

# Final Project Specification

ECE4514 CRN:13112

Jacob Abel

April 24, 2019

# Project Objectives

This project intends to provide a deliverable of a Synthesisable SystemVerilog GameBoy DMG emulator. With consideration for the time constraints, this emulator will be able to run the game "Pokemon Red" with some limitations. The serial link cable and infra-red communication subsystems will not be implemented so there will be no support for any multi-device interactions. Additionally, the device resets will be divided into soft and hard resets. Soft resets will not reset the save Catridge/MBC RAM while hard resets will. The controller will be emulated by a PS/2 keyboard and the 160x144 LCD display will be up-scaled to 800x720 and rendered on a 1024x768 VGA display running at 60Hz. The hardware and controls will map as shown below.

<b>Controller:</b>	Up	Left	Down	Right	A	B	Start	Select
<b>Keyboard:</b>	W	A	S	D	Space	Escape	Tab	Tilde

Figure 1: Controller→Keyboard Mappings

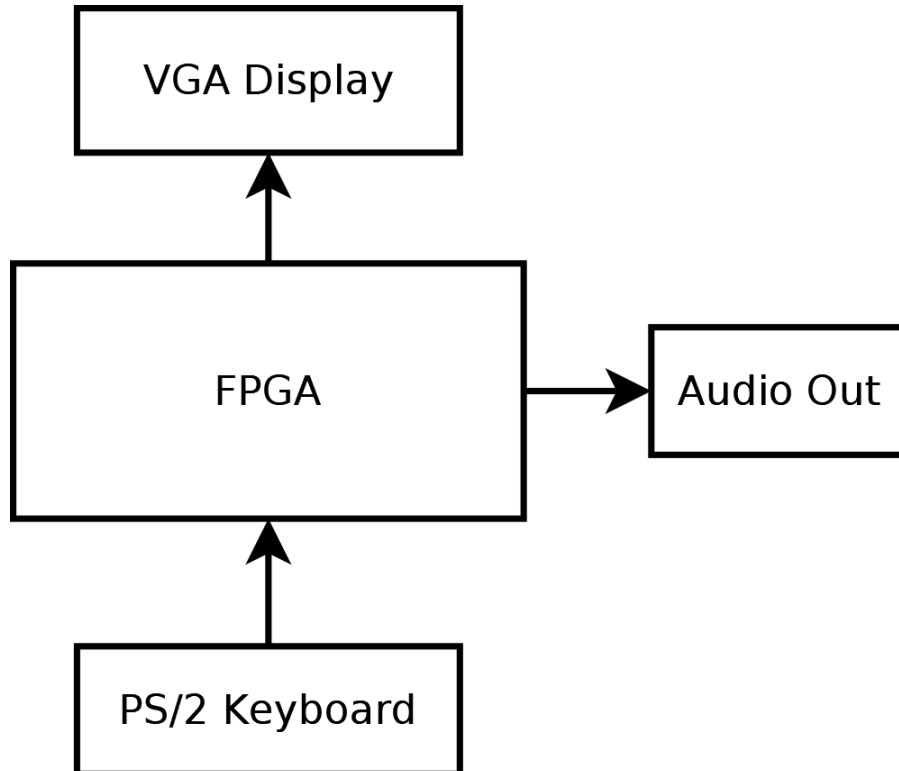


Figure 2: Physical Hardware Layout

# Design Description

The final design will replicate the operation of the GameBoy DMG handheld console for use with the game "Pokemon Red" with reasonable accuracy and within the scope outlined in [Project Objectives](#). The design description below offers an initial starting point however the design is free to be restructured as necessary provided that all required observable behaviour is replicated and a final updated project specification is offered detailing the up-to-date structuring of the deliverable.

## PS/2 Module

The PS/2 submodule is an adaptor for the PS/2 Keyboard and will map keyboard keys to the game boy controller button outputs as detailed in [Figure 1](#). The adaptor will use a PS/2 IP core for handling the protocol.

## DAC Module

The DAC submodule mostly reuses the audio modules from prior assignments with modifications as necessary to integrate with and upscale the LCD display output.

## VGA Module

The VGA submodule reuses the XGA modules from prior assignments with modifications as necessary to facilitate the vertical and horizontal interrupts on the primary DMG module.

## RAM & VRAM Modules

These modules instantiate the Work RAM and Video RAM for the DMG module. These modules instantiate 64 kilobits/8 kilobytes of Work RAM and VRAM. The VRAM(in conjunction with the OAM RAM discussed later) stores all of the visual sprites and textures as well as coordinate maps necessary for the LCD Controller to generate the final rendered display. The Work RAM on the other hand is a general RAM for the instructions executing on the CPU core.

## Memory Bank Controller Module

This module implements the MBC3 controller with 1 megabyte of ROM (for the game code) and 32 kilobytes of RAM as well. The RAM and ROM can be addressed by the CPU and the data stored in RAM persists through soft reset. This is because this particular

MBC3 module that is being emulated has an onboard battery to maintain the RAM contents. As such, this is where save data is persisted.

## Gameboy DMG Module

This is the primary submodule of the project. It will replicate the functionality of the DMG as specified by the accompanying GameBoy DMG documentation in [Annex A](#) and [Annex C](#). This module will contain the submodules for each module present in [Figure 4](#). Because of the broad yet relatively shallow nature of this module, only the major functionality of each primary subsystem will be outlined beyond what is already provided in [Annex A](#).

### CPU Core

The CPU core is largely similar to the Intel 8080 and Zilog Z80 CPUs however it has some major differences as outlined in [CPU Comparision with Z80](#). For the most part it is a Zilog Z80 running at 4MHz with all instruction execution times being rounded up to 4 clock cycles. An additional element of note is that there is a small internal RAM for periods where the primary data bus is in use. The rest of the CPU operations are detailed under [Annex A](#) and [Chapter 1.2](#) and [Chapter 4](#) of Annex C.

### Memory Subsystem

The memory map subsystem is effectively an IOMMU or IO memory management unit. This system controls which device is accessed or written to when a specific memory address is accessed. [The Memory Map section of Annex A](#) details which subsystem is routed to by each address range. Primarily, the important addressable ranges are the MBC ROM, MBC RAM, Work RAM, and IO Registers. The full details for each of the registers is available in [Annex A](#) and [Annex C](#). This module will primarily be composed of a large range select chain and routing outputs.

### Sound Subsystem

The Audio subsystem is divided into six parts: the four audio generators, a small RAM, and a signal mixer. Audio generators one and two produce pulse width tone patterns, generator three is a static noise generator. It creates either clean or dirty static. Audio generator four produces a looping waveform based on the contents of the small RAM. The RAM obviously holds audio data. Finally, the signal mixer combines all four signals and outputs it to the DAC subsystem. More details are available in [Audio Subsystem Hardware](#), [Audio Subsystem Control](#), and [Chapter 3 of Annex C](#).

## Timer Subsystem

This subsystem is divided into the divider and timer functionalities. The divider is a clock divider circuit that controls the rate at which the timer is incremented. The divider counter increments at 16384Hz and can be used to drive the timer at 4096Hz, 16384Hz, 65536Hz, and 262144 Hz. The timer is 8 bit timer that when it overflows is reset with a specified value and can trigger an interrupt. Further details on timer operation can be found under [Timer Behaviour in Annex A](#) or [Sections 2.4.2 and 2.4.3 of Annex C](#)

## Interrupt Subsystem

The interrupt subsystem controls all hardware interrupts in the system. On a signal from one of the connected subsystems it suspends execution of the CPU, pushes the current program counter onto the stack, and sets the current program counter to the interrupt handler's jump address. The interrupt subsystem has 4 internal interrupts and 1 external interrupt. The internal interrupts are on LCD Vertical Blanking, on a configurable status change from the display controller, on timer overflow, or on completion of serial data transfer(out of the scope of this project). The external interrupt is triggered by the pressing of any controller buttons (negative edge on inputs). All interrupts can be enabled or disabled and function addresses can be assigned to the interrupt handler jump addresses. Further details can be found in [the Interrupt Section of Annex A](#).

## Graphics Subsystem

The graphics subsystem is divided into three parts: the display controller, the VRAM interface, the OAM RAM, and the DMA controller. The display controller requests all the sprites and textures from the VRAM and OAM RAM and then composites them into a final output frame. The VRAM interface connects the display controller to the VRAM and maps requests by data type to actual addresses. The OAM RAM contains the non-background object drawn on the screen. These objects each contain the coordinates and texture map ids necessary to render them. The DMA (Direct Memory Access) controller transfers data to the OAM RAM from the VRAM. During DMA operations, the CPU operates from its internal RAM to prevent bus conflicts. All relevant information can be found in [the Video Subsystem section of Annex A](#) and [Chapter 2 of Annex C](#).

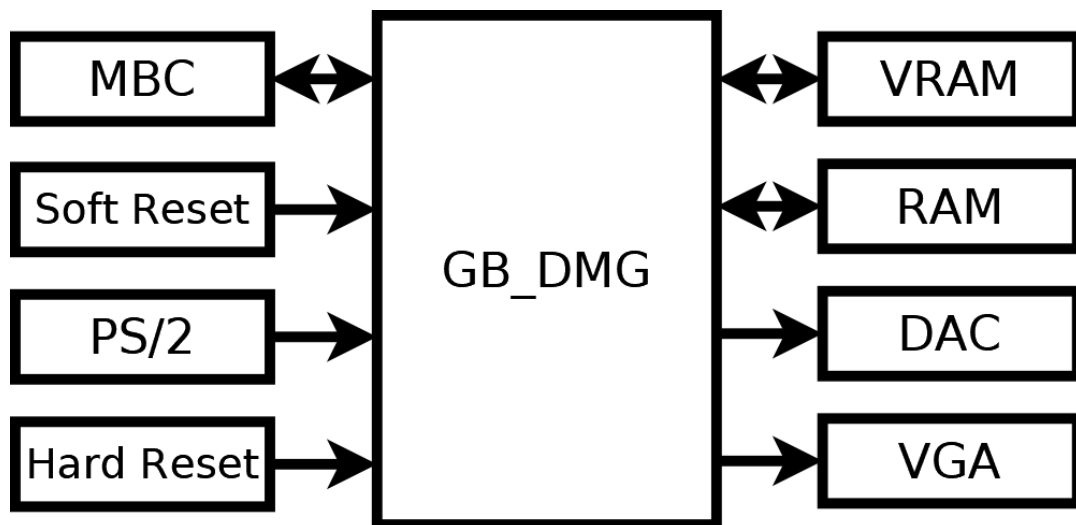


Figure 3: GameBoy Emulator Layout

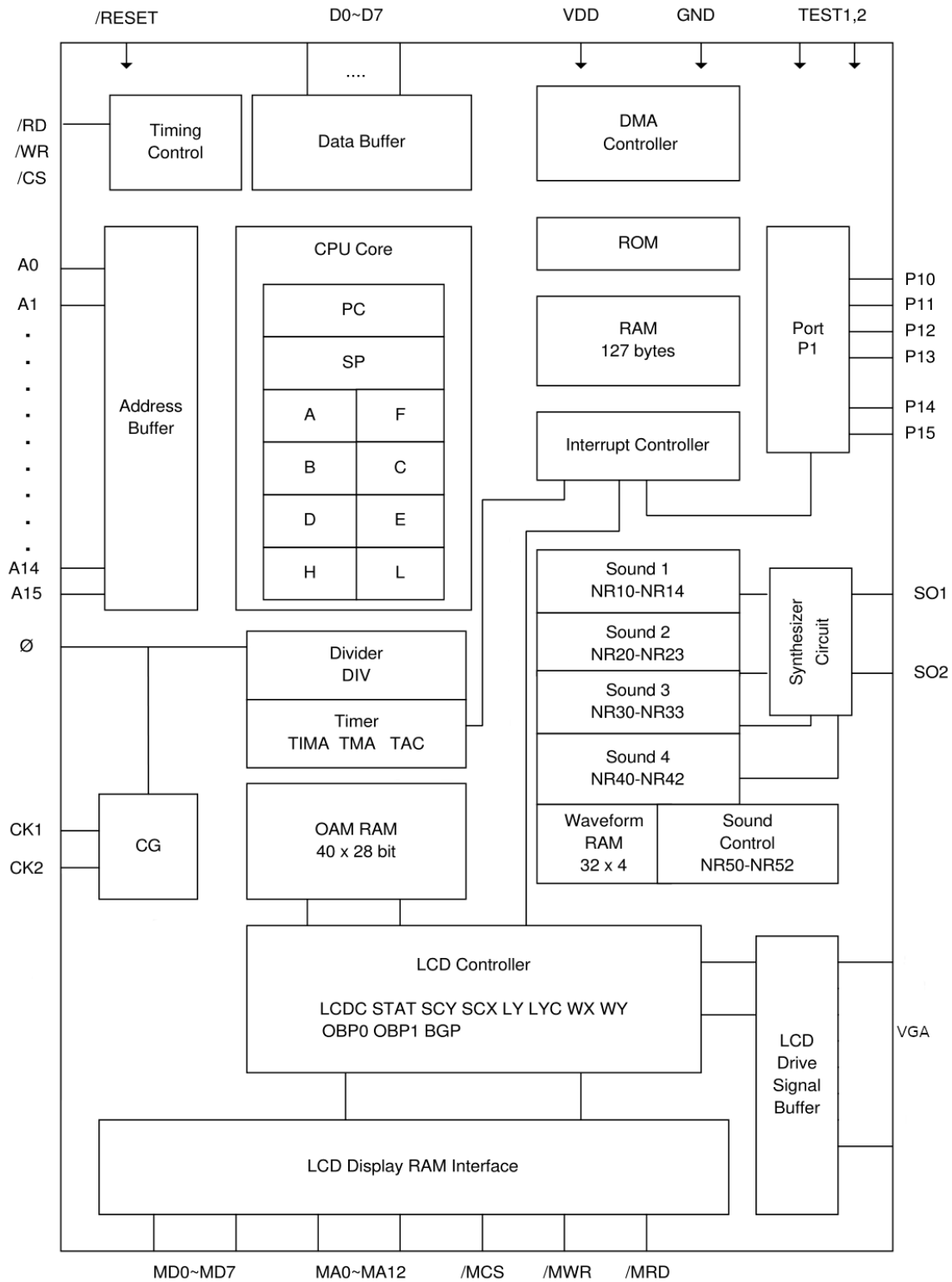


Figure 4: GameBoy DMG Layout

# Development Plan

The development plan will primarily be divided into two phases: development and testing. The development period has already begun and completion is targeted for the end of May 2nd. The testing period will fill the remaining time.

## Development Period

The development period will focus primarily on translating the specification to SystemVerilog then getting it to a synthesisable and mostly functional state. Each new module under development will have assertions specifying the intended operation embedded and testbenches will be developed using constraint based random verification to validate the modules both independently and together. Each module (beyond basic wrapper and pre-existing modules) will have some extent of testbench infrastructure.

## Testing Period

The testing period will focus on physical testing and verifying that the game in question (Pokemon Red) runs properly on the device. This period will focus on mental model-checking, on-chip testing, and general TLC. It is expected that given the constraints of the project that the game should run respectably on the device at the end of this period.

## Documentation

This project will use the documentation provided in Annexes A, B, and C. Additionally the WM8731 Audio DAC datasheet, Phillips I2C specification, and DE1-SOC documentation will be provided. The datasheet for the PS/2 IP Core will also be provided once it is selected.

As for deliverable documentation, an updated Design Description will be provided at the end of the project illustrating the final design.



# Annex A: Pan Docs GameBoy Documentation

## **CPU Comparision with Z80:**

[http://gbdev.gg8.se/wiki/articles/CPU\\_Comparision\\_with\\_Z80](http://gbdev.gg8.se/wiki/articles/CPU_Comparision_with_Z80)

## **CPU Registers and Flags:**

[http://gbdev.gg8.se/wiki/articles/CPU\\_Registers\\_and\\_Flags](http://gbdev.gg8.se/wiki/articles/CPU_Registers_and_Flags)

## **CPU Instruction Set:**

[http://gbdev.gg8.se/wiki/articles/CPU\\_Instruction\\_Set](http://gbdev.gg8.se/wiki/articles/CPU_Instruction_Set)

## **Memory Map:**

[http://gbdev.gg8.se/wiki/articles/Memory\\_Map](http://gbdev.gg8.se/wiki/articles/Memory_Map)

## **Video Subsystem:**

[http://gbdev.gg8.se/wiki/articles/Video\\_Display](http://gbdev.gg8.se/wiki/articles/Video_Display)

## **Audio Subsystem Control:**

[http://gbdev.gg8.se/wiki/articles/Sound\\_Controller](http://gbdev.gg8.se/wiki/articles/Sound_Controller)

## **Audio Subsystem Hardware:**

[http://gbdev.gg8.se/wiki/articles/Gameboy\\_sound\\_hardware](http://gbdev.gg8.se/wiki/articles/Gameboy_sound_hardware)

## **Interrupts:**

<http://gbdev.gg8.se/wiki/articles/Interrupts>

## **Timer Behaviour:**

[http://gbdev.gg8.se/wiki/articles/Timer\\_Obscure\\_Behaviour](http://gbdev.gg8.se/wiki/articles/Timer_Obscure_Behaviour)

## **Power Up Sequence:**

[http://gbdev.gg8.se/wiki/articles/Power\\_Up\\_Sequence](http://gbdev.gg8.se/wiki/articles/Power_Up_Sequence)

## **Summary of Subsystem Operations:**

<https://copetti.org/projects/consoles/game-boy/>

# Annex B: VESA Timing Guidelines

## VESA MONITOR TIMING STANDARD

Adopted: 9/10/91 (VESA #901101A)  
 Resolution: 1024 x 768 at 60 Hz (non-interlaced)  
 EDID ID: DMT ID: 10h; STD 2 Byte Code: (61, 40)h; CVT 3 Byte Code: n/a  
 BIOS Modes: 104h, 105h, 116h, 117h, & 118h (4, 8, 15, 16, & 24 bpp)  
 Method: **\*\*\* NOT CVT COMPLIANT \*\*\***

### Detailed Timing Parameters

Timing Name	=	<b>1024 x 768 @ 60Hz;</b>		
Hor Pixels	=	<b>1024;</b>	// Pixels	
Ver Pixels	=	<b>768;</b>	// Lines	
Hor Frequency	=	48.363;	// kHz	= 20.7 usec / line
Ver Frequency	=	60.004;	// Hz	= 16.7 msec / frame
Pixel Clock	=	<b>65.000;</b>	// MHz	= 15.4 nsec ± 0.5%
Character Width	=	<b>8;</b>	// Pixels	= 123.1 nsec
Scan Type	=	<b>NONINTERLACED;</b>	// H Phase	= 5.1 %
Hor Sync Polarity	=	<b>NEGATIVE;</b>	// HBlank	= 23.8% of HTotal
Ver Sync Polarity	=	<b>NEGATIVE;</b>	// VBlank	= 4.7% of VTotal
Hor Total Time	=	20.677;	// (usec)	= 168 chars = 1344 Pixels
Hor Addr Time	=	15.754;	// (usec)	= 128 chars = 1024 Pixels
Hor Blank Start	=	15.754;	// (usec)	= 128 chars = 1024 Pixels
Hor Blank Time	=	4.923;	// (usec)	= 40 chars = 320 Pixels
Hor Sync Start	=	16.123;	// (usec)	= 131 chars = 1048 Pixels
// H Right Border	=	0.000;	// (usec)	= 0 chars = 0 Pixels
// H Front Porch	=	0.369;	// (usec)	= 3 chars = 24 Pixels
Hor Sync Time	=	2.092;	// (usec)	= 17 chars = 136 Pixels
// H Back Porch	=	2.462;	// (usec)	= 20 chars = 160 Pixels
// H Left Border	=	0.000;	// (usec)	= 0 chars = 0 Pixels
Ver Total Time	=	16.666;	// (msec)	= 806 lines HT – (1.06xHA)
Ver Addr Time	=	15.880;	// (msec)	= 768 lines = 3.98
Ver Blank Start	=	15.880;	// (msec)	= 768 lines
Ver Blank Time	=	0.786;	// (msec)	= 38 lines
Ver Sync Start	=	15.942;	// (msec)	= 771 lines
// V Bottom Border	=	0.000;	// (msec)	= 0 lines
// V Front Porch	=	0.062;	// (msec)	= 3 lines
Ver Sync Time	=	0.124;	// (msec)	= 6 lines
// V Back Porch	=	0.600;	// (msec)	= 29 lines
// V Top Border	=	0.000;	// (msec)	= 0 lines

**Definition of Terms:** Refer to section 3.3.

# **Annex C:**

## **Official GameBoy Documentation**

**GAME BOY<sup>®</sup>**

*PROGRAMMING MANUAL*

Version 1.1

DMG-06-4216-001-B  
Released 12/03/1999

“Confidential”

This document contains confidential and proprietary information of Nintendo and is also protected under the copyright laws of the United States and foreign countries. No part of this document may be released, distributed, transmitted or reproduced in any form or by any electronic or mechanical means, including information storage and retrieval systems, without permission in writing from Nintendo.

© 1999, 2000 Nintendo of America Inc.

TM and ® are trademarks of Nintendo

## INTRODUCTION

This manual is a combination and reorganization of the information presented in the Game Boy Development Manual, revision G, and the Game Boy Color User's Guide, version 1.3. In addition, it incorporates all information related to Game Boy programming, including programming for Super Game Boy and the Game Boy Pocket Printer.

The abbreviations used in this manual represent the following:

- DMG:** Game Boy (monochrome), introduced on April 21, 1989
- MGB:** Game Boy Pocket (monochrome), introduced on July 21, 1996
- MGL:** Game Boy Light (monochrome), introduced on April 14, 1998
- CGB:** Game Boy Color (color), introduced on October 21, 1998

**Note:** *Where it is not necessary to distinguish between the different monochrome models, DMG is used to refer to both monochrome models, and CGB is used to denote the Color Game Boy. Only where it is necessary to distinguish between the monochrome models is MGB used to denote Game Boy and MGL used to denote Game Boy Light.*

- SGB:** Super Game Boy, introduced on June 14, 1994
- SGB2:** Super Game Boy 2, introduced on January 30, 1998

**Note:** *SGB is used to denote both SGB and SGB2 when no distinction is necessary. SGB2 is used only in cases where distinction is necessary.*

***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

## **PREFACE: TO PUBLISHERS**

### **NINTENDO GAME BOY COLOR SOFTWARE PRE-APPROVAL REQUIREMENTS**

Prior to submitting your CGB software to Lot Check for approval, it is required that you submit it to the Licensee Product Support Group for pre-approval. To assist us with the evaluation of your CGB software and/or product proposal(s), please refer to the following requirements when submitting materials\* for approval.

\* Please do not send original artwork or materials, as they will not be returned.

CGB software and/or product proposals are evaluated based on the following criteria:

- Use of Color  
To ensure that the expectations of the Game Boy Color consumer are met, Mario Club will evaluate the use of color in all CGB games (dual or dedicated) using the following criteria:
  - ◇ **Differentiation** - If a game is to be considered CGB-compatible, then it must appear significantly more colorful than a monochrome Game Boy game when "colorized" by the CGB hardware. The principal measure of this is the number of colors in the background (BG) and the number of colors in the objects (OBJ).
  - ◇ **Simultaneous Colors** - Because CGB hardware automatically "colorizes" monochrome games with up to four colors in the BG palette and up to six colors for two OBJ palettes (three colors per palette), a game typically must display more colors than this automatic "colorization" to be considered a CGB game.
  - ◇ **Appropriate use of Color** - Objects in the game that are based on reality (trees, rocks, animals, and so on) should be a color that we would normally associate with them. For fictional objects, colors should be chosen to show appropriate detail and, when needed, to differentiate unlike objects.
  - ◇ **Variety of Colors** - The CGB is capable of producing a wide range of colors (32,768 to be exact -- albeit not all at the same time). A CGB game should use this capability of the hardware to yield distinctly different colors for objects, characters, areas, and so on.
  - ◇ **Contrast & Saturation** - Two of the elements that make a game look colorful are high contrast and "saturated" or vibrant colors. Pastel colors on a white background will not seem nearly as colorful as the same colors on a dark background. Not every game can use a dark background, but the intensity of the colors should still be maximized as much as possible.

Please detail or demonstrate how your game will utilize color capabilities of the CGB. Use whatever means will best allow you to do so, such as artists renderings, programmed demos, ROM images, written descriptions, and so on.



### ***Game Boy Programming Manual***

- **Game Concept Content**

*We do not require an explanation of, or evaluate game concept content for original CGB titles.* However, if you are planning to “colorize” a previously released monochrome game we require that it include game-play enhancements (beyond simply adding color) to differentiate it from its monochrome counterpart. Such game-play enhancements may include, but are not limited to: additional stages, levels, or areas; new characters; additional items; game-play based on color; and so on. These enhancements must be readily apparent to players familiar with the original monochrome game.

Please submit a written proposal of the enhancements to us for pre-approval. Use whatever additional means that will best allow you to communicate the game-play enhancements, such as storyboards, treatments, videotapes, programmed demos, and so on.

- **Interim ROM Submissions**

We require at least one interim ROM submission to Mario Club (at approximately 50% completion) for preliminary review of the use of color in every CGB game. By reviewing the interim ROM and providing you with feedback in the early stages, we also help ensure that your projects stay on schedule. Final pre-approval is based on Mario Club’s evaluation of a ROM near completion of game development.

*If you wish to arrange electronic transfer of the ROM image, please contact Terral Dunn in our Testing and Engineering department at (425) 861-2670 or by e-mail at “Terraldu@noa.nintendo.com”. Please notify him when you have made an electronic submission for our review.*

- **Proposed Developer**

Please supply us with the name, address and phone number of the proposed developer. If the developer is not an Authorized Nintendo CGB Developer, please contact Melody Morgan at “melomo01@noa.nintendo.com” or 425-861-2618, and she will provide you with the application information.

- **Schedule Information**

Please provide us with an estimated product schedule, including interim ROM submission(s), final Mario Club submission, submission of the master ROM to Lot Check, and the release date.

- **Game Pak Configuration & Game Type**

Please provide us with the estimated Game Pak size in Megabits (Mb) and the RAM size if internal memory is to be used to save game information. Also state whether the game will be compatible with the monochrome Game Boy hardware or if it is dedicated to CGB hardware. For the current Game Pak prices and configurations available, please contact Nintendo’s Licensing Department.

You will be contacted with the evaluation results when the Licensee Product Support Group has completed its evaluation of your ROM or concept submission.

## Table of Contents

	<i>Page Number</i>
<b>Introduction .....</b>	<b>3</b>
<b>Preface: To Publishers.....</b>	<b>5</b>
<b>Chapter 1 System.....</b>	<b>9</b>
<b>Chapter 2 Display Functions .....</b>	<b>47</b>
<b>Chapter 3 Sound Functions.....</b>	<b>77</b>
<b>Chapter 4 CPU Instruction Set .....</b>	<b>93</b>
<b>Chapter 5 Miscellaneous General Information.....</b>	<b>125</b>
<b>Chapter 6 The Super Game Boy System .....</b>	<b>137</b>
<b>Chapter 7 Super Game Boy Sound .....</b>	<b>187</b>
<b>Chapter 8 Game Boy Memory Controllers(MBC) .....</b>	<b>215</b>
<b>Chapter 9 Pocket Printer.....</b>	<b>235</b>
<b>Appendix 1 Programming Cautions.....</b>	<b>253</b>
<b>Appendix 2 Register and Instruction Set Summaries.....</b>	<b>267</b>
<b>Appendix 3 Software Submission Requirements .....</b>	<b>285</b>

*THIS PAGE WAS INTENTIONALLY LEFT BLANK.*

<b>CHAPTER 1: SYSTEM .....</b>	<b>11</b>
<i>Revision History.....</i>	<i>10</i>
<b>1. GENERAL SYSTEM .....</b>	<b>11</b>
1.1 System Overview .....	11
1.2 Game Boy Block Diagram .....	13
1.3 Memory Configuration.....	14
1.4 Memory Map .....	15
1.5 Feature Comparison .....	16
1.6 Register Comparison.....	17
<b>2. CPU.....</b>	<b>18</b>
2.1 Overview of CPU Features .....	18
2.2 CPU Block Diagram .....	20
2.3 Description of CPU Functions .....	22
2.4 CPU Functions (Common to DMG/CGB①).....	24
2.5 CPU Functions (Common to DMG/CBG②).....	28
2.6 CPU Functions (CGB only).....	34

Revision History		
Date	Section	Description
12/3/99	2.6.3	Revision of description for Infrared Communication

## CHAPTER 1: SYSTEM

### 1. GENERAL SYSTEM INFORMATION

#### 1.1 System Overview

##### Structure

At the heart of the DMG/CGB system is a CPU with a built-in LCD controller designed for DMG/CGB use.

##### System

[DMG]	[CGB]
<ul style="list-style-type: none"> <li>➤ Dot-matrix LCD unit capable of grayscale display</li> <li>➤ 64 Kbit – SRAM (for LCD display)</li> <li>➤ 64 Kbit – SRAM (working memory)</li> </ul>	<ul style="list-style-type: none"> <li>➤ Color dot-matrix LCD unit capable of RGB with 32 grayscale shades</li> <li>➤ 128 Kbit – SRAM (for LCD display)</li> <li>➤ 256 Kbit – SRAM (working memory)</li> <li>➤ Infrared communication link (photo transistor, photo LED)</li> </ul>

##### Features common to DMG/CGB

- 32-pin connector (for ROM cartridge connection)
- 6-pin subconnector (for external serial communication)
- DC-DC converter for power source
- Sound amp
- Keys for operation
- Speaker
- Stereo headphone connector
- Input connector for external power source

##### Types of Game Pak Supported

- 1 Game Boy Game Pak  
(Software that uses only the Game Boy functions. When used with Game Boy Color, 4-10 colors are displayed.)
- 2 Game Boy Color Game Pak
  - Game Pak supported by CGB (for use with both CGB and DMG)
  - Game Pak for CGB only (software that runs only on CGB)

***Operating Modes (the following modes apply only to CGB)***

- 1 DMG Mode (when using software for DMG)  
The new registers, expanded memory area, and new features for CGB are not used. Color applications previously associated with palette data BGP, OBP0, and OBP1 are performed by the system.
- 2 CGB Mode (when using software supported or used exclusively by CGB )  
The new registers, expanded memory area, and new features of CGB are available.

**Note:** *To operate in CGB mode, specific code must first be placed in the ROM data area of the user program. For more information, see Chapter 5, Section 2, Recognition Data for CGB(CGB only) in ROM-Registered Data.*

***Power Source***

- Battery/AC adapter/Battery charger

***Accessories (as of April 1999)***

DMG Accessories

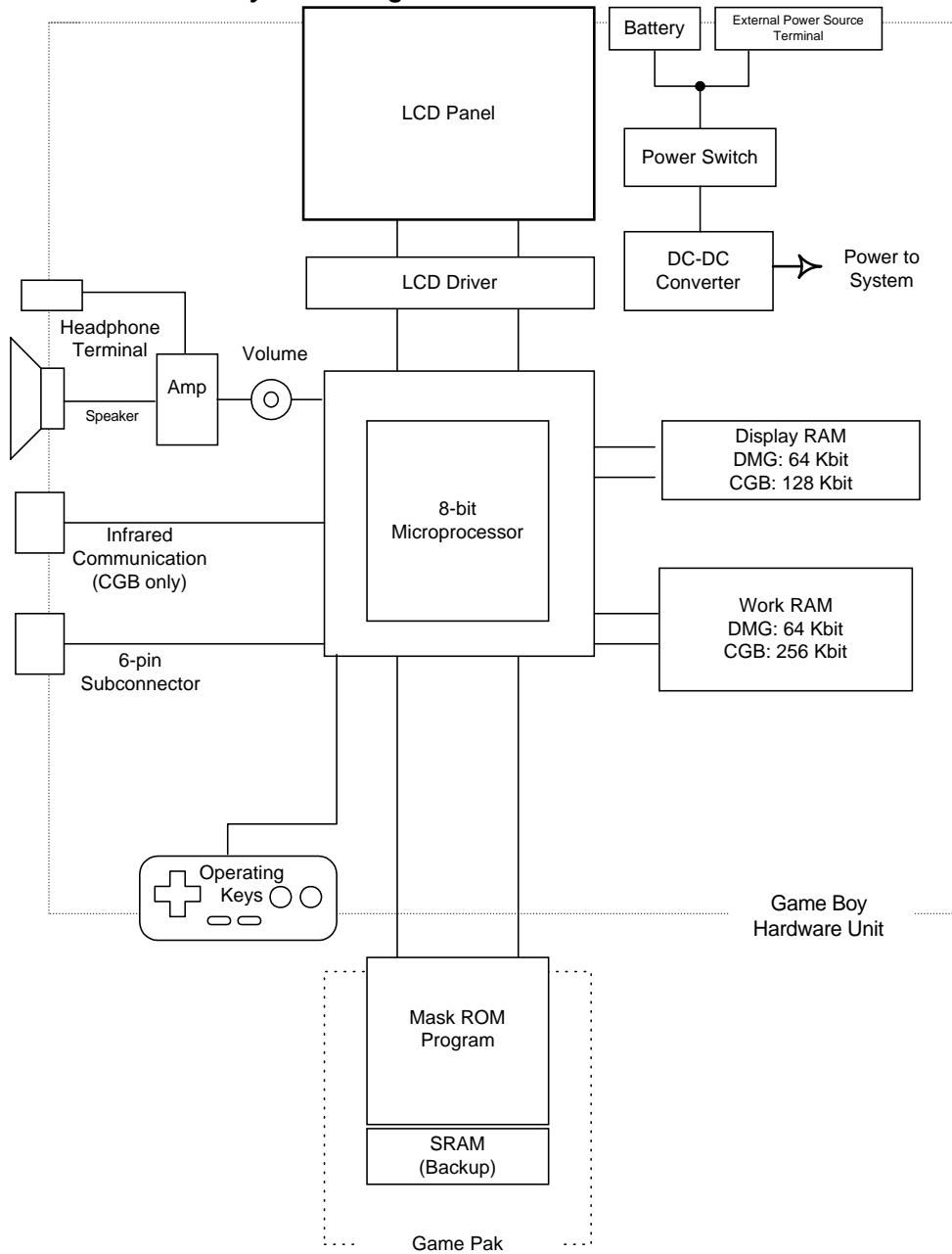
- Communication Cable
- Battery Charger Adapter

MGB/CGB Accessories

- Communication Cable
- AC Adapter
- Battery Pack Charger Set

The 6-pin serial communication subconnector and the AC adapter input connector of the DMG hardware that preceded MGB are shaped differently than those of MGB and CGB. Thus, two types of accessories are available — those exclusively for DMG and those exclusively for MGB/CGB. In addition, a conversion connector is necessary for communication between DMG and MGB/CGB.

## 1.2 GAME Boy Block Diagram





### **1.3 Memory Configuration**

In DMG and CGB, the 32 KB from 0h to 7FFFh is available as program area.

000h-0FFh: Allocated as the destination address for RST instructions and the starting address for interrupts.

100h-14Fh: Allocated as the ROM area for storing data such as the name of the game.

150h: Allocated as the starting address of the user program.

The 8 KB from 8000h to 9FFFh is used as RAM for the LCD display. In CGB, the amount of RAM allocated for this purpose is 16 KB (8 KB x 2), twice the amount allocated for the LCD display in DMG, and this RAM can be used in 8 KB units using bank switching. The 8 KB RAM areas are divided into the following 2 areas.

- 1 An area for character data
- 2 An area for BG (background) display data (Character code and attribute)

The 8 KB from A000h to BFFFh is the area allocated for external expansion RAM.

The 8 KB from C000h to DFFFh is the work RAM area.

In DMG, the 8 KB of working RAM is implemented without change. In CGB, bank switching is used to provide 32 KB of working RAM. This 32 KB area is divided into 8 areas of 4 KB each.

- 1 The 4 KB from C000h to CFFFh is fixed as Bank 0.
- 2 The 4 KB from D000h to DFFFh can be switched between banks 1 through 7.

**Note:** *Use of the area from E000h to FFFFh is prohibited.*

FE00h to FFFFh is allocated for CPU internal RAM.

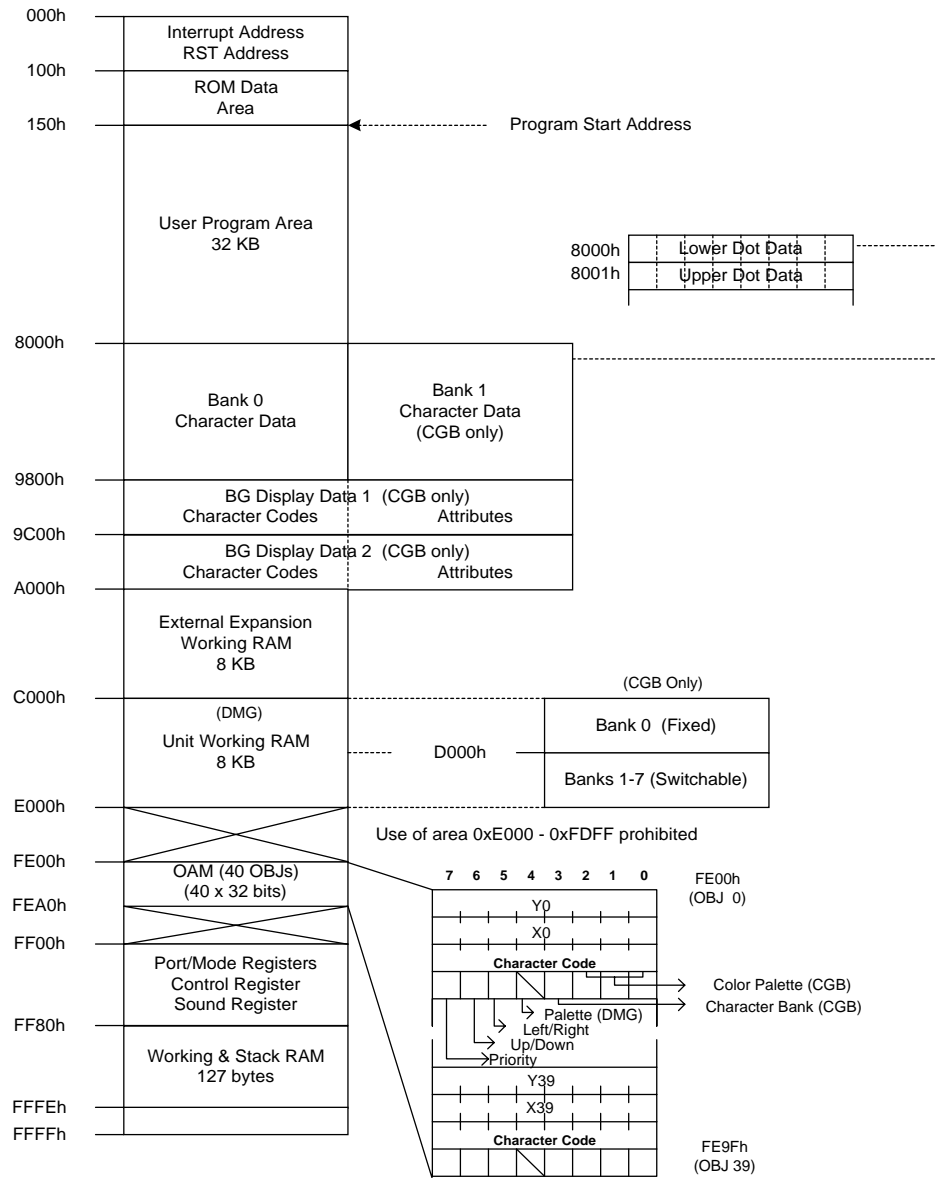
FE00h-FE9Fh: OAM-RAM (Holds display data for 40 objects)

FF00h-FF7Fh & FFFFh: Specified for purposes such as instruction registers and system controller flags.

FF80h-FFFEh: Can be used as CPU work RAM and/or stack RAM.

## 1.4 Memory Map

**Note:** In DMG, there is no bank switching at 8000h-9FFFh and C000h-DFFFh.



## 1.5 Feature Comparison

Item	DMG CPU	CGB CPU
<b>CPU Speed</b> (system operating frequency)	1.05 MHz	1.05 MHz (normal mode) 2.10 MHz (double-speed mode)
<b>Game Boy RAM</b> Work and Stack RAM Work RAM OAM For LCD display	127 x 8 bits 8,192 bytes 40 x 28 bits 8,192 bytes	← 32,768 bytes 40 x 32 bits 16,384 bytes
<b>Game Pak Memory Space</b> ROM (without MBC) RAM (without MBC)	32,768 bytes 8,192 bytes	← ←
<b>LCD Controller</b> Display Capacity Block Structure BG, window Object Number of Usable Characters BG OBJ 8 x 8 8 x 16 Grayscale: BG, window  Grayscale: Object  Object priority Different x coordinates  Same x coordinates	160 x 144 dots  8 x 8 dots 8 x 8 dots or 8 x 16 dots  256 256 128 4 shades, 1 palette  3 shades, 2 palettes  Object with smallest x coord .  Object with lowest OBJ number	160 x 144 x RGB dots  ← ←  512 512 256 4 colors, 8 palettes ( DMG mode: 4 colors, 1 palette) 3 colors, 8 palettes (DMG mode: 3 colors, 2 palettes)  Object with lowest OBJ number (DMG mode: Object with lowest x coord.) ←
<b>Timer &amp; Divider Stages</b>	8-bit timer x 1 16 stages x 1	← ←
<b>Serial Input/Output</b> Baud Rate	8 bits x 1 8 K	← 8K/256K (16K/512K in high-speed mode)
<b>DMA Controller</b> Existing DMA Horizontal blank DMA General-purpose DMA	8000h~DFFFh→OAM --- ---	0h~DFFFh→OAM Game Pak & Work RAM→VRAM Game Pak & Work RAM→VRAM
<b>Interrupt features</b> Internal Interrupts External Interrupts	4 types (maskable) 1 type (maskable)	← ←
<b>Input/Output Ports</b> Serial Input/Output Ports Infrared Communication Port	SIN, SCK, SOUT ---	← R0, R1, R2, R3
<b>Sound Output Circuit</b>	4 sounds	← Monaural (VIN) External Sound Mixable Input

←: Same as in column at left

## 1.6 Register Comparison

Use	DMG CPU		CGB CPU	
	Register	Address	Register	Address
Port/Mode Registers	P1	FF00	←	←
	SB	FF01	←	←
	SC	FF02	←	←
	DIV	FF04	←	←
	TIMA	FF05	←	←
	TMA	FF06	←	←
	TAC	FF07	←	←
	---	---	KEY1	FF4D
	---	---	RP	FF56
Bank Control Registers		---	VBK	FF4F
		---	SVBK	FF70
Interrupt Flags	IF	FF0F	←	←
	IE	FFFF	←	←
	IME		←	
LCD Display Registers	LCDC	FF40	←	←
	STAT	FF41	←	←
	SCY	FF42	←	←
	SCX	FF43	←	←
	LY	FF44	←	←
	LYC	FF45	←	←
	DMA	FF46	←	←
	BGP	FF47	←	←
	OBP0	FF48	←	←
	OBP1	FF49	←	←
	WY	FF4A	←	←
	WX	FF4B	←	←
		---	HDMA1	FF51
		---	HDMA2	FF52
		---	HDMA3	FF53
		---	HDMA4	FF54
		---	HDMA5	FF55
		---	BCPS	FF68
		---	BCPD	FF69
		---	OCPS	FF6A
		---	OCPD	FF6B
	OAM	FE00~FE9F	←	←
Sound Registers	NR x x	FF10~FF26	←	←
	Waveform RAM	FF30~FF3F	←	←

←: Same as in column at left

## **2. CPU**

### **2.1 Overview of CPU Features**

The CPUs of DMG and CGB are ICs customized for DMG/CGB use, and have the following features.

#### **CPU Features**

Central to the 8-bit CPU are the following features, including an I/O port and timer.

- 127 x 8 bits of built-in RAM (working and stack)
- RAM for LCD Display: <DMG> 8 KB/<CGB>16 KB ( )
- Working RAM: <DMG> 8KB/<CGB> 32 KB
- Built-in 16-stage Frequency Divider
- Built-in 8-bit Timer
- 4 types of Internal Interrupts (maskable)
- 1 type of External Interrupt (maskable)
- Built-in DMA Controller
- Input Ports P10 ~ P13
- Output Ports P14 and P15
- Serial I/O Ports SIN, SCK, SOUT
- Infrared I/O Port <CGB only>

#### **LCD Controller Functions**

Game Boy is equipped with functions that provide control of the images displayed on the LCD. Character data used for display is held in system RAM.

- DMG: 4 shades of gray; CGB: 32 shades of gray for each RGB color
- 160 x 144-dot liquid crystal display
- 8 x 8-dot composition of background and window characters
- 8 x 8 or 8 x 16-dot composition of OBJ characters
- Up to 40 objects displayable in 1 screen
- Up to 10 objects displayable on 1 horizontal line
- 40 x 32 bits of built-in RAM (OBJ-RAM for LCD)
- Control of 256 x 256-dot background
- Vertically and horizontally scrollable background
- Window-like functions

**Sound Functions**

Each system is equipped with 4 types of sound synthesis circuitry.

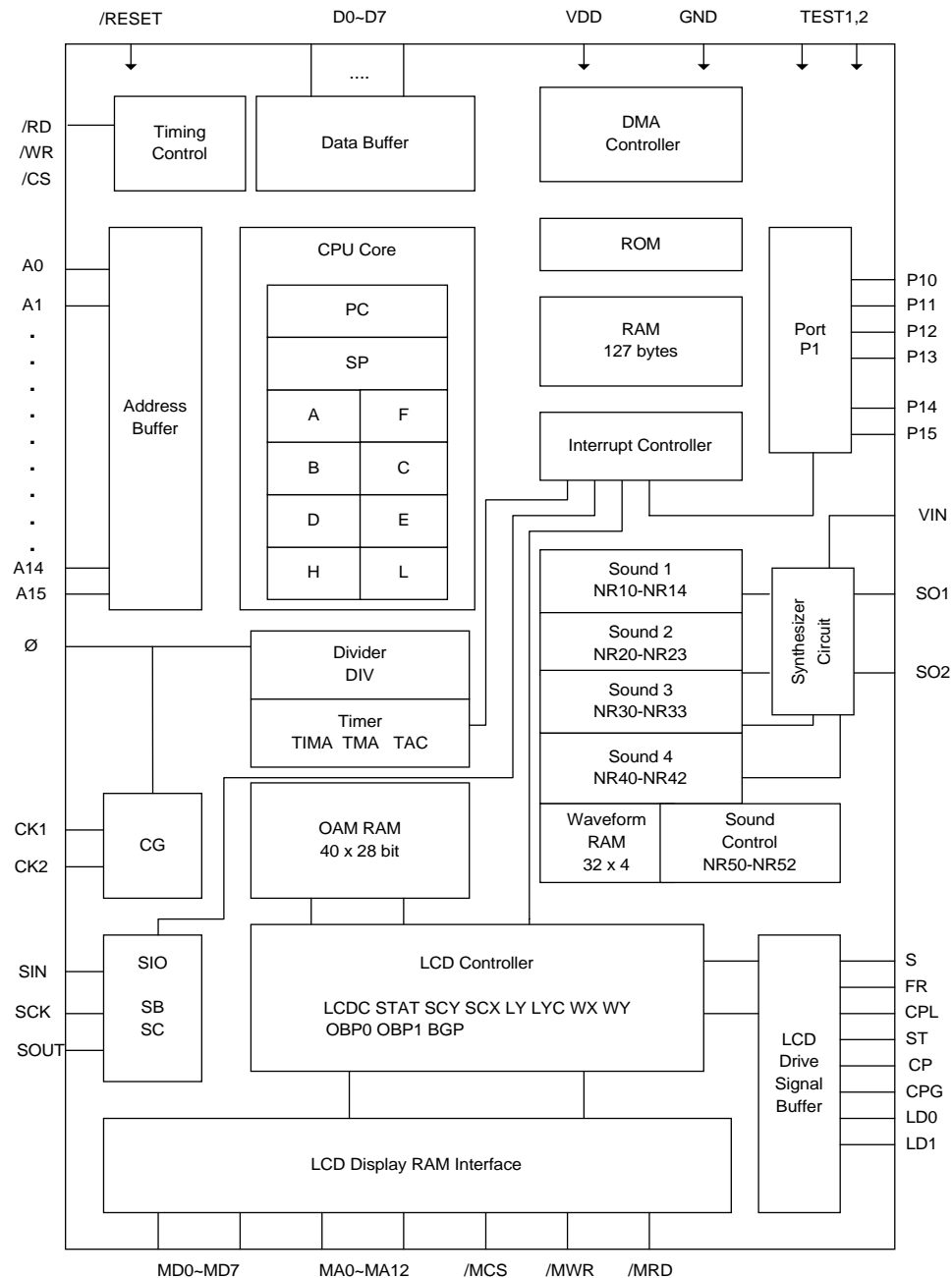
- Sound 1: Quadrangular waveform, sweep and envelope functions
- Sound 2: Quadrangular waveform, envelope functions
- Sound 3: Arbitrary waveform, generated
- Sound 4: White noise, generated
- 2 output channels (output can be allocated to a channel)
- Synthesized output with external sound input <CGB only>

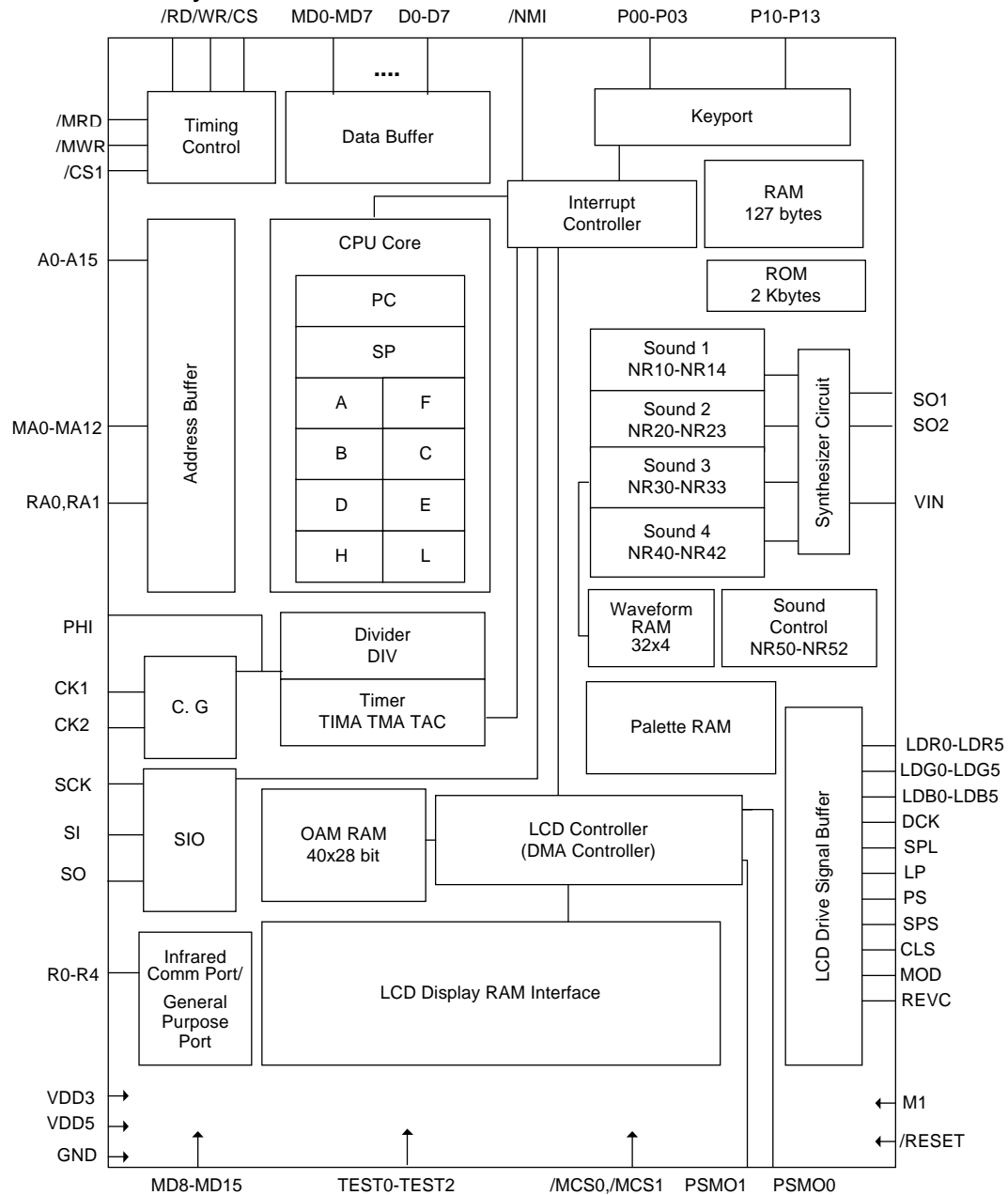
**Miscellaneous**

- An internal monitor program is built into DMG/CGB CPUs. When power is turned on or the Game Boy is reset, the internal monitor program first initializes components such as the ports, then passes control to the user program.
- Instruction cycles
  - <DMG> 0.954  $\mu$ s (source oscillation: 4.1943 MHz)
  - <CGB> 0.954  $\mu$ s/0.477  $\mu$ s, switchable (source oscillation: 8.3886 MHz)

## 2.2 CPU Block Diagram

### Game Boy (DMG/MGB) CPU



**Game Boy Color CPU**



## 2.3 Description of CPU Functions

### Interrupts

There are five types of interrupts available, including 4 types of maskable internal interrupts and 1 type of maskable external interrupt. The IE flag is used to control interrupts. The IF flag indicates which type of interrupt is set.

- LCD Display Vertical Blanking
- Status Interrupts from LCDC (4 modes)
- Timer Overflow Interrupt
- Serial Transfer Completion Interrupt
- End of Input Signal for ports P10-P13

### DMA Transfers

DMA transfers are controlled by the DMA registers.

#### <DMG>

DMG allows 40 x 32-bit DMA transfers from 8000h-DFFFh to OAM (FE00h-FE9Fh).

The transfer start address can be specified in increments of 100h for 8000h-DFFFh.

#### <CGB>

In addition to the DMA transfers method for DMG (from 0000h-DFFFh in CGB), CGB enables two new types of DMA transfer — horizontal blanking and general-purpose DMA transfers.

Note, however, that when performing a DMG-type DMA transfer on CGB, some consideration must be given to specifying the destination RAM area.

For more information, see the DMA Functions section in Chapter 2.

#### 1 Horizontal Blanking DMA Transfer

Sixteen bytes of data are automatically transferred for each horizontal blanking period during a DMA transfer from the user program area (0000h-7FFFh) or external and hardware working RAM area (A000h-DFFFh) to the LCD display RAM area (8000h-9FFFh).

#### 2 General-Purpose DMA Transfer

Between 16 and 2048 bytes of data (specified in 16-byte increments) are transferred from the user program area (0000h-7FFFh) or external and hardware working RAM area (A000h-DFFFh) to the LCD display RAM area (8000h-9FFFh), during the Vertical Blanking Period.

### Timer

The timer is composed of the following:

- TIMA (timer counter)
- TMA (timer modulo register)
- TAC (timer control register)

### Controller Connections

- P10-P13: Input ports
- P14-P15: The key matrix structure is composed of the output ports.

At user program startup, the status of the CPU port registers and mode registers are as follows.

Register	Status
P1	0
SC	0
TIMA	0
TAC	0
IE	0
LCDC	\$83 BG/OBJ ON, LCDC OPERATION
SCY	0
SCX	0
LYC	0
WY	0
W	0
Interrupt Enable (IE)	DI

Stack: FFFEh

**Standby Modes**

The standby functions are HALT mode, which halts the system clock, and STOP mode, which halts oscillation (source oscillation).

**HALT Mode**

Game Boy switches to HALT mode when a HALT instruction is executed.

The system clock and CPU operation halt in this mode. However, operation of source oscillation circuitry between terminals CK1 and CK2 continues. Thus, the functions that do not require the system clock (e.g., DIV, SIO, timer, LCD controller, and sound circuit) continue to operate in this mode.

HALT mode is canceled by the following events, which have the starting addresses indicated.

- 1) A LOW signal to the /RESET terminal  
Starting address: 0000h
- 2) The interrupt-enable flag and its corresponding interrupt request flag are set  
IME = 0 (Interrupt Master Enable flag disabled)  
Starting address: address following that of the HALT instruction  
IME = 1 (Interrupt Master Enable flag enabled)  
Starting address: each interrupt starting address

**STOP Mode**

Game Boy switches to STOP mode when a STOP instruction is executed.

The system clock and oscillation circuitry between the CK1 and CK2 terminals are halted in this mode. Thus, all operation is halted except that of the SIO external clock. STOP mode is canceled by the following events, and started from the starting address.

- 3) A LOW signal to the /RESET terminal  
Starting address: 0000h
- 4) A LOW signal to terminal P10, P11, P12, or P13  
Starting address: address following that of STOP instruction

When STOP mode is canceled, the system clock is restored after 217 times the oscillation clock (DMG: 4 MHz, CGB: 4 MHz/8 MHz), and the CPU resumes operation.

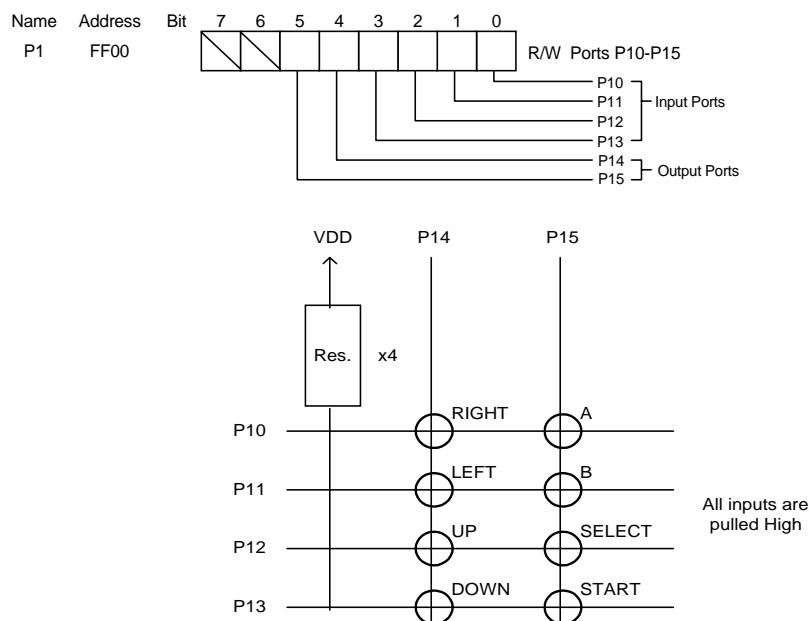
When STOP mode is entered, the STOP instruction should be executed after all interrupt-enable flags are reset, and meanwhile, terminals P10-P13 are all in a HIGH period.

## 2.4 CPU Functions (Common to DMG/CGB<sup>①</sup>)

The CPU functions described here are those that are identical in DMG and CGB. CPU functions that are enhanced in CGB are described in Section 2.5, *CPU Functions (Common to DMG/CGB<sup>②</sup>)*. CPU functions that cannot be used for DMG are described in Section 2.6, *CPU Function (CGB only)*.

### 2.4.1 Controller Data

The P1 ports are connected with a matrix for reading key operations.



When key input is read, a brief interval is interposed between P14 and P15 output and reading of the input, as shown below.

```

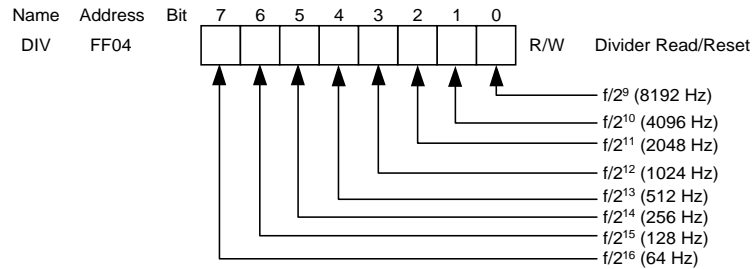
Example: KEY          LD      A, $20          ; Read U, D, L, R keys
                      LD      ($FF00), A      ; Port P14 ← LOW output
                      LD      A, ($FF00)      ; A Register ← Port P10-P13
                      LD      A, ($FF00)      ; Perform this operation twice
                      .
                      .
                      LD      A, ($10)        ; Reads keys A, B, SE, ST
                      LD      ($FF00), A      ; Port P15 ← LOW output

                      LD      A, ($FF00)      ; A Register ← Ports P10-P13
                      LD      A, ($FF00)      ; Perform this operation 6 times
                      LD      A, ($FF00)      ;
                      .
                      .
                      .
                      LD      A, $30          ; Port reset
                      LD      ($FF00), A
                      .
                      RET
  
```

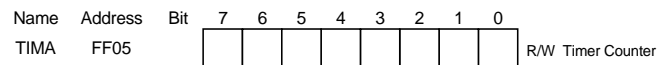
The interrupt request flag (IF: 4) is set by negative edge input at one of the P13-P10 terminals. Negative edge input requires a LOW period of  $2^4$  times source oscillation (DMG = 4 MHz, CGB = 4 MHz/8 MHz).

The interrupt request flag (IF: 4) also is set when a reset signal is input to the /RESET terminal with a P13~P10 terminal in the LOW state.

### 2.4.2 Divider Registers

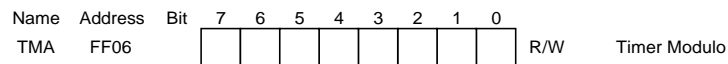


The upper 8 bits of the 16-bit counter that counts the basic clock frequency (f) can be referenced. If an LD instruction is executed, these bits are cleared to 0 regardless of the value being written. f = (4.194304 MHz).

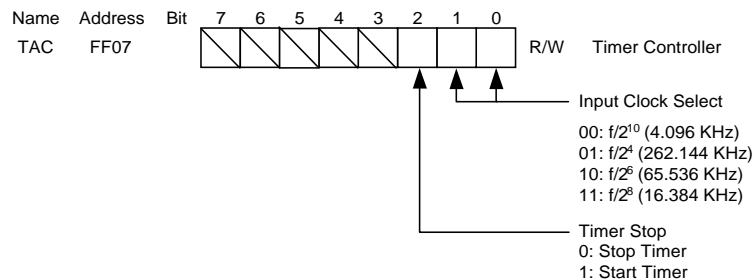


### 2.4.3 Timer Registers

The main timer unit. Generates an interrupt when it overflows.



The value of TMA is loaded when TIMA overflows.



The timer consists of TIMA, TMA, and TAC.

The timer input clock is selected by TAC.

TIMA is the timer itself and operates using the clock selected by TAC.

TMA is the modulo register of TIMA. When TIMA overflows, the TMA data is loaded into TIMA.

Writing 1 to the 2<sup>nd</sup> bit of TAC starts the timer.

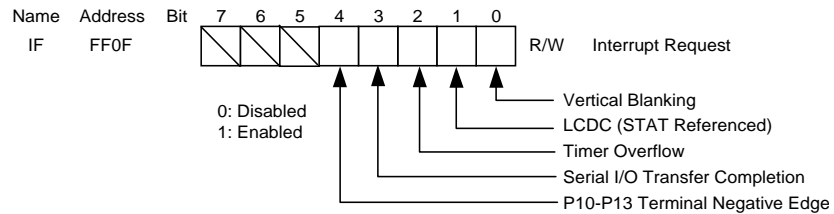
The timer should be started (the TAC start flag set) after the count up pulse is selected. Starting the timer before or at the same time as the count up pulse is selected may result in excessive count up operation.

Example:

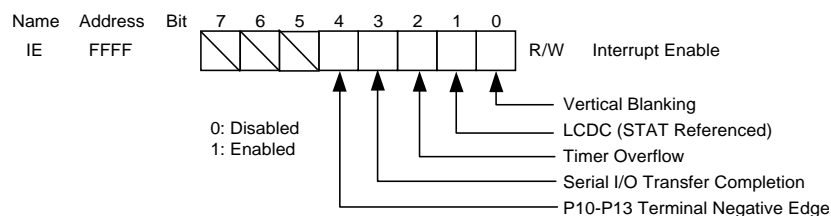
```
LD      A, 3      ;Select a count pulse of f/28
LD      (07), A   ;TAC ← 3 set
LD      A, 7      ;Start timer
LD      (07), A   ;
```

If a TMA write is executed with the same timing as that with which the contents of the modulo register TMA are transferred to TIMA as the result of a timer overflow, the same data is transferred to TIMA.

#### 2.4.4 Interrupt Flags



Bit reset enabled



Name

IME Interrupt Master Enable  
0: Reset by DI instruction, prohibits all interrupts  
1: Set by EI instruction, the interrupts set by the IE registers are enabled

Bit reset enabled

Interrupts are controlled by the IE (interrupt enable) flag.

The IF (interrupt request) flag can be used to determine which interrupt was requested.

The 5 types of interrupts are as follows:

Cause of Interrupt	Priority	Interrupt starting address
Vertical blanking	1	0040h
LCDC status interrupt	2	0048h
Timer overflow	3	0050h
Serial transfer completion	4	0058h
P10-P13 input signal goes low	5	0060h

The LCDC interrupt mode can be selected (see STAT register).

Mode 00  
Mode 01  
Mode 10  
LYC=LY  
consist

When multiple interrupts occur simultaneously, the IE flag of each is set, but only that with the highest priority is started. Those with lower priorities are suspended.

When using an interrupt, set the IF register to 0 before setting the IE register.

The interrupt process is as follows:

- 1 When an interrupt is processed, the corresponding IF flag is set.
- 2 Interrupt enabled.  
If the IME flag (Interrupt Master Enable) and the corresponding IE flag are set, the interrupt is performed by the following steps.
- 3 The IME flag is reset, and all interrupts are prohibited.
- 4 The contents of the PC (program counter) are pushed onto the stack RAM.
- 5 Control jumps to the interrupt starting address of the interrupt.

The resetting of the IF register that initiates the interrupt is a hardware reset.

The interrupt processing routine should push the registers during interrupt processing.

When an interrupt begins, all other interrupts are prohibited, but processing of the highest level interrupt is enabled by controlling the IME and IE flags with instructions.

Return from the interrupt routine is performed by the RET1 and RET instructions.

If the RET1 instruction is used for the return, the IME flag is automatically set even if a DI instruction is executed in the interrupt processing routine.

If the RET instruction is used for the return, the IME flag remains reset unless an EI instruction is executed in the interrupt routine.

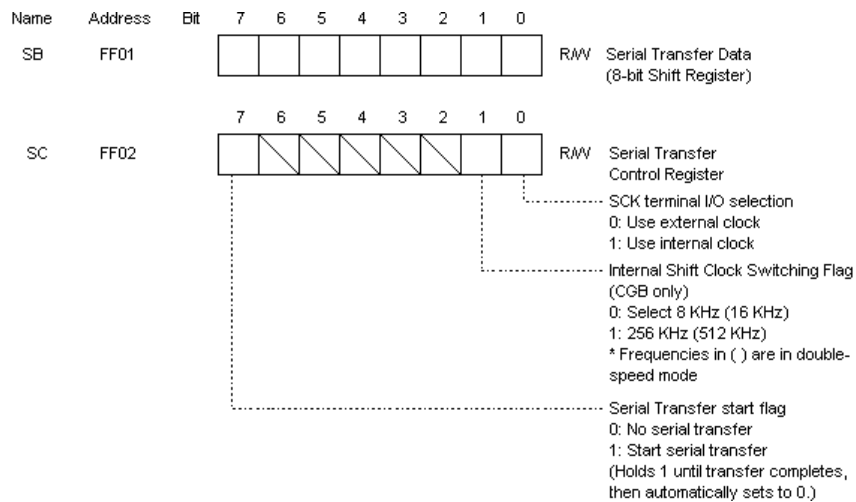
Each interrupt request flag of the IF register can be individually tested using instructions.

Interrupts are accepted during the op code fetch cycle of each instruction.

## 2.5 CPU Functions (Common to DMG/CGB®)

This section describes the CPU functions that have been enhanced in CGB. Functions that are identical in DMG and CGB are described in Section 2.4, *CPU Functions (Common to DMG/CGB®)*. CPU functions not available in DMG are described in Section 2.6, *CPU Functions (CGB only)*.

### 2.5.1 Serial Cable Communication



**Note:** In DMG mode, bit 1 of the SC register is set to 1 and cannot be changed, but the transfer speed is fixed at 8 KHz.

Serial I/O (SIO) is controlled by the SB and SC registers.

The lowest bit (SC0) of the SC register can be used to select shift clock to be either the external clock from the SCK terminal or the internal shift clock.

Sending and receiving occur simultaneously with a serial transfer.

If the data to be sent is set in the SB register and the serial transfer is then started, the received data is set in the SB register when the transfer is finished.

Serial transfer procedure:

- 1 The data is set in the SB register.
- 2 Setting the highest SC register bit (SC 7) to 1 starts the transfer.
- 3 The 3-bit counter is reset and after 8 counts of the shift clock, the transfer is performed until overflow occurs.
- 4 SC7 is reset.
- 5 If the serial transfer completion interrupt is enabled, the CPU is interrupted.

When the shift clock goes low, the contents of the SB register are shifted leftward and the data is output from the highest bit. When the shift clock goes high, input data from the SIN terminal are output to the lowest bit of the SB register.

When the shift clock goes low, the contents of the SB register are shifted leftward and the data is output from the highest bit. When the shift clock goes high, input data from the SIN terminal are output to the lowest bit of the SB register.

When the SCK terminal is in external-clock mode, it is pulled up to VDD.

If the highest bit of the SC register (SC7) is set, reading and writing to the SB register is prohibited.

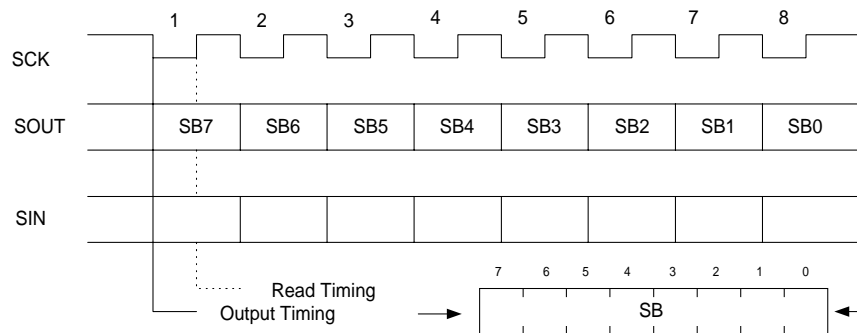
An SIO serial transfer should be started (highest SC bit set) after the external or internal shift clock is selected. Excessive shifting may result if the transfer is started before or at the same time as the shift clock is selected.

If a transfer is performed using the external clock, the data is first set in the SB register, then the SC register start flag is set and input from the external clock is awaited. The transfer start flag must be set each time data is transferred.

The maximum setting for an external clock is 500 KHz.

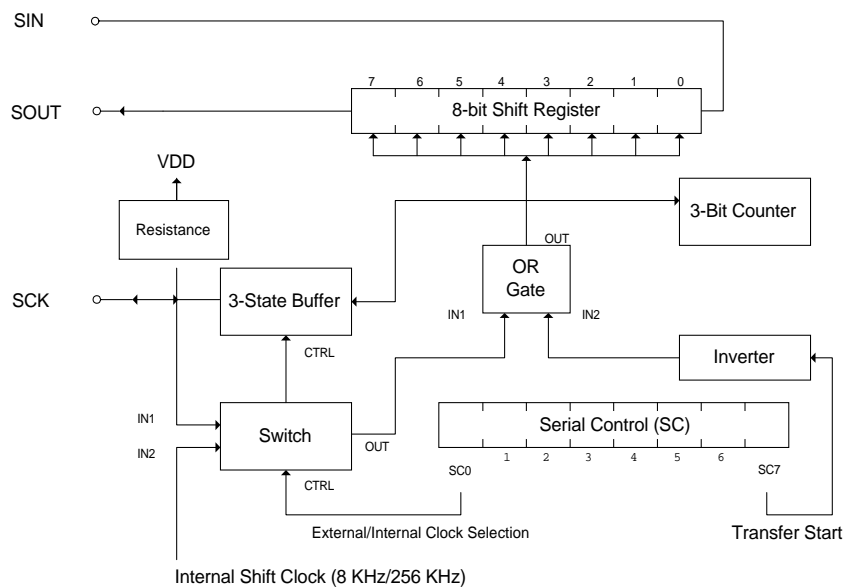
Serial communication (SIO) specifications are essentially the same for DMG and CGB. In CGB, however, the operating speed of the internal shift clock can be set to high by specifying a speed in bit 1.

#### SIO Timing Chart



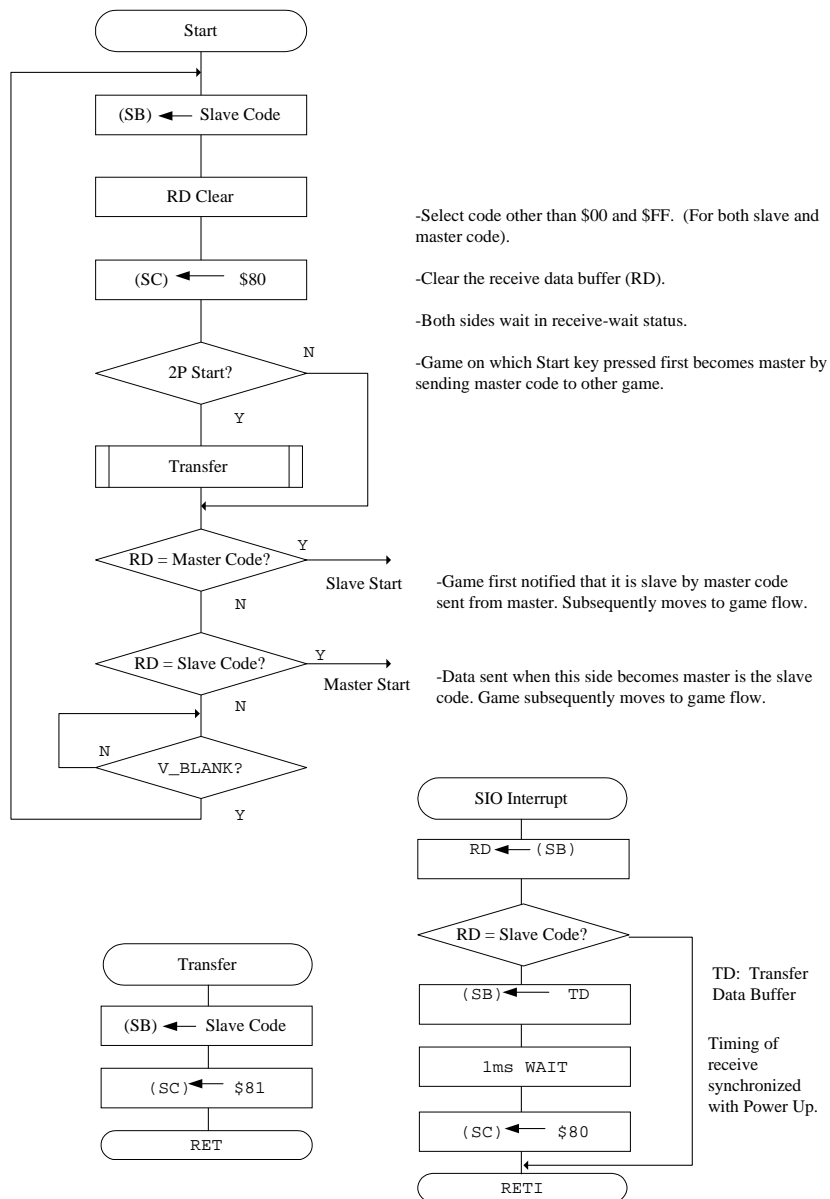


# SIO Block Diagram

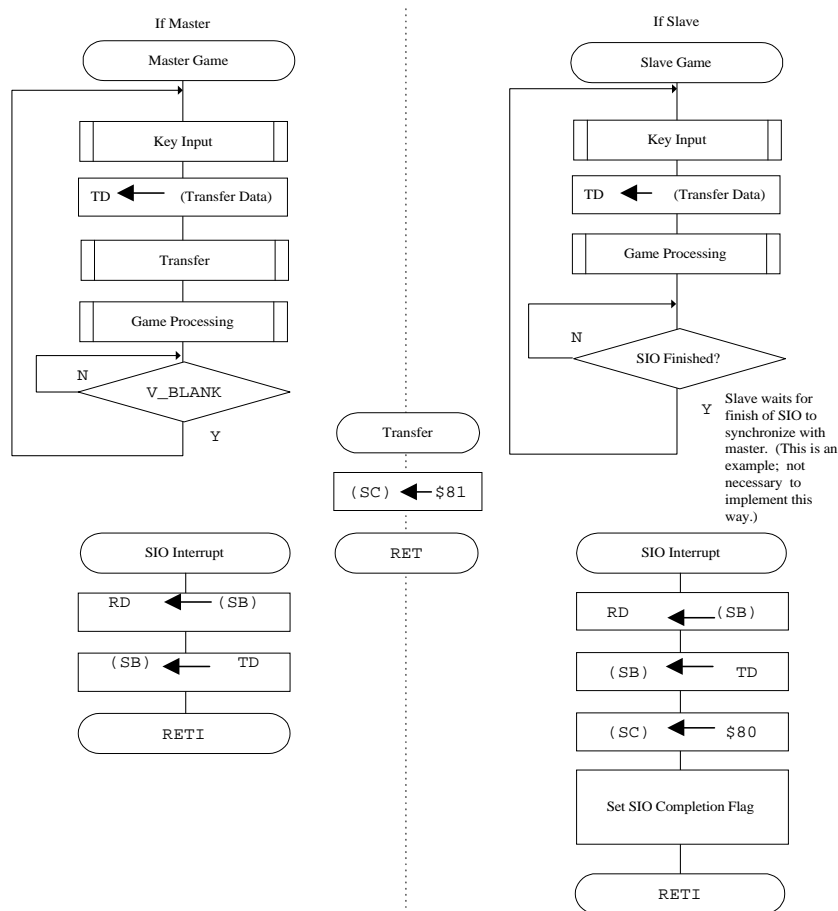


## 2.5.2 Serial Cable Communication: Reference flowchart

## Flow until start of game



**Flow after game start**



Data subsequently sent by the master is placed in (SB) and then sent to the slave at the same time as the (SC) is set to \$81. At exactly that same time, the master receives the slave data. An SIO interrupt is then set in the slave and, as the flowchart indicates, the slave sets the data to be sent to the master (current data).

Because the data sent from the slave are those loaded at the time of the previous interrupt, the data sent to the master are one step (one pass through the main program) behind the current slave data. Exactly the converse is true when this process is viewed from the perspective of the slave. An SIO interrupt is set in the master, and the master sets the data to be sent to the slave (current data). In this case, because the data sent from the master are those loaded at the time of the previous interrupt, the data sent to slave are one step (one pass through main program) behind the current master data. (\*The data of the master and slave can be synchronized by setting the data for each back 1 pass.)

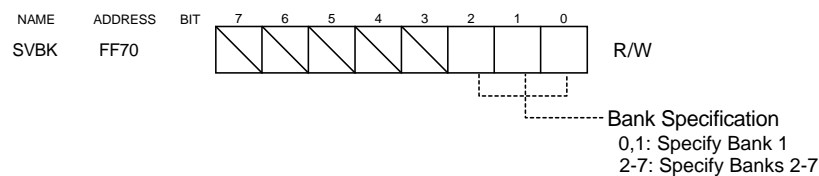
In the example, 1 byte is sent per frame. (This is not required.) If several bytes are sent continuously, a transmission interval longer than the processing time of other interrupts (e.g. V\_BLANK) should be used (usually around 1 mS). The reason is that if an attempt is made to communicate with the slave during another interrupt, the slave cannot receive the data until after the interrupt is finished. If the next data is transmitted before the other interrupt is finished, the slave will be unable to receive the initial data of the transmission.

## 2.6 CPU Functions (CGB only)

This section describes CPU functions that can be used only with CGB. Functions that are identical in DMG and CGB are described in Section 2.4, *CPU Functions (Common to DMG/CGB①)*. For information on CPU functions enhanced in CGB, see Section 2.5, *CPU Functions (Common to DMG/CGB②)*.

### 2.6.1 Bank Register for Game Boy Working RAM

The 32 KB of Game Boy working RAM is divided into 8 banks of 4 KB each. The CPU memory space C000h-CFFFh is set to Bank 0, and the space D000h-DFFFh is switched between banks 1-7. Switching is performed using the lowest 3 bits of the bank register, SVBK. (If 0 is specified, Bank 1 is selected.)



**Note:** This register cannot be written to in DMG mode.

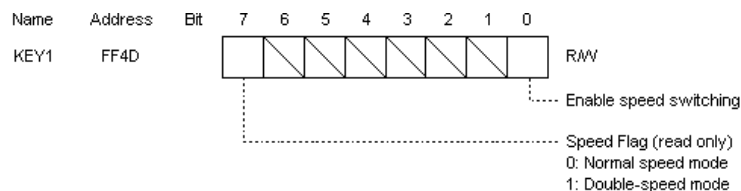
### 2.6.2 CPU Operating Speed

The speed of the CGB CPU can be changed to suit different purposes. In normal mode, each block operates at the same speed as with the DMG CPU. In double-speed mode, all blocks except the liquid crystal control circuit and the sound circuit operate at twice normal speed.

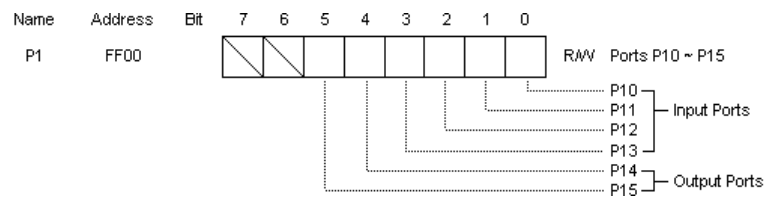
Normal mode: 1.05 MHz (CPU system clock)  
 Double-speed mode: 2.10 MHz (CPU system clock)

#### Switching the CPU Operating Speed

Immediately after the CGB CPU is reset (immediately after reset cancellation), it operates in normal mode. The CPU mode is switched by executing a STOP instruction with bit 0 of register Key 1 set to a value of 1. If this is done in normal mode, the CPU is switched to double-speed mode; otherwise it is switched to normal mode. Bit 0 of register Key 1 is automatically reset after the operating speed is switched. In addition, bit 7 of register Key 1 serves as the CPU speed flag, indicating the current CPU speed.



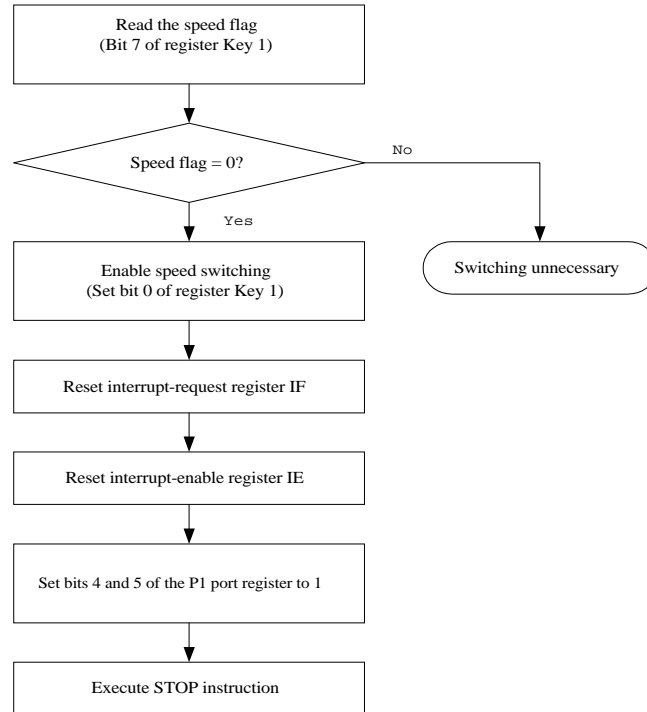
**Note:** When bit 0 of register Key 1 is set to 1, the standby function cannot be used. When using the standby function, always confirm that bit 0 of register Key 1 is set to 0. When switching the CPU speed, all interrupt-enable flags should be reset and a STOP instruction executed with bits 4 and 5 of the P1 port register set to 1, as with the standby function (STOP mode). When the CPU speed is switched, a return from STOP mode is automatic, so it is not necessary to generate a STOP mode cancellation. However, until the CPU speed has been changed and the system clock returns, bits 4 and 5 of the P1 port register should be made to hold the value 1.



Approximately 16 ms is required to switch from normal to double-speed mode, and approximately 32 ms is needed to switch from double-speed to normal mode. In double-speed mode, the DIV register (FF04h) and the TIMA register (FF05h) both operate at double speed. Battery life is shorter in double-speed mode than in normal mode. The use of double-speed mode requires the corresponding mask ROM and MBC.

### Flow of Switching (when switching to double-speed mode)

In case the CPU operating speed needed to be switched, the current speed should always be checked first using the speed flag (bit 7 of the KEY 1 register). This ensures that the speed will be switched to the intended speed.



### Switching Routine (example)

```

LD      HL, KEY1
BIT     7, (HL)
JR      NZ, _NEXT
SET     0, (HL)
XOR     A
LD      (IF), A
LD      (IE), A
LD      A, $30
LD      (P1), A
STOP
  
```

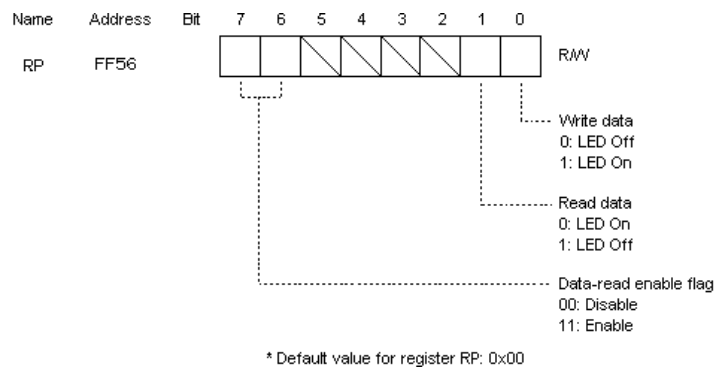
\_NEXT

### 2.6.3 Infrared Communication

#### 2.6.3.1 Port Register

The CGB system is equipped with an infrared communication function. An infrared signal can be output by writing data to bit 0 of RP register. A received infrared signal is latched internally in the CPU by positive edge of the system clock. (System clock goes to HIGH from LOW.) The latched data can be read beginning from bit 1 of RP register by setting bits 6 and 7 to 1.

**Note:** When data is not sent or received, always set the values of RP register to 00h. This register cannot be written to in DMG mode.



#### 2.6.3.2 Controlling Infrared Communication

Sender:

Setting bit 0 of the RP register to 1 causes the LED to emit light; setting it to 0 turns off the LED.

Receiver:

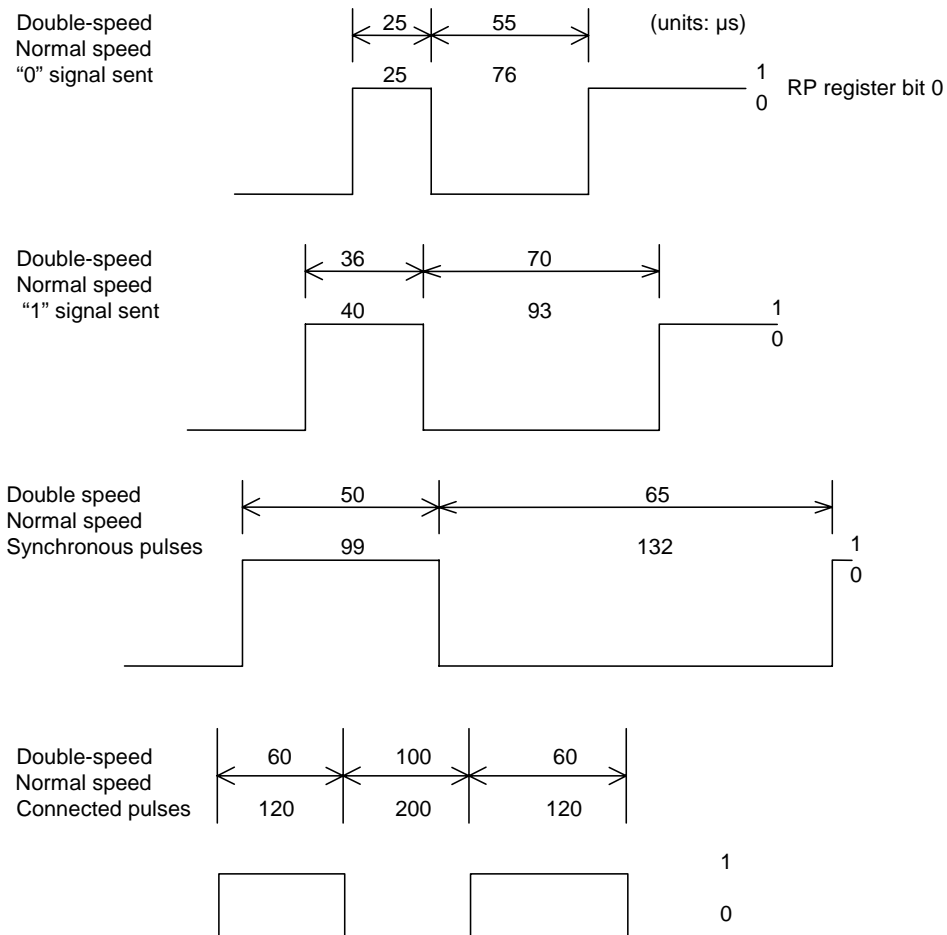
If the photo transistor detects infrared light, bit 1 of the RP register is set to 0; if no infrared light is detected, this bit is set to 1.

#### 2.6.3.3 Basic Format

When the receiver recognizes the unmodified signal from the sender as a logical value of 1 or 0, the receiver actually cannot distinguish between the continuous transmission of 1s and the absence of received infrared light. The status of the receiver is identical under these conditions. Consequently, to ensure proper data transmission from sender to receiver in Game Boy Color infrared communication, signals are distinguished by the size of the interval between the rising edge of the pulse of one received signal to the rising edge of the subsequent received signal.



The following illustrates signals from a sender.



Scatter in the source oscillation of Game Boy Color produces slight individual differences.

#### **2.6.3.4 Preparing for Data Transmission and Reception**

To use infrared communication, data reception must be enabled by setting bits 6 and 7 of Game Boy Color RP register to 1. However, even with both of these bits set to 1, data cannot immediately be received. After setting bits 6 and 7 to 1, at least 50 ms should be allowed to pass before using the infrared port.

### 2.6.3.5 Transmitted Data

When data is transmitted and received, it is transmitted in packets. Each packet comprises the 4 parts shown below, and each part is sandwiched between synchronous pulses. For more information, see Section 2.6.3.7, Details of Data Transmission and Reception.

The data that comprises a packet is transmitted 1 bit at a time beginning from the MSB.

#### Transmission Packet

Connector	Header	Data	Checksum
-----------	--------	------	----------

Connector:

Signal that implements an infrared communication connection between two Game Boy Color machines. This is always required in the initial packet. When the receiver receives the connector and recognizes it as a connecting pulse, the receiver returns the same pulse to the sender. The sender then determines whether this signal is a normal connecting pulse. If it is not recognized as a normal pulse, transmission is interrupted at this stage.

Header:

Data indicating the type of data being sent and the total number of bytes.

Byte 1: Communication command

5Ah: transmission of raw data

At present, any value other than 5Ah causes an error.

(To be used for by other devices in future)

Byte 2: Total number of data in data portion of the packet

01h-FFh: Number of data

00h: Indicates completion of communication to receiver.

Data:

The transmitted data itself. Maximum of 255 bytes.

There are no data if completion of communication is indicated to the receiver.

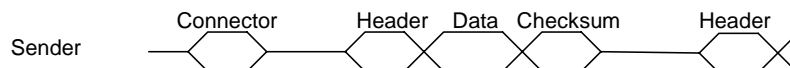
(The data portion of the packet consists only of a synchronous pulse.)

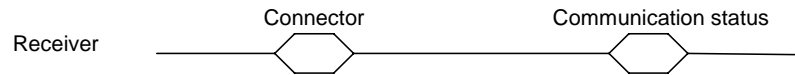
Checksum:

2 bytes of data consisting of the sum of the header and all data in the data portion of the packet. Following this, the communication status is returned from receiver to sender.

### 2.6.3.6 Flow of Data Transmission and Reception

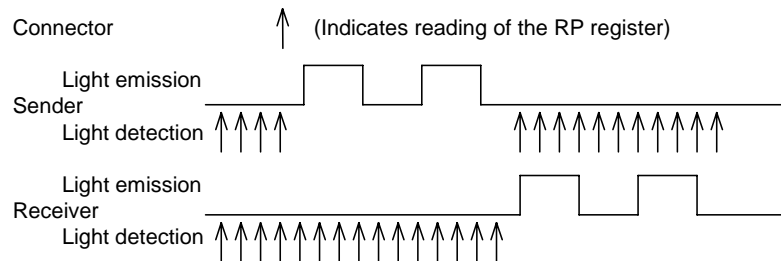
When data is transmitted and received, both Game Boy Color units are first placed in receive status. The one with the send indicator is then designated as the sender, and the other one is designated as the receiver. The flow of data transmission is shown below.





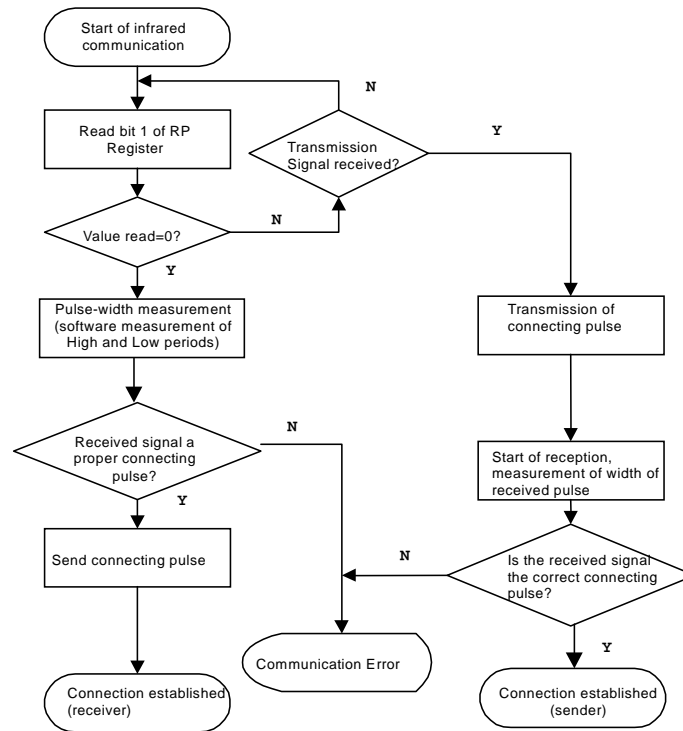
- 1 Sender transmits connecting pulse.
- 2 The receiver calculates the width of the received connecting pulse. If the value is correct, the receiver returns the same connecting pulse to the sender.
- 3 The sender calculates the width of the connecting pulse returned by the receiver. If the value is correct, the sender determines that a connection has been properly established.
- 4 The header is transmitted.
- 5 The data is transmitted.
- 6 The checksum is transmitted.
- 7 The receiver returns the communication status to the sender.
- 8 When communication is complete, the header of the subsequently transmitted packet is set to 00h + 00h.

### 2.6.3.7 Details of Data Transmission and Reception

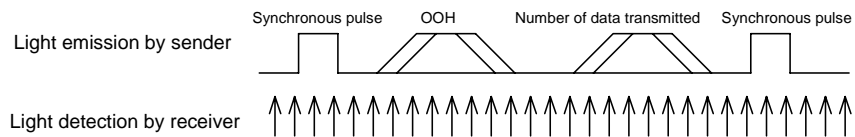


The two Game Boy Color machines perform initial data reception, then the one designated as the sender (e.g., by operations such as pressing button A) begins transmission.

The following illustrates the flow for implementing a connection.

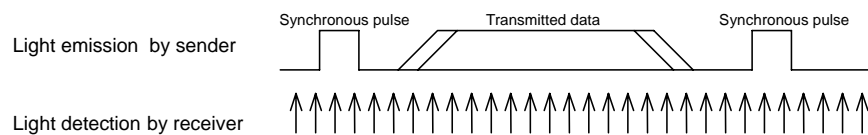


#### Header



One byte indicating the data type and 1 byte indicating the number of transmitted data are sandwiched between synchronous pulses.

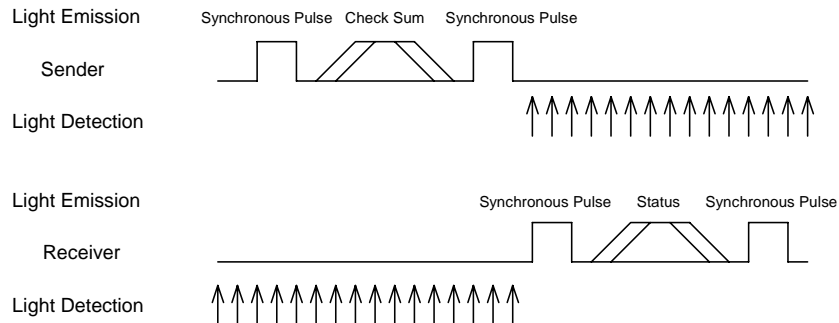
#### Data



Between 1 and 255 bytes of transmitted data are sandwiched between synchronous pulses.

## Game Boy Programming Manual

### Checksum



A 2-byte checksum consisting of the sum of the header and transmitted data is sandwiched between synchronous pulses. The receiver uses the checksum to determine whether the transmission was performed properly and notifies the sender of the results of communication status.

The following section describes the details of communication status determination.

#### 2.6.3.8 Communication Status

8Bh : Communication OK

04h : Checksum error

The results of the checksum calculated by the receiver do not agree with the checksum sent by the sender.

In the following cases, the communication status cannot be returned to the sender even if an error is generated during communication (no response from receiver).

- The wrong communication protocol is used.
- Data is transmitted using the wrong pulse width.
- One of the Game Boy Color units is operating in double-speed mode and the other is operating in normal mode.
- Communication is affected by sunlight or obstruction of the signal light.

**2.6.3.9 Communication Error Processing**

If an error described above in Communication Status is generated, the following error codes are returned by subroutine.

Error Code	Error Description
04h	Checksum error (same for sender and receiver): The results of the checksum calculated by the receiver and the checksum sent by the sender do not agree.
10h	Pulse width error: Generated by the receiver when the width of the pulse of the signal sent by the sender is too wide or narrow. Generated by the sender when the width of the pulse of the signal sent by the receiver is too wide or narrow.
20h	Communication error: Communication prevented by other causes. The subroutine provided by Nintendo treats as an error the case when the data value of the second byte of the received header exceeds the number of data items to be received, as determined beforehand by the receiver. The routine also generates an error if the communication command value of byte 1 of the header is not 5A.

### **2.6.3.10 Usage Notes**

When programming use of the infrared port, please note the following.

- When transmitting more than 256 bytes of data, ensure that the receiver keeps track of which packet number is being received. When a communication error (status not returned even though data was received) is generated, the sender will re-send the data, and the receiver may lose track of the packet number (see note 1 of previous section).
- The sender is prone to entering an endless loop when the packet signifying transmission completion is received. Therefore, the receiver should remain in receive status for approximately 300  $\mu$ s after returning the status (see note 2 of previous section).
- Depending on the power reserve of the battery, infrared communication may cause a sudden drop in battery voltage and a complete loss of power.
- Ensure that the speed of the two communicating Game Boy Color machines is the same (both double-speed or both normal speed during communication).
- Noise can be heard from the speaker and headphones during communication, but this does not indicate a problem with the hardware.
- Be careful that malfunctions/lock-ups do not occur when infrared communication signals are input from other game software and devices. Use particular care when using the same subroutine to communicate between various types of games because malfunctions/lock-ups are especially likely to occur in such cases. (Before performing data communication, confirm that the other hardware involved in the transmission is using the same game. This can be accomplished by means such as exchanging a unique key code.)
- Though very rare, it is possible that at the final communication stage, one Game Boy will terminate normally and the other abnormally due to an unexpected external disturbance.

The following are items to note when using an infrared communication subroutine other than that provided by Nintendo.

- Ensure that error-handling is implemented to prevent the program from entering an endless loop when communication is interrupted by sunlight or obstruction of the signal light.
- To reduce power consumption, use a maximum infrared LED emission pulse duration of 150  $\mu$ s and a duty ratio of approximately 1/2.
- Do not leave the infrared LED or photo transistor(Amplifier and Read Enable) ON when not using infrared communication.

**2.6.3.11 Specifications**

- 1) Communication Speed  
Normal-speed mode: approximately 7.5 Kbps  
Double-speed mode: approximately 9.5 Kbps
- 2) Communication distances: Minimum, 10 cm, Typical, 15 cm
- 3) Recommended directional angle: approximately  $\pm 15^\circ$



***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

## **CHAPTER 2: DISPLAY FUNCTIONS..... 48**

<b>1. GENERAL DISPLAY FUNCTIONS .....</b>	<b>48</b>
1.1 Character Composition.....	48
1.2 LCD Display RAM .....	49
1.3 Character RAM .....	50
1.4 BG Display .....	54
1.5 LCD Screen .....	56
1.6 LCD Display Registers.....	57
1.7 OAM Registers.....	62
1.8 DMA Registers.....	64
1.9 OBJ Display Priority.....	70
<b>2. LCD COLOR DISPLAY (CGB ONLY) .....</b>	<b>72</b>
2.1 Color Palettes .....	72
2.2 Color Palette Composition .....	73
2.3 Writing Data to a Color Palette.....	73
2.4 Overlapping OBJ and BG .....	75
2.5 Display Using Earlier DMG Software (DMG mode).....	76

## CHAPTER 2: DISPLAY FUNCTIONS

### 1. GENERAL DISPLAY FUNCTIONS

#### 1.1 Character Composition

- The basic character size is an 8 x 8-dot composition.
- With characters of the basic size:
  - 128 OBJ-only characters are available (256 with CGB)
  - 128 BG-only characters are available (256 with CGB)
  - 128 characters can be registered both as OBJ and BG characters (256 with CGB)
- On DMG, characters can be represented using 4 shades of gray (including transparent). On CGB, characters can be represented using 32 shades for each color of RGB.
- The basic character size can be switched to an 8 x 16-dot composition for OBJ characters only. In this case, however, only even-numbered character codes can be specified. Even if an odd-numbered character code is specified, the display will be the same as that seen with an even-numbered code.
- Up to 40 OBJ characters can be displayed in a single screen, and up to 10 characters can be displayed on each horizontal line.
- The display data for OBJ characters are as follows:
  - y-axis coordinate
  - x-axis coordinate
  - Character code
  - Attribute data
- Data are written to OAM from working RAM by DMA transfer.
- OBJ characters are automatically displayed to the screen using the data written to OAM.
- Data specification ranges for OBJ characters:
  - $00 \leq \text{character code} \leq \text{FFh}$
  - $00 \leq X \leq \text{FFh}$
  - $00 \leq Y \leq \text{FFh}$

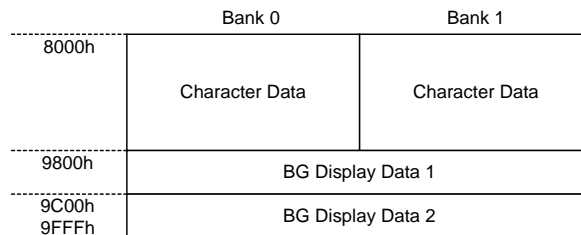
## 1.2 LCD Display RAM

The DMG CPU has 8 KB (64 Kbits) of built-in LCD display RAM.

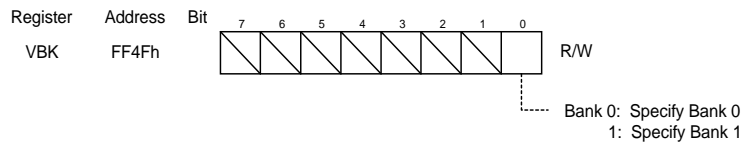
In CGB, 16 KB of memory can be joined in the 8 MB (64-Mbit) memory area (8000h-9FFFh) by bank switching using the register VBK (FF4Fh). Bank switching is used exclusively in CGB and cannot be used in DMG mode.

### ➤ Mapping of LCD Display RAM

The 16 KB of memory in CGB is partitioned into 2 x 8 KB by register VBK.



### ➤ Bank Register (CGB) for LCD Display RAM



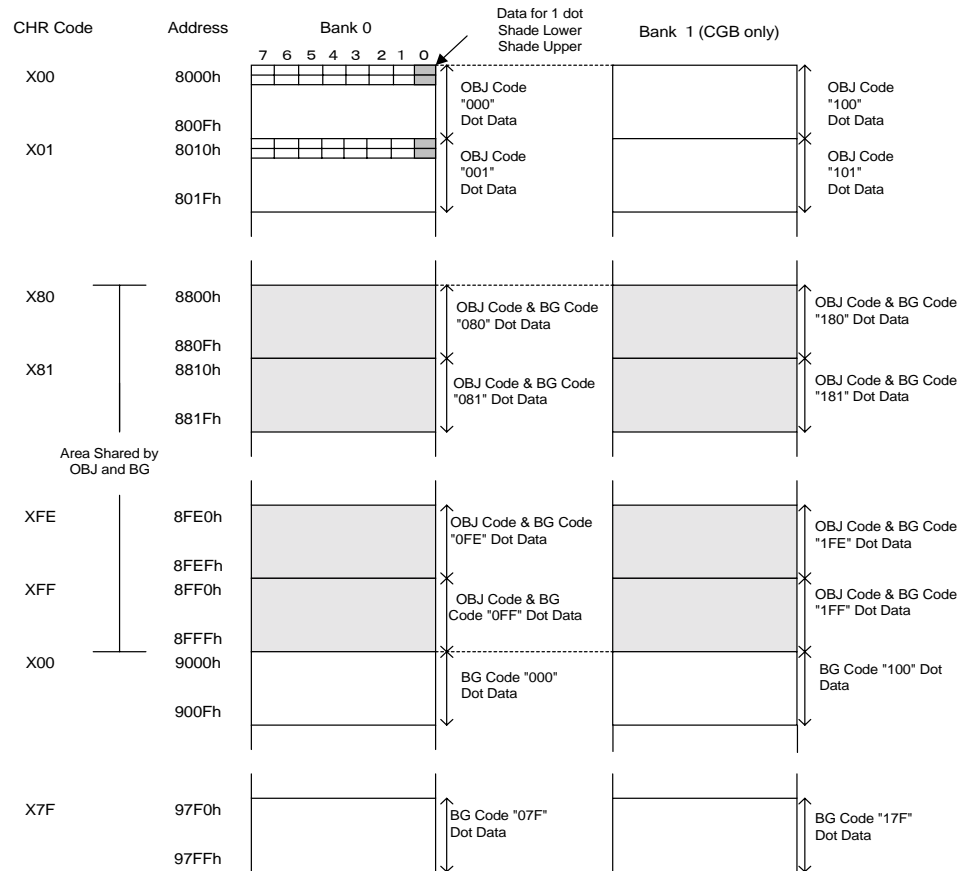
Bank 0 is selected immediately after cancellation of a reset signal.

This function is available only in CGB. In DMG mode, bit 0 is forcibly set 0, and its value cannot be changed to 1.

### **1.3 Character RAM**

- Character data can be written to the 6144 bytes from 8000h to 97FFh.
- The area from 8000h to 8FFFh is allocated for OBJ character data storage.
- The register LCDC can be used to select either 8000h-8FFFh or 8800h-97FFh as the area for storing BG and window character data.
- If the BG character data are allocated to 8000h-8FFFh, these data share an area with OBJ data, and the character dot data that correspond to the CHR codes also are the same.
- By means of bank switching, CGB can store twice the amount of character data in LCD display RAM that DMG can store. In this case, both Bank 1 and Bank 0 have the same mapping as the area in DMG.

## Character Code Mapping



With BG character data allocated to 8800h-97FFh:

- The case of 8 x 8 dots/block for both BG and OBJ:

CHR Codes:

<DMG>

OBJ: 256 x 1

BG: 256 x 1

<CGB>

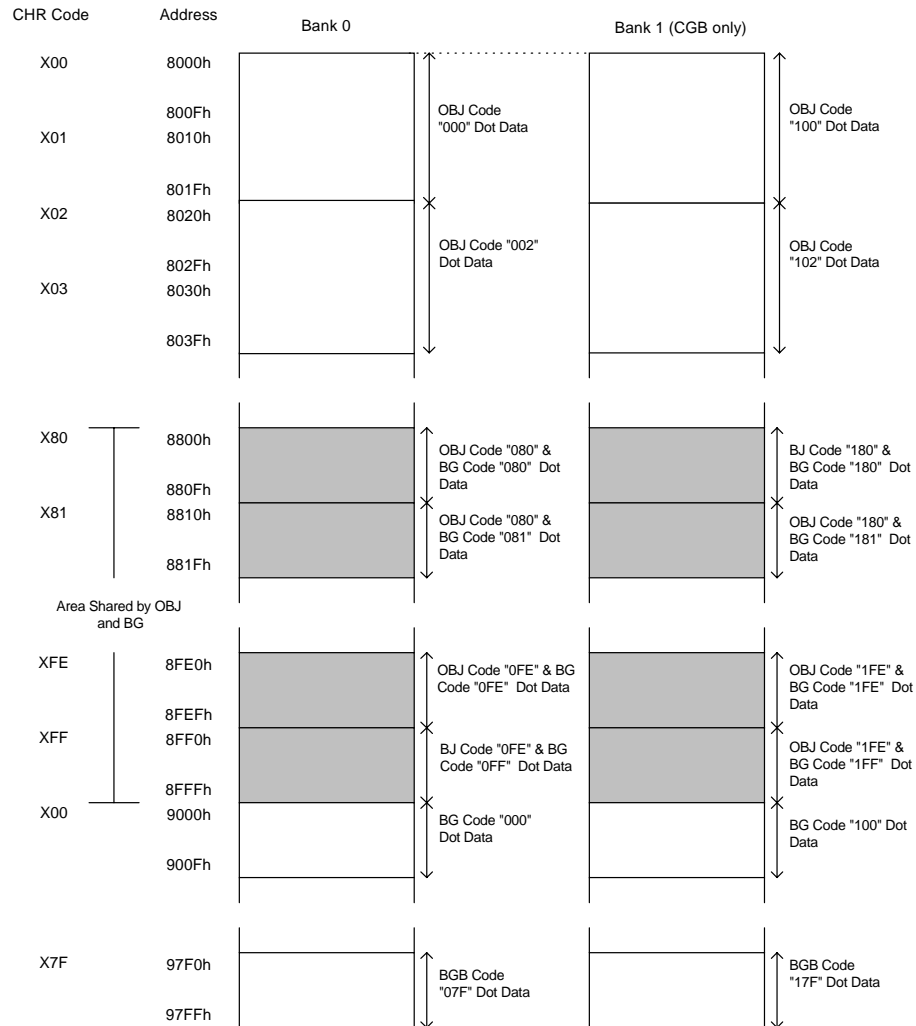
OBJ: 256 x 2

BG: 256 x 2

**Note:** Because bank switching is not available in DMG mode, Bank 1 on the right side of the figure is not available in this mode.

*Game Boy Programming Manual*

8 x 16 dots/block (OBJ) and 8 x 8 dots/block (BG):



CHR Codes:

<DMG>

OBJ: 128 x 1

BG: 256 x 1

<CGB>

OBJ: 128 x 2

BG: 256 x 2

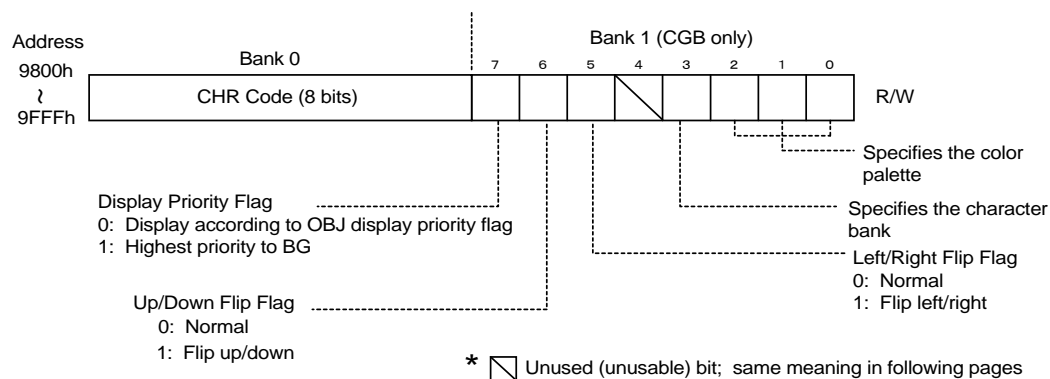
If BG character data are allocated to 8000h-8FFFh, these data share an area with OBJ data, and

the dot data that correspond to the CHR codes also are the same.

**Note:** *Because bank switching is not available in DMG mode, Bank 1 on the right side of the figure is not present in this mode.*



## 1.4 BG Display



Two screens of BG display can be held, Data 1 or Data 2.

Whether the BG display data are allocated to 9800h-9BFFh or to 9C00h-9FFFh is determined by bit 3 of the LCD register (FF40h).

Because bank switching is not available in DMG mode, Bank 1 on the right side of the figure is not present in this mode.

	Bank 0	Bank 1 (CGB only)
9800h	BG Display Data 1	
9C00h		
9FFFh	BG Display Data 2	

Data for 32 x 32 character codes (256 x 256 dots) can be specified from 9800h or 9C00h as BG display data. Of these, data for 20 x 18 character codes (160 x 144 dots) are displayed to the LCD screen.

The screen can be scrolled vertically or horizontally one dot at a time by changing the values of scroll registers SCX and SCY.



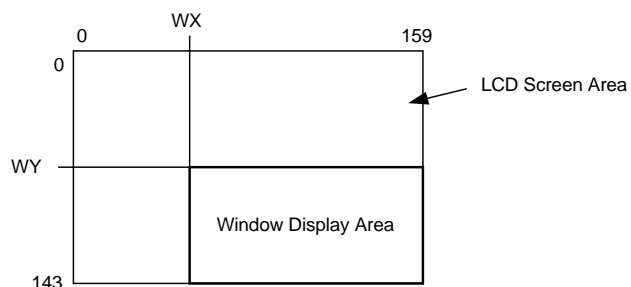
## 1.5 LCD Screen

### ➤ Window Display

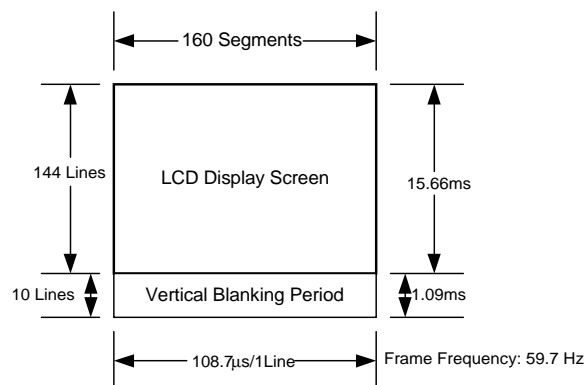
Specifying a position on the LCD screen using registers WX and WY causes the window to open downward and rightward beginning from that position.

Window display data also can be specified as character codes, beginning from 9800h or 9C00h in external SRAM.

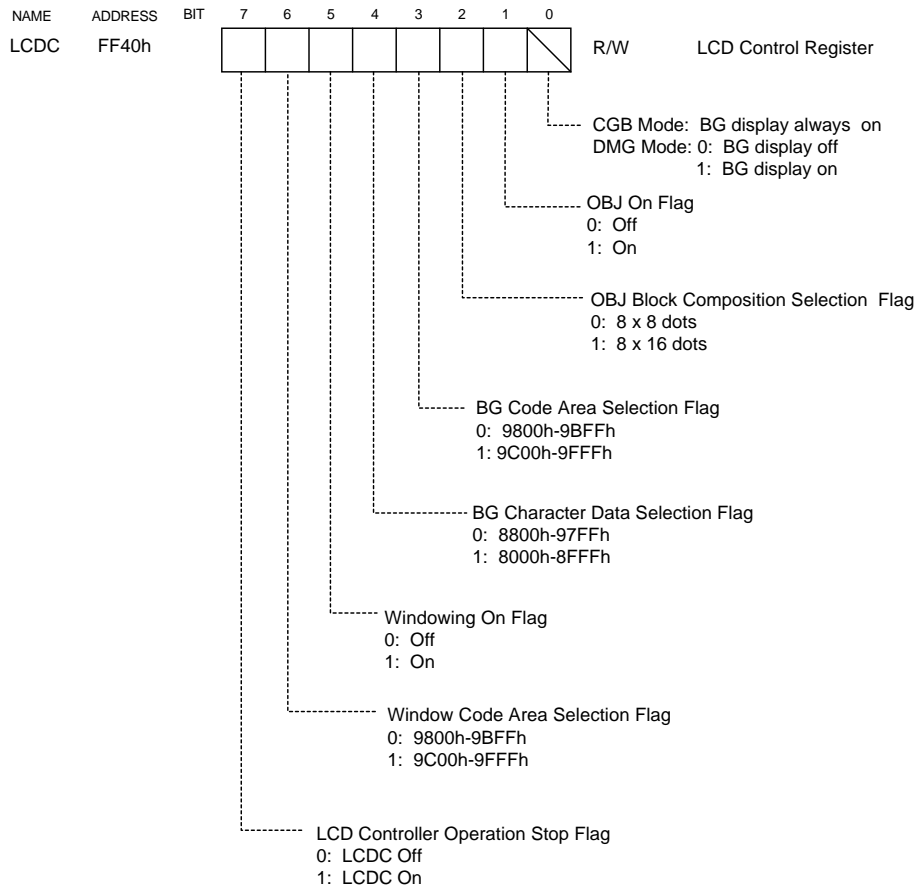
OBJ character data are displayed in the window in the same way as the BG screen.



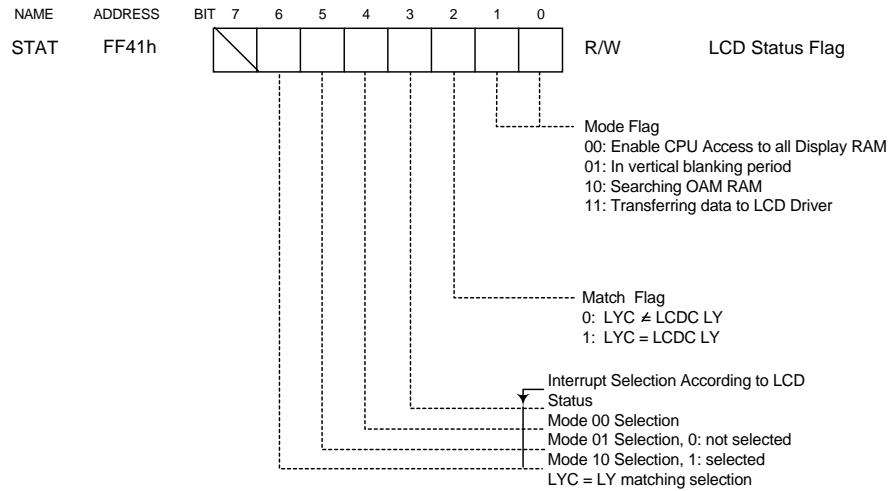
### ➤ Screen Timing



## 1.6 LCD Display Registers



\* In CGB, the liquid crystal protection circuit functions when the LCDC is turned on. Consequently, a white screen is displayed for up to 2 frames. In DMG, the LCDC should be off during vertical blanking periods.



STAT indicates the current status of the LCD controller.

Mode 00: A flag value of 1 represents a horizontal blanking period and means that the CPU has access to display RAM (8000h-9FFFh).

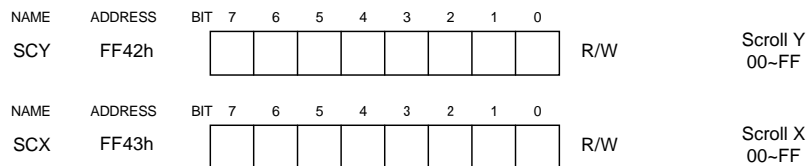
When the value of the flag is 0, display RAM is in use by the LCD controller.

Mode 01: A flag value of 1 indicates a vertical blanking period and means that the CPU has access (approximately 1 ms) to display RAM (8000h-9FFFh).

Mode 10: A flag value of 1 means that OAM (FE00h-FE90h) is being used by the LCD controller and is inaccessible by the CPU.

Mode 11: A flag value of 1 means that the LCD controller is using OAM (FE00h-FE90h) and display RAM (8000h-9FFFh). The CPU cannot access either of these areas.

In addition, the register allows selection of 1 of the 4 types of interrupts from the LCD controller. Executing a write instruction for the match flag resets that flag but does not change the mode flag.



Changing the values of SCY and SCX scrolls the BG screen vertically and horizontally one bit at a time.

NAME	ADDRESS	BIT	7	6	5	4	3	2	1	0		
LY	FF44h										R	LCDC y-coordinate

LY indicates which line of data is currently being transferred to the LCD driver. LY takes a value of 0-153, with 144-153 representing the vertical blanking period.

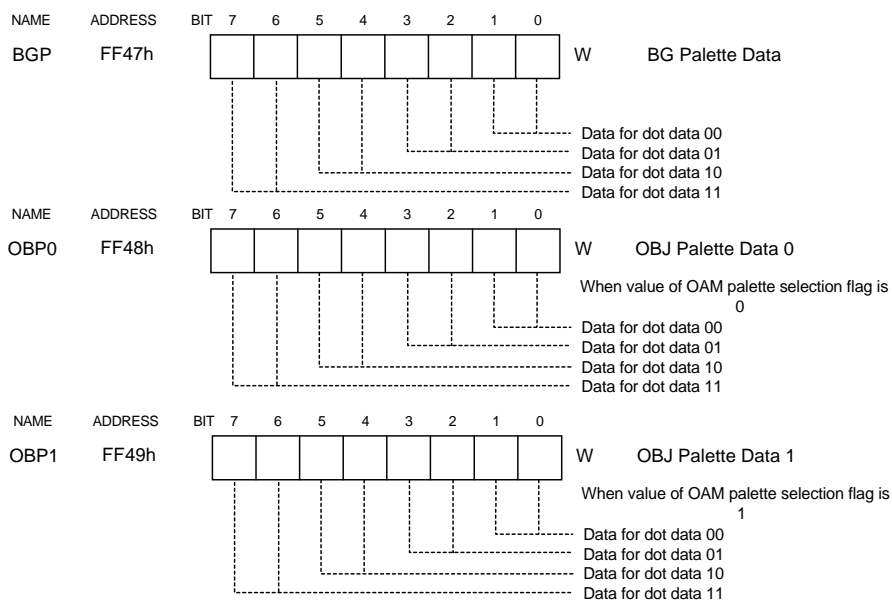
When the value of bit 7 of the LCDC register is 1, writing 1 to this again does not change the value of register LY.

Writing a value of 0 to bit 7 of the LCDC register when its value is 1 stops the LCD controller, and the value of register LY immediately becomes 0. (Note: Values should not be written to the register during screen display.)

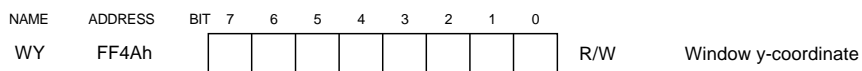
NAME	ADDRESS	BIT	7	6	5	4	3	2	1	0		
LYC	FF45h										R/W	LY Compare

Register LYC is compared with register LY. If they match, the Matchflag of the STAT register is set.

**Note:** *The following 3 registers (BGP, OBP0, and OBP1) are valid in DMG and CGB modes. For information of CGB color palette settings, see section 2, LCD Color Display.*

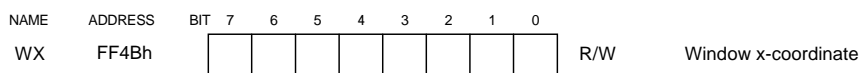


The grayscales (2 bit) for the character dot data are converted by the palette data (BG: register BGP; OBJ: OBP0 or OBP1) and output to the LCD driver as data representing 4 shades (including transparent).



$$0 \leq WY \leq 143$$

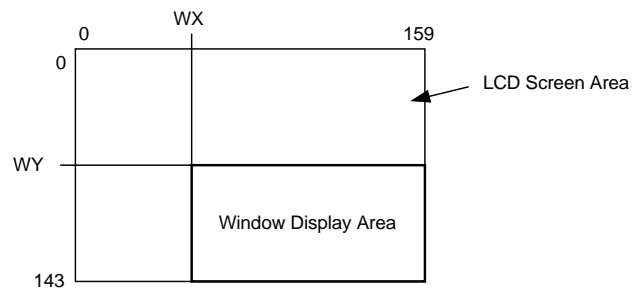
With WY = 0, the window is displayed from the top edge of the LCD screen.



$$7 \leq WX \leq 166$$

With WX = 7, the window is displayed from the left edge of the LCD screen.

Values of 0-6 should not be specified for WX.



OBJ characters are displayed in the same manner in the window as on BG.

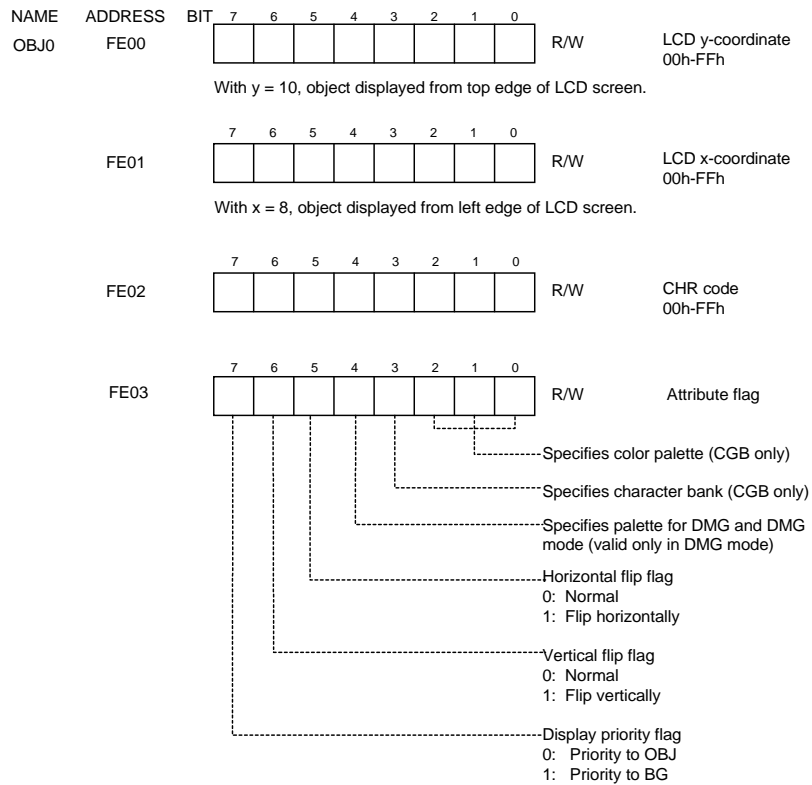


## **1.7 OAM Registers**

### **OBJ (Object)**

- Data for 40 objects (OBJ) can be loaded into internal OAM RAM in the CPU (FE00h-FE9Fh), and 40 objects can be displayed to the LCD. Up to 10 objects can be displayed on the same Y line.
- Each object consists of a y-coordinate (8 bits), x-coordinate (8 bits), and CHR code (8 bits) and specifications for BG and OBJ display priority (1 bit), vertical flip (1bit), horizontal flip (1 bit), DMG-mode palette, (1 bit), character bank (1bit), and color palette (3 bits), for a total of 32 bits.
- An 8 x 8- or 8 x 16-bit block composition can be specified for an OBJ using bit 2 of the LCDC register. With an 8 x 16-bit composition, the CHR code is specfied as an even number, as in DMG.

## OAM Register



OBJ1-OBJ30 have the same composition as OBJ0.

**Note:** In DMG mode, the lower 4 bits of the attribute flag are invalid; only the flags in the upper 4 bits starting from the palette flag are valid.

## 1.8 DMA Registers

### 1.8.1 DMA Transfers in DMG

DMA transfers of 40 x 32 bits of data can be performed from the RAM area (8000h-DFFFh) to OAM (FE00h-FE9Fh). The transfer time is 160  $\mu$ s.

Note that in DMG, data cannot be transferred by DMA from RAM area 0000h-7FFFh.

The starting address of a DMA transfer can be specified as 8000h-DFFFh in increments of 100h.

Note that the method used for transfers from 8000h-9FFFh (display RAM) is different from that used for transfers from other addresses.

#### Example 1

The following example shows how to perform a DMA transfer of 40 x 32 bits from the expansion RAM area (C000h-C09Fh) to OAM (FE00h-FE9Fh).

During DMA, the CPU is run using the internal RAM area (FF80h-FFFEh) to prevent external bus conflicts.

1. The program writes the following instructions to internal RAM (FF80h-FFFEh):

Address	Machine Code	Label	Instruction	Comment
FF80	3E C0		LD A, 0C0H	
	E0 46		LD (DMA), A;C000-C09F→OAM	
	3E 28		LD A, 40	;160-cycle wait
	3D	L1:	DEC A	
	20 FD		JR NZ, L1	
	C9		RET	

2. Example of program that writes the above instructions to internal RAM starting from 0xFF80:

	Label	Instruction
	LD	C, 80H
	LD	B, 10
	LD	HL, DMADATA
L2:	LD	A, (HL)
	LD	(C), A
	INC	C
	DEC	B
	JR	NZ, L2
	•	
	•	
	•	
DMADATA	DB	3EH, 0C0H, 0E0H, 46H, 3EH
	DB	28H, 3DH, 20H, 0FDH, 0C9H

- 3) When the DMA transfer is performed, the subroutine written to internal RAM shown in 1) above is executed:

```
.
CALL 0FF80h      ;DMA transfer
.
```

**Note:** *The preceding program is used for DMA transfers performed within routines for processing interrupts implemented by vertical blanking. In all other cases, however, the program written to internal RAM should be as shown below to prevent interrupts during a transfer.*

Address	Command	Label	Instruction	Comment
FF80	F3		DI	;Interrupt disabled
	3E C0		LD A, 0C0H	
	E0 46		LD (DMA), A	;C000-C09F.OAM
	3E 28		LD A, 40	;160-cycle wait
	3D	L1:	DEC A	
	20 FD		JR NZ, L1	
	FB		EI	;Interrupt enabled
	C9		RET	

#### Example 2

The example below shows a DMA transfer of 40 x 32 bits of data from the display RAM area (9F00h-9F9Fh) to OAM (FE00-FE9Hh).

Machine Code	Label	Instruction	Comment
3E 9F		LD A, 9FH	
E0 46		LD (DMA), A	;9F00-9F9F.OAM

Data can be transferred by DMA from 8000h-9F9Fh to OAM either by the method shown in Example 1 or by using only the above instructions.

### 1.8.2 DMA Transfers in CGB

#### Using the Earlier DMA Transfer Method

This DMA method transfers only 40 x 32 bits of data from 0-DFFFh to OAM (FE00h-FE9Fh). The transfer starting address can be specified as 0-DFFFh in increments of 100h. The transfer method is the same as that used in DMG, but when data are transferred from 8000h-9FFFh (LCD display RAM area), the data transferred are those in the bank specified by bit 0 of register VBK. When transferring data from D000h-DFFFh (unit working RAM area), the data transferred are those in the bank specified by the lower 3 bits of register SVBK.

**Note:** *When the CPU is operating at double-speed, the transfer rate is also doubled.*

#### Using the New DMA Transfer Method

The DMA transfer method provided for DMG has been augmented in CGB with the following DMA transfer functions.

##### 1. Horizontal Blanking DMA Transfer

Sixteen bytes of data are automatically transferred from the user program area (0-7FFFh) and external and unit working RAM area (A000h-DFFFh) to the LCD display RAM area (8000h-9FFFh) during each horizontal blanking period. The number of lines transferred by DMA in a horizontal blanking period can be specified as 1-128 by setting register HDMA5. CPU processing is halted during a DMA transfer period.

##### 2. General-Purpose DMA Transfers

Between 16 and 2048 bytes (specified in 16-byte increments) are transferred from the user program area (0-7FFFh) and external and unit working RAM area (A000h-DFFFh) to the LCD display RAM area (8000h-9FFFh). As with horizontal blanking DMA transfers, CPU operation is halted during the DMA transfer period.

The unit working RAM area (D000h-DFFFh) selected as the transfer source is the bank specified by register SVBK.

The LCD display RAM area (8000h-9FFFh) selected as the transfer destination is the bank specified by register VBK.

Special Notes

- The number of bytes transferred by the new DMA method must be specified in 16-byte increments; byte counts that are not a multiple of 16 cannot be transferred.
- With the new DMA transfer method, transfers are performed at a fixed rate regardless of whether the CPU is set to operate at normal or double-speed.
- Horizontal blanking DMAs should always be started with the LCDC on and the STAT mode set to a value other than 00.
- General-purpose DMAs should be performed with the LCDC off or during a vertical blanking period.
- When the new DMA transfer method is used to transfer data from the user program area (0-7FFFh), mask ROM and MBC for double-speed mode are required.

1.8.3 DMA Control Register: DMG and CGB

NAME	ADDRESS	BIT	7	6	5	4	3	2	1	0		
DMA	FF46h										W	DMA Transfer and Starting Address

**1.8.4 New DMA Control Registers: CGB only**

NAME	ADDRESS	BIT	7	6	5	4	3	2	1	0	
HDMA1	FF51										W Specifies higher-order transfer source address 00h-7Fh(program ROM) A0h-DFh (external and unit working RAM)
HDMA2	FF52										W Specifies lower-order transfer source address 0Xh-FXh  Combined with HDMA1, specifies the upper 12 bits of the transfer source area (0x000X-0x7FFX or 0xA00X-0xDFFX)
HDMA3	FF53										W Specifies higher-order transfer destination address 00h-1Fh
HDMA4	FF54										W Specifies lower-order transfer destination address 0Xh-FXh  Combined with HDMA3, specifies the upper 12 bits of the transfer destination area (800Xh-9FFXh)
HDMA5	FF55										R/W Transfer start and number of bytes to transfer
<div style="display: flex; justify-content: space-between; align-items: flex-start;"> <div style="width: 40%;"> <p>(n)</p> <p>Horizontal blanking DMA No. of lines to transfer = (n + 1) Total no. of bytes to transfer = 16 x (n+1) (Max = 2,048 bytes)</p> <p>General-purpose DMA Total no. of bytes transfer = 16 x (n + 1) (Max = 2,048)</p> </div> <div style="width: 55%;"> <p>Value of 1 written: After 1 is written, horizontal blanking DMA transfer is started from the first horizontal blanking period. (DMA should always be started with LCDC on and value other than 00 for STAT mode.)</p> <p>* When a value of 0 has been subsequently written, DMA transfer stops beginning with the next horizontal blanking period.</p> </div> </div> <div style="margin-top: 10px;"> <p>Value of 0 written (the following applies only when the bit is already 0): General-purpose DMA starts (DMA should be started with LCDC off or during a horizontal blanking period. Ensure that the transfer period does not overlap with STATE mode settings of 10 or 11.)</p> <p>* Only input of a reset signal can halt a general-purpose DMA transfer in progress.</p> </div>											
<div style="display: flex; justify-content: space-between;"> <div style="width: 30%;">Horizontal blanking DMA</div> <div style="width: 65%;"></div> </div> <div style="margin-top: 10px;"> <div style="display: flex; justify-content: space-between;"> <div style="width: 30%;">General-purpose DMA</div> <div style="width: 65%;"></div> </div> </div>											

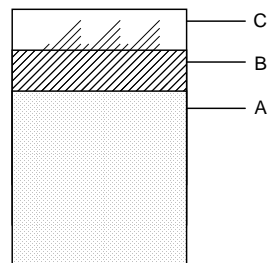
**Note:** These registers cannot be written to in DMG mode.



## 1.9 OBJ Display Priority

As a rule, when objects overlap, the one with the lower OBJ number is given priority. In DMG or CGB in DMG mode, among overlapping objects with different x-coordinates, priority is given to the object with the smallest x-coordinate.

1. Same x-coordinate: For both DMG and CGB



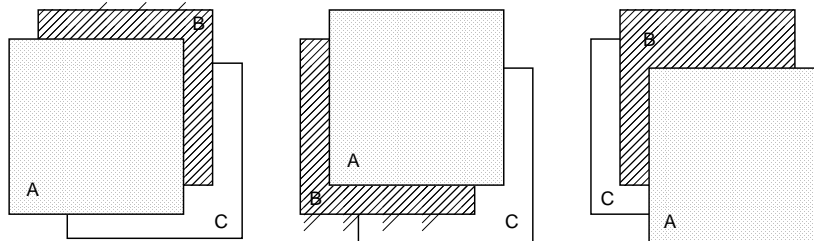
a = No. of OBJ A  
b = No. of OBJ B  
c = No. of OBJ C

When  $a < b < c$ , objects are displayed as indicated in the figure at left.

2. Different x-coordinates: CGB Only

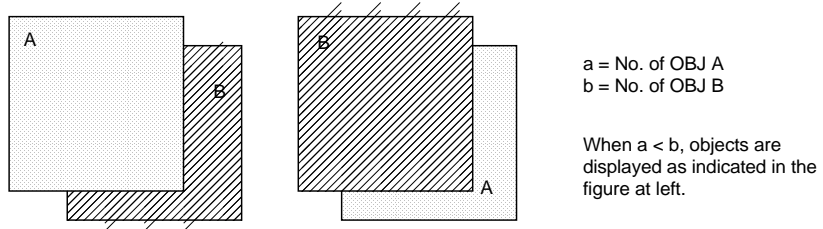
a = No. of OBJ A  
b = No. of OBJ B  
c = No. of OBJ C

When  $a < b < c$ , objects are displayed as indicated in the figures below.



3. Different x-coordinates: DMG/CGB in DMG Mode

In DMG mode and with objects with different x-coordinates, the object with the smallest x-coordinate is given priority.

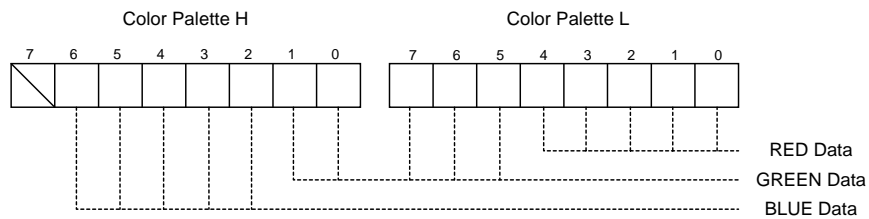


## 2 LCD COLOR DISPLAY (CGB ONLY)

The LCD unit of the CGB system can display 32 shades each for RGB, for a total 32,768 colors. A single color palette consists of 4 colors selected from among these 32,768. One of 8 palettes can be selected for each BG and OBJ character. However, because each OBJ includes transparent data, each OBJ color palette consists of 3 colors. The color palettes for BG and OBJ are independent of one another.

### 2.1 Color Palettes

- Eight palettes each are provided for BG and OBJ.
- Each palette consists of 4 colors and is specified by the display dot data (2 bits) (palette data nos. 0-3).
- The color palettes represent each color with 2 bytes, with 5 bits of data for each color of RGB (32,768 displayable colors).



## 2.2 Color Palette Composition

### 1. BG Color Palettes

Color Palette No.			Palette Data No.
Color palette 0	Color palette H00	Color palette L00	0
	Color palette H01	Color palette L01	1
	Color palette H02	Color palette L02	2
	Color palette H03	Color palette L03	3
Color palettes 1-7			

### 2. OBJ Color Palettes

OBJ color palettes have the same composition as shown in the previous figure.

## 2.3 Writing Data to a Color Palette

Data are written to color palettes using the write-specification and write-data registers. The lower 6 bits of the write-specification register specifies the write address. Data are written to the write-data register, at the address specified by the write-specification register. If the most significant bit of the write-specification register is set to 1, the write address is then automatically incremented to specify the next address. (The next address is read from the lower 6 bits of the write-specification register.)

The write-specification and write-data registers also are used to read data from color palettes. Data are read from the write-data register, and the read data are those at the address specified by the write-specification register. When data are read, the specified address is not incremented even if the most-significant bit of the write-specification register is set to 1.

NAME	ADDRESS	BIT	
BCPS	FF68	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	R/W Specifies a BG write
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies H/L (H: 1, L: 0)
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies the palette data no.
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies the palette no.
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	1: With each write, specifies the next palette
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	0: Values of bits 0-5 fixed
BCPD	FF69	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	R/W Specifies the BG write data
OCPS	FF6A	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	R/W Specifies the OBJ write data
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies H/L (H: 1; L: 0)
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies the palette data no.
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	Specifies the palette no.
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	1: With each write, specifies the next palette
		<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	0: Values of bits 0-5 fixed
OCPD	FF6B	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	R/W OBJ write data

**Note:** These registers cannot be written to in DMG mode.

## 2.4 Overlapping OBJ and BG

When objects are displayed, overlapping objects and background are displayed according to the display priority flags for OBJ and BG, as indicated below. The BG display priority flag can be used to assign BG display priority to individual characters.

Display Priority Flag		Dot Data		Screen Display	
BG	OBJ	OBJ	BG	Palette	Data
0:  Use OBJ Priority	0: Priority to OBJ	00	00	BG	00
		00	bg	BG	bg
		obj	00	OBJ	obj
		obj	bg	OBJ	obj
	1: Priority to BG	00	00	BG	00
		00	bg	BG	bg
		obj	00	OBJ	obj
		obj	bg	BG	bg
1: Highest Priority to BG (by character)	0  1	00	00	BG	00
		00	bg	BG	bg
		obj	00	OBJ	obj
		obj	bg	BG	bg

**Note:** *obj and bg represent dot data (01, 10, 11) for OBJ and BG, respectively.*

## **2.5 Display Using Earlier DMG Software (DMG mode)**

When earlier DMG software is used, coloring is performed automatically by the system using registers BGP, OBP0, and OBP1. However, the display uses 3 palettes, 1 for BG, with 4 colors, and 2 for OBJ, each with 3 colors (excluding transparent; maximum of 10 colors in 1 screen).

### **1. BG Display**

Colors specified in BG color palette No. 0 are displayed by the dot data (2 bits) whose grayscales are specified by register BGP.

### **2. OBJ Display**

Colors specified in OBJ color palettes No. 0 and No. 1 are displayed by the dot data (2 bits) whose grayscales are specified by registers OBP0 and OBP1.

The CGB hardware automatically selects the display color according to the color palette pre-registered in the CGB (cannot be changed by a program). However, when turning on power to the CGB, the player can select from a combination of the 12 colors registered in the unit. This function is available only in DMB mode.

**CHAPTER 3: SOUND FUNCTIONS.....79**  
    **Revision History..... 76**  
    **1. OVERVIEW OF SOUND FUNCTIONS.....79**  
    **2. SOUND CONTROL REGISTERS.....81**  
        *2.1 Sound 1 Mode Registers .....81*  
        *2.2 Sound 2 Mode Registers .....84*  
        *2.3 Sound 3 Mode Registers .....85*  
        *2.4 Sound 4 Mode Registers .....87*  
        *2.5 Sound Control Registers.....90*  
    **3. VIN TERMINAL USAGE NOTES.....91**



## Revision History

Date	Section	Description
12/3/99	1	Addition of "Usage Notes"
12/3/99	2.1 2.2 2.4	Revision and addition of Notes in Mode Registers for Sound 1, 2, 4

## CHAPTER 3: SOUND FUNCTIONS

### 1. OVERVIEW OF SOUND FUNCTIONS

The sound circuitry consists of circuits that generate 4 types of sounds (Sounds 1-4). It can also synthesize external audio input waveforms and output sounds. (External audio input is a function available only in CGB).

Sound 1: Generates a rectangle waveform with sweep and envelope functions.

Sound 2: Generates a rectangle waveform with an envelope function.

Sound 3: Outputs any waveform from waveform RAM.

Sound 4: Generates white noise with an envelope function.

Each sound has two modes, ON and OFF.

- ◆ ON Mode

Sounds are output according to data in the mode register for each sound.

The mode register data can be specified as needed while outputting sound.

- ◆ Initialization Flag

When the default envelope values are set and the length counter is restarted, the initialization flag is set to 1 and the data is initialized.

- ◆ Mute

In the following instances, the synthesizer will enter mute status. No sound will be output regardless of the ON flag setting.

Sounds 1, 2, and 4:

- When the output level is 0 with the default envelope value set to a value other than 0000 and in DOWN mode

- When the step is 0 with the default envelope value set to a value of 0000 and in UP mode (NR12, NR22, and NR42 set to 0x08 and the initialization flag set)

Sound 3:

With the output level set to mute  
(bits 5 and 6 of NR32 set to 0)

- ◆ Stop Status

In the following cases, the ON flag is reset and sound output is halted.

- Sound output is halted by the length counter.

- With Sound 1, during a sweep operation, an overflow occurs in addition mode.

◆ OFF Mode

Stops operation of the frequency counter and D/A converter and halts sound output.

◆ Sounds 1, 2, and 4:

-When the default level is set to 0000 with the envelope in DOWN mode (initialization not required)

◆ Sound 3:

-When the Sound OFF flag (bit 7 of NR30) is set to 0.

Setting the Sound OFF flag to 1 cancels OFF mode.

Sound 3 is started by re-initialization.

◆ All Sounds OFF mode

-Setting the All Sounds ON/OFF flag (bit 7 of NR52) to 0 resets all of the mode registers (for sounds 1, 2, 3, and 4) and halts sound output. Setting the All Sounds ON/OFF flag to 1 cancels All Sounds OFF mode.

**Note:** *The sound mode registers should always be set after All Sound OFF mode is cancelled. The sound mode registers cannot be set in All Sound OFF mode.*

◆ Sound Usage Notes

Use one of the following methods to halt sounds 1, 2, or 4.

1) Use NR51.

2) Set NR12, NR22, and NR42 to 0x08.

3) Set NR14, NR24, and NR44 to 0x80.

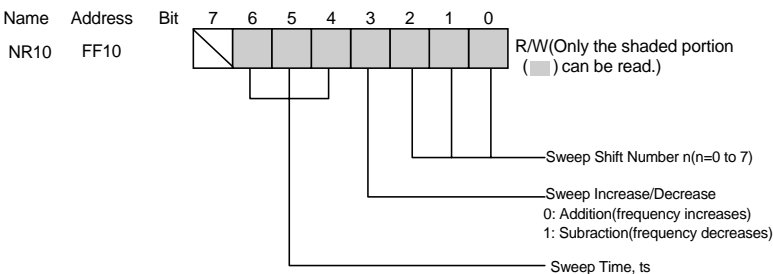
Switch to OFF mode during the scene you stop the BGM. Unless you switch to OFF mode, a faint whining noise can be heard due to the sound circuit structure.

## 2. SOUND CONTROL REGISTERS

### 2.1 Sound 1 Mode Registers

Sound 1 is a circuit that generates a rectangle waveform with sweep and envelope functions. It is set by registers NR10, NR11, NR12, NR13, and NR14.

◆ Sweep Shift Number



◆ Sweep Shift Number

The frequency with one shift (NR13 and NR14) is determined by the following formula.  
$$X(t) = X(t-1) + X(t-1) / 2 \quad n = 0 \text{ to } 7$$
$$X(0) = \text{default data} \quad X(t-1) \text{ is the previous output frequency}$$

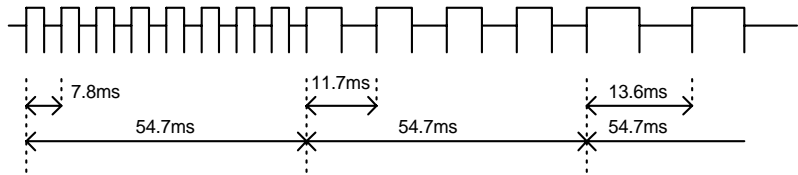
If the result of this formula is a value consisting of more than 11 bits, sound output is stopped and the Sound 1 ON flag of NR52 (bit 0) is reset.  
In a subtraction operation, if the subtrahend is less than 0, the result is the pre-calculation value  $X(t) = X(t-1)$ . However, if  $n = 0$ , shifting does not occur and the frequency is unchanged.

◆ Sweep time (ts)

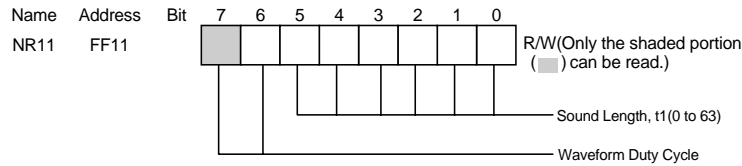
Frequency varies with each value of ts.  
000: Sweep OFF  
001:  $ts=1/f_{128}$  (7.8ms)  
010:  $ts=2/f_{128}$  (15.6ms)  
011:  $ts=3/f_{128}$  (23.4ms)  
100:  $ts=4/f_{128}$  (31.3ms)  
101:  $ts=5/f_{128}$  (39.1ms)  
110:  $ts=6/f_{128}$  (46.9ms)  
111:  $ts=7/f_{128}$  (54.7ms)

$f_{128}=128\text{Hz}$

Example: When NR10 = 79h and the default frequency = 400h, the sweep waveform appears as follows.



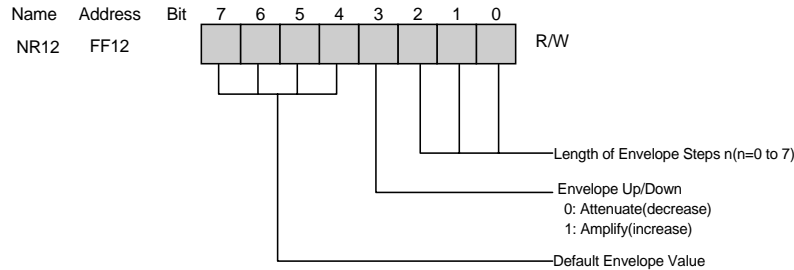
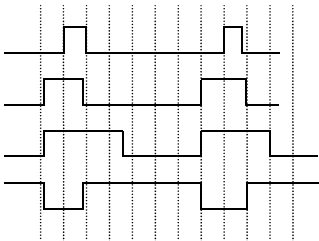
**Note:** When the sweep function is not used, the increase/decrease flag should be set to 1 (subtraction mode).



Sound length = (64 - t1) x (1/256) sec

Waveform Duty Cycles

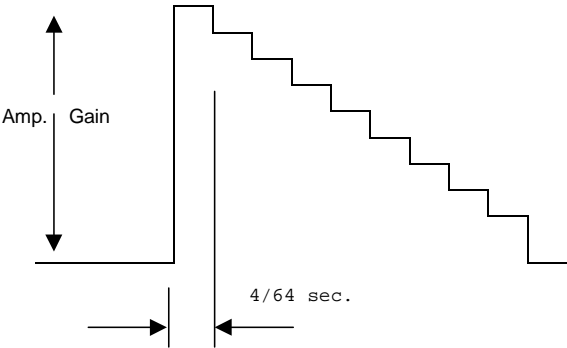
- 00 : 12.5%
- 01 : 25%
- 10 : 50%
- 11 : 75%



Length of Envelope Steps:  
Sets the length of each step of envelope amplification or attenuation.  
Length of 1 step = N x (1/64) sec  
When N = 0, the envelope function is stopped.

Default Envelope Value (0000 to 1111B):  
16 step levels can be specified using the 4-bit D/A circuit.  
Maximum is 1111B, and 0000 is the mute setting.

Example: When NR12 = 94h, the Amp Gain is as follows.



**Note:** By Setting the envelope register only nothing will be reflected in the output. Always set the initial flag.

Name	Address	Bit	7	6	5	4	3	2	1	0	
NR13	FF13										R/W(Low-order Frequency Data)

Name	Address	Bit	7	6	5	4	3	2	1	0	
NR14	FF14										R/W(Only the shaded portion ( ) can be read.)
											High-order Frequency Data(3 bits)
											Counter Continuous Selection
											Initialize

Counter/Continuous Selection

- 0: Outputs continuous sound regardless of length data in register NR11.  
1: Outputs sound for the duration specified by the length data in register NR11.  
When sound output is finished, bit 0 of register NR52, the Sound 1 ON flag, is reset.

Initialize

Setting this bit to 1 restarts Sound 1.  
With the 11-bit frequency data specified in NR13 and NR14 represented by x, the frequency, f, is determined by the following formula.  
 $f = 4194304 / (4 \times 2 \times (2048 - X))$  Hz  
Thus, the minimum frequency is 64 Hz and the maximum is 131.1 KHz.

◆ Sound 1 Usage Notes

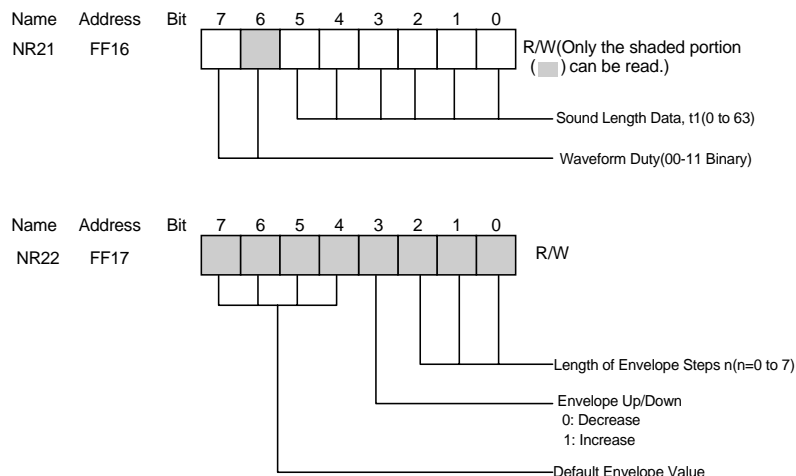
-When no sweep function is used with Sound 1, the sweep time should be set to 0 (sweep OFF). In addition, either the sweep increase/decrease flag should be set to 1 or the sweep shift number set to 0 (set to 08h-0Fh or 00h in NR10).

-Sound may not be produced if the sweep increase/decrease flag of NR10 is set to 0 (addition mode), the sweep shift number set to a value other than 0, and the mode set to sweep OFF (e.g. NR10 = 01h)

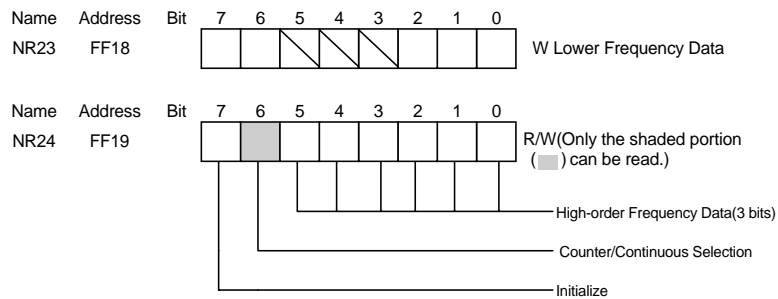
-When a value is written in the envelope register, the sound output becomes unstable till the initial flag is set. Therefore, set the initial flag immediately after writing a value in the envelope register.

## 2.2 Sound 2 Mode Registers

Sound 2 is a circuit that generates a rectangle waveform with an envelope function. It is set by registers NR21, NR22, NR23, and NR24.



**Note: By Setting the envelope register only nothing will be reflected in the output. Always set the initial flag.**



#### Counter/Continuous Selection

- 0: Outputs continuous sound regardless of length data in register NR21.  
 1: Outputs sound for the duration specified by the length data in register NR21. When sound output is finished, bit 1 of register NR52, the Sound 2 ON flag, is reset.

#### Initialize

Setting this bit to 1 restarts Sound 2.

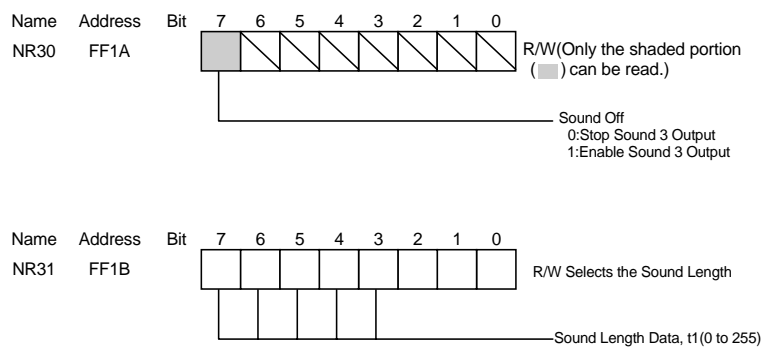
#### ◆ Sound 2 Usage Notes

When a value is written in the envelope register, the sound output becomes unstable until the initial flag is set. Hence, set the initial flag immediately after writing a value in the envelope register.

### 2.3 Sound 3 Mode Registers

Sound 3 is a circuit that generates user-defined waveforms. It automatically reads a waveform pattern (1 cycle) written to waveform RAM at FF30h-FF3Fh, and it can output a sound while changing its length, frequency, and level by registers NR30, NR31, NR32, NR33, and NR34.

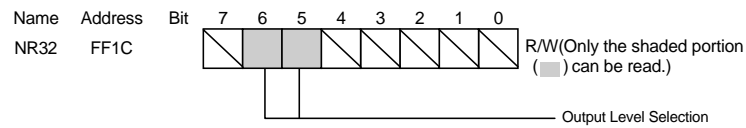
The settings of the sound length and frequency functions and data are the same as for the Sound 1 circuit.



$$\text{Sound Length} = (256 - t1) \times (1/256) \text{ sec}$$



## Game Boy Programming Manual



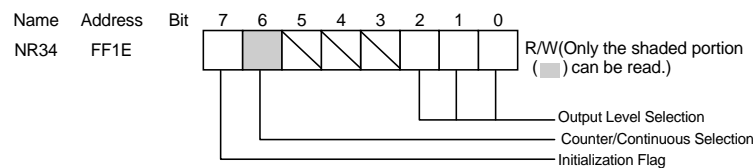
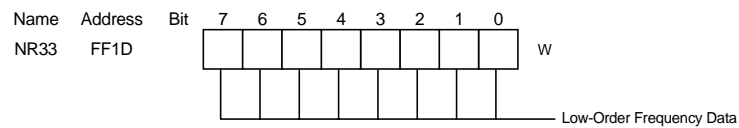
### Output Level:

00: Mute

01: Output waveform RAM data (4-bit length) unmodified.

10: Output waveform RAM data (4-bit length) shifted 1 bit to the right (1/2).

11: Output waveform RAM data (4-bit length) shifted 2 bits to the right (1/4).



### Counter/Continuous Selection

0: Outputs continuous sound regardless of length data in register NR31.

1: Outputs sound for the duration specified by the length data in register NR31.

When sound output is finished, bit 2 of register NR52, the Sound 3 ON flag, is reset.

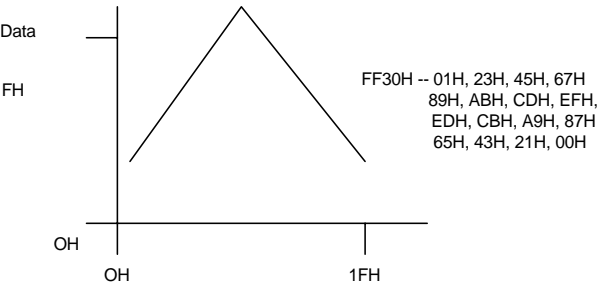
### Initialization Flag

When the Sound OFF flag (bit 7, NR30) is set to 1, setting this bit to 1 restarts Sound 3.

- ◆ Sound 3 Usage Notes
  - The initialization flag should not be set when the frequency is changed during Sound 3 output.
  - Setting the initialization flag during Sound 3 operation (Sound 3 ON flag = 1) may destroy the contents of waveform RAM.
  - Setting the initialization flags for Sound 1, Sound 2, or Sound 4 does not cause a problem.
- ◆ Waveform RAM Composition
  - Waveform RAM consists of waveform patterns of 4 bits x 32 steps.

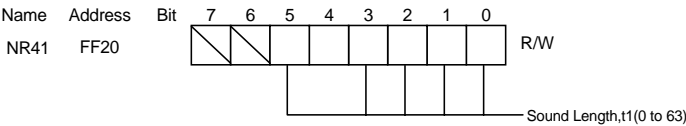
Address	D7	D6	D5	D4	D3	D2	D1	D0
FF30		Step 0				Step 1		
FF31		Step 2				Step 3		
FF32		Step 4				Step 5		
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
FF3F		Step 30				Step 31		

Example: Triangular Wave

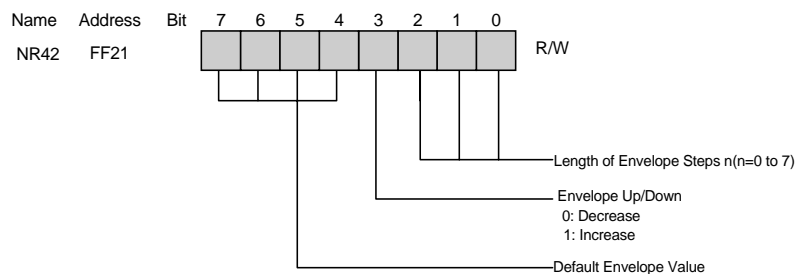


2.4 Sound 4 Mode Registers

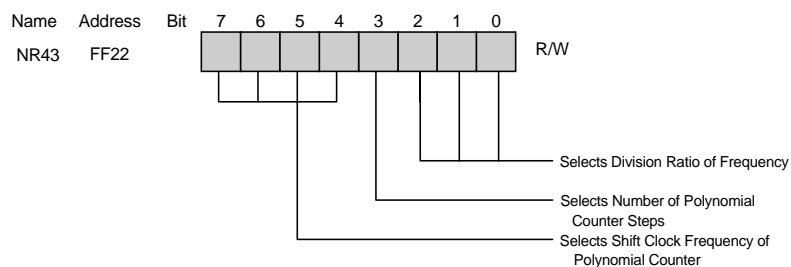
Sound 4 is a white-noise generating circuit. It can output sound while switching the number of steps of the polynomial counter for random number generation and changing the frequency dividing ratio and envelope data by registers NR41, NR42, NR43, and NR44.



## Game Boy Programming Manual



**Note:** By only setting the envelope register nothing will be reflected in the output. Always set the initial flag.



Selecting the dividing ratio of the frequency:

Selects a 14-step prescaler input clock to produce the shift clock for the polynomial counter.

000 :  $fx1/2^3 \times 2$

001 :  $fx1/2^3 \times 1$

010 :  $fx1/2^3 \times 1/2$

011 :  $fx1/2^3 \times 1/3$

100 :  $fx1/2^3 \times 1/4$

101 :  $fx1/2^3 \times 1/5$

110 :  $fx1/2^3 \times 1/6$

111 :  $fx1/2^3 \times 1/7$

$f=4/19430\text{MHz}$

Selecting the number of steps for the polynomial counter:

0: 15 steps

1: 7 steps

Selecting the shift clock frequency of the polynomial counter:

0000: Dividing ratio frequency  $\times 1/2$

0001: Dividing ratio frequency  $\times 1/2^2$

0010: Dividing ratio frequency  $\times 1/2^3$

0011: Dividing ratio frequency  $\times 1/2^4$

:

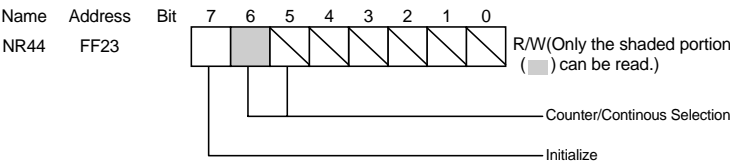
:

:

1101: Dividing ratio frequency  $\times 1/2^4$

1110: Prohibited code

1111: Prohibited code

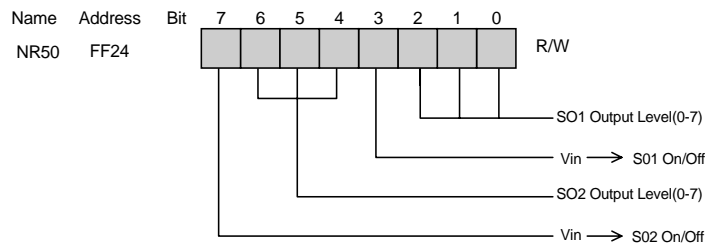


Counter/Continuous Selection:  
0: Outputs continuous sound regardless of length data in register NR41.  
1: Outputs sound for the duration specified by the length data in register NR41. When sound output is finished, bit 3 of register NR52, the Sound 4 ON flag, is reset.

Initialize:  
Setting this bit to 1 restarts Sound 4.

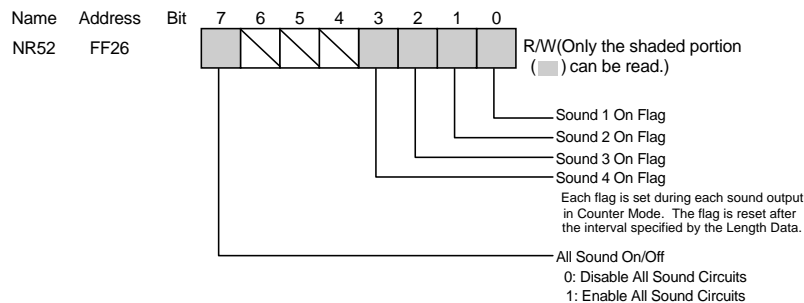
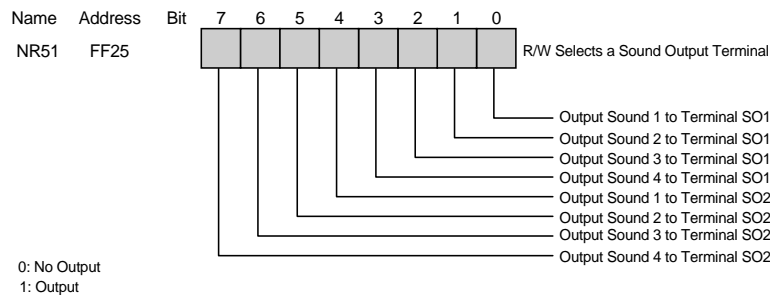
◆ Sound 4 Usage Notes  
When a value is written in the envelope register, the sound output becomes unstable until the initial flag is set. Hence, set the initial flag immediately after writing a value in the envelope register.

2.5 Sound Control Registers



Output Level:  
000: Minimum level (Maximum level 8)  
:  
:  
111: Maximum level

V i n→SO1 ON/OFF (V i nSO2→ON/OFF)  
Synthesizes audio input from Vin terminal with sounds 1-4 and ouputs the result.  
0: No output  
1: Output



### 3. VIN TERMINAL USAGE NOTES

- The VIN terminal can be used normally only in CGB. (Since the signal from the VIN terminal is too low to be used, the VIN terminal cannot be used in DMG.)
- The maximum amplitude of the synthesized output is 3V.
- The design prevents the maximum amplitude from exceeding 3V when only sounds 1-4 are used, even when the output level for each sound is set to the maximum.

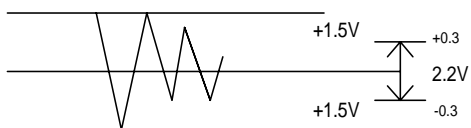
When the output level is set to 0Fh, each sound is output at 0.75V.

$$0.75V \times 4 = 3V$$

- The maximum amplitude of the synthesized sound output also must be limited to 3V or less when the VIN terminal is used to input external sound.

Example: Using Sounds 1-4 and the VIN terminal

Use software to adjust the output levels of sounds 1-4 so that they do not exceed 0.6V (3V ÷ 5). Also limit the output level of the VIN terminal to 0.6V or less (input range of 1.9 - 2.5V).



- The input voltage from the VIN terminal also can be increased if the levels of the internal sounds are low or if not all 4 sounds are used (total output level of 3V or less).

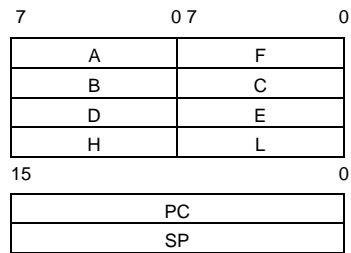
***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

<b>CHAPTER 4: CPU INSTRUCTION SET.....</b>	<b>94</b>
<b>1. GENERAL PURPOSE REGISTERS.....</b>	<b>94</b>
<b>2. DESCRIPTION OF INSTRUCTIONS.....</b>	<b>95</b>
2.1 8-Bit Transfer and Input/Output Instructions.....	95
2.2 16-Bit Transfer Instructions .....	100
2.3 8-Bit Arithmetic and Logical Operation Instructions .....	102
2.4 16-Bit Arithmetic Operation Instructions .....	107
2.5 Rotate Shift Instructions .....	109
2.6 Bit Operations .....	114
2.7 Jump Instructions.....	116
2.8 Call and Return Instructions .....	118
2.9 General-Purpose Arithmetic Operations/CPU Control Instructions....	122

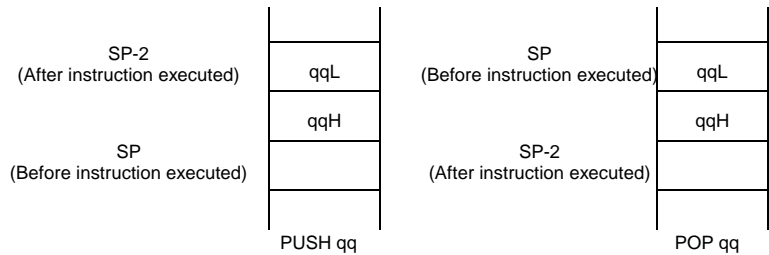


CHAPTER 4: CPU INSTRUCTION SET

1. GENERAL PURPOSE REGISTERS



- Accumulator: A  
An 8-bit register for storing data and the results of arithmetic and logical operations.
- Auxiliary registers: B, C, D, E, F, H, and L  
These serve as auxiliary registers to the accumulator. As register pairs (BC, DE, HL), they are 8-bit registers that function as data pointers.
- Program counter: PC  
A 16-bit register that holds the address data of the program to be executed next. Usually incremented automatically according to the byte count of the fetched instructions. When an instruction with branching is executed, however, immediate data and register contents are loaded.
- Stack pointer: SP  
A 16-bit register that holds the starting address of the stack area of memory. The contents of the stack pointer are decremented when a subroutine CALL instruction or PUSH instruction is executed or when an interrupt occurs and incremented when a return instruction or pop instruction is executed.



- **Flag Register: F**  
Consists of 4 flags that are set and reset according to the results of instruction execution. Flags CY and Z are tested by various conditional branch instructions.

7	6	5	4	3	2	1	0
Z	N	H	CY				

Z: Set to 1 when the result of an operation is 0; otherwise reset.  
N: Set to 1 following execution of the subtraction instruction, regardless of the result.  
H: Set to 1 when an operation results in carrying from or borrowing to bit 3.  
CY: Set to 1 when an operation results in carrying from or borrowing to bit 7.

## 2. DESCRIPTION OF INSTRUCTIONS

## 2.1 8-Bit Transfer and Input/Output Instructions

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	$r, r'$	$r \leftarrow r'$	--	--	--	--	1	0	1	$r$		$r'$		

Loads the contents of register r' into register r.

Codes for registers  $r$  and  $r'$

Register	r, r'
A	111
B	000
C	001
D	101
E	011
H	100
L	101

Examples: LD A, B ;  $A \leftarrow B$   
LD B, D ;  $B \leftarrow D$

**Game Boy Programming Manual**

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0	
LD	r,	n	r ← n			--	--	--	--	2	00	r			110				
											← n →								

Loads 8-bit immediate data n into register r.

Example: LD B, 24h ; B ← 24h

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	r, (HL)	r ← (HL)				2	01	r	110					

Loads the contents of memory (8 bits) specified by register pair HL into register r.

Example: When (HL) = 5Ch,  
LD H, (HL) ; H ← 5Ch

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(HL), r	(HL) ← r				2	01	110	r					

Stores the contents of register r in memory specified by register pair HL.

Example: When A = 3Ch, HL = 8AC5h  
LD (HL), A ; (8AC5h) ← 3Ch

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(HL), n	(HL) ← n				--	--	--	--	3	00	110	110					
											← n →							

Loads 8-bit immediate data n into memory specified by register pair HL.

Example: When HL = 8AC5h,  
LD (HL), 0 ; 8AC5h ← 0

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (BC)	A ← (BC)				2	00	001	010					

Loads the contents specified by the contents of register pair BC into register A.

Example: When (BC) = 2Fh,  
LD A, (BC) ; A ← 2Fh

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (DE)	A ← (DE)				2	00	011	010					

Loads the contents specified by the contents of register pair DE into register A.

Example: When (DE) = 5Fh,  
LD A, (DE) ; A ← 5Fh

## Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (C)	A ← (FF00H+C)				--	--	--	--	2	11	110	010	

Loads into register A the contents of the internal RAM, port register, or mode register at the address in the range FF00h-FFFFh specified by register C.

Example: When C = 95h,  
LD A, (C) ; A ← contents of (FF95h)

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(C), A	(FF00H+C) ← A				--	--	--	--	2	11	100	010	

Loads the contents of register A in the internal RAM, port register, or mode register at the address in the range FF00h-FFFFh specified by register C.

Example: When C = 9Fh,  
LD (C), A ; (FF9Fh) ← A

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (n)	A ← (n)		--	--	--	--	3	11	110	000			
									← n →					

Loads into register A the contents of the internal RAM, port register, or mode register at the address in the range FF00h-FFFFh specified by the 8-bit immediate operand n.

Note, however, that a 16-bit address should be specified for the mnemonic portion of n, because only the lower-order 8 bits are automatically reflected in the machine language.

Example: To load data at FF34h into register A, type the following.  
LD A, (FF34)

Typing only LD A, (34) would cause the address to be incorrectly interpreted as 0034, resulting in the instruction LD A, (0034) .

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(n), A	(n) ← A		--	--	--	--	3	11	100	000			
									← n →					

Loads the contents of register A to the internal RAM, port register, or mode register at the address in the range FF00h-FFFFh specified by the 8-bit immediate operand n.

Note, however, that a 16-bit address should be specified for the mnemonic portion of n, because only the lower-order 8 bits are automatically reflected in the machine language.

Example: To load the contents of register A in 0xFF34, type the following.  
LD (FF34), A

Typing only LD (34), A would cause the address to be incorrectly interpreted as 0034, resulting in the instruction LD (0034), A .

**Game Boy Programming Manual**

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (nn)	A ← (nn)		--	--	--	--	4	11	111	010			
									← n →					
									← n →					

Loads into register A the contents of the internal RAM or register specified by 16-bit immediate operand nn.

Example: LD A, (FF44h) ; A ← (LY)  
LD A, (8000h) ; A ← (8000h)

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(nn), A	(nn) ← A				--	--	--	--	4	11	101	010	
											← n →			
											← n →			

Loads the contents of register A to the internal RAM or register specified by 16-bit immediate operand nn.

Example: LD (FF44h), A ; (LY) ← A  
LD (8000h), A ; (8000h) ← A

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (HLI)	A ← (HL) HL ← HL+1				--	--	--	--	2	00	101	010	

Loads in register A the contents of memory specified by the contents of register pair HL and simultaneously increments the contents of HL.

Example: When HL = 1FFh and (1FFh) = 56h,  
LD A, (HLI) ; A ← 56h, HL ← 200h

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	A, (HLD)	A ← (HL) HL ← HL-1				--	--	--	--	2	00	111	010	

Loads in register A the contents of memory specified by the contents of register pair HL and simultaneously decrements the contents of HL.

Example: When HL = 8A5Ch and (8A5Ch) = 3Ch,  
LD A, (HLD) ; A ← 3Ch, HL ← 8A5Bh

#### Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(BC), A	(BC) ← A		--	--	--	--	2	00	000	010			

Stores the contents of register A in the memory specified by register pair BC.

Example: When BC = 205Fh and A = 3Fh,  
LD (BC), A ; (205Fh) ← 3Fh

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(DE), A	(DE) ← A		--	--	--	--	2	00	010	010			

Stores the contents of register A in the memory specified by register pair DE.

Example: When DE = 205Ch and A = 00h,  
LD (DE), A ; (205Ch) ← 00h

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(HLI), A	(HL) ← A HL ← HL+1		--	--	--	--	2	00	100	010			

Stores the contents of register A in the memory specified by register pair HL and simultaneously increments the contents of HL.

Example: When HL = FFFFh and A = 56h,  
LD (HLI), A ; (0xFFFF) ← 56h, HL = 0000h

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	(HLD), A	(HL) ← A HL ← HL-1		--	--	--	--	2	00	110	010			

Stores the contents of register A in the memory specified by register pair HL and simultaneously decrements the contents of HL.

Example: HL = 4000h and A = 5h,  
LD (HLD), A ; (4000h) ← 5h, HL = 3FFFh

## 2.2 16-Bit Transfer Instructions

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	dd,	nn	dd ← nn			--	--	--	--	3	00 dd0 001							
											L-ADRS							
											H-ADRS							
											← n →							
											← n →							

Loads 2 bytes of immediate data to register pair dd.

dd codes are as follows:

Register Pair	dd
BC	00
DD	01
HL	10
SP	11

Example: LD HL, 3A5Bh ; H ← 3Ah, L ← 5Bh

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD	SP, HL	SP ← HL				--	--	--	--	2	11	111	001	

Loads the contents of register pair HL in stack pointer SP.

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
PUSH	qq	(SP - 1) ← qqH (SP - 2) ← qqL SP ← SP-2				--	--	--	--	4	11	qq0	101	

Pushes the contents of register pair qq onto the memory stack. First 1 is subtracted from SP and the contents of the higher portion of qq are placed on the stack. The contents of the lower portion of qq are then placed on the stack. The contents of SP are automatically decremented by 2.

qq codes are as follows:

Register Pair	qq
BC	00
DE	01
HL	10
AF	11

Example: When SP = FFFEh,  
PUSH BC ; (FFFCh), (FFFCh) ← B, SP ← FFFCh

## Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
POP	qq	$qq_L \leftarrow (SP)$ $qq_H \leftarrow (SP+1)$ $SP \leftarrow SP+2$		--	--	--	--	3	11	qq0		001		

Pops contents from the memory stack and into register pair qq.

First the contents of memory specified by the contents of SP are loaded in the lower portion of qq. Next, the contents of SP are incremented by 1 and the contents of the memory they specify are loaded in the upper portion of qq. The contents of SP are automatically incremented by 2.

Example: When  $SP = FFFCh$ ,  $(FFFCh) = 5Fh$ , and  $(FFFDh) = 3Ch$ ,  
POP BC ;  $B \leftarrow 3Ch$ ,  $C \leftarrow 5Fh$ ,  $SP \leftarrow FFFEh$

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LDHL	$SP, e$	$HL \leftarrow SP+e$	*	*	0	0	3	11	111	000				
										* Varies with instruction results				
										$\leftarrow e \rightarrow$				
										$e = -128 \text{ to } +127$				

The 8-bit operand e is added to SP and the result is stored in HL.

Flag            Z: Reset  
                  H: Set if there is a carry from bit 11; otherwise reset.  
                  N: Reset  
                  CY: Set if there is a carry from bit 15; otherwise reset.

Example: When  $SP = 0xFFFF8$ ,  
LDHL SP, 2 ;  $HL \leftarrow 0xFFFA$ ,  $CY \leftarrow 0$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$ ,  $Z \leftarrow 0$

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
LD (nn), SP	(nn) ← SP <sub>L</sub>	--	--	--	--	5	00	001	000					
	(nnH) ← SP <sub>H</sub>							L-ADRS ← n →						
							H-ADRS ← n →							

Stores the lower byte of SP at address nn specified by the 16-bit immediate operand nn and the upper byte of SP at address  $nn + 1$ .

Example: When  $SP = FFF8h$ ,  
LD (C100h), SP ;  $C100h \leftarrow F8h$   
                           $C101h \leftarrow FFh$



### 2.3 8-Bit Arithmetic and Logical Operation Instructions

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADD	A, r	$A \leftarrow A + r$		*	*	0	*	1	10	000	r			

Adds the contents of register r to those of register A and stores the results in register A.

Flag  
 Z: Set if the result is 0; otherwise reset.  
 H: Set if there is a carry from bit 3; otherwise reset.  
 N: Reset  
 CY: Set if there is a carry from bit 7; otherwise reset.

Example: When A = 0x3A and B = 0xC6,  
 ADD A, B ;  $A \leftarrow 0$ ,  $Z \leftarrow 1$ ,  $H \leftarrow 1$ ,  $N \leftarrow 0$ ,  $CY \leftarrow 1$

							CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADD	A,	n	$A \leftarrow A + n$				*	*	0	*	2	11 000 110							
												$\longleftrightarrow n \longrightarrow$							

Adds 8-bit immediate operand n to the contents of register A and stores the results in register A..

Example: When A = 3Ch,  
 ADD A, FFh ;  $A \leftarrow 3Bh$ ,  $Z \leftarrow 0$ ,  $H \leftarrow 1$ ,  $N \leftarrow 0$ ,  $CY \leftarrow 1$

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADD	A, (HL)	$A \leftarrow A + (HL)$		*	*	0	*	2	10	000	110			

Adds the contents of memory specified by the contents of register pair HL to the contents of register A and stores the results in register A..

Example: When A = 3Ch and (HL) = 12h,  
 ADD A, (HL) ;  $A \leftarrow 4Eh$ ,  $Z \leftarrow 0$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$ ,  $CY \leftarrow 0$

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADC	A, s	$A \leftarrow A + s + CY$		*	*	0	*	--	--	--	--			

Adds the contents of operand s and CY to the contents of register A and stores the results in register A..  
 r, n, and (HL) are used for operand s.

## Chapter 4: CPU Instruction Set

		CYCL							
		7	6	5	4	3	2	1	0
ADC	A, r	1	10	001	r				
ADC	A, n	2	11	001	110				
			← n →						
ADC	A, (HL)	2	10	001	110				

Examples: When A = E1h, E = 0Fh, (HL) = 1Eh, and CY = 1,  
 ADC A, E ;  $A \leftarrow F1h, Z \leftarrow 0, H \leftarrow 1, CY \leftarrow 0$   
 ADC A, 3Bh ;  $A \leftarrow 1Dh, Z \leftarrow 0, H \leftarrow 0, CY \leftarrow -1$   
 ADC A, (HL) ;  $A \leftarrow 00h, Z \leftarrow 1, H \leftarrow 1, CY \leftarrow 1$

		CY H N Z CYCL 7 6 5 4 3 2 1 0									
SUB	s	A ← A-s	*	*	1	*	--	--	--	--	--

Subtracts the contents of operand s from the contents of register A and stores the results in register A.  
 r, n, and (HL) are used for operand s.

		CYCL							
		7	6	5	4	3	2	1	0
SUB	r	1	10	010	r				
SUB	n	2	11	010	110				
			← n →						
SUB	(HL)	2	10	010	110				

Flag  
 Z: Set if result is 0; otherwise reset.  
 H: Set if there is a borrow from bit 4; otherwise reset.  
 N: Set  
 CY: Set if there is a borrow; otherwise reset.

Examples: When A = 3Eh, E = 3Eh, and (HL) = 40h,  
 SUB E ;  $A \leftarrow 00h, Z \leftarrow 1, H \leftarrow 0, N \leftarrow 1, CY \leftarrow 0$   
 SUB 0Fh ;  $A \leftarrow 2Fh, Z \leftarrow 0, H \leftarrow 1, N \leftarrow 1, CY \leftarrow 0$   
 SUB (HL) ;  $A \leftarrow FEh, Z \leftarrow 0, H \leftarrow 0, N \leftarrow 1, CY \leftarrow 1$

		CY H N Z CYCL 7 6 5 4 3 2 1 0									
SBC	A, s	A ← A-s-CY	*	*	1	*	--	--	--	--	--

Subtracts the contents of operand s and CY from the contents of register A and stores the results in register A.  
 r, n, and (HL) are used for operand s.

**Game Boy Programming Manual**

			CYCL							
			7	6	5	4	3	2	1	0
SBC	A,	r	1	10	011	r				
SBC	A,	n	2	11	011	110				
				← n →						
SBC	A.	(HL)	2	10	011	110				

Examples: When A = 3Bh, (HL) = 4Fh, H = 2Ah, and CY = 1,  
 SBC A, H ; A ← 10h, Z ← 0, H ← 0, N ← 1 CY ← 0  
 SBC A, 3Ah ; A ← 00h, Z ← 1, H ← 0, N ← 1 CY ← 0  
 SBC A, (HL) ; A ← EBh, Z ← 0, H ← 1, N ← 1 CY ← 1

		CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
AND	s	A ← A ∧ s	0	1	0	*	--	--	--	--	--	--	--

Takes the logical-AND for each bit of the contents of operand s and register A, and stores the results in register A.  
 r, n, and (HL) are used for operand s.

		CYCL							
		7	6	5	4	3	2	1	0
AND	r	1	10	100	r				
AND	n	2	11	100	110				
			← n →						
AND	(HL)	2	10	100	110				

Examples: When A = 5Ah, L = 3Fh and (HL) = 0h,  
 AND L ; A ← 1Ah, Z ← 0, H ← 1, N ← 0 CY ← 0  
 AND 38h ; A ← 18h, Z ← 0, H ← 1, N ← 0 CY ← 0  
 AND (HL) ; A ← 00h, Z ← 1, H ← 1, N ← 0 CY ← 0

		CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
OR	s	AVs	0	0	0	*	--	--	--	--	--	--	--

Takes the logical-OR for each bit of the contents of operand s and register A and stores the results in register A. r, n, and (HL) are used for operand s.

#### Chapter 4: CPU Instruction Set

			CYCL							
			7	6	5	4	3	2	1	0
OR	r	1	10	110	r					
OR	n	2	11	110	110					
			← n →							
OR	(HL)	2	10	110	110					

Examples: When A = 5Ah, (HL) = 0Fh,  
 OR A ; A ← 5Ah, Z ← 0  
 OR 3 ; A ← 5Bh, Z ← 0  
 OR (HL) ; A ← 5Fh, Z ← 0

		CY H N Z CYCL 7 6 5 4 3 2 1 0							
XOR	s	A ⊕ s	0	0	0	*	--	--	--

Takes the logical exclusive-OR for each bit of the contents of operand s and register A. and stores the results in register A. r, n, and (HL) are used for operand s.

		CYCL							
		7	6	5	4	3	2	1	0
XOR	r	1	10	101	r				
XOR	n	2	11	101	110				
			← n →						
XOR	(HL)	2	10	101	110				

Examples: When A = FFh and (HL) = 8Ah,  
 XOR A ; A ← 00h, Z ← 1  
 XOR 0x0F ; A ← F0h, Z ← 0  
 XOR (HL) ; A ← 75h, Z ← 0

		CY H N Z CYCL 7 6 5 4 3 2 1 0							
CP	s	A — s	*	*	1	*	--	--	--

Compares the contents of operand s and register A and sets the flag if they are equal. r, n, and (HL) are used for operand s.

**Game Boy Programming Manual**

		CYCL					7	6	5	4	3	2	1	0
CP	r	1	10	111	r									
CP	n	2	11	111	110									
			<div>← n →</div>											
CP	(HL)	2	10	111	110									

Examples: When A = 3Ch, B = 2Fh, and (HL) = 40h,  
 CP B ;  $Z \leftarrow 0, H \leftarrow 1, N \leftarrow 1, CY \leftarrow 0$   
 CP 3Ch ;  $Z \leftarrow 1, H \leftarrow 0, N \leftarrow 1, CY \leftarrow 0$   
 CP (HL) ;  $Z \leftarrow 0, H \leftarrow 0, N \leftarrow 1, CY \leftarrow 1$

CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
INC	r	$r \leftarrow r + 1$		--	*	0	*	1	00	r	100

Increments the contents of register r by 1.

Example: When A = FFh,  
 INC A ;  $A \leftarrow 0, Z \leftarrow 1, H \leftarrow 1, N \leftarrow 0$

CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
INC	(HL)	$(HL) \leftarrow (HL) + 1$		--	*	0	*	3	00	110	100

Increments by 1 the contents of memory specified by register pair HL.

Example: When (HL) = 0x50,  
 INC (HL) ;  $(HL) \leftarrow 0x51, Z \leftarrow 0, H \leftarrow 0, N \leftarrow 0$

CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
DEC	r	$r \leftarrow r - 1$		--	*	1	*	1	00	r	101

Subtract 1 from the contents of register r by 1.

Example: When L = 01h,  
 DEC L ;  $L \leftarrow 0, Z \leftarrow 1, H \leftarrow 0, N \leftarrow 1$

CY	H	N	Z	CYCL 7 6 5 4 3 2 1 0							
DEC	(HL)	$(HL) \leftarrow (HL) - 1$		--	*	1	*	3	00	110	101

Decrements by 1 the contents of memory specified by register pair HL.

Example: When (HL) = 00h,  
 DEC (HL) ;  $(HL) \leftarrow FFh, Z \leftarrow 0, H \leftarrow 1, N \leftarrow 1$

## 2.4 16-Bit Arithmetic Operation Instructions

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADD	HL <sub>i</sub> ss	HL ← HL + ss	*	*	0	--	2	00	ss1	001				

Adds the contents of register pair **ss** to the contents of register pair **HL** and stores the results in **HL**. **ss** codes are as follows:

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

Flag            Z: No change  
                  H: Set if there is a carry from bit 11; otherwise reset.  
                  N: Rest  
                  CY: Set if there is a carry from bit 15; otherwise reset.

Example: When HL = 8A23h, BC = 0605h,  
 ADD HL, BC ; HL  $\leftarrow$  9028h, H  $\leftarrow$  1, N  $\leftarrow$  0, CY  $\leftarrow$  0  
 ADD HL, HL ; HL  $\leftarrow$  1446h, H  $\leftarrow$  1, N  $\leftarrow$  0, CY  $\leftarrow$  1

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
ADD	SP, e		*	*	0	0	4	11	101	000				
								<div> <div>←</div> <div>e</div> <div>→</div> </div>						

Adds the contents of the 8-bit immediate operand *e* and SP and stores the results in SP.

Example: SP = FFF8h  
ADD SP, 2 ; SP ← 0xFFFA, CY ← 0, H ← 0, N ← 0, Z ← 0

			CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
INC	SS	SS ← SS + 1	--	--	--	--	2	00		ss0			011		

Increments the contents of register pair ss by 1.

Example: When DE = 235Fh,  
INC DE ; DE ← 2360h

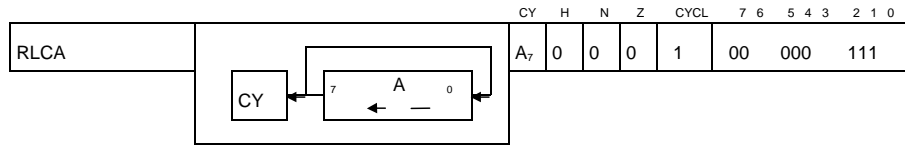
*Game Boy Programming Manual*

		CY H N Z				CYCL	7	6	5	4	3	2	1	0
DEC	ss	ss ← ss - 1				--	--	--	--	2	00	ss1	011	

Decrements the contents of register pair ss by 1.

Example: When DE = 235Fh,  
DEC DE ; DE ← 235Eh

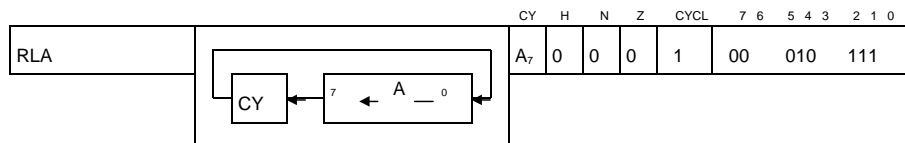
## 2.5 Rotate Shift Instructions



Rotates the contents of register A to the left.

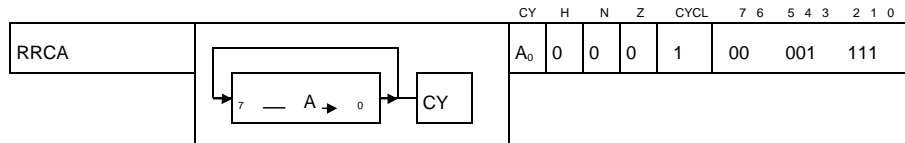
That is, the contents of bit 0 are copied to bit 1 and the previous contents of bit 1 (the contents before the copy operation) are copied to bit 2. The same operation is repeated in sequence for the rest of the register. The contents of bit 7 are placed in both CY and bit 0 of register A..

Example: When A = 85h and CY = 0,  
 RLCA ; A ← 0Ah, CY ← 1, Z ← 0, H ← 0, N ← 0



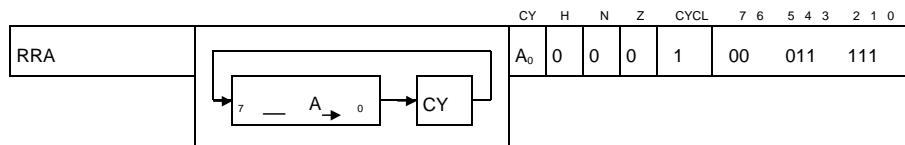
Rotates the contents of register A to the left.

Example: When A = 95h and CY = 1,  
 RLA ; A ← 2Bh, C ← 1, Z ← 0, H ← 0, N ← 0



Rotates the contents of register A to the right.

Example: When A = 3Bh and CY = 0,  
 RRCA ; A ← 9Dh, CY ← 1, Z ← 0, H ← 0, N ← 0

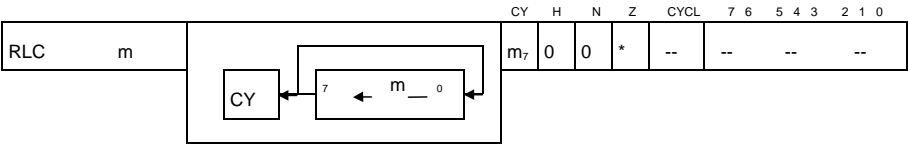


Rotates the contents of register A to the right.

Example: When A = 81h and CY = 0,  
 RRA ; A ← 40h, CY ← 1, Z ← 0, H ← 0, N ← 0



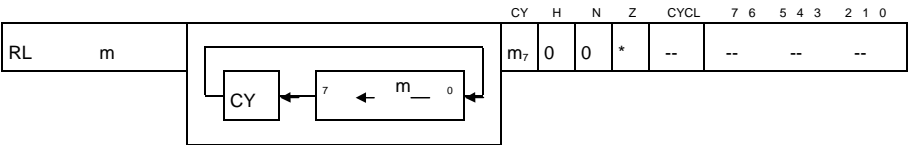
**Game Boy Programming Manual**



Rotates the contents of operand m to the left.  
r and (HL) are used for operand m.

		CYCL	7 6	5 4 3	2 1 0
RLC	r	2	11	001	011
			00	000	r
RLC	(HL)	4	11	001	011
			00	000	110

Examples: When B = 85h, (HL) = 0, and CY = 0,  
RLC B ; B ← 0Bh, CY ← 1, Z ← 0, H ← 0, N ← 0  
RLC (HL) ; (HL) ← 00h, CY ← 0, Z ← 1, H ← 0, N ← 0

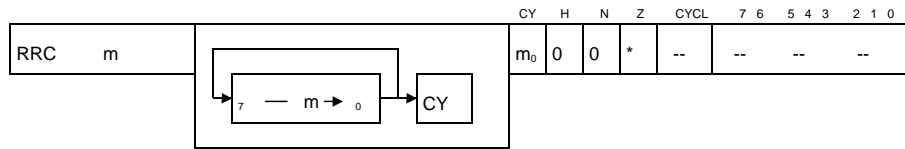


Rotates the contents of operand m to the left.  
r and (HL) are used for operand m.

		CYCL	7 6	5 4 3	2 1 0
RL	r	2	11	001	011
			00	010	r
RL	(HL)	4	11	001	011
			00	010	110

Examples: When L = 80h, (HL) = 11h, and CY = 0,  
RL L ; L ← 00h, CY ← 1, Z ← 1, H ← 0, N ← 0  
RL (HL) ; (HL) ← 22h, CY ← 0, Z ← 0, H ← 0, N ← 0

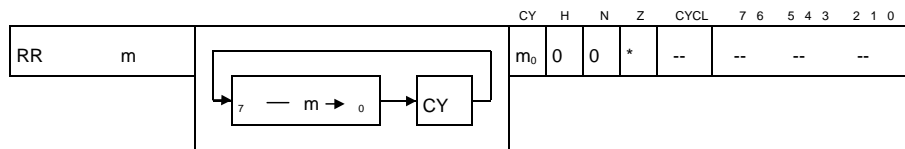
# Chapter 4: CPU Instruction Set



Rotates the contents of operand m to the right.  
r and (HL) are used for operand m.

		CYCL	7 6	5 4 3	2 1 0
RRC	r	2	11	001	011
			00	001	r
RRC	(HL)	4	11	001	011
			00	001	110

Examples: When C = 1h, (HL) = 0h, CY = 0,  
RRC C ; C ← 80h, CY ← 1, Z ← 0, H ← 0, N ← 0  
RRC (HL) ; (HL) ← 00h, CY ← 0, Z ← 1, H ← 0, N ← 0

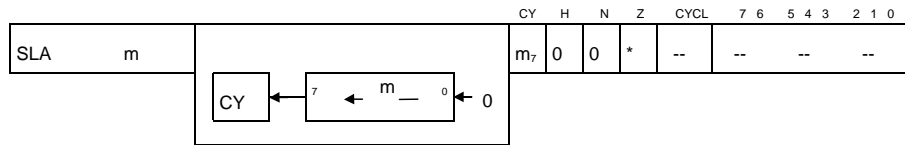


Rotates the contents of operand m to the right.  
r and (HL) are used for operand m.

		CYCL	7 6	5 4 3	2 1 0
RR	r	2	11	001	011
			00	011	r
RR	(HL)	4	11	011	011
			00	011	110

Examples: When A = 1h, (HL) = 8Ah, CY = 0,  
RR A ; A ← 00h, CY ← 1, Z ← 1, H ← 0, N ← 0  
RR (HL) ; (HL) ← 45h, CY ← 0, Z ← 0, H ← 0, N ← 0

**Game Boy Programming Manual**

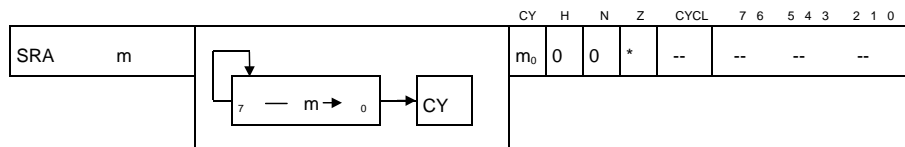


Shifts the contents of operand *m* to the left. That is, the contents of bit 0 are copied to bit 1 and the previous contents of bit 1 (the contents before the copy operation) are copied to bit 2. The same operation is repeated in sequence for the rest of the operand. The content of bit 7 is copied to CY, and bit 0 is reset.

*r* and (HL) are used for operand *m*.

		CYCL		7	6	5	4	3	2	1	0
SLA	<i>r</i>	2	11	001	011	00	100				<i>r</i>
SLA	(HL)	4	11	011	011	00	100				110

Examples: When *D* = 80h, (HL) = FFh, and CY = 0,  
 SLA *D* ; *D* ← 00h, CY ← 1, Z ← 1, H ← 0, N ← 0  
 SLA (HL) ; (HL) ← FEh, CY ← 1, Z ← 0, H ← 0, N ← 0

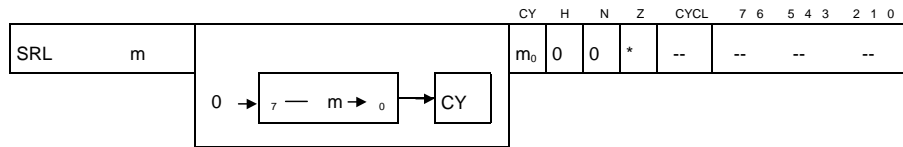


Shifts the contents of operand *m* to the right. That is, the contents of bit 7 are copied to bit 6 and the previous contents of bit 6 (the contents before the copy operation) are copied to bit 5. The same operation is repeated in sequence for the rest of the operand. The contents of bit 0 are copied to CY, and the content of bit 7 is unchanged.

*r* and (HL) are used for operand *m*.

		CYCL		7	6	5	4	3	2	1	0
SRA	<i>r</i>	2	11	001	011	00	101				<i>r</i>
SRA	(HL)	4	11	001	011	00	101				110

Example: When *A* = 8Ah, (HL) = 01h, and CY = 0,  
 SRA *D* ; *A* ← C5h, CY ← 0, Z ← 0, H ← 0, N ← 0  
 SRA (HL) ; (HL) ← 00h, CY ← 1, Z ← 1, H ← 0, N ← 0

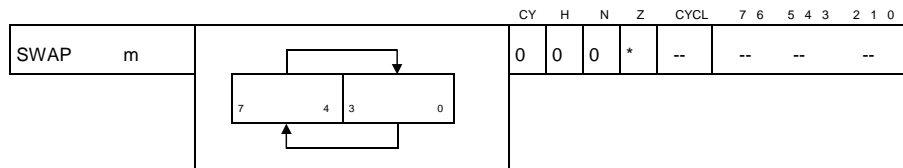


Shifts the contents of operand  $m$  to the right. That is, the contents of bit 7 are copied to bit 6 and the previous contents of bit 6 (the contents before the copy operation) are copied to bit 5. The same operation is repeated in sequence for the rest of the operand. The contents of bit 0 are copied to CY, and bit 7 is reset.

$r$  and (HL) are used for operand  $m$ .

		CYCL	7	6	5	4	3	2	1	0
SRL	$r$	2	11	001	011					
			00	111				$r$		
SRL	(HL)	4	11	001	011					
			00	111	110					

Examples: When  $A = 01h$ ,  $(HL) = FFh$ ,  $CY \leftarrow 0$ ,  
 SRL  $A$  ;  $A \leftarrow 00h$ ,  $CY \leftarrow 1$ ,  $Z \leftarrow 1$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$   
 SRL (HL) ;  $(HL) \leftarrow 7Fh$ ,  $CY \leftarrow 1$ ,  $Z \leftarrow 0$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$



Shifts the contents of the lower-order 4 bits (0-3) of operand  $m$  unmodified to the higher-order 4 bits (4-7) of that operand and shifts the contents of the higher-order 4 bits to the lower-order 4 bits.  
 $r$  and (HL) are used for operand  $m$ .

		CYCL	7	6	5	4	3	2	1	0
SWAP	$r$	2	11	001	011					
			00	110				$r$		
SWAP	(HL)	4	11	001	011					
			00	110	110					

Examples: When  $A = 00h$  and  $(HL) = F0h$ ,  
 SWAP  $A$  ;  $A \leftarrow 00h$ ,  $Z \leftarrow 1$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$ ,  $CY \leftarrow 0$   
 SWAP (HL) ;  $(HL) \leftarrow 0Fh$ ,  $Z \leftarrow 0$ ,  $H \leftarrow 0$ ,  $N \leftarrow 0$ ,  $CY \leftarrow 0$

2.6 Bit Operations

										CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
BIT	b,	r	$Z \leftarrow \overline{r_b}$					--	1	0	$\overline{r_b}$	2	11    001    011									
													01    b    r									

Copies the complement of the contents of the specified bit in register r to the Z flag of the program status word (PSW).

The codes for b and r are as follows.

Bit	b	Register	r
0	000	A	111
1	001	B	000
2	010	C	001
3	011	D	010
4	100	E	011
5	101	H	100
6	110	L	101
7	111		

Examples: When A = 80h and L = EFh  
BIT 7, A ; Z ← 0, H ← 1, N ← 0  
BIT 4, L ; Z ← 1, H ← 1, N ← 0

								CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
BIT	b,	(HL)	$Z \leftarrow \overline{(HL)_b}$	--	1	0	$\overline{(HL)_b}$	3												
									01      b      110											

Copies the complement of the contents of the specified bit in memory specified by the contents of register pair HL to the Z flag of the program status word (PSW).

Examples: When (HL) = FEh,  
BIT 0, (HL) ; Z ← 1, H ← 1, N ← 0  
BIT 1, (HL) ; Z ← 0, H ← 1, N ← 0

#### Chapter 4: CPU Instruction Set

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
SET	b,	r	$r_b \leftarrow 1$	--	--	--	--	2	11	001	011							
										11	b	r						

Sets to 1 the specified bit in specified register r.

Example: When A = 80h and L = 3Bh,  
 SET 3, A ;  $A \leftarrow 0x84$   
 SET 7, L ;  $L \leftarrow 0xBB$

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
SET	b,	(HL)	(HL) <sub>b</sub> ← 1	--	--	--	--	4	11	001	011							
										11	b	110						

Sets to 1 the specified bit in the memory contents specified by registers H and L.

Example: When 00h is the memory contents specified by H and L,  
 SET 3, (HL) ;  $(HL) \leftarrow 04H$

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
RES	b,	r	$r_b \leftarrow 0$			--	--	--	--	2	11		001			011		
											10		b			r		

Resets to 0 the specified bit in the specified register r.

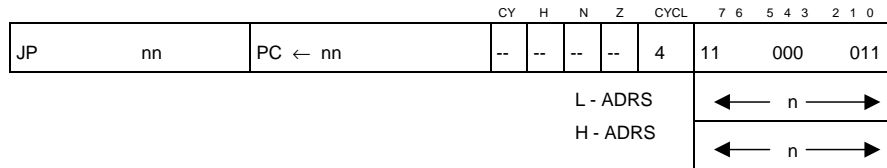
Example: When A = 80h and L = 3Bh,  
 RES 7, A ;  $A \leftarrow 00h$   
 RES 1, L ;  $L \leftarrow 39h$

						CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
RES	b,	(HL)	(HL) <sub>b</sub> ← 0			--	--	--	--	4	11		001			011		
											10		b			110		

Resets to 0 the specified bit in the memory contents specified by registers H and L.

Example: When 0xFF is the memory contents specified by H and L,  
 RES 3, (HL) ;  $(HL) \leftarrow F7h$

## 2.7 Jump Instructions

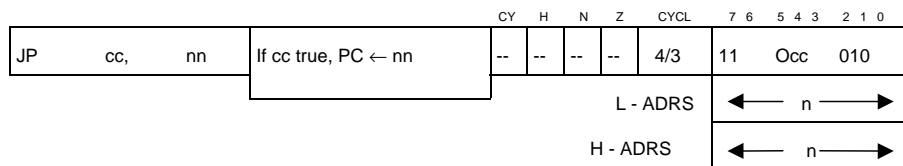


Loads the operand nn to the program counter (PC).

**nn** specifies the address of the subsequently executed instruction.

The lower-order byte is placed in byte 2 of the object code and the higher-order byte is placed in byte 3.

Example: JP 8000h ; Jump to 8000h.



\*Cycle no. is 3 when cc does not match

Loads operand nn in the PC if condition cc and the flag status match.

The subsequent instruction starts at address nn.

If condition cc and the flag status do not match, the contents of the PC are incremented, and the instruction following the current JP instruction is executed.

The relation between conditions and cc codes are as follows.

Cc	Condition	Flag
00	NZ	Z = 0
01	Z	Z = 1
10	NC	CY = 0
11	C	CY = 1

Example: When Z = 1 and C = 0,

JP NZ, 8000h ; Moves to next instruction after 3 cycles.

JP Z, 8000h ; Jumps to address 8000h.

JP C, 8000h ; Moves to next instruction after 3 cycles.

JP NC, 8000h ; Jumps to address 8000h.

#### Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
JR	e	PC ← PC + e	--	--	--	--	3	00	011	000				
								← e - 2 →						

e = -127 to +129

Jumps -127 to +129 steps from the current address.

							CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
JR	cc,	e	If cc true, PC ← PC + e				--	--	--	--	3/2	00	1cc	000					
												← e - 2 →							

e = -127 to +129

If condition cc and the flag status match, jumps -127 to +129 steps from the current address. If cc and the flag status do not match, the instruction following the current JP instruction is executed.

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
JP	(HL)	PC ← HL	--	--	--	--	1	11	101	001				

Loads the contents of register pair HL in program counter PC.  
The next instruction is fetched from the location specified by the new value of PC.

Example: When HL = 8000h,  
JP (HL) ; Jumps to 8000h.



## 2.8 Call and Return Instructions

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
CALL	nn	(SP - 1) ← PC <sub>H</sub> (SP - 2) ← PC <sub>L</sub> PC ← nn SP ← SP-2		--	--	--	--	6	11	001	101			
									L - ADRS		← n →			
									H - ADRS		← n →			

In memory, pushes the PC value corresponding to the instruction at the address following that of the CALL instruction to the 2 bytes following the byte specified by the current SP. Operand nn is then loaded in the PC.

The subroutine is placed after the location specified by the new PC value.

When the subroutine finishes, control is returned to the source program using a return instruction and by popping the starting address of next instruction, which was just pushed, and moving it to the PC.

With the push, the current value of the SP is decremented by 1, and the higher-order byte of the PC is loaded in the memory address specified by the new SP value. The value of the SP is then again decremented by 1, and the lower-order byte of the PC is loaded in the memory address specified by that value of the SP.

The lower-order byte of the address is placed in byte 2 of the object code, and the higher-order byte is placed in byte 3.

Examples: When PC = 8000h and SP = FFFEh,

Address

8000h CALL 1234H ; Jumps to address 1234h, and  
 8003h (FFFDH) ← 80H  
 (FFFCH) ← 03H  
 SP ← FFFCH

				CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
CALL	cc,	nn	If cc true, (SP - 1) ← PC <sub>H</sub> (SP - 2) ← PC <sub>L</sub> PC ← nn SP ← SP - 2	--	--	--	--	6/3	11	Occ	100					
				L-ADRS					← n →							
				H-ADRS					← n →							

If condition cc matches the flag, the PC value corresponding to the instruction following the CALL instruction in memory is pushed to the 2 bytes following the memory byte specified by the SP. Operand nn is then loaded in the PC.

Examples: When Z = 1,

Address

7FFCh CALL NZ, 1234h ; Moves to next instruction after 3 cycles.  
 8000h CALL Z, 1234h ; Pushes 8003h to the stack,  
 8003h and jumps to 1234h.

#### Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
RET	$PC_L \leftarrow (SP)$	--	--	--	--	4	11		001		001			
	$PC_H \leftarrow (SP + 1)$													
	$SP \leftarrow SP + 2$													

Pops from the memory stack the PC value pushed when the subroutine was called, returning control to the source program.

In this case, the contents of the address specified by the SP are loaded in the lower-order byte of the PC, and the content of the SP is incremented by 1. The contents of the address specified by the new SP value are then loaded in the higher-order byte of the PC, and the SP is again incremented by 1. (The value of SP is 2 larger than before instruction execution.)

The next instruction is fetched from the address specified by the content of PC.

Examples:      Address  
                  8000H    CALL 9000H  
                  8003H  
                  9000H  
                  RET                      ; Returns to address 0x8003

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
RETI	$PC_L \leftarrow (SP)$	--	--	--	--	4	11		011		001			
	$PC_H \leftarrow (SP + 1)$													
	$SP \leftarrow SP + 2$													

Used when an interrupt-service routine finishes.  
 The execution of this return is as follows.

The address for the return from the interrupt is loaded in program counter PC.  
 The master interrupt enable flag is returned to its pre-interrupt status.

Examples:      0040h      RETI                      ; Pops the stack and returns to address 8001h.  
                  8000H      INC L                      :An external interrupt occurs here.  
                  8001H

Game Boy Programming Manual

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
RET	If cc true, PC <sub>L</sub> ← (SP) PC <sub>H</sub> ← (SP+1) SP ← SP + 2	--	--	--	--	5/2	11		Occ			000		

If condition cc and the flag match, control is returned to the source program by popping from the memory stack the PC value pushed to the stack when the subroutine was called.

Example:      Address  
              8000h   CALL   9000h  
              8003h  
  
9000h       CP       0  
              RET     Z               ; Returns to address 8003h if Z = 1.  
                                      Moves to next instruction after 2 cycles if Z = 0.

		CY H N Z				CYCL				7 6 5 4 3 2 1 0			
RST	t	$(SP - 1) \leftarrow PC_H$ $(SP - 2) \leftarrow PC_L$ $SP \leftarrow SP - 2$ $PC_H \leftarrow 0 \quad PC_L \leftarrow P$				4				11 t 111			

Pushes the current value of the PC to the memory stack and loads to the PC the page 0 memory addresses provided by operand t.  
Then next instruction is fetched from the address specified by the new content of PC.

With the push, the content of the SP is decremented by 1, and the higher-order byte of the PC is loaded in the memory address specified by the new SP value. The value of the SP is then again decremented by 1, and the lower-order byte of the PC is loaded in the memory address specified by that value of the SP.

The RST instruction can be used to jump to 1 of 8 addresses.

Because all of the addresses are held in page 0 memory, 0x00 is loaded in the higher-order byte of the PC, and the value of P is loaded in the lower-order byte.

The relation between the t codes and P are as follows.

Operand	t	(PC <sub>H</sub> )	P (PC <sub>L</sub> )
0	000	00h	00h
1	001	00h	08h
2	010	00h	10h
3	011	00h	18h
4	100	00h	20h
5	101	00h	28h
6	110	00h	30h
7	111	00h	38h

Example: Address 8000h RST 1 ; Pushes 8001h to the stack ,  
8001h and jumps to 0008h.

## 2.9 General-Purpose Arithmetic Operations and CPU Control Instructions

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
DAA	Decimal adjust acc	*	0	--	*	1	00	100	111					

When performing addition and subtraction, binary coded decimal representation is used to set the contents of register A to a binary coded decimal number (BCD).

The following table shows the processing that accompanies execution of the DAA instruction immediately following execution of addition (ADD and ADC) and subtraction (SUB and SBC) instructions.

Instruction Before Execution	CY Contents before Execution	Bits 4-7 Register A	H Contents before Execution	Bits 0-3 Register A	Number Added to Register A	CY Contents after Execution
ADD ADC	0	0h- 9h	0	0h- 9h	00h	0
	0	0h- 8h	0	Ah-Fh	06h	0
	0	0h- 9h	1	0h- 3h	06h	0
	0	Ah-Fh	0	0h- 9h	60h	1
	0	9h-Fh	0	Ah-Fh	66h	1
	0	Ah-Fh	1	0h- 3h	66h	1
	1	0h- 2h	0	0h- 9h	60h	1
	1	0h- 2h	0	Ah-Fh	66h	1
(N = 0)	1	0h- 3h	1	0h- 3h	66h	1
SUB SBC	0	0h-9h	0	0h- 9h	00h	0
	0	0h-8h	1	6h- Fh	FAh	0
	1	7h-Fh	0	0h- 9h	A0h	1
	1	6h-Fh	1	6h- Fh	9Ah	1

Examples: When A = 45h and B = 38h,

ADD A, B ; A ← 7Dh, N ← 0

DAA ; A ← 7Dh + 06h (83h), CY ← 0

SUB A, B ; A ← 83h – 38h (4Bh), N ← 1

DAA ; A ← 4Bh + FAh (45h)

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
CPL	$\overline{A \leftarrow A}$	--	1	1	--	1	00	101	111					

Takes the one's complement of the contents of register A.

Example: When A = 35h,

CPL ; A ← CAh

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
NOP	No operation	--	--	--	--	1	00	000	000					

Only advances the program counter by 1; performs no other operations that have an effect.

#### Chapter 4: CPU Instruction Set

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
CCF	$CY \leftarrow (\overline{CY})$	$\overline{(CY)}$	0	0	--	1	00	111	111					

Flips the carry flag CY.

Example: When CY = 1,  
CCF ; CY  $\leftarrow$  0

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
SCF	$CY \leftarrow 1$	1	0	0	--	1	00	110	111					

Sets the carry flag CY.

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
DI	$IME \leftarrow 0$	--	--	--	--	1	11	110	011					

Resets the interrupt master enable flag and prohibits maskable interrupts.

**Note:** Even if a DI instruction is executed in an interrupt routine, the IME flag is set if a return is performed with a RETI instruction.

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
EI	$IME \leftarrow 1$	--	--	--	--	1	11	111	011					

Sets the interrupt master enable flag and enables maskable interrupts.

This instruction can be used in an interrupt routine to enable higher-order interrupts.

**Note:** The IME flag is reset immediately after an interrupt occurs. The IME flag reset remains in effect if control is returned from the interrupt routine by a RET instruction. However, if an EI instruction is executed in the interrupt routine, control is returned with IME = 1.

## Game Boy Programming Manual

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
HALT	Halt	--	--	--	--	1	01	110	110					

After a HALT instruction is executed, the system clock is stopped and HALT mode is entered. Although the system clock is stopped in this status, the oscillator circuit and LCD controller continue to operate.

In addition, the status of the internal RAM register ports remains unchanged.

HALT mode is canceled by an interrupt or reset signal.

The program counter is halted at the step after the HALT instruction. If both the interrupt request flag and the corresponding interrupt enable flag are set, HALT mode is exited, even if the interrupt master enable flag is not set.

Once HALT mode is canceled, the program starts from the address indicated by the program counter.

If the master enable flag is set, the contents of the program counter are pushed to the stack and control jumps to the starting address of the interrupt.

If the RESET terminal goes LOW in HALT mode, the mode becomes that of a normal reset.

		CY	H	N	Z	CYCL	7	6	5	4	3	2	1	0
STOP	Stop	--	--	--	--	1	00	010	000					
							00	000	000					

Execution of a STOP instruction stops both the system clock and oscillator circuit. STOP mode is entered, and the LCD controller also stops.

However, the status of the internal RAM registers ports remains unchanged.

STOP mode can be canceled by a reset signal.

If the RESET terminal goes LOW in STOP mode, it becomes that of a normal reset status.

The following conditions should be met before a STOP instruction is executed and STOP mode is entered.

- All interrupt-enable (IE) flags are reset.
- Input to P10 — P13 is LOW for all.

## **CHAPTER 5: MISCELLANEOUS GENERAL INFORMATION**

.....	126
1. MONITOR ROM .....	126
2. RECOGNITION DATA FOR CGB ONLY IN ROM-REGISTERED DATA.....	127
3. POWER-SAVING ROUTINES FOR THE MAIN PROGRAM .....	128
4. SOFTWARE CREATED EXCLUSIVELY FOR CGB.....	130
5. SOFTWARE CREATED TO OPERATE ON CGB .....	131
6. SOFTWARE CREATED TO OPERATE ON CGE: EXAMPLE .....	132
6.1 Program Specifications .....	132
6.2 CGB Recognition Method.....	133
6.3 Flowcharts .....	134



## CHAPTER 5: MISCELLANEOUS GENERAL INFORMATION

### 1. MONITOR ROM

The DMG and CGB CPU includes internal monitor ROM.

When power on the hardware is turned on, the monitor ROM checks for errors in the 'Nintendo' logo character data within the game software.

If the data is correct, the Nintendo logo is displayed and the program is then started. If there is an error in the data, the screen flashes repeatedly.

For information on registering the Nintendo logo character data, refer to Appendix 3 of this manual, *Submission Requirements*.

**The conditions required for starting the user program are as follows.**

Starting Address	150h (default value)	The starting address can be freely set by writing a jump destination address at 102h and 103h.
LCDC value	91h	
Stack value	FFFEh	

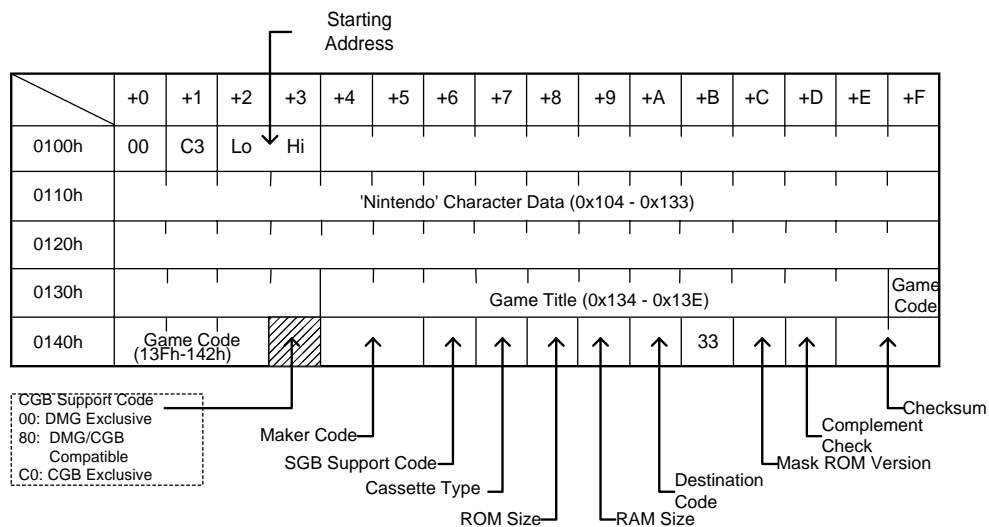
## 2. RECOGNITION DATA FOR CGB (CGB ONLY) IN ROM-REGISTERED DATA

As with software created for DMG, software for CGB (including software only for CGB) must place data concerning items such as the name of the game and Game Pak specifications in the 80 bytes of the program area between 100h and 14Fh. In the system, a code indicating whether the software is for CGB should be set at address 143h.

**Note:** For an overall description of the ROM area shown below, please refer to Appendix 3, Submission Requirements.

Setting a value of 80h or C0h at this address causes the system to recognize the software as being for CGB.

If 00h or any value less than 7Fh (existing DMG software) is set at this address, the software is recognized as non-CGB software and CGB functions (registers) are not available.



CGB/CGB Only: When operating on CGB, up to 56 colors can be displayed on a single screen.  
 Non-CGB: When operating on CGB, up to 10 colors can be displayed on a single screen.

**Note:** Regardless of the type of game, the following fixed values should be stored at the following addresses.

- Address 100h=00h
- Address 101h=C3h
- Address 14Bh=33h
- Addresses 104h – 133h='Nintendo' character data

### 3. POWER-SAVING ROUTINES FOR THE MAIN PROGRAM

To minimize battery power consumption and extend battery life, inclusion of programs such as those shown below is recommended.

During waiting for vertical blanking, halt the CPU system clock to reduce power consumption by the CPU and ROM.

```

*****
;
*****
;
*****
;
Main Routine
*****
*****

MAIN
    CALL    CONT      :   Keypad input.
    CALL    GAME      :   Game or other processing.

VBLK_WT
    HALT          :   Halt the system clock.
                  :   Return from HALT mode if an interrupt is
                  :   generated.
    NOP          :   Wait for a vertical blanking interrupt.
                  :   Used to avoid bugs in the rare case that the
                  :   instruction. after the HALT instruction is not
                  :   executed.

    LD        A, (VBLK_F)
    AND       A        :   Generate a V-blank interrupt?
    JR        Z, VBLK_WT :   Jump if a non-V-blank interrupt.
    XOR       A
    LD        (VBLK_F), A
    JR        MAIN

*****
;
*****
;
*****
;
Vertical Blanking Routine
*****
*****

VBLK
    PUSH     AF
    PUSH     BC
    PUSH     DE
    PUSH     HL

    CALL     DMA

    LD        A, 1
    LD        (VBLK_F), A      :   Set the V-blank completion flag.

    POP      HL
    POP      DE
    POP      BC
    POP      AF
    RETI

```

## **Chapter 5: Miscellaneous General Information**

HALT instructions should not be executed while CGB horizontal blanking DMA is executed. (See Appendix 1, *Programming Cautions*.)

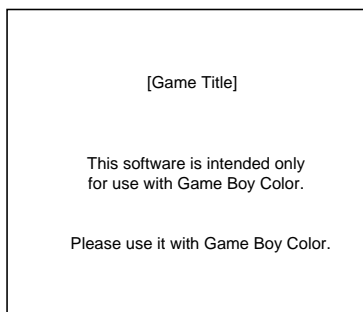
## **4. SOFTWARE CREATED EXCLUSIVELY FOR CGB**

Because the shape of the Game Pak for CGB-only software is the same as that for DMG, CGB-only Game Paks also can be inserted in DMG. Therefore, a program that displays a message such as that shown below when a CGB-only Game Pak is mistakenly inserted in DMG should always be included in the software. The upper part of the message screen should display the official title of the game.

If the title is similar to that of other software (e.g., series software), a subtitle should also be displayed to distinguish the programs from one another.

For information on software methods of distinguishing game units, see Section 6 of this chapter, *Software Created for CGB: Example*.

### *Sample Message Display*



## 5. SOFTWARE CREATED TO OPERATE ON CGB

As is shown below, CGB and DMG differ slightly in their specifications and operation. When creating software to operate on CGB, please give appropriate consideration to these differences.

CGB	DMG
When objects with different x-coordinates overlap, the object with the lowest OBJ NO. is given display priority.	When objects with different x-coordinates overlap, the object with the smallest x-coordinate is given display priority.
In CGB mode, BG display CANNOT be turned off using bit 0 of the LCDC register (address FF40h).	BG display CAN be turned on and off using bit 0 of the LCDC register (address FF40h).
_____	When the value of register WX (address FF4Bh) is 166, the window is partially displayed.
_____	When an instruction that register pair increment is used, if the value of the register pair is an address that specifies OAM (FE00h-FE9Fh), OAM may be destroyed.

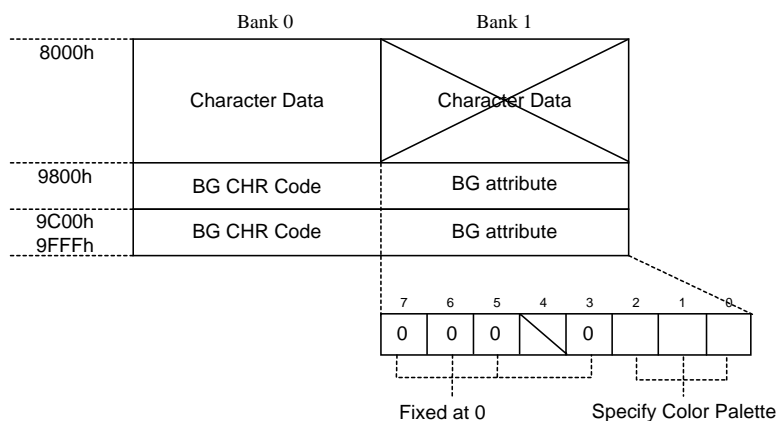
## 6. SOFTWARE CREATED TO OPERATE ON CGB: EXAMPLE

When creating software for CGB, a CGB support code is set in the ROM data area, and processing branches according to the hardware used internally by the program. For more information, see the flowchart in Part 1 of Section 6.3 of this chapter. Limiting the functions used, as shown below, allows the same processing to be used for different units without branching. For more information, see the flowchart in Part 2 of Section 6.3 of this chapter.

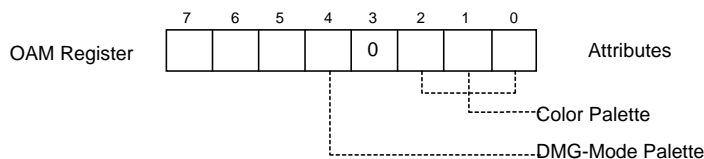
The following example describes how to create a program that operates on both CGB and DMG and allows display of 56 colors when running on CGB. Such means can be used to maintain compatibility with earlier hardware (DMG) while using CGB functions.

### 6.1 Program Specifications

- Only bank 0 is used as the character data area.
- Only the bits that specify the color palette (bits 0-2 of bank 1) are used for BG attributes.



- Both the color palette and DMG-mode palette are set as attribute flags in the OAM register.



- None of the other expanded CGB functions are used.

## **6.2 CGB Recognition Method**

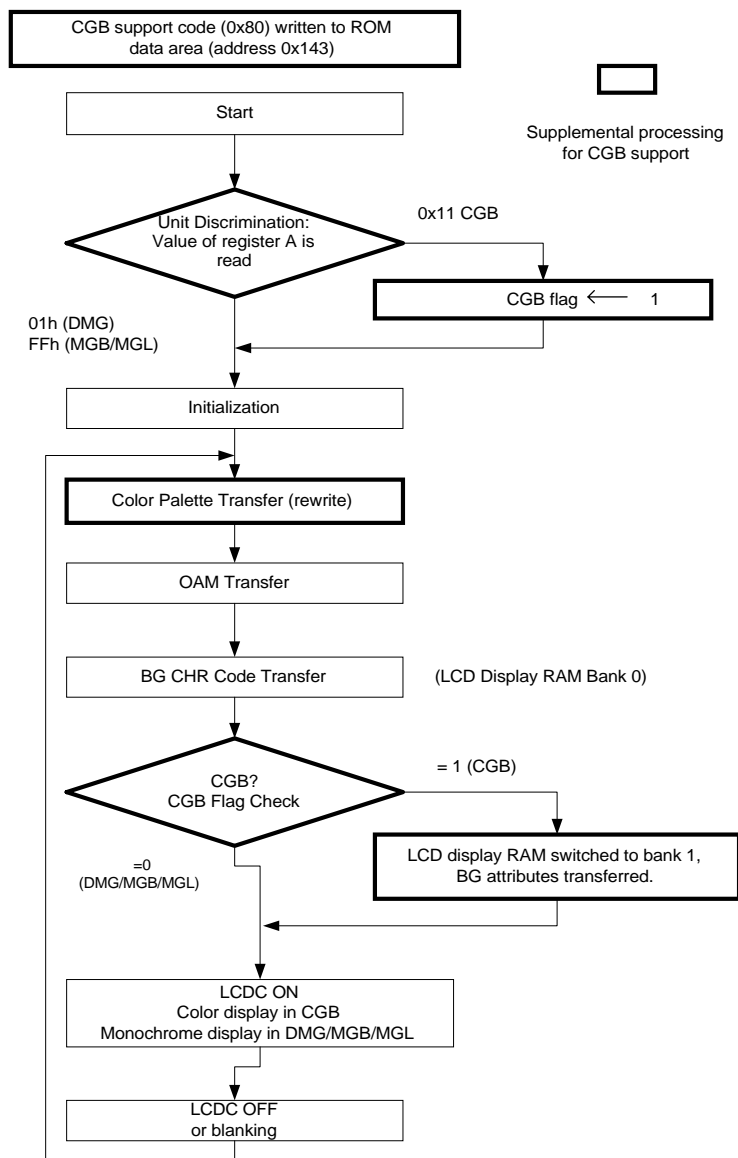
Immediately after program startup, the initial value of the accumulator (register A) is read to determine whether the hardware on which the program is operating is a DMG (SGB), MGB/MGL (SGB2), or CGB.

01h → DMG (SGB)  
FFh → MGB/MGL (SGB2)  
11h → CGB

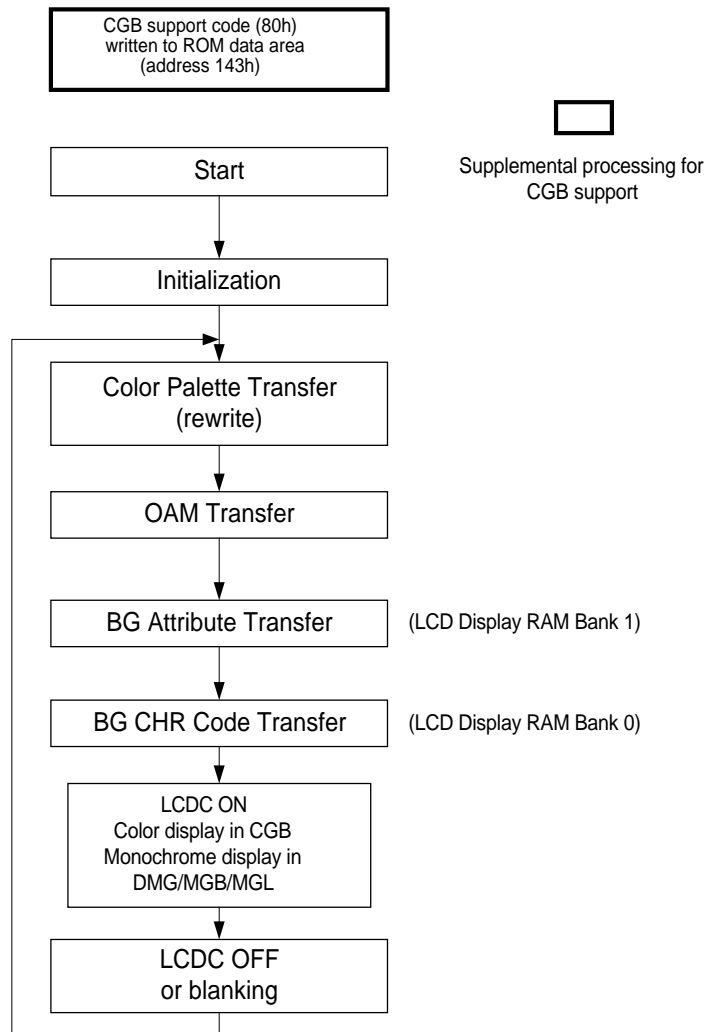


### 6.3 Flowcharts

#### 1) Branched Processing for CGB and DMG/MGB/MGL



2) Uniform processing for CGB and DMG/MGB/MGL



**Note:** The BG attributes should always be transferred before the BG character code.

Even if only the BG attributes are changed, always transfer the character code from that same address.

***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

## CHAPTER 8: GAME BOY MEMORY CONTROLLERS (MBC) ..216

<b>1. MBC1 .....</b>	<b>216</b>
1.1 Overview .....	216
1.2 Description of Registers .....	216
1.3 Memory Map .....	218
<b>2. MBC2 .....</b>	<b>219</b>
2.1 Overview .....	219
2.2 Description of Registers .....	219
2.3 Memory Map .....	219
2.4 Backup RAM .....	219
<b>3. MBC3 .....</b>	<b>220</b>
3.1 Overview .....	220
3.2 Description of Registers .....	220
3.3 Accessing the Clock Counters .....	221
3.4 Memory Map .....	221
3.5 Programming Items to Note .....	222
<b>4. MBC5 .....</b>	<b>224</b>
4.1 Overview .....	224
4.2 Registers .....	224
4.3 Memory Map .....	224
4.4 Description of Registers .....	225
4.5 Programming Cautions .....	226
4.6 Examples of MBC5 programs on DMG and CGB .....	227
<b>5. MBC5 (WITH RUMBLE FEATURE) .....</b>	<b>228</b>
5.1 Overview .....	228
5.2 Registers .....	228
5.3 Memory Map .....	229
5.4 Description of Registers .....	229
5.5 Motor Control .....	230
5.6 Programming Cautions .....	231
5.7 Physical Effects of Vibration on the Body .....	233

## CHAPTER 8: GAME BOY MEMORY CONTROLLERS (MBC)

### 1. MBC1

#### 1.1 Overview

MBC1 is a memory controller that enables the use of 512 Kbits (64 Kbytes) or more of ROM and 256 Kbits (32 Kbytes) of RAM. It can be used as follows.

- To control up to 4 Mbits of ROM  
When used to control up to 4 Mbits (512 Kbytes) of ROM, MBC1 can control up to 256 Kbits (32 Kbytes) of RAM.
- To control 8 Mbits or more of ROM  
When MBC1 is used to control up to 8 Mbits (1 MB) or 16 Mbits (2 MB) of ROM, the following conditions apply
  - When used to control 8 Mbits of ROM  
MCB cannot use ROM addresses 080000h-083FFFh (Bank 20h)
  - When used to control 16 Mbits of ROM  
MBC1 cannot use ROM Addresses 

8000h-083FFFh (Bank 20h)
100000h-103FFFh (Bank 40h)
180000h-183FFFh (Bank 60h)

RAM use by MBC1 is restricted to 64 Kbits (8 Kbytes).

#### 1.2 Description of Registers

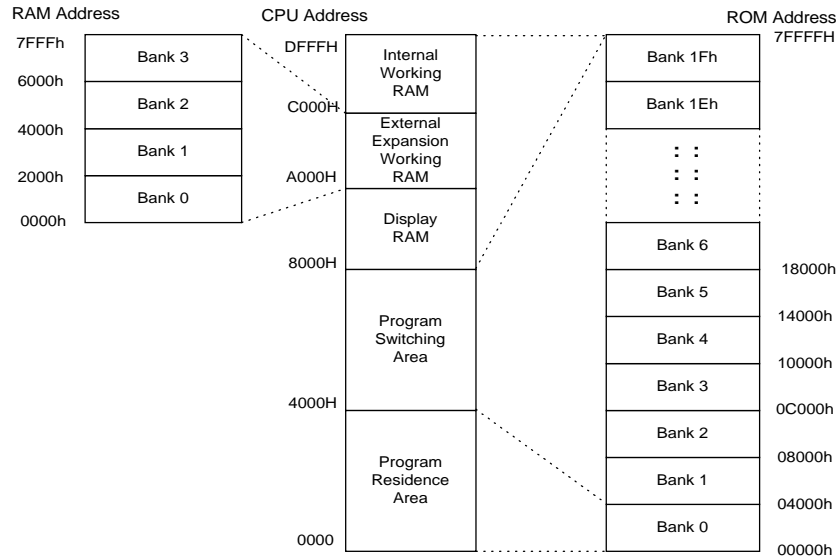
- Register 0: RAMCS gate data (serves as write-protection for RAM)  
Write addresses: 0000h-1FFFh Write data: 0Ah  
Writing 0Ah to 0h-1FFFh causes the CS to be output, allowing access to RAM.
  - Register 1: ROM bank code  
Write addresses: 2000h-3FFFh Write data: 01h-1Fh  
The ROM bank can be selected.
  - Register 2: Upper ROM bank code when using 8 Mbits or more of ROM (and register 3 is 0)  
Write addresses: 4000h-5FFFh Write data: 0-3  
The upper ROM banks can be selected in 512-Kbyte increments.
    - Write value of 0 selects banks 01h-1Fh
    - Write value of 1 selects banks 21h-3Fh
    - Write value of 2 selects banks 41h-5Fh
    - Write value of 3 selects banks 61h-7Fh
- : RAM bank code when using 256 Kbits of RAM (and register 3 is 1)
- Write addresses: 4000h-5FFFh Write data: 0-3  
The RAM bank can be selected in 8-Kbyte increments.

## ***Chapter 8: Game Boy Memory Controllers (MBC)***

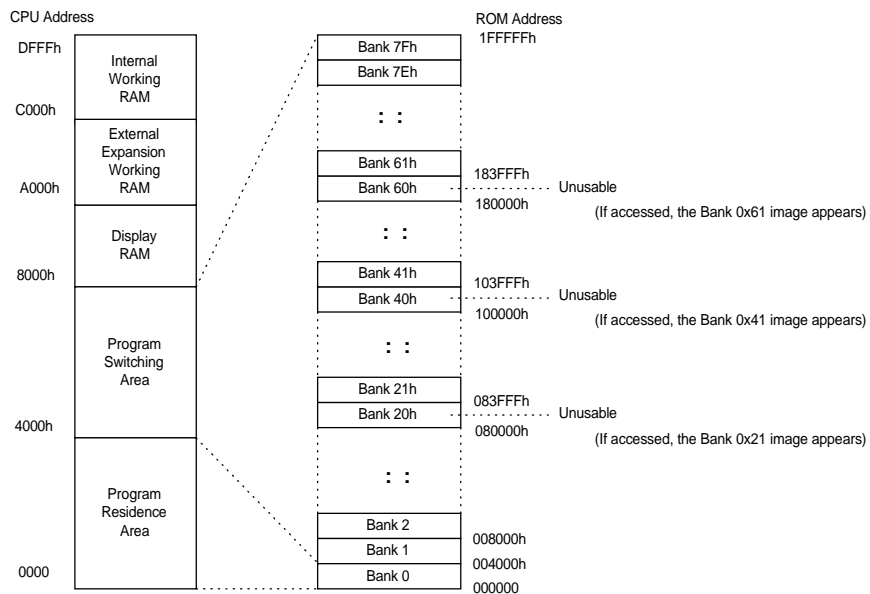
- Register 3: ROM/RAM change  
Write addresses: 6000h-7FFFh Write Data: 0-1  
Writing 0 causes the register 2 output to control switching of the higher ROM bank.  
Writing 1 causes the register 2 output to control switching of the RAM bank.

### 1.3 Memory Map

- When Used to Control up to 4 Mbits of ROM



- When Used to Control up to 8 Mbits of ROM



## 2. MBC2

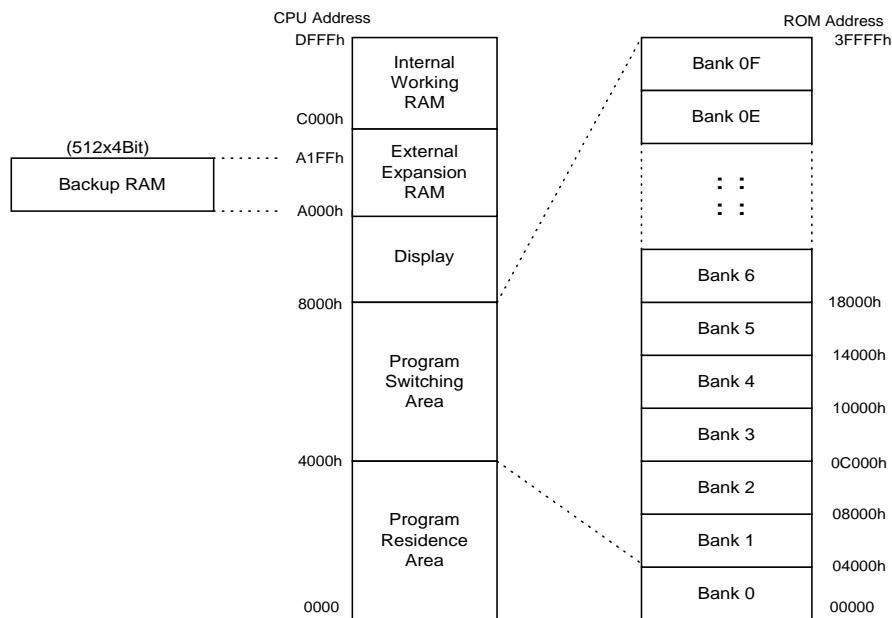
### 2.1 Overview

Controller for up to 2 Mbits (256 Kbytes) of ROM with built-in backup RAM (512 x 4 bits).

### 2.2 Description of Registers

- Register 0: RAMCS gate data (serves as write-protection for RAM)  
Write addresses: 000h-0FFFh Write data: 0Ah  
Writing 0Ah to 000-0FFFh causes the CS to be output, allowing access to RAM.
- Register 1: ROM bank code  
Write addresses: 2100h-21FFh Write data: 01h-0Fh  
The ROM bank can be selected.

### 2.3 Memory Map



### 2.4 Backup RAM

Allocated to the D0-D3 areas of CPU addresses A000h-A1FFh  
Backup RAM is write-protected by a power-on reset.  
To protect backup data, avoid removing write protection unless necessary.



MBC3 is the memory bank controller that allows use of between 512 Kbits (64 Kbytes) and 16 Mbits (2 MB) of ROM and 256 Kbits (32 Kbytes) of RAM. Built into the controller are clock counters that operate by means of an external crystal oscillator (32.768 KHz). The clock counters are accessed by RAM bank switching. RAM and clock counter data can be backed up by an external lithium battery.

Settings for control registers 0-3 are specified by writing data to the ROM area.

- Register 0: Write protects RAM and the clock counters (default: 0)  
Write addresses: 0000h-1FFFh Write data: 0Ah\_\_\_\_  
Allows access to RAM and the clock counter registers.
- Register 1: ROM bank code (default: 0, selects ROM bank 1)  
Write addresses: 2000h-3FFFh Write data: 01h-7Fh  
Allows the ROM bank to be selected in 16-Kbyte increments.
- Register 2: RAM bank code (default: 0, selects RAM bank 0)  
Write addresses: 4000h-5FFFh Write data: 0-3  
Allows the RAM bank to be selected in 8-Kbyte increments.

Data	Register	Range of Values	Function
08h	RTC_S	0-59 (0-3Bh)	Seconds counter (6 bits)
09h	RTC_M	0-59 (0-3Bh)	Minutes counter (6 bits)
0Ah	RTC_H	0-23 (0-17h)	Hours counter (5 bits)
0Bh	RTC_DL	0-255 (0-FFh)	Lower-order 8 bits of days counter
0Ch	RTC_DH	<div style="text-align: center;"> <div style="display: flex; justify-content: space-between; width: 100%;"> <span>bit7</span> <span>bit0</span> </div> <div style="display: flex; align-items: center; justify-content: center;"> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px;"></div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px;"></div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px; text-align: center;">0</div> <div style="border: 1px solid black; width: 20px; height: 20px; margin: 0 5px;"></div> </div> </div>	<p>Higher-order bit and carry bit of days counter.</p> <p>HALT starts and stops the clock counters.</p>

- \* The days counter consists of a 9-bit counter + a carry bit. Thus, it can count from 0 to 511 (000h-1FFh).
  - \* Once the carry bit is set to 1, it remains 1 until 0 is written.
  - \* The counters operate when HALT is 0 and stop when HALT is 1.
  - \* Values outside the given counter ranges will not be correctly written.
- Register 3: Latches the data for all clock counters (default: 0)

Write addresses: 6000h-7FFFh Write Data: 0 → 1

Writing 0 → 1 causes all counter data to be latched. The latched contents are retained until 0 → 1 is written again.

### 3.3 Accessing the Clock Counters

The clock counter registers are assigned to the external expansion RAM area of the CPU address space. To access the clock counters, RAM bank switching must first be performed.

External expansion RAM Area (A000h-BFFFh) Bank Map

Bank	Device	Notes
00h	RAM BANK 0	
01h	RAM BANK 1	
02h	RAM BANK 2	
03h	RAM BANK 3	
		Not used
08h	Seconds counter	
09h	Minutes counter	
0Ah	Hours counter	
0Bh	Days counter (L)	
0Ch	Days counter (H)	
:		Not used

The following are examples of accessing the clock counters.

#### 3.3.1 Reading

The clock counters are accessed by first writing 0x0A to register 0. This opens the gate used to access the counters. To read clock counter values, write 1 to register 3 to latch the values of all the registers. If the value of register 3 is already 1, first set it to 0 and then to 1. While this register is set to 1, the clock counters will operate but the latched values of all of the clock counters will not change. This allows the clock counters to be read.

For example, the seconds counter register can be accessed and read by first setting the RAM bank to 8, then reading from any CPU address between A000h and BFFFh.

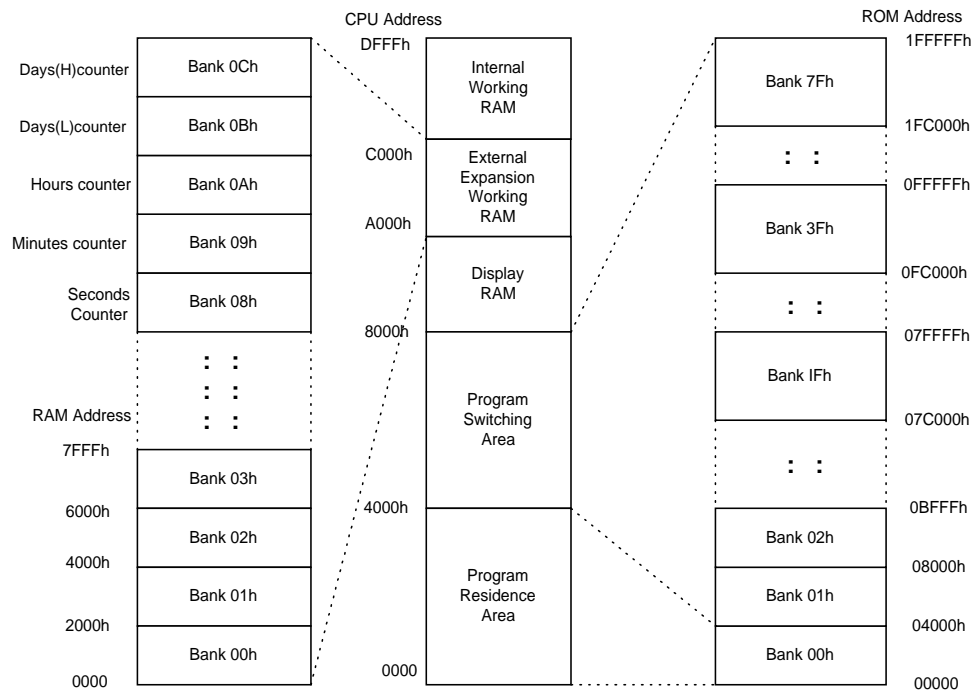
#### 3.3.2 Writing

Writing 0Ah to register 0 opens the access gate, allowing each clock counter register to be written to.

### 3.4 Memory Map

- ROM bank 0 is assigned to the program residence area (0000h-3FFFh) of the CPU memory space (unchangeable).
- One bank from among ROM banks 01h-7Fh can be assigned to the program switching area (4000h-7FFFh) of the CPU memory space.

- One bank from among RAM banks 0-3 and the clock counter registers (RAM banks 08h-0Ch) can be assigned to the external expansion working RAM area (A000h-BFFFh) of the CPU memory space.



### 3.5 Programming Cautions

#### 3.5.1 Accessing the Clock Counters

Although counting up of the clock counters themselves and accessing the clock counters from the CPU are performed asynchronously, clock counter failure may result if both operations are performed at the same time. To prevent this, MBC3 provides an interface circuit for WR signals from the CPU. Use of this circuit necessitates a delay when accessing control register 3 and the clock counter registers (RTC\_S, RTC\_M, RTC\_H, RTC\_DL, and RTC\_DH). Thus, whenever accessing these registers consecutively, interpose a delay of 4 cycles between accesses.

When reading clock counter data:

- Latch all clock counter data using control register 3.
- ↓ 4-cycle delay required
- Read the data in the clock counter registers.

When writing values to the clock counters:

- Set data in clock counter register RTC\_S.  
↓ 4-cycle delay required
- Set data in clock counter register RTC\_M.  
↓ 4-cycle delay required
- Set data in clock counter register RTC\_H.  
↓ 4-cycle delay required
- Set data in clock counter register RTC\_DL.  
↓ 4-cycle delay required
- Set data in clock counter register RTC\_DH.

### 3.5.2 Condensation

MBC3 uses a crystal oscillator for its clock counter operation, and condensation on the oscillator may halt its oscillation, preventing the clocks from counting up. Once the condensation disappears, the clocks will resume counting up from where they stopped. However, please ensure that the counter stoppage does not result in a loss of program control.

### 3.5.3 Control Register Initialization

Although control registers 0-3 are initialized (see Section 3.2, *Description of Registers*) when Game Boy power is turned on, they are not initialized by a hard reset of SNES when Super Game Boy is used. Therefore, please be sure to implement a software reset of these registers.

### 3.5.4 Clock Counter Registers

When commercial Game Boy software that uses MBC3 is shipped from the factory, the values of the clock counter registers are undefined. Therefore, please ensure that these registers are initialized.

## 4. MBC5

### 4.1 Overview

Supports CGB double-speed mode.

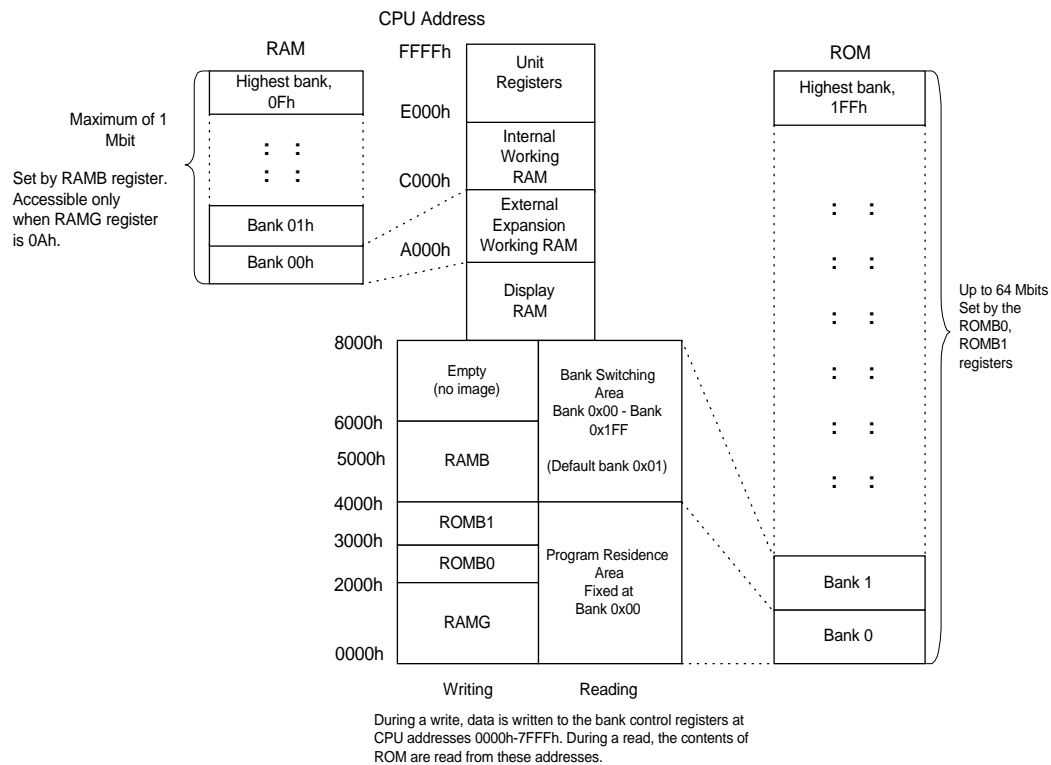
MBC5 can use up to 64 Mbits of ROM (512 banks of 128 bits each) and 1 Mbit of RAM (16 banks of 64 Kbits each).

Upwardly compatible with MBC1.

### 4.2 Registers

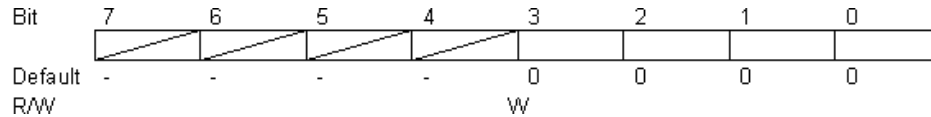
Name	Addresses (hex)
RAMG	0000-1FFF
ROMB 0	2000-2FFF
ROMB 1	3000-3FFF
RAMB	4000-5FFF

### 4.3 Memory Map

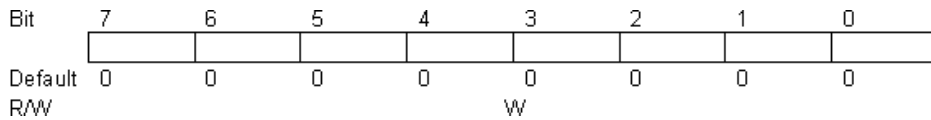


#### 4.4 Description of Registers

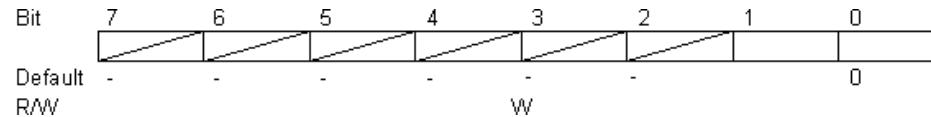
- Register for Specifying External Expansion Memory (RAMG)  
Specifies whether external expansion RAM is accessible. Access to this RAM is enabled by writing 0Ah to the RAMG register space, 0000h-1FFFh. Writing any other value to this register disables reading to and writing from RAM.



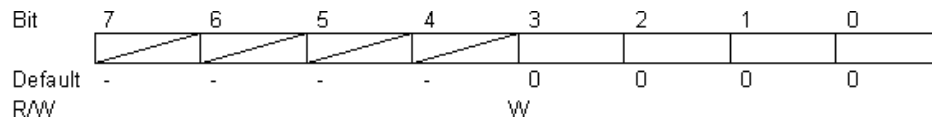
- Lower ROM Bank Register (ROMB0)  
Specifies the lower-order 8 bits of a 9-bit ROM bank.  
The ROM bank can be changed by writing the desired ROM bank number to the ROMB0 register area, 2000h-2FFFh.

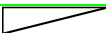


- Upper ROM Bank Register (ROMB1)  
Specifies the higher-order 1 bit of a 9-bit ROM bank.  
The ROM bank can be changed by writing the desired ROM bank number to the ROMB1 register area, 3000h-3FFFh.



- RAM Bank Register (RAMB)  
Specifies the RAM bank  
The RAM bank can be changed by writing the desired RAM bank number to the RAMB register area, 4000h-5FFFh.



**Note** Although the bits marked with  are ignored by MBC5, they should be used after being set to 0. The default values are set automatically when power is turned on.

## **4.5 Programming Cautions**

### **4.5.1 When Migrating from MBC1 to MBC5**

- Use of Register 1  
If an MBC1 program uses register 1 (ROM bank control register) addresses 3000h-3FFFh, the bank intended for selection by ROMB1 in MBC5 will not be selected. Addresses 2000h-2FFFh of register 1 should be used by programs that use MBC1.
- Use of Register 2  
Note that in MBC1, programs that use 8 Mbits or more use register 2 (ROM or RAM bank control register) for the high ROM bank. Consequently, in MBC5 the RAM bank is different while the ROM bank is unchanged.
- ROM Banks 20h, 40h, and 60h  
ROM banks 20h, 40h, and 60h cannot be used in MBC1, but they can be used in MBC5.
- MBC1 Register 3 (ROM/RAM change)  
Because the addresses of ROM and RAM are independent of each other in MBC5, ROM/RAM switching is unnecessary.  
Any write instructions to register3 left in a program that uses MBC1 are ignored by MBC5 and have no effect.

### **4.5.2 General Notes**

- Memory Image  
If a memory device is used that uses less than the maximum amount of memory available (ROM: 64 Mbits; RAM: 1 Mbit) , a memory image is generated for the empty bank area. Therefore, please do not develop software that uses an image, because it may cause failures.
- RAM Data Protection  
To protect RAM data, it is recommended that RAM access be disabled when RAM is not being accessed (RAMG ← 00h) .
- Specifying External Sound Input (VIN)  
Always use the sound control register (NR50) with bits 7 and 3 (VIN function OFF) set to 0. Because the VIN terminal is used in development flash ROM cartridges, using the register with VIN set to ON will produce sound abnormalities.

#### 4.6 Examples of MBC5 programs on DMG and CGB

- Set the bank switching area (4000h-7FFFh) to 1FFh.

```
LD A,$FF
LD ($2000),A    ;ROMB0 setting
LD A,$01
LD ($3000),A    ;ROMB1 setting
|
|
|
```

- Set the external expansion memory area (A000h-BFFFh) to 0Fh.

```
LD A,$0F
LD ($4000),A    ; RAMB setting
LD A,$0A
LD ($0000),A    ; Enable access to RAM
|               } RAM Access Processing
|               }
|               }
LD A,$00
LD ($0000), A    ; Disable access to RAM
```



## 5. MBC5 (WITH RUMBLE FEATURE)

### 5.1 Overview

This cartridge is the same as the previous MBC5 cartridge but also includes a rumble motor and size AAA battery to power the motor. The motor is controlled by the program using the MBC5 RAM bank register (RAMB, bit 3).

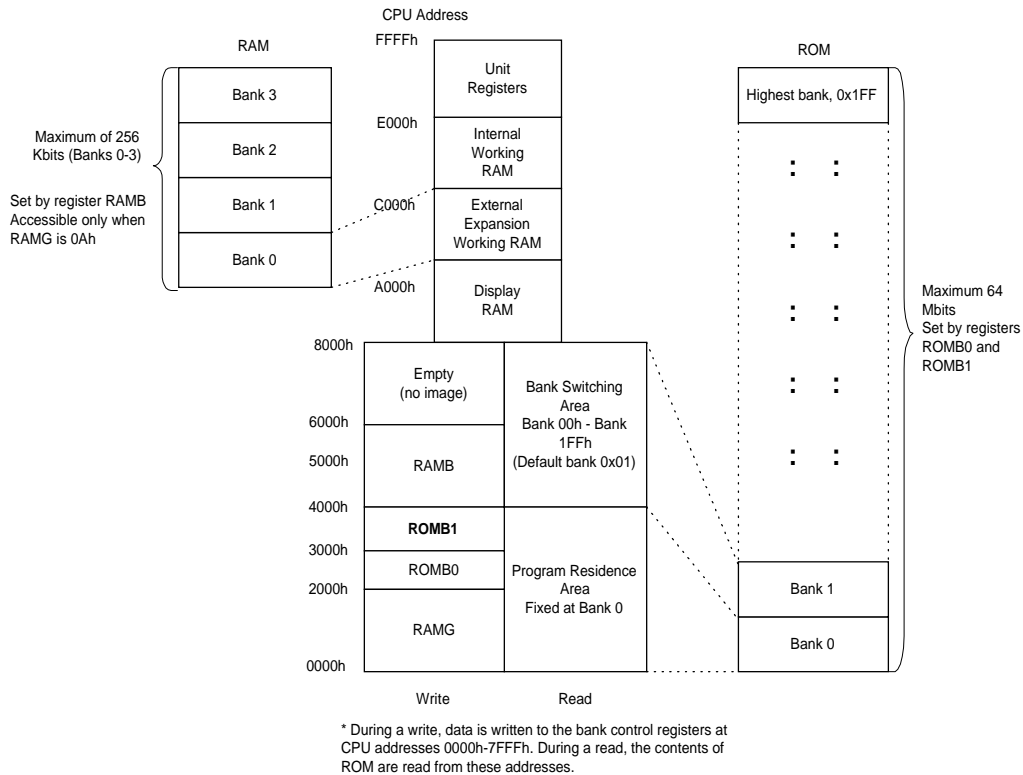
MBC5 supports CGB normal- and double-speed modes.

Up to 64 Mbits (512 banks of 128 Kbits each) of ROM and 256 Kbits of RAM (4 banks of 64 Kbits each) can be used.

### 5.2 Registers

Name	Addresses (hex)	Notes
RAMG	0000-1FFF	Each register executes its control using any one of the address spaces at left.
ROMB 0	2000-2FFF	
ROMB 1	3000-3FFF	
RAMB	4000-5FFF	

### 5.3 Memory Map











### 5.4 Description of Registers

- **Register for Specifying External Expansion Memory (RAMG)**  
Specifies whether external expansion RAM is accessible. Access to this RAM is enabled by writing 0Ah to the RAMG register (any single address in 0000h-1FFFh). Writing any other value to this register disables reading to and writing from RAM.

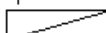
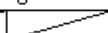
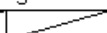
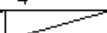
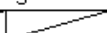
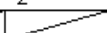
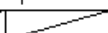

Bit	7	6	5	4	3	2	1	0
Default	0	0	0	0	0	0	0	0
R/W					W			

- **Lower ROM Bank Register (ROMB0)**  
Specifies the lower-order 8 bits of a 9-bit ROM bank. The ROM bank can be changed by writing the desired ROM bank number to the ROMB0 register (any single address in 2000h-2FFFh).

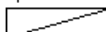
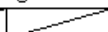
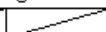
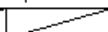
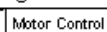
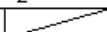


## Game Boy Programming Manual

Bit	7	6	5	4	3	2	1	0
								
Default	0	0	0	0	0	0	0	0
R/W					W			


- Upper ROM Bank Register (ROMB1)  
Specifies the higher-order 1 bit of a 9-bit ROM bank.  
The ROM bank can be changed by writing the desired ROM bank number to the ROMB1 register (any single address in 3000h-3FFFh).

Bit	7	6	5	4	3	2	1	0
								
Default	-	-	-	-	-	-	-	0
R/W					W			

- RAM Bank Register (RAMB)  
Specifies the RAM bank.  
The RAM bank can be changed by writing the desired RAM bank number to the RAMB register (any single address in 4000h-5FFFh).

Bit	7	6	5	4	3	2	1	0
								
Default	-	-	-	-	0	-	0	0
R/W					W			

Bits 0-1: Register for RAM bank setting  
Bit 3: Motor control register (1: motor ON; 0: motor OFF)

**Note:** Be sure to set the bits marked with  to 0 before using them.  
The default values are set automatically when power is turned on.

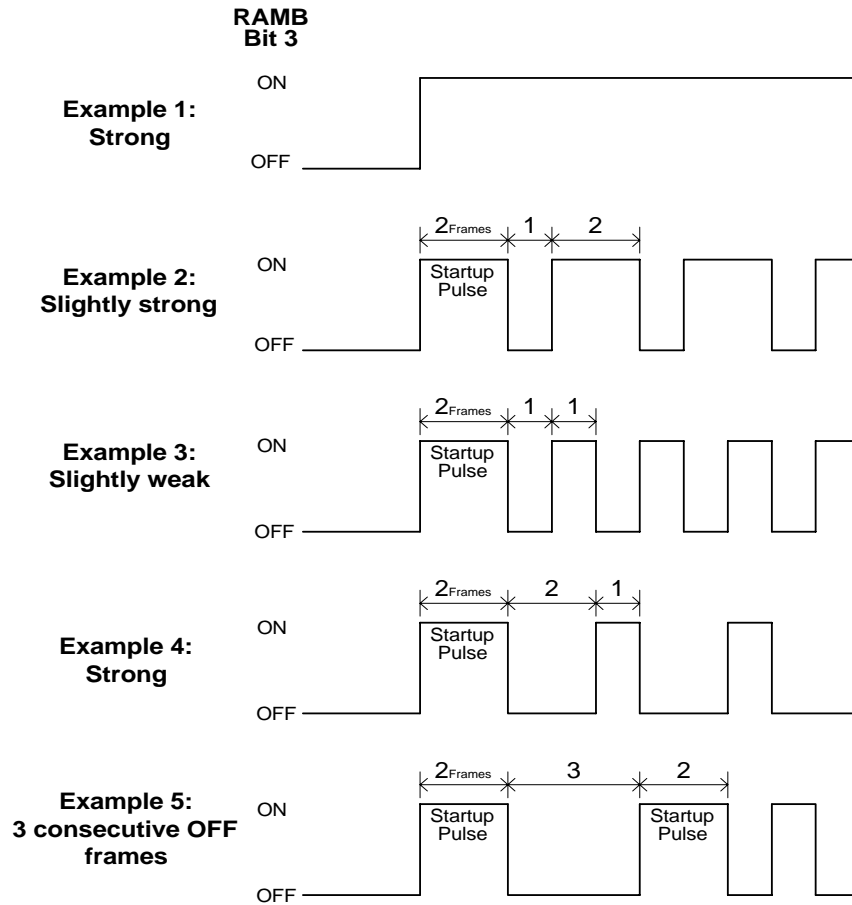
## 5.5 Motor Control

### 5.5.1 Vibration Level

Control of the rumble motor consists of setting it to ON or OFF.  
The vibration level can be controlled by sending pulses of combined ON/OFF instructions in short cycles. Please comply with the following points when implementing vibration control.

- (1) Set the frame rate to 1 frame per 1/60 second and control vibration frame by frame.
- (2) At the start of vibration control, send a startup pulse (at least 2 ON frames).  
A startup pulse also should be sent if the width of an OFF pulse is 3 or more consecutive frames. This is necessary because startup from a complete stop requires a certain amount of time. (see Ex. 5)

### 5.5.2 Vibration Pulse Examples



### 5.6 Programming Cautions

#### **IMPORTANT**

#### 5.6.1 Memory Image

If a memory device is used that uses less than the maximum amount of memory available (ROM: 64 Mbits; RAM: 256 Kbits) , an empty bank area (memory image) results. Please do not access this empty bank area. Doing so may result in faulty operation.

#### 5.6.2 RAM Data Protection

To protect RAM data, it is recommended that RAM access be disabled (RAMG 00h) when RAM is not being accessed.

### **5.6.3 Specifying External Sound Input (VIN)**

Always use the sound control register (NR50) with bits 7 and 3 set to 0 (VIN function OFF). Because the VIN terminal is used in development flash ROM cartridges, using the register with VIN set to ON will produce sound abnormalities.

### **5.6.4 Disabling Vibration Using the SGB, SGB2, or 64GB Pak**

When MBC5 is used by SGB, SGB2, or the 64GB Pak, vibration should be turned off by the program to prevent failures caused by a faulty connection. For methods of recognizing SGB and SGB2, see the description of the MLT\_REQ command in Chapter 6, Section 3.2, *System Command Details*. With the 64GB Pak, vibration is controlled by the N64 software. Therefore, N64 software programs that support MBC5 should not write data to bit 3 of the RAM bank register.

### **5.6.5 Limiting the Period of Continuous Vibration**

To prevent physical effects in the user such as numbness as a result of continuous vibration, limit the duration of continuous vibration as indicated below, regardless of the vibration strength (see Section 6.5.2, *Vibration Pulse Examples*).

- The duration of continuous vibration should generally be limited to a maximum of 1 minute.
- The period of no vibration between the finish of one period of vibration and the start of the next period generally must be at least as long as the vibration time.

The above points are guidelines that should be followed in most cases. However, if adhering to these guidelines is made difficult by factors such as the game content, take appropriate measures while keeping in mind the points noted in Section 6.7, *Effects of Vibration on the Body*.

### **5.6.6 Disabling Vibration for Resets and Pauses**

Vibration should be halted during resets and pauses.

When power is turned on, the unit should not be vibrated until some input is received from the controller.

### **5.6.7 Rumble Feature Selection**

The user should be allowed to set the rumble feature to ON or OFF or to select strong, mild, or OFF by means such as an initial-settings screen at the start of the game. In addition, the program should allow the user to easily change these settings even during a game if, for example, they are uncomfortable with the vibration. Such changes also should be allowed a pause.

### **5.6.8 Changes in Vibration Level with Battery Use**

If the battery that powers the motor (Size AAA alkaline battery) wears out, the perceived vibration level will be reduced even if the requested vibration level remains the same.

Therefore, rumble operation should be checked both when the battery is new (1.6 V) and when it is at the end of its life (1.1 V).

### **5.7 Physical Effects of Vibration on the Body**

Users have occasionally experienced numbness for some time after continuous vibration lasting several tens of seconds to several minutes. This may occur regardless of the strength of the vibration (see Section 6.5.2, *Vibration Pulse Examples*).

Unfortunately, the effects of continuous vibration on the body are not yet clear. Thus, the guidelines presented in Section 6.6.5, *Limiting the Period of Continuous Vibration*, are intended to give priority to user safety. However, software development requires free thinking and original ideas, and there may well be cases in which the use of continuous vibration in a game is desirable.

Because each game is different, the limitations presented in Section 6.6.5 are by their nature not restrictions that should be enforced digitally. It is instead preferable for the developer to adequately consider user safety when determining the game's content.

For example, even supposing that continuous vibration does last for more than 1 minute, it may not pose a safety problem if it is used infrequently, such as only when special events occur. Conversely, if vibrations lasting several seconds to several tens of seconds are repeated at short intervals, the effects on the user may be the same as with continuous, long-term vibration.

Thus, the guidelines presented in Section 6.6.5 are not absolute restrictions. However, even if a program varies from these guidelines, the following points should be considered minimum requirements and strictly observed.

- Continuous vibration should not exceed 3 minutes for any reason.
- Because the effects of continuous vibration vary from person to person, the strength of these effects on the user should not be determined independently by the developer. Rather, this determination should be arrived at after considering the opinions of many others during debugging and other phases of development.

***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

<b>APPENDIX 1: PROGRAMMING CAUTIONS .....</b>	<b>254</b>
<b>1. USING THIS APPENDIX.....</b>	<b>254</b>
<b>2. PROGRAMMING CAUTIONS REGARDING GAME BOY .....</b>	<b>255</b>
2.1 LCDC/VRAM .....	255
Reference Notes: .....	256
2.2 Communication.....	256
2.3 Sound.....	256
2.4 Miscellaneous Notes .....	257
<b>3. PROGRAMMING CAUTIONS REGARDING MBCs .....</b>	<b>260</b>
3.1 All MBCs .....	260
3.2 MBC3.....	260
3.3 MBC5.....	261
<b>4. SGB PROGRAMMING CAUTIONS .....</b>	<b>263</b>
4.1 ROM Data (Required).....	263
4.2 Default Data (Required).....	263
4.3 SOU_TRN Default Data (Required).....	263
<b>5. PROGRAMMING CAUTION REGARDING POCKET PRINTER..</b>	<b>264</b>
5.1 Transfer Time Intervals (Required).....	264
5.2 Printing Multiple Sheets Continuously (Recommended) .....	264
5.3 Print Density (Recommended).....	264
5.4 Operation After the Motor is Stopped (Required) .....	264
5.5 Feeds (Required).....	264
5.6 Point of Caution During Debugging (Recommended) .....	264
5.7 Sample Program Provided by Nintendo (Recommended) .....	264
<b>6. PROGRAMMING CAUTIONS FOR U.S. PROGRAMMERS.....</b>	<b>265</b>



## **APPENDIX 1: PROGRAMMING CAUTIONS**

### **1. USING THIS APPENDIX**

#### Purpose and Scope

These programming notes provide information on how to avoid easily made mistakes during program development, information on unique Game Boy programming issues that require special attention, and special issues regarding peripheral devices.

#### Items Covered in this Manual

Many of the topics covered in this appendix also are covered elsewhere in different chapters of this manual. This appendix consolidates the discussion of these topics. Topics that would be more easily comprehensible to the reader when presented separately will also be discussed in another chapter, even though this may duplicate the discussion in this appendix.

**Note:** Although these notes were created to make every effort to eliminate potential sources of trouble once on the market, they do not represent a guarantee that various potential problems can be completely avoided.

## 2. PROGRAMMING CAUTIONS REGARDING GAME BOY

Covers:

DMG: DMG, MGB, and MGL

SGB: SGB and SGB2

CGB: CGB

### 2.1 LCDC/VRAM

#### 2.1.1 Setting the LCDC to OFF (Recommended)

Covers: DMG and CGB

In early DMGs, a black horizontal line appears on the screen if the LCDC is stopped (LCDC register bit 7  $\leftarrow$  0) at any time other than during vertical blanking. Therefore, the LCDC should be set to OFF during V-blanking. If the occurrence of V-blanking cannot be confirmed, the LCDC should be set to OFF when the value of the LY register is 145 (91h) or greater. These restrictions do not apply in CGB. Thus, when creating software for use on CGB only, the timing of setting the LCDC to OFF need not be considered.

#### 2.1.2 Window x-coordinate Register (Required)

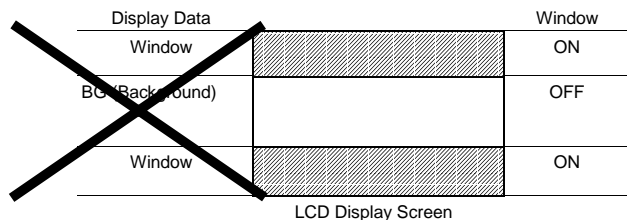
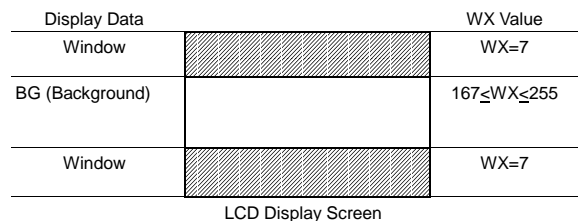
Covers: DMG, SGB, and CGB

When the window is displayed, the window x-coordinate register (register WX, address FF4Bh) must be set in the range 7-165. A setting of 0-6 or 166 is prohibited. Specifying a value of 167 or greater causes the window not to be displayed.

#### 2.1.3 Displaying Multiple Windows (Required)

Covers: CGB

Multiple windows that divide the screen horizontally into upper and lower areas can be displayed by setting the window x-coordinate register (WX) to a value of 167 or greater during a horizontal blanking period. Attempting to display multiple windows by switching the window ON and OFF during H-blanking may result in the lower window not being displayed.



**Reference Notes:**

1. Accessing VRAM Outside of a V-blanking period  
In early DMGs, accessing VRAM outside of a V-blanking period would corrupt the screen.
2. Length of H-blanking  
The length of the H-blanking period changes depending on the conditions of OBJ use, so caution is recommended when using H-blanking.

**2.2 Communication**

**2.2.1 Communication Rate (Required)**

Covers: DMG, SGB2, and CGB  
Data may be corrupted if the data transfer rate is too high.  
The maximum external clock setting should be 512KHz between CGBs.  
It should be 500KHz for others including DMG and SGB2.  
Also, it should be 256KHz for communication between CGB and DMG.

**2.2.2 Communication Errors (Recommended)**

Covers: DMG, SGB2, and CGB  
When using the communication function (infrared), the communicating data may be corrupted by noise and such. Therefore, the program should not go out of control by such data corruption on both the sending and receiving side.  
When using the communication function (serial), depending on how the program is made, it is confirmed that communication errors happen rarely.  
SIO interrupt processing may be delayed by factors such as the processing of other interrupts.  
This type of error should be avoided by establishing a proper communication interval that allows a problem-free exchange of data.

**2.2.3 Effects of Other Infrared Devices (Recommended)**

Covers: CGB  
Adequate care should be taken to ensure against faulty operation and loss of program control even when infrared communication signal input is received from other game software and devices.  
Note that such problems may particularly occur in communication between multiple games that use the same subroutines. (Before performing data communication, use means such as exchanging a unique key code to check whether the same game is on the other hardware.)

**2.3 Sound**

**2.3.1 Using Sounds 1, 2, and 3 (Required)**

Covers: CGB  
With continuous operation mode selected (bit 6 of NR\*4 set to 0) for sounds 1, 2, and 3, if the higher-order frequency data (lower-order 3 bits of NR\*4) are changed, the sound length (bits 0-5 of NR\*1) must be set to 0 after the frequency data is set. If the sound length is not set to 0, the sound may stop during playback.

**2.3.2 Using Sound 3 (Required)**

Covers: DMG, SGB, and CGB

When sound 3 is used, data should always first be specified for addresses FF30h-FF3Fh of waveform RAM. If the initial flag is set during sound 3 operation (sound 3 ON flag = 1), the contents of waveform RAM will be destroyed.

## **2.4 Miscellaneous Notes**

### **2.4.1 Using Interrupts (Required)**

Covers: DMG, SGB, and CGB

When interrupts are used, the IF register should be cleared before the IE register is set. If the IF register is not first cleared, an interrupt may be generated immediately after interrupts are enabled.

### **2.4.2 Reading Keys (Required)**

Covers: DMG

An interval of approximately 18 cycles should be used between output from P14 and P15 and reading of input. Without this interval, normal key input cannot be read.

### **2.4.3 Using the Timer (Required)**

Covers: DMG, SGB, and CGB

The timer should be started (TAC start flag set) after the count-up pulse is selected. Starting the timer before or at the same time as the pulse is selected may result in an extra count-up operation at the time of pulse selection.

Example:

```
LD  A,3      ;Selects f/256 as the count-up pulse.
LD  (07),A   ;Sets TAC ← 3
LD  A,7      ;Starts the timer
LD  (07),A
```

If a write instruction is executed for the modulo register TMA with the same timing as the contents of that register are transferred to TIMA as a result of a timer overflow, the same write data also will be transferred to TIMA..

### **2.4.4 Using STOP Mode (Required)**

Covers: DMG, SGB, and CGB

When STOP mode is used, all interrupt-enable (IE) flags should be reset before execution of a STOP instruction. Otherwise, if an interrupt is generated during the period of oscillation stabilization (HALT mode) following STOP mode cancellation, HALT mode will immediately be canceled, preventing a stable system clock from being provided.

### **2.4.5 Using Paired Registers (Required)**

Covers: DMG, SGB, and CGB

With instructions that use paired registers BC, DE, and HL, such as the following, there is some chance that OAM RAM may be destroyed. Therefore, ensure that these paired registers are not set to a value in the range FE00h-FE9Eh.

```
INC  ss      ; ss : BC, DE, HL
DEC  ss
LD   A,(HLI)
```

```
LD  A,(HLD)
LD  (HLI),A
LD  (HLD),A
```

#### **2.4.6 Using the HALT Instruction (Required)**

Covers: DMG, SGB, and CGB

When using a HALT instruction, always add an NOP instruction immediately after the HALT instruction. Not adding the NOP instruction may in rare cases cause the instruction after the HALT instruction not to be executed.

#### **2.4.7 Switching the CPU Operating Speed (Recommended)**

Covers: CGB

When switching the CPU operating speed, first confirm the current speed by checking the speed flag (bit 7 of register KEY1). In double-speed mode, both the divider (DIV) and timer (TIMA) registers will also be set for double-speed operation.

#### **2.4.8 Using Horizontal Blanking DMA (Required)**

Horizontal blanking DMA should always be started (bit 7 of HDMA5 set to 1) when the STAT mode is not set to 00. If horizontal blanking DMA is started when STAT mode is 00, depending on the timing, the data in LCD display RAM may be destroyed. In addition, execution of a HALT instruction during horizontal blanking DMA may prevent normal cancellation of the HALT mode or DMA. Therefore, HALT instructions should not be used while horizontal blanking DMA is being started.

#### **2.4.9 Using General-Purpose DMA (Required)**

General-purpose DMA should be started (bit 7 of HDMA5 set to 0) with the LCDC off or during V-blanking. However, when transferring data during V-blanking, ensure that the transfer period does not overlap with STAT modes 10 or 11.

#### **2.4.10 DMA Transfers to OAM (Required)**

In DMG and in CGB in DMG mode, when transferring data to OAM by DMA, the user program area (00h-7FFFh) should not be used as the starting address of the transfer. In some cases, data cannot be transferred from the user program area normally. CGB mode, however, does enable DMA transfers from the user program area.

#### **2.4.11 Status Interrupts (Required)**

Covers: DMG, SGB, and CGB

When using a status interrupt in DMG or in CGB in DMG mode, register IF should be set to 0 after the value of the STAT register is set. (In DMG, setting the STAT register value changes the value of the IF register, and an interrupt is generated at the same time as interrupts are enabled.)

**2.4.12 Chattering (Recommended)**

Covers: DMG, SGB, and CGB

To prevent buttons from inadvertently being pressed twice, an interval should be provided between key reads. (Although this varies with the software, keys are normally read approximately once per frame.)

### 3. PROGRAMMING CAUTIONS REGARDING MBCS

#### 3.1 All MBCs

##### 3.1.1 Protecting RAM Data (Recommended)

To protect RAM data, access to RAM should be disabled (RAMG←00h) when it is not being accessed.

#### 3.2 MBC3

##### 3.2.1 Accessing the Clock Counters (Required)

If the clock counters themselves are counted up, accessing of the clock counters by the CPU is performed asynchronously. However, if these operations are performed simultaneously, the clock counters may fail. To prevent this, MBC3 provides an interface circuit for WR signals from the CPR. Use of this circuit necessitates a delay when accessing control register 3 and the clock counter registers (RTC\_S, RTC\_M, RTC\_H, RTC\_DL, and RTC\_DH). Thus, whenever accessing these registers consecutively, interpose a delay of 4 cycles between accesses.

*When reading clock counter data:*

- Latch all clock counter data using control register 3.



4-cycle delay required

- Read the data in the clock counter registers.

*When writing values to the clock counters:*

- Set data in clock counter register RTC\_S.



4-cycle delay required

- Set data in clock counter register RTC\_M.



4-cycle delay required

- Set data in clock counter register RTC\_H.



4-cycle delay required

- Set data in clock counter register RTC\_DL.



4-cycle delay required

- Set data in clock counter register RTC\_DH.

### **3.2.2 Condensation (Required)**

MBC3 uses a crystal oscillator for its clock counter operation, and condensation on the oscillator may halt its oscillation, preventing the clocks from counting up. Once the condensation disappears, the clocks will resume counting up from where they stopped. However, please ensure that the counter stoppage does not result in a loss of program control.

### **3.2.3 Control Register Initialization (Required)**

Although control registers 0-3 are initialized (see Section 3.2, Description of Registers) when the Game Boy power is turned on, they are not initialized by a hard reset of Super NES when Super Game Boy is used. Therefore, please be sure to implement a software reset of these registers.

### **3.2.4 Clock Counter Registers (Required)**

When commercial GB software that use MBC3 are shipped from the factory, the values of the clock counter registers are undefined. Therefore, please ensure that these registers are initialized.

## **3.3 MBC5**

### **3.3.1 Memory Image (Required)**

If a memory device is used that uses less than the maximum amount of memory available (ROM: 64 Mbits; RAM: 1 Mbit) , a memory image is generated for the empty bank area. Therefore, please do not develop software that uses an image, because it may cause failures.

### **3.3.2 Specifying External Sound Input (VIN) (Required)**

Always use the sound control register (NR50) with bits 7 and 3 (VIN function OFF) set to 0. Because the VIN terminal is used in development flash ROM cartridges, using the register with VIN set to ON will produce sound abnormalities.

### **3.3.3 Disabling Vibration Using the SGB, SGB2, or 64GB Pak (Recommended)**

When MBC5 with rumble feature is used by SGB, SGB2, or the 64GB Pak, vibration should be turned off by the program to prevent failures caused by a faulty connection. For methods of recognizing SGB and SGB2, see Chapter 6, section 4.2, Recognizing SGB. With the 64GB Pak, vibration is controlled by the N64 software. Therefore, N64 software programs that support MBC5 should not write data to bit 3 of the RAM bank register.

### **3.3.4 Disabling Vibrations for Resets and Pauses (Recommended)**

Vibration should be halted during resets and pauses.  
When power is turned on, the hardware should not be vibrated until some input is received from the controller.



### **3.3.5 Limiting the Period of Continuous Vibration (Recommended)**

To prevent physical effects in the user such as numbness as a result of continuous vibration, limit the duration of continuous vibration as indicated below, regardless of the vibration strength.

- Limit the duration of continuous vibration to 1 minute.
- If the nature of the game makes longer periods of continuous vibration unavoidable, limit these periods to 3 minutes.

### **3.3.6 Rumble Feature Selection (Recommended)**

The user should be allowed to set the rumble feature to ON or OFF or to select strong, mild, or OFF by means such as an initial-settings screen at the start of the game. In addition, the program should allow the user to easily change these settings even during a game if, for example, they are uncomfortable with the vibration. Such changes also should be allowed a pause.

### **3.3.7 Changes in Vibration Level with Battery Use (Recommended)**

If the battery that powers the motor (size AAA alkaline battery) wears out, the perceived vibration level will be reduced even if the requested vibration level remains the same. Therefore, rumble operation should be checked both when the battery is new (1.6 V) and when it is at the end of its life (1.1 V).

## **4. SGB PROGRAMMING CAUTIONS**

### **4.1 ROM Data (Required)**

To use the functions of SGB (system commands), the following values must be stored in ROM at the locations indicated.

146h ← 03h, 14Bh ← 33h

### **4.2 Default Data (Required)**

When writing programs that use the functions of SGB, use the initialization routine of the game program to send default data (see Chapter 6) to the register file.

### **4.3 SOU\_TRN Default Data (Required)**

When using the SOU\_TRN system command, send the SOU\_TRN default data (see Chapter 6) to the register file before SOU\_TRN is used.

## **5. PROGRAMMING CAUTIONS REGARDING POCKET PRINTER**

### **5.1 Transfer Time Intervals (Required)**

Transfer time intervals vary depending on the manufacturer. The timings indicated in Chapter 9 should be used to avoid faulty operation with a printer from a particular manufacturer.

### **5.2 Printing Multiple Sheets Continuously (Recommended)**

Between 2 and 255 sheets can be printed continuously by an application. However, because this may take a long time, the user should be given a means of halting a print job in progress.

### **5.3 Print Density (Recommended)**

Because it is very inconvenient to adjust the density each time the program is started, the print density data should be backed up whenever possible.

### **5.4 Operation After the Motor is Stopped (Required)**

If a print instruction packet is sent within 100 msec of when the motor is stopped, the print starting position may be incorrect. Therefore, print instruction packets should always be sent at least 100 msec after the motor is stopped.

### **5.5 Feeds (Required)**

In setting the number of line feeds to be inserted before and after printing (byte 2 of the data portion of the print instruction packet), always specify a value of 1 or greater for the number of feeds before printing and 3 or greater for the number after printing. Otherwise, problems can arise, such as double printing twice on a single line or failure of the last line of print to reach the paper cutter.

### **5.6 Point of Caution During Debugging (Recommended)**

There are two types printers, each made by a different manufacturer (Seiko Instruments and Hosiden). As part of final debugging, the program should be checked with at least one printer of each type.

### **5.7 Sample Program Provided by Nintendo (Recommended)**

Modifying the program to suit the intended use is permitted. However, in creating the original program, values for timing and other parameters were calculated to allow normal operation. These parameters must therefore be carefully considered when modifying the program.

## **6. PROGRAMMING CAUTIONS FOR U.S. PROGRAMMERS**

If you are unable to verify that the system is a Super Game Boy, and the Accumulator returns the same value if the game is inserted in the Super Game Boy or original Game Boy hardware, follow the instructions below.

If your game is Super Game Boy (SGB) enhanced, then you just need to use the MLT\_REQ function. Otherwise, you must use the SGB libraries to verify if the game is in an SGB. (These libraries are located in the CGB files section of Wario World under SGBlib.zip.) You will need to call the SGBCHK function from these libraries right after the Soft Reset label. To use this function, you must set the ROM Registration area for SGB (\$146h) to \$03, which allows access to the SGB REgisters. (Don't forget to readjust the Complement Check.)

Also, on the Software Submission sheet, make sure you note that the game has a \$03 in address \$146, but in the remarks section, explain that the game doesn't use any of the SGB features.

***THIS PAGE WAS INTENTIONALLY LEFT BLANK.***

**APPENDIX 2: REGISTER AND INSTRUCTION SET SUMMARIES.. 268**

- 1. CONTROL REGISTER SUMMARY..... 268**
- 2. SOUND REGISTER SUMMARY..... 273**
- 3. CPU INSTRUCTION SET SUMMARY..... 277**

## APPENDIX 2: REGISTER AND INSTRUCTION SET SUMMARIES

### 1. CONTROL REGISTER SUMMARY

Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
P1  Port P15- P10	FF00	/	/	P15	P14	P13	P12	P11	P10	R/W Control of transfer data by P14, P15
SB  Serial transfer register	FF01									R/W  Transfer data
SC  Serial control	FF02	Transfer start 0: No start 1: Start	/	/	/	/	/	Clock speed 0: 8 KHz 1: 256 KHz	Shift clock 0: External 1: Internal	R/W In double- speed mode clock speed also doubled
DIV  Divider	FF04	$f/2^{16}$ 64Hz	$f/2^{15}$ 128Hz	$f/2^{14}$ 256Hz	$f/2^{13}$ 512Hz	$f/2^{12}$ 1024Hz	$f/2^{11}$ 2048Hz	$f/2^{10}$ 4096Hz	$f/2^9$ 8192Hz	R/W With clear normal by LD instruction: f=4194304 Double speed: f=8388608
TIMA  Timer	FF05									R/W Timer unit. Operates at double-speed in double- speed mode
TMA  Timer modulo	FF06									R/W Preset register for timer
TAC  Timer control	FF07	/	/	/	/	/	Timer stop 0: Stop 1: Operate	Frequency selection bit 00: $f/2^{10}$ 10: $f/2^6$ 01: $f/2^4$ 11: $f/2^8$		R/W Normal-speed: f=4194304 Double-speed: f=8388608
IF  Interrupt request flag	FF0F	/	/	/	Terminals P10-P13 HIGH	End of serial transfer	Timer overflow	LCDC Controller STAT	V-blank	R/W Bit reset valid
IE  Interrupt enable flag	FFFF				Terminals P10-P13 LOW	End of serial transfer	Timer overflow	LCDC Controller STAT	V-blank	R/W 0: Disabled 1: Enabled

**Appendix 2. Register and Instruction Set Summaries**

Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
IME Interrupt master enable										Reset with DI; set with EI 0: Disable interrupts 1: Enable interrupts
LCDC LCDC Control	FF40	Controller 0: Stop 1: Operate	WIN Area 0: 9800- 1: 9C00-	Window 0: OFF 1: ON	BG Characters 0: 8800- 1: 8000-	BG Area 0: 9800- 1: 9C00-	OBJ Block 0: 8x8 1: 8x16	OBJ Display 0: OFF 1: ON	BG Displa y 0: OFF 1: ON	R/W Bit 0 fixed to display BG ON in CGB mode only
STAT LCDC status information	FF41		LCDC status interrupt selection flags				LYC agreement 0: 1: LYC=LY	Mode 00: RAM access 10: OBJ search 01: V-blank 11: LCD transfer		R/W Bits 3-6 Interrupt 0: Not selected 1: Selected
SCY Scroll Y register	FF42									R/W 00h – FFh
SCX Scroll X register	FF43									R/W 00h – FFh
LY LCDC y- coordinate	FF44									R y-coordinate during display
LYC LY compare register	FF45									R/W Agreement flag set with LYC=LY
DMA DMA Transfer	FF46									W 00h – DFh Transfer starts at the same time as address set



**Game Boy Programming Manual**

Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
BGP BG Palette Data	FF47	Palette data for character dot data 11 in DMG mode.		Palette data for character dot data 10 in DMG mode.		Palette data for character dot data 01 in DMG mode.		Palette data for character dot data 00 in DMG mode.		W
OBP0 OBJ palette data 0	FF48	Palette data for character dot data 11 in DMG mode.		Palette data for character dot data 10 in DMG mode.		Palette data for character dot data 01 in DMG mode.		Palette data for character dot data 00 in DMG mode.		W When attribute bit 4 is 0.
OBP1 OBJ palette data 1	FF49	Palette data for character dot data 11 in DMG mode.		Palette data for character dot data 10 in DMG mode.		Palette data for character dot data 01 in DMG mode.		Palette data for character dot data 00 in DMG mode.		W When attribute bit 4 is 1.
WY Window y-coordinate	FF4A									R/W 0 – 143 Top edge when WY=0
WY Window x-coordinate	FF4B									R/W 7 – 165 Left edge when WY=7
KEY1 CPU speed switching	FF4D	Current speed: 0: Normal 1: Double-speed								Enable speed switching  R/W Switch by setting bit 0 to 1 and issuing a STOP instruction
VBK VRAM bank specification	FF4F								Bank 0: Bank 0 1: Bank 1	R/W Bank 0 selected immediately after a reset signal.
HDMA1 Higher-order address of HDMA transfer source	FF51									W 00h – 7Fh (ROM) A0h – DFh (WRAM)
HDMA2 Lower-order address of HDMA transfer source	FF52									W 0Xh – FXh
HDMA3 Higher-order address of HDMA transfer destination	FF53									W 00h – 1Fh

Appendix 2. Register and Instruction Set Summaries

Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
HDMA4 Lower-order address of HDMA transfer destination	FF54									W 0Xh – FXh
HDMA5 H-blank and general- purpose DMA control	FF55	DMA selection 0: General purpose 1: H- blank	$0 \leq n \leq 127$ Total number of transferred bytes: $16 \times (n+1)$							W H-blanking stopped by setting bit 7 to 0; General- purpose stopped by resetting
RP Infrared communicatio n port	FF56	Data read-enable flag 00: Disable 11: Enable						Read data 0: LED-ON 1: LED-OFF	Write data 0: LED-OFF 1: LED-ON	R/W
BCPS Color palette BG write specification	FF68	Increment 0: OFF 1: ON		Palette No. 0 - 7			Palette No. 0 - 3		H/L specification 0: L 1: H	R/W Not incremented automatically with a read
BCPD Color palette BG write data	FF69									R/W
OCPS Color palette OBJ write specification	FF6A	Increment 0: OFF 1: ON		Palette No. 0 - 7			Palette No. 0 - 3		H/L specification 0: L 1: H	R/W Not incremented automatically with a read
OCPD Color palette OBJ write data	FF6B									R/W

**Game Boy Programming Manual**

Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
SVBK WRAM Bank specification	FF70						Banks specification 0,1: Specifies bank 1 2-7: Specifies banks 2-7			R/W
OBJ0 <sup>*1</sup> LCD y-coordinate	FE0 0									R/W 00h – FFh Top edge when Y=10h
LCD x-coordinate	FE0 1									R/W 00h – FFh Left edge when X=08h
Character code	FE0 2									R/W 00h – FFh
Attribute flag	FE0 3	Display priority 0: OBJ 1: BG	Vertical flip 0: Normal 1: Flip	Horizontal flip 0: Normal 1: Flip	Palette specification for DMG mode	VRAM bank 0: bank 0 1: bank 1	Color Palette No. 0 - 7			R/W
<sup>*1</sup> OBJ1 - OBJ39 same as OBJ0										

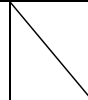
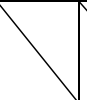
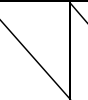
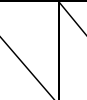
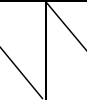
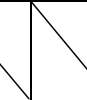
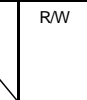







The dark frame  indicates a flag or register unique to CGB.

## 2. SOUND REGISTER SUMMARY

All values shown in the following table apply to normal mode. The values for double-speed mode should be calculated by doubling the system clock frequency.

	Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment	
S O U N D  1	NR10	FF10		Sweep time: 010:15.6ms 101:39.1ms 000:OFF 011:23.4ms 110:46.9ms 001:7.8ms 100:31.3ms 111:54.7ms			Sweep increase /decrease 0: + f hi 1: - f low	Number of sweep shifts: 0 - 7			R/W f <sub>128</sub> =128Hz	
	NR11 Duty cycle/sound length	FF11	Waveform duty cycle 00: 12.5% 10: 50% 01: 25% 11: 75%		Sound length data t1 : 0 - 63 Sound length = (64-t1) * (1/256) sec							R/W
	NR12  Envelope	FF12	Initial envelope value: 0x00 – 0x0F Mute when 0x00 Maximum when 0x0F				Envelope U/D 0: Attenuate 1: Amplify	Number of envelope steps N = 0 - 7 Length of 1 step = N*(1/64) sec Envelope function stopped when N=0			R/W Initial value of 00 sets to OFF when in DOWN mode	
	NR13  Lower-order frequency data	FF13	Lower-order 8 bits of frequency data									W
	NR14  Higher-order frequency data/ other	FF14	Restart when initialize flag set to 1	Length selection 0: Consecutive 1: NR11				Higher order 3 bits of frequency data With x =11-bit frequency data, f = 4194304 / (4*2 <sup>3</sup> (2048-x)) Hz			R/W f = 64Hz - 131KHz	

**Game Boy Programming Manual**

	Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
S O U N D 2	NR21 Duty cycle/sound length	FF16	Waveform duty cycle 00: 12.5% 10: 50% 01: 25% 11: 75%		Sound length t1: 0 - 63 Sound length = (64-t1) * (1/256) sec						R/W
	NR22  Envelope	FF17	Initial envelope value 0x00 – 0x0F Mute when 0x00 Maximum when 0x0F				Envelope U/D 0: Attenuate 1: Amplify		Number of envelope steps N = 0-7 Length of 1 step = N*(1/64) sec Envelope function stops when N=0		R/W Initial value of 00 sets to OFF when in DOWN mode
	NR23  Lower-order frequency data	FF18	Lower-order 8 bits of frequency data,								W
	NR24  Higher- order /other frequency data	FF19	Restart when initialize flag set to 1	Length selection 0: Consecutive 1: NR21					Higher order 3 bits of frequency data With x =11-bit frequency data, f = 4194304 / (4*2 <sup>3</sup> (2048-x)) Hz		R/W f = 64Hz - 131KHz
S O U N D 3	NR30  Sound OFF	FF1A	Sound OFF 0: OFF 1: ON								R/W
	NR31  Sound length data	FF1B	Sound length data t1 : 0 - 255 Sound length = (256-t1) * (1/256) sec								R/W
	NR32  Output level	FF1C		Output level 00: Mute 10: 1/2 01: Max 11: 1/4							R/W
	NR33  Lower-order frequency data	FF1D	Lower-order 8 bits of frequency data								W
	NR34  Higher- order frequency data/other	FF1E	Restart when initialize flag set to 1	Length selection 0: Consecutive 1: NR31					Higher-order 3 bits of frequency data With x =11-bit frequency data, f = 4194304 / (4*2 <sup>3</sup> (2048-x)) Hz		R/W f = 64Hz - 131KHz

## Appendix 2. Register and Instruction Set Summaries

	Register	Address	D7	D6	D5	D4	D3	D2	D1	D0	Comment
S O U N D  4	NR41  Sound length data	FF20			Sound length data t1 : 0 - 63 Sound length = (64+t1) * (1/256) sec						R/W
	NR42  Envelope	FF21	Initial envelope value 0x00 – 0x0F Mute when 0x00 Max when 0x0F			Envelope U/D 0: Attenuate 1: Amplify	Number of envelope steps N = 0-7 Length of 1 step = N*(1/64) sec Envelope function stops when N=0				R/W  Initial value of 00 sets to OFF when in DOWN mode
	NR43  Polynomial counter	FF22	Polynomial counter clock frequency selection Prohibited codes 0000: $f_b * 1/2$ 1100: $f_b * 1/2^{13}$ 1110 0001: $f_b * 1/2^2$ 1101: $f_b * 1/2^{14}$ 1111			Step no. selection 0: 15 steps 1: 7 steps	Selection of frequency dividing ratio $f_b$ 000: $f^*1/2^{3*2}$ 110: $f^*1/2^{3*1}/6$ 001: $f^*1/2^{3*1}/1$ to 000: $f^*1/2^{3*1}/7$				W  f = 4.194304MHz
	NR44  Initialize/length	FF23	Restart when initialize 0: flag set to 1 1: NR41	Length selection 0: Consecutive 1: NR41							R/W
C O N T R O L	NR50  SO1 / SO2 level	FF24	VIN input 0: SO2 OFF 1: SO2 output	SO2 output level control 000 (min) – 111 (max)			VIN input 0: SO1 OFF 1: SO1 output	SO1 output level control 000 (min) – 111 (max)			R/W
	NR51  Distribution to SO1/SO2	FF25	Sound 4 to SO2	Sound 3 to SO2	Sound 2 to SO2	Sound 1 to SO2	Sound 4 to SO1	Sound 3 to SO1	Sound 2 to SO1	Sound 1 to SO1	R/W  0: No output 1: Output
	NR52  Sound-end flag	FF26	All sounds 0: Stop 1: Play				Sound 4 ON flag	Sound 3 ON flag	Sound 2 ON flag	Sound 1 ON flag	R/W

Game Boy Programming Manual

Waveform RAM

Waveform RAM is made up of waveform patterns consisting of 4 bits x 32 steps.

Address	D7	D6	D5	D4	D3	D2	D1	D0
FF30	Step 0				Step 1			
FF31	Step 2				Step 3			
FF32	Step 4				Step 5			
FF3F	Step 30				Step 31			

## 3. CPU INSTRUCTION SET SUMMARY

	MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE																	
			CY	H	N	Z	CYCL	76 543 210	COMMENT																
8-Bit Transfer/Input-Output Instructions	LD r,r'	r←r'	--	--	-	--	1	01 r r'	<table><tr><td>Register</td><td>r,r'</td></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr></table>	Register	r,r'	A	111	B	000	C	001	D	010	E	011	H	100	L	101
	Register	r,r'																							
	A	111																							
	B	000																							
	C	001																							
	D	010																							
	E	011																							
	H	100																							
	L	101																							
	LD r,n	r←n	--	--	-	--	2	00 r 110																	
								← n →																	
	LD r,(HL)	r← (HL)	--	--	-	--	2	01 r 110																	
	LD (HL),r	(HL) ←r	--	--	-	--	2	01 110 r																	
	LD (HL),n	(HL) ←n	--	--	-	--	3	00 110 110																	
								← n →																	
	LD A,(BC)	A← (BC)	--	--	-	--	2	00 001 010																	
	LD A,(DE)	A← (DE)	--	--	-	--	2	00 011 010																	
	LD A,(C)	A←(FF00H+C)	--	--	-	--	2	11 110 010																	
	LD (C),A	(FF00H+C) ←A	--	--	-	--	2	11 100 010																	
	LD A,(n)	A←(n)	--	--	-	--	3	11 110 000																	
							← n →																		
LD (n),A	(n) ←A	--	--	-	--	3	11 100 000																		
							← n →																		
LD A,(nn)	A←(nn)	--	--	-	--	4	11 111 010																		
							← n →																		
							← n →																		
LD (nn),A	(nn) ←A	--	--	-	--	4	11 101 010																		
							← n →																		
							← n →																		
LD A,(HLI)	A←(HL) HL←HL+1	--	--	-	--	2	00 101 010																		
LD A,(HLD)	A←(HL) HL←HL-1	--	--	-	--	2	00 111 010																		
LD (BC),A	(BC) ←A	--	--	-	--	2	00 000 010																		



	MNEMONIC	SYMBOLIC OPERATION	FLAGS				CYCL	OP-CODE 76 543 210
			CY	H	N	Z		
16-Bit Transfer Instructions	LD (DE),A	(DE) ← A	--	--	--	--	2	00 010 010
	LD (HLI),A	(HL) ← A HL ← HL + 1	--	--	--	--	2	00 100 010
	LD (HLD),A	(HL) ← A HL ← HL - 1	--	--	--	--	2	00 110 010
	LD dd,nn	dd ← nn	--	--	--	--	3	00 dd0 001
			L-ADRS					← n →
			H-ADRS					← n →
	LD SP,HL	SP ← HL	--	--	--	--	2	11 111 001
	PUSH qq	(SP-1) ← qqH (SP-2) ← qqL SP ← SP-2	--	--	--	--	4	11 qq0 101
	POP qq	qqL ← (SP) qqH ← (SP+1) SP ← SP+2	--	--	--	--	3	11 qq0 001
	LDHL SP,e	HL ← SP+e	*	*	0	0	3	11 111 000
LD (nn),SP	(nn) ← SPL (nn+1) ← SPH	--	--	-	--	5	00 001 000	
		L-ADRS					← n →	
		H-ADRS					← n →	

Register Pair	dd
BC	00
Register Pair	dd
DE	01
HL	10
SP	11

Register Pair	qq
BC	00
DE	01
HL	10
AF	11

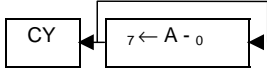
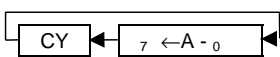
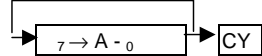
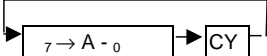
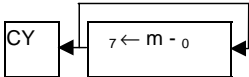
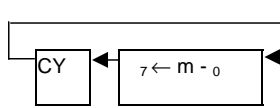
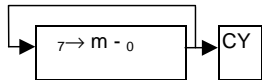
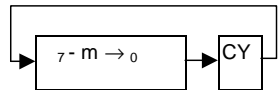
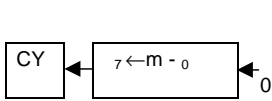
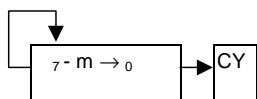
e=-128~+127

Appendix 2. Register and Instruction Set Summaries

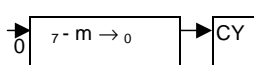
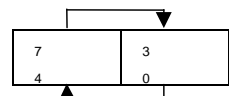
	MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE			COMMENT		
			CY	H	N	Z	CYCL	76	543	210			
8-Bit Arithmetic and Logical Operation Instructions	ADD A,r	$A \leftarrow A+r$	*	*	0	*	1	10	000	r	s is any of r,n,(HL)  CYCL 1: s is r 2: s is n or (HL)		
	ADD A,n	$A \leftarrow A+n$	*	*	0	*	2	11	000	110			
									$\leftarrow n \rightarrow$				
	ADD A,(HL)	$A \leftarrow A+(HL)$	*	*	0	*	2	10	000	110			
	ADC A,s	$A \leftarrow A+s+CY$	*	*	0	*	1,2	--	---	---			
	SUB s	$A \leftarrow A-s$	*	*	1	*	1,2	--	---	---			
	SBC A,s	$A \leftarrow A-s-CY$	*	*	1	*	1,2	--	---	---			
	AND s	$A \leftarrow A \wedge s$	0	1	0	*	1,2	--	---	---			
	OR s	$A \vee s$	0	0	0	*	1,2	--	---	---			
	XOR s	$A \oplus s$	0	0	0	8	1,2	--	---	---			
	CP s	$A-s$	*	*	1	*	1,2	--	---	---			
	INC r	$r \leftarrow r+1$	--	*	0	*	1	00	r	100			
	INC (HL)	$(HL) \leftarrow (HL)+1$	--	*	0	*	3	00	110	100			
	DEC r	$r \leftarrow r-1$	--	*	1	*	1	00	r	101			
DEC (HL)	$(HL) \leftarrow (HL)-1$	--	*	1	*	3	00	110	101				
16-Bit Arithmetic Operation Instructions	ADD HL,ss	$HL \leftarrow HL+ss$	*	*	0	--	2	00	ss1	001	Register Pair	ss	
	ADD SP,e	$SP \leftarrow SP+e$	*	*	0	0	4	11	101	000	BC	00	
									$\leftarrow e \rightarrow$			DE	01
	INC ss	$ss \leftarrow ss+1$	--	--	--	--	2	00	ss0	011	HL	10	
	DEC ss	$ss \leftarrow ss-1$	--	--	--	--	2	00	ss1	011	SP	11	
												e=-128~+127	

The flag is affected according to the result of the operation.

Z: Zero flag. z=1 if the result of the operation is 0  
C: Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result  
H: Half-carry flag.  
N: Add/Subject flag.

	MNEMONIC	SYMBOLIC OPERATION	FLAGS				OP-CODE	COMMENT																	
			CY	H	N	Z	76 543 210																		
Rotate Shift Instructions	RLCA		A <sub>7</sub>	0	0	0	1	00 000 111	<table><tr><td>Register</td><td>r</td></tr><tr><td>A</td><td>111</td></tr><tr><td>B</td><td>000</td></tr><tr><td>C</td><td>001</td></tr><tr><td>D</td><td>010</td></tr><tr><td>E</td><td>011</td></tr><tr><td>H</td><td>100</td></tr><tr><td>L</td><td>101</td></tr></table>	Register	r	A	111	B	000	C	001	D	010	E	011	H	100	L	101
	Register	r																							
	A	111																							
	B	000																							
	C	001																							
	D	010																							
	E	011																							
	H	100																							
	L	101																							
RLA		A <sub>7</sub>	0	0	0	1	00 010 111																		
RRCA		A <sub>0</sub>	0	0	0	1	00 001 111																		
RRA		A <sub>0</sub>	0	0	0	1	00 011 111																		
RLC m		m <sub>7</sub>	0	0	*	--	-- --- ---	<table><tr><td>m is any of r, (HL)</td><td>CYCL</td></tr><tr><td>RLC r</td><td>2</td></tr><tr><td>RLC (HL)</td><td>4</td></tr></table>	m is any of r, (HL)	CYCL	RLC r	2	RLC (HL)	4											
m is any of r, (HL)	CYCL																								
RLC r	2																								
RLC (HL)	4																								
RL m		m <sub>7</sub>	0	0	*	--	-- --- ---	<table><tr><td>RL r</td><td>2</td></tr><tr><td>RL (HL)</td><td>4</td></tr></table>	RL r	2	RL (HL)	4													
RL r	2																								
RL (HL)	4																								
RRC m		m <sub>0</sub>	0	0	*	--	-- --- ---	<table><tr><td>RRC r</td><td>2</td></tr><tr><td>RRC (HL)</td><td>4</td></tr></table>	RRC r	2	RRC (HL)	4													
RRC r	2																								
RRC (HL)	4																								
RR m		m <sub>0</sub>	0	0	*	--	-- --- ---	<table><tr><td>RR r</td><td>2</td></tr><tr><td>RR (HL)</td><td>4</td></tr></table>	RR r	2	RR (HL)	4													
RR r	2																								
RR (HL)	4																								
SLA m		m <sub>7</sub>	0	0	*	--	-- --- ---	<table><tr><td>SLA r</td><td>2</td></tr><tr><td>SLA (HL)</td><td>4</td></tr></table>	SLA r	2	SLA (HL)	4													
SLA r	2																								
SLA (HL)	4																								
SRA m		m <sub>0</sub>	0	0	*	--	-- --- ---	<table><tr><td>SRA r</td><td>2</td></tr><tr><td>SRA (HL)</td><td>4</td></tr></table>	SRA r	2	SRA (HL)	4													
SRA r	2																								
SRA (HL)	4																								

**Appendix 2. Register and Instruction Set Summaries**

MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE			COMMENT				
		CY	H	N	Z	CYCL	76	543	210					
SRL m		m <sub>0</sub>	0	0	*	--	--	---	---	<table><tr><td>SRL</td><td>2</td></tr><tr><td>r</td><td></td></tr></table>	SRL	2	r	
SRL	2													
r														
										<table><tr><td>SRL</td><td>4</td></tr><tr><td>(HL)</td><td></td></tr></table>	SRL	4	(HL)	
SRL	4													
(HL)														
SWAP m		0	0	0	*	--	--	---	---	<table><tr><td>SWAP</td><td>2</td></tr><tr><td>r</td><td></td></tr></table>	SWAP	2	r	
SWAP	2													
r														
										<table><tr><td>SWAP</td><td>4</td></tr><tr><td>(HL)</td><td></td></tr></table>	SWAP	4	(HL)	
SWAP	4													
(HL)														

Game Boy Programming Manual

	MNEMONIC	SYMBOLIC OPERATION	FLAGS				CYCL	OP-CODE			COMMENT																																				
			CY	H	N	Z		76	543	210																																					
Bit Operations	BIT b,r	$Z \leftarrow r_b$	--	1	0	$r_b$	2	11	001	011	<table><tr><td>Bit</td><td>b</td><td>Register</td><td>r</td></tr><tr><td>0</td><td>000</td><td>A</td><td>111</td></tr><tr><td>1</td><td>001</td><td>B</td><td>000</td></tr><tr><td>2</td><td>010</td><td>C</td><td>001</td></tr><tr><td>3</td><td>011</td><td>D</td><td>010</td></tr><tr><td>4</td><td>100</td><td>E</td><td>011</td></tr><tr><td>5</td><td>101</td><td>H</td><td>100</td></tr><tr><td>6</td><td>110</td><td>L</td><td>101</td></tr><tr><td>7</td><td>111</td><td></td><td></td></tr></table>	Bit	b	Register	r	0	000	A	111	1	001	B	000	2	010	C	001	3	011	D	010	4	100	E	011	5	101	H	100	6	110	L	101	7	111		
	Bit	b	Register	r																																											
	0	000	A	111																																											
	1	001	B	000																																											
	2	010	C	001																																											
	3	011	D	010																																											
4	100	E	011																																												
5	101	H	100																																												
6	110	L	101																																												
7	111																																														
BIT b,(HL)	$Z \leftarrow (HL)_b$	--	1	0	$(HL)_b$	3	11	001	011																																						
SET b,r	$r_b \leftarrow 1$	--	-	--	--	2	11	001	011																																						
SET b,(HL)	$(HL)_b \leftarrow 1$	--	-	--	--	4	11	001	011																																						
RES b,r	$r_b \leftarrow 0$	--	-	--	--	2	11	001	011																																						
RES b,(HL)	$(HL)_b \leftarrow 0$	--	-	--	--	4	11	001	011																																						
Jump Instructions	JP nn	$PC \leftarrow nn$	--	-	--	--	4	11	000	011	<p>* No. of cycles is 3 when no cc agreement</p> <table><tr><td>CC</td><td>Condition</td><td>Flag</td></tr><tr><td>00</td><td>NZ</td><td>Z=0</td></tr><tr><td>01</td><td>Z</td><td>Z=1</td></tr><tr><td>10</td><td>NC</td><td>CY=0</td></tr><tr><td>11</td><td>C</td><td>CY=1</td></tr></table> <p>e=-127~+129</p>	CC	Condition	Flag	00	NZ	Z=0	01	Z	Z=1	10	NC	CY=0	11	C	CY=1																					
			CC	Condition	Flag																																										
			00	NZ	Z=0																																										
			01	Z	Z=1																																										
	10	NC	CY=0																																												
	11	C	CY=1																																												
	L-ADRS						$\leftarrow n \rightarrow$																																								
	H-ADRS						$\leftarrow n \rightarrow$																																								
	JP cc,nn	If cc true, $PC \leftarrow nn$	--	-	--	--	4/3	11	0cc	010																																					
			L-ADRS						$\leftarrow n \rightarrow$																																						
H-ADRS						$\leftarrow n \rightarrow$																																									
JR e	$PC \leftarrow PC+e$	--	-	--	--	3	00	011	000																																						
JR cc,e	If cc true, $PC \leftarrow PC+e$	--	-	--	--	3/2	00	1cc	000																																						
								$\leftarrow e-2 \rightarrow$																																							
JP (HL)	$PC \leftarrow HL$	--	-	--	--	1	11	101	001																																						

Appendix 2. Register and Instruction Set Summaries

	MNEMONIC	SYMBOLIC OPERATION	FLAGS				CYCL	OP-CODE			COMMENT	
			CY	H	N	Z		76	543	210		
Call/Return Instructions	CALL nn	(SP-1) ← PC <sub>H</sub> (SP-2) ← PC <sub>L</sub> PC←nn SP←SP-2	--	-	--	--	6	11	001	101		
			L-ADRS						← n →			
			H-ADRS						← n →			
	CALL cc,nn	If cc true, (SP-1) ← PC <sub>H</sub> (SP-2) ← PC <sub>L</sub> PC←nn SP←SP-2	--	-	--	--	6/3	11	0cc	100		
			L-ADRS						← n →			
			H-ADRS						← n →			
	RET	PC <sub>L</sub> ←(SP) PC <sub>H</sub> ←(SP+1) SP←SP+2	-	--	--	--	4	11	001	001		
	RETI	PC <sub>L</sub> ←(SP) PC <sub>H</sub> ←(SP+1) SP←SP+2	-	--	--	--	4	11	011	001		
	RET cc	If cc true, PC <sub>L</sub> ←(SP) PC <sub>H</sub> ←(SP+1) SP←SP+2	-	--	--	--	5/2	11	0cc	000		
	RST t	(SP-1) ← PC <sub>H</sub> (SP-2) ← PC <sub>L</sub> SP←SP-2 PC <sub>H</sub> ←0 PC <sub>L</sub> ←P	-	--	--	--	4	11	t	111		
	Gen-Purpose Arithmetic/CPU Control Instructions	DAA	Decimal Adjust acc	*	0	--	*	1	00	100		111
CPL		A ← $\overline{A}$	-	1	1	--	1	00	101	111		
NOP		No operation	-	--	--	--	1	00	000	000		
CCF		CY← $\overline{CY}$	$\overline{CY}$	0	0	--	1	00	111	111		
SCF		CY←1	1	0	0	--	1	00	110	111		
DI		IME←0	-	--	--	--	1	11	110	011		
EI		IME←1	-	--	--	--	1	11	111	011		
HALT		Halt		--	--	--	1	01	110	110		
STOP		Stop	-	--	--	--	1	00	010	000		
								00	000	000		

Operand	t	(PC <sub>H</sub> )	(PC <sub>L</sub> )
0	000	00h	00h
1	001	00h	08h
2	010	00h	10h
3	011	00h	18h
4	100	00h	20h
5	101	00h	28h
6	110	00h	30h
7	111	00h	38h