第 1 章　嵌入式系统概述

在科技高速发展的今天，几乎随处可见嵌入式设备，嵌入式系统已经全面渗入我们的日常生活中，如便携式、嵌入式终端产品：MP3、MP4、PDA、手机、数码相机、GPS 导航等，大型的嵌入式设备如车载电子、智能家电等。如今，人们的生活几乎离不开嵌入式系统，它带给了我们许多人性化的便捷服务，使得我们的生活和工作更加高效。本章将讲述嵌入式系统基本概念、系统组成，介绍几种主流的嵌入式操作系统，以及本书后续内容涉及的三星 S3C2440 嵌入式微处理器的硬件资源、编程模型和 ARM 指令集，最后介绍了本文中所使用开发板的硬件结构。

1.1　嵌入式系统基本概念

嵌入式系统（Embedded system）是一种为特定应用而设计的专用计算机系统，英国电器工程师协会（U.K. Institution of Electrical Engineer）定义嵌入式系统是控制、监视或者辅助设备、机器和车间运行的装置。目前国内普遍认为嵌入式系统是以应用为中心、以计算机技术为基础，软硬件可裁剪，应用系统对功能、成本、体积、功耗、可靠性严格要求的专用计算机系统。

我们可以这样认为，嵌入式系统是一种专用的计算机系统，作为装置或设备的一部分。与通用的个人计算机（PC）系统不同，嵌入式系统通常执行的是带有特定要求的预先定义的任务。由于嵌入式系统只针对一项特殊的任务，设计人员能够对它进行优化，减小系统尺寸，从而降低生产成本。通过以下嵌入式系统的特点，我们可以更系统地了解嵌入式系统的基本概念：

（1）系统内核小。由于嵌入式系统一般资源相对有限，因此系统内核普遍都很小。比如 Enea 公司的 OSE 分布式系统，内核只有 5KB。

（2）专用性强。嵌入式系统的个性化很强，其中的软件系统和硬件的结合非常紧密，应用于不同硬件的嵌入式操作系统一般要针对硬件进行系统的移植。

（3）系统精简。嵌入式系统一般没有系统软件和应用软件的明显区分，功能设计及实现上无需过于复杂，这样既利于控制系统成本，也实现了系统安全。

（4）高实时性。嵌入式软件要求固态存储，代码要求高质量和高可靠性，以提高速度和保证高实时性。

（5）嵌入式系统开发需要开发工具和环境。嵌入式系统本身不具备自主开发能力，开发者必须通过一套开发工具和环境才能对嵌入式系统进行开发。

（6）多任务的操作系统。区别于普通单片机系统，为了合理地调度多个任务，利用系统资源、系统函数以及专家库函数接口，用户必须自行配置实时操作系统（Real-Time Operating System，RTOS）开发平台，这样才能保证程序执行的实时性、可靠性，并减少开发时间，保证软件质量。

实际上，凡是与产品结合在一起的、具有嵌入式特点的控制系统都可以称为嵌入式系统，一个手持的 MP3 和一个 PC104 的微型工业控制计算机都可以认为是嵌入式系统，因此有时很难给它下一个准确的定义。总之，嵌入式系统是采用“量体裁衣”的方式把所需要的功能嵌入到各种应用系统中。

1.2　嵌入式系统组成

嵌入式系统是由嵌入式处理器、存储器等硬件系统和嵌入式操作系统、应用程序等软件系统组成，如图 1.2.1 所示。

图 1.2.1 嵌入式系统组成结构

1．嵌入式硬件系统

嵌入式系统的硬件系统包括嵌入式微处理器、存储器（SDRAM、ROM、Flash 等）、通用设备接口及 I/O 接口。以嵌入式微处理器为核心，配置必要的外围接口部件，如电源电路、时钟电路和存储器电路等，就构成了一个嵌入式核心控制模块。

1）嵌入式微处理器

嵌入式微处理器与通用 CPU 最大的区别在于嵌入式微处理器大多工作在为特定用户群所专门设计的系统中，它将通用 CPU 许多由板卡完成的任务集成在芯片内部，从而有利于嵌入式系统在设计时趋于小型化，同时还具有很高的效率和可靠性。

嵌入式微处理器的体系结构可以采用冯·诺依曼体系或哈佛体系结构，指令系统可以选用精简指令系统（Reduced Instruction Set Computer，RISC）或复杂指令系统（Complex Instruction Set Computer，CISC）。嵌入式微处理器有各种不同的体系，即使在同一体系中也可能具有不同的时钟频率和数据总线宽度，或集成了不同的外设和接口。目前，市场上主流的体系有 ARM、MIPS、PowerPC、X86 和 SH 等，但与全球 PC 市场不同的是，没有一种嵌入式微处理器可以主导市场。仅以 32 位的微处理器产品而言，就有 100 种以上，用户可以根据具体的应用决定选择不同的嵌入式微处理器。

为提高嵌入式系统的实时性和高可靠性，同时满足体积、功耗及成本要求，在嵌入式系统设计中，应尽可能选择适用于系统功能接口的 SoC 芯片，以最少的外围部件构成一个应用系统。

2）存储器

嵌入式系统需要存储器来存放和执行代码。嵌入式系统的存储器包含 Cache、主存储器和辅助存储器。

Cache 是一种容量小、速度快的存储器阵列。它位于主存和嵌入式微处理器内核之间，全部功能由硬件实现。Cache 存放的是最近一段时间微处理器使用最多的程序代码和数据，可分为数据 Cache、指令 Cache 或混合 Cache，在需要进行数据读取操作时，微处理器尽可能从 Cache 中读取数据，而不是从主存中读取，这样就大大改善了系统的性能，提高了微处理器和主存之间的数据传输速率。Cache 的大小因不同处理器而定，一般中高档的嵌入式微处理器才会把 Cache 集成进去。

主存是嵌入式微处理器能直接访问的寄存器，用来存放系统和用户的程序及数据。它可以位于微处理器的内部或外部，其容量为 256KB～1GB，根据具体的应用而定。一般片内存储器容量小，速度

快，片外存储器容量大。常用作主存的存储器有：NOR Flash、EPROM 和 PROM 等只读存储器（Read-Only Memory），以及 SRAM、DRAM 和 SDRAM 等随机存储器（Random Access Memory）。其中 NOR Flash 凭借其可擦写次数多、存储速度快、存储容量大、价格便宜等优点，在嵌入式领域内得到了广泛应用。

辅助存储器用来存放大数据量的程序代码或信息，它的容量大但读取速度比主存慢很多，常用来长期保存用户的信息。嵌入式系统中常用的外存有：硬盘、NAND Flash、CF 卡、MMC 和 SD 卡等。

3）通用设备接口及 I/O 接口

嵌入式系统通过通用设备接口及 I/O 接口与外界实现一定形式的交互，外设再通过与片外其他设备或传感器的连接来实现微处理器的输入/输出功能。目前嵌入式系统中常用的通用设备接口有 A/D（模/数转换接口）、D/A（数/模转换接口），I/O 接口有 UART（串行通信接口）、Ethernet（以太网接口）、USB（通用串行总线接口）、音频接口、VGA 视频输出接口、I^2C（现场总线）、SPI（串行外围设备接口）和 IrDA（红外线接口）等。这些也是我们设计嵌入式系统时经常会涉及的接口。

2．嵌入式软件系统

嵌入式系统软件一般可分为 4 个部分：设备驱动、实时操作系统（RTOS）、应用程序接口（API）及应用程序。也有些书籍将应用程序接口归属于实时操作系统。

1）设备驱动

驱动是嵌入式系统中不可缺少的重要部分，使用任何外设都需要有相应驱动程序的支持，它为上层软件提供了控制设备的接口。上层软件不用理会设备的具体内部操作，只需调用驱动程序提供的接口即可。驱动程序一般包括硬件抽象层 HAL、板极支持包 BSP 和设备驱动程序。

2）实时操作系统（RTOS）

嵌入式实时操作系统是一种用途广泛的系统软件，具有通用操作系统的基本特点，如负责嵌入式系统的全部软硬件资源的分配、任务调度及控制、协调并发活动；同时，它必须体现其所在嵌入式系统的特征，如能够通过装卸某些模块来达到系统所要求的功能，再通过内核映像的形式下载到目标系统中。内核中通常必需的基本部件是进程调度、进程间通信及内存管理；其他部件如文件系统、驱动程序、网络协议等都可根据用户要求进行配置。

较之通用操作系统，嵌入式操作系统在系统实时高效性、硬件的相关依赖性、软件固化以及应用的专用性等方面具有较突出的特点。在下节中将具体介绍几种常用的嵌入式操作系统。

3）应用程序接口（API）

应用程序接口（Application Programming Interface，API）是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。嵌入式操作系统下的 API 和一般操作系统下的 API 在功能、含义及知识体系上完全一致。可这样理解 API：在计算机系统中有很多可通过硬件或外部设备去执行的功能，这些功能的执行可通过计算机操作系统或硬件预留的标准指令调用，而软件人员在编写应用程序时，就不需要为每种可通过硬件或外设执行的功能重新编写程序，只需按系统或某些硬件事先提供的 API 调用即可完成功能的执行。因此在操作系统中提供标准的 API 函数，可加快用户应用程序的开发，统一应用程序的开发标准，也为操作系统版本的升级带来了方便。在 API 函数中，提供了大量的常用模块，可大大简化用户应用程序的编写。

4）嵌入式系统应用程序

嵌入式应用软件是基于特定的硬件平台开发的，直接为终端用户完成某些特定功能所设计的程序。开发者主要通过调用系统的 API 函数对系统进行操作，完成应用功能的开发。在用户的应用程序

中，也可创建用户自己的任务，任务之间的协调主要依赖于系统的消息队列。其特点在于，软件要求固化在存储器中、代码质量要求高和软件实时性高。大多数嵌入式设备的应用软件和操作系统是紧密结合的，这也是嵌入式系统与通用系统最大的区别。

1.3 主流嵌入式操作系统

随着集成电路规模的不断提高，涌现出大量价格低廉、结构小巧、功能强大的微处理器，这样给嵌入式系统提供了丰富的硬件平台。从 20 世纪 80 年代开始，陆续出现了一些嵌入式操作系统。这些操作系统经过不断的发展，已经逐渐成熟，在各个领域得到了广泛的应用。目前，比较常用的嵌入式操作系统有 VxWorks、μC/OS-II、Linux 和 Windows CE 等。

1.3.1 VxWorks

VxWorks 是美国 Wind River System 公司（即风河公司，简称 WRS 公司）于 1983 年设计开发的一种无内存管理单元（Memory Management Unit，MMU）的嵌入式实时操作系统。它具有良好的持续发展能力、高性能的内核以及友好的用户开发环境，特别的 VxWorks 只占用很小的存储空间，并可以高度裁剪，保证了系统能以高效率运行，因此被广泛应用于通信、军事、航空等高精度技术领域及实时性要求极高的领域，如卫星通信、军事演习、弹道制导、飞机导航等。如美国 F-16 与 FA-18 战斗机、爱国者导弹及火星探测车上都使用了 VxWorks 操作系统。

Tornado 是为开发 VxWorks 应用系统提供的集成开发环境。Tornado 中包含的工程管理软件可将用户自己写的代码与 VxWorks 的核心有效的组合起来，可按照用户的具体需求，裁剪配置 VxWorks 内核。VxSim 原型仿真器可让程序员在不用目标机的情况下，直接开发系统原型，做出系统评估。CrossWind 调试器可提供任务级和系统级的调试模式，还可实现多目标机的联调，功能十分强大。优化分析工具可帮助程序员以多种方式真正地观察、跟踪系统的运行，排除人的思维很难检查出的逻辑错误，优化性能。

1.3.2 μC/OS-II

μC/OS-II 是由美国工程师 Jean J. Labrosse 开发的实时操作系统内核。这个内核的产生与 Linux 优点相似，他花了一年多的时间开发了这个最初名为 μC/OS 的实时操作系统，于 1992 年将介绍文章发表在嵌入式系统编程杂志上，其源代码公布在该杂志的网站上，1993 年出书。这本书的热销以及源代码的公开推动了 μC/OS-II 本身的发展，已有成千上万的开发者把它成功地应用于各种系统，安全性和稳定性也已得到认证。μC/OS-II 目前已经被移植到 Intel、Philips、Motorola 等公司不同的处理器上。

μC/OS-II 是一种可移植，可裁剪，抢占式的典型实时操作系统内核，它总是执行处于就绪队列中优先级最高的任务。μC/OS-II 公开源代码，它只包含了进程调度、时钟管理、内存管理和进程间的通信与同步等基本功能，没有 I/O 管理、文件系统、网络等额外模块，全部核心代码只有 8.3KB，可以简单的视其为一个多任务调度器。

μC/OS-II 的源代码绝大部分是用 C 语言编写的，经过编译就能在 PC 上运行。仅有与 CPU 密切相关的一部分是用汇编语言写成的。在 μC/OS-II 操作系统中涉及系统移植的源代码文件只有 3 个，只要编写 4 个汇编语言的函数、6 个 C 函数、定义 3 个宏和 1 个常量，代码长度不过二三百行，移植起来并不困难。它的应用非常广泛，如医疗器械、音响设备、发动机控制、高速公路电话系统、自动提款机及航空航天领域等。

1.3.3　Windows CE

Windows CE 是 Microsoft 公司于 1996 年推出的一个 32 位、多线程、多任务的嵌入式软实时操作系统。它的核心全部是由 C 语言开发的，操作系统本身还包含许多由各个厂家用 C 语言和汇编语言开发的驱动程序。Windows CE 的内核提供内存管理、抢先多任务和中断处理功能。内核的上面是图形用户界面（Graphical User Interface，GUI）和桌面应用程序。在 GUI 内部运行着所有的应用程序，而且多个应用程序可以同时运行。

与 Windows 98/NT 的 API 相比，Windows CE 的 API 是微缩版的 Win32 API，且专门为体积小、资源要求低、便携的机器而设计，是桌面 Windows 系统 API 的一个子集。因此，许多基于微软桌面 Windows 操作系统开发的应用程序只需要经过少许改动就能应用于 Windows CE 中。

Windows CE 拥有完善的软件支持开发工具。Windows CE 的核心移植和驱动开发使用专门的操作系统定制工具：Windows CE Platform Builder。而应用程序的开发则有嵌入式开发工具包 Embedded Visual C++和 Embedded Visual Basic 等。在 Embedded Visual Tools 下还可以进行部分驱动程序的开发。同时，在 Windows CE 中还提供了用于 Windows CE 开发的 Bootloader：Eboot。

Windows CE 是针对有限资源的平台而设计的多线程、完整优先权、多任务的操作系统，但它不是一个强实时操作系统，高度模块化是它的重要特性，它适合作为可裁剪的 32 位嵌入式操作系统。Windows CE 凭借其优秀的人机交互界面，既适用于工业设备的嵌入式控制模块，也适用于消费类电子产品，如电话、机顶盒和掌上计算机等。针对不同的目标设备硬件环境，可以在内核基础上添加各种模块，从而形成一个定制的嵌入式操作系统。但是，由于它内核相对 VxWorks 等操作系统显得很笨拙，占用内存过大，应用程序庞大，实时性不强，因此在通信、军事、航空、航天等高端技术领域、实时性要求极高的领域，以及系统硬件资源非常有限的应用中无法得到使用。

1.3.4　嵌入式 Linux

Linux 是由芬兰赫尔辛基大学学生 Linux Torvalds 在 1991 年开发的源码开放的类 UNIX 操作系统。通过对标准 Linux 进行内核裁剪和优化后形成了适用于专用场合的软实时、多任务嵌入式 Linux。由于它是免费的，没有其他商业性嵌入式操作系统需要的许可证费用，所以得到了 IT 巨头的支持。嵌入式 Linux 的设计也随着广泛应用而获得了巨大的成功。

目前，Linux 已经拥有了许多版本，包括强实时的嵌入式 Linux（RT-Linux）和一般的嵌入式 Linux（μClinux）。其中，RT-Linux 通过把通常的 Linux 任务优先级设为最低，而所有的实时任务的优先级都高于它，以达到既兼容通常的 Linux 任务，又保证强实时性的目的；而 μClinux 对 Linux 经过小型化裁剪后，能够固化在容量只有几百 KB 或几 MB 的存储器芯片或单片机中，是针对没有 MMU 的处理器而设计的。它不能使用处理器的虚拟内存管理技术，即对内存的访问是直接的，所有程序中的地址都是实际的物理地址。

嵌入式 Linux 可移植到多个不同结构的嵌入式微处理器和硬件平台上，性能稳定，支持升级能力，而且开发容易。其特点有：

① 源代码开放，易于定制裁剪，在价格上极具竞争力；

② 内核小、功能强大、运行稳定、效率高；

③ 支持多种嵌入式微处理器体系；

④ 有多种成熟的开发工具和开发环境；

⑤ 可方便地获得众多第三方软硬件厂商的支持；

⑥ Linux 内核的结构在网络方面非常完整，它提供了对 10M/100M/1000M 以太网、无线网络、令

牌网、光纤网、卫星等多种联网方式的全面支持。此外在图像处理、文件管理及多任务支持等方面也都非常出色。

一个可用的 Linux 系统包括两个部分：内核和应用程序。Linux 内核为应用程序提供了一个虚拟的硬件平台，以统一的方式对资源进行访问，并且透明地支持多任务。它可分为 6 部分：进程调度、内存管理、文件管理、进程间通信、网络和驱动程序。Linux 应用程序包括系统的部分初始化、基本的人机界面和必要的命令等内容。由于嵌入式系统越来越追求数字化、网络化和智能化，因此要求整个系统必须是开放的、提供标准的 API，并能够方便地与第三方的软硬件沟通，这也是 Linux 成为主流嵌入式操作系统的主要原因。

1.4 ARM 处理器系列

ARM（Advanced RISC Machines）既是一个公司的名字，也是一类微处理器的通称。1990 年 11 月，ARM 公司成立于英国剑桥，是专门从事基于 RISC 技术芯片设计开发的公司，主要出售芯片设计技术的授权，作为知识产权供应商，本身不直接从事芯片生产，靠转让设计许可由合作公司生产各具特色的芯片。半导体生产商从 ARM 公司购买其设计的 ARM 微处理器核，根据各自不同的应用领域，加入适当的外围电路，从而形成自己的 ARM 微处理器芯片进入市场。

1．ARM7 系列

ARM7 系列微处理器包括 ARM7TDMI、ARM7TDMI-S、ARM720T、ARM7EJ 几种类型，是低功耗的 32 位嵌入式 RISC 处理器。其中 ARM7TDMI 是目前使用最广泛的 ARM7 处理器，主频最高可达 130 MIPS，采用能够提供 0.9 MIPS/MHz 的三级流水线结构，支持 16 位 Thumb 指令集，支持片上 Debug，内嵌硬件乘法器（Multiplier），嵌入式 ICE，支持片上断点和调试点。指令系统与 ARM9 系列、ARM9E 系列和 ARM10E 系列兼容，支持 Windows CE、Linux、Palm OS 等操作系统。

2．ARM9 系列

ARM9 系列微处理器包含 ARM920T、ARM922T 和 ARM940T 几种类型，在高性能和低功耗两方面都有出色表现。采用 5 级整数流水线，指令执行效率更高。提供 2.1 MIPS/MHz 的哈佛结构，支持 32 位 ARM 指令集和 16 位 Thumb 指令集，支持数据 Cache 和指令 Cache，具有更高的指令和数据处理能力，支持 32 位的高速 AMBA 总线接口，全性能的 MMU，支持 Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统，MPU 支持实时操作系统。

3．ARM9E 系列

ARM9E 系列微处理器包含 ARM926EJ-S、ARM946E-S 和 ARM966E-S 三种类型，使用单一的处理器内核提供了微控制器、DSP、Java 应用系统的解决方案，减少了芯片的面积和系统的复杂程度。ARM9E 系列微处理器提供了增强的 DSP 处理能力，适合于那些需要同时使用 DSP 和微控制器的应用场合。ARM9E 系列微处理器支持 DSP 指令集，采用 5 级整数流水线，支持数据 Cache 和指令 Cache，主频最高可达 300 MIPS，支持 32 位 ARM 指令集和 16 位 Thumb 指令集，支持 32 位的高速 AMBA 总线接口，支持 VFP9 浮点处理协处理器，MMU 支持 Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统，MPU 支持实时操作系统。

4．ARM10E 系列

ARM10E 系列微处理器包含 ARM1020E、ARM1022E 和 ARM1026EJ-S 几种类型，由于采用了新

的体系结构，与同等的 ARM9 器件相比较，在同样的时钟频率下，性能提高了近 50%。同时采用了两种先进的节能方式，使其功耗极低。ARM10E 系列微处理器支持 DSP 指令集，采用 6 级整数流水线，指令执行效率更高，支持数据 Cache 和指令 Cache，主频最高可达 400MIPS，支持 32 位 ARM 指令集和 16 位 Thumb 指令集，支持 32 位的高速 AMBA 总线接口，支持 VFP10 浮点处理协处理器，MMU 支持 Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统，内嵌并行读/写操作部件。

5．SecurCore 系列

SecurCore 系列微处理器包含 SecurCore SC100、SecurCore SC110、SecurCore SC200 和 SecurCore SC210 几种类型。SecurCore 系列微处理器除了具有 ARM 体系结构各种主要特点外，在系统安全方面还有如下出色表现：

① 带有灵活的保护单元，以确保操作系统和应用数据的安全；

② 采用软内核技术，防止外部对其进行扫描探测；

③ 可集成用户自己的安全特性和其他协处理器。

6．StrongARM 系列

StrongARM 系列处理器包含 SA110 处理器、SA1100、SA1110PDA 系统芯片和 SA1500 多媒体处理器芯片等，是 Intel 公司开发的 ARM 体系结构高度集成的 32 位 RISC 微处理器，采用在软件上兼容 ARMv4 且具有 Intel 技术优点的体系结构。

7．XScale 微处理器

Intel XScale 是一款基于 ARMv5TE 体系结构的高性能、超低功耗微处理器。它分别拥有 32KB 的数据 Cache 和 32KB 的指令 Cache 以及 7 级超级流水线，指令执行效率更高。它在 0.75V 时工作频率最高可达 150 MHz，在 1.0V 时工作频率可达 400MHz，在 1.65V 下工作频率则可高达 800 MHz。超低功率与高性能的特色使得 Intel XScale 处理器广泛应用于互联网接入设备，数字移动电话，个人数字处理产品等领域。

1.5　S3C2440A 处理器

1.5.1　S3C2440A 简介

S3C2440A 是三星公司开发生产的 16/32 位精简指令集（RISC）微处理器，其处理器核心是由 ARM 公司设计的 16/32 位 ARM920T 的 RISC 处理器。ARM920T 实现了 MMU，AMBA 总线和哈佛结构高速缓冲体系结构。这一结构具有独立的 16KB 指令 Cache 和 16KB 数据 Cache。每个 Cache 都是由具有 8 字节长的行组成。它采用了新的总线架构如先进微控制总线构架（AMBA）。低功耗、简单、精致，且全静态的独特设计使其适合于对成本和功率敏感型的应用。为了降低整体系统成本，S3C2440A 还提供了丰富的内部设备。通过提供一套完整的通用系统外设，S3C2440A 开发者无需配置额外的组件，减少了整体系统成本。图 1.5.1 所示为 S3C2440A 系统结构框图。以下将根据结构框图，简单介绍 S3C2440A 处理器特性。

1．体系结构

（1）手持设备的完整系统和普通嵌入式应用；

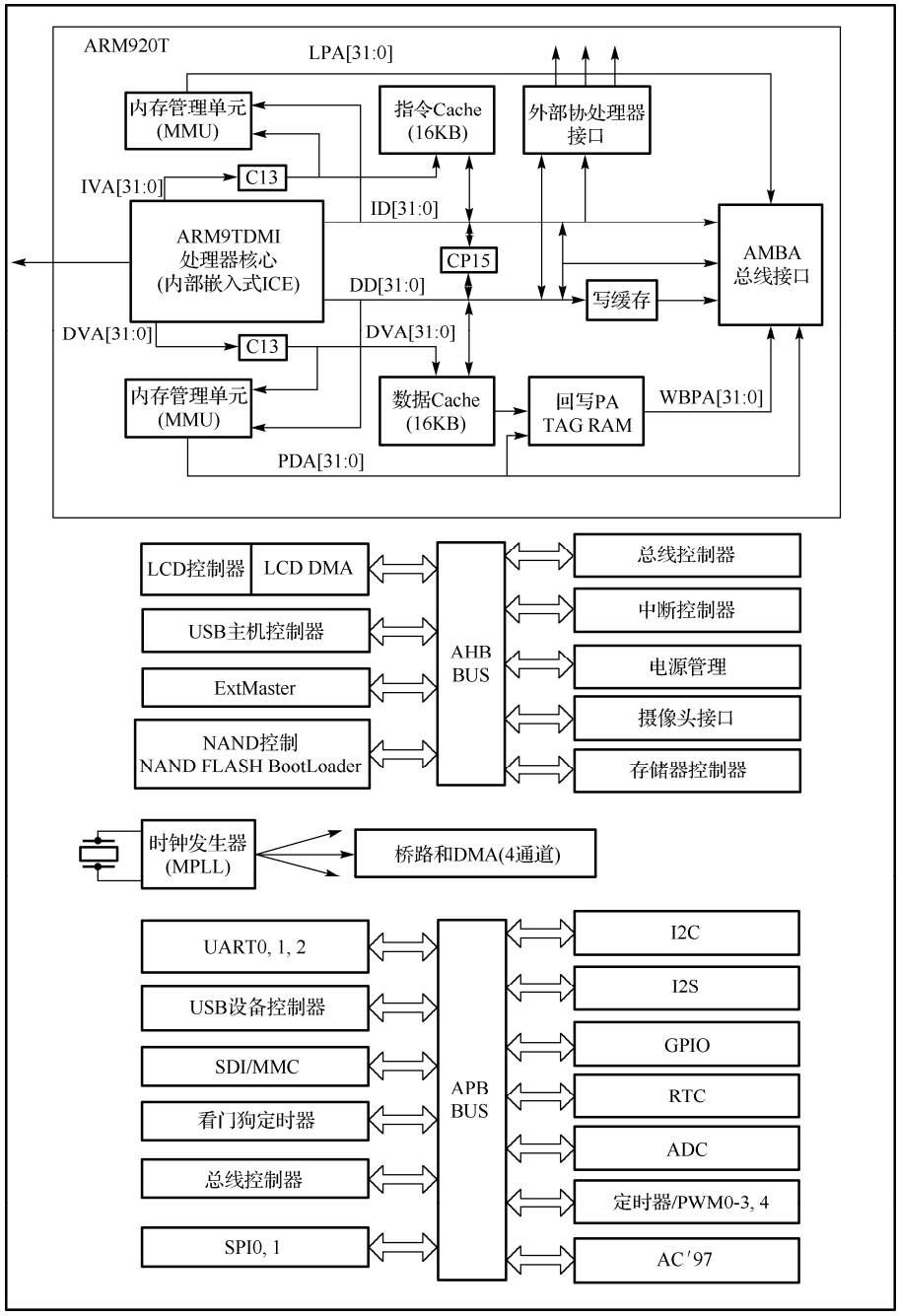

图 1.5.1　S3C2440A 系统结构框图

（2）16/32 位 RISC 体系架构和 ARM920T CPU 核心的强大的指令集；

（3）增强型 ARM 架构 MMU 以支持 WinCE，EPOC 32 和 Linux；

（4）指令高速缓存，数据高速缓存，写缓冲和物理地址 TAG RAM 以减少执行主存储器带宽和延迟性能的影响；

（5）ARM920T CPU 核支持 ARM 调试架构；

（6）内部先进微控制器总线架构（AMBA2.0，AHB/APB）。

2．系统管理

（1）支持大/小端；
（2）地址空间：每 Bank 128M 字节（总共 1G 字节）；
（3）支持可编程的每 Bank 8/16/32 位数据总线宽度；
（4）BANK 0 到 BANK 6 固定 Bank 的起始地址；
（5）BANK 7 具有可编程 Bank 起始地址和大小；
（6）8 个存储器 Bank：其中 6 个存储器 Bank 为 ROM，SRAM 和其他；2 个存储器 Bank 为 ROM/SRAM/SDRAM；
（7）所有存储器具备完整可编程访问周期；
（8）支持外部等待信号来扩展总线周期；
（9）支持 SDRAM 掉电时自刷新模式；
（10）支持从各种类型 ROM 启动（NOR/NAND Flash，EEPROM 或其他）。

3．NAND Flash 启动引导（BootLoader）

（1）支持从 NAND Flash 启动；
（2）4KB 的启动内部缓冲区；
（3）支持启动后 NAND Flash 作为存储器；
（4）支持先进 NAND Flash。

4．高速缓存存储器

（1）64 路指令缓存（16KB）和数据缓存（16KB）的组相联高速缓存；
（2）每行 8 字节长度，其中含一个有效位和两个 dirty 位；
（3）伪随机或循环 robin 置换算法；
（4）执行直写或回写高速缓存刷新主存储器；
（5）写缓冲区可以保存 16 字的数据和 4 个地址。

5．时钟和电源管理

（1）片上 MPLL 和 UPLL：UPLL 产生时钟运作 USB 主机/设备；MPLL 产生时钟运作 1.3V 下最高 400MHz 的 MCU。
（2）用软件可以有选择的提供时钟给各功能模块。
（3）电源模式：普通、慢速、空闲和睡眠模式。其中普通模式：正常运行模式；慢速模式：无 PLL 的低频率时钟；空闲模式：只停 CPU 的时钟；睡眠模式：关闭包括所有外设的核心电源。
（4）EINT[15:0]或 RTC 闹钟中断触发从睡眠模式中唤醒。

6．中断控制器

（1）60 个中断源（1 个看门狗，5 个定时器，9 个 UART，24 个外部中断，4 个 DMA，2 个 RTC，2 个 ADC，1 个 IIC，2 个 SPI，1 个 SDI，2 个 USB，1 个 LCD ，1 个电池故障，1 个 NAND ，2 个摄像头，1 个 AC′97）；
（2）外部中断源中电平/边沿模式；
（3）可编程边沿和电平的极性；
（4）支持快速中断请求（FIQ ）给非常紧急的中断请求。

7．脉宽调制（PWM）定时器

（1）4 通道 16 位具有 PWM 功能的定时器，1 通道 16 位基于 DMA 或基于中断运行的内部定时器；
（2）可编程的占空比，频率和极性；
（3）能产生死区；
（4）支持外部时钟源。

8．内部设备

（1）1.2V 内核供电，1.8V/2.5V/3.3V 储存器供电，3.3V 外部 I/O 供电，具备 16KB 的指令缓存和 16KB 的数据缓存和 MMU 的微处理器；
（2）外部存储控制器（SDRAM 控制和片选逻辑）；
（3）LCD 控制器（最大支持 4K 色 STN 和 256K 色 TFT）提供 1 通道 LCD 专用 DMA；
（4）4 通道 DMA 并有外部请求引脚；
（5）3 通道 UART（IrDA1.0，64 字节发送 FIFO 和 64 字节接收 FIFO）；
（6）2 通道 SPI；
（7）1 通道 IIC 总线接口（支持多主机）；
（8）1 通道 IIS 总线音频编码器接口；
（9）AC’97 编解码器接口；
（10）兼容 SD 主接口协议 1.0 版和 MMC 协议 2.11 兼容版；
（11）2 通道 USB 主机/1 通道 USB 设备（1.1 版）；
（12）4 通道 PWM 定时器和 1 通道内部定时器/看门狗定时器；
（13）8 通道 10 位 ADC 和触摸屏接口；
（14）具有日历功能的 RTC；
（15）摄像头接口（最大支持 4096×4096 像素输入，2048×2048 像素输入支持缩放）；
（16）130 个通用 I/O 口和 24 通道外部中断源；
（17）具有普通，慢速，空闲和掉电模式；
（18）具有 PLL 片上时钟发生器。

1.5.2 基本编程模型

（1）处理器运行状态

S3C2440 的处理器运行状态一共有两种：ARM 状态和 Thumb 状态。ARM 状态下执行的是 ARM 指令。Thumb 状态下执行的是 Thumb 指令。ARM 指令是以 32 位对齐的，而 Thumb 指令则是 16 位对齐的。

（2）状态切换

Thumb 状态和 ARM 状态可以互相切换，当程序执行一个 BX 指令的时候（设置操作数寄存器），处理器就进入了 Thumb 状态。如果 Thumb 状态下发生异常，当异常返回时，处理器将自动切换回 Thumb 状态。类似的，也可以通过执行 BX 指令进入 ARM 状态。值得注意的是当处理器发生异常时，将会进入 ARM 状态。

（3）存储格式

S3C2440A 可以处理大端格式和小端格式。

（4）数据类型

ARM920T 支持字节（8 位）、半字（16 位）和字（32 位）的数据类型。字必须按 4 字节对齐边界，半字必须按 2 字节对齐边界。

（5）运行模式

S3C2440 中一共有 7 种运行模式，分别为用户模式、快中断模式，中断模式、管理模式、中止模式、系统模式、未定义模式。它们的具体作用如下。

用户模式：当程序正常执行的情况下，处理器运行在用户模式。

快中断模式：快中断模式是 S3C2440A 为了数据的高速传输和通道处理而设计的。

中断模式：中断模式是为了满足处理器的一般中断处理而设计的。

管理模式：管理模式是操作系统所使用的保护模式。

中止模式：中止模式用于数据的存储保护。

系统模式：系统模式是操作系统的特权模式，可以放完所有的资源，并且可以进行处理器模式的切换。

1.5.3　ARM 寄存器

S3C2440A 中的寄存器一种有 37 个。包括程序寄存器在内，通用的寄存器有 31 个。通用寄存器都是 32 位的。除了通用寄存器，S3C2440A 还有 6 个状态寄存器。

通用寄存器包括：R0～R15、R13_svc、R14_svc、R13_abt、R14_abt、R13_und、R13_irq、R14_irq、R8_frq～R14_fig。其中 R13 通常为堆栈指针，R14 为链接寄存器，R15 为程序计数器。

状态寄存器包括：CPSR、SPSR_sev、SPSR_abt、SPSR_und、SPSR_irq、SPSR_fiq。

S3C2440A 一共有 7 种处理器模式，在不同的模式下所能访问的寄存器是不同的。其中所有的处理器模式下，都能够访问的有 15 个通用寄存器（R0～R15）、一个或者两个状态寄存器以及程序计数器。

1．未分组寄存器（R0～R7）

在处理器中，未分组寄存器在所有的运行模式中都是共用的。在中断或者异常处理的时候，会引起处理器运行模式的切换，这些切换可能会引起寄存器中数据的破坏。所以，在程序开发过程中必须引起足够的重视。

2．分组寄存器（R8～R14）

分组寄存器所对应的物理寄存器与处理器的运行模式有关。其中，R8～R12 分别对应了两组不同的物理寄存器，在快速中断模式下，访问 R8_fiq～R12_fiq。在除了快速中断以外的模式下，访问 R8_usr～R12_usr。而 R13～R14 中，每个寄存器对应了 6 个不同的物理寄存器。除了系统模式和用户模式是共同使用同一个物理寄存器以外，其他 5 种运行模式都有自己独有的物理寄存器。这些寄存器可以使用采用 R13_<mode>或者 R14<mode>的方式来表示，其中 mode 表示运行模式，分别是：usr、svc、abt、und、irq、fiq。

ARM 指令中，寄存器 R13 一般用作堆栈指针 SP，当然也可以使用其他寄存器当做 SP，只是习惯上将 R13 作为堆栈指针。在 Thumb 指令集中，有一些指令必须使用 R13 作为 SP。

R14 称为链接寄存器。当程序中执行 BL 或者 BLX 指令来调用子程序的时候，R14 会将程序计数器 PC 中的值复制并保存起来。当子程序程序返回时，R14 中保存的值被重新复制到程序计数器 PC 中。当发生异常的时候，对应的异常模式的 R14（R14_svc、R14_irq、R14_fiq、R14_abt、R14_und）用来保存程序寄存器 PC 中的值。R14 也可以当作通用寄存器来使用。

3．程序计数器（PC）

R15 通常用作程序计数器 PC。在 ARM 状态下，位[1:0]为 0，位[31:2]为 PC 值；当处于 Thumb 状态下，位[0]为 0，位[31:1]为 PC 值。由于 ARM 采用了流水线技术，PC 值所指向的地址是正在取指

令的地址，而不是正在执行的地址，PC 指向的是当前正在执行指令的下两条指令的地址。通常情况下，R15 不会作为通用寄存器来使用。

4．CPSR 与 SPSR

CPSR 指的是当前程序状态寄存器，其中存储着当前程序的运行状态。CPSR 可以在所有的处理器模式下访问，而且各运行模式共享同一个 CPSR。ARM 中除了用户模式和系统模式外，其余的异常模式中都有一个独立的备份的程序状态寄存器，称为 SPSR。当发生异常时，SPSR 负责保存 CPSR 中的值，当异常返回时，可以使用 SPSR 中的值来恢复 CPSR。

CPSR 中包含条件标志位、中断禁止位、当前处理器模式与一些状态和控制信息。其中条件标志位分别为 N、Z、C、V。大多数的 ARM 指令会根据这些标志位有条件地执行，而在 Thumb 状态下，只有分支指令是有条件执行的。CPSR 中的低 8 位（I、F、T、M[4:0]）称为控制位。I 位和 F 位是中断禁止位。当 I 位置位时，IRQ 中断被禁止。当 F 位置位时，FIQ 中断被禁止。T 位是处理器的工作状态选择位，当 T 位置位时，处理器运行在 Thumb 状态；当 T 位清零时，处理器运行在 ARM 状态。M[4:0]为处理器模式选择位，决定了处理器的运行模式。

1.5.4 ARM 异常的种类

当正常的程序被打断而发生暂停 S3C2440A 就会进入异常模式。比如发生一个中断的时候，首先处理器将当前的状态信息进行保存，然后处理中断，当中断服务处理完成后，将现场恢复然后继续执行向下执行程序。

ARM 支持的异常类型一共有 7 种，分别是复位、未定义指令、软件中断、指令预取中止、数据中止、外部中断请求以及快速中断请求。

（1）复位：当处理器的复位电平有效的时候，会发生复位异常，程序将跳转到复位异常的处理程序处执行。系统加电以及系统复位都将引起复位异常的发生。

（2）未定义指令：当 ARM 处理器或者协处理器遇到无法处理的指令时，将发生未定义指令的异常中断。通常使用该异常进行软件仿真。

（3）软件中断：当处理器执行 SWI 指令时，将发生软件中断。一般用在用户模式下调用特权操作指令。可以利用软件中断可以进入管理模式，请求特定的管理功能。

（4）指令预取中止：当处理器预取的指令的地址不存在或者该地址无法访问的时候，存储器发送中止信号给处理器。应该注意的是，当预取的指令被执行时，指令预取中止异常才会发生。

（5）数据中止：当处理器访问的数据的地址不存在或者该数据的地址没有访问权限时，将发生数据中止异常。

（6）外部中断请求：当处理器的外部中断引脚有效且 CPSR 的 I 位为 0 的时候，处理器将发生外部中断。系统外设使用该中断来请求相应的中断服务程序。

（7）快速中断请求：当处理器的快速中断引脚有效且 CPSR 的 F 位为 0 时，处理器将发生快速中断请求。快速中断通常是为了支持数据传输和通道处理。

1.5.5 ARM 异常的处理

当处理器发生异常的时候，ARM 处理器会执行以下的步骤：

（1）为了让异常返回后程序能够从正确的地址开始执行，首先将下一条指令的地址存入链接寄存器（LR）中。需要注意的是，如果是 ARM 状态下，LR 保存的是下一条指令的地址。如果是 Thumb 状态下，LR 保存的则是 PC 的偏移量。

（2）将 CPSR 中的值复制到 SPSR 中，从而保存处理器当前的处理器状态，中断禁止位、以及其他的状态和控制信息。

（3）根据发生的异常，设置 CPSR 中相应的运行模式位。

（4）将异常向量地址填入程序计数器 PC 中，从而使得程序跳转到相应的异常处理服务程序中去。

当异常返回的时候，ARM 处理器执行以下步骤：

（1）将链接寄存器（LR）中的值减掉相应的偏移量后赋值给 PC 中。

（2）将 SPSR 中的值复制到 CPSR 中。

（3）如果进入异常时禁止了中断，在该步骤中清除。

1.6　ARM 指令集介绍

1.6.1　ARM 指令集概述

ARM 处理器是基于精简指令集处理器架构设计的。ARM 处理器的指令集分为 ARM 指令和 Thumb 指令。Thumb 指令是 ARM 指令的子集。其中 ARM 指令是 32 位的，以字对齐方式保存在存储器中，ARM 指令效率高但代码密度比较低。Thumb 指令是 16 位的，以半字对齐方式保存在存储器中，Thumb 具有较高的代码密度，而且保存了 ARM 大多数性能上的优势。

ARM 指令主要分为跳转指令、数据处理指令、程序状态寄存器（PSR）传输指令、Load/Store 指令、协处理器指令以及异常中断产生指令。

ARM 指令的格式如下：

```
<opcode>{<cond>} {S}  <Rd>,<Rn> {,<operand2>}
```

- <opcode>：指令助记符，表示不同的指令。
- <cond>：可选的条件码，为指令的执行条件。
- {S}：可选的后缀，表示指令是否影响 CPSR 中的值。
- <Rd>：此处为目标寄存器。
- <Rn>：此处为第一个操作数的寄存器。
- <operand2>：此处为第二个操作数。

1.6.2　数据处理指令

数据处理指令主要用来完成算术和逻辑运算，主要有数据传送指令、算数逻辑运算指令、比较指令以及乘法指令。

1．数据传送指令

（1）MOV 传送指令

```
MOV{cond} {S} <Rd>, <operand2>
```

MOV 指令将一个数据<operand2>传送到目标寄存器<Rd>中。

MOV 指令的例子如下：

```
MOV R1,#0x20 ; R1=0x20;
```

（2）MVN 传送指令

```
MVN{cond} {S} <Rd>, <operand2>
```

MVN 指令将<operand2>的数据按位取反后传送给目标寄存器< Rd>中。

MVN 指令的例子如下：

```
MVN R1 ,#0; R1=-1
```

2．算数逻辑运算指令

（1）ADD 加法运算指令

```
ADD{<cond>} {S} <Rd>, <Rn>, <operand2>
```

ADD 指令将<operand2>中的数据与寄存器<Rn>中的值相加后传送到目标寄存器<Rd>中。

ADD 指令的例子如下：

```
ADD R1, R1, #1; R1=R1+1
```

（2）SUB 减法运算指令

```
SUB{<cond>} {S} <Rd>, <Rn>, <operand2>
```

SUB 指令将寄存器<Rn>中的值减去<operand2>中的数据，然后将结果传送到目标寄存器<Rd>中。

SUB 指令的例子如下：

```
SUB R1, R1,#1; R1=R1-1
```

（3）ADC 带进位的加法

```
ADC{<cond>} {S} <Rd>, <Rn>, <operand2>
```

ADC 指令将<operand2>中的值与寄存器<Rn>中的值相加，然后再加上 CPSR 中 C 条件标志位，最后将结果传送到目标寄存器<Rd>。

ADC 指令的例子如下：

```
ADC R1 , R2 ,R3;
```

（4）RSB 逆向减法指令

```
RSB{<cond>} {S} <Rd>, <Rn>, <operand2>
```

RSB 指令将<operand2>中的值减去寄存器<Rn>，然后将结果传送到目标寄存器<Rd>中。

RSB 指令的例子如下：

```
RSB R2, R1, #0; R2=-R1
```

（5）SBC 带位减法指令

```
SBC{<cond>} {S} <Rd>, <Rn>, <operand2>
```

SBC 指令将寄存器<Rn>减去<operand2>中的数据，再减去 CPSR 中 C 条件标志位的非（如果 C 标志位为 0，则减去 1），然后将结果传送到目标寄存器<Rd>中。

SBC 指令的例子如下：

```
SBC R0, R0, R1;
```

（6）RSC 带位逆向减法指令

```
RSC{<cond>} {S} <Rd>, <Rn>, <operand2>
```

RSC 指令将<operand2>中的数据减去寄存器<Rn>中的值，再减去 CPSR 中 C 条件标志位，最后将计算结果传送到目标寄存器<Rd>中。

RSC 指令的例子如下：

```
RSC R0, R1, R2;
```

（7）AND 逻辑与操作指令

```
AND{<cond>} {S} <Rd>, <Rn>, <operand2>
```

AND 指令将<operand2>中的数据与寄存器<Rn>中的值按位作逻辑与操作，然后将结果传送到目标寄存器<Rd>中。

AND 指令的例子如下：

```
AND R1, R2, R3; R1=R2&R3
```

（8）ORR 逻辑或操作指令

```
ORR{<cond>} {S} <Rd>, <Rn>, <operand2>
```

ORR 指令将<operand2>中的数据与寄存器<Rn>中的值按位作逻辑或操作，然后将结果传送到目标寄存器<Rd>中。

ORR 指令的例子如下：

```
ORR R1, R1, #3;
```

（9）EOR 逻辑异或操作指令

```
EOR{<cond>} {S} <Rd>, <Rn>, <operand2>
```

EOR 指令将<operand2>中的数据与寄存器<Rn>中的值按位作逻辑异或操作，然后将结果传送到目标寄存器<Rd>中。

EOR 指令的例子如下：

```
EOR R1, R1, #3;
```

（10）BIC 位清除指令

```
BIC{<cond>} {S} <Rd>, <Rn>, <operand2>
```

BIC 指令将<operand2>中的数据与寄存器<Rn>中的值的反码按位作逻辑与操作，然后将结果传送到目标寄存器<Rd>中。

BIC 指令的例子如下：

```
BIC R2, R2, #0x0F;
```

3．比较指令

（1）CMP 比较指令

```
CMP{<cond>} <Rn>, <operand2>
```

CMP 指令将寄存器<Rn>中的值减去<operand2>中的数据，然后根据结果更新 CPSR 中的相应的条件标志位。

CMP 指令的例子如下：

```
CMP R0, R1;
```

（2）CMN 负数比较指令

```
CMN{<cond>} <Rn>, <operand2>
```

CMN 指令将寄存器<Rn>中的值加上<operand2>中的数据，然后根据结果更新 CPSR 中的相应的条件标志位。

CMN 指令的例子如下：

```
CMN R1, #1;
```

（3）TST 位测试指令

```
TST{<cond>} <Rn>, <operand2>
```

TST 指令将<operand2>中的数据与寄存器<Rn>中的值按位作逻辑与操作，然后根据结果更新 CPSR 中的相应的条件标志位。

TST 指令的例子如下：

```
TST R1, #0x01;
```

（4）TEQ 相等测试指令

```
TEQ{<cond>} <Rn>, <operand2>
```

TEQ 指令将<operand2>中的数据与寄存器<Rn>中的值按位作逻辑异或操作，然后根据结果更新 CPSR 中相应的条件标志位。

TEQ 指令的例子如下：

```
TEQ R1, R2;
```

4．乘法指令

（1）MUL 指令

```
MUL {<cond>} {S} <Rd>, <Rm>, <Rs>
```

MUL 是 32 位乘法指令，该指令将寄存器<Rm>中的值与寄存器<Rs>中的值相乘，然后取结果的低 32 位传送到寄存器<Rd>中。

MUL 指令的例子如下；

```
MUL R1, R2, R3;
```

（2）MLA 指令

```
MLA{<cond>} {S} <Rd>, <Rm>, <Rs>, <Rn>
```

MLA 是 32 位乘加指令，该指令先将寄存器<Rm>中的值与寄存器<Rs>中的值相乘，然后再加上第 3 个操作数（即<Rn>中的数据），最后取结果的低 32 位传送到寄存器<Rd>中。

MLA 指令的例子如下：

```
MLA R1, R0, R2, R3;
```

（3）SMULL 指令

```
SMULL{<cond>} {S} <RdLo>, <RdHi>, <Rm>, <Rs>
```

SMULL 是 64 位有符号乘法指令，该指令将寄存器<Rm>中的值与寄存器中的值作有符号相乘，然后将计算结果的低 32 位传送到寄存器<RdLo>中，将计算结果的高 32 位保存到寄存器<RdHi>中。

SMULL 指令的例子如下：

```
SMULL R1, R2, R3, R4;
```

（7）SMLAL 指令

```
SMLAL{<cond>} {S} <RdLo>, <RdHi>, <Rm>, <Rs>
```

SMLAL 是 64 位有符号乘加指令，该指令先将寄存器<Rm>中的值与寄存器<Rs>中的值相乘，然后再与<RdHi>和<RdLo>中的值相加，将计算结果的高 32 位传送到寄存器<RdHi>中，将计算结果的低 32 位保持到寄存器<RdLo>中。

SMLAL 指令的例子如下：

```
SMLAL R1, R2, R3, R4;
```

1.6.3　分支指令

分支指令用于实现程序的跳转。在 ARM 处理器中一共有两种方式实现程序的跳转，一种是使用分支指令，另一种则是直接向 PC 寄存器中赋值。

分支和带链接的分支指令如下。

```
B{L}{cond}<expression>
```

分支指令 B 和带链接的分支指令 BL 都用来完成程序的跳转。其中 B 指令只是单纯完成跳转操作，而 BL 指令先将下一条指令的地址复制到 LR 中，然后完成跳转操作。

1.6.4　程序状态寄存器（PSR）传输指令

（1）MRS 指令

```
MRS{cond} Rd, <psr>
```

MRS 指令将状态寄存器中的内容传送到寄存器 Rd 中。

MRS 指令的例子如下：

```
MRS R1, CPSR;
```

（2）MSR 指令

```
MSR{cond} <psr>, Rm
```

MRS 指令将寄存器 Rm 中的内容传送到状态寄存器中。

```
MSR{cond} <psrf>, Rm
```

MSR 指令将寄存器 Rm 中的内容传送到 PSR 的状态标志位。

```
MSR{cond} <psrf>,<#expression>
```

MSR 指令将立即数传送到 PSR 的状态标志位。
MSR 指令的例子如下：

```
MSR CPSR, R0;
```

1.6.5 Load/Store 指令

（1）LDR 指令

```
LDR{cond} Rd, <Address>
```

LDR 指令将<Address>所在内存地址中的字读取到目标寄存器 Rd 中。
LDR 指令的例子如下：

```
LDR R1, [R2];
```

（2）STR 指令

```
STR{cond} Rd, <Address>
```

STR 指令将寄存器 Rd 中的 32 位的字数据保存到内存地址为<Address>的内存中。
STR 指令的例子如下：

```
STR R1, [R2,#12];
```

（3）LDM 指令

```
LDM{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}
```

LDM 指令可以将从连续内存中读取到寄存器列表中的寄存器中。
LDM 指令的例子如下：

```
LDMIA R1!,{R3-R9};
```

（4）STM 指令

```
STM{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}
```

STM 指令可以将寄存器列表中的寄存器中的值写入到连续的内存中。
STM 指令的例子如下：

```
STMIA R0!, {R3-R9};
```

1.6.6 协处理器指令

（1）CDP 指令

```
CDP{cond} p#,<expression1>, cd, cn, cm {,<expression2>}
```

CDP 指令用于通知 ARM 协处理器完成特定的操作。其中 p#为协处理器独有的标识号，cd、cn、cm 均为协处理器。cd 为目标寄存器，cn 和 cm 为存放操作数的协处理器。<expression1>与<expression2>