Chapter 2

Game Boy Architecture In A Nutshell

A solid understanding of the hardware specifications and capabilities of the Game Boy Advance is necessary in order to write the most efficient code for this platform. Programming any console is a rewarding experience because there are no layers between you and the hardware itself. When you modify a bit here and a bit there, things happen! There is no operating system, no game library, and in the case of the Game Boy Advance, not even the luxury of a graphics processing unit (GPU). Despite what might seem like limited hardware specifications, one must remember that when working directly with the hardware, without any layers, things move along very quickly. That is truly what makes Game Boy Advance programming so rewarding. A programmer's most natural habitat is as close to the hardware as possible, and consoles uniquely provide that environment.

This chapter presents an overview of the Game Boy Advance's hardware specifications, explaining how it works, or what makes it tick, so to speak. The display screen, memory architecture, and main processors are covered in detail. This chapter also examines previous Game Boy models, comparing and contrasting them with the Game Boy Advance. As a hardware reference and guide, feel free to refer back to this chapter at any time when you need some specifics on the hardware. The pace is somewhat fast, and I don't explain every single detail at this time, because many of these concepts are covered in later chapters.

Here is a summary of the subjects covered in this chapter:

- Game Boy handheld systems
- Direct hardware access
- Memory architecture
- The central processor
- Graphics and sound capabilities

Game Boy Handheld Systems

The Game Boy Advance has a long and fruitful history that goes clear back to 1989 when the original Game Boy came out. The Game Boy Advance is sort of the great-grandchild of that first Game Boy, because it is the fourth Game Boy model. New as it may be, however, the Game Boy Advance has now been supplanted by the Game Boy Advance SP. Granted, the internal hardware is architecturally the same, but this new SP model has some considerable new options, not the least of which is a backlit screen. Let's peruse all of the Game Boys that have made their way into our hearts over the years. Table 2.1 shows an overview of the Game Boy models and their specifications.

Table 2.1 Game Boy Specifications

Model	CPU	Memory	Display	Colors
Game Boy	8-bit Z80 4.17 MHz	64 Kbits	160 x 144	4
Game Boy Pocket	8-bit Z80 4.17 MHz	64 Kbits	160 x 144	4
Game Boy Color	8-bit Z80 8.0 MHz	384 Kbits	160 x 144	56
Game Boy Advance	32-bit ARM7 16.7 MHz	3,072 Kbits	240 x 160	32,768
Game Boy Advance SP	32-bit ARM7 16.7 MHz	3,072 Kbits	240 x 160	32,768

Game Boy, 1989

The original Nintendo Game Boy (shown in Figure 2.1) was released in 1989, only four years after the NES came out in the United States. Operating on four AA batteries, the Game Boy was not a revolutionary console by any means (an attribute shared with the NES), but it had a relatively long battery life. This first Game Boy was equipped with a Zilog Z80 microprocessor—the same one used on many electronic devices in the 1980s. In fact, all of the Game Boy models up to and including Game Boy Color featured a Z80 CPU, although later models were faster. The Game Boy's CPU runs at 4.17 MHz, which is comparable to the first IBM PC at 4.77 MHz. Not bad for a tiny little handheld!

The first Game Boy came with 64 Kbits of memory, which is a very limited amount! However, due to the small display screen, with a resolution of 160 x 144 and only four-color grayscale, very little memory was required for graphics. Four colors is really insignificant, memory-wise; that's what you might call 2-bit color. Since two binary digits can store the numbers 0, 1, 2, or 3, there are your four colors! Obviously, color 0 was black. That didn't

leave much for artists, so to state that the Game Boy had primitive graphics is exactly right on target.

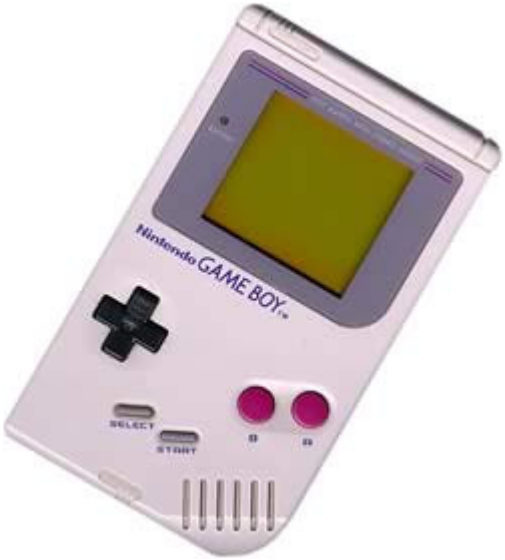

Figure 2.1
Nintendo Game Boy, 1989.

To put this in perspective, the Atari 2600 had better graphics than the Game Boy, and it was a decade older. However, a very low memory footprint for 2-bit graphics does allow for some interesting games, even if the color support is limited, and this kept manufacturing costs down for Nintendo. A typical 8 x 8 sprite would use up only 128 bits of memory (that's just 16 bytes), and the 160 x 144 video buffer would have required only 5,760 bytes of memory (although there was no "video buffer", per se, on the Game Boy). As a side note, the Game Boy was capable of handling only 8 x 8 and 8 x 16 pixel sprites, and only a maximum of 40 at a time.

Game Boy Pocket, 1996

The Game Boy Pocket (shown in Figure 2.2) was a slimmed-down version of the Game Boy. Although the hardware specifications were essentially the same, the Game Boy Pocket required less voltage to operate (3 volts instead of 6) and thus could be powered by only two AAA batteries, which are much smaller than AA and suited the small size of the Game Boy Pocket.

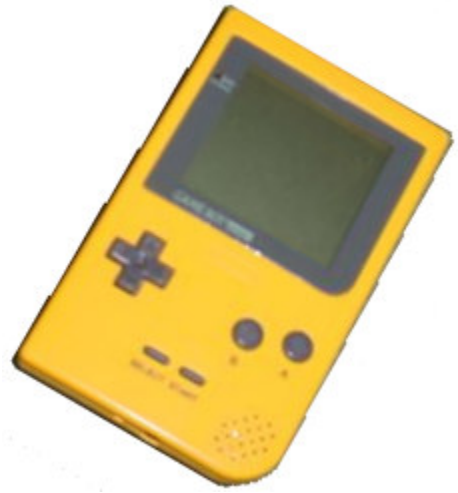

Figure 2.2
Nintendo Game Boy Pocket, 1994.

Game Boy Color, 1998

The Game Boy Color (shown in Figure 2.3) was significantly more capable than the previous two models and greatly aided game developers with a faster CPU and more memory, while still retaining compatibility with the older game cartridges. The Game Boy Color only requires two AA batteries and was therefore much lighter than the original Game Boy.

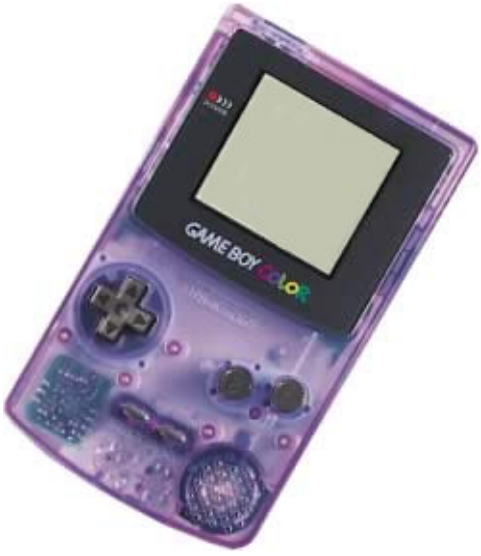

Figure 2.3
Nintendo Game Boy Color, 1998.

The Game Boy Color not only was capable of displaying 56 colors on the screen at once but also enhanced existing Game Boy games to 32 colors, greatly improving their appearance and playability. Grayscale games came to life on the Game Boy Color with multiple shades of color. Obviously, backward compatibility was an important factor for Nintendo, primarily for marketing reasons. Claiming that a just-released console (such as the Game Boy Color)

has a game library of several hundred titles is an impressive feat! Of course, a large percentage of those games are very low-quality Game Boy titles, due to the limited capabilities of the GB. Many excellent titles were released for Game Boy Color, including the phenomenally successful *Super Mario Bros. Deluxe* and *The Legend of Zelda: Link's Awakening DX*.

Game Boy Advance, 2001

The Game Boy Advance (shown in Figure 2.4) is significantly more powerful than any previous Game Boy model, with nearly twice the screen resolution, 10 times more memory than the Game Boy Color, and a blazing-fast RISC CPU (more than twice as fast as Game Boy Color). In addition, the Game Boy Advance incorporates the original Z80 CPU from the Game Boy Color, providing for complete backward compatibility with all previous Game Boy cartridges. I would surmise that Nintendo was primarily concerned with supporting Game Boy Color titles rather than the older grayscale Game Boy games, although all previous cartridges *will* work!

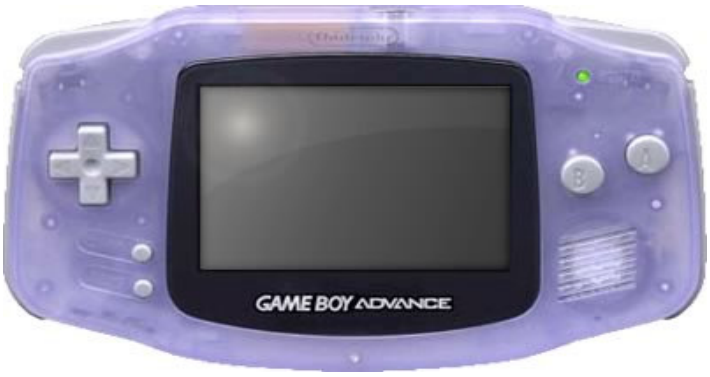

Figure 2.4
Nintendo Game Boy Advance, 2001.

Basically, what happens is that the Game Boy Advance detects the type of cartridge that has been inserted and boots up on either the ARM7 or the Z80 CPU, based on the cartridge. This ingenious architectural design allows the Game Boy Advance to run all previous games, all the way back to 1989—a significant achievement of electronics engineering. If you think about it, how many other consoles today are capable of running games from 1989—original games, in their original cartridges? None! Only the Game Boy Advance is capable of this feat (and the Game Boy Color before it).

Before getting on with the next model, the Game Boy Advance SP, I want to show you an awesome accessory for your GBA. If you already own a GBA and are considering purchasing a Game Boy Advance SP model only for the backlight, you have another option. Figure 2.5

shows a Game Boy Advance in normal indoor lighting with an Afterburner installed. This is an inexpensive internal flat-surface LED that is positioned in front of the LCD, providing a remarkable improvement in screen visibility regardless of the lighting conditions.

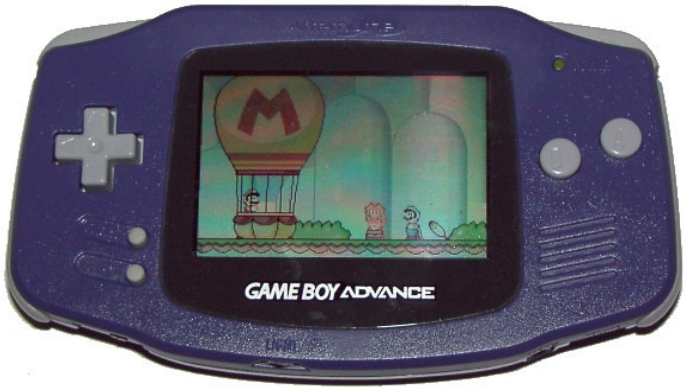

Figure 2.5
This Game Boy Advance is equipped with an Afterburner to brighten the screen.

Some soldering is required, but the work is relatively easy to do (that is, if you have any experience with a soldering iron—if not, you should ask a friend who is experienced with one to help you). For development work, the Afterburner is an essential add-on. It is very inexpensive (under $30) and may be ordered from online Game Boy Advance retailers, such as Lik-Sang (http://www.lik-sang.com). I suggest this alternative because I prefer the original Game Boy Advance design.

Game Boy Advance SP, 2003

The Game Boy Advance SP (shown in Figure 2.6) is a variation of the Game Boy Advance with an internally lighted screen, long-lasting rechargeable battery (built in), and folding clamshell design. In all other respects, the SP has the same internal components as the Game Boy Advance, and the changes are cosmetic. While the rechargeable battery is internal (and therefore not replaceable without disassembly), it does promise longer life than the AA batteries used in a Game Boy Advance (and there are rechargeable battery packs for Game Boy Advance as well). One interesting aspect of the SP is that, when the screen is opened, it resembles the original Game Boy!

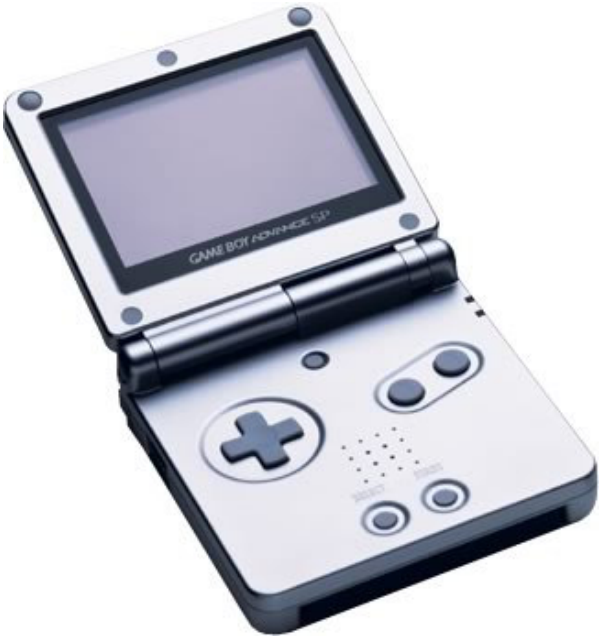

Figure 2.6
Nintendo Game Boy Advance SP, 2003.

Direct Hardware Access

The Game Boy Advance is a video game console, and yet it is similar to a PC in many ways. Both have one or more processors, random-access memory (RAM), a display screen (or monitor) with many different video modes, a digital signal processor (DSP) for sound, and some form of intuitive user input. Both the Game Boy Advance and a PC have a motherboard with a power supply and a basic input/output system (BIOS) chip that causes the hardware to boot up.

A PC provides access to the hardware primarily through the operating system, while a console primarily operates by storing all system functions inside the executable program (the game). The Game Boy Advance has no operating system. Game Boy Advance games have complete control over the hardware, at the lowest level. This gives the programmer a great deal of control over the console, but also great responsibility. Many software engineers specialize in applications, operating systems, network communications, or device drivers. As a Game Boy Advance programmer, you will touch on all of these areas and more, each time you write a game, because no one has paved the way, so to speak. Each new program you write for the Game Boy Advance must incorporate all the code necessary to display things on the screen, play sound effects and music, and detect button presses. Most of these features are programmed using direct memory address functionality.

On the PC, there are interrupts provided by the operating system that you can use to make things happen. On a GBA, however, you make things happen by reading or writing a number

to a specific portion of memory (called a *hardware register*). The Game Boy Advance changes instantly when you make such a change, because those memory addresses are directly tied to the hardware. If you turn on a bit somewhere in video memory, the screen will change to another video mode or perhaps even display a pixel.

As is usually the case when charting new territory, it is useful to draw a map along the way. Fortunately, someone has already provided all the memory addresses for us, so we don't need to fire numbers into random memory locations to see what happens—in fact, that would likely have an adverse effect on the Game Boy; who knows, it could even be damaged by it. Let's look at the general features of the Game Boy Advance to get an idea of the terrain ahead.

Memory Architecture

You may be aware that your PC has three distinct types of memory. You have your main RAM, which holds all the programs and data you're actively working with (ignore virtual memory for now since this looks to your programs like lots of main RAM). There is the hard disk, which stores information for long periods. And there is display memory on your video card.

The Game Boy Advance has similar kinds of memory. Like the PC, each address refers to a single 8-bit byte (which means that it is byte addressable). Also like the PC, the ARM ("Advanced RISC Machine") processor in the Game Boy Advance can access 8, 16, or 32 bits at a time. Things are a little more complicated, though, and you need to understand a little more about how the different parts are used. Table 2.2 lists the types of memory accesses the memory allows and the wait states each access incurs.

The CPU reads memory by sending an address to the memory at the start of a cycle and then reading the data at the end of that cycle. Ideally, the memory is fast enough to provide the data that quickly. Slower memory tells the CPU to wait for one or more additional cycles before reading the data. These extra cycles are called wait states. The fastest memories, such as IWRAM and VRAM, have no wait states (abbreviated 0WS) and therefore return data in a single cycle. EWRAM is 2WS memory, returning data in three cycles—one normal cycle plus two wait states.

Table 2.2 Game Boy Memory Access		
Memory Type	Access Widths	Comments
IWRAM	8, 16, 32	32 KB "internal" working RAM. Typically used for fast scratchpad RAM and for time-critical code.
EWRAM	8, 16	256 KB "external" working RAM. Typically used for main data storage and multiboot code.
VRAM	8, 16	96 KB video RAM. Stores all graphics data. Can only write 16 bits at a time.
ROM	8, 16	ROMs can be read in either slow (4/2) or fast (3/1) mode. See chapter text for more details.
Game Save RAM	8, 16	The game save RAM is part of the cartridge. See chapter text for more details.

Internal Working RAM

Internal working RAM (IWRAM) is the only memory directly accessible on the 32-bit internal data bus of the CPU core, because it is actually built into the CPU itself. This is why it's called *internal WRAM* as opposed to external WRAM (covered next). IWRAM is the fastest memory in the Game Boy Advance and is also the only memory that can be accessed 32 bits at a time. The speed and width of this memory makes it ideal for running ARM code at full speed. Unfortunately, there are only 256 Kbits of this memory.

As far as the ARM processor is concerned, a word is 32 bits, while a halfword is 16 bits, and a byte is 8 bits.

External Working RAM

One might think something named external working RAM (EWRAM) would on a cartridge or something. However, it is built into the GBA. It's called *external* because it sits outside the CPU's core on the 16-bit data bus. We've got 2,048 Kbits of this to play with. This RAM, with each access taking three cycles, is slower than IWRAM.

EWRAM is where you will store large data items. You may cache graphics here before transferring them to VRAM for display. You can also place programs here using the multiboot

protocol. Programs will run a little faster here than in fast ROM and quite a bit faster than in slow ROM.

The amount of memory that can be addressed by a particular number of address bits is always a power of two. You can address 2^{16} (or 65,536) locations by using 16 address bits. The closest power of two to 1,000 is 2^{10} (or 1,024). This has become the usual meaning for 1 KB in measuring memory. Therefore, 32 KB = 32 x 1,024 = 32,768 bytes. Similarly, 1 MB = 1,024 K = 1,048,576.

Cartridge Memory

The game cartridge contains the game's program and data stored in ROM. This is much like a CD-ROM for your PC in that it cannot be changed and is typically larger than the RAM you have available. Some cartridges also contain memory for saving games. Special cartridges, such as those made by Visoly, have memory called flash cartridges. These devices look to the Game Boy Advance like normal game cartridges and yet can be programmed with your own data much like a hard disk in your PC, using a flash linker device (which is covered in the next chapter).

Game ROM

Like EWRAM, the ROM is accessible via the 16-bit data bus. There are two speeds at which ROMs can operate and two modes in which they can be accessed. Speeds for the ROMs are given as a pair of wait-state values, such as 3/1. The overriding factor is whether each access to the ROM can be classified as sequential or nonsequential.

A nonsequential access occurs whenever a new area of the ROM is read. Sending the memory address to the ROM takes extra time, and these accesses take the number of wait states indicated by the first number. In this example, the nonsequential access will take four cycles (three wait states plus the normal cycle).

A sequential access occurs when the very next access to the ROM is at the next address. In this case the ROM already has the next address available and only takes the smaller number of wait states. This use of sequential accesses means that a consecutive sequence of instructions that have no other data accesses can run with only 1 wait state for faster ROMs. Even slower ROMs (running with 4/2 wait states) can equal EWRAM speed for these short bursts, while fast ROMs can outpace EWRAM during such a run. What this means, basically, is that a game cartridge is capable of slow-mode and fast-mode access.

On average, however, even fast ROMs will fall behind EWRAM because long runs of sequential accesses are not the norm.There is a prefetch buffer in the memory controller

that allows some sequential accesses have no wait states. Unfortunately, this speed comes at a cost in power usage.

Bytes, Words, and Halfwords

The ARM processor uses some terminology for memory sizes differently than in the PC world. To understand why, let's look at the history of some of the terms.

Most of us are used to a *byte* being 8 bits, and indeed the ARM doesn't change this. Looking back in history, however, we find that a byte actually refers to the size of a native character for the computer's output device.

Just as the byte has had different sizes, a computer *word* is commonly defined as the normal amount of memory the computer processes at one time. In the early 1980s, IBM designed the IBM PC around the Intel 8086 and Apple built the Macintosh with the Motorola 68000. Both of these machines processed 16 bits at a time and used 8-bit ASCII character sets. These two machines have cemented the terms *byte*, *word*, and *double-word* as meaning 8, 16, and 32 bits, respectively, for most of personal computerdom.

For more than a decade, Intel and Motorola microprocessors have processed 32 bits at a time in their normal operations, and new 64-bit processors are now available for PCs (such as the AMD *Opteron* and *Athlon 64*). The word size of 32-bit computers is therefore 32 bits, while a word on a 64-bit processor is 64 bits. The terminology surrounding software for them, however, has maintained the older terms because the operating system APIs and data structures have evolved using the terms rooted in their 16-bit ancestors. But the important factor to remember is that a word usually comprises the same number of bits that are handled by the processor natively, and the byte has remained to this day a fixed 8-bit value.

The ARM has no 16-bit predecessor and need not retain backward compatibility with any overriding architecture (as is the case with the x86), although the Game Boy Advance does include the Z80 for backward compatibility. Imagine a modern PC with an old 80486 chip included along with a newer processor! ARM terminology follows the normal definitions for byte and word: an ARM byte is 8 bits, while an ARM word is 32 bits, while a 16-bit number is called a *halfword*. Throughout this book, I avoid the confusion by simply calling memory addresses and variables by their bit depth.

Game Save Memory

Some cartridges (including the Visoly flash cartridges) have nonvolatile memory for storing saved games on the cartridge. There are currently three different kinds of memory used for this: battery-backed static RAM (SRAM), Flash ROM, and serial EEPROM. Each type has unique methods of access. Some of the Visoly cartridges support all three types of game save memory, while some support only SRAM and Flash ROM (because of the special writer needed for an EEPROM).

Graphics Memory

There are three sections of memory that deal exclusively with video memory and the display screen: video memory, palette memory, and object attribute memory (OAM, for handling sprites).

Video Memory

Video memory is where all graphics data must be stored for display on the screen. VRAM is zero wait-state memory like IWRAM but sits on the 16-bit data bus, so you can only move data half as fast as in IWRAM. How VRAM is used depends greatly upon the video mode and other features that your program selects. One property of this RAM that I will point out many times is that it can only be written 16 bits at a time (while the bus is capable of a full 32 bits). Trying to write a byte will actually write 2 bytes of the same value. This seeming flaw can actually be useful in certain circumstances, such as the ability to quickly redraw the screen.

Care must be taken when writing to VRAM during the time when the screen is being drawn. Attempting to change memory that is being used to draw the screen can result in graphics glitches and image tearing (and event where the image is being drawn while the screen is also being refreshed, resulting in an uneven image). Furthermore, VRAM accesses during screen time can be delayed while the CPU waits for the video hardware to perform its accesses.

Palette Memory

Most of the Game Boy Advance's video modes use palettes to specify the colors being used. The Game Boy Advance has two separate 256-color palettes: one for background images and one for sprites. Each of these palettes is further divided into 16 palettes of 16 colors in some modes, allowing graphics data to be compacted even more. Color 0 of any palette is

defined to be transparent no matter what value is actually stored in the palette memory. Palettes are usually updated during the vertical blanking (VBlank) period, during which time the screen is not being drawn. There is also one video mode that doesn't use a palette but directly addresses colors in each pixel (mode 4).

Object Attribute Memory

Object attribute memory (OAM) is where you store the attributes, or descriptions, of what a sprite is to display, how, and where. The actual graphics data comes from VRAM, but the sprite's position, size, and other information come from the OAM. OAM is commonly updated during VBlank, and one often mirrors this data set in main memory for improved speed.

The Central Processor

The processor in the Game Boy Advance is an ARM7TDMI chip. This is a 32-bit RISC processor with a three-stage pipeline, a hardware multiplier, and lots of registers—it is quite versatile. I can recommend two reference books for more detail about the ARM, if you are looking to get into some serious low-level programming, or if you are just curious.

- ***ARM Architecture Reference Manual***, edited by David Seal, Addison-Wesley, ISBN 0-201-73719-1. Also known as the *ARM ARM*, this is a detailed description of the many ARM processors in use today. This book is invaluable when working in assembly language.
- ***ARM System-on-Chip Architecture* (2000)**, by Steve Furber, Addison-Wesley, ISBN 9-201-67519-6. This book gives more of the how and why about using the ARM. Whereas the *ARM ARM* is the definitive reference book, Furber gives a more readable text and fills in some of the details about why things were done the way they were.

Two Instruction Sets

The ARM is a RISC (reduced instruction set computer) processor design. As with most RISC processors the instruction set is very regular, meaning that there are few ways to encode an instruction. RISC design makes it very easy to build a fast and inexpensive CPU. It does not, however, lead to very compact code. In general, *code density*—the number of instructions per unit of memory—is lower in RISC designs than it is in CISC (complex instruction set

computer) designs. While the ability to understand assembly language is a valuable skill when writing console code, I don't use it very much in this book, aside from a primer in chapter 11, "ARM7 Assembly Language Primer." I have only included this chapter to help you interface with assembly, not to teach you full-blown assembly language programming, because this book focuses on the C language. Knowing how to interface with assembly routines can be helpful, so that is the focus of that chapter.

The designers of the ARM decided that they could create a denser instruction set—which they dubbed the Thumb instruction set—by cutting out some features and making the instruction encoding a little less regular. They did this in a way that allows the two instruction sets to work together. The CPU essentially converts Thumb instructions into their equivalent ARM instructions on the fly. Each Thumb instruction is 16 bits, whereas the ARM instructions are 32 bits. There is a performance benefit to using 16-bit instructions in a computer with 16-bit memory, although I don't get into Thumb mode at all in this book, as it is an advanced topic. True, there may be cases for using Thumb instructions to speed up parts of a game, but there are also cases for using regular ARM instructions as well.

CPU Registers

The ARM has 16 registers accessible at any time. A register is a physical component of the processor, and may be thought of as part of the "thinking" component. One of these registers, called R15, is the program counter, which keeps track of the current instruction being executed. A couple of the other registers have defined uses for some instructions. All the registers hold 32 bits each.

There are also some additional registers that are only used under certain conditions. There are, for example, registers that replace R13 and R14 (usually the stack pointer and link register) when interrupts occur. There are exceptions to most of these, but knowing this list will help you when looking at compiled code or reading through assembly language examples.

The ARM Procedure Call Standard (APCS) defines a convention for compilers to use when calling functions. This is the way the C compiler calls functions (except under certain optimizing conditions). You don't have to follow this convention when you write your own assembly functions, but you can't call those functions from C if you don't. Table 2.3 details the register uses for APCS. When one procedure calls another there is an assumption that

some of the registers will not be changed by the called procedure. The called procedure must save and restore these values if it needs to change one of these registers.

Table 2.3 ARM Procedure Call Standard

Registers	Usage
R0–R3	These registers are used for passing parameters to functions. Any parameters that don't fit here get passed on the stack.
R4–R10	These registers are used primarily for register variables. Registers 9 and 10 are also used for stack manipulation when switching modules, but you'll seldom, if ever, have to worry about this.
R11	This is the frame pointer. This register is typically set and restored during the prologue or epilogue code since it is the pointer through which all local variables are accessed.
R12	This is the interlink pointer. For most Game Boy Advance code this is a scratch register.
R13	This is the stack pointer. This points to the last item pushed on the stack. The stack is a "full descending" stack meaning that the stack grows downward (toward lower addresses) in memory and the pointer always points to the next item to pop from the stack.
R14	This is the link register. This register holds the return address for the subroutine. This register is often pushed on the stack and then popped directly into the program counter for the return.
R15	This is the program counter. You only directly change it to execute a jump.

The Graphics System

Truly the most important aspect of a console is the graphics system and its capabilities. The Game Boy Advance has an intriguing selection of possible video modes from which to choose. There are three tile-based modes that resemble the previous Game Boy graphics systems. In addition, there are three new bitmap-based modes that provide more creative freedom with a game.

It is important to note some specific numbers that don't change and thus may be relied upon from one Game Boy Advance unit to the next. The refresh rate equates to 280,896 clock cycles per frame (the time it takes to display the entire video buffer), and this provides a refresh rate of 59.73 hertz (Hz). Therefore, the maximum frame rate on the

Game Boy Advance is ~60 FPS. Any game or demo that claims to achieve more than 60 FPS is simply subframing the display, which may have practical uses for special effects, but in general this is an upper limit on the frame rate of the screen. Most consoles aspire to such a frame rate, so don't take this number by any means to reflect poorly on the GBA. It is, in fact, a significant refresh rate.

Tile-Based Modes (0–2)

There are three tile-based modes, as mentioned previously. A "tile" is a small 8 x 8 section of the screen, and this is how the first Game Boy handled all backgrounds. Of course, you may still use sprites that move over the tiled background. I will summarize each of these video modes in the following subsections.

Mode 0

In this mode four text background layers can be shown. In this mode backgrounds 0–3 all count as "text" backgrounds and cannot be scaled or rotated. Check out the section on text backgrounds for details on this.

Mode 1

This mode is similar in most respects to mode 0, the main difference being that only three backgrounds are accessible—0, 1, and 2. Backgrounds 0 and 1 are text backgrounds, while background 2 is a rotation/scaling background.

Mode 2

Like modes 0 and 1, mode 2 uses tiled backgrounds. It uses backgrounds 2 and 3, both of which are rotate/scale backgrounds.

Bitmap-Based Modes (3–5)

There are three bitmap-based modes, also mentioned previously. These are the more familiar video modes that resemble those found on a PC, with a given resolution and a video buffer. I will summarize each of these video modes in the following subsections. You don't need to remember all of this information right now. It will become second nature to you in time, as you actually use these memory addresses and so forth in actual code.

Mode 3

This is a standard 16-bit bitmapped (nonpaletted) 240 x 160 mode. The map starts at 0 x 06000000 and is 76,800 bytes long. See the color format table above for the format of these bytes. This allows the full color range to be displayed at once. Unfortunately, the frame buffer in this mode is too large for page flipping (a method of reducing flicker on the screen—covered in Part Two) to be possible. One option to get around this would be to copy a frame buffer from work RAM into VRAM during the retrace.

Mode 4

This is an 8-bit bitmapped (paletted) mode at a resolution of 240 x 160. The bitmap starts at either 0x06000000 or 0x0600A000, depending on bit 4 of the REG_DISPCNT register. Swapping the map by flipping bit 4 and drawing in the one that isn't displayed allows for page-flipping techniques to be used. The palette is at 0x5000000 and contains 256 16-bit color entries.

Mode 5

This is another 16-bit bitmapped mode, but at a smaller resolution of 160 x 128. The display starts at the upper-left corner of the screen but can be shifted using the rotation and scaling registers for background 2. The advantage of using this mode is presumably that there are two frame buffers available, and this can be used to perform page-flipping effects that cannot be done in mode 3 due to the smaller memory requirements of mode 5. Bit 4 of the REG_DISPCNT register sets the start of the frame buffer to 0 x 06000000 when it is zero, and 0x600A000 when it is one.

The Sound System

The sound system in the Game Boy Advance comprises four FM synthesis channels for generating sound effects and music, primarily for backward compatibility. The Game Boy Advance also features two new 16-bit stereo digital sound channels capable of outputting sampled sound effects and music tracks. There is no built-in sound mixer for synchronous sound playback, so programmers must write their own sound mixers or use a third-party library. A sound mixer allows multiple sounds to be played at the same time. Conceptually, it does this by "mixing" them together, which is where the name comes from.Without a sound mixer, it is only possible to play one sound at a time, which is called *asynchronous playback*. I will cover the sound system in more detail in Chapter 9, "The Sound System."

Summary

The Game Boy Advance is truly an advanced handheld video game system that is worthy of the accolades it has received in the development community and by gamers themselves. Many of the best games of all time are being ported to Game Boy Advance because it has the processing power to handle high-end games that are either 2D or 3D. By understanding at least the high-level view of the Game Boy Advance architecture, you are able to better judge the type of game that is or is not possible—and then challenge those possibilities!

Chapter Quiz

The following quiz will help to further reinforce the information covered in this chapter. The quiz consists of 10 multiple-choice questions with up to four possible answers. The answers may be found in Appendix D, "Answers To The Chapter Quizzes."

1. What is the name of the new Game Boy model released in 2003?
    A. Game Boy Advance SP
    B. Game Boy SP
    C. Super Game Boy
    D. Game Boy Advance Pro

2. What processor was used in the Game Boy Color?
    A. 6802
    B. 8086
    C. Z80
    D. 68000

3. How much memory does the Game Boy Advance have?
    A. 8 KB
    B. 64 KB
    C. 128 KB
    D. 384 KB

4. When was the first Game Boy released in the United States?
    A. 1879
    B. 1983
    C. 1989
    D. 1991

5. What is the full name of the main processor inside the Game Boy Advance?

A. 80386 DX4-100
B. ARM7TDMI
C. 68002
D. 6501

6. What is the designation for the memory that is built into the processor?

A. EWRAM
B. VRAM
C. IWRAM
D. CRAM

7. What is the maximum frame rate of the Game Boy Advance as limited by the hardware refresh?

A. 120
B. 30
C. 80
D. 60

8. What are video modes 0, 1, and 2 called?

A. Tile-based modes
B. Bitmap-based modes
C. Sprite-based modes
D. Background modes

9. What are video modes 3, 4, and 5 called?

A. Tile-based modes
B. Bitmap-based modes
C. Sprite-based modes
D. Background modes

10. How many sound channels, overall, does the Game Boy Advance have?

A. 2
B. 4
C. 6
D. 8