counter reaches 0. After normal DMA ends, an IRQ is generated from the DMA circuit to the SA-1 CPU to set the DMAIRQ flag in the CFR register (2301H) to "1".

### 9.3 DMA TRANSMISSION SPEED

The transmission speeds for Normal DMA are as follows.:

Type of DMA	Frequency
Game Pak ROM to SA-1 I-RAM	10.74 MHz
Game Pak ROM to BW-RAM	5.37 MHz
BW-RAM to SA-1 I-RAM	5.37 MHz
SA-1 I-RAM to BW-RAM	5.37 MHz

Table 1-9-3 DMA Transmission Speed

When the Super NES CPU's general purpose DMA or H-DMA generates an access during the SA-1's internal DMA transmission, the SA-1 internal DMA is put in the "wait" state. Hence, the Super NES CPU's DMA has priority.

## **Chapter 1      *Introduction to Super FX<sup>TM</sup>***

The Super FX is a Graphic Support processing Unit (GSU) designed to greatly improve the Super NES graphics and mathematical functions through the use of the following special features.

### **1.1 FEATURES**

#### **1.1.1 RISC-LIKE INSTRUCTIONS**

Instructions which are utilized often consist of only one byte and are executed in one cycle in an instruction cache.

#### **1.1.2 HIGH SPEED CLOCK OPERATION**

The current version of the Super FX operates at a clock speed of 10.74MHz. This is six times as fast as the Super NES CPU.

#### **1.1.3 BUILT-IN INSTRUCTION CACHE**

A 512-byte cache RAM is installed in order to perform the instructions at high speed. (Refer to "Cache RAM".)

#### **1.1.4 SUPER NES CPU'S MEMORY MAY BE USED**

The Super FX uses game pak ROM and RAM which is currently used by the Super NES CPU. (Refer to "Memory Mapping".)

#### **1.1.5 INDEPENDENT ROM AND RAM BUSES**

The Super FX can access game pak ROM and RAM in parallel. Program processing speed is maximized, as buffers are provided to read from ROM and write to RAM. (Refer to "Program Execution".)

#### **1.1.6 PARALLEL OPERATIONS WITH SUPER NES CPU**

The Super NES CPU and Super FX may execute processing in parallel. Thus, high speed operations can be performed.

#### **1.1.7 GRAPHICS FUNCTION**

A fast plot process can be performed by specifying a coordinate corresponding with the Super NES PPU format. (Refer to "Bitmap Emulation", under "Super FX Special Functions".)

#### **1.1.8 PIPELINE PROCESSING**

Pipeline processing reduces the number of processing cycles and enables high speed operation. (Refer to "Pipeline Processing", under "Instruction Set General Description".)

## 1.2 SPECIAL CONVENTIONS

Unless otherwise specified, addresses will be written with a 2 digit hexadecimal bank number and a 4 digit hexadecimal address separated by a colon (:). The following example demonstrates this convention.

3F:0000H

In this example “3F” represents the bank number, while “0000” represents the hexadecimal address.

### 1.3 SYSTEM CONFIGURATION

The GSU is installed on each game pak with ROM and RAM as demonstrated below. The Super NES CPU and the GSU share game pak ROM and RAM. Additional ROM for the Super NES CPU and back-up RAM may also be installed.

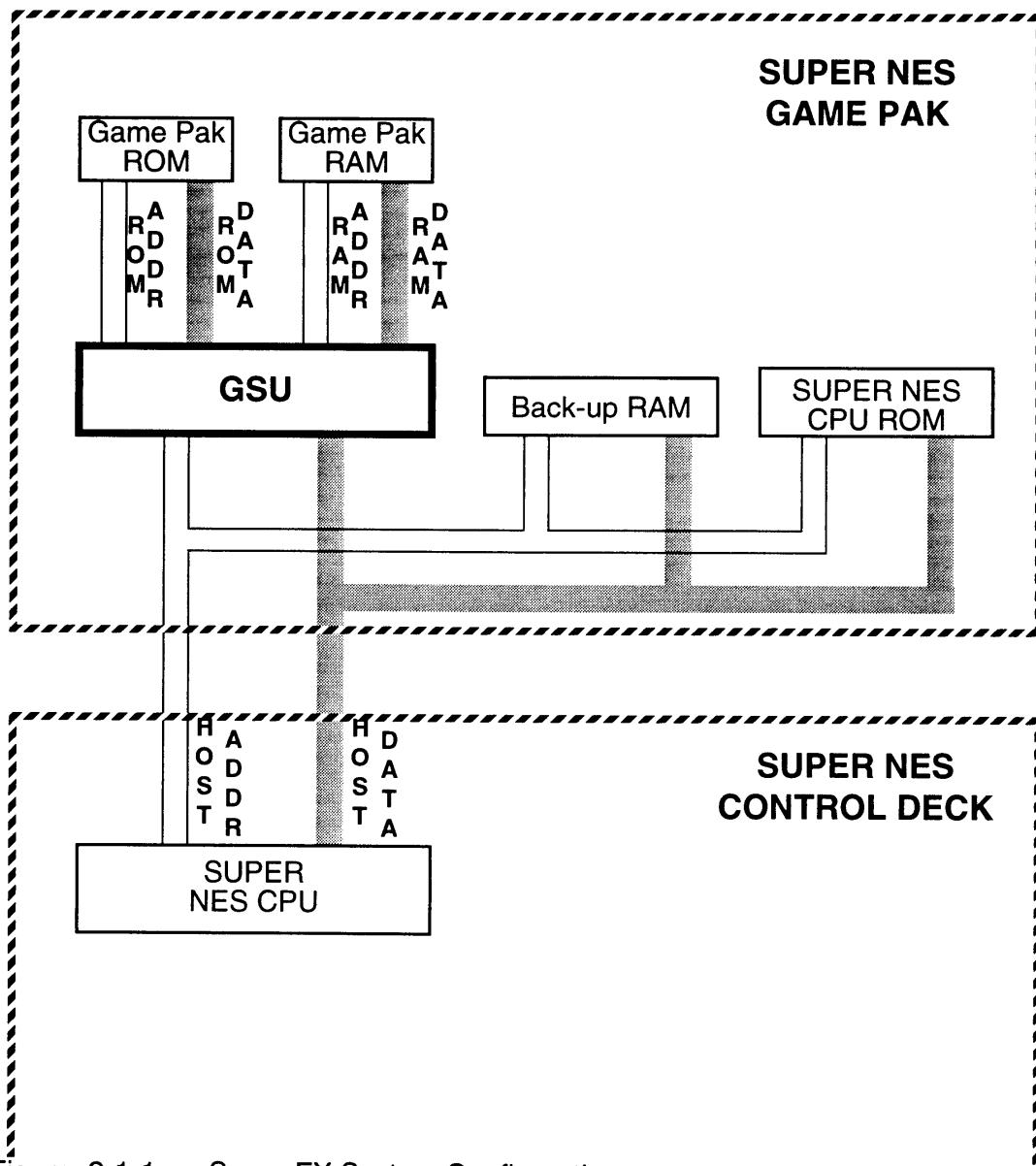


Figure 2-1-1    Super FX System Configuration

## 1.4 SYSTEM OPERATION

Although the Super NES CPU and GSU share game pak ROM and RAM, the processors can not access either simultaneously. The GSU has a flag, controlled by the Super NES CPU program, which determines whether the CPU or GSU have access to game pak ROM and/or RAM. This is demonstrated in the following figure.

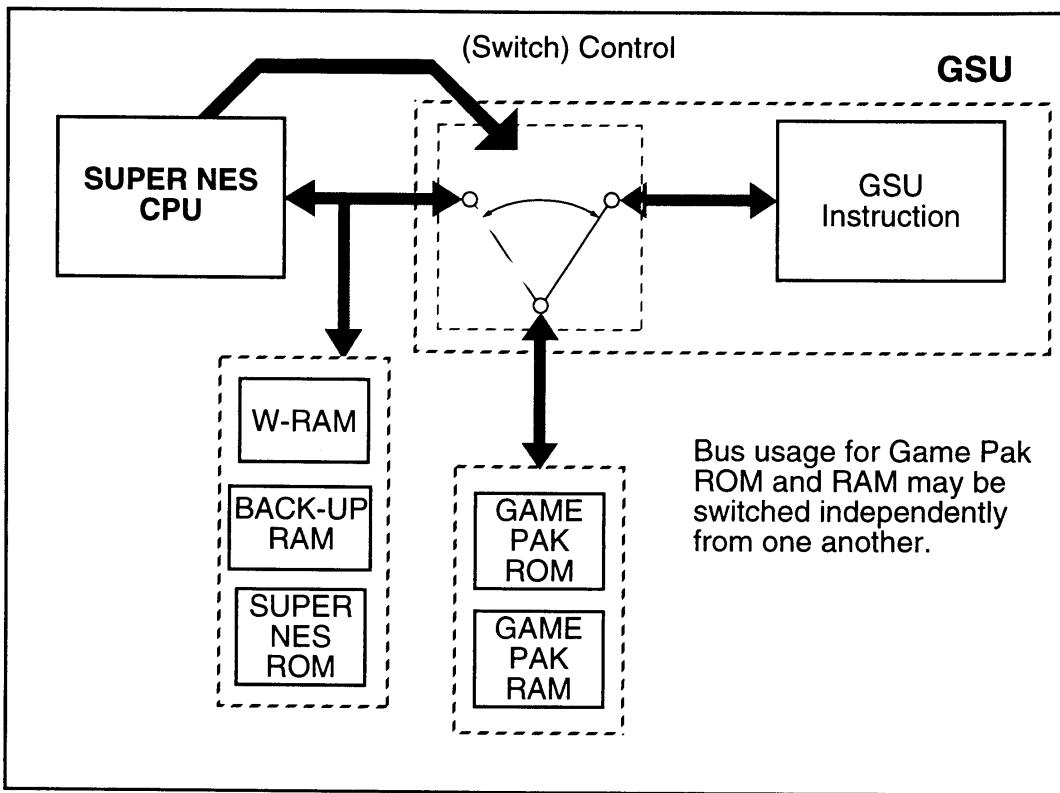


Figure 2-1-2 Game Pak ROM/RAM Bus Diagram

When using the GSU, the program must be written and executed with these points in mind. The following example demonstrates recommended usage of the GSU.

## 1.5 EXAMPLE OF USAGE

### 1.5.1 RESET SUPER NES

When the Super NES is reset, the GSU is also reset. In this condition the game pak ROM and RAM busses are connected to the Super NES CPU. The program stored in game pak ROM is processed by the Super NES CPU. The GSU is idle during this period.

### **1.5.2 WRAM**

The Super NES CPU is used to move the program from game pak ROM to the work RAM (WRAM) mounted within the Super NES Control Deck. The Super NES CPU may then be operated by this WRAM program.

### **1.5.3 ACTIVATION OF GSU**

The GSU flag is set by the Super NES CPU. This allows the GSU to process instructions stored in game pak ROM and store results in game pak RAM.

### **1.5.4 GSU STOP COMMAND**

When the GSU completes the desired processing, a stop command is executed. The GSU stops processing and generates an interrupt to the Super NES CPU. This notifies the Super NES CPU that the GSU has completed its processing.

### **1.5.5 GSU DISCONNECT**

When the GSU stops, game pak ROM and RAM busses are again connected to the Super NES CPU. This permits the Super NES CPU to process the results of the GSU's computations.

### **1.5.6 EXAMPLE SUMMARY**

This process may have been used, for example, to produce game video data. These programming steps are then repeated, as necessary, to accomplish the programmer's desired result.

### **1.5.7 CURRENT CONSUMPTION**

A game pak which contains the Super FX is required to have a built-in safety program to prevent it from operating in excess of the maximum current rating of the AC Adapter. For example, a game pak which contains the Super FX can not be used with Multi Player 5 because this would exceed the maximum current rating. A program must be included within the game pak which will check accessory IDs and activate the Super FX only if an acceptable accessory is connected. If an accessory ID other than those acceptable is detected, a warning message must be displayed and the Super FX must halt.

Some accessories may be used, depending upon the size of ROM and RAM included in the game pak and the Super FX operating frequency. The user should contact Nintendo's Licensee Support Group for assistance, in advance, if use of an accessory other than the standard controller is desired.

## Chapter 2 GSU FUNCTIONAL OPERATION

### 2.1 GSU FUNCTIONAL BLOCK DIAGRAM

The GSU is comprised of the following 6 functional blocks. These are demonstrated in the figure below.

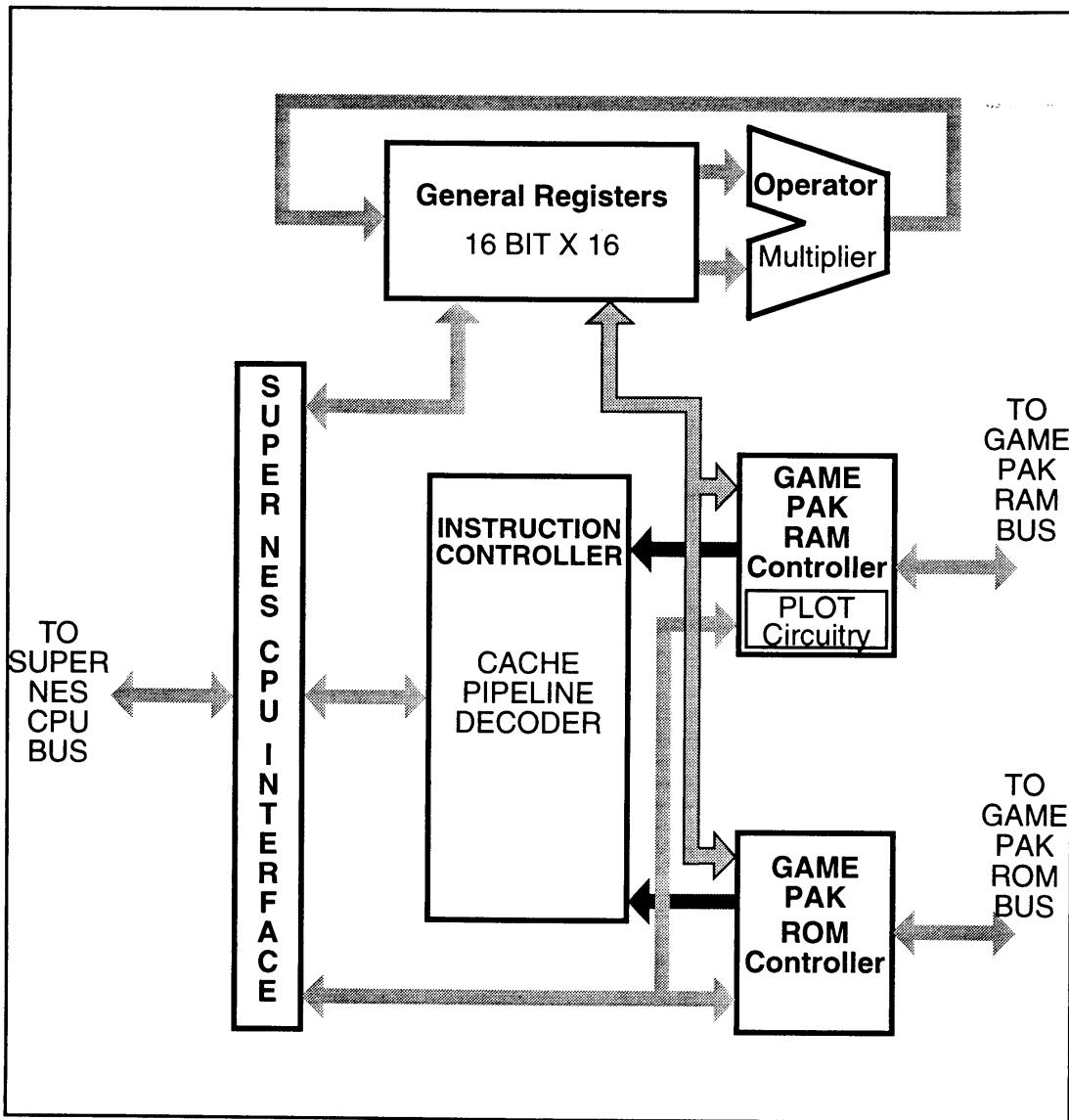


Figure 2-2-1 GSU Functional Block Diagram

### 2.1.1 SUPER NES CPU INTERFACE

The Super NES CPU Interface performs the following functions:

1. Controls data transfer between the Super NES CPU, game pak ROM/RAM, and the general registers.
2. Controls instruction data transfer between Super NES CPU and the cache.
3. Controls activation of GSU.
4. Controls interrupt to Super NES CPU.

### 2.1.2 INSTRUCTION CONTROLLER

This controls fetch instructions, decode instructions, and various other blocks based upon these instructions; loaded from game pak ROM, game pak RAM, or the cache.

Note: Pipeline and cache circuits enable high speed execution of instructions.

### 2.1.3 GAME PAK ROM CONTROLLER

The game pak ROM controller performs the following functions:

1. Controls data transfer between the Super NES CPU and game pak ROM.
2. Loads instructions from game pak ROM to the GSU.
3. Transfers data from the game pak ROM to the GSU internal registers.

Note: Data transfer from the game pak ROM to the GSU is accomplished using a ROM buffering system. This enables instructions from the game pak RAM and cache to be executed and operated in an array.

### 2.1.4 GAME PAK RAM CONTROLLER

The game pak RAM controller functions as follows:

1. Controls data transfer between the Super NES CPU and game pak RAM.
2. Loads instructions from game pak RAM to the GSU.
3. Transfers data between game pak RAM and GSU internal registers.
4. Bitmap emulation.

Note: Data transfer from the game pak RAM to the GSU is accomplished using a RAM buffering system. This enables instructions from the game pak ROM and cache to be executed and operated in an array.

### 2.1.5 GENERAL REGISTERS

These registers are used for general operations and data transfer.

Note: The GSU is equipped with sixteen, 16-bit registers. All GSU operations are performed using the general registers.

### 2.1.6 OPERATOR

The Operator executes 16-bit arithmetic operations and logical operations.

## 2.2 REGISTERS

A list of GSU internal registers is provided in the table below.

FUNCTIONAL GROUP	REGISTER NAME
General Registers Group	General Register R0 ~ R13 ROM Address Pointer R14 Program Counter R15 Status/Flag Register SFR
Registers Related to Memory Operations	Program Bank Register PBR Game Pak ROM Bank Register ROMBR Game Pak RAM Bank Register RAMBR Cache Base Register CBR
Plot Related Registers	Screen Base Register SCBR Screen Mode Register SCMR Color Register COLR Plot Option Register POR
Other Registers	Back-up RAM Register BRAMR Version Code Register VCR CONFIG Register CFGR Clock Select Register CLSR

Table 2-2-1 Registers Listed by Functional Group

### 2.2.1 GENERAL REGISTERS

#### 2.2.1.1 R0 ~ R13

These registers are used to execute various instructions as GSU General Registers during GSU operation. There are special functions available for some instructions (refer to "GSU Internal Register Configuration"). These can also be accessed by the Super NES CPU when the GSU is in the idle state.

**2.2.1.2 R14**

This register functions as a data pointer for game pak ROM during GSU operation. Data addressed in this register is automatically stored in the ROM buffer. As with R0 ~ R13, this register may be used as a GSU general register. It can also be accessed by the Super NES CPU when the GSU is in the idle state.

**2.2.1.3 R15**

This register is the GSU Program Counter. If an address is written to this register from the Super NES CPU, while the GSU is idle, the GSU will be activated.

**2.2.1.4 STATUS/FLAG REGISTER (SFR)**

The “flags” in this register indicate GSU status and operation results. This register can be referenced by the Super NES CPU even while the GSU is operating.

**2.2.2 REGISTERS RELATED TO MEMORY OPERATIONS****2.2.2.1 PROGRAM BANK REGISTER (PBR)**

This register specifies the memory bank when an instruction is read. Its value must be assigned from the Super NES CPU before the GSU is activated. This is changed during GSU operation using the LJMP instruction.

**2.2.2.2 GAME PAK ROM BANK REGISTER (ROMBR)**

This register specifies the game pak ROM bank when data are read from the game pak ROM using the ROM buffering system. Its value is changed during GSU operation using the ROMB instruction.

**2.2.2.3 GAME PAK RAM BANK REGISTER (RAMBR)**

This register specifies the game pak RAM bank when data are read/written from/to the game pak RAM. Its value is changed during GSU operation using the RAMB instruction.

**2.2.2.4 CACHE BASE REGISTER (CBR)**

This register specifies the starting address when loading data from the game pak ROM or RAM to the cache RAM. The value for CBR is updated during GSU operation whenever the CACHE instruction or LJMP instruction is executed.

**2.2.3 PLOT RELATED REGISTERS****2.2.3.1 SCREEN BASE REGISTER (SCBR)**

This register is used to specify the start address in the character data storage area. Its value must be assigned from the Super NES CPU prior to activating the GSU.

### 2.2.3.2 SCREEN MODE REGISTER (SCMR)

This register assigns the color and screen mode when PLOT processing is performed. Its value must be assigned from the Super NES CPU prior to activating the GSU.

### 2.2.3.3 COLOR REGISTER (COLR)

This register specifies the color when PLOT processing is performed. Its value is changed during GSU operation using the COLOR instruction or GETC instruction. It cannot be accessed from the Super NES CPU.

### 2.2.3.4 PLOT OPTION REGISTER (POR)

This register assigns the mode when executing the COLOR, GETC, or PLOT instructions. When these instructions are used, the value of the plot option register must be assigned before execution, using the CMODE instruction.

## 2.2.4 OTHER REGISTERS

### 2.2.4.1 B-RAM REGISTER (BRAMR)

Back-up RAM enable/disable can be controlled by this register. The register's value must be assigned from the Super NES CPU.

### 2.2.4.2 VERSION CODE REGISTER (VCR)

This assigns the GSU version code. Its value can be read only from the Super NES CPU.

### 2.2.4.3 CONFIG REGISTER (CFGR)

This register assigns the execution speed for GSU multiplication instructions and enables/disables the interrupt signal to the Super NES CPU. Its value must be assigned from the Super NES CPU prior to GSU activation.

### 2.2.4.4 CLOCK SELECT REGISTER (CLSR)

This register is used to assign the operating frequency for the Super FX. Its value must be assigned from the Super NES CPU prior to activation of the Super FX.

## 2.3 INSTRUCTION SET

There are 98 instructions available in the GSU. These instructions and their functions are given in the following table.

CLASSIFICATION	INSTRUCTION	FUNCTION
D I A N T S A T T R R U A C N T S I F O E N R S	GETB	Get byte from ROM buffer
	GETBH	Get high byte from ROM buffer
	GETBL	Get low byte from ROM buffer
	GETBS	Get signed byte from ROM buffer
	GETC	Get byte from ROM to color register
	LDW (Rm)	Load word data from RAM
	LDB (Rm)	Load byte data from RAM
	LM Rn, (xx)	Load word data from RAM using 16 bits
	LMS Rn, (yy)	Load word data from RAM, short address
	STW (Rm)	Store word data to RAM
From register to game pak RAM (RAM buffer)	STB (Rm)	Store byte data to RAM
	SM (xx), Rn	Store word data to RAM using 16 bits
	SMS (yy), Rn	Store word data to RAM, short address
	SBK	Store word data, last RAM address used
	MOVE Rn, Rn'	Move word data
From register to register	MOVES Rn, Rn'	Move word data and set flags
	IWT Rn, #xx	Load immediate word data
Immediate data to register	IBT Rn, #pp	Load immediate byte data
	ADD Rn	Add
Arithmetic Operation Instructions	ADD #n	
	ADC Rn	Add with carry
	ADC #n	
	SUB Rn	Subtract
	SUB #n	
	SBC Rn	Subtract with carry
	CMP Rn	Compare
	MULT Rn	
	MULT #n	Signed multiply
	UMULT Rn	
	UMULT #n	Unsigned multiply
	FMULT	Fractional signed multiply
	LMULT	16x16 signed multiply
	DIV2	Divide by 2
	INC Rn	Increment
	DEC Rn	Decrement

Table 2-2-2 Instruction Set (Sheet 1)

CLASSIFICATION	INSTRUCTION	FUNCTION
Logical Operation Instructions	AND Rn AND #n	Logical AND
	OR Rn OR #n	Logical OR
	NOT	Invert all bits
	XOR Rn XOR #n	Logical exclusive OR
	BIC Rn BIC #n	Bit clear mask
	ASR LSR ROL ROR	Arithmetic shift right Logical shift right Rotate left through carry Rotate right through carry
Byte Transfer Instructions	HIB	Value of high byte of register
	LOB	Value of low byte of register
	MERGE	Merge high byte of R8 and R7
	SEX	Sign extend register
	SWAP	Swap low and high byte
Jump, Branch, Loop Instructions	JMP Rn	Jump
	LJMP Rn	Long jump
	BRA e	Branch always
	BGE e	Branch on greater than or equal to zero
	BLT e	Branch on less than zero
	BNE e	Branch on not equal
	BEQ e	Branch on equal
	BPL e	Branch on plus
	BMI e	Branch on minus
	BCC e	Branch on carry clear
	BCS e	Branch on carry set
	BVC e	Branch on overflow clear
	BVS e	Branch on overflow set
	LOOP	Loop
	LINK #n	Link return address
Bank Set-up Instructions	ROMB	Set ROM data bank
	RAMB	Set RAM data bank
Plot-related Instructions	CMODE	Set plot mode
	COLOR	Set plot color
	PLOT	Plot pixel
	RPIX	Read pixel color

Table 2-2-2 Instruction Set (Sheet 2)

CLASSIFICATION	INSTRUCTION	FUNCTION
Prefix Flag Instructions	ALT1	Set ALT1 mode
	ALT2	Set ALT2 mode
	ALT3	Set ALT3 mode
Prefix Register Instructions	FROM Rn	Set Sreg
	TO Rn	Set Dreg
	WITH Rn	Set Sreg and Dreg
GSU Control Instructions	CACHE	Set cache base register
	NOP	No operation
	STOP	Stop processor
Macro Instructions	MOVEW Rn, (Rn')	Load word data from RAM
	MOVEB Rn, (Rn')	Load byte data from RAM
	MOVE Rn, (xx)	Load word data from RAM using 16 bits
	MOVEW (Rn'), Rn	Store word data to RAM
	MOVEB (Rn'), Rn	Store byte data to RAM
	MOVE (xx), Rn	Store word data to RAM using 16 bits
	MOVE Rn, #xx	Load immediate word data
	LEA Rn, xx	Load effective address

Table 2-2-2 Instruction Set (Sheet 3)

## ***Chapter 3      Memory Mapping***

### **3.1 SUPER NES CPU MEMORY MAP**

The figure on the following page depicts the memory map for the Super NES CPU. Refer to this figure while reading the sub-paragraphs below.

#### **3.1.1 GSU INTERFACE**

This area (A) is mapped to address 3000H ~ 32FFH in banks 00H ~ 3FH and 80H ~ BFH. (Refer to “GSU Internal Register Configuration”.)

#### **3.1.2 GAME PAK ROM**

Game pak ROM (B) is mapped to 2 Mbytes starting from 00:8000H. Two Mbytes from 40:0000H (B') are used for the ROM image. This image is stored in blocks of 32 Kbytes, as indicated on the memory map by circled numbers (i.e., area; ①' is the image of area ①, ②' is the image of area ②, and so forth).

#### **3.1.3 GAME PAK RAM**

Game pak RAM (C) is mapped to 128 Kbytes starting from 70:0000H. Eight Kbytes from address 6000H (C') in each of banks 00~3F and 80~BF are used for RAM image.

#### **3.1.4 BACK-UP RAM**

Back-up RAM (D) is mapped to 128 Kbytes from 78:0000H.

#### **3.1.5 SUPER NES CPU ROM**

Six Mbyte of ROM (E) is mapped from 80:8000H.

SUPER NES CPU MEMORY MAP

## **MEMORY MAPPING**

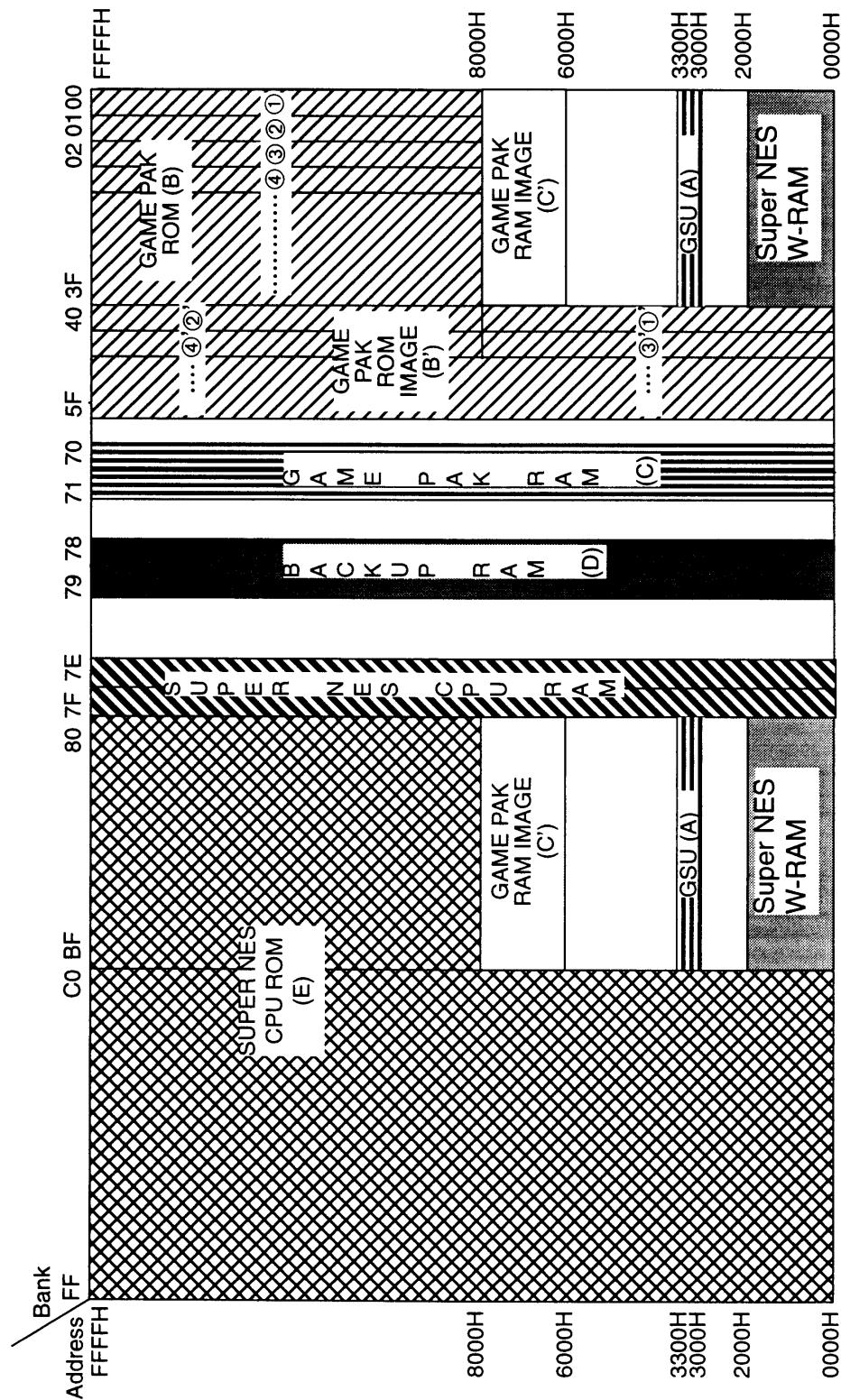


Figure 2-3-1 Super NES CPU Memory Map

## 3.2 GSU MEMORY MAPPING

The GSU memory map is depicted on the following page.

### 3.2.1 GAME PAK ROM

The game pak ROM (A) is mapped to 2 Mbytes starting from 00:8000H. Two Mbytes from 40:0000H (A') are used for the ROM image. This image is stored in blocks of 32 Kbytes, as indicated on the memory map by circled numbers (i.e., area; ①' is the image of area ①, ②' is the image of area ②, and so forth). Other areas should not be used for this purpose.

### 3.2.2 GAME PAK RAM

Game pak RAM (B) is mapped to 128 Kbytes starting from 70:0000H. When the GSU accesses memory, it specifies bank addresses using three bank registers. These are; Program Bank Register (PBR), ROM Bank Register (ROMBR), and RAM Bank Register (RAMBR).

## SUPER FX MEMORY MAP

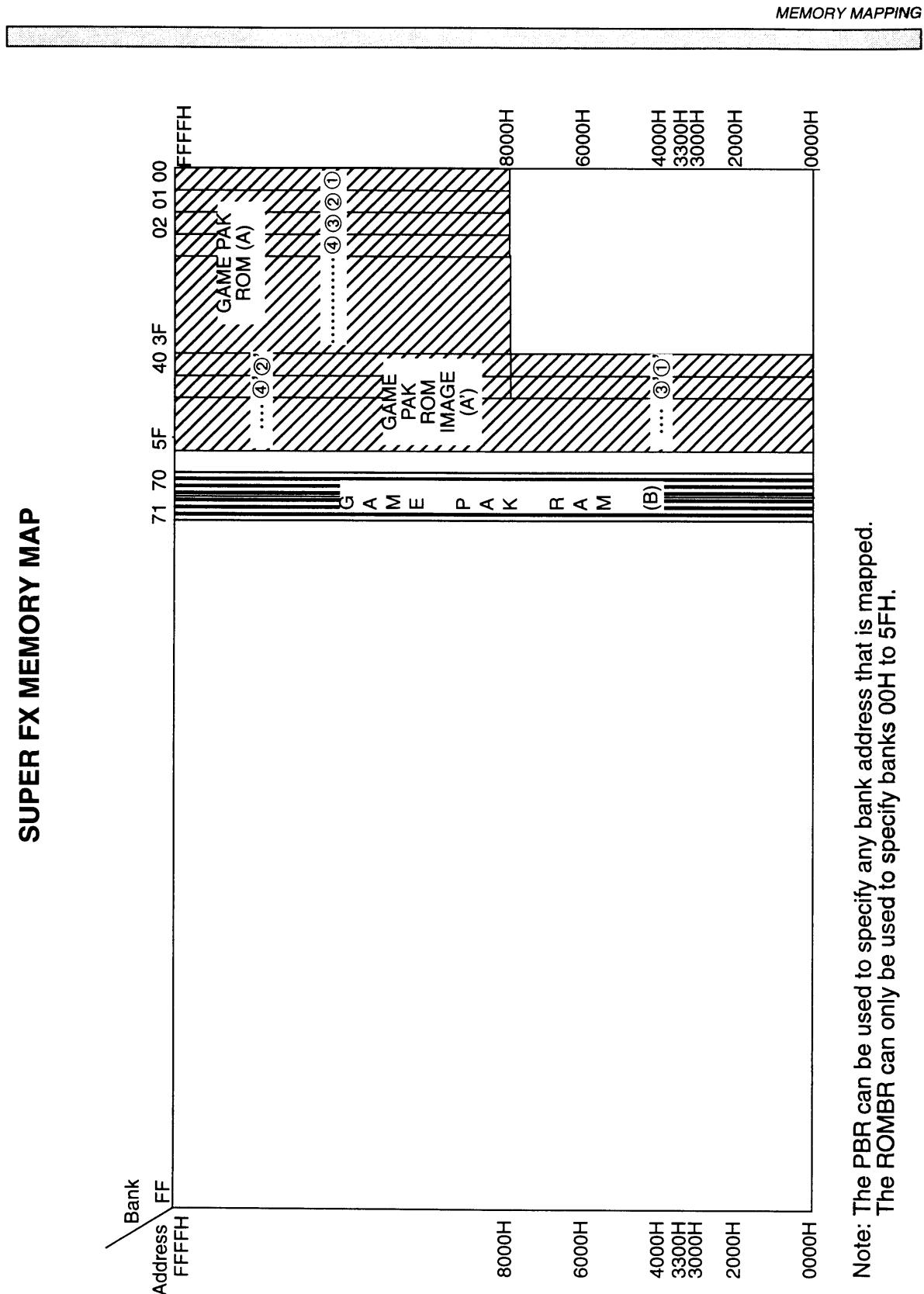


Figure 2-3-2    Super FX Memory Map

## Chapter 4 GSU Internal Register Configuration

The GSU internal registers will be described in detail in this chapter. Although many of these registers may be accessed from the Super NES CPU, none can be accessed in this way during operation of the GSU, with the exception of the Status/Flag Register (SFR) and Version Code Register (VCR). In addition, when addressing the 16-bit registers from the Super NES CPU, the low byte must be accessed first.

**All addresses denoted with (\*\*) can be accessed in banks 00H ~ 3FH and 80H ~ BFH.**

### 4.1 GENERAL REGISTERS (R0 ~ R13)

Access from Super NES CPU: R/W

Register Size: 16 bits

GSU Access Method: Various transfer instructions (LDW (Rn))  
Various Operation Instructions (ADD Rn)  
Other Instructions

Register Name	Super NES CPU Address	Special Functions	Initial Value
R0	** :3000H, 3001H	Default source/destination register	Invalid
R1	** :3002H, 3003H	PLOT instruction, X coordinate	0000H
R2	** :3004H, 3005H	PLOT instruction, Y coordinate	0000H
R3	** :3006H, 3007H		Invalid
R4	** :3008H, 3009H	LMULT instruction, lower 16 bits	Invalid
R5	** :300AH, 300BH		Invalid
R6	** :300CH, 300DH	FMULT and LMULT instructions, multiplication	Invalid
R7	** :300EH, 300FH	MERGE instruction, source 1	Invalid
R8	** :3010H, 3011H	MERGE instruction, source 2	Invalid
R9	** :3012H, 3013H		Invalid
R10	** :3014H, 3015H		Invalid
R11	** :3016H, 3017H	LINK instruction destination register	Invalid
R12	** :3018H, 3019H	LOOP instruction counter	Invalid
R13	** :301AH, 301BH	LOOP instruction branch	Invalid

Table 2-4-1 GSU General Registers

For LINK and LOOP special functions refer to “Instruction Execution”, for other special functions refer to the instruction name in the chapter titled “Description of Instructions”.

### R0

D15	D14	D13	D12	D11	D10	D9	D8	3001H
D7	D6	D5	D4	D3	D2	D1	D0	3000H

Figure 2-4-1 Example of General Register

## 4.2 GAME PAK ROM ADDRESS POINTER (R14)

Access from Super NES CPU: R/W

Super NES CPU Addresses: \*\* :301CH, 3011DH

Register Size: 16 bits

GSU Access Method: Various transfer instructions (LDW (Rn))  
Various operation instructions (ADD Rn)  
Other instructions

D7	D6	D5	D4	D3	D2	D1	D0	
A15	A14	A13	GAME PAK ROM		A10	A9	A8	301DH
A7	A6	A5	GAME PAK ROM		A2	A1	A0	301CH

R14 is a pointer that specifies the game pak ROM address when data are loaded from the game pak ROM to an internal register. Typically, the ROM buffering system will be used for this process.

### 4.3 PROGRAM COUNTER (R15)

Access from Super NES CPU: R/W  
 Super NES CPU Addresses: \*\* :301EH, 3011FH  
 Register Size: 16 bits  
 Default Address: 0000H  
 GSU Access Method: Various branching instructions (JMP Rn)  
 Other instruction

D7	D6	D5	D4	D3	D2	D1	D0	
PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	301FH
PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	301EH

R15 is the GSU program counter. If its value is changed by a transfer instruction or operation instruction, the program jumps to the address of the new value.

## 4.4 STATUS/FLAG REGISTER (SFR)

Access from Super NES CPU: R/W  
 Super NES CPU Addresses: \*\* :3030H, 3031FH  
 Register Size: 16 bits  
 Default Address: 0000H

D7	D6	D5	D4	D3	D2	D1	D0	
IRQ	*	*	B	IH	IL	ALT2	ALT1	3031H Status Portion
*	R	G	OV	S	CY	Z	*	3030H Flag Portion

\* This bit is 0 when this register is read.

Flag	Description
Z	Zero flag
CY	Carry flag
S	Sign flag
OV	Overflow flag
G	Go flag (set to 1 when the GSU is running)
R	Set to 1 when reading ROM using R <sub>14</sub> address.
ALT1	Mode set-up flag for the next instruction
ALT2	Mode set-up flag for the next instruction
IL	Immediate lower 8-bit flag
IH	Immediate higher 8-bit flag
B	Set to 1 when the WITH instruction is executed.
IRQ	Interrupt flag

Table 2-4-2 GSU Status Register Flags

The Status/Flag register indicates the status of the GSU. It may be accessed from the Super NES CPU during GSU operation to determine GSU status.

## 4.5 PROGRAM BANK REGISTER (PBR)

Access from Super NES CPU: R/W  
 Super NES CPU Addresses: \*\* :3034H  
 Register Size: 8 bits  
 Default Address: Undefined  
 GSU Access Method: LJMP instruction

D7	D6	D5	D4	D3	D2	D1	D0	
A23	A22	A21	A20	Program Bank A19	A18	A17	A16	3034H

The program bank register specifies the memory bank register to be accessed when the GSU is loading the program code.

## 4.6 GAME PAK ROM BANK REGISTER (ROMBR)

Access from Super NES CPU: R  
 Super NES CPU Addresses: \*\* :3036H  
 Register Size: 8 bits  
 Default Address: Undefined  
 GSU Access Method: ROMB instruction

D7	D6	D5	D4	D3	D2	D1	D0	
A23	A22	A21	A20	ROM Data Bank A19	A18	A17	A16	3036H

The game pak ROM bank register specifies the game pak ROM bank when loading data from game pak ROM using the ROM buffering system.

## 4.7 GAME PAK RAM BANK REGISTER (RAMBR)

Access from Super NES CPU: R  
 Super NES CPU Addresses: \*\* :303CH  
 Register Size: 1 bit  
 Default Address: Undefined  
 GSU Access Method: RAMB instruction

D7	D6	D5	D4	D3	D2	D1	D0
RAM Data Bank							A16
*	*	*	*	*	*	*	303CH

Bank = 70H when D0 = 0

Bank = 71H when D0 = 1

\* This bit is 0 when this register is read.

The game pak RAM bank register specifies the game pak RAM bank when data are read/written between game pak RAM and the GSU internal registers. The RAMB instruction specifies bank 70H or 71H for game pak RAM access.

## 4.8 CACHE BASE REGISTER (CBR)

Access from Super NES CPU: R  
 Super NES CPU Addresses: \*\* :303EH, 303FH  
 Register Size: 12 bits  
 Default Address: 0000H  
 GSU Access Method: LJMP, CACHE instructions

D7	D6	D5	D4	D3	D2	D1	D0
Cache Base Address							303FH
A15	A14	A13	A12	A11	A10	A9	A8
Cache Base Address							303EH
A7	A6	A5	A4	*	*	*	*

\* This bit is 0 when this register is read.

The cache base register specifies the starting address when data are loaded from game pak ROM or RAM to the cache RAM.

## 4.9 SCREEN BASE REGISTER (SCBR)

Access from Super NES CPU: W  
Super NES CPU Addresses: \*\* :3038H  
Register Size: 8 bits  
Default Address: Undefined  
GSU Access Method: None

D7	D6	D5	D4	D3	D2	D1	D0	
A17	A16	A15	A14	A13	A12	A11	A10	3038H

The screen base register is used to specify the start address in the character data storage area.

## 4.10 SCREEN MODE REGISTER (SCMR)

Access from Super NES CPU: W  
 Super NES CPU Addresses: \*\* :303AH  
 Register Size: 6 bits  
 Default Address: 00H  
 GSU Access Method: None

D7	D6	D5	D4	D3	D2	D1	D0	
Screen Height Select							Color Gradient	
-	-	HT1	RON	RAN	HT0	MD1	MD0	303AH

The screen mode register specifies the color gradient and screen height during PLOT processing and controls game pak ROM and RAM bus assignments.

### 4.10.1 SCREEN HEIGHT

Ht 1	Ht 0	Mode
0	0	128 (pixels)
0	1	160 (pixels)
1	0	192 (pixels)
1	1	OBJ mode

Table 2-4-3 Screen Height

### 4.10.2 COLOR GRADIENT

Mod 1	Mod 0	Mode
0	0	4-color mode
0	1	16-color mode
1	0	Not used
1	1	256-color mode

Table 2-4-4 Color Gradient

### 4.10.3 ROM/RAM ENABLE FLAGS

When:

RON = 0, the Super NES CPU has game pak ROM bus access.  
 1, the GSU has game pak ROM bus access.

RAN = 0, the Super NES CPU has game pak RAM bus access.  
 1, the GSU has game pak RAM bus access.

## 4.11 COLOR REGISTER (COLR)

Access from Super NES CPU: Disabled  
 Super NES CPU Addresses:  
 Register Size: 8 bits  
 Default Address: Undefined  
 GSU Access Method: COLOR, GETC instructions

D7	D6	D5	D4	D3	D2	D1	D0
CD7	CD6	CD5	Color Data CD4	CD3	CD2	CD1	CD0

The color register contains data which specifies the colors to be plotted when PLOT processing is performed.

## 4.12 PLOT OPTION REGISTER (POR)

Access from Super NES CPU: Disabled  
 Super NES CPU Addresses:  
 Register Size: 5 bits  
 Default Address: Undefined  
 GSU Access Method: CMODE instruction

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	OBJ Flag	Freeze High Flag	High Nibble Flag	Dither Flag	Trans-parent Flag

The plot option register contains flags which specify the mode to be used when a COLOR, GETC, or PLOT instruction is executed.

### 4.13 BACK-UP RAM REGISTER (BRAMR)

Access from Super NES CPU: W  
 Super NES CPU Addresses: \*\* :3033H  
 Register Size: 1 bit  
 Default Address: 00H  
 GSU Access Method: None

D7	D6	D5	D4	D3	D2	D1	D0	
BRAM Flag								3033H
-	-	-	-	-	-	-	-	

When:

BRAM Flag = 0, BRAM is disabled.  
1, BRAM is enabled.

Data becomes "protected" when the BRAM flag is reset ("0") after saving data to the Back-up RAM.

### 4.14 VERSION CODE REGISTER (VCR)

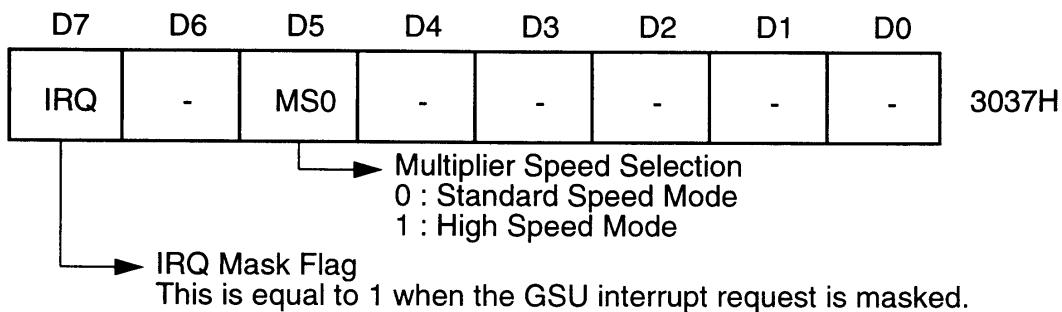
Access from Super NES CPU: R  
 Super NES CPU Addresses: \*\* :303BH  
 Register Size: 8 bit  
 Default Address: Undefined  
 GSU Access Method: None

D7	D6	D5	D4	D3	D2	D1	D0	
Version Code								303BH
VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	

The version code register permits the user to read the GSU version code.

## 4.15 CONFIG REGISTER (CFGR)

Access from Super NES CPU: W  
 Super NES CPU Addresses: \*\* :3037H  
 Register Size: 8 bit  
 Default Address: 00H  
 GSU Access Method: None

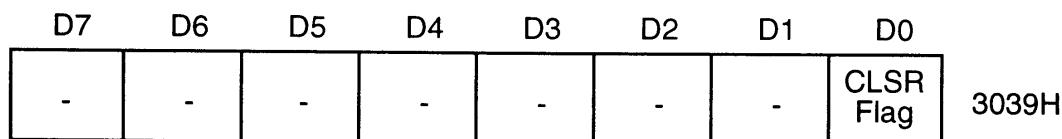


The CONFIG register selects the operating speed of the multiplier in the GSU and sets up a mask for the interrupt signal.

**Note:** When the Super FX operates at 21 MHz (when the CLSR flag of the Clock Select Register is "1"), MS0 flag should be fixed at "0".

## 4.16 CLOCK SELECT REGISTER (CLSR)

Access from Super NES CPU: W  
 Super NES CPU Addresses: \*\*:3039H  
 Register Size: 1 bit  
 Default Address: 00H  
 GSU Access Method: None



When:

CLSR Flag = 0, Super FX operates at 10.7 MHz  
 = 1, Super FX operates at 21.4 MHz

This register assigns the Super FX operating frequency.

## ***Chapter 5 GSU Program Execution***

## 5.1 STARTING THE GSU

The GSU is placed in the idle state when the Super NES control deck is reset. The GSU is started by writing to its internal program counter (R15) from the Super NES. The GSU programs operate on the game pak ROM, RAM, or cache RAM, but the GSU activation method differs depending upon which memory is accessed. The various methods are described below.

### **5.1.1 STARTING GSU PROGRAM IN GAME PAK ROM**

The GSU is started by the following method when the GSU program is to operate in the game pak ROM.

### **5.1.1.1 BUS CONTROL**

In order for the Super NES CPU to pass game pak ROM bus access to the GSU, the Super NES CPU program used to start the GSU in an area other than the game pak ROM (such as WRAM) is transferred to the GSU and the GSU jumps to that program.

However, if the optional ROM for the Super NES is being used, the GSU can be started by running the start program in Super NES ROM, making the above transfer unnecessary.

### 5.1.1.2 REGISTER ADDRESSING

In the Super NES CPU program for starting the GSU, first assign the following registers.

- PBR (Super NES CPU Address, \*\*:3034H)
  - SCBR (Super NES CPU Address, \*\*:3038H)
  - SCMR (Super NES CPU Address, \*\*:303AH)

Note: RON absolutely must be set to "1".

- CFGR (Super NES CPU Address, \*\*:3037H)
  - CLSR (Super NES CPU Address, \*\*:3039H)

Subsequently, when the lead address of the GSU program is written from the Super NES CPU to R15 (Super NES CPU address, \*\*:301EH), the GSU can be started from that address.

An example of the program required for starting the GSU from the Super NES is demonstrated on the following page.

```
mem8
lda    #clock data
sta    3039H      ;Sets operating frequency
sta    3037H      ;Sets CONFIG register
lda    #screen base
sta    3038H      ;Sets screen base
lda    #program bank
sta    3034H      ; Sets program code bank
lda    #screen size mode
ora    18H         ; Sets RON, RAN flag, screen size, and color number
sta    303aH

mem16
rep    #00100000B
lda    #program address
sta    301EH       ; Sets program counter
```

### **5.1.2 STARTING GSU PROGRAM IN GAME PAK RAM**

The following procedure is used to start the GSU when its program is to operate in game pak RAM.

### **5.1.2.1 TRANSFER GSU PROGRAM**

The Super NES CPU first transfers the GSU program from the game pak ROM to game pak RAM. If the GSU will not be using game pak ROM, the Super NES CPU does not need to pass the game pak ROM bus access to the GSU.

### 5.1.2.2 REGISTER ADDRESSING

In the Super NES CPU program for starting the GSU, first assign the following registers.

- PBR (Super NES CPU Address, \*\*:3034H)
  - SCBR (Super NES CPU Address, \*\*:3038H)
  - SCMR (Super NES CPU Address, \*\*:303AH)

Note: RAN absolutely must be set to "1".

- CFGR (Super NES CPU Address, \*\*:3037H)
  - CLSR (Super NES CPU Address, \*\*:3039H)

Subsequently, when the lead address of the GSU program is written from the Super NES CPU to R15 (Super NES CPU address, \*\*:301EH), the GSU can be started from that address.

### 5.1.3 STARTING GSU PROGRAM IN CACHE RAM

The following procedure is used to start the GSU when its program is to operate in cache RAM.

#### 5.1.3.1 TRANSFER GSU PROGRAM

The Super NES CPU first transfers the GSU program from the game pak ROM to cache RAM. If the GSU will not be using game pak ROM or RAM, the Super NES CPU does not need to pass the game pak ROM or RAM bus access to the GSU.

#### 5.1.3.2 REGISTER ADDRESSING

In the Super NES CPU program for starting the GSU, first assign the following registers.

- PBR (Super NES CPU Address, \*\*:3034H)
- SCBR (Super NES CPU Address, \*\*:3038H)
- SCMR (Super NES CPU Address, \*\*:303AH)
- CFGR (Super NES CPU Address, \*\*:3037H)
- CLSR (Super NES CPU Address, \*\*:3039H)

Subsequently, when the lead address of the GSU program is written from the Super NES CPU to R15 (Super NES CPU address, \*\*:301EH), the GSU can be started from that address.

## 5.2 STOPPING THE GSU

The following two methods may be used to stop the GSU.

- GSU auto-stop using the STOP instruction
- Forced stop from the Super NES CPU using the GO flag

#### 5.2.1 GSU AUTO-STOP USING STOP INSTRUCTION

The STOP instruction is one of the instructions in the GSU instruction set. When the GSU reads the STOP instruction, it resets the GO flag, sends an interrupt (IRQ) to the Super NES CPU (to inform the CPU that processing is complete), and goes into the idle state.

The value in R15 after the GSU has executed a STOP instruction varies depending upon the instruction that was executed immediately prior to the STOP instruction.

Instruction Type	Value of R15
Transfer Data to R15	R15 Data + 1
Jump or Branch	Jump or branch destination address + 1
CACHE Instruction	Address of STOP instruction + 1
Other Instruction	Address of STOP instruction + 1

### 5.2.2 FORCED STOP FROM SUPER NES CPU USING GO FLAG

The GSU can be forceably stopped by writing a "0" from the Super NES CPU to the GO flag in the status/flag register (Super NES CPU address, \*\* :3030H). This clears the data in the cache and resets the cache base register to 0000H.

### 5.3 MEMORY ACCESS FROM SUPER NES CPU DURING GSU OPERATION

If a "0" is written from the Super NES CPU to the RON flag in the status/flag register (Super NES CPU address, \*\* :303AH) during GSU operation, the GSU will shift to WAIT status when it requires game pak ROM access. This makes it temporarily possible to access game pak ROM from the Super NES CPU.

The WAIT status is subsequently canceled by writing a "1" to RON from the Super NES CPU. This causes the GSU to resume processing. In a similar manner, game pak RAM can be temporarily accessed by the Super NES CPU, using the RAN flag in the screen mode register.

### 5.4 INTERRUPTS

#### 5.4.1 SUPER NES CPU INTERRUPT VECTOR

Game pak ROM access from the Super NES CPU is inhibited during GSU operation and when the RON flag is "1". If an interrupt (NMI) is generated to the Super NES CPU under these conditions, an interrupt vector from the game pak ROM will not be available for the Super NES CPU. This will cause an error. In order to avoid this problem, when a Super NES CPU interrupt vector is read, the GSU outputs a dummy vector on the data bus. The table below expresses the relationship between the Super NES CPU interrupt vector addresses and the dummy vectors. By placing interrupt routines in all the memories except the game pak ROM and encoding a jump instruction to each of the interrupt routines at WRAM addresses 00:0104H, 00:0100H, 00:0108H, and 00:010CH, interrupt processing can be executed without accessing the game pak ROM.

Interrupt Vector Address	Dummy Vector
00:FFE4	00:0104
00:FFE6	00:0100
00:FFE8	00:0100
00:FFEA	00:0108
00:FFEE	00:010C

Table 2-5-1      Dummy Interrupt Vector Addresses

Note: If the game pak ROM is accessed from the Super NES CPU during GSU operation when GO and RON are "1", the dummy data can be read using the value of the lower 4 bits of that address. This will generate the dummy addresses described above. The table below demonstrates this.

Lower 4 Bits of Address	Dummy Data
0H, 2H, 6H, 8H, CH	00H
4H	04H
AH	08H
EH	0CH
Other	01H

Table 2-5-2      Dummy Data

#### 5.4.2 INTERRUPT FROM GSU TO SUPER NES CPU

The STOP instruction generates an IRQ from the GSU to the Super NES CPU. Therefore, the Super NES CPU can continue its own processing without having to periodically monitor the GSU for the end of its routine. Since there are instances in which an IRQ is generated for some other reason, the Super NES CPU must determine if the GSU was the source of the IRQ. There is an IRQ flag at bit 15 of the GSU status register. If this flag is "1", the IRQ was generated by the completion of GSU processing. When bit 15 of this status register is read, the bit is reset to "0". The IRQ output by the GSU can be disabled by setting bit 7 in the CONFIG register to "1".

## ***Chapter 6      Instruction Execution***

### **6.1    READING INSTRUCTION CODE**

#### **6.1.1    EXECUTION IN GAME PAK ROM/RAM**

The GSU executes a program by reading the instruction codes from the game pak ROM or RAM at the addresses specified by the PBR and program counter (R15). The contents of the PBR determines whether the instruction code is to be read from game pak ROM or RAM (refer to "Memory Mapping").

The RON flag must be set (1) when an instruction code is read from game pak ROM. If the RON flag is reset (0), the GSU will be placed in the WAIT state when a game pak ROM instruction code is loaded. Likewise, the RAN flag must be set (1) when an instruction code is read from game pak RAM. If the RAN flag is reset (0), the GSU will be placed in the WAIT state when a game pak RAM instruction code is loaded.

#### **6.1.2    EXECUTION IN CACHE RAM**

If the GSU's program counter (R15) is in a cache area determined by the cache base register and the data in the cache are valid, the GSU will read the instruction code from the cache RAM and execute it. When a program is being executed in the cache, even if RON or RAN is reset (0), the GSU will not stop when an instruction code is loaded. Consequently, it becomes possible to access the game pak ROM or RAM from the Super NES CPU.

### **6.2    PIPELINE PROCESSING**

The GSU employs a "pipeline" for high-speed operation. This "pipeline" is a mechanism that, in parallel with the execution of an instruction, loads the next step and prepares it as the next instruction. The program counter (R15) indicates the next address following the instruction currently being executed.

Normally, it is not particularly necessary to be aware of this processing, but it must be considered when using instructions that change the program counter (R15), such as branch or jump instructions. When a branching process is executed, the instruction code at the next address is loaded into the pipeline. This instruction code is then executed in parallel with a load of the instruction code at the branch destination address into the pipeline. This is demonstrated in example 1 on the following page.

## (Example 1)

```
BNE FROG  
INC R1  
:  
:  
FROG: ADD R2
```

When the program in Example 1 is executed, the INC instruction will be executed regardless of the presence of a branch instruction, since it is loaded into the pipeline while processing the BNE instruction.

Note: Be especially careful when placing an instruction of 2 bytes or more after an instruction that changes the program counter.

## (Example 2)

```
BNE LOP1  
BRA LOP2  
:  
:  
LOP1: TO R1
```

When the program in Example 2 is executed, the program jumps to LOP1 when the Z flag is 0, but the first byte of the code “BRA LOP2” has already been loaded into the pipeline. Therefore, the code 11H at the jump destination “TO R1” will be processed as the offset value of the BRA instruction, causing “BRA \*\*\*\*” to be executed instead of “TO R1”.

Note: The value for \*\*\*\* = LOP1+1+11H.

In this situation, a NOP instruction should be inserted after the BNE instruction, as shown below.

## (Example 3)

```
BNE LOP1  
NOP  
BRA LOP2  
:  
:  
LOP1: TO R1
```

## 6.3 PROGRAM COUNTER

The GSU program counter is assigned to R15. When the value for R15 is changed by an instruction, the program jumps to the address indicated by that value.

(Example 4)

IWT	R0,#0010H
IWT	R4,#0020H
IWT	R15,#Address
NOP	
:	
:	
Address:	ADD R4
	INC R3

In example 4, the program jumps to the specified address at the IWT instruction on the third line. Due to pipeline processing, the ADD instruction in the 7th line will be executed after the NOP instruction in the 4th line is executed. In addition, the address following the instruction currently being executed can be identified by moving the contents of R15 to another register.

## 6.4 FLAG PREFIXES

In the GSU, the action of the next instruction code to be executed varies depending upon the values of the status flags (ALT1, ALT2, B), set by instructions such as the ALT1 instruction.

(Example 5)

The instruction code 53H will perform the processing shown below depending upon the values for ALT1 and ALT2.

When ALT1=0, ALT2=0	Sreg+R3→Dreg	(ADD R3)
When ALT1=1, ALT2=0	Sreg+R3+CY→Dreg	(ADC R3)
When ALT1=0, ALT2=1	Sreg+3→Dreg	(ADD #3)
When ALT1=1, ALT2=1	Sreg+3+CY→Dreg	(ADC #3)

## (Example 6)

The instruction code 11H will perform the processing shown below depending on the value of the B flag.

When B=0      Set Dreg to R1      (TO R1)

When B=1      Sreg→R1      (MOVE R1,Rn n=value for Sreg)

The ALT1 instruction is used to set the ALT1 flag to 1. Likewise, the ALT 2 instruction is used to set the ALT2 flag to 1. The ALT3 instruction sets both the ALT1 flag and ALT2 flag. The WITH instruction is used to set the B flag.

Normally, the flags which were set by these instructions are cleared after the next instruction is executed. The flags are not cleared when the next instruction is a FROM, TO, WITH, ALT1, ALT2, ALT3, or a branch instruction.

For instance, since the TO and FROM instructions become MOVE and MOVES instructions, respectively; when the B flag is set, these flags will be cleared after the instructions are executed. They will also be cleared after the execution of a NOP instruction.

Since ALT1, ALT2, and ALT3 instructions are used in combination with the next instruction, they do not need to be thought of as independent instructions. For instance, there is no need to be specifically aware that "if ADD R3 is executed after setting the ALT1 flag with an ALT1 instruction, the instruction becomes ADC R3". The process can simply be seen as the two-byte instruction "ADC R3". In the assembler, as well, it is normally unnecessary to specifically code an ALT1 instruction or to write a MOVE instruction as a WITH instruction and a TO instruction.

However, as demonstrated in the following examples, these things need to be kept in mind when accelerating program processing by effectively using the pipeline.

## (Example 7)

	IWT	R3,#100H	
LOP1:	ADC	R0	; ALT1+ADD R0
	PLOT		
	:		
	DEC	R3	
	BNE	LOP1	
	NOP		

Due to pipeline processing, the code following a branching instruction will be executed regardless of the presence of a branch. In Example 7, the NOP instruction after the BNE instruction will always be executed, but this program can be substituted as demonstrated below.

(Example 8)

```
IWT      R3,#100H
ALT1
NEWLOP1: ADD      R0
          PLOT
          :
DEC      R3
BNE      NEWLOP1
ALT1
```

In this example, the branch destination “ADC R0” is divided into “ALT1” and “ADD R0”. ALT1 is placed after BNE, changing the address of the branch destination. Thus, the pipeline code at the time of the branch becomes useful.

A different situation is demonstrated below.

(Example 9)

```
IWT      R3,#100H
LOP2:   PLOT
:
MOVE    R4,R5      ; WITH R5+TO R4
DEC     R3
BNE     LOP2
NOP
```

This program can be substituted as shown in Example 10.

(Example 10)

```
IWT      R3,#100H
LOP2:   PLOT
:
DEC     R3
WITH   R5
BNE     LOP2
TO     R4
```

In example 10, “MOVE R4,R5” is split into “WITH R5” and “TO R4”. This kind of rewrite is possible because the B flag is not changed by the branch instruction.

## 6.5 REGISTER PREFIXES

Most of the GSU instructions use a source register (Sreg) and destination register (Dreg). The Sreg indicates the general register used for the source of the instruction, while the Dreg indicates the general register used to store the result. The Sreg and Dreg can be assigned in the GSU using the TO, FROM and WITH register prefix instructions. The Sreg is assigned using the FROM instruction and the Dreg using the TO instruction. The Sreg and Dreg can both be assigned using the WITH instruction. The Sreg and Dreg return to the default R0 when any instruction other than TO, FROM, WITH, ALT, or a branch is executed.

If a TO instruction or FROM instruction follows a WITH instruction, as demonstrated below, they will be executed as MOVE or MOVES instructions, causing Sreg and Dreg to return to the defaults after the instructions are executed. These registers also return to the defaults after a NOP instruction is executed.

(Example 11)

The program used to execute R3=R4-R5 is as follows.

TO	R3
FROM	R4
SUB	R5

The operation R0=R4-R5 can be performed by executing the following program, omitting the TO instruction.

FROM	R4
SUB	R5

The operation R0=R0-R5 can be performed using the following program. The FROM instruction is omitted.

SUB	R5
-----	----

After a normal instruction has been executed, with the exception of TO, FROM, WITH, ALT, or a branch, Sreg and Dreg are both assigned the default register (R0). Consequently, in the following program, the initial SUB instruction will execute R3=R4-R5, but the second SUB instruction will execute R0=R0-R5.

TO	R3
FROM	R4
SUB	R5
SUB	R5

The WITH instruction not only assigns Sreg and Dreg, but also sets the B flag within the status/flag register. The TO and FROM instructions act as different instructions when the B flag is set.

- When a TO instruction is next, it performs a MOVE instruction (instruction to move between registers).
- When a FROM instruction is next, it performs a MOVES instruction (instruction to move between registers and set flags according to the data loaded).

## 6.6 LOOP

The LOOP instruction is provided for efficient loop processing in the GSU. The LOOP instruction decrements the value in R12 by 1 and, when the result is not 0, loads the address in R13 into the program counter. When the result is 0, the next instruction is executed without branching.

Consequently, when performing loop processing using the LOOP instruction, it is necessary to store the loop count number in R12 and the loop return destination address in R13.

(Example 12)

```

IWT      R14,#DATA ;R14=ROM Address for Read Data
IWT      R12,#0100H ;R12=Loop Count Number
MOVE    R13,R15   ;R13=REPEAT (Loop Back Address)

REPEAT:
    GETB
    INC     R14
    LOOP           ;R12=R12-1. IF (R12<>0) THEN PC=R13
    PLOT

```

## 6.7 SUBROUTINES

The GSU does not have any instructions for making subroutine calls. Therefore, when using a subroutine, it will be necessary to specify the return destination address in the program.

(Example 13)

```

A000  FB 07 A0          IWT    R11,#RETURN
A003  FF 03 A1          IWT    R15,#SUB1 ;Jump to SUB1
A006  01                NOP    ;Dummy
A007  D0    RETURN:    INC    R0      ;Return Address
    :
    :
    :
A103  96    SUB1:     ASR
A104  96            ASR
A105  2B 1F          MOVE   R15,R11 ;Return to Main Routine
A107  01            NOP    ;Dummy

```

In Example 13, the program jumps to the subroutine after the return address in R11 has been specified. In the subroutine, the program finally returns to the main program by loading the value for R11 to the program counter (R15).

The LINK instruction is used in the GSU for specifying the return address. LINK adds a value from 1 to 4, depending upon the operand, to the address of the instruction following LINK. The result is stored in R11.

#### (Example 14)

The call side of the routine in Example 13 can be rewritten as follows using the LINK instruction.

A000	94	LINK	#4	;R11=A005
A001	FF 03 A1	IWT	R15,#SUB1	;Jump to SUB1
A004	01	NOP		
A005	D0	RETURN:	INC R0	;Return Address

## 6.8 CACHE RAM

A 512-byte instruction cache is built into the Super FX. Because instruction code is read six times as fast as reading from game pak ROM or RAM, a program in cache RAM runs at high speed. If a program is run in cache memory, access to the game pak ROM or RAM can be performed at the same time the instruction is executed. Therefore, a program can be executed at a higher speed.

### 6.8.1 USING CACHE INSTRUCTIONS

The CACHE instruction is used to control the cache. If the CACHE instruction is executed, any subsequent instruction codes will be sequentially loaded into the cache RAM whether they are loaded from game pak ROM or game pak RAM.

For instance, if the CACHE instruction is executed immediately prior to loop processing, the program can be made to operate in the cache RAM beginning with the second repetition.

Program loops exceeding 512 bytes in size will not perform efficiently since the portion not handled in cache RAM will always be executed in game pak ROM or game pak RAM. Dividing the program into several loops so that the loops fit within the 512 byte limit will enable higher speed operation when the CACHE instruction is executed immediately prior to these loops.

### 6.8.2 CACHE OPERATION

When the CACHE instruction is executed, the beginning address for data to be loaded from game pak ROM or RAM to cache RAM is stored in the CBR (cache base register). The cache area will be 512 bytes beginning with the address stored in the CBR. The 512-byte cache area is further divided into 32 blocks of 16 bytes each. A "cache flag" is assigned to each of these 32 blocks.

When the program counter indicates the cache area, the cache flag that corresponds with that address is read. If the cache flag is not set, the instructions are loaded to cache RAM while the program executes in game pak ROM or RAM. The cache flag is set when the 16-byte block has been entirely loaded with instruction code. If the cache flag has already been set, the program is executed in cache RAM. The cache flags are all reset when the CACHE instruction is executed.

Since the low 4 bits of the CBR are fixed at 0, the beginning address stored in the CBR after execution of a CACHE instruction will be the value of the address following the CACHE instruction with its low 4 bits set to 0 (XXX0H). If the low 4 bits of the address following the CACHE instruction are other than 0, the program jumps to the address in the CBR and loads the code from the game pak ROM or RAM into the cache RAM, after the CACHE instruction is executed.

If a branch occurs before all 16 bytes of instruction code in a block can be loaded (before the cache flag is set), the program will branch after the remaining instruction code in that block has been entirely loaded. This operation is the same within the same block. If the program has branched to an address other than the block header address (XXX0H), the code between the block header address and the branch address will be loaded before the instruction at the branch address is executed. Refer to the illustration on the following page.

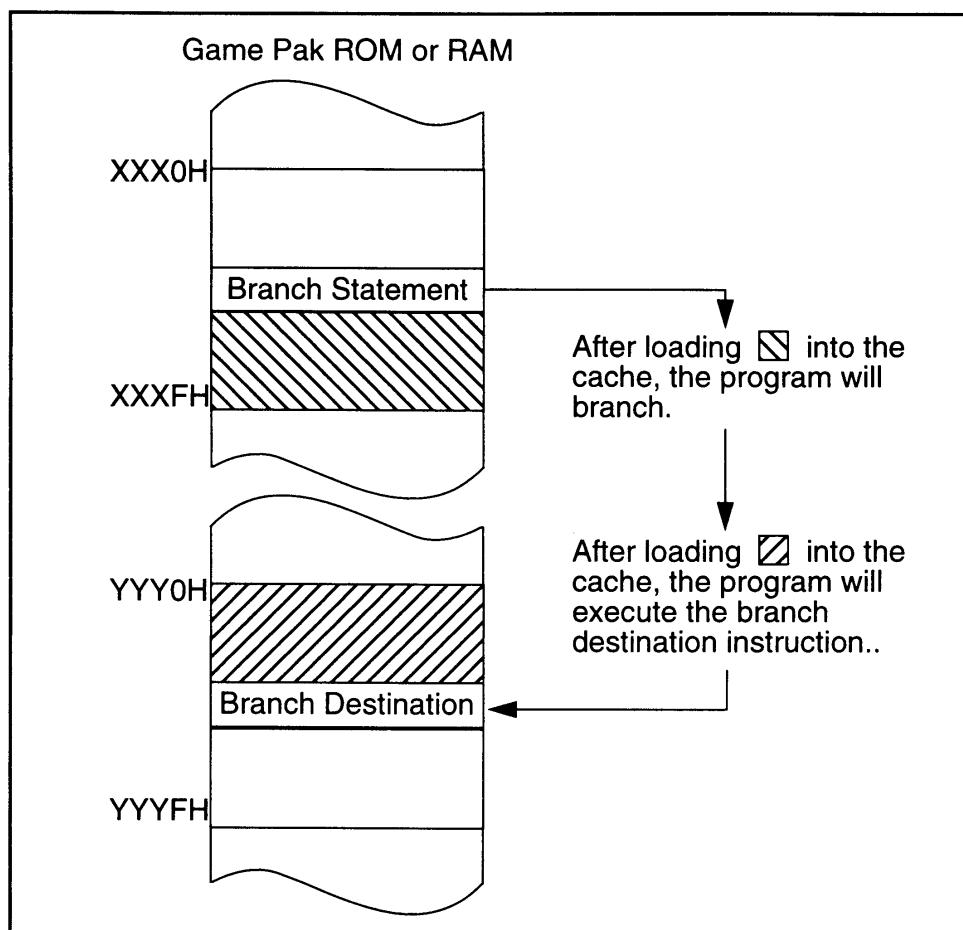


Figure 2-6-1 Load to Cache RAM While Branching

Since the CBR does not have any bank information, when an LJMP instruction is executed, all cache flags are cleared and the CBR is reset to a value with the low 4 bits of the jump destination address at 0 (XXX0H). This operation is the equivalent of executing another CACHE instruction.

In addition, when the Super NES CPU writes a 0 to the GO flag of the GSU's status/flag register (a forced end if the GSU is operating), all of the cache flags are cleared and the CBR value is set to 0000H. If the GSU is stopped by a STOP instruction, the contents of the CBR, cache flags and cache RAM are all saved. Consequently, when the GSU is restarted, a 0 must be written to the GO flag to reset the CBR and cache flags.

### 6.8.3 CACHE RAM ACCESS FROM THE SUPER NES

It is possible for the Super NES CPU to read and write to the GSU's cache RAM. The cache RAM is divided into 512-byte addresses from 3100H in any of banks 00H~3FH or 80H~BFH in the Super NES memory map. When the GSU is not operating, data can be freely read and written from/to the Super NES CPU.

However, the CBR does not necessarily comply with address 3100H in the Super NES memory map. Caution should be observed when reading cache memory contents after the CACHE instruction has been executed. The address in the CBR cache RAM complies with the address indicated by the value of the low 9 bits of the CBR. Therefore, the CBR address on the Super NES is calculated as follows.

$$\text{CBR address on Super NES} = 3100H + (\text{CBR AND } 01FFH)$$

When cache data is loaded from the CBR complied address to 32FFH, continuous data is loaded from 3100H to the CBR complied address minus 1.

For example; when the CBR is C3A0H,

Instruction Memory Address	Super NES Complied Address
C3A0H~C3FFH	32A0H~32FFH
C400H~C59FH	3100H~329FH

When writing data from Super NES CPU to cache RAM, instructions must be written in 16-byte blocks. If data are written only part way through the 16 bytes, the flag will not be set for that block. In this case, the GSU will process as though cache data did not exist in that block. To set the cache flag, write any data to the XXXFH address of that block.

### 6.8.4 GSU EXCLUSIVE OPERATION IN CACHE RAM

By activating the GSU after code has been written from the Super NES CPU to the cache RAM, it is possible to operate the program exclusively in cache RAM. The CBR value is stored from the Super NES CPU by resetting the GO flag. This causes the CBR value to become 0000H. The program addresses in cache are normally 0000H through 01FFH, so the GSU is activated with addresses in this range stored in the program counter.

Please be aware that, even when a STOP instruction is executed, the next code has been loaded into the pipeline. If the address of the STOP instruction is XXXFH, the GSU will try to read code from external RAM unless the cache flag for the block containing the next address (XXX0H) has been set.

## ***Chapter 7 Data Access***

### **7.1 GAME PAK ROM DATA**

The GSU uses a function called the “ROM buffering system” as a method of loading data from game pak ROM during program execution. Using the ROM buffering system, register R14 is assigned as the address pointer to game pak ROM. When a value is set in register R14, the game pak ROM data at the address specified by ROMBR and register R14 are loaded to an internal buffer called the “ROM buffer”.

#### **7.1.1 GSU PROGRAM RUNNING IN CACHE RAM OR GAME PAK RAM**

When the program is running in cache RAM or game pak RAM, game pak ROM data can be loaded in parallel with the execution of instructions. Therefore, it is most efficient to sandwich several instructions between an instruction that changes R14 and a GETB instruction.

Care is required when performing the following operations while data are being loaded into the ROM buffer.

- If the value for R14 is updated, the initial loading process is interrupted and a new loading process is started.
- If a ROMB instruction is fetched, the program will wait until the data are loaded into the ROM buffer. The ROMBR value will be changed after data is loaded and program execution will resume.
- If a GETB or similar instruction is fetched, the program will pause while the data is loaded into the ROM buffer.

In the following examples, it is presumed that the program is being executed in cache RAM and bit 0 of the CLSR is “1” (Super FX operating frequency is 21.4 MHz).

#### **CAUTIONS**

If cache instructions are executed immediately after the value is set at R14, while the program is running on cache RAM, the proper value is not read to the ROM buffer. Please use caution when reading data from ROM.

- During 21.7 MHz operation, do not insert a CACHE instruction during the first 7 machine cycles after an instruction that changes the content of R14.
- During 10.7 MHz operation, do not insert a CACHE instruction during the first 4 machine cycles after an instruction that changes the content of R14.

## (Example 1)

Cycle	Instruction	Comment
2	MOVE R14,R1	;Start Fetching
5	GETB	;Get The Byte Into R0
1	TO R1	
1	FROM R2	
1	ADD R3	;Perform R1=R2+R3
1	TO R4	
1	FROM R5	
1	ADD R6	;Perform R4=R5+R6
1	ADD R8	;R0=R0+R8

Fourteen cycles are required to execute the program in the previous example. Since R0 is not used until the last instruction, the GETB instruction can be moved to the line before "ADD R8", as demonstrated below.

## (Example 2)

Cycle	Instruction	Comment
2	MOVE R14,R1	;Start Fetching
1	TO R1	
1	FROM R2	
1	ADD R3	;Perform R1=R2+R3
1	TO R4	
1	FROM R5	
1	ADD R6	;Perform R4=R5+R6
1	GETB	;Get The Byte Into R0
1	ADD R8	;R0=R0+R8

Only 10 cycles are required to execute this program. Read timing for game pak ROM access is as follows.

- Operating frequency 21.4 MHz: 5 cycles
- Operating frequency 10.7 MHz: 3 cycles

### 7.1.2 GSU PROGRAM RUNNING IN GAME PAK ROM

When the GSU program is running in game pak ROM, it is necessary to use the ROM buffering system even when loading game pak ROM data. The instruction following a change in register R14 will not begin execution until the ROM buffer is loaded.

## 7.2 GAME PAK RAM DATA

The GSU uses a function called the “RAM buffering system” as a method of loading data from game pak RAM during program execution. Using the RAM buffering system, the game pak RAM address and data to be written are moved to an internal buffer. The operation of writing to RAM is started by executing a STB, STW, SM, SMS, or SBK instruction.

### 7.2.1 GSU PROGRAM RUNNING IN CACHE RAM OR GAME PAK ROM

When the program is running in cache RAM or game pak ROM, its write data will be written to game pak RAM while the subsequent program is being executed. Therefore, it is most efficient to sandwich several instructions between STW instructions.

Care is required when performing the following operations while writing to game pak RAM.

- Execution of a command that updates the register which was used as the address in a STB or STW instruction will have absolutely no effect on the write operation to game pak RAM and will not wait.
- If a RAMB instruction is fetched, the program will wait until the data are written to game pak RAM. The RAMBR value will be changed after the write is completed and execution of the program will resume.
- If a STW instruction is fetched, the program will wait until the data are written to game pak RAM.

In the following examples, it is presumed that the program is being executed in cache RAM and bit 0 of the CLSR is “1” (Super FX operating frequency is 21.4 MHz).

(Example 3)

Cycle	Instruction	Comment
1	FROM R8	;Store R8 Into (R10)
1	STW (R10)	
10	STW (R11)	;Store R0 Into (R11)
1	TO R1	
1	FROM R2	
1	ADD R3	;Perform R1=R2+R3
1	FROM R5	
1	ADD R6	;Perform R0=R5+R6

Seventeen cycles are required to execute the program in the previous example. Since the value for R0 is not changed until the last instruction, the second STW instruction can be moved to the line immediately before that instruction. This is demonstrated on the following page.

#### (Example 4)

Cycle	Instruction	Comment
1	FROM R8	
1	STW (R10)	;Store R8 Into (R10)
1	TO R1	
1	FROM R2	
1	ADD R3	;Perform R1=R2+R3
7	STW (R11)	;Store R0 Into (R11)
1	FROM R5	
1	ADD R6	;Perform R0=R5+R6

Only 14 cycles are required to execute the program in Example 4. This is more efficient than Example 3, a wait period of 2 cycles is still required to write to game pak RAM.

#### 7.2.2 GSU PROGRAM RUNNING IN GAME PAK RAM

When the GSU program is running in game pak RAM, it is necessary to use the RAM buffering system described above even when writing game pak RAM data. The instruction following a STB or similar instruction is executed after completion of the write operation to game pak RAM.

### 7.3 BULK PROCESSING

Normally during bulk processing, data are loaded from game pak RAM, some processing is performed, and a process is executed to return the data to the same address. Waste can be avoided if the process can be completed without having to specify the address in RAM a second time.

When an instruction that performs a data transfer between the game pak RAM and an internal register is executed in the GSU, the game pak RAM address used in that instruction will be stored in memory. The SBK instruction stores the RAM address in which the register contents are stored. Since it does not require an operand, it can be executed more quickly than the SM or SMS instructions. The difference is demonstrated in the following two examples.

## (Example 5)

In the following example the SBK instruction is not used. In this case, word data have been read from game pak RAM address 1234H, the register contents are incremented, and again written to 1234H.

Cycle		Instruction	Comment
14		LM R0,(1234H)	;R0←(1234H)
1		INC R0	;R0←R0+1
4		SM (1234H),R0	;(1234H)←R0

Nineteen cycles are required to execute the above program. If the SBK instruction is used, the following occurs.

## (Example 6)

Cycle		Instruction	Comment
14		LM R0,(1234H)	;R0←(1234H)
1		INC R0	;R0←R0+1
1		SBK	;(1234H)←R0

In this example, only 16 cycles are required. The memory required to handle the program is also decreased.

## ***Chapter 8 GSU Special Functions***

The GSU performs various special functions to realize high-speed operations. These functions are described below.

### **8.1 BITMAP EMULATION**

Since a character mapping system is used with the Super NES PPU, its CPU can not efficiently perform processing such as; placing a point, drawing a line or painting a plane (bitmap graphics). Prior to display on the screen, this data must be converted to character data. Thereby, emulating the bitmap data.

The GSU is equipped with functions that support “Plot Processing”. These functions, “place a point of a specified color at a specified coordinate position.” Consequently; after setting the screen mode (CMODE instruction), the color data (COLOR, GETC instructions), and the X,Y coordinates; the PLOT instruction is performed.

In this manner, the GSU converts plotted (bitmapped) data to character data which can be utilized by the Super NES PPU and writes them to game pak RAM. In order to be displayed on screen, character data produced in the game pak RAM must be transferred by the Super NES CPU to the V-RAM of the Super NES.

#### **8.1.1 SET SCREEN MODE**

To begin GSU plot processing, screen mode assignments must be made. This is performed using the screen mode register (SCMR) and the screen base register (SCBR). The plot options are assigned using the CMODE instruction.

##### **8.1.1.1 SCREEN MODE REGISTER (SCMR)**

The GSU conversion process from bitmapped data to character data requires a screen mode selection. This determines how the characters will be aligned and the bit mode to be used. This is performed by assigning a mode to the SCMR using the Super NES CPU.

The GSU has 4 modes. A BG character array may be selected with screen heights of 128 dot, 160 dot and 192 dot. The fourth mode is an OBJ character array.

The character data conversion processing by the GSU is performed assuming that the character array is aligned as demonstrated in the following figures for BG 128 dot, BG 160 dot, BG 192 dot, or OBJ; respectively. Consequently, when the converted data are used as BG or OBJ character data for the Super NES, it is necessary to assign the screen mode and store the screen data in the VRAM.

256 DOT							
000	010	020	•	•	•	•	1F0
001	011	021	•	•	•	•	1F1
002	012	022	•	•	•	•	1F2
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
00F	01F	02F	•	•	•	•	1FF

Figure 2-8-1 128 Dot High BG Character Array (numbers are hexadecimal)

256 DOT							
000	014	028	•	•	•	•	26C
001	015	029	•	•	•	•	26D
002	016	02A	•	•	•	•	26E
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
013	027	03B	•	•	•	•	27F

Figure 2-8-2 160 Dot High BG Character Array (numbers are hexadecimal)

256 DOT							
000	018	030	•	•	•	•	2E8
001	019	031	•	•	•	•	2E9
002	01A	032	•	•	•	•	2EA
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
017	02F	047	•	•	•	•	2FF

Figure 2-8-3 192 Dot High BG Character Array (numbers are hexadecimal)

256 DOT											
000	001	002	•	•	•	•	00F	100	•	•	•
010	011	012	•	•	•	•	01F	110	•	•	11F
020	021	022	•	•	•	•	02F	120	•	•	12F
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	SC Data
•	•	•	•	•	•	•	•	•	•	•	•
0F0	0F1	0F2	•	•	•	•	OFF	1F0	•	•	1FF
200	201	202	•	•	•	•	20F	300	•	•	30F
210	211	212	•	•	•	•	21F	310	•	•	31F
220	221	222	•	•	•	•	22F	320	•	•	32F
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•	•	•	•	•
2F0	2F1	2F2	•	•	•	•	2FF	3F0	•	•	3FF

Figure 2-8-4 OBJ Character Array (numbers are hexadecimal)

To calculate the total number of bytes of character data required, the following formula is derived from the bit mode and the screen height and width.

Total number of bytes of character data =

$$\begin{array}{c}
 (\text{Number of dots high}/8) \quad \times \quad (\text{Number of dots wide}/8) \quad \times \quad (8n) \\
 \boxed{\text{Number of vertical characters}} \qquad \boxed{\text{Number of horizontal characters}} \qquad \boxed{\text{Number of bytes/char}}
 \end{array}$$

Where n equals the number of bits per dot (2,4, or 8).

#### 8.1.1.2 SCREEN BASE REGISTER (SCBR)

The start address of the area in game pak RAM where character data will be handled must be assigned in advance from the Super NES CPU. This information is stored in the SCBR.

The start address is calculated using the following formula.

$$(\text{Start Address}) = 70:0000H + \text{SCBR} \times 400H$$

For example, when the value 11H is stored in the SCBR, in 4-bit mode, with a height of 128 dots, width of 192 dots;

$$(\text{Start Address}) = 70:0000\text{H} + 11\text{H} \times 400\text{H} = 70:4400\text{H}$$

$$\begin{aligned} &(\text{Total number of bytes of character data}) \\ &= (128/8) \times (192/8) \times (8 \times 4) = 3000\text{H} \end{aligned}$$

game pak RAM addresses 70:4400H through 70:73FFH are used for the character data area.

#### 8.1.1.3 CMODE INSTRUCTION

The CMODE instruction must be stored in the plot option register (POR) to enable the PLOT instruction and COLOR or GETC instructions to be selected. The relationship between plot processing and the CMODE instruction is covered in more detail under “Plot Function and CMODE”, later in this chapter.

#### 8.1.2 SET COLOR (COLOR, GETC)

The color data used in plot processing must be stored in the GSU’s color register (COLR) using the COLOR instruction or the GETC instruction. If the COLOR instruction is used, the value for the source register is stored, while the GETC instruction stores the value for the ROM buffer.

#### 8.1.3 PLOT PROCESSING (PLOT)

The PLOT instruction plots the color data, stored by the COLOR or GETC instruction, to the X and Y coordinates stored in general registers R<sub>1</sub> and R<sub>2</sub>. The X coordinate value must be in R<sub>1</sub> and the Y coordinate value in R<sub>2</sub>. Color data plotted by the PLOT instruction are converted to character data and written to the game pak RAM.

Since it would be inefficient to perform a direct write to game pak RAM for each PLOT instruction, caching is performed in an 8-bit (1 pixel) x 8-bit memory inside the GSU. This corresponds with the 1 vertical pixel x 8 horizontal pixel blocks into which the screen is divided. This memory is called the “pixel cache” and the blocks that are cached are called “character blocks”.

There are two pixel cache memories in the GSU. The color data produced by the PLOT instruction is cached in the “primary pixel cache.” These data are copied to the “secondary pixel cache,” then written from the “secondary pixel cache” to game pak RAM. Each pixel cache has an 8-bit flag called the primary and secondary bit-pend flags. These indicate whether or not the color data in each pixel cache is valid.

When the PLOT instruction is executed, the offset address of game pak RAM where color data are written is calculated from the value in bit 7 through bit 3 of the X coordinate ( $R_1$ ) and the value in bit 7 through bit 0 of the Y coordinate ( $R_2$ ). These values are held in the GSU. When another PLOT instruction is executed, the GSU compares the new coordinate values to those stored. If the coordinates have not changed, plotting is performed to the same character block (stored in secondary cache) is written to game pak RAM.

The flow of GSU plot processing will be demonstrated below using two cases. In the first description, the character block which was stored by the previous PLOT instruction is to be written. The second case demonstrates plotting to a different block.

#### 8.1.3.1 PLOTTING TO SAME CHARACTER BLOCK

Color data are written to the pixel cache and the corresponding bit-pend flag is set. When all of the bit-pend flags are set (all 8 pixels of the cache block have been written), write processing to game pak RAM is performed in the following manner.

First, the contents of the primary pixel cache and the primary bit-pend flag are transferred to the secondary pixel cache and secondary bit-pend flag. If the contents of the secondary pixel cache are in the process of being written to the game pak RAM, this process is placed in WAIT status until the secondary pixel cache is empty.

After transfer processing, all of the primary bit-pend flags are cleared. Then the GSU executes the instruction following the PLOT instruction. Since the primary pixel cache can be used, the next instruction could be a PLOT instruction without requiring a WAIT status. Parallel with the execution of the next instruction, the GSU converts the color data in the secondary pixel cache into character data and writes them to the game pak RAM.

#### 8.1.3.2 PLOTTING TO A DIFFERENT CHARACTER BLOCK

The contents of the primary pixel cache and the primary bit-pend flag are transferred to the secondary pixel cache and secondary bit-pend flag. If the contents of the secondary pixel cache are in the process of being written to the game pak RAM, this process is placed in WAIT status until the secondary pixel cache is empty. Thereafter, color data are written to the primary pixel cache and the corresponding bit-pend flag is set.

The GSU then executes the instruction following the PLOT instruction. Parallel with the execution of this instruction, the GSU converts the color data in the secondary pixel cache to character data and writes it to game pak RAM.

The data in the corresponding character block are read from the game pak RAM and converted back, while the color data correspond with the flags which are not set in the secondary bit-pend flag are set in the secondary pixel cache. The GSU then converts the color data in the secondary pixel cache into character data and writes them to the game pak RAM.

Thus, the operation of writing to game pak RAM using two pixel caches can be performed in parallel with the execution of instructions, making PLOT processing very efficient. In addition, since the PLOT instruction increments the value for R1 after processing, there is no need to specify coordinates when writing the pixels continuously toward the right.

#### CAUTION

Do not change the setting of the screen mode, described under "Set Screen Mode," during plot operations. Also, when screen plot processing is completed, execute the RPIX instruction to write all of the data contained in the pixel caches to the game pak RAM.

#### (Example 1)

The following program is executed under the following conditions.

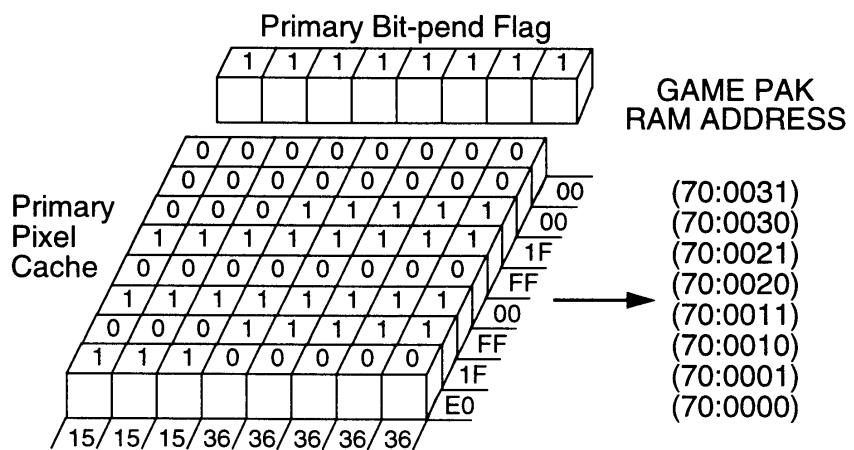
SCBR=00H, Color Mode=256, and Screen Mode=BG 128 dot high

```

IBT    R1,#0
IBT    R2,#0      ;Set the plot starting coordinate to (0,0)
IBT    R0,#0
CMODE
IBT    R0,#15H
COLOR
PLOT
PLOT
PLOT      ;Plot (0,0) through (2,0)
IBT    R0,#36H
COLOR
PLOT
PLOT
PLOT
PLOT      ;Plot (3,0) through (7,0)

```

The primary pixel cache becomes the cache RAM for the character block from coordinates (0,0) through (7,0). When the program is executed, the following values are stored in the primary pixel cache and the primary bit-pend flag.



Since all 8 pixels in a character block are set with the final PLOT instruction, they are transferred from the primary pixel cache to the secondary pixel cache and the game pak RAM write begins. This process clears the primary bit-pend flags and the primary pixel cache is released.

(Example 2)

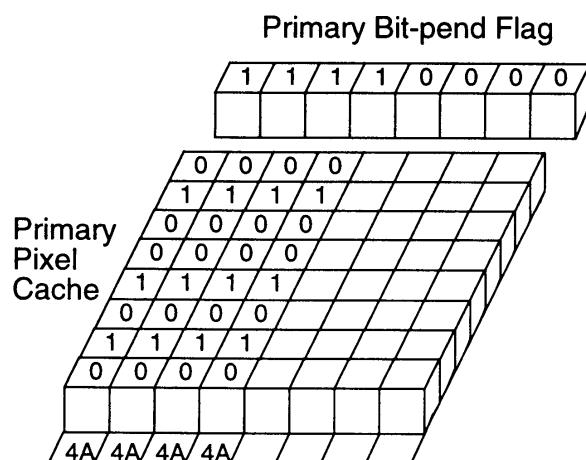
Continuing from Example 1, the following program is executed.

```

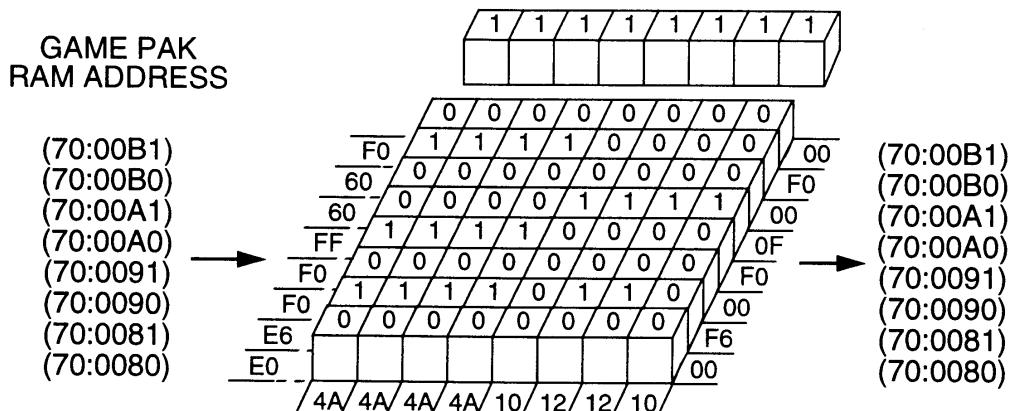
IBT    R0,#4AH
COLOR           ;Load 4AH to the color register
PLOT
PLOT
PLOT
PLOT           ;Plot (8,0) through (11,0)
IBT    R1,#10H  ;Change X coordinate to 16
PLOT           ;Plot (16,0)

```

The primary pixel cache becomes the cache for the character data from coordinates (8,0) through (15,0). Immediately after the 4th PLOT instruction is executed, the primary pixel cache and primary bit-pend flags are as shown below.



Since the last PLOT instruction writes to a different character block, RAM write processing is performed. First, a transfer is performed from the primary pixel cache and primary bit-pend flag to the secondary pixel cache and secondary bit-pend flag. Then, game pak RAM write processing is performed, but the pixels in the secondary pixel cache which have not been plotted are written after a game pak RAM read operation has been executed.



### 8.1.3.3 RPIX INSTRUCTION

The RPIX instruction reads the character block containing the specified coordinates from game pak RAM into the pixel cache and performs processing to calculate the pixel values after the contents of the pixel cache have been written to the game pak RAM. When the screen drawing routine is complete, it is advisable to execute the RPIX instruction to insure that all of the PLOT data have been written.

If consecutive RPIX instructions are executed, game pak RAM read data processing will always be performed because the instruction does not discern whether or not there are color data at the specified coordinates in the pixel cache.

#### CAUTION

Even when consecutive RPIX instructions read color data from the same character block, data will always be read from the game pak RAM.

### 8.1.4 PLOT FUNCTION AND CMODE

The CMODE instruction is used to determine how the color register value will be handled by the PLOT instruction. The modes which can be specified with CMODE are shown in the table below.

BIT	Flag Name	Operation when 0	Operation when 1	Related Instructions
0	Transparent Flag	Do not PLOT color 0	PLOT color 0	PLOT
1	Dither Flag	PLOT value of low 4 bits of color register	Alternately PLOT high 4 bits and low 4 bits of color register	PLOT
2	High Nibble Flag	Set value of low 4 bits in color register	Set value of high 4 bits in color register	COLOR, GETC
3	Freeze High Nibble Flag	Set all 8 bits in color register	Set only low 4 bits in color register with high 4 bits fixed	COLOR,GETC, PLOT
4	OBJ Mode Flag	Set mode with SCMR (ht0,ht1)	OBJ mode	PLOT,RPIX

Table 2-8-1 Functions of CMODE

The PLOT instruction is related to bit 3, but it is also used during PLOT processing for selecting the number of bits to be used (0=8 Bit, 1=4 Bit) for transparent processing.

#### 8.1.4.1 BIT 0

The Super NES has multiple hardware BG screens. When one BG screen is laid over another BG screen, the 0 portions of the color in the top BG screen become "transparent" and the colors of the bottom BG are displayed. The GSU uses color mode 0 to perform this function.

When Bit 0=0 and all of the effective COLR bits are 0, the PLOT circuit refreshes only the X coordinate and no PLOT operation is performed. Normal PLOT operation is performed for anything other than 0.

#### 8.1.4.2 BIT 1

When the number of colors that can be displayed at once is low (16 color mode), techniques can be used to apparently increase the number of colors through dither processing. The GSU is able to process this with extreme ease. The example below demonstrates the difficulties encountered when this function is not used.

(Example 3)

Routine for drawing a horizontal line of a specified length from a specified coordinate using two alternating specified colors.

R1:Start X position  
 R2:Start Y position  
 R3:Color 1  
 R4:Color 0  
 R12:Line length

```

MOVE   R13,R15 ;Set LOOP return address.
;LOOP return address
FROM   R1
XOR    R2
AND    #1      ;Execute [R0=(R1 XOR R2)And 1].
BNE    DOPLOT
FROM   R3      ;When not zero, set R3 (color 1) to Sreg.
FROM   R4      ;When zero, set R4 (color 0) to Sreg.
DOPLOT: COLOR   ;Set value of Sreg in COLR.
              PLOT
              LOOP
              NOP

```

Thus, if only the plotting functions are used, it takes time to determine which of the two colors to PLOT at a specified time.

The bit 1 dither flag may be used to efficiently perform this type of drawing process. The dither mode is only functional in 4 color mode and 16 color mode.

When dither mode is set, the PLOT circuit checks the bit 0 value of the result when an XOR operation is performed on R1 (X coordinate) and R2 (Y coordinate). If the resultant bit 0=0, the low 4 bits of the COLR register are used as the color data for the PLOT instruction. However, if the resultant bit 0=1, the high 4 bits of the COLR register are used.

When the program in the previous example is written using the CMODE instruction, only the PLOT instruction is looped, as demonstrated below.

(Example 4)

```

IBT      R0,#2
CMODE    ;Set to transparent and dither mode.
FROM     R3
ADD      R3
ADD      R0
ADD      R0
ADD      R0      ;Shift low 4 bits of COLOR1 to high 4 bits.
ADD      R4      ;Add value of R4 (COLOR0) to R0.
COLOR    ;Set COLR.
MOVE     R13,R15
;LOOP return address
LOOP
PLOT      ;Plot pixel.

```

Since the processing to determine whether or not a color is transparent is performed in parallel with the generation of plot data, dithering cannot be performed between a transparent color and a normal color. This mode can also be used in the 4 color mode.

#### 8.1.4.3 BIT 2

To efficiently perform rotation/enlargement/reduction of OBJ data, a system is used in which each pixel of color data is stored at one address. When displaying a 16 color OBJ, half of the memory is wasted using this method. Memory may be conserved by storing two pixels of color data together in one byte. However, this requires a method for extracting two pixels of color data from one byte of data. Bit 2 of CMODE is used by the GSU to perform this function.

When the COLOR or GETC instruction is executed with bit 2 of CMODE set, the high 4 bits of the source register are written to the color register. If different OBJ data are stored in the high 4 bits and low 4 bits of the same memory area, this function permits the packed 8-bit data to be used without shift processing. This mode can also be used in 4 color mode.

#### 8.1.4.4 BIT 3

If the COLOR or GETC instruction is executed in 256 color mode with bit 3 of CMODE set, only the low 4 bits of the COLR register can be written to the color register. The high 4 bits are fixed. This function enables the high 4 bits of the color register to be used in place of a palette in 256 color mode. In other words, characters of different colors can be drawn by plotting 16 color mode data while changing the value of the high 4 bits of the color register.

#### 8.1.4.5 BIT 4

When bit 4 of CMODE is set, the mode which enables character data to be produced for OBJ. When this bit is 0, the mode is specified by HT0,HT1 of the SCMR. When switching the OBJ mode by changing this bit, it will be necessary to use the RPIX instruction to write the data to the game pak RAM which have already been written to the pixel caches.

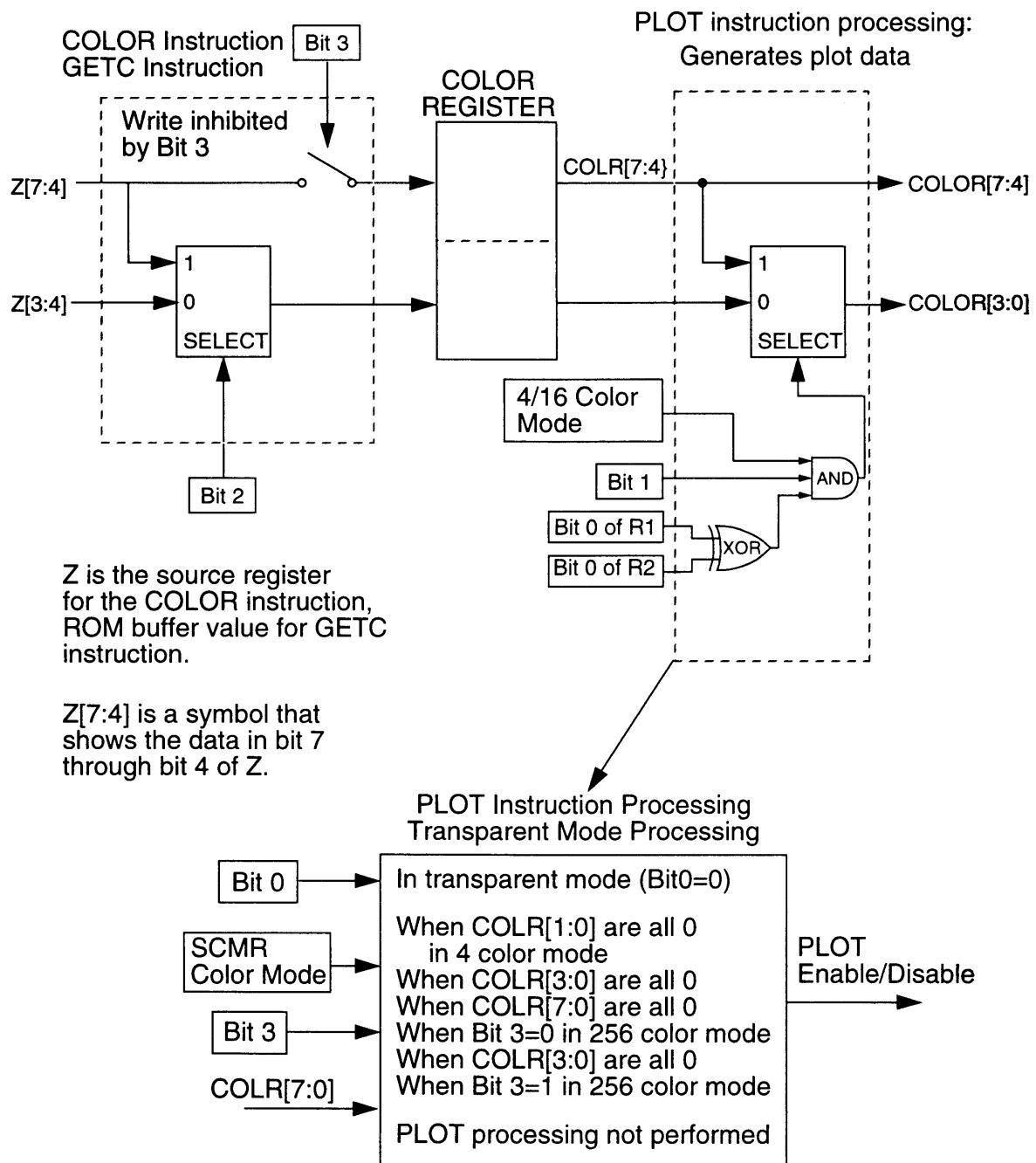


Figure 2-8-5 Plot Operations Assigned by CMODE

### 8.1.5 PLOT DATA ADDRESS CALCULATION METHODS

The addresses to which plot data are written are determined using the following data.

- X and Y coordinates are specified by the low bytes of R<sub>1</sub> and R<sub>2</sub>.
- The screen color mode and height mode are specified by the SCMR.
- SCBR

The following examples demonstrate the method of calculating this address. In the calculations below, "X[7:3]" indicates the value of bit 7 through 3 for the value of X. The expression "X4," indicates the value of bit 4 for X.

1. Calculate the character number (CN) containing the specified coordinates. CN is the value of SC data in the character arrays previously described.

(a) Height, 128 Dot Mode

$$CN [9:0] = (X[7:3] \times 10H) + Y[7:3]$$

X7	X6	X5	X4	X3	Y7	Y6	Y5	Y4	Y3
+					Y7	Y6	Y5	Y4	Y3
CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0

(b) Height, 164 Dot Mode

$$CN [9:0] = (X[7:3] \times 14H) + Y[7:3]$$

X7	X6	X5	X4	X3	X4	X3	Y7	Y6	Y5	Y4	Y3
+			X7	X6	X5	X4					
CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0		

(c) Height, 192 Dot Mode

$$CN [9:0] = (X[7:3] \times 18H) + Y[7:3]$$

X7	X6	X5	X4	X3	X4	X3	Y7	Y6	Y5	Y4	Y3
+			X7	X6	X5	X4					
CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0		

## (d) OBJ Mode

$$CN[9:0] = (Y[7] \times 200H) + (X[7] \times 100H) + (Y[6:3] \times 10H) + Y[6:3]$$

X7		X6	X5	X4	X3		
+ Y7	Y6	Y5	Y4	Y3			
CN9	CN8	CN7	CN6	CN5	CN4		
				CN3	CN2	CN1	CN0

2. The addresses to be written to are then calculated as follows.

$$\begin{aligned} A[19:0] = & (CN[9:0] \times \text{CHAR\_SIZE}) \\ & + (\text{SB}[7:0] \times 4000H) \\ & + (Y[2:0] \times 2) \\ & + (\text{PL}[2] \times 200H) + (\text{PL}[1] \times 100H) + \text{PL}[0] \end{aligned}$$

Where CHAR\_SIZE is the number of bytes used for one character. This is 16 bytes for 4 color mode, 32 bytes for 16 color mode, and 64 bytes for 256 color mode. The expression "PL[2:0]" indicates a plane number. The expression "SB[7:0]" indicates the value stored at the SCBR. The following examples demonstrate this calculation.

## (a) 4 Color Mode

$$\begin{array}{ccccccccccccc} & SB7 & SB6 & SB5 & SB4 & SB3 & SB2 & SB1 & SB0 \\ & CN9 & CN8 & CN7 & CN6 & CN5 & CN4 & CN3 & CN2 & CN1 & CN0 \\ + & & & & & & & & & & & & & & \\ & A19 & A18 & A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 & A8 & A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \end{array} \quad \begin{array}{ccc} Y2 & Y1 & Y0 \\ & & PL0 \end{array}$$

## (b) 16 Color Mode

$$\begin{array}{ccccccccccccc} & SB7 & SB6 & SB5 & SB4 & SB3 & SB2 & SB1 & SB0 \\ & CN9 & CN8 & CN7 & CN6 & CN5 & CN4 & CN3 & CN2 & CN1 & CN0 \\ + & & & & & & & & & & & & & & & & & & & \\ & A19 & A18 & A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 & A8 & A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \end{array} \quad \begin{array}{ccc} Y2 & Y1 & Y0 \\ & & PL1 \\ & & PL0 \end{array}$$

## (c) 256 Color Mode

$$\begin{array}{ccccccccccccc} & SB7 & SB6 & SB5 & SB4 & SB3 & SB2 & SB1 & SB0 \\ & CN9 & CN8 & CN7 & CN6 & CN5 & CN4 & CN3 & CN2 & CN1 & CN0 \\ + & & & & & & & & & & & & & & & & & & & \\ & A19 & A18 & A17 & A16 & A15 & A14 & A13 & A12 & A11 & A10 & A9 & A8 & A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \end{array} \quad \begin{array}{ccc} Y2 & Y1 & Y0 \\ & & PL2 \\ & & PL1 \\ & & PL0 \end{array}$$

## 8.2 MULTIPLICATION INSTRUCTIONS

The 4 multiplication instructions shown below are available in the GSU.



There is an 8 bit x 8 bit multiplier built into the GSU. Since this multiplier is used only once with the MULT and UMULT instructions, these instructions can be executed at high speed. A 16 bit x 16 bit answer is calculated, for the LMULT and FMULT instructions, by performing an 8 bit x 8 bit multiplication 4 times.

The execution speed of each multiplication instruction can be changed using bit 5 of the CFGR. Normally, the standard speed mode (bit 5=0) is used. When the Super FX operates at 10.7 MHz (when bit 0 of the CCSR is "0"), the high speed mode (bit 5=1) can be used. If R4 is specified as the destination register with the LMULT instruction, the high 16 bits of the operation results are stored in R4.

**CAUTION**

If R4 is specified as the destination register with the FMULT instruction, the operation results will not be stored in R4 and the results will be lost. Do not specify R4 as the destination register for the FMULT instruction.

### 8.2.1 INTERNAL PROCESSING OF FMULT AND LMULT

For LMULT and FMULT instructions, 16 bit x 16 bit multiplication is performed by repeating an 8 bit x 8 bit multiplication circuit whose signed and unsigned numbers could both be present 4 times. The processing flow for the FMULT and LMULT instructions is explained below. The FMULT and LMULT instructions share the circuit, but notice that there are processes that can only be performed by the LMULT instruction.

Initially, an 18 bit buffer used to hold the partial results during multiplication, called the partial product buffer, is cleared.

The first multiplication is performed.

Low 8 bits of Sreg (unsigned) x Low 8 bits of R6 (unsigned)  
→ 16 bit result (unsigned)

The high 8 bits of the result are stored in the low 8 bits of the partial product buffer. For LMULT, the low 8 bits of the result are stored in the low 8 bits of R4.

The second multiplication is performed.

High 8 bits of Sreg (signed) x Low 8 bits of R6 (unsigned)  
→ 16 bit result (signed)

The result is expanded to 18 bits with the sign and added to the partial product buffer.

The third multiplication is performed.

Low 8 bits of Sreg (unsigned) x High 8 bits of R6 (signed)  
→ 16 bit result (signed)

The result is expanded to 18 bits with the sign and added to the partial product buffer. For LMULT, the low 8 bits of the partial product buffer are further stored in the high 8 bits of R4.

The fourth multiplication is performed.

High 8 bits of Sreg (signed) x High 8 bits of R6 (signed)  
→ 16 bit result (signed)

The result (16 bits) is added to the high 10 bits of the partial product buffer. For LMULT if the Dreg is R4, the value in the partial product buffer is stored in R4. If the Dreg is not R4, the value of the partial product buffer is stored in the Dreg.

If R4 is specified as the destination register for the LMULT instruction when performing the above processing, the high 16 bits of the operation result will be stored in R4. However, if R4 is specified as the destination register for the FMULT instruction, the operation result will not be stored as the value for R4.

## ***Chapter 9 Description of Instructions***

This chapter provides a detailed description of each instruction and its function. ROM and RAM execution times listed for each instruction refer to the game pak ROM and RAM. Special indicators and symbols are used throughout this chapter. These are defined in the following 3 tables.

### **9.1 OPERAND DESCRIPTIONS**

INDICATOR	DESCRIPTION
$R_0$	Indicates internal register $R_0$ .
$R_n$	A 16-bit general use register.
$R_n'$	A 16-bit general use register.
$(R_m)$	Indicates the value stored in the memory location specified by the contents of register $R_m$ .
$(xx)$	Indicates the value stored in the memory location specified by the 16-bit value $xx$ .
$(yy)$	Indicates the value stored in the memory location specified by the 9-bit value $yy$ . ( $0 \leq yy \leq 510$ )
#n	Indicates 4-bit immediate data.
#xx	Indicates 16-bit immediate data. ( $0 \leq xx \leq 65535$ )
#pp	Indicates 8-bit immediate data. ( $-128 \leq pp \leq 127$ )
e	1-byte data $-128 \leq e \leq 127$ , that expresses the displacement in the relative addressing mode.

### **9.2 FLAG DESCRIPTIONS**

SYMBOL	DESCRIPTION
1	Set
0	Reset
*	Set or reset according to results.
-	No change

### 9.3 OPERATOR FUNCTIONS

INDICATOR	DESCRIPTION
$R_0$	Indicates internal register $R_0$ .
$R_n, R_n'$	A 16-bit general use register specified by n.
$(R_m)$	Indicates a value stored in a memory location specified by the contents of register $R_m$ .
$(xx)$	Indicates a value stored in a memory location specified by the 16-bit value xx.
$(yy)$	Indicates a value stored in a memory location specified by the 9-bit value yy.
#n	Indicates 4-bit immediate data.
#xx	Indicates 16-bit immediate data. ( $0 \leq xx \leq 65535$ )
#pp	Indicates 8-bit immediate data. ( $-128 \leq pp \leq 127$ )
e	1-bit data ( $-128 \leq e \leq 127$ ), that expresses displacement in the relative addressing mode.
$S_{reg}$	Source register
$D_{reg}$	Destination register
High-Byte	Upper byte of 16-bit data
Low-Byte	Lower byte of 16-bit data
$\rightarrow$	Indicates direction of movement of data
+	Add
-	Subtract
*	Multiply
$\overline{R_n}, \overline{\#n}$	1's compliment
ALT1	ALT1 Flag
ALT2	ALT2 Flag
CY	Carry Flag
O/V	Overflow Flag
Z	Zero Flag
S	Sign Flag
B	B Flag
GO	Go Flag

## 9.4 ADC R<sub>n</sub>

- Operation:  $S_{reg} + R_n + CY \text{ Flag} \rightarrow D_{reg}$  (n=0~15)
- Description: This instruction adds the source register, the operand, and the carry flag. The result is stored in the destination register.
- Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.
- The operand can be any of registers R<sub>0</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B: Reset  
 ALT1: Reset  
 ALT2: Reset  
 O/V: Set on signed overflow.  
 S: Set if result is negative, else reset  
 CY: Set on unsigned carry, else reset  
 Z: Set if result is zero.

Opcode:

ADC R <sub>n</sub>	(MSB)								(LSB)	
	0	0	1	1	1	1	0	1	(3DH)	n (0H~FH)

- Machine Cycles:
- |                          |          |
|--------------------------|----------|
| ROM execution time       | 6 cycles |
| RAM execution time       | 6 cycles |
| Cache RAM execution time | 2 cycles |

- Example:
- |                     |  |
|---------------------|--|
| ADC R <sub>1</sub>  | ; R <sub>0</sub> +R <sub>1</sub> +CY→R <sub>0</sub>      |
| WITH R <sub>2</sub> | ; Set the source/destination registers to R <sub>2</sub> |
| ADC R <sub>3</sub>  | ; R <sub>2</sub> +R <sub>3</sub> +CY→R <sub>2</sub>      |
| ADC R <sub>2</sub>  | ; R <sub>0</sub> +R <sub>2</sub> +CY→R <sub>0</sub>      |

## 9.5 ADC #n

Operation:  $S_{reg} + \#n + CY \text{ Flag} \rightarrow D_{reg}$  (n=0~15)

Description: This instruction adds the source register, the immediate data specified by the operand #n, and the carry flag. The result is stored in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be immediate data from 0~15.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B: Reset  
 ALT1: Reset  
 ALT2: Reset  
 O/V: Set on signed overflow.  
 S: Set if result is negative, else reset  
 CY: Set on unsigned carry, else reset  
 Z: Set if result is zero, else reset.

Opcode:

ADC #n	(MSB)								(LSB)	
	0	0	1	1	1	1	1	1	(3FH)	n (0H~FH)

Machine Cycles: ROM execution time 6 cycles

RAM execution time 6 cycles

Cache RAM execution time 2 cycles

Example:

ADC	#9H	; R <sub>0</sub> +0009H+CY→R <sub>0</sub>
FROM	R <sub>3</sub>	; Set the source register to R <sub>3</sub>
ADC	#5H	; R <sub>3</sub> +0005H+CY→R <sub>0</sub>
ADC	#0AH	; R <sub>0</sub> +000AH+CY→R <sub>0</sub>

## 9.6 ADD R<sub>n</sub>

Operation:  $S_{\text{reg}} + R_n \rightarrow D_{\text{reg}}$  (n=0~15)

Description: This instruction adds the source register and the register specified by the operand R<sub>n</sub>. The result is stored in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be any of registers R<sub>0</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B:	Reset
ALT1:	Reset
ALT2:	Reset
O/V:	Set on signed overflow.
S:	Set if result is negative, else reset
CY:	Set on unsigned carry, else reset
Z:	Set if result is zero.

Opcode:

ADD R <sub>n</sub>	(MSB)				(LSB)	
	0	1	0	1	n (0H~FH)	(5nH)

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycle

Example: Under the following conditions:

S<sub>reg</sub>: R<sub>0</sub>, D<sub>reg</sub>: R<sub>0</sub>, R<sub>0</sub>=4283H, R<sub>4</sub>=2438H

R<sub>0</sub>=66BBH when ADD R<sub>4</sub> is executed.

```

ADD  R4      ; R0+R4→R0
TO   R5      ; Set the destination register to R5
ADD  R6      ; R0+R6→R5
ADD  R3      ; R0+R3→R0

```

## 9.7 ADD #n

- Operation:  $S_{\text{reg}} + \#n \rightarrow D_{\text{reg}}$  (n=0~15)
- Description: This instruction adds the source register to the immediate data specified by the operand #n. The result is stored in the destination register.
- Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.
- The operand can be immediate data from 0-15.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B: Reset  
 ALT1: Reset  
 ALT2: Reset  
 O/V: Set on signed overflow, else reset.  
 S: Set if result is negative, else reset  
 CY: Set on unsigned carry, else reset.  
 Z: Set on zero result, else reset.

Opcode:

ADD #n	(MSB)								(LSB)	
	0	0	1	1	1	1	1	0	(3EH)	n (0H~FH)

- Machine Cycles: ROM execution time 6 cycles  
 RAM execution time 6 cycles  
 Cache RAM execution time 2 cycles

Example: Under the following conditions:

$S_{\text{reg}}: R_4, D_{\text{reg}}; R_7, R_4=3682H$   
 $R_7$  is 368AH when ADD #8H is executed.

```

ADD  #8H      ; R4+0008H→R7
WITH R7      ; Set the source and destination registers to R7
ADD  #2H      ; R7+0002H→R7
ADD  R7      ; R0+R7→R0
  
```

## 9.8 ALT1

### FLAG PREFIX INSTRUCTION

**Operation:** 1 → ALT1 Flag

**Description:** ALT1 is a prefix instruction used in combination with the instruction which follows. When ALT1 is executed, the Super FX sets the ALT1 flag in bit 8 of the status flag register (3030, 3031H).

The ALT1 flag specifies the mode for the next instruction.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
-	1	-	-	-	-	-

ALT1: Set

**Opcode:**

ALT1	(MSB)				(LSB)			
	0	0	1	1	1	1	0	1
	(3DH)							

**Machine Cycles:** ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

**Example:** Execution of the ALT1 instruction sets the ALT1 flag. Various instructions can be executed, depending upon the instruction which follows the ALT1 prefix.

(Refer to, “ALT1 (\$3D) +”, in the Super FX Opcode Matrix at the end of this chapter.)

## 9.9 ALT2

### FLAG PREFIX INSTRUCTION

Operation:  $1 \rightarrow \text{ALT2 Flag}$

Description: ALT2 is a prefix instruction used in combination with the instruction which follows. When ALT2 is executed, the Super FX sets the ALT2 flag in bit 9 of the status flag register (3030, 3031H).

The ALT2 flag specifies the mode for the next instruction.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	1	-	-	-	-

ALT2: Set

Opcode:

ALT2	(MSB)								(LSB)	
	0	0	1	1	1	1	1	0	(3EH)	

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

Example: Execution of the ALT2 instruction sets the ALT2 flag. Various instructions can be executed, depending upon the instruction which follows the ALT2 prefix.

(Refer to, "ALT2 (\$3E) +", in the Super FX Opcode Matrix at the end of this chapter.)

## 9.10 ALT3

### FLAG PREFIX INSTRUCTION

**Operation:**       $1 \rightarrow \text{ALT1 Flag}$   
                          $1 \rightarrow \text{ALT2 Flag}$

**Description:**      ALT3 is a prefix instruction used in combination with the instruction which follows. When ALT3 is executed, the Super FX sets the ALT1 and ALT2 flags in bits 8 and 9 of the status flag register (3030, 3031H).

These flags specify the mode for the next instruction.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
-	1	1	-	-	-	-

ALT1:      Set  
               ALT2:      Set

**Opcode:**

ALT3	(MSB)				(LSB)			
	0	0	1	1	1	1	1	(3FH)

**Machine Cycles:**      ROM execution time      3 cycles

RAM execution time      3 cycles

Cache RAM execution time      1 cycles

**Example:**      Execution of the ALT3 instruction sets the ALT 1 and ALT 2 flags. Various instructions can be executed, depending upon the instruction which follows the ALT3 prefix.

(Refer to, “ALT3 (\$3F) +”, in the Super FX Opcode Matrix at the end of this chapter.)

## 9.11 AND R<sub>n</sub>

Operation: S<sub>reg</sub> AND R<sub>n</sub> → D<sub>reg</sub> (n=1~15)

Description: This instruction performs logical AND on corresponding bits of the source register and the operand R<sub>n</sub>. The result is stored in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be any of registers R<sub>1</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B: Reset  
 ALT1: Reset  
 ALT2: Reset  
 S: Set if result is negative, else reset.  
 Z: Set on zero result, else reset.

Opcode: (MSB) (LSB)

ADD R<sub>n</sub>

0	1	1	1	n (1H~FH)
---	---	---	---	-----------

 (7nH)

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

Example:

```

AND  R8      ; R0 AND R8 → R0
        (163AH) (00FFH) → (003AH)
FROM R9      ; Set the source register to R9
TO   R10     ; Set the destination register to R10
AND  R7      ; R9 AND R7 → R10
        (55AAH) (FF00H) → (5500H)

```

## 9.12 AND #n

Operation: Sreg AND #n → Dreg (n=1~15)

**Description:** This instruction performs logical AND on corresponding bits of the source register and the immediate data specified by the operand #n. The result is stored in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be immediate data from 1~15.

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B: Rese

**ALT1:** Reset

**ALT2:** Reset

S: Set if result is negative, else reset

Z: Set on zero result, else reset.

## Opcode:

(MSB) (LSB)

(LSB)

ADD #n	0	0	1	1	1	1	1	0	(3EH)
	0	1	1	1			n (1H~FH)		(7nH)

Machine Cycles: ROM execution time 6 cycles

RAM execution time 6 cycles

Cache RAM execution time 2 cycles

Example: When register B<sub>0</sub> is “3E5DH (0011 1110 0101 1101B)”

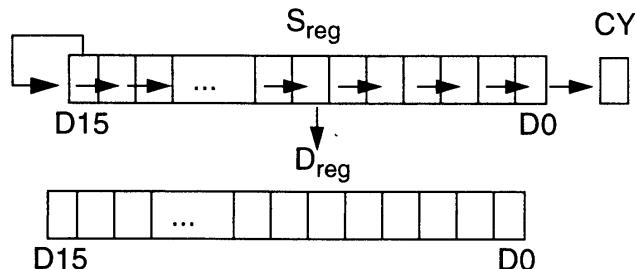
AND #6H

will result in

$R_0 = "0004H\ (0000\ 0000\ 0000\ 0100B)"$ .

## 9.13 ASR

Operation:



Description: This instruction shifts all bits in the source register one bit to the right. Bit 0 goes into the carry flag and bit 15 is unaffected. The result is stored in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

B: Reset  
 ALT1: Reset  
 ALT2: Reset  
 S: Set if result is negative, else reset  
 CY: Set if bit 0 in the source register is "1", else reset  
 Z: Set on zero result, else reset.

Opcode: (MSB) (LSB)

1	0	0	1	0	1	1	0	(96H)
---	---	---	---	---	---	---	---	-------

Machine Cycles: ROM execution time 3 cycles  
 RAM execution time 3 cycles  
 Cache RAM execution time 1 cycles

Example: Under the following conditions,

$S_{reg} : R_{10}$ ,  $D_{reg} : R_1$

CY	bit 15	bit0
0	R <sub>10</sub>	0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 1 (4F7BH)

When ASR is executed, the carry flag and R<sub>1</sub> are:

CY	bit15	bit0
1	R <sub>1</sub>	0 0 1 0 0 1 1 1 1 0 1 1 1 1 0 1 (27BDH)

**9.14 BCC e**

Operation: If CY Flag=0  
 then  $R_{15}+e \rightarrow R_{15}$  ( $e = -128 \sim +127$ )  
 $R_{15}$  identifies the next address for the BCC instruction (2 bytes)

Description: If the carry flag is "0", add "e" to the contents of the program counter  $R_{15}$  and JUMP to the address indicated by the resulting value in the program counter.

If the carry flag is "1", do not jump.

The relative offset can be -128 to +127 bytes from the address following the code for "e".

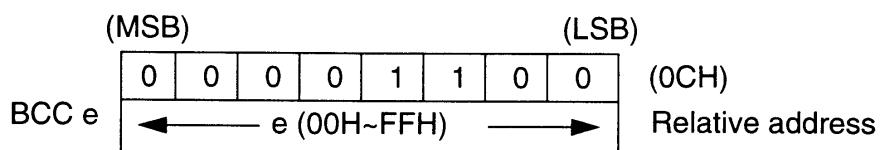
If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode:



Note: The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles: ROM execution time 6 cycles  
 RAM execution time 6 cycles  
 Cache RAM execution time 2 cycles

**Example:** In the following example, the carry flag is zero and the program jumps forward 5 bytes from the execution address of the instruction.

BCC \$+5H

The relationship between the program and the program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
51E	0C <input type="text"/>	(BCC \$+5H)
51F	03 <input type="text"/>	
520	←PC before jump <input type="text"/>	
521		
522		
523	←PC after jump <input style="border: 1px solid black; width: 20px; height: 15px; vertical-align: middle;" type="text"/> ←	Execute instruction at address 520 and jump.
.	.	
.	.	
.	.	

**9.15 BCS e**

Operation: If CY Flag=1  
 then  $R_{15}+e \rightarrow R_{15}$  ( $e = -128 \sim +127$ )  
 $R_{15}$  identifies the next address for the BCS instruction (2 bytes)

Description: If the carry flag is "1", add "e" to the program counter  $R_{15}$  and JUMP to the address indicated by the resulting value in the program counter.

If the carry flag is "0", do not jump.

The relative offset can be -128 to +127 bytes from the address following the code for "e".

If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode: (MSB) (LSB)  
 BCS e      

0	0	0	0	1	1	0	1
← e (00H~FFH) →							

 (0DH)  
 Relative address

Note: The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles: ROM execution time 6 cycles  
 RAM execution time 6 cycles  
 Cache RAM execution time 2 cycles

**Example:** In the following example, the carry flag is set and the program jumps backward 1 byte from the execution address of the instruction.

BCS \$-1H

The relationship between the program and the program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
42D	←PC after jump	
42E	0D	(BCS \$-1H)
42F	FD	
430	←PC before jump	
431		
.	.	
.	.	

Execute instruction at address 430 and jump.

9.16 BEQ e

Operation: If Z Flag=1

then  $R_{15+e} \rightarrow R_{15}$

(e= -128~+127)

R<sub>15</sub> identifies the next address for the BEQ instruction (2 bytes)

Description: If the zero flag is “1”, add “e” to the program counter R<sub>15</sub> and JUMP to the address indicated by the resulting value in the program counter.

If the zero flag is “0”, do not jump.

The relative offset can be -128 to +127 bytes from the address following the code for "e".

If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

### Opcode:

(MSB)

(LSB)

Note:-

The number “e” (number, label, formula) which shows the jump destination is given in the assembler as an operand

### Machine Cycles:

### ROM execution time

6 cycles

## BAM execution time

6 cycles

Cache RAM execution time 3 cycles

Example: In the following example, the zero flag is set and the program jumps ahead 5 bytes from the execution address of the instruction.

BEQ \$+5H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
15FD		
15FE	09	
15FF	03	□ (BEQ \$+5H)
1600		←PC before jump
1601		
1602		
1603		←PC after jump

Execute instruction at address 1600 and jump.

9.17 BGE e

Operation: If  $(S \text{ XOR } O/V) = 0$

then  $R_{15} + e \rightarrow R_{15}$

(e= -128~+127)

R<sub>15</sub> identifies the next address for the BGE instruction (2 bytes)

Description: If the sign flag and the overflow flag are equal, add “e” to the program counter  $R_{15}$  and JUMP to the address indicated by the resulting value in the program counter.

If the values are different, do not jump.

The relative offset can be -128 to +127 bytes from the address following the code for “e”.

If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

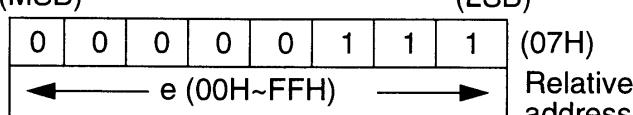
No flags affected

### Opcode:

(MSB)

(1 SB)

BGF e



Note:

The number “e” (number, label, formula) which shows the jump destination is given in the assembler as an operand.

### Machine Cycles:

### ROM execution time

6 cycles

### RAM execution time

6 cycles

### Cache RAM execution time

2 cycles

**Example:** In the following example, the sign flag and over flag are set and the program jumps backward 3 bytes from the execution address of the instruction.

BGE \$-3H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>
22FA	
22FB	←PC after jump
22FC	
22FD	
22FE	07 █ (BGE \$-3H)
22FF	FB █
2300	←PC before jump

Execute instruction at address 2300 and jump.

## 9.18 BIC R<sub>n</sub>

Operation: S<sub>reg</sub> AND  $\overline{R_n} \rightarrow D_{reg}$  (n=1~15)

Description: This instruction performs logical AND on corresponding bits of the source register and the 1's complement of register specified in the operand R<sub>n</sub>. The result is stored in the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be any of registers R<sub>1</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set if result is negative, else reset  
 Z : Set on zero result, else reset.

Opcode:

BIC R <sub>n</sub>	(MSB)				(LSB)			
	0	0	1	1	1	1	0	1
	(3DH)							
	(7nH)							

Machine Cycles:

ROM execution time	6 cycles
RAM execution time	6 cycles
Cache RAM execution time	2 cycles

Example: Under the following conditions:

S<sub>reg</sub>: R<sub>2</sub>, D<sub>reg</sub>: R<sub>0</sub>

R<sub>2</sub>=75CEH (0111 0101 1100 1110B),  
R<sub>1</sub>=3846H (0011 1000 0100 0110B)

R<sub>0</sub> is 4588H (0100 0101 1000 1000B) when  
BIC R<sub>1</sub>  
is executed.

## 9.19 BIC #n

**Operation:**  $S_{reg} \text{ AND } \overline{n} \rightarrow D_{reg}$  (n=1~15)

**Description:** This instruction performs logical AND on corresponding bits of the source register and the 1's complement of the immediate data specified in the operand #n. The result is stored in the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

The operand can be immediate data from 1~15.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set if result is negative, else reset  
 Z : Set on zero result, else reset.

Opcode: BIC #n	(MSB)								(LSB)				
	0	0	1	1	1	1	1	1	(3FH)				
	0	1	1	1	n (1H~FH)								(7nH)

<b>Machine Cycles:</b>	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** Under the following conditions:

$S_{reg}$ : R<sub>4</sub>,  $D_{reg}$ : R<sub>5</sub>  
 $R_4 = 364BH$  (0011 0110 0100 1011B)  
R<sub>5</sub> is 3640H (0011 0110 0100 0000B) when  
BIC #F  
is executed.

## 9.20 BLT e

Operation: If  $(S \text{ XOR } O/V) = 1$   
               then  $R_{15} + e \rightarrow R_{15}$  ( $e = -128 \sim +127$ )  
                      $R_{15}$  identifies the next  
                     address for the BLT  
                     instruction (2 bytes)

Description: If the sign flag and the overflow flag are different, add "e" to the program counter  $R_{15}$  and read the next instruction at the location indicated by the resulting value in the program counter.  
                     If the values are the same, do not jump.  
                     The relative offset can be -128 ~ +127 bytes from the address following the code for "e".  
                     If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode: (MSB) (LSB)  
                     0 0 0 0 0 1 1 0 (06H)  
                     ← e (00H~FFH) → Relative address  
                     BLT e

Note: The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** In the following example, the sign flag is set and the overflow flag is reset. The program jumps forward 4 bytes from the execution address of the instruction.

**BLT \$+4H**

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>
BBB	06
BBE	02
BBF	02
BC0	←PC before jump
BC1	←PC after jump
BC2	← Execute instruction at address BC0 and jump.

### 9.21 BMI e

**Operation:**      If      S Flag = 1

then  $R_{15} + e \rightarrow R_{15}$

(e= -128~+127)

$R_{15}$  identifies the next address for the BMI instruction (2 bytes)

Description: If the sign flag is “1”, add “e” to the program counter  $R_{15}$  and read the next instruction at the location indicated by the resulting value in the program counter.

If the sign flag is “0”, do not jump.

The relative offset can be -128 ~ +127 bytes from the address following the code for “e”.

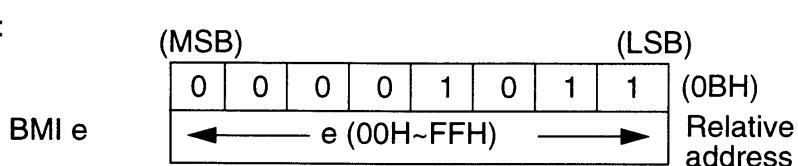
If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

### Opcode:



**Note:** The number “e” (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** In the following example, the sign flag is set and the program jumps forward 5 bytes from the execution address of the instruction.

BMI \$+5H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>
57D	
57E	0B
57F	03
580	←PC before jump
581	
582	
583	←PC after jump

Execute instruction at address 580 and jump.

## 9.22 BNE e

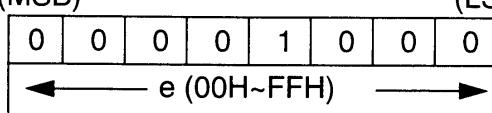
Operation: If Z Flag = 0  
 then  $R_{15} + e \rightarrow R_{15}$  ( $e = -128 \sim +127$ )  
 $R_{15}$  identifies the next address for the BNE instruction (2 bytes)

Description: If the zero flag is "0", add "e" to the program counter  $R_{15}$  and read the next instruction at the location indicated by the resulting value in the program counter.  
 If the zero flag is "1", do not jump.  
 The relative offset can be -128 ~ +127 bytes from the address following the code for "e".  
 If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode: (MSB) (LSB)  
 BNE e   
 Relative address

Note: The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

Example: In the following example, the zero flag is reset and the program jumps backward 2 bytes from the execution address of the instruction.

BNE \$-2H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
35FB		
35FC	←PC after jump	←
35FD		
35FE		
35FF	08	
3600	FC	Execute instruction at address 3600 and jump.
3601	( BNE \$-2H )	
3600	←PC before jump	
3601		

9.23 BPL e

Operation: If S Flag = 0

then  $R_{15} + e \rightarrow R_{15}$

(e= -128~+127)

R<sub>15</sub> identifies the next address for the BPL instruction (2 bytes)

Description: If the sign flag is “0”, add “e” to the program counter R<sub>15</sub> and read the next instruction at the location indicated by the resulting value in the program counter.

If the sign flag is “1”, do not jump.

The relative offset can be -128 ~ +127 bytes from the address following the code for “e”.

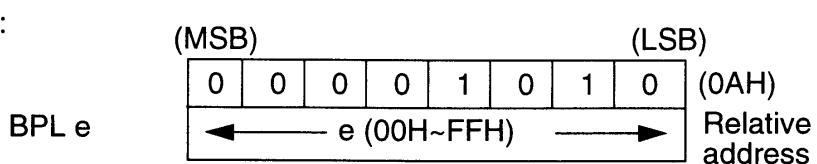
If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

## No flags affected

## Opcode:



Note: The number “e” (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** In the following example, the sign flag is reset and the program jumps forward 4 bytes from the execution address of the instruction.

BPL \$+4H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
95D		
95E	0A	
95F	02	▀ (BPL \$+4H)
960		←PC before jump
961		←PC after jump
962		← Execute instruction at address 960 and jump.
963		

## 9.24 BRA e

Operation:  $R_{15} + e \rightarrow R_{15}$  ( $e = -128 \sim +127$ )  
 $R_{15}$  identifies the next address for the BRA instruction (2 bytes)

Description: Regardless of the status of the flags, add "e" to the program counter  $R_{15}$  and read the next instruction at the location indicated by the resulting value in the program counter.  
The relative offset can -128 ~ +127 bytes from the address following the code for "e".

When a JUMP occurs, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode: (MSB) (LSB)  
BRA e      

0	0	0	0	0	1	0	1
← e (00H~FFH) →							

 (05H)  
Relative address

Note: The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

**Example:** In the following example, the program jumps backward to the execution address of the instruction.

**BRA \$0H**

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
B0FC		
B0FD		
B0FE	05	←PC after jump
B0FF	FE	←PC before jump
B100	(BRA \$0H)	Execute instruction at address B100 and jump.
B101		

**9.25 BVC e**

Operation: If O/V Flag=0

then  $R_{15}+e \rightarrow R_{15}$

( $e = -128 \sim +127$ )

$R_{15}$  identifies the next address for the BVC instruction (2 bytes)

Description: If the overflow flag is "0", add "e" to the program counter  $R_{15}$  and read the next instruction at the location indicated by the resulting value in the program counter.

If the overflow flag is "1", do not jump.

The relative offset can be -128 ~ +127 bytes from the address following the code for "e".

If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

Flags affected:

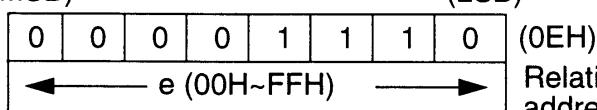
B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode:

(MSB) (LSB)

BVC e



Note:

The number "e" (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:

ROM execution time 6 cycles

RAM execution time 6 cycles

Cache RAM execution time 2 cycles

**Example:** In the following example, the overflow flag is reset and the program jumps forward 4 bytes from the execution address of the instruction.

BVC \$+4H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>
288D	
288E	0E
288F	02 □ (BVC \$+4H)
2890	←PC before jump
2891	
2892	←PC after jump
2893	

Diagram annotations:

- A bracket spans from the PC at address 2890 to the PC at address 2893, labeled "Execute instruction at address 2890 and jump."
- An arrow points from the PC at address 2891 to the PC at address 2892.

9.26 BVS e

**Operation:**      If      O/V Flag=1

then  $R_{15} + e \rightarrow R_{15}$

(e= -128~+127)

$R_{15}$  identifies the next address for the BVS instruction (2 bytes)

**Description:** If the overflow flag is “1”, add “e” to the program counter R<sub>15</sub> and read the next instruction at the location indicated by the resulting value in the program counter.

If the overflow flag is “0”, do not jump.

The relative offset can be -128 ~ +127 bytes from the address following the code for “e”.

If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

### No flags affected

## Opcode:

(MSB) (LSB)

0	0	0	0	1	1	1	1
$\longleftrightarrow e \text{ (00H~FFH)} \longrightarrow$							

(0FH)  
Relative address

Note: The number “e” (number, label, formula) which shows the jump destination is given in the assembler as an operand.

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** In the following example, the overflow flag is set and the program jumps backward 2 bytes from the execution address of the instruction.

BVS \$-2H

The relationship between the program and program counter is as follows:

<u>PC ADDRESS</u>	<u>Object Code</u>	
68B		
68C	←PC after jump	
68D		←
68E	0F [ ] (BVS \$-2H)	
68F		Execute instruction at address 690 and jump.
690	FC [ ]	
691	←PC before jump	[ ]

**9.27 CACHE**

Operation: If CACHE BASE REGISTER $\leftrightarrow$ (R<sub>15</sub> & 0FFF0H)  
 then (R<sub>15</sub> & 0FFF0H)→CACHE BASE REGISTER

Description: When the cache base register is equal to the address with the lower 4 bits of the program counter at 0, nothing occurs. When it is not equal to this address, reset all cache flags and set the cache base register to that value.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)  
 CACHE 

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

 (02H)

Machine Cycles: ROM execution time 3~4 cycles  
 RAM execution time 3~4 cycles  
 Cache RAM execution time 1 cycle

## 9.28 CMODE

Operation:  $S_{reg} (b4\sim b0) \rightarrow \text{PLOT OPTIONS REGISTER}$

Description: This instruction loads the lower 5 bits of the source register into the plot options register. The instruction is used to specify the PLOT, COLOR, and GETC execution modes.

Bit 0 - Transparency Flag

0 = Transparency ON

If transparency is on and the color register is “0”, the plot circuit only changes the X coordinate. When transparency is on and the color register is other than “0”, the normal plotting operation is performed.

1 = Transparency OFF

The normal plotting operation is performed when transparency is off.

Bit 1 - Dither Flag

Bit 1 is only valid in the 16-color mode. When Bit 1 is “1” and the values of bit 0 in registers R1 and R2 are the same, the lower 4 bits in the color register are plotted. When bit 0 of registers R1 and R2 are different, the upper 4 bits in the color register are plotted.

Note: When transparency is on and the 4 bits to be plotted are “0”, only the X coordinate is changed.

Bit 2 - Upper 4 Bits Color

Bit 2 is valid in the 16-color and 256-color modes. In the 256-color mode, Bit 3 must be set to a logic “1”.

When Bit 2 is “1”, the upper 4 bits in the source register are stored in the lower 4 bits of the color register while processing the COLOR and GETC instructions. This allows the data for two pixels to be stored in one byte.

Bit 3 - 256 Color Mode Only

Set Bit 3, “1”, in the 256-color mode to fix the upper 4 bits of the color register while processing the COLOR and GETC instructions and change the lower 4 bits only.

Bit 4 - Sprite Mode

Set Bit 4, “1”, to specify the bitmap in the sprite mode.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

CMODE	0	0	1	1	1	1	0	1	(3DH)
	0	1	0	0	1	1	1	0	(4EH)

Machine Cycles: ROM execution time 6 cycles  
 RAM execution time 6 cycles  
 Cache RAM execution time 2 cycles

Example: Under the following conditions,

Sreg: R<sub>0</sub>, R<sub>0</sub>= 0002H

the transparency and dithering modes are set when

CMODE

is executed.

## 9.29 CMP R<sub>n</sub>

Operation: S<sub>reg</sub> – R<sub>n</sub> (n=0~15)

Description: This instruction subtracts the operand R<sub>n</sub> from the source register and sets the flags accordingly. The result of the subtraction is not stored.

The source register is specified in advance using a FROM or WITH instruction. When not specified, the source register defaults to R<sub>0</sub>.

The operand can be R<sub>0</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
O/V	: Set on overflow, else reset
S	: Set when the result is negative, else reset.
CY	: Set on unsigned borrow, else reset.
Z	: Set on zero result, else reset

Opcode:

CMP R <sub>n</sub>	(MSB)								(LSB)	
	0	0	1	1	1	1	1	1	(3FH)	
	0	1	1	0	n (0H~FH)					(6nH)

Machine Cycles:

ROM execution time 6 cycles

RAM execution time 6 cycles

Cache RAM execution time 2 cycles

Example: Under the following conditions,

S<sub>reg</sub>: R<sub>1</sub>, R<sub>1</sub>= 8000H, R<sub>3</sub>= 2FFFH

the overflow and carry flags are set and sign and zero flags are reset when

CMP R<sub>3</sub>

is executed.

## 9.30 COLOR

Operation:  $S_{reg} \rightarrow$  Color register

Description: This instruction loads the lower 8 bits of the source register into the color register as the color value.

Note: The value in the color register is stored in the color matrix (8 rows x 8 columns) with the PLOT instruction. When the PLOT instruction has been executed eight times or either of registers R<sub>1</sub> or R<sub>2</sub> is changed, the data is changed automatically to character data format and stored in the game pak RAM.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

COLOR	0	1	0	0	1	1	1	0	(4EH)
-------	---	---	---	---	---	---	---	---	-------

Machine Cycles: ROM execution time 3 cycles  
 RAM execution time 3 cycles  
 Cache RAM execution time 1 cycles

Example: Under the following conditions:

$S_{reg}$ : R<sub>6</sub>, R<sub>6</sub>= 9830H

the color register becomes 30H when

COLOR

is executed.

### 9.31 DEC R<sub>n</sub>

Operation:  $R_n - 1 \rightarrow R_n$  (n=0~14)

Description: This instruction decrements the register specified in the operand  $R_n$  by 1 and stores the result back in the same register. The register used can be  $R_0-R_{14}$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set when the result is negative, else reset.
Z	: Set on zero result, else reset

Opcode: (MSB) (LSB)

DEC R <sub>n</sub>	1 1 1 0	n (0H~EH)	(EnH)
--------------------	---------	-----------	-------

Machine Cycles: ROM execution time 3 cycles  
 RAM execution time 3 cycles  
 Cache RAM execution time 1 cycles

Example: Under the following conditions:

$R_9 = A3F7H$

when the following instruction is executed

DEC R<sub>9</sub>

$R_9$  becomes A3F6H.

**9.32 DIV2**

Operation:      If  $(S_{reg}) = -1$       then  $0 \rightarrow (D_{reg})$   
                   else ASR  $(S_{reg}) \rightarrow (D_{reg})$

Description:      This instruction automatically shifts all bits in the source register right one place. The result is stored in the destination register. (Refer to ASR instruction for details.) If the source register data is FFFFH, the result stored in the destination register is 0000H.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set when the result is negative, else reset.
CY	: Set when Bit 0 of the source register is "1" and reset when "0".
Z	: Set on zero result, else reset

Opcode:

DIV2	(MSB)				(LSB)				(3DH)
	0	0	1	1	1	1	0	1	
	1	0	0	1	0	1	1	0	(96H)

Machine Cycles:      ROM execution time      6 cycles  
                   RAM execution time      6 cycles  
                   Cache RAM execution time      2 cycles

Example: Under the following conditions,

$S_{reg}$ : R<sub>7</sub>,  $D_{reg}$ : R<sub>2</sub>

CY      Bit15  
0      R<sub>7</sub>: [ 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 ] (4635H)

becomes

CY      Bit15  
1      R<sub>2</sub>: [ 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 ] (231AH)

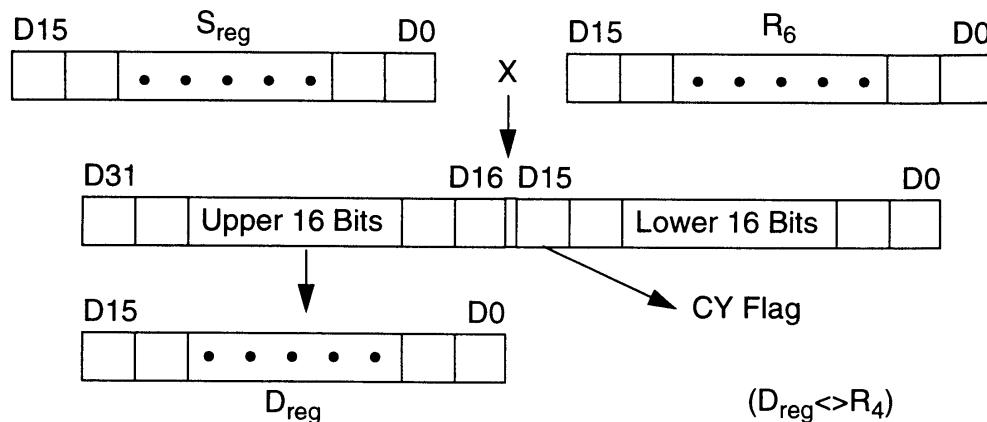
when

DIV2

is executed.

### 9.33 FMULT

Operation:



Description: This instruction performs a 16 x 16-bit signed multiplication with the source register and  $R_6$ . The upper 16 bits of the 32-bit result are stored in the destination register. Bit 15 of the 32-bit result becomes the carry flag.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, these registers default to  $R_0$ .

Note: Any register,  $R_0 \sim R_{15}$ , except  $R_4$  may be assigned as the destination register.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- S : Set when the result is negative, else reset.
- CY : Set when Bit 15 of the result is "1" and reset when "0".
- Z : Set if the upper 16 bits of result are zero, else reset.

Opcode: (MSB) (LSB)

FMULT	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	0	0	1	1	1	1	1	(9FH)
1	0	0	1	1	1	1	1			

Machine Cycles:	ROM execution time	11 or 7 cycles
	RAM execution time	11 or 7 cycles
	Cache RAM execution time	8 or 4 cycles

Note: The number of machine cycles depends on the CFGR register.

Example: Under the following conditions,

$S_{reg}$ : R<sub>5</sub>, D<sub>reg</sub>: R<sub>2</sub>, R<sub>5</sub>= 4AAAH, R<sub>6</sub>= DAABH

R<sub>2</sub> becomes F51CH and the carry flag and sign flag are set when

FMULT

is executed.

## 9.34 FROM R<sub>n</sub>

### REGISTER PREFIX INSTRUCTION

Operation: If B =0      then set S<sub>reg</sub> to R<sub>n</sub>      (n=0~15)  
                   else R<sub>n</sub> → D<sub>reg</sub>

Description: This instruction specifies which of the registers, R<sub>0</sub>~R<sub>15</sub>, is to be used as the source register. If the B flag is set, the contents of the specified operand R<sub>n</sub> are stored in the destination register D<sub>reg</sub>, which is specified using the WITH instruction. (Refer to the MOVES instruction.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

No flags affected

Opcode: (MSB) (LSB)  
         FROM R<sub>n</sub>

1	0	1	1	n (0H~FH)	(BnH)
---	---	---	---	-----------	-------

Machine Cycles: ROM execution time      3 cycles  
                   RAM execution time      3 cycles  
                   Cache RAM execution time      1 cycles

Example: Execute

FROM R<sub>2</sub>

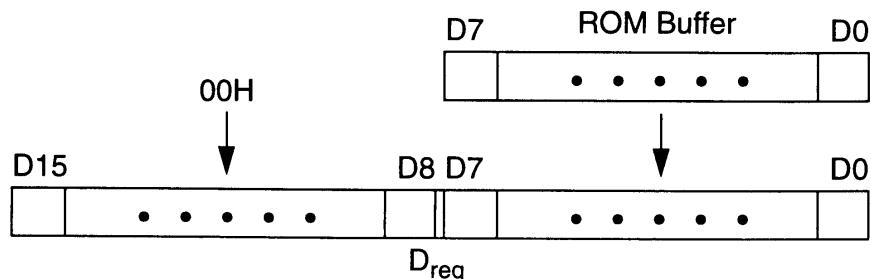
to set R<sub>2</sub> as the source register.

To perform R<sub>2</sub>+ R<sub>3</sub>= R<sub>0</sub>, write:

FROM R<sub>2</sub> ;Sets the source register to R<sub>2</sub>  
                   ADD R<sub>3</sub> ;Executes R<sub>2</sub> + R<sub>3</sub>→R<sub>0</sub>

## 9.35 GETB

Operation:



**Description:** This instruction loads one byte of data stored in the ROM buffer into the lower 8 bits of the destination register and resets the upper 8 bits of the destination register. Register R<sub>14</sub> is the ROM address pointer when data is loaded from the game pak ROM into the ROM buffer. Using the value stored at R<sub>14</sub> for the game pak ROM address, data is read from game pak ROM to the ROM buffer.

Banks are specified in advance using the ROMB instruction. However, changing banks using the ROMB instruction does not in itself trigger a ROM load.

The destination register is specified in advance using a WITH or TO instruction. When not specified, this register defaults to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

GETB	1	1	1	0	1	1	1	1	(EFH)
------	---	---	---	---	---	---	---	---	-------

Machine Cycles:

ROM execution time	3~8 cycles
RAM execution time	3~9 cycles
Cache RAM execution time	1~6 cycles

Note: Because the ROM buffer is used, the number of execution cycles varies with each program.

Example: Under the following conditions,

ROM buffer=0075H, D<sub>reg</sub>:R<sub>0</sub>

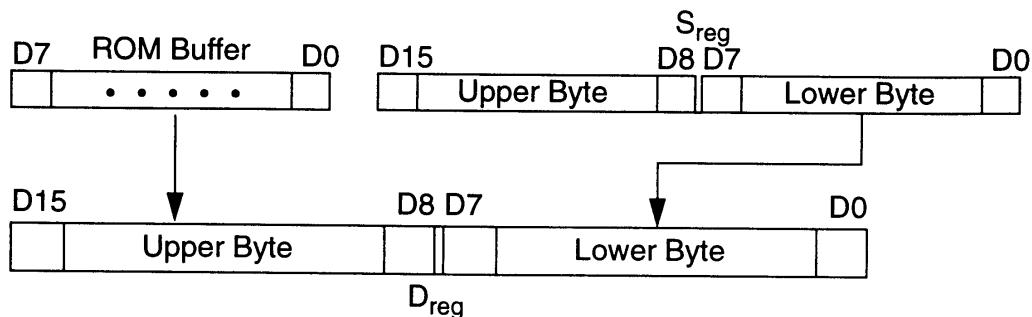
R<sub>0</sub> becomes 0075H when

**GETB**

is executed.

## 9.36 GETBH

Operation:



Description: This instruction loads the data contained in the ROM buffer to the high byte of the destination register and the low byte of the source register to the low byte of the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

Note: Refer to the GETB instruction and "Memory Mapping" for information to load data from game pak ROM to the ROM buffer.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

GETBH	0	0	1	1	1	1	0	1	(3DH)
	1	1	1	0	1	1	1	1	(EFH)

Machine Cycles: ROM execution time 6~10 cycles  
 RAM execution time 6~9 cycles  
 Cache RAM execution time 2~6 cycles

Note: Because the ROM buffer is used, the number of execution cycles varies with each program.

**Example:** Under the following conditions,

(ROM buffer) = 75H, S<sub>reg</sub>: R<sub>2</sub>, D<sub>reg</sub>: R<sub>6</sub>, R<sub>2</sub>= 4ABDH

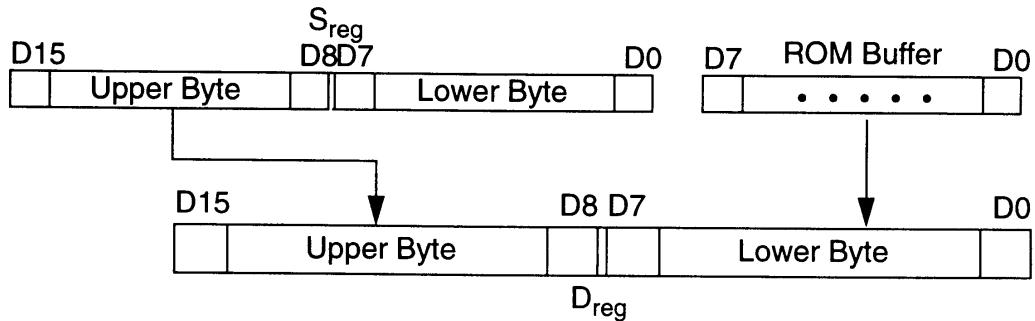
R<sub>6</sub> becomes 75BDH when

**GETBH**

is executed.

### 9.37 GETBL

Operation:



Description: This instruction loads the data contained in the ROM buffer to the low byte of the destination register and the high byte of the source register to the high byte of the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

Note: Refer to the GETB instruction and "Memory Mapping" for information to load data from game pak ROM to the ROM buffer.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

GETBL	0 0 1 1 1 1 1 0	(3EH)
	1 1 1 0 1 1 1 1	(EFH)

Machine Cycles: ROM execution time 6~10 cycles  
 RAM execution time 6~9 cycles  
 Cache RAM execution time 2~6 cycles

Note: Because the ROM buffer is used, the number of execution cycles varies with each program.

Example: Under the following conditions,

(ROM buffer) = 75H, S<sub>reg</sub>: R<sub>2</sub>, D<sub>reg</sub>: R<sub>6</sub>, R<sub>2</sub>= 4ABDH

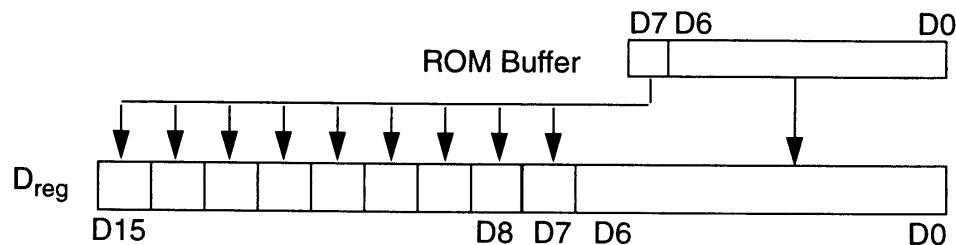
R<sub>6</sub> is 4A75H when

**GETBL**

is executed.

## 9.38 GETBS

Operation:



**Description:** This instruction loads the data contained in the ROM buffer to the low byte of the destination register and the data contained in Bit 7 of the ROM buffer to Bits 8~15 of the destination register.

The destination register is specified in advance using a WITH or TO instruction. When not specified, this register defaults to R<sub>0</sub>.

**Note:** Refer to the GETB instruction and "Memory Mapping" for information to load data from game pak ROM to the ROM buffer.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

**Opcode:** (MSB) (LSB)

GETBS	0	0	1	1	1	1	1	(3FH)
	1	1	1	0	1	1	1	(EFH)

**Machine Cycles:** ROM execution time 6~10 cycles  
RAM execution time 6~9 cycles  
Cache RAM execution time 2~6 cycles

**Note:** Because the ROM buffer is used, the number execution cycles varies with each program.

Example: Under the following conditions,

(ROM buffer) = 85H, D<sub>reg</sub>: R<sub>8</sub>

R<sub>8</sub> becomes FF85H when

GETBS

is executed.

## 9.39 GETC

**Operation:** (ROM buffer) → (COLOR register)

**Description:** This instruction loads the data contained in the ROM buffer into the color register as color data.

**Note:** Refer to the GETB instruction and "Memory Mapping" for information to load data from game pak ROM to the ROM buffer. Refer to COLOR and "Bitmap Emulation" for information concerning the color register and how to plot.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

**Opcode:** (MSB) (LSB)  
GETC      

1	1	0	1	1	1	1	1
---	---	---	---	---	---	---	---

 (DFH)

**Machine Cycles:** ROM execution time 3~10 cycles  
RAM execution time 3~9 cycles  
Cache RAM execution time 1~6 cycles

**Note:** Because the ROM buffer is used, the number of execution cycles varies with each program.

**Example:** Under the following conditions,

(ROM buffer) = 4BH

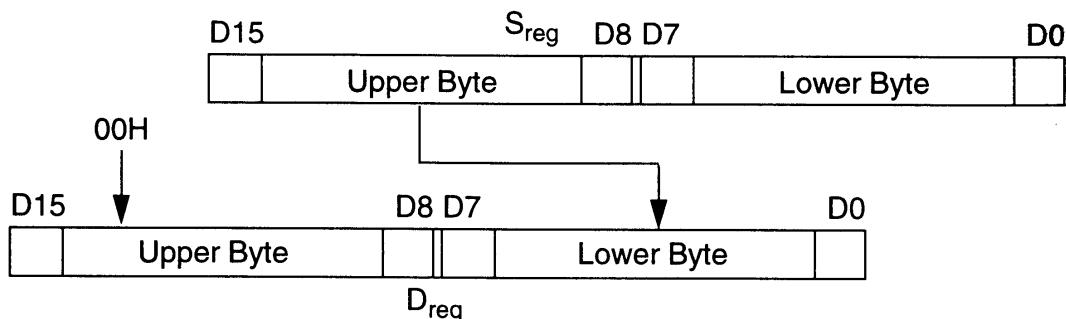
4BH is loaded to the color register when

GETC

is executed.

## 9.40 HIB

Operation:



Description: This instruction loads the high byte of the source register into the low byte of the destination register. The high byte of the destination register is loaded with 00H.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, these registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- S : Set if a negative number is loaded to the low byte of the destination register, else reset.
- Z : Set if zero is loaded to low byte of the destination register, else reset.

Opcode:

HIB	(MSB)	(LSB)
	1 1 0 0 0 0 0 0	(C0H)

Machine Cycles:

ROM execution time	3 cycles
RAM execution time	3 cycles
Cache RAM execution time	1 cycles

**Example:** Under the following conditions,

$S_{reg}$ : R<sub>11</sub>, D<sub>reg</sub>=R<sub>1</sub>, R<sub>11</sub>=8A43H

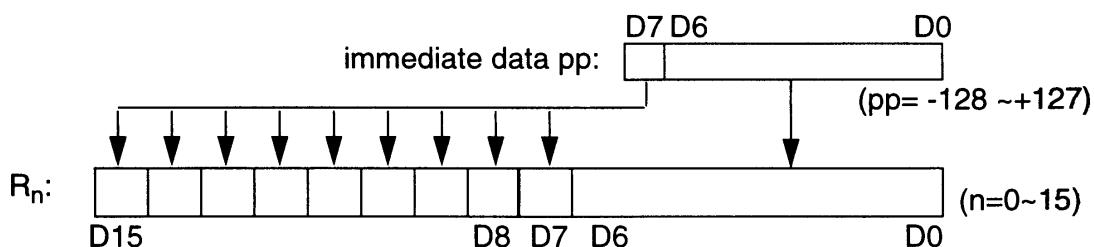
R<sub>1</sub> becomes 008AH and the sign flag is set when

HIB

is executed.

## 9.41 IBT R<sub>n</sub>, #pp

Operation:

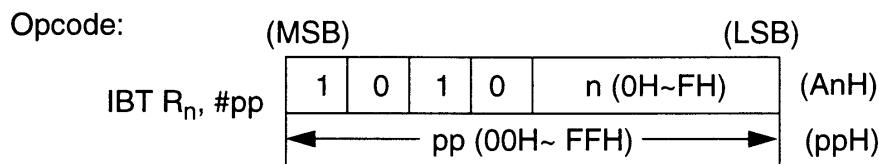


Description: This instruction loads one byte of immediate data (hexadecimal) into the low byte of register R<sub>n</sub>. Bit 7 of the immediate data is loaded into bits 8 through 15 of R<sub>n</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset



Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: Since hexadecimal numbers are handled in the assembler as integers, without signs, a hexadecimal number of 80H or greater that is entered as an operand is processed as a number greater than +128, exceeding the range -128~+127. When this occurs, the assembler will specify the low byte as the immediate data of the IBT instruction.

IBT	R <sub>8</sub> ,	#4	...	0004H → R <sub>8</sub>
IBT	R <sub>8</sub> ,	#-128	...	FF80H → R <sub>8</sub>
IBT	R <sub>8</sub> ,	#0A4H	...	FFA4H → R <sub>8</sub>

## 9.42 INC R<sub>n</sub>

Operation:  $R_n + 1 \rightarrow R_n$  ( $n = 0 \sim 14$ )

Description: This instruction increments the contents of the register specified in the operand  $R_n$  by one and stores the result back into the same register.

The operand can be  $R_0 \sim R_{14}$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set if result is negative, else reset.
Z	: Set on zero result, else reset.

Opcode: (MSB) (LSB)  
INC R<sub>n</sub>

1	0	1	1	n (0H~EH)	(DnH)
---	---	---	---	-----------	-------

Machine Cycles: ROM execution time 3 cycles  
RAM execution time 3 cycles  
Cache RAM execution time 1 cycles

Example: When register R<sub>12</sub> is 65B1H, R<sub>12</sub> becomes 65B2H when  
INC R<sub>12</sub>  
is executed.

### 9.43 IWT R<sub>n</sub>, #xx

Operation: #xx (2-byte hexadecimal immediate data) → R<sub>n</sub>  
 (n = 0~15, #xx=0~65535)

Description: This instruction loads two bytes of immediate data, #xx (hexadecimal), to the register specified in the operand, R<sub>n</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

ITW R <sub>n</sub> , #xx	1	1	1	1	n (0H~FH)	(FnH)
					x (00H~FFH)	(Lower Byte)
					x (00H~FFH)	(Upper Byte)

The two-byte immediate data in the op code is loaded low byte first, followed by the high byte.

Machine Cycles: ROM execution time 9 cycles  
 RAM execution time 9 cycles  
 Cache RAM execution time 3 cycles

Example: Register R<sub>0</sub> becomes 4583H when  
 IWT R<sub>0</sub>, #4583H  
 is executed.

## 9.44 JMP R<sub>n</sub>

Operation:  $R_n \rightarrow R_{15}$  (PC) (n=8~13)

Description: This instruction loads the contents of the register specified in the operand R<sub>n</sub> to R<sub>15</sub> (program counter) and initiates a program fetch from the resulting location specified by the program counter.

The next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

The operand can be register R<sub>8</sub>~R<sub>13</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

Opcode: JMP R <sub>n</sub>	B	: Reset						
	ALT1	: Reset						
	ALT2	: Reset						
	(MSB)	(LSB)						
	<table border="1"> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>n (8H~DH)</td> <td>(9nH)</td> </tr> </table>	1	0	0	1	n (8H~DH)	(9nH)	
1	0	0	1	n (8H~DH)	(9nH)			

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

Example: When register R<sub>10</sub> is 0555H and the following program is executed,

PC	Opcode
0444H	JMP R <sub>10</sub>
0445H	INC R <sub>10</sub>

. .  
. .  
. .

the jump destination is 0555H.

**9.45 LDB (R<sub>m</sub>)**

Operation:  $(R_m) \rightarrow D_{reg}$  (Low Byte)  $(m=0\sim11)$

$00H \rightarrow D_{reg}$  (High Byte)

Description: This instruction loads one byte of data located at the game pak RAM address contained in the register specified in the operand R<sub>m</sub> and stores this data in the destination register. The upper byte of the destination register is loaded with 00H.

Use the RAMB instruction to set the RAM bank. (Refer to RAMB.)

The destination register is specified in advance using a WITH or TO instruction. When not specified, this register defaults to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

0	0	1	1	1	1	0	1
0	1	0	0	m (0H~BH)			

(3DH)
(4mH)

Machine Cycles: ROM execution time 11 cycles  
RAM execution time 13 cycles  
Cache RAM execution time 6 cycles

Note: The GSU waits while the data is loaded from game pak RAM. The cycles required for this are included in the execution times given above.

**Example:** Under the following conditions,

$D_{reg} = R_7$ ,  $R_1 = 3482H$ ,  $(70:3482H) = 51H$   
RAMBR:70H

and when the following program is executed,

LDB (R<sub>1</sub>)

R<sub>7</sub> becomes 0051H.

## 9.46 LDW ( $R_m$ )

Operation:  $(R_m) \rightarrow D_{reg}$  (Low Byte)  $(m=0\sim11)$   
 $(R_{m\pm1}) \rightarrow D_{reg}$  (High Byte) When the contents of  $R_m$  is:  
 even,  $(R_{m+1})$   
 odd,  $(R_{m-1})$   
 is loaded to the high byte.

Description: The word data located in the game pak RAM address that equals the contents of register  $R_m$  are stored in the destination register. The game pak RAM address bank is specified using the RAMB instruction (refer to RAMB).

The destination register is specified in advance using a WITH or TO instruction. When not specified, this register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)  
 LDW ( $R_m$ ) 

0	1	0	0	$m (0H\sim BH)$
---	---	---	---	-----------------

 (4mH)

Machine Cycles: ROM execution time 10 cycles  
 RAM execution time 12 cycles  
 Cache RAM execution time 7 cycles

Note: While a load is performed from the game pak ROM, the GSU is in the WAIT state. This execution time is included in the above machine cycles.

Example: Under the following conditions,

$D_{reg}:R_5$ ,  $R_3=6480H$ ,  $(70:6480H)=C0H$ ,  $RAMBR=70H$

and when the following program is executed,

LDW  $(R_3)$

the register  $R_5$  becomes C02EH.

## 9.47 LEA R<sub>n</sub>, xx (Refer to IWT R<sub>n</sub>, #xx)

Operation:  $R_n \leftarrow xx$  (n=0~15, xx=0~65535)

Description: This instruction loads two bytes of immediate data, #xx (hexadecimal), to the register specified in the operand R<sub>n</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

LEA R <sub>n</sub> , xx	1	1	1	1	n (0H~FH)	(FnH)
	x (00H~FFH)					(Lower Byte)
	x (00H~FFH)					(Upper Byte)

The two-byte immediate data in the op code is loaded low byte first, followed by the high byte.

Machine Cycles: ROM execution time 9 cycles  
 RAM execution time 9 cycles  
 Cache RAM execution time 3 cycles

Example: Register R<sub>3</sub> becomes 4853H when

LEA R<sub>3</sub>, #4853H

is executed.

**9.48 LINK #n**

Operation:  $R_{15} + \#n \rightarrow R_{11}$  (n=1~4)  
 $R_{15}$  contains address  
following LINK instruction

Description: This instruction adds the operand  $\#n$  to the value contained in register  $R_{15}$  (program counter) and stores the result in register  $R_{11}$ . Operand  $\#n$  can be a number from 1~4. This instruction can be used to specify a return address in register  $R_{11}$  when jumping to a subroutine.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

LINK #n	1	0	0	1	n (1H~4H)	(9nH)
---------	---	---	---	---	-----------	-------

Machine Cycles: ROM execution time 3 cycles  
RAM execution time 3 cycles  
Cache RAM execution time 1 cycles

Example: Under the following conditions,

$R_{15}$ : 4368H

and when the following program is executed,

4368	LINK #4
4369	IWT $R_{15}$ , #74FFH
436C	NOP
436B	IBT $R_1$ , #12H

register  $R_{11}$  becomes  $4369H + 2 = 436BH$

## 9.49 LJMP R<sub>n</sub>

Operation:  $R_n \rightarrow R_{15}$  (PC)  $(n=8\sim13)$

$S_{reg} \rightarrow$  Program Bank Register (PBR)

Description: This instruction loads the register specified as operand,  $R_n$ , into the program counter,  $R_{15}$  and loads the lower byte of the source register to the program bank register. This allows the program to jump to addresses in different banks.

The next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not included in the machine cycles listed below.)

The operand can be any of registers  $R_8\sim R_{13}$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)  
LJMP R<sub>n</sub>

0	0	1	1	1	1	0	1
1	0	0	1	n (8H~DH)			

 (3DH) (9nH)

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: Under the following conditions,

R<sub>1</sub>:0001H

the program jumps from 00:8006H to 01:0002H when the following program is executed.

Bank	Address	Syntax
00	:8000H	IWT R <sub>10</sub> , #0002H
00	:8003H	FROM R <sub>1</sub>
00	:8004H	LJMP R <sub>10</sub>
00	:8006H	NOP

### 9.50 LM R<sub>n</sub>, (xx)

**Description:** This instruction loads the data contained in the game pak RAM address specified in the second operand  $xx$  and stores the data in the register specified in the first operand  $R_n$ . The RAMB instruction is used to specify the bank of the RAM address.

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B	: Reset
ALT1	: Reset
ALT2	: Reset

Opcode:	(MSB)	(LSB)																																
LM R <sub>n</sub> , (xx)	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>1</td><td>1</td><td colspan="4">n (0H~FH)</td></tr> <tr> <td colspan="8" style="text-align: center;">x (00H~FFH)</td></tr> <tr> <td colspan="8" style="text-align: center;">x (00H~FFH)</td></tr> </table>	0	0	1	1	1	1	0	1	1	1	1	1	n (0H~FH)				x (00H~FFH)								x (00H~FFH)								(3DH) (FnH) (ADRS Lower Byte) (ADRS Upper Byte)
0	0	1	1	1	1	0	1																											
1	1	1	1	n (0H~FH)																														
x (00H~FFH)																																		
x (00H~FFH)																																		

Machine Cycles:	ROM execution time	20 cycles
	RAM execution time	21 cycles
	Cache RAM execution time	11 cycles

Note: While a load is performed from the game pak RAM, the GSU is in the WAIT state. This execution time is included in the above machine cycles.

**Example:** Under the following conditions.

(70:BACCH) = 28H, (70:BACDH) = 96H, RAMBR=70H

register R<sub>9</sub> becomes 9628H when the following program is executed:

LM R<sub>0</sub>, (OBACCH)

## 9.51 LMS R<sub>n</sub>, (yy)

Operation: RAM (yy) → R<sub>n</sub> (low byte) (n=0~15, yy=0~510\*)  
RAM (yy+1) → R<sub>n</sub> (high byte)

\*Note: Selectable RAM address (yy) must be an even number.

Description: This instruction uses a short address method to perform the LM instruction. The address is shortened by reducing the number of bytes in the instruction opcode. The instruction loads data from the game pak RAM address equal to the immediate number yy and stores the data in register R<sub>n</sub>. The selectable game pak RAM address may be an even number of 0~510. The RAMB instruction is used to specify the bank of the RAM address.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode:

LMS R <sub>n</sub> , (yy)	(MSB)				(LSB)			
	0	0	1	1	1	1	0	1
	1	0	1	0	n (0H~FH)			
kk (00H~FFH)							(Address)	

[Short address method]

This method is used by LMS, SMS, and other instructions to reduce the number of bytes in the instruction opcode. Only one byte is used. The actual game pak RAM address is twice that of the address code. The relationship between yy in the above syntax and kk in the opcode is:

$$yy = kk \times 2$$

Machine Cycles: ROM execution time 17 cycles  
RAM execution time 17 cycles  
Cache RAM execution time 10 cycles

Note: The GSU waits while data is loaded from game pak RAM. The execution time required for this is included in the machine cycles given above.

*DESCRIPTION OF INSTRUCTIONS*

Example: Under the following conditions,

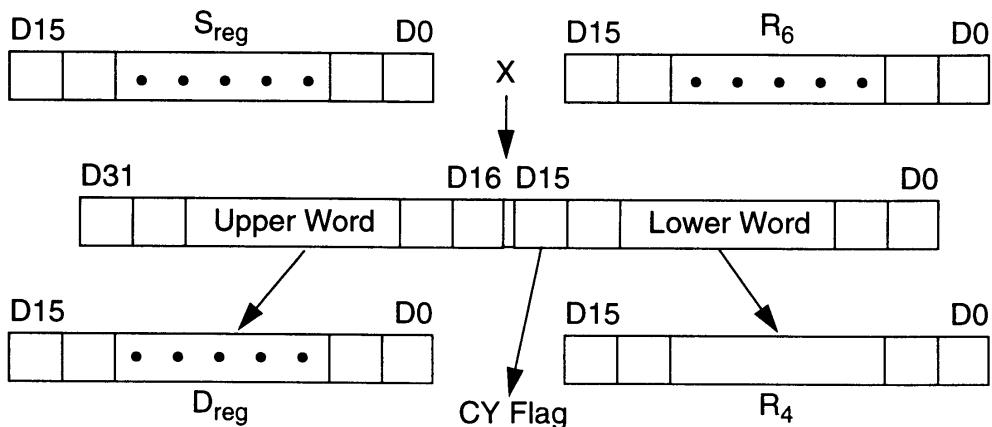
(70:1AAH) = 32H, (70:1ABH) = 92H, RAMBR:70H

register R<sub>3</sub> becomes 9232H when the following program is executed:

Syntax	Opcode
LMS R <sub>3</sub> , (1AAH)	3D A3 D5

## 9.52 LMULT

Operation:



**Description:** This instruction performs 16 x 16-bit signed multiplication using the source register and register R<sub>6</sub>. The upper 16 bits of the result are stored in the destination register, and the lower 16 bits are stored in R<sub>4</sub>. If Bit 15 of R<sub>6</sub> is set, the carry flag is also set to "1".

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>. If R<sub>4</sub> is specified as the destination register, the result will be invalid.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- S : Set if the result is negative, else reset
- CY : Set if Bit 15 of R<sub>6</sub> is "1", reset if "0"
- Z : Set if the destination register result is zero, else reset.

Opcode:

LMULT	(MSB)								(LSB)	
	0	0	1	1	1	1	0	1	(3DH)	
	1	0	0	1	1	1	1	1	(9FH)	

**DESCRIPTION OF INSTRUCTIONS**

Machine Cycles:	ROM execution time	10 or 14 cycles
	RAM execution time	10 or 14 cycles
	Cache RAM execution time	5 or 9 cycles

Note: The number of cycles varies depending upon the CFGR register setting.

Example: Under the following conditions,

$S_{reg}$ : R<sub>9</sub>, D<sub>reg</sub>: R<sub>8</sub>  
R<sub>9</sub>= B556H, R<sub>6</sub>= DAABH

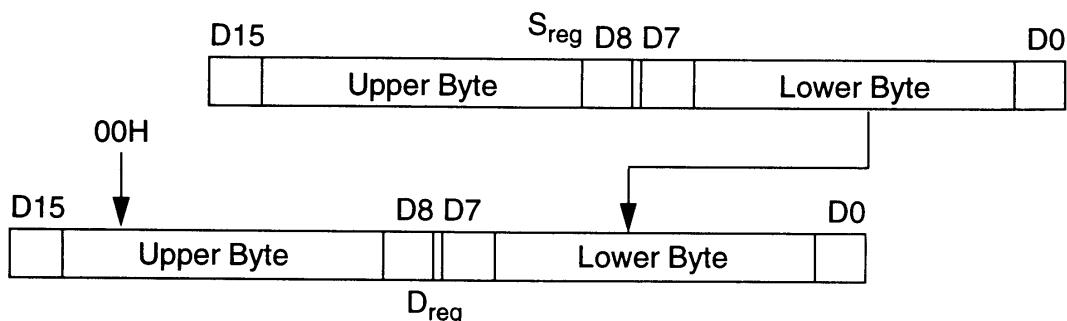
the register R<sub>8</sub> becomes 0AE3H and R<sub>4</sub> 5C72H when

LMULT

is executed.

## 9.53 LOB

Operation:



Description: This instruction loads the lower byte of the source register to the low byte of the destination register. The high byte of the destination register is loaded with 00H.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- S : Set if the low byte of the source register is negative, else reset.
- Z : Set if low byte of the source register is zero, else reset.

Opcode: (MSB) (LSB)

LOB	1   0   0   1   1   1   1   0	(9EH)
-----	-------------------------------	-------

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

**Example:** Under the following conditions,

$S_{reg}$ : R<sub>10</sub>, D<sub>reg</sub>: R<sub>12</sub>, R<sub>10</sub>= FB23H

the register R<sub>12</sub> becomes 0023H when

LOB

is executed.

## 9.54 LOOP

Operation:  $R_{12} - 1 \rightarrow R_{12}$

If Z Flag=0 then  $R_{13} \rightarrow R_{15}$  (PC)

Description: This instruction decrements  $R_{12}$  by 1. If the result does not set the zero flag, the contents of  $R_{13}$  are loaded into  $R_{15}$  and the program is fetched from the resulting location specified by the program counter.

If the zero flag is set, the program counter is incremented and the next instruction is executed.

The instruction at the address following the LOOP instruction is already loaded into the pipeline. The branch is taken after this instruction is executed.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set if the register $R_{12}$ is negative, else reset.
Z	: Set if the register $R_{12}$ is zero, else reset.

Opcode: (MSB) (LSB)  
 LOOP 

0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

 (3CH)

Machine Cycles: ROM execution time 3 cycles  
 RAM execution time 3 cycles  
 Cache RAM execution time 1 cycles

Example: In the following program,

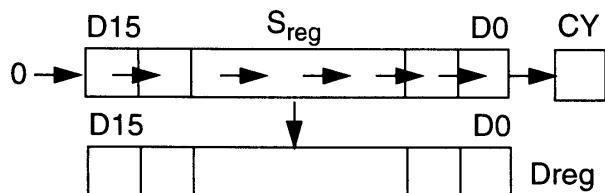
```

00:8014  INC R7
00:8015  INC R6
00:8016  LOOP
00:8017  NOP
00:8018  ADD R4
  
```

if  $R_{13}$  is 8014H and  $R_{12}$  is other than 0001H, the program jumps to 00:8014H after the NOP instruction is executed. If  $R_{12}$  is 0001H, the jump does not happen and the instruction ADD is executed.

9.55 LSR

## Operation:



**Description:** This instruction shifts all bits in the source register one bit to the right and stores the result in the destination register. Bit 15 becomes “0” and the value of Bit 0 is stored in the carry flag.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to  $R_0$ .

### Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	0	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Reset
CY	: Set if Bit 0 in source register is “1”, else reset
Z	: Set on zero result, else reset.

Opcode: (MSB) (LSB)

LSR 0 0 0 0 0 0 1 1 (03H)

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache BAM execution time 1 cycles

**Example:** Under the following conditions

$S_{req}: R_8, D_{req}: R_0$

bit 15

hit0

R<sub>8</sub>: 

1	0	1	1	0	1	0	1	0	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

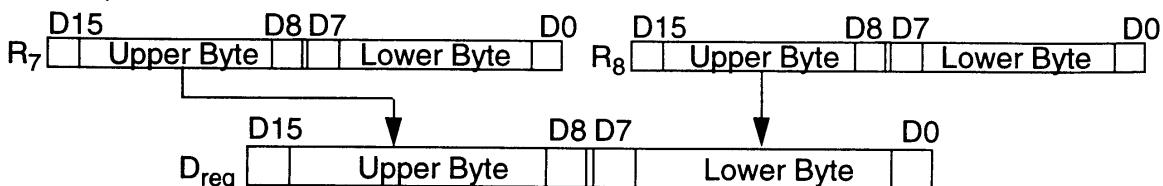
 (B53FH)

## LSR execution results in:

**R<sub>0</sub>:** bit15 bit0 CY  
0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 (5A9FH) 1

## 9.56 MERGE

Operation:



Description: This instruction stores the high byte of  $R_7$  in the high byte of the destination register and the high byte of  $R_8$  in the low byte of the destination register.

The destination register is specified in advance using a WITH or TO instruction. When not specified, the register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- O/V : Set if the result of (B6 or B7 or B14 or B15) is "1", and reset if "0".
- S : Set if the result of (B7 or B15) is "1", and reset if "0".
- CY : Set if the result of (B5 or B6 or B7 or B13 or B14 or B15) is "1", reset if "0".
- Z : Set if the result of (B4 or B5 or B6 or B7 or B12 or B13 or B14 or B15) is "1", reset if "0"

Opcode: (MSB) (LSB)

MERGE	0   1   1   1   0   0   0   0	(70H)
-------	-------------------------------	-------

Machine Cycles:

ROM execution time	3 cycles
RAM execution time	3 cycles
Cache RAM execution time	1 cycles

*DESCRIPTION OF INSTRUCTIONS*

**Example:** Under the following conditions:

$D_{reg}; R_9, R_7=05AAH, R_8=FC33H$

$R_9$  becomes 05FCH and the sign, over flow, carry and zero flags are set when

**MERGE**

is executed.

## 9.57 MOVE R<sub>n</sub>, R<sub>n'</sub>

Operation: R<sub>n'</sub> → R<sub>n</sub> (n, n' = 0~15)

Description: This instruction loads the contents of register R<sub>n'</sub>, specified in the second operand, to register R<sub>n</sub>, specified in the first operand.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

MOVE R <sub>n</sub> , R <sub>n'</sub>	0	0	1	0	n' (0H~FH)	(2n'H)
	0	0	0	1	n (0H~FH)	(1nH)

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: Under the following conditions,

$$R_{14} = 4983H, R_8 = 9264H$$

the register R<sub>8</sub> becomes 4983H when

MOVE R<sub>8</sub>, R<sub>14</sub>

is executed.

## 9.58 MOVE R<sub>n</sub>, #xx

## **MACRO INSTRUCTION**

Conditions: IF (-128≤xx≤127): (if unsigned, (0≤xx≤127) or  
then, use an IBT instruction (65408≤xx≤65535))  
else, use an IWT instruction.

**Description:** This instruction directly loads hexadecimal immediate data into register  $R_n$ , specified in the first operand. This is a macro instruction and is stored in memory as “IWT  $R_n$ , #xx” or “IBT  $R_n$ , #pp.” The assembler automatically recognizes whether this should be replaced with an IBT instruction or IWT instruction, depending upon the value of immediate data.

If immediate data is -128 ~ 127 (unsigned, 0~127 or 65408~65535), it is replaced with an IBT instruction. Otherwise, it is replaced with an IWT instruction. Refer to “IBT R<sub>n</sub>, #pp” or “IWT R<sub>n</sub>, #xx” for machine cycles, flags affected, and opcode.

### Example:

MOVE R <sub>8</sub> , #070H	;0070H→R <sub>8</sub>	(IBT R <sub>8</sub> , #070H)
MOVE R <sub>8</sub> , #0A4H;	00A4H→R <sub>8</sub>	(IWT R <sub>8</sub> , #0A4H)
MOVE R <sub>8</sub> , #-128;	FF80→R <sub>8</sub>	(IBT R <sub>8</sub> , #-128)

## 9.59 MOVE R<sub>n</sub>, (xx)

### MACRO INSTRUCTION

Operation:  $(xx) \rightarrow R_n$  (low byte)  $(n=0\sim15, xx=0\simFFFFH)$   
 $(xx\pm1) \rightarrow R_n$  (high byte)

Note: When the value xx is even, the contents of (xx+1) are loaded to the high byte of R<sub>n</sub>. When the value of xx is odd, the contents of (xx-1) are loaded to the high byte of R<sub>n</sub>.

Conditions: If  $0000H \leq xx \leq 01FFH$  and xx is even:  
then, use an LMS instruction  
else, use an LM instruction.

Description: This instruction loads hexadecimal data contained in the game pak RAM address specified in the second operand and stores the data in register R<sub>n</sub>, specified in the first operand.. The RAMB instruction is used to specify the bank of the game pak RAM address (refer to RAMB).

This is a macro instruction and is stored in memory as "LM R<sub>n</sub>, (xx)" or "LMS R<sub>n</sub>, (yy)." The assembler automatically recognizes whether it should be replaced with an LM instruction or an LMS instruction, depending upon the value of the game pak RAM address specified.

When the game pak RAM address is an even number of 0~1FFH, it is replaced with an LMS instruction. Otherwise, it is replaced with an LM instruction. Refer to "LM R<sub>n</sub>, (xx)" or "LMS R<sub>n</sub>, (yy)" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

$(70:BACCH) = 28H, (70:BACDH) = 96H, RAMBR=70H$

the register R<sub>9</sub> becomes 9628H when the following program is executed:

```
MOVE R9, (0BACCH) ;(70:BACCH)→R9(Low Byte) (LM R9, (0BACCH))
                  ;(70:BACDH)→R9(High Byte)
```

Also, under the following conditions,

(71:01AAH) = 32H, (71:01ABH) = 92H, RAMBR=71H

the register R<sub>3</sub> becomes 9232H when the following program is executed:

MOVE R<sub>3</sub>, (1AAH) ;(71:01AAH)→R<sub>3</sub>(Low Byte) (LMS R<sub>3</sub>, (01AAH))  
;(71:01ABH)→R<sub>3</sub>(High Byte)

## 9.60 MOVE (xx), R<sub>n</sub>

## **MACRO INSTRUCTION**

Note: If the value of  $xx$  is even, store the high byte of  $R_n$  at  $(xx+1)$ . If the value of  $xx$  is odd, store the high byte of  $R_n$  at  $(xx-1)$ .

**Conditions:** If  $(0000H \leq xx \leq 01FFH)$  and  $xx$  is even:  
then, use an SMS instruction,  
else, use an SM instruction.

**Description:** This instruction stores the contents (hexadecimal data) of register  $R_n$  specified in the second operand in the game pak RAM address specified in the first operand. The RAMB instruction is used to specify the bank of the game pak RAM address (refer to “RAMB”).

This macro instruction is stored in memory as "SM (xx), R<sub>n</sub>" or "SMS (yy), R<sub>n</sub>." The assembler automatically recognizes whether it should be replaced with an SM instruction or an SMS instruction, depending upon the value of the game pak RAM address specified.

When the game pak RAM address is an even number of 0~1FFH, it is replaced with an SMS instruction. Otherwise, it is replaced with an SM instruction. Refer to "SM (xx), R<sub>n</sub>" and "SMS (yy), R<sub>n</sub>" for machine cycles, flags affected, and opcode.

**Example:** Under the following conditions,

R<sub>9</sub>: BACDH, and RAMBR=71H

when the following program is executed,

MOVE (9CDEH), R<sub>9</sub> ;R<sub>9</sub> (Low Byte)→(71:9CDEH) (SM (9CDEH), R<sub>9</sub>)  
;R<sub>9</sub> (High Byte)→(71:9CDFH)

the result is (71:9CDEH)=CDH, (71:9CDFH)=BAH

Also, under the following conditions,

R<sub>2</sub>: 3248H, and RAMBR=70H

when the following program is executed,

MOVE (136H), R<sub>2</sub> ;R<sub>2</sub> (Low Byte)→(70:0136H) (SMS (136H), R<sub>2</sub>)  
;R<sub>2</sub> (High Byte)→(70:0137H)

the result is (70:0136H)=48H, (70:0137H)=32H

## 9.61 MOVEB R<sub>n</sub>, (R<sub>n'</sub>)

### MACRO INSTRUCTION

Operation:  $(R_{n'}) \rightarrow R_n$  (Low Byte)  $(n=0\sim15, n'=0\sim11)$   
 $00H \rightarrow R_n$  (High Byte)

Conditions: If  $n=0$ :  
then, use only LDB instruction,  
else, use TO instruction and LDB instruction.

Description: This instruction loads one byte of data located at the game pak RAM address equal to the contents of register R<sub>n'</sub>, specified by the second operand and stores this data in the register specified in the first operand. The high byte of the destination register is loaded with 00H. The register identified in the second operand is selectable from R<sub>0</sub>~R<sub>11</sub>. The RAMB instruction is used to specify the game pak RAM bank (refer to "RAMB").

This macro instruction is stored in memory as "LDB (R<sub>m</sub>)" or "TO R<sub>n</sub>" + "LDB (R<sub>m</sub>).". The assembler automatically recognizes whether or not the TO instruction is required. When n does not equal 0, the TO instruction is added. Refer to "LDB (R<sub>m</sub>)" and "TO R<sub>n</sub>" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

$R_1=3482H, (70:3482H)=51H, RAMBR=70H$

when the following program is executed,

```
MOVEB R7, (R1) ;(R1)→R7 (Low Byte) (TO R7+LDB (R1))
;00H→R7 (High Byte)
```

register R<sub>7</sub> becomes 0051H.

Also, under the following conditions,

$R_3=3581H, (70:3581H)=9AH, RAMBR=70H$

when the following program is executed,

```
MOVEB R0, (R3) ;(R3)→R0 (Low Byte) (LDB (R3))
;00H→R0 (High Byte)
```

register R<sub>0</sub> becomes 009AH.

### 9.62 MOVEB (R<sub>n</sub>'), R<sub>n</sub>

## **MACRO INSTRUCTION**

Operation:  $R_n$  (low byte)  $\rightarrow (R_{n'})$  ( $n=1\sim 15, n'=0\sim 11$ )

Conditions: If  $n=0$ :  
then, use only STB instruction,  
else, use FROM instruction and STB instruction.

**Description:** This instruction stores the contents of the low byte of register  $R_n$  specified in the second operand at the game pak RAM address equal to the contents of register  $R_{n'}$ , specified in the first operand. The register identified in the first operand is selectable from  $R_0 \sim R_{11}$ . The RAMB instruction is used to specify the game pak RAM bank (refer to “RAMB”).

This macro instruction is stored in memory as “STB (R<sub>m</sub>)” or “FROM R<sub>n</sub>” + “STB (R<sub>m</sub>).” The assembler automatically recognizes whether or not the FROM instruction is required. When n does not equal 0, the FROM instruction is added. Refer to “STB (R<sub>m</sub>) and “FROM R<sub>n</sub>” for machine cycles, flags affected, and op-code.

**Example:** Under the following conditions,

R<sub>5</sub>=3843H, R<sub>11</sub>=94F1H, RAMBR=71H

when the following program is executed,

MOVEB (R<sub>11</sub>), R<sub>5</sub> ;R<sub>5</sub> (Low Byte)→(R<sub>11</sub>) (FROM R<sub>5</sub>+STB (R<sub>11</sub>))

the result is (71:94F1H)=43H.

Also, under the following conditions,

R<sub>0</sub>=89E0H, R<sub>3</sub>=438BH, RAMBR=70H

when the following program is executed,

**MOVEB** ( $R_3$ ),  $R_0$  ; $R_0$  (Low Byte)  $\rightarrow$  ( $R_3$ ) (STB ( $R_3$ ))

the result is (70:438BH)=43H.

## 9.63 MOVES R<sub>n</sub>, R<sub>n'</sub>

Operation: R<sub>n'</sub> → R<sub>n</sub> (n, n' = 0~15)

Description: This instruction loads the contents of register R<sub>n'</sub>, specified in the second operand, to register R<sub>n</sub>, specified in the first operand. Flags are set according to the data loaded.  
(Refer to MOVE R<sub>n</sub>, R<sub>n'</sub>.)

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
O/V	: Set if Bit 7 is "1", else reset
S	: Set if Bit 15 is "1", else reset
Z	: Set when data is zero, else reset

Opcode:

MOVES R <sub>n</sub> , R <sub>n'</sub>	(MSB)					(LSB)	
	0	0	1	0	n' (0H~FH)	(2n' H)	
	1	0	1	1	n (0H~FH)	(BnH)	

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: When R<sub>7</sub> is 4983H and

MOVES R<sub>10</sub>, R<sub>7</sub>

is executed, the register R<sub>10</sub> becomes 4983H and the overflow flag is set.

## 9.64 MOVEW R<sub>n</sub>, (R<sub>n'</sub>)

### MACRO INSTRUCTION

Operation:  $(R_{n'}) \rightarrow R_n$  (Low Byte)  $(n=0\sim15, n'=0\sim11)$   
 $(R_{n'}+1) \rightarrow R_n$  (High Byte)

Note: If the contents of R<sub>n'</sub> are even, store the address equal to the contents of (R<sub>n'</sub>+1) in the high byte of R<sub>n</sub>. If the contents of R<sub>n'</sub> are odd, store the address equal to (R<sub>n'</sub>-1) in the high byte of R<sub>n</sub>.

Conditions: If n=0:  
then, use only LDW instruction,  
else, use TO instruction and LDW instruction.

Description: This instruction loads hexadecimal data from the game pak RAM address equal to the contents of register R<sub>n'</sub> specified in the second operand and stores it into register R<sub>n</sub> specified by the first operand. The game pak RAM address bank is specified using the RAMB instruction (refer to RAMB).

This macro instruction is stored in memory as "LDW (R<sub>m</sub>)" or "TO R<sub>n</sub>" + "LDW (R<sub>m</sub>).". The assembler automatically recognizes whether or not the TO instruction is required. When n is not equal to 0, the TO instruction is added. Refer to "LDW (R<sub>m</sub>)" and "TO R<sub>n</sub>" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

R<sub>3</sub>=6480H, (71:6480H)=2EH, (71:6481H)=C0H,  
RAMBR=71H

and when the following program is executed,

MOVEW R<sub>5</sub>, (R<sub>3</sub>) ;(R<sub>3</sub>) $\rightarrow$ R<sub>5</sub>(Low Byte) (TO R<sub>5</sub> + LDB (R<sub>3</sub>))  
;(R<sub>3</sub>+1) $\rightarrow$ R<sub>5</sub>(High Byte)

register R<sub>5</sub> becomes C02EH.

Also, under the following conditions,

R<sub>6</sub>=0822H, (70:0822H)=43H, (70:0823H)=96H,  
RAMBR=70H

and when the following program is executed,

MOVEW R<sub>0</sub>, (R<sub>6</sub>) ;(R<sub>6</sub>) $\rightarrow$ R<sub>0</sub>(Low Byte) (LDB (R<sub>6</sub>))  
;(R<sub>6</sub>+1) $\rightarrow$ R<sub>0</sub>(High Byte)

register R<sub>0</sub> becomes 9643H.

## 9.65 MOVEW ( $R_n'$ ), $R_n$

### MACRO INSTRUCTION

Operation:  $R_n$  (low byte)  $\rightarrow (R_{n'})$   $(n=0\sim 15, n'=0\sim 11)$   
 $R_n$  (high byte)  $\rightarrow (R_{n'} \pm 1)$

Note: If the contents of  $R_{n'}$  are even, store the high byte of  $R_n$  into the address equal to the contents of  $(R_{n'}+1)$ . If the contents of  $R_{n'}$  are odd, store the high byte of  $R_n$  into the address equal to the contents of  $(R_{n'}-1)$ .

Conditions: If  $n=0$ :  
then, use only STW instruction,  
else, use FROM instruction and STW instruction.

Description: This instruction stores the contents (hexadecimal data) of register  $R_n$  specified in the second operand into the game pak RAM address which is equal to the value of register  $R_{n'}$  specified in the first operand. The game pak RAM address bank is specified using the RAMB instruction (refer to RAMB). The operand  $n'$  can be a register from  $R_0\sim R_{11}$ .

This macro instruction is stored in memory as "STW ( $R_m$ )" or "FROM  $R_n$ " + "STW ( $R_m$ ).". The assembler automatically recognizes whether or not the FROM instruction is required. When  $n$  is not equal to 0, the FROM instruction is added. Refer to "STW ( $R_m$ )" and "FROM  $R_n$ " for machine cycles, flags affected, and op-code.

Example: Under the following conditions,

$R_9=BFA3H, R_{10}=4444H, RAMBR=71H$

and when the following program is executed,

MOVEW ( $R_{10}$ ),  $R_9$  ; $R_9$ (Low Byte) $\rightarrow(R_{10})$  (FROM  $R_9+STW (R_{10})$ )  
; $R_9$ (High Byte) $\rightarrow(R_{10}+1)$

the result is  $(71:4444H)=A3H, (71:4445H)=BFH$ .

Also, under the following conditions,

$R_0=3151H$ ,  $R_6=92A0H$ ,  $RAMBR=71H$

and when the following program is executed,

MOVEW  $(R_6)$ ,  $R_0$  ; $R_0$  (Low Byte)  $\rightarrow (R_6)$  (STW ( $R_6$ ))  
; $R_0$  (High Byte)  $\rightarrow (R_6+1)$

the result is  $(71:92A0H)=51H$ ,  $(71:92A1H)=31H$ .

## 9.66 MULT R<sub>n</sub>

**Operation:** S<sub>reg</sub> (low byte) \* R<sub>n</sub> (low byte) → D<sub>reg</sub> (n=0~15)

**Description:** This instruction performs 8 x 8-bit signed multiplication using the low byte of the source register and the low byte of register R<sub>n</sub>. The result is stored in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

The operand can be a register R<sub>0</sub>~R<sub>15</sub>.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

**Opcode:**

MULT R <sub>n</sub>	(MSB)	(LSB)					
	<table border="1"> <tr> <td>1</td> <td>0</td> <td>0</td> <td>0</td> <td>n (0H~FH)</td> </tr> </table>	1	0	0	0	n (0H~FH)	(8nH)
1	0	0	0	n (0H~FH)			

**Machine Cycles:**

ROM execution time	3 or 5 cycles
RAM execution time	3 or 5 cycles
Cache RAM execution time	1 or 2 cycles

**Note:** The number of cycles depends upon the CFGR register.

**Example:** Under the following conditions,

S<sub>reg</sub>: R<sub>5</sub>, D<sub>reg</sub>: R<sub>2</sub>  
 R<sub>5</sub>= 52CFH, R<sub>1</sub>= 63CFH

the register R<sub>2</sub> becomes 0961H when

MULT R<sub>1</sub>

is executed.

**9.67 MULT #n**

Operation:  $S_{reg}$  (low byte) \* #n  $\rightarrow D_{reg}$  (n=0~15)

Description: This instruction performs 8 x 8-bit signed multiplication using the low byte of the source register and the immediate data specified in the operand #n. The result is stored in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

The operand can be immediate data from 0~15.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

Opcode: (MSB) (LSB)

MULT #n	<table border="1"> <tr> <td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td><td>0</td><td colspan="4">n (0H~FH)</td></tr> </table>	0	0	1	1	1	1	1	0	1	0	0	0	n (0H~FH)				(3EH)	(8nH)
0	0	1	1	1	1	1	0												
1	0	0	0	n (0H~FH)															

Machine Cycles: ROM execution time 6 or 8 cycles  
 RAM execution time 6 or 8 cycles  
 Cache RAM execution time 2 or 3 cycles

Note: The number of cycles depends upon the CFGR register.

Example: Under the following conditions,

$S_{reg}$ : R<sub>3</sub>,  $D_{reg}$ : R<sub>4</sub>, R<sub>3</sub>=95C6H

the register R<sub>4</sub> becomes FDF6H when

MULT #9

is executed.

## 9.68 NOP

Operation:  $PC \leftarrow PC+1$

Description: This instruction causes the processor to idle for one cycle and increment the program counter by one.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode:

(MSB)	(LSB)								
NOP	<table border="1"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td> </tr> </table> (01H)	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1		

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

**9.69 NOT**

Operation:  $\overline{\text{Sreg}} \rightarrow \text{Dreg}$

**Description:** This instruction calculates the 1's complement of the source register and stores the result in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

Opcode:	(MSB)	(LSB)
NOT	0 1 0 0 1 1 1 1	(4FH)

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

**Example:** Under the following conditions.

$S_{\text{req}}: R_9, D_{\text{req}}: R_{13}$

R<sub>9</sub>: Bit15 Bit0 (B764H)

1	0	1	1	0	1	1	1	0	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## **the execution of**

NOT

results in:

R<sub>13</sub>: Bit15 Bit0  
0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 1 (489BH)

## 9.70 OR R<sub>n</sub>

Operation: S<sub>reg</sub> OR R<sub>n</sub> → D<sub>reg</sub> (n=1~15)

Description: This instruction performs logical bit-wise OR on corresponding bits of the source register and the register specified in the operand R<sub>n</sub>. The result is stored in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

The operand can be a register R<sub>1</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

Opcode:

	(MSB)	(LSB)					
OR R <sub>n</sub>	<table border="1"> <tr> <td>1</td> <td>1</td> <td>0</td> <td>0</td> <td>n (1H~FH)</td> </tr> </table>	1	1	0	0	n (1H~FH)	(CnH)
1	1	0	0	n (1H~FH)			

Machine Cycles:

ROM execution time	3 cycles
RAM execution time	3 cycles
Cache RAM execution time	1 cycles

**Example:** Under the following conditions,

$S_{reg}$ :  $R_4$ ,  $D_{reg}$ :  $R_5$

Bit15 Bit0  
R4: 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0 0 (6368H)

R<sub>2</sub>: Bit15 Bit0  
0 0 0 1 0 1 1 0 1 0 0 0 1 1 0 0 (168CH)

the register  $R_5$  becomes:

Bit15 Bit0  
R<sub>5</sub>: 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 0 (77ECH)

when

OR R<sub>2</sub>

is executed.

**9.71 OR #n**

Operation: Sreq OR #n → Dreq (n=1~15)

**Description:** This instruction performs logical bit-wise OR on corresponding bits of the source register and the immediate data specified in the operand #n. The result is stored in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set when the result is negative, else reset.
Z	: Set on zero result, else reset.

Opcode: (MSB) (LSB)

OR #n	0	0	1	1	1	1	1	0	(3EH)
	1	1	0	0	n (1H~FH)				(CnH)

Machine Cycles:	ROM execution time	6 cycles
	RAM execution time	6 cycles
	Cache RAM execution time	2 cycles

**Example:** Under the following conditions.

$S_{req}: R_7, D_{req}: R_5$

Bit15

Bit0

R7: 

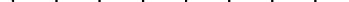
0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (5FA2H)

the register  $R_5$  becomes:

**Bit15**

Bit0

R<sub>5</sub>:  (5FA7H)

when

QB #5H

is executed

## 9.72 PLOT

**Description:** This instruction plots the color code specified by the COLOR or GETC instruction to locations X and Y specified by R<sub>1</sub> and R<sub>2</sub>. After plotting, R<sub>1</sub> will be incremented.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

**Opcode:** (MSB) (LSB)

PLOT	0	1	0	0	1	1	0	0	(4CH)
------	---	---	---	---	---	---	---	---	-------

**Machine Cycles:** ROM execution time 3~48 cycles  
 RAM execution time 3~51 cycles  
 Cache RAM execution time 1~48 cycles

**Note:** Because this instruction uses the RAM buffer, the number of machine cycles varies depending upon the program.

## 9.73 RAMB

Operation:  $S_{reg} \rightarrow \text{RAMBR}$

Description: This instruction moves the low byte of the source register into the game pak RAM bank register in order to specify the game pak RAM bank when transferring data between game pak RAM and multi-purpose registers. Note that the SCBR is used with the RAMBR to specify the bank for plotting. The game pak RAM bank register can only be changed with the RAMB instruction. The initial value of this register is invalid.

The source register is specified in advance using a FROM or WITH instruction. When not specified, the register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: RAMB	(MSB)	(LSB)															
	<table border="1"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> </table>	0	0	1	1	1	1	1	0	1	1	0	1	1	1	1	1
0	0	1	1	1	1	1	0										
1	1	0	1	1	1	1	1										
		(DFH)															

Machine Cycles:

ROM execution time	6 cycles
RAM execution time	6 cycles
Cache RAM execution time	2 cycles

Example: Under the following conditions,

$S_{reg}: R_3, R_3 = 0170H$

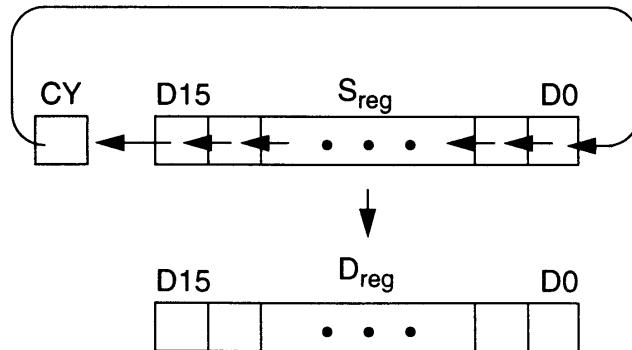
the RAM bank register becomes 70H when

RAMB

is executed.

## 9.74 ROL

Operation:



Description: This instruction shifts all bits in the source register one bit to the left. Bit 15 is shifted to the carry flag and the carry flag is shifted to Bit 0. The result is stored in the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set if result is negative, else reset.
CY	: Set if Bit 15 in source register is "1", else reset.
Z	: Set on zero result, else reset.

Opcode: (MSB) (LSB)

ROL	0 0 0 0 0 1 0 0	(04H)
-----	-----------------	-------

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

Example: Under the following conditions,

$S_{reg}$ : R<sub>8</sub>,  $D_{reg}$ : R<sub>4</sub>

CY              bit15  
1              R<sub>8</sub>: 

0	0	0	1	1	1	0	1	0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (1D4BH)

executing ROL results in:

CY              bit15  
0              R<sub>4</sub>: 

0	0	1	1	1	0	1	0	1	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (3A97H)

## 9.75 ROMB

Operation:  $S_{reg} \rightarrow ROMBR$

Description: This instruction moves the low byte of the source register into the game pak ROM bank register in order to specify the game pak ROM bank when loading data from game pak ROM. The game pak ROM bank register can only be changed with the ROMB instruction, but the contents can not be read. The initial value of this register is invalid.

The source register is specified in advance using a FROM or WITH instruction. When not specified, the source register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

ROMB	0	0	1	1	1	1	1	1	(3FH)
	1	1	0	1	1	1	1	1	(DFH)

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: Under the following conditions,

$S_{reg}: R_5, R_5 = 0046H$

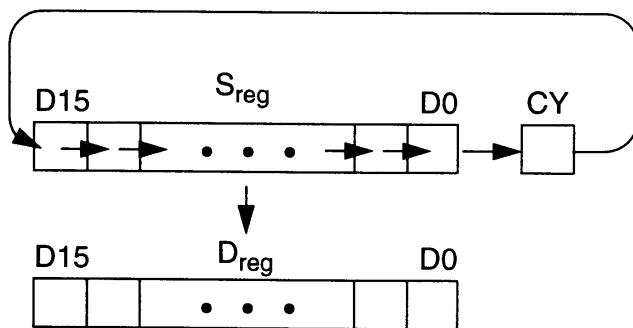
the ROMBR becomes 46H when

ROMB

is executed.

## 9.76 ROR

Operation:



**Description:** This instruction shifts all bits in the source register one bit to the right. Bit 0 is shifted to the carry flag and the carry flag is shifted to Bit 15. The result is stored in the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	*	*

- B : Reset
- ALT1 : Reset
- ALT2 : Reset
- S : Set if result is negative, else reset.
- CY : Set if Bit 0 in source register is "1", else reset.
- Z : Set on zero result, else reset.

Opcode:

	(MSB)				(LSB)											
ROR	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>				1	0	0	1	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td><td>1</td><td>1</td></tr></table> (97H)				0	1	1	1
1	0	0	1													
0	1	1	1													

Machine Cycles:

ROM execution time

3 cycles

RAM execution time

3 cycles

Cache RAM execution time

1 cycles

Example: Under the following conditions,

Sreg: R<sub>10</sub>, Dreg: R<sub>12</sub>

CY              bit15  
1    R<sub>10</sub>: 

0	0	0	1	1	1	0	1	0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (1D4BH)

executing ROR results in:

CY              bit15  
1    R<sub>12</sub>: 

1	0	0	0	1	1	1	0	1	0	1	0	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (8EA5H)

## 9.77 RPIX

**Operation:** PIXEL COLOR from game pak RAM → D<sub>reg</sub>

**Description:** This instruction loads the color data stored in game pak RAM and stores it in the destination register. Because data in game pak RAM is in the PPU format, it is first read to the color matrix and subsequently stored in the destination register. The data is then read from game pak RAM to the color matrix.

**Flags affected:**

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

**Opcode:** (MSB) (LSB)

RPIX	0	0	1	1	1	1	0	1	(3DH)
	0	1	0	0	1	1	0	0	(4CH)

**Machine Cycles:** ROM execution time 24~80 cycles

RAM execution time 24~78 cycles

Cache RAM execution time 20~74 cycles

**9.78 SBC R<sub>n</sub>**

Operation:  $S_{reg} - R_n - \overline{CY\ Flag} \rightarrow D_{reg}$  (n=0~15)

Description: This instruction subtracts the contents of the register specified in the operand and the carry flag from the source register and stores the result in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
O/V	: Set on signed overflow, else reset
S	: Set when the result is negative, else reset.
CY	: Set on unsigned overflow, else reset
Z	: Set on zero result, else reset

Opcode:

SCB R <sub>n</sub>	(MSB)								(LSB)	
	0	0	1	1	1	1	0	1	(3DH)	
	0	1	1	0	n (0H~FH)				(6nH)	

Machine Cycles: ROM execution time 6 cycles

RAM execution time 6 cycles

Cache RAM execution time 2 cycles

Example: Under the following conditions:

$S_{reg}: R_4, D_{reg}: R_6, R_4=5682H, R_5=3609H, CY\ Flag=1$

register R<sub>6</sub> becomes 2079H and the carry flag is reset when

SBC R<sub>5</sub>

is executed.

## 9.79 SBK

Operation:  $S_{reg} \rightarrow$  (Last game pak RAM address used)

Description: The game pak RAM address accessed when data is transferred between game pak RAM and a multi-purpose register, for example the LD and ST instructions, is buffered internally. When data is to be stored to the last accessed game pak RAM address, this buffer is used so that the address does not have to be specified again in the op code. This is called "bulk processing".

This instruction uses bulk processing to store the word data contained in the source register to RAM.

The source register is specified in advance using a WITH or FROM instruction. When not specified, the register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset

Opcode: (MSB) (LSB)

SBK	1 0 0 1 0 0 0 0	(90H)
-----	-----------------	-------

Machine Cycles: ROM execution time 3~8 cycles  
 RAM execution time 7~11 cycles  
 Cache RAM execution time 1~6 cycles

Example: Under the following conditions,

(70:3230H)=51H, (70:3231H)=49H, RAMBR=70H

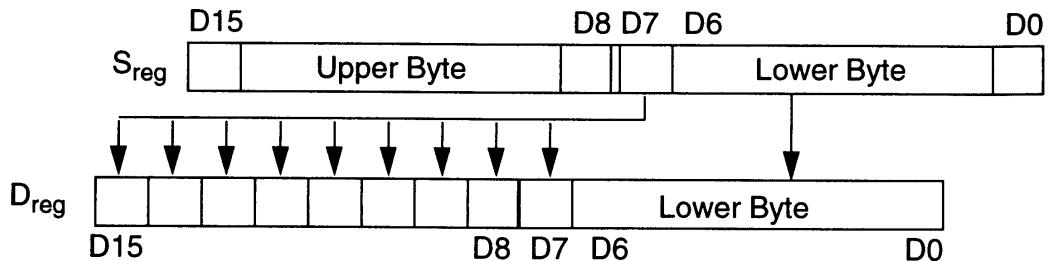
executing,

LM	$R_1$ , (3230H)
INC	$R_1$
SBK	

will result in  $R_1=4952H$ , (70:3230H)=52H, and (70:3231H)=49H.

## 9.80 SEX

Operation:



Description: This instruction performs signed expansion of the low byte of the source register, converts it to word data and stores it in the destination register.

This means that Bit 7 of the source register is stored in Bits 8 ~ 15 of the destination register. The low byte is loaded directly from the source register to the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
S	: Set if the result is negative, else reset.
Z	: Set on zero result, else reset.

Opcode: (MSB) (LSB)

SEX	1   0   0   1   0   1   0   1	(95H)
-----	-------------------------------	-------

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

**Example:** Under the following conditions,

$S_{reg}: R_5, D_{reg}: R_1, R_5 = 9284H$

the register  $R_1$  becomes  $FF84H$  when

**SEX**

is executed.

### 9.81 SM (xx), R<sub>n</sub>

Operation:	$R_n$ (low byte) $\rightarrow$ (xx)      ( $n=0\sim 15$ , $xx=0\sim 65535$ )	
	$R_n$ (high byte) $\rightarrow$ (xx+1)	When the contents of $R_n$ are even, the high byte is stored at address ( $R_n+1$ ); When the contents of $R_n$ are odd, the high byte is stored at address ( $R_n-1$ ).

**Description:** This instruction stores the contents of register R<sub>n</sub>, specified in the second operand, to the game pak RAM address which equals the value of (xx), the first operand. The RAM bank must be specified with the RAMB instruction. (Refer to RAMB.)

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B	: Reset
ALT1	: Reset
ALT2	: Reset

Opcode:	(MSB)	(LSB)																																
	<table border="1"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td><td>1</td><td>n (0H~FH)</td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td>x (00H~FFH)</td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td>x (00H~FFH)</td><td></td><td></td><td></td></tr> </table>	0	0	1	1	1	1	1	0	1	1	1	1	n (0H~FH)								x (00H~FFH)								x (00H~FFH)				(3EH) (FnH) (ADRS Lower Byte) (ADRS Upper Byte)
0	0	1	1	1	1	1	0																											
1	1	1	1	n (0H~FH)																														
				x (00H~FFH)																														
				x (00H~FFH)																														
SM (xx), R <sub>n</sub>																																		

Machine Cycles:	ROM execution time	12~17 cycles
	RAM execution time	16~20 cycles
	Cache RAM execution time	4~9 cycles

Note: Because this instruction uses the RAM buffer, the number of cycles varies depending upon the program.

**Example:** Under the following conditions,

**R<sub>4</sub>**=438CH and RAMBR=70H

the following program execution,

SM (0B492H), R<sub>4</sub>

will result in (70:B492H) =8CH, (70:B493H) = 43H.

## 9.82 SMS (yy), R<sub>n</sub>

Operation: R<sub>n</sub> (low byte) → (yy) (n=0~15, yy=0~510\*)  
R<sub>n</sub> (high byte) → (yy+1)

\*Note: Selectable RAM address (yy) must be an even number.

Description: Similar to SM, this instruction loads word data from register R<sub>n</sub>, specified in the second operand, and stores it in the game pak RAM address equal to the value specified in the first operand, yy. The selectable address is an even number 0~510. The bank is specified with the RAMB instruction. This instruction uses the short address method to reduce the number of bytes in the instruction code.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

SMS (yy), R <sub>n</sub>	(MSB)								(LSB)	
	0	0	1	1	1	1	1	0	(3EH)	
	1	1	1	1	n (0H~FH)				(AnH)	
	kk (00H~FFH)									(Address)

[Short address method]

This method is used by LMS, SMS and other instructions to reduce the number of bytes in the instruction code. One byte is used for the address. The selectable address may be an even number 0~510. The relationship between yy in the syntax and kk in the opcode is:

$$yy = kk \times 2$$

Machine Cycles:	ROM execution time	9~14 cycles
	RAM execution time	13~17 cycles
	Cache RAM execution time	3~8 cycles

Note: Because this instruction uses the RAM buffer, the number of machine cycles varies depending upon the program.

**DESCRIPTION OF INSTRUCTIONS**

**Example:** Under the following conditions,

Register R<sub>11</sub>= ABCDH, RAMBR=71H

the following program is execution,

Syntax	Opcode
SMS (194H), R <sub>11</sub>	3E AB CA

will result in (71:0194H) = CDH, (71:0195H) = ABH. The relationship between syntax and opcode is as shown above.

### 9.83 STB (R<sub>m</sub>)

Operation: S<sub>reg</sub> (low byte) → (R<sub>m</sub>) (m=0~11)

Description: This instruction stores the low byte of the source register in the game pak RAM address equal to the value in the register specified in the operand. The operand can be a register R<sub>0</sub>~R<sub>11</sub>. The game pak RAM bank must be specified with the RAMB instruction.

The source register is specified in advance using a WITH or FROM instruction. When not specified, the register defaults to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

0	0	1	1	1	1	0	1
0	0	1	1	m (0H~BH)			

(3DH)
(3mH)

STB (R<sub>m</sub>)

Machine Cycles: ROM execution time 6~9 cycles  
RAM execution time 8~14 cycles  
Cache RAM execution time 2~5 cycles

Note: Because this instruction uses the RAM buffer, the number of machine cycles varies depending upon the program.

Example: Under the following conditions,

S<sub>reg</sub>:R<sub>5</sub>, R<sub>5</sub>=216CH, R<sub>8</sub>=9A34H, RAMBR=70H

and when the following program is executed,

STB (R<sub>8</sub>)

the result is (70:9A34H)=6CH.

## 9.84 STOP

Operation:  $0 \rightarrow \text{Go flag}$

Description: This instruction resets the GSU GO flag and stops the processor. When this instruction is executed and the GSU stops, the Super NES IRQ signal is initiated.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)

STOP	0	0	0	0	0	0	0	(00H)
------	---	---	---	---	---	---	---	-------

Machine Cycles:	ROM execution time	3 cycles
	RAM execution time	3 cycles
	Cache RAM execution time	1 cycles

## 9.85 STW ( $R_m$ )

Operation:  $S_{reg}$  (low byte)  $\rightarrow (R_m)$   $(m=0\sim11)$   
 $S_{reg}$  (high byte)  $\rightarrow (R_m + 1)$

When the contents of  $R_m$  are even, the high byte is stored at address  $(R_m+1)$ ; When the contents of  $R_m$  are odd, the high byte is stored at address  $(R_m-1)$ .

Description: This instruction stores the contents of the source register into the game pak RAM address specified in the operand,  $R_m$ . The RAM bank must be specified with the RAMB instruction. The operand can be a register from  $R_0\sim R_{11}$ .  
The source register is specified in advance using WITH or FROM. When not specified, the register defaults to  $R_0$ .

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset  
ALT1 : Reset  
ALT2 : Reset

Opcode: (MSB) (LSB)  
STW ( $R_m$ ) 

0	0	1	1	m (0H~BH)
---	---	---	---	-----------

 (3mH)

Machine Cycles: ROM execution time 3~8 cycles  
RAM execution time 7~11 cycles  
Cache RAM execution time 1~6 cycles

Note: Because this instruction uses the RAM buffer, the number of cycles varies depending upon the program.

Example: Under the following conditions,

$S_{reg}:R_{10}$ ,  $R_{10}=9326H$ ,  $R_2:5872H$ ,  $RAMBR=70H$   
and when the following program is executed,

STW  $(R_2)$

the result is  $(70:5872H)=26H$ ,  $(70:5873H)=93H$ .

**9.86 SUB R<sub>n</sub>**

Operation:  $S_{reg} - R_n \rightarrow D_{reg}$  (n=0~15)

Description: This instruction subtracts the contents of the register specified in the operand from the source register and stores the result in the destination register.

Source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

The operand can be any of registers R<sub>0</sub>~R<sub>15</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
O/V	: Set on signed overflow, else reset.
S	: Set if the result is negative, else reset
CY	: Set on unsigned overflow, else reset (Set on adder overflow.)
Z	: Set if result is zero.

Opcode: (MSB) (LSB)

SUB R <sub>n</sub>	<table border="1"> <tr> <td>0</td><td>1</td><td>1</td><td>0</td><td>n (0H~FH)</td></tr> </table>	0	1	1	0	n (0H~FH)	(6nH)
0	1	1	0	n (0H~FH)			

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

Example: Under the following conditions:

$S_{reg}: R_5, D_{reg}: R_4, R_5=735AH, R_8=426BH$

the register R<sub>4</sub> becomes 30EFH when

SUB R<sub>8</sub>

is executed.

## 9.87 SUB #n

Operation:  $S_{reg} - \#n \rightarrow D_{reg}$  (n=0~15)

Description: This instruction subtracts the immediate data specified in the operand from the contents of the source register and stores the result in the destination register.

The source and destination registers are specified in advance using a WITH, FROM, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

The operand can be immediate data from 0-15.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	*	*	*	*

B	: Reset
ALT1	: Reset
ALT2	: Reset
O/V	: Set on signed overflow, else reset.
S	: Set if the result is negative, else reset
CY	: Set on unsigned borrow, else reset
Z	: Set if result is zero.

Opcode: (MSB) (LSB)

SUB #n	0	0	1	1	1	1	0	(3EH)
	0	1	1	0	n (0H~FH)			(6nH)

Machine Cycles: ROM execution time 6 cycles  
RAM execution time 6 cycles  
Cache RAM execution time 2 cycles

Example: Under the following conditions:

$S_{reg}$ : R<sub>0</sub>, D<sub>reg</sub>: R<sub>0</sub>, R<sub>0</sub>=329BH

the register R<sub>0</sub> becomes 3291H when

SUB #10

is executed.

## 9.88 SWAP

Operation:  $S_{reg}$  (low byte)  $\rightarrow D_{reg}$  (high byte)

$S_{reg}$  (high byte)  $\rightarrow D_{reg}$  (low byte)

Description: This instruction swaps the low byte and high byte of the source register and stores the result in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

Opcode: (MSB) (LSB)

SWAP	0   1   0   0   1   1   0   1	(4DH)
------	-------------------------------	-------

Machine Cycles: ROM execution time 3 cycles

RAM execution time 3 cycles

Cache RAM execution time 1 cycles

Example: Under the following conditions:

$S_{reg}$ : R<sub>3</sub>, D<sub>reg</sub>: R<sub>13</sub>, R<sub>3</sub>=48D0H

the register R<sub>13</sub> becomes D048H when

SWAP

is executed.

**9.89 TOR<sub>n</sub>**

## **REGISTER PREFIX INSTRUCTION**

**Description:** This instruction specifies register  $R_n$  as the destination register. The destination register can be any of registers  $R_0 \sim R_{15}$ .

If the B flag has been set (i.e., if a WITH instruction was executed immediately prior to this instruction) the contents of the source register are loaded to  $R_n$  (refer to MOVE  $R_n, R_n'$ ).

## Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
-	-	-	-	-	-	-

## No flags affected

## Opcode:

(MSB)

(LSB)

TO R<sub>n</sub>

0	0	0	1	n (0H~FH)
---	---	---	---	-----------

(1nH)

## Machine Cycles:

## ROM execution time

3 cycles

### RAM execution time

3 cycles

Cache BAM execution time 1 cycles

### Example:

**I** Under the following conditions:

$R_c = 7106H$   $R_o = 0028H$

the register R<sub>4</sub> becomes 712EH when the following program is executed.

FROM	R <sub>6</sub>
TO	R <sub>4</sub>
ADD	R <sub>2</sub>

## 9.90 UMULT R<sub>n</sub>

Operation: S<sub>reg</sub> (low byte) \* R<sub>n</sub> (low byte) → D<sub>reg</sub>

Description: This instruction performs 8 x 8-bit unsigned multiplication using the low byte of the source register and the low byte of register R<sub>n</sub>, specified in the operand. The result is stored in the destination register.

The source and destination registers are specified in advance using a FROM, WITH, or TO instruction. When not specified, the source and destination registers default to R<sub>0</sub>.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	*	-	*

B : Reset  
 ALT1 : Reset  
 ALT2 : Reset  
 S : Set when the result is negative, else reset.  
 Z : Set on zero result, else reset.

Opcode: (MSB) (LSB)

UMULT R <sub>n</sub>	0	0	1	1	1	1	0	1	(3DH)
	1	0	0	0	n (0H~FH)				(8nH)

Machine Cycles: ROM execution time 6 or 8 cycles  
 RAM execution time 6 or 8 cycles  
 Cache RAM execution time 2 or 3 cycles

Note: The number of cycles depends on the CONFIG register setting.

Example: Under the following conditions,

S<sub>reg</sub>: R<sub>3</sub>, D<sub>reg</sub>: R<sub>0</sub>, R<sub>3</sub>=364FH, R<sub>8</sub>=B2CFH

the register R<sub>0</sub> becomes 3FE1H when

UMULT R<sub>8</sub>

is executed.



## ***Chapter 1      Introduction to DSP1***

Digital Signal Processor (DSP1) is a 16-bit fixed point digital signal processor designed as a co-processor for the Super Nintendo Entertainment System (Super NES). It provides the Super NES programmer with advanced, high speed, pseudo three-dimensional programming capabilities. These functions are possible through the use of a command set held by the DSP1's internal ROM.

### **1.1 SUPER NES CPU SUPPORT**

DSP1 supports processing of the Super NES CPU through parallel operation. The increased processing speed and advanced processing capability greatly improves the realism of Super NES games.

### **1.2 PSEUDO 3-DIMENSIONAL GRAPHICS**

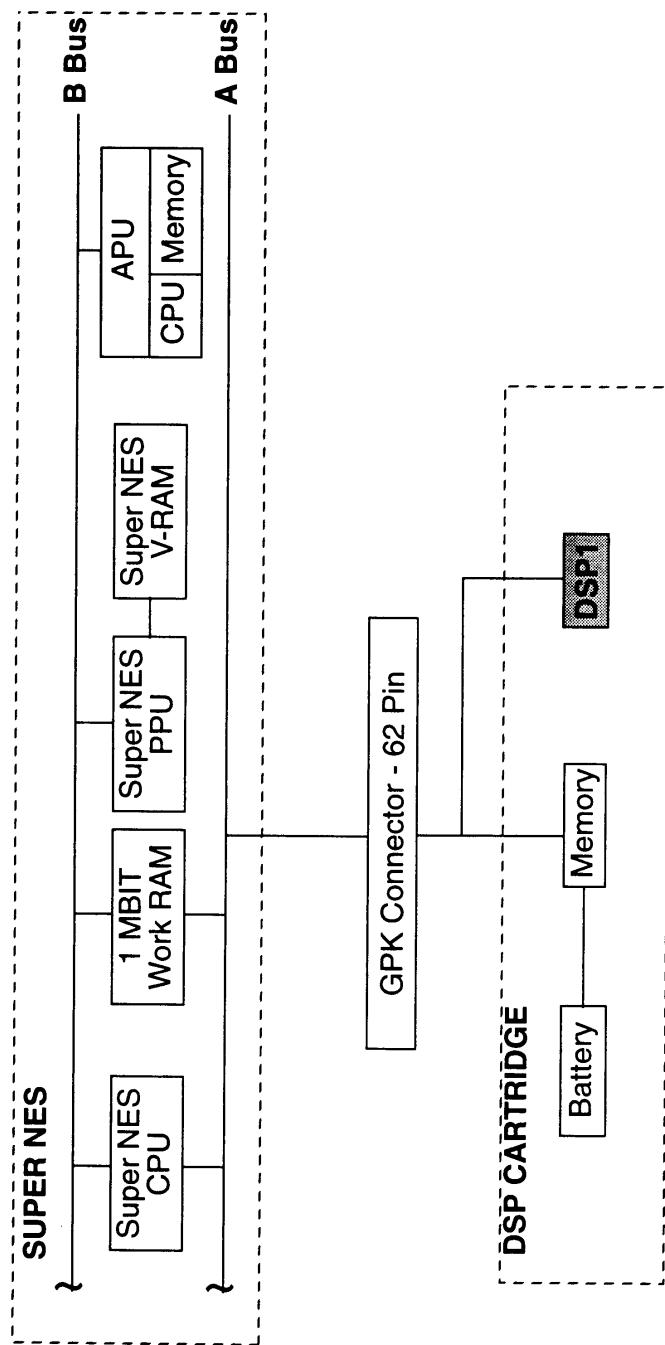
Because numerous commands for 3-dimensional graphics are incorporated, DSP1 is especially useful for 3-dimensional games, such as those involving flight simulation.

### **1.3 COMPLEX MATH PROCESSING**

General purpose commands for complex math calculation are also included within the DSP1 ROM. Calculations can be executed much faster than with the Super NES CPU. Therefore, DSP1 is useful in games which require high speed multiplication, division, and calculation of trigonometric functions.

### **1.4 SYSTEM BLOCK DIAGRAM**

The system block diagram, on the following page, illustrates the means by which the DSP1 is connected to the Super NES.



The DSP1 and Super NES CPU are connected by Bus A.  
The Super NES CPU executes the LOAD/STORE commands for DSP Data I/O.

Figure 3-1-1 System Block Diagram (DSP1)

## 1.5 DSP1 OPERATION

### 1.5.1 COMMAND EXECUTION

The DSP1 receives commands from the Super NES CPU and returns the results of its computations.

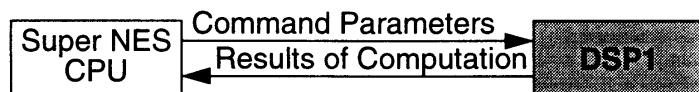


Figure 3-1-2 Super NES CPU and DSP1 Communications

Command execution between the Super NES and DSP1 is demonstrated below.

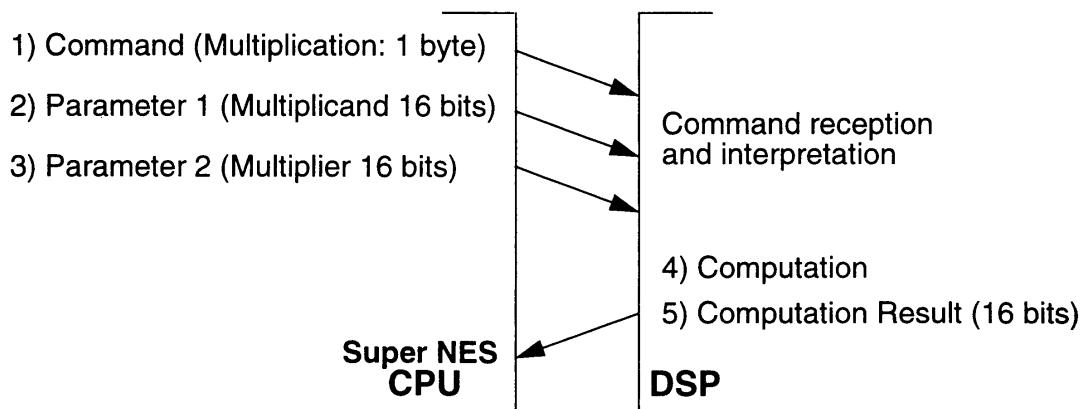


Figure 3-1-3 DSP1 Command Execution

## 1.6 MEMORY MAPPING

### 1.6.1 MODE 20/DSP

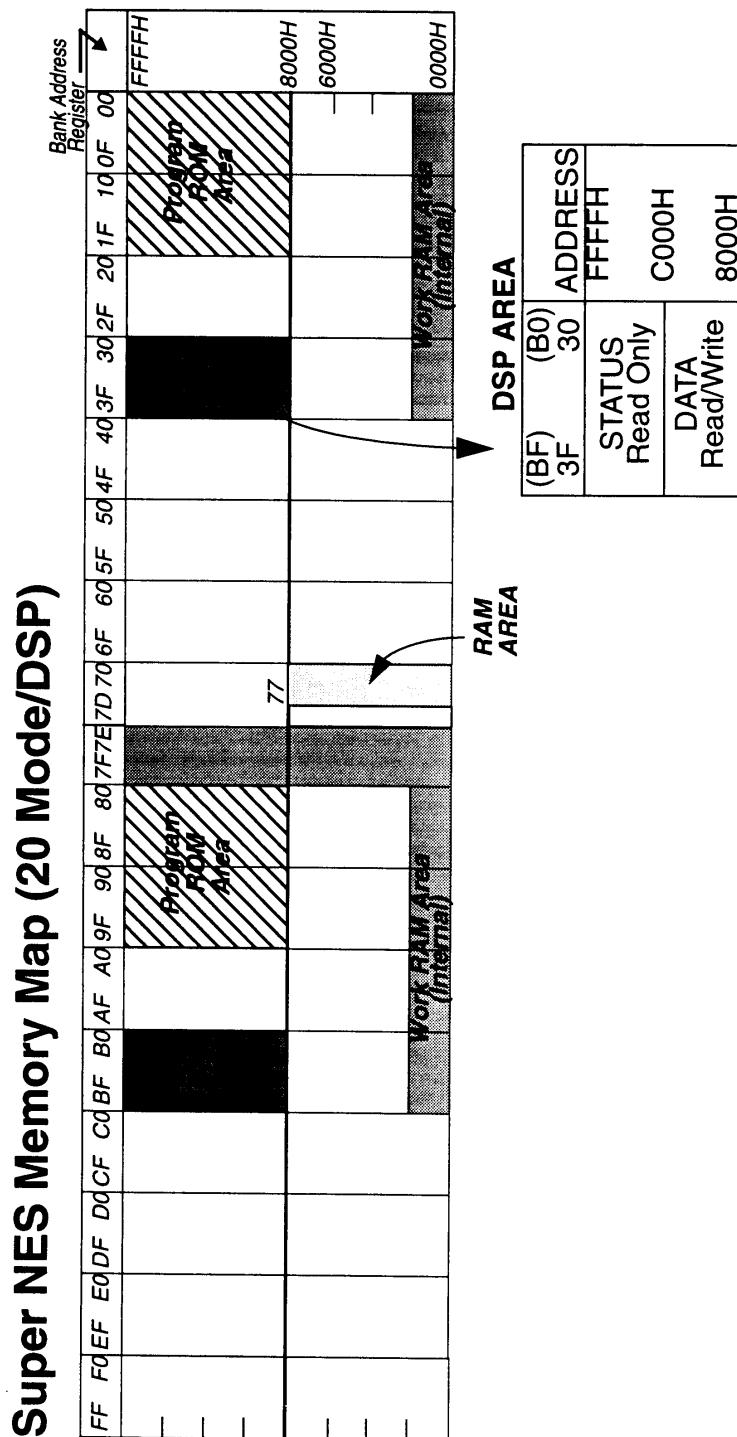
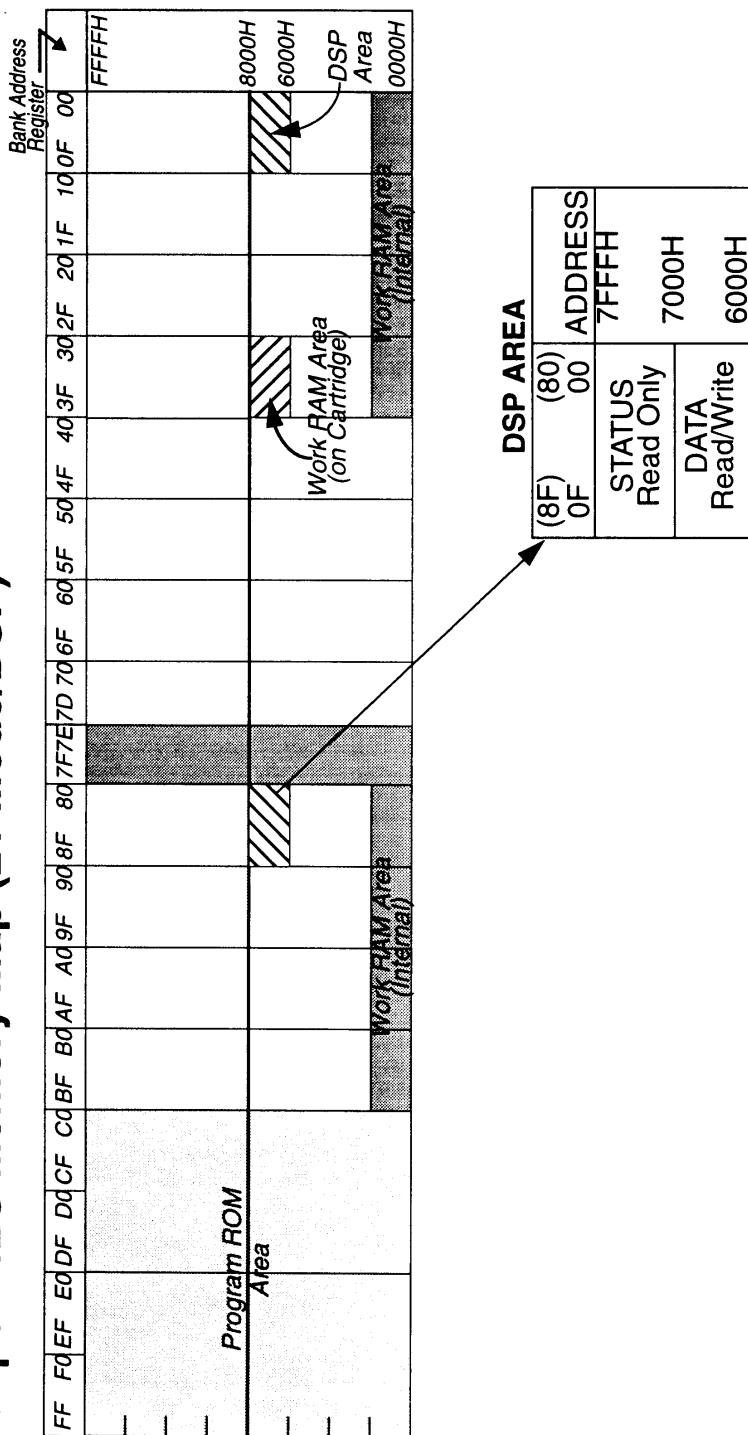


Figure 3-1-4 Mode 20/DSP Memory Map

Note 1: Use 8000H/C000H as the read/write port for DSP1.  
 Note 2: The maximum ROM capacity is 8 MBits in Mode 20.

### 1.6.2 MODE 21/DSP

**Super NES Memory Map (21 Mode/DSP)**



Note 1: Use 6000H/7000H as the read/write port for DSP1 operations.

Figure 3-1-5 Mode 21/DSP Memory Map

## Chapter 2 Command Summary

TYPE	COMMAND NAME	FUNCTION	CODE	CLOCK CYCLES	PROCESSING TIME (μsec)
General Calculation	Multiply	16-bit multiplication (decimal, integer)	00H	26	3.4
	Inverse	Inverse calculation (floating point)	10H	98	12.9
	Triangle	Trigonometric calculation (sin, cos)	04H	59	7.8
Vector Calculation	Radius	Vector size calculation	08H	34	4.5
	Range	Vector size comparison	18H	38	5.0
	Distance	Vector absolute value calculation	28H	156	20.5
Coordinate Calculation	Rotate	2-D coordinate rotation	0CH	65	8.6
	Polar	3-D coordinate rotation	1CH	147	19.3
Projection Calculation	Parameter	Projection parameter setting	02H	892	117.4
	Raster	Raster data calculation	0AH	224+209(n-1) 224+208(n-1)	29.5+27.5(n-1) 29.5+27.4(n-1)
	Project	Object projection calculation	1AH		
	Target	Coordinate calculation of a selected point on the screen	06H 0EH	627 228	82.5 30.0
Attitude Control	Attitude	Set attitude	01H 11H 21H	164 164 164	21.6 21.6 21.6
	Objective	Convert from global to object coordinates	0DH 1DH 2DH	45 45 45	5.9 5.9 5.9
	Subjective	Convert from object to global coordinates	03H 13H 23H	44 44 44	5.8 5.8 5.8
	Scalar	Calculation of inner product with the forward attitude and a vector	0BH 1BH 2BH	36 36 36	4.7 4.7 4.7
New Angle Calculation	Gyrate	3-D angle rotation	14H	444	58.4

Table 3-2-1 DSP1 Command Summary

- Note 1. The "n" in the processing speed and clock cycle columns indicates the number of times a process is repeated.
- Raster data calculation: The number of rasters calculated.
  - Data ROM read: Number of words in the ROM read.

Note 2. For commands with multiple codes, refer to the description of each command.

## Chapter 3 Parameter Data Type

The conventions used in the table below are employed throughout this manual when referring to parameters.

PARAMETER	DESCRIPTION	# BITS	DATA RANGE	UNIT
A	Angle	8	$-\pi \sim +\pi$ ( $-180^\circ \sim +180^\circ$ )	$2\pi/2^{16}$
T	Fixed Point Decimal	16	-1.0 ~ +0.999969...	$2^{-15}$
I8	Integer with decimal part (fixed point)	16	-128.0 ~ +127.996093...	$2^{-8}$
I	Integer	16	-32768 ~ +32767	1
2I	Double integer	17	-65536 ~ +65534	2
CI	Cyclic integer	16	► -32768 ~ +32767 □	1
U	Integer without a sign	16	0 ~ 65535	1
D	Double precision integer	32	-2147483648 ~ +2147483647	1
L	Low digit of double precision integer	16	---	---
H	High digit of double precision integer	16	---	---
D2	Double precision half integer	32	-1073741824 ~ +1073741823	$2^{-1}$
L2	Low digit of double precision half integer	16	---	---
H2	High digit of double precision half integer	16	---	---
M	Floating point coefficient	16	-1.0 ~ +0.999969...	1
C	Floating point exponent	16	-32768 ~ +32767	1

Table 3-3-1 Parameter Data Type

Note 1. The data transfer between the Super NES CPU and DSP1 is carried out in 16 bits regardless of the number of bits in each parameter selection shown in the above table.

Note 2. Though the resolution of the double precision semi-integer (D2) is  $2^{-1}$ , it is actually handled as an integer because the lowest bit is always used as 0.

Note 3. The exponent of a floating point number (C) can be stored in the range of 8002H to 7FFFH (-32766 to 32767).

## Chapter 4      Use of DSP1

### 4.1    DSP1 DR REGISTER

DSP1 processes Super NES CPU commands and parameters using an internal DR register that is mapped in the Super NES CPU "A" bus.

Commands and parameters are sent from the Super NES CPU to DSP1. Specifically, data is written to the memory-mapped DR register using the STORE command. The Super NES CPU and DSP1 do not perform handshaking operations. The Super NES CPU waits while DSP1 processes data, before sending the next data.

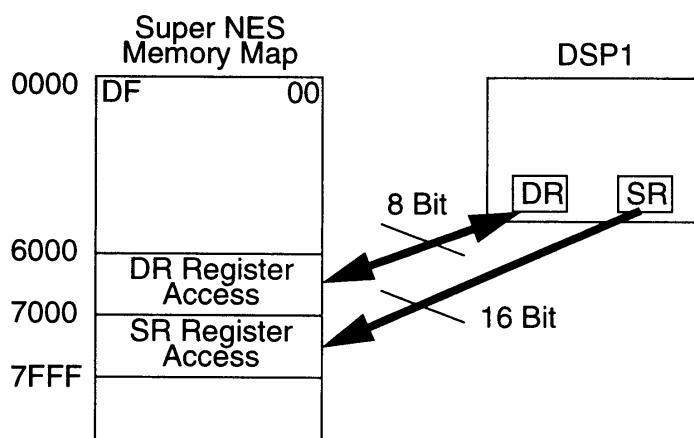


Figure 3-4-1    Super NES/DSP1 Memory Mapping (Mode 21)

DSP1 decodes commands, processes them according to the assigned parameters, and writes the results to the DR register. The Super NES CPU waits, while DSP1 processes the data, then reads the DR register using the LOAD command to obtain the results.

The DR register has 2 input/output modes, 8 bit and 16 bit. The DSP1 receives each command in the 8 bit mode. Once the command is received, the DR register is changed to the 16 bit mode. All input/output data is transferred in the 16 bit mode. The DR register mode is controlled by the DSP1 Status register.

## 4.2 DSP1 STATUS REGISTER

The status register is a 16-bit register which holds the status bits needed by the DSP1 to transfer data to and from external devices. The upper 8 bits can be read from an external device through pins D0 through D7 of DSP1. Only bit 15 is used by the Super NES. This bit is referred to as "RQM"

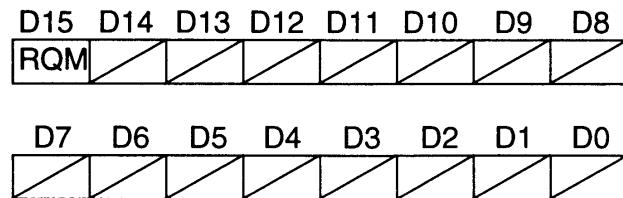


Figure 3-4-2 DSP1 Status Register Configuration

## 4.3 RQM

This bit indicates that the DSP1 is requesting data read from the Super NES CPU. The bit is "0" when the DSP1 is busy and "1" when it is ready to read or write.

## 4.4 DMA TRANSFER

Although DSP1 is capable of DMA data transfer, it is not supported by the Super NES system due to current hardware configuration.

## 4.5 OPERATION SUMMARY

The following figure shows the relationship and basic operations of the Super NES CPU and DSP1.

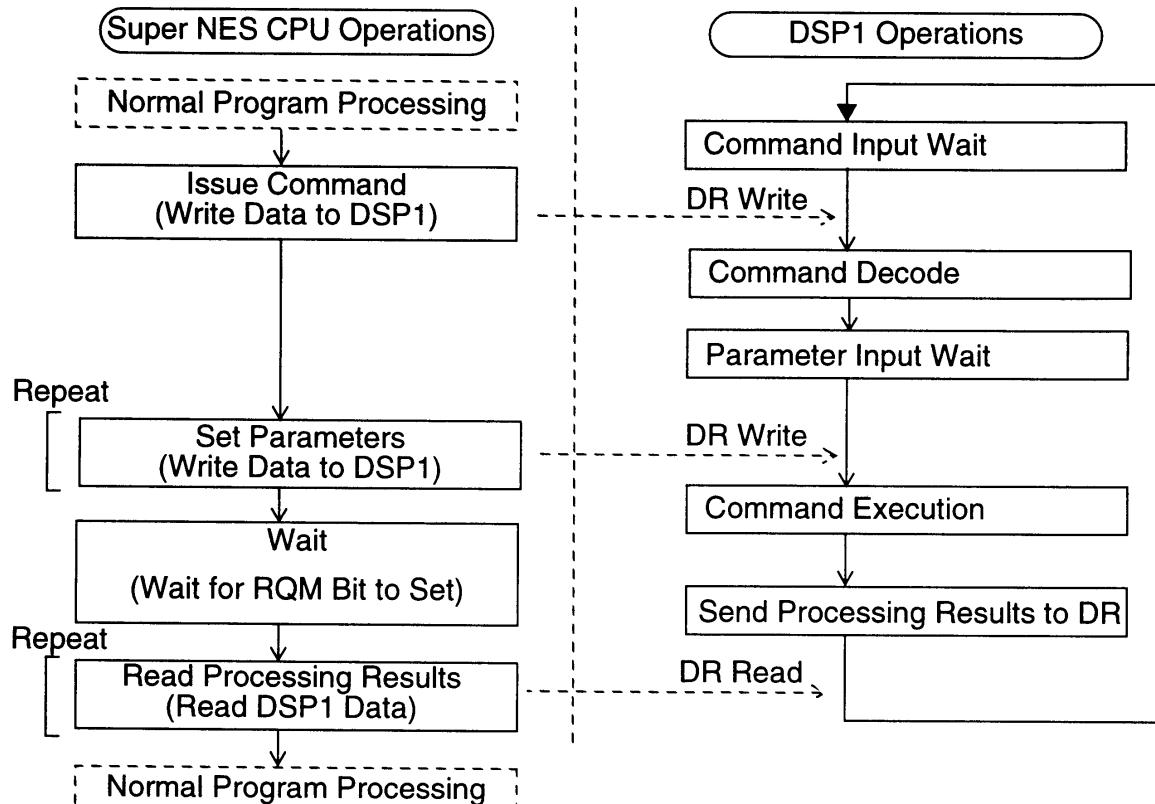


Figure 3-4-3 DSP1 Operations Flow Diagram

### Super NES CPU/DSP1 Operational Timing

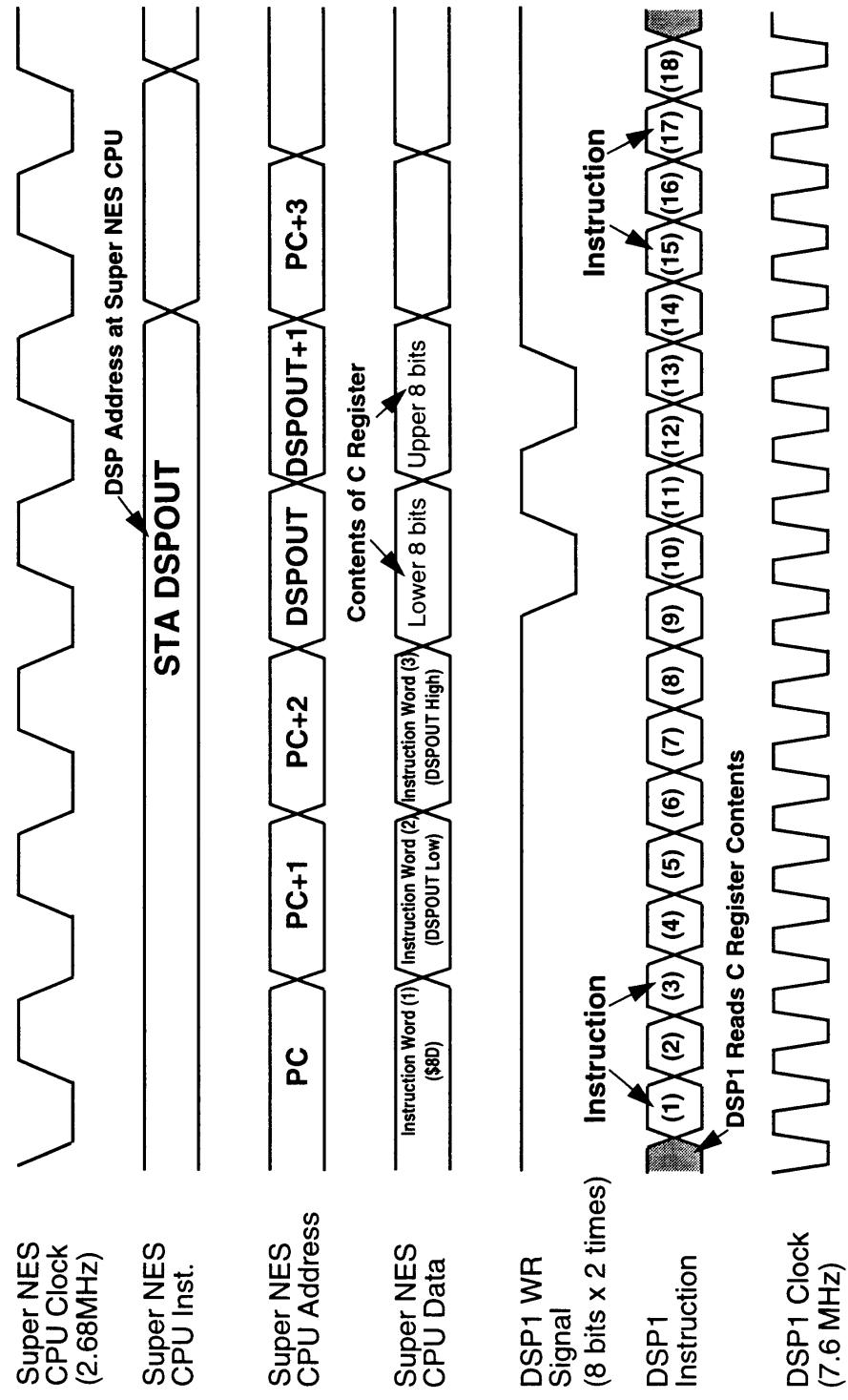


Figure 3-4-4 Super NES CPU/DSP1 Operational Timing

## ***Chapter 5      Description of DSP1 Commands***

### **5.1 GENERAL CALCULATION**

#### **5.1.1 16-BIT MULTIPLICATION (DECIMAL, INTEGER)**

Name:	Multiply		
Code:	00H <sup>*3</sup>		
Parameters:	Input	k[T/I] I[T/I]	Multiplicand Multiplier
	Output	M[T/H2]	Product (rounded fraction ≤ 15 bits)
Function:	This command determines the product, M, of decimal K and I. The command can also determine the product of integers [I], wherein the result of the calculation is a double precision half integer (H2).		

Equation 5-1:

$$k \times I = M$$

Number of Process Cycles:	Input	1. Command Input	6
	2. k input		12
	3. I input		4
	Output	1. M output	4

\*Notes: 1. Parameters are input/output via the DR register.

2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.
3. 00H is a hexadecimal code.

Example: This is a general command used in all types of calculations.

### 5.1.2 INVERSE CALCULATION (FLOATING POINT)

Name: Inverse  
 Code: 10H  
 Parameters: Input: a[M] Coefficient  
               b[C] Exponent (8002-7FFFH)  
               Output: A[M] Coefficient  
                  B[C] Exponent (8002-7FFFH)  
 Function: This command determines the inverse of a floating point decimal number.

Equation 5-2:

$$\frac{1}{a \times 2^b} = A \times 2^B$$

Number of Process Cycles:	Input	1. Command Input	6
	2. a input		13
	3. b input		73
	Output	1. A output	2
		2. B output	4

- \*Notes: 1. Parameters are input/output via the DR register.  
 2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: This is a general command used in all types of calculations.

### 5.1.3 TRIGONOMETRIC CALCULATION

Name: Triangle

Code: 04H

Parameters:	Input:	$\theta[A]$	Angle
		$r[T/I]$	Radius
	Output:	$S[T/I]$	sin
		$C[T/I]$	cos

Function: This command determines the product of the sin of angle  $\theta$  and radius  $r$ , and the product of the cosine and radius  $r$ . When the radius is an integer [I], the results are also an integer.

Equation 5-3:

$$C = r(\cos\theta) \quad S = r(\sin\theta)$$

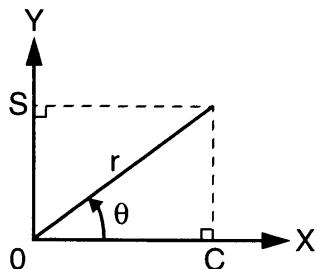


Figure 3-5-1 Trigonometric Calculation

Number of Process Cycles: Input

1. Command Input	6
2. $\theta$ input	12
3. $r$ input	34
Output	
1. S output	3
2. C output	4

\*Notes: 1. Parameters are input/output via the DR registers.

2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [sinθ, cosθ calculation]

Set  $r=1$  to calculate  $\sin\theta$  and  $\cos\theta$ .

[Vector component calculation]

Determines the X and Y components for a two-dimensional vector whose size and direction are known.

This is a general command which can be used in other types of calculations.

## 5.2 VECTOR CALCULATION

### 5.2.1 VECTOR SIZE

Name:	Radius															
Code:	08H															
Parameters:	<table border="0"> <tr> <td>Input:</td> <td>x[I]</td> <td>X component of the vector</td> </tr> <tr> <td></td> <td>y[I]</td> <td>Y component of the vector</td> </tr> <tr> <td></td> <td>z[I]</td> <td>Z component of the vector</td> </tr> <tr> <td>Output:</td> <td>L_L[L2]</td> <td>Vector size squared (lower)</td> </tr> <tr> <td></td> <td>L_H[H2]</td> <td>Vector size squared (upper)</td> </tr> </table>	Input:	x[I]	X component of the vector		y[I]	Y component of the vector		z[I]	Z component of the vector	Output:	L_L[L2]	Vector size squared (lower)		L_H[H2]	Vector size squared (upper)
Input:	x[I]	X component of the vector														
	y[I]	Y component of the vector														
	z[I]	Z component of the vector														
Output:	L_L[L2]	Vector size squared (lower)														
	L_H[H2]	Vector size squared (upper)														
Function:	This command determines vector size (square of the absolute value).															

Equation 5-4:

$$(x^2 + y^2 + z^2) = L$$

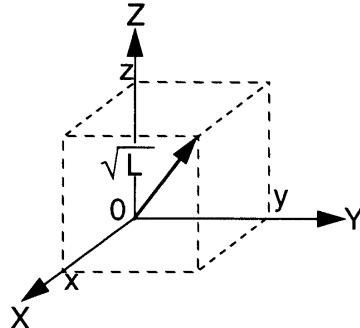


Figure 3-5-2 Vector Calculation

The absolute value of the vector  $R = \sqrt{L}$  is determined by the Distance command.

Number of Process Cycles:	Input	1. Command Input	6
	2. x input		14
	3. y input		4
	4. z input		4
	Output	1. L_L output	2
		2. L_H output	4

- \*Notes: 1. Parameters are input/output via the DR registers.  
 2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Distance between two points]

This command is useful for calculating the distance between two points. The command calculates the square of distance between two points, and may be used for calculating comparative data. One point of the vector is assumed to be X=0, Y=0 and Z=0.

### 5.2.2 VECTOR SIZE COMPARISON

Name:	Range	
Code:	18H	
Parameters:	Input:      x[T/I]      X component of the vector y[T/I]      Y component of the vector z[T/I]      Z component of the vector r[T/I]      Range to be compared against the vector size (sphere radius)	
	Output:      D[T/H2]      Difference between the vector size and the specified range.	
Function:	This command subtracts the square of the specified range from the square of the vector size. This command compares the vector size and the distance from a particular point, and so may be used to determine if a point is within the sphere. The parameters can be either decimal or integer.	

Equation 5-5:

$$x^2 + y^2 + z^2 - r^2 = D$$

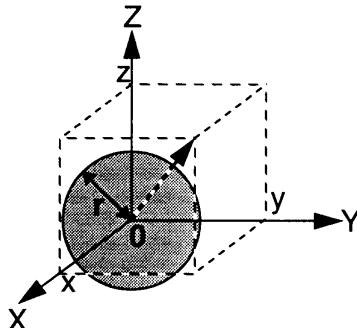


Figure 3-5-3 Vector Size Comparison

Number of Process Cycles: Input

1. Command Input	6
2. x input	12
3. y input	4
4. z input	4
5. r input	8

Output      1. D output      4

- \*Notes:
1. Parameters are input/output via the DR registers.
  2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Detects a collision in three-dimension]

This command determines if an object is within a certain range of a point. It can be used to detect three-dimensional collisions.

### 5.2.3 VECTOR ABSOLUTE VALUE CALCULATION

Name:	Distance	
Code:	28H	
Parameters:	Input:	x[I/T]      X component of the vector y[I/T]      Y component of the vector z[I/T]      Z component of the vector
	Output:	R[I/T]      Vector size
Function:	This command determines vector size (absolute value). The parameters can be either decimal or integer.	

Equation 5-6:  $\sqrt{x^2 + y^2 + z^2} = R$

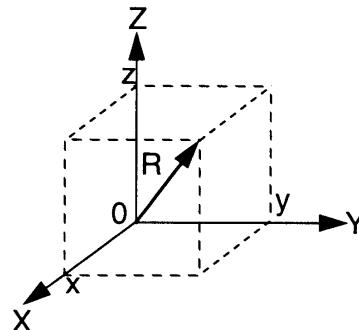


Figure 3-5-4 Vector Absolute Value Calculation

Number of Process Cycles:	Input	1. Command Input	6
	2. x input	15	
	3. y input	4	
	4. z input	127	
	Output	1. R output	4

- \*Notes:
1. Parameters are input/output via the DR registers.
  2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Distance between two points]

This command calculates the distance between two 3-D points on the coordinate. In contrast to the Radius command.

## 5.3 COORDINATE CALCULATION

### 5.3.1 TWO-DIMENSIONAL COORDINATE ROTATION

Name:	Rotate
Code:	0CH
Parameters:	Input: $\theta[A]$ Angle of rotation about the Z axis (counterclockwise)
	$x_1[I]$ X coordinate before rotation
	$y_1[I]$ Y coordinate before rotation
	Output: $x_2[I]$ X coordinate after rotation
	$y_2[I]$ Y coordinate after rotation
Function:	This command determines the (X,Y) coordinates after rotating (x,y) counterclockwise for $\theta$ .

Equation 5-7:

$$(x_1, y_1) \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = (x_2, y_2)$$

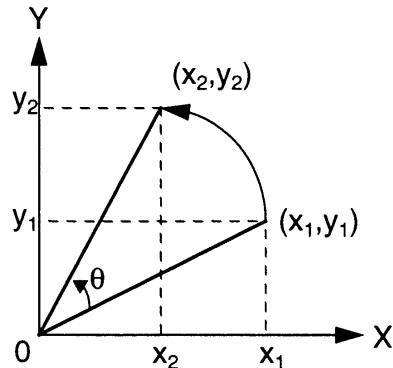


Figure 3-5-5 Two-Dimensional Coordinate Rotation

Number of Process Cycles: Input	1. Command Input	6
	2. $\theta$ input	12
	3. $x_1$ input	3
	4. $y_1$ input	37
Output	1. $x_2$ output	2
	2. $y_2$ output	4

- \*Notes:
1. Parameters are input/output via the DR registers.
  2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

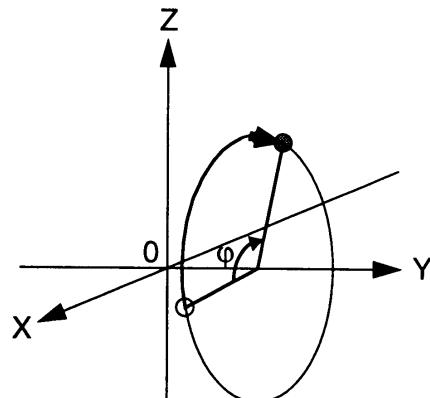
Example: [Coordinate calculation for rotating an object on a surface]  
 This command calculates the coordinates of an object after it is rotated on a surface.

### 5.3.2 THREE-DIMENSIONAL COORDINATE ROTATION

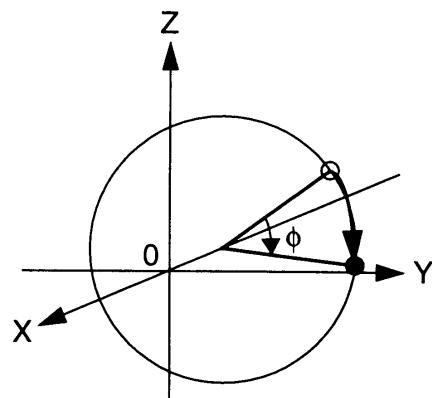
Name:	Polar		
Code:	1CH		
Parameters	Input:	$\theta[A]$	Angle of rotation about the Z axis (positive from the Y axis to the X axis)
		$\phi[A]$	Angle of rotation about the X axis (positive from the Z axis to the Y axis)
		$\varphi[A]$	Angle of rotation about the Y axis (positive from the X axis to the Z axis)
	x[I]	X coordinate before rotation	
	y[I]	Y coordinate before rotation	
	z[I]	Z coordinate before rotation	
	Output:	X[I]	X coordinate after rotation
		Y[I]	Y coordinate after rotation
		Z[I]	Z coordinate after rotation
Function:	This command determines the (X,Y,Z) coordinates when rotating (x,y,z) three-dimensionally. Rotation is performed in the order of $\phi$ about the Y axis, $\varphi$ about the X axis, and $\theta$ about the Z axis.		

Equation 5-8:

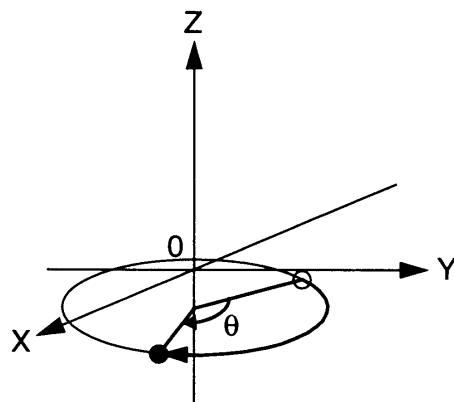
$$(x, y, z) \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = (X, Y, Z)$$



Rotation on Y axis



Rotation on X axis

**Rotation on Z axis**

**Note:** To be compatible with the projection and attitude-control commands, the X axis shall be east-west (east = +), the Y axis shall be north-south (north = +), and the Z axis shall be up and down (up = +).

Number of Process Cycles: Input	1. Command Input	6
	2. θ input	13
	3. φ input	3
	4. ϕ input	2
	5. x input	2
	6. y input	2
	7. z input	107
Output	1. X output	6
	2. Y output	2
	3. Z output	4

- \*Notes: 1. Parameters are input/output via the DR registers.  
 2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

**Example:** [Coordinate calculation for three-dimensional rotation of an object]

This command calculates the coordinates of an object after three-dimensional rotation. (Refer to the diagram on the following page.)

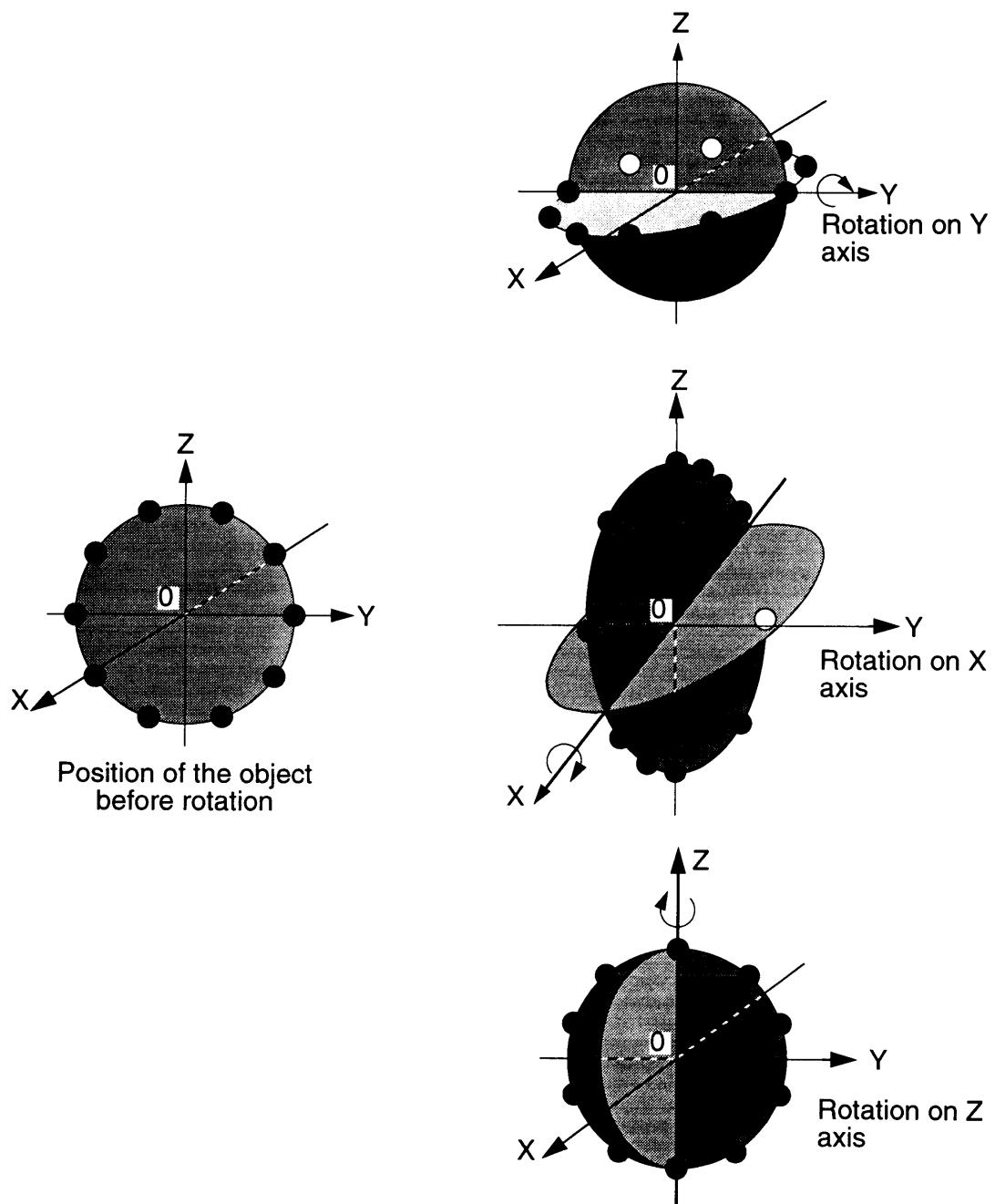


Figure 3-5-6 Examples of Three-Dimensional Rotation

## 5.4 PROJECTION CALCULATION

### 5.4.1 PROJECTION PARAMETER SETTING

Name:	Parameter	
Code:	02H	
Parameters:	Input:	F <sub>x</sub> [CI] X coordinate of base point (global coordinates) F <sub>y</sub> [CI] Y coordinate of base point (global coordinates) F <sub>z</sub> [CI] Z coordinate of base point (global coordinates) L <sub>fe</sub> [U] Distance between base point and viewpoint (Sets screen-sprite ratio.) L <sub>es</sub> [U] Distance between viewpoint and screen (The effect of screen angle considered; the screen horizontal distance is 256) A <sub>as</sub> [A] Azimuth angle of view line with respect to global coordinates. (East is 0° and positive toward the north) A <sub>zs</sub> [A] Zenith angle of view line with respect to global coordinates. (Zenith is 0°, 0°~180°).
	Output:	V <sub>of</sub> [I] Raster number of imaginary center V <sub>va</sub> [I] Raster number representing horizontal line. C <sub>x</sub> [CI] X coordinate of the point projected on the center of the screen (ground coordinates) C <sub>y</sub> [CI] Y coordinate of the point projected on the center of the screen (ground coordinates)
Function:	This command sets various projection parameters and calculates the basic data used in subsequent processes. The command places the viewer behind a fixed point such as an airplane. If the distance between the fixed point and the view point is set to 0, then the viewer sees the display from the perspective of the airplane.	

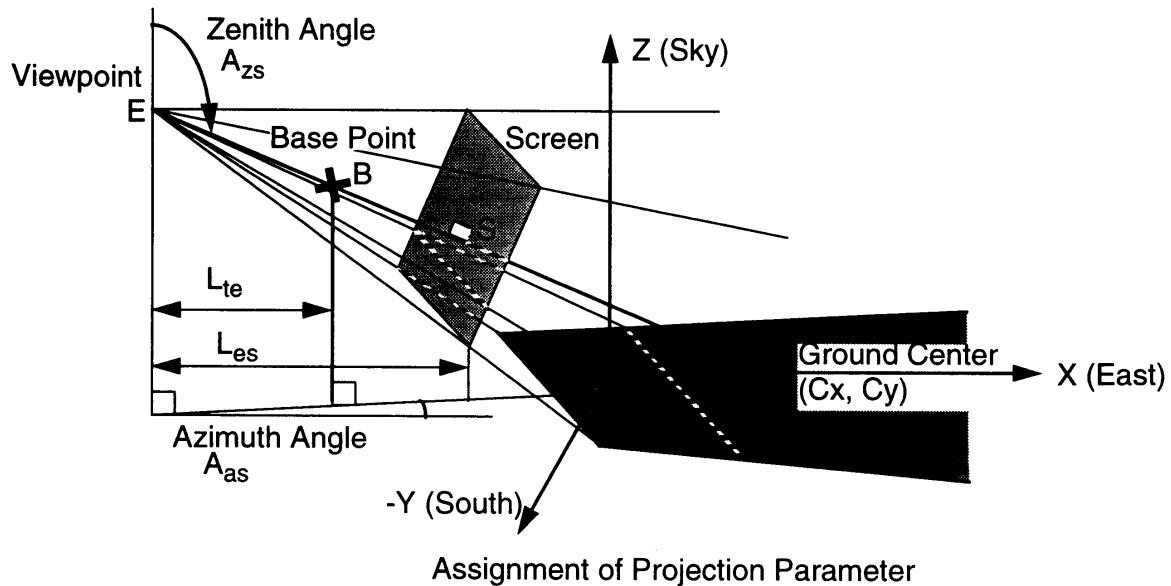


Figure 3-5-7 Assignment of Projection Parameter

$V_{va}$  (number of the raster used to display a horizontal line) indicates the border between background environments such as sky or cloud and a horizontal plane such as earth or sea. For raster numbers larger than  $V_{va}$  (representing the area below the horizon line), a horizontal plane is displayed on the screen, but the matrix elements for each raster are calculated individually using the RASTER command.

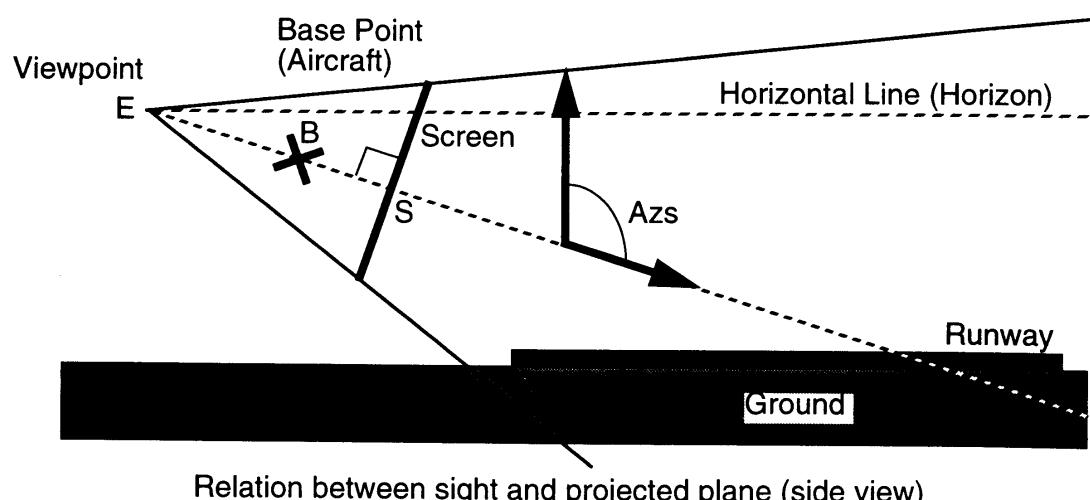


Figure 3-5-8 Relationship of Sight and Projected Plane

Cx and Cy (global coordinates for the point projected on the center of the screen) are the center coordinates used for rotation, and must be specified to the PPU.

Number of Process Cycles:	Input	1. Command Input	6
		2. F <sub>x</sub> input	11
		3. F <sub>y</sub> input	2
		4. F <sub>z</sub> input	2
		5. L <sub>fe</sub> input	3
		6. L <sub>es</sub> input	3
		7. A <sub>as</sub> input	4
		8. A <sub>zs</sub> input	839
	Output	1. V <sub>of</sub>	2
		2. V <sub>va</sub>	10
		3. C <sub>x</sub>	5
		4. C <sub>y</sub>	5

Example: [Parameter setting necessary for projection]

Pilot Wings displays the view seen from the view point directly behind an airplane which is at the fixed point. When the distance between the screen and view point is set to 256 (when the horizontal width of the screen is 256), the horizontal screen angle is 50°.

### 5.4.2 RASTER DATA CALCULATION

Name:	Raster	
Code:	0AH (To output result of calculation via DMA.) 1AH (When result of calculation is not output via DMA.)	
Parameters:	Input:	V <sub>s</sub> [I]
	Output:	A <sub>n</sub> [I8]
		B <sub>n</sub> [I8]
		C <sub>n</sub> [I8]
		D <sub>n</sub> [I8]
Function:	<p>This command calculates the linear transformation matrix elements (A, B, C, D) for each raster based on the various projection parameters specified with the Parameter command in internal RAM. Effects of Perspective can be achieved by specifying the matrix elements for each raster to the PPU to display distant objects (small) and near objects (large). Results of these calculations can be output in one of two modes. Normally, the results are read from the Super NES CPU using software. The results are output successively in the order of A⇒B⇒C⇒D⇒A⇒B••• until the command is completed. The command is ended by writing 8000H to the DR instead of reading element D.</p>	

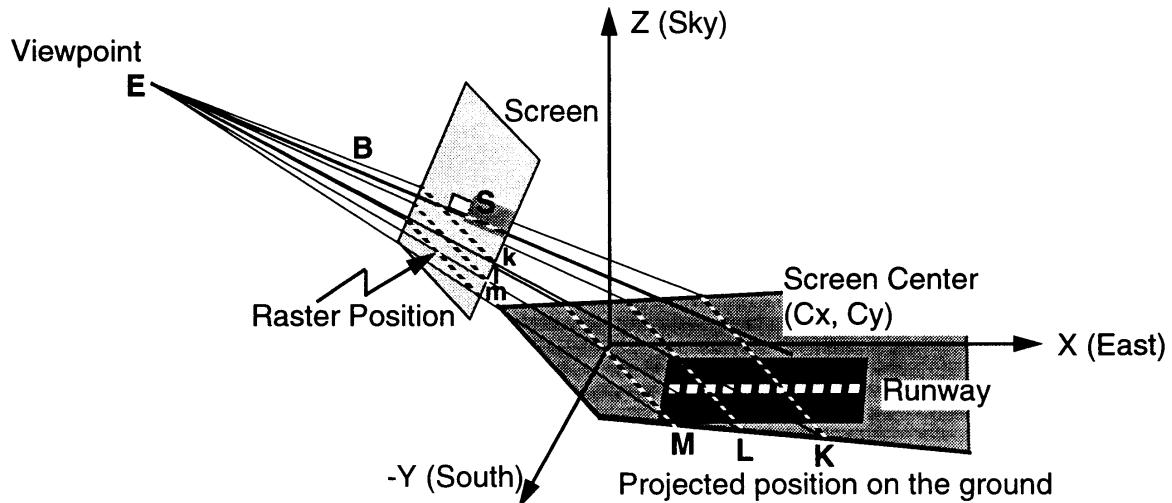


Figure 3-5-9 Calculation of Raster Data

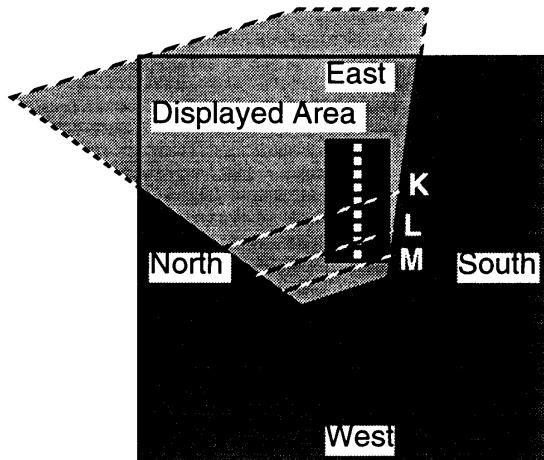


Figure 3-5-10 BG Screen and Displayed Area

Number of Process Cycles: Input	1. Command Input	6
	2. $V_s$ input	211
Output	1. $A_n$	3
	2. $B_n$	3
	3. $C_n$	3
	4. $D_n$	$200^{*1}$
	5. $D_n$	$7^{*2}$

\*Notes: 1. Until  $A_{n+1}$  is output.

2. Until the command is interrupted and the next command can be selected.

**Example:** [Calculation of linear transformation matrix elements for projection]

This command is used frequently for projection of the ground objects (airplane runway, sky diving target point, etc.) in Pilot Wings.

### 5.4.3 OBJECT PROJECTION CALCULATION

Name:	Project		
Code:	06H		
Parameters:	Input:	x[I]	X coordinate of the object (global coordinates)
		y[I]	Y coordinate of the object (global coordinates)
		z[I]	Z coordinate of the object (global coordinates)
	Output:	H[I]	H coordinate of the object projected on the screen (screen coordinates, right is positive).
		V[I]	V coordinate of the object projected on the screen (screen coordinates, down is positive).
		M[I]	Enlargement ratio for projected object.
Function:	This command calculates the location and size of the projection of an object on the screen based on various projection parameters specified with the Parameter command in internal RAM. The center of the screen is the origin of the screen coordinates (0,0).		

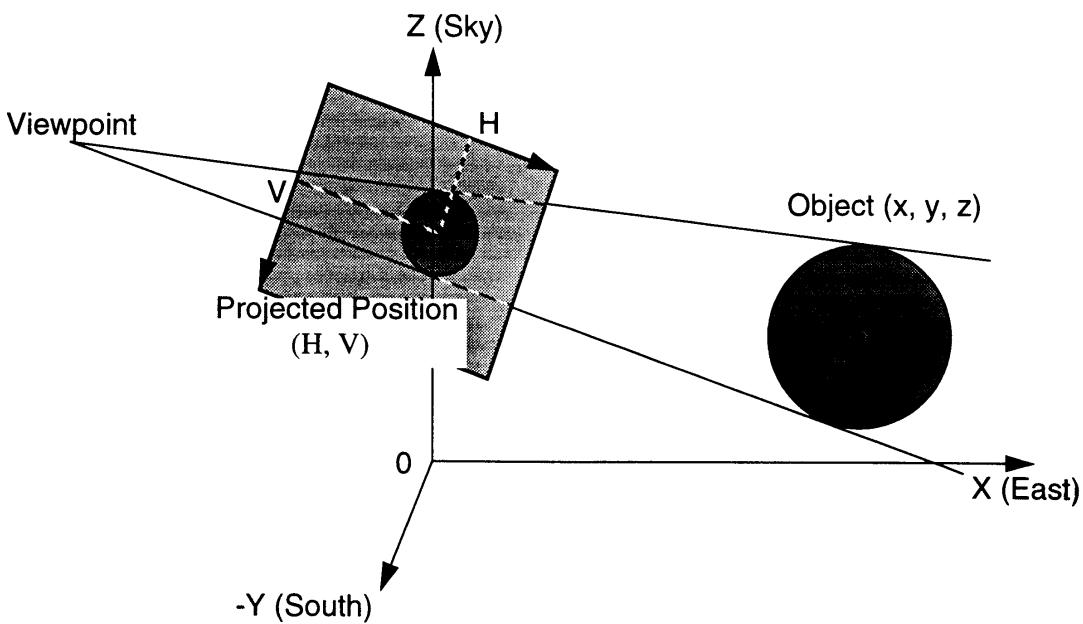


Figure 3-5-11 Calculation of Projected Position of Object

Number of Process Cycles: Input	1. Command Input	6
	2. x input	12
	3. y input	4
	4. z input	596
Output	1. H output	3
	2. V output	2
	3. M output	4 <sup>*1</sup>

\*Notes: 1. Until the next command can be selected.

Example: [Calculation of the projected location (on the screen) of a floating object]

This command is used in Pilot Wings to project a ring consisting of floating balls. The location and size of the balls projected on the screen are calculated based on the balls' global coordinates. By changing the location and size of the balls' sprite, three-dimensional display of the ring projected on the screen can be achieved.

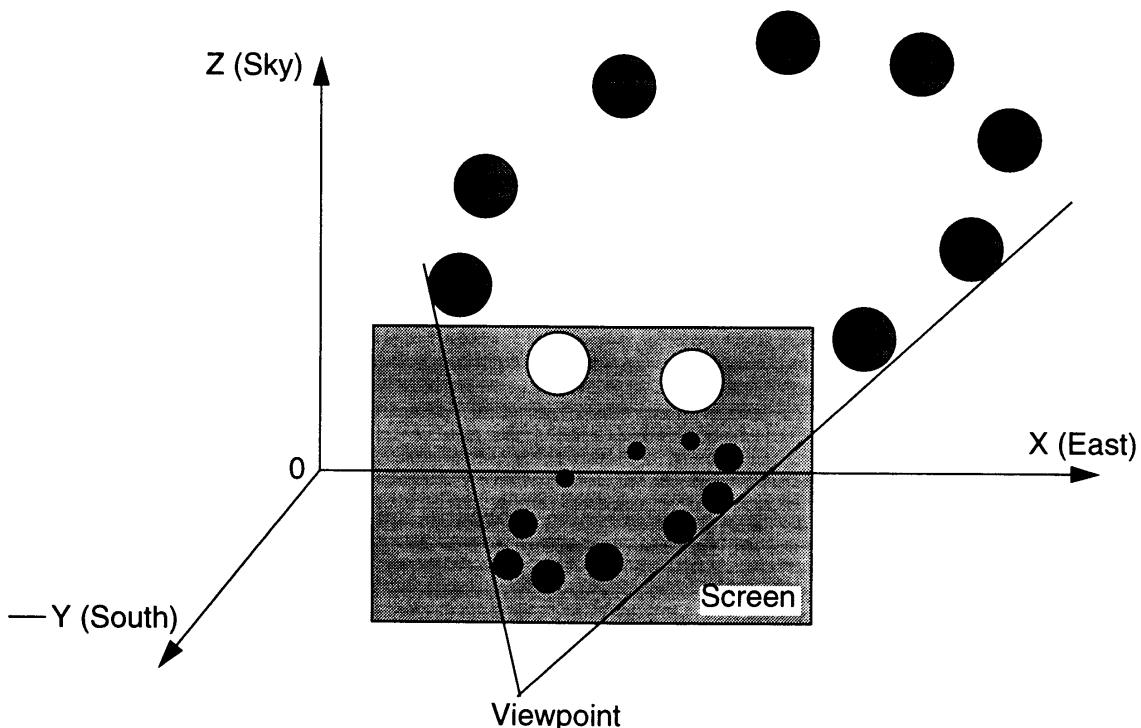


Figure 3-5-12 Projection Image of Object

#### **5.4.4 COORDINATE CALCULATION OF A SELECTED POINT ON THE SCREEN**

Name: Target

Code: 0EH

Parameters:	Input:	$h[l]$	H coordinate of selected point on the screen (screen coordinates, right is positive)
		$v[l]$	V coordinate of selected point on the screen (screen coordinates, down is positive)

Note: The origin coordinates of the screen designate the center of the screen.

Output:	X[I]	X coordinate of selected point (global coordinates, east is positive).
	Y[I]	Y coordinate of selected point (global coordinates, south is positive).

**Function:** This command calculates the coordinates of a selected “ground” point on the screen. The command calculates the global coordinates (X,Y) (the Z coordinate is zero) of the point projected on a point selected by a cursor or target mark based on the screen coordinates (H,V) of the selected point.

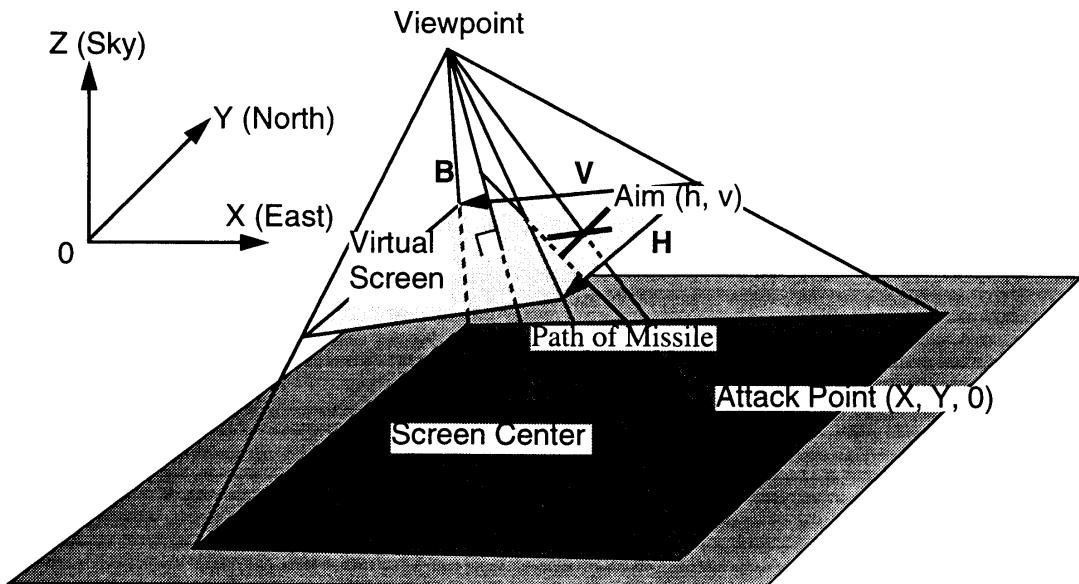


Figure 3-5-13 Calculation of Coordinates for the Indicated Point on the Screen

Number of Process Cycles: Input	1. Command Input	6
	2. h input	11
	3. v input	203
Output	1. X output	4
	2. Y output	4 <sup>*1</sup>

\*Notes: 1. Until the next command can be selected.

Example: [Calculation of the target on the ground when attacking from the sky]

This command is used in Pilot Wings when the helicopter attacks a target on the ground using a missile scope. When the missile launch button is pressed, the location of the point on the ground which is targeted in the scope is calculated and a missile is launched on that vector. The trajectory of the missile is a straight line toward that point and is not affected by the velocity of the helicopter at the time of the launch.

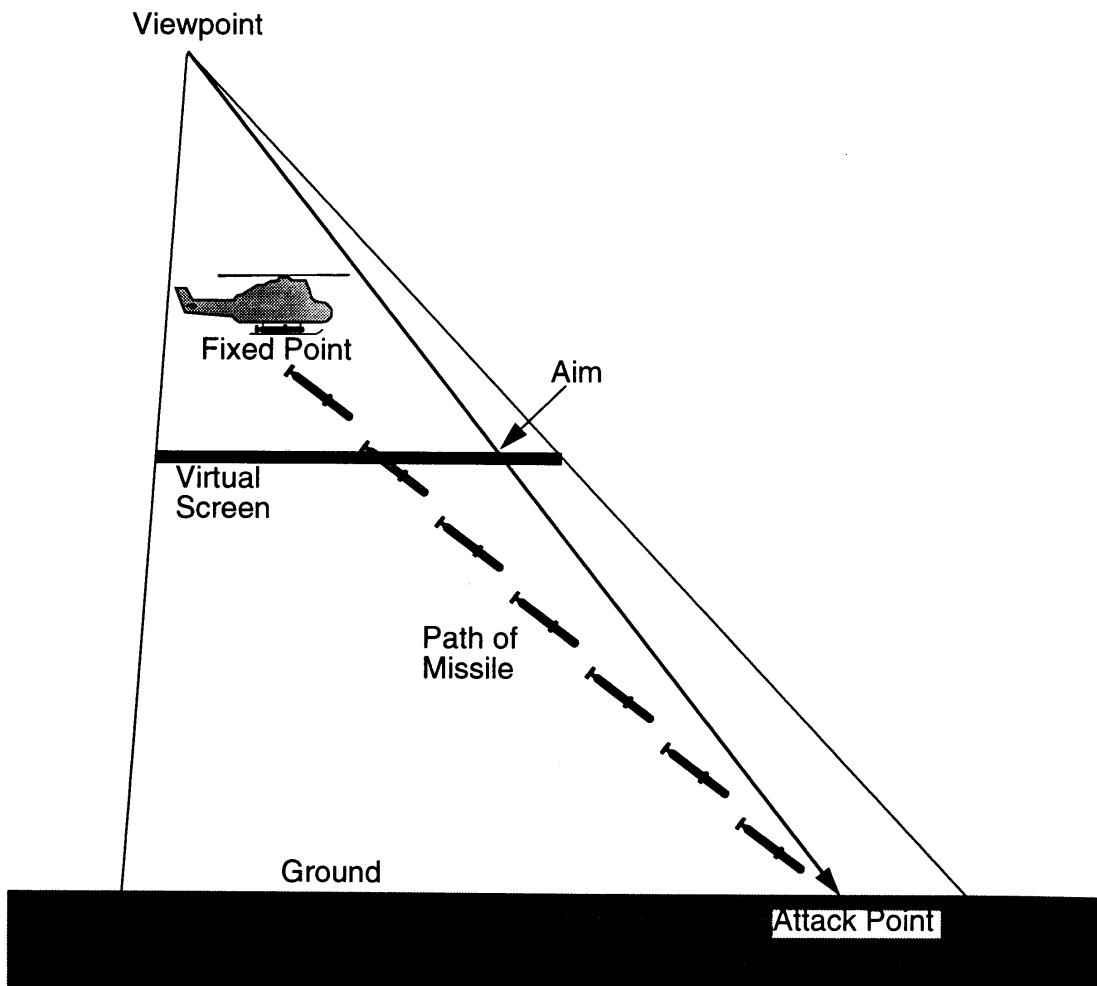


Figure 3-5-14 Attack Point and Position Indicated on Screen (Side View)

## 5.5 ATTITUDE CONTROL

### 5.5.1 SET ATTITUDE

Name:	Attitude	
Code:	01H (To select attitude matrix A) 11H (To select attitude matrix B) 21H (To select attitude matrix C)	
Parameters:	Input:      m[T]      Constant $\theta[A]$ Rotational angle about the Z axis (from Y axis to X axis is +) $\phi[A]$ Rotational angle about the X axis (from Z axis to Y axis is +) $\psi[A]$ Rotational angle about the Y axis (from X axis to Z axis is +)	
Function:	This command calculates a matrix which represents a three-dimensional rotation (attitude change). The order of rotation is $\phi$ about the Y axis (north-south), $\psi$ about the X axis (east-west), and $\theta$ about the Z axis (up-down). By applying the attitude matrix to the object coordinates (FLU coordinates), the global coordinates (XYZ coordinates) can be obtained (the SUBJECTIVE command). By applying the inverse of the attitude matrix (transpose matrix) to the global coordinates, the object coordinates can be calculated (the OBJECTIVE command).	
	Calculates attitude matrix A when the code is 01H (M=A)	
	Calculates attitude matrix B when the code is 11H (M=B)	
	Calculates attitude matrix C when the code is 21H (M=C)	

Equation 5-9:

$$m \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = M$$

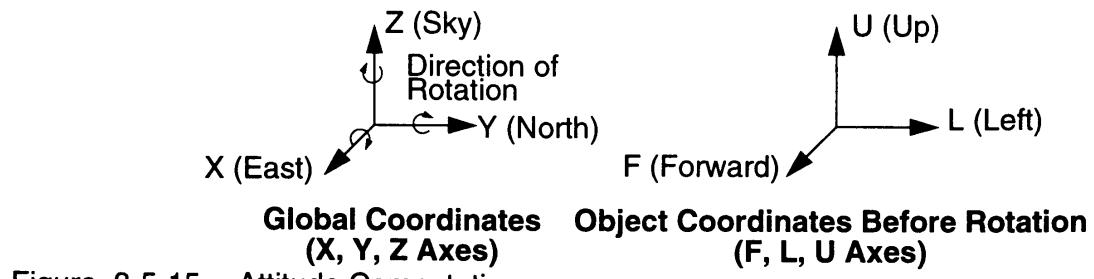


Figure 3-5-15 Attitude Computation

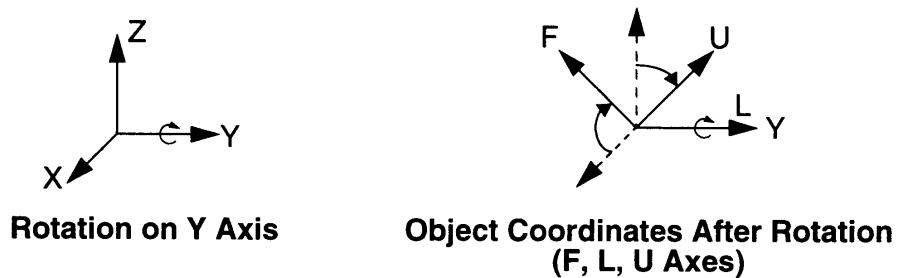


Figure 3-5-16 Object Coordinate Rotated on Y Axis

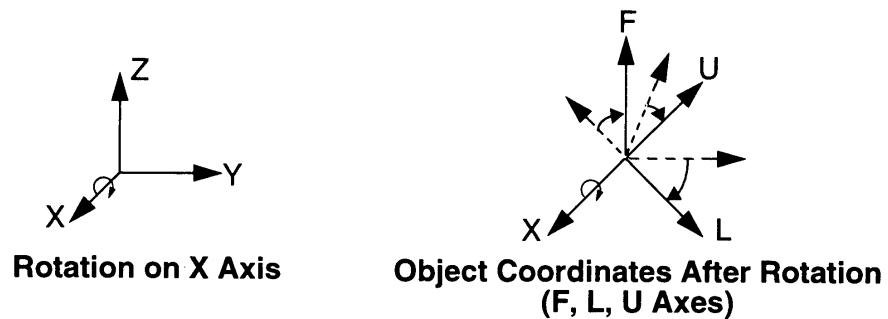


Figure 3-5-17 Object Coordinate Rotated on X Axis

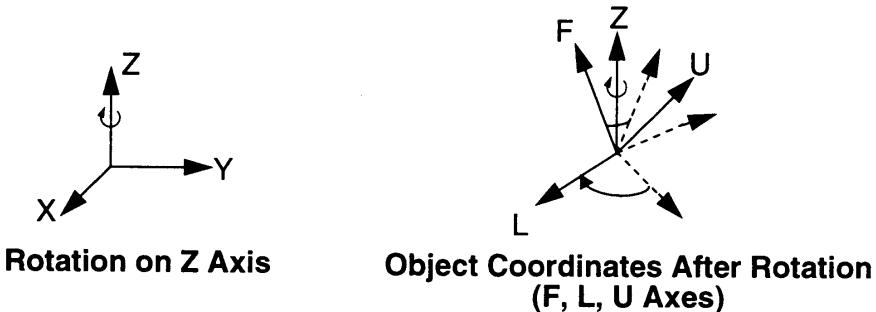


Figure 3-5-18 Object Coordinate Rotated on Z Axis

Number of Process Cycles: Input	1. Command Input	6
	2. m input	13
	3. θ input	4
	4. φ input	4
	5. ϕ input	137 <sup>*1</sup>

\*Notes: 1. Until the next command can be selected.

Example: [Calculation of attitude matrix for global-object coordinate conversion]

This command is used to calculate necessary attitude matrices using 3 commands for attitude control. When the attitude changes, this command must be used to renew attitude control matrices.

### 5.5.2 CONVERT FROM GLOBAL TO OBJECT COORDINATES

Name:	Objective		
Code:	0DH (To select attitude matrix A) 1DH (To select attitude matrix B) 2DH (To select attitude matrix C)		
Parameters:	Input:	x[I]	X coordinate of object (global coordinates, east)
		y[I]	Y coordinate of object (global coordinates, north)
		z[I]	Z coordinate of object (global coordinates, up)
	Output:	F[2I]	F coordinate of object (object coordinates, forward)
		L[2I]	L coordinate of object (object coordinates, left)
		U[2I]	U coordinate of object (object coordinates, up)
Function:	<p>Attitude matrices (A,B,C) represent the three-dimensional relationship between rotation angles of the object coordinates (the FLU axes) and global axes (the XYZ axes). The global coordinates are obtained by multiplying the object coordinates with attitude matrices (i.e., by rotating three-dimensionally). Inversely, the object coordinates are obtained by multiplying the global coordinates with inverse of the attitude matrices (i.e., by rotating in the opposite direction and order).</p> <p>Calculates the product with inverse of the matrix A when the code is 0DH (<math>M^{-1}=A^{-1}</math>).</p> <p>Calculates the product with inverse of the matrix B when the code is 1DH (<math>M^{-1}=B^{-1}</math>).</p> <p>Calculates the product with inverse of the matrix C when the code is 2DH (<math>M^{-1}=C^{-1}</math>).</p>		

Equation 5-10:

$$\frac{1}{2} (x, y, z) M^{-1} = (F, L, U)$$

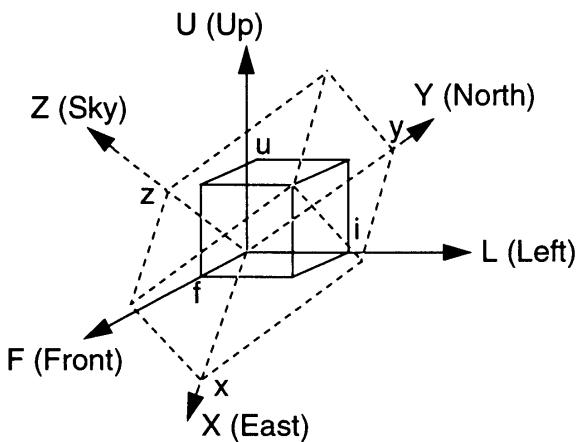


Figure 3-5-19 Conversion of Global to Objective Coordinates

Number of Process Cycles: Input	1. Command Input	6
	2. x input	14
	3. y input	4
	4. z input	7
Output	1. F output	5
	2. L output	5
	3. U output	4

- \*Notes:
1. Parameters are input/output via the DR registers.
  2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Conversion from the global coordinates to object coordinates]

In Pilot Wings, the conversion of objective coordinates to global coordinates for the aircraft is calculated using wind effects. Using these calculations, the course and speed of the aircraft may be altered by wind direction and speed.

### 5.5.3 CONVERSION FROM OBJECT TO GLOBAL COORDINATES

Name:	Subjective	
Code:	03H (To select attitude matrix A) 13H (To select attitude matrix B) 23H (To select attitude matrix C)	
Parameters:	Input: F[21] F coordinate of object (object coordinates, forward) L[21] L coordinate of object (object coordinates, left) U[21] U coordinate of object (object coordinates, up)	
	Output: X[I] X coordinate of object (global coordinates, east) Y[I] Y coordinate of object (global coordinates, north) Z[I] Z coordinate of object (global coordinates, up)	
Function:	Attitude matrices (A,B,C) represent the three-dimensional relationship between rotation angles of the object coordinates (FLU axes) and global axes (XYZ axes). The global coordinates are obtained by multiplying the object coordinates with attitude matrices (i.e., by rotating three-dimensionally).	
	Calculates product with attitude matrix A when the code is 03H (M=A) Calculates product with attitude matrix B when the code is 13H (M=B) Calculates product with attitude matrix C when the code is 23H (M=C)	

Equation 5-11:

$$\frac{1}{2} (F, L, U) M = (X, Y, Z)$$

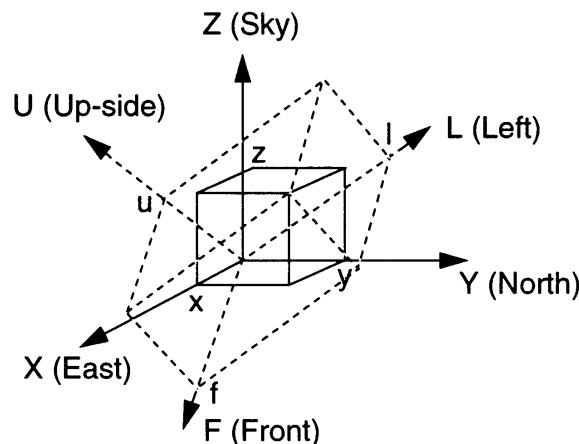


Figure 3-5-20 Conversion of Object to Global Coordinates

Number of Process Cycles: Input	1. Command Input	6
	2. F input	13
	3. L input	4
	4. U input	7
Output	1. X output	5
	2. Y output	5
	3. Z output	4

- \*Notes:
1. Parameters are input/output via the DR registers.
  2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Calculation of the global coordinates after change in the object's attitude]

In Pilot Wings, the object coordinates of the ring of balls remain the same unless the size of the balls or the shape or size of the ring is changed because there is one object coordinate system dedicated for the ring. When the direction (attitude) of the ring is changed, the ATTITUDE command is used to renew the attitude matrices. The ring with the new attitude can be displayed by calculating the global coordinates using the new attitude matrices and calculating the location of balls' projection using the PROJECT command. The same process takes place when the object coordinates change without a change in attitude or when both attitude and object coordinates change.

### 5.5.4 CALCULATION OF INNER PRODUCT WITH FORWARD ATTITUDE AND A VECTOR

Name: Scalar

Code: 0BH (To select attitude matrix A)  
1BH (To select attitude matrix B)  
2BH (To select attitude matrix C)

Parameters: Input:  $x[I]$  X component of vector.  
 $y[I]$  Y component of vector.  
 $z[I]$  Z component of vector.

Output:  $S[I]$  Inner product

Function: This command selects an attitude matrix based on the code. It calculates the inner product of a vector and the first row of the selected matrix.

When the code is 0BH,  $S = x \cdot A_{fx} + y \cdot A_{fy} + z \cdot A_{fz}$

When the code is 1BH,  $S = x \cdot B_{fx} + y \cdot B_{fy} + z \cdot B_{fz}$

When the code is 2BH,  $S = x \cdot C_{fx} + y \cdot C_{fy} + z \cdot C_{fz}$

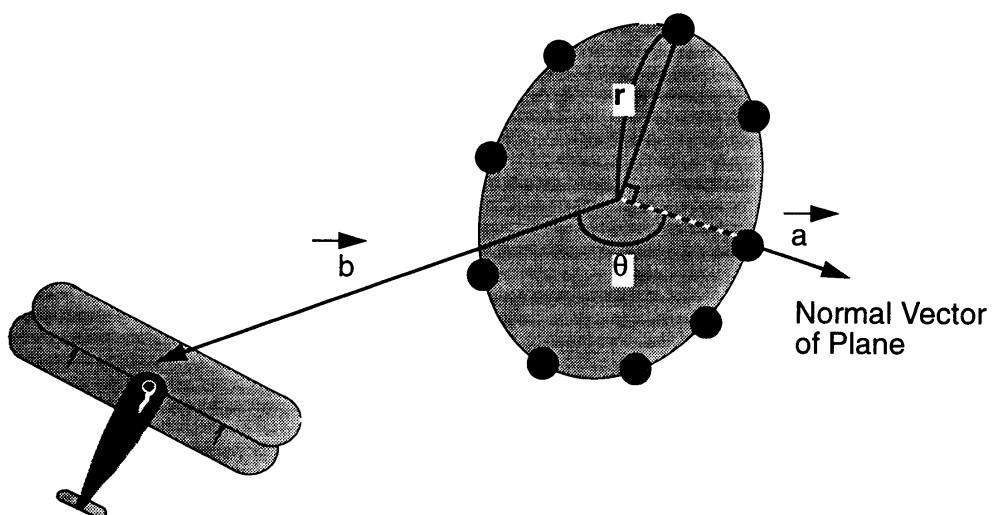


Figure 3-5-21 Calculation of Inner Product with Forward Attitude

Note: As shown below, the first row of the attitude matrix represents global coordinates of a unity vector (1,0,0) in the forward direction in the object coordinate system.

Equation 5-12:

$$S = (X, Y, Z) (1, 0, 0) \begin{bmatrix} M_{fx} & M_{fy} & M_{fz} \\ M_{lx} & M_{ly} & M_{lz} \\ M_{ux} & M_{uy} & M_{uz} \end{bmatrix} = (M_{fx} M_{fy} M_{fz})$$

M is equal to A, B, or C; depending upon selected code.

Number of Process Cycles: Input	1. Command Input	6
	2. x input	15
	3. y input	4
	4. z input	7
Output	1. S output	4

\*Notes: 1. Parameters are input/output via the DR registers.

2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Detection of three-dimensional collision]

This command is used in Pilot Wings to see if the airplane flew through the ring of balls. The sign of the inner product of the forward vector of an object and the vector connecting the object and the airplane changes when the airplane crosses the plane containing the ring (the inner product is zero when the airplane is on the plane). When the sign change occurs, the distance from the center of the ring to the airplane and the radius of the ring can be compared with the RANGE command to see if the airplane was able to fly through the ring.

Vector passes through  
the plane and ring.

$$\vec{a} \cdot \vec{b} = 0$$

Vector passes through  
the inside of ring.

$$|\vec{b}| < r$$

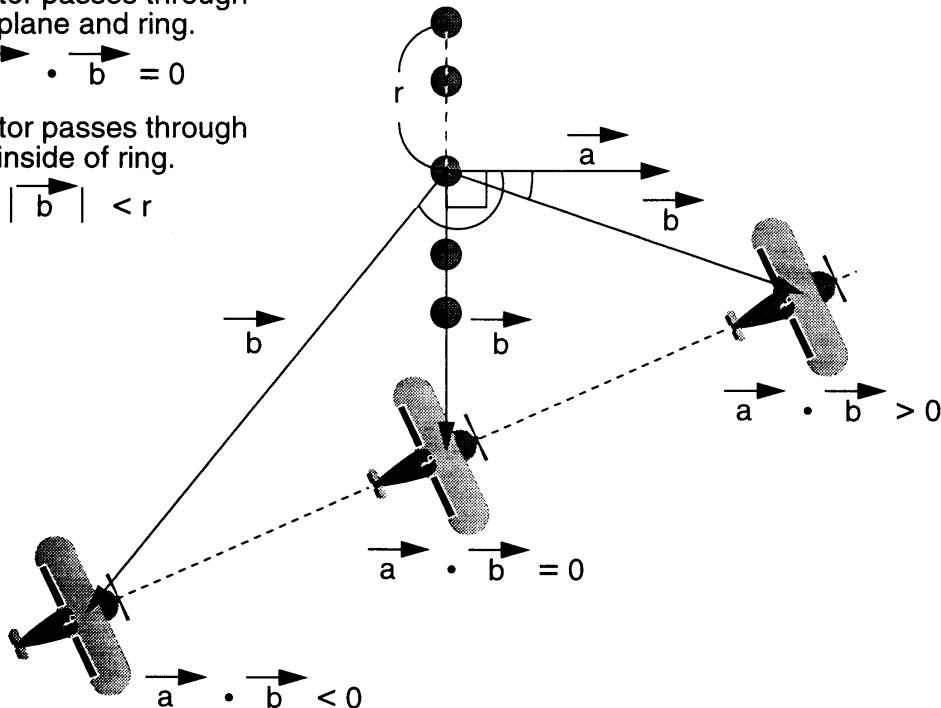


Figure 3-5-22 Position of Aircraft and Vector Code

## 5.6 NEW ANGLE CALCULATION

### 5.6.1 THREE-DIMENSIONAL ANGLE ROTATION

Name:	Gyrate	
Code:	14H	
Parameters:	$\theta_i[A]$	Angle of rotation about the Z axis (+ from the Y axis to the X axis)
	$\phi_i[A]$	Angle of rotation about the X axis (+ from the Z axis to the Y axis)
	$\varphi_i[A]$	Angle of rotation about the Y axis (+ from the X axis to the Z axis)
	$d\theta[A]$	U axis displacement angle. (+ from the L axis to the F axis)
	$d\phi[A]$	F axis displacement angle. (+ from the U axis to the L axis)
	$d\varphi[A]$	L axis displacement angle. (+ from the F axis to the U axis)
Output:	$\theta_o[A]$	Rotational angle about the Z axis.
	$\phi_o[A]$	Rotational angle about the X axis.
	$\varphi_o[A]$	Rotational angle about the Y axis.

Note: F, L, U axes represent the X, Y, Z axes when rotated  $\phi_i$ ,  $\varphi_i$ ,  $\theta_i$  only.

Function: This command determines the attitude angles ( $\theta_o$ ,  $\phi_o$ ,  $\varphi_o$ ) of the body coordinates after the body with the attitude angle ( $\theta_i$ ,  $\phi_i$ ,  $\varphi_i$ ) with respect to the global coordinates are rotated by the minor displacement ( $d\theta$ ,  $d\phi$ ,  $d\varphi$ ). The body axes are rotated about the XYZ axes by ( $\theta_i$ ,  $\phi_i$ ,  $\varphi_i$ ) to obtain the FLU axes. The FLU axes are then rotated by ( $d\theta$ ,  $d\phi$ ,  $d\varphi$ ). This command calculates the angles of the new FLU axes with respect to the XYZ axes. The order of rotation is Y axis, X axis, and Z axis (L, F, and U axis).

Equation 5-13:

$$\begin{aligned}\theta_i + \sec \phi_i (d\theta \cos \varphi_i - d\phi \sin \varphi_i) &= \theta_o \\ \phi_i + (d\theta \sin \varphi_i + d\phi \cos \varphi_i) &= \phi_o \\ \varphi_i - \tan \phi_i (d\theta \cos \varphi_i + d\phi \sin \varphi_i) + d\varphi &= \varphi_o\end{aligned}$$

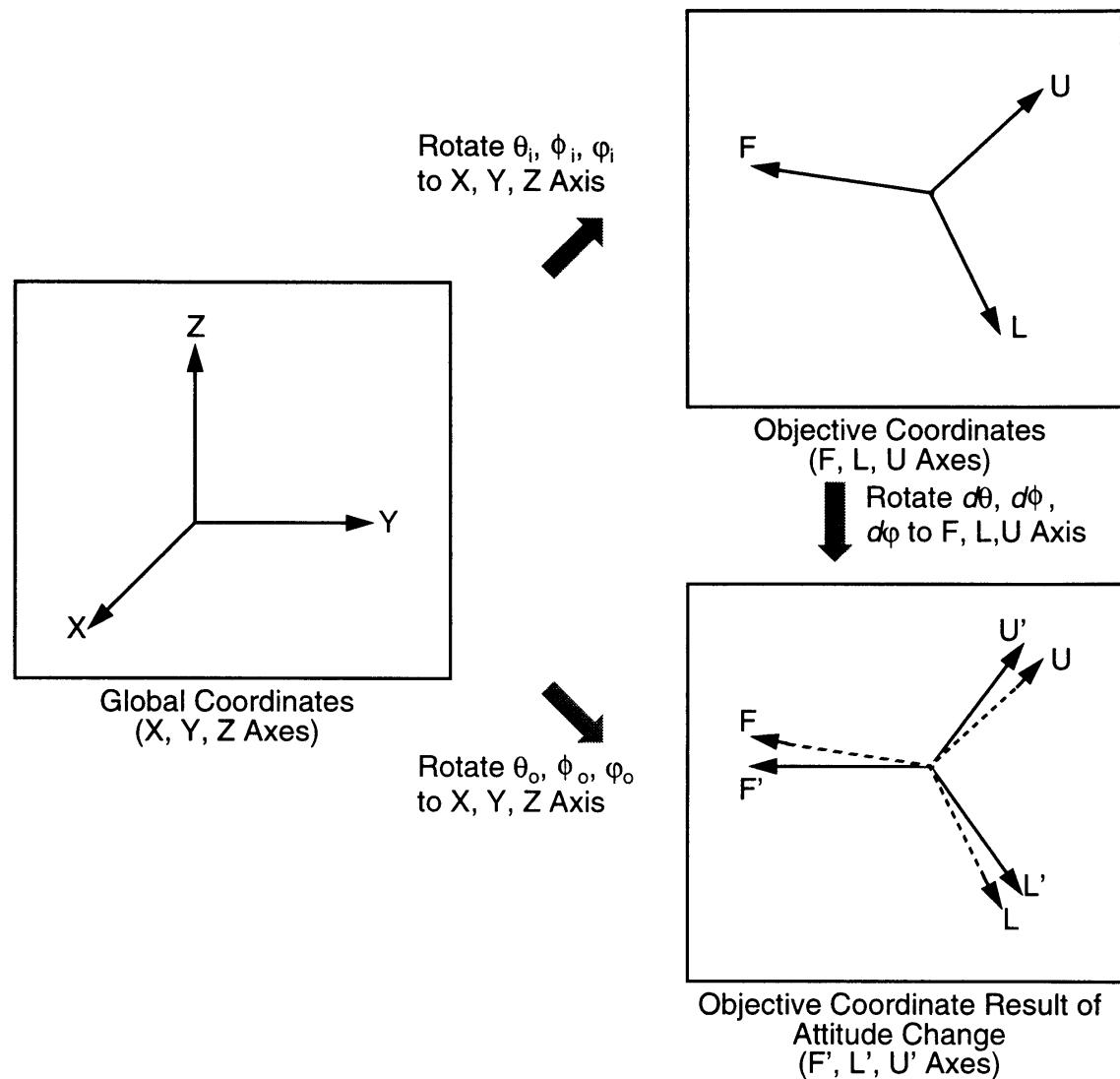


Figure 3-5-23 Calculation of Rotation Angle After Attitude Change

Number of Process Cycles:	Input	1. Command Input	6
	2. $\theta_i$ input	14	
	3. $\phi_i$ input	2	
	4. $\varphi_i$ input	2	
	5. $d\theta$ input	2	
	6. $d\phi$ input	2	
	7. $d\varphi$ input	406	
	Output	1. $\theta_o$ output	2
		2. $\phi_o$ output	4
		3. $\varphi_o$ output	4

\*Notes: 1. Parameters are input/output via the DR registers.

2. Parameters are input/output in the order shown above. The number of cycles is the period until the next parameter can be selected or the results of the calculation can be read.

Example: [Calculation for object attitude (directions) change]

This command is used to calculate the attitude angles of an object that is steadily moving. The command determines the attitude angles with respect to the global coordinates by specifying the angles of change to the current attitude angles. The command may be used continuously to determine changing attitude angles.

## **Chapter 6      Math Functions and Equations**

The following is a summary of the mathematical functions and equations used in this manual.

### **6.1 MULTIPLY**

$$k \times l = M$$

### **6.2 INVERSE**

$$\frac{1}{a \times 2^b} = A \times 2^B$$

### **6.3 TRIANGLE**

$$\begin{aligned} r(\cos\theta) &= C \\ r(\sin\theta) &= S \end{aligned}$$

### **6.4 RADIUS**

$$x^2 + y^2 + z^2 = L$$

### **6.5 RANGE**

$$x^2 + y^2 + z^2 - r^2 = D$$

### **6.6 DISTANCE**

$$\sqrt{x^2 + y^2 + z^2} = R$$

### **6.7 GYRATE**

$$\begin{aligned} \theta_i + \sec\phi_i(d\theta\cos\phi_i - d\phi\sin\phi_i) &= \theta_o \\ \phi_i + (d\theta\sin\phi_i + d\phi\cos\phi_i) &= \phi_o \\ \phi_i - \tan\phi_i(d\theta\cos\phi_i + d\phi\sin\phi_i) + d\phi &= \phi_o \end{aligned}$$

### **6.8 ROTATE**

$$(x, y) \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} = (X, Y)$$

**6.9 POLAR**

$$(x, y, z) \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = (X, Y, Z)$$

**6.10 ATTITUDE**

$$m \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = M$$

**6.11 OBJECTIVE**

$$\frac{1}{2} (x, y, z) M^{-1} = (F, L, U)$$

**6.12 SUBJECTIVE**

$$\frac{1}{2} (f, l, u) M = (X, Y, Z)$$

**6.13 SCALAR**

$$S = (X, Y, Z) (1, 0, 0) \begin{bmatrix} M_{fx} & M_{fy} & M_{fz} \\ M_{lx} & M_{ly} & M_{lz} \\ M_{ux} & M_{uy} & M_{uz} \end{bmatrix} = (M_{fx} M_{fy} M_{fz})$$

## *Chapter 1. The Super NES Super Scope System*

## 1.1 INTRODUCTION TO THE SUPER NES SUPER SCOPE SYSTEM

The Super NES Super Scope is a light sensitive system for use with the Super NES. The Super NES Super Scope was developed to give the Super NES added value and eliminate all of the problems of heretofore existing devices. Features of the Super NES Super Scope are as follows. It is composed of two units; the Super NES Super Scope (light sensitive device) and a receiver/transmitter (Super NES Super Scope-RX).

### **1.1.1 TARGETING**

The Super NES Super Scope detects where the device is aimed, unlike the existing Nintendo Entertainment System device (Zapper), which detects targets. The wireless system utilizes an infra-red beam.

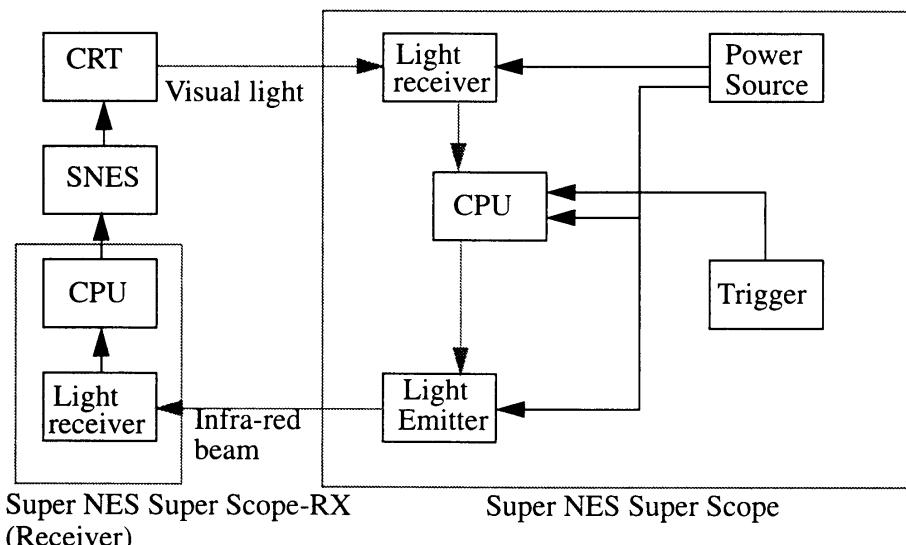


Figure 4-1-1 Signal Flow

The Super NES Super Scope utilizes the external latch function of the Super NES horizontal/vertical counters. The Super NES Super Scope detects CRT scanner timing with a light receiver, and transmits the timing pulse to the Super NES external latch pin to detect the aim location on the CRT. (Same principle as a light pen.)

When the Super NES Super Scope is triggered, the Super NES Super Scope sends a beam of infra-red light to the Super NES and transmits raster timing pulses for a few frames.

When the CPU in the Super NES Super Scope RX recognizes the trigger signal, it opens the gate for an appropriate duration to provide the Super NES with the timing pulses.

### 1.1.2 SUPER NES SUPER SCOPE SIGHT ADJUSTMENT

The most precise alignment of the Super NES Super Scope's sight occurs when the end of its barrel is 3 meters (about 10 feet) away from the television screen. Please refer to the illustration below.

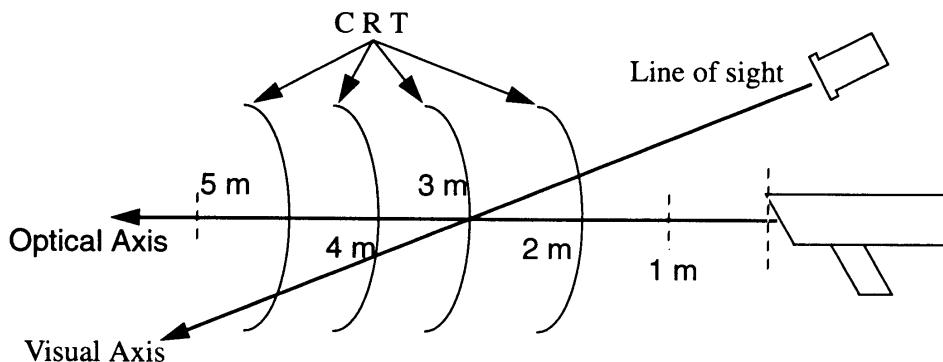
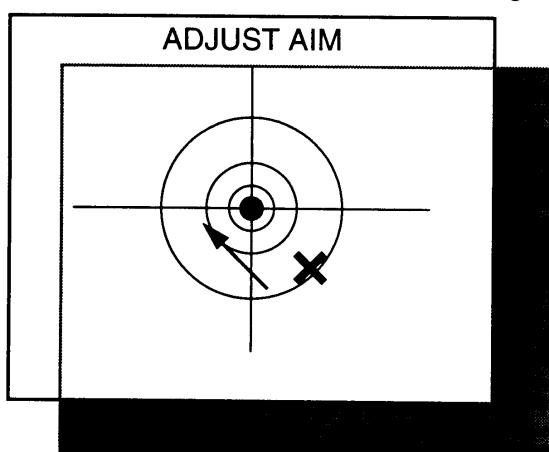


Figure 4-1-2 Optical Alignment

The line of sight (visual axis) virtually "sees" what the lens (optical axis) "sees" when the distance between the television screen and the end of the Super NES Super Scope barrel is 3 meters (10 feet). As demonstrated above, an offset occurs as this range is moved away from 3 meters, in either direction. The function of the "ADJUST AIM" and "TEST AIM" portion of the game is to adjust the optical axis for proper sight alignment through software at the beginning of the game. This adjustment takes into account all electrical delay times. When the adjustment is performed, an insensitive area is created at the edge of the screen. The greater the offset adjustment, the larger this insensitive area becomes.

The following illustration demonstrates an example of the difference between what your eye might see through the Super NES Super Scope and what the lens sees, during the Adjust Aim mode.



- What your eye sees (visual axis)

- ✗ What the lens sees (optical axis)

- Television screen

- Virtual screen (coordinate)

Figure 4-1-3 Virtual Screen Alignment

In order for proper alignment to occur, the virtual screen must be moved in the direction of the arrow. As the virtual screen is moved up and to the left an insensitive (shaded) area is created at the edges of the screen. This shaded area cannot be processed. For this reason, the Super NES Super Scope operation manual recommends that the Super NES Super Scope be used at a range of 3 meters (about 10 feet) from the television for optimum performance. At this distance the insensitive area at the edge of the screen is, for all practical purposes, eliminated.

## 1.2 BASIC SUPER NES SUPER SCOPE SPECIFICATIONS

- Range: 3.28 ~ 16.4ft (with fully charged batteries)
- Resolution: About 1 character (8 dots, in x and y orientation)
- Lens:  $f = 150 \text{ mm}$ ,  $30 \phi$
- Batteries: Six size AA batteries
- Controls:
  - Power switch
  - Single shot/multiple shot selection switch (This is a three-position switch, which is also used as the power switch.)
  - Pause switch (See Note 1)
  - Cursor switch (See Note 2)
  - Trigger switch

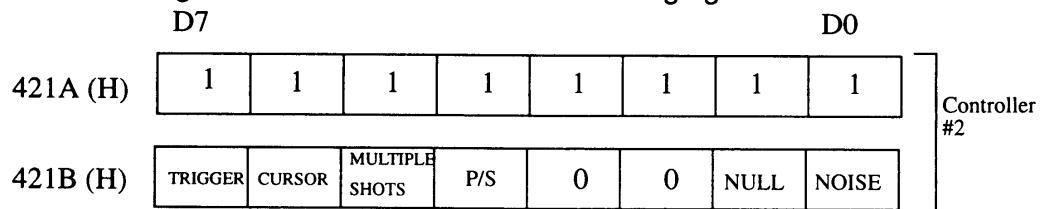
Note 1: This function varies depending on the software, and is used to pause during a game or change screens.

Note 2: The cursor is displayed on the screen while this switch is held down. (The location signal is transmitted continuously.)

## 1.3 SUPER NES PROGRAM ADDRESS

### 1.3.1 REGISTER BIT ASSIGNMENT

The connector for #2 controller serves as the interface between the Super NES Super Scope-RX and the Super NES. Like a standard controller, the Super NES controller can read signals automatically. Address and bit assignments are indicated in the following figures..



421A (H) is always FF (H). D7, 6, 5 and 4 of 421A(H) are unspecified bits. D3, 2, 1, and 0 of 421A (H) and D2 and 3 of 421B (H) are Super NES Super Scope ID codes.

213C (H)      Horizontal counter latch

213D (H)      Vertical counter latch

The horizontal/vertical counter is a hard counter whose latch trigger is set by the Super NES Super Scope.

213F (H)	EXT						
----------	-----	--	--	--	--	--	--

D6 of 213F (H) is the external latch flag.

Figure 4-1-4    Address and Bit Assignments

ITEM	ACTIVITY LEVEL	EXPLANATION
Trigger	High	Indicates that the trigger has been pulled.
Cursor	High	Indicates cursor mode.
Single/multiple	High	Indicates single or multiple shot mode.
Pause	High	Indicates that the pause button is pressed.
Noise	High	Indicates that noise disturbance is impairing operations.
Null	High	Indicates that a valid raster signal could not be found.
H counter	---	The H-position of the hit
V counter	---	The V-position of the hit
EXT latch	High	Indicates that the data was set to the HV counter.

The external latch only can be reset by read. (It cannot be reset by the write command.)

Table 4-1-1 Signal Bit Definitions

## Chapter 2. *Principles of the Super NES Super Scope*

### 2.1 PRINCIPLES OF THE SUPER NES SUPER SCOPE

A comprehensive explanation of the Super NES Super Scope's operation would involve a wide spectrum of topics and require more space than is allowable here. The following is a basic, if cursory, description.

The Super NES projects 60 pictures per second on the television screen. That is, every 1/60 second, a picture frame is projected on the television. But before explaining how the picture is drawn, it is necessary to describe the Braun tube or CRT in the television set.

A fluorescent material (phosphor coating) is fused to the inside of the Braun tube's glass screen. Light is emitted when electrons bombard this fluorescent material.

The inside of the Braun tube resembles a funnel (refer to the figure below) and an "electron gun" is located at the rear of the tube. (This is the section which extends from the back of a television.)

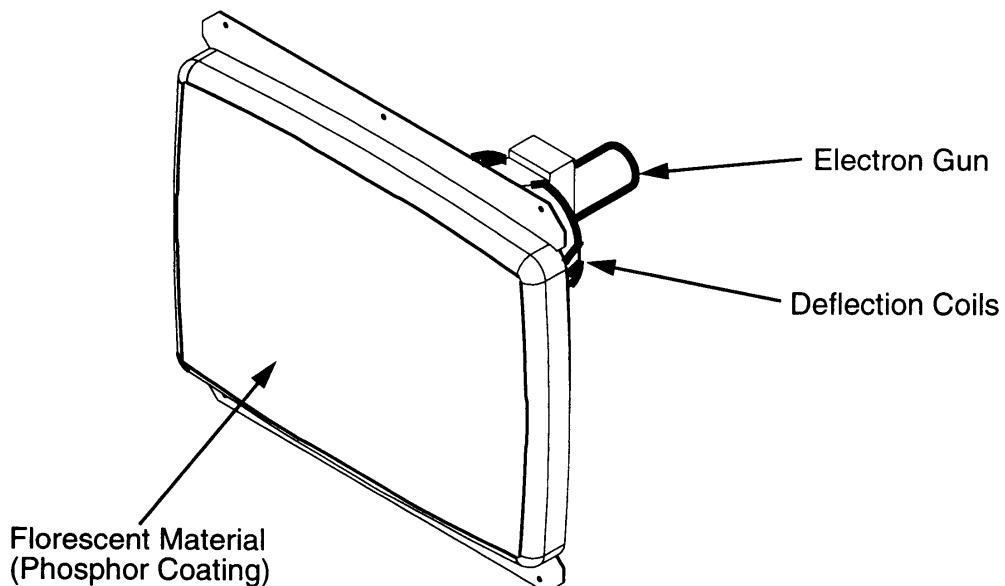


Figure 4-2-1 Picture Tube

The electron gun discharges a beam of electrons toward the screen. This, by itself, would only light a fixed spot where the electron beam hit the screen; however, deflecting coils are attached to the base of the tube and a signal is transmitted to the coils to drive the electron beam in the direction desired.

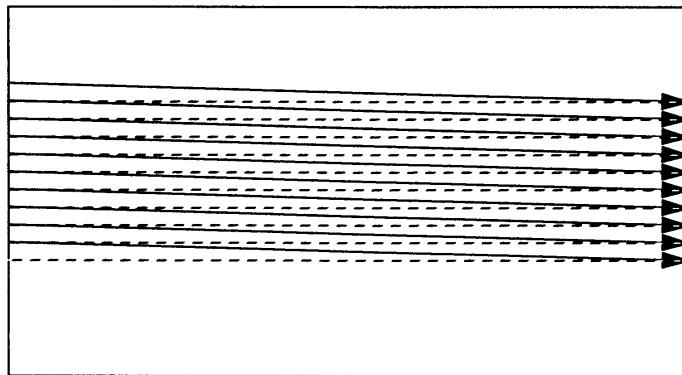


Figure 4-2-2 Scanning

Using this technique, the electron beam scans from left to right beginning at the top left of the screen and moving successively down the screen, as shown in the above figure. Each horizontal line formed by the scan is called a scan line or a raster. Light and dark areas are created by varying the intensity of the electron beam as it scans across the fluorescent material. This is how each picture is drawn.

The Super NES contains a PPU (picture processing unit), for controlling the picture projected on the screen. Inside the PPU is a "raster counter" (or "HV counter") with a register which holds the X and Y coordinates of the electron beam in the Braun tube as it scans.

When the Super NES Super Scope is aimed at the screen, a small area on the screen is seen by the Super NES Super Scope.

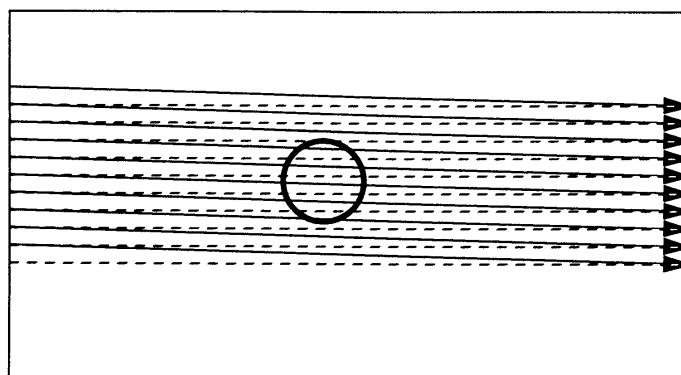


Figure 4-2-3 Area Seen by Super NES Super Scope

As shown in the previous figure, the instant the electron beam scans across the area seen by the Super NES Super Scope, it sends a signal to the Super NES. The Super NES registers the value of the PPU raster counter using this timing signal. With this data, the Super NES can detect the point on the screen where the Super NES Super Scope is aimed.

## 2.2 SUPER NES SUPER SCOPE PROGRAMMING

We assume that most readers are involved in programming Super NES Super Scope games..

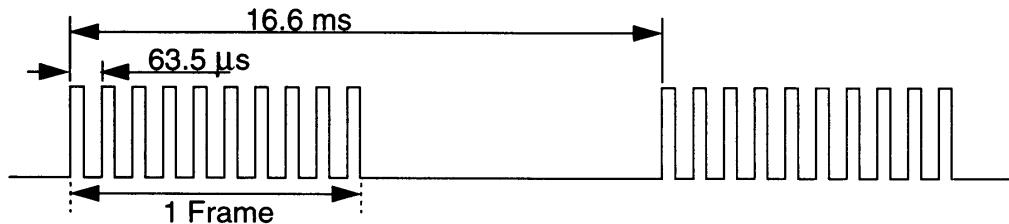


Figure 4-2-4 Vertical Positioning

The above figure depicts the output of the Super NES Super Scope's light reception amplifier under these conditions. Each of the pulses represents a raster in the Braun tube. The Super NES Super Scope system picks a pulse and transmits it to the Super NES raster counter. Pulse selection determines the vertical location on the screen by the raster count. This is done under a fixed set of conditions by the Super NES Super Scope's internal CPU.

The horizontal position is determined by the timing of pulses with respect to the Super NES Control Deck's horizontal synchronization signal. (Refer to the figure below.)

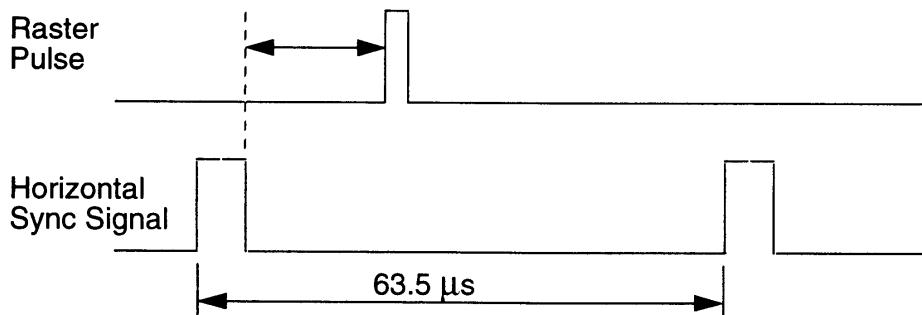


Figure 4-2-5 Horizontal Positioning

The time corresponding to one dot on the screen is an amazing 180 nsec. This processing speed cannot be achieved by most micro-computers, and in the Super NES Super Scope system, the raster pulse is not processed directly by the CPU. Signal transmission and selection is done by opening and closing the raster gate, controlled by the CPU, and is depicted in the block diagram in Chapter 1. An area of caution for Super NES Super Scope programs is that Super NES Super Scope operations are not synchronized with the Super NES. The timing relationship between the Super NES Super Scope, the Super NES screen scan, and the program, described later, should be kept in mind when programming.

## 2.3 THE SUPER NES HORIZONTAL/VERTICAL COUNTER

The horizontal/vertical counter of the Super NES plays a critical role in the Super NES Super Scope system, yet is not described in much detail in the Super NES programming manual or other documents. For this reason, we will present an overview here.

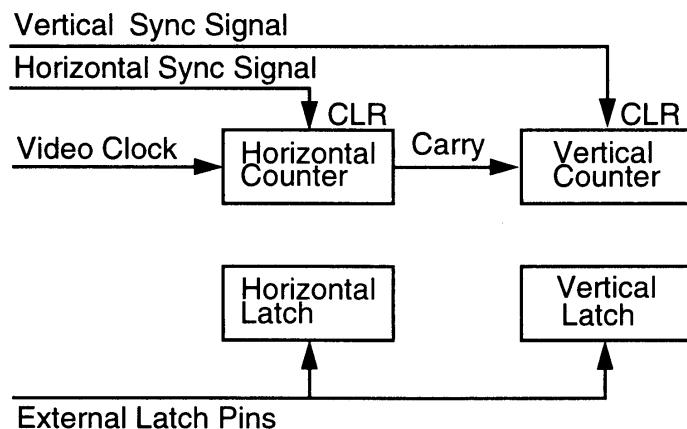


Figure 4-2-6     Horizontal/Vertical Counter

The horizontal counter value corresponds to the horizontal location of the raster and the vertical counter value corresponds to the vertical location of the raster.

These values can be stored by sending a pulse to the external latch pin. The Super NES software then reads this, and is able to detect the location on the screen which corresponds to the external latch pulse.

In the Super NES Control Deck, a flag is set when the horizontal/vertical latch is set. This flag does not operate in synchronization with the programming flow, and interrupts are not supported by the Super NES Control Deck. Hence, programming precautions should be taken.

## **Chapter 3. Super NES Super Scope Functional Operation**

### **3.1 SUPER NES SUPER SCOPE CPU**

The Super NES Super Scope CPU is a one-chip CPU for processing Super NES Super Scope key input (trigger, cursor, etc.), data pulse generation, and transmission of screen timing signals.

#### **3.1.1 KEYS**

Trigger	Trigger
Cursor	Continuous input
Pause	One-shot input
Multiple/single shot	Switches between continuous trigger input and one-shot input

#### **3.1.2 KEY PRIORITY**

Priority is given in the order of the trigger, cursor and pause keys. Two types of trigger codes are generated by switching between the multiple and single shot modes.

#### **3.1.3 KEY RECOGNITION**

A key is recognized as "on" after it is on for 1 msec or more, and "off" after 20 msec or more.

#### **3.1.4 SIMULTANEOUS KEY INPUT**

Only the trigger and cursor keys can be input at the same time. Other key combinations are not recognized.

### **3.2 SUPER NES SUPER SCOPE BLOCK DIAGRAM**

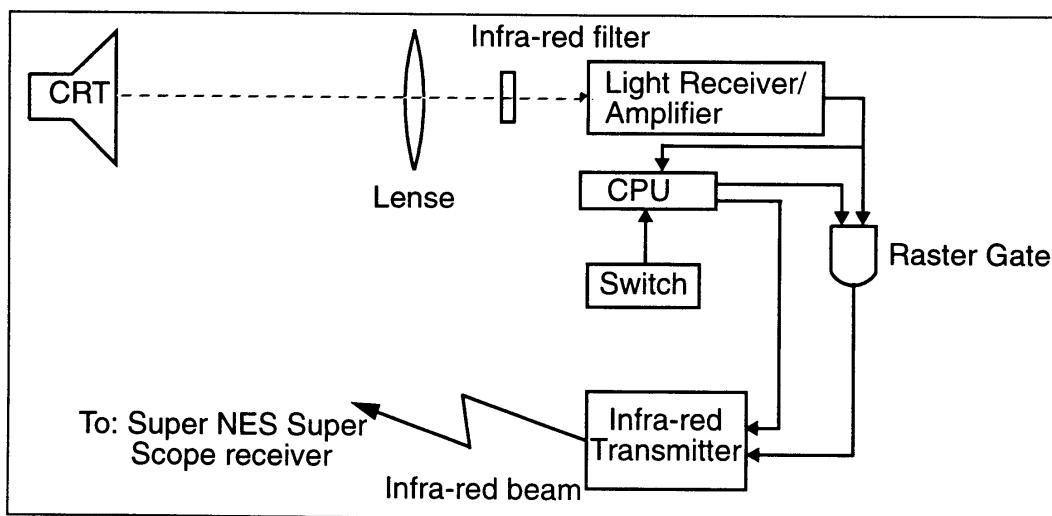


Figure 4-3-1 Super NES Super Scope Block Diagram

### 3.2.1 LIGHT RECEIVER/AMPLIFIER

The light receiver/amplifier receives the light signal from the CRT, converts it to pulses, and transmits the pulses to the Super NES Super Scope CPU. It consists of a pin photo-diode H-amp, and an M-amp for signal amplification and pulse conversion.

### 3.2.2 SUPER NES SUPER SCOPE CPU (SM595)

The Super NES Super Scope CPU reads the Super NES Super Scope, generates the corresponding code, controls the raster gate, and sends the raster signal to the Super NES Super Scope receiver.

### 3.2.3 LIGHT OUTPUT

This converts the pulse generated by the CPU into an infra-red beam. It consists of an infra-red LED and its driver.

## 3.3 SUPER NES SUPER SCOPE FLOW DIAGRAM

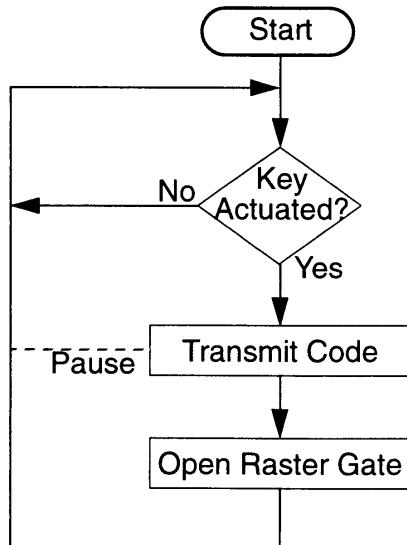


Figure 4-3-2 Super NES Super Scope Flow Diagram

The Super NES Super Scope does not process the raster signal.

## 3.4 INFRA-RED DATA TRANSMISSION FORMAT

### 3.4.1 Overview

The Super NES Super Scope infra-red signal is composed of two segments. The first segment contains a digital code, which defines the single-shot trigger, multiple-shot trigger, cursor, and pause. The second segment is the raster segment. The Super NES Super Scope CPU opens the raster gate and connects the light receiver/amplifier and light output. The raster signal is output from the CRT for a set duration of time.

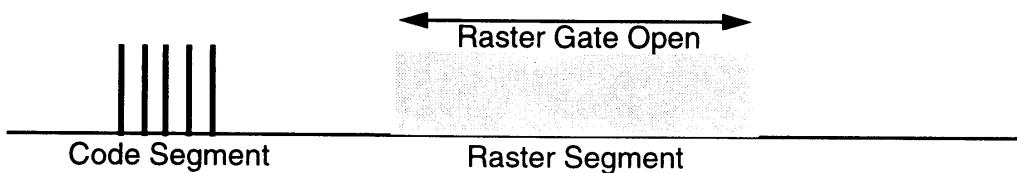


Figure 4-3-3 Raster Signal

### 3.4.2 DESCRIPTION OF ONE BYTE

The Super NES Super Scope system can generate four types of codes based on the status of the keys. One byte is defined as follows.

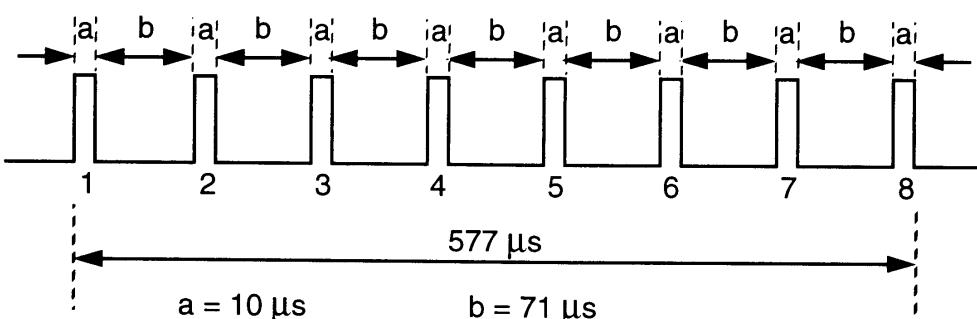


Figure 4-3-4 Definition of one byte

One byte is composed of a block of eight pulses as shown above.

The code is generated by combining five one-byte blocks as shown below.

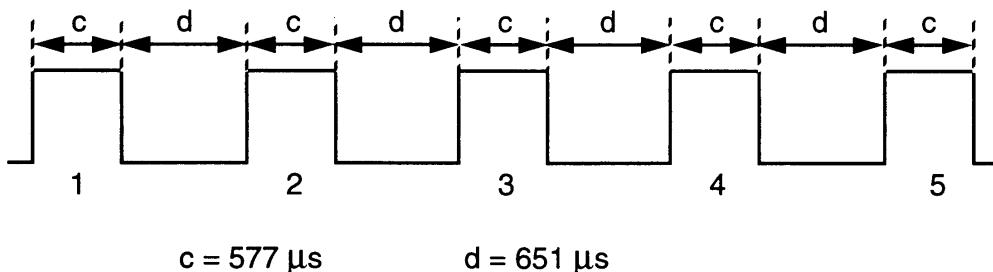


Figure 4-3-5 Output Signal Code

Byte 1 is the switch byte.

Byte 5 is the end byte.

Bits 2, 3 and 4 are data bits

### 3.4.3 COMMUNICATION CODES

Four codes are defined as follows.

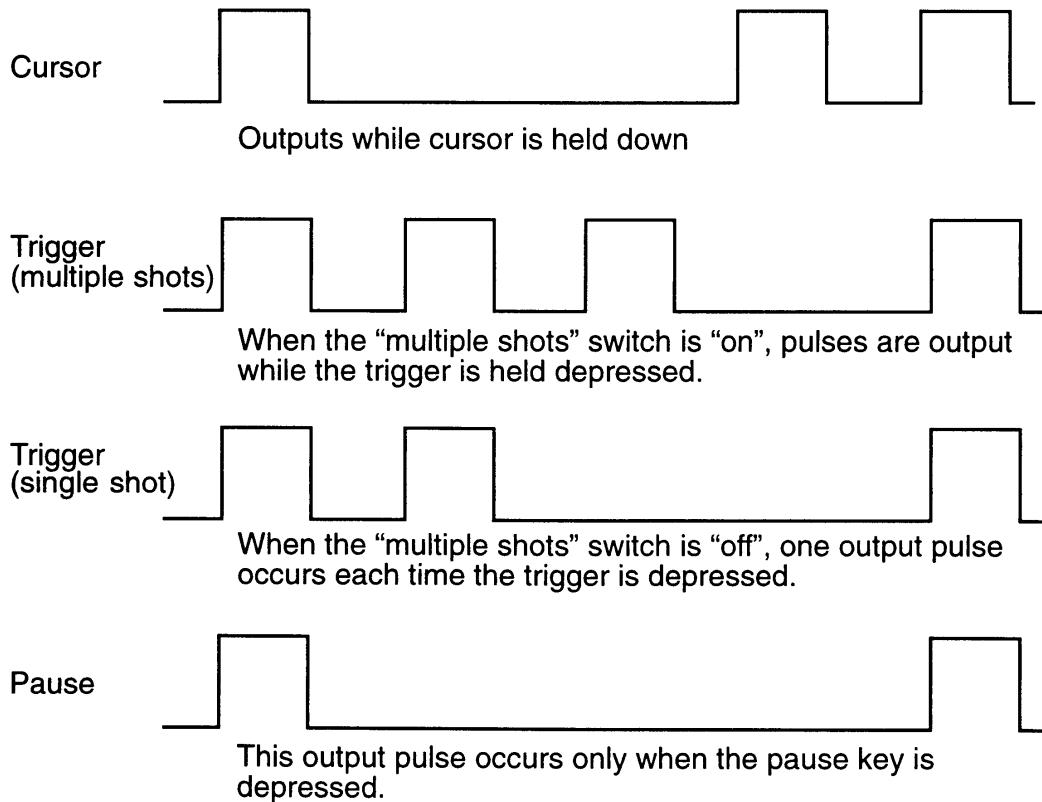
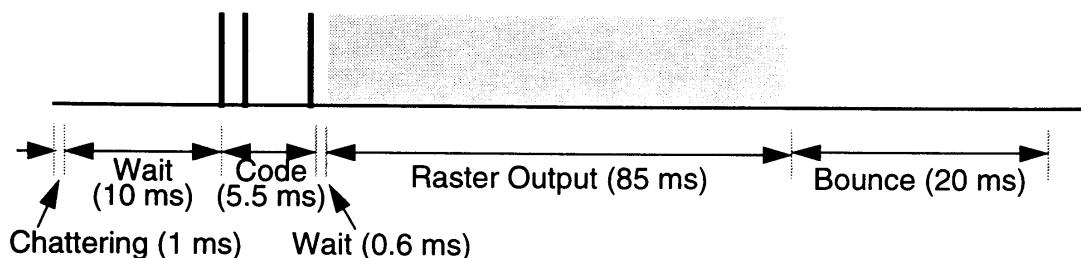


Figure 4-3-6 Definitions of codes.

### 3.4.4 RASTER SIGNAL TRANSMISSION TIMING

a. Trigger (single shot)



b. Trigger (multiple shots)

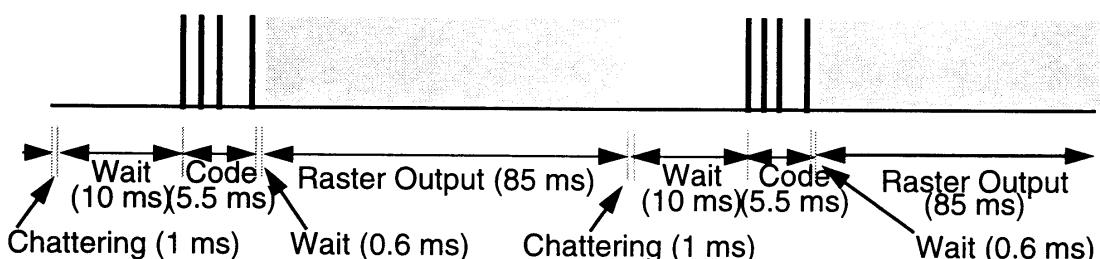
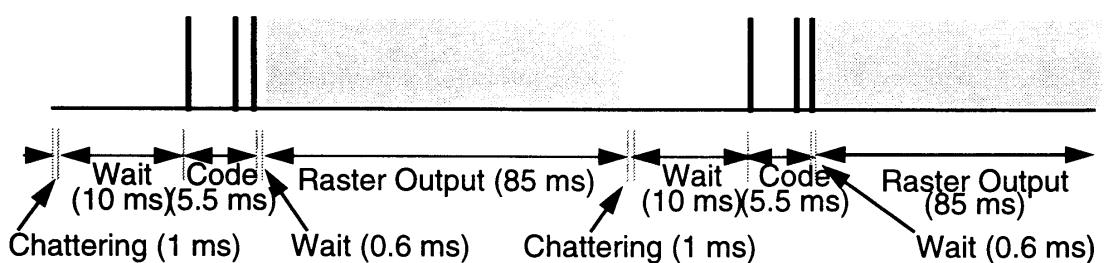


Figure 4-3-7 Raster Signal Transmission Timing, part 1

The cycle above is repeated while the trigger is held down. When the trigger is released, a single shot cycle occurs as the final cycle

c. Cursor



d. Pause

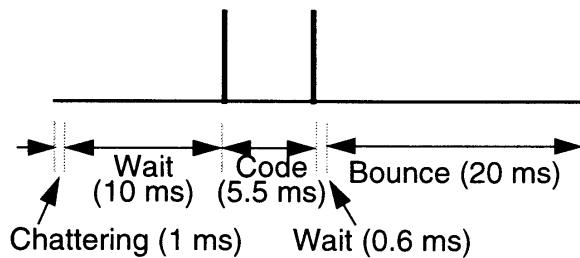


Figure 4-3-7 Raster Signal Transmission Timing, part 2

The raster gate opens during raster output and the raster pulses are transmitted to the Super NES Super Scope receiver. The raster pulse timing is not defined. The Super NES Super Scope and Super NES Control Deck are not synchronous.

## Chapter 4. Super NES Super Scope Receiver Functions

### 4.1 SUPER NES SUPER SCOPE RECEIVER BLOCK DIAGRAM

The Super NES Super Scope receiver first receives the infra-red signal from the Super NES Super Scope, and transmits the key switches and screen timing signals to Super NES Control Deck.

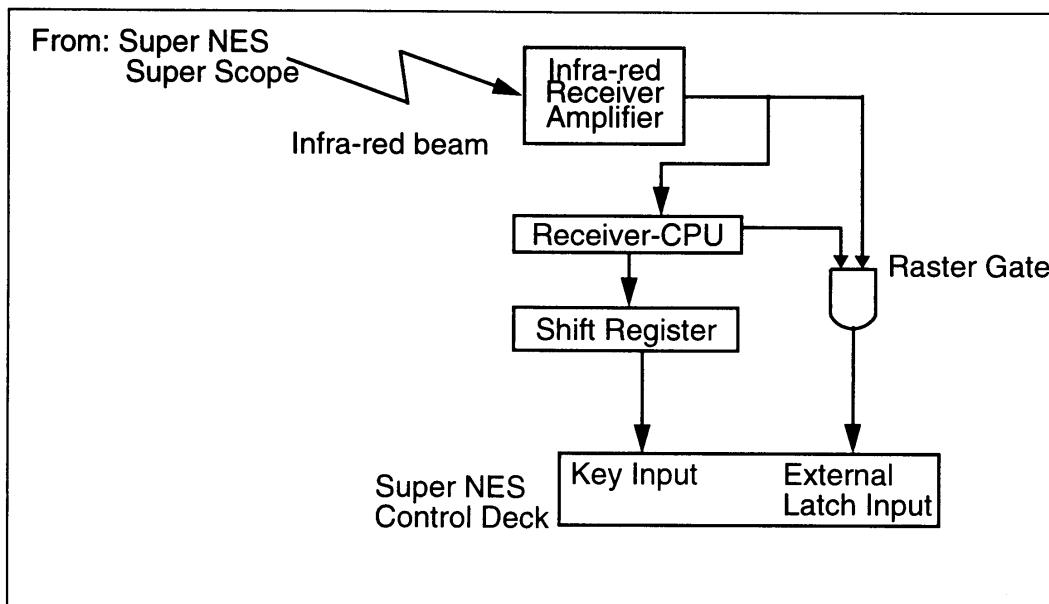


Figure 4-4-1 Receiver Block Diagram

#### 4.1.1 INFRA-RED LIGHT RECEIVER/AMPLIFIER

Receives the infra-red signal from the Super NES Super Scope, converts it to pulses, and transmits the pulses to the Super NES Super Scope receiver CPU. It consists of a pin photo diode H-amp and an M-amp for signal amplification and pulse conversion.

#### 4.1.2 SUPER NES SUPER SCOPE RECEIVER CPU

The CPU analyzes the code signal from the Super NES Super Scope, controls the shift register flag and raster gate, and sends the raster pulses to the Super NES external latch pin.

#### 4.1.3 SHIFT REGISTER

This is the interface between the Super NES Super Scope receiver CPU and the Super NES Control Deck, and is similar to the type of interface found in a controller.

#### 4.1.4 OPERATIONS FLOW DIAGRAM

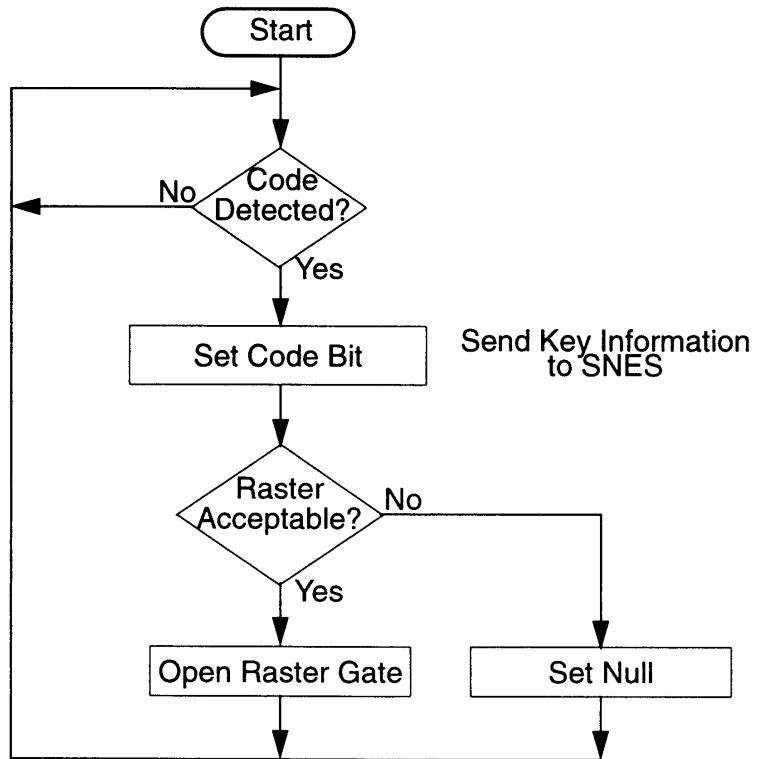


Figure 4-4-2 Operation Flow Diagram

In addition, a pulse check is performed during code detection for noise detection.

## 4.2 SUPER NES SUPER SCOPE RECEIVER INTERFACE

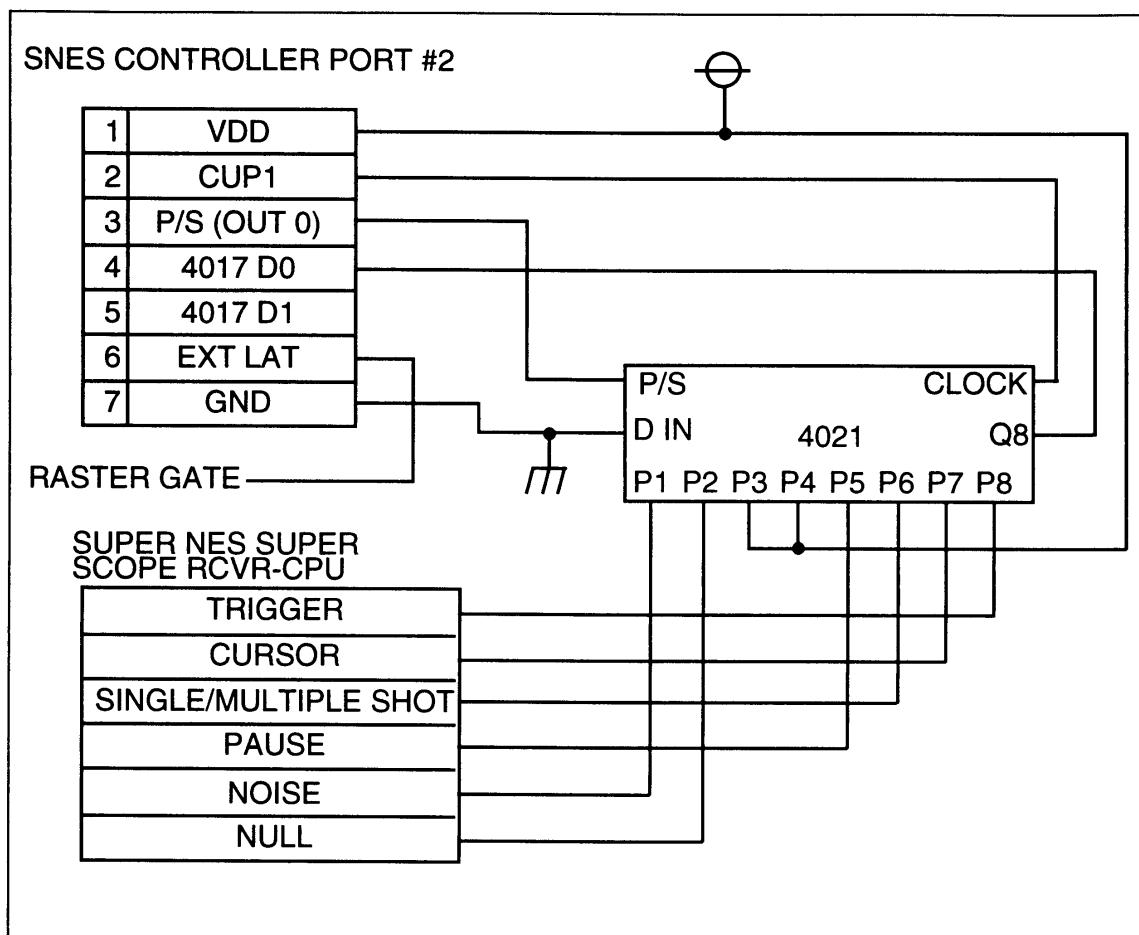


Figure 4-4-3 Receiver Interface Schematic

## 4.3 CODE PULSE DETECTION

### 4.3.1 ONE BIT CODE DETECTION

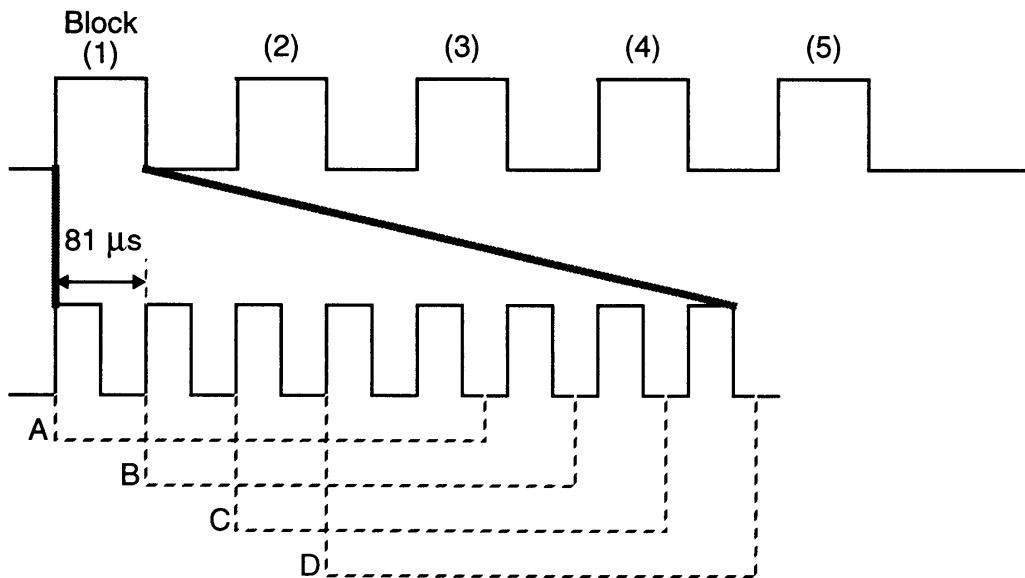


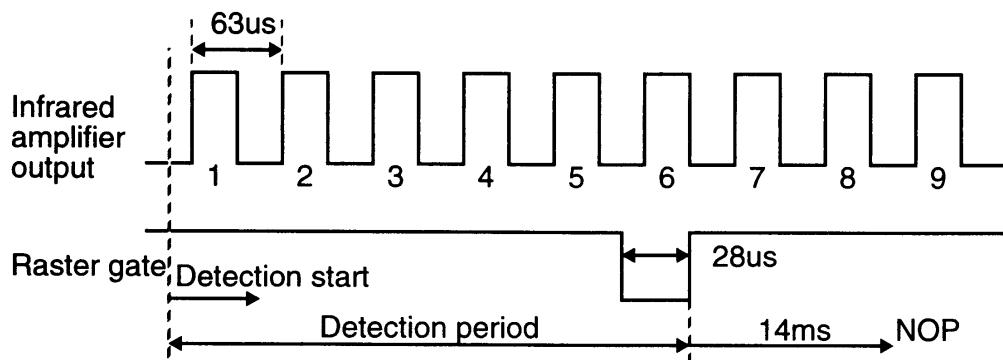
Figure 4-4-4 One Bit Code Detection

A block is good if five - 81  $\mu$ sec pulses are detected in succession in any of the ranges, A, B, C and D shown above.

A noise flag is set if the "high" level is detected 36-39  $\mu$ sec after the rising edge of a pulse is detected.

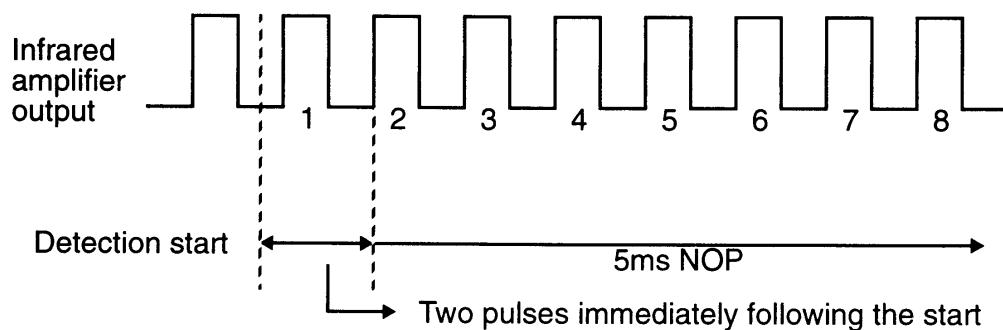
### 4.3.2 RASTER PULSE DETECTION

The start of detection and input of raster pulses do not coincide in the example below.



The latch gate opens when pulses 1~6 are detected with the precise cycle time.

In the next example, the start of detection and input of raster pulses coincide.



If two raster pulses are detected immediately following the start of raster pulse detection, it is determined that the detection cycle occurred at the same time as raster pulse input. In this case, the receiver CPU would perform time calculations for 5 msec. In this frame, the CPU does not attempt to output the raster signal.

An error occurs when a raster exceeds 5 msec. (With the existing optical system, this may happen 1.64 feet away from a 14-inch television screen.)

## 4.4 FUNCTIONAL DESCRIPTION

### 4.4.1 CURSOR MODE

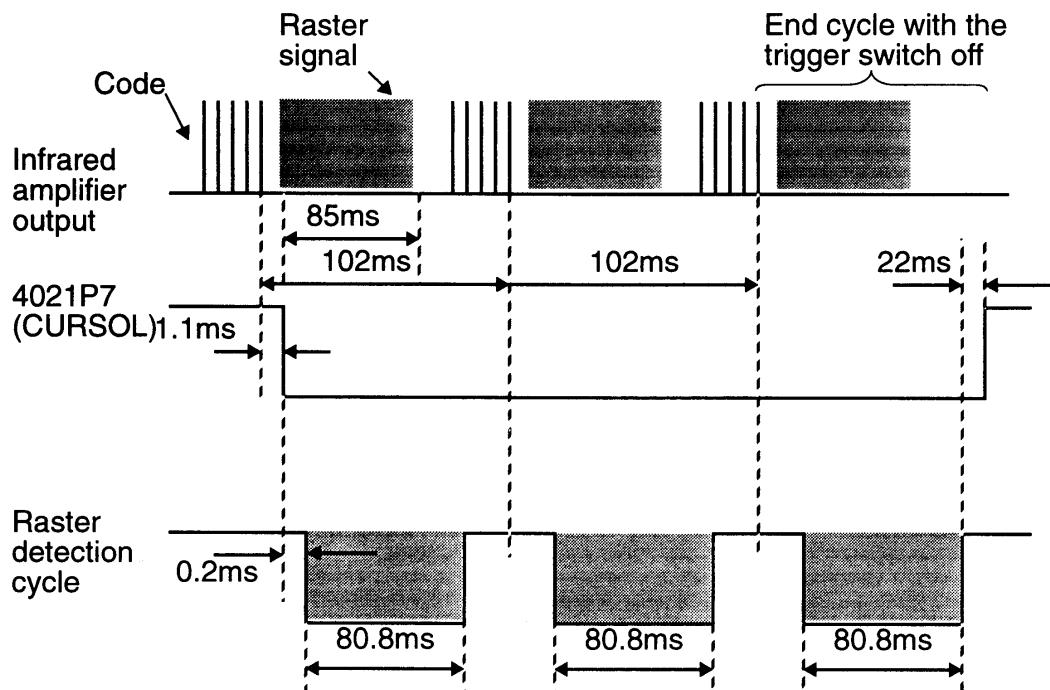


Figure 4-4-5 Cursor Mode Raster Detection Cycle

In the cursor mode, the cursor is displayed continuously on the screen. To accomplish this, raster pulses are transmitted for five frames (85 msec) after code data is sent from the Super NES Super Scope.

#### 4.4.2 TRIGGER MODE (SINGLE SHOT)

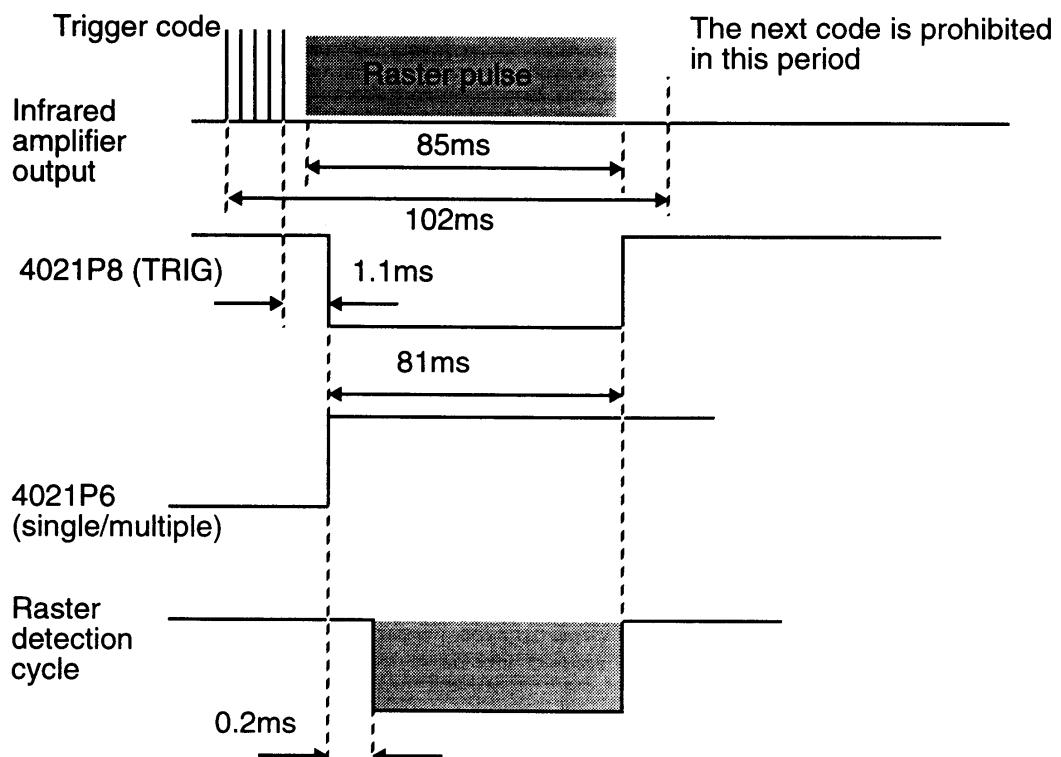


Figure 4-4-6 Trigger Mode, Single Shot

#### 4.4.3 TRIGGER MODE (MULTIPLE SHOTS)

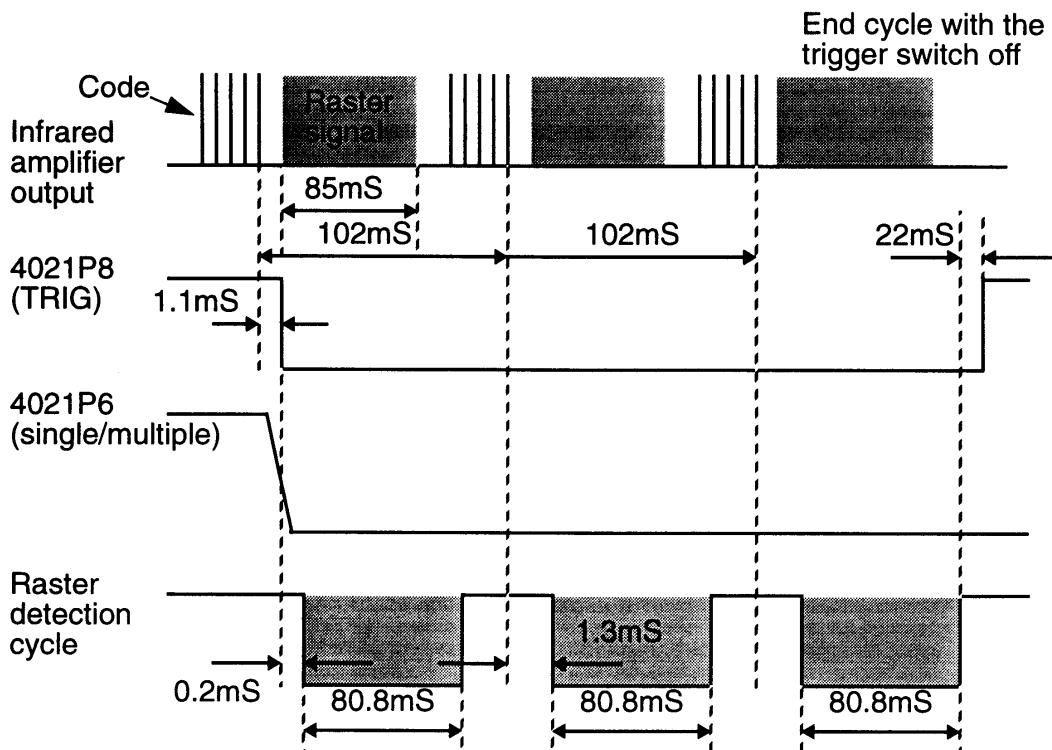


Figure 4-4-7 Trigger Mode, Multiple Shots

#### 4.4.4 NOISE FLAG

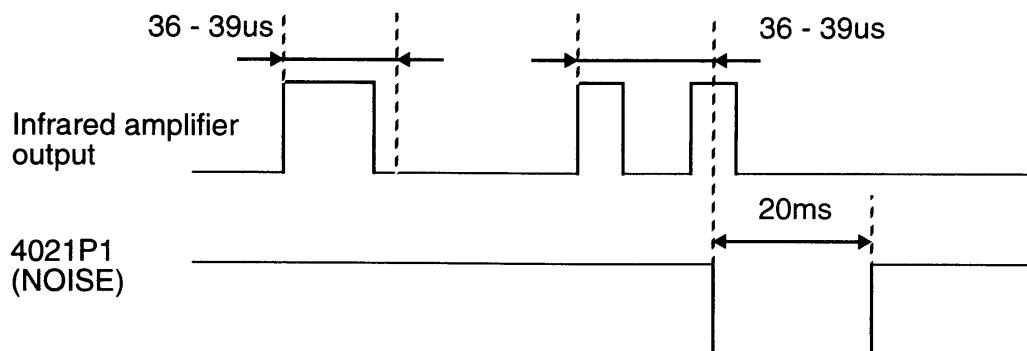


Figure 4-4-8 Noise Flag

Under the timing shown above, the noise flag is set when a pulse with a cycle time different from that used by the Super NES Super Scope system is detected while waiting for the code.

#### 4.4.5 NULL BIT

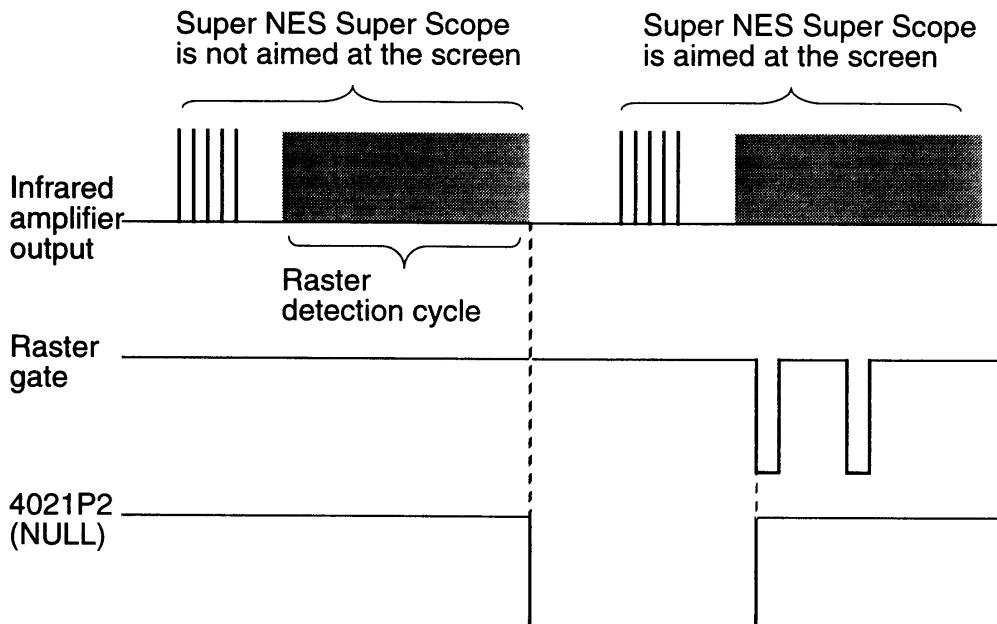


Figure 4-4-9 Null Bit

The null flag is set if a valid raster signal is not detected during a raster detection cycle. It is reset if a valid raster signal is detected in a subsequent cycle and the raster gate is opened.

#### 4.4.6 PAUSE BIT

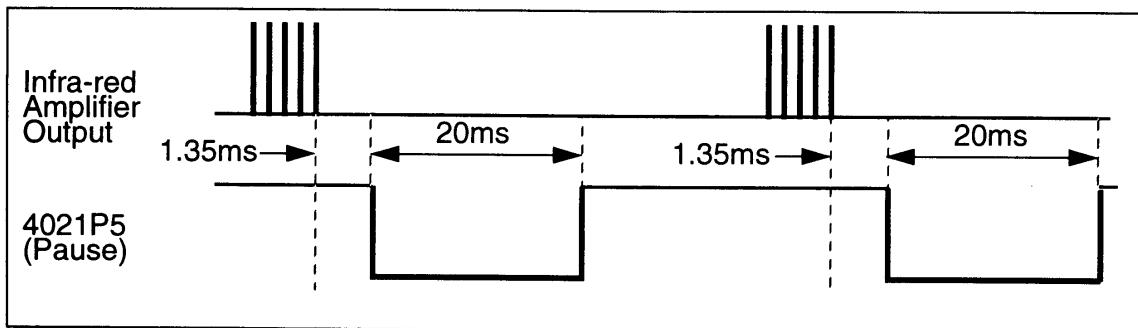


Figure 4-4-10 Pause Bit

This flag is set when a pause code is received from the Super NES Super Scope.

#### 4.4.7 CURSOR + TRIGGER CYCLE

##### 4.4.7.1 TRIGGER (SINGLE SHOT)

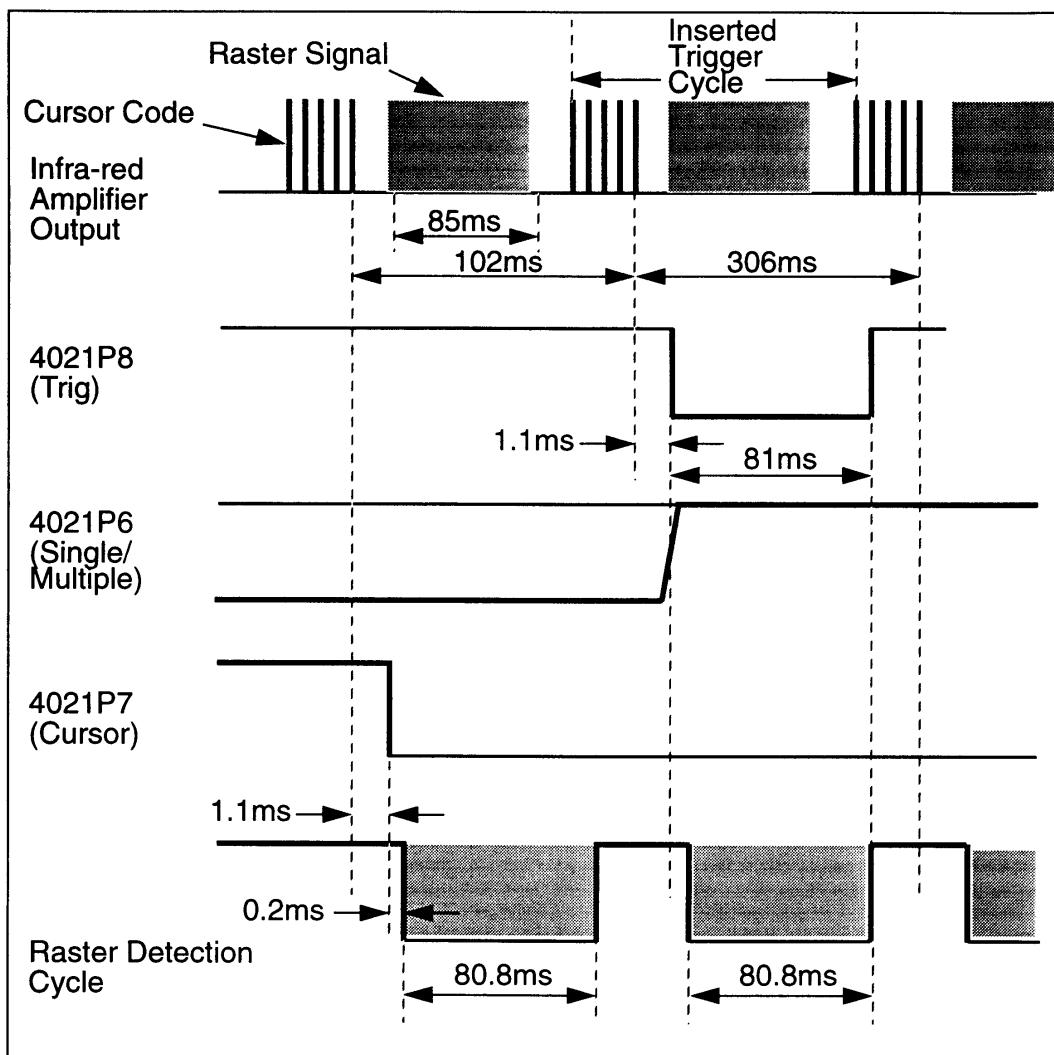


Figure 4-4-11 Trigger, Single Shot

## 4.4.7.2 TRIGGER (MULTIPLE SHOTS)

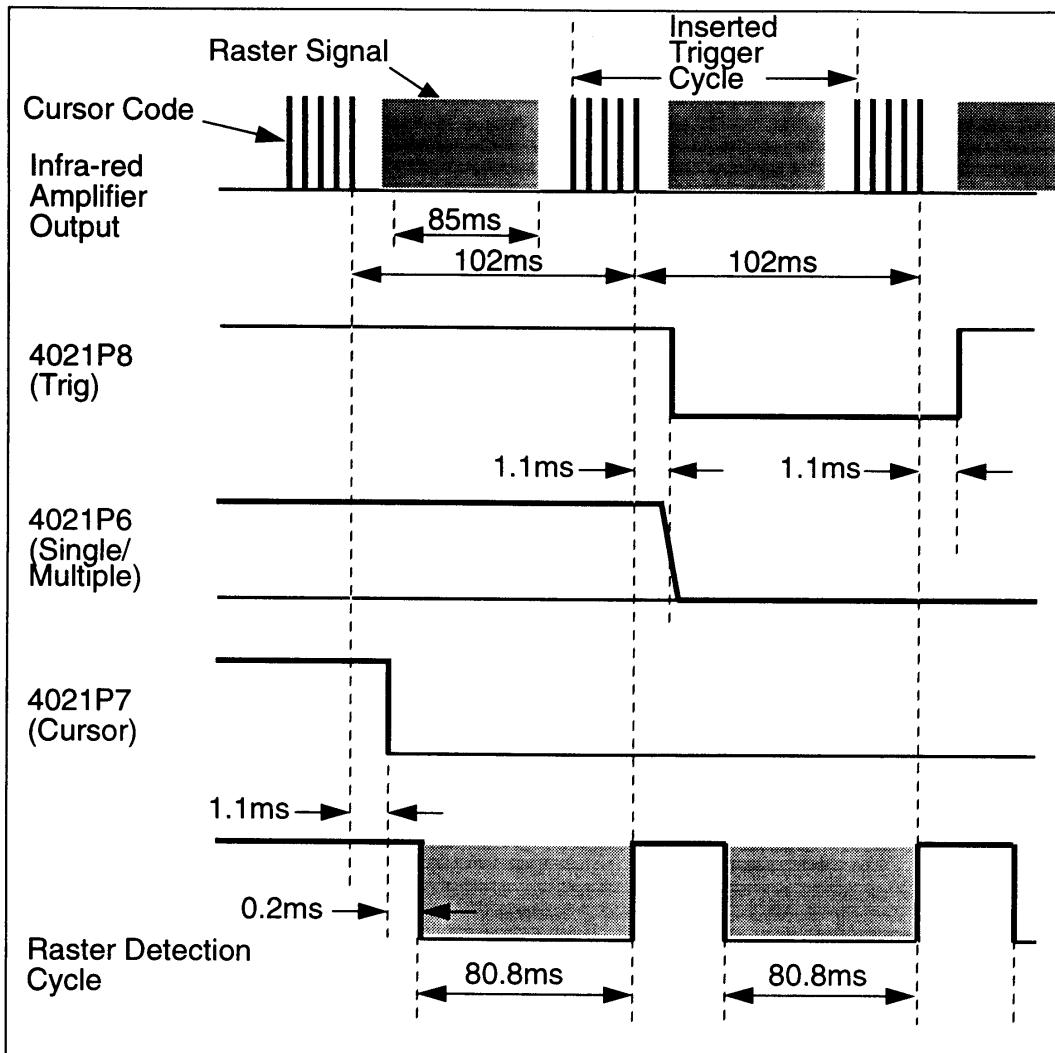


Figure 4-4-12 Trigger, Multiple Shots

Same as the cursor mode except the trigger flag and single/multiple shot flags vary.

Note: In this section, the timing charts for each 4021Px flag (trigger, cursor, single/multiple shot, etc.) are expressed in negative logic (active low); however, these are positive logic (active high) in the Super NES program.

## *Chapter 5. Graphics*

### 5.1 LIMITATIONS ON GRAPHICS

Because Super NES Super Scope operations are based on the detection of rasters on a television screen, the screen used must have a minimum level of brightness.

Of particular concern is the fact that the Super NES Super Scope is not sensitive to the color red. This is due to differences in the afterglow characteristics of the fluorescent materials used in the Braun tube for the three colors, red, green and blue. The period of fluorescence for red is relatively longer, as shown in the table below, and hence the change in the volume of light over time is smaller (16 KHz horizontal synchronization frequency component), and raster timing is more difficult to detect.

Red	1.2 msec
Green	300 $\mu$ sec
Blue	250 $\mu$ sec

The minimum level of brightness which the Super NES Super Scope can detect is very difficult to predict due to the various factors involved (television type, year of make, screen adjustment, etc.). An Optical Color Sensitivity Chart is provided on the following page for programming reference.

If you wish to detect the location on the screen in one-dot increments or draw a dark picture, such as of outer space, you may wish to insert a bright single-color screen for one frame.

When accuracy is important, be careful of the variation in luminosity across the screen. On a 14-inch screen there is about 1.5 times variation in luminosity between the center and the perimeter of the screen. When the screen is dark, the Super NES Super Scope signal may be delayed, and the location detected will be shifted to the right. This may be corrected in the program or the Super NES color operation function may be used to correct for luminosity.

**NOTE: Nintendo's products and game programs, designed in accordance with these specifications, are subject to claims of patent and patent pending owned and/or licensed by Nintendo exclusively for the benefit of Nintendo and its authorized licensees. Nintendo does not license such rights for any other use or purpose. Nintendo does not warrant or represent against claims of patent infringement by third parties.**

## 5.2 SUPER NES SUPER SCOPE OPTICAL COLOR SENSITIVITY CHART

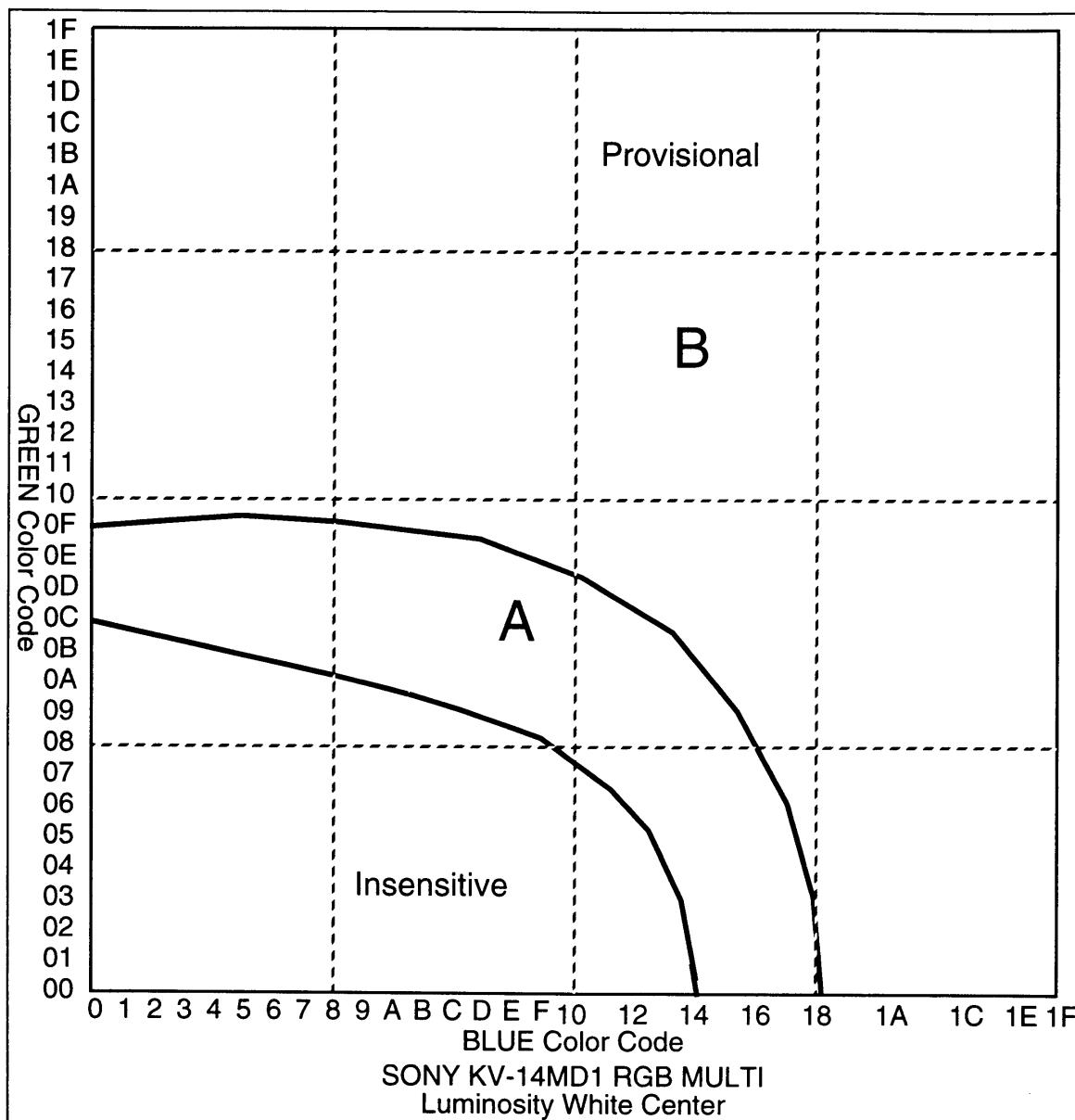


Figure 4-5-1 Optical Color Sensitivity Chart

The Super NES Super Scope is not sensitive to red at all. The error increases in area "A", above. There is no problem in area "B". This chart is based on the measurement of a single color on the screen and should be used as a reference only, since the screen pattern does introduce variations.

## ***Chapter 6. Super NES Mouse Specifications***

### **6.1 INTRODUCTION TO SUPER NES MOUSE**

The Super NES Mouse is a special purpose serial mouse. Displacement data detected in the mouse is processed on a custom chip. Data is input to the Super NES console via the 7 pin connector as key data. The mouse does not burden the program in any way. The programmer need only call the standard basic input/output system (BIOS) subroutine for processing mouse data. Thus, the Super NES Mouse is substituted for the standard controller. The mouse has three tracking speeds. A speed selection switch inside the mouse can be controlled by the following two methods.

- Game software which allows user selection
- Game software which provides a fixed speed

## 6.2 SUPER NES MOUSE DATA FLOW

Super NES Mouse data is transmitted to the Super NES control deck in a serial input format, like the standard controller. A 32 bit data string is transmitted; however, only 24 bits are used. The figure below shows a valid data string transmitted to the Super NES control deck, from the Super NES Mouse. Signals from the Super NES Mouse are transmitted in negative logic and converted to positive logic data strings by the input inversion buffer in the Super NES control deck. Note that all the data shown has already been loaded into the Super NES control deck.

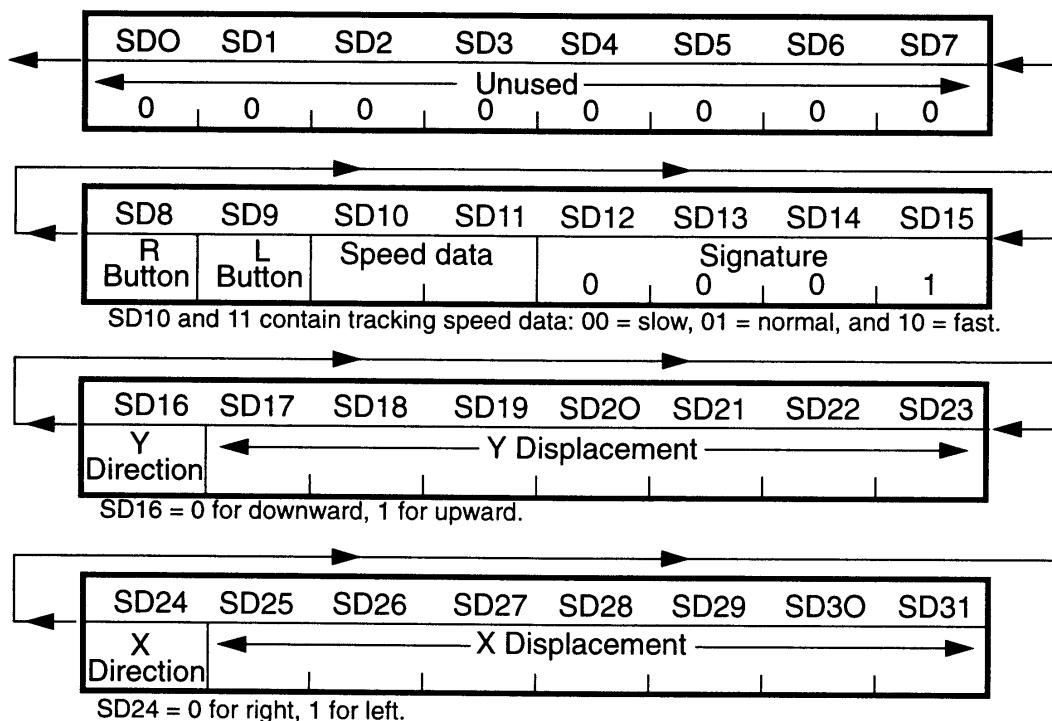


Figure 4-6-1 Valid Super NES Mouse Data String

### 6.2.1 DATA TRANSMISSION

The Super NES Mouse has four 8-bit shift registers. These registers are serially connected as indicated by the arrows in the figure on the previous page. The Super NES Mouse transmits 32 bits of data to the Super NES control deck following each OUT0 pulse, using CUP0 as a clock pulse. The Super NES control deck transmits this OUT0 pulse at a fixed interval. The sequence is from SD0 to SD31.

### 6.2.2 READ METHODS

For details concerning the manner in which the Super NES control deck reads serial controller data, refer to "Joy Controller" in the "Software" section of this manual.

#### 6.2.2.1 METHOD 1

Sixteen bits are read by hardware and 16 bits are read by software. Any complications arising from the use of this method may be avoided by using the enclosed standard BIOS, "mouse\_read".

#### 6.2.2.2 METHOD 2

Thirty-two bits are read by software.

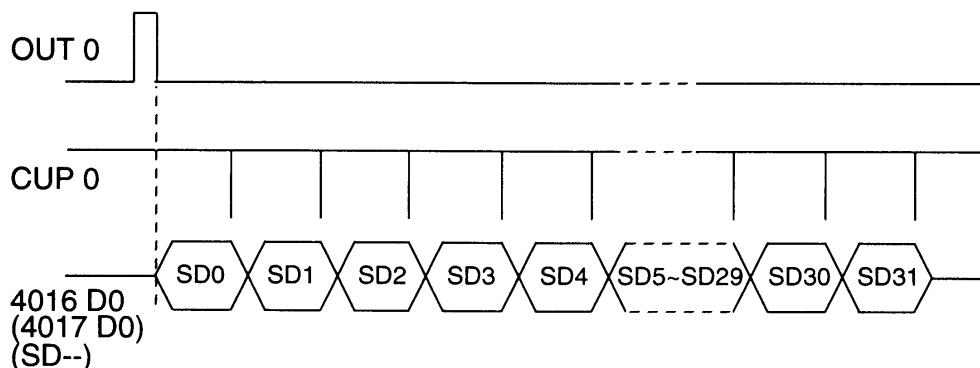


Figure 4-6-2 Serial Data Read Timing

## 6.3 SPEED SWITCHING

Super NES Mouse speed can be switched as described in the following paragraphs.

### 6.3.1 USING SOFTWARE

The programmer should write 1 in D0 of 4016H (OUT0 is HI), and immediately read 4016H. (Read 4017H for controller 2). Then, set OUT0 to LOW, and immediately read 4016H again. (Read 4017H for controller 2). The mouse speed will switch to the next setting, in the order of slow, normal, fast, and back to slow, each time this operation is performed.

### 6.3.2 USE OF OUT0 AND CUP0 SIGNALS

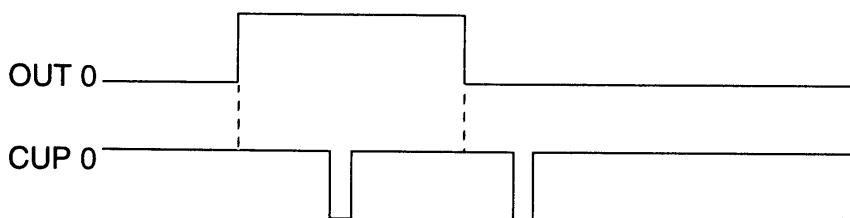
Set OUT0 to HI, and set CUP0 once to [LOW → HIGH] (read 4016H). Next, set OUT0 to LOW, and once again set CUP0 to [LOW → HIGH]. This changes the mouse tracking speed by one setting. The speed is changed by two settings if CUP0 is set LOW to HI twice while OUT0 is HI.

### 6.3.3 CAUTIONS

Once switched, the speed mode is output to SD10 and SD11. Note that the speed setting in SD10 and SD11 may not be the same as the speed setting in the mouse. The mouse tracking speed should always be switched once immediately after connecting the mouse to ensure that the mouse tracking speed and the speed setting in SD10 and SD11 are the same. This should also be done when the mouse is accidentally disconnected during a game.

The sample software MOUSE.X65 contains a subroutine for switching speeds called speed\_change.

(Refer to "Mouse Speed Switching Routine" in the following chapter.)



## 6.4 DATA

### 6.4.1 SIGNATURE (SD12~SD15)

The signature is stored in SD12~SD15. Use this code to identify what is currently connected to the 7 pin console connector. (When using the standard BIOS, check the connection with mouse\_con in the Super NES register. Refer to "Using the Standard BIOS".) When the mouse is connected, the code is 0001B. Check the signature to verify whether or not the mouse is connected. If a different signature appears (signatures up to 1111B may be assigned to input devices other than a mouse), input data should be inhibited. When nothing is connected or a standard controller is connected, the signature is 0000B.

### 6.4.2 SPEED DATA (SD10 and SD11)

The speed data identifies whether the speed mode in the mouse is set to slow (00B), normal (01B) or fast (10B). The mouse contains an internal speed switching circuit which switches between the three different tracking speeds. Switching between speeds is done using software in the Super NES console. (Refer to "Speed Selection and Cursor Movement" to switch the tracking speed). SD10 and SD11 contain the data the mouse transmits to the Super NES console to inform the Super NES console which speed mode is currently active.

### 6.4.3 LEFT AND RIGHT ACTUATORS (SD8 and SD9)

Bit SD9 is "1" when the left mouse actuator is pressed, and SD8 is "1" when the right actuator is pressed.

#### 6.4.4 X, Y ABSOLUTE DISPLACEMENT (SD16~SD31)

When moving an object or BG with the mouse in a positive direction (SD16 and SD24 = 0), add the X and Y data to the respective horizontal and vertical positions. When moving an object or BG in the negative direction (SD16 and SD24 = 1), subtract the seven bits, which are the X and Y data less the direction bits (SD16 and SD24) from the positions. Note that SD16 and SD24 are the most significant bits and SD23 and SD31 are the least significant bits.

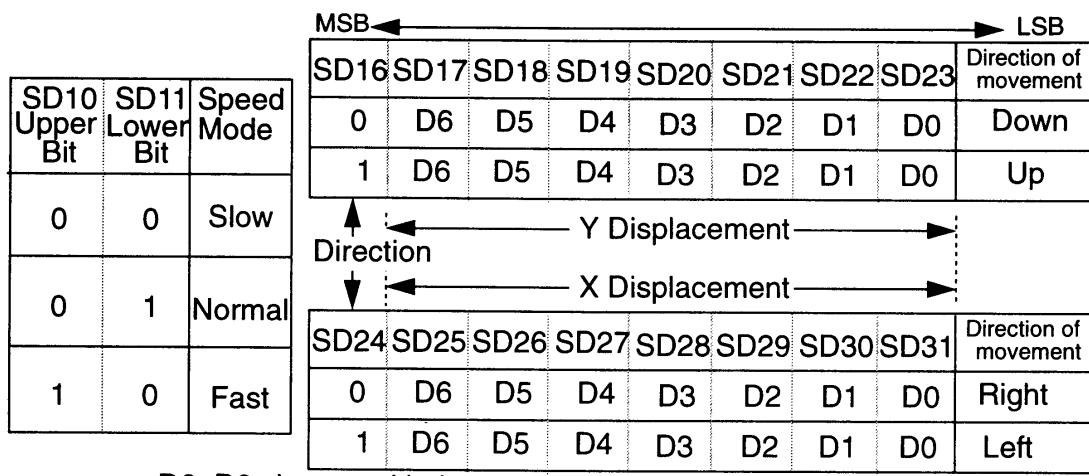


Figure 4-6-3 Explanation of Data Strings 2 Bits or Longer

## 6.5 SUPER NES MOUSE SPECIFICATIONS

### 6.5.1 ELECTRICAL SPECIFICATIONS

Operating voltage:  $5 \text{ V} \pm 10\%$

Current consumption: 50 mA (maximum)

### 6.5.2 OPERATIONAL AND ENDURANCE SPECIFICATIONS

Resolution: 50 counts/inch  $\pm 15\%$

Tracking speed: 250 mm/sec (maximum)

Useable Life: 5000 hours in powered state (min.)

(with vertical load of 100 g and voltage of  $5 \text{ V} \pm 5\%$ .)

Actuators: two tact switches (guaranteed to endure at least 2.5 million engagements.)

### 6.5.3 DIMENSIONS

Length: 98 mm

Width: 64 mm

Height: 35 mm

Cable length: 1.4 m

Weight: approximately 140 g

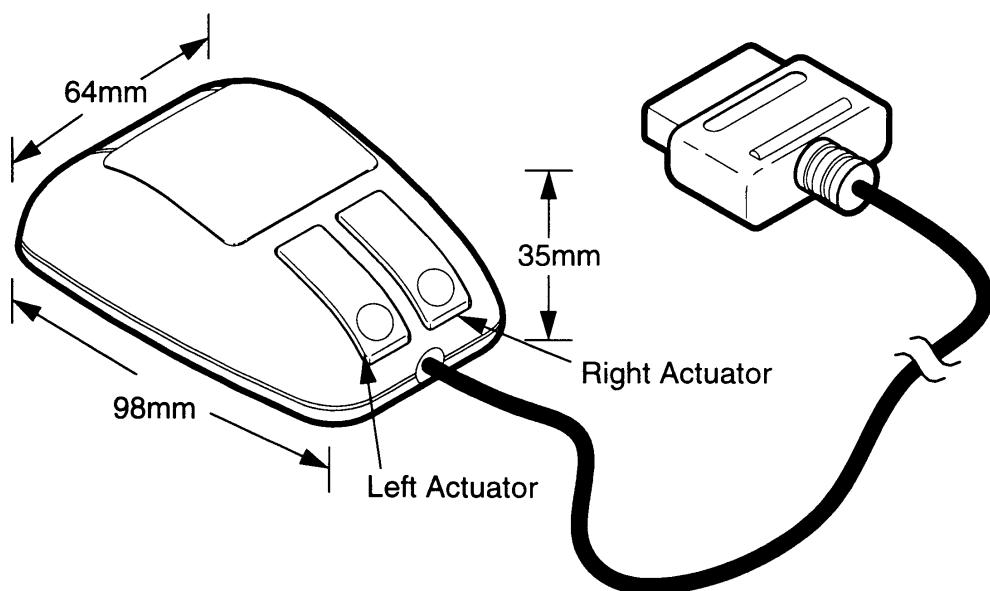


Figure 4-6-4 Super NES Mouse Dimensions

## ***Chapter 7. Using the Standard BIOS***

### **7.1 THE STANDARD BIOS**

Nintendo strongly recommends the use of the following standard BIOS with all Super NES Mouse related programming. If the standard BIOS is not used, future modifications to the mouse, the Super NES control deck, or related software, hardware, or accessories will likely impair or limit the future use and/or compatibility of such non-standard programs.

The software enclosed contains a file called MOUSE.X65. This file has two subroutine programs.

1. `mouse_read`: reads serial data from the mouse.
2. `speed_change`: switches the mouse speed.

Whenever `mouse_read` is used, `speed_change` should also be used. An explanation of how to use these sub-routines is given below. Refer to "Registers" for a summary of the registers needed to use the standard BIOS, `mouse_read`, and `speed_change`.

## 7.2 MOUSE SERIAL DATA READ ROUTINE (`mouse_read`)

This routine is used in the same way the key data read subroutine is used with a standard controller. `Mouse_read` must be called as a subroutine in the main program at every frame. All information needed for using the mouse can be found in the registers shown in the figure, "Standard BIOS Output Register", on the following page. (Data is read when the mouse is connected to either connector 1 or 2.)

Cautions concerning this program:

1. The program, `mouse_read`, uses an automatic key data read function to read the serial data from the mouse. Therefore, the automatic read function must always be turned on when the standard BIOS, `mouse_read`, is used.
2. Do not call this subroutine during the automatic read (hardware read). Refer to "Joy Controller" in the Software section of this manual to circumvent the timing problem.
3. Always use `mouse_read` and `speed_change` together. The mouse tracking speed must always be switched once immediately after connecting the mouse to the Super NES control deck, `mouse_read` uses `speed_change` to do this automatically. The paragraph titled "Super NES Mouse Speed Switching Routine" describes how to use the subroutine, `speed_change`.

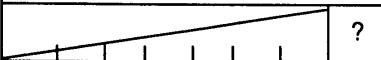
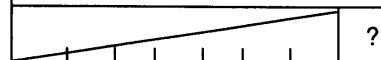
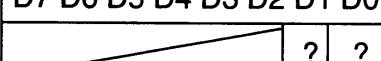
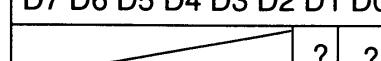
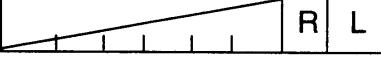
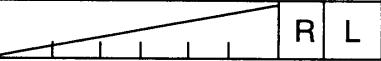
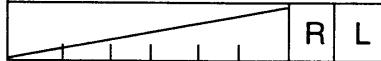
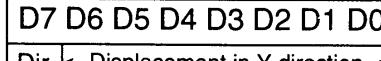
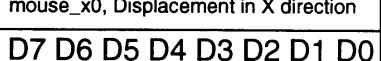
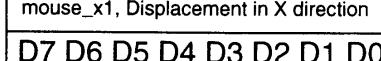
When connected to connector 1	When connected to connector 2	Significance of data
mouse_con0 mouse connection info. D7 D6 D5 D4 D3 D2 D1 D0  ?	mouse_con1 mouse connection info. D7 D6 D5 D4 D3 D2 D1 D0  ?	0 / not connected 1 / connected
mouse_sp0 mouse speed information D7 D6 D5 D4 D3 D2 D1 D0  ? ?	mouse_sp1 mouse speed information D7 D6 D5 D4 D3 D2 D1 D0  ? ?	D1:0 D0:0 slow D1:0 D0:1 normal D1:1 D0:0 fast
mouse_sw0 mouse switch continuous D7 D6 D5 D4 D3 D2 D1 D0  R L	mouse_sw1 mouse switch continuous D7 D6 D5 D4 D3 D2 D1 D0  R L	R: right button L: left button 0 / OFF 1 / ON
mouse_swt0 mouse switch trigger D7 D6 D5 D4 D3 D2 D1 D0  R L	mouse_swt1 mouse switch trigger D7 D6 D5 D4 D3 D2 D1 D0  R L	R: right button L: left button 0 / OFF 1 / ON
mouse_y0, Displacement in Y direction D7 D6 D5 D4 D3 D2 D1 D0 Dir. <--Displacement in Y direction--> 	mouse_y1, Displacement in Y direction D7 D6 D5 D4 D3 D2 D1 D0 Dir. <--Displacement in Y direction--> 	Dir: Direction bit 0 / down 1 / up
mouse_x0, Displacement in X direction D7 D6 D5 D4 D3 D2 D1 D0 Dir. <--Displacement in X direction--> 	mouse_x1, Displacement in X direction D7 D6 D5 D4 D3 D2 D1 D0 Dir. <--Displacement in X direction--> 	Dir: Direction bit 0 / right 1 / left

Figure 4-7-1 Standard BIOS, Output Register

## 7.3 SUPER NES MOUSE SPEED SWITCHING ROUTINE / speed\_change (Screen cursor, OBJ and BG move speed switching)

This section describes the speed switching program, speed\_change, found in the "MOUSE.X65" program (supplied on sample diskette).

Connector 1. Set the X register to "0"

Set the number corresponding to the desired speed in the mouse\_sp\_set0 register, where slow = 0, normal = 1 and fast = 2.

Connector 2. Set the X register to "1".

Set the number corresponding to the desired speed in the mouse\_sp\_set1 register.

After setting the X and mouse\_sp\_set0 or mouse\_sp\_set1 registers, call the speed\_change subroutine. The speed will be switched to the desired setting in one step. (Because the mouse tracking speed can only be switched in a rotary switch fashion, the speed\_change subroutine is useful when switching the speed twice; for example, to switch from "normal" to "slow".)

When the mouse tracking speed is changed, the new speed data is transmitted by the mouse, and mouse\_sp0 and mouse\_sp1 data are rewritten.

### 7.3.1 CAUTION

Do not forget to set the X and mouse\_sp\_set0 or mouse\_sp\_set1 registers.

Figure 4-7-2 Examples of Speed Switching Program Subroutine Call

#### Example 1

```
Idx  #$00      ; Connector 1
lda  #$01      ; Switch to "normal" speed
sta  mouse_sp_set0
jsr  speed_change
```

#### Example 2

```
Idx  #$01      ; Connector 2
lda  mouse_sp0 ; Look at the current speed, and increase the speed
inc  a          ; to the next highest setting
cmp  #$03
bne  change    ; If the current speed is "fast", it changes to "slow"
lda  #$00
change
sta  mouse_sp_set0
jsr  speed_change
```

### 7.3.2 USING THE PROGRAM

Mouse\_read automatically completes the above speed switching at the time the mouse is connected. (Refer to "Programming Cautions", Item 3 later in this section). If mouse\_sp\_set0 and mouse\_sp\_set1 have been cleared, then the mouse speed is "slow" when the mouse is connected.

If the mouse becomes disconnected and reconnected during a game and this program is not being used, the speed must be switched once. Mouse\_read does this automatically when the mouse is re-connected. The speed setting in that case is the same as immediately before the mouse became disconnected.

If mouse\_read is used, the entire process is done automatically. No additional steps need be taken. Mouse\_read also constantly monitors the speed data (mouse\_sp0 and mouse\_sp1), thus allowing speed changes to be programmed at any time during a game.

## 7.4 SPEED SELECTION AND CURSOR MOVEMENT

### 7.4.1 Fast (10B)

The ratio of cursor displacement to mouse displacement is automatically adjusted between 6 levels, from 1:1 to 6:1. The ratio varies according to the speed the mouse is moved. When the mouse is moved slowly, the ratio is 1:1 and when the mouse is moved quickly, the ratio increases to a maximum 6:1. To move the cursor a short distance, the mouse is moved slowly. To move the cursor a long distance, the mouse is moved quickly. When the mouse is set to "fast", the cursor moves a longer distance the faster the mouse is moved so that the distance the mouse must be moved on the table is minimized.

### 7.4.2 Normal (01B)

The ratio of cursor displacement to mouse displacement is also automatically adjusted, as with the "fast" setting. The ratio, however, is smaller.

### 7.4.3 Slow (00B)

The ratio of cursor displacement to mouse displacement is 1:1. This ratio is always fixed. For example, if the cursor moves 5 cm when the mouse is moved 10 cm, then the cursor will move 10 cm when the mouse is moved 20 cm. The distance the cursor moves is always proportionate to the distance the mouse is moved whether the mouse is moved quickly or slowly. When the mouse is set to "slow", the mouse must be moved a long distance on the table to move the cursor a long distance.

Note: 00B, 01B, and 10B are the mouse\_sp0 and mouse\_sp1 D1 and D0 bit data.

## 7.5 REGISTERS

The registers required for these subroutines are as follows.

mouse_con0,	mouse_con1	Mouse connection status (indicates the connector to which the mouse is connected.)
mouse_y0,	mouse_y1	Mouse Y axis data for connectors 1(Y0) and 2 (Y1)
mouse_x0,	mouse_x1	Mouse X axis data for connectors 1(X0) and 2 (X1)
mouse_sw0,	mouse_sw1	Actuator status for connectors 1 and 2 (01H = right actuator, 02H = left actuator)
mouse_swt0,	mouse_swt1	Trigger status for connectors 1 and 2.
mouse_sp0,	mouse_sp1	Mouse speed mode for connectors 1 and 2 (00H = slow, 01H = normal, 02H = fast)
mouse_sb0,	mouse_sb1	Work register for trigger status
mouse_sp_set0, mouse_sp_set1		For speed changes
connect_st0,	connect_st1	DS1 connection start check.
reg0l,	reg0h	Multi-purpose work register

```
*****
;*
;*      mouse.x65
;*      Super NES Mouse System file
;*      March 11, 1992
;*      (c) 1992 Nintendo of America
;*****



*****;
;*****;
;*      Mouse Driver Routine (Ver 1.00)
;*****;
*****;

db      'START OF MOUSE BIOS'          ;do not delete
=====;
;*      RAM Definition
=====;

reg0
reg0l      ds     1      ; Work registers
reg0h      ds     1      ;

mouse_con
mouse_con0    ds     1      ; Mouse connection port D0=4016
mouse_con1    ds     1      ; Mouse connection port D0=4017

mouse_sp_set
mouse_sp_set0  ds     1      ; Mouse speed setting (joy1)
mouse_sp_set1  ds     1      ; Mouse speed setting (joy2)

mouse_sp
mouse_sp0    ds     1      ; Mouse speed (joy1)
mouse_sp1    ds     1      ; Mouse speed (joy2)

mouse_y0    ds     1      ; Mouse Y direction (joy 1)
mouse_y1    ds     1      ; Mouse Y direction (joy 2)
mouse_x0    ds     1      ; Mouse X direction (joy 1)
mouse_x1    ds     1      ; Mouse X direction (joy 2)

mouse_sw
```

```
mouse_sw0      ds     1      ; Mouse button turbo
mouse_sw1      ds     1      ; Mouse button turbo

mouse_swt
mouse_swt0    ds     1      ; Mouse button trigger
mouse_swt1    ds     1      ; Mouse button trigger

mouse_sb          ; Previous switch status
mouse_sb0    ds     1      ;
mouse_sb1    ds     1      ;

cursol_x      ds     1      ;Cursor X position
cursol_y      ds     1      ;Cursor Y position
```

```
*****
;=====
;*      mouse_read
;=====

;*      If this routine is called every frame, then the mouse status will be set to the
;*      appropriate registers.

;* INPUT
;*      None (Mouse key read automatically)

;*OUTPUT
;*      Connection status (mouse_con)    D0=1 Mouse connected to Joy1
;*                                         D1=1 Mouse connected to Joy2
;*      Switch (mouse_sw0,1)           D0=left switch turbo
;*                                         D1=right switch turbo
;*      Switch (mouse_swt0,1)         D0=left switch trigger
;*                                         D1=right switch trigger
;*      Mouse movement (ball) value
;*          (mouse_x)                 D7=0 Positive turn, D7=1 Negative turn
;*                                         D6-D0 X movement value
;*          (mouse_y)                 D7=0 Positive turn, D7=1 Negative turn
;*                                         D6-D0 X movement value
*****
```

## mouse\_read

```
php
sep    #$30
_10
    lda    $4212
    and    #00000001
    bne    _10        ; Automatic read ok?

    idx    #$01        ; Joy2
    lda    $421a
    jsr    mouse_data

    lda    connect_st1
    beq    _20

    jsr    speed_change
    stz    connect_st1
```

```

plp
rts
_20
dex
lda    $4218      ; joy1
jsr    mouse_data

lda    connect_st0
beq    _30

jsr    speed_change
stz    connect_st0
_30
plp
rts

mouse_data
sta    reg0l      ; (421a 4218 save to reg0)
and    #%00001111
cmp    #$01        ; Is the mouse connected?
beq    _m10

stz    mouse_con0,x ; No connection.

stz    mouse_x0,x
stz    mouse_y0,x
stz    mouse_sw0,x
stz    mouse_swt0,x
stz    mouse_sb0,x

rts
_m10
lda    mouse_con0,x ; When mouse is connected, speed will change.
bne    _m20          ; Previous connection status
                  ; (mouse.com judged by lower 1 bit)
lda    #$01          ; Connection check flag on
sta    mouse_con0,x
sta    connect_st0,x
rts

```

```
_m20
    ldy #16          ; Read 16 bit data.

_m30
    lda $4016,x
    lsr a
    rol mouse_x0,x
    rol mouse_y0,x
    dey
    bne _m30

    stz mouse_sw0,x

    rol reg0l
    rol mouse_sw0,x
    rol reg0l
    rol mouse_sw0,x ; Switch turbo

    lda mouse_sw0,x
    eor mouse_sb0,x ; Get switch trigger
    bne _m40

    stz mouse_swt0,x

    rts

_m40
    lda mouse_sw0,x
    sta mouse_swt0,x
    sta mouse_sb0,x

    rts
```

```

;*****
;=====
;*      Speed_change
;=====

;* Set speed to mouse_sp_set. Give mouse port the value of x and call this routine.
;* If this routine is called without setting mouse_sp_set, then the previous speed will be
;* assigned to the current speed.
;* Normally, the mouse speed data will be saved to mouse_sp.
;* If the mouse speed cannot be set, then the error code will be set to mouse_sp.
;* INPUT
;*      X=connection port (X:0=joy1 1=joy2)
;*      MOUSE_SP_SET0= JOY1 setting speed
;*      MOUSE_SP_SET1= JOY2 setting speed
;* OUTPUT
;*      MOUSE_SP0 = Joy1 Mouse speed
;*                  (0=slow, 1=medium, 2=fast, $80=error code)
;*      MOUSE_SP1 = Joy2 Mouse speed
;*                  (0=slow, 1=medium, 2=fast, $80=error code)
;*****
;
```

```

speed_change
    php
    sep  #$30

    lda   mouse_con,x
    beq  _s25

    lda   #$10
    sta   reg0h

_s10
    lda   #$01
    sta   $4016
    lda   $4016,x      ; Speed change (1 step).
    stz   $4016

    lda   #$01          ; Read speed data.
    sta   $4016          ; Shift register clear.
    lda   #$00
    sta   $4016

```

```

sta    mouse_sp0,x ; Speed register clear.

ldy    #10           ; Shift register read has no meaning.

_s20
lda    $4016,x
dey
bne    _s20

lda    $4016,x      ; Read speed
lsr    a
rol    mouse_sp0,x

lda    $4016,x

lsr    a
rol    mouse_sp0,x
lda    mouse_sp0,x

cmp    mouse_sp_set0,x ;Set speed or not?
beq    _s30

dec    reg0h         ; For error check
bne    _s10

_s25
lda    #$80          ; Speed change error.
sta    mouse_sp0,x

_s30
plp
rts

db    'NINTENDO SNES MOUSE BIOS Ver1.00'      ;do not delete.

;If user modifies program, then change to
; 'MODIFIED FROM SNES MOUSE BIOS Ver1.00'
db    'END OF MOUSE BIOS'                      ;do not delete.

```

## ***Chapter 8 Programming Cautions***

Programs should be written so that controller input can be used from the time the power is turned on until the menu screen appears. (From the demo screen until the actual start point).

### **8.1 CAUTION #1**

The explanation given in Chapter 6 is based on data read by the Super NES control deck. Note that the data sent by the Super NES Mouse is in negative logic, and is inverted inside the Super NES control deck. (There is a bit inversion buffer after the Super NES controller connector.)

### **8.2 CAUTION #2**

When not using the standard BIOS, constantly check the mouse connection code, not just at start up. Take precautions to prevent problems when changing from a mouse to another input device during a game. This will protect the software from data input through other input devices. When using the standard BIOS, the mouse connection code is automatically checked constantly. If the mouse is replaced by another input device, data will not be received at that time.

This holds true for other input devices as well. If, when using a program requiring the standard controller, the programmer constantly checks that the connection code is "0000B", no errors will occur even if another input device is connected.

### **8.3 CAUTION #3**

As mentioned earlier, the mouse speed and speed data are initially undetermined. When not using the standard BIOS, always switch the speed of the mouse once after connecting it. Otherwise, the speed data (SD10,SD11) and actual speed setting of the mouse may be different. (Although they might mismatch initially, after the speed is switched automatically or manually once, the speed data and speed setting are always in agreement.) The speed switching program should be executed before any data is transmitted by the mouse. (If the mouse becomes disconnected during a game, always run the speed switching program once immediately after re-connecting the mouse.) When using the standard BIOS, the speed switching program is run automatically whenever the mouse is connected, and no additional steps need be taken.

### **8.4 CAUTION #4**

The standard BIOS, `mouse_read`, can be included in the program without modification and may be treated like a controller read routine. Call `mouse_read` as a subroutine.

Note that the standard BIOS, `mouse_read`, is designed for mouse-only software. Take caution when using a standard controller and mouse at the same time.

## **8.5 CAUTION #5**

The standard BIOS is written entirely in the eight-bit mode. Therefore, the commands php, plp and sep are executed after it is called and before returning to the main program. They may be removed when the eight-bit and sixteen bit modes are carefully managed.

## **8.6 CAUTION #6**

Refer to "Mouse Specifications", for mouse characteristics such as tracking speed = 250 mm/sec., when writing any software.

Note about the enclosed software:

The disk contains sample software which uses the standard BIOS (MOUSE.COM). MOUSE.COM displays data on the screen transmitted by the mouse and stored in each register. The number strings shown at the bottom represent 32-bit mouse data strings. The cursor will follow the movement of the mouse horizontally or vertically on the screen. Move the cursor to the heart symbol and push the left mouse actuator to change the cursor tracking speed.

## ***Chapter 9 MultiPlayer 5 Specifications***

### **9.1 INTRODUCTION TO MULTIPLAYER 5**

The Super NES MultiPlayer 5 is a standard term referring to any controller or adapter used to accommodate 3 ~ 5 players. The adapter is connected to the Super NES control deck and allows up to five people to play at one time. The adapter references all controller data simultaneously, and does not give an unfair advantage to any one controller during a game. The adapter's controller ports are identical to the Super NES controller port. Therefore, many devices which can be connected to the controller port may also be connected to MultiPlayer 5.

The adapter should be equipped with a switch which is user selectable between a 2 player (2P) mode and a 5 player (5P) mode (for three to five players). When the adapter is in the 2P mode, the software treats MultiPlayer 5 controller port #2 as an extension of controller port #2 of the Super NES control deck. A BIOS is provided on 3.5" diskette to read the multiple controller data input to MultiPlayer 5.

This chapter describes how data is read from peripheral devices connected to MultiPlayer 5. For reliable operation, the supplied BIOS should always be used. Refer to the following chapter for details on the supplied BIOS.

There are no standard entries that are required in manuals provided with games that use MultiPlayer 5. However, the manual should explain how to connect and operate MultiPlayer 5 when playing a multi-player game. A MultiPlayer 5 logo is available for use on packaging and advertising. The logo artwork may be obtained through the NOA Licensing Department.

## 9.2 HARDWARE CONNECTIONS

The figure below demonstrates a typical hardware arrangement using the Super NES control deck and a MultiPlayer 5 device.

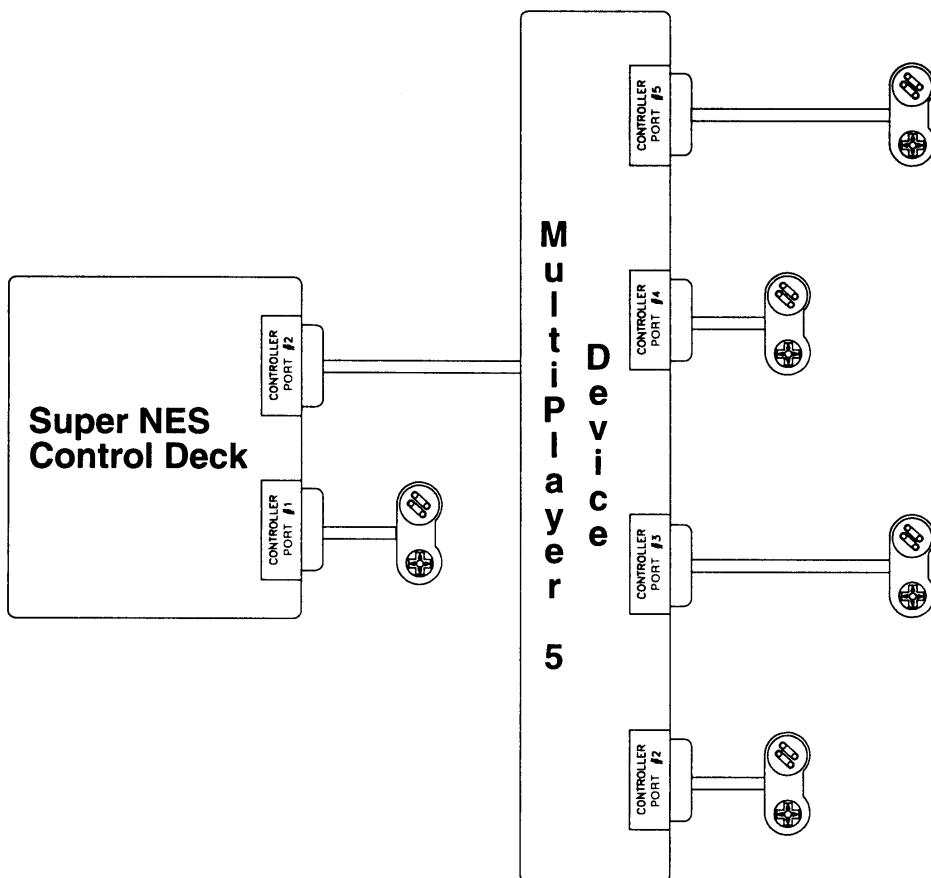


Figure 4-9-1     MultiPlayer 5 Device Hardware Connections

The MultiPlayer 5 device is connected to the Super NES control deck through controller port #2. The MultiPlayer 5 device should not be used with controller port #1 of the control deck. This should be carefully explained and addressed in all software and related manuals.

### 9.3 MODES OF OPERATION

Each MultiPlayer 5 device is equipped with a switch for changing between the 2P and 5P modes. The function of this switch is demonstrated in the table below.

PIN #	SYMBOL	2 PLAYER MODE			5 PLAYER MODE				
		②	EXPANSION CONNECTORS			②	EXPANSION CONNECTORS		
			③	④	⑤		③	④	⑤
1	+5V	X	NC	NC	NC	X	X	X	X
2	CUP	X				X	X	X	X
3	OUT0	X				X	X	X	X
4	D0	X				X	X	X	X
5	D1	X				NC			
6	PP	X				NC			
7	GND	X				X	X	X	X
Adapter Connection Status Detection		NOT AVAILABLE			AVAILABLE				

X = Connected  
NC = Not Connected

Table 4-9-1 MultiPlayer 5 Switch Function

#### 9.3.1 TWO PLAYER MODE

In the 2P mode, only controller port #2 of the MultiPlayer 5 device can be used. In this mode, MultiPlayer 5 controller port #2 performs the same functions as controller port #2 of the Super NES control deck.

#### 9.3.2 FIVE PLAYER MODE

In the 5P mode all connectors of the MultiPlayer 5 device can be used. This permits up to 5 players to play a game at one time (counting controller port #1 of the Super NES control deck).

## 9.4 PROGRAMMING CAUTIONS FOR COMPATIBLE SOFTWARE

### 9.4.1 CAUTION #1

Games should be programmed to use the MultiPlayer 5 device only when the device is connected to controller port #2 of the Super NES control deck. Games should display the following warning message and the program should halt, when the MultiPlayer 5 device is connected to controller port #1 of the Super NES control deck and the MultiPlayer 5 is in the 5P mode.

“The Super NES MultiPlayer 5 Adapter must be connected to Controller Socket #2.”

### 9.4.2 CAUTION #2

Games should be programmed so that game play can be continued if the MultiPlayer 5 or one of the devices connected to it becomes disconnected.

### 9.4.3 CAUTION #3

The Super NES Super Scope can not be used with the MultiPlayer 5. The following error message should be displayed and the program should halt if the Super NES Super Scope is connected to the MultiPlayer 5 using the 5P mode.

“The Super NES MultiPlayer 5 Adapter is not designed for use with the Super NES Super Scope.”

### 9.4.4 CAUTION #4

The Super NES Mouse can not be used with the MultiPlayer 5. The following error message should be displayed and the program should halt if the Super NES Mouse is connected to the MultiPlayer 5 using the 5P mode.

“The Super NES MultiPlayer 5 Adapter is not designed for use with the Super NES Mouse.”

### 9.4.5 CAUTION #5

Use the supplied BIOS whenever possible to ensure hardware and software compatibility. If a custom BIOS is used, read connector #2 and #3, followed by connector #4 and #5; because PP7 changes from a logic 0 to 1 slowly. Refer to “Reading Data” on the following page.

### 9.4.6 CAUTION #6

Programs can not detect whether the MultiPlayer 5 is connected when the MultiPlayer 5 is in the 2P mode.

### 9.4.7 CAUTION #7

Software should be evaluated using the MultiPlayer Development Assembly prior to submission. This assembly may be obtained through the NOA Parts Department. Refer to “Super NES Parts List” in the “Supplemental Information” section of this manual.

#### 9.4.8 CAUTION #8

When using the MultiPlayer 5 with the supplied BIOS, use caution in the order of the BIOS call (refer to "Supplied BIOS Execution" in the following chapter).

### 9.5 READING DATA

#### 9.5.1 STANDARD CONTROLLER CONNECTED (5P MODE)

When the MultiPlayer 5 is in the 5P mode, data from the four connected controllers is read in two groups; controllers 2 and 3, and controllers 4 and 5. Data from each of these groups is read in parallel starting from <4017H> D0 and D1. The bit at PP7 (<4201H> D7) is used to switch between the two groups. The normal condition of PP7 is 1. If changed to 0, it should be set back to 1 immediately.

PP7 = 1	Read controller 2 data from <4017H> D0
	Read controller 3 data from <4017H> D1
PP7 = 0	Read controller 4 data from <4017H> D0
	Read controller 5 data from <4017H> D1

##### 9.5.1.1 READ TIMING

Read timing is demonstrated in the figure below.

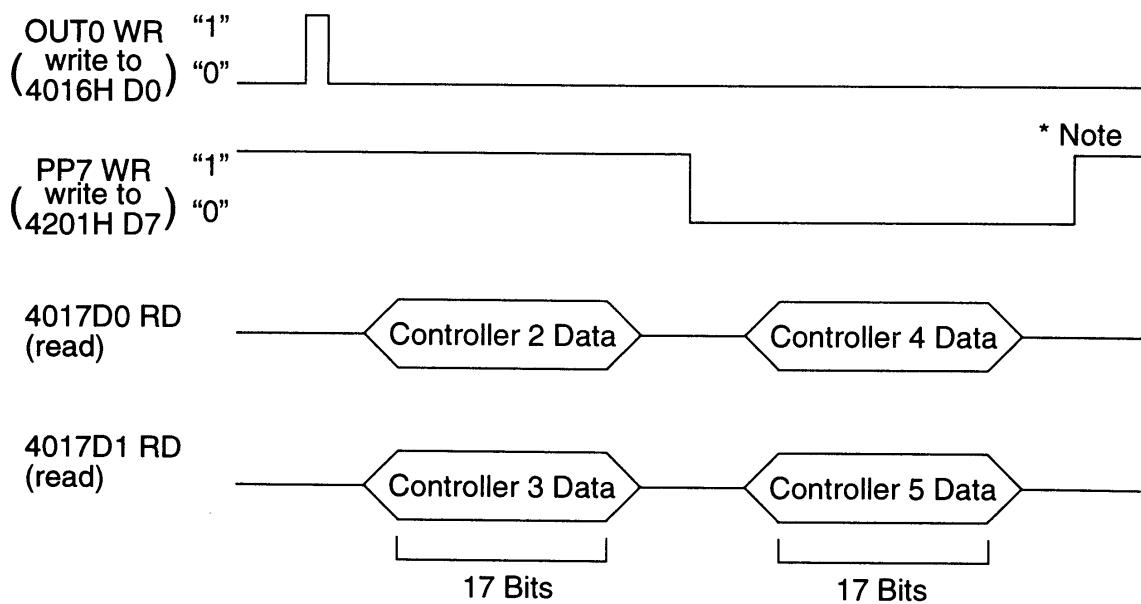


Figure 4-9-2 MultiPlayer 5 Read Timing Chart, 5P Mode

Note: The normal state outputs "1" to PP7. After reading Controller Data 4 and 5, the state should be returned to "1".

### 9.5.1.2 DATA FORMAT

The following table lists the MultiPlayer 5 data format when controllers are connected to connectors 2 through 5. An asterisk (\*) is used to show that the indicated data is 0 when that controller is not connected.

1	Output “1” in advance to PP7 (<4201H> D7) Change OUT0 (<4016H> D0) from “0” to “1” to “0”			
2	Content of <4017H>			
	D7~D2	D1	D0	
<4017H> 1st read	undefined	Controller 3 B button	Controller 2 B button	
<4017H> 2nd read	undefined	Controller 3 Y button	Controller 2 Y button	
<4017H> 3rd read	undefined	Controller 3 select button	Controller 2 select button	
.	.	.	.	
.	.	.	.	
.	.	.	.	
<4017H> 15th read	undefined	0	0	
<4017H> 16th read	undefined	0	0	
<4017H> 17th read	undefined	1 (*)	1 (*)	
3	Change the output going to PP7 (<4201H> D7) from “1” to “0”			
4	<4017H> 18th read	undefined	Controller 5 B button	Controller 4 B button
	<4017H> 19th read	undefined	Controller 5 Y button	Controller 4 Y button
	<4017H> 20th read	undefined	Controller 5 select button	Controller 4 select button
.	.	.	.	
.	.	.	.	
.	.	.	.	
<4017H> 32nd read	undefined	0	0	
<4017H> 33rd read	undefined	0	0	
<4017H> 34th read	undefined	1 (*)	1 (*)	
5	After controller data has been read, change the output to PP7 (<4201H> D7) from “0” to “1”			

Table 4-9-2 MultiPlayer 5 Data Format

## 9.5.2 PERIPHERAL DEVICE CONNECTIONS

The MultiPlayer 5 connectors are identical in shape to the controller ports of the Super NES control deck. Peripheral devices other than controllers can be connected. However, some types of devices are not compatible with the MultiPlayer 5.

### 9.5.2.1 INCOMPATIBLE DEVICES

The following devices cannot be used with MultiPlayer 5 except for those devices marked with an asterisk (\*), which can be used only when MultiPlayer 5 is in the 2P mode. If any of the devices marked with an asterisk (\*) are used when MultiPlayer 5 is in the 5P mode, they either will not operate or may not operate normally.

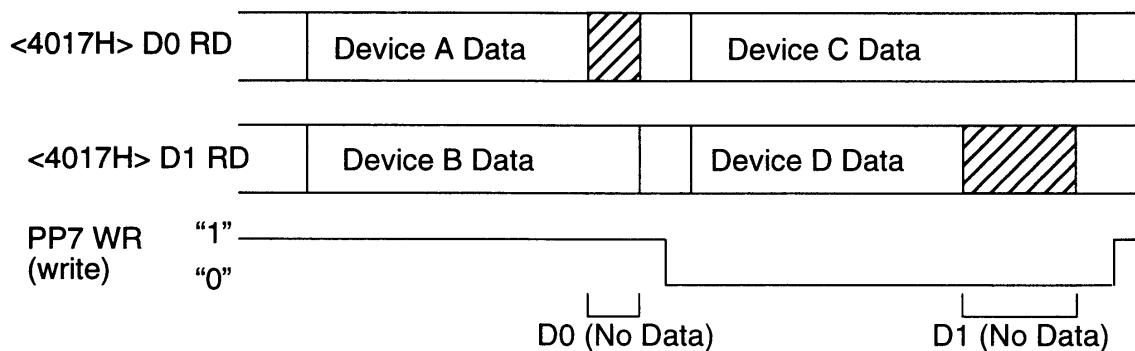
- 1\*. Any device which uses <4016H> D1 or <4017H> D1 for its data read.
- 2\*. Any device which uses <4201H> or <4213H>.
3. Any device with an electrical consumption of 17mA or more per unit.
- 4\*. Any device which detects a CUP signal while OUT 0 is "1".
5. Any device which transmits data while OUT 0 is "1".
6. Any adapter used to connect other devices.

Examples of devices which can not be used with MultiPlayer 5:

- Super NES mouse (for reason 3).
- Super NES Super Scope (for reason 2)
- MultiPlayer 5 (for reason 6)

### 9.5.2.2 DISSIMILAR DEVICES

Dissimilar devices can be used simultaneously as long as any one device is not contained in the previous incompatibility list. Differences in data composition and length between the various devices will not result in any problems. An example of data read timing for dissimilar devices is provided below.



#### Device Physical Connections:

Device A = Connector 2	Device C = Connector 4
Device B = Connector 3	Device D = Connector 5

**Figure 4-9-3     Data Read Timing for Dissimilar Devices**

When the data length between two devices that are read in parallel is different, the excess part (shaded) is read in with no data. The above setting is only one example and all four devices do not need to be connected.

## 9.6 IDENTIFYING DEVICES CONNECTED TO MULTIPLAYER 5

### 9.6.1 SIGNATURES

Nintendo has a standard for each "signature" which allows software to detect the type of device connected. Software uses the signature to select the appropriate operations mode and menu for the connected device and to inhibit data from being read from incompatible devices.

The peripheral device signature is contained in bits 13 ~ 16 of the OUT 0 latch pulse (<4016H> D0 WR) when read serially from <4016H> D0 (<4017H> D0). Refer to the chapter "CPU Registers" in book 1 for more information concerning these registers.

The signature for a standard controller is 0000. Refer to device programming documentation for the signature of other devices.

### 9.6.2 MULTIPLAYER 5 SIGNATURE

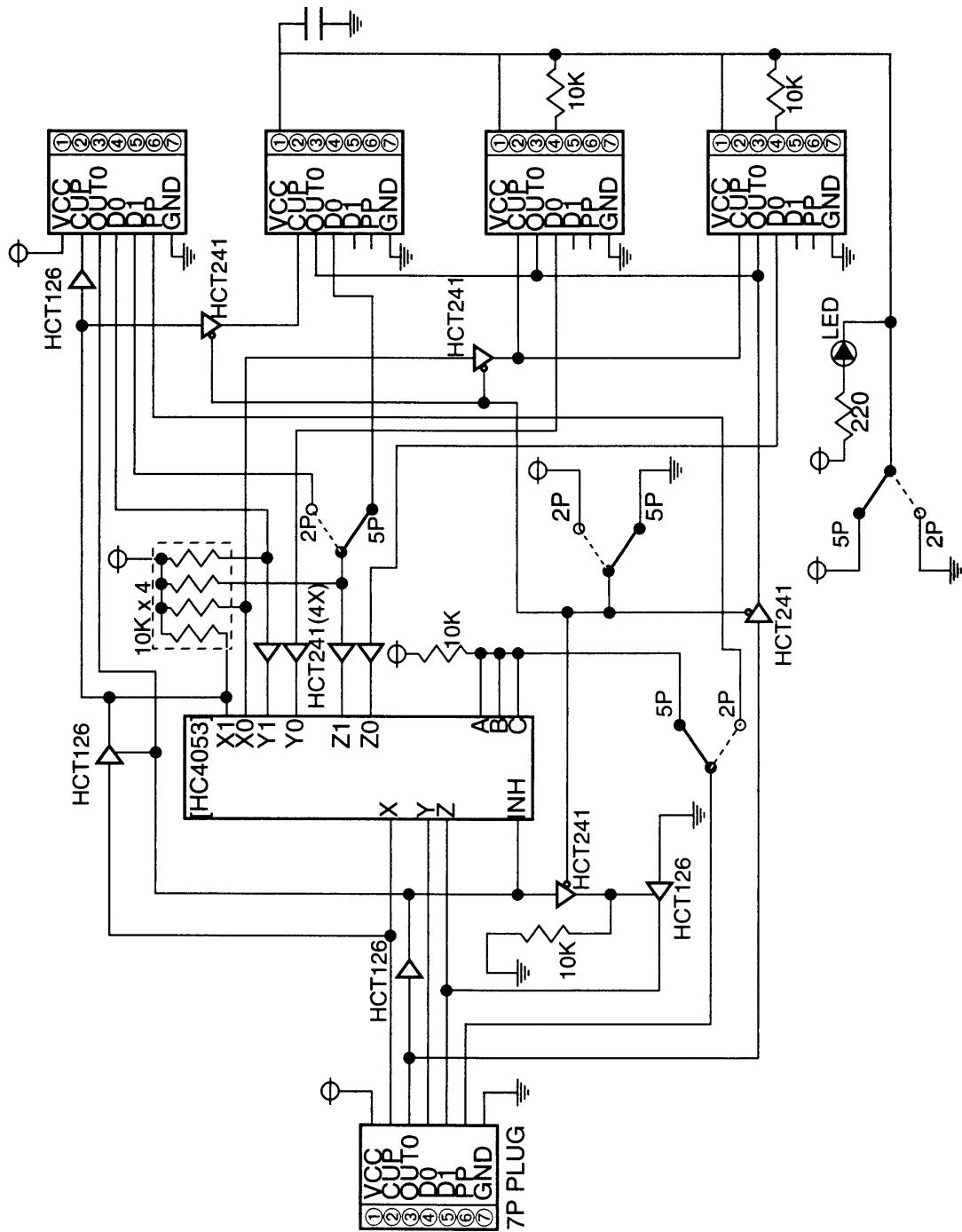
MultiPlayer 5 simply passes on the signature codes for devices connected to controller ports 2 ~ 5 and does not have a signature code of its own. However, the following procedure will verify that MultiPlayer 5 is connected. When performing this procedure, it does not matter whether or not a device is connected to MultiPlayer 5 controller ports 2 ~ 5.

1. Output "1" to register <4016H> D0.
2. Read register <4017H> D1 eight times and verify that it is "11111111 (FFH)".
3. Output "0" to register <4016H> D0.
4. Read register <4017H> D1 eight times and verify that it is not "11111111 (FFH)".

If items 1~4 are all satisfied, MultiPlayer 5 is connected to controller port #2 of the Super NES control deck and the 2P/5P mode switch is in the 5P mode. The Super NES cannot detect if MultiPlayer 5 is connected when MultiPlayer 5 is in the 2P mode. To verify that MultiPlayer 5 is connected to controller port #1 of the Super NES control deck, complete the same test procedure using register <4016H> D1.

## 9.7 MULTIPLAYER 5 SCHEMATIC DIAGRAM

**SUPER NES MultiPlayer 5 - SCHEMATIC DIAGRAM** (Rev 2.3) May 1, 1992



## 9.8 READING CONTROLLER DATA

In order to understand the process by which MultiPlayer 5 data is read, the user must first understand the method by which normal controller data is read. This method is described in the following paragraphs.

### 9.8.1 CONTROLLER DATA STORAGE

Controller data is stored at <4218H> ~ <421BH> in the Super NES CPU. This data, originally transmitted in serial form by the controller, has been automatically expanded by the CPU internal hardware. The controller automatic read function operates during the PPU V-blank period. Therefore, the controller status for the previous V-blank is stored at <4218H> ~ <421BH>. Refer to "Joy Controller" in the "Software" section of this manual.

Note: Super NES CPU registers <421CH> ~ <421FH> are provided for expansion of controller data storage. However, no data is stored in this area by MultiPlayer 5 and data held by these registers is ignored.

In addition to reading controller and other external device data automatically, the Super NES can read data serially using software. Data can also be read using a combination of the automatic read function (up to 16 bits) and software (from the 17th bit).

## 9.8.2 CONTROLLER I/O PORTS

There are four Super NES I/O ports used for reading controller (or peripheral device) data in serial format.

### 9.8.2.1 REGISTER <4016H> (D0, D1 READ)

Bits D0 and D1 of this register read peripheral devices connected to controller port #1 of the Super NES control deck.

### 9.8.2.2 REGISTER <4017H> (D0, D1 READ)

Bits D0 and D1 of this register read peripheral devices connected to controller port #2 of the Super NES control deck.

### 9.8.2.3 REGISTER <4016H> (D0 WRITE)

This is the controller shift registers' parallel load control.

### 9.8.2.4 REGISTER <4201H> (D6, D7 WRITE)

Bit D6 enables serial output for controller port #1 and bit D7 enables serial output for controller port #2.

### 9.8.2.5 REGISTER <4213h> (D6, D7 READ)

Bits D6 and D7 read inputs from the parallel I/O ports.

Only specially designed devices allow data input from registers <4016H> bit D1 and <4017H> bit D1. When a controller is used by itself (directly connected to the Super NES), this data is undefined.

The following figure demonstrates a valid controller data string. The shaded area indicates data that is automatically read.

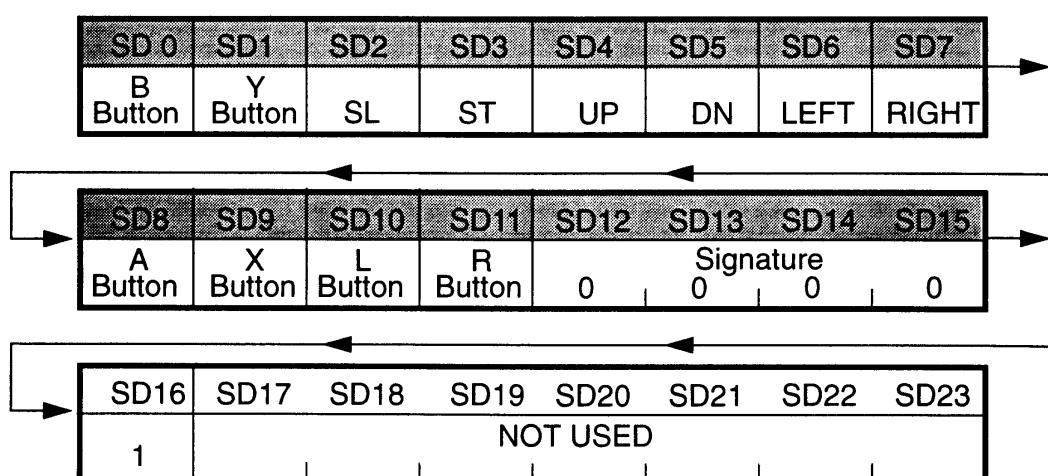


Figure 4-9-4 Valid Controller Data String

The data for each button is transmitted as “1” when pressed and “0” when not pressed. The SD16 data bit is used to verify a controller is connected. A controller is connected to the port when the signature code is 0000 and SD16 = 1. When the controller is not connected, the signature code is 0000 and SD16 = 0.

## ***Chapter 10 MultiPlayer 5 Supplied BIOS***

Super NES hardware and any MultiPlayer 5 program which does not use the supplied BIOS may not be fully compatible. (When any minor hardware changes are made in the future, maintaining the compatibility at the BIOS level will have the first priority.)

The enclosed diskette includes the following two files, which compose the BIOS program.

- M\_CHECK.X65, Version X.XX
- MULTI5.X65, Version X.XX

### **10.1 FILE DESCRIPTION**

The file "M\_CHECK.X65" determines whether a MultiPlayer 5 device is connected to the Super NES. The file "MULTI5.X65" reads controller data for 5 players. The diskette contains the following 8 files. These files were written using the Super NES Emulator development system.

#### **10.1.1 BIOS FILES**

- MULTI5. X65
- M\_CHECK. X65

#### **10.1.2 SAMPLE PROGRAM FILES**

- TEST. X65
- INIT. X65
- FONT. X65
- MAKE. BAT
- TEST. ISX
- TEST.COM

### **10.2 SAMPLE PROGRAM EXECUTION**

The enclosed disk also contains a sample program for checking MultiPlayer 5 operations. Using the MAKE file on the enclosed disk, run the program using the Super NES Emulator development tool or the EPROM evaluation board (1Mbit or larger capacity).

#### **10.2.1 OPERATION PARAMETERS**

Assign the following parameters when running the sample program.

Memory map mode: 20 mode  
Memory bank to be used: Bank 00, 80H  
Use the high speed mode: (3.58 MHz)

### 10.2.2 SAMPLE PROGRAM UTILIZATION

When power is applied, the program displays the button engagement status of the connected controller(s). The program displays a different number of controllers depending on whether the MultiPlayer 5 is in the 5P mode or the 2P mode. Button names are not displayed when a controller is not connected. An error message is displayed when the adapter is connected to controller port #1 of the Super NES control deck.

The program proceeds through the following display format when the Super NES reset button is pressed.

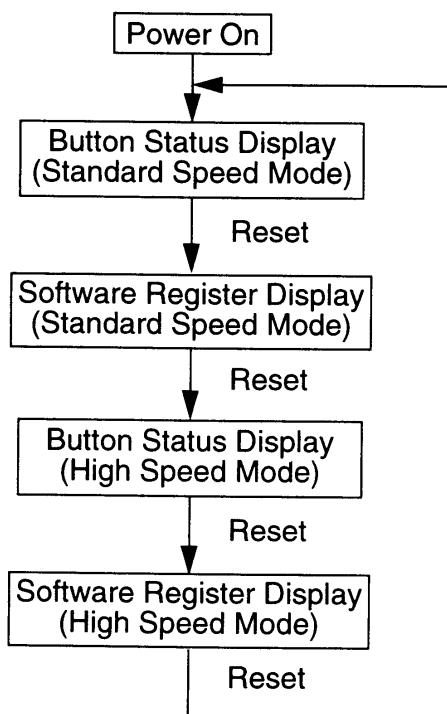


Figure 4-10-1 Sample Program Display Format

### 10.3 SUPPLIED BIOS EXECUTION

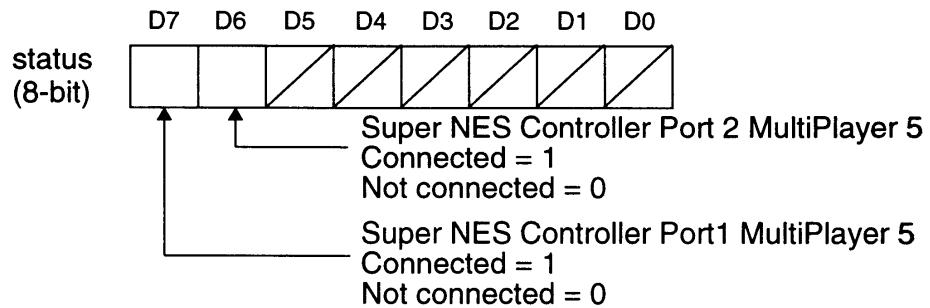
The supplied BIOS program assumes it is running in synchronization with the Super NES PPU's NMI interrupt. The program uses the Super NES CPU controller data automatic read function, so the automatic read function must be enabled when the BIOS is called (<4200H> D0=1).

The data for 5 controllers is read when the BIOS is called with the automatic read function enabled. Since the supplied BIOS uses the automatic read function, the BIOS can not be called more than once per frame (the period from one automatic read to the next automatic read).

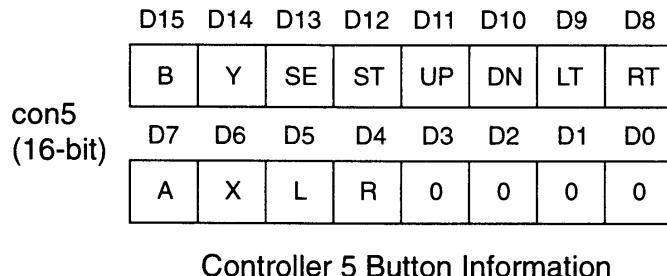
In this BIOS, the OUT0 signal is controlled by the Controller Automatic Read function. The user must ensure that the BIOS is called in the proper order. After the Super NES CPU Automatic Read period (215 µs from the start of NMI), call "MULTI5.ASM (X65)" followed by "M\_CHECK.ASM (X65)". The BIOS must be called in this order for proper operation.

## 10.4 SUPPLIED BIOS OUTPUT REGISTER

M\_CHECK.X65

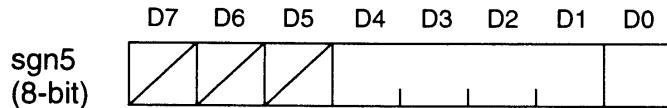


MULTI5.X65



The same format is used for con4~con1 (16 bits each).

- con 4 = Controller 4
- con 3 = Controller 3
- con 2 = Controller 2
- con 1 = Controller 1 (Super NES controller port #1)



D0 is xxx00000 when no device is connected to the Super NES controller port. D0 is xxx00001 when a controller is connected. D0 is undefined for all other devices.

The same format is used for sgn4~sgn1(8 bits each).

sgn 4: Connector 4

sgn 3: Connector 3

sgn 2: Connector 2

sgn 1: Connector 1 (Super NES controller port #1)

## 10.5 SUPPLIED BIOS CAUTIONS

### 10.5.1 CAUTION #1

MULTI5.X65 reads data under the assumption that MultiPlayer 5 is in the 5P mode with all 4 controllers connected and that a controller is connector to controller port #1 of the Super NES control deck. Therefore, if MultiPlayer 5 is not connected or a device other than a controller is connected, the contents of con1~5 are invalid. Refer to status obtained by M\_CHECK.X65 and data in sgn1~5 to check the status of device connections.

### 10.5.2 CAUTION #2

Since the supplied BIOS uses the automatic read function, the BIOS can not be called more than once per frame (the period from one automatic read to the next automatic read). Do not overlap the execution of the BIOS with the automatic read execution period (about 215 µs from the start of the NMI). Refer to the chapter "Joy Controller" under "Software" in this manual.

### 10.5.3 CAUTION #3

Nintendo does not assume responsibility for any problems which arise from using all or part of this BIOS. Developers should use the BIOS only after fully understanding its operations and usage.

### 10.5.4 CAUTION #4

Change the BIOS end code, at the end of the BIOS, when partial changes are made to the BIOS. This is demonstrated below.

- M\_CHECK.X65  
"NINTENDO SHVC MULTI5 CONNECT CHECK Ver X.XX"  
⇒ "MODIFIED FROM SHVC MULTI5 CONNECT CHECK Ver X.XX"
- MULTI5.X65  
"NINTENDO SHVC MULTI5 BIOS Ver X.XX"  
⇒ "MODIFIED FROM SHVC MULTI5 BIOS Ver X.XX"

#### **10.5.5 CAUTION #5**

When consecutively calling “MULTI5.ASM (X65)” AND “M\_CHECK.ASM (X65)”, the user must call “MULTI5.ASM (X65)” first to ensure the expected results.

## 10.6 MULTIPLAYER 5 SUPPLIED BIOS PROGRAM LISTINGS

The following are program listings contained on the MultiPlayer 5 Supplied BIOS diskette. These programs are in the I.S. assembler format.

[M\_CHECK.X65]

```

        ON816
        PUBALL
        ASSUME 0,0
MEM16  macro
        ON16A
        endm
MEM8   macro
        OFF16A
        endm
IDX16  macro
        ON16I
        endm
IDX8   macro
        OFF16I
        endm
*****
;
*****  

; MultiPlayer connection check routine ver x.xx
; Date
; © 199x Nintendo
*****  

; BANK80 GROUP 080H
=====  

; MultiPlayer connection check BIOS start code
; Please do not delete this code
=====  

        DB      'START OF MULTI5 CONNECT CHECK'  

=====  

; RAM define table
=====  

BANKEQU GROUP 0  

        EXTERN status
        EXTERN reg0l,reg0h,reg1l,reg1h
c_ad1   EQU    4016H
c_ad2   EQU    4017H

```

BANK80 GROUP 080H

```

;*****
;      MultiPlayer connection check ver x.xx
;*****
;(Caution)
;Contents of register A, B, X, Y will be destroyed after this routine.
;
check_mpa
    PHP
    IDX8
    MEM8
    SEP     #30H
    STZ     .status

;<automatic controller read enabled?>
_c00
    LDA     4212H
    AND     #01H
    BNE     _c00

;<determine if MPA is connected or not?>
    STZ     c_ad1    ;output "0" to out0
    LDA     #01H
    STA     c_ad1    ;output "1" to out0
    LDX     #08H

_c10
    LDA     c_ad1
    LSR     A
    LSR     A
    ROL     .reg0h   ;read d1 of 4016h and store it to reg0h
    LDA     c_ad2
    LSR     A
    LSR     A
    ROL     .reg1h   ;read d1 of 4017h and store it to reg1h
    DEX
    BNE     _c10

_c20
    STZ     c_ad1    ; output 0 to out0
    LDX     #08H

    LDA     c_ad1
    LSR     A
    LSR     A
    ROL     .reg0l   ;read d1 of 4016h and store it to reg0l
    LDA     c_ad2

```

```

        LSR      A
        LSR      A
        ROL      .reg11 ;read d1 of 4017h and store it to reg11
        DEX
        BNE      _c20
;

```

**MULTIPLAYER 5 SUPPLIED BIOS**

```
*****
;      MultiPlayer BIOS end code
;      Please do not delete this code
*****
DB      'END OF MULTI5 CONNECT CHECK'

END
```

```
[MULTI5.X65]
        ON816
        PUBALL
        ASSUME 0,0
MEM16    macro
        ON16A
        endm
MEM8     macro
        OFF16A
        endm
IDX16   macro
        ON16I
        endm
IDX8    macro
        OFF16I
        endm
*****
;
*****
;
;
; MultiPlayer driver routine ver x.xx
; Date
; © 199x Nintendo
*****
;
*****
;
(Caution)
; 1. Enable controller automatic read when read_mpa routine is used.
; 2. This BIOS is for the standard controller only.
; 3. This BIOS is called once every frame.
;
BANK80 GROUP 080H

=====
; MultiPlayer BIOS start code
; Please do not delete this code
=====
DB      'START OF MULTI5 BIOS'

=====
; RAM define table
=====
BANKEQU GROUP 0
        ORG 0010H

status DS 1 ; status of device connection
con5  DS 2 ;status of controller #5 (MPA #4)
```

```

con4    DS     2      ;status of controller #4 (MPA #3)
con3    DS     2      ;status of controller #3 (MPA #2)
con2    DS     2      ;status of controller #2 (MPA #1)
con1    DS     2      ;status of controller #1 (front connector #1)

sgn5    DS     1      ;signature of controller #5 (MPA #4)
sgn4    DS     1      ;signature of controller #4 (MPA #3)
sgn3    DS     1      ;signature of controller #3 (MPA #2)
sgn2    DS     1      ;signature of controller #2 (MPA #1)
sgn1    DS     1      ;signature of controller #1 (front connector #1)

reg0l   DS     1      ; Work register
reg0h   DS     1      ; Work register
reg1l   DS     1      ; Work register
reg1h   DS     1      ; Work register

c_ad1   EQU    4016H
c_ad2   EQU    4017H

BANK80 GROUP  080H

;*****
; (Caution)
; Contents of register A, B, X, Y will be destroyed after this routine.
;
read_mpa
    PHP
    IDX8
    MEM8
    SEP     #30H
    STZ     <status

;<automatic read of controller data enable?>
_10
    LDA    4212H
    AND    #01H
    BNE    _10

;<store data of controller #1>
    LDA    4219H
    STA    con1+1
    LDA    4218H
    STA    con1    ;store data of controller #1 to con1 (1 byte)
    AND    #0FH
    STA    sgn1
    LDA    c_ad1

```

```

        LSR      A
        ROL      sgn1    ;store signature of controller #1 to sgn1

;<store data of controller #2 and #3>
        LDA      421BH
        STA      con2+1
        LDA      421AH
        STA      con2    ;store data of controller #2 to con2
        AND      #0FH
        STA      sgn2
        LDA      421FH
        STA      con3+1
        LDA      421EH
        STA      con3    ;store data of controller #3 to con3
        AND      #0FH
        STA      sgn3
        LDA      c_ad2
        LSR      A
        ROL      sgn2    ;store signature of controller #2 to sgn2
        LSR      A
        ROL      sgn3    ;store signature of controller #3 to sgn3

;<output "0" to PP7>
        LDA      #7FH
        STA      4201H

;<read and store data of controller #4 and #5>
        LDY      #10H
_20
        LDA      c_ad2
        MEM16
        REP      #20H
        LSR      A
        ROL      con4    ;store data of controller #4 to con4
        LSR      A
        ROL      con5    ;store data of controller #5 to con5
        MEM8
        SEP      #20H
        DEY
        BNE      _20
        LDA      con4
        AND      #0FH
        STA      sgn4
        LDA      con5
        AND      #0FH
        STA      sgn5

```

```
LDA      c_ad2
LSR      A
ROL      sgn4    ;store signature of controller #4 to sgn4
LSR      A
ROL      sgn5    ;store signature of controller #5 to sgn5

;< output "1" to PP7>
LDA      #0FFH
STA      4201H

PLP
RTS

=====
;
; MultiPlayer driver routine ver x.xx
; (Caution)
; When this routine is used as is, please don't delete this code.
; If this routine is modified, please use the following code instead.
; 'MODIFIED FROM SHVC MULTI5 BIOS Ver x.xx'
;
*****
DB      'NINTENDO SHVC MULTI5 BIOS Ver x.xx'

*****
;
; MultiPlayer BIOS end code
; Please do not delete this code
*****
DB      'END OF MULTI5 BIOS'

END
```

## **10.7 MULTIPLAYER DEVELOPMENT ASSEMBLY**

Nintendo has created a breadboard for evaluation of MultiPlayer 5 programs. This breadboard is manufactured according to the standard MultiPlayer 5 circuit specifications and is the standard evaluation tool for MultiPlayer 5 programs. All master programs should be tested using this device prior to submission for approval.

Nintendo also uses this breadboard to test for proper operation as part of lot checks.

If the breadboard is desired for program development, contact the NOA Parts Department at (800) 531-4048. Ask for the MultiPlayer Development Assembly.

## *Chapter 1. Super NES Parts List*

Part #	Description	Remarks
22945	Control Deck (SNS)	
21712	Control Deck (SFX)	
25306	GPK Super Mario World (SNS)	
21713	GPK Super Mario World (SFX)	
23089	Cable AV (Stereo) - (ACC)	
21715	AC Adapter (SFX)	
21716	Cable RGB	
23090	Cable S-VHS (ACC)	
22424	Cable AV Mono	
21943	IC D411 CIC	
25100	IC D413 CIC (PAL)	
21326	RAM S-WRAM 1M SNS/SHVC Custom	
22423	Fuse 1.5A	
22939	Housing GPK Front (SNS)	
22940	Housing GPK Back (SNS)	
21940	Housing GPK Front (SFX)	
21941	Housing GPK Back (SFX)	
7879	Screw GPK M2x5.9	
22536	PCB SHVC-1A0N (bare)	
22537	PCB SHVC-1A1B (bare)	
22538	PCB SHVC-1A3B (bare)	
22539	PCB SHVC-1A5B (bare)	
22540	PCB SHVC-1B0N (bare)	
24468	PCB SHVC-1B5B (bare)	
26424	PCB SHVC-1K1B (bare) (Super Mario Kart)	
27441	PCB SHVC- 4PV5B Evaluation Kit	25 PCBs
28761	PCB SHVC- 2P3B Evaluation Kit	25 PCBs
22427	PCB Assy SHVC- 2P3B	
21945	PCB Assy SHVC- 1P0N	
24470	PCB Assy SHVC- 2Q5B	
25474	PCB Assy SHVC-4PV5B	
26011	PCB Assy SHVC-2QW5B	
28626	PCB Assy SHVC-8PV5B	
28760	PCB Assy SHVC-4QW5B	
28625	PCB Assy SHVC-1RA3B6S	
33366	PCB Assy SHVC-4PV7B	
32321	PCB Assy SHVC-8X7B	
22410	Multi Checker SFX	
27124	Multi Checker (20/21 Modes)	
22742	EPROM 64K MBM27C64 Fujitsu (blank)	
22743	EPROM 128K MBM27C128 Fujitsu (blank)	
22744	EPROM 266K MBM27C256 Fujitsu (blank)	
22745	EPROM 512K MBM27C512 Fujitsu (blank)	
22746	EPROM 1M NH27C101 Hitachi (blank)	
22748	EPROM 2M FUJITSU MBM 27C2001 (blank)	
22749	EPROM 4M TC574000D Toshiba (blank)	

FOR PARTS ORDERS CALL: 1-800-531-4048

<b>Part #</b>	<b>Description</b>	<b>Remarks</b>
21283	Connector Expansion 28 Pin Male (SFX)	
22771	Connector Expansion 28 Pin Female	
26882	Super NES Emulator-SE	
21321	IC RF5A22 CPU SHVC	
21322	IC RF5C77 PPU1 SHVC	
21323	IC RF5C78 PPU2 SHVC	
22943	DSP1	
23367	RAM 16K - S, Low Power Small	
23884	RAM 64K - S, Low Power Large	
23368	RAM 256K - S, Low Power Large	
27448	Multiplayer Development Assy	
25715	RAM, TC551001PL-85 (Emulator Upgrade)	
24966	Super NES Development Manual, Book 1	
27457	Super NES Development Manual, Book 2	

***INDEX*** (Book I)**A**

Absolute Addressing 1-17-4  
 Absolute Multiplication 1-15-1  
 Addition/Subtraction Screen 1-7-1  
 ADSR Mode 2-7-3  
 Audio Processing Unit 1-22-1

**B**

BG Mode 1-3-1, 1-27-3  
 Bit Rate Reduction 2-2-1  
 Brightness 1-27-1  
 BRR 2-2-1, 2-7-9  
 BRR Filter 2-2-1, 2-2-2  
 BRR Filter Number 2-2-1  
 BRR Format 2-2-1  
 BRR Range 2-2-1, 2-2-2

**C**

CG-RAM 1-8-1, 1-27-11  
 Channels 1-17-1  
 Clock Speed 1-21-1  
 Color Constant 1-7-2  
 Color Constant Addition/Subtraction 1-7-5, 1-9-1  
 Color Generator RAM 1-22-2  
 Colors 1-2-1  
 Controller 1-13-1, 1-14-1  
 CPU Clock 1-21-1

**D**

Data Bank Register 3-3-2, 3-4-5, 3-4-8  
 Data Transfer 1-17-1  
 Direct Page Flag 2-8-7  
 Direct Register 3-4-8  
 Direct Select 1-27-16  
 Division 1-15-1  
 DMA 1-13-1, 1-17-1  
 DMA, General Purpose 1-13-1, 1-17-1

**E**

Echo Delay 2-7-9  
 Echo Enable 2-7-8  
 Echo Feed-Back 2-7-9  
 Echo Filter Coefficients 2-7-1  
 Echo Start Address 2-7-9  
 Emulation Mode 3-1-1, 3-2-1  
 Expanded Connector 1-13-1  
 ExtBG Mode 1-5-1, 1-27-19  
 External Latch Flag 1-27-22, 4-1-3  
 External Synchronization 1-27-19

**F**

Fixed Color Addition 1-6-1

**G**

Gain Mode 2-7-3

**H**

H-Blank 1-17-4  
 H-DMA 1-6-1, 1-12-1, 1-17-1  
 Horizontal Blanking 1-1-2

**I**

Indirect Addressing 1-17-4  
 Interface 1-14-1  
 Interlace 1-1-1, 1-1-2, 1-18-1  
 Interrupt 1-16-1  
 IPL ROM 2-1-1

**J**

Joy Controller Enable 1-28-1

**M**

Main Screen 1-7-1, 1-7-5  
 Mode 20 1-21-3  
 Mode 21 1-21-4  
 Mosaic 1-4-1, 1-27-3  
 Multiplication 1-27-20

**N**

Native Mode 3-2-1

NMI 1-13-1

**O**

OAM Priority Rotation 1-27-2

Object Attribute Memory 1-22-2, 1-27-2

Object Size 1-27-1

**P**

Pallets 1-2-1

Priority 1-2-1

Priority Order 1-20-2

Processor Status Register 3-9-2

Programmable I/O Port 1-14-1, 1-28-1

Program Bank Register 3-3-3, 3-4-7

Program Counter 3-3-3

Program Status Word 2-8-6

**R**

Resolution 1-3-1, 1-18-1

**S**

Screen Addition/Subtraction 1-6-1, 1-7-5,  
1-9-1

Screen Repetition 1-27-4

Scroll 1-12-1

Scroll, Vertical Partial 1-12-1

Sony SPC700 2-8-1

Stack Pointer 3-3-3

Sub Screen 1-7-1, 1-7-5

Synchronization 1-16-1

**T**

Timer 1-16-1

Timer Enable 1-28-1

Transparency 1-7-2

Two's Complement 1-10-1

**V**

Vertical Blanking 1-1-2

**W**

Window 1-6-1, 1-12-1, 1-27-12

Window Logic 1-27-13

**INDEX (Book II)****COMMANDS/INSTRUCTIONS**

- ADC Rn 2-2-6, 2-9-3  
 ADC #n 2-2-6, 2-9-4  
 ADD Rn 2-2-6, 2-9-5  
 ADD #n 2-2-6, 2-9-6  
 ALT1 2-2-8, 2-9-7  
 ALT2 2-2-8, 2-9-8  
 ALT3 2-2-8, 2-9-9  
 AND Rn 2-2-7, 2-9-10  
 AND #n 2-2-7, 2-9-11  
 ASR 2-2-7, 2-9-12  
 ATTITUDE 3-5-22  
 BCC e 2-2-7, 2-9-14  
 BCS e 2-2-7, 2-9-16  
 BEQ e 2-2-7, 2-9-18  
 BGE e 2-2-7, 2-9-20  
 BIC Rn 2-2-7, 2-9-22  
 BIC #n 2-2-7, 2-9-23  
 BLT e 2-2-7, 2-9-24  
 BMI e 2-2-7, 2-9-26  
 BNE e 2-2-7, 2-9-28  
 BPL e 2-2-7, 2-9-30  
 BRA e 2-2-7, 2-9-32  
 BVC e 2-2-7, 2-9-34  
 BVS e 2-2-7, 2-9-36  
 CACHE 2-2-8, 2-9-38  
 CMODE 2-2-7, 2-9-39  
 CMP Rn 2-2-6, 2-9-41  
 COLOR 2-2-7, 2-9-42  
 DEC Rn 2-2-6, 2-9-43  
 DISTANCE 3-5-7  
 DIV2 2-2-6, 2-9-44  
 FMULT 2-2-6, 2-9-46  
 FROM Rn 2-2-8, 2-9-48  
 GETB 2-2-6, 2-9-49  
 GETBH 2-2-6, 2-9-51  
 GETBL 2-2-6, 2-9-53  
 GETBS 2-2-6, 2-9-55  
 GETC 2-2-6, 2-9-57  
 GYRATE 3-5-31  
 HIB 2-2-7, 2-9-58  
 IBT Rn, #pp 2-2-6, 2-9-60  
 INC Rn 2-2-6, 2-9-61  
 INVERSE 3-5-2
- IWT Rn, #xx 2-2-6, 2-9-62  
 JMP Rn 2-2-7, 2-9-63  
 LDB (Rn) 2-2-6, 2-9-64  
 LDW (Rn) 2-2-6, 2-9-66  
 LEA Rn, xx 2-2-8, 2-9-67  
 LINK #n 2-2-7, 2-9-68  
 LJMP Rn 2-2-7, 2-9-69  
 LM Rn, (xx) 2-2-6, 2-9-70  
 LMS Rn, (yy) 2-2-6, 2-9-71  
 LMULT 2-2-6, 2-9-73  
 LOB 2-2-7, 2-9-75  
 LOOP 2-2-7, 2-9-77  
 LSR 2-2-7, 2-9-78  
 MERGE 2-2-7, 2-9-79  
 MOVE Rn, Rn' 2-2-8, 2-9-81  
 MOVE Rn, #xx 2-2-8, 2-9-82  
 MOVE Rn, (xx) 2-2-8, 2-9-83  
 MOVE (xx), Rn 2-2-8, 2-9-85  
 MOVEB Rn, (Rn') 2-2-8, 2-9-87  
 MOVEB (Rn'), Rn 2-2-8, 2-9-88  
 MOVES Rn, Rn' 2-2-8, 2-9-89  
 MOVEW Rn,(Rn') 2-2-8, 2-9-90  
 MOVEW (Rn'), Rn 2-2-8, 2-9-91  
 MULT Rn 2-2-6, 2-9-93  
 MULT #n 2-2-6, 2-9-94  
 MULTIPLY 3-5-1  
 NOP 2-2-8, 2-9-95  
 NOT 2-2-7, 2-9-96  
 OBJECTIVE 3-5-25  
 OR Rn 2-2-7, 2-9-97  
 OR #n 2-2-7, 2-9-99  
 PARAMETER 3-5-12  
 PLOT 2-2-7, 2-9-100  
 POLAR 3-5-9  
 PROJECT 3-5-18  
 RADIUS 3-5-4  
 RAMB 2-2-7, 2-9-101  
 RANGE 3-5-6  
 RASTER 3-5-15  
 ROL 2-2-7, 2-9-102  
 ROMB 2-2-7, 2-9-104

*Index* (Continued)**COMMANDS/INSTRUCTIONS** (Continued)

ROR 2-2-7, 2-9-105  
ROTATE 3-5-8  
RPIX 2-2-7, 2-9-107  
SBC Rn 2-2-6, 2-9-108  
SBK 2-2-6, 2-9-109  
SCALAR 3-5-29  
SEX 2-2-7, 2-9-110  
SM (xx), Rn 2-2-6, 2-9-112  
SMS (yy), Rn 2-2-6, 2-9-113  
STB(Rn) 2-2-6, 2-9-115  
STOP 2-2-8, 2-9-116  
STW (Rn) 2-2-6, 2-9-117  
SUB Rn 2-2-6, 2-9-118  
SUB #n 2-2-6, 2-9-119  
SUBJECTIVE 3-5-27  
SWAP 2-2-7, 2-9-120  
TARGET 3-5-20  
TO Rn 2-2-8, 2-9-121  
Triangle 3-5-3  
UMULT Rn 2-2-6, 2-9-122  
UMULT #n 2-2-6, 2-9-123  
WITH Rn 2-2-8, 2-9-124  
XOR Rn 2-2-7, 2-9-125  
XOR #n 2-2-7, 2-9-126

**SUBJECT - Alphabetical Listing****A**

Accelerator Mode 1-5-6  
Access Modes 2-4-8, 2-5-2, 2-5-4, 2-6-1  
ADC #n 2-2-6, 2-9-4  
ADC Rn 2-2-6, 2-9-3  
ADD #n 2-2-6, 2-9-6  
ADD Rn 2-2-6, 2-9-5  
ALT1 2-2-8, 2-9-7  
ALT2 2-2-8, 2-9-8  
ALT3 2-2-8, 2-9-9  
AND #n 2-2-7, 2-9-11  
AND Rn 2-2-7, 2-9-10  
ASR 2-2-7, 2-9-12  
Attitude 2-5-10, 2-5-22, 2-5-24, 2-5-25,  
2-5-27, 2-5-28, 2-5-29, 2-5-31,  
2-5-32, 2-5-33  
Auto-increment Mode 1-8-3

**B**

Barrel Shift 1-8-4, 1-8-5  
BCC e 2-2-7, 2-9-14  
BCS e 2-2-7, 2-9-16  
BEQ e 2-2-7, 2-9-18  
BGE e 2-2-7, 2-9-20  
BIC #n 2-2-7, 2-9-23  
BIC Rn 2-2-7, 2-9-22  
Bitmap 1-8-14  
Bitmap Access 1-6-3  
Bitmap Emulation 1-8-1  
Bitmap Format 1-6-1  
BLT e 2-2-7, 2-9-24  
BMI e 2-2-7, 2-9-26  
BNE e 2-2-7, 2-9-28  
BPL e 2-2-7, 2-9-30  
BRA e 2-2-7, 2-9-32  
Bulk Processing 2-7-4  
BVC e 2-2-7, 2-9-34  
BVS e 2-2-7, 2-9-36  
BW-RAM 1-1-1, 1-1-2, 1-1-3, 1-1-4, 1-2-2,  
1-2-4, 1-6-6

*Index* (Continued)**C**

Cache 2-6-1, 2-8-4, 2-8-5, 2-8-6, 2-8-7, 2-9-38  
 Cache RAM 2-6-1, 2-6-2, 2-8-8  
 Character Conversion 1 1-6-1, 1-6-7, 1-6-8  
 Character Conversion 2 1-6-2, 1-6-10, 1-6-11  
 CMODE 2-8-1, 2-8-9, 2-8-11, 2-8-12, 2-9-39  
 CMP Rn 2-9-41  
 Color 2-8-1, 2-8-4, 2-8-6, 2-8-10, 2-8-11,  
     2-8-12, 2-8-13, 2-9-41, 2-9-42  
 COLR 2-2-3, 2-2-5, 2-4-9, 2-8-4, 2-8-10,  
     2-8-11, 2-8-12, 2-8-13  
 Cumulative Arithmetic 1-1-2  
 Cumulative Sum 1-7-1, 1-7-3

**D**

DEC Rn 2-2-6, 2-9-43  
 Distance 3-5-4, 3-5-7  
 Dither 2-4-9, 2-8-9, 2-8-10, 2-8-11  
 DIV2 2-2-6, 2-9-44  
 Division 1-7-1, 1-7-2  
 DMA 1-9-1

**E**

External Latch 4-1-4  
 External Latch Flag 4-1-3

**F**

Fixed Mode 1-8-2  
 FMULT 2-2-6, 2-4-1, 2-8-16, 2-8-17, 2-9-46  
 FROM 2-6-4, 2-6-6, 2-6-7, 2-6-11, 2-7-1,  
     2-7-2, 2-7-3, 2-7-4, 2-8-10, 2-8-11  
 FROM Rn 2-2-8, 2-9-48

**G**

GETB 2-2-6, 2-9-49  
 GETBH 2-2-6, 2-9-51  
 GETBL 2-2-6, 2-9-53  
 GETBS 2-2-6, 2-9-55  
 GETC 2-2-6, 2-8-1, 2-8-4, 2-8-9, 2-8-12,  
     2-8-13, 2-9-57  
 Gyrate 3-5-31

**H**

H Counter 4-1-4  
 HIB 2-2-7, 2-9-58  
 Horizontal Counter Latch 4-1-3  
 HV Timer 1-1-2, 1-10-1  
**I**  
 IBT Rn, #pp 2-2-6, 2-9-60  
 INC Rn 2-2-6, 2-9-61  
 Inverse 3-5-2  
 I-RAM 1-1-1, 1-1-3, 1-1-4, 1-2-2, 1-2-5, 1-3-5  
 IWT Rn, #xx 2-2-6, 2-9-62

**J**

JMP Rn 2-2-7, 2-4-3, 2-9-63

**L**

LDB (Rn) 2-2-7, 2-9-64  
 LDW (Rn) 2-2-7, 2-9-66  
 LEA Rn, xx 2-2-8, 2-9-67  
 Linear Timer 1-10-1  
 LINK #n 2-2-7, 2-9-68  
 LJMP Rn 2-2-7, 2-9-69  
 LM Rn, (xx) 2-2-7, 2-9-70  
 LMS Rn, (yy) 2-2-7, 2-9-71  
 LMULT 2-2-6, 2-4-1, 2-8-16, 2-8-17, 2-9-73  
 LOB 2-2-7, 2-9-75  
 LOOP 2-2-7, 2-9-77  
 LSR 2-2-7, 2-9-78

*Index* (Continued)**M**

Masked Interrupt 1-5-3  
**MERGE** 2-2-7, 2-9-79  
**Message** 1-5-3  
**Mixed Processing Mode** 1-5-8  
**MOVE (xx), Rn** 2-2-8, 2-9-85  
**MOVE Rn, #xx** 2-2-8, 2-9-82  
**MOVE Rn, (xx)** 2-2-8, 2-9-83  
**MOVE Rn, Rn'** 2-2-6, 2-9-81  
**MOVEB (Rn')**, Rn 2-2-8, 2-9-88  
**MOVEB Rn, (Rn')** 2-2-8, 2-9-87  
**MOVES Rn, Rn'** 2-2-6, 2-9-89  
**MOVEW (Rn')**, Rn 2-2-8, 2-9-91  
**MOVEW Rn,(Rn')** 2-2-8, 2-9-90  
**MULT #n** 2-2-6, 2-8-16, 2-9-94  
**MULT Rn** 2-2-6, 2-8-16, 2-9-93  
**Multiplication** 1-7-1, 1-7-2  
**Multiply** 3-5-1

**N**

**NOP** 2-2-8, 2-6-2, 2-6-3, 2-6-4, 2-6-5, 2-6-7,  
   2-6-9, 2-8-10, 2-9-95  
**Normal Color** 2-8-11  
**Normal DMA** 1-9-2  
**NOT** 2-2-8, 2-9-96

**O**

**Objective** 3-5-22, 3-5-25, 3-5-26  
**OBJ Rotation** 2-8-11  
**OBJ Scaling** 2-8-11  
**OR #n** 2-2-7, 2-9-99  
**OR Rn** 2-2-7, 2-9-97

**P**

**Parallel Processing Mode** 1-5-7  
**Parameter** 3-3-1, 3-5-1  
**Pipeline Processing** 2-6-1, 2-6-3, 2-6-5  
**Pixel Cache** 2-8-4, 2-8-5, 2-8-6, 2-8-7, 2-8-9  
**Plot** 2-2-7, 2-4-1, 2-4-8, 2-4-9, 2-8-1, 2-8-4, 2-  
   8-5, 2-8-6, 2-8-7, 2-8-8, 2-8-9, 2-8-10, 2-  
   8-11, 2-8-13, 2-9-100  
**Polar** 3-5-9  
**Project** 3-5-10, 3-5-12, 3-5-13, 3-5-14,  
   3-5-15, 3-5-17, 3-5-18, 3-5-19,  
   3-5-20, 3-5-28

**R**

**Radius** 3-5-3, 3-5-4, 3-5-6, 3-5-7, 3-5-30  
**RAMB** 2-2-7, 2-4-6, 2-7-3, 2-9-101  
**RAN** 2-4-8, 2-5-2, 2-5-4, 2-6-1  
**Range** 3-5-6, 3-5-30  
**Raster** 3-2-1, 3-5-12, 3-5-13, 3-5-15, 3-5-16  
**Register Prefix** 2-6-6  
**ROL** 2-2-7, 2-9-102  
**ROMB** 2-2-7, 2-4-5, 2-7-1, 2-9-104  
**RON** 2-4-8, 2-5-2, 2-5-4, 2-6-1  
**ROR** 2-2-7, 2-9-105  
**Rotate** 3-5-8, 3-5-23  
**RPIX** 2-2-7, 2-8-6, 2-8-9, 2-8-12, 2-9-107

*Index (Continued)***S**

SBC Rn 2-2-6, 2-9-108  
 SBK 2-2-6, 2-9-109  
 SBK Instruction 2-7-2, 2-7-4, 2-7-5  
 Scalar 3-5-29  
 SCR 2-8-14  
 SEX 2-2-7, 2-9-110  
 Shared Memory 1-5-4  
 SM (xx), Rn 2-2-6, 2-9-112  
 SMS (yy), Rn 2-2-6, 2-9-113  
 Sprite Rotation 2-8-11  
 Sprite Scaling 2-8-11  
 STB(Rn) 2-2-6, 2-9-115  
 STOP 2-2-8, 2-9-116  
 STW (Rn) 2-2-6, 2-9-117  
 SUB #n 2-2-6, 2-9-119  
 SUB Rn 2-2-6, 2-9-118  
 Subjective 3-5-22, 3-5-27  
 Super MMC 1-1-1, 1-3-3, 1-3-4  
 SWAP 2-2-7, 2-9-120

**X**

XOR #n 2-2-7, 2-9-126  
 XOR Rn 2-2-7, 2-9-125

**T**

Target 3-5-17, 3-5-20, 3-5-21  
 TO 2-6-2, 2-6-4, 2-6-6, 2-6-7  
 TO Rn 2-2-8, 2-9-121  
 Transparent 2-8-9, 2-8-10, 2-8-11, 2-8-13  
 Triangle 3-5-3

**U**

UMULT #n 2-2-6, 2-8-16, 2-9-123  
 UMULT Rn 2-2-6, 2-8-16, 2-9-122

**V**

V Counter 4-1-4  
 Variable-length Data 1-8-1, 1-8-4  
 Vector Switching 1-5-4  
 Vertical Counter Latch 4-1-3  
 Virtual VRAM 1-1-2

**W**

WITH 2-6-4, 2-6-6, 2-6-7  
 WITH Rn 2-2-8, 2-9-124