基于ARM的嵌入式系统设计与实现

Design and Implementation of ARM Based Embedded Systems

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

1.ARM处理器介绍（1）

ARM(Advanced RISC Machines)公司是全球领先的16／32位RISC微处理器知识产权设计供应商。ARM公司通过转让RISC微处理器，外围和系统芯片设计技术给合作伙伴，使他们能用这些技术来生产各具特色的芯片。

Copyright © 2003-2007 SinoSys. All Rights Reserved

1.ARM处理器介绍（2）

ARM已成为移动通信、手持设备、多媒体数字消费嵌入式解决方案的RISC标准。 ARM处理器有三大特点：

◆体积小、功耗低、低成本而高性能；

◆16／32位双指令集；

◆全球众多的合作伙伴。

Copyright © 2003-2007 SinoSys. All Rights Reserved

1.ARM处理器介绍（3）

ARM处理器目前有5个系列产品：ARM7、ARM9、ARM9E、ARMI0和SecurCore。其中ARM7是低功耗的32位核，最适合应用于对价位和功耗敏感的产品，它又分为应用于实时环境的ARM7TDMI、ARM7TDMI-S，适用于开放平台的ARM720T，以及适用于DSP运算及支持Java的ARM7EJ等。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2. ARM处理器编程

- 2.1 ARM 处理器概述
- 2.2 ARM 处理器结构
- 2.3 ARM 处理器的编程模型
- 2.4 ARM 处理器的指令系统

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(1)

- ARM （Advanced RISC Machines）公司成立于1990年11月，其前身为 Acorn计算机公司，该公司主要设计ARM系列RISC处理器内核
- 授权ARM内核给生产和销售半导体的合作伙伴

  ARM 公司不生产芯片

  IP(Intelligence Property)
- 另外也提供基于ARM架构的开发设计技术

  软件工具, 评估板, 调试工具,应用软件,

  总线架构, 外围设备单元，等等

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(2)

一、ARM微处理器的特点

- 采用RISC架构的ARM微处理器一般具有如下特点
- 1、体积小、低功耗、低成本、高性能；
- 2、支持Thumb（16位）/ARM（32位）双指令集，能很好的兼容8位/16位器件；
- 3、大量使用寄存器，指令执行速度更快；
- 4、大多数数据操作都在寄存器中完成；
- 5、寻址方式灵活简单，执行效率高；
- 6、指令长度固定；

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(3)

二、ARM微处理器系列

- ARM微处理器目前包括下面几个系列。

1. ARM7系列
2. ARM9系列
3. ARM9E系列
4. ARM10E系列
5. SecurCore系列
6. Intel的Xscale
7. Intel的StrongARM

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

2.1 ARM处理器概述(4)

1. ARM7微处理器系列

- ARM7微处理器系列具有如下特点：
- — 具有嵌入式ICE－RT逻辑，调试开发方便。
- — 极低的功耗，适合对功耗要求较高的应用，如便携式产品。
- — 能够提供0.9MIPS/MHz的三级流水线结构。
- — 代码密度高并兼容16位的Thumb指令集。
- — 对操作系统的支持广泛，包括Windows CE、Linux、Palm OS等。
- — 指令系统与ARM9系列、ARM9E系列和ARM10E系列兼容，便于用户的产品升级换代。
- — 主频最高可达130MIPS，高速的运算处理能力能胜任绝大多数的复杂应用。
- 主要应用领域为：工业控制、Internet设备、网络和调制解调器设备、移动电话等多种多媒体和嵌入式应用。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(5)

- ARM7系列微处理器包括如下几种类型的核：ARM7TDMI、ARM7TDMI-S、ARM720T、ARM7EJ。其中，ARM7TMDI是目前使用最广泛的32位嵌入式RISC处理器，属低端ARM处理器核。TDMI的基本含义为：
- T： 支持16为压缩指令集Thumb；
- D： 支持片上Debug；
- M：内嵌硬件乘法器（Multiplier）
- I： 嵌入式ICE，支持片上断点和调试点；

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(6)

2. ARM9微处理器系列

- 具有以下特点：
- 一 5级整数流水线，指令执行效率更高。
- 一 提供1.1MIPS/MHz的哈佛结构。
- 一 支持32位ARM指令集和16位Thumb指令集。
- 一 支持32位的高速AMBA总线接口。
- 一 全性能的MMU，支持Windows CE、Linux、Palm OS等多种主流嵌入式操作系统。
- 一 MPU支持实时操作系统。
- 一 支持数据Cache和指令Cache，具有更高的指令和数据处理能力。
- 主要应用于无线设备、仪器仪表、安全系统、机顶盒、高端打印机、数字照相机和数字摄像机等。
- ARM9系列微处理器包含ARM920T、ARM922T和ARM940T。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(7)

3. ARM9E微处理器系列

- ARM9E系列微处理器的主要特点如下：
- － 支持DSP指令集，适合于需要高速数字信号处理的场合。
- － 5级整数流水线，指令执行效率更高。
- － 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。
- － 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。
- － 支持数据Cache和指令Cache，具有更高的指令和数据处理能力。
- － 主频最高可达300MIPS。
- 主要应用于下一代无线设备、数字消费品、成像设备、工业控制、存储设备和网络设备等领域。
- ARM9E系列微处理器包含ARM926EJ-S、ARM946E-S和ARM966E-S

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(8)

4.ARM10E微处理器系列

- ARM10E系列微处理器的主要特点如下：
- － 支持DSP指令集，适合于需要高速数字信号处理的场合。
- － 6级整数流水线，指令执行效率更高。
- － 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。
- － 全性能的MMU，支持Windows CE、Linux、Palm OS等多种主流嵌入式操作系统。
- － 支持数据Cache和指令Cache，具有更高的指令和数据处理能力
- － 主频最高可达400MIPS。
- － 内嵌并行读/写操作部件。
- 主要应用于下一代无线设备、数字消费品、成像设备、工业控制、通信和信息系统等领域。
- ARM10E系列微处理器包含ARM1020E、ARM1022E和ARM1026EJ-S。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(9)

5. SecurCore微处理器系列

- 除具有ARM体系结构主要特点外，还在系统安全方面具如下特点：
- － 带有灵活的保护单元，以确保操作系统和应用数据的安全。
- － 采用软内核技术，防止外部对其进行扫描探测。
- － 可集成用户自己的安全特性和其他协处理器。
- 主要应用于一些对安全性要求较高的应用产品及应用系统，如电子商务、电子政务、电子银行业务、网络和认证系统等领域。
- SecurCore系列微处理器包含SecurCore SC100、SecurCore SC110、SecurCore SC200和SecurCore SC210四种类型。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(10)

- **6. StrongARM微处理器系列**
- Intel StrongARM SA-1100处理器是采用ARM体系结构高度集成的32位RISC微处理器。它融合了Intel公司的设计和处理技术以及ARM体系结构的电源效率，采用在软件上兼容ARMv4体系结构、同时采用具有Intel技术优点的体系结构。
- Intel StrongARM处理器是便携式通讯产品和消费类电子产品的理想选择，已成功应用于多家公司的掌上电脑系列产品。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.1 ARM处理器概述(11)

- **7. Xscale处理器**
- Xscale 处理器是基于ARMv5TE体系结构的解决方案，是一款全性能、高性价比、低功耗的处理器。它支持16位的Thumb指令和DSP指令集，已使用在数字移动电话、个人数字助理和网络产品等场合。
- Xscale 处理器是Intel目前主要推广的一款ARM微处理器。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.2 ARM处理器结构(1)

- **一、 RISC体系结构**
- RISC —Reduced Instruction Set Computer，精简指令集计算机RISC体系结构应具有如下特点：
- – 采用固定长度的指令格式，指令归整、简单、基本寻址方式有2～3种。
- – 使用单周期指令，便于流水线操作执行。
- – 大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/ 存储指令可以访问存储器，以提高指令的执行效率。
- 除此以外，ARM体系结构还采用了一些特别的技术，在保证高性能的前提下尽量缩小芯片的面积，并降低功耗：
- – 所有的指令都可根据前面的执行结果决定是否被执行，从而提高指令的执行效率。
- – 可用加载/存储指令批量传输数据，以提高数据的传输效率。
- – 可在一条数据处理指令中同时完成逻辑处理和移位处理。
- – 在循环处理中使用地址的自动增减来提高运行效率。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.2 ARM处理器结构(2)

二、 ARM处理器的寄存器结构

- ARM处理器共有37个寄存器，这些寄存器包括：
- － 31个通用寄存器，包括程序计数器（PC指针），均为32位的寄存器。
- － 6个状态寄存器，用以标识CPU的工作状态及程序的运行状态，均为32位，目前只使用了其中的一部分。
- 同时，ARM处理器又有7种不同的处理器模式，在每一种处理器模式下均有一组相应的寄存器与之对应。即在任意一种处理器模式下，可访问的寄存器包括15个通用寄存器（R0～R14）、一至二个状态寄存器和程序计数器。在所有的寄存器中，有些是在7种处理器模式下共用的同一个物理寄存器，而有些寄存器则是在不同的处理器模式下有不同的物理寄存器。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.2 ARM处理器结构(3)

- 三、 **ARM处理器的指令结构**
- ARM微处理器的在较新的体系结构中支持两种指令集：ARM指令集和Thumb指令集。
- 其中，ARM指令为32位的长度，Thumb指令为16位长度。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.3 ARM 处理器的编程模型

- 本节的主要内容：

一　ARM微处理器的工作状态

二　ARM体系结构的存储器格式

三　指令长度及数据类型

四　ARM微处理器的工作模式

五　ARM体系结构的寄存器组织

七　ARM微处理器的异常处理

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.3 ARM 处理器的编程模型

- 首先对字（Word）、半字（Half-Word）、字节（Byte）的概念作一个说明：
- 字（Word）：在ARM体系结构中，字的长度为32位，而在8位/16位处理器体系结构中，字的长度一般为16位，请读者在阅读时注意区分。
- 半字（Half-Word）：在ARM体系结构中，半字的长度为16位，与8位/16位处理器体系结构中字的长度一致。
- 字节（Byte）：在ARM体系结构和8位/16位处理器体系结构中，字节的长度均为8位。

Copyright © 2003-2007 SinoSys. All Rights Reserved

一 ARM微处理器的工作状态(1)

- ARM微处理器的工作状态一般有两种，并可在两种状态之间切换：
  - 第一种为ARM状态，此时处理器执行32位的字对齐的ARM指令；
  - 第二种为Thumb状态，此时处理器执行16位的、半字对齐的Thumb指令。
- 处理器工作状态的转变并不影响处理器的工作模式和相应寄存器中的内容。

Copyright © 2003-2007 SinoSys. All Rights Reserved

一 ARM微处理器的工作状态(2)

- 状态切换方法：
- ARM微处理器在开始执行代码时，应该处于ARM状态。
- 进入Thumb状态：当操作数寄存器的状态位（位0）为1时，可采用执行BX指令的方法，使微处理器从ARM状态切换到Thumb状态。此外，当处理器处于Thumb状态时发生异常（如IRQ、FIQ、Undef、Abort等），则异常处理返回时，自动切换到Thumb状态。
- 进入ARM状态：当操作数寄存器的状态位为0时，执行BX指令时可使微处理器从Thumb状态切换到ARM状态。此外，在处理器进行异常处理时，把PC指针放入异常模式链接寄存器中，并从异常向量地址开始执行程序，也可以使处理器切换到ARM状态。

Copyright © 2003-2007 SinoSys. All Rights Reserved

二 ARM体系结构的存储器格式(1)

- ARM体系结构将存储器看作是从零地址开始的字节的线性组合。从零字节到三字节放置第一个存储的字数据，从第四个字节到第七个字节放置第二个存储的字数据，依次排列。作为32位的微处理器，ARM体系结构所支持的最大寻址空间为4GB（2^{32}字节）。
- ARM体系结构可以用两种方法存储字数据，称之为大端格式和小端格式。

Copyright © 2003-2007 SinoSys. All Rights Reserved

二 ARM体系结构的存储器格式(2)

- 大端格式：
- 在这种格式中，字数据的高字节存储在低地址中，而字数据的低字节则存放在高地址中，如图

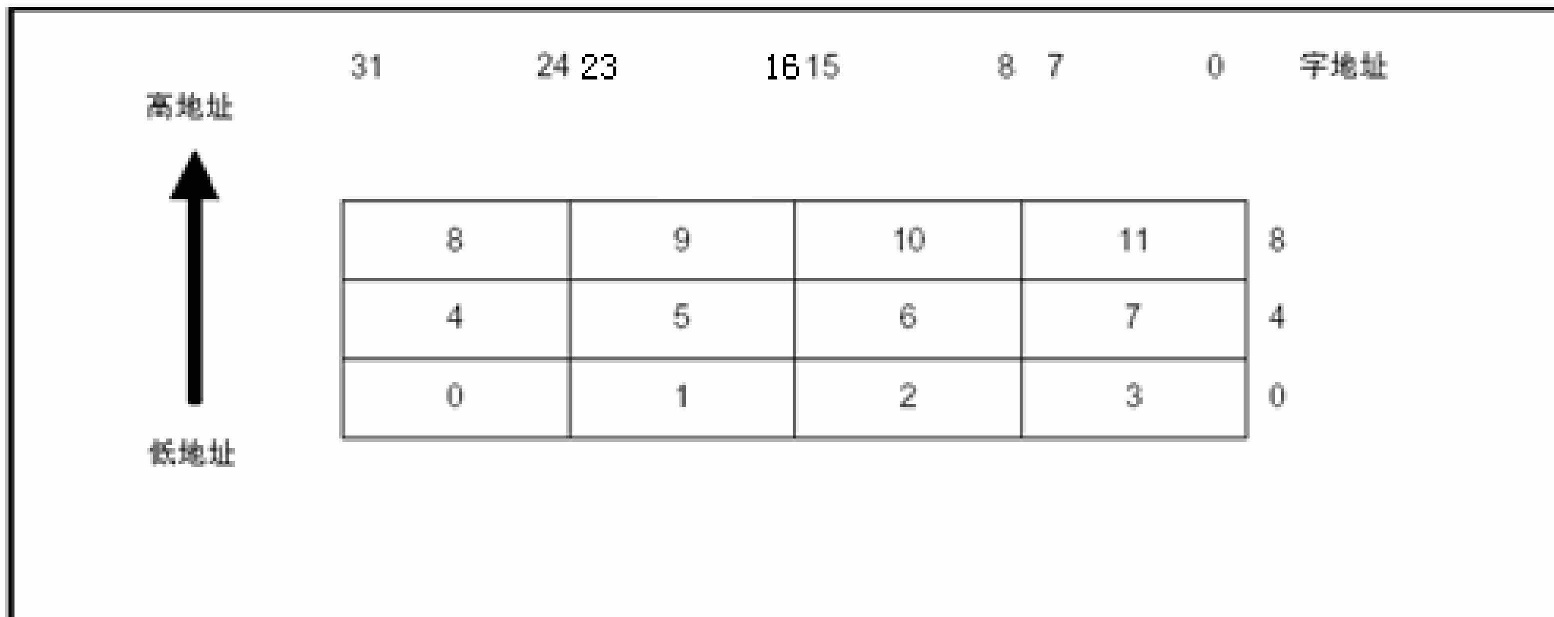


图 以大端格式存储字数据

Copyright © 2003-2007 SinoSys. All Rights Reserved
双实科技 SinoSys

二 ARM体系结构的存储器格式(3)

- 小端格式：
- 与大端存储格式相反，在小端存储格式中，低地址中存放的是字数据的低字节，高地址存放的是字数据的高字节。如图

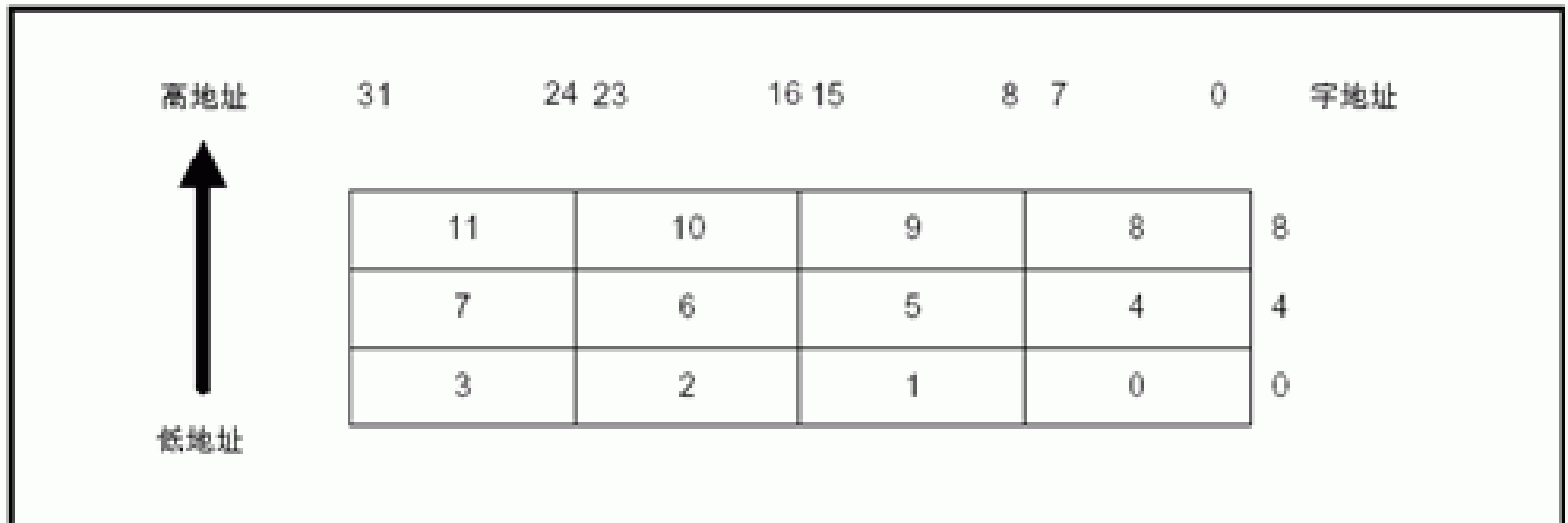


图　以小端格式存储字数据

Copyright © 2003-2007 SinoSys. All Rights Reserved

三 指令长度及数据类型

- ARM 采用的是32位架构。
- ARM 约定：Byte ： 8 bits
  Halfword ：16 bits (2 byte)
  Word : 32 bits (4 byte)
- 大部分ARM core 提供：
  ARM 指令集（32-bit）
  Thumb 指令集（16-bit ）

Copyright © 2003-2007 SinoSys. All Rights Reserved

四 ARM处理器工作模式(1)

ARM 有7个基本工作模式:

User:非特权模式，大部分任务执行在这种模式
- 正常程序执行的模式

FIQ:当一个高优先级(fast)中断产生时将会进入这种模式
- 高速数据传输和通道处理

IRQ:当一个低优先级(normal)中断产生时将会进入这种模式
- 通常的中断处理

Copyright © 2003-2007 SinoSys. All Rights Reserved

四 ARM处理器工作模式(2)

Supervisor:当复位或软中断指令执行时将会进入这种模式
- 供操作系统使用的一种保护模式

Abort: 当存取异常时将会进入这种模式
- 虚拟存储及存储保护

Undef: 当执行未定义指令时会进入这种模式
- 软件仿真硬件协处理器

System: 使用和User模式相同寄存器集的特权模式
- 特权级的操作系统任务

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(1)

ARM微处理器共有37个32位寄存器，其中31个为通用寄存器，6个为状态寄存器。但是这些寄存器不能被同时访问，具体哪些寄存器是可编程访问的，取决微处理器的工作状态及具体的运行模式。但在任何时候，通用寄存器R14～R0、程序计数器PC、一个或两个状态寄存器都是可访问的。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(2)

1. ARM状态下的寄存器组织

通用寄存器：

- 通用寄存器包括R0～R15，可以分为三类：
- — 未分组寄存器R0～R7；
- — 分组寄存器R8～R14
- — 程序计数器PC(R15)

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(3)

- 未分组寄存器R0～R7：
- 在所有的运行模式下，未分组寄存器都指向同一个物理寄存器，他们未被系统用作特殊的用途。（在进行程序设计时应注意）
- 分组寄存器R8～R14
- 对于分组寄存器，他们每一次所访问的物理寄存器与处理器当前的运行模式有关。
- 对于R8～R12来说，每个寄存器对应两个不同的物理寄存器，当使用fiq模式时，访问寄存器R8_fiq～R12_fiq；当使用除fiq模式以外的其他模式时，访问寄存器R8_usr～R12_usr。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(4)

- 对于R13、R14来说，每个寄存器对应6个不同的物理寄存器，其中的一个是用户模式与系统模式共用，另外5个物理寄存器对应于其他5种不同的运行模式。
- 采用以下的记号来区分不同的物理寄存器：
- R13_<mode>
- R14_<mode>
- 其中，mode为以下几种模式之一：usr、fiq、irq、svc、abt、und。
- 寄存器R13在ARM指令中常用作堆栈指针。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(5)

- 注意：由于处理器的每种运行模式均有自己独立的物理寄存器R13，在用户应用程序的初始化部分，一般都要初始化每种模式下的R13，使其指向该运行模式的栈空间，这样，当程序的运行进入异常模式时，可以将需要保护的寄存器放入R13所指向的堆栈，而当程序从异常模式返回时，则从对应的堆栈中恢复，采用这种方式可以保证异常发生后程序的正常执行。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(6)

- R14也称作子程序连接寄存器或连接寄存器LR。
- 当执行BL子程序调用指令时，R14中得到PC的备份。其他情况下，R14用作通用寄存器。与之类似，当发生中断或异常时，对应的分组寄存器R14_svc、R14_irq、R14_fiq、R14_abt和R14_und用来保存R15的返回值。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(7)

- 寄存器R14常用在如下的情况：
- 在每一种运行模式下，都可用R14保存子程序的返回地址，当用BL或BLX指令调用子程序时，将PC的当前值拷贝给R14，执行完子程序后，又将R14的值拷贝回PC，即可完成子程序的调用返回。以上的描述可用指令完成：
- 1、执行以下任意一条指令：

  MOV PC，LR

  BX LR
- 2、在子程序入口处使用以下指令将R14存入堆栈：

  STMFD SP！,{<Regs>,LR}
- 对应的，使用以下指令可以完成子程序返回：

  LDMFD SP！,{<Regs>,PC}

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(6)

- 程序计数器PC(R15)
- 寄存器R15用作程序计数器(PC)。在ARM状态下，位[1:0]为0，位[31:2]用于保存PC；在Thumb状态下，位[0]为0，位[31:1]用于保存PC（在ARM状态下，PC的0和1位是0，在Thumb状态下，PC的0位是0）。
- ARM体系结构采用了多级流水线技术，对于ARM指令集而言，PC总是指向当前指令的下两条指令的地址，即PC的值为当前指令的地址值加8个字节。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(9)

- 此图说明在每一种运行模式下，哪一些寄存器是可以访问的。

图 ARM 状态下的寄存器组织

Copyright © 2003-2007 Sin

五 ARM体系结构的寄存器组织(10)

- 寄存器R16：
- 寄存器R16用作CPSR(当前程序状态寄存器)，CPSR可在任何运行模式下被访问，它包括条件标志位、中断禁止位、当前处理器模式标志位，以及其他一些相关的控制和状态位。
- 每一种运行模式下又都有一个专用的物理状态寄存器，称为SPSR（备份的程序状态寄存器），当异常发生时，SPSR用于保存CPSR的当前值，从异常退出时则可由SPSR来恢复CPSR。
- 由于用户模式和系统模式不属于异常模式，他们没有SPSR，当在这两种模式下访问SPSR，结果是未知的

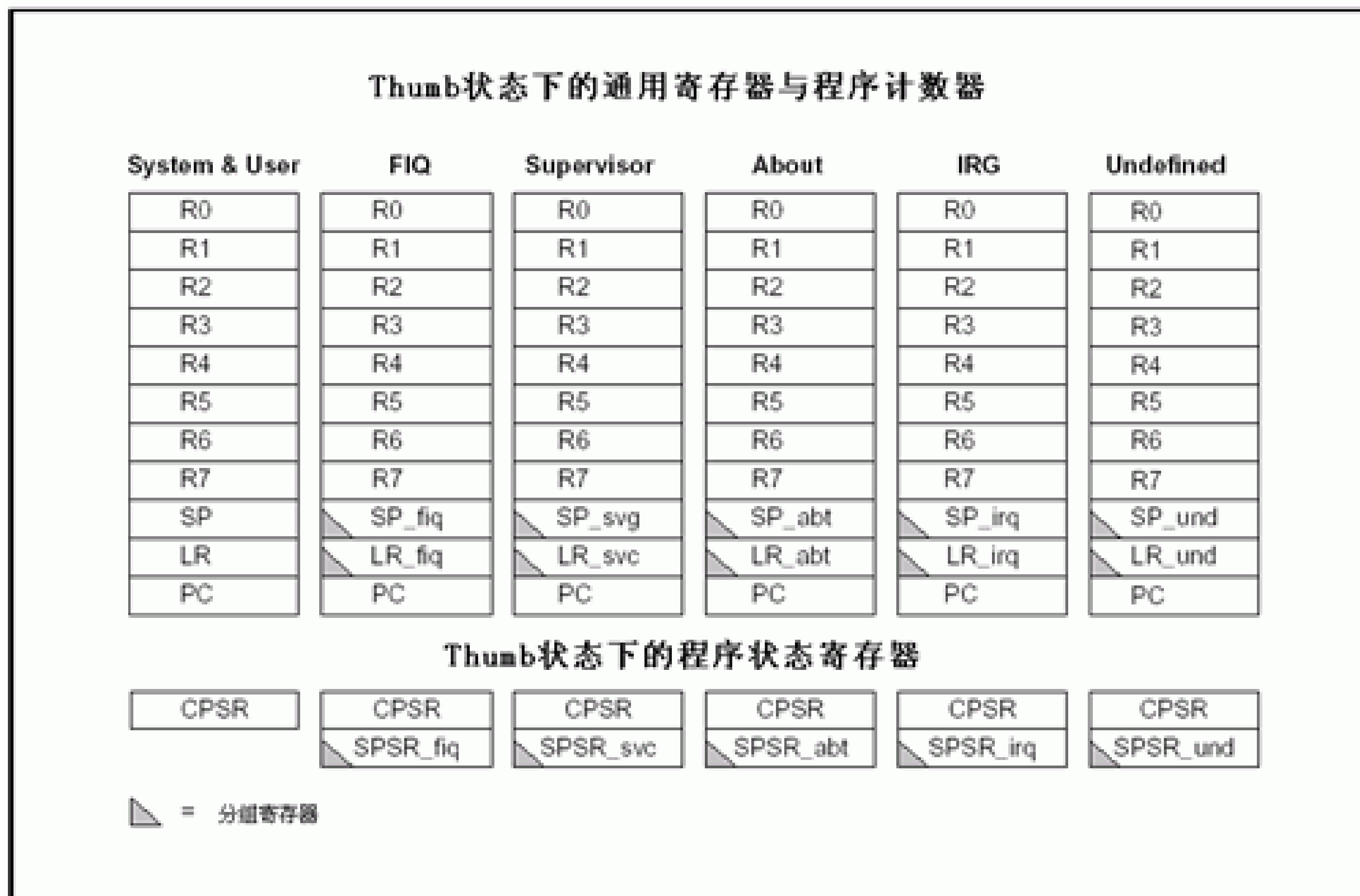
Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(11)

- **2. Thumb状态下的寄存器组织**
- Thumb状态下的寄存器集是ARM状态下寄存器集的一个子集，程序可以直接访问8个通用寄存器（R7～R0）、程序计数器（PC）、堆栈指针（SP）、连接寄存器（LR）和CPSR。同时，在每一种特权模式下都有一组SP、LR和SPSR。下图表明Thumb状态下的寄存器组织。

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(12)



Thumb状态下的通用寄存器与程序计数器

System & User	FIQ	Supervisor	About	IRG	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	SP_fiq	SP_svg	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

Thumb状态下的程序状态寄存器

System & User	FIQ	Supervisor	About	IRG	Undefined
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

◺ = 分组寄存器

图 Thumb 状态下的寄存器组织

Copyrigh

SinoSys

五 ARM体系结构的寄存器组织(13)

- Thumb状态下的寄存器组织与ARM状态下的寄存器组织的关系：
- — Thumb状态下和ARM状态下的R0～R7是相同的。
- — Thumb状态下和ARM状态下的CPSR和所有的SPSR是相同的。
- — Thumb状态下的SP对应于ARM状态下的R13。
- — Thumb状态下的LR对应于ARM状态下的R14。
- — Thumb状态下的程序计数器对应于ARM状态下R15
- 以上的对应关系如图所示：

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(14)

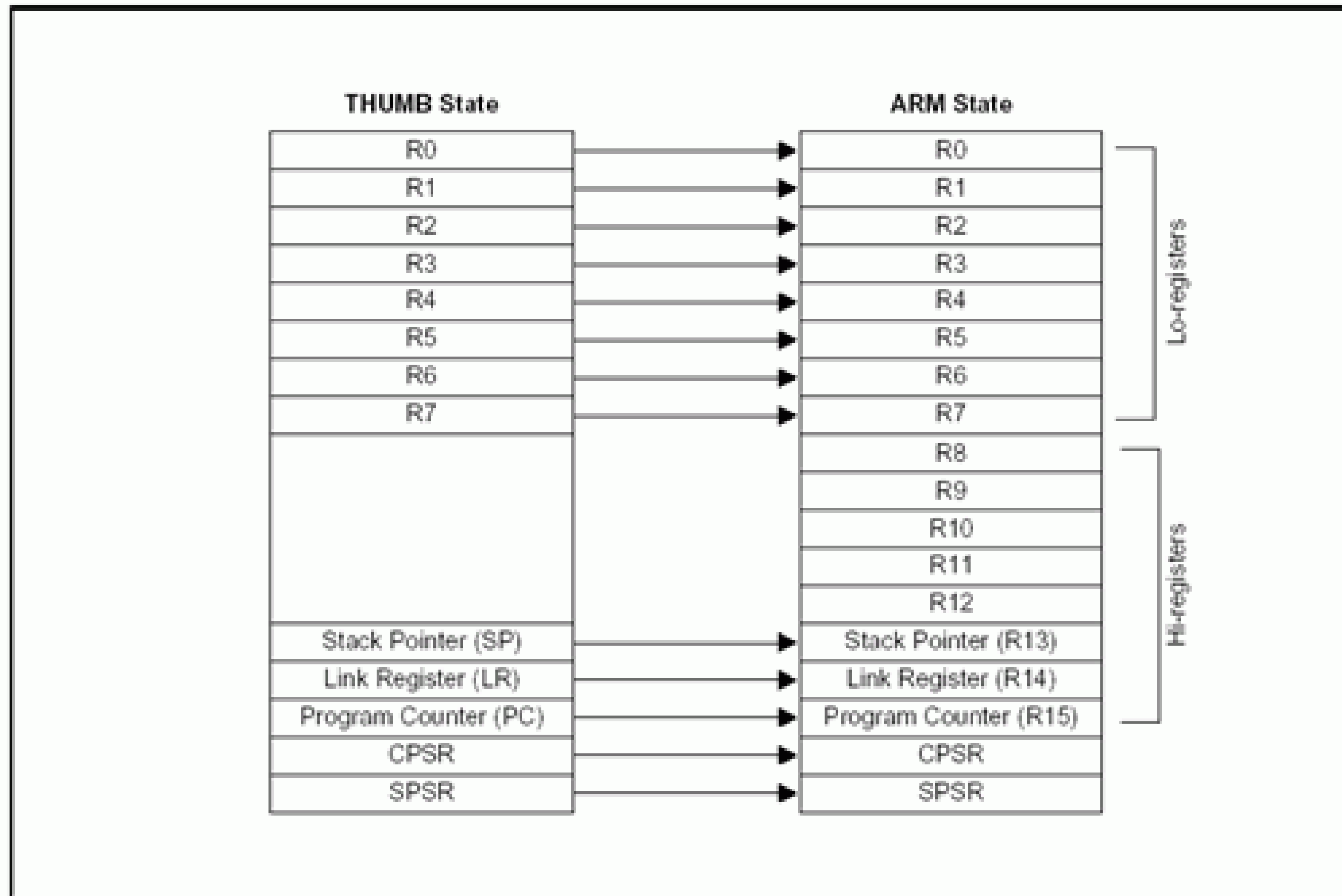


图 Thumb 状态下的寄存器组织

Copyright © 2003-2007 SinoSys. All Rights Reserved

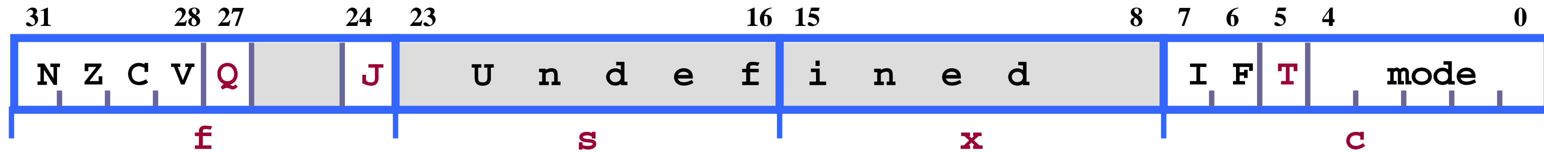
双实科技 SinoSys

五 ARM体系结构的寄存器组织(15)

- 3.程序状态寄存器
- ARM体系结构包含一个当前程序状态寄存器（CPSR）和五个备份的程序状态寄存器（SPSRs）。备份的程序状态寄存器用来进行异常处理，其功能包括：
- — 保存ALU中的当前操作信息
- — 控制允许和禁止中断
- — 设置处理器的运行模式
- 程序状态寄存器的每一位的安排如图所示：

Copyright © 2003-2007 SinoSys. All Rights Reserved

五 ARM体系结构的寄存器组织(16)



- 条件位：
  N = 1-结果为负,0-结果为正或0
  Z = 1-结果为0,0-结果不为0
  C =1-进位，0-借位
  V =1-结果溢出，0结果没溢出
- Q 位：
  仅ARM 5TE/J架构支持
  指示增强型DSP指令是否溢出
- J 位
  仅ARM 5TE/J架构支持
  J = 1: 处理器处于Jazelle状态
- 中断禁止位：
  I = 1: 禁止 IRQ.
  F = 1: 禁止 FIQ.
- T Bit
  仅ARM xT架构支持
  T = 0: 处理器处于 ARM 状态
  T = 1: 处理器处于 Thumb 状态
- Mode位(处理器模式位):

0b10000	User
0b10001	FIQ
0b10010	IRQ
0b10011	Supervisor
0b10111	Abort
0b11011	Undefined
0b11111	System

Copyright © 2003-2007 SinoSys. All Rights Reserved

六 ARM微处理器的异常处理(1)

- 当正常的程序执行流程发生暂时的停止时，称之为异常，例如处理一个外部的中断请求。
- ARM体系结构中的异常，与8位/16位体系结构的中断有很大的相似之处，但异常与中断的概念并不完全等同。
- ARM体系结构所支持的异常及具体含义如下表所示。

Copyright © 2003-2007 SinoSys. All Rights Reserved

六 ARM微处理器的异常处理(2)

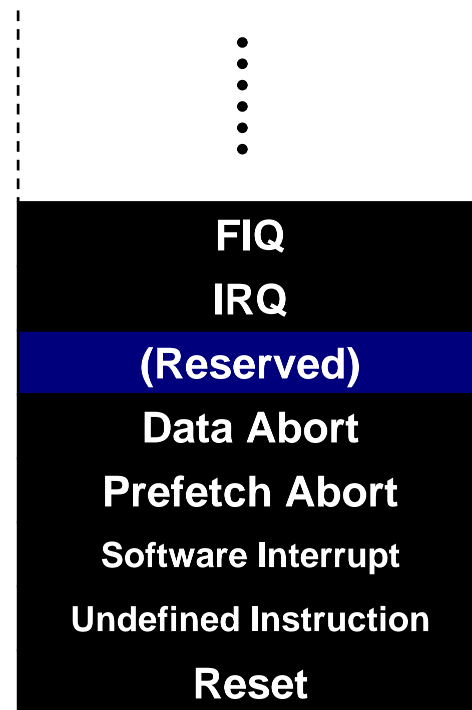
异常类型	具体含义
复位	当处理器的复位电平有效时，产生复位异常，程序跳转到复位异常处理程序处执行。
未定义指令	当ARM处理器或协处理器遇到不能处理的指令时，产生未定义指令异常。可使用该异常机制进行软件仿真。
软件中断	该异常由执行SWI指令产生，可用于用户模式下的程序调用特权操作指令。可使用该异常机制实现系统功能调用。
指令预取中止	若处理器预取指令的地址不存在，或该地址不允许当前指令访问，存储器会向处理器发出中止信号，但当预取的指令被执行时，才会产生指令预取中止异常。
数据中止	若处理器数据访问指令的地址不存在，或该地址不允许当前指令访问时，产生数据中止异常。
IRQ（外部中断请求）	当处理器的外部中断请求引脚有效，且CPSR中的I位为0时，产生IRQ异常。系统的外设可通过该异常请求中断服务。
FIQ（快速中断请求）	当处理器的快速中断请求引脚有效，且CPSR中的F位为0时，产生FIQ异常。

Copyright © 2003-2007 SinoSys. All Rights Reserved

六 ARM微处理器的异常处理(3)

- 当异常产生时, ARM core:
  拷贝 CPSR 到 SPSR_<mode>
  设置适当的 CPSR 位：
  - 改变处理器状态进入 ARM 态
  - 改变处理器模式进入相应的异常模式
  - 设置中断禁止位禁止相应中断 (如果需要)

  保存返回地址到 LR_<mode>
  设置 PC 为相应的异常向量
- 返回时, 异常处理需要:
  从 SPSR_<mode>恢复CPSR
  从LR_<mode>恢复PC
  Note:这些操作只能在 ARM 态执行.



Vector Table

Vector table can be at
`0xFFFF0000` on ARM720T
and on ARM9/10 family devices

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

六 ARM微处理器的异常处理(4)

- 异常向量（Exception Vectors）
- 异常向量表

地 址	异 常	进入模式
0x0000,0000	复位	管理模式
0x0000,0004	未定义指令	未定义模式
0x0000,0008	软件中断	管理模式
0x0000,000C	中止（预取指令）	中止模式
0x0000,0010	中止（数据）	中止模式
0x0000,0014	保留	保留
0x0000,0018	IRQ	IRQ
0x0000,001C	FIQ	FIQ

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

六 ARM微处理器的异常处理(5)

- 异常优先级（Exception Priorities）

优先级	异　常
1（最高）	复位
2	数据中止
3	FIQ
4	IRQ
5	预取指令中止
6（最低）	未定义指令、SWI

Copyright © 2003-2007 SinoSys. All Rights Reserved

六 ARM微处理器的异常处理(6)

- 应用程序中的异常处理
- 当系统运行时，异常可能会随时发生，为保证在ARM处理器发生异常时不至于处于未知状态，在应用程序的设计中，首先要进行异常处理，采用的方式是在异常向量表中的特定位置放置一条跳转指令，跳转到异常处理程序，当ARM处理器发生异常时，程序计数器PC会被强制设置为对应的异常向量，从而跳转到异常处理程序，当异常处理完成以后，返回到主程序继续执行。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(1)

一. ARM微处理器的指令集概述

1. ARM微处理器的指令的分类与格式

- ARM微处理器的指令集是加载/存储型的，也即指令集仅能处理寄存器中的数据，而且处理结果都要放回寄存器中，而对系统存储器的访问则需要通过专门的加载/存储指令来完成。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(2)

•ARM微处理器的指令集可以分为以下六大类：

1. 跳转指令
2. 数据处理指令
3. 程序状态寄存器（PSR）处理指令
4. 加载/存储指令
5. 协处理器指令
6. 异常产生指令

具体的指令及功能如下表所示

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(3)

助记符	指令功能描述	助记符	指令功能描述
ADC	带进位加法指令	MRC	从协处理器寄存器到ARM寄存器的数据传输指令
ADD	加法指令	MRS	传送CPSR或SPSR的内容到通用寄存器指令
AND	逻辑与指令	MSR	传送通用寄存器到CPSR或SPSR的指令
B	跳转指令	MUL	32位乘法指令
BIC	位清零指令	MLA	32位乘加指令
BL	带返回的跳转指令	MVN	数据取反传送指令
BLX	带返回和状态切换的跳转指令	ORR	逻辑或指令
BX	带状态切换的跳转指令	RSB	逆向减法指令
CDP	协处理器数据操作指令	RSC	带借位的逆向减法指令
CMN	比较反值指令	SBC	带借位减法指令
CMP	比较指令	STC	协处理器寄存器写入存储器指令
EOR	异或指令	STM	批量内存字写入指令
LDC	存储器到协处理器的数据传输指令	STR	寄存器到存储器的数据传输指令
LDM	加载多个寄存器指令	SUB	减法指令
LDR	存储器到寄存器的数据传输指令	SWI	软件中断指令
MCR	从ARM寄存器到协处理器寄存器的数据传输指令	SWP	交换指令
MLA	乘加运算指令	TEQ	相等测试指令
MOV	数据传送指令	TST	位测试指令

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(4)

2. 指令的条件域

- 当处理器工作在ARM状态时，几乎所有的指令均根据CPSR中条件码的状态和指令的条件域有条件的执行。当指令的执行条件满足时，指令被执行，否则指令被忽略。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(5)

- 每一条ARM指令包含4位的条件码，位于指令的最高4位[31:28]。条件码共有16种，每种条件码可用两个字符表示，这两个字符可以添加在指令助记符的后面和指令同时使用。例如，跳转指令B可以加上后缀EQ变为BEQ表示“相等则跳转”，即当CPSR中的Z标志置位时发生跳转。
- 在16种条件标志码中，只有15种可以使用，如表所示，第16种（1111）为系统保留，暂时不能使用。

Copyright © 2003-2007 SinoSys. All Rights Reserved

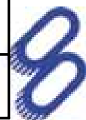
2.4 ARM 处理器的指令系统(6)

- 表 指令的条件码

条件码	助记符后缀	标 志	含 义
0000	EQ	Z置位	相等
0001	NE	Z清零	不相等
0010	CS	C置位	无符号数大于或等于
0011	CC	C清零	无符号数小于
0100	MI	N置位	负数
0101	PL	N清零	正数或零
0110	VS	V置位	溢出
0111	VC	V清零	未溢出
1000	HI	C置位Z清零	无符号数大于
1001	LS	C清零Z置位	无符号数小于或等于
1010	GE	N等于V	带符号数大于或等于
1011	LT	N不等于V	带符号数小于
1100	GT	Z清零且（N等于V）	带符号数大于
1101	LE	Z置位或（N不等于V）	带符号数小于或等于
1110	AL	忽略	无条件执行

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技



2.4 ARM 处理器的指令系统(7)

二. ARM指令的寻址方式

- 寻址方式：所谓寻址方式就是处理器根据指令中给出的地址信息来寻找物理地址的方式。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(8)

- 目前ARM指令系统支持如下几种常见的寻址方式。
- 立即寻址
- 寄存器寻址
- 寄存器间接寻址
- 基址变址寻址
- 多寄存器寻址
- 相对寻址
- 堆栈寻址

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(9)

- 1 立即寻址（立即数寻址）
- 操作数本身就在指令中给出，只要取出指令也就取到了操作数。例如以下指令：
- ADD R0，R0，＃1 ；R0←R0＋1
- ADD R0，R0，＃0x3f ；R0←R0＋0x3f
- 在以上两条指令中，第二个源操作数即为立即数，要求以“＃”为前缀，对于以十六进制表示的立即数，还要求在“＃”后加上“0x”或“&”。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(10)

- **2 寄存器寻址**
- 寄存器寻址就是利用寄存器中的数值作为操作数，这种寻址方式是各类微处理器经常采用的一种方式，也是一种执行效率较高的寻址方式。
- 以下指令：
- ADD R0，R1，R2 ；R0←R1＋R2
- 该指令的执行效果是将寄存器R1和R2的内容相加，其结果存放在寄存器R0中。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(11)

- **3** 寄存器间接寻址
- 寄存器间接寻址就是以寄存器中的值作为操作数的地址，而操作数本身存放在存储器中。
- 例如以下指令：
- ADD R0，R1，[R2] ；R0←R1＋[R2]
- LDR R0，[R1] ；R0←[R1]
- STR R0，[R1] ；[R1]←R0

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(12)

- 4 基址变址寻址
- 基址变址寻址就是将寄存器（基址寄存器）的内容与指令中给出的地址偏移量相加，从而得到一个操作数的有效地址。
- 采用变址寻址方式的指令常见有以下几种形式
- LDR R0，[R1，＃4] ；R0←[R1＋4]
- LDR R0，[R1，＃4]！ ；R0←[R1＋4]、R1←R1＋4
- LDR R0，[R1]，＃4 ；R0←[R1]、R1←R1＋4
- LDR R0，[R1，R2] ；R0←[R1＋R2]

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(13)

- **5 多寄存器寻址**
- 采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送。这种寻址方式可以用一条指令完成传送最多16个通用寄存器的值。以下指令：
- LDMIA R0，{R1，R2，R3，R4} ；R1←[R0]

  ；R2←[R0＋4]

  ；R3←[R0＋8]

  ；R4←[R0＋12]
- 该指令的后缀IA表示在每次执行完加载/存储操作后，R0按字长度增加，因此，指令可将连续存储单元的值传送到R1～R4。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(14)

- 6 相对寻址
- 与基址变址寻址方式相类似，相对寻址以程序计数器PC的当前值为基地址，指令中的地址标号作为偏移量，将两者相加之后得到操作数的有效地址。以下程序段完成子程序的调用和返回，跳转指令BL采用了相对寻址方式：
- BL NEXT ；跳转到子程序NEXT处执行
- ……
- NEXT
- ……
- MOV PC，LR ；从子程序返回

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(15)

- 7 堆栈寻址
- 堆栈是一种数据结构，按先进后出（FILO）的方式工作，使用一个称作堆栈指针的专用寄存器指示当前的操作位置，堆栈指针总是指向栈顶。
- 当堆栈指针指向最后压入堆栈的数据时，称为满堆栈（，而当堆栈指针指向下一个将要放入数据的空位置时，称为空堆栈。
- 同时，根据堆栈的生成方式，又可以分为递增堆栈和递减堆栈，当堆栈由低地址向高地址生成时，称为递增堆栈，当堆栈由高地址向低地址生成时，称为递减堆栈，这样就有四种类型的堆栈工作方式。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(16)

- ARM微处理器支持这四种类型的堆栈工作方式，即：
- － 满递增堆栈：堆栈指针指向最后压入的数据，且由低地址向高地址生成。
- － 满递减堆栈：堆栈指针指向最后压入的数据，且由高地址向低地址生成。
- － 空递增堆栈：堆栈指针指向下一个将要放入数据的空位置，且由低地址向高地址生成。
- － 空递减堆栈：堆栈指针指向下一个将要放入数据的空位置，且由高地址向低地址生成。

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(17)

三、ARM 指令集

数据处理指令：

```
SUB     r0,r1,#5             ;r1-5->r0
ADD     r2,r3,r3,LSL #2 ;R3x4+r3->r2
ANDS    r4,r4,#0x20          ;r4+0x20->r4,更新条
                             件码标志位
ADDEQ   r5,r5,r6            ;r5+r6->r5(条件-相等)
```

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(18)

存储器存取指令：

```
LDR    r0,[r1],#4  ;r1+4->r0
STRNEB r2,[r3,r4]  ;NE符合-将r2低8位
                       数写到[r3+r4]内存单元
LDRSH  r5,[r6,#8]!  ;[r6+8]->r5(半字
          节)，r5中高16位设置成该字节的符号位
STMFD  sp!,{r0,r2-r7,r10}     ；出栈
```

Copyright © 2003-2007 SinoSys. All Rights Reserved

2.4 ARM 处理器的指令系统(19)

ARM 跳转分支指令

- B <label>

  PC ±32 Mbyte .

- BL <子程序>

  保存返回地址到 LR

  返回时从 LR 恢复 PC

  对于 non-leaf 函数, LR 必须压栈保存

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.基于ARM的硬件系统的启动及初始化

3.1 引言

3.2 硬件系统初始化过程

3.3 Boot loader概述

3.4 常用基于ARM9的Boot loader

3.5 小结

3.6 实验

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.1 引言

对比嵌入式系统和PC。PC的引导代码就是BIOS，一般BIOS由主板厂商提供，我们无须也很难修改（事实上，有些BIOS的代码是保密的）。嵌入式系统中的引导代码则是嵌入式开发的难点之一，同时是系统运行的一个基本前提条件，没有这段和硬件紧密相关的代码，多么精悍的内核也不能发挥作用。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2 硬件系统初始化过程

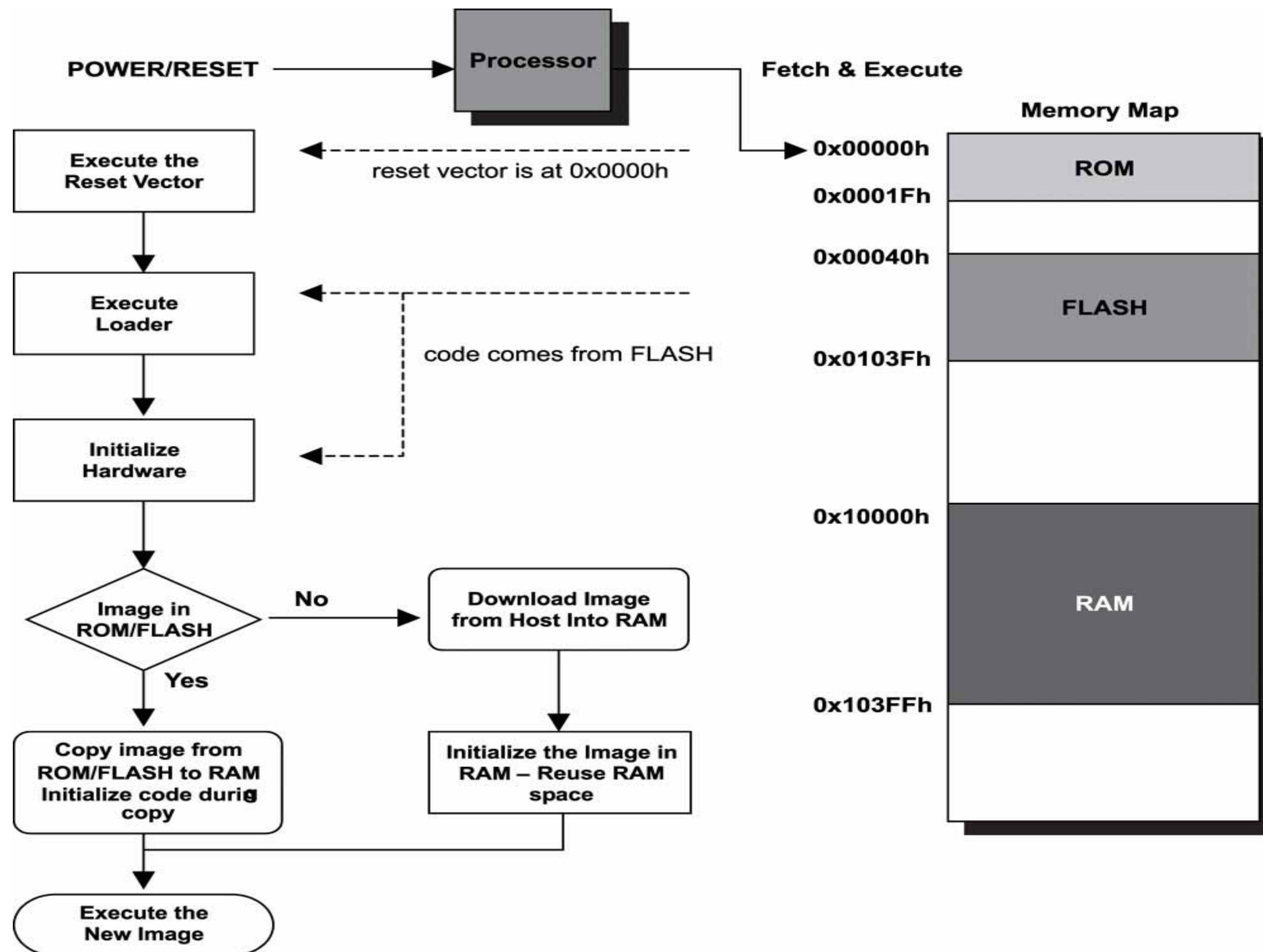
3.2.1 硬件系统启动模式

3.2.2 硬件系统初始化

3.2.3 基于ARM9的嵌入式系统的初始化

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.1 硬件系统启动模式(1)



Copyright © 2003-2007 SinoSys. All Rights Reserved

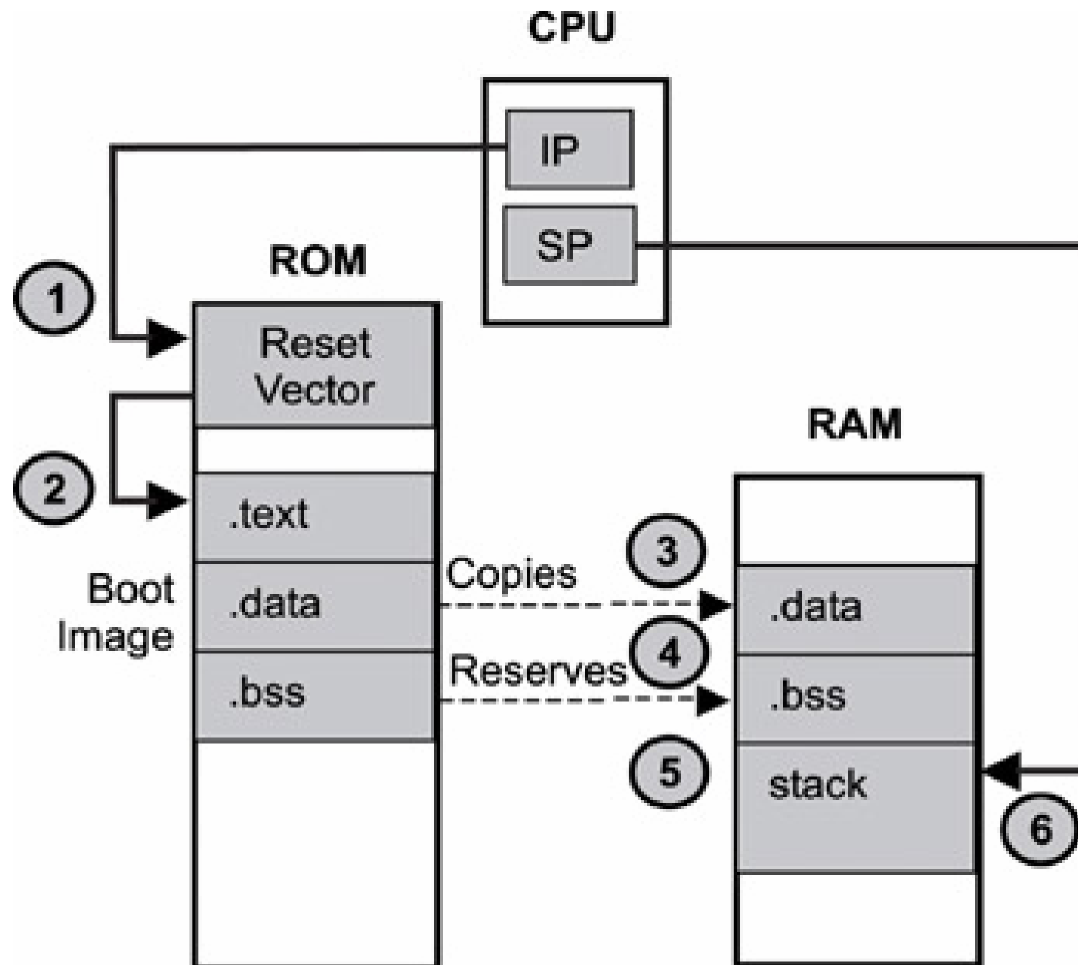
3.2.1 硬件系统启动模式(2)

- 我们描述三种运行模式：
  - 从ROM运行，数据放在RAM。
  - 程序从ROM拷贝到RAM后，从RAM运行。
  - 程序主机端传到RAM后，从RAM运行。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.1 硬件系统启动模式(3)

- 从ROM运行，数据放在RAM



Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.1 硬件系统启动模式(4)

从ROM运行的启动顺序：

1、CPU的IP指针指向内存的第一条指令。（复位向量）

2、复位向量的跳转指令跳到初始化代码影象文件的.text段的第一条指令。这个.text段驻留在ROM里；CPU使用IP指针去运行.text段，初始化存储系统（包括RAM）。

3、代码影像文件的.data段被拷贝到RAM区里，因为他们要求读写操作。

4、RAM分配空间被用做代码影像文件的.bss段。.bss段是空的，所以不存在拷贝传输的事情。

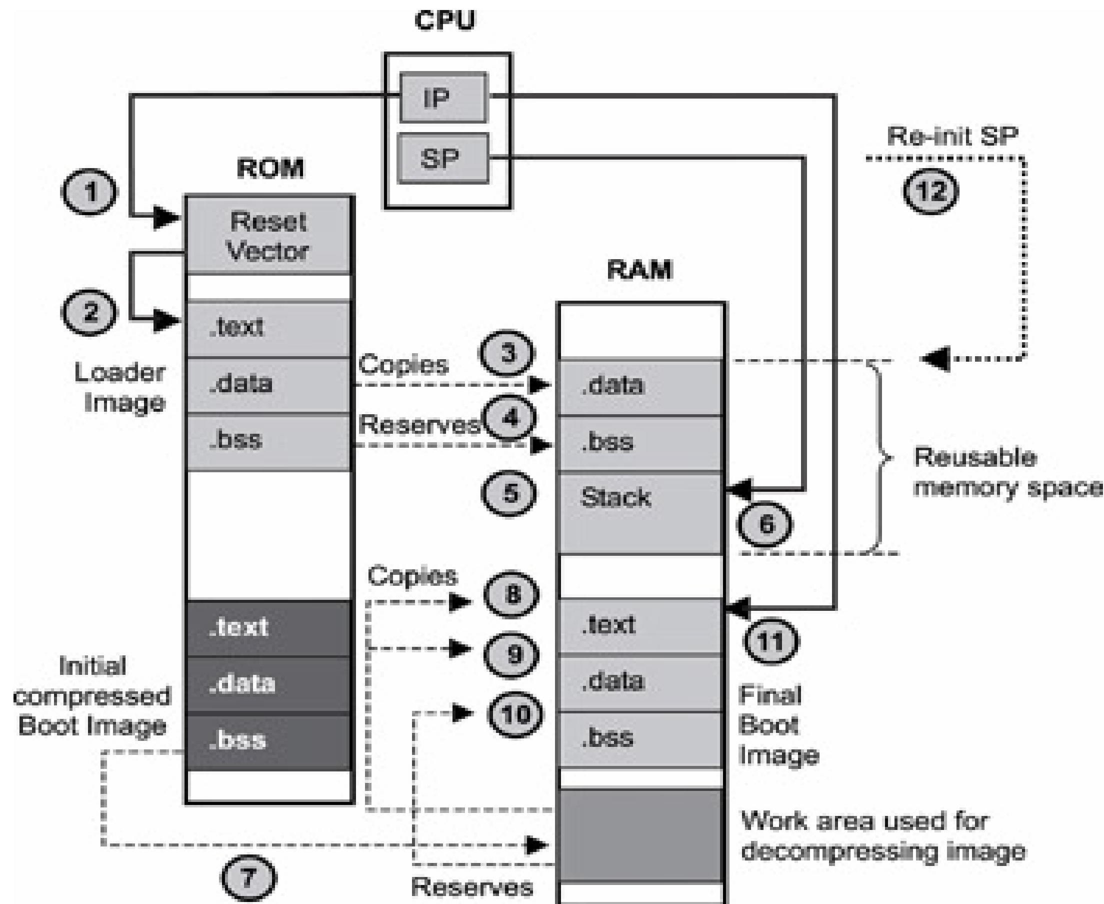
5、RAM分配堆栈空间。

6、CPU的SP堆栈指针指向新建立的堆栈空间的开始地址。启动完成后，CPU继续运行直到系统断电或代码执行结束。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.1 硬件系统启动模式(5)

- 程序从ROM拷贝到RAM后，从RAM运行



Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

3.2.1 硬件系统启动模式(6)

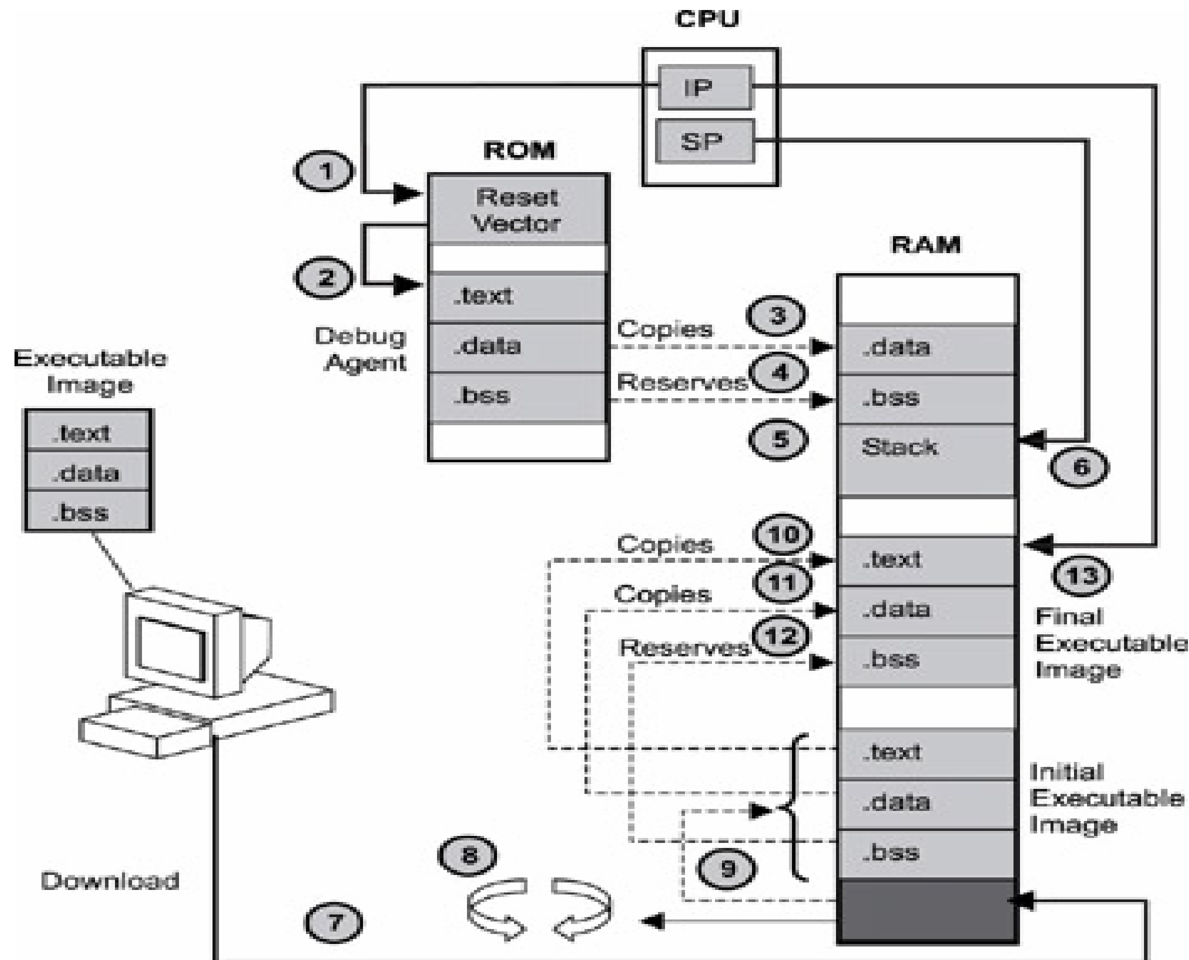
前六个步骤是与前面一个模式相同，在完成上面六个步骤后，处理器继续执行：

7、压缩的应用程序影像文件从ROM拷贝到RAM
8-10、解压缩流程完成的初始化部分。
11、引导程序将控制权交给应用程序。
12、重新初始化堆栈指针。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.1 硬件系统启动模式(7)

- 程序主机端传到RAM后，从RAM运行



Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

3.2.1 硬件系统启动模式(8)

前六个步骤是与前面一个模式相同，在完成上面六个步骤后，处理器继续执行：

7、应用程序影像文件从主机开发系统传过来。

8、检验应用程序影像文件的完整性。

9、如果必要文件被解压缩。

10-12、目标调试工具下载应用程序影像文件各自的段到RAM的他们运行的地址里

13、运行程序

Copyright © 2003-2007 SinoSys. All Rights Reserved

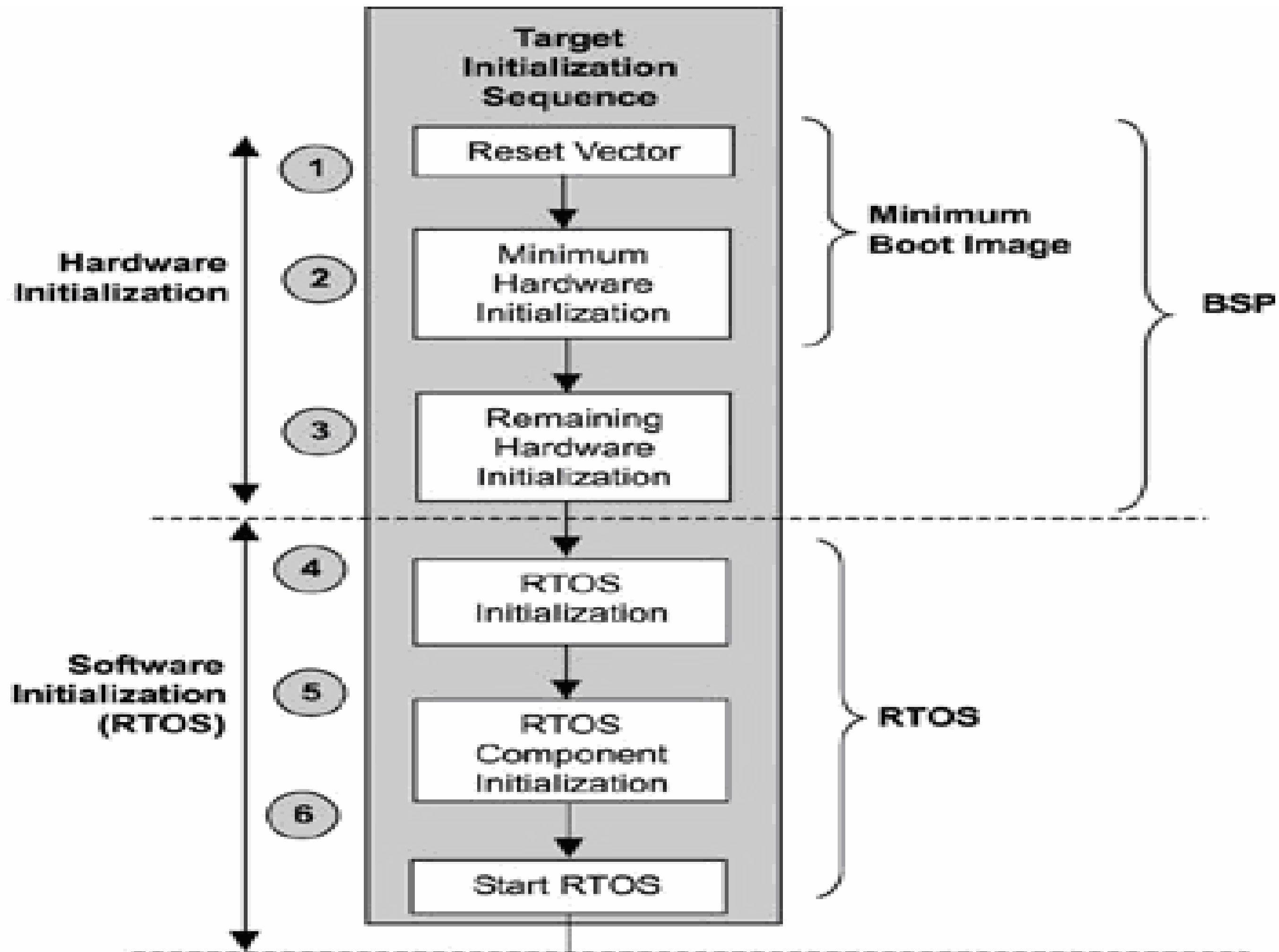
3.2.2 硬件系统初始化(1)

最小的硬件初始化要求引导程序执行,包括 ：

- 从复位向量处开始执行
- 通过设置合适的寄存器使处理器处于一个已知的状态：
  获得处理器的型号
  获得或设置CPU的时钟速度
- 禁止中断和缓冲
- 初始化内存控制器，内存芯片和存储单元:
  获得内存的初始地址
  获得内存大小
  如果有要求，执行初步存储测试

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.2 硬件系统初始化(2)



Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.2 硬件系统初始化(3)

RTOS初始化

图的步骤4为RTOS的初始化。关键部分在图步骤4和6：

- 初始化 RTOS
- 如果存在，初始化不同的RTOS 对象和服务:
  - 任务对象
  - 信号对象
  - 消息队列对象
  - 定时器服务
  - 中断服务
  - 存储管理服务
- 为RTOS创建需要的栈
- 初始化增加的RTOS栈，例如:
  - TCP/IP 栈
  - 文件系统
- 启动RTOS，初始化任务

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(1)

- 系统启动程序所执行的操作跟具体的目标系统和开发系统相关，一般通用的内容包括：
  - 中断向量表；
  - 初始化存储器系统；
  - 初始化堆栈；
  - 初始化有特殊要求的端口、设备；
  - 初始化应用程序执行环境；
  - 改变处理器模式；
  - 呼叫主应用程序。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(2)

- 中断向量表

地址	向量	说明
0x1C	FIQ	外部快速中断
0x18	IRQ	普通外部中断
0x14	(Reserved)	保留
0x10	Data Abort	数据异常中断
0x0C	Prefetch Abort	指令预取中断
0x08	Software Interrupt	软中断
0x04	Undef	未定义指令中断
0x00	Reset	复位中断

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

3.2.3 基于ARM9的嵌入式系统的初始化(3)

- 初始化存储器系统

（1）存储器类型和时序配置

（2）存储器地址分布

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(4)

- 初始化堆栈

  因为ARM处理器有7种执行状态，每一种状态的堆栈指针寄存器(SP)都是独立的(System和User模式使用相同的SP寄存器)。因此，对程序中需要用到的每一种模式都要给SP寄存器定义一个堆栈地址。方法是改变状态寄存器(CPSR)内的状态位，使处理器切换到不同的状态，然后给SP赋值。注意：不要切换到User模式进行User模式的堆栈设置，因为进入User模式后就不能再操作CPSR回到别的模式了。可能会对接下去的程序执行造成影响。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(5)

- 初始化有特殊要求的端口、设备

这要由具体的系统和用户需求而定。一般的外设初始化可以在系统初始化之后进行。

比较典型的应用是驱动一些简单的输出设备LED等，来指示系统启动的进程和状态。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(6)

- 初始化应用程序执行环境

ZI(Zero initialized R/W Data)	只定义了变量名的全局变量
RW(R/W Data)	定义时带初始化的全局变量
RO(Code+RO Data)	编译结果

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(7)

- 改变处理器模式

ARM处理器(V4架构以后的版本)一共有7种执行模式：

User：用户模式

FIQ：快速中断响应模式

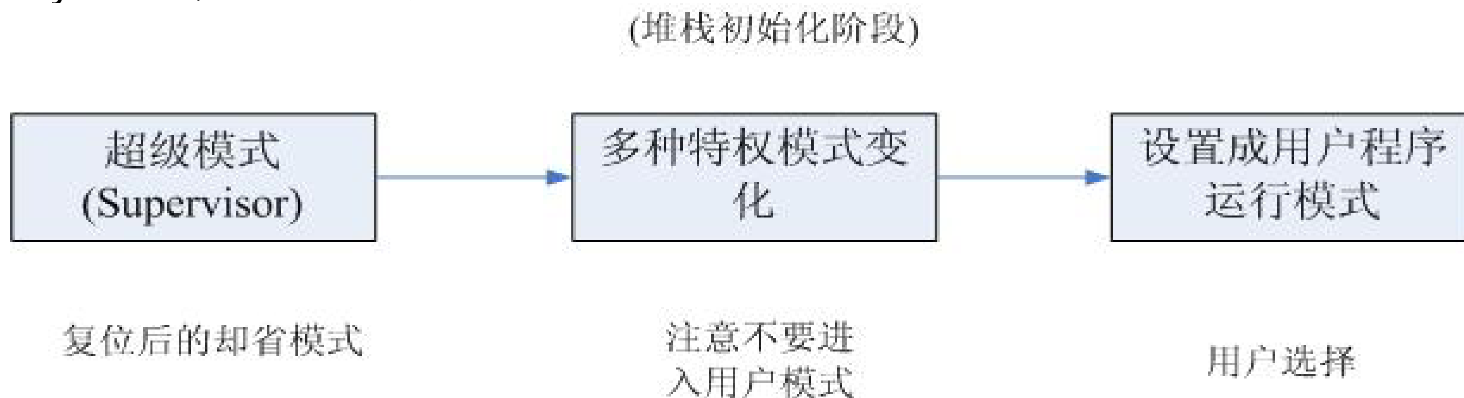
IRQ：一般中断响应模式

Supervisor超级模式

Abort；出错处理模式

Undef：未定义模式

System；系统模式



Copyright © 2003-2007 SinoSys. All Rights Reserved

3.2.3 基于ARM9的嵌入式系统的初始化(8)

当所有的系统初始化工作完成之后，就需要把程序流程转入主应用程序。最简单的—种情况是：

```
IMPORT  main  ;get the label main
B       main  ;jump to main()
```

在ARM ADS环境中，还另外提供了一套系统级的呼叫机制。

```
IMPORT   __main
B        __main
```

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.3 Boot loader概述

3.3.1 Boot loader的概念
3.3.2 Boot loader的典型结构框架
3.3.3 Boot loader的stage1
3.3.4 Boot loader的stage2

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.3.1 Boot loader的概念(1)

- 简单地说，Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。
- 通常，Boot Loader 是严重地依赖于硬件而实现的，特别是在嵌入式世界。因此，在嵌入式世界里建立一个通用的 Boot Loader 几乎是不可能的。尽管如此，我们仍然可以对 Boot Loader 归纳出一些通用的概念来，以指导用户特定的 Boot Loader 设计与实现。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.3.1 Boot loader的概念(2)

Boot Loader 归纳出一些通用的概念：

- 1、Boot Loader 所支持的 CPU 和嵌入式板型
- 2、Boot Loader 的安装媒介（Installation Medium）
- 3、用来控制 Boot Loader 的设备或机制
- 4、Boot Loader 的启动过程是单阶段（Single Stage）还是多阶段（Multi-Stage）
- 5、Boot Loader 的操作模式 (Operation Mode)
- 6、Boot Loader 与主机之间进行文件传输所用的通信设备及协议

Copyright © 2003-2007 SinoSys. All Rights Reserved

双实科技 SinoSys

3.3.2 Boot loader的典型结构框架

- 由于 Boot Loader 的实现依赖于 CPU 的体系结构，因此大多数 Boot Loader 都分为 stage1 和 stage2 两大部分。依赖于 CPU 体系结构的代码，比如设备初始化代码等，通常都放在 stage1 中，而且通常都用汇编语言来实现，以达到短小精悍的目的。而 stage2 则通常用C语言来实现，这样可以实现复杂的功能，而且代码会具有更好的可读性和可移植性。

Copyright © 2003-2007 SinoSys. All Rights Reserved

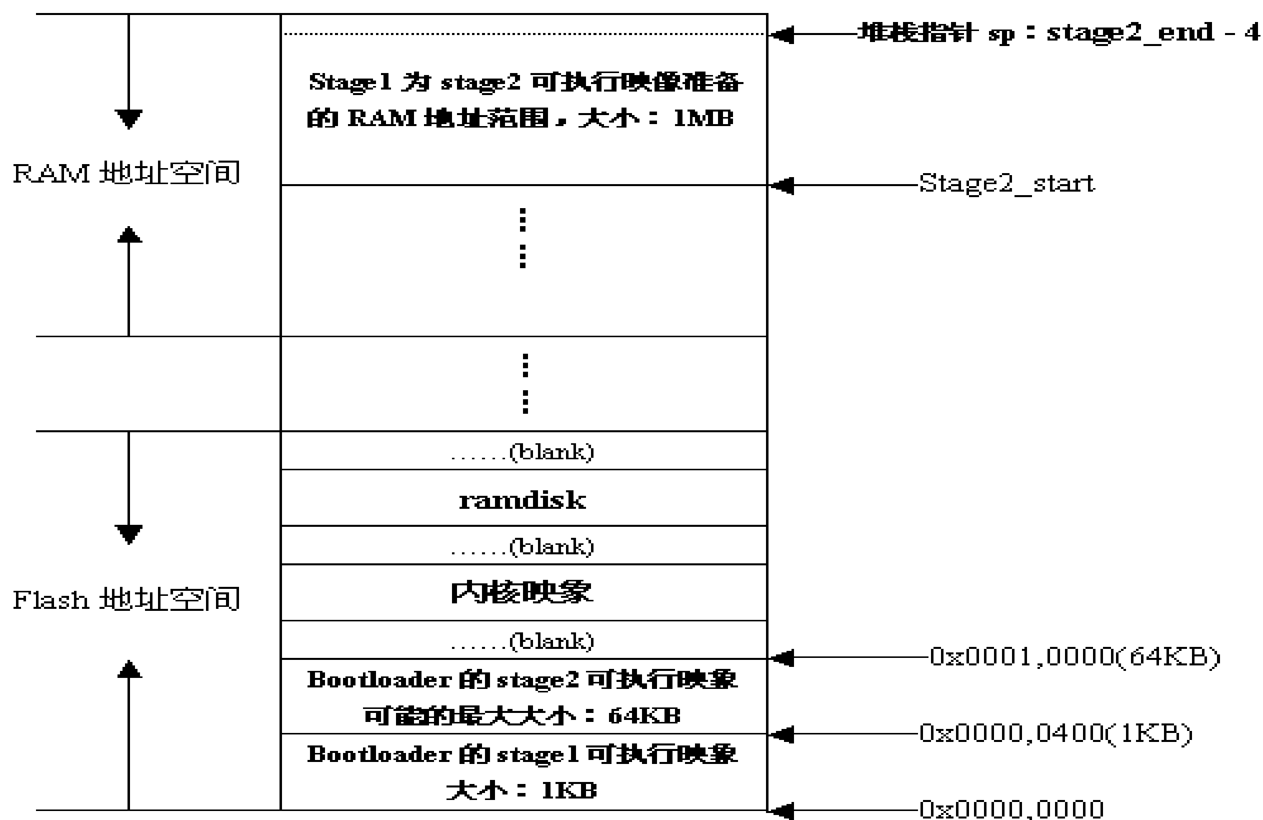
3.3.3 Boot loader的stage1(1)

- 硬件设备初始化。
- 为加载 Boot Loader 的 stage2 准备 RAM 空间。
- 拷贝 Boot Loader 的 stage2 到 RAM 空间中。
- 设置好堆栈。
- 跳转到 stage2 的 C 入口点。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.3.3 Boot loader的stage1(2)

- **Boot loader 的 stage2 可执行映象被拷贝到 RAM 空间时的系统内存布局**



Copyright © 2003-2007 SinoSys. All Rights Reserved

3.3.4 Boot loader的stage2

- 初始化本阶段要使用到的硬件设备。
- 检测系统内存映射(memory map)。
- 将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中。
- 为内核设置启动参数。
- 调用内核。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4 常用基于ARM9的Boot loader

5.4.1 vivi的设计及实现

5.4.2 U-Boot的设计及实现

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.1 vivi的设计及实现(1)

- vivi是韩国Mizi公司开发的Boot loader，适用于ARM9处理器。vivi也有前面说过的两种工作模式，启动加载模式可以在一段时间后(这个时间可更改)自行启动Linux内核，这是vivi的默认模式。在下载模式下，vivi为用户提供一个命令行接口，通过该接口可以使用vivi提供的一些命令：
- Load把二进制文件载入Flash或者RAM
- Part操作MTD分区信息。显示、增加、删除、复位、保存MTD分区
- Param设置参数
- Boot启动系统
- Flash管理Flash，如删除Flash的数据

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.1 vivi的设计及实现(2)

vivi包括下面几个目录。

- arch：此目录包括了所有vivi支持的目标板的子目录，这里只有s3c2440目录
- drivers：其中包括了引导内核需要的设备的驱动程序（MTD和串口）。MTD目录下分map、nand和nor三个目录。
- init：这个目录只有main.c和version.c两个文件。和普通的C程序一样，vivi将从main函数开始运行。
- lib：一些平台公共的代码，比如：time.c里的udelay( )和mdelay( )。
- include：头文件的公共目录，其中的s3c2440.h定义了这块处理器的一些寄存器，以及NAND FLASH的一些寄存器等。Platform/smdk2440.h定义了与这块开发板的相关的资源配置参数，我们往往只需要修改这个文件就可以配置目标板的参数。如：波特率、引导参数、物理内存映射等。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.1 vivi的设计及实现(3)

- vivi的第一阶段

既然是Boot loader，那么vivi的运行也可以分两个阶段。在第一阶段完成含依赖于CPU的体系结构硬件初始化的代码，包括禁止中断、初始化串口、复制自身到RAM等。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.1 vivi的设计及实现(4)

vivi的第二阶段

- Boot loader的第二阶段是用C语言完成的。但是与普通C语言应用程序不同的是：在编译和链接Boot loader时，不能使用glibc库中的任何支持函数。这就产生一个问题：从哪里跳转进main()函数呢？直接把main()函数的起始地址作为整个stage2执行映像的点或许是最直接的想法，但是这样做有两个缺点：

  无法通过main()函数传递参数；

  无法处理main()函数返回的情况。

- 一种更巧妙的方法是利用trampoline的概念，即用汇编语言写一段trampoline程序，并将这段trampoline程序作为stage2可执行映像的执行入口点。之后可以在trampoline程序中用处理器的跳转指令跳入main()函数中去执行。当main()函数返回时，CPU执行路径将再次回到原来的trampoline程序。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.1 vivi的设计及实现(5)

- 同一般的C语言程序一样，vivi从main()函数开始，该函数在／init／main.c文件中，总共可以分为8个步骤。函数的开始，通过putstr(vivi_banner)打印出vivi的版本。
- 第二步，对开发板进行初始化(board_init函数)，board_init是与开发板紧密相关的。
- 如果初始化正确，则开始内存映射初始化和内存管理单元的初始化工作，即第三步。
- 第四步，初始化堆，heap_init( ) 。
- 第五步，初始化mtd设备，mtd_dev_init( )。
- 第六步，初始化私有数据，init_priv_data( )。
- 第七步，初始化内置命令，init buildn cmds()。
- 第八步，启动vivi-boot_or_vivi()。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.2 U-Boot的设计及实现(1)

- U-Boot代码采用了一种高度模块化的编程方式，与移植相关的有以下几个目录。

- ✓ board：这个目录存放了所有U-Boot支持的目标板的子日录，如board／smdk2440／*就是我们所关心的。要将U-Boot移植到自己的S3C2440X目标板上，必须参考这个目录下的内容，比如对Flash以及Flash宽度和大小的定制等就要修改其中的flash.c。
- ✓ cpu：这个目录存放了U-Boot支持的CPU类型，我们只关心cpu／arm920t，CPU相关的文件主要是初始化一个执行环境，包括中断的初始化；start.S是整个u-boot.bin目标可执行代码的第一段代码，它们是从Flash开始运行的，其主要工作就是对整个U-Boot目标代码的重定位，即将U-Boot转移到内存中去运行。
- ✓ common：这个目录存放了U-Boot的一些公共命令的实现，像那些以cmd_*.c为名字的文件就是对应U-Boot的每个命令的实现代码，我们通常关心cmd_boot．c和cmd bootm．c(它们和内核的引导相关)。
- ✓ drivers：这个目录中存放了各种外设接口的驱动程序。
- ✓ fs：这个目录中存放了U-Boot支持的文件系统。
- ✓ lib_arm：这个目录存放了ARM平台公共的接口代码。
- ✓ include：这个目录存放头文件的公共目录，其中的include/configs/smdk2440.h定义了所有和S3C2440X相关的资源的配置参数，我们往往只需修改这个文件就可以配置目标板的参数，如波特率、引导参数、物理内存映射等。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.4.2 U-Boot的设计及实现(2)

- 常用命令及功能

命令名	功能
help/?	帮助命令
bdinfo	查看目标板参数和变量、硬件配置等
setenv	设置环境变量
printenv	查看环境变量
saveenv	保存设置的环境变量到 FLASH
mw	写内存
md	查看内存
mm	修改内存
finfo	查看 FLASH 的信息
erase[起始地址　结束地址]	查除 FLASH 内容，必须以扇区为单位进行擦除
cp[源地址　目标地址大小]	内存复制，可以在 RAM 和 FLASH 中交换数据
imi[起始地址	查看内核映像文件
bootm[起始地址]	从某个地址启动内核
tftpboot[起始地址　镜像名]	通过 tftp 从主机系统下载内核映像文件
reset	复位

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.5 小结

- 本章讲述了Boot Loader的概念，重点分析系统初始化过程以及两个引导程序：vivi和U-Boot。Boot Loader与硬件联系非常紧密，特别是处理器的启动流程部分。在开发Boot Loader程序时，一定要先掌握原理然后进行移植，否则当程序出现问题时，不知从哪里下手查找问题的所在。

Copyright © 2003-2007 SinoSys. All Rights Reserved

3.6 实验

3-1 ARM9处理器启动文件的分析和设计

Copyright © 2003-2007 SinoSys. All Rights Reserved