嵌入式系统

第二讲 arm处理器

arm处理器

- ARM体系结构
- ARM编程模型
- ARM指令集

V1版架构

- 该版架构只在原型机ARM1出现过,其基本性能：
- 基本的数据处理指令(无乘法)
- 字节、半字和字的LOAD/STORE指令
- 转移指令,包括子程序调用及链接指令
- 软件中断指令
- 寻址空间：64M字节(226)
- 支持
  - 基本数据处理指令（不含乘法）
  - 字节、字、半字的load，store
  - 分支指令，包括子程序调用
  - 软件中断指令，进行操作系统调用
  - 26位寻址
- 没有使用

V2版架构

- 该版架构对V1版进行了扩展,如ARM2与ARM3(V2a版)架构,增加了以下功能：
- 乘法和乘加指令
- 支持协处理器操作指令
- 快速中断模式
- SWP/SWPB的最基本存储器与寄存器交换指令
- 寻址空间：64M字节
- 支持
  - 增加乘法，乘加
  - 协处理器支持
  - 快速中断中2个以上分组的寄存器
  - 称为SWP与SWPB的原子性加载与存储
  - 26位寻址
- 不使用了

V3版架构

- 把寻址空间增至32位(4G字节),
- 增加了当前程序状态寄存器CPSR(Current Program Status Register)和程序状态保存寄存器SPSR(Saved Program Status Register)以便于异常(Exception)的处理。
- 增加了中止(Abort)和未定义二种处理器模式。ARM6就采用该版架构。指令集变化如下：
- 增加了MRS/MSR指令,以访问新增的CPSR/SPSR寄存器
- 增加了从异常处理返回的指令功能。
- 支持
  - 寻址32位
  - 增加CPSR(Current Program Status Register)
  - 增加SPSR(Saved Program Status Register)
  - 增加访问CPSR与SPSR的指令
  - 修改了过去用于异常返回的指令的功能
  - 与26位寻址模式兼容

V4版架构

- V4版架构是目前应用最广的ARM体系结构,对V3版架构进行了进一步扩充,有的还引进了16位的Thumb指令集,使ARM使用更加灵活。ARM7、ARM8、ARM9和StrongARM都采用该版架构。指令集中增加了以下功能：
- 符号化和非符号化半字及符号化字节的存/取指令
- 增加了16位Thumb指令集
- 完善了软件中断SWI指令的功能
- 处理器系统模式引进特权方式时使用用户寄存器操作
- 把一些未使用的指令空间捕获为未定义指令
- 支持
  - 半字load，store
  - 加载与进行字节和半字节带符号扩展
  - 在T变量中，一个转换到Thumb状态的指令
  - 使用用户模式寄存器的新的特权处理器模式
  - 不再要求与26位寻址模式兼容

V5版架构

- 这是最近推出ARM架构,在V4版基本上增加了一些新的指令,ARM10和XScale都采用该版架构,这些新增指令有：
- 带有链接和交换的转移BLX指令
- 计数前导零CLZ指令
- BRK中断指令
- 增加了数字信号处理指令(V5TE版)
- 为协处理器增加更多可选择的指令
- 支持
  - 提高T变量中ARM/Thumb切换效率
  - 让非T变量同T变量一样，使用相同的代码生成技术
  - 增加一个计数前导零指令
  - 增加软件断点指令
  - 为协处理器设计者增加更多可选择指令
  - 对乘法如何设置标志严密定义

v6版架构

- 2001年发布的
- 350-500Mhz
- 130nm工艺
- 功耗0.4mw/mhz
- 增加了SIMD功能扩展
- 适合使用电池供电的便携式设备
- SIMD功能扩展
  - 包括音频/视频处理在内的应用系统提供了优化功能
  - 可以使音频/视频处理性能提高4倍
- 首先在2002年发布的ARM11处理器中使用

ARM系列产品表示

- ARM系列产品很多,以ARM7系列为例,其内核ARM7TDMI表示为：
- ·ARM7： ARM系列具有32位整数运算核
- ·T： 内含16位压缩指令集Thumb
- ·D： 支持片内Debug调试,
- ·M： 采用增强型乘法器(Multiplier),
- ·I： 内含嵌入式ICE宏单元
- 另外,各产品的后缀提供了各种形式与功能的选择：
- ·-S：可综合的软核Softcore
- ·-E：具有DSP的功能
- ·-J：Jazeller,允许直接执行Java字节码

arm处理器系列

- ARM 7 （V4版架构）
  - 系列产品
    - ARM7TDMI/ARM7TDMI/ARM720T
    - ARM7EJ－最低功耗
  - 具有：
    - 嵌入式ICE-RT逻辑－硬件上提供片上调试断点支持
    - 非常低的功耗
    - 没有MMU
    - 提供1MIPS/MHz的三级流水线和冯.诺依曼体系
- ARM 9 （V4版架构）
  - 系列产品
    - ARM920T与ARM922T
    - ARM940T
  - 具有：
    - 5级流水线，1.1MIPS/MHz的哈佛结构
- ARM 9E （V4版架构）
  - 系列产品
    - ARM966E-S
    - ARM946E-S
    - ARM926EJ-S
  - 提供
    - DSP扩充
    - 嵌入式ICE-RT调试逻辑
    - 1.1MIPS/MHz的5级流水线和哈佛结构
    - 紧耦合存储器（TCM）接口，可使存储器以最高处理器速度运转，可直接连到内核上
- ARM 10E （V5版架构）
  - 产品系列
    - ARM1022E rev0/ARM1020E rev1
  - 提供
    - DSP扩展
    - 嵌入式ICE-RT
    - 全性能MMU
    - Cache
    - 对于指令与数据，64位AHB接口
    - 6级流水线
    - 内部64位数据通道
    - 1.25MIPS/MHz
    - 比同等ARM9器件，同样时钟下，性能提高50％
- ARM 11 （V6版架构）
  - 2002
  - 面向高性能
  - ARMv6
  - 8级流水，支持SIMD
- Securcore
  - 专为安全设计
  - 抗窜改（resist tampering）
  - 逆向工程（reverse engineering）
  - 保护机构－确保操作系统与数据安全

ARM体系结构v7版本

- ARM Cortex系列简介

基于ARMv7版本的ARM Cortex系列产品由A、R、M三个系列组成，具体分类延续了一直以来ARM面向具体应用设计CPU的思路。

ARM体系结构v7版本

- ARMv7-A　针对复杂操作系统设计，支持虚拟地址
- ARMv7-R　面向实时系统 ，不支持虚拟地址
- ARMv7-M　针对成本和功耗敏感的嵌入式控制器，只支持thumb指令集

ARM微处理器系列

- Cortex™-M3处理器简介

该处理器是首款基于ARMv7-M架构的处理器，采用了纯Thumb2指令的执行方式，具有极高的运算能力和中断相应能力。

Cortex-M3主要应用于汽车车身系统，工业控制系统和无线网络等对功耗和成本敏感的嵌入式应用领域。目前最便宜的基于该内核的ARM单片机售价为1美元。功耗为200 μA/MHz。

ARM微处理器系列

- Cortex™-R4处理器简介

该处理器是首款基于ARMv7架构的高级嵌入式处理器，其主要目标为产量巨大的高级嵌入式应用系统，如硬盘，喷墨式打印机，以及汽车安全系统等等。

- Cortex™-R4F处理器简介

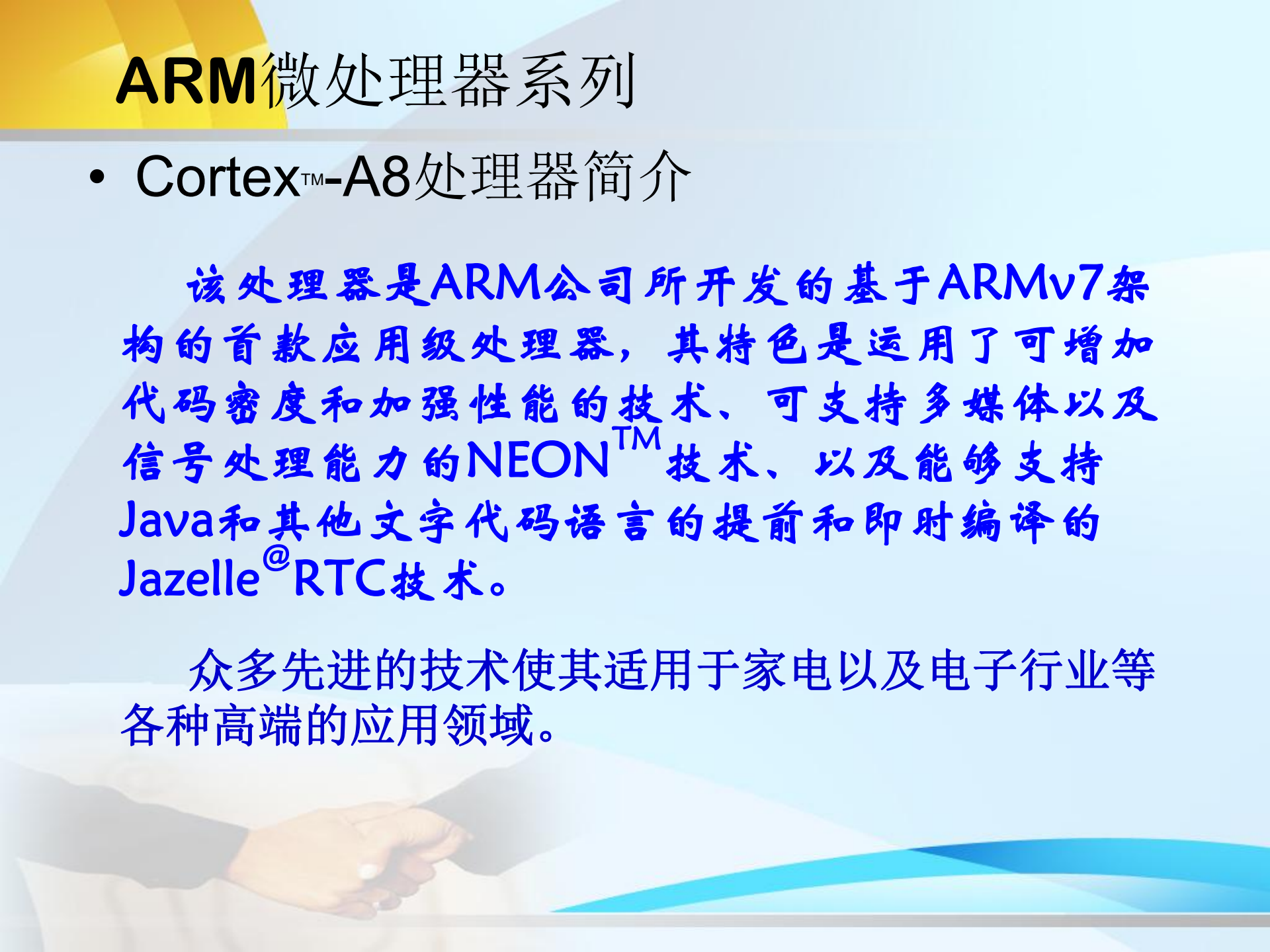
该处理器在Cortex™-R4处理器的基础上加入了代码错误校正(ECC)技术，浮点运算单元(FPU)以及DMA综合配置的能力，增强了处理器在存储器保护单元、缓存、紧密耦合存储器、DMA访问以及调试方面的能力。

ARM微处理器系列

- Cortex™-A8处理器简介

该处理器是ARM公司所开发的基于ARMv7架构的首款应用级处理器，其特色是运用了可增加代码密度和加强性能的技术、可支持多媒体以及信号处理能力的NEON™技术、以及能够支持Java和其他文字代码语言的提前和即时编译的Jazelle®RTC技术。

众多先进的技术使其适用于家电以及电子行业等各种高端的应用领域。



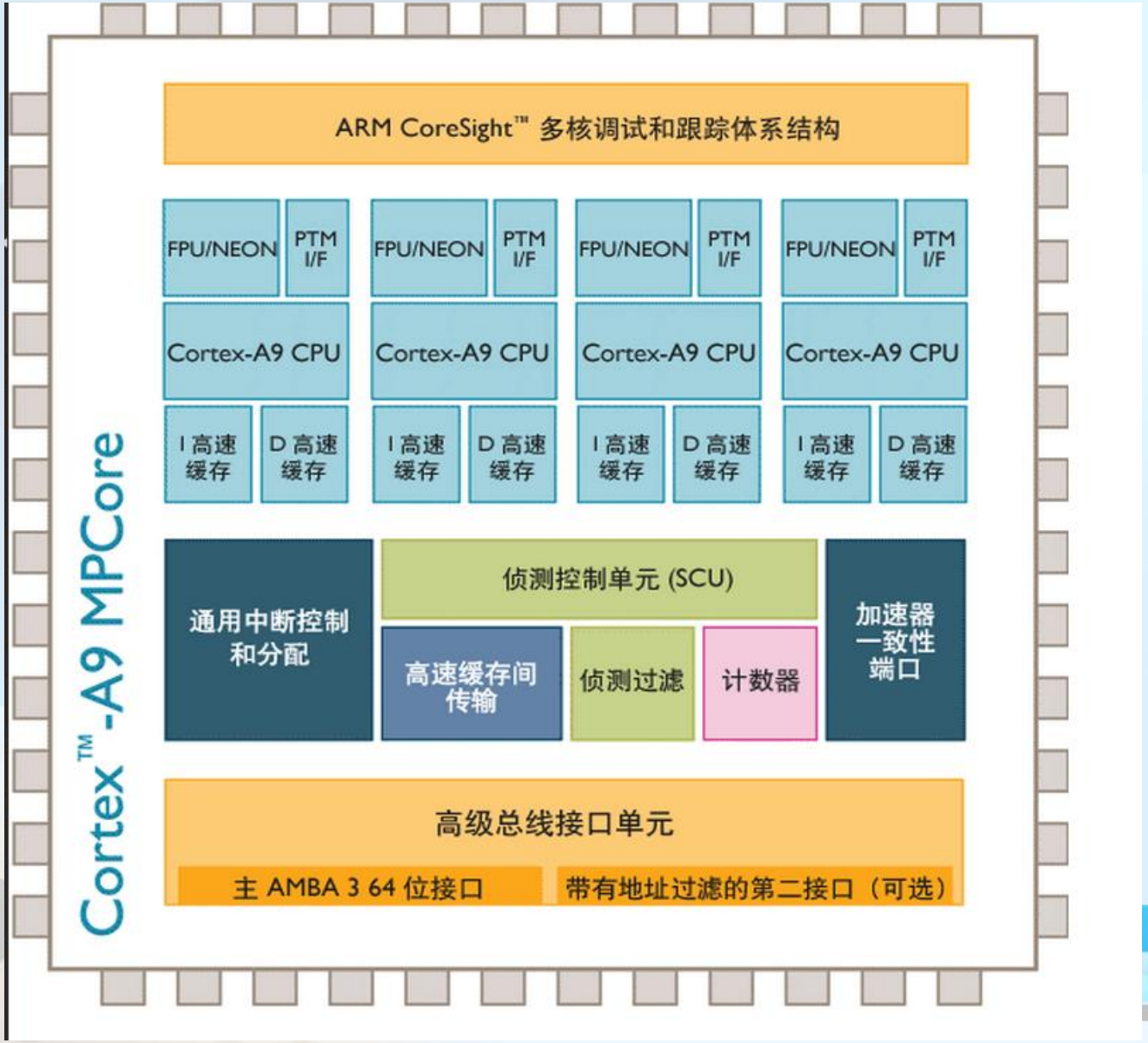
ARM微处理器系列

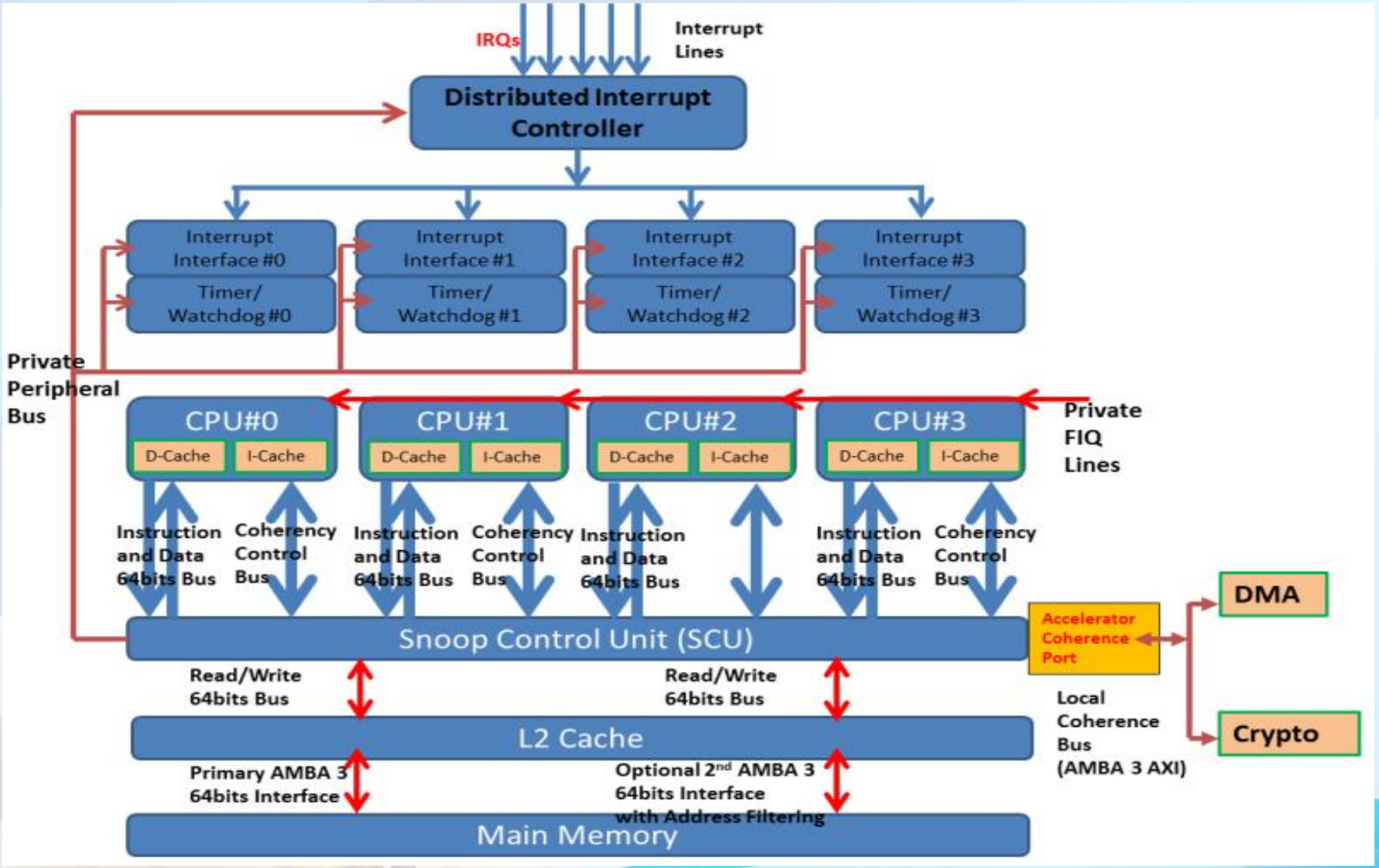
- Cortex™-A9处理器简介(MPCore架构)
  - **SMP（Symmetric Multi-Processing）**

对称多处理结构，是指各CPU之间共享内存子系统以及总线结构等资源。多个 CPU 对称工作，无主次或从属关系。作系统管理着一个任务队列，每个处理器依次处理队列中的进程。

  - **NUMA(Non-Uniform Memory Access)**

非一致存储访问结构，具有多个 CPU 模块，每个 CPU 模块由多个 CPU(如 4 个) 组成，并且具有独立的本地内存、 I/O 槽口等。节点之间可以通过互联模块 (如称为 Crossbar Switch) 进行连接和信息交互。


Cortex™-A9 MPCore
ARM CoreSight™ 多核调试和跟踪体系结构
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
FPU/NEON
PTM I/F
Cortex-A9 CPU
Cortex-A9 CPU
Cortex-A9 CPU
Cortex-A9 CPU
I 高速缓存
D 高速缓存
I 高速缓存
D 高速缓存
I 高速缓存
D 高速缓存
I 高速缓存
D 高速缓存
通用中断控制和分配
侦测控制单元 (SCU)
高速缓存间传输
侦测过滤
计数器
加速器一致性端口
高级总线接口单元
主 AMBA 3 64 位接口
带有地址过滤的第二接口（可选）


IRQs
Interrupt Lines
Distributed Interrupt Controller
Interrupt Interface #0
Interrupt Interface #1
Interrupt Interface #2
Interrupt Interface #3
Timer/ Watchdog #0
Timer/ Watchdog #1
Timer/ Watchdog #2
Timer/ Watchdog #3
Private Peripheral Bus
CPU#0
CPU#1
CPU#2
CPU#3
D-Cache
I-Cache
Private FIQ Lines
Instruction and Data 64bits Bus
Coherency Control Bus
Snoop Control Unit (SCU)
Accelerator Coherence Port
DMA
Crypto
Local Coherence Bus (AMBA 3 AXI)
Read/Write 64bits Bus
L2 Cache
Primary AMBA 3 64bits Interface
Optional 2nd AMBA 3 64bits Interface with Address Filtering
Main Memory

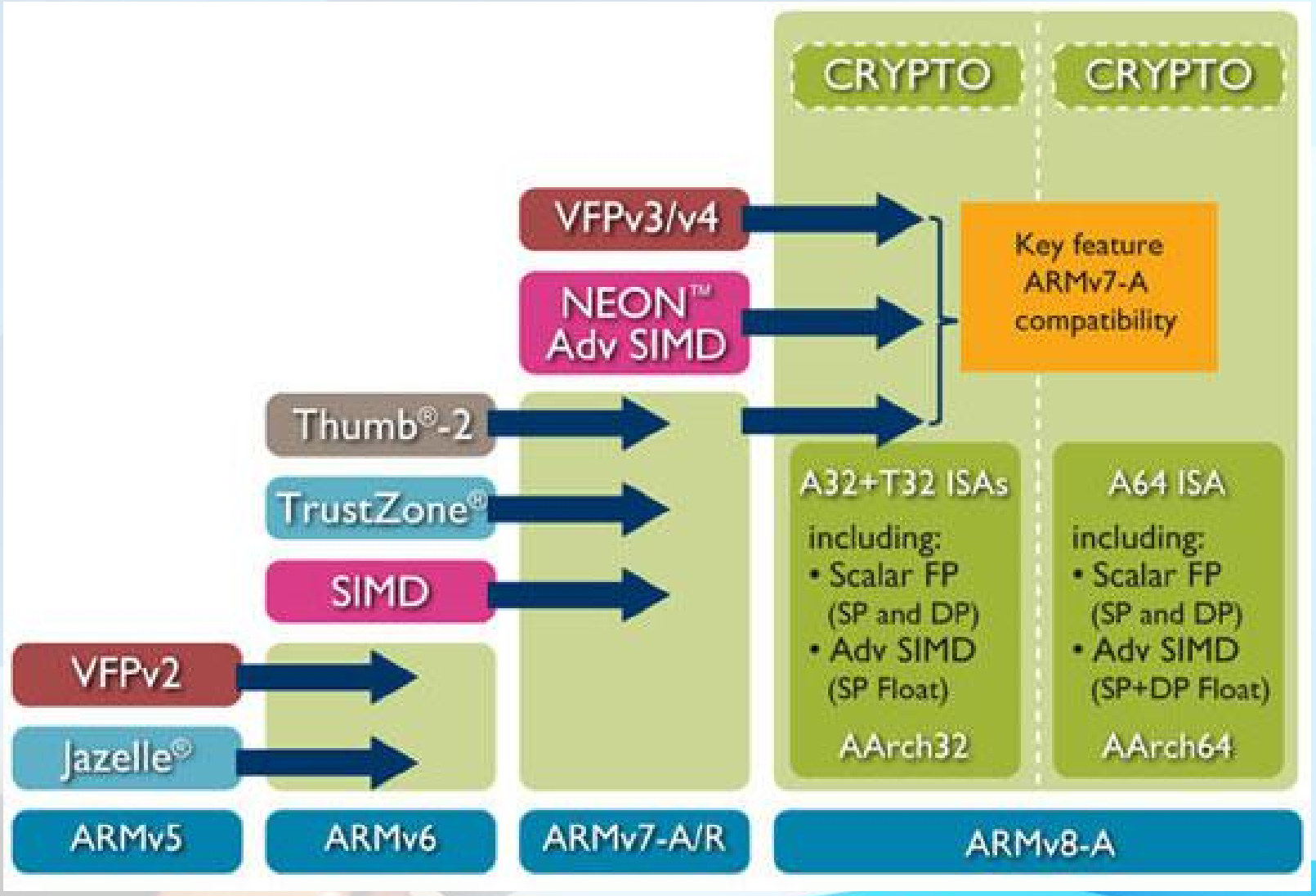
ARM体系结构v8版本

- ◆将64位架构支持引入ARM架构中
- ◆支持三个主要指令集

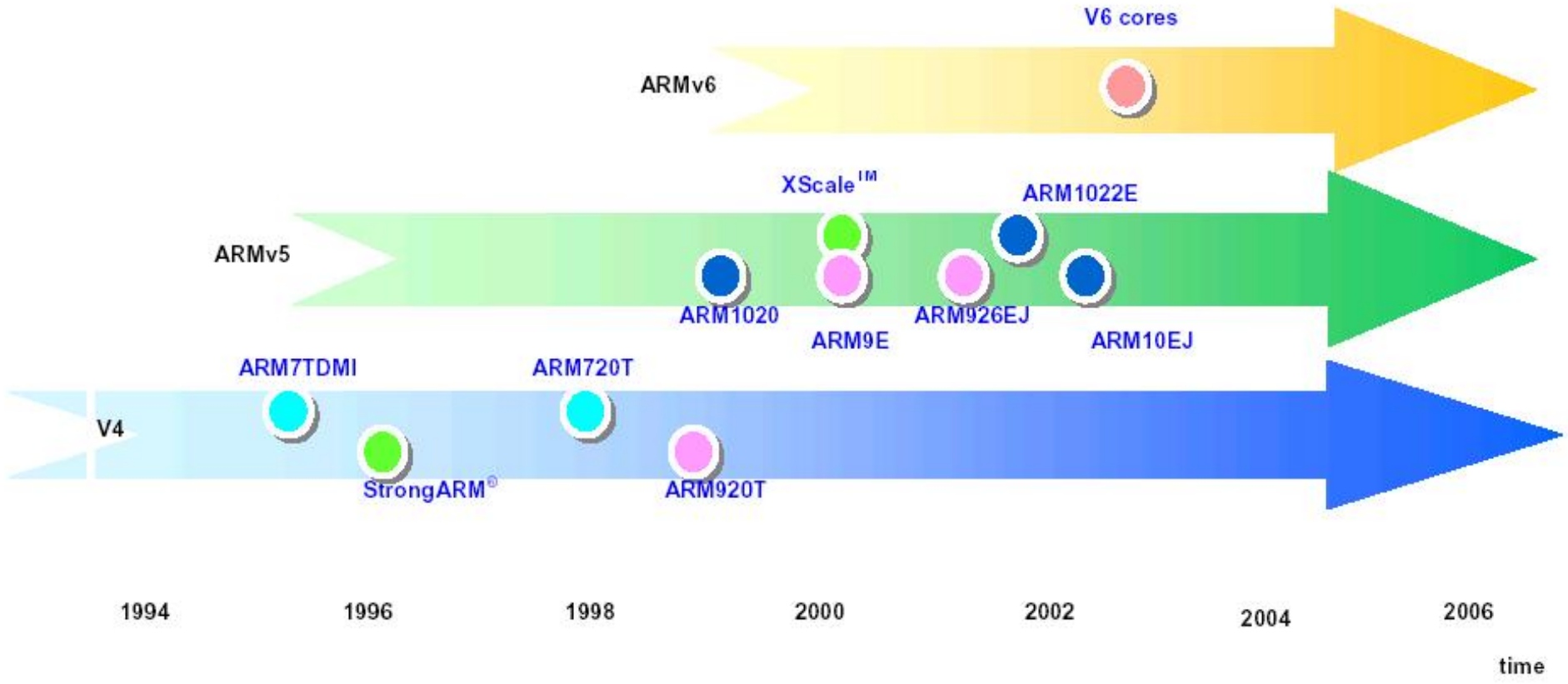
A32（或 ARM）：32 位固定长度指令集

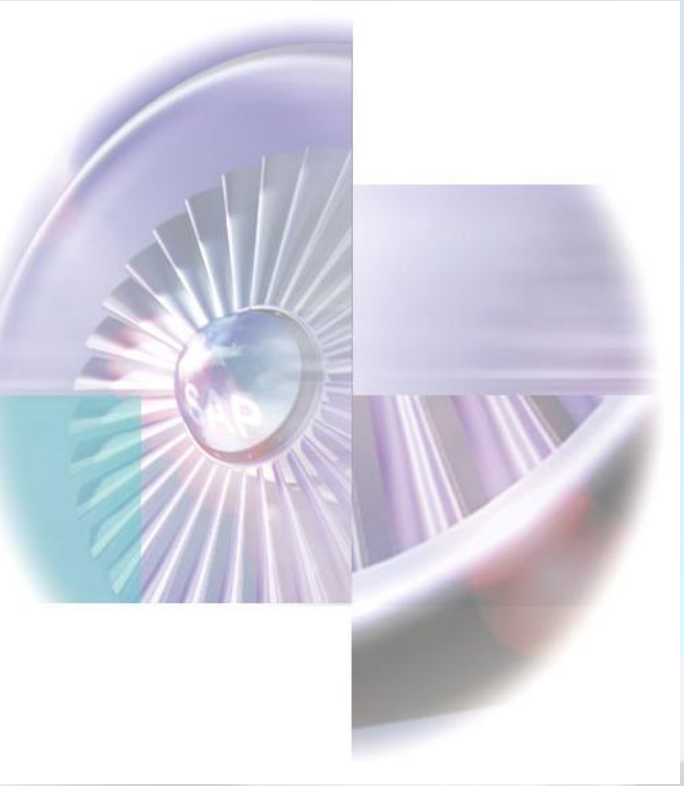
T32 (Thumb)：16 位固定长度指令集

A64：提供与 ARM 和 Thumb 指令集类似功能的 32 位固定长度指令集


CRYPTO
CRYPTO
VFPv3/v4
NEON™
Adv SIMD
Key feature
ARMv7-A
compatibility
Thumb®-2
TrustZone®
SIMD
A32+T32 ISAs
including:
• Scalar FP
(SP and DP)
• Adv SIMD
(SP Float)
AArch32
A64 ISA
including:
• Scalar FP
(SP and DP)
• Adv SIMD
(SP+DP Float)
AArch64
VFPv2
Jazelle®
ARMv5
ARMv6
ARMv7-A/R
ARMv8-A

ARM 体系结构更新

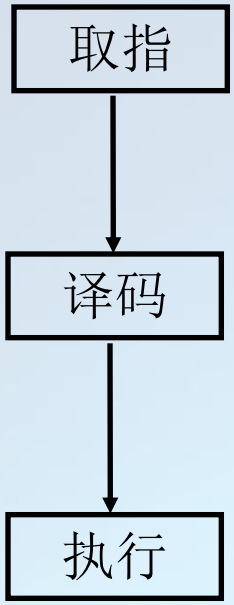

V6 cores
ARMv6
XScale™
ARM1022E
ARMv5
ARM1020
ARM926EJ
ARM9E
ARM10EJ
ARM7TDMI
ARM720T
V4
StrongARM®
ARM920T
1994
1996
1998
2000
2002
2004
2006
time



ARM编程模型

ARM7的三级指令流水线

每条指令可以分3个阶段执行



从存储器取指

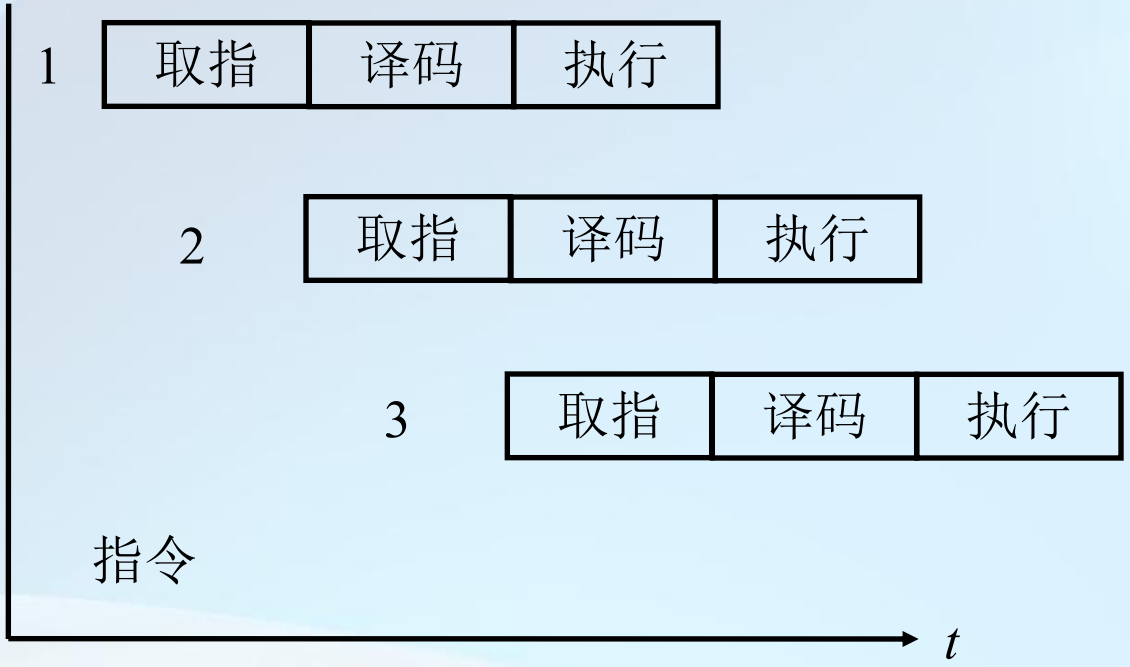
指令所用的寄存器译码

从从寄存器组中读寄存器

移位和ALU操作

把寄存器写回到存储器组

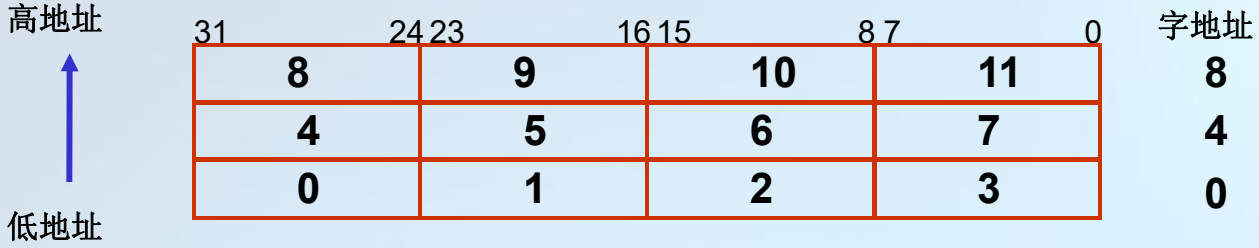
ARM7的三级指令流水线



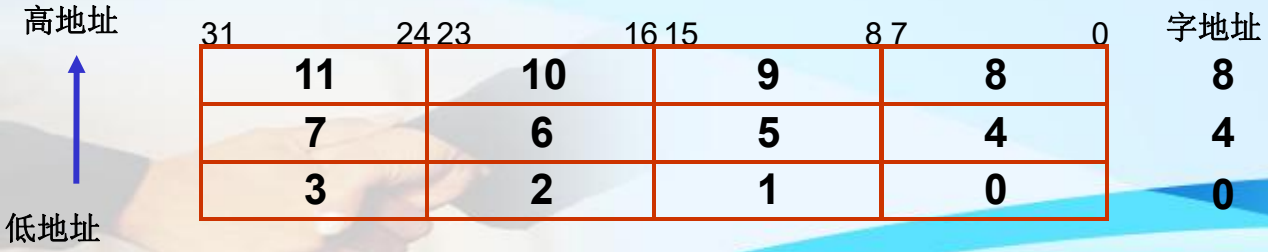
注:程序计数器PC指向正在取指的指令而不是正在执行的指令

存储器模式

- 大端模式
  - 最高位字节保存在最低位地址
  - 字由最低位字节的字节地址寻址



- 小端模式
  - 最低位字节保存在最低位地址
  - 字由最低位字节的字节地址寻址



处理器模式

ARM微处理器支持7种运行模式，分别为：

- 用户模式（usr）： ARM处理器正常的程序执行状态。
- 快速中断模式（fiq）： 用于高速数据传输或通道处理。
- 外部中断模式（irq）： 用于通用的中断处理。
- 管理模式（svc）： 操作系统使用的保护模式。
- 数据访问终止模式(abt)： 当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护。
- 系统模式（sys）： 运行具有特权的操作系统任务。
- 未定义指令中止模式（und）： 当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真。

处理器模式

- 特权模式

处理器模式	说明	备注
用户 (usr)	正常程序工作模式	不能直接切换到其它模式
系统 (sys)	用于支持操作系统的特权任务等	与用户模式类似，但具有可以直接切换到其它模式等特权
快中断 (fiq)	支持高	
中断 (irq)	用于通	
管理 (svc)	操作系	
中止 (abt)	用于支护	
未定义 (und)	支持硬	

除用户模式外，其它模式均为特权模式。ARM内部寄存器和一些片内外设在硬件设计上只允许（或者可选为只允许）特权模式下访问。此外，特权模式可以自由的切换处理器模式，而用户模式不能直接切换到别的模式。

处理器模式

- 异常模式

处理器模式	说明	备注
用户 (usr)	正常程序工作模式	不能直接切换到其它模式
系统 (sys)	用于支持操作系统的特权任务等	与用户模式类似，但具有可以直接切换到其它模式等特权
快中断 (fiq)		
快中断 (fiq)		
中断 (irq)		
管理 (svc)		
中止 (abt)		
未定义 (und)		

这五种模式称为异常模式。它们除了可以通过程序切换进入外，也可以由特定的异常进入。当特定的异常出现时，处理器进入相应的模式。每种异常模式都有一些独立的寄存器，以避免异常退出时用户模式的状态不可靠。

处理器模式

- 用户和系统模式

处理器模式	说明	备注
用户 (usr)	正常程序工作模式	不能直接切换到其它模式
	用于支持操作系统的特权任务等	与用户模式类似，但具有可以直接切
中断 (irq)	用	
管理 (svc)	操	
中止 (abt)	用 器	
未定义 (und)	支	

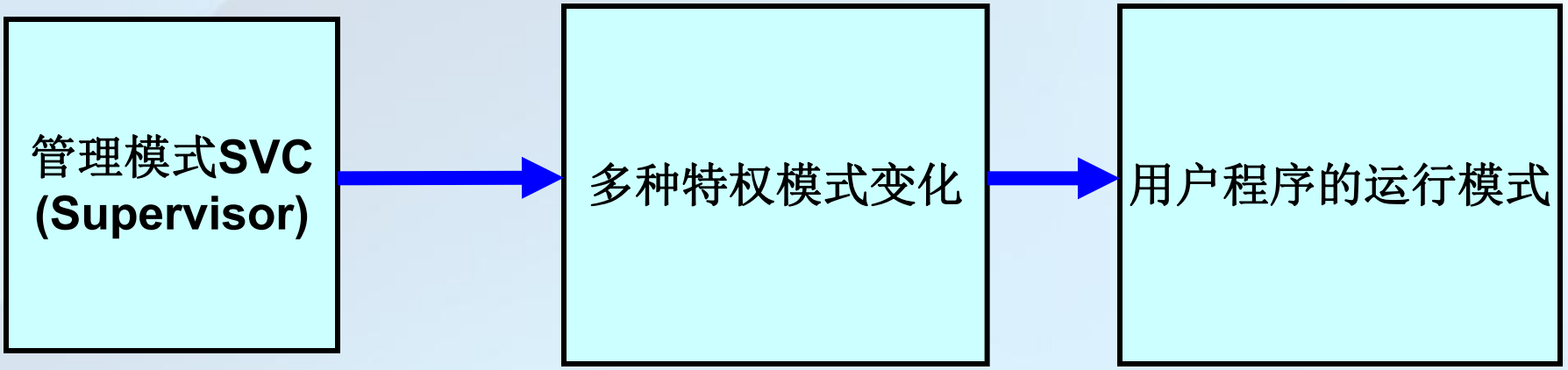
用户 (usr)

系统 (sys)

这两种模式都不能由异常进入，而且它们使用完全相同的寄存器组。系统模式是特权模式，不受用户模式的限制。操作系统在该模式下访问用户模式的寄存器就比较方便，而且操作系统的一些特权任务可以使用这个模式访问一些受控的资源。

处理器模式

处理器启动时的模式转换图



复位后
缺省模式

主要完成各
模式的堆栈
设置，

一般为用户
模式User

寄存器组织

- 37个寄存器
  - 31个32位通用寄存器
  - 6个32位状态寄存器
- ARM微处理器中的寄存器不能被同时访问，具体哪些寄存器是可以访问的，取决处理器的运行模式。

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称						
通用寄存器和程序计数器	R0(a1)	R0					
	R1(a2)	R1					
	R2(a3)	R2					
	R3(a4)	R3					
	R4(v1)	R4					
		R5					
		R6					
		R7					
		R8					R8_fiq
		R9					R9_fiq
		R10					R10_fiq
		R11					R11_fiq
	R11(FP,v8)	R12					R12_fiq
	R12(IP)	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R13(SP)	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R14(LR)	R15					
	R15(PC)						
状态寄存器	CPSR	CPSR					
	SPSR	无	SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

所有的37个寄存器，分成两大类：

- 31个通用32位寄存器；
- 6个状态寄存器。

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)	R5						
	R6(v3)	R6						
	R7(v4)	R7						
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM状态下的寄存器组织

1.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)。

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄程序计	R0(a1)	R0						
		R1						
		R2						
		R3						
		R4						
		R5						
		R6						
		R7						
		R8						R8_fiq
		R9						R9_fiq
		R10						R10_fiq
		R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

在汇编语言中寄存器R0～R13为保存数据或地址值的通用寄存器。它们是完全通用的寄存器，不会被体系结构作为特殊用途，并且可用于任何使用通用寄存器的指令。

ARM状态下的寄存器组织

2.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，因此，在中断或异常处理进行运行模式转换时，由于不同的处理器运行模式均使用相同的物理寄存器，可能会造成寄存器中数据的破坏，这一点在进行程序设计时应引起注意。

ARM状态下的寄存器组织

其中R0～R7为未分组的寄存器，也就是说对于任何处理器模式，这些寄存器都对应于相同的32位物理寄存器。

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器					
通用寄存器、程序计数器	R0	R0					
	R1	R1					
	R2	R2					
	R3	R3					
	R4	R4					
	R5	R5					
	R6	R6					
	R7	R7					
	R8(v5)	R8					R8_fiq
	R9(SB,v6)	R9					R9_fiq
	R10(SL,v7)	R10					R10_fiq
	R11(FP,v8)	R11					R11_fiq
	R12(IP)	R12					R12_fiq
	R13(SP)	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15					
状态寄存器	CPSR	CPSR					
	SPSR	无	SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM状态下的寄存器组织

3.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。

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。

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
		R4						
		R5						
		R6						
		R7						
		R8						R8_fiq
		R9						R9_fiq
		R10						R10_fiq
		R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寄存器R8～R14为分组寄存器。它们所对应的物理寄存器取决于当前的处理器模式，几乎所有允许使用通用寄存器的指令都允许使用分组寄存器

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)							
	R1(a2)							
	R2(a3)							
	R3(a4)							
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)	R8						R8_fiq
	R9(SB,v6)	R9						R9_fiq
	R10(SL,v7)	R10						R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寄存器R8～R12有两个分组的物理寄存器。一个用于除FIQ模式之外的所有寄存器模式，另一个用于FIQ模式。这样在发生FIQ中断后，可以加速FIQ的处理速度。

ARM状态下的寄存器组织

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)	R4						
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							R8_fiq
	R9(SB,v6)							R9_fiq
	R10(SL,v7)							R10_fiq
	R11(FP,v8)	R11						R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寄存器R13、R14分别有6个分组的物理寄存器。一个用于用户和系统模式，其余5个分别用于5种异常模式。

ARM状态下的寄存器组织

- 寄存器R13在ARM指令中常用作堆栈指针，但这只是一种习惯用法，用户也可使用其他的寄存器作为堆栈指针。
- 而在Thumb指令集中，某些指令强制性的要求使用R13作为堆栈指针。

堆栈指针寄存器R13（SP）

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							R8_fiq
	R9(SB,v6)							R9_fiq
	R10(SL,v7)							R10_fiq
	R11(FP,v8)							R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)							
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

寄存器R13常作为堆栈指针（SP）。在ARM指令集当中，没有以特殊方式使用R13的指令或其它功能，只是习惯上都这样使用。但是在Thumb指令集中存在使用R13的指令。

ARM状态下的寄存器组织

- R14也称作子程序链接寄存器（Subroutine Link Register）或链接寄存器LR。
- 当执行BL子程序调用指令时，R14中得到R15（程序计数器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。

链接寄存器R14（LR）

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)							
	R3(a4)							
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							R8_fiq
	R9(SB,v6)							R9_fiq
	R10(SL,v7)							R10_fiq
	R11(FP,v8)							R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)							
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R14为链接寄存器（LR），在结构上有两个特殊功能：

- **在每种模式下，模式自身的R14版本用于保存子程序返回地址；**
- **当发生异常时，将R14对应的异常模式版本设置为异常返回地址（有些异常有一个小的固定偏移量）。**

程序计数器R15（PC）

寄存器类别	寄存器在汇编中的名称	各模式下实际访问的寄存器						
		用户	系统	管理	中止	未定义	中断	快中断
通用寄存器和程序计数器	R0(a1)	R0						
	R1(a2)	R1						
	R2(a3)	R2						
	R3(a4)	R3						
	R4(v1)							
	R5(v2)							
	R6(v3)							
	R7(v4)							
	R8(v5)							R8_fiq
	R9(SB,v6)							R9_fiq
	R10(SL,v7)							R10_fiq
	R11(FP,v8)							R11_fiq
	R12(IP)	R12						R12_fiq
	R13(SP)	R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
	R14(LR)	R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
	R15(PC)	R15						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_abt	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

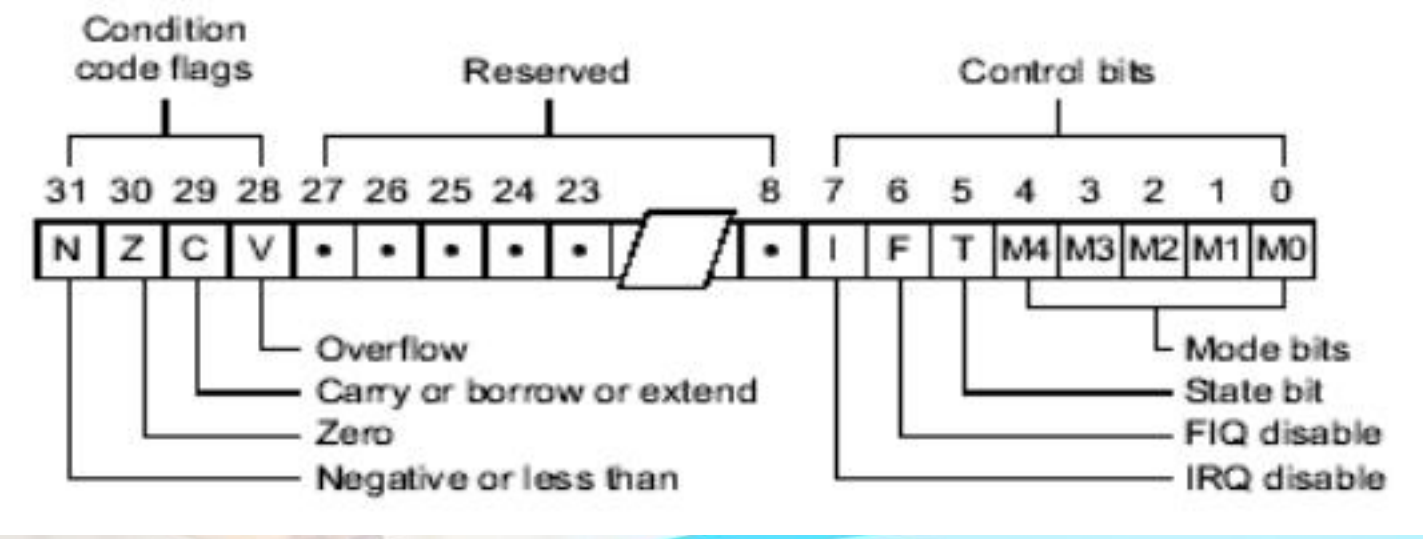
寄存器R15常作为程序计数器(PC)。R15虽然也可用作通用寄存器，但一般不这么使用，因为对R15的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。

程序状态寄存器 – 1

- ARM 核包含当前程序状态寄存器（CPSR），加上5个程序状态保存寄存器SPSR，当异常发生时，用于保存CPSR的状态
- 这些寄存器的功能是：
  - 包括关于最近执行的ALU操作的信息
  - 控制中断的使能和禁止
  - 设置处理器操作模式

程序状态寄存器 – 2

- N, Z, C and V 条件码标志
  - 可以在处理器中作为数学和逻辑操作改变
  - 可以被所有的指令测试，以决定指令是否被执行
  - N : Negative. Z : Zero. C : Carry. V : Overflow
- I and F 位是中断禁止位
- M0, M1, M2, M3 and M4 位是模式位



程序状态寄存器PSR的模式位

M[4:0]	Mode	Visible Thumb-state registers	Visible ARM-state registers
10000	User	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR
10001	FIQ	r0–r7, SP_fiq, LR_fiq, PC, CPSR, SPSR_fiq	r0–r7, r8_fiq–r14_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	r0–r7, SP_irq, LR_irq, PC, CPSR, SPSR_irq	r0–r12, r13_irq, r14_irq, PC, CPSR, SPSR_irq
10011	Supervisor	r0–r7, SP_svc, LR_svc, PC, CPSR, SPSR_svc	r0–r12, r13_svc, r14_svc, PC, CPSR, SPSR_svc
10111	Abort	r0–r7, SP_abt, LR_abt, PC, CPSR, SPSR_abt	r0–r12, r13_abt, r14_abt, PC, CPSR, SPSR_abt
11011	Undefined	r0–r7, SP_und, LR_und, PC, CPSR, SPSR_und	r0–r12, r13_und, r14_und, PC, CPSR, SPSR_und
11111	System	r0–r7, SP, LR, PC, CPSR	r0–r14, PC, CPSR

异常 - 1

- 异常——内部或外部中断源产生并引起处理器处理一个事件。
  - 处理异常之前必须保留处理器的状态
- 异常类型
  - FIQ
  - IRQ(Interrupt ReQuest)
  - 未定义指令
  - 预取中止
  - 数据中止
  - 复位
  - 软件中断Software interrupt
    - 通过软件中断产生
    - 进行管理员模式中获得
    - 通常要求特殊的管理功能，如操作系统支持

异常 - 2

- 未定义的指令陷井
  - 当ARM接受到一条不能处理的指令,ARM把这条指令提供给任何一个协处理器执行
  - 如果协处理器可以执行这条指令但此时协处理器忙,ARM将等待直到协处理器准备好或中断发生
  - 如果没有协处理器处理这条指令,那么ARM将处理未定义的指令陷井

- 异常优先级
  - (1) Reset (highest priority)
  - (2) Data abort
  - (3) FIQ
  - (4) IRQ
  - (5) Prefetch abort
  - (6) 未定义指令, Software interrupt (最低优先级)

异常 – 3

- 只要产生异常就会导致正常的程序流程被临时停止, 例如外围中断。
- 在异常被处理前, 当前的处理器状态必须被保存, 以便处理程序完成后, 最后的程序可以被恢复。

异常向量

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	*Reserved*	*Reserved*
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

进入异常的操作

- 在相应的链接寄存器LR（r14)中保存下一条指令的地址
- 将CPSR复制到相应的SPSR中
- 强制使CPSR模式位置成对应异常类型的值
- 强制使程序计数器指向相应异常向量,取下一条指令

例子：用户模式到 FIQ模式*

退出异常的操作

- 将LR寄存器中的值减去相应的偏移量送到PC中

  Linux：SWI、UND不变

  IRQ、FIQ、取址异常减4

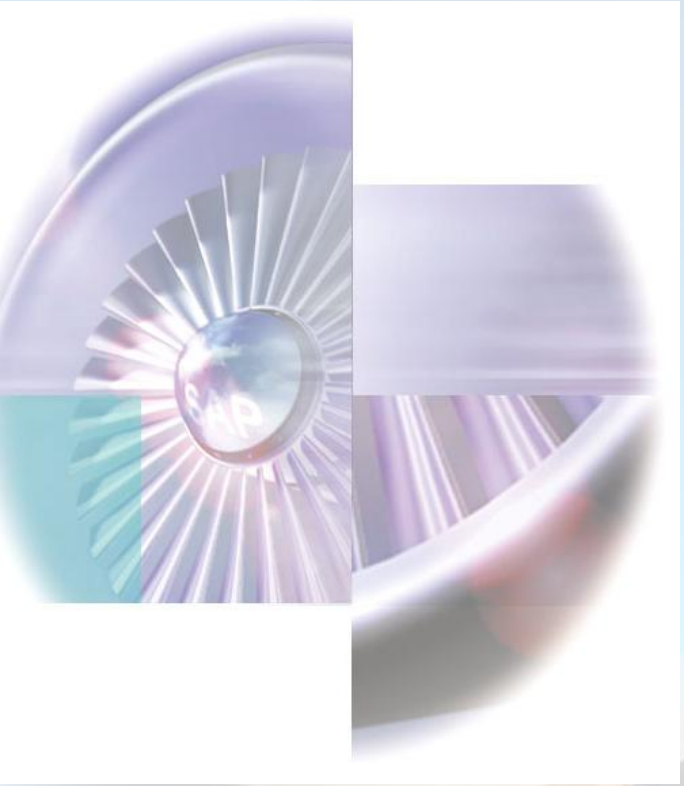
  数据异常减8

- 将 SPSR 复制回 CPSR

- 清除禁止中断标志，如果它被设置成禁止

进入/退出异常概述

	Return Instruction	Previous State	
		ARM R14_x	THUMB R14_x
BL	MOV PC, R14	PC + 4	PC + 2
SWI	MOVS PC, R14_svc	PC + 4	PC + 2
UDEF	MOVS PC, R14_und	PC + 4	PC + 2
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8
RESET	NA	-	-



ARM指令集

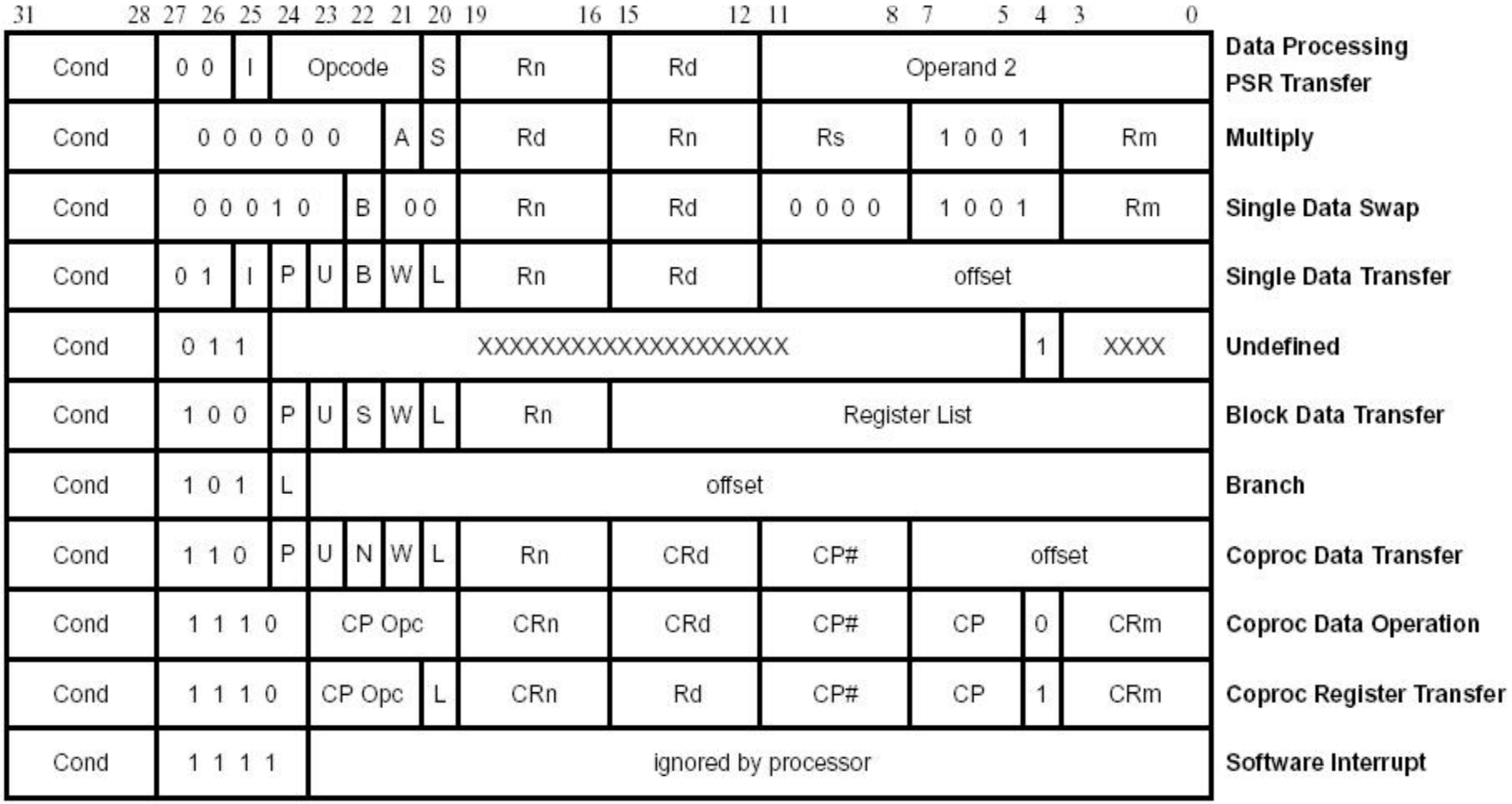
指令长度

- 指令集可以是以下任一种
  - 32 bits 长 (ARM状态)
  - 16 bits 长 (Thumb 状态)
- ARM（v4） 支持3种数据类型
  - 字节 (8-bit)
  - 半字 (16-bit)
  - 字 (32-bit)
- 字必须被排成4个字节边界对齐,半字必须被排列成2个字节边界对齐

ARM 指令集

- Load-store 结构
  - load -从存储器中读某个值到寄存器;
  - store –把寄存器的值写回到存储器中;
- 指令分类
  - 数据处理指令 – 使用和改变寄存器的值
  - 数据传送指令 – 把存储器的值拷贝到寄存器中(load) or 把寄存器中的值拷贝到存储器中(store)
  - 控制流指令
    - 分支
    - 分支和链接, 保存返回的地址,以恢复最先的次序
    - 陷入系统代码

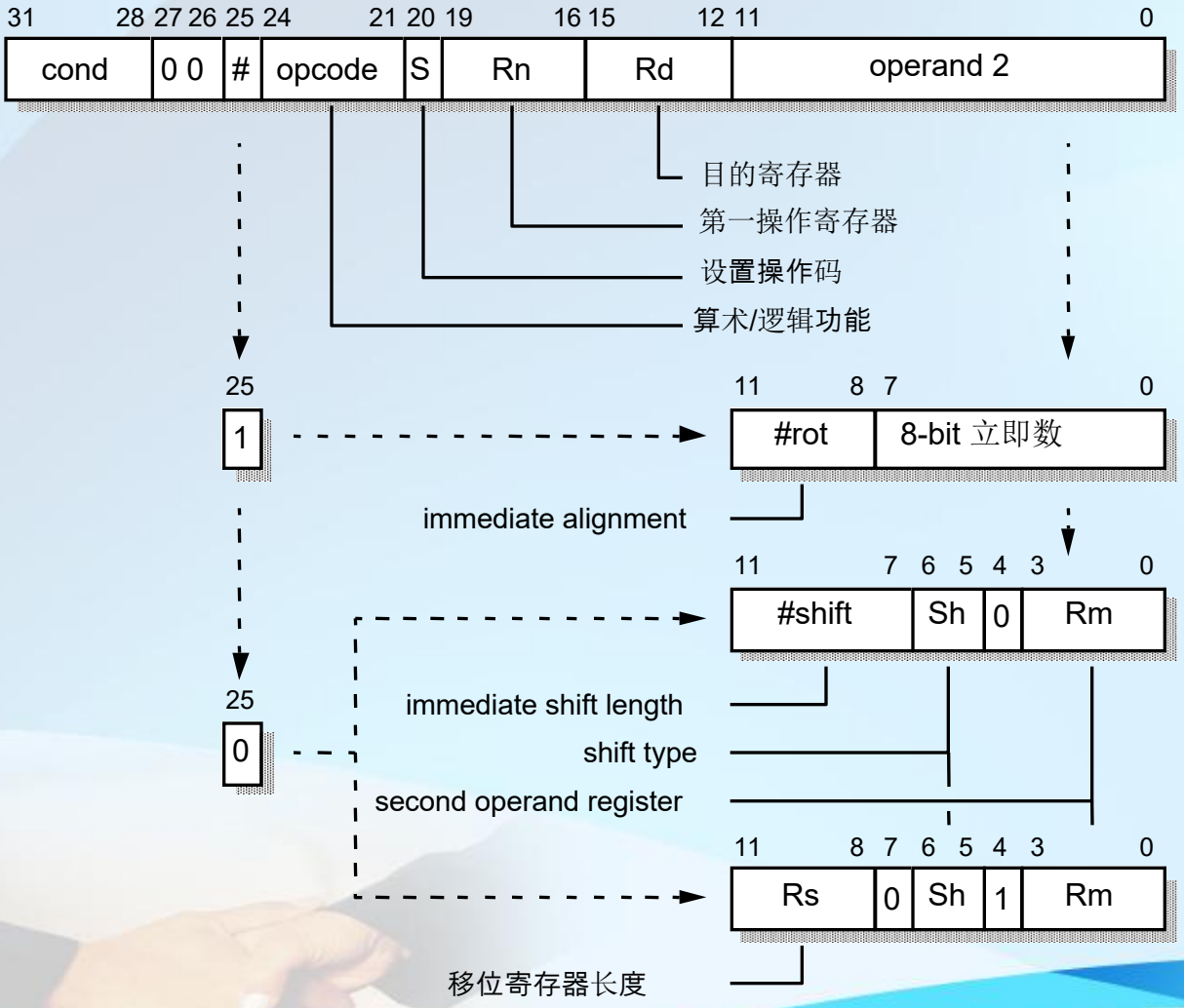
ARM指令集编码*


31 28 27 26 25 24 23 22 21 20 19 16 15 12 11 8 7 5 4 3 0
Cond 0 0 I Opcode S Rn Rd Operand 2 Data Processing PSR Transfer
Cond 0 0 0 0 0 0 0 A S Rd Rn Rs 1 0 0 1 Rm Multiply
Cond 0 0 0 1 0 B 0 0 Rn Rd 0 0 0 0 1 0 0 1 Rm Single Data Swap
Cond 0 1 I P U B W L Rn Rd offset Single Data Transfer
Cond 0 1 1 XXXXXXXXXXXXXXXXXXX 1 XXXX Undefined
Cond 1 0 0 P U S W L Rn Register List Block Data Transfer
Cond 1 0 1 L offset Branch
Cond 1 1 0 P U N W L Rn CRd CP# offset Coproc Data Transfer
Cond 1 1 1 0 CP Opc CRn CRd CP# CP 0 CRm Coproc Data Operation
Cond 1 1 1 0 CP Opc L CRn Rd CP# CP 1 CRm Coproc Register Transfer
Cond 1 1 1 1 ignored by processor Software Interrupt

数据处理指令 – 1

- 数据处理指令的类别
  - 算术操作
  - 按位逻辑操作
  - 寄存器移位操作
  - 比较操作
- 操作数：32-bits 宽；3种指定操作数的方式
  - 来自寄存器
  - 第二操作数可以是常数(立即数)
  - 移位寄存器操作数
- 结果：32-bits 宽，放在寄存器中
  - 长乘法产生64位结果

数据处理指令 – 2*


31 28 27 26 25 24 21 20 19 16 15 12 11 0
cond
0 0
#
opcode
S
Rn
Rd
operand 2
目的寄存器
第一操作寄存器
设置操作码
算术/逻辑功能
25
1
11 8 7 0
#rot
8-bit 立即数
immediate alignment
11 7 6 5 4 3 0
#shift
Sh
0
Rm
25
0
immediate shift length
shift type
second operand register
11 8 7 6 5 4 3 0
Rs
0
Sh
1
Rm
移位寄存器长度

数据处理指令 – 3

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

数据处理指令 – 4

算术操作

ADD r0, r1, r2	r0 := r1 + r2
ADC r0, r1, r2	r0 := r1 + r2 + C
SUB r0, r1, r2	r0 := r1 - r2
SBC r0, r1, r2	r0 := r1 - r2 + C - 1
RSB r0, r1, r2	r0 := r2 – r1
RSC r0, r1, r2	r0 := r2 – r1 + C - 1

按位逻辑操作

AND r0, r1, r2	r0 := r1 and r2
ORR r0, r1, r2	r0 := r1 or r2
EOR r0, r1, r2	r0 := r1 xor r2
BIC r0, r1, r2	r0 := r1 and (not) r2

寄存器移位

MOV r0, r2	r0 := r2
MVN r0, r2	r0 := not r2

比较操作

CMP r1, r2	set cc on r1 - r2
CMN r1, r2	set cc on r1 + r2
TST r1, r2	set cc on r1 and r2
TEQ r1, r2	set cc on r1 xor r2

数据处理指令 – 5

- 立即数操作:
  立即数操作 = (0->255) x 22n, 0 <= n <= 12

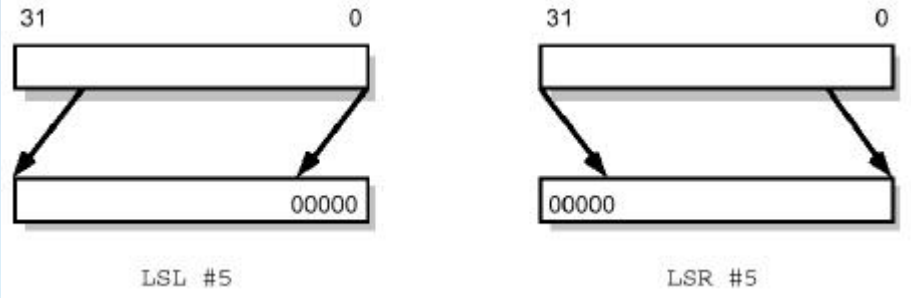
```
ADD r3, r3, #3          r3 := r3 + 3
AND r8, r7, #&ff        r8 := r7[7:0], & for hex
```

- 移位寄存器操作数
  - 第二个操作数在与第一个操作数合成之前, 是服从于移位操作的.

```
ADD r3, r2, r1, LSL #3      r3 := r2 + 8 x r1
ADD r5, r5, r3, LSL r2      r5 := r5 + 2^r2 x r3
```

数据处理指令 - 6

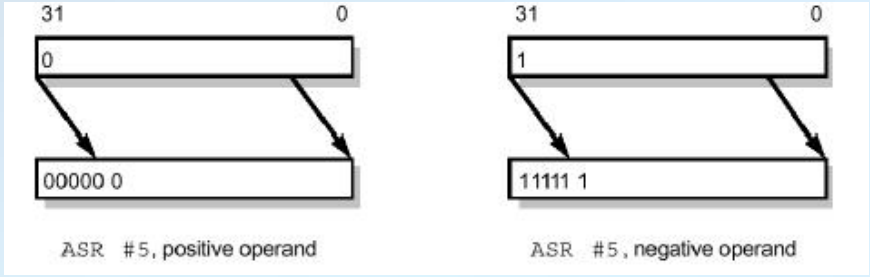
- 移位操作
  - 在任何数据处理指令中,第二个寄存器操作数可以有应用该操作数的移位操作.
  - 逻辑移位

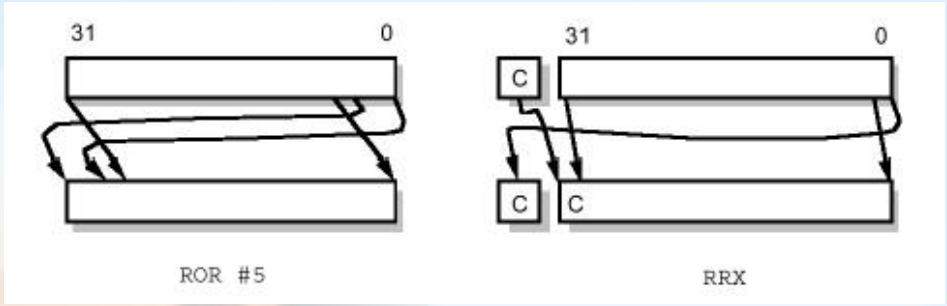


  - LSL:逻辑左移
    - 字的最小位空位清零
  - LSR:逻辑右移字的最大位空位清零.

数据处理指令 – 7

- 算术移位
  - ASR: = LSR
  - ASL: 算术左移
- 循环移位: ROR, RRX





条件码标志

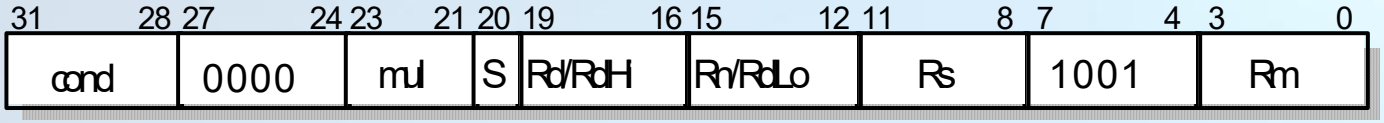
- 任何数据处理指令都可以设置条件码（N, Z, V, and C)
  - 适用于除比较操作外的所有数据处理指令
  - 特殊的请求必须在汇编语言中实现,这种请求是通过把"S"增加到选择代码中指定的

```
ADDS r2, r2, r0      ; carry out to C
```

- 算术操作设置所有的标志位（N, Z, C, and V）
- 逻辑和移位操作设置 N and Z

乘法指令集

- 在寄存器产生32位值



Opcode [23:21]	Mnemonic	Meaning	Effect
000	MUL	Multiply (32-bit result)	Rd := (Rm * Rs) [31:0]
001	MLA	Multiply-accumulate (32-bit result)	Rd := (Rm * Rs + Rn) [31:0]
100	UMULL	Unsigned multiply long	RdHi:RdLo := Rm * Rs
101	UMLAL	Unsigned multiply-accumulate long	RdHi:RdLo += Rm * Rs
110	SMULL	Signed multiply long	RdHi:RdLo := Rm * Rs
111	SMLAL	Signed multiply-accumulate long	RdHi:RdLo += Rm * Rs

乘法

- 例子（乘法，乘法累加器）

  MUL r4, r3, r2　　r4 := [r3 x r2]$_{<31:0>}$

  MLA r4, r3, r2, r1　　r4 := [r3 x r2 + r1] $_{<31:0>}$

- 注意
  - 最低 32-bits 置于结果寄存器中,其余被忽略
  - 不支持第二立即操作数
  - 结果寄存器与源寄存器必须不同
  - if `S` bit is set the V is preserved and the C is rendered meaningless

数据传送指令 - 1

- 单数据传送 (LDR, STR)
  - 单字(32bit), 半字(6 bit) 以及字节(8 bit) 传送
  - 寻址
    - 寄存器偏移
      - 地址 =基址 ± 寄存器偏移
    - 立即数偏移
      - 地址 = 基址 ± 立即数常数
    - 后变址Post-indexing: modify address after use
    - 前变址Pre-indexing: modify address before use
  - 回写
    - 如果可能,更新基址寄存器

数据传送指令 – 2

- 多数据传送指令（LDM, STM）
  - load（LDM）或 store（STM）当前可见寄存器的任意子集
  - 使用
    - 堆栈: maintaining full or empty stacks which can grow up or down memory
    - 上下文切换: 保存或重新存储工作寄存器
    - 块拷贝:在主存储器中移动大数据块
  - 寻址
    - Pre/Post indexing
    - Auto increment or decrement
    - 回写到基址寄存器Write back the base register

数据传送指令 – 3

- 单数据交换（SWAP）
  - 在寄存器和外部存储器之间交换字节或字
  - 读存储器和写存储器是是放在一起的
    - 原子指令
    - 执行时不能中断
    - 当‘LOCK’信号输出操作时，外部存储器管理单元被锁定，当
    - 多线程操作时使程序同步(OS支持)
    - 锁定
    - 信号量

数据传送指令 – 4

单寄存器 load and store

寄存器间接寻址

LDR r0, [r1]　　r0 := mem_{32}[r1]

STR r0, [r1]　　mem_{32}[r1] := r0

Note: r1 keeps a word address (2 LSBs are 0)

LDRB r0, [r1]　　r0 := mem_{8}[r1]

Note: no restrictions for r1

基址+偏移量寻址
(offset of up to 4Kbytes)

LDR r0, [r1, #4]　　r0 := mem_{32}[r1 +4]

自动变址寻址

LDR r0, [r1, #4]!　　r0 := mem_{32}[r1 + 4]
r1 := r1 + 4

后变址寻址

LDR r0, [r1], #4　　r0 := mem_{32}[r1]
r1 := r1 + 4

数据传送指令 – 5

```
COPY:       ADR r1, TABLE1        ; r1 points to TABLE1
ADR r2, TABLE2        ; r2 points to TABLE2
LOOP:       LDR r0, [r1]
STR r0, [r2]
ADD r1, r1, #4
ADD r2, r2, #4
...
TABLE1:     ...
TABLE2:...
```

```
COPY:       ADR r1, TABLE1        ; r1 points to TABLE1
ADR r2, TABLE2        ; r2 points to TABLE2
LOOP:       LDR r0, [r1], #4
STR r0, [r2], #4
...
TABLE1:     ...
TABLE2:...
```

数据传送指令 – 6

多寄存器数据传送 [Increment After]

LDMIA r1, {r0, r2, r5}

r0 := mem_{32}[r1]
r2 := mem_{32}[r1 + 4]
r5 := mem_{32}[r1 + 8]

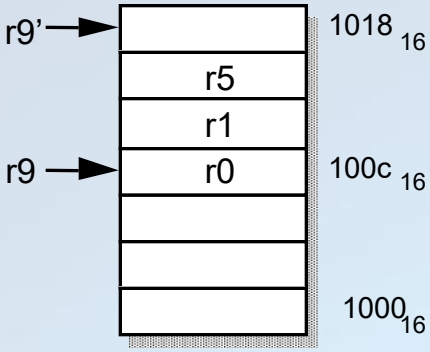
Note: 寄存器的部分或全部都可以用单指令传送

Note: 在表中的寄存器顺序并不重要

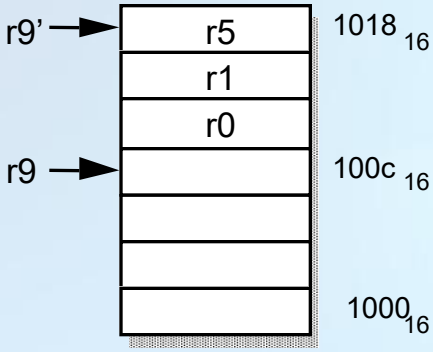
Note: 在表中包括 r15 i将造成控制流的改变

- 块拷贝
  - 数据被存贮在基本寄存器的上面地址或下面地址
  - 地址增加或减少是在存贮第一个值之前或之后开始的

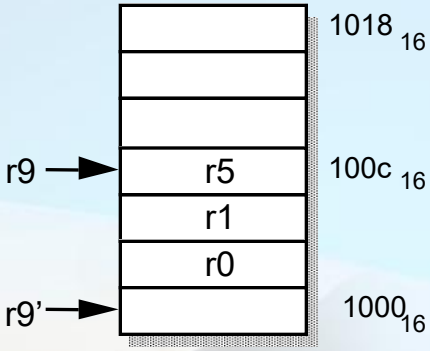
多寄存器传送寻址模式



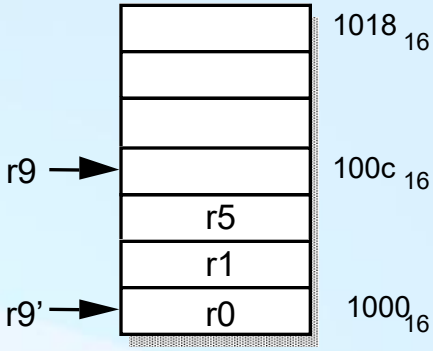
STMIA r9!, {r0,r1,r5}



STMIB r9!, {r0,r1,r5}



STMDA r9!, {r0,r1,r5}



STMDB r9!, {r0,r1,r5}

条件执行

- 所有的ARM指令都可以条件执行
- 指令的执行与否取决于CPSR寄存器的N, Z, C and V标志位
- 所有的Thumb指令都可以解压成全部条件指令
- Condition Field in instruction

31	27	0
Cond		

- 0000 = EQ - Z set (equal)
- 0001 = NE - Z clear (not equal)
- 0010 = CS - C set (unsigned higher or same)
- 0011 = CC - C clear (unsigned lower)
- 0100 = MI - N set (negative)
- 0101 = PL - N clear (positive or zero)
- 0110 = VS - V set (overflow)
- 0111 = VC - V clear (no overflow)
- 1000 = HI - C set and Z clear (unsigned higher)
- 1001 = LS - C clear or Z set (unsigned lower or same)
- 1010 = GE - N set and V set, or N clear and V clear (greater or equal)
- 1011 = LT - N set and V clear, or N clear and V set (less than)
- 1100 = GT - Z clear, and either N set and V set, or N clear and V clear (greater than)
- 1101 = LE - Z set, or N set and V clear, or N clear and V set (less than or equal)
- 1110 = AL - always
- 1111 = NV - never

控制流指令

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

条件执行

- 条件执行避免使用分支指令
- Example

```
CMP r0, #5;
BEQ BYPASS          ; if (r0!=5) {
ADD r1, r1, r0      ;     r1:=r1+r0-r2
SUB r1, r1, r2      ; }
BYPASS:   ...
```

使用条件执行

```
CMP r0, #5;
ADDNE r1, r1, r0    ;
SUBNE r1, r1, r2    ;
...
```

Note: add 2 –letter condition after the 3-letter opcode

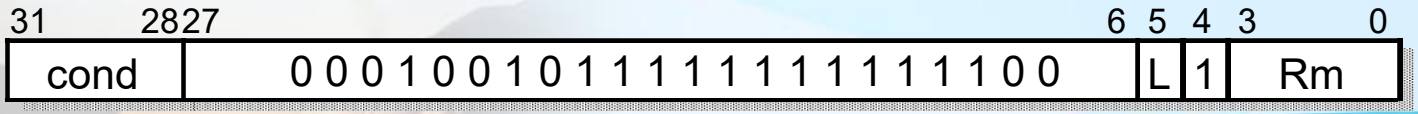
```
; if ((a==b) && (c==d)) e++;

CMP r0, r1
CMPEQ r2, r3
ADDEQ r4, r4, #1
```

控制和分支指令

- 控制指令
  - 分支和分支连接
    - 跳到希望的指令中
    - 保存当前的PC并返回 (with ‘L’ bit)

  - 分支和交换
    - 跳到期望的指令中与指令集交换
      - Rm[0] == 1: Subsequent inst. are THUMB.
      - Rm[0] == 0: Subsequent inst. are ARM.



分支和链接指令

- 分支子程序（r14 serves as a link register）

```
BL SUBR    ; branch to SUBR
..         ; return here


SUBR:      ..          ; SUBR entry point
MOV pc, r14 ; return
```

Full Descending

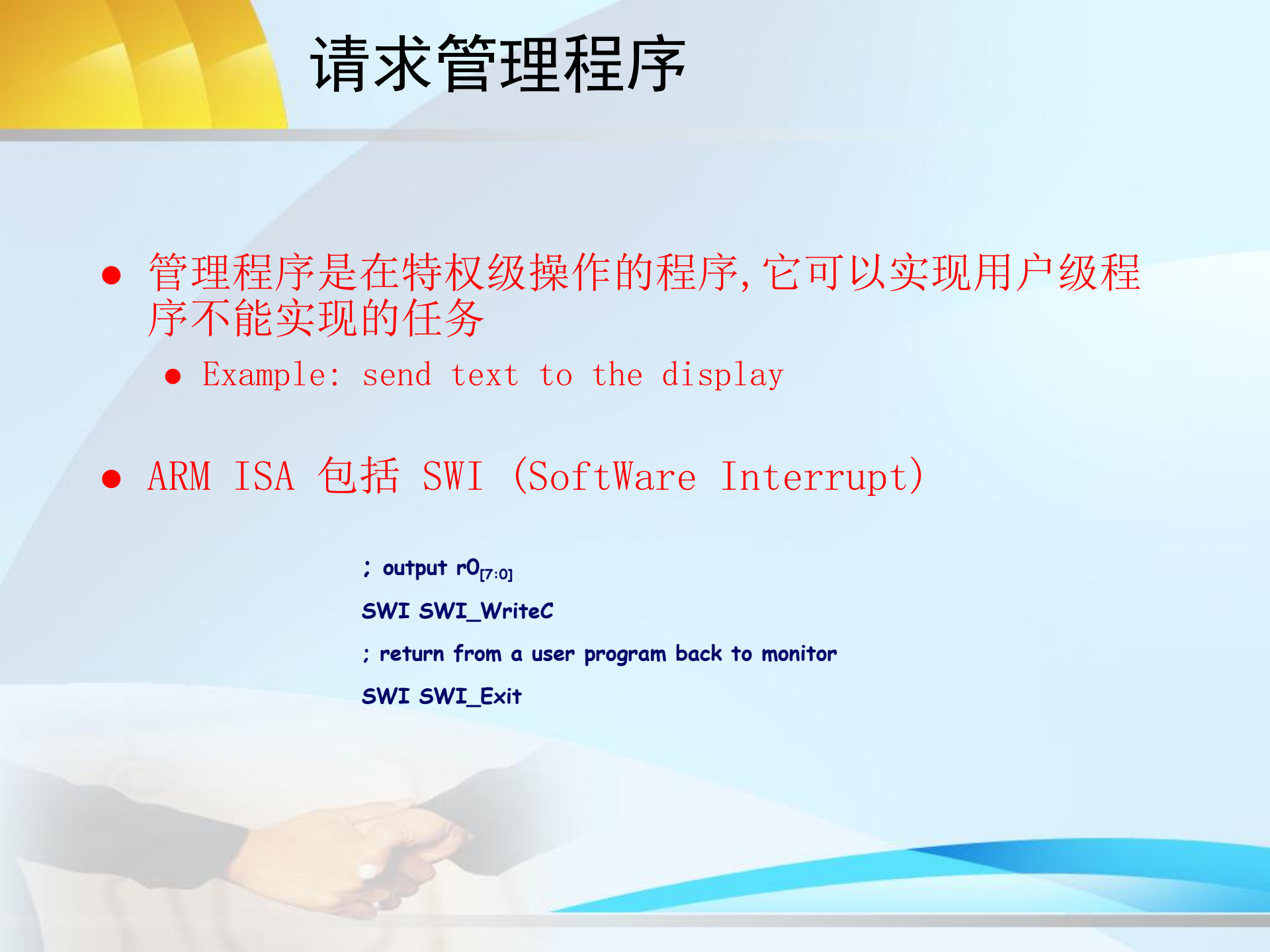
- 嵌套子程序

```
BL SUB1
           ..
SUB1:      ; save work and link register
STMFD r13!, {r0-r2,r14}
BL SUB2
..
LDMFD r13!, {r0-r2,pc}
SUB2:      ..
MOV pc, r14 ; copy r14 into r15
```

请求管理程序

- 管理程序是在特权级操作的程序,它可以实现用户级程序不能实现的任务
  - Example: send text to the display
- ARM ISA 包括 SWI (SoftWare Interrupt)

```
; output r0[7:0]
SWI SWI_WriteC
; return from a user program back to monitor
SWI SWI_Exit
```



转移表

- 根据程序计算值调用一个子程序

```
BL JTAB
...
JTAB:       CMP r0, #0
BEQ SUB0
CMP r0, #1
BEQ SUB1
CMP r0, #2
BEQ SUB2
```

Note: slow when the list is long, and all subroutines are equally frequent

```
BL JTAB
...
JTAB:       ADR r1, SUBTAB
CMP r0, #SUBMAX ; overrun?
LDRLS pc, [r1, r0, LSL #2]
B ERROR
SUBTAB:    DCD SUB0
DCD SUB1
DCD SUB2
...
```

Example:Hello ARM World!

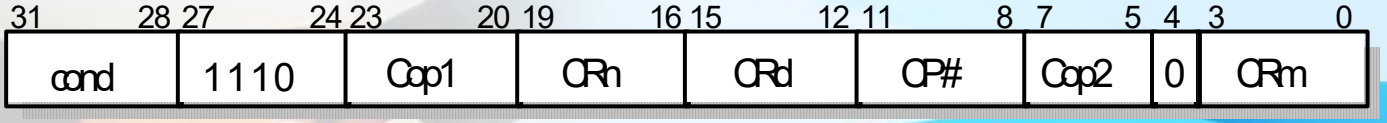
```
                AREA    HelloW,CODE,READONLY  ;声明代码区
SWI_WriteC      EQU  &0                       ；输出r0中的字符
SWI_Exit        EQU  &11                      ；程序结束
                ENTRY                         ；代码入口
START       ADR     r1,TEXT                   ；r1---“Hello World”
LOOP        LDRB    r0,[r1],#1                ；读取下一字节
            CMP     r0,#0                     ；检查文本终点
            SWINE   SWI_WriteC                ；若非终点，则打印
            BNE     LOOP                      ；并返回LOOP
            SWI     SWI_Exit                  ；执行结束
TEXT        =       “Hello World”,&0a,&0d,0
            END                               ；程序结束
```

PSR 指令

- PSR 指令(MRS, MSR)
  - The MRS and MSR 指令是从数据处理指令子集形成的.instructions are formed from a subset of the Data Processing operations
  - 这些指令允许访问 CPSR and SPSR 寄存器:
    - The MRS 指令允许把 CPSR or SPSR_<mode>寄存器中的内容移到通用寄存器中
    - The MSR 指令允许把通用寄存器中的内容移到CPSR or SPSR_<mode> 寄存器中

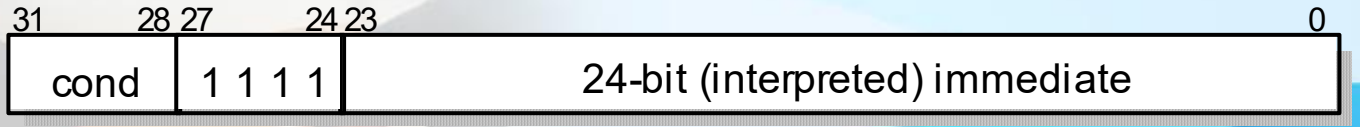
协处理器指令 – 1

- 协处理器
  - 一般原理是通过增加核扩展指令集
  - Example ： 如 MMU & cache. FPU等系统控制器
  - 寄存器
    - 协处理器专用
    - ARM 控制数据流
      - 协处理器只包含数据处理和存贮器传送操作
- 协处理器数据操作 (CDP)
  - 这类指令是用来告诉协处理器执行某些内部操作
  - 无结果返回ARM,ARM并不等待操作完成



协处理器指令 – 2

- 协处理器数据传送（LDC, STC）
  - Load（LDC）or store（STC）一个协处理器寄存器的子集直接到存储器
  - ARM is 负责提供存储器地址,协处理器提供或接收大量传送的数据或控制指令
- 协处理器寄存器传送（MRC, MCR）
  - 在ARM和协处理器之间的直接通讯信息
- 软件中断指令（SWI）
  - 在控制方式中用于进入管理员模式
  - 该指令造成软件中断陷刊产生,它会影响模式改变



结束