SUPER NINTENDO®
ENTERTAINMENT SYSTEM

DEVELOPMENT MANUAL

BOOK II

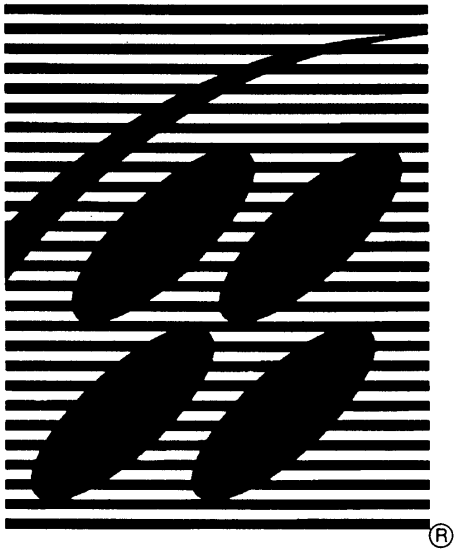


Table of Contents

BOOK II

SUBJECT **PAGE**

Table of Contents (Continued)

SUBJECT **PAGE**

List of Figures

BOOK II

List of Figures (Continued)

List of Figures (Continued)

List of Tables

BOOK II

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 (Σ(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.

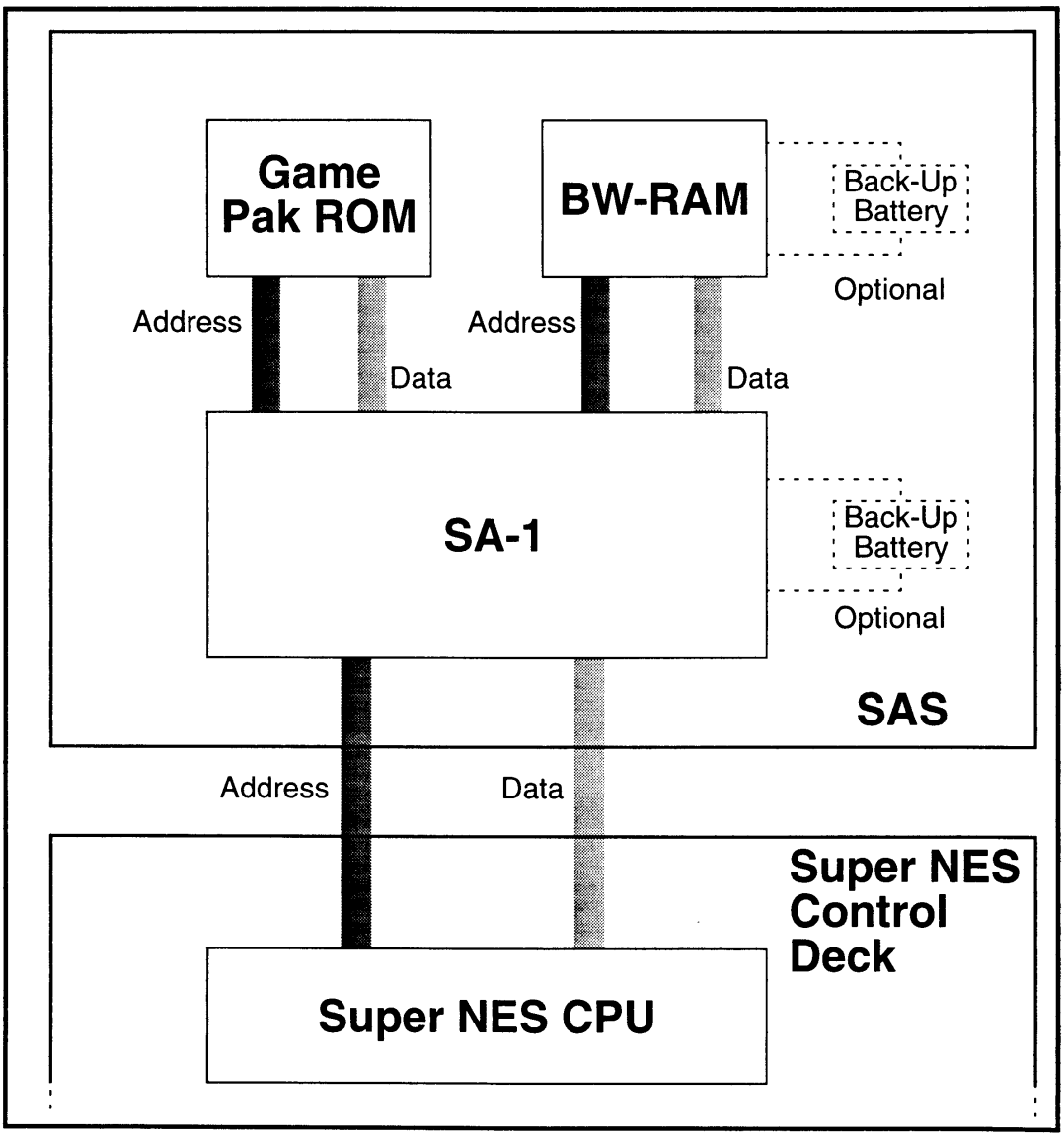


Figure 1-1-1 Super Accelerator System Configuration

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.

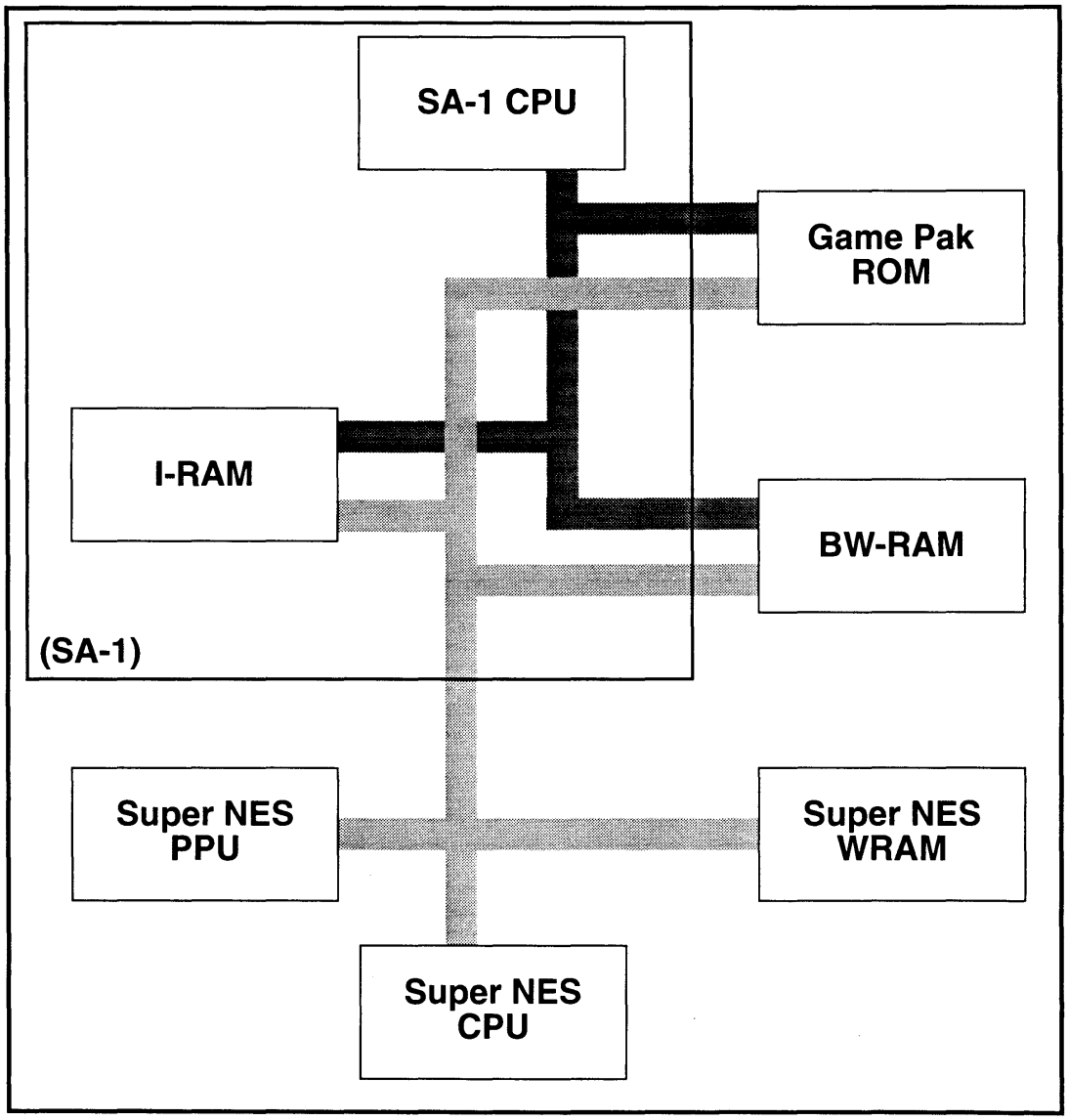


Figure 1-1-2 SAS Bus Image

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.

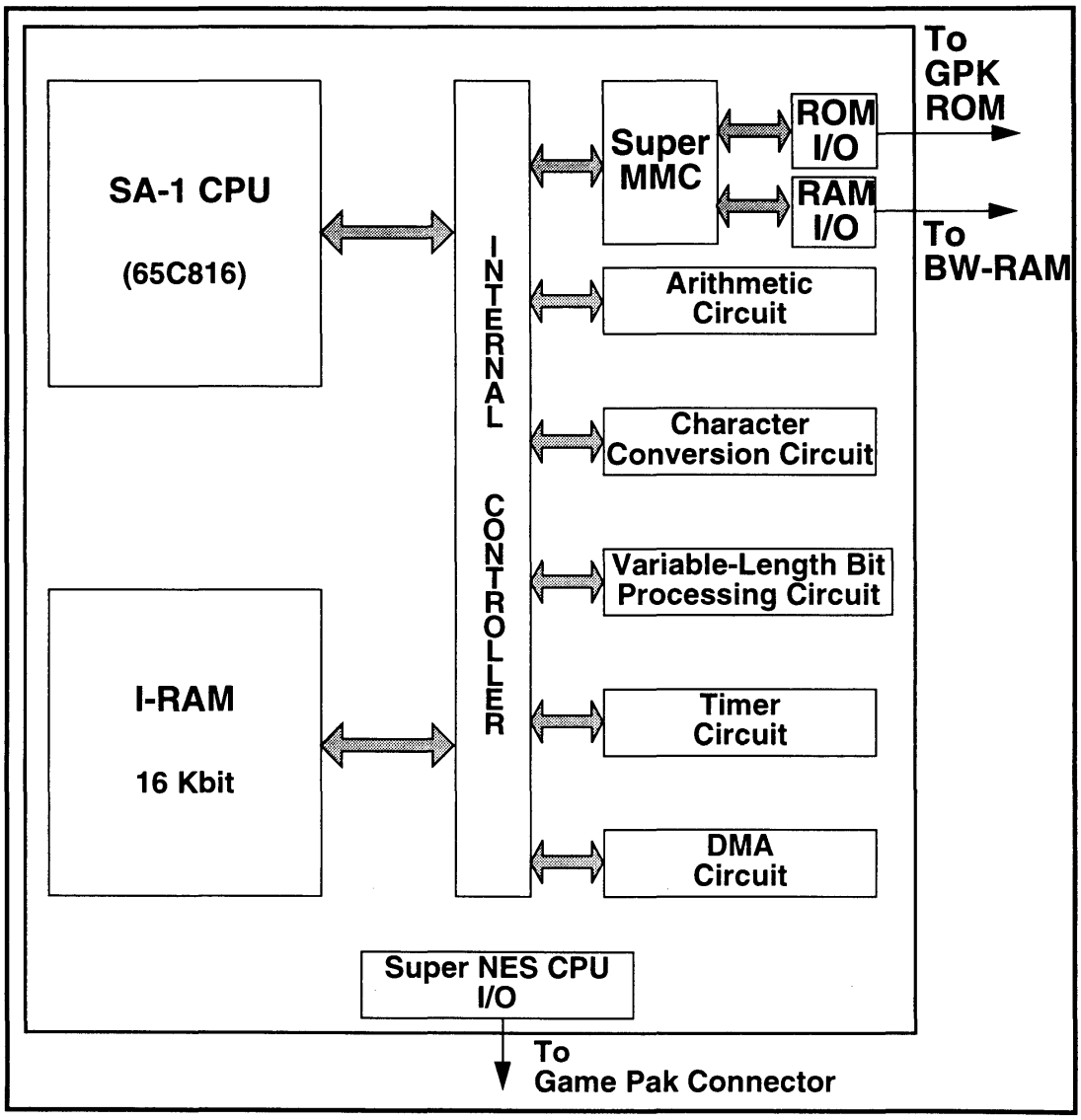


Figure 1-2-1 SA-1 Block Diagram

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

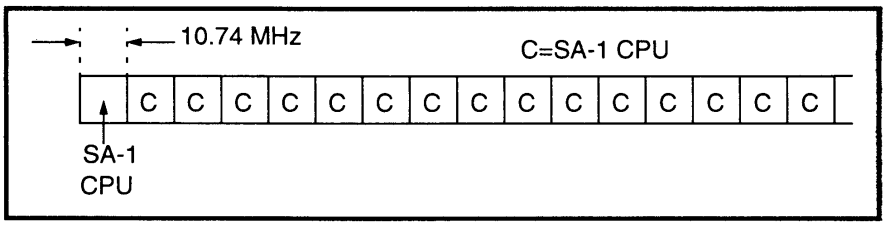
The Super NES CPU and SA-1 CPU share the entire game pak ROM area and can both freely access it. This is known as 2-phase access.

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.

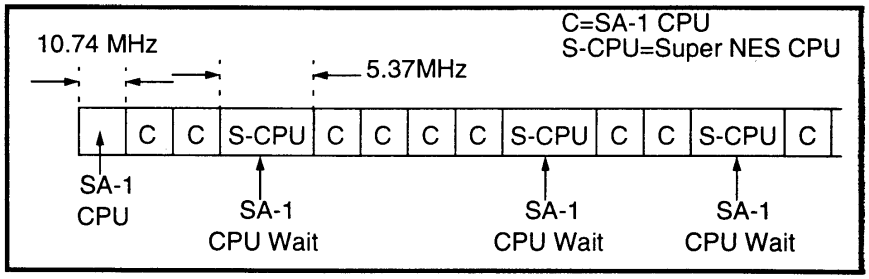
When both the Super NES CPU and SA-1 CPU execute a program on game pak ROM, the SA-1 CPU runs at 5.37 MHz and the Super NES CPU runs at 2.68 MHz.

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.

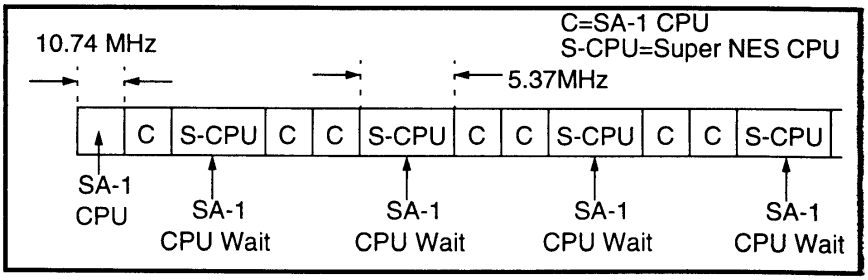
2.2.1.1 ONLY SA-1 CPU USES ROM



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)

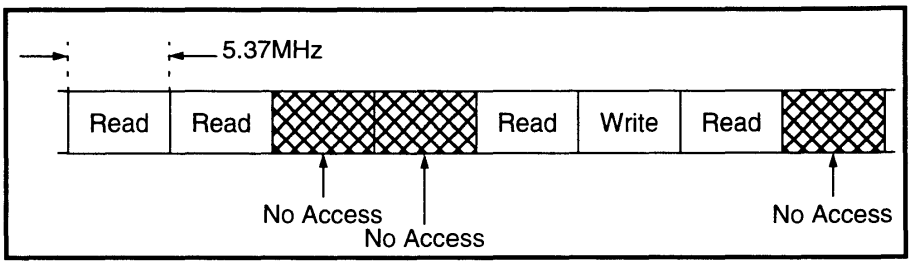


2.2.2 BW-RAM ACCESS

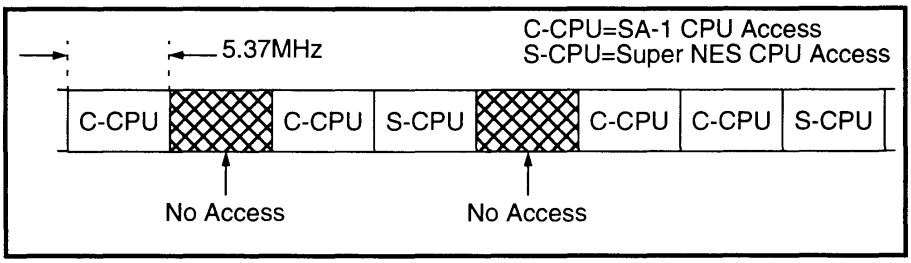
The Super NES CPU and SA-1 CPU share all areas of BW-RAM and can freely access it (two-phase access).

The SA-1 CPU accesses BW-RAM at 5.37 MHz and the Super NES CPU accesses BW-RAM at 2.68 MHz.

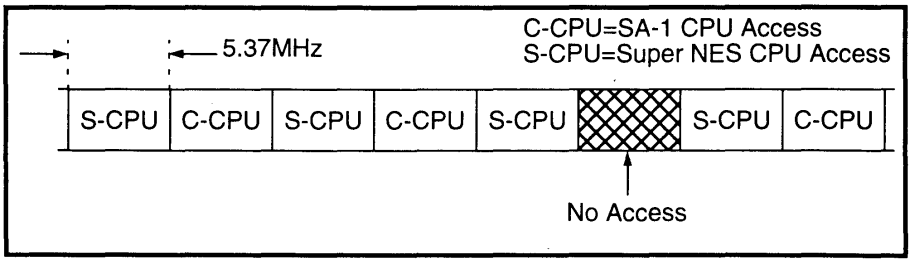
2.2.2.1 ONLY SA-1 CPU USES BW-RAM



2.2.2.2 SUPER NES CPU ACCESSES BW-RAM DURING SA-1 CPU OPERATIONS



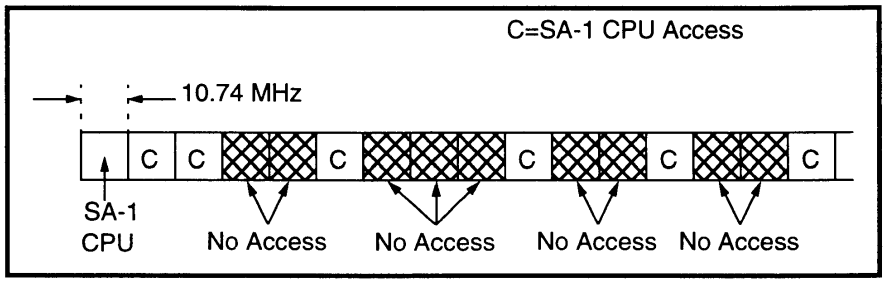
2.2.2.3 BOTH PROCESSORS ACCESS BW-RAM (2-PHASE ACCESS)



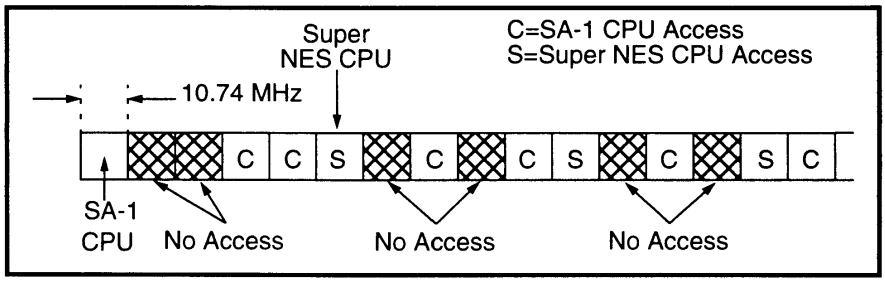
2.2.3 SA-1 I-RAM ACCESS

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



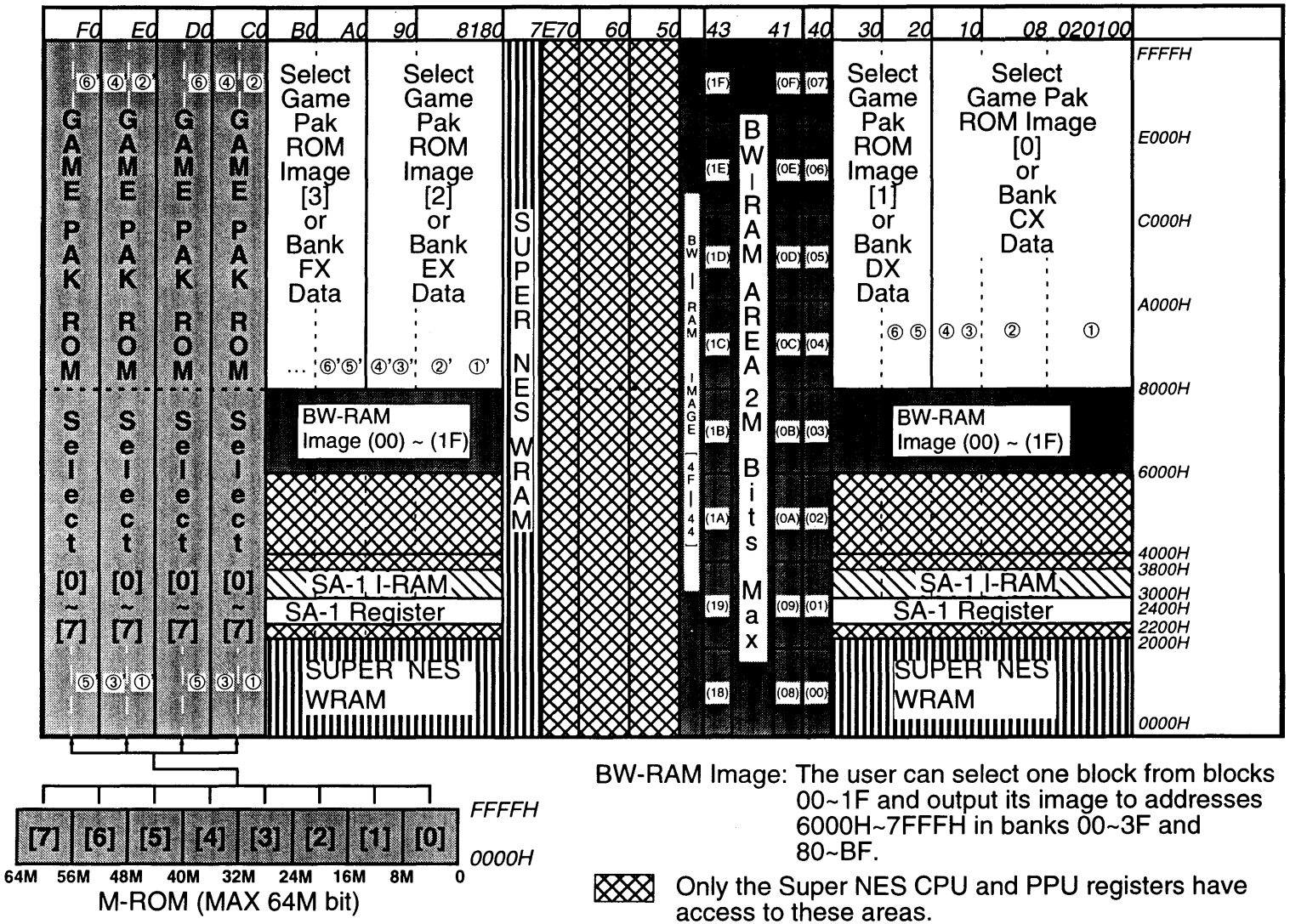
2.2.3.2 BOTH SA-1 CPU AND SUPER NES CPU ACCESS I-RAM



Chapter 3 Super Accelerator Memory Map

3.1 MEMORY MAP FROM SUPER NES CPU PERSPECTIVE

Super NES Memory Map (SAS, Super NES CPU)

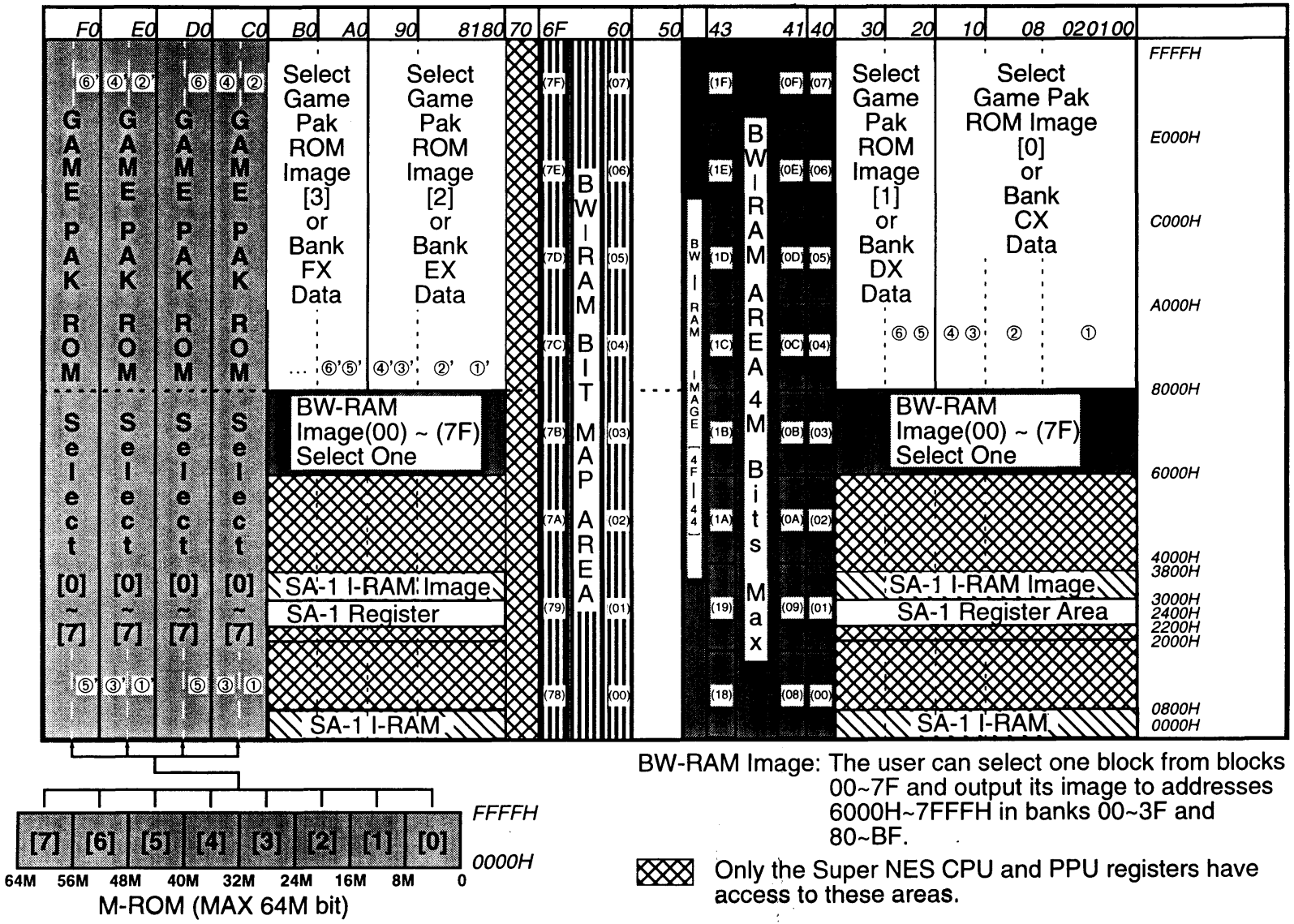


BW-RAM Image: The user can select one block from blocks 00~1F and output its image to addresses 6000H~7FFFH in banks 00~3F and 80~BF.

Only the Super NES CPU and PPU registers have access to these areas.

3.2 MEMORY MAP FROM SA-1 CPU PERSPECTIVE

Super NES Memory Map (SAS, SA-1 CPU)



BW-RAM Image: The user can select one block from blocks 00~7F and output its image to addresses 6000H~7FFFH in banks 00~3F and 80~BF.

Only the Super NES CPU and PPU registers have access to these areas.

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~40:07FFH
64K RAM:	40:0000H~40:1FFFH
256K RAM:	40:0000H~40:7FFFH
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	
		SA-1 CPU Reset Vector						2203H
CRV7	CRV6	CRV5	CRV4	CRV3	CRV2	CRV1	CRV0	
		SA-1 CPU Reset Vector						2204H
CRV15	CRV14	CRV13	CRV12	CRV11	CRV10	CRV9	CRV8	

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	
		SA-1 CPU NMI Vector (Low)						2205H
CNV7	CNV6	CNV5	CNV4	CNV3	CNV2	CNV1	CNV0	
		SA-1 CPU NMI Vector (High)						2206H
CNV15	CNV14	CNV13	CNV12	CNV11	CNV10	CNV9	CNV8	

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	
		SA-1 CPU IRQ Vector (Low)						2207H
CIV7	CIV6	CIV5	CIV4	CIV3	CIV2	CIV1	CIV0	
		SA-1 CPU IRQ Vector (High)						2208H
CIV15	CIV14	CIV13	CIV12	CIV11	CIV10	CIV9	CIV8	

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)								220CH
SNV7	SNV6	SNV5	SNV4	SNV3	SNV2	SNV1	SNV0	
Super NES CPU NMI Vector (High)								220DH
SNV15	SNV14	SNV13	SNV12	SNV11	SNV10	SNV9	SNV8	

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)								220EH
SIV7	SIV6	SIV5	SIV4	SIV3	SIV2	SIV1	SIV0	
Super NES CPU IRQ Vector (High)								220FH
SIV15	SIV14	SIV13	SIV12	SIV11	SIV10	SIV9	SIV8	

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	
			H-Count (Low)					2212H
H7	H6	H5	H4	H3	H2	H1	H0	
			H-Count (High)					2213H
0	0	0	0	0	0	0	H8	

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	
			V-Count (Low)					2214H
V7	V6	V5	V4	V3	V2	V1	V0	
			V-Count (High)					2215H
0	0	0	0	0	0	0	V8	

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)

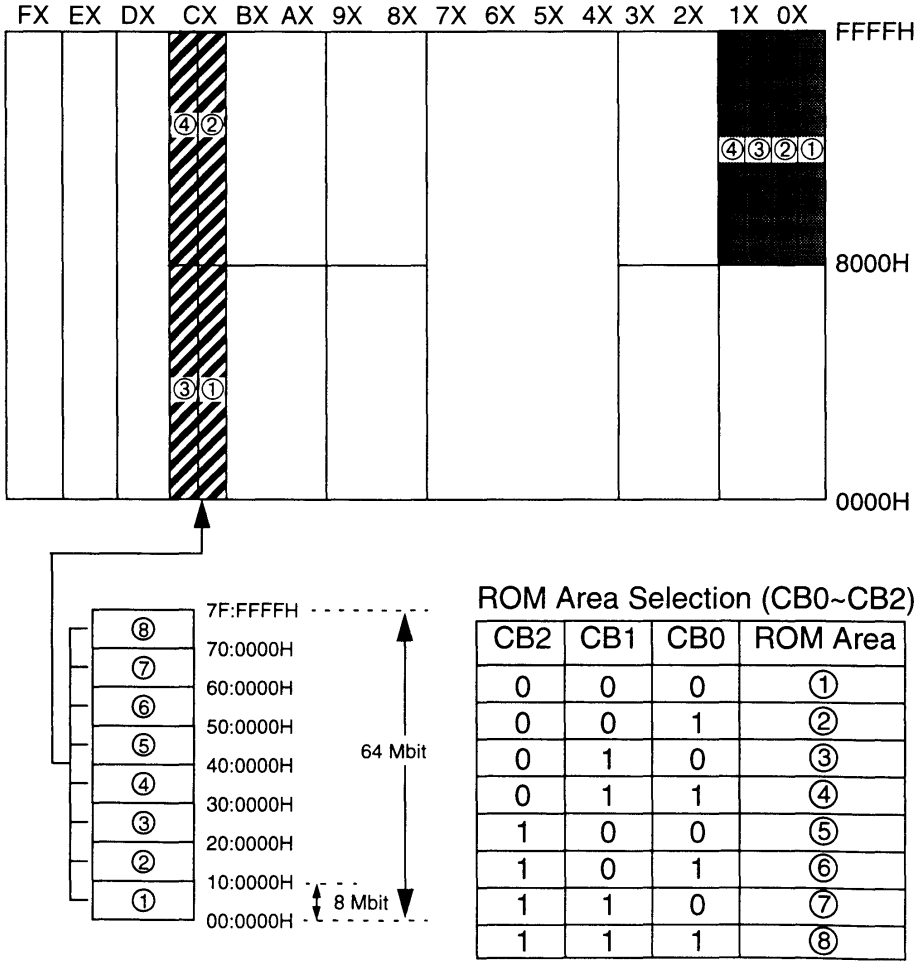
Access: Super NES CPU Write
Address: **2220H
Size: 8 bits
Initial value: 00H

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.



ROM Area Selection (CB0~CB2)

CB2	CB1	CB0	ROM Area
0	0	0	①
0	0	1	②
0	1	0	③
0	1	1	④
1	0	0	⑤
1	0	1	⑥
1	1	0	⑦
1	1	1	⑧

4.1.17 SET SUPER MMC BANK D (DXB)

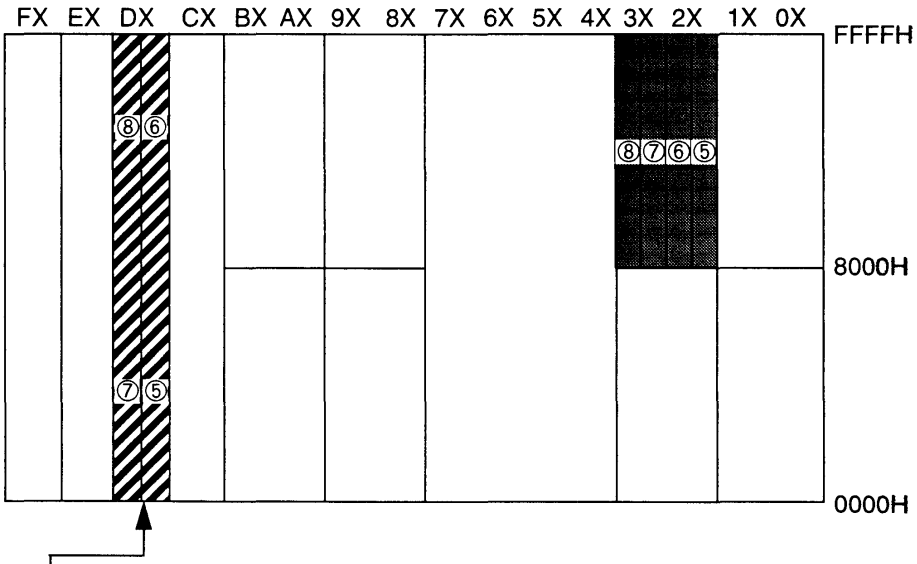
Access: Super NES CPU Write
Address: **2221H
Size: 8 bits
Initial value: 01H

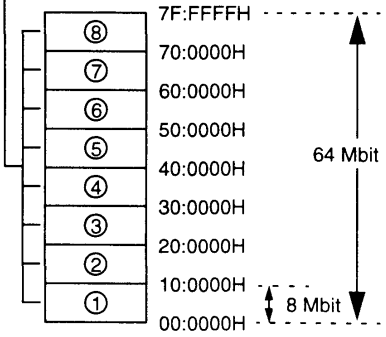
D7	D6	D5	D4	D3	D2	D1	D0	
DBM	0	0	0	0	CB2	CB1	CB0	2221H

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.





ROM Area Selection (CB0~CB2)

CB2	CB1	CB0	ROM Area
0	0	0	①
0	0	1	②
0	1	0	③
0	1	1	④
1	0	0	⑤
1	0	1	⑥
1	1	0	⑦
1	1	1	⑧

4.1.18 SET SUPER MMC BANK E (EXB)

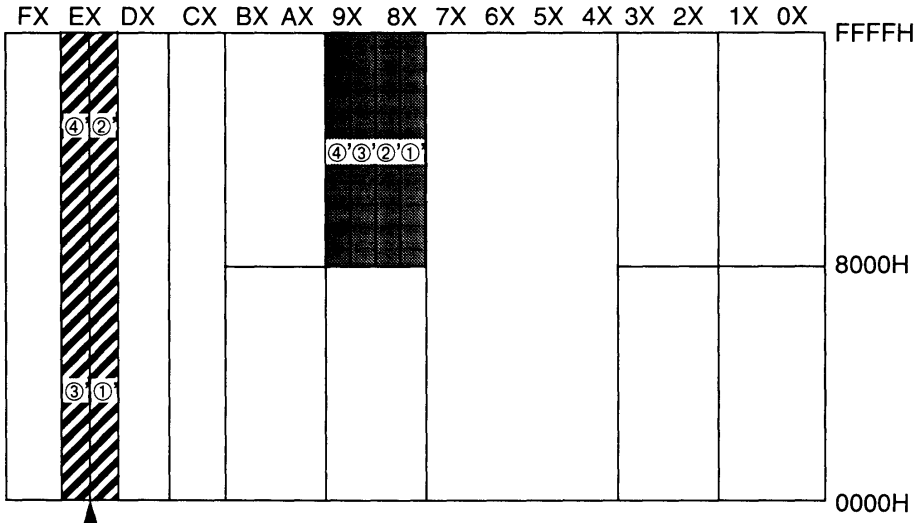
Access: Super NES CPU Write
Address: **2222H
Size: 8 bits
Initial value: 02H

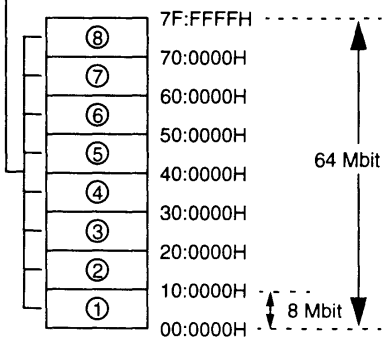
D7	D6	D5	D4	D3	D2	D1	D0	
EBM	0	0	0	0	CB2	CB1	CB0	2222H

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.





ROM Area Selection (CB0~CB2)

CB2	CB1	CB0	ROM Area
0	0	0	①
0	0	1	②
0	1	0	③
0	1	1	④
1	0	0	⑤
1	0	1	⑥
1	1	0	⑦
1	1	1	⑧

4.1.19 SET SUPER MMC BANK F (FXB)

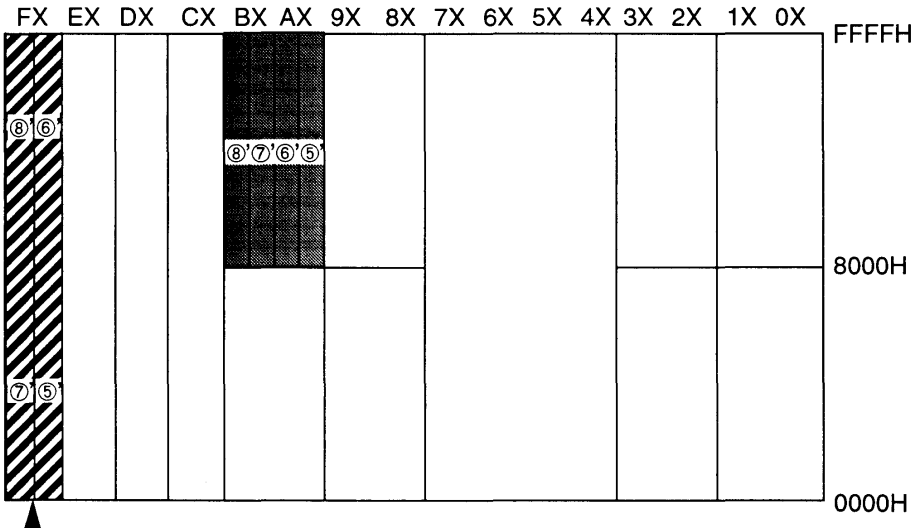
Access: Super NES CPU Write
Address: **2223H
Size: 8 bits
Initial value: 03H

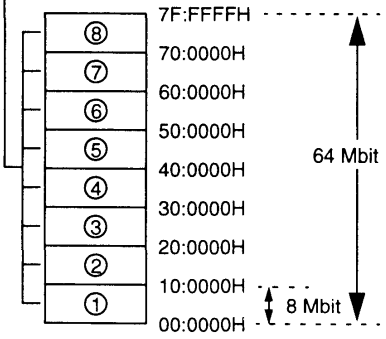
D7	D6	D5	D4	D3	D2	D1	D0	
FBM	0	0	0	0	CB2	CB1	CB0	2223H

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.





ROM Area Selection (CB0~CB2)

CB2	CB1	CB0	ROM Area
0	0	0	①
0	0	1	②
0	1	0	③
0	1	1	④
1	0	0	⑤
1	0	1	⑥
1	1	0	⑦
1	1	1	⑧

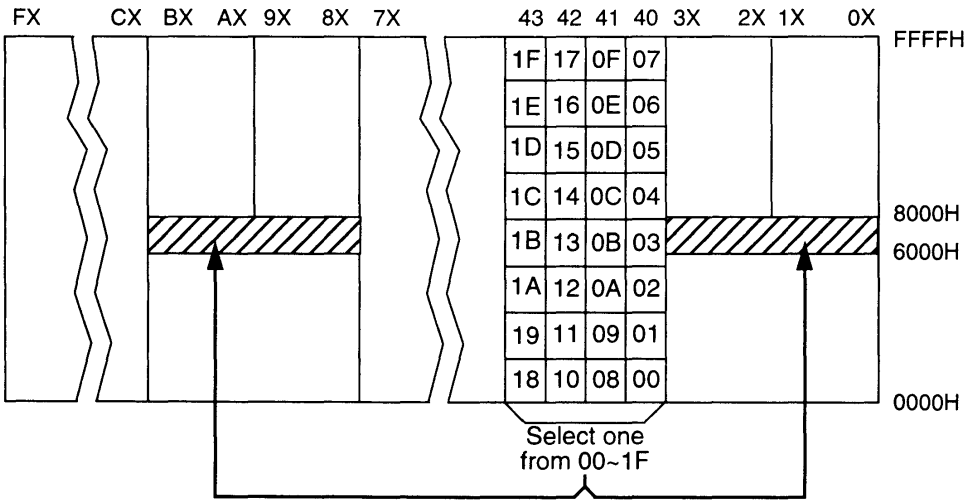
4.1.20 SUPER NES CPU BW-RAM ADDRESS MAPPING (BMAPS)

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)

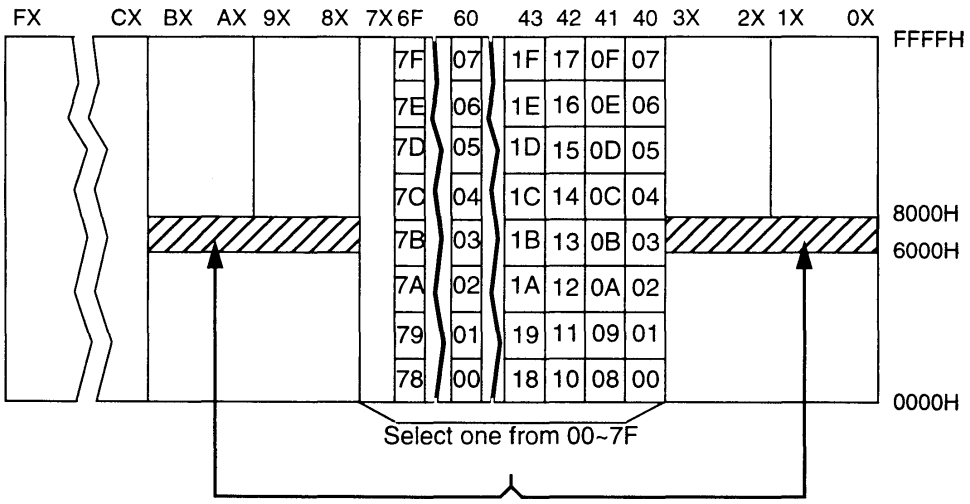
Access: SA-1 CPU Write
Address: **2225H
Size: 8 bits
Initial value: 00H

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)

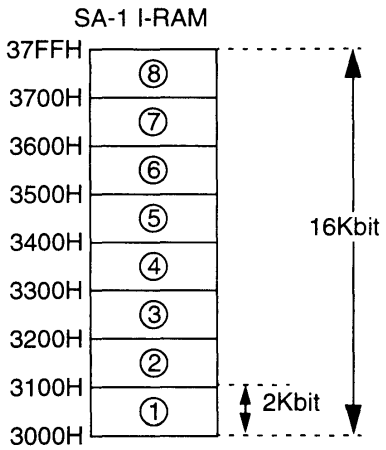
Access: Super NES CPU Write
Address: **2229H
Size: 8 bits
Initial value: 00H

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)

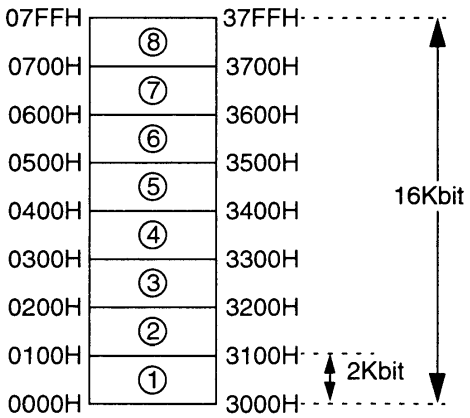
Access: SA-1 CPU Write
Address: **222AH
Size: 8 bits
Initial value: 00H

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	
DMA Source Device A Start Address (Low)								2232H
DSA7	DSA6	DSA5	DSA4	DSA3	DSA2	DSA-1	DSA0	
DMA Source Device A Start Address (Middle)								2233H
DSA-15	DSA-14	DSA-13	DSA-12	DSA-11	DSA-10	DSA9	DSA8	
DMA Source Device A Start Address (High)								2234H
DSA23	DSA22	DSA21	DSA20	DSA-19	DSA-18	DSA-17	DSA-16	

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	
DMA Destination Device Start Address (Low)								2235H
DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	
DMA Destination Device Start Address (Middle)								2236H
DDA15	DDA14	DDA13	DDA12	DDA11	DDA10	DDA9	DDA8	
DMA Destination Device Start Address (High)								2237H
DDA23	DDA22	DDA21	DDA20	DDA19	DDA18	DDA17	DDA16	

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	
DMA Terminal Counter (Low)								2238H
T7	T6	T5	T4	T3	T2	T1	T0	
DMA Terminal Counter (High)								2239H
T15	T14	T13	T12	T11	T10	T9	T8	

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	
Bitmap Register File 0								
BM07	BM06	BM05	BM04	BM03	BM02	BM01	BM00	2240H
Bitmap Register File 1								
BM17	BM16	BM15	BM14	BM13	BM12	BM11	BM10	2241H
Bitmap Register File 2								
BM27	BM26	BM25	BM24	BM23	BM22	BM21	BM20	2242H
Bitmap Register File 3								
BM37	BM36	BM35	BM34	BM33	BM32	BM31	BM30	2243H
Bitmap Register File 4								
BM47	BM46	BM45	BM44	BM43	BM42	BM41	BM40	2244H
Bitmap Register File 5								
BM57	BM56	BM55	BM54	BM53	BM52	BM51	BM50	2245H
Bitmap Register File 6								
BM67	BM66	BM65	BM64	BM63	BM62	BM61	BM60	2246H
Bitmap Register File 7								
BM77	BM76	BM75	BM74	BM73	BM72	BM71	BM70	2247H

Figure 1-4-4 Bitmap Register Files 0 ~ 7

		Bitmap Register File 8						2248H
BM87	BM86	BM85	BM84	BM83	BM82	BM81	BM80	
		Bitmap Register File 9						2249H
BM97	BM96	BM95	BM94	BM93	BM92	BM91	BM90	
		Bitmap Register File A						224AH
BMA7	BMA6	BMA5	BMA4	BMA3	BMA2	BMA1	BMA0	
		Bitmap Register File B						224BH
BMB7	BMB6	BMB5	BMB4	BMB3	BMB2	BMB1	BMB0	
		Bitmap Register File C						224CH
BMC7	BMC6	BMC5	BMC4	BMC3	BMC2	BMC1	BMC0	
		Bitmap Register File D						224DH
BMD7	BMD6	BMD5	BMD4	BMD3	BMD2	BMD1	BMD0	
		Bitmap Register File E						224EH
BME7	BME6	BME5	BME4	BME3	BME2	BME1	BME0	
		Bitmap Register File F						224FH
BMF7	BMF6	BMF5	BMF4	BMF3	BMF2	BMF1	BMF0	

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)								2259H
VA7	VA6	VA5	VA4	VA3	VA2	VA1	VA0	
Variable-Length Bit Game Pak ROM Start Address (Middle)								225AH
VA15	VA14	VA13	VA12	VA11	VA10	VA9	VA8	
Variable-Length Bit Game Pak ROM Start Address (High)								225BH
VA23	VA22	VA21	VA20	VA19	VA18	VA17	VA16	

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	
		Timer H-Count Read (Low)						2302H
H7	H6	H5	H4	H3	H2	H1	H0	
		Timer H-Count Read (High)						2303H
--	--	--	--	--	--	--	H8	

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	
		Timer V-Count Read (Low)						2304H
V7	V6	V5	V4	V3	V2	V1	V0	
		Timer V-Count Read (High)						2305H
--	--	--	--	--	--	--	V8	

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								2306H
D7	D6	D5	D4	D3	D2	D1	D0	
Read Arithmetic Result (product/quotient/cumulative sum) W1								2307H
D15	D14	D13	D12	D11	D10	D9	D8	
Read Arithmetic Result (product/remainder/cumulative sum) W2								2308H
D23	D22	D21	D20	D19	D18	D17	D16	
Read Arithmetic Result (product/remainder/cumulative sum) W3								2309H
D31	D30	D29	D28	D27	D26	D25	D24	
Read Arithmetic Result (cumulative sum) W4								230AH
D39	D38	D37	D36	D35	D34	D33	D32	

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) x 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)								230CH
VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	
Variable-Length Data Read Port (High)								230DH
VD15	VD14	VD13	VD12	VD11	VD10	VD9	VD8	

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.

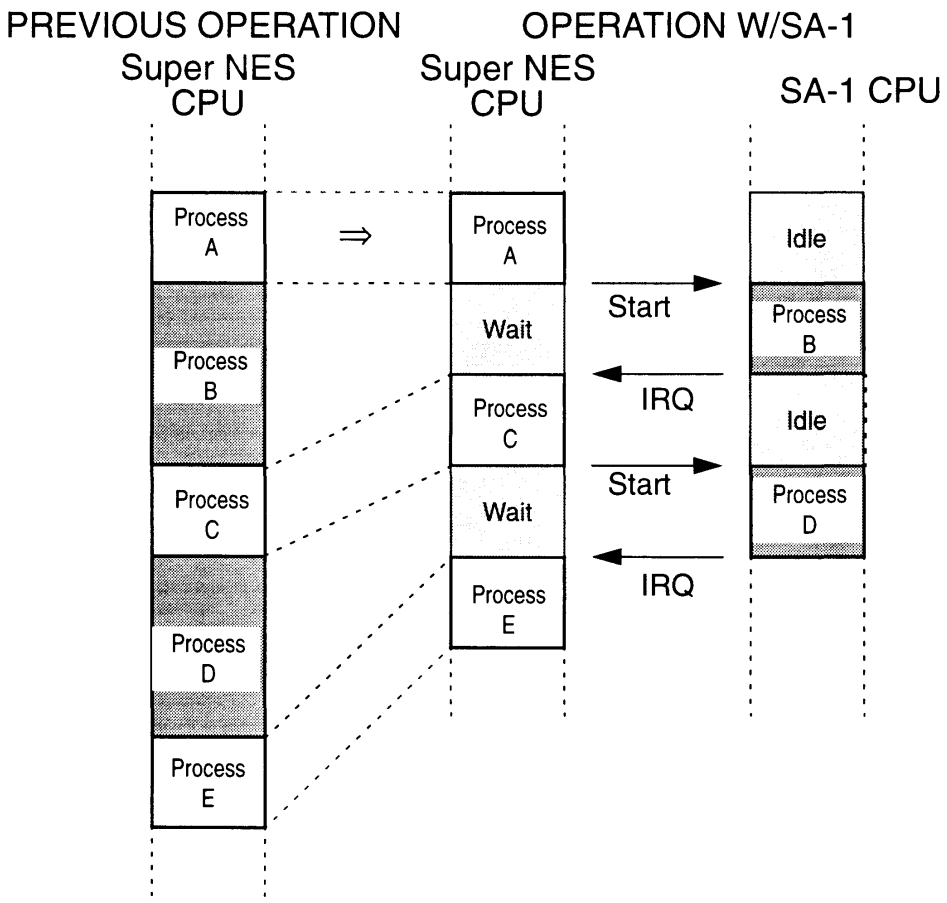


Figure 1-5-1 Accelerator Mode

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 hand-shakes. 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.

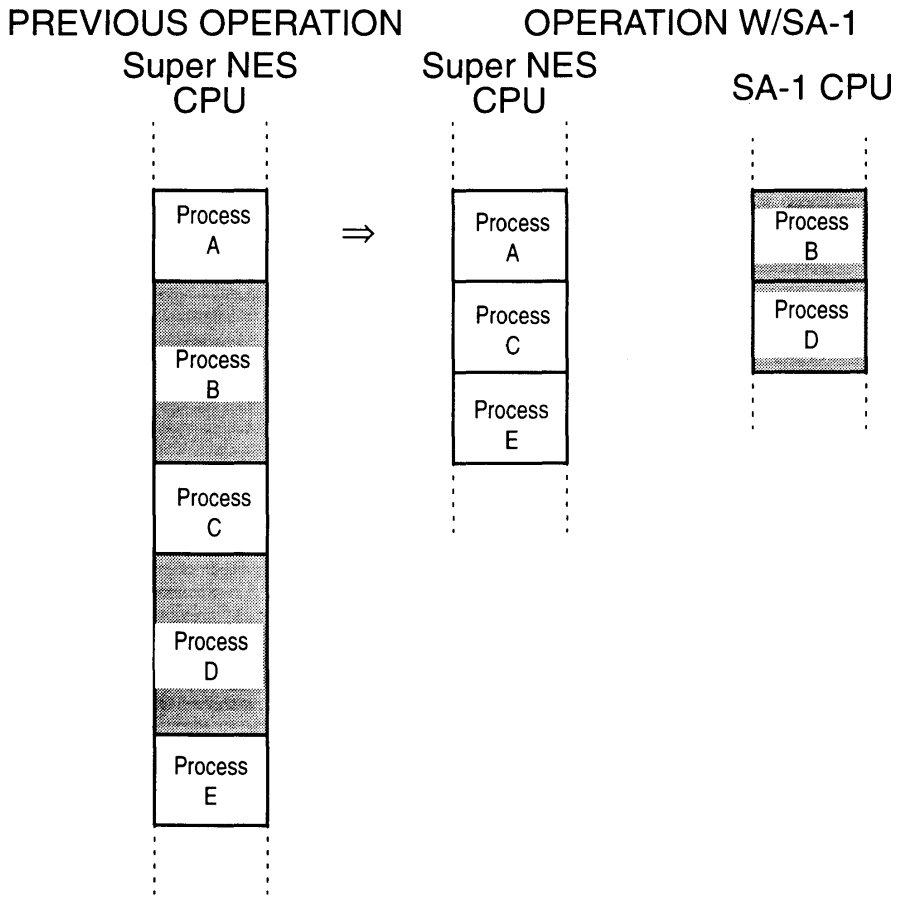


Figure 1-5-2 Parallel Processing Mode

In the parallel processing mode the highest processing efficiency can be achieved, as both MPUs operate without waiting for one anther. 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.

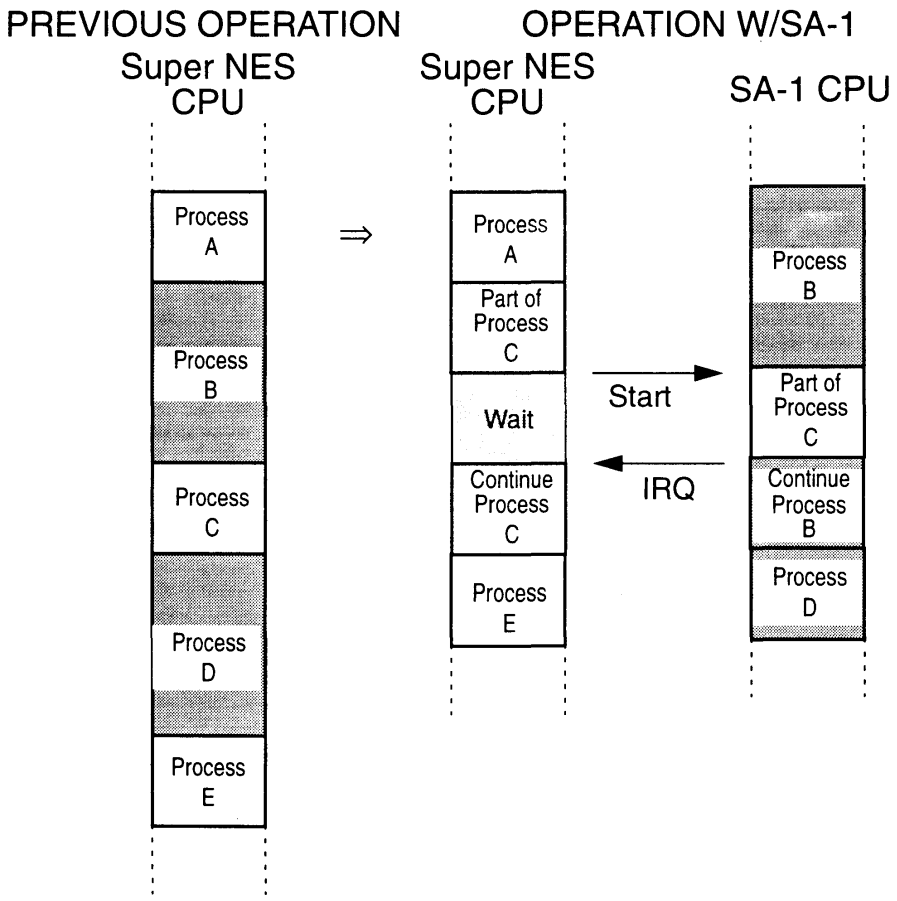


Figure 1-5-3 Mixed Processing Mode

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.

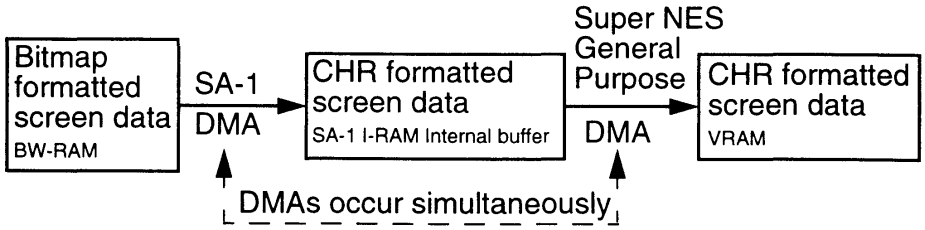


Figure 1-6-1 Character Conversion 1

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,

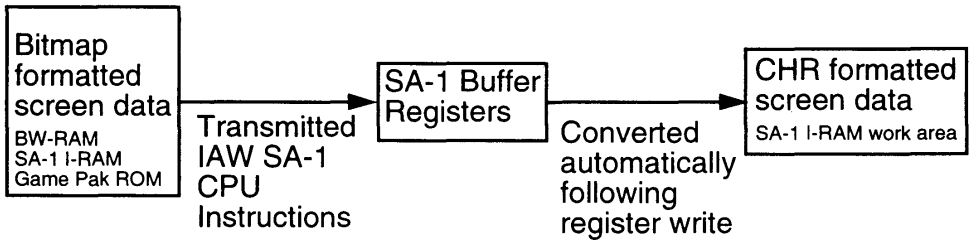


Figure 1-6-2 Character Conversion 2

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.

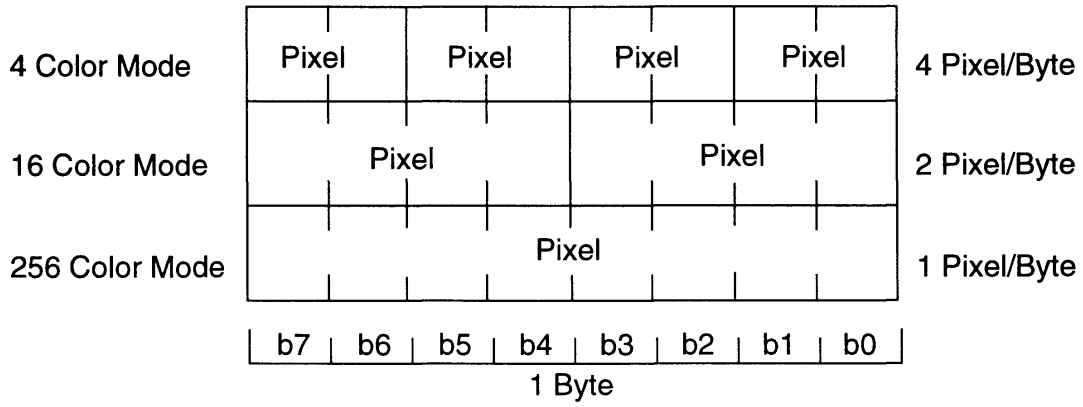


Figure 1-6-3 Compressed Bitmap Data

6.3.1 BW-RAM IMAGE PROJECTION

Within the SA-1, the BW-RAM image is projected into 6 x H banks in the SA-1 CPU's memory map. When BW-RAM is accessed in these 6 x H banks, it can be accessed at one pixel per byte in either the 4 color or 16 color modes.

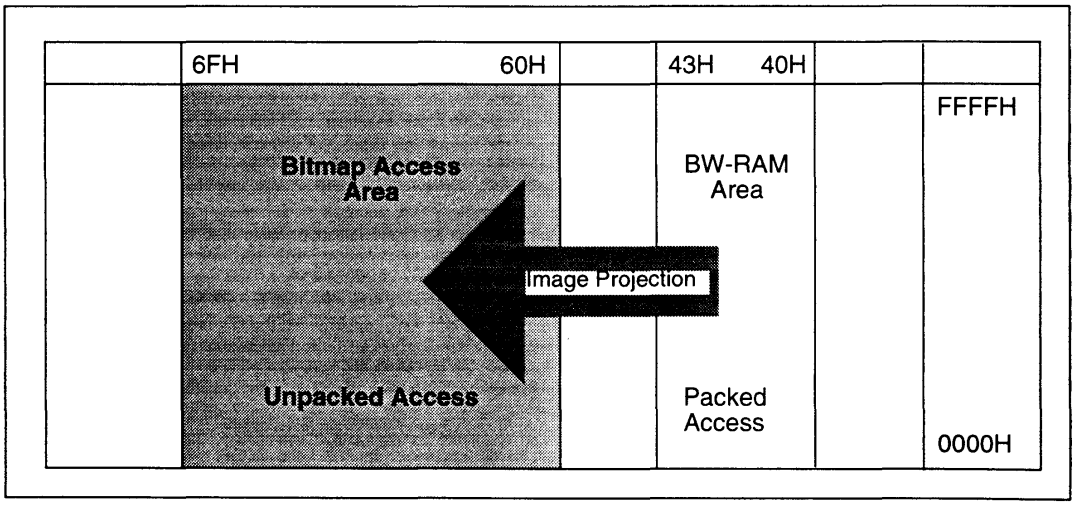


Figure 1-6-4 Bitmap Image Projection

For 64 Kbit BW-RAM:

BW-RAM		Bitmap (16 color)	/ Bitmap (4 color)
40:0000H~40:1FFFH	⇒	60:0000H~60:3FFFH	/ 60:0000H~60:7FFFH

For 256 Kbit BW-RAM:

BW-RAM		Bitmap (16 color)	/ Bitmap (4 color)
40:0000H~40:7FFFH	⇒	60:0000H~60:FFFFH	/ 60:0000H~61:FFFFH

For 2 Mbit BW-RAM:

BW-RAM		Bitmap (16 color)	/ Bitmap (4 color)
40:0000H~43:FFFFH	⇒	60:0000H~67:FFFFH	/ 60:0000H~6F:FFFFH

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.

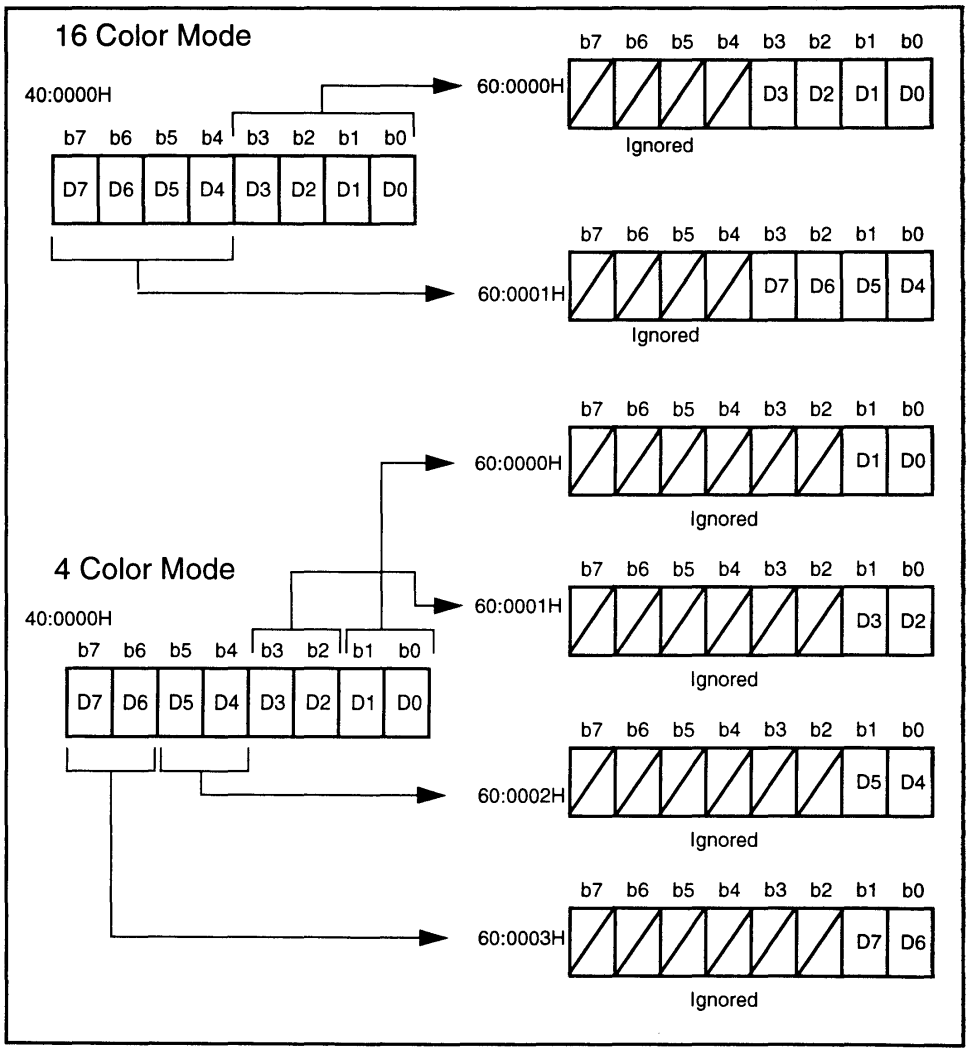


Figure 1-6-5 Bitmap Data Expansion

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.

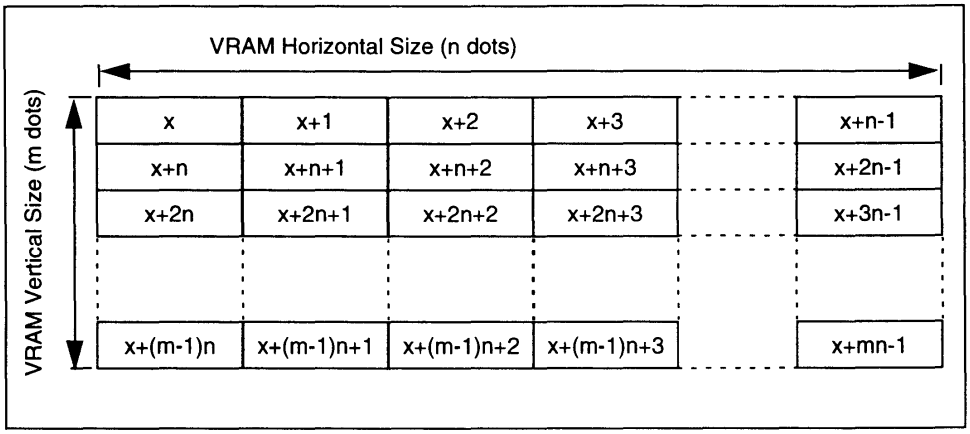


Figure 1-6-6 Memory Addresses for the Bitmap Area

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.

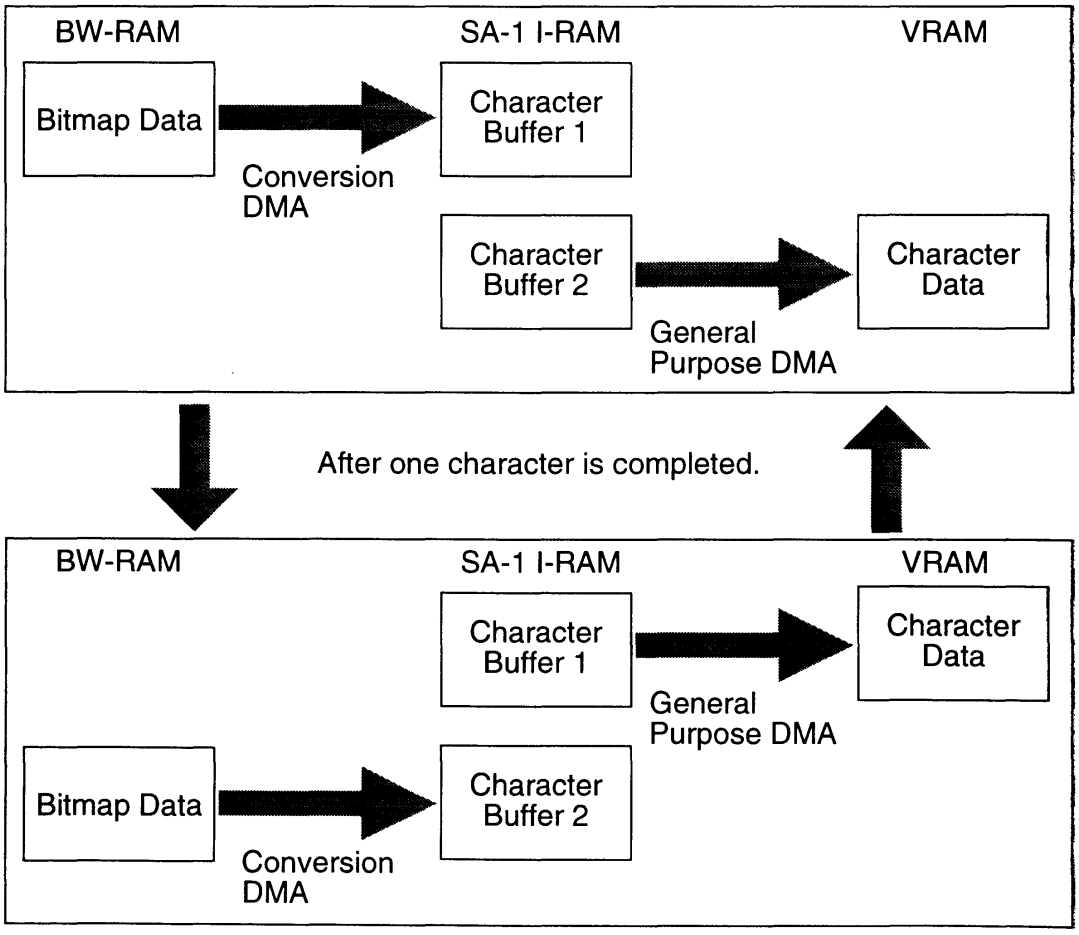


Figure 1-6-7 Character Conversion Buffers

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	16	256	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	32
Zero Bits	4	5	6	7	8	9	5	6	7	8	9	10	6	7	8	9	10	11

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, or256) 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 (CHRIRQ) 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 SA! 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 arithmentc circuit installed in the Super NES PPU. The SA-1 arithmentc 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)	X	16 bits (S)	=	32 bits (S)

2. DIVISION

Dividend		Divisor		Result
16 bits (S)	÷	16 bits (U)	=	16 bits (S) 16 bits (U) Remainder

3. CUMULATIVE SUM

Multiplicand		Multiplier		Result
Σ(16 bits (S)	X	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.

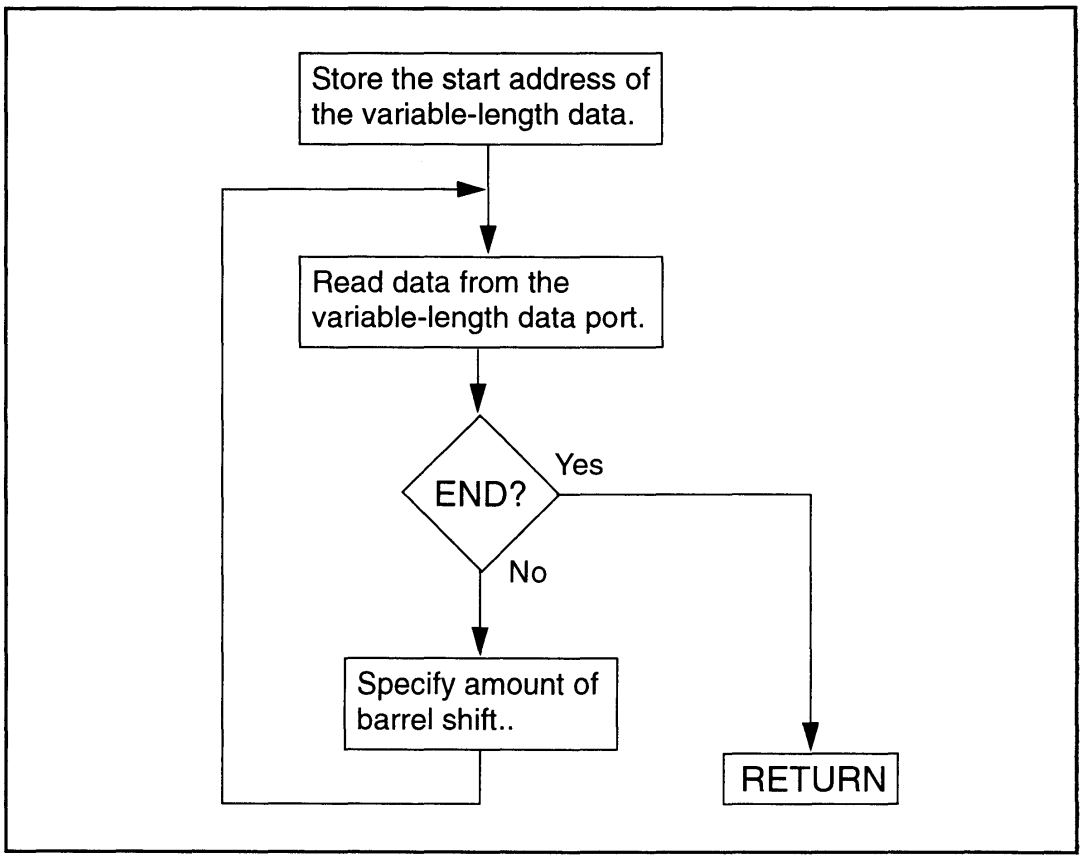


Figure 1-8-1 Fixed Mode Process Flow Diagram

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.

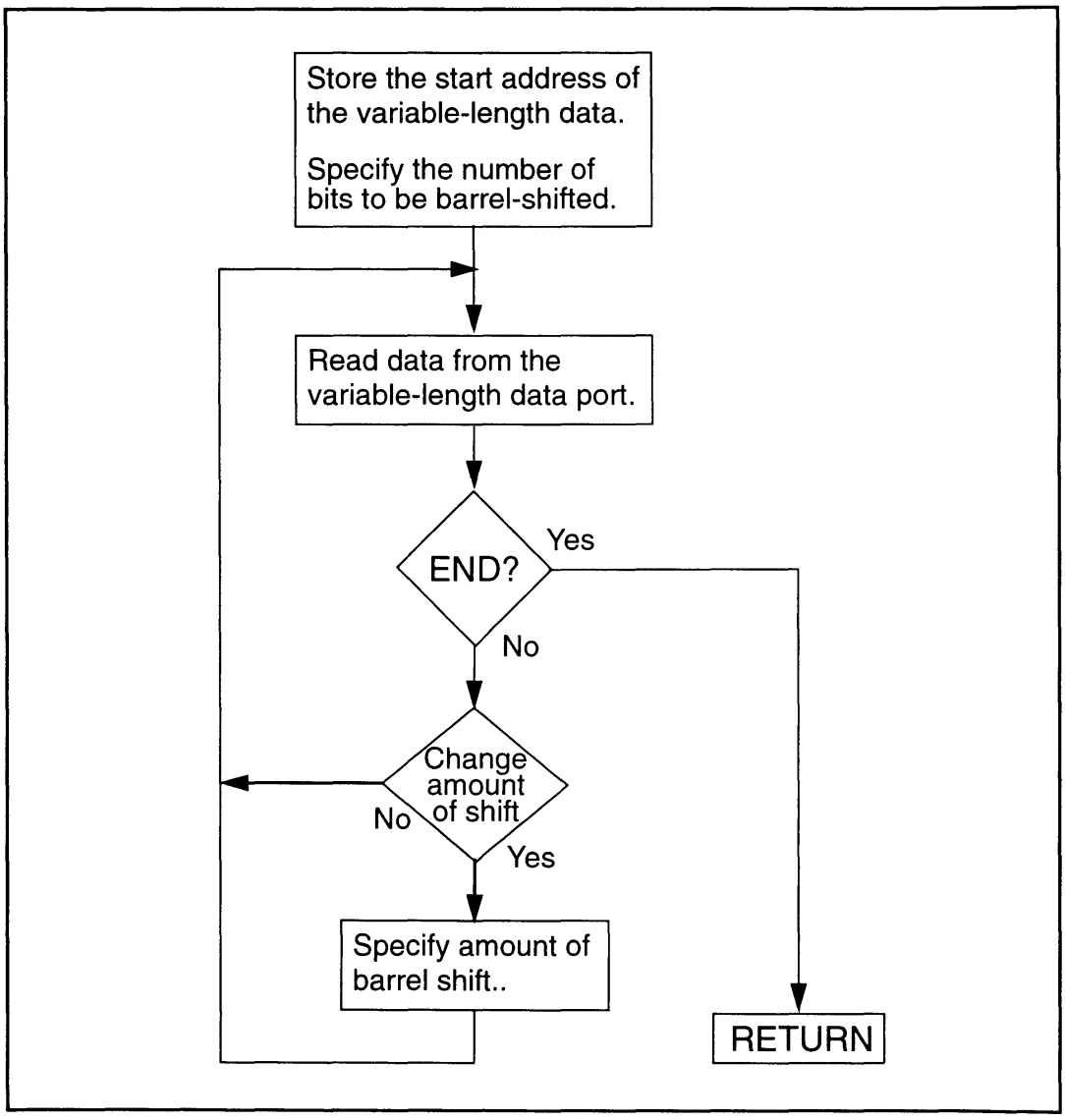


Figure 1-8-2 Auto-increment Mode Process Flow Diagram

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.

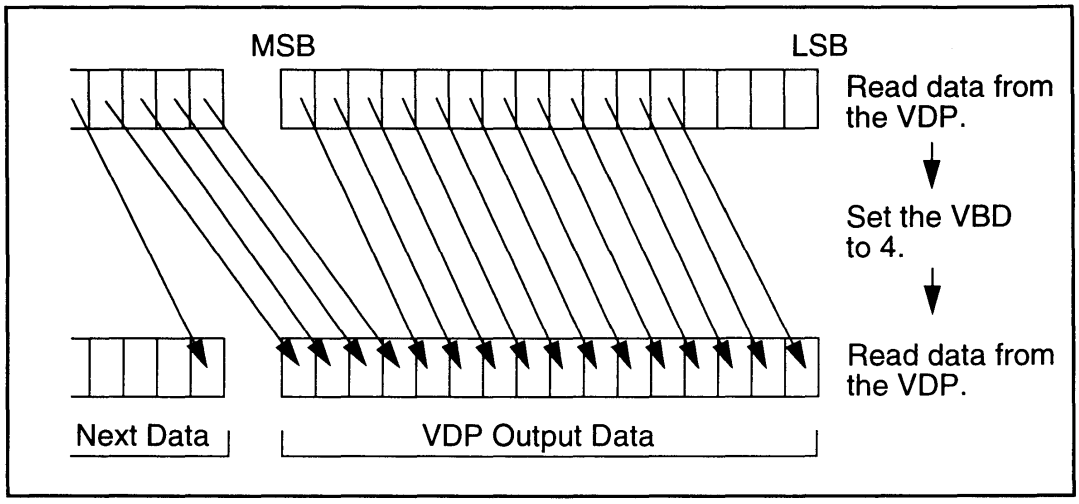


Figure 1-8-3 Barrel Shift Process

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.

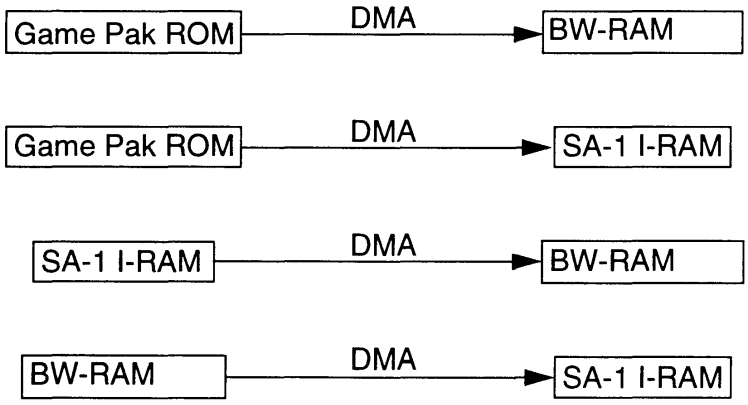


Figure 1-9-1 Normal DMA

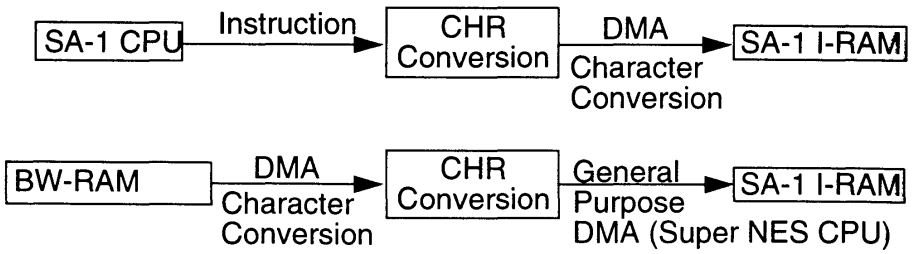


Figure 1-9-2 Character Conversion DMA

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

SD1	SD0	Device
0	0	Game Pak ROM
0	1	BW-RAM
1	0	SA-1 I-RAM

Destination Device

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.

DPrio=0: SA-1 CPU priority (Instructions can be executed during transmission)
DPrio=1: DMA priority (SA-1 CPU waits during DMA)

NOTE: The DPrio 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™

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.

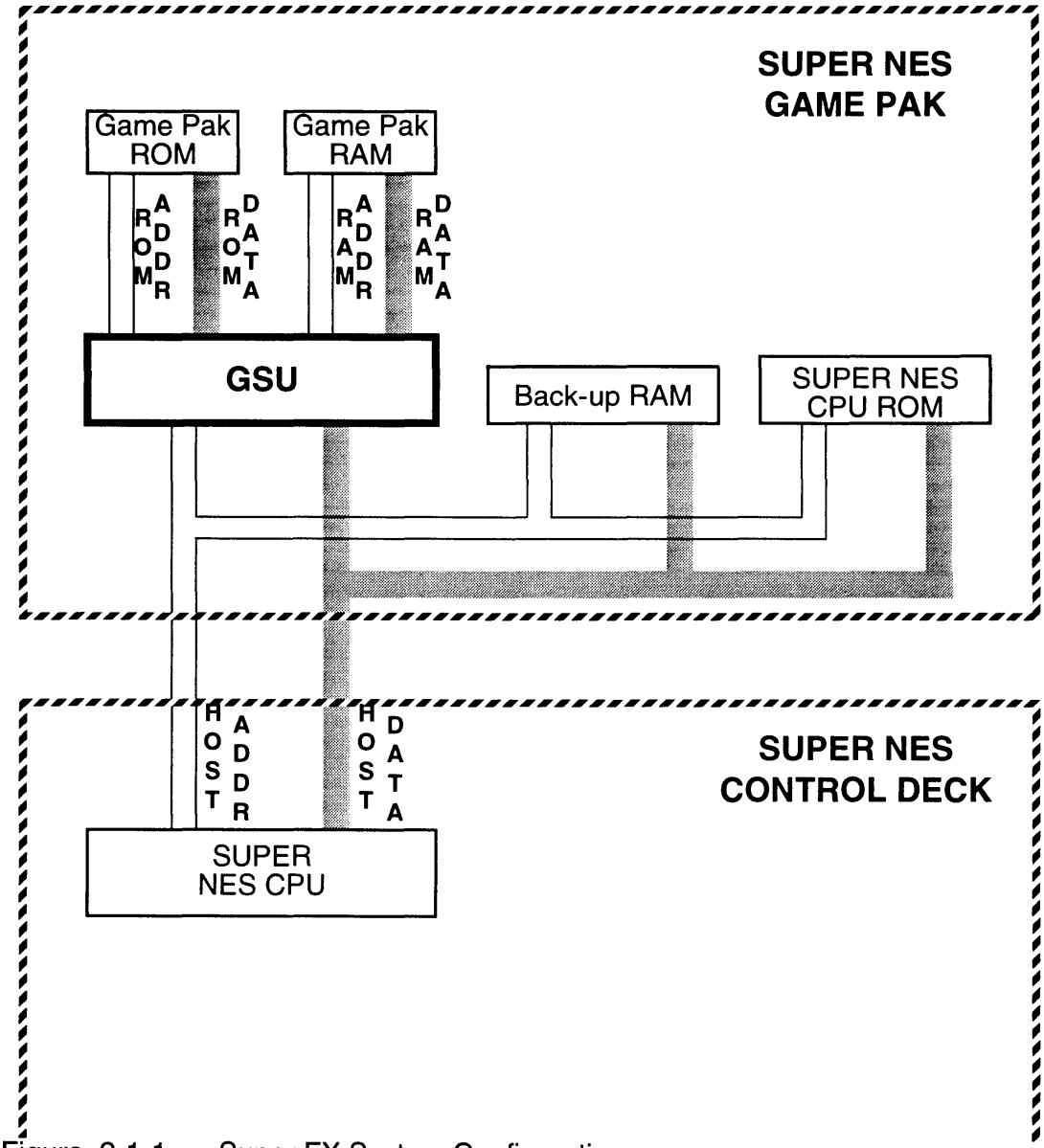


Figure 2-1-1 Super FX System Configuration

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.

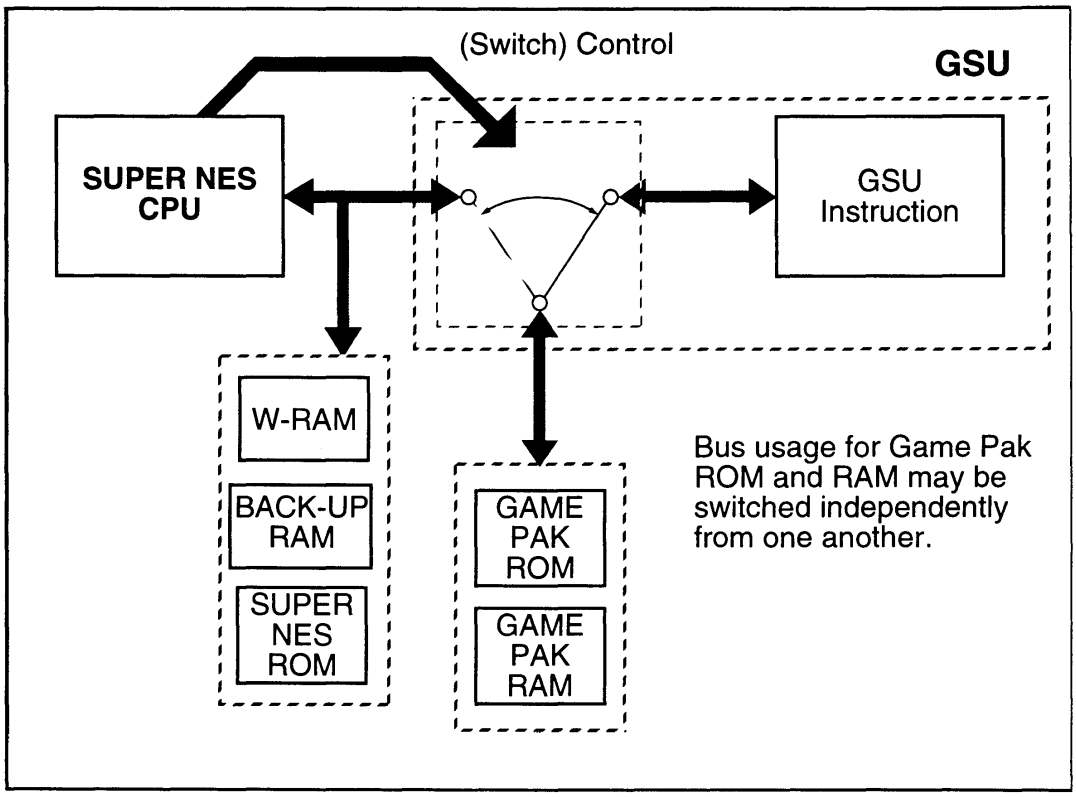


Figure 2-1-2 Game Pak ROM/RAM Bus Diagram

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.

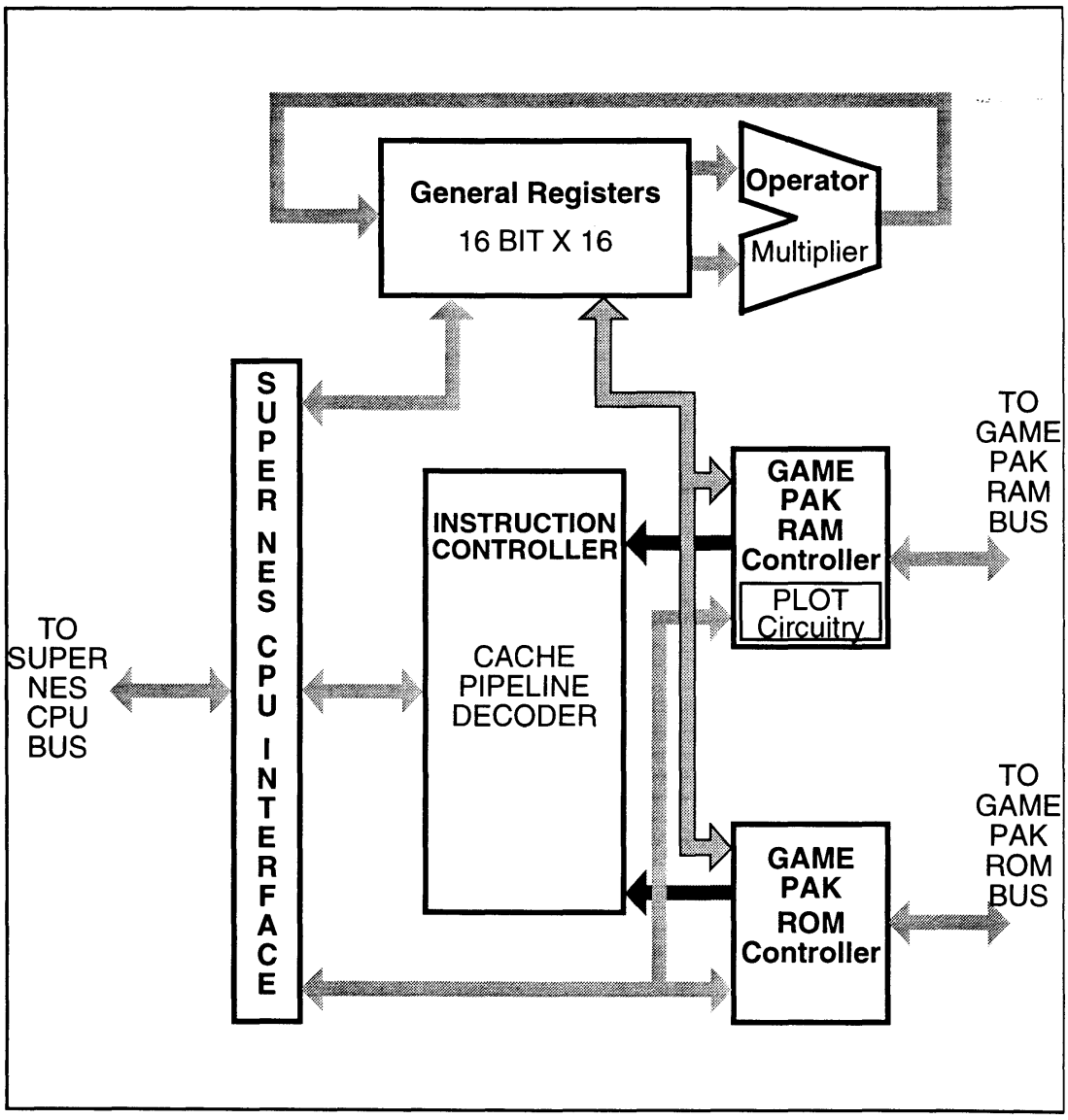


Figure 2-2-1 GSU Functional Block Diagram

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
DATA TRANSFER INSTRUCTIONS	From game pak ROM (ROM buffer) to register	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
	From game pak RAM to 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
	From register to game pak RAM (RAM buffer)	STW (Rm)	Store word data to RAM
		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
	From register to register	MOVE Rn, Rn'	Move word data
		MOVES Rn, Rn'	Move word data and set flags
	Immediate data to register	IWT Rn, #xx	Load immediate word data
		IBT Rn, #pp	Load immediate byte data
Arithmetic Operation Instructions		ADD Rn ADD #n	Add
		ADC Rn ADC #n	Add with carry
		SUB Rn SUB #n	Subtract
		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
Shift Instructions	ASR	Arithmetic shift right
	LSR	Logical shift right
	ROL	Rotate left through carry
	ROR	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 to128 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

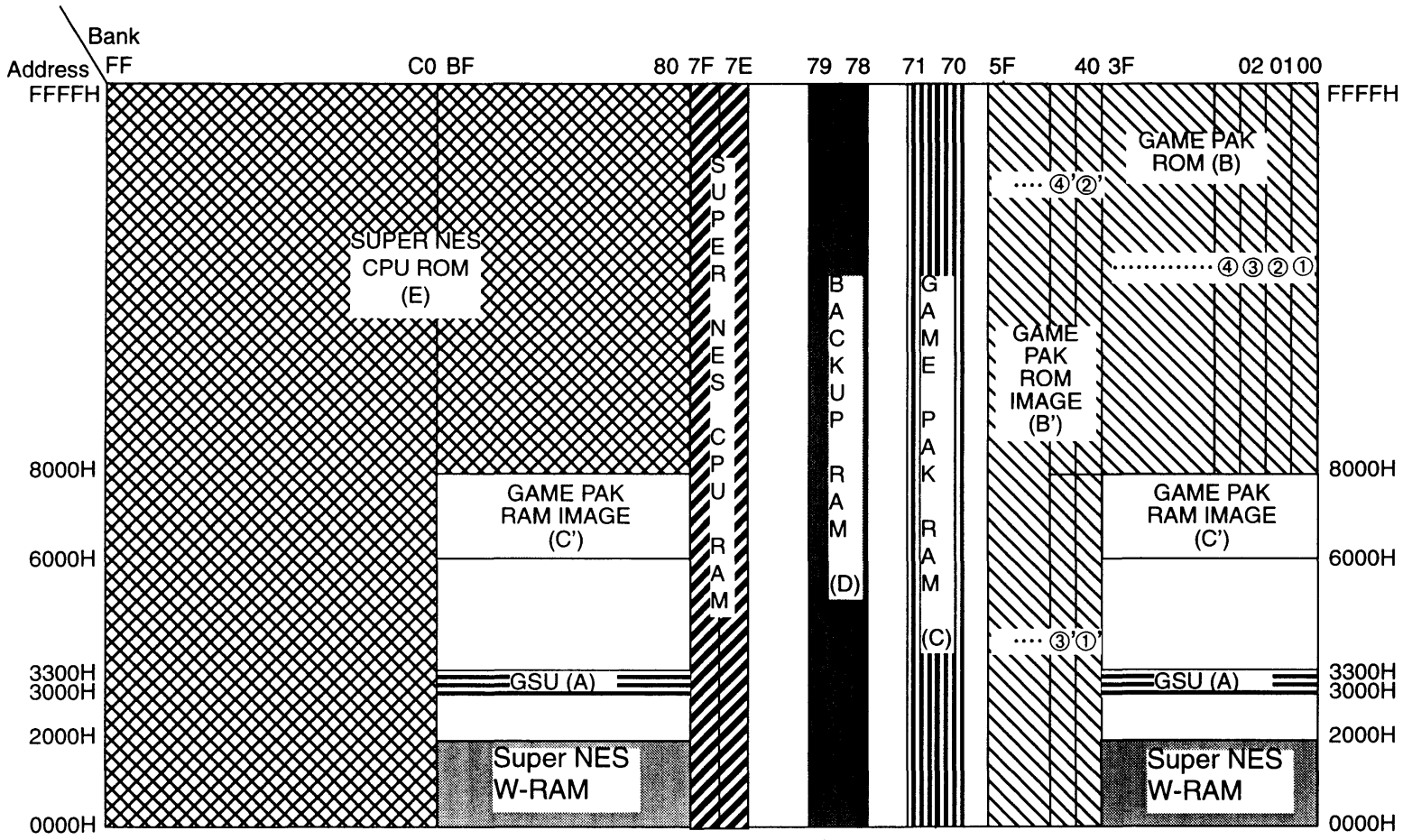


Figure 2-3-1 Super NES CPU Memory Map

3.2 GSU MEMORY MAPPING

The GSU memory map is depicted on the following page.

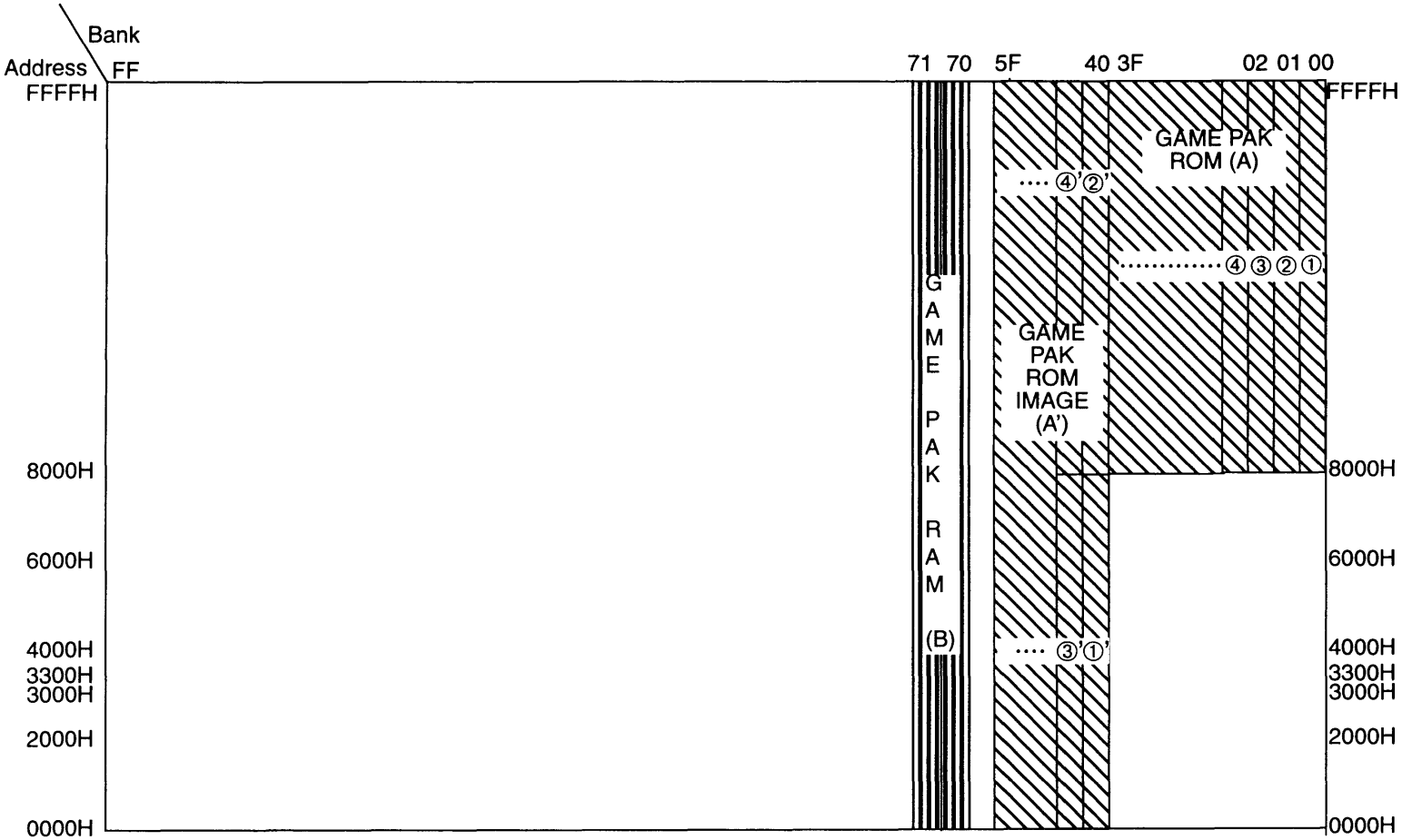
3.2.1 GAME PAK ROM

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 to128 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



Note: The PBR can be used to specify any bank address that is mapped.
The ROMBR can only be used to specify banks 00H to 5FH.

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	
GAME PAK ROM								
A15	A14	A13	A12	A11	A10	A9	A8	301DH
GAME PAK ROM								
A7	A6	A5	A4	A3	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	
Program Counter								
PC15	PC14	PC13	PC12	PC11	PC10	PC9	PC8	301FH
Program Counter								
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_{14} 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	
			Program Bank					
A23	A22	A21	A20	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	
			ROM Data Bank					
A23	A22	A21	A20	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								
A15	A14	A13	A12	A11	A10	A9	A8	303FH
Cache Base Address								
A7	A6	A5	A4	*	*	*	*	303EH

* 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	
Screen Base Address								
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
			Color Data				
CD7	CD6	CD5	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								
VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	303BH

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

D7	D6	D5	D4	D3	D2	D1	D0	
IRQ	-	MS0	-	-	-	-	-	3037H

MS0 → Multiplier Speed Selection
0 : Standard Speed Mode
1 : High Speed Mode

IRQ → IRQ Mask Flag
This is equal to 1 when the GSU interrupt request is masked.

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

D7	D6	D5	D4	D3	D2	D1	D0	
-	-	-	-	-	-	-	CLSR Flag	3039H

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.

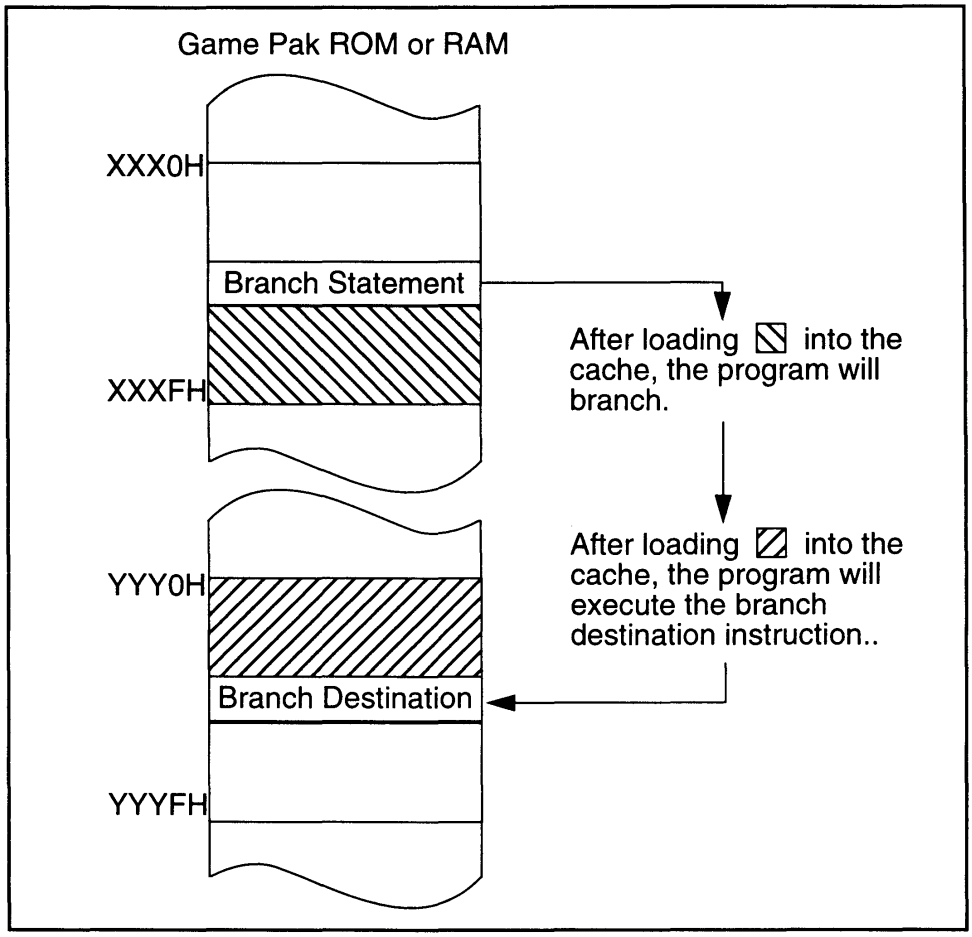


Figure 2-6-1 Load to Cache RAM While Branching

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.

CBR address on Super NES = 3100H + (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 that 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.

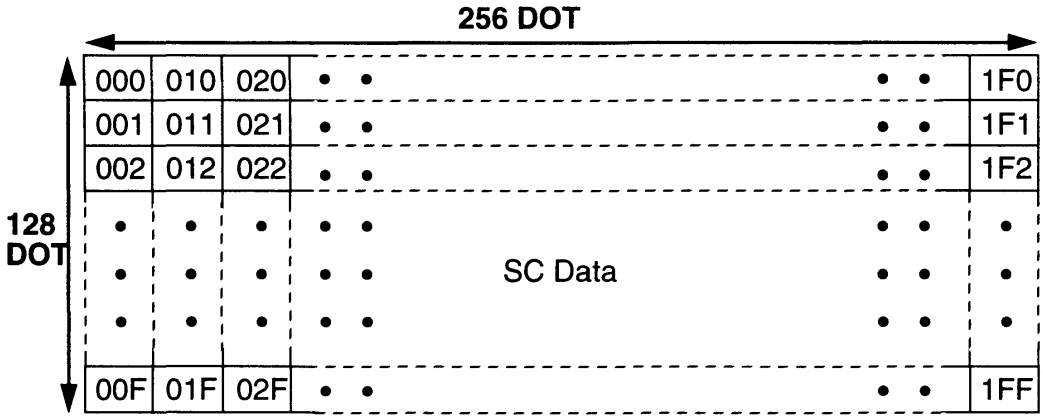


Figure 2-8-1 128 Dot High BG Character Array (numbers are hexadecimal)

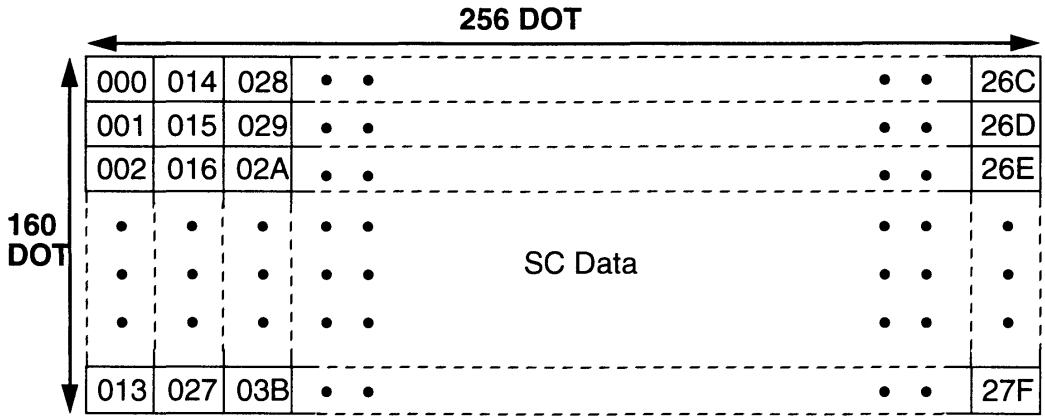


Figure 2-8-2 160 Dot High BG Character Array (numbers are hexadecimal)

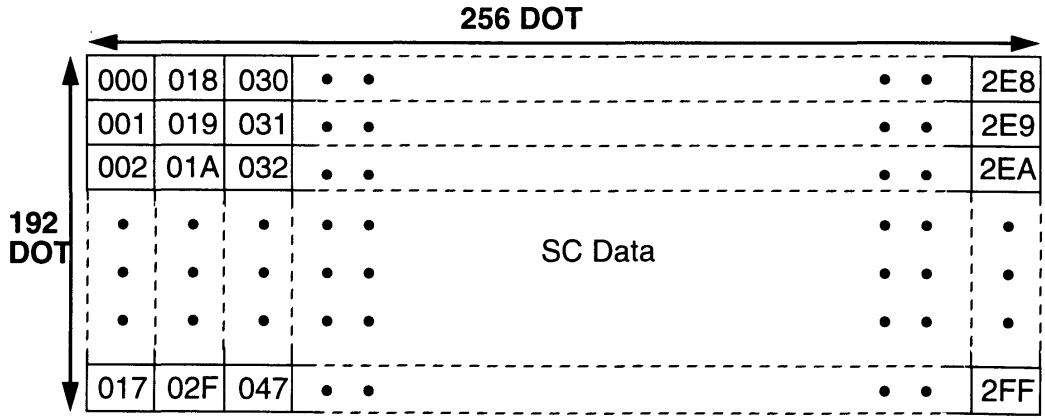


Figure 2-8-3 192 Dot High BG Character Array (numbers are hexadecimal)

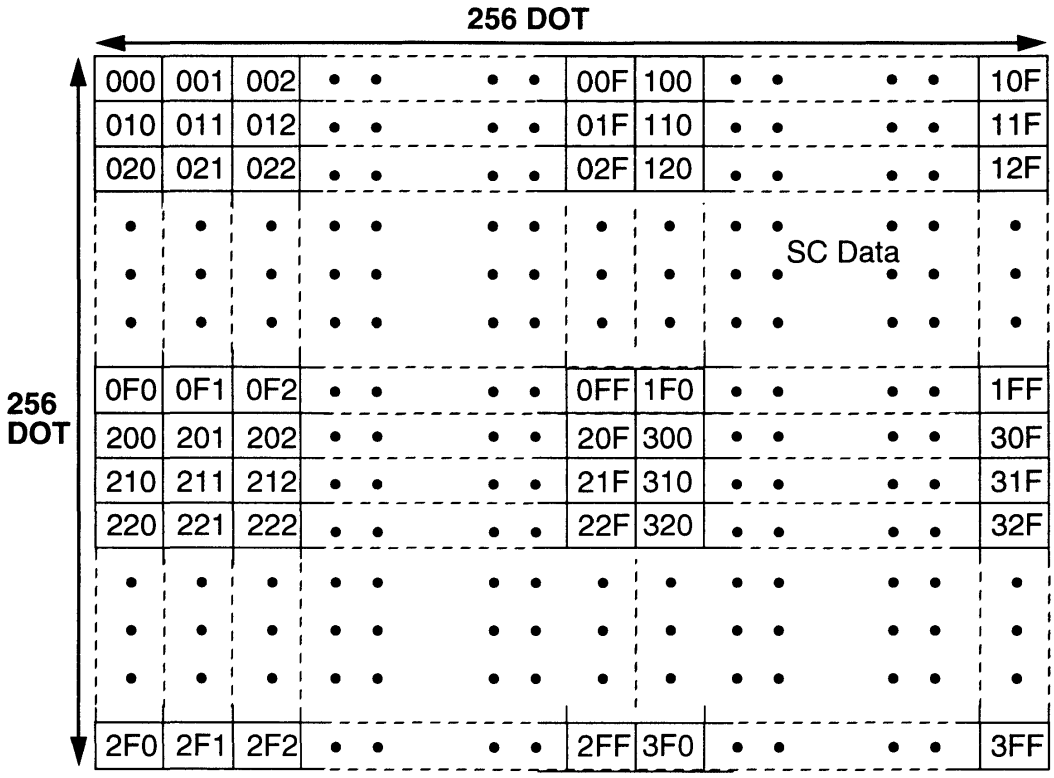


Figure 2-8-4 OBJ Character Array (numbers are hexadecimal)

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 =

(Number of dots high/8) X (Number of dots wide/8) X (8n)

[Number of vertical characters] [Number of horizontal characters] [Number of bytes/char]

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.

(Start Address) = 70:0000H+SCBRx400H

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;

(Start Address) = 70:0000H+11Hx400H = 70:4400H

(Total number of bytes of character data)
= (128/8)x(192/8)x(8x4) = 3000H

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_1 and R_2. The X coordinate value must be in R_1 and the Y coordinate value in R_2. 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.

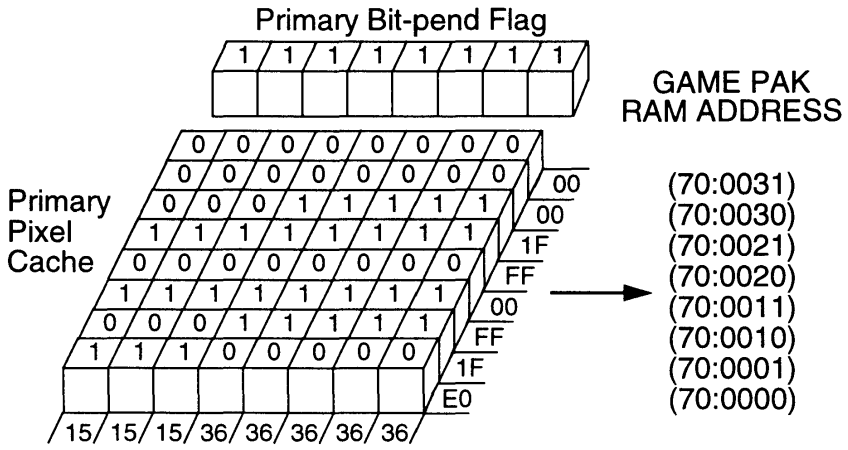
(Example 1)

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             ;Reset POR
IBT     R0,#15H
COLOR             ;Load 15H to the color register
PLOT
PLOT
PLOT              ;Plot (0,0) through (2,0)
IBT     R0,#36H
COLOR             ;Load 36H to the color register
PLOT
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.



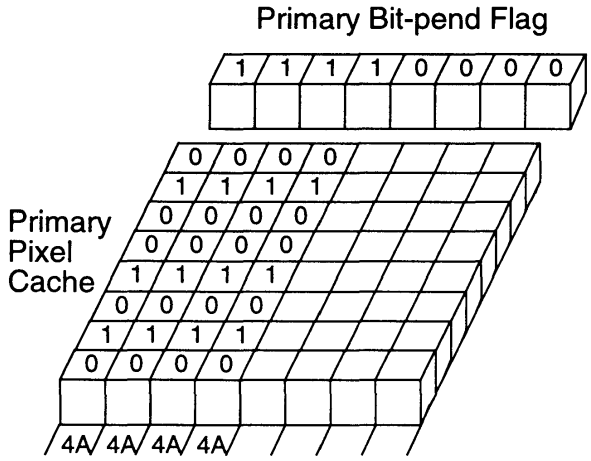
Since all 8 pixels in a character block are set with the final PLOT instruction, they are transferred from the primary pixel cache to the secondary pixel cache and the game pak RAM write begins. This process clears the primary bit-pend flags and the primary pixel cache is released.

(Example 2)

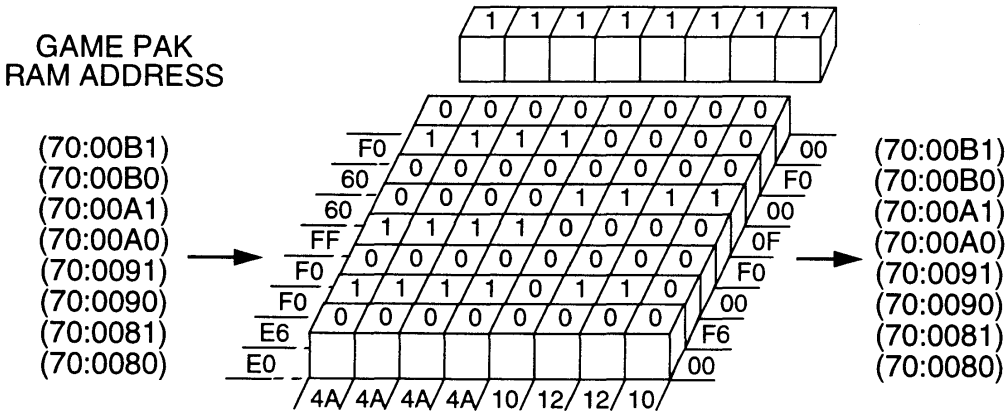
Continuing from Example 1, the following program is executed.

```
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.

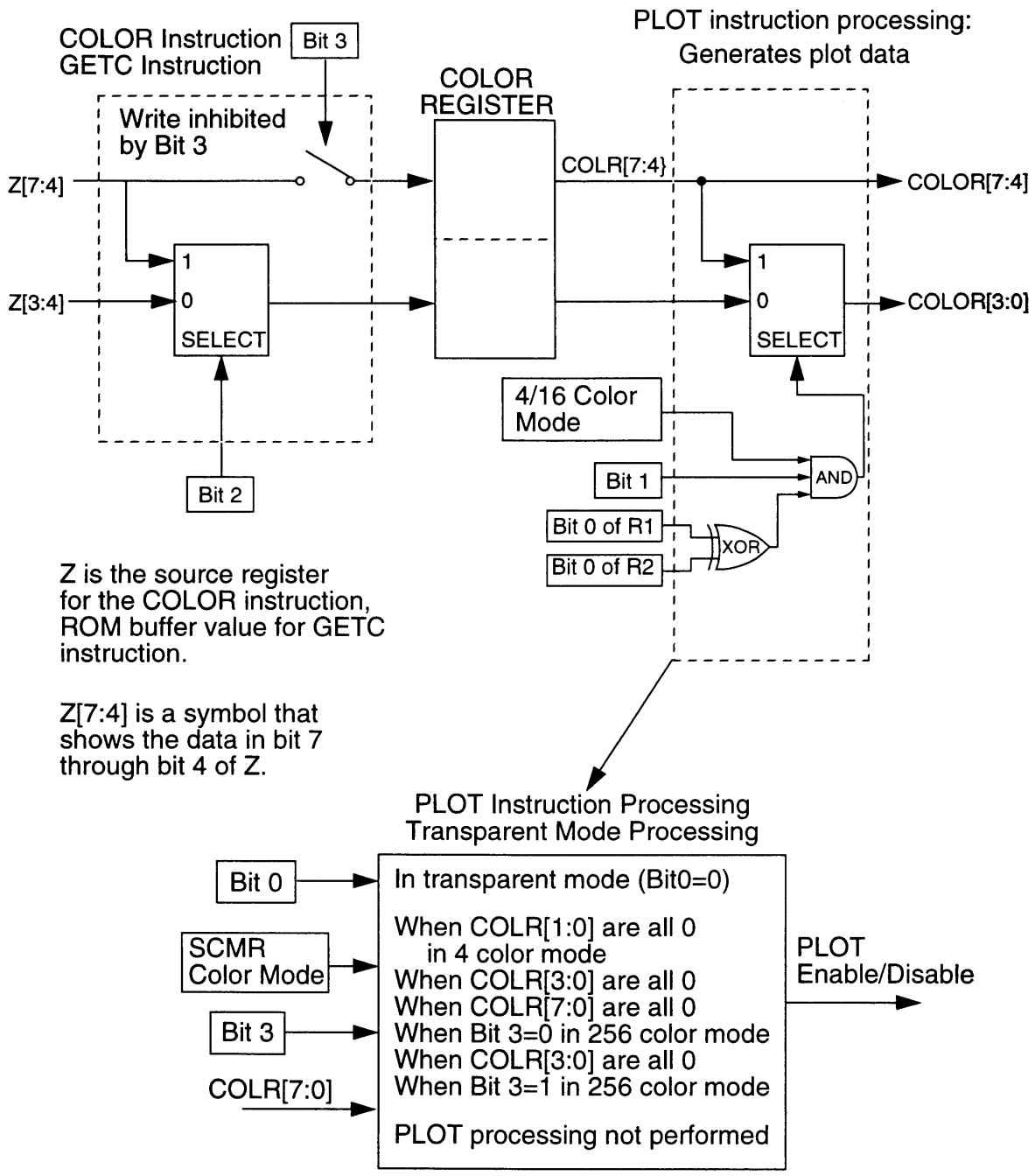


Figure 2-8-5 Plot Operations Assigned by CMODE

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_1 and R_2.
- 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] x 10H) + Y[7:3]

	X7	X6	X5	X4	X3				
+					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] x 14H) + Y[7:3]

	X7	X6	X5	X4	X3				
			X7	X6	X5	X4	X3		
+					Y7	Y6	Y5	Y4	Y3
CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0

(c) Height, 192 Dot Mode

CN [9:0] = (X[7:3] x 18H) + Y[7:3]

	X7	X6	X5	X4	X3				
		X7	X6	X5	X4	X3			
+					Y7	Y6	Y5	Y4	Y3
CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0

(d) OBJ Mode

CN [9:0] = (Y[7] x 200H) + (X[7] x100H) + (Y[6:3] x 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.

A[19:0] = (CN[9:0] x CHAR_SIZE)
+ (SB[7:0] x 4000H)
+ (Y[2:0] x 2)
+ (PL[2] x 200H) + (PL[1] x 100H) + PL[0]

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

		SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0										
						CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0				
																Y2	Y1	Y0	
+																			PL0
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

(b) 16 Color Mode

		SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0										
					CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0					
																Y2	Y1	Y0	
+															PL1				PL0
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

(c) 256 Color Mode

		SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0										
				CN9	CN8	CN7	CN6	CN5	CN4	CN3	CN2	CN1	CN0						
																Y2	Y1	Y0	
+														PL2	PL1				PL0
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

8.2 MULTIPLICATION INSTRUCTIONS

The 4 multiplication instructions shown below are available in the GSU.

• MULT instruction	Signed 8 bits Low 8 bits of Sreg	x	Signed 8 bits Low 8 bits of operand	→	Signed 16 bits Dreg
• UMULT instruction	Unsigned 8 bits Low 8 bits of Sreg	x	Unsigned 8 bits Low 8 bits of operand	→	Unsigned 16 bits Dreg
• LMULT instruction	Signed 16 bits Sreg	x	Signed 16 bits R6	→	Signed 32 bits High Dreg Low R4
• FMULT instruction	Signed 16 bits Sreg	x	Signed 16 bits R6	→	Signed 32 bits High Dreg

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 CLSR 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. ($0 \le yy \le 510$)
#n	Indicates 4-bit immediate data.
#xx	Indicates 16-bit immediate data. ($0 \le xx \le 65535$)
#pp	Indicates 8-bit immediate data. ($-128 \le pp \le 127$)
e	1-byte data $-128 \le e \le 127$, 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. ($0 \le xx \le 65535$)
#pp	Indicates 8-bit immediate data. ($-128 \le pp \le 127$)
e	1-bit data ($-128 \le e \le 127$), 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
→	Indicates direction of movement of data
+	Add
-	Subtract
*	Multiply
$\overline{Rn}$, $\overline{\#n}$	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_n

Operation: $S_{reg} + R_n + CY\ Flag \rightarrow D_{reg}$ (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_0.

The operand can be any of registers R_0~R_{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.

Opcode:

	(MSB)							(LSB)	
ADC R_n	0	0	1	1	1	1	0	1	(3DH)
	0	1	0	1	n (0H~FH)				(5nH)

Machine Cycles:

ROM execution time	6 cycles
RAM execution time	6 cycles
Cache RAM execution time	2 cycles

Example:

```
ADC   R1    ; R0+R1+CY→R0
WITH  R2    ; Set the source/destination registers to R2
ADC   R3    ; R2+R3+CY→R2
ADC   R2    ; R0+R2+CY→R0
```

9.5 ADC #n

Operation: S_{reg} + #n + CY Flag → 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_0.

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:

	(MSB)							(LSB)	
ADC #n	0	0	1	1	1	1	1	1	(3FH)
	0	1	0	1	n (0H~FH)				(5nH)

Machine Cycles:

ROM execution time	6 cycles
RAM execution time	6 cycles
Cache RAM execution time	2 cycles

Example:

```
ADC   #9H    ; R0+0009H+CY→R0
FROM  R3     ; Set the source register to R3
ADC   #5H    ; R3+0005H+CY→R0
ADC   #0AH   ; R0+000AH+CY→R0
```

9.6 ADD R_n

Operation: $S_{reg} + R_n \rightarrow D_{reg}$ (n=0~15)

Description: This instruction adds the source register and the register specified by the operand R_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_0.

The operand can be any of registers R_0~R_{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.

Opcode:

	(MSB)				(LSB)	
ADD R_n	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_{reg}: R_0, D_{reg}: R_0, R_0=4283H, R_4=2438H

R_0=66BBH when ADD R_4 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_{reg} + #n→ D_{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_0.

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:

	(MSB)							(LSB)	
ADD #n	0	0	1	1	1	1	1	0	(3EH)
	0	1	0	1	n (0H~FH)				(5nH)

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_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:

(MSB) (LSB)

ALT1

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 → 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:

	(MSB)							(LSB)	
ALT2	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 → ALT1 Flag
1 → 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:

	(MSB)							(LSB)	
ALT3	0	0	1	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_n

Operation: S_{reg} AND $R_n \rightarrow D_{reg}$ (n=1~15)

Description: This instruction performs logical AND on corresponding bits of the source register and the operand R_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_0.

The operand can be any of registers R_1~R_{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:

(MSB) (LSB)

ADD R_n

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_0.

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:

	(MSB)							(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 R_0 is, "3E5DH (0011 1110 0101 1101B)",

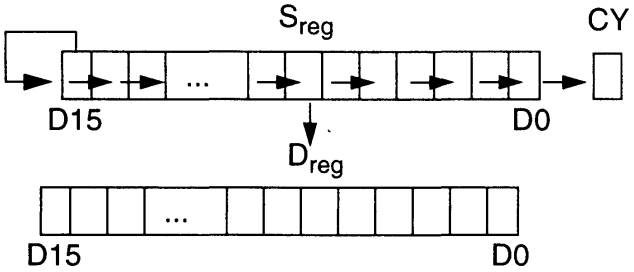
AND #6H

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)	
ASR	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_{10}	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_1 are:

CY		bit15															bit0	
1	R_1	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 ~+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:

	(MSB)							(LSB)	
	0	0	0	0	1	1	0	0	(0CH)
BCC e	← e (00H~FFH) →								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 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:

PC ADDRESS	Object Code	
51E	0C	(BCC $+5H)
51F	03	
520	←PC before jump	Execute instruction at address 520 and jump.
521		
522		
523	←PC after jump	
.	.	
.	.	
.	.	

9.15 BCS e

Operation: If CY Flag=1

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+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)	
	0	0	0	0	1	1	0	1	(0DH)
BCS e	← e (00H~FFH) →								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:

PC ADDRESS	Object Code	
42D	←PC after jump	
42E	0D (BCS $-1H)	
42F	FD	Execute instruction
430	←PC before jump	at address 430 and
431		jump.
.	.	
.	.	
.	.	

9.16 BEQ e

Operation: If Z Flag=1

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+127) R_{15} identifies the next address for the BEQ instruction (2 bytes)

Description: If the zero 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 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)	
	0	0	0	0	1	0	0	1	(09H)
BEQ e	← e (00H~FFH) →								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 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:

PC ADDRESS	Object Code	
15FD		
15FE	09	(BEQ $+5H)
15FF	03	
1600	←PC before jump	Execute instruction at address 1600 and jump.
1601		
1602		
1603	←PC after jump	

9.17 BGE e

Operation: If (S XOR O/V)=0

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+127)
R_{15} 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)							(LSB)	
	0	0	0	0	0	1	1	1	(07H)
BGE e	← e (00H~FFH) →								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 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:

PC ADDRESS	Object Code	
22FA		
22FB	←PC after jump	Execute instruction at address 2300 and jump.
22FC		
22FD		
22FE	07 (BGE $-3H)	
22FF	FB	
2300	←PC before jump	

9.18 BIC R_n

Operation: S_{reg} AND $\overline{Rn} \rightarrow D_{reg}$ (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_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_0.

The operand can be any of registers R_1~R_{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:

	(MSB)							(LSB)	
BIC R_n	0	0	1	1	1	1	0	1	(3DH)
	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: Under the following conditions:

S_{reg}: R_2, D_{reg}: R_0

R_2=75CEH (0111 0101 1100 1110B),
R_1=3846H (0011 1000 0100 0110B)

R_0 is 4588H (0100 0101 1000 1000B) when

BIC R_1

is executed.

9.19 BIC #n

Operation: S_{reg} 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_0.

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)

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_5
R_4= 364BH (0011 0110 0100 1011B)

R_5 is 3640H (0011 0110 0100 0000B) when

BIC #F

is executed.

9.20 BLT e

Operation: If (S XOR O/V)=1

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+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)
BLT e	← e (00H~FFH) →								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 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:

PC ADDRESS	Object Code	
BBD		
BBE	06	(BLT $+4H)
BBF	02	
BC0	←PC before jump	Execute instruction at address BC0 and jump.
BC1		
BC2	←PC after 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".

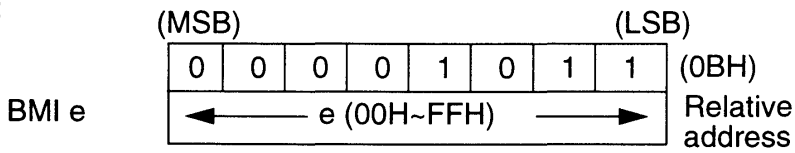
If the decision results in a JUMP, the next instruction to be executed will already be in the instruction pipeline of the processor. For this reason one byte from the pipeline will be executed before the instruction at the branch destination is executed. (The execution time for this instruction is not 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:

PC ADDRESS	Object Code	
57D		
57E	0B	(BMI $+5H)
57F	03	
580	←PC before jump	Execute instruction at address 580 and jump.
581		
582		
583	←PC after jump	

9.22 BNE e

Operation: If Z Flag = 0

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+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)	
	0	0	0	0	1	0	0	0	(08H)
BNE e	← e (00H~FFH) →								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:

PC ADDRESS	Object Code	
35FB		
35FC	←PC after jump	Execute instruction at address 3600 and jump.
35FD		
35FE	08 (BNE $-2H)	
35FF	FC	
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_{15} identifies the next address for the BPL instruction (2 bytes)

Description: If the sign 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 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:

	(MSB)							(LSB)	
	0	0	0	0	1	0	1	0	(0AH)
BPL e	◄—— e (00H~FFH) ——►								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 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:

PC ADDRESS	Object Code	
95D		
95E	0A	(BPL $+4H)
95F	02	
960	←PC before jump	Execute instruction at address 960 and jump.
961		
962	←PC after jump	
963		

9.24 BRA e

Operation: $R_{15}+e \rightarrow R_{15}$ (e= -128~+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)	
	0	0	0	0	0	1	0	1	(05H)
BRA e	← e (00H~FFH) →								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:

PC ADDRESS	Object Code	
B0FC		
B0FD		
B0FE	05	←PC after jump
B0FF	FE	(BRA $0H)
B100		←PC before jump
B101		

Execute instruction at address B100 and jump.

9.25 BVC e

Operation: If O/V Flag=0

then $R_{15}+e \rightarrow R_{15}$ (e= -128~+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)	
	0	0	0	0	1	1	1	0	(0EH)
BVC e	← e (00H~FFH) →								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 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:

PC ADDRESS	Object Code	
288D		
288E	0E	(BVC $+4H)
288F	02	
2890	←PC before jump	Execute instruction at address 2890 and jump.
2891		
2892	←PC after jump	
2893		

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_{15} 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:

BVS e

(MSB)							(LSB)	
0	0	0	0	1	1	1	1	(0FH)
← e (00H~FFH) →								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:

PC ADDRESS	Object Code	
68B		
68C	←PC after jump	Execute instruction at address 690 and jump.
68D		
68E	0F (BVS $-2H)	
68F	FC	
690	←PC before jump	
691		

9.27 CACHE

Operation: If CACHE BASE REGISTER<>(R_{15} & 0FFF0H)

then (R_{15} & 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~b0) → 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:

CMODE

(MSB)							(LSB)	
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_0, R_0= 0002H

the transparency and dithering modes are set when

CMODE

is executed.

9.29 CMP R_n

Operation: $S_{reg} - R_n$ (n=0~15)

Description: This instruction subtracts the operand R_n 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_0.

The operand can be R_0~R_{15}.

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:

	(MSB)							(LSB)	
CMP R_n	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_{reg}: R_1, R_1= 8000H, R_3= 2FFFH

the overflow and carry flags are set and sign and zero flags are reset when

CMP R_3

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_1 or R_2 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_6, R_6= 9830H

the color register becomes 30H when

COLOR

is executed.

9.31 DEC R_n

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_n	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_9

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_0.

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:

	(MSB)							(LSB)	
DIV2	0	0	1	1	1	1	0	1	(3DH)
	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_7, D_{reg}: R_2

CY		Bit15															Bit0	
0	R_7:	0	1	0	0	0	1	1	0	0	0	1	1	0	1	0	1	(4635H)

becomes

CY		Bit15															Bit0	
1	R_2:	0	0	1	0	0	0	1	1	0	0	0	1	1	0	1	0	(231AH)

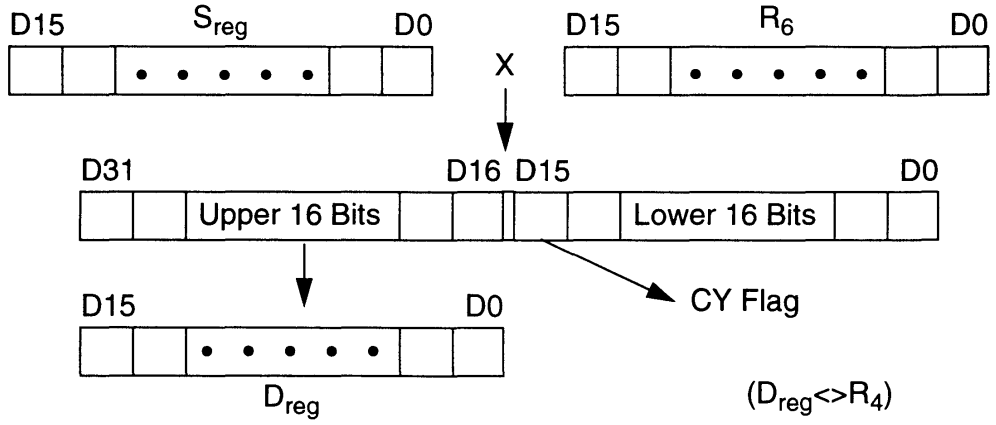
when

DIV2

is executed.

9.33 FMULT

Operation:



Description: This instruction performs a 16 x16-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~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	1	0	0	1	1	1	1	1	(9FH)

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_5, D_{reg}: R_2, R_5= 4AAAH, R_6= DAABH

R_2 becomes F51CH and the carry flag and sign flag are set when

FMULT

is executed.

9.34 FROM R_n

REGISTER PREFIX INSTRUCTION

Operation: If B =0 then set S_{reg} to R_n (n=0~15)
else $R_n \rightarrow D_{reg}$

Description: This instruction specifies which of the registers, R_0~R_{15}, is to be used as the source register. If the B flag is set, the contents of the specified operand R_n are stored in the destination register D_{reg}, 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_n	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_2

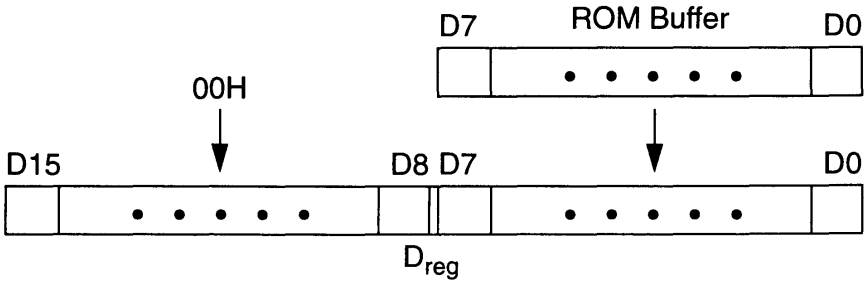
to set R_2 as the source register.

To perform R_2+ R_3= R_0, write:

```
FROM R2        ;Sets the source register to R2
ADD R3         ;Executes R2 + R3→R0
```

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_{14} is the ROM address pointer when data is loaded from the game pak ROM into the ROM buffer. Using the value stored at R_{14} 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_0.

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_{reg}:R_0

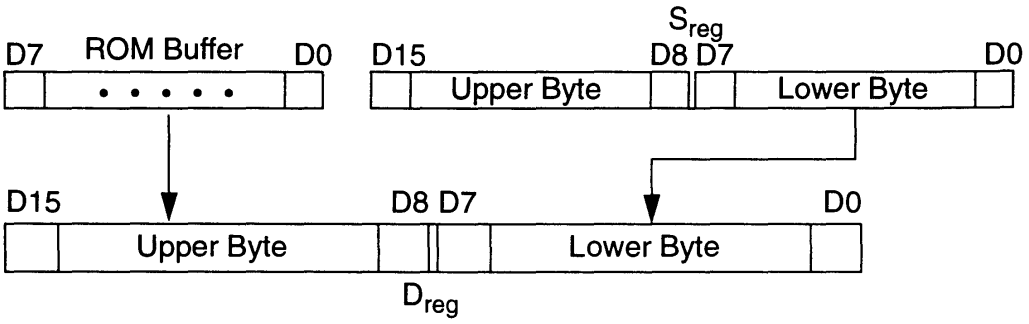
R_0 becomes 0075H when

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_0.

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_{reg}: R_2, D_{reg}: R_6, R_2= 4ABDH

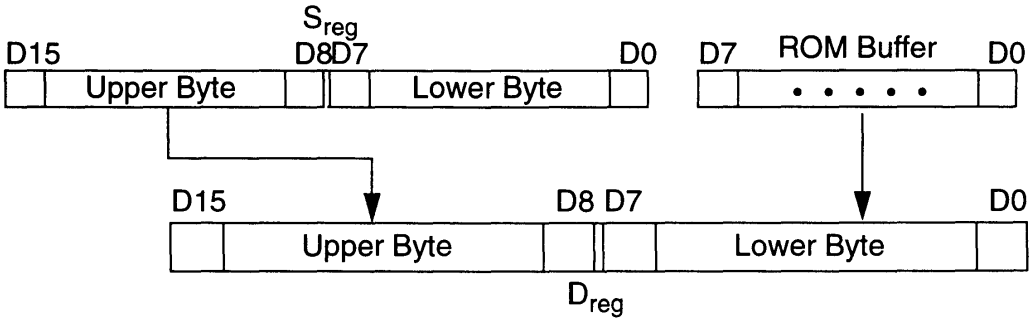
R_6 becomes 75BDH when

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_0.

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_{reg}: R_2, D_{reg}: R_6, R_2= 4ABDH

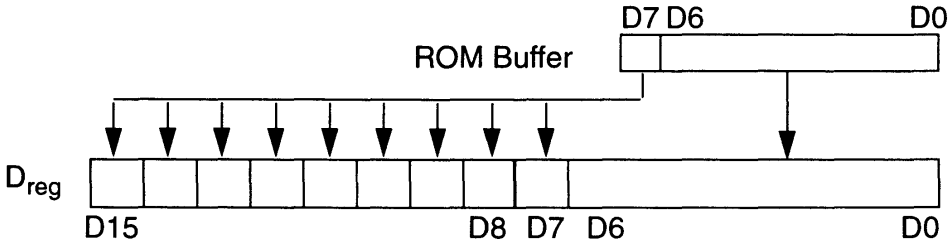
R_6 is 4A75H when

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_0.

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	1	(3FH)
	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 execution cycles varies with each program.

Example: Under the following conditions,

(ROM buffer) = 85H, D_{reg}: R_8

R_8 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

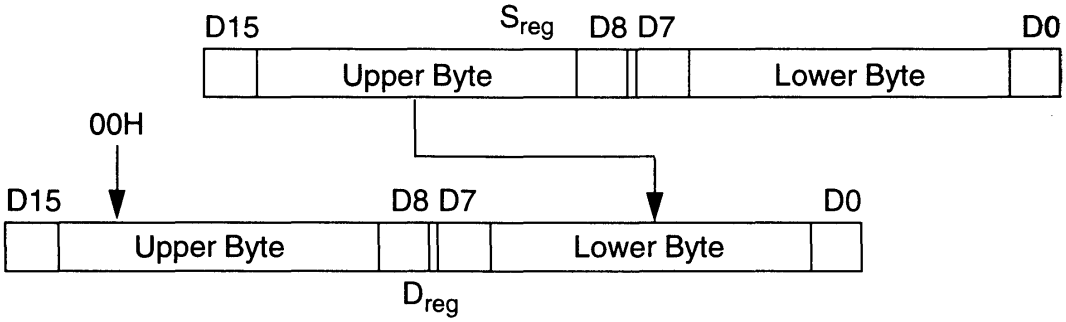
4BH is loaded to the color register when

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_0.

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:

	(MSB)							(LSB)	
HIB	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_{11}, D_{reg}=R_1, R_{11}= 8A43H

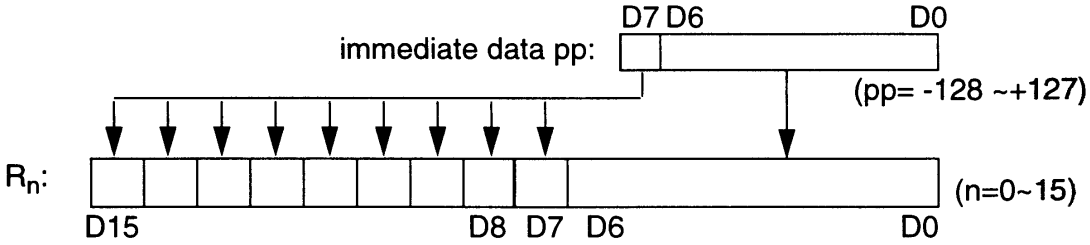
R_1 becomes 008AH and the sign flag is set when

HIB

is executed.

9.41 IBT R_n, #pp

Operation:



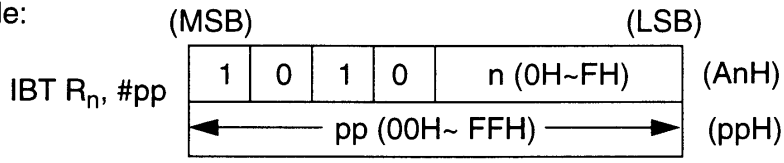
Description: This instruction loads one byte of immediate data (hexadecimal) into the low byte of register R_n. Bit 7 of the immediate data is loaded into bits 8 through 15 of R_n.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:



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 intergers, 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   R8,   #4      ... 0004H → R8
IBT   R8,   #-128   ... FF80H → R8
IBT   R8,   #0A4H   ... FFA4H → R8
```

9.42 INC R_n

Operation: $R_n + 1 \rightarrow R_n$ (n = 0~14)

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~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_n

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_{12} is 65B1H, R_{12} becomes 65B2H when

INC R_{12}

is executed.

9.43 IWT R_n, #xx

Operation: #xx (2-byte hexadecimal immediate data) → R_n

(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_n.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:

	(MSB)				(LSB)	
ITW R_n, #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_0 becomes 4583H when

IWT R_0, #4583H

is executed.

9.44 JMP R_n

Operation: $R_n \rightarrow R_{15}$ (PC) (n=8~13)

Description: This instruction loads the contents of the register specified in the operand R_n to R_{15} (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_8~R_{13}.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:

(MSB) (LSB)

JMP R_n	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_{10} is 0555H and the following program is executed,

PC	Opcode
0444H	JMP R_{10}
0445H	INC R_{10}
.	.
.	.
.	.

the jump destination is 0555H.

9.45 LDB (R_m)

Operation: (R_m) → D_{reg} (Low Byte) (m=0~11)

00H → 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_m 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_0.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:

	(MSB)							(LSB)	
LDB (R_m)	0	0	1	1	1	1	0	1	(3DH)
	0	1	0	0	m (0H~BH)				(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_1)

R_7 becomes 0051H.

9.46 LDW (R_m)

Operation: (R_m) → D_{reg} (Low Byte) (m=0~11)

(R_m±1) → 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~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     (R3)
```

the register R_5 becomes C02EH.

9.47 LEA R_n, xx (Refer to IWT R_n, #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_n.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:

LEA R_n, xx

(MSB)					(LSB)	
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_3 becomes 4853H when

LEA R_3, #4853H

is executed.

9.48 LINK #n

Operation: R_{15} + #n → 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 R15, #74FFH
436C    NOP
436B    IBT R1, #12H
```

register R_{11} becomes 4369H + 2=436BH

9.49 LJMP R_n

Operation: $R_n \rightarrow R_{15}$ (PC) (n=8~13)

$S_{reg} \rightarrow$ 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~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_n	0	0	1	1	1	1	0	1	(3DH)
	1	0	0	1	n (8H~DH)				(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_1:0001H

the program jumps from 00:8006H to 01:0002H when the following program is executed.

Bank	:Address	Syntax
00	:8000H	IWT R_{10}, #0002H
00	:8003H	FROM R_1
00	:8004H	LJMP R_{10}
00	:8006H	NOP

9.50 LM R_n, (xx)

Operation: RAM (xx) → R_n (low byte) (n=0~15, xx=0~65535)

RAM (xx±1) → R_n (high byte) When the value of xx is:
even, (xx+1)
odd, (xx-1)
is loaded to the high byte.

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:

LM R_n, (xx)

(MSB)							(LSB)	
0	0	1	1	1	1	0	1	(3DH)
1	1	1	1	n (0H~FH)				(FnH)
x (00H~FFH)								(ADRS Lower Byte)
x (00H~FFH)								(ADRS Upper Byte)

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_9 becomes 9628H when the following program is executed:

```
LM      R9, (0BACCH)
```

9.51 LMS R_n, (yy)

Operation: RAM (yy) → R_n (low byte) (n=0~15, yy=0~510*)
RAM (yy+1) → R_n (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_n. 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:

	(MSB)							(LSB)	
	0	0	1	1	1	1	0	1	(3DH)
LMS R_n, (yy)	1	0	1	0	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 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 x 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.

Example: Under the following conditions,

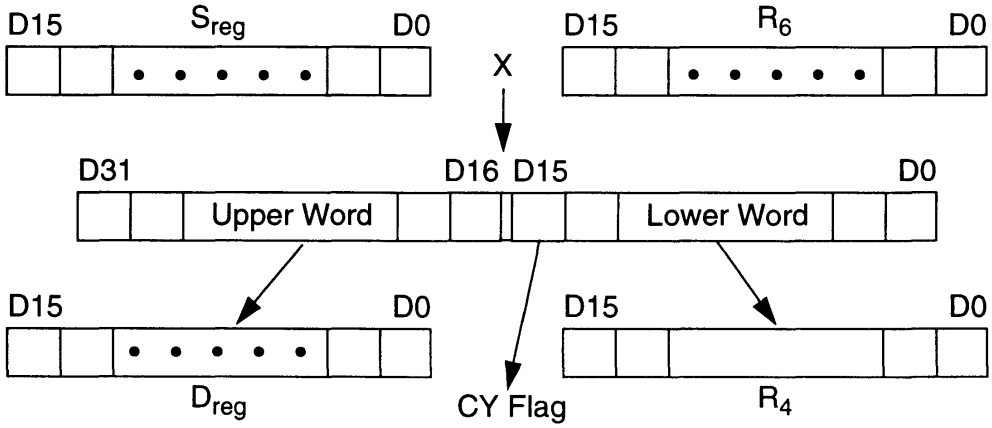
(70:1AAH) = 32H, (70:1ABH) = 92H, RAMBR:70H

register R_3 becomes 9232H when the following program is executed:

Syntax	Opcode
LMS R_3, (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_6. The upper 16 bits of the result are stored in the destination register, and the lower 16 bits are stored in R_4. If Bit 15 of R_6 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_0. If R_4 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_6 is "1", reset if "0"
Z : Set if the destination register result is zero, else reset.

Opcode:

	(MSB)							(LSB)	
LMULT	0	0	1	1	1	1	0	1	(3DH)
	1	0	0	1	1	1	1	1	(9FH)

Machine Cycles:	ROM execution time	10 or14 cycles
	RAM execution time	10 or14 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_9, D_{reg}: R_8
R_9= B556H, R_6= DAABH

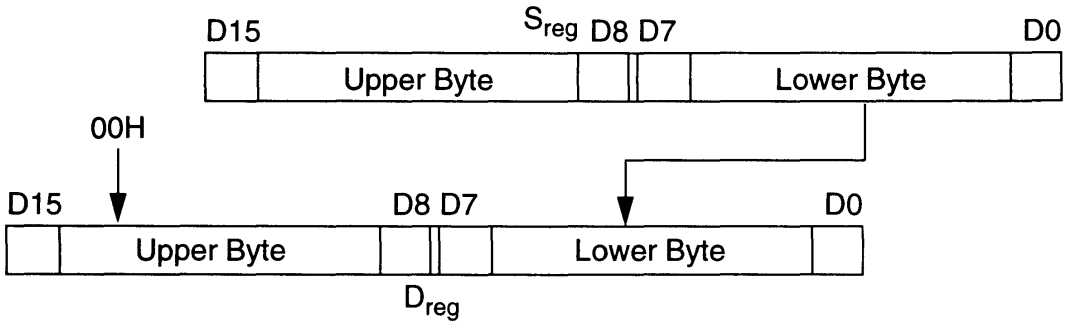
the register R_8 becomes 0AE3H and R_4 5C72H when

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_0.

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_{10}, D_{reg}: R_{12}, R_{10}= FB23H

the register R_{12} 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

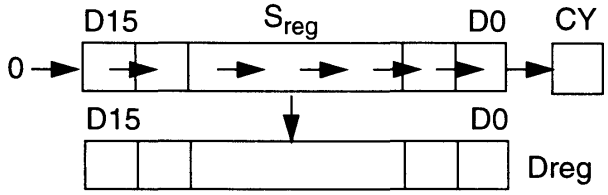
Example: In the following program,

```
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 RAM execution time	1 cycles

Example: Under the following conditions,

S_{reg}: R_8, D_{reg}: R_0

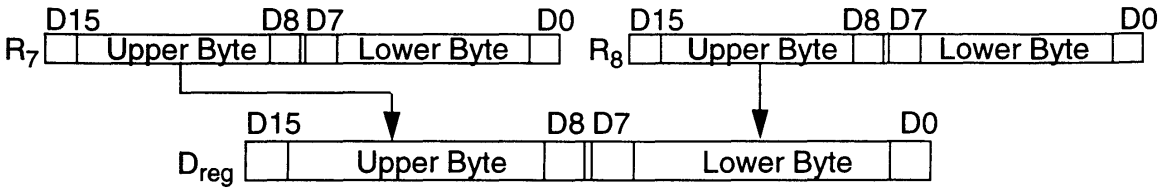
	bit 15															bit0	
R_8:	1	0	1	1	0	1	0	1	0	0	1	1	1	1	1	1	(B53FH)

LSR execution results in:

	bit15															bit0		CY
R_0:	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

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_n, R_n'

Operation: $R_n' \rightarrow R_n$ (n, n' = 0~15)

Description: This instruction loads the contents of register R_n', specified in the second operand, to register R_n, 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_n, R_n'	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_8 becomes 4983H when

MOVE R_8, R_{14}

is executed.

9.58 MOVE R_n, #xx

MACRO INSTRUCTION

Operation: #xx → R_n (n = 0~15, #xx=-32768~65535)
(if unsigned, #xx=0~65535)

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_n, #pp" or "IWT R_n, #xx" for machine cycles, flags affected, and opcode.

Example:

MOVE R_8, #070H	;0070H→R_8	(IBT R_8, #070H)
MOVE R_8, #0A4H;	00A4H→R_8	(IWT R_8, #0A4H)
MOVE R_8, #-128;	FF80→R_8	(IBT R_8, #-128)

9.59 MOVE R_n, (xx)

MACRO INSTRUCTION

Operation: (xx) → R_n (low byte) (n=0~15, xx=0~FFFFH)

(xx±1) → R_n (high byte)

Note: When the value xx is even, the contents of (xx+1) are loaded to the high byte of R_n. When the value of xx is odd, the contents of (xx-1) are loaded to the high byte of R_n.

Conditions: If (0000H≤xx≤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_n, 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_n, (xx)" or "LMS R_n, (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_n, (xx)" or "LMS R_n, (yy)" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

(70:BACCH) = 28H, (70:BACDH) = 96H, RAMBR=70H

the register R_9 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_3 becomes 9232H when the following program is executed:

```
MOVE    R3, (1AAH)      ;(71:01AAH)→R3(Low Byte)    (LMS R3, (01AAH))
                        ;(71:01ABH)→R3(High Byte)
```

9.60 MOVE (xx), R_n

MACRO INSTRUCTION

Operation: R_n (low byte) → (xx) (n=0~15, xx=0~FFFFH)

R_n (high byte) → (xx±1)

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≤xx≤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_n" or "SMS (yy), R_n." 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_n" and "SMS (yy), R_n" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

R_9: BACDH, and RAMBR=71H

when the following program is executed,

```
MOVE    (9CDEH), R9     ;R9 (Low Byte)→(71:9CDEH)   (SM (9CDEH), R9)
                        ;R9 (High Byte)→(71:9CDFH)
```

the result is (71:9CDEH)=CDH, (71:9CDFH)=BAH

Also, under the following conditions,

R_2: 3248H, and RAMBR=70H

when the following program is executed,

```
MOVE    (136H), R2        ;R2 (Low Byte)→(70:0136H)    (SMS (136H), R2)
                          ;R2 (High Byte)→(70:0137H)
```

the result is (70:0136H)=48H, (70:0137H)=32H

9.61 MOVEB R_n, (R_n')

MACRO INSTRUCTION

Operation: (R_n') → R_n (Low Byte) (n=0~15, n'=0~11)
00H →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_n', 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_0~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 "LDB (R_m)" or "TO R_n" + "LDB (R_m)." 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_m)" and "TO R_n" 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_7 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_0 becomes 009AH.

9.62 MOVEB (R_n'), R_n

MACRO INSTRUCTION

Operation: R_n (low byte) → (R_n') (n=1~15, n'=0~11)

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~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 (Rm)" or "FROM Rn" + "STB (R_m)." 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 (Rm) and "FROM Rn" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

R_5=3843H, R_{11}=94F1H, RAMBR=71H

when the following program is executed,

MOVEB (R_{11}), R_5 ;R_5 (Low Byte)→(R_{11}) (FROM R_5+STB (R_{11}))

the result is (71:94F1H)=43H.

Also, under the following conditions,

R_0=89E0H, R_3=438BH, RAMBR=70H

when the following program is executed,

MOVEB (R_3), R_0 ;R_0 (Low Byte)→(R_3) (STB (R_3))

the result is (70:438BH)=43H.

9.63 MOVES R_n, R_n'

Operation: $R_n' \rightarrow R_n$ (n, n' = 0~15)

Description: This instruction loads the contents of register R_n', specified in the second operand, to register R_n, specified in the first operand. Flags are set according to the data loaded. (Refer to MOVE R_n, R_n'.)

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:

	(MSB)				(LSB)	
MOVES R_n, R_n'	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_7 is 4983H and

MOVES R_{10}, R_7

is executed, the register R_{10} becomes 4983H and the overflow flag is set.

9.64 MOVEW R_n, (R_n')

MACRO INSTRUCTION

Operation: (R_n') → R_n (Low Byte) (n=0~15, n'=0~11)
(R_n'±1) → R_n (High Byte)

Note: If the contents of R_n' are even, store the address equal to the contents of (R_n'+1) in the high byte of R_n. If the contents of R_n' are odd, store the address equal to (R_n'-1) in the high byte of R_n.

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_n' specified in the second operand and stores it into register R_n 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_m)" or "TO R_n" + "LDW (R_m)." 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_m)" and "TO R_n" for machine cycles, flags affected, and opcode.

Example: Under the following conditions,

R_3=6480H, (71:6480H)=2EH, (71:6481H)=C0H,
RAMBR=71H

and when the following program is executed,

```
MOVEW   R5, (R3)   ;(R3)→R5(Low Byte)      (TO R5 + LDB (R3))
                   ;(R3+1)→R5(High Byte)
```

register R_5 becomes C02EH.

Also, under the following conditions,

R_6=0822H, (70:0822H)=43H, (70:0823H)=96H,
RAMBR=70H

and when the following program is executed,

```
MOVEW   R0, (R6)   ;(R6)→R0(Low Byte)      (LDB (R6))
                   ;(R6+1)→R0(High Byte)
```

register R_0 becomes 9643H.

9.65 MOVEW (R_n'), R_n

MACRO INSTRUCTION

Operation: R_n (low byte) → (R_n') (n=0~15, n'=0~11)

R_n (high byte) → (R_n' ±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~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 opcode.

Example: Under the following conditions,

R_9=BFA3H, R_{10}=4444H, RAMBR=71H

and when the following program is executed,

```
MOVEW   (R10), R9   ;R9(Low Byte)→(R10)  (FROM R9+STW (R10))
                    ;R9(High Byte)→(R10+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   (R6), R0    ;R0 (Low Byte)→(R6)   (STW (R6))
                    ;R0 (High Byte)→(R6+1)
```

the result is (71:92A0H)=51H, (71:92A1H)=31H.

9.66 MULT R_n

Operation: S_{reg} (low byte) * R_n (low byte) → 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 low byte of register R_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_0.

The operand can be a register R_0~R_{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 R_n

1	0	0	0	n (0H~FH)

(8nH)

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_{reg}: R_5, D_{reg}: R_2
R_5= 52CFH, R_1= 63CFH

the register R_2 becomes 0961H when

MULT R_1

is executed.

9.67 MULT #n

Operation: S_{reg} (low byte) * #n → 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_0.

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	0	0	1	1	1	1	1	0	(3EH)
	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 upon the CFGR register.

Example: Under the following conditions,

S_{reg}: R_3, D_{reg}:R_4, R_3= 95C6H

the register R_4 becomes FDF6H when

MULT #9

is executed.

9.68 NOP

Operation: PC ← 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	0	0	0	0	0	0	0	1	(01H)

Machine Cycles:

ROM execution time	3 cycles
RAM execution time	3 cycles
Cache RAM execution time	1 cycles

9.69 NOT

Operation: $\overline{Sreg} \rightarrow Dreg$

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_0.

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_{reg}: R_9, D_{reg}: R_{13}

	Bit15															Bit0	
R_9:	1	0	1	1	0	1	1	1	0	1	1	0	0	1	0	0	(B764H)

the execution of

NOT

results in:

	Bit15															Bit0	
R_{13}:	0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	1	(489BH)

9.70 OR Rn

Operation: S_{reg} OR $R_n \rightarrow D_{reg}$ (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_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_0.

The operand can be a register R_1~R_{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)	
OR R_n	1	1	0	0	n (1H~FH)	(CnH)

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	
R_4:	0	1	1	0	0	0	1	1	0	1	1	0	1	0	0	0	(6368H)

	Bit15															Bit0	
R_2:	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_5:	0	1	1	1	0	1	1	1	1	1	1	0	1	1	0	0	(77ECH)

when

OR R_2

is executed.

9.71 OR #n

Operation: Sreg OR #n → Dreg (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_0.

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_{reg}: R_7, D_{reg}: R_5

	Bit15															Bit0	
R_7:	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_5:	0	1	0	1	1	1	1	1	1	0	1	0	0	1	1	1	(5FA7H)

when

OR #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_1 and R_2. After plotting, R_1 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: $S_{reg} \rightarrow$ 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:

	(MSB)							(LSB)	
RAMB	0	0	1	1	1	1	1	0	(3EH)
	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

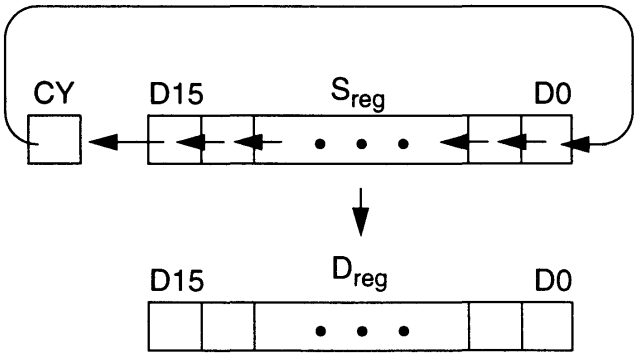
the RAM bank register becomes 70H when

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_8, D_{reg}: R_4

CY		bit15															bit0	
1	R_8:	0	0	0	1	1	1	0	1	0	1	0	0	1	0	1	1	(1D4BH)

executing ROL results in:

CY		bit15															bit0	
0	R_4:	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

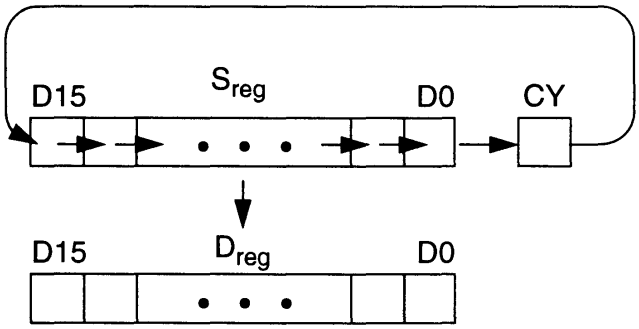
the ROMBR becomes 46H when

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_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 source register is "1", else reset.
Z : Set on zero result, else reset.

Opcode:

	(MSB)							(LSB)	
ROR	1	0	0	1	0	1	1	1	(97H)

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_{10}, Dreg: R_{12}

CY		bit15															bit0	
1	R_{10}:	0	0	0	1	1	1	0	1	0	1	0	0	1	0	1	1	(1D4BH)

executing ROR results in:

CY		bit15															bit0	
1	R_{12}:	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_{reg}

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_n

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_0.

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:

	(MSB)							(LSB)	
SCB R_n	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, R5=3609H, CY Flag=1

register R_6 becomes 2079H and the carry flag is reset when

SBC R_5

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

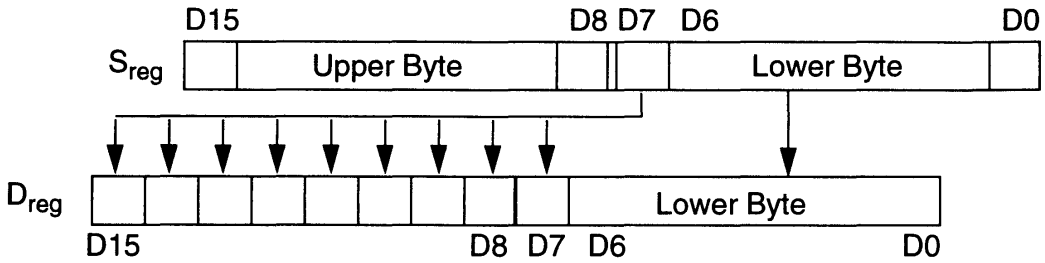
executing,

```
LM      R1, (3230H)
INC     R1
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_0.

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_n

Operation: R_n (low byte) → (xx) (n=0~15, xx=0~65535)

R_n (high byte) → (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_n, 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:

SM (xx), R_n

(MSB)							(LSB)	
0	0	1	1	1	1	1	0	(3EH)
1	1	1	1	n (0H~FH)				(FnH)
x (00H~FFH)								(ADRS Lower Byte)
x (00H~FFH)								(ADRS Upper Byte)

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_4= 438CH and RAMBR=70H

the following program execution,

SM (0B492H), R_4

will result in (70:B492H) =8CH, (70:B493H) = 43H.

9.82 SMS (yy), R_n

Operation: R_n (low byte) → (yy) (n=0~15, yy=0~510*)
R_n (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_n, 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

Opcode:

	(MSB)							(LSB)	
SMS (yy), R_n	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 x 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.

Example: Under the following conditions,

Register R_{11}= ABCDH, RAMBR=71H

the following program is execution,

Syntax	Opcode
SMS (194H), R_{11}	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_m)

Operation: S_{reg} (low byte) → (R_m) (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_0~R_{11}. 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_0.

Flags affected:

B	ALT1	ALT2	O/V	S	CY	Z
0	0	0	-	-	-	-

B : Reset
ALT1 : Reset
ALT2 : Reset

Opcode:

	(MSB)							(LSB)	
STB (R_m)	0	0	1	1	1	1	0	1	(3DH)
	0	0	1	1	m (0H~BH)				(3mH)

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_{reg}:R_5, R_5=216CH, R_8=9A34H, RAMBR=70H

and when the following program is executed,

```
STB     (R8)
```

the result is (70:9A34H)=6CH.

9.84 STOP

Operation: 0 → 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	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) → (R_m) (m=0~11)

S_{reg} (high byte) → (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~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_n

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_0.

The operand can be any of registers R_0~R_{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 overflow, else reset (Set on adder overflow.)
Z : Set if result is zero.

Opcode:

	(MSB)				(LSB)	
SUB R_n	0	1	1	0	n (0H~FH)	(6nH)

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_4 becomes 30EFH when

SUB R_8

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_0.

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:

SUB #n

(MSB)							(LSB)	
0	0	1	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_0, D_{reg}: R_0, R_0=329BH

the register R_0 becomes 3291H when

SUB #10

is executed.

9.88 SWAP

Operation: S_{reg} (low byte) → D_{reg} (high byte)

S_{reg} (high byte) → 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_0.

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_3, D_{reg}: R_{13}, R_3=48D0H

the register R_{13} becomes D048H when

SWAP

is executed.

9.89 TO R_n

REGISTER PREFIX INSTRUCTION

Operation: If B Flag = 0 then set D_{reg} to R_n
else $S_{reg} \rightarrow R_n$ (n=0~15)

Description: This instruction specifies register R_n as the destination register. The destination register can be any of registers R_0 ~ 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_n	0	0	0	1	n (0H~FH)		(1nH)

Machine Cycles:

ROM execution time	3 cycles
RAM execution time	3 cycles
Cache RAM execution time	1 cycles

Example: Under the following conditions:

R_6= 7106H, R_3=0028H

the register R_4 becomes 712EH when the following program is executed.

```
FROM   R6
TO     R4
ADD    R3
```

9.90 UMULT R_n

Operation: S_{reg} (low byte) * R_n (low byte) $\rightarrow$ D_{reg}

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_n, 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_0.

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_n	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_{reg}: R_3, D_{reg}: R_0, R_3= 364FH, R_8= B2CFH

the register R_0 becomes 3FE1H when

UMULT R_8

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

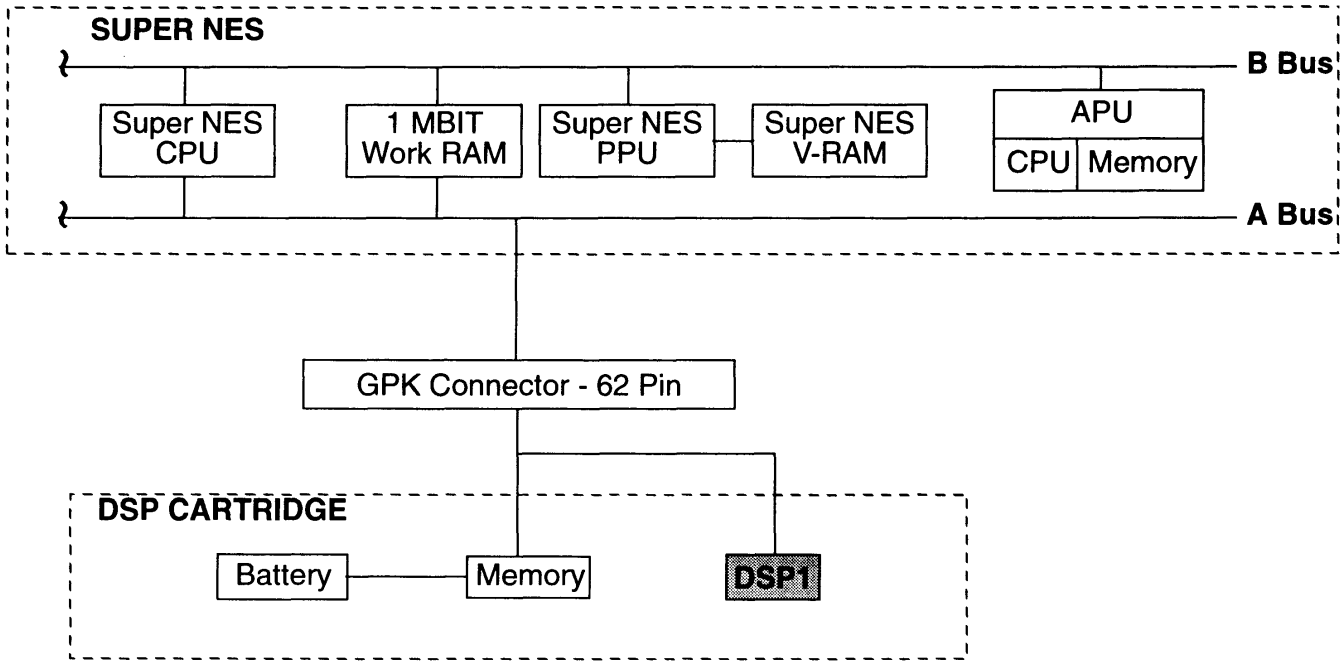
Because numerous commands for 3-dimensional graphics are incorporated, DSP1 is especially useful for 3-dimensional games, such as those involving flight simulation.

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.

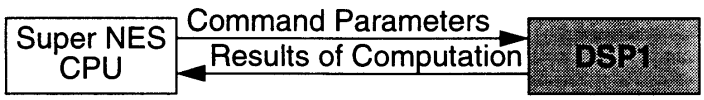


Figure 3-1-2 Super NES CPU and DSP1 Communications

Command execution between the Super NES and DSP1 is demonstrated below.

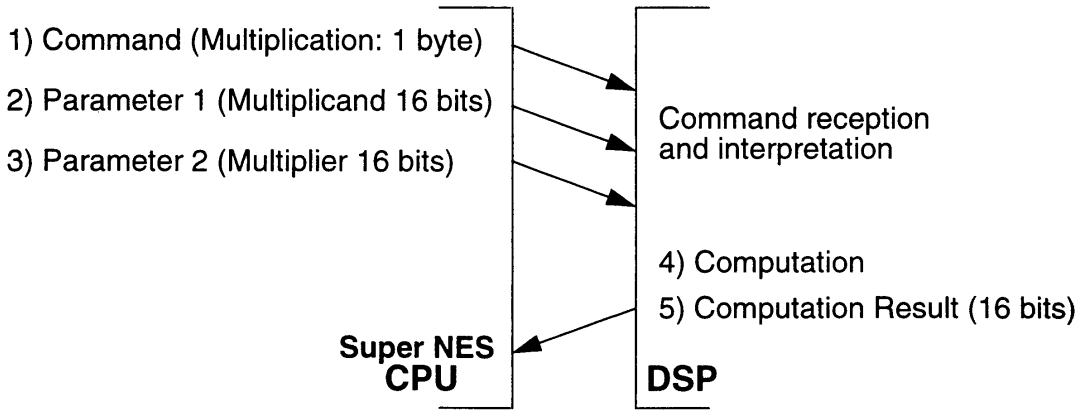


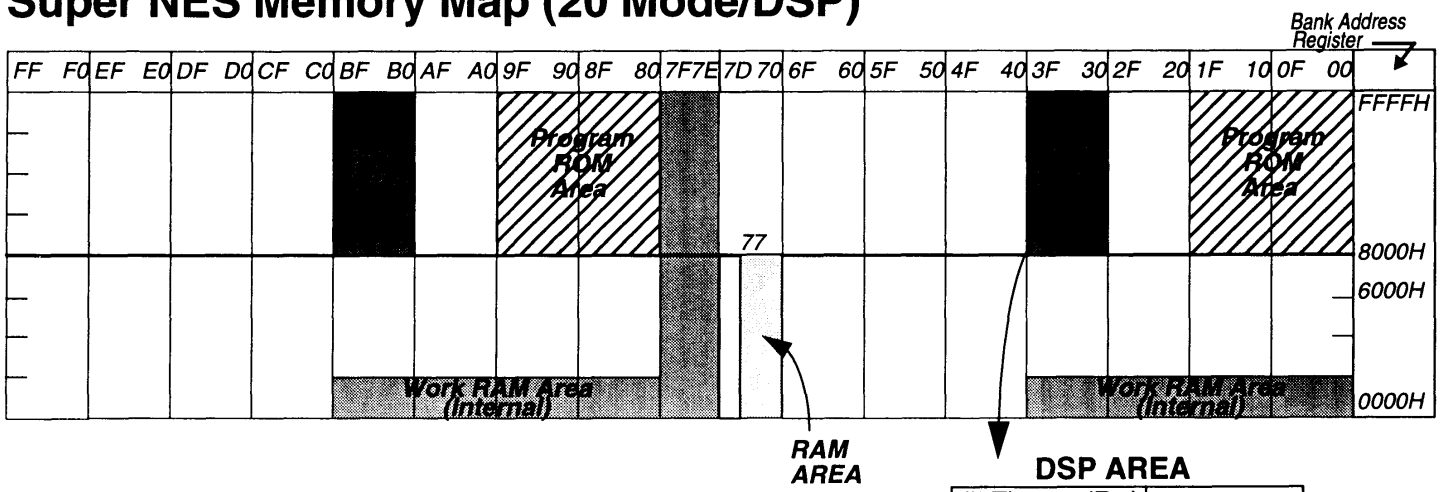
Figure 3-1-3 DSP1 Command Execution

1.6 MEMORY MAPPING

1.6.1 MODE 20/DSP

Figure 3-1-4 Mode 20/DSP Memory Map

Super NES Memory Map (20 Mode/DSP)



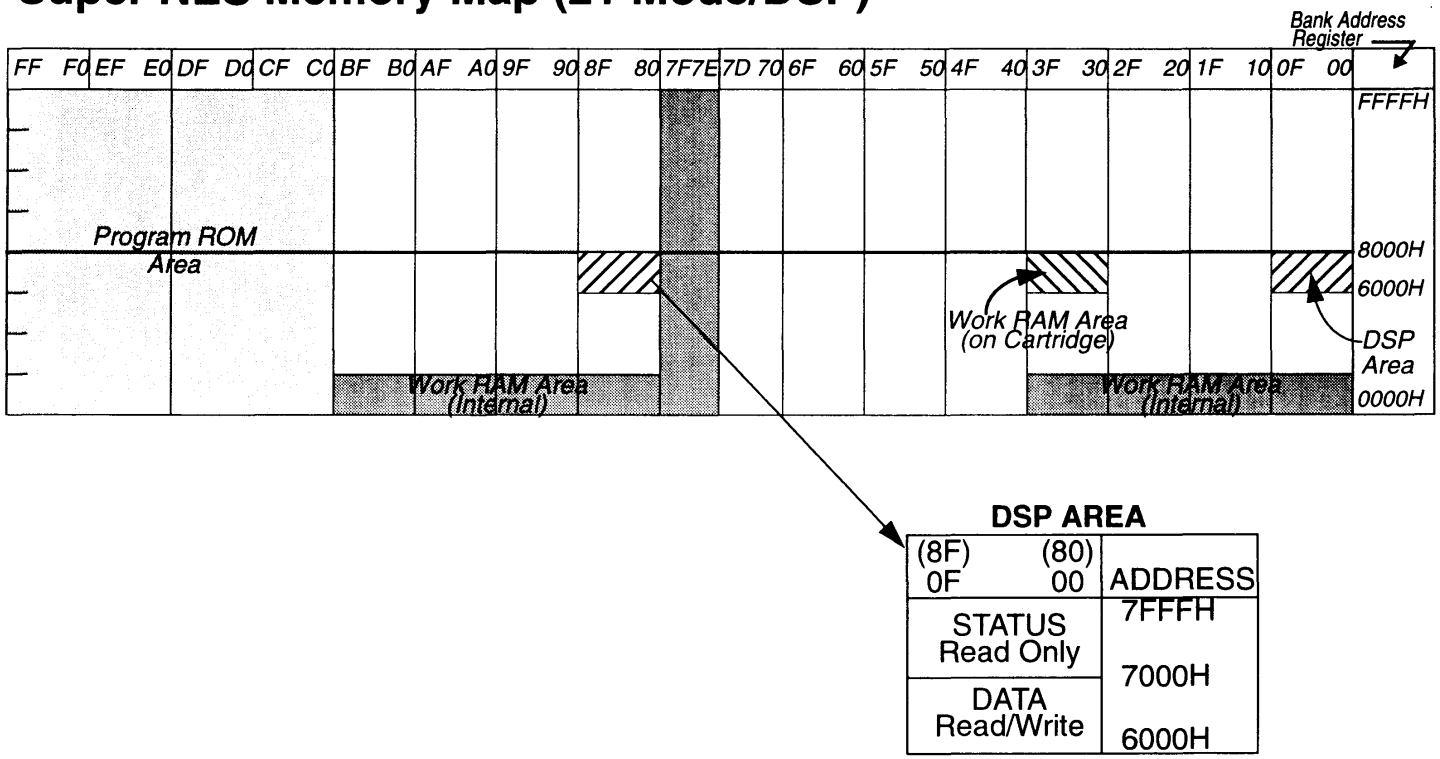
DSP AREA

(BF) 3F	(B0) 30	ADDRESS
STATUS Read Only		FFFFH
DATA Read/Write		C000H
		8000H

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, interger)	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 1AH	224+209(n-1) 224+208(n-1)	29.5+27.5(n-1) 29.5+27.4(n-1)
	Project	Object projection calculation	06H	627	82.5
	Target	Coordinate calculation of a selected point on the screen	0EH	228	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.

PARA-METER	DESCRIPTION	# BITS	DATA RANGE	UNIT
A	Angle	8	$-\pi \sim +\pi$ (-180°~+180°)	$2\pi/2^{16}$
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.

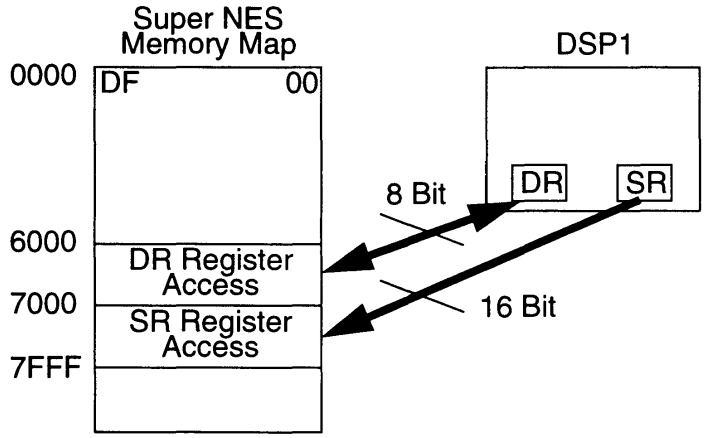


Figure 3-4-1 Super NES/DSP1 Memory Mapping (Mode 21)

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"

D15	D14	D13	D12	D11	D10	D9	D8
RQM							

D7	D6	D5	D4	D3	D2	D1	D0

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.

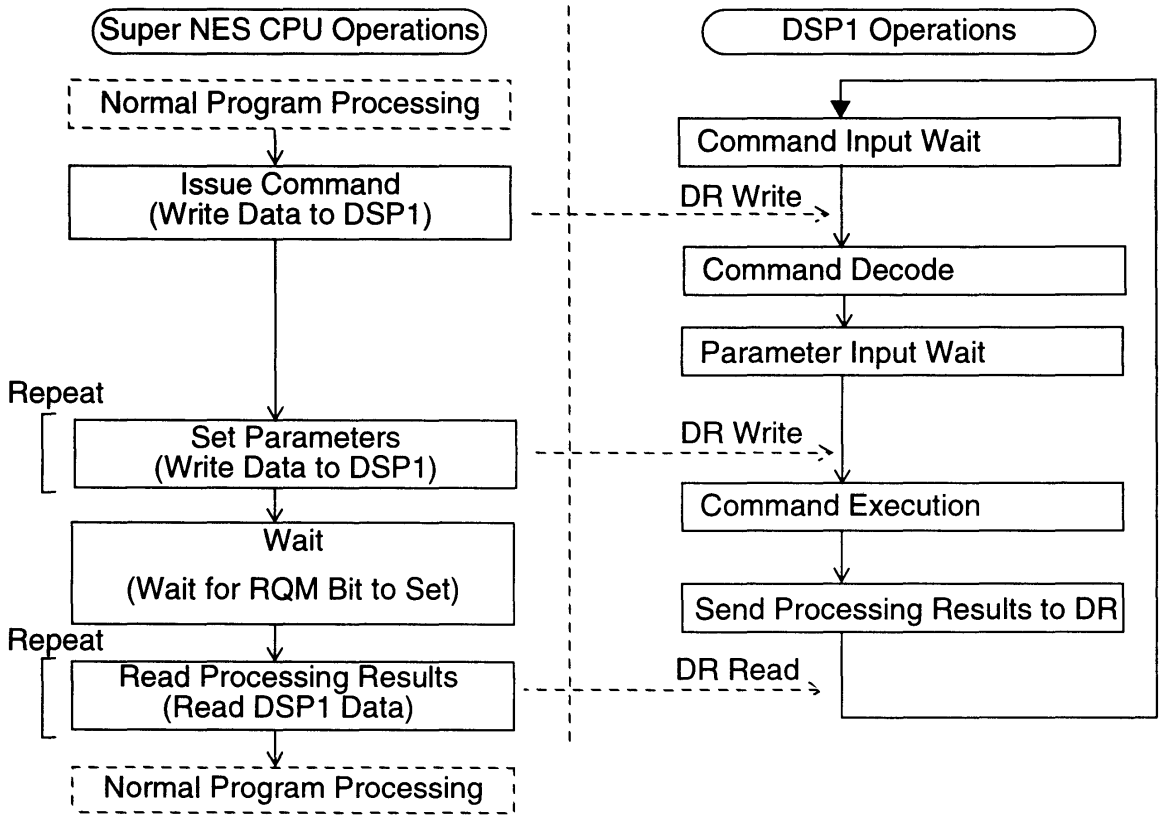


Figure 3-4-3 DSP1 Operations Flow Diagram

Super NES CPU/DSP1 Operational Timing

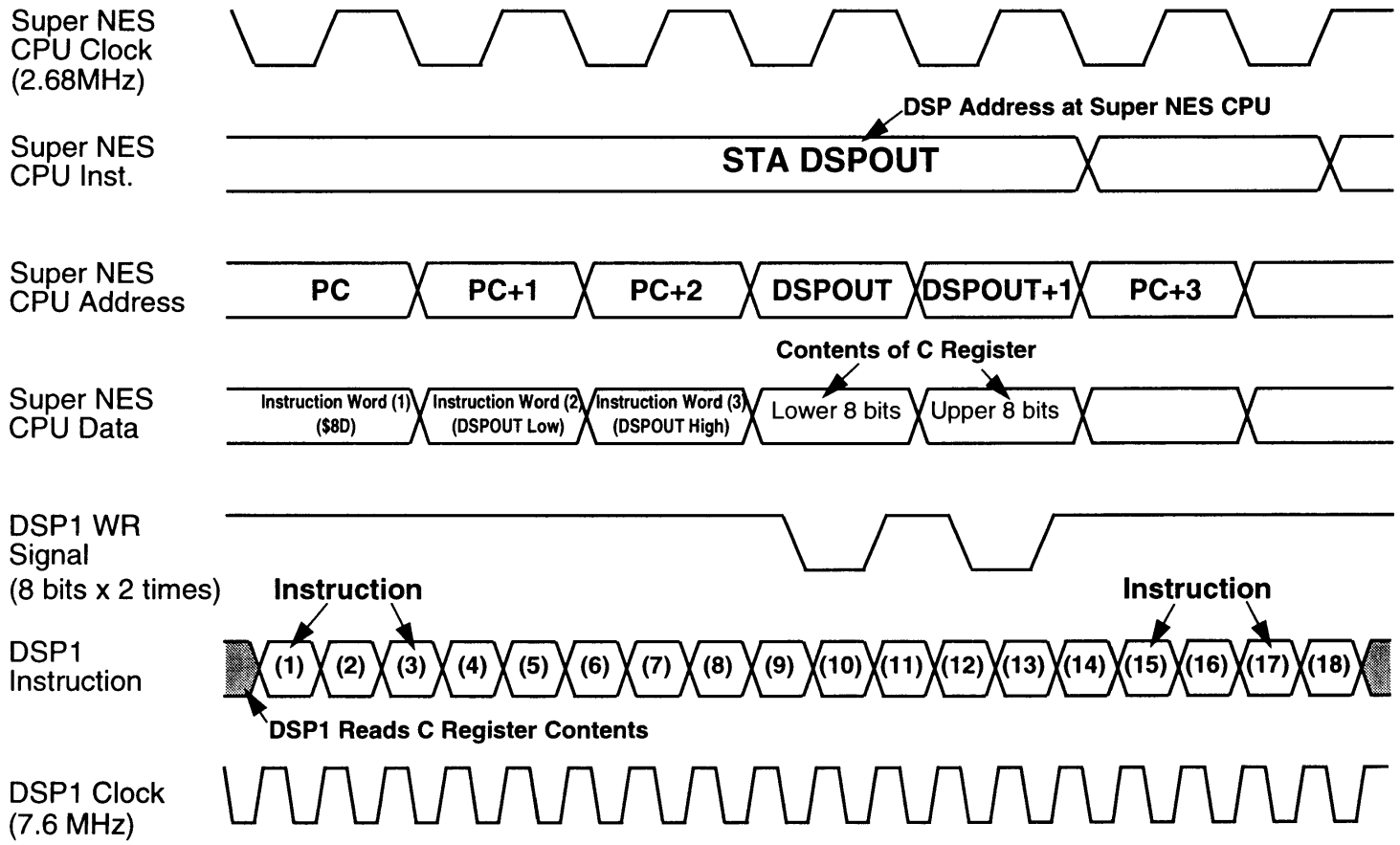


Figure 3-4-4 Super NES CPU/DSP1 Operational Timing

Chapter 5 Description of DSP1 Commands

5.1 GENERAL CALCULATION

5.1.1 16-BIT MULTIPLICATION (DECIMAL, INTEGER)

Name:	Multiply		
Code:	00H*3		
Parameters:	Input	k[T/I]	Multiplicand
		l[T/I]	Multiplier
	Output	M[T/H2]	Product (rounded fraction ≤ 15 bits)

Function: This command determines the product, M, of decimal K and l. 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:

$$k \times l = M$$

Number of Process Cycles:	Input	1. Command Input	6
		2. k input	12
		3. l 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:	θ[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 θ 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:

$$C = r(\cos\theta) \qquad S = r(\sin\theta)$$

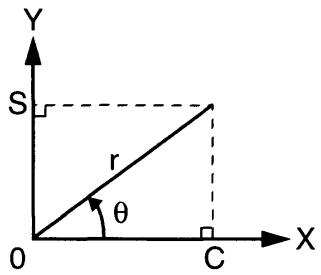


Figure 3-5-1 Trigonometric Calculation

Number of Process Cycles:

Input	1. Command Input	6
	2. θ 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 sinθ and cosθ.

[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:

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:

$$(x^2 + y^2 + z^2) = L$$

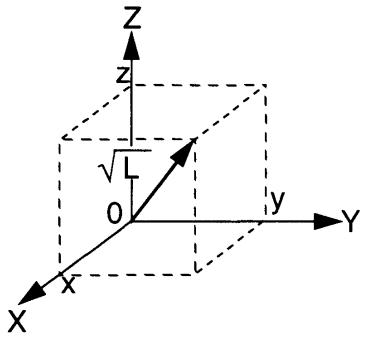


Figure 3-5-2 Vector Calculation

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:

$$x^2 + y^2 + z^2 - r^2 = D$$

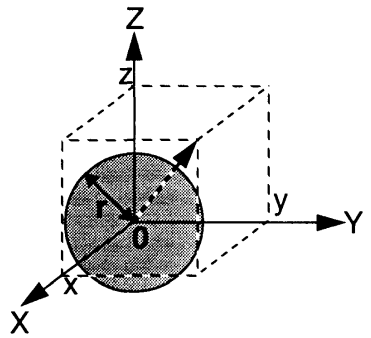


Figure 3-5-3 Vector Size Comparison

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: $\sqrt{x^2 + y^2 + z^2} = R$

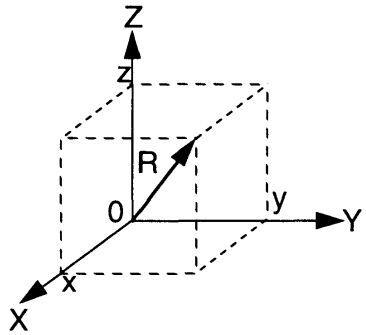


Figure 3-5-4 Vector Absolute Value Calculation

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 commands 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:	θ[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 θ.

Equation 5-7:

$$(x_1, y_1)\begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix} = (x_2, y_2)$$

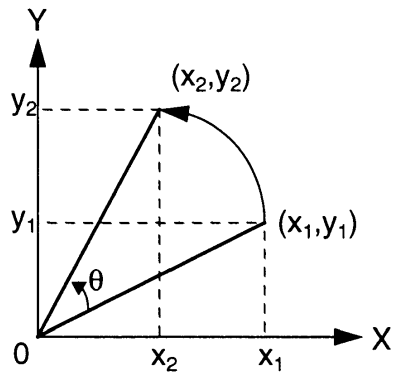


Figure 3-5-5 Two-Dimensional Coordinate Rotation

Number of Process Cycles:

Input	1. Command Input	6
	2. θ 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

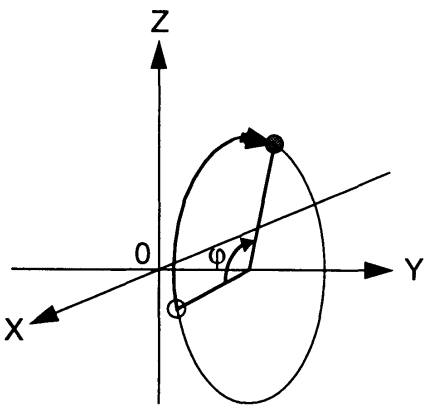
Parameters

Input:	θ[A]	Angle of rotation about the Z axis (positive from the Y axis to the X axis)
	ϕ[A]	Angle of rotation about the X axis (positive from the Z axis to the Y axis)
	φ[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

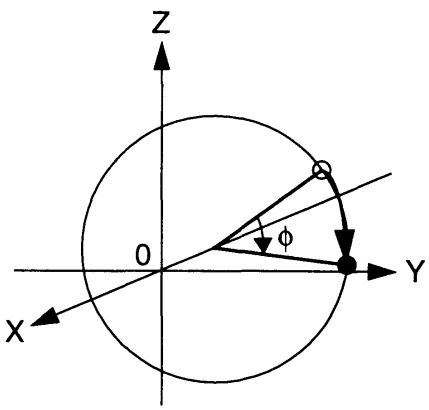
Function: This command determines the (X,Y,Z) coordinates when rotating (x,y,z) three-dimensionally. Rotation is performed in the order of ϕ about theY axis, φ about the X axis, and θ about the Z axis.

Equation 5-8:

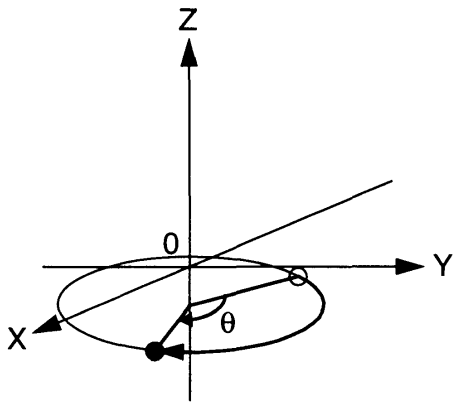
$$(x, y, z)\begin{bmatrix}\cos\varphi & 0 & \sin\varphi\\ 0 & 1 & 0\\ -\sin\varphi & 0 & \cos\varphi\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} = (X, Y, Z)$$



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.)

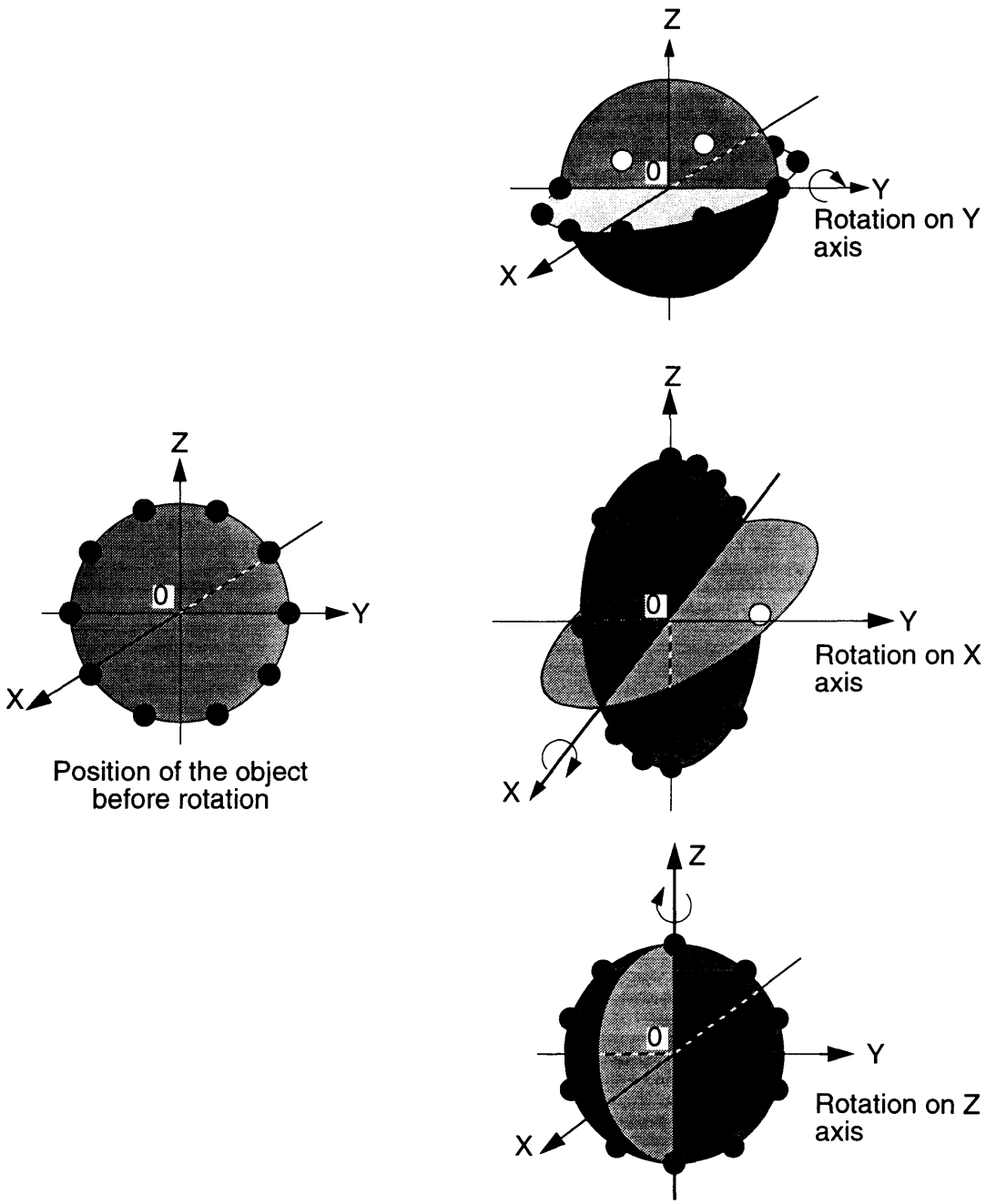


Figure 3-5-6 Examples of Three-Dimensional Rotation

5.4 PROJECTION CALCULATION

5.4.1 PROJECTION PARAMETER SETTING

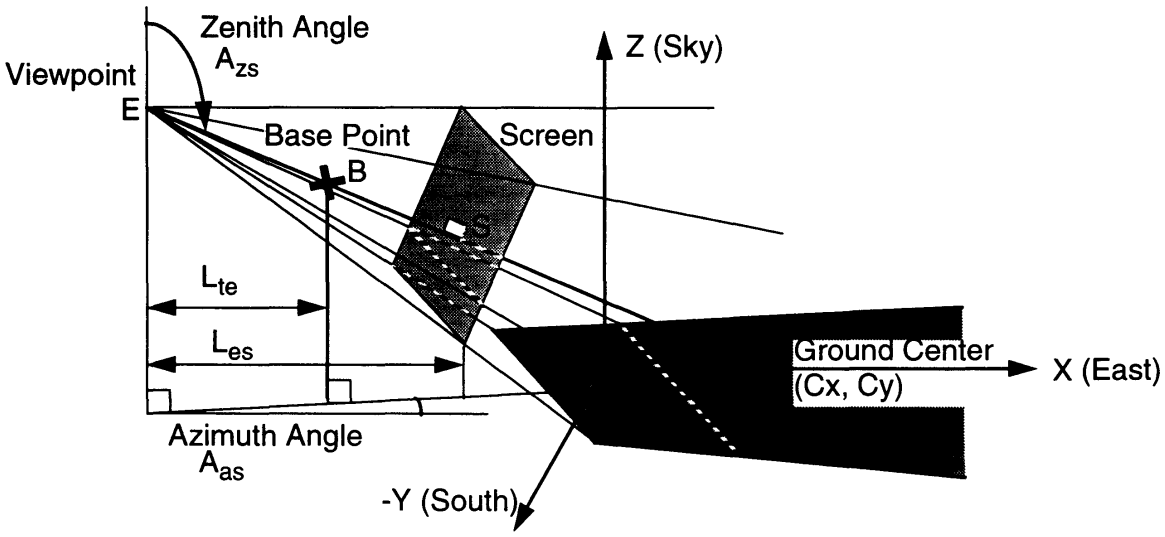
Name: Parameter

Code: 02H

Parameters:

Input:	F_x[CI]	X coordinate of base point (global coordinates)
	F_y[CI]	Y coordinate of base point (global coordinates)
	F_z[CI]	Z coordinate of base point (global coordinates)
	L_{fe}[U]	Distance between base point and viewpoint (Sets screen-sprite ratio.)
	L_{es}[U]	Distance between viewpoint and screen (The effect of screen angle considered; the screen horizontal distance is 256)
	A_{as}[A]	Azimuth angle of view line with respect to global coordinates. (East is 0° and positive toward the north)
	A_{zs}[A]	Zenith angle of view line with respect to global coordinates. (Zenith is 0°, 0°~180°).
Output:	V_{of}[I]	Raster number of imaginary center
	V_{va}[I]	Raster number representing horizontal line.
	C_x[CI]	X coordinate of the point projected on the center of the screen (ground coordinates)
	C_y[CI]	Y coordinate of the point projected on the center of the screen (ground coordinates)

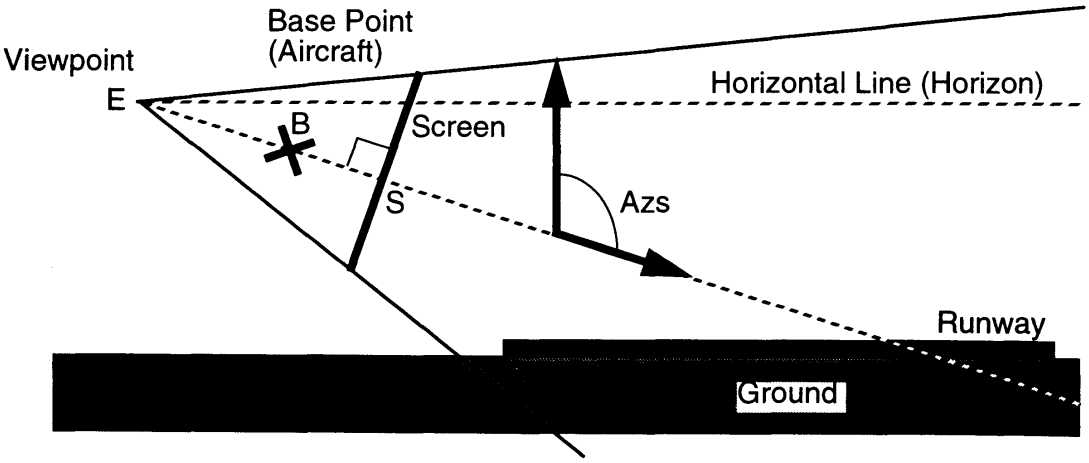
Function: This command sets various projection parameters and calculates the basic data used in subsequent processes. The command places the viewer behind a fixed point such as an airplane. If the distance between the fixed point and the view point is set to 0, then the viewer sees the display from the perspective of the airplane.



Assignment of Projection Parameter

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.



Relation between sight and projected plane (side view)

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_x input	11
		3. F_y input	2
		4. F_z input	2
		5. L_{fe} input	3
		6. L_{es} input	3
		7. A_{as} input	4
		8. A_{zs} input	839
	Output	1. V_{of}	2
		2. V_{va}	10
		3. C_x	5
		4. C_y	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_s[I]	Raster number where projection display begins.
Output:	A_n[I8]	Linear transformation matrix element A for each raster
	Bn[I8]	Linear transformation matrix element B for each raster
	C_n[I8]	Linear transformation matrix element C for each raster
	D_n[I8]	Linear transformation matrix element D for each raster

Function: 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.

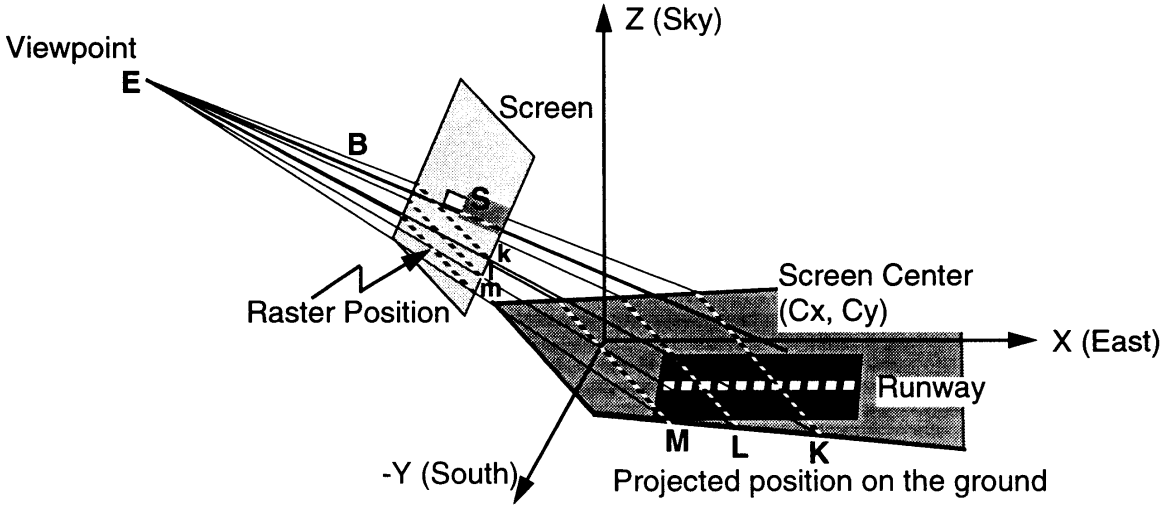


Figure 3-5-9 Calculation of Raster Data

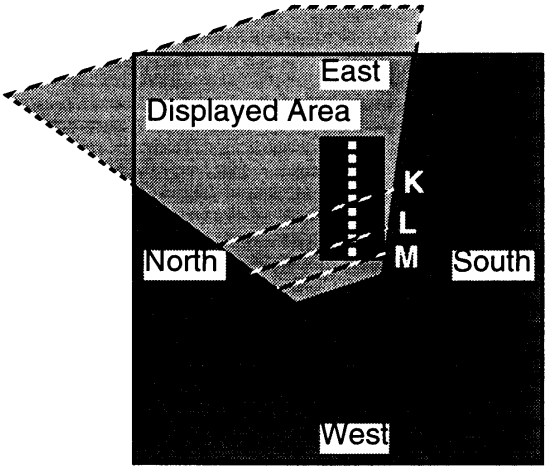


Figure 3-5-10 BG Screen and Displayed Area

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).

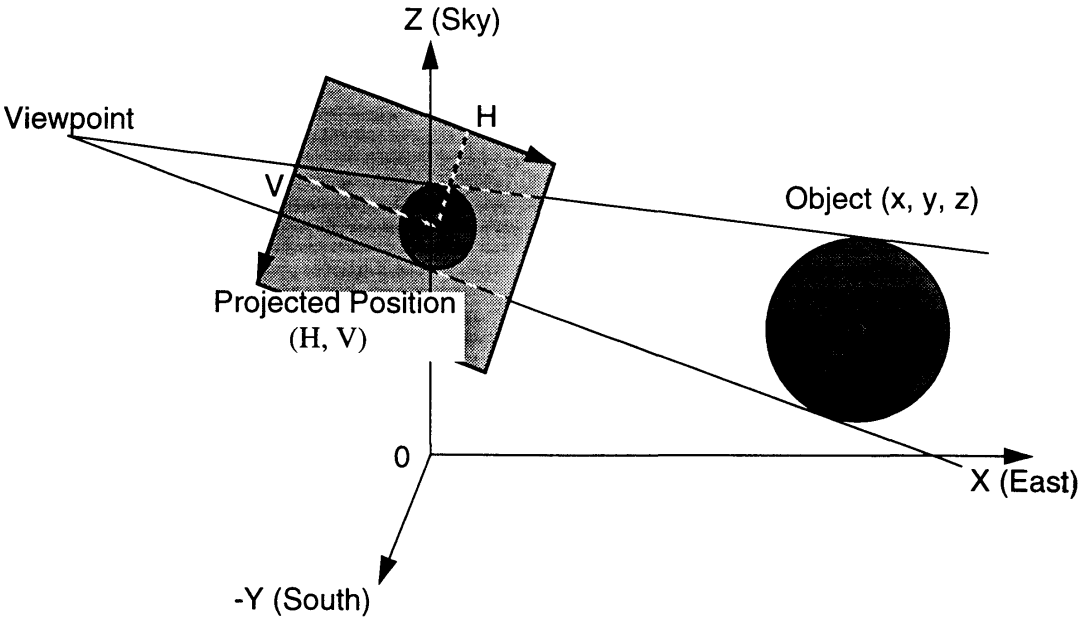


Figure 3-5-11 Calculation of Projected Position of Object

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*1

*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.

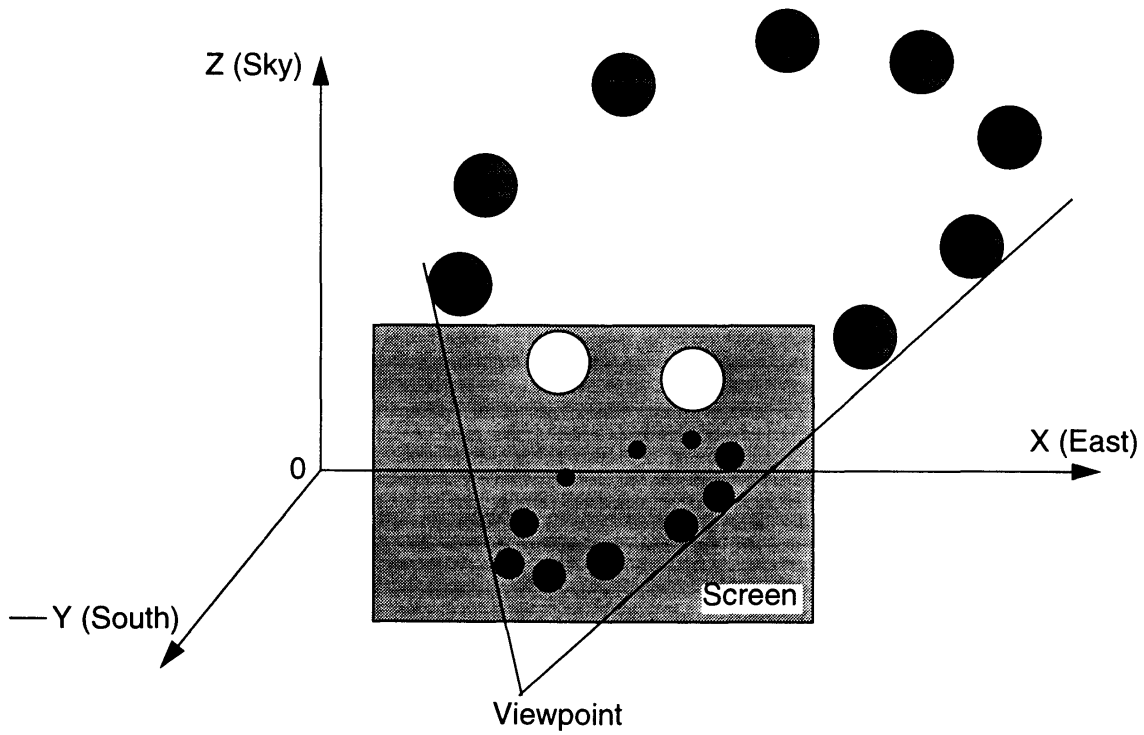


Figure 3-5-12 Projection Image of Object

5.4.4 COORDINATE CALCULATION OF A SELECTED POINT ON THE SCREEN

Name: Target

Code: 0EH

Parameters:

Input:

h[I]	H coordinate of selected point on the screen (screen coordinates, right is positive)
v[I]	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.

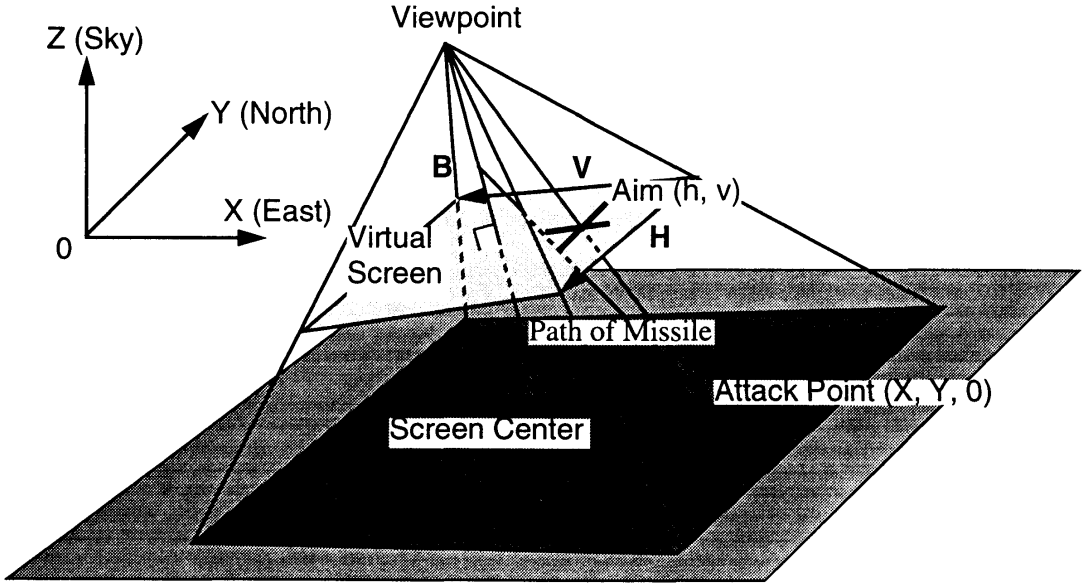


Figure 3-5-13 Calculation of Coordinates for the Indicated Point on the Screen

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*1

*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.

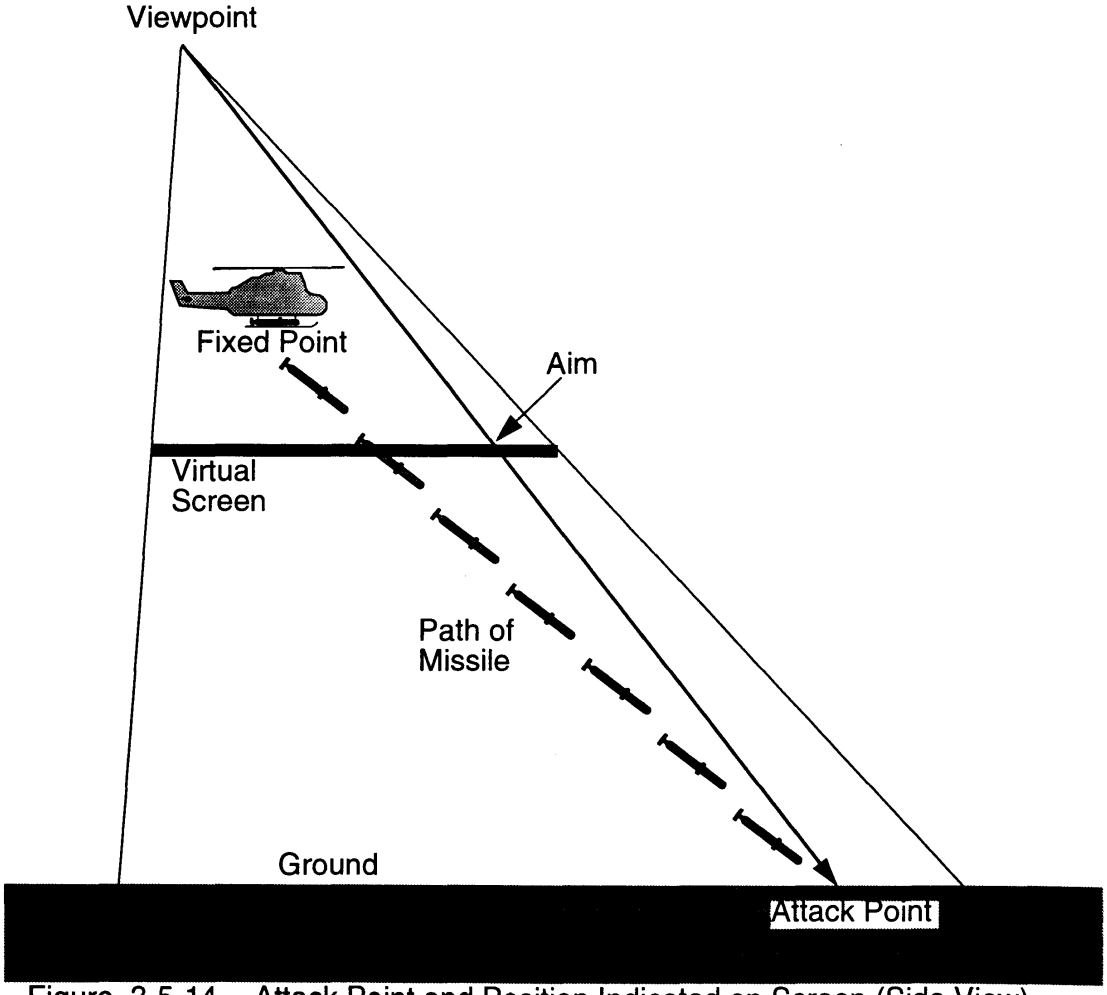


Figure 3-5-14 Attack Point and Position Indicated on Screen (Side View)

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
	θ[A]	Rotational angle about the Z axis (from Y axis to X axis is +)
	ϕ[A]	Rotational angle about the X axis (from Z axis to Y axis is +)
	φ[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 ϕ about the Y axis (north-south), φ about the X axis (east-west), and θ 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:

$$m\begin{bmatrix}\cos\varphi & 0 & -\sin\varphi\\ 0 & 1 & 0\\ \sin\varphi & 0 & \cos\varphi\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & \sin\phi\\ 0 & -\sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} = M$$

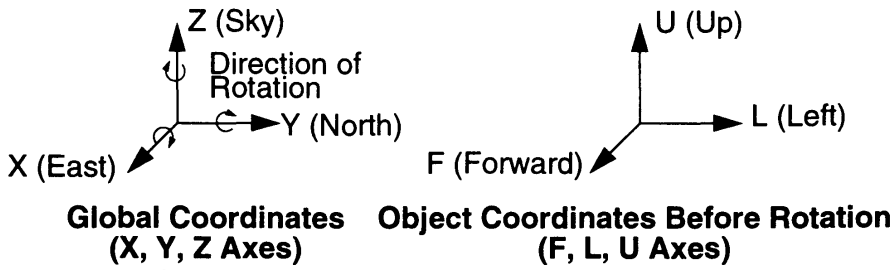


Figure 3-5-15 Attitude Computation

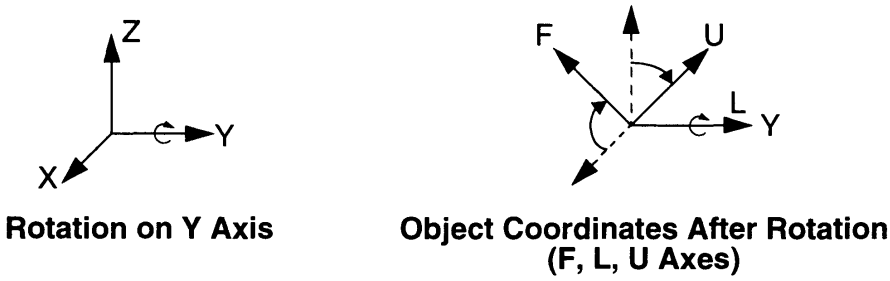


Figure 3-5-16 Object Coordinate Rotated on Y Axis

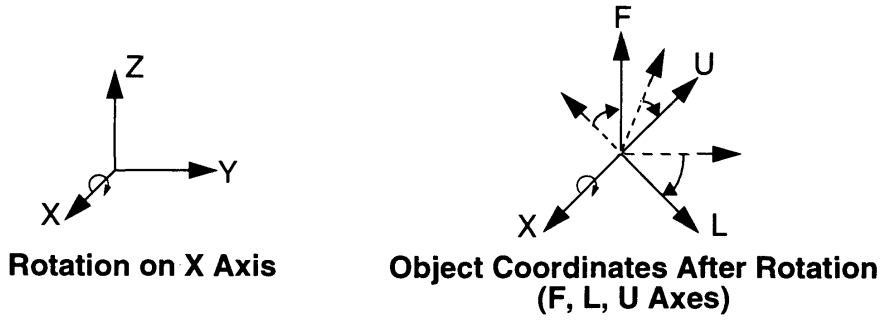


Figure 3-5-17 Object Coordinate Rotated on X Axis

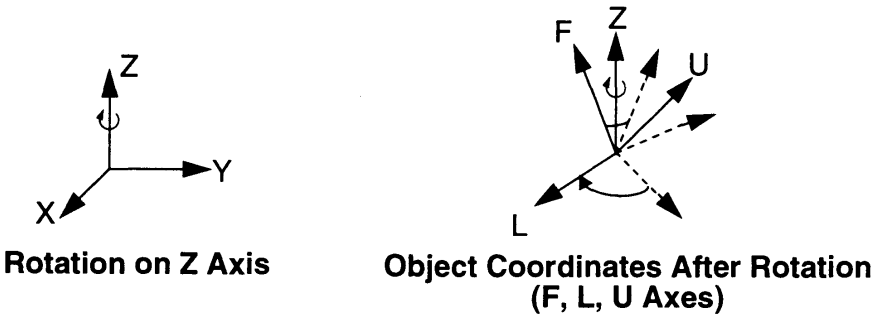


Figure 3-5-18 Object Coordinate Rotated on Z Axis

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*1

*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: 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).

Calculates the product with inverse of the matrix A when the code is 0DH ($M^{-1}=A^{-1}$).

Calculates the product with inverse of the matrix B when the code is 1DH ($M^{-1}=B^{-1}$).

Calculates the product with inverse of the matrix C when the code is 2DH ($M^{-1}=C^{-1}$).

Equation 5-10:

$$\frac{1}{2}(x, y, z)\, M^{-1} = (F, L, U)$$

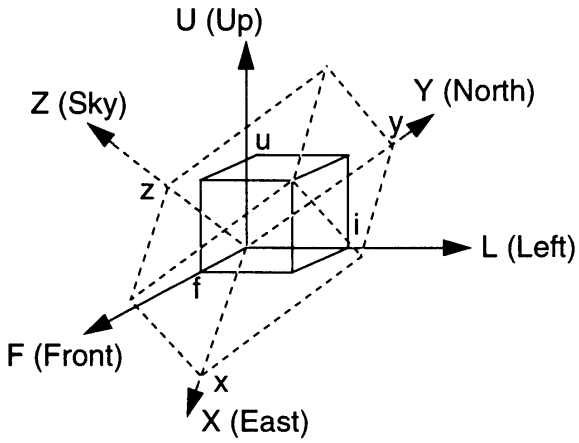


Figure 3-5-19 Conversion of Global to Objective Coordinates

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:

$$\frac{1}{2}(F, L, U)\,M = (X, Y, Z)$$

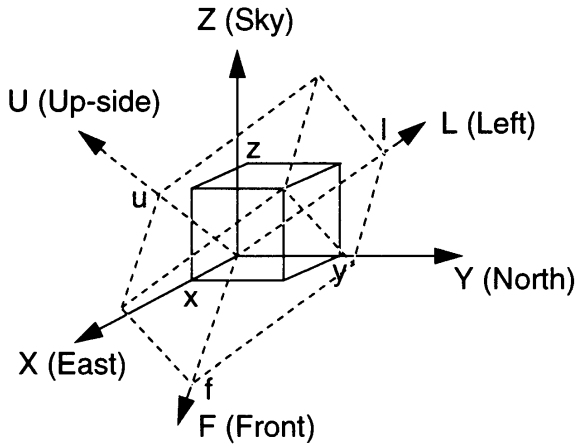


Figure 3-5-20 Conversion of Object to Global Coordinates

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 ATTITIDE 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, $S = x \bullet A_{fx} + y \bullet A_{fy} + z \bullet A_{fz}$
When the code is 1BH, $S = x \bullet B_{fx} + y \bullet B_{fy} + z \bullet B_{fz}$
When the code is 2BH, $S = x \bullet C_{fx} + y \bullet C_{fy} + z \bullet C_{fz}$

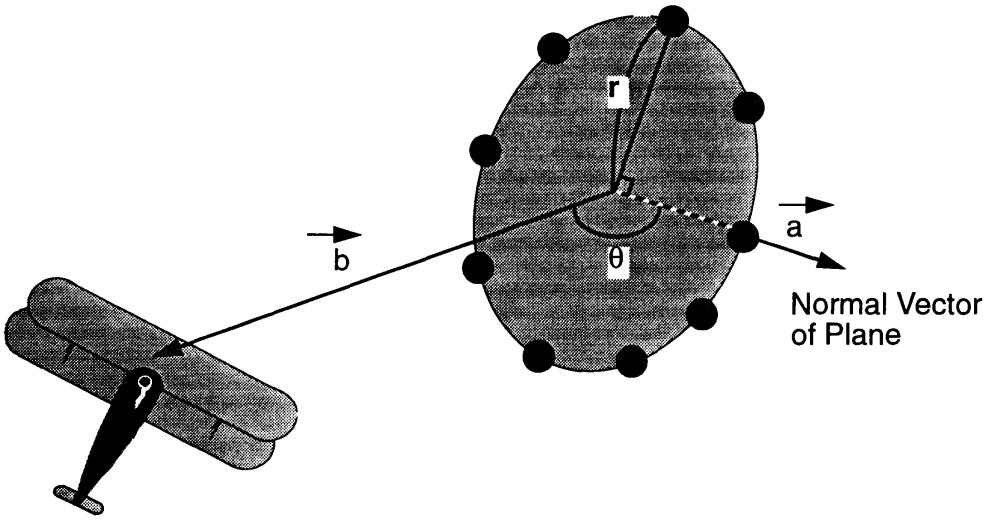


Figure 3-5-21 Calculation of Inner Product with Forward Attitude

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.

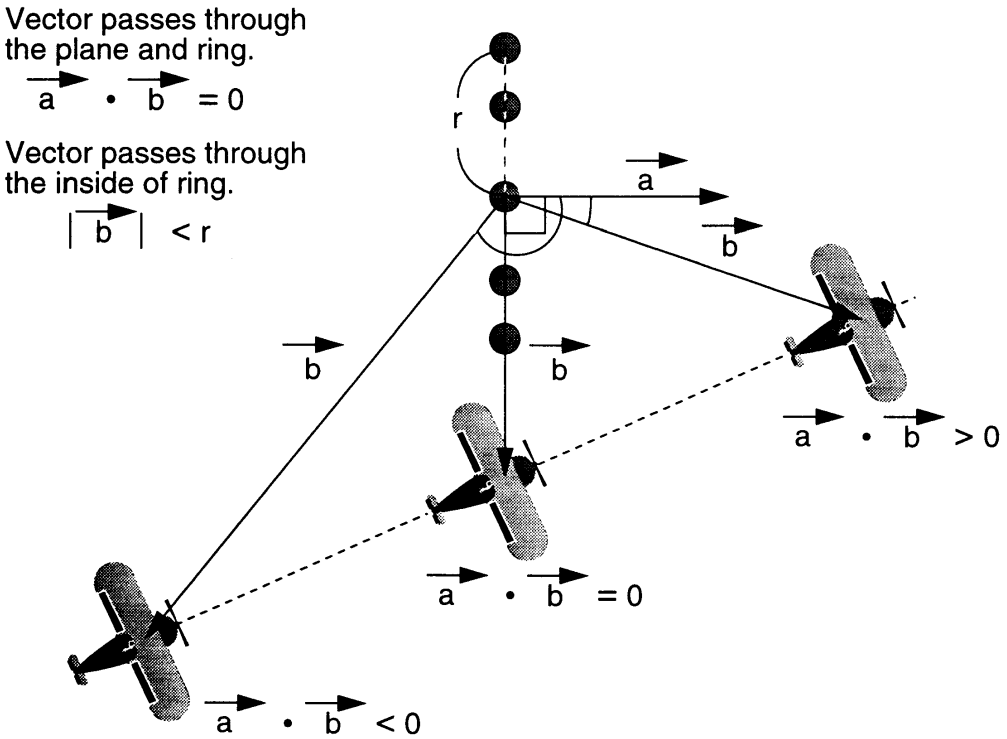


Figure 3-5-22 Position of Aircraft and Vector Code

5.6 NEW ANGLE CALCULATION

5.6.1 THREE-DIMENSIONAL ANGLE ROTATION

Name: Gyrate

Code: 14H

Parameters:

Input:	θ_i[A]	Angle of rotation about the Z axis (+ from the Y axis to the X axis)
	ϕ_i[A]	Angle of rotation about the X axis (+ from the Z axis to the Y axis)
	φ_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:	θ_o[A]	Rotational angle about the Z axis.
	ϕ_o[A]	Rotational angle about the X axis.
	φ_o[A]	Rotational angle about the Y axis.

Note: F, L, U axes represent the X, Y, Z axes when rotated ϕ_i, φ_i, θ_i only.

Function: This command determines the attitude angles (θ_o, ϕ_o, φ_o) of the body coordinates after the body with the attitude angle (θ_i, ϕ_i, φ_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 (θ_i, ϕ_i, φ_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:

$$\theta_i + \sec\phi_i\,(d\theta\cos\varphi_i - d\phi\sin\varphi_i) = \theta_o$$

$$\phi_i + (d\theta\sin\varphi_i + d\phi\cos\varphi_i) = \phi_o$$

$$\varphi_i - \tan\phi_i\,(d\theta\cos\varphi_i + d\phi\sin\varphi_i) + d\varphi = \varphi_o$$

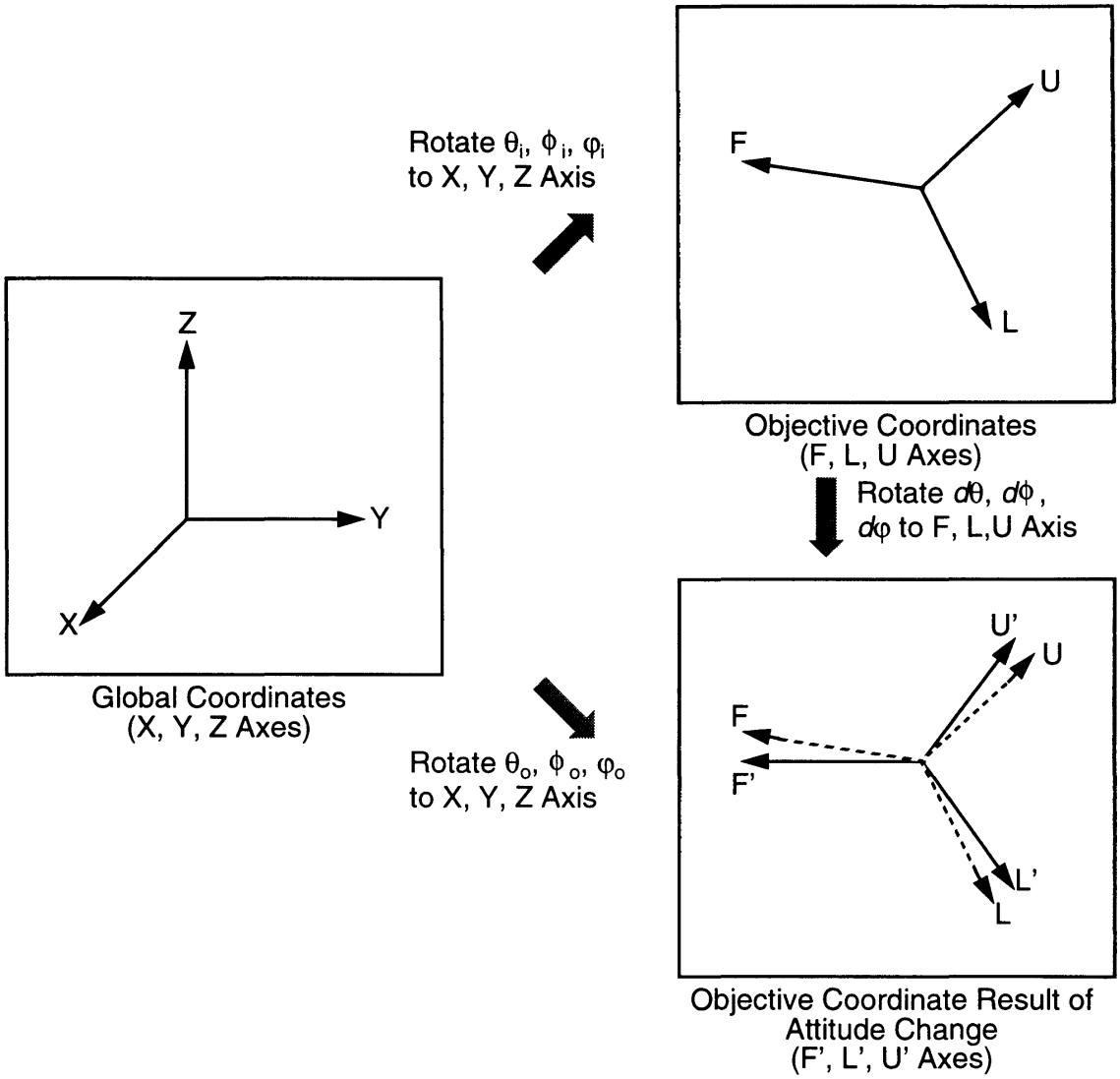


Figure 3-5-23 Calculation of Rotation Angle After Attitude Change

Number of Process Cycles:			
	Input	1. Command Input	6
		2. θ_i input	14
		3. ϕ_i input	2
		4. φ_i input	2
		5. $d\theta$ input	2
		6. $d\phi$ input	2
		7. $d\varphi$ input	406
	Output	1. θ_o output	2
		2. ϕ_o output	4
		3. φ_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

$$r(\cos\theta) = C$$
$$r(\sin\theta) = S$$

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

$$\theta_i + \sec\phi_i\,(d\theta\cos\varphi_i - d\phi\sin\varphi_i) = \theta_o$$
$$\phi_i + (d\theta\sin\varphi_i + d\phi\cos\varphi_i) = \phi_o$$
$$\varphi_i - \tan\phi_i\,(d\theta\cos\varphi_i + d\phi\sin\varphi_i) + d\varphi = \varphi_o$$

6.8 ROTATE

$$(x, y)\begin{bmatrix}\cos\phi & -\sin\phi \\ \sin\phi & \cos\phi\end{bmatrix} = (X, Y)$$

6.9 POLAR

$$(x, y, z)\begin{bmatrix}\cos\varphi & 0 & \sin\varphi\\ 0 & 1 & 0\\ -\sin\varphi & 0 & \cos\varphi\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} = (X, Y, Z)$$

6.10 ATTITUDE

$$m\begin{bmatrix}\cos\varphi & 0 & -\sin\varphi\\ 0 & 1 & 0\\ \sin\varphi & 0 & \cos\varphi\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & \sin\phi\\ 0 & -\sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} = M$$

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.

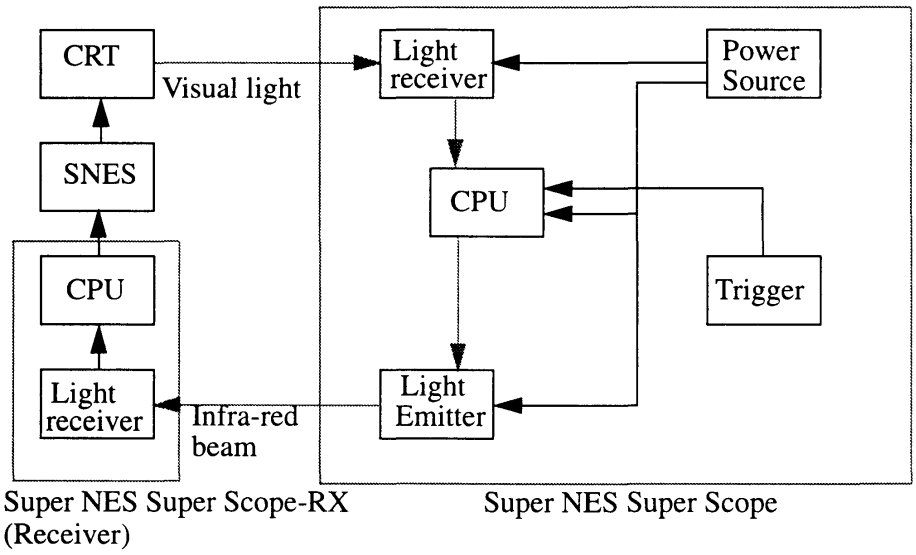


Figure 4-1-1 Signal Flow

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.

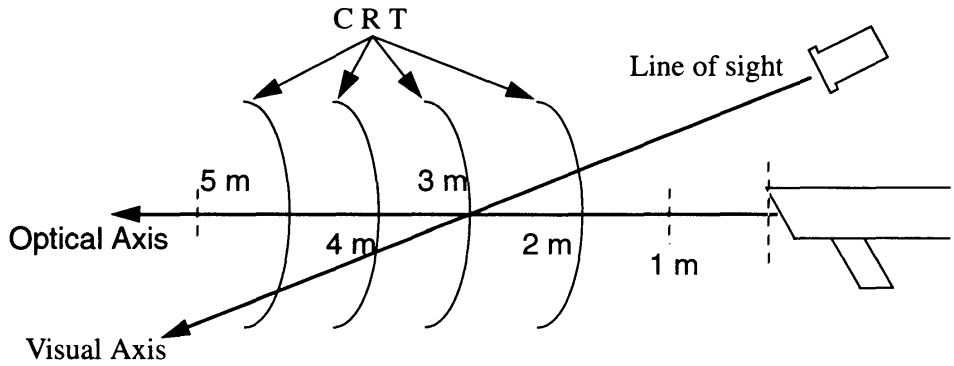


Figure 4-1-2 Optical Alignment

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.

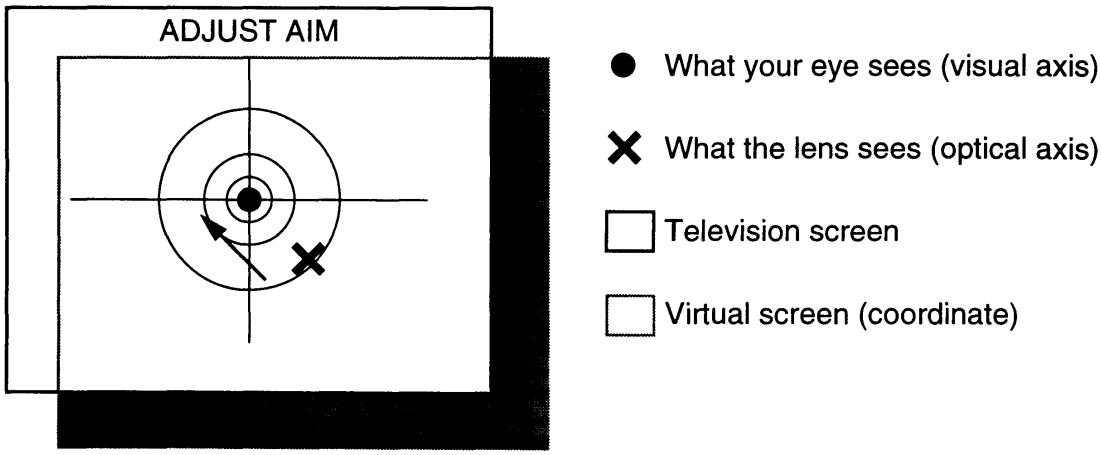


Figure 4-1-3 Virtual Screen Alignment

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 mm, 30 ϕ
- 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..

	D7							D0	
421A (H)	1	1	1	1	1	1	1	1	Controller #2
421B (H)	TRIGGER	CURSOR	MULTIPLE SHOTS	P/S	0	0	NULL	NOISE	

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 florescent material (phosphor coating) is fused to the inside of the Braun tube's glass screen. Light is emitted when electrons bombard this florescent 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.)

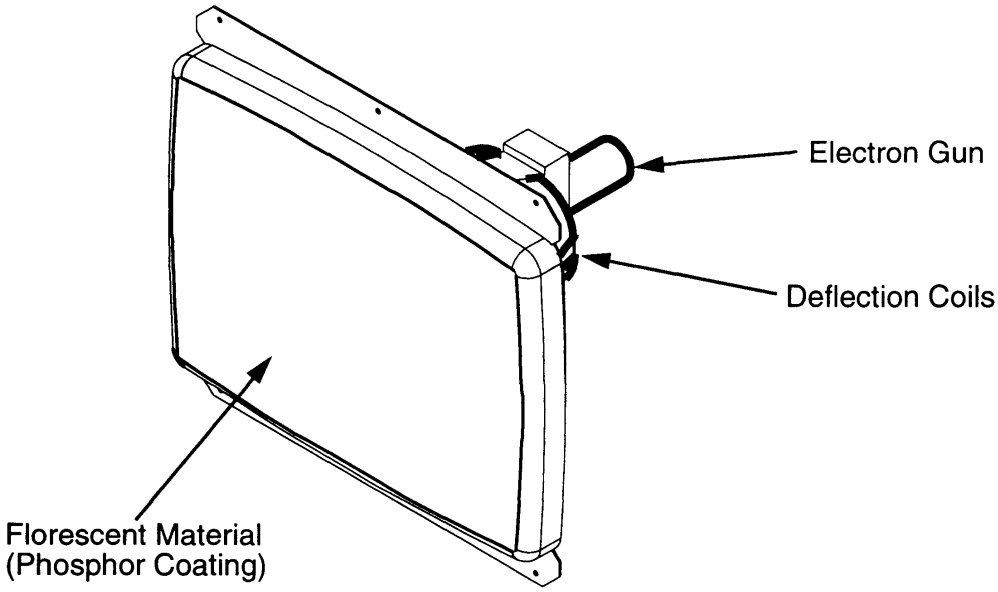


Figure 4-2-1 Picture Tube

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.

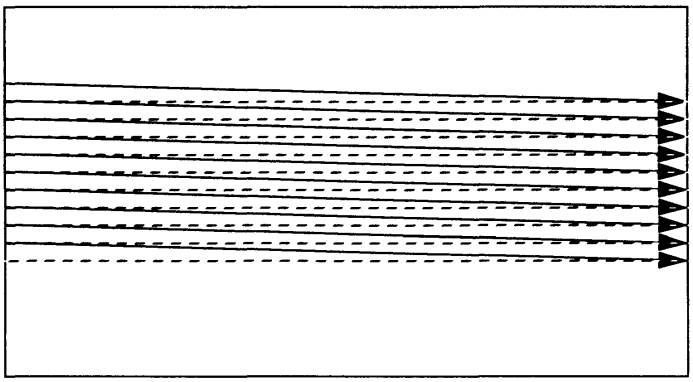


Figure 4-2-2 Scanning

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 florescent 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.

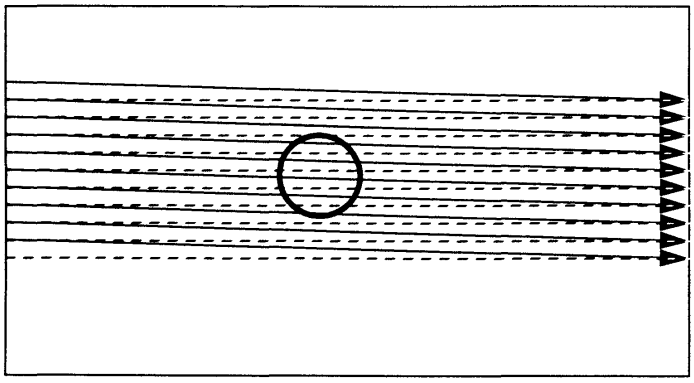


Figure 4-2-3 Area Seen by Super NES Super Scope

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..

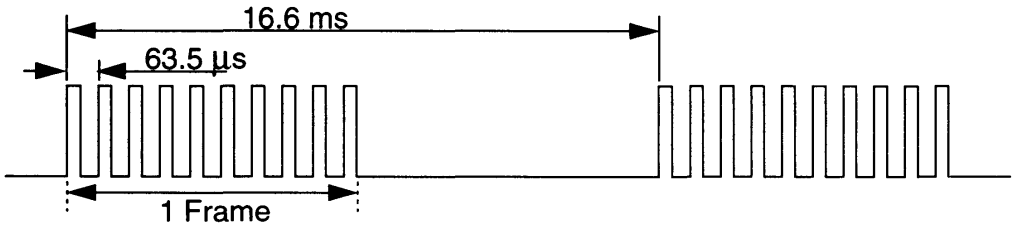


Figure 4-2-4 Vertical Positioning

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.)

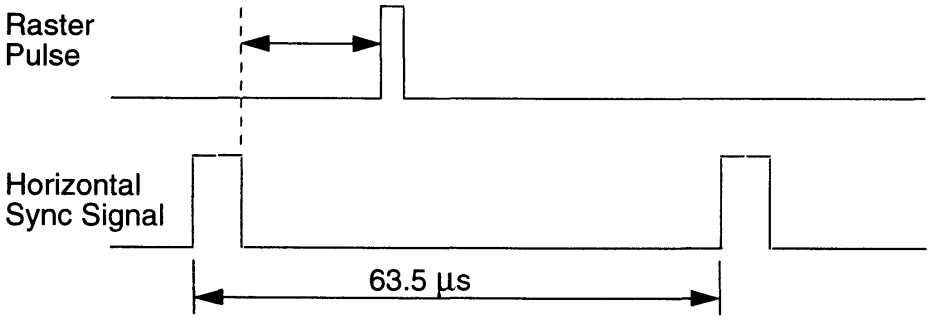


Figure 4-2-5 Horizontal Positioning

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.

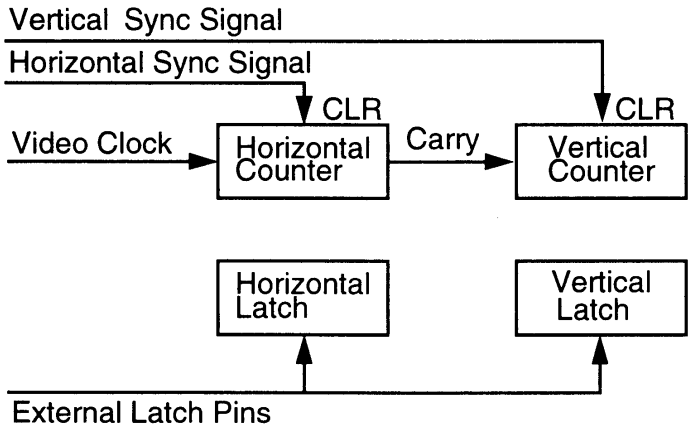


Figure 4-2-6 Horizontal/Vertical Counter

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

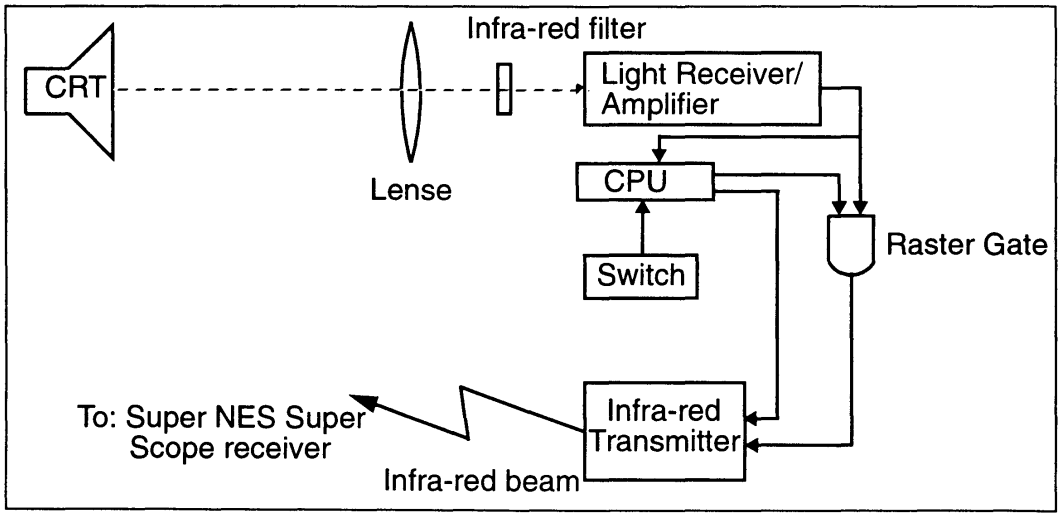


Figure 4-3-1 Super NES Super Scope Block Diagram

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

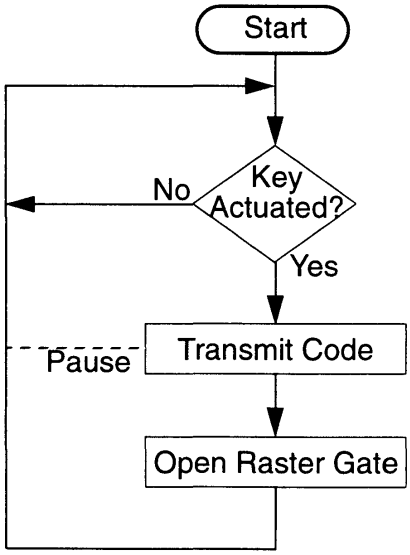


Figure 4-3-2 Super NES Super Scope Flow Diagram

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.

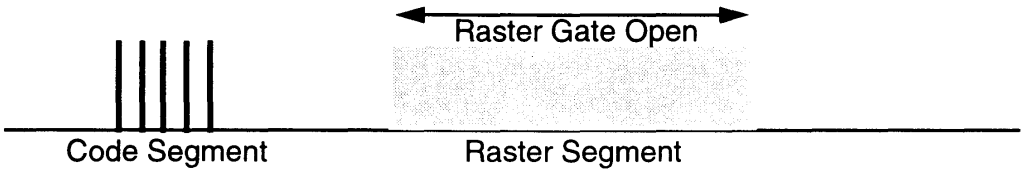


Figure 4-3-3 Raster Signal

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.

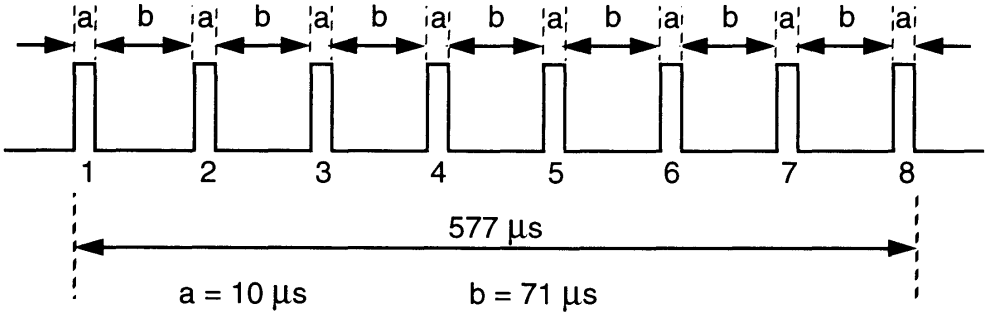
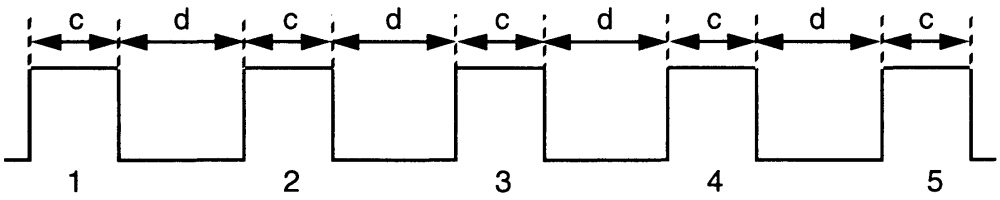


Figure 4-3-4 Definition of one byte

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.



c = 577 µs d = 651 µs

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

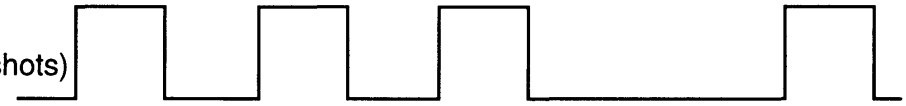
3.4.3 COMMUNICATION CODES

Four codes are defined as follows.

Cursor

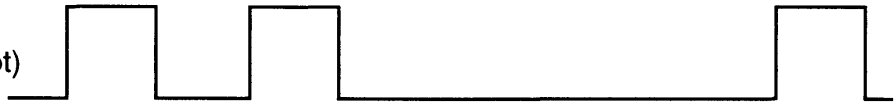
Outputs while cursor is held down

Trigger (multiple shots)



When the "multiple shots" switch is "on", pulses are output while the trigger is held depressed.

Trigger (single shot)



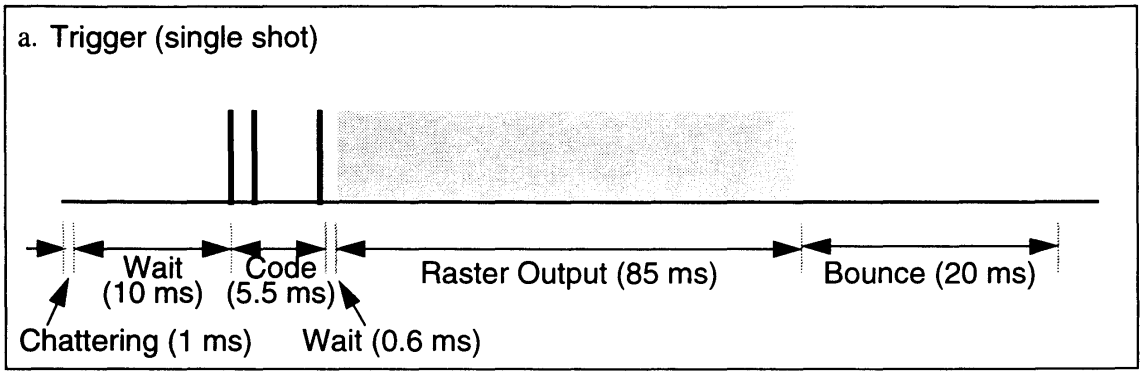
When the "multiple shots" switch is "off", one output pulse occurs each time the trigger is depressed.

Pause

This output pulse occurs only when the pause key is depressed.

Figure 4-3-6 Definitions of codes.

3.4.4 RASTER SIGNAL TRANSMISSION TIMING



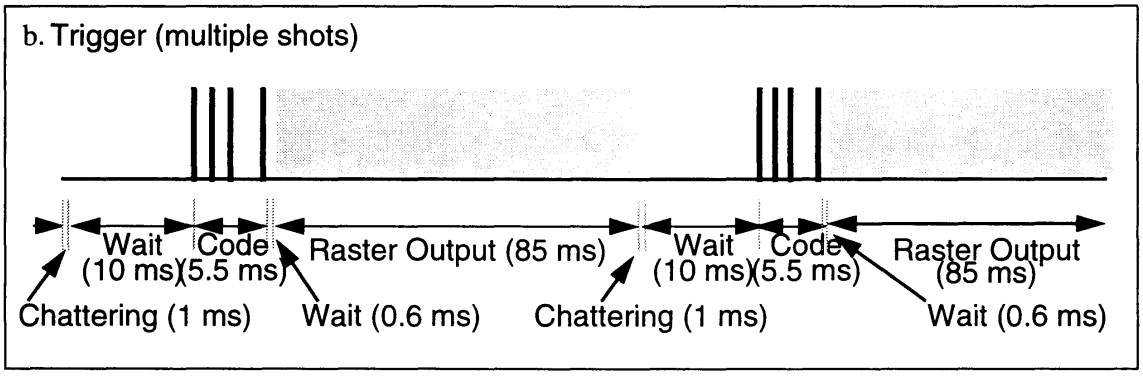
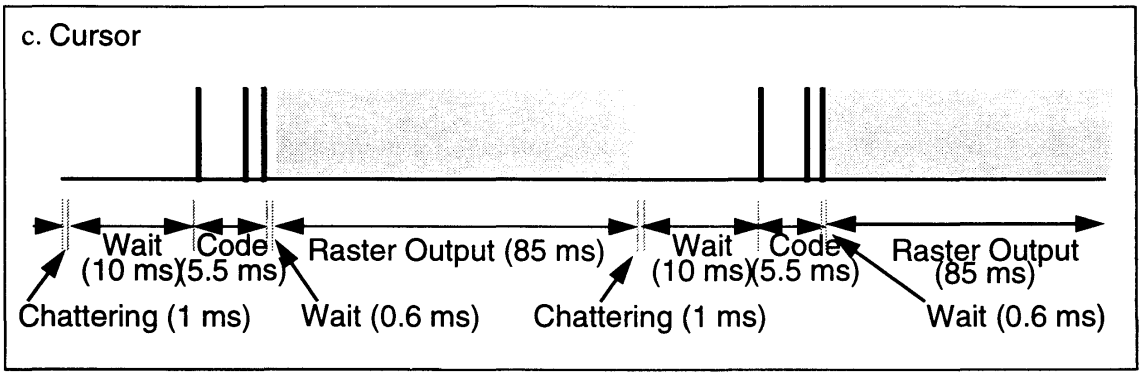


Figure 4-3-7 Raster Signal Transmission Timing, part 1

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



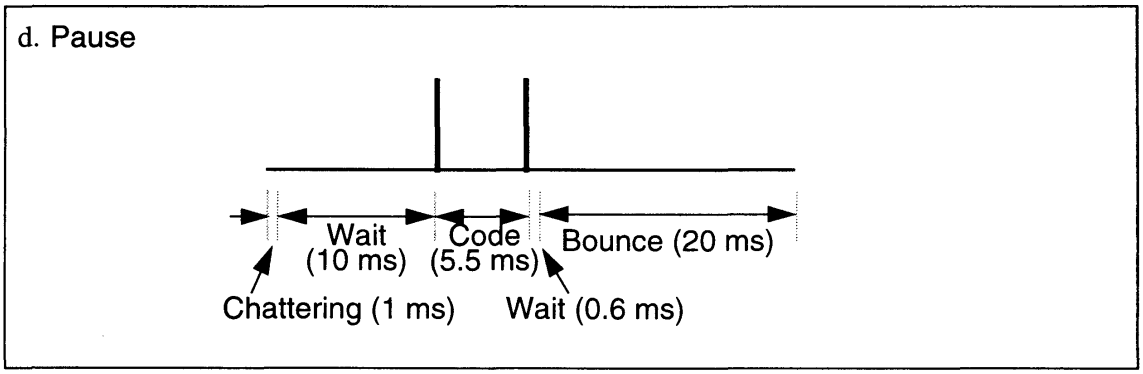


Figure 4-3-7 Raster Signal Transmission Timing, part 2

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.

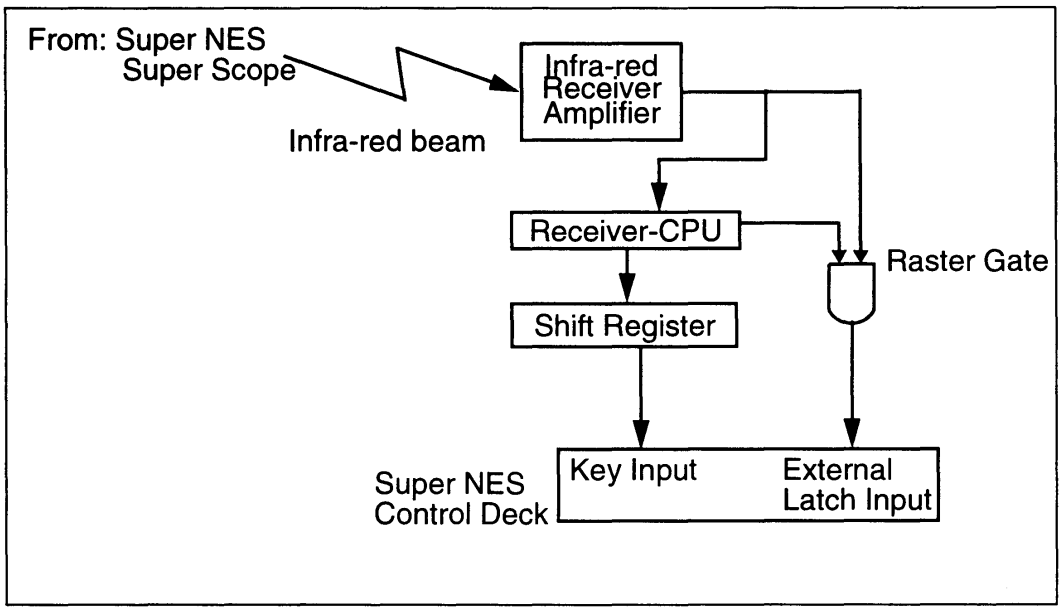


Figure 4-4-1 Receiver Block Diagram

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

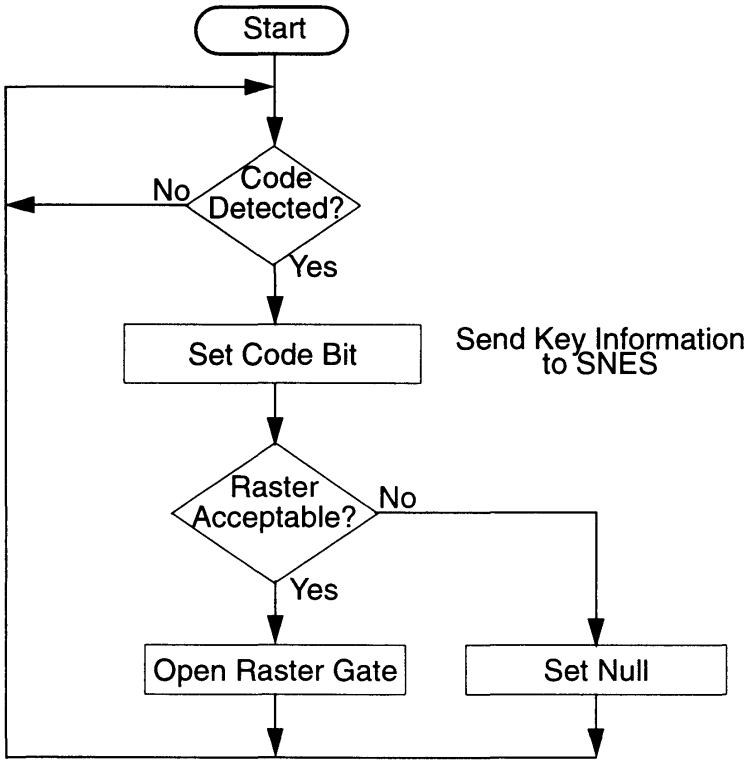


Figure 4-4-2 Operation Flow Diagram

In addition, a pulse check is performed during code detection for noise detection.

4.2 SUPER NES SUPER SCOPE RECEIVER INTERFACE

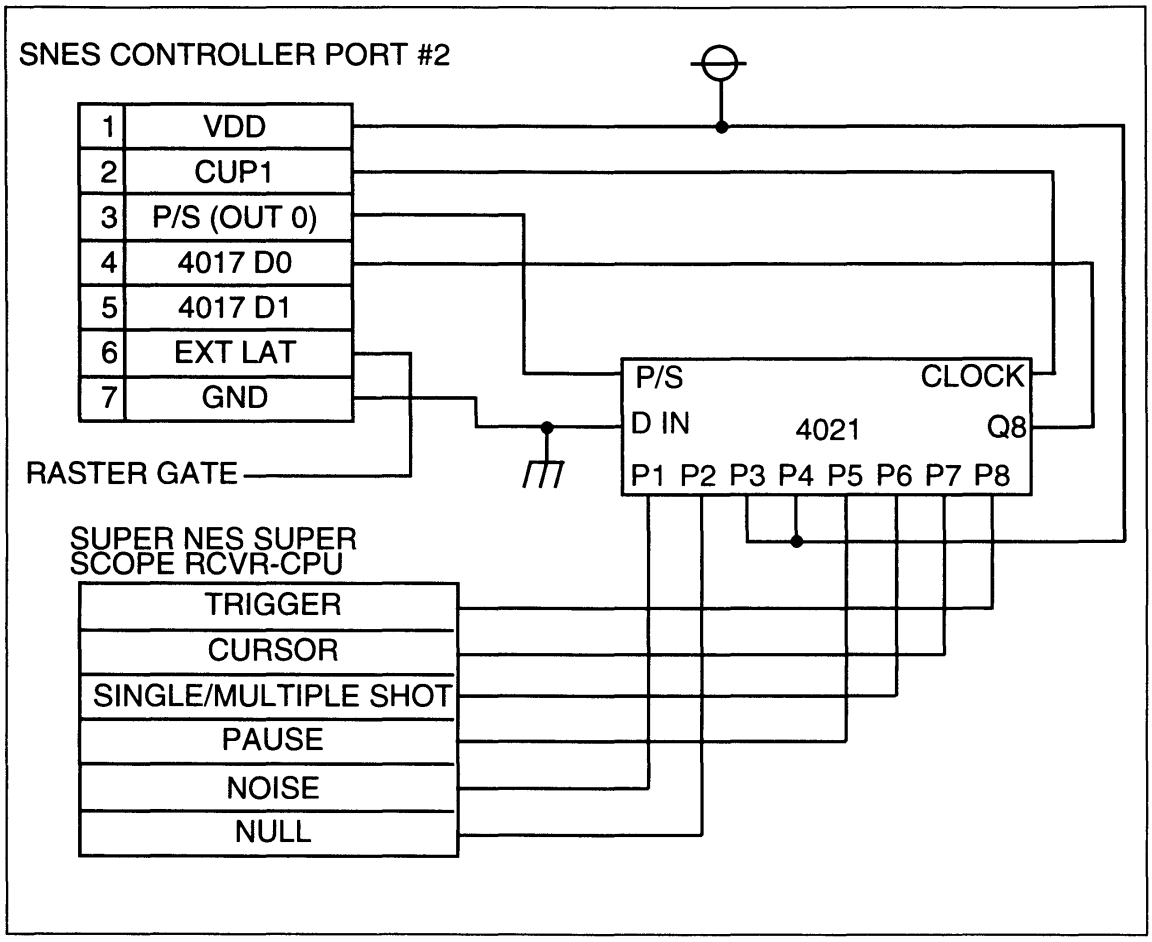


Figure 4-4-3 Receiver Interface Schematic

4.3 CODE PULSE DETECTION

4.3.1 ONE BIT CODE DETECTION

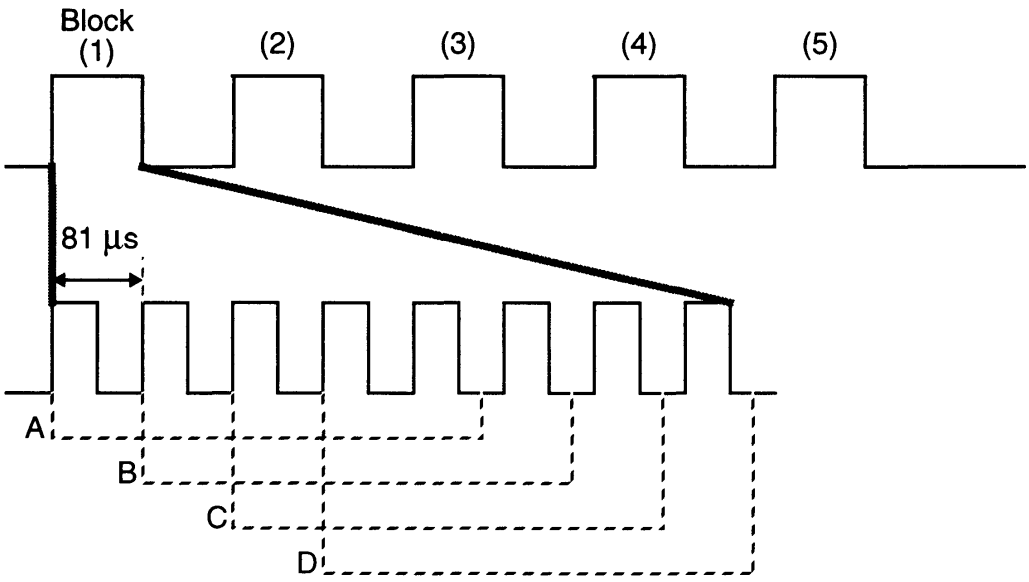


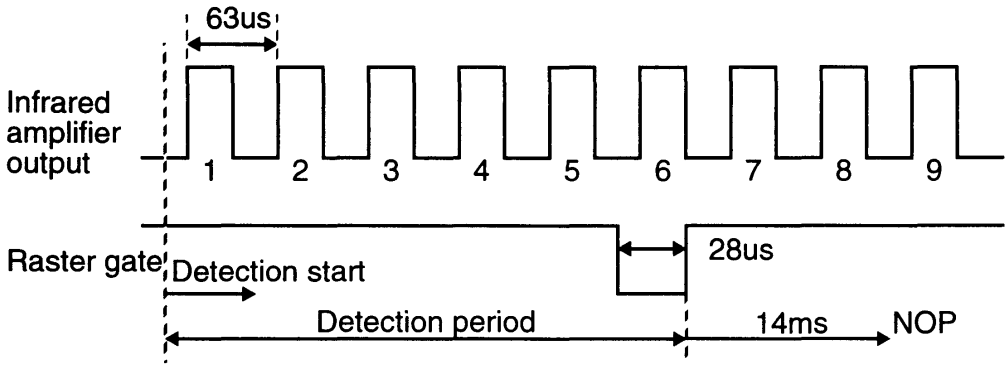
Figure 4-4-4 One Bit Code Detection

A block is good if five - 81 µsec 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 µsec after the rising edge of a pulse is detected.

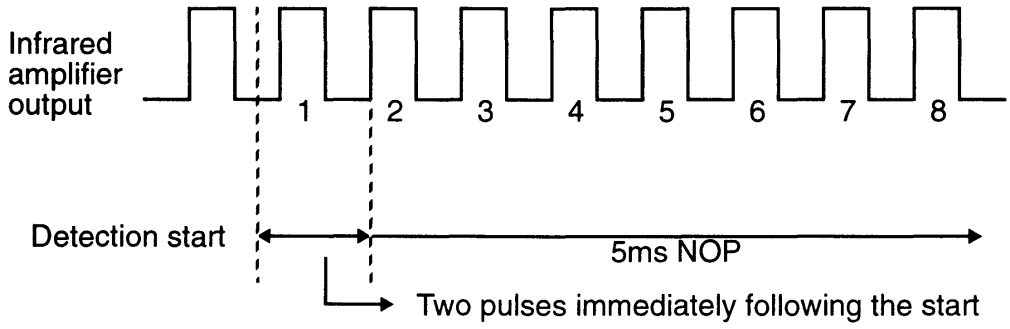
4.3.2 RASTER PULSE DETECTION

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

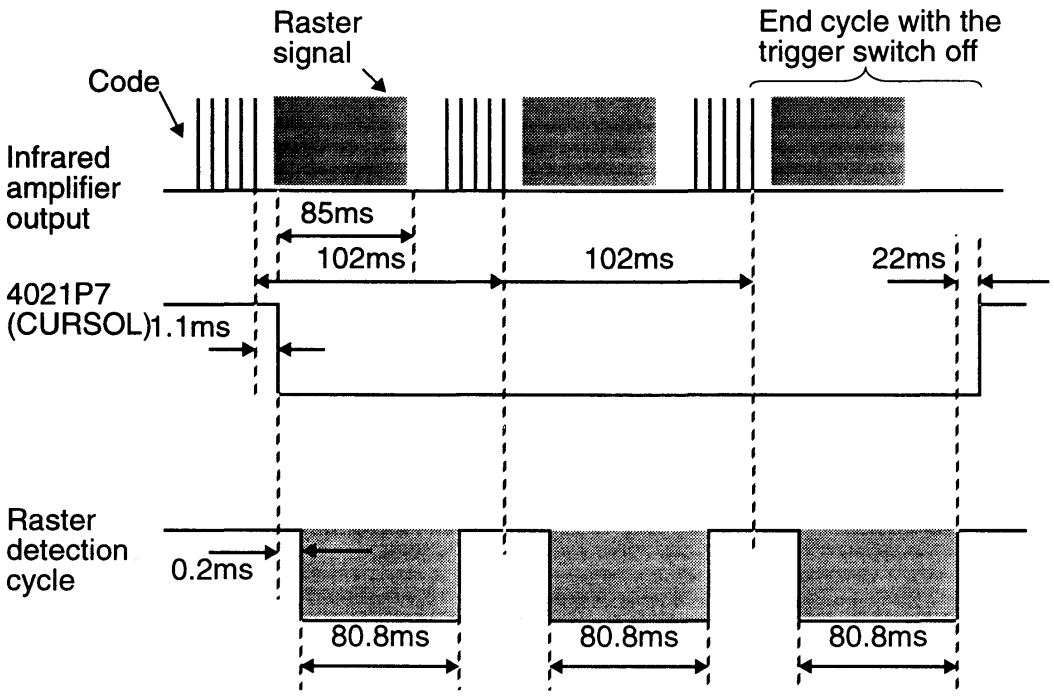


Figure 4-4-5 Cursor Mode Raster Detection Cycle

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)

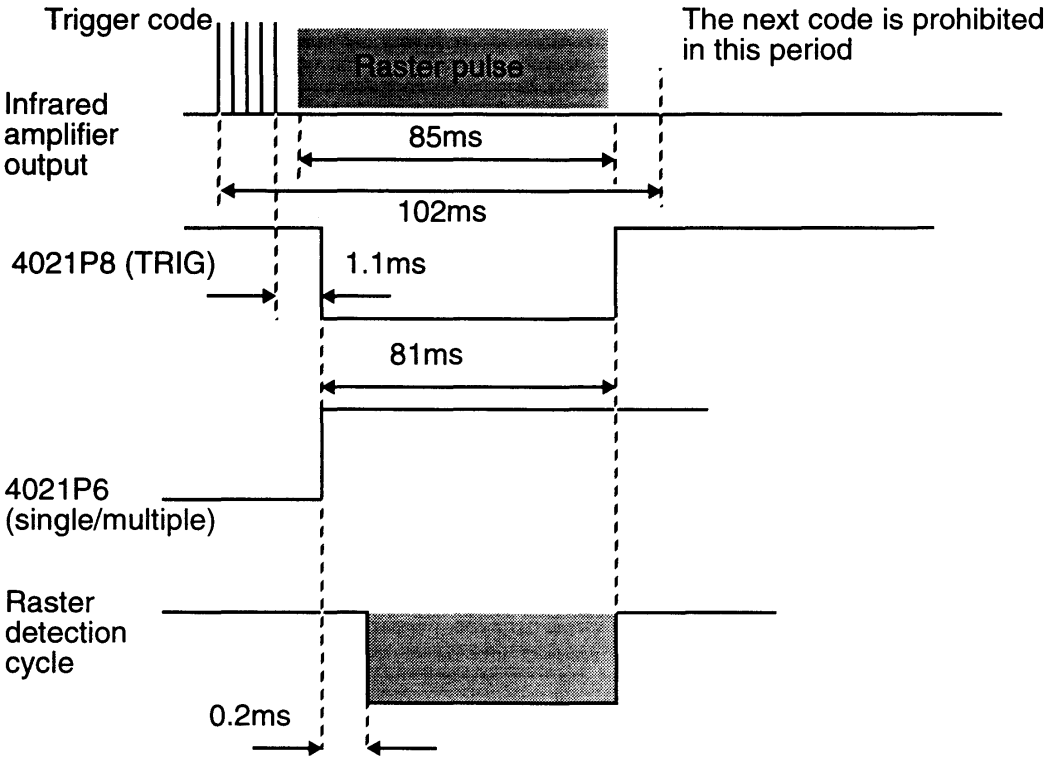


Figure 4-4-6 Trigger Mode, Single Shot

4.4.3 TRIGGER MODE (MULTIPLE SHOTS)

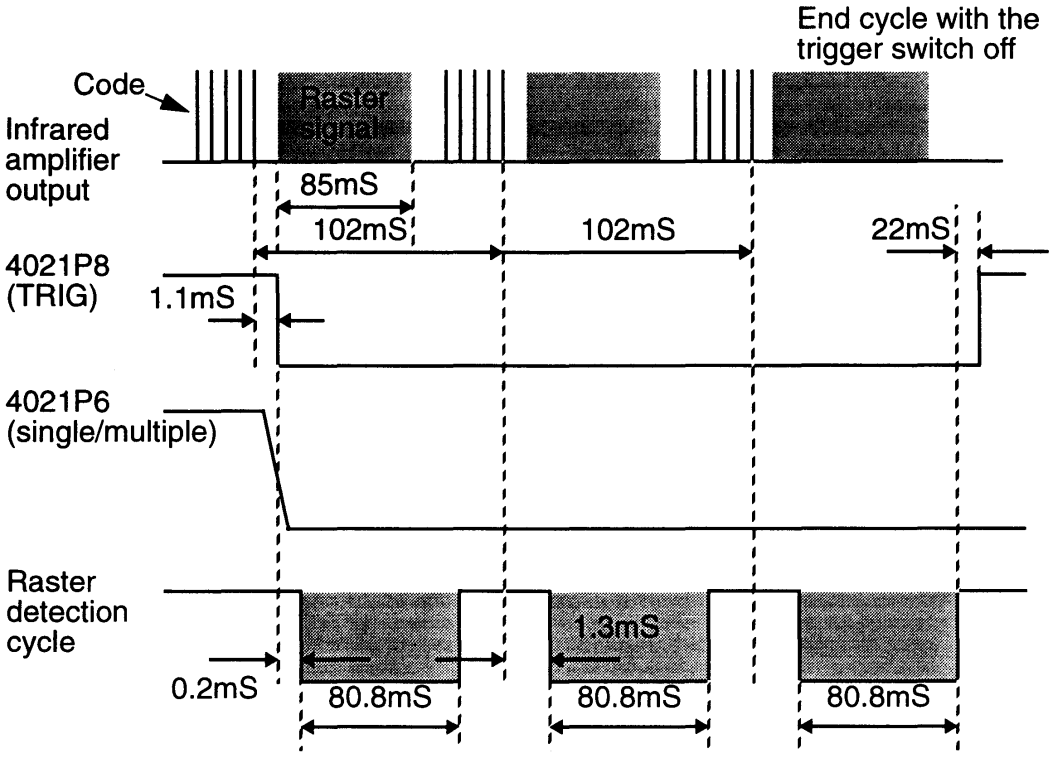


Figure 4-4-7 Trigger Mode, Multiple Shots

4.4.4 NOISE FLAG

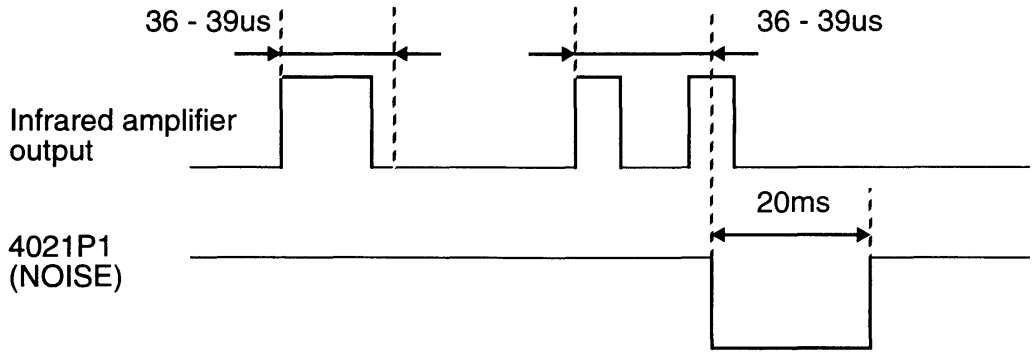


Figure 4-4-8 Noise Flag

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

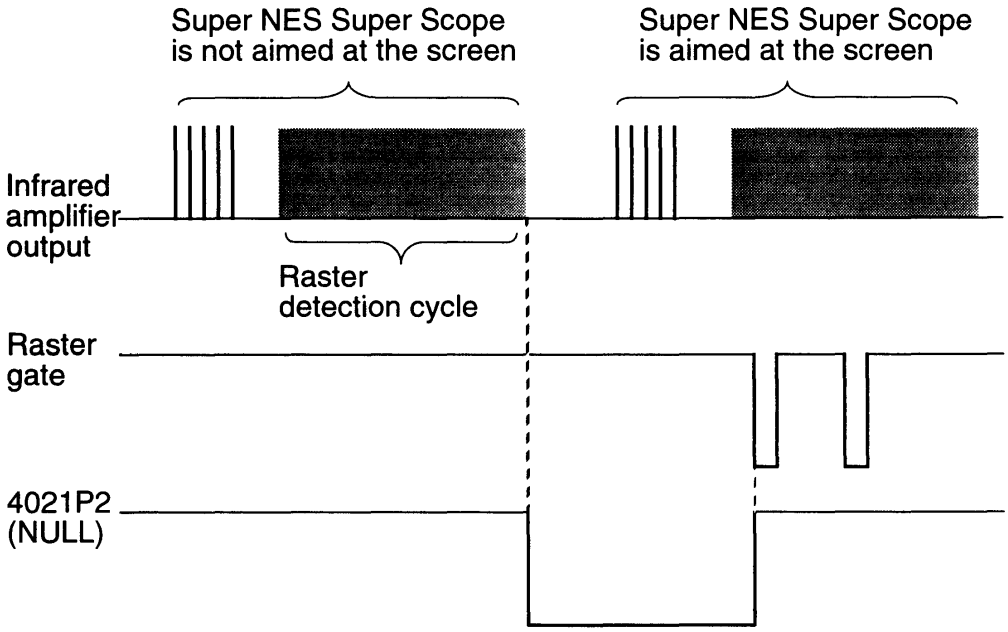


Figure 4-4-9 Null Bit

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

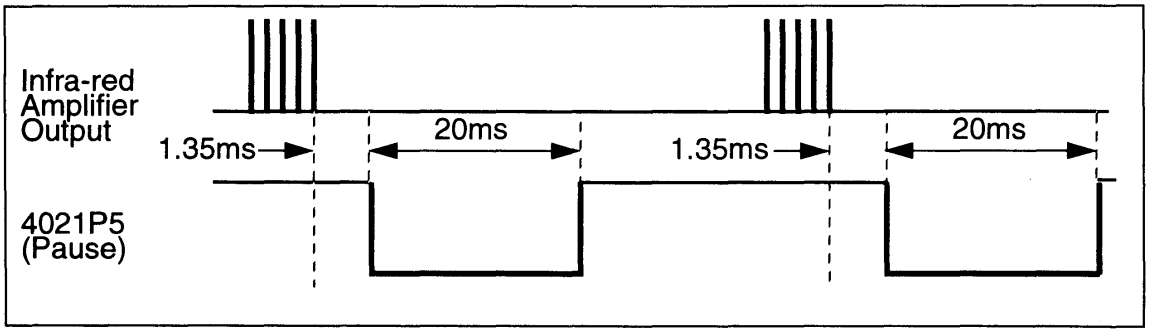


Figure 4-4-10 Pause Bit

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)

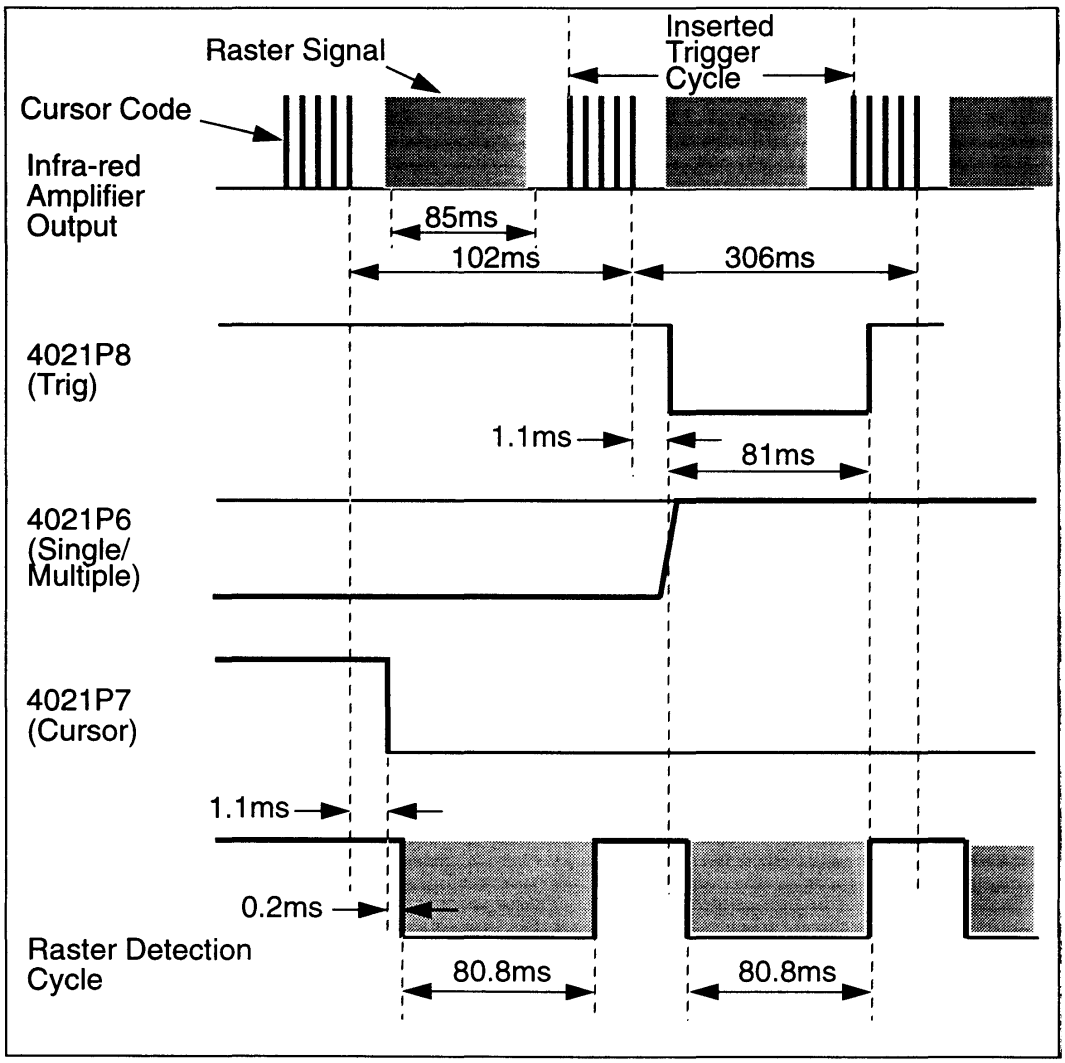


Figure 4-4-11 Trigger, Single Shot

4.4.7.2 TRIGGER (MULTIPLE SHOTS)

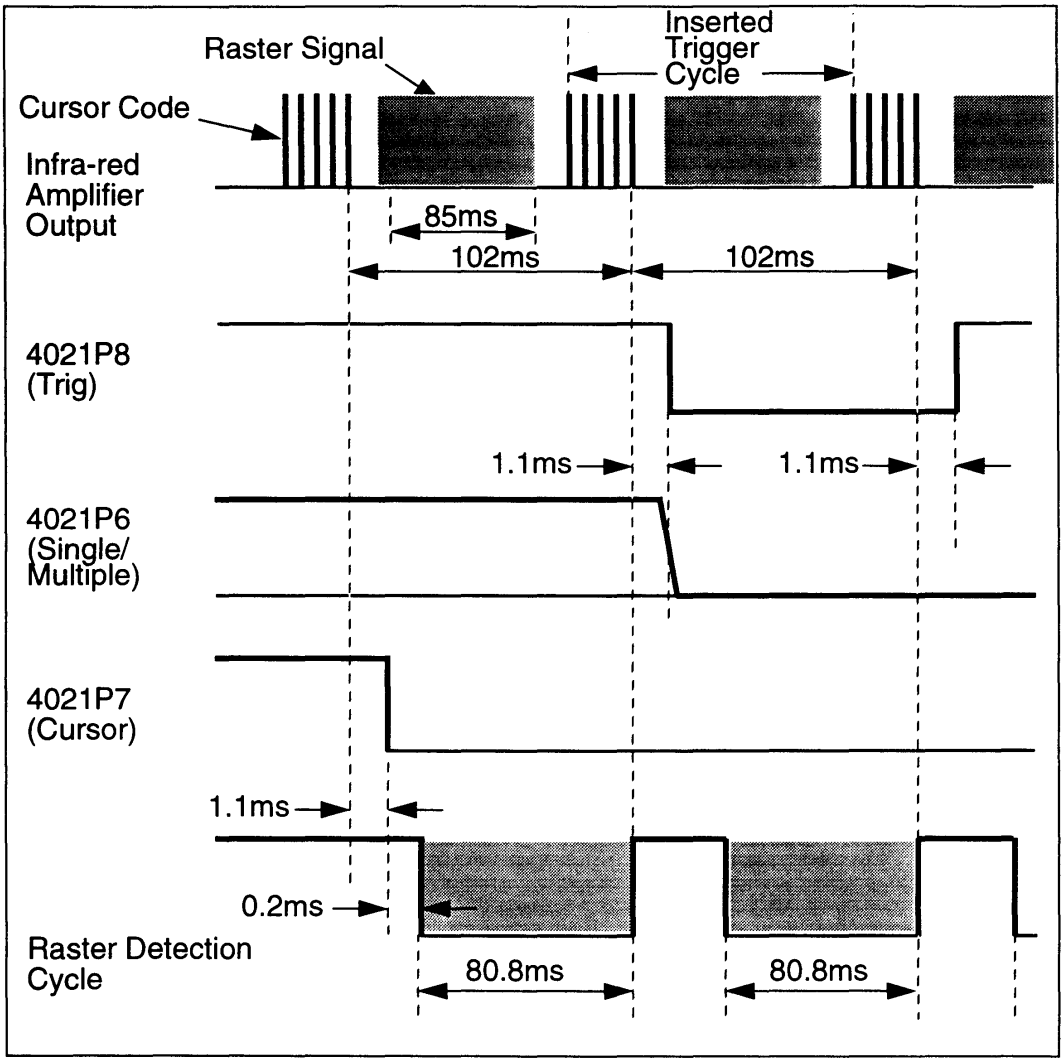


Figure 4-4-12 Trigger, Multiple Shots

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 florescence 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 µsec
Blue	250 µsec

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

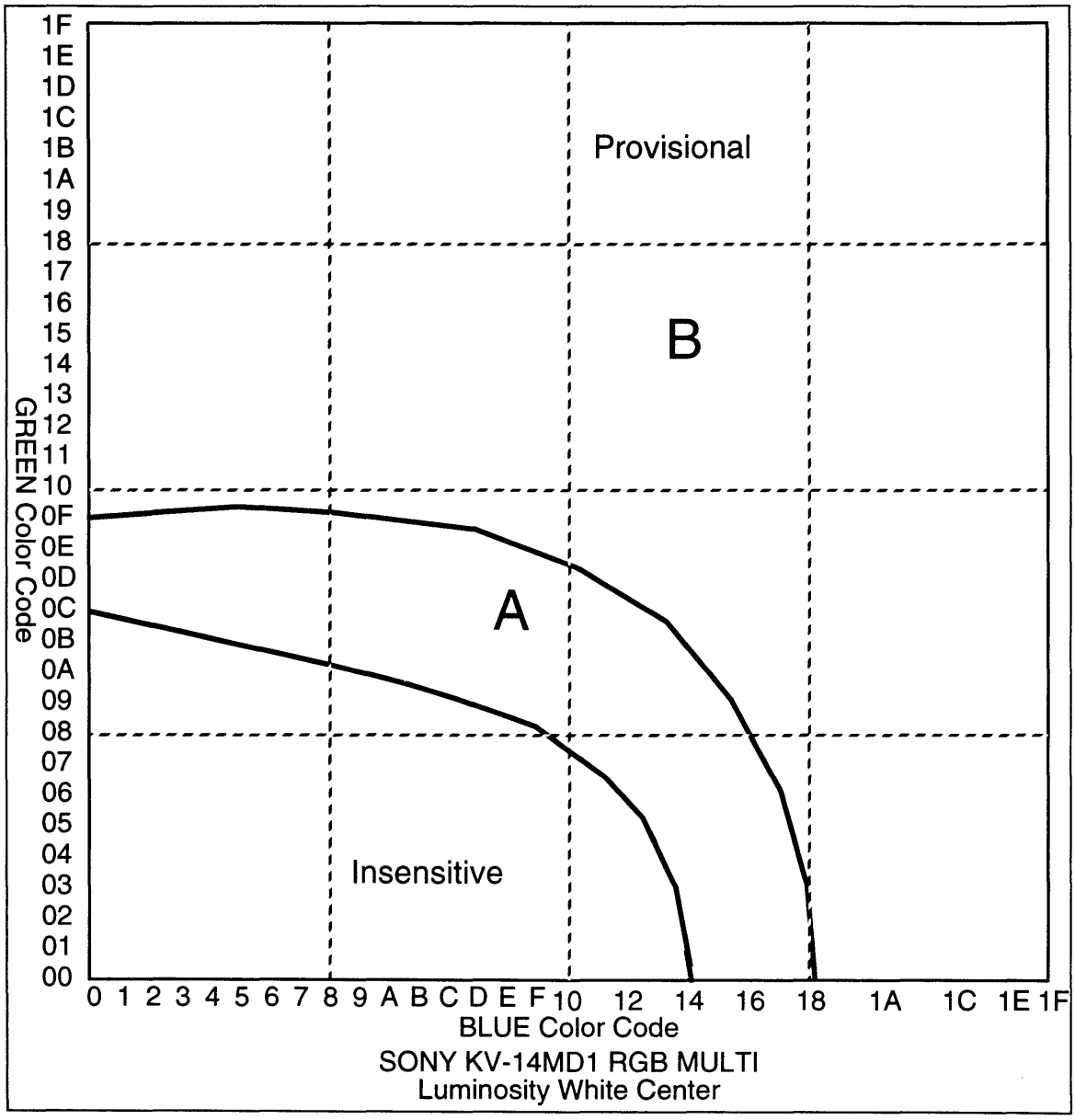


Figure 4-5-1 Optical Color Sensitivity Chart

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.

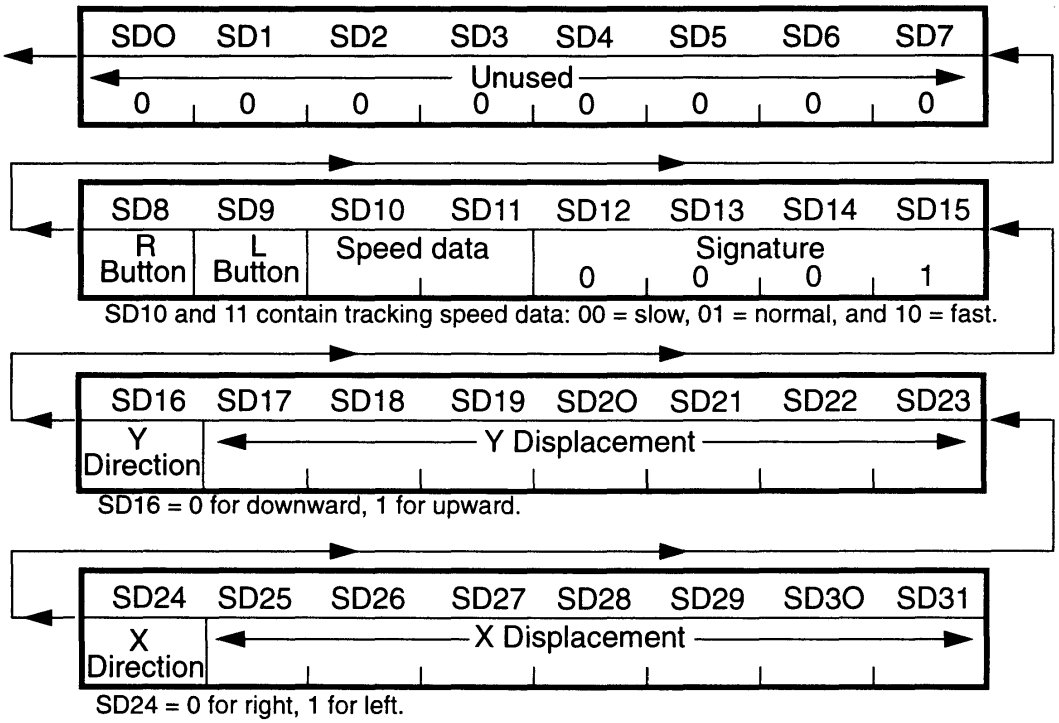


Figure 4-6-1 Valid Super NES Mouse Data String

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.

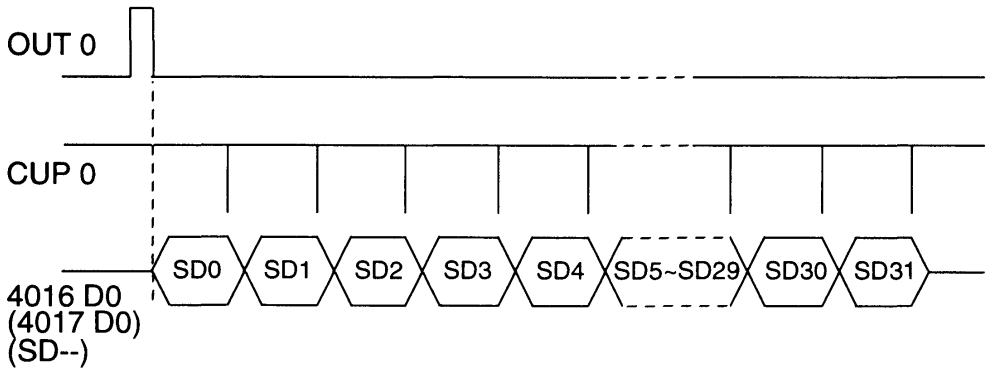


Figure 4-6-2 Serial Data Read Timing

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

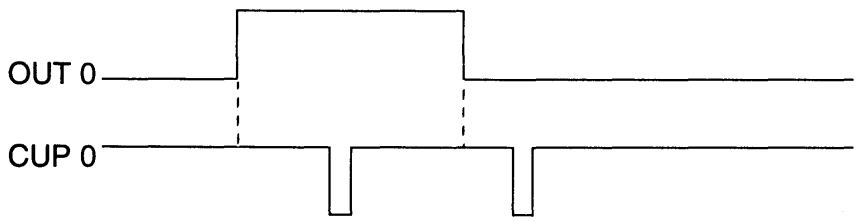
Set OUT0 to HI, and set CUP0 once to [LOW → HIGH] (read 4016H). Next, set OUT0 to LOW, and once again set CUP0 to [LOW → HIGH]. This changes the mouse tracking speed by one setting. The speed is changed by two settings if CUP0 is set LOW to HI twice while OUT0 is HI.

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.

SD10 Upper Bit	SD11 Lower Bit	Speed Mode
0	0	Slow
0	1	Normal
1	0	Fast

MSB ← → LSB

SD16	SD17	SD18	SD19	SD20	SD21	SD22	SD23	Direction of movement
0	D6	D5	D4	D3	D2	D1	D0	Down
1	D6	D5	D4	D3	D2	D1	D0	Up

Direction

Y Displacement

X Displacement

SD24	SD25	SD26	SD27	SD28	SD29	SD30	SD31	Direction of movement
0	D6	D5	D4	D3	D2	D1	D0	Right
1	D6	D5	D4	D3	D2	D1	D0	Left

D6~D0 change with the amount of mouse displacement. (Max. 3F)

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 V ± 10%

Current consumption: 50 mA (maximum)

6.5.2 OPERATIONAL AND ENDURANCE SPECIFICATIONS

Resolution: 50 counts/inch ± 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 V ± 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

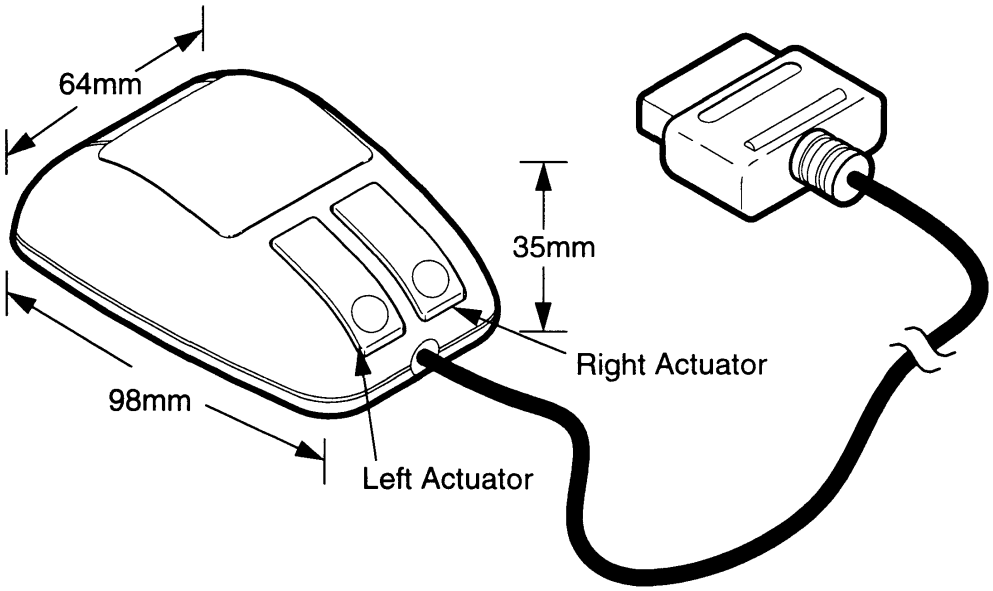


Figure 4-6-4 Super NES Mouse Dimensions

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 compatability 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.	mouse_con1 mouse connection info.	
D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	0 / not connected 1 / connected
?	?	
mouse_sp0 mouse speed information	mouse_sp1 mouse speed information	
D7 D6 D5 D4 D3 D2 D1 D0	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	mouse_sw1 mouse switch continuous	
D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	R: right button L: left button 0 / OFF 1 / ON
R L	R L	
mouse_swt0 mouse switch trigger	mouse_swt1 mouse switch trigger	
D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	R: right button L: left button 0 / OFF 1/ ON
R L	R L	
mouse_y0, Displacement in Y direction	mouse_y1, Displacement in Y direction	
D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	Dir: Direction bit 0 / down 1 / up
Dir. <--Displacement in Y direction-->	Dir. <--Displacement in Y direction-->	
mouse_x0, Displacement in X direction	mouse_x1, Displacement in X direction	
D7 D6 D5 D4 D3 D2 D1 D0	D7 D6 D5 D4 D3 D2 D1 D0	Dir: Direction bit 0 / right 1 / left
Dir. <--Displacement in X direction-->	Dir. <--Displacement in X direction-->	

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

```
        ldx   #$00          ; Connector 1
        lda   #$01          ; Switch to "normal" speed
        sta   mouse_sp_set0
        jsr   speed_change
```

Example 2

```
        ldx   #$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?

        ldx     #$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.

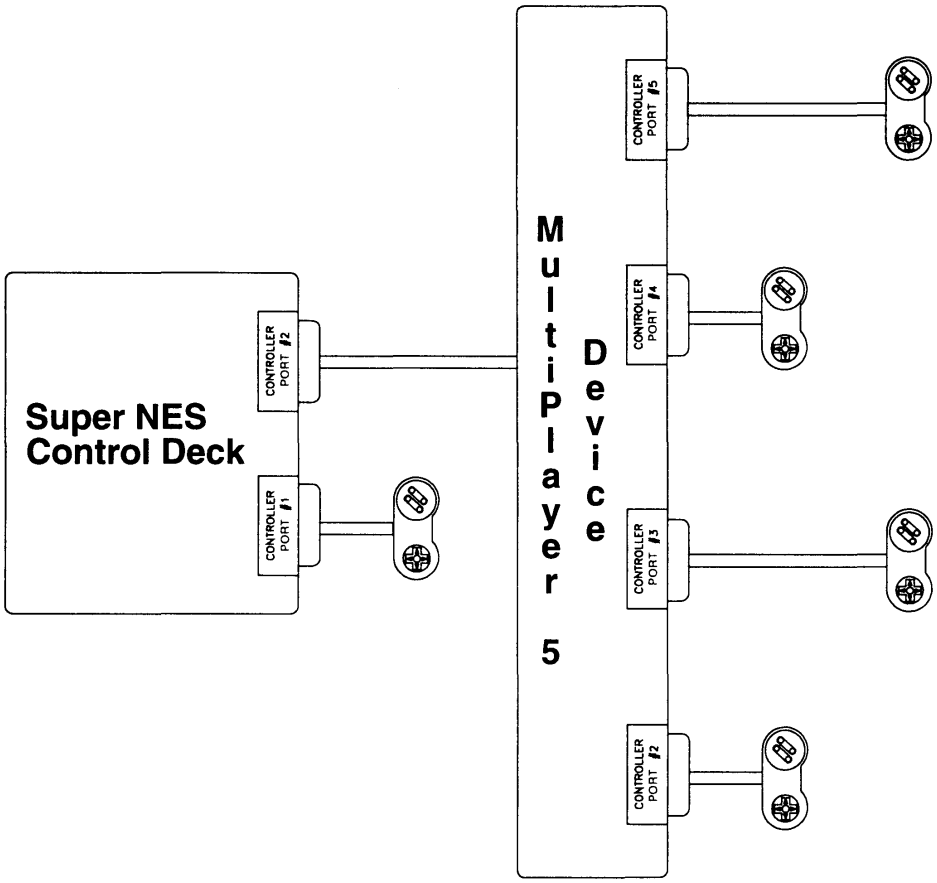


Figure 4-9-1 MultiPlayer 5 Device Hardware Connections

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			
			EXPANSION CONNECTORS				EXPANSION CONNECTORS		
		②	③	④	⑤	②	③	④	⑤
1	+5V	X	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							
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.

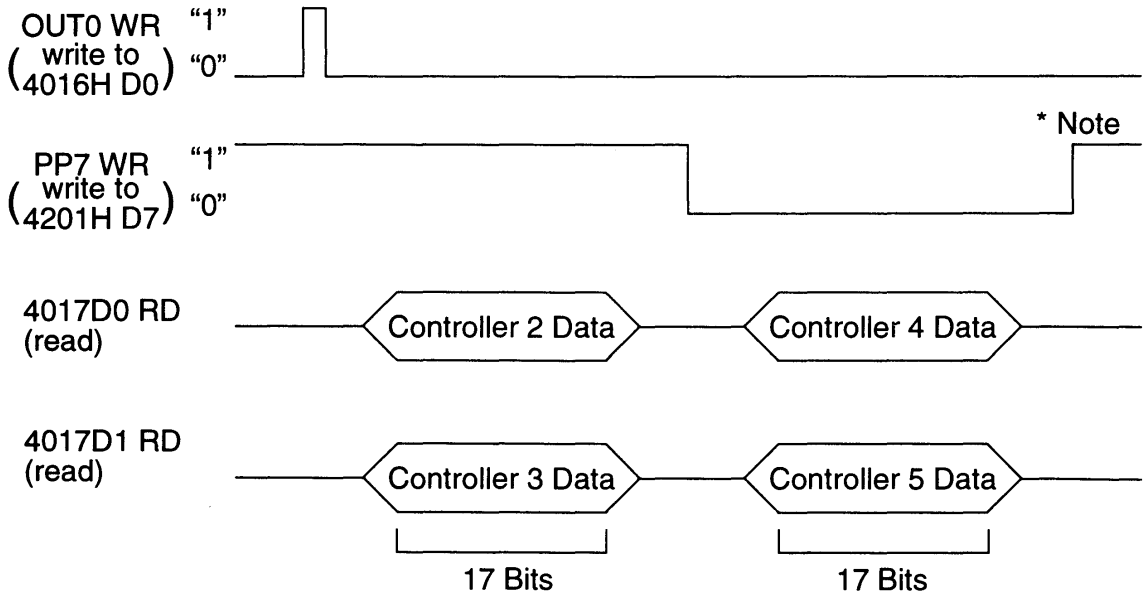


Figure 4-9-2 MultiPlayer 5 Read Timing Chart, 5P Mode

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 <4017H> 16th read <4017H> 17th read	undefined undefined undefined	0 0 1 (*)	0 0 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 <4017H> 33rd read <4017H> 34th read	undefined undefined undefined	0 0 1 (*)	0 0 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.

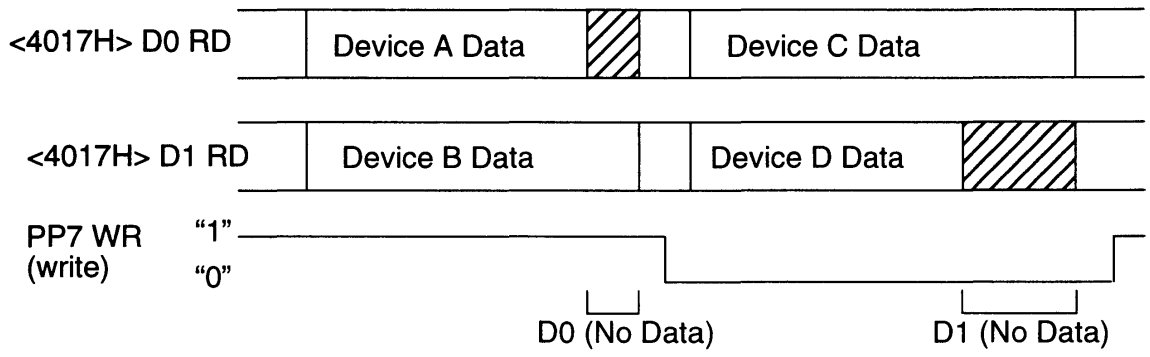
1*. Any device which uses <4016H> D1 or <4017H> D1 for its data read.
2*. Any device which uses <4201H> or <4213H>.
3. Any device with an electrical consumption of 17mA or more per unit.
4*. Any device which detects a CUP signal while OUT 0 is "1".
5. Any device which transmits data while OUT 0 is "1".
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

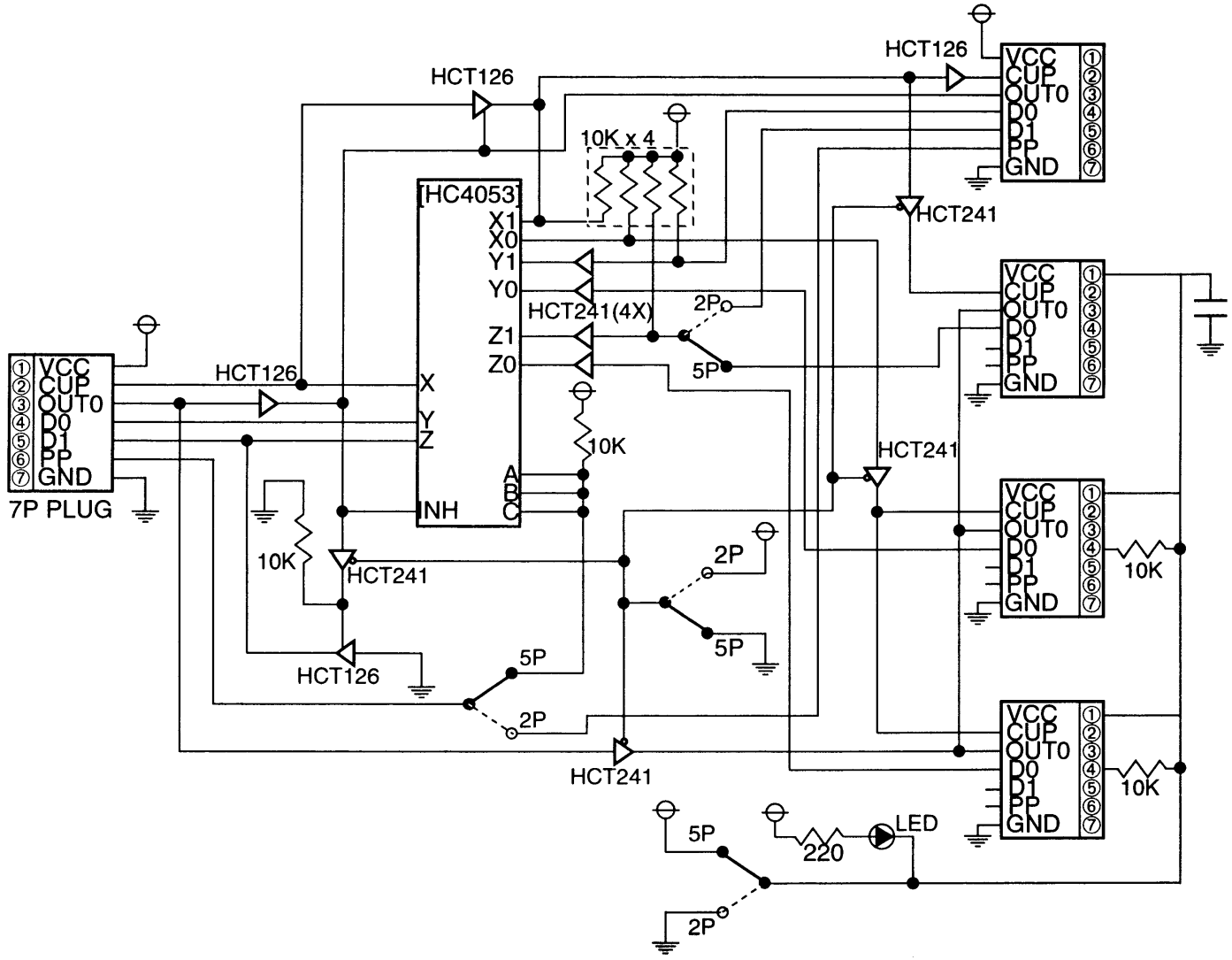
MultiPlayer 5 simply passes on the signature codes for devices connected to controller ports 2 ~ 5 and does not have a signature code of its own. However, the following procedure will verify that MultiPlayer 5 is connected. When performing this procedure, it does not matter whether or not a device is 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 <u>not</u> "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 to16 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.

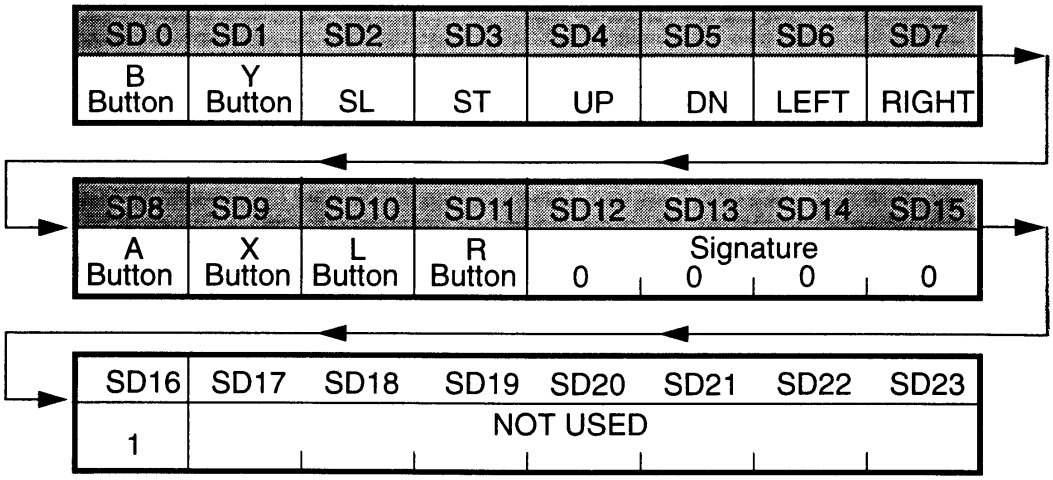


Figure 4-9-4 Valid Controller Data String

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.

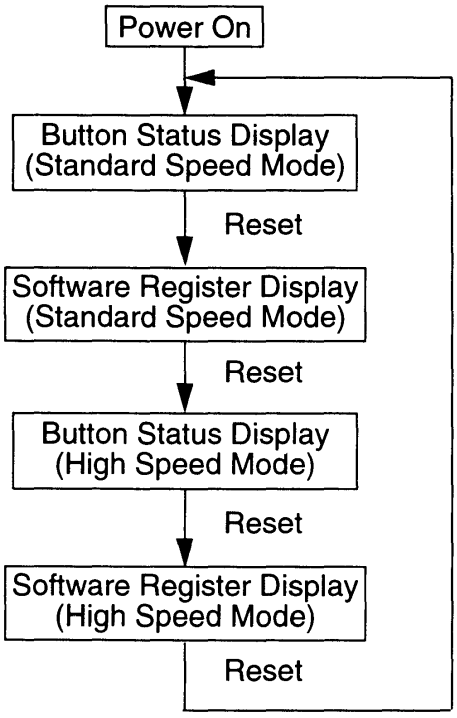


Figure 4-10-1 Sample Program Display Format

10.3 SUPPLIED BIOS EXECUTION

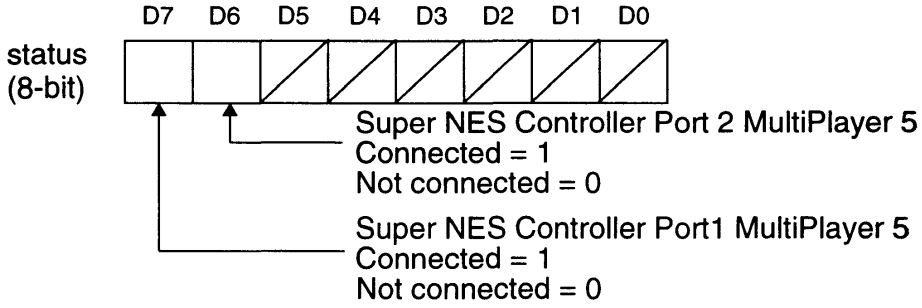
The supplied BIOS program assumes it is running in synchronization with the Super NES PPU's NMI interrupt. The program uses the Super NES CPU controller data automatic read function, so the automatic read function must be enabled when the BIOS is called (<4200H> D0=1).

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).

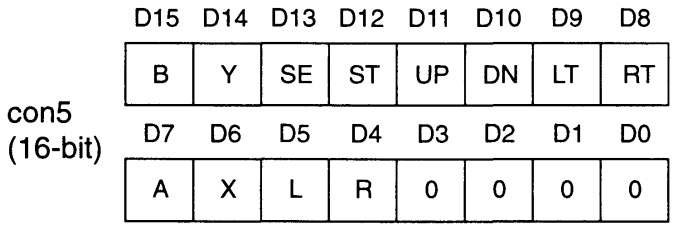
In this BIOS, the OUT0 signal is controlled by the Controller Automatic Read function. The user must ensure that the BIOS is called in the proper order. After the Super NES CPU Automatic Read period (215 µs from the start of NMI), call "MULTI5.ASM (X65)" followed by ""M_CHECK.ASM (X65)". The BIOS must be called in this order for proper operation.

10.4 SUPPLIED BIOS OUTPUT REGISTER

M_CHECK.X65



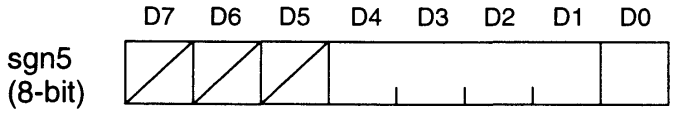
MULTI5.X65



Controller 5 Button Information

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 connecter 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

        STZ     c_ad1   ; output 0 to out0
        LDX     #08H
_c20
        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     .reg1l      ;read d1 of 4017h and store it to reg1l
        DEX
        BNE     _c20
;<determine if special device or MPA is connected?>
;
;<Check controller port1>
        LDA     .reg0h
        CMP     #0FFH       ;Is reg0h=$FF?
        BNE     _c30        ; YES->determine if MPA or special device
                            ; NO->branch, check connection on port2
        LDA     .reg0l
        CMP     #0FFH       ;Is reg0l=$FF?
        BEQ     __C30       ; YES->special device connected to port1, jmp
                            ; NO->MPA connected to port1, set status
        LDA     #80H
        STA     .status

;<Check controller port2>
_c30
        LDA     .reg1h
        CMP     #0FFH       ;Is reg1h=$FF?
        BNE     _c40        ; YES->determine if MPA or special device
                            ; NO->branch and return from routine
        LDA     .reg1l
        CMP     #0FFH       ;Is reg1l=$FF?
        BEQ     _c40        ; YES->special device connected to port2, rts
                            ; NO->MPA connected to port2, set status
        LDA     #40H
        ORA     .status
        STA     .status
_c40
        PLP
        RTS


;==============================================================
;       MultiPlayer connection check routine version 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 CONNECT CHECK VER x.xx'
;
;*****************************************************************************************
;
        DB      'NINTENDO SHVC MULTI5 CONNECT CHECK Ver1.00'
```

```
;*****************************************************************************************
;
;           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

Part #	Description	Remarks
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)

N

O

P

R

S

T

V

W

INDEX (Book II)

COMMANDS/INSTRUCTIONS

Index (Continued)

COMMANDS/INSTRUCTIONS (Continued)

SUBJECT - Alphebetical Listing

A

B

Index (Continued)

Index (Continued)

Index (Continued)