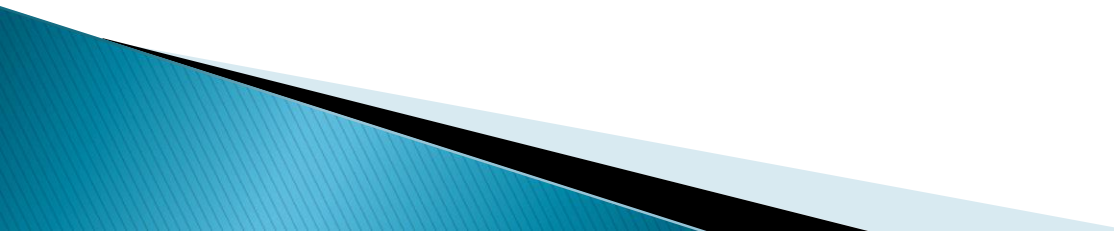


第2章 ARM处理器体系结构



目 录

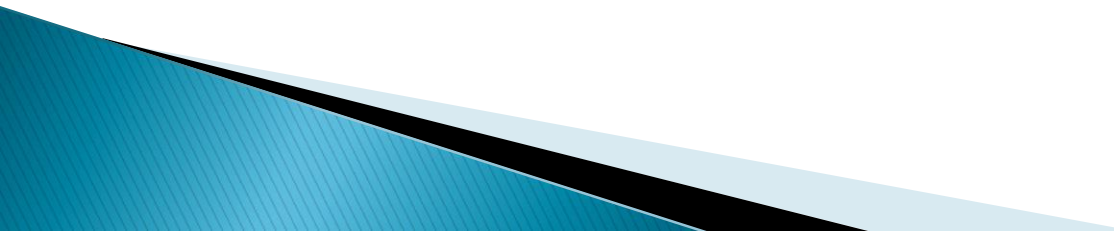
- 2.1 ARM处理器概述
 - 2.2 Cortex-A8处理器架构
 - 2.3 Cortex-A8处理器工作模式和状态
 - 2.4 Cortex-A8存储器管理
 - 2.5 Cortex-A8异常处理
 - 2.6 本章小结
- 

ARM (Advanced RISC Machines) 处理器是一种**RISC** (精简指令集) 结构的高性价比、低功耗处理器，广泛用于各种嵌入式系统设计中。

目前，各种采用**ARM**技术知识产权 (**IP核**) 的**ARM**微处理器，已遍及工业控制、消费类电子产品、通信系统、网络系统、无线系统等各类产品市场。

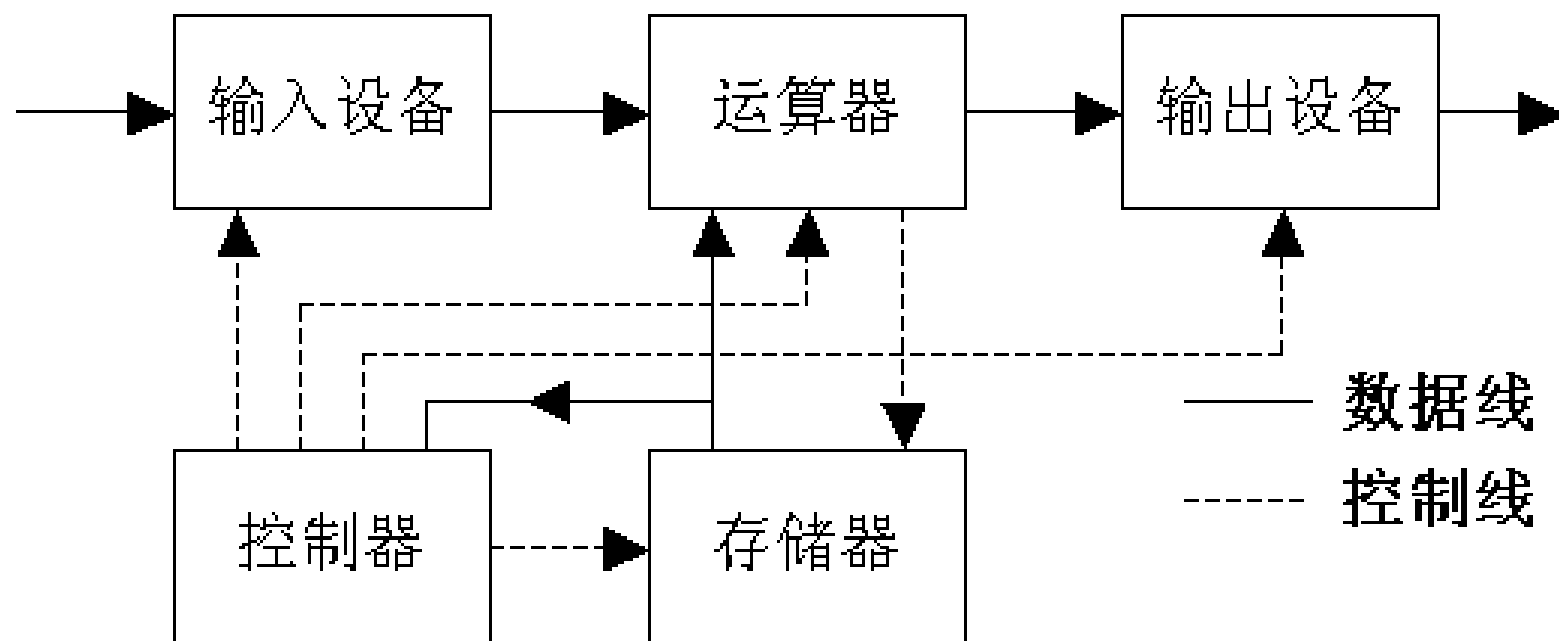
基于**ARM** 技术的微处理器应用约占据了**32**位**RISC** 微处理器**80%**以上的市场份额，**ARM**技术正在逐步渗入到我们生活的各个方面。

本章主要内容：

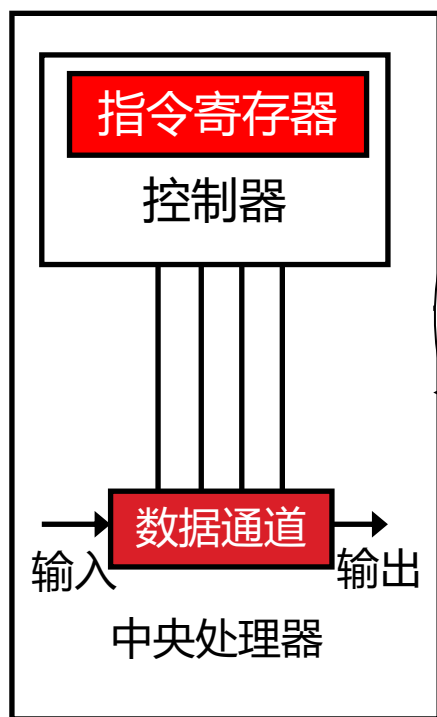
- 1、介绍**ARM体系结构**的不同版本以及比较有代表性的**ARM产品**，并对相关**开发工具**进行了阐述；
 - 2、对**ARM Cortex-A8**处理器的**组成结构、寄存器组织、运行模式和状态**以及**存储管理**方法进行了说明；
 - 3、详细介绍了**ARM Cortex-A8**处理器的**异常处理**原理及过程。
- 

计算机体系结构

- 计算机中，按内存的组成成分两种典型的结构：
 - 冯·诺依曼结构/普林斯顿结构



冯·诺依曼体系结构

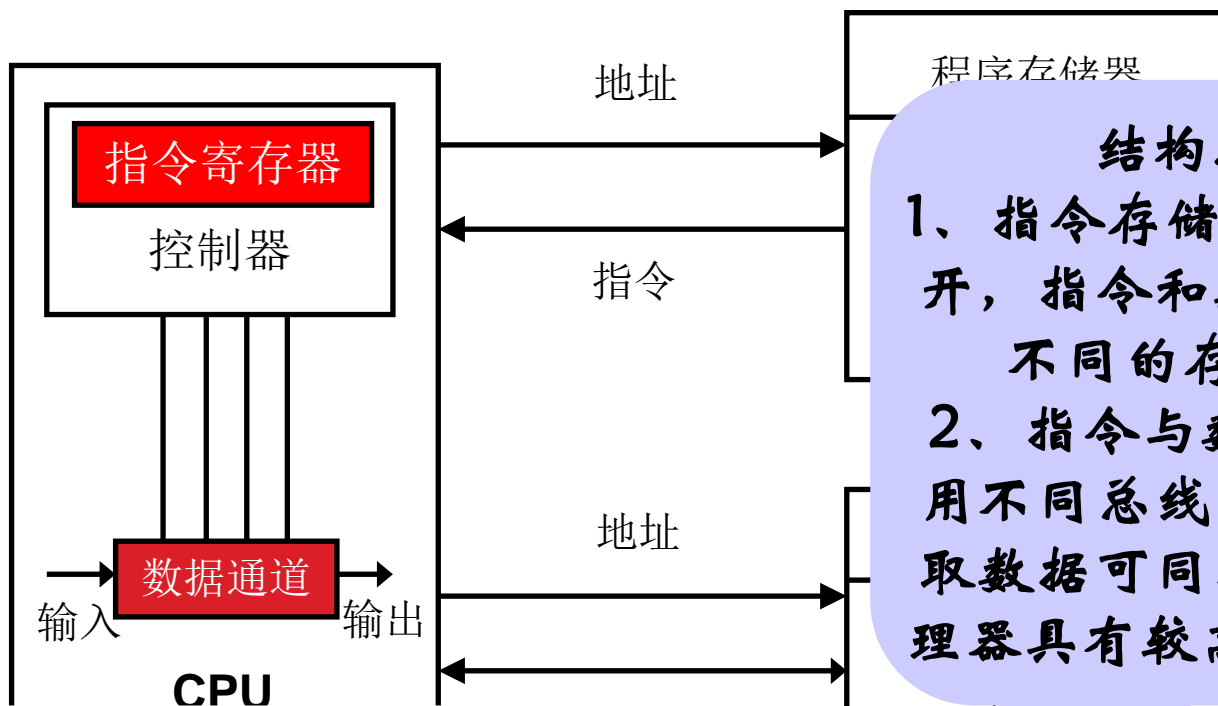


结构特点：1、指令和数据存储在相同的内存空间，但存储地址不同。
2、处理器利用相同的总线处理内存中的指令和数据，指令和数据具有相同的数据宽度，指令与数据无法同时存取。

数据1
数据2

ARM7嵌入式微处理器亦采用此结构

哈佛体系结构



结构特点:

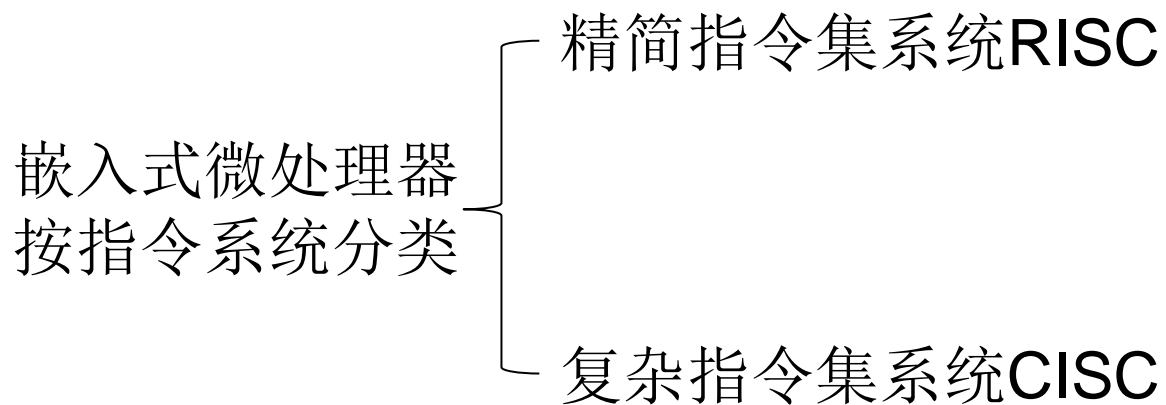
- 1、指令存储和数据存储分开，指令和数据分别位于不同的存储空间。
- 2、指令与数据的存取采用不同总线，取指令和存取数据可同时进行，微处理器具有较高的执行效率。

数字信号处理器DSP通常采用哈佛结构，ARM9嵌入式微处理器亦采用此结构。

2.1 Part One

ARM处理器概述

2.1.1 ARM处理器简介



CISC和RISC

高代码
密度

CISC: 复杂指令集 (Complex Instruction Set Computer)

具有大量的指令和寻址方式

8/2原则: 80%的程序只使用20%的指令

大多数程序只使用少量的指令就能够运行。

RISC: 精简指令集 (Reduced Instruction Set Computer)

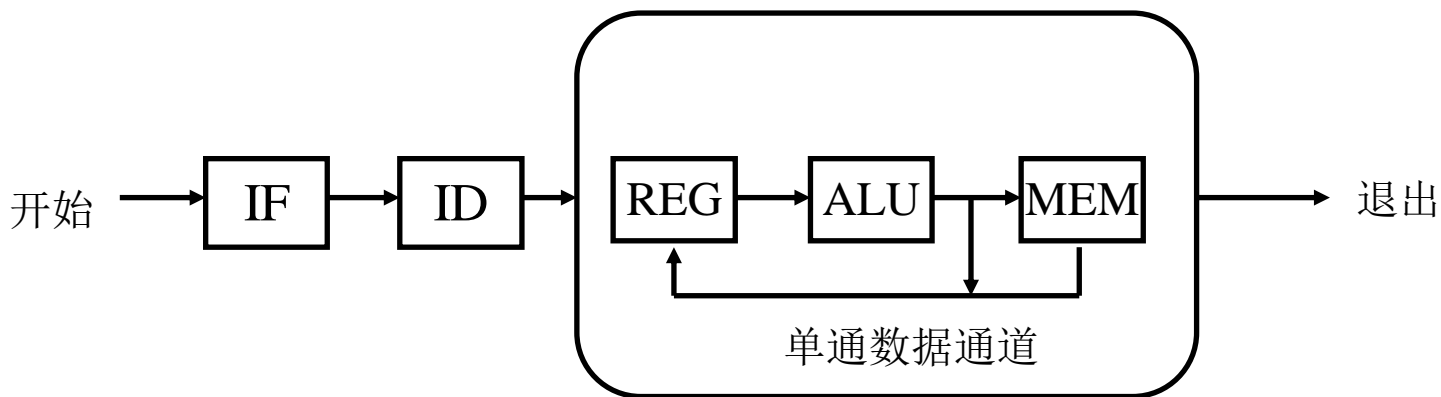
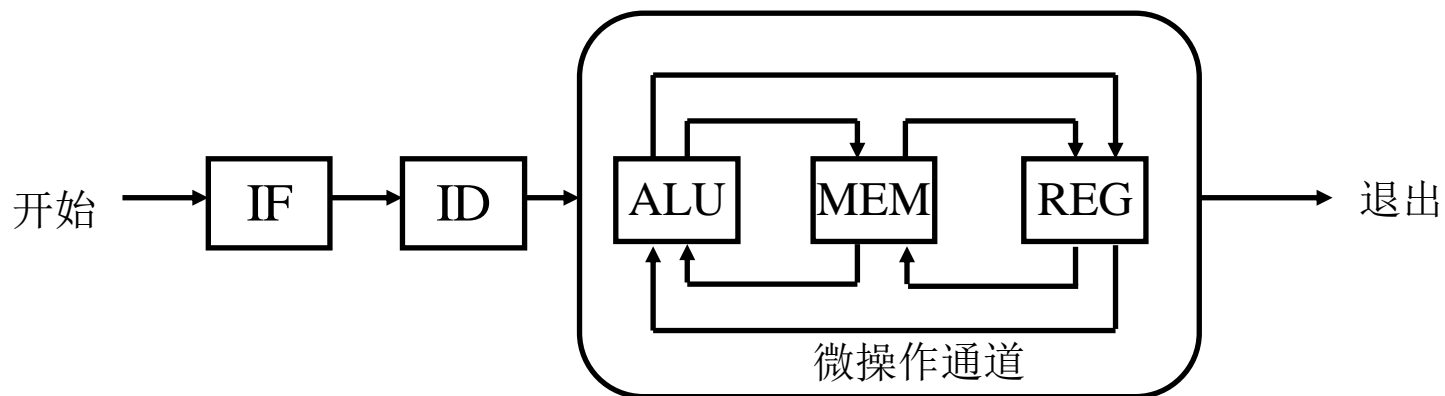
低代码
密度

在通道中只包含最有用的指令

确保数据通道快速执行每一条指令

使CPU硬件结构设计变得更为简单

CISC与RISC的数据通道



CISC的主要缺点

- ▶ 指令使用频度不均衡。
 - 高频度使用的指令占据了绝大部分的执行时间，扩充的复杂指令往往是低频度指令。
- ▶ 大量复杂指令的控制逻辑不规整，不适于VLSI工艺
 - VLSI的出现，使单芯片处理机希望采用规整的硬联逻辑实现，而不希望用微程序，因为微程序的使用反而制约了速度提高。(微码的存控速度比CPU慢5-10倍)。
- ▶ 软硬功能分配
 - 复杂指令增加硬件的复杂度，使指令执行周期大大加长，直接访存次数增多，降低了CPU性能。
- ▶ 不利于先进指令级并行技术的采用
 - 流水线技术

2.1.1 ARM处理器简介

RISC结构一般具有如下特点：

- (1) 单周期的执行。
- (2) 采用高效的流水线操作。
- (3) 无微代码的硬连线控制。
- (4) 指令格式的规格化和简单化。
- (5) 采用面向寄存器组的指令。
- (6) 采用Load/Store（装载/存储）指令结构。
- (7) 注重编译的优化，力求有效地支撑高级语言程序。

2.1.2 ARM体系结构发展

一种**CPU**的体系结构定义了其支持的指令集和基于该体系结构下的处理器编程模型。

相同的体系结构下，由于所面向的应用不同，对性能的要求不同，会有多种处理器。

到目前为止，**ARM**处理器的体系结构发展了v1～v8共8个版本。



体系结构	ARM内核版本
v1	ARM1
v2	ARM2
v2a	ARM2aS、ARM3
v3	ARM6、ARM600、ARM610、ARM7、ARM700、 ARM710
v4	Strong ARM、ARM8、ARM810
v4T	ARM7TDMI、ARM720T、ARM740T、ARM9TDMI、 ARM920T、ARM940T
v5TE	ARM9E-S、ARM10TDMI、ARM1020E
v6	ARM11、ARM1156T2-S、ARM1156T2F-S、 ARM1176JZF-S、ARM11JZF-S
v7	ARM Cortex-M、ARM Cortex-R、ARM Cortex-A
v8	Cortex-A53/57、Cortex-A72等

1. v1版本

v1版本ARM处理器没有商品化，只出现在ARM1原型机上。它的主要特点有：

- (1) 26位的地址空间，寻址空间64MB；
- (2) 只有基本的数据处理指令，甚至没有乘法指令；
- (3) 基于字节、半字和字的Load/Store存储器访问指令；
- (4) 子程序调用指令（BL）和链接指令；
- (5) 操作系统调用的软件中断指令（SWI）。

2. v2版本

对v1版本进行了扩展和完善。仍旧采用**26**位地址空间和**64M**寻址空间。它的主要特点有：

- (1) 增加了**32**位乘法指令和乘加指令；
- (2) 支持协处理器指令；
- (3) 对快速中断模式支持；
- (4) 支持最基本的存储器与寄存器交换指令SWP/SWPB。

3. v3版本

该版本在体系结构上较以前的版本有很大变化。基于该版本的**ARM6**处理器，做为**IP**核独立的处理器，具有片上高速缓存、**MMU**和写缓存的集成**CPU**。它的主要特点有：

- （1）寻址空间增加到**32**位（**4G**）；
- （2）增加了当前程序状态寄存器（**CPSR**）保存当前程序运行的状态信息；
- （3）增加了备份程序状态寄存器（**SPSR**），在程序运行被异常中断时保存现场；
- （4）增加了**MRS/MSR**指令，以访问新增的**CPSR/SPSR**寄存器；
- （5）增加了中止和未定义两种异常模式，以方便操作系统使用数据访问中止异常、指令预取中止异常和未定义指令异常；
- （6）改进了从异常返回指令。

4. v4版本

该版本在v3版本的基础上做了进一步的扩充，是曾经被应用最广的ARM体系结构，ARM7TDMI、ARM9、StrongARM等都采用该结构。它的主要特点有：

- （1）增加了对有符号、无符号半字及有符号字节的存/取指令；
- （2）增加 T变种，引入Thumb状态，处理器工作在该状态下时，指令集为新增的16位Thumb指令集；
- （3）增加了系统模式，该模式下处理器使用用户寄存器；
- （4）完善了软件中断（SWI）指令功能；
- （5）把一些未使用的指令空间捕获为未定义指令。

5. v5版本

在v4版本的基础上增加了一些新的指令。

ARM9E、ARM10和Intel的XScale处理器都采用该版本结构。它的主要特点有：

（1）改进了**ARM指令集**和**Thumb指令集**的混合使用效率；

（2）增加了带有链接和交换的转移指令（**BLX**）、计数前导零指令（**CLZ**）、软件断点指令（**BKPT**）；

（3）**v5TE**版本中增加了**DSP**（数字信号处理）指令集，包括全部算法和**16位指令集**；

支持新的**Java**，提供字节代码执行的硬件和优化软件加速性能。

6. v6版本

该版本2001年发布，并应用在2002年发布的ARM11处理器中。该版本降低耗电量的同时提高了图像处理能力，适合无线和消费类电子产品；高数据吞吐量和高性能的结合。它的主要特点有：

- （1）支持多微处理器内核；
- （2）Thumb代码压缩技术；
- （3）引入Jazelle技术，提高了Java性能，降低了Java应用程序对内存的空间占用；
- （4）通过SIMD（单指令多数数据流）技术，提高了音/视频处理能力。

7. v7版本

v7版本架构是在v6版本的基础上诞生的，对于早期的ARM处理器软件提供了较好的兼容性。它的主要特点有：

（1）采用了在Thumb代码压缩技术上发展的Thumb-2技术，比纯32位代码减少了31%的内存占用，减小了系统开销，能够提供比基于Thumb技术的解决方案高出38%的性能；

（2）首次采用NEON信号处理扩展集，它是一个结合64位和128位的SIMD指令集，对H.264和MP3等媒体解码提供加速，将DSP和媒体处理能力提高了近4倍，并支持改良的浮点运算；

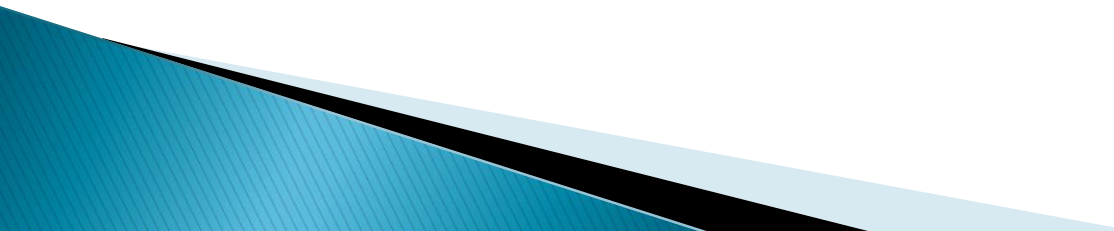
（3）支持改良的运行环境，迎合不断增加的JIT（Just In Time）和DAC（Dynamic Adaptive Compilation）技术的使用。

该架构定义了三大系列：

Cortex-A系列：面向基于虚拟内存的操作系统和用户应用，主要用于运行各种嵌入式操作系统（Linux、WindowsCE、Android、Symbian等）的消费娱乐和无线产品；

Cortex-M系列：主要面向微控制器领域，用于对成本和功耗敏感的终端设备，如智能仪器仪表、汽车和工业控制系统、家用电器、传感器、医疗器械等；

Cortex-R系列：该系列主要用于具有严格的实时响应限制的深层嵌入式实时系统。



8. v8版本

2011年11月，ARM公司发布了新一代处理器架构ARMv8的部分技术细节，这是ARM公司的首款支持64位指令集的处理器架构，将被首先用于对扩展虚拟地址和64位数据处理技术有更高要求的产品领域，如企业应用、高档消费电子产品。目前的ARMv7架构的主要特性都将在ARMv8架构中得以保留或进一步拓展，如TrustZone技术、虚拟化技术及NEON advanced SIMD技术等。ARMv8 架构将64位架构支持引入ARM架构中，其中包括：

- （1）64位通用寄存器、SP（堆栈指针）和PC（程序计数器）；
- （2）64位数据处理和扩展的虚拟寻址；
- （3）两种主要执行状态：AArch64（64位执行状态）和AArch32（32位执行状态）；

两种执行状态支持三个主要指令集：

A32（或ARM）：32位固定长度指令集，通过不同架构变体增强部分32位架构执行环境，现在称为AArch32；

T32 (Thumb)：以16位固定长度指令集的形式引入的，随后在引入Thumb-2技术时增强为**16位和32位**混合长度指令集，部分32位架构执行环境现在称为AArch32。

A64：提供与ARM和Thumb指令集类似功能的32位固定长度指令集，随ARMv8一起引入，它是一种AArch64 指令集。

2.1.3 ARM处理器系列主要产品

1.ARM7系列

ARM7系列主要包括ARM7TDMI、ARM7TDMI-S、带有高速缓存处理器宏单元的ARM720T等。

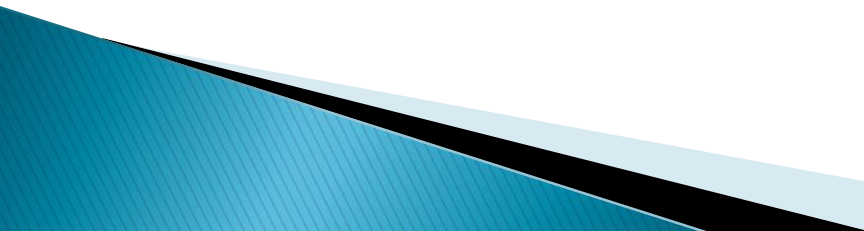
ARM7TDMI基于ARM体系结构V4版本，是目前低端的ARM核。ARM7TDMI中的T代表Thumb 架构扩展，提供两个独立的指令集，D代表内核具有Debug扩展结构，M代表EmbeddedICE 逻辑，I代表增强乘法器，支持64位结果。

ARM7TDMI属于ARM v4体系结构，采用冯诺依曼结构（指令和数据在存储中统一存放，采用同一套总线分时传输），3级流水处理，平均0.9DMIPs/Mhz性能。

ARM7TDMI没有MMU（Memory Management Unit）和Cache，所以仅支持那些不需要MMU和Cahce的小型实时操作系统，如VxWorks、uC/OS-II和uLinux等RTOS。

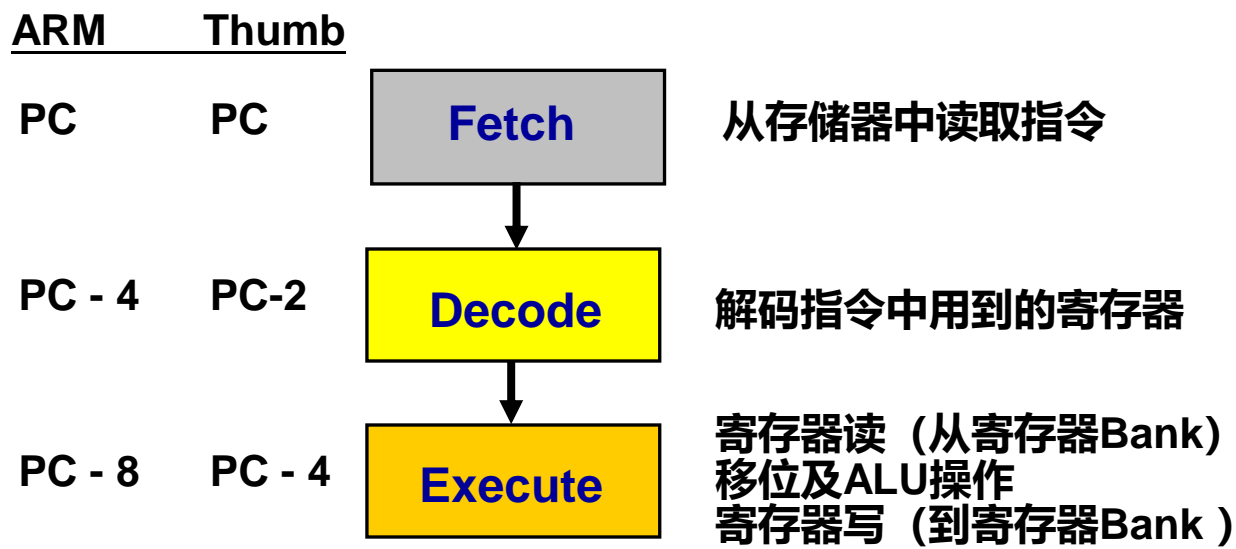
ARM7 系列使用**三级流水线**。该流水线允许多个操作同时处理，而非顺序执行。而**ARM**处理器中的程序计数器**PC**指向正被取指的指令，而非正在执行的指令。

ARM7的三级流水线技术首先从存储器中读取指令，然后解码指令中用到的寄存器，接下来进行执行操作，主要包括从寄存器组中读寄存器值或者移位及**ALU**操作或者在寄存器组中写入寄存器值操作等。



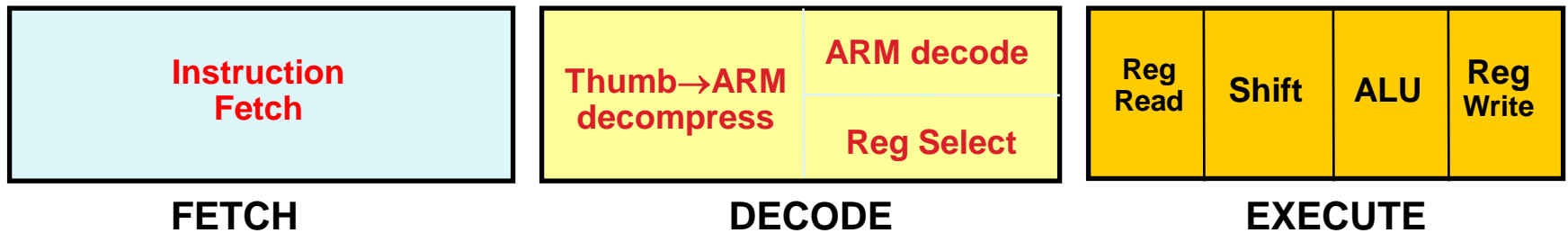
ARM7TDMI指令流水线

- ▶ 为增加处理器指令流的速度，ARM7 系列使用3级流水线。
 - 允许多个操作同时处理，而非顺序执行。
 - PC指向正被取指的指令，而非正在执行的指令。

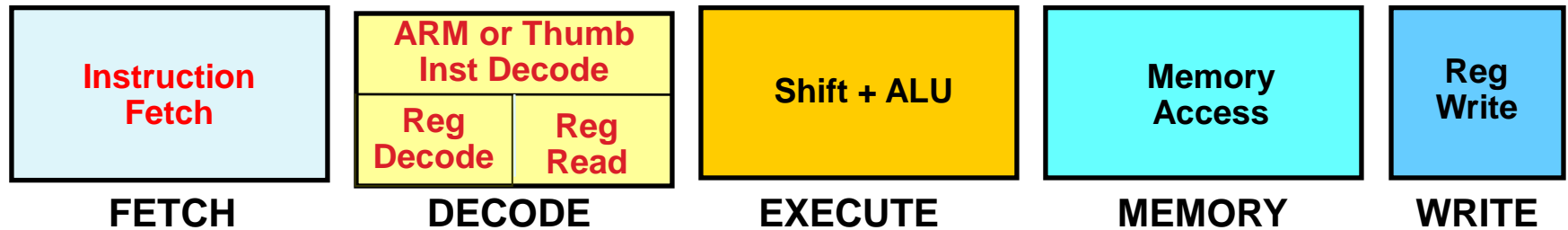


ARM指令流水线

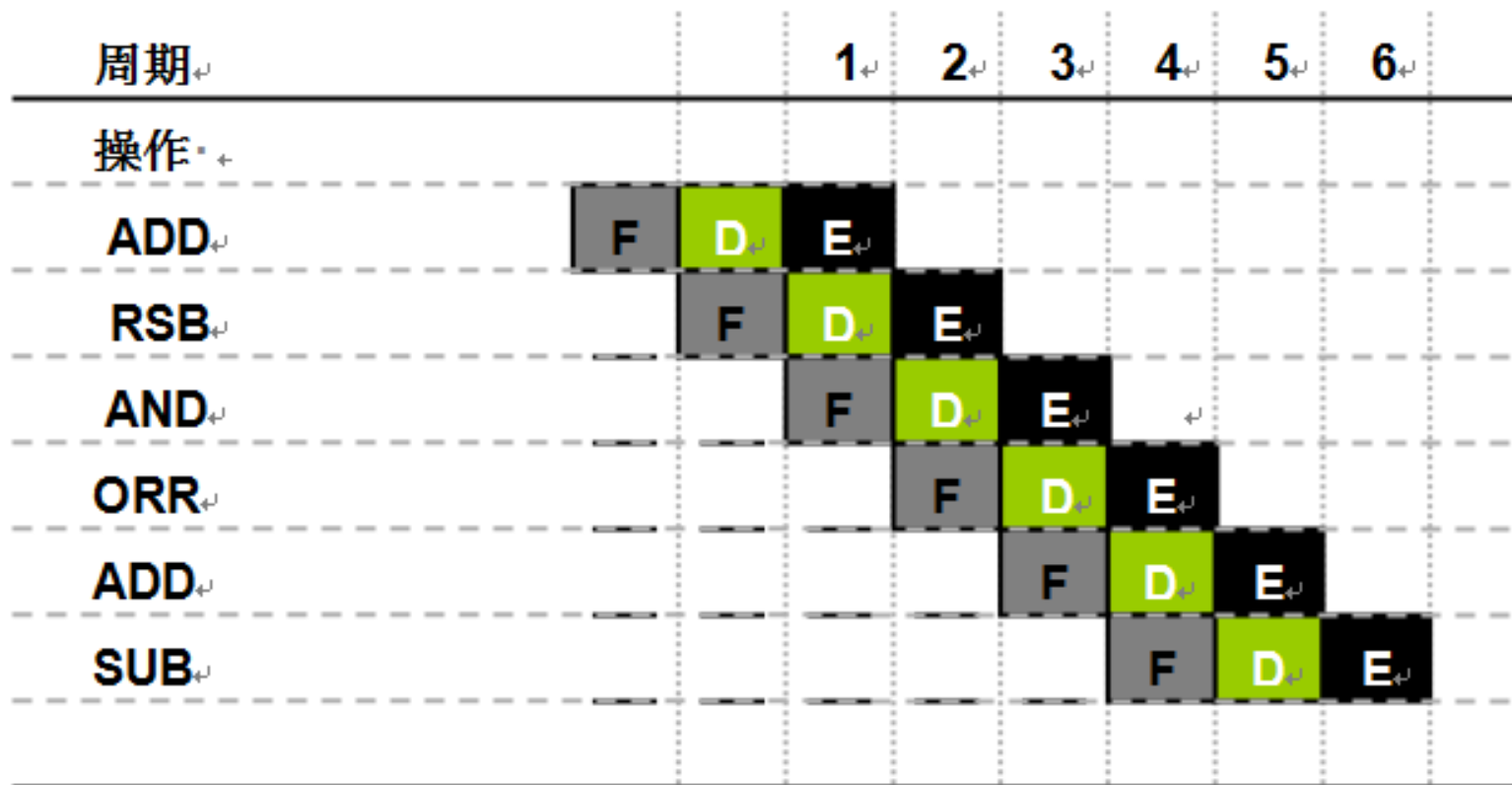
ARM7TDMI



ARM9TDMI



理想的3级流水线 (ARM7TDMI:无访存操作)



F ← 取指令 → D ← 译码 → E ← 执行

ARM的LDR流水线

周期		1	2	3	4	5	6	7	8
操作									
ADD		F	D	E					
RSB			F	D	E				
LDR			F	D	E	E	E		
ADD				F	D	S	S	E	
SUB					F	S	S	D	E
ORR							F	D	E

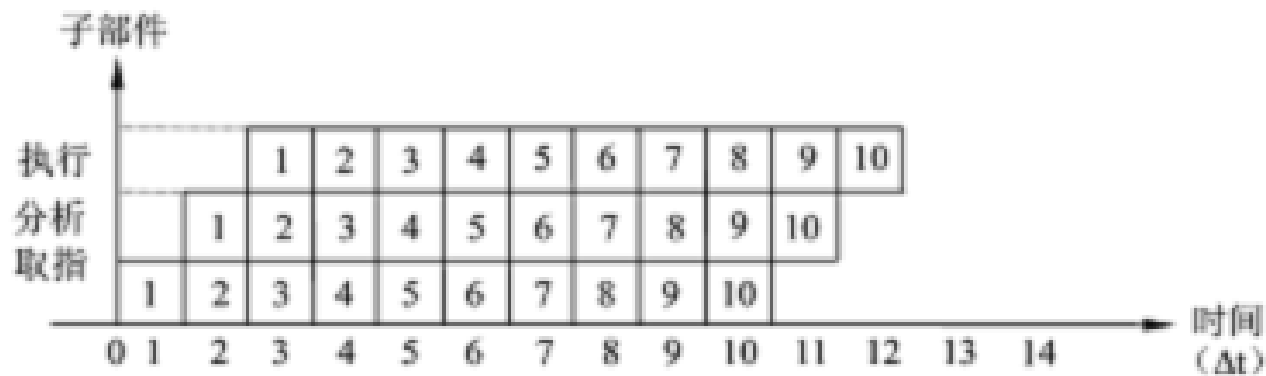
F -- 取指令 → D -- 译码 → E -- 执行 · M -- 访存 · → W -- 回写 · S -- 延迟

思考题

- ▶ 设指令由取指、分析、执行3个子部件完成，每个子部件的工作周期均为 Dt ，采用常规标量单流水线处理机。若连续执行10条指令，则共需时间 Dt 。
- ▶ A. 8 B. 10 C. 12 D. 14

分析

- ▶ 本题考查指令流水的概念。
- ▶ 顺序执行时，每条指令都需三步才能执行完，没有重叠。
- ▶ 所以连续执行10条指令后，共需时间为 $2 + 10 = 12Dt$ 。参考答案C



2.ARM9系列和ARM9E系列

ARM9采用哈佛体系结构，指令和数据分开存放于不同的存储器，分别采用各自的总线进行传输。

ARM9TDMI相比ARM7TDMI，将流水级数提高到五级从而增加了处理器的时钟频率，并使用指令和数据存储器分开的哈佛结构以改善CPI和提高处理器性能。

在ARM9TDMI基础上又有ARM920T、ARM940T和ARM922T，其中ARM940T增加了MPU（Memory Protect Unit）和Cache；ARM920T和ARM922T加入了MMU、Cache和ETM9，从而更好的支持像Linux和WinCE这样的多线程、多任务操作系统。

ARM9E系列属于ARM v5TE，其中ARM926EJ-S是最具代表性的。通过DSP和Java的指令扩展，可获得70%的DSP处理能力和8倍的Java处理性能提升。另外分开的指令和数据Cache结构进一步提升了软件性能；指令和数据TCM（Tightly Couple Memory：紧耦合存储器）接口支持零等待访问存储器；双AMBA AHB总线接口等。

ARM9处理器的主要特点：

- (1) 32bit定点RISC处理器，改进型ARM/Thumb代码交织，增强性乘法器设计。支持实时（**real-time**）调试；
- (2) 片内指令和数据SRAM，而且指令和数据的存储器容量可调；
- (3) 片内指令和数据高速缓冲器（**cache**）容量从4K字节到1M字节；
- (4) 设置保护单元（**protection unit**），非常适合嵌入式系统应用中对存储器进行分段和保护；
- (5) 采用AMBA AHB总线接口，为外部设备提供统一的地址和数据总线；
- (6) 支持外部协处理器，指令和数据总线有简单的握手信令支持；
- (7) 支持标准基本逻辑单元扫描测试方法，而且支持BIST(built-in-self-test)；
- (8) 支持嵌入式跟踪宏单元，支持实时跟踪指令和数据。

3.ARM11系列

ARM11系列主要有ARM1136、ARM1156、ARM1176和ARM11 MP-Core等，它们都是v6体系结构，相比v5系列增加了SIMD多媒体指令，获得1.75x多媒体处理能力的提升。

除了ARM1136外，其他的处理器都支持AMBA3.0-AXI总线。

ARM11系列内核最高的处理速度可达500Mhz以上（其中90nm工艺下，ARM1176可达到750Mhz）以及600DMIPS的性能。

基于ARMv6架构的ARM11系列处理器是根据消费类电子、无线设备、网络应用和汽车电子产品等需求而制定的。它的媒体处理能力和低功耗特点使它特别适合于无线和消费类电子产品；高数据吞吐量和高性能的结合非常适合网络处理应用；另外，在实时性能和浮点处理等方面ARM11可以满足汽车电子应用的需求。

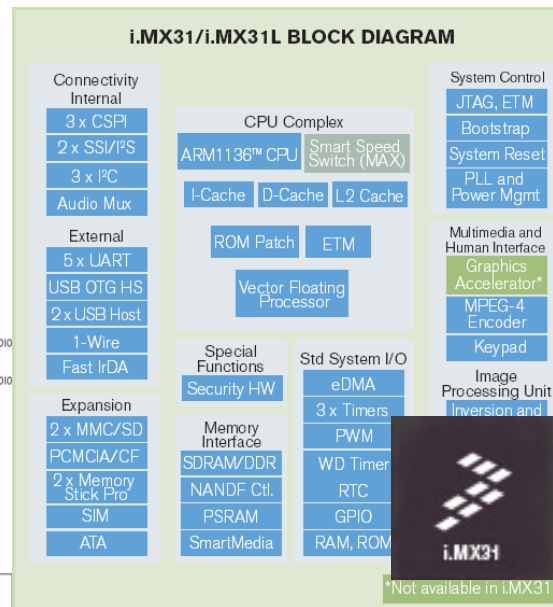
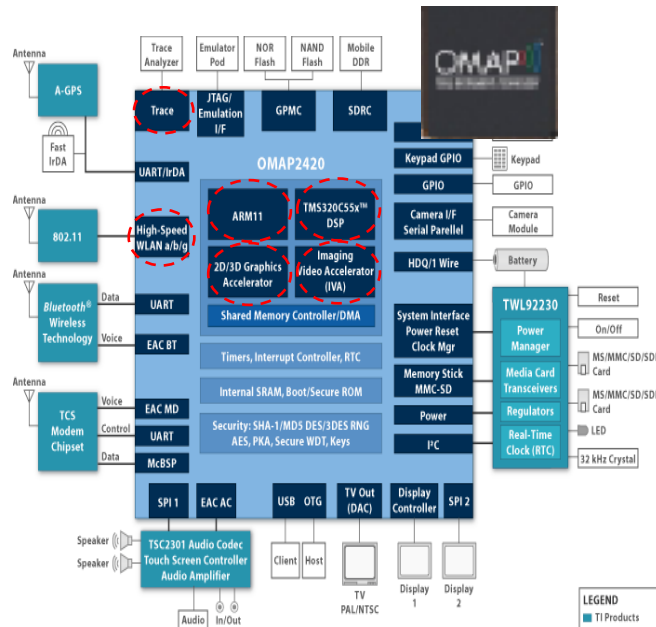
ARM11 芯片

► ARM11

- 300-700+ MHz
- SIMD 指令扩展支持更丰富的多媒体应用
- 40家授权芯片公司



Quatro™ 4230



Qualcomm
Segmentation Strategy
Integration Yields Cost-effective Solutions at All Tiers

	value platform	multimedia platform	enhanced platform	convergence platform
AUDIO	MP3	MP3/AAC/WMA	MP3/AAC/WMA	MP3/AAC/WMA
GRAPHICS	NA	50k Triangles	100k Triangles	3M-4M Triangles
CAMERA	NA	1.2 MPixel	2.4 MPixel	4.8 MPixel
VIDEO	Sub QCIF	15 fps QCIF	30 fps CIF	30 fps VGA
UPROCESSOR	ARM 7 - 50MHz	ARM 9 - 150MHz	ARM 9 - 225 MHz	Dual CPU incl. ARM 11 - 400MHz - 1GHz
LCD Resolution	QCIF [176 x 144]	QCIF [176 x 144]	QVGA [320 x 240]	VGA [640 x 480]

Figures for CDMA2000 Solutions only

FOMA N902i

First ARM11 based phone



4. XScale系列

Intel公司开发的Xscale系列是基于ARM v5TE的 ARM体系结构的内核，在架构扩展的基础上同时也保留了对于以往产品的向下兼容，因此获得了广泛的应用。相比于ARM处理器，XScale功耗更低，系统伸缩性更好，支持16位的Thumb指令和DSP指令集，同时核心频率也得到提高，达到了400Mhz甚至更高。XScale系列处理器还支持高效通讯指令，可以和同样架构处理器之间达到高速传输。XScale系列处理器的另外一个主要扩展是使用了无线MMX，这是一种64位的SIMD指令集，并在新款的Xscale处理器中集成有SIMD协处理器，可以有效的加快视频、3D图像、音频以及其他SIMD传统元素处理。

基于XScale PXA250微处理器性能如下：

- （1）内核工作频率：100-400MHZ；I-Cache 32KB和D-Cache 32KB；I-MMU + D-MMU （各32路变换后备缓冲器TLB快表）；7/8级流水线。
- （2）系统存储器接口：100MHZ SDRAM；4-256MB SDRAM；支持16-256MB DRAM；4个SDRAM区，每个区支持64MB存储器；支持2个PCMCIA/CF卡插槽。
- （3）外围接口：具有16个通道的DMA控制器；LCD控制器（支持被动DSTN和主动TFT显示，最大分辨率800*600*16；系统集成模块（GPIO、中断控制器、PWM）；USB，3个UART，红外（FIR）、I2C总线接口、多媒体通信口、动态电源管理技术。

5. ARM Cortex系列

在ARM11系列之后，Cortex系列是ARM公司目前最新内核系列，属于v7架构。该架构定义了三大系列：**Cortex-A**系列、**Cortex-M**系列和**Cortex-R**系列。

ARM Cortex-A 系列应用型处理器可向具备操作系统平台和用户应用程序的设备提供全方位的解决方案，从超低成本手机、智能手机、移动计算平台、数字电视到企业网络、打印机服务器解决方案。

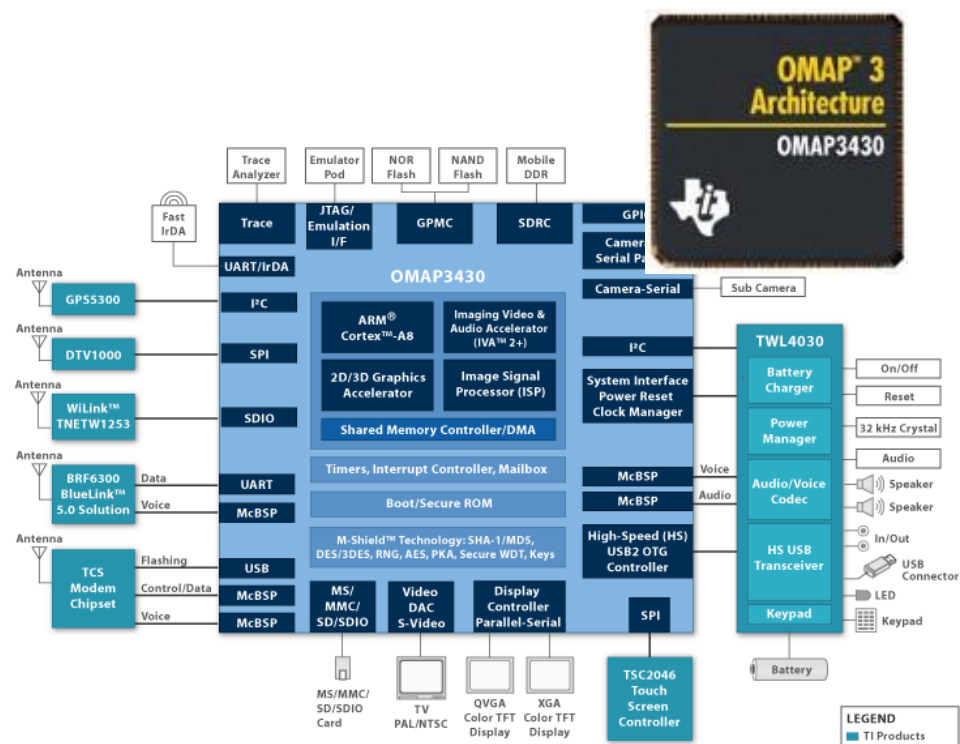
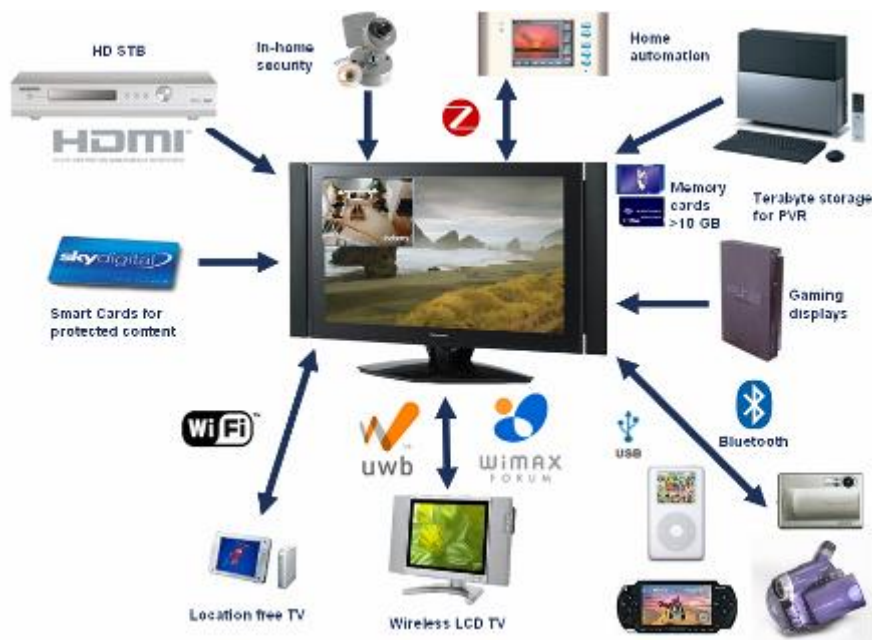
常见Cortex系列型号：

- (1) Cortex-A5 处理器
- (2) Cortex-A7 处理器
- (3) Cortex-A8 处理器
- (4) Cortex-A9处理器
- (5) Cortex-A15 处理器
- (6) ARM Cortex-R处理器
- (7) ARM Cortex-M处理器

高性能的ARM嵌入式处理器

ARM Cortex A8 Application Processor

- ▶ 最快的处理器提供超过2000 DMIPS 的性能
 - 运行于 1GHz 频率 (90nm or 65nm 制造工艺)
- ▶ 功耗小于 300mW



Cortex-M3 实现 \$1 ARM芯片

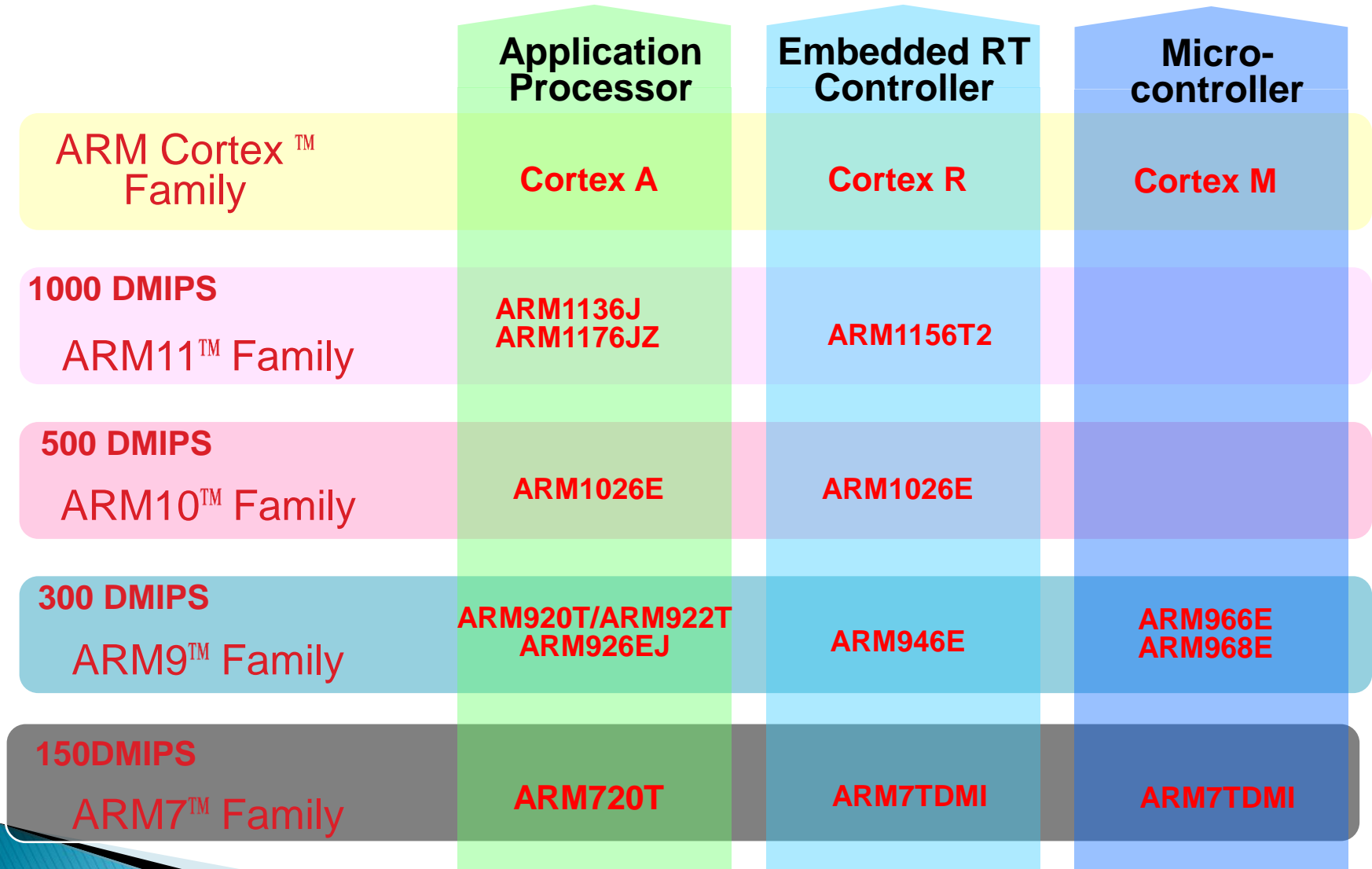
ARM Cortex-M3 微控制器内核，专门针对MCU应用领域而设计，突出低成本、低功耗和高效率。

- ARM Cortex Architecture
- Thumb-2 ISA
- 3 Stage Pipeline
- 1.22 DMIPS/MHz – 30% over ARM7TDMI
- 33K gates – 30% smaller than ARM7TDMI



- Luminary Micro的Stellaris系列MCU产品售价仅1美元

ARM Family



A performance matrix for the ARM Family of processors. The matrix is organized into rows representing different processor families and their performance in DMIPS, and columns representing different processor types. A blue arrow on the left points upwards, indicating increasing performance from bottom to top. The background features a blue and black abstract design at the bottom left.

	Application Processor	Embedded RT Controller	Micro-controller
ARM Cortex™ Family	Cortex A	Cortex R	Cortex M
1000 DMIPS ARM11™ Family	ARM1136J ARM1176JZ	ARM1156T2	
500 DMIPS ARM10™ Family	ARM1026E	ARM1026E	
300 DMIPS ARM9™ Family	ARM920T/ARM922T ARM926EJ	ARM946E	ARM966E ARM968E
150DMIPS ARM7™ Family	ARM720T	ARM7TDMI	ARM7TDMI

2.1.4 ARM开发工具简介

ARM开发工具就是ARM公司为庞大的各领域工程师和开发人员装备的完整的开发工具链，帮助迅速搭建开发平台，降低开发的成本和难度，缩短开发周期，让工程师们充分针对ARM架构处理器进行开发。

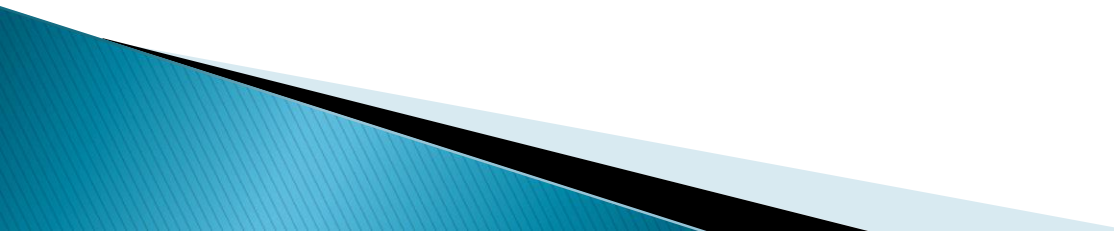
根据开发目标平台的不同，ARM提供不同的工具解决方案。最常见的是MDK-ARM、RVDS，ARM DS5。他们分别针对低端和高端ARM处理器应用。

1. MDK-ARM

RealView Microcontroller Development Kit(MDK) 支持基于包括ARM7, ARM9, Cortex-M3微控制处理器等在内的众多处理器, 例如Atmel, Freescale, Luminary, NXP, OKI, Samsung, Sharp, ST, TI等厂家的产品。MDK提供工业标准的编译工具和强大的调试支持。

MDK是专为MCU的用户开发嵌入式软件而设计的一套开发工具。包括根据器件定制的调试仿真支持, 丰富的项目模版, 固件示例以及为内存优化的RTOS库。MDK上手容易, 功能强大, 适合微控制器应用程序开发。

MDK主要是为终端客户提供价格低廉, 功能强大的开发工具。集成了RealView编译工具, Keil uVision开发环境, 支持基于ARM7, ARM9, Cortex-M1, Cortex-M3, Cortex-R4等ARM产品的仿真。



2.RVDS

RVDS（**RealView Development Suite**）是ARM公司推出的专为**SOC**，**FPGA** 以及**ASIC**用户开发复杂嵌入式应用程序或者和操作系统平台组件接口而设计的开发工具，被业界称为最好的**ARM**开发工具。**RVDS**支持器件设计，支持多核调试，支持基于所有**ARM** 和**Cortex**系列**CPU**的程序开发。**RVDS**还可以和第三方软件进行很好的连接。

RVDS 是ARM公司继**SDT** 与**ADS1.2**之后主推的新一代开发工具，目前最高版本是**4.1**。**RVDS**对代码密度的提升、代码执行速度的提高，都可以由**ARM**开发工具自动实现，而不需要软件开发人员花费过多的时间手动优化高级语言代码。这是**RVDS**的优势所在。

RVDS包含有四个模块：

(1) IDE：**RVDS**中集成了**Eclipse IDE**，用于代码的编辑和管理。支持语句高亮和多颜色显示，以工程的方式管理代码，支持第三方**Eclipse**功能插件。

(2) RVCT：**RVCT**是业界最优秀的编译器，支持全系列的**ARM**和**XSCALE**架构，支持汇编语言、**C**语言和**C++**语言。**RVDS**的编译器根据最新的**ARM**架构进行特别的优化，针对每个**ARM**架构都提供最好的代码执行性能，最优的代码密度。可以根据需要选择调试信息级别，以及不同的代码优化方向和优化级别。

(3) RVD：是**RVDS**中的调试软件，功能强大，支持**Flash**烧写和多核调试，支持多种调试手段，快速错误定位。

(4) RVISS：是指令集仿真器，支持外部设备虚拟，可以使软件开发和硬件开发同步进行，同时可以分析代码性能，加快软件开发速度。

3. ARM DS5

ARM DS5，也叫ARM DS-5，是一款支持开发所有ARM内核芯片的集成开发环境，也是一套针对 ARM 支持的linux和android平台的全面的端到端软件开发工具套件。ARM DS5提供具有跟踪、系统范围性能分析器、实时系统模拟器和编译器的应用程序和内核空间调试器。这些功能包含在定制的、功能强大且用户友好的基于Eclipse的IDE中。借助于该工具套件，可以很轻松地地为ARM支持的系统开发和优化基于Linux的系统，缩短开发和测试周期，并且可帮助工程师创建资源利用效率高的软件。

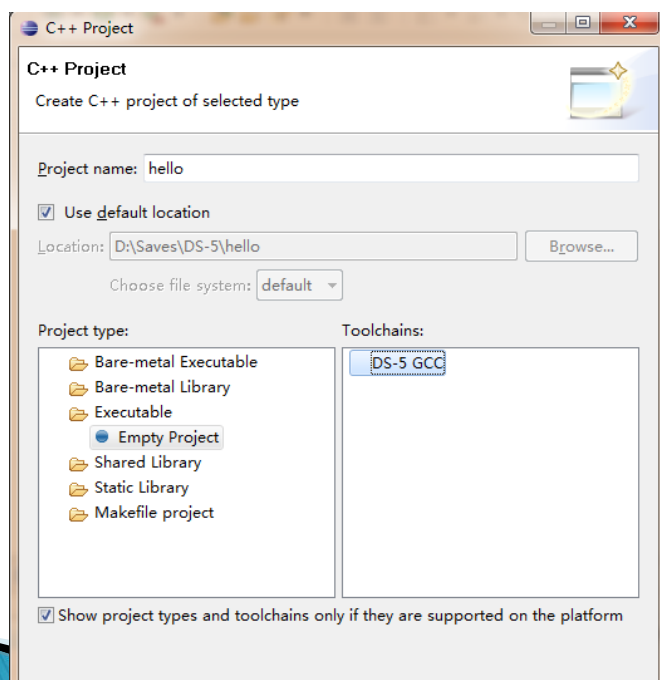


图2-1 ARM DS5的工程配置

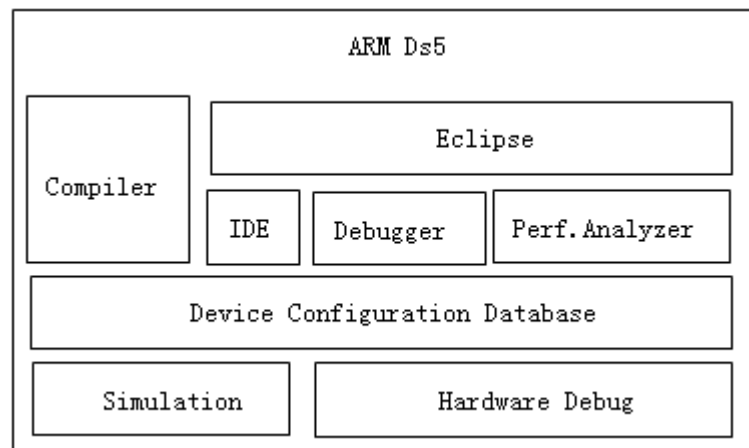


图2-2 ARM DS-5功能框架

2.2

Part Two

Cortex-A8处理器架构

Cortex-A8处理器是第一款基于**ARMv7**构架的的应用处理器,使用了能够带来更高性能、更低功耗和更高代码密度的**Thumb-2**技术,新增了**130**条指令。新增的功能使用户在进行终端服务时无需在**ARM**和**Thumb**模式间进行切换,同时可访问整套处理器寄存器。产生的代码保持**Thumb**指令的传统代码密度,却可以实现**32**位**ARM**代码的性能。

ARM处理器中首次采用面向音频、视频和3D图形的NEON媒体和信号处理技术。这是一个64/128位混合SIMD架构。通过使用NEON技术执行典型的多媒体功能，Cortex-A8处理器可在275MHz以30帧/秒的速度解码MPEG-4 VGA视频（包括去环状块、解块过滤和YUV至RGB的变换）并以350MHz解码H.264视频。

Cortex-A8还采用了Jazelle-RCT Java加速技术，可以支持Java程序的预编译与实时编译，对实时(JIT)和动态调整编译(DAC)提供最优化,将即时（JIT）字节码应用程序的内存占用削减到原先的三分之一。代码变少可增强性能并降低功率。

Cortex-A8中采用的TrustZone技术可确保消费类产品（如运行开放式操作系统的移动电话、个人数字助理和机顶盒）中的数据隐私和DRM得到保护。



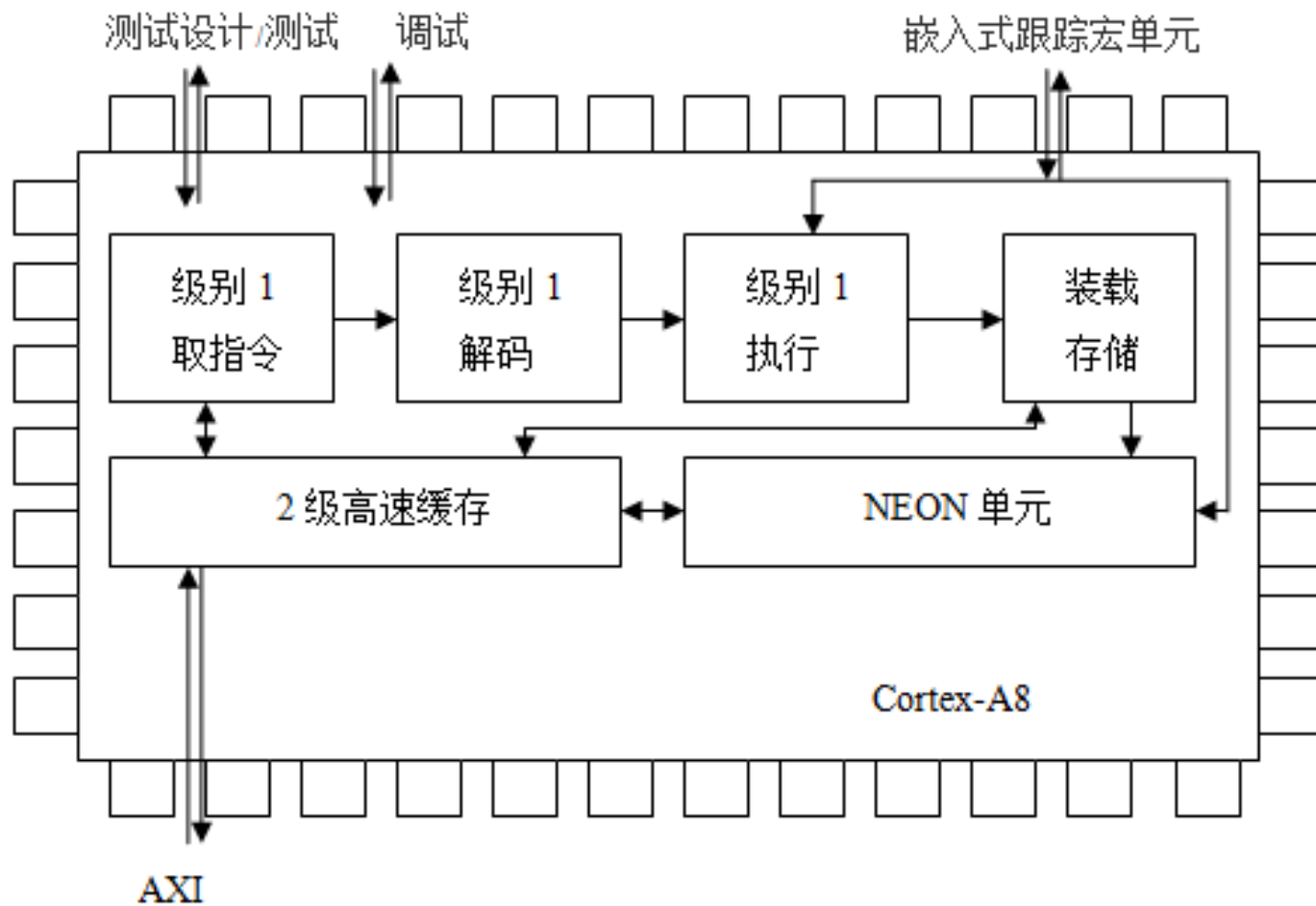


图2-3 ARM Cortex-A8内核系统框图

1.指令读取单元（Instruction Fetch）

指令读取单元的主要构成如下图所示，指令读取单元对指令流进行预测，从L1指令缓存中取出指令放到译码流水线中。在此过程中使用到了TLB

（Translation Lookaside Buffers，转换旁路缓冲器）。

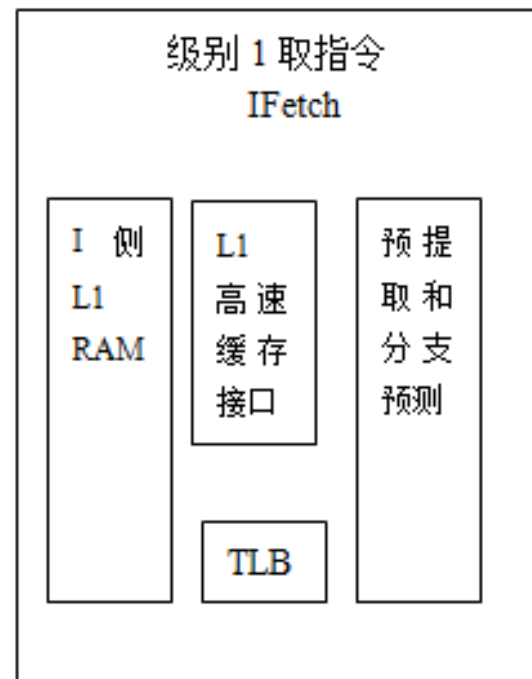


图2-4 指令读取单元主要构成

2. 指令译码（解码）单元（**Instruction Decode**）

指令译码单元对所有的ARM、Thumb-2指令进行译码排序，包括调试控制协处理器**CP14**的指令、系统控制协处理器**CP15**的指令。

指令译码单元处理指令的顺序是：异常、调试事件、复位初始化、存储器内嵌自测**CM13IST**、等待中断、其它事件。

3. 指令执行单元（Instruction Execute）

指令执行单元包括两个对称的ALU（算术逻辑单元）流水线（ALU0和ALU1），它们都可以处理大多数算术指令。ALU流水线0始终执行一对旧的发射指令。Cortex-A8处理器还配备乘法器和加载存储流水线，但这些都不对两条ALU流水线执行额外指令，可将它们视为“独立”流水线。使用它们必需同时使用两条ALU流水线中的一个，乘法器流水线只能与ALU1流水线中的指令结合使用，而加载存储流水线只能与任一ALU中的指令结合使用。

指令执行单元的功能：

- 执行所有整数ALI运算和乘法运算，并修改标志位；
- 根据要求产生用于存取的虚拟地址以及基本回写值；
- 将要存放的数据格式化，并将数据和标志向前发送；
- 处理分支及其它指令流变化，并评估指令条件码。

4. 数据存取单元（Load/Store）

数据存取单元包括全部的L1数据缓存部分和整数存取流水线，流水线可在每个周期接收一次数据存或取，可以在流水线0或流水线1上。

数据存取单元由以下几个部分组成：

- L1数据缓存；
- 数据TLB；
- 整数存储缓存；
- NEON存储缓存；
- 取整数数据对齐、格式化单元；
- 存整数数据对齐、格式化单元。

5. L2缓存单元（L2 Cache）

Cortex-A8处理器的L2缓存单元大小可配置为从64k到2M，包括L2 Cache和缓冲接口单元BIU。

L1数据缓存的内容与L2缓存不兼容，L1指令缓存的内容是L2缓存的子集，当指令预取单元和数据存取单元在L1 Cache中未命中时，L2 Cache将为它们提供服务。L2缓存与L1缓存之间采用低延迟、高带宽专用接口，这将L1缓存行填充的延迟时间降至最低，并且与主系统总线不会产生流量冲突。

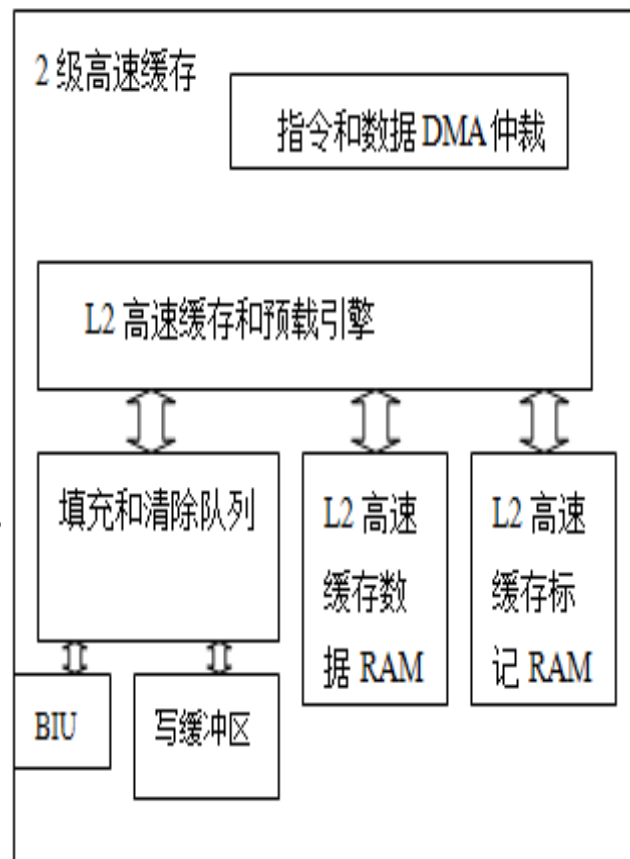


图2-5 2级高速缓存主要构成

6. NEON媒体处理引擎

ARM NEON技术是适用于ARM Cortex-A系列处理器的一种SIMD(Single Instruction, Multiple Data,单指令、多数据)扩展结构。Cortex-A8处理器的NEON媒体处理引擎包括一个10段流水线以及高级SIMD多媒体指令集。

NEON媒体处理引擎主要由以下几个部分组成：

- NEON指令队列；
- NEON数据队列；
- NEON译码逻辑的两个流水线；
- 三条用于SIMD整数指令的执行流水线；
- 两条用于SIMD的单精度浮点流水线；
- 一条加载存储/交换流水线；
- 一个非流水线向量浮点单元（VFPLite），执行VFPv3数据处理指令。

7. ETM单元（嵌入式跟踪宏单元）

ETM单元是一个非侵入跟踪宏单元，对嵌入式处理器内核提供了实时跟踪能力。它向一个跟踪端口输出处理器执行的信息。在系统调试和系统性能分析时，可以对指令和数据进行跟踪，并能对跟踪信息进行过滤和压缩。

ETM单元直接连接到ARM处理器内核，通过一个称为ATB（高级跟踪总线）的外部接口与处理器外部连接。

8. 外部接口

Cortex-A8有丰富的外部接口。主要有：

- **AMBA AXI总线接口**：一种面向高性能、高带宽、低延迟的片内总线。AXI总线接口是可配置的64位或128位AMBA高速总线接口，用于执行L2 Cache的填充和L1 Cache指令及数据的访问。
- **AMBA APB接口**：Cortex-A8处理器通过一个APB接口来访问ETM、CTI和调试寄存器。APB接口与CoreSight调试体系结构（ARM多处理器跟踪调试体系）兼容。
- **AMBA ATB接口**：Cortex-A8处理器通过一个ATB接口输出调试信息。ATB接口兼容CoreSight调试体系结构。
- **DFT（Design For Test）接口**：DFT接口为生产时使用MBIST（内存内置自测试）和ATPG（自动测试模式生成）进行内核测试提供支持。

2.3

Part Three

Cortex-A8处理器工作模式和状态

2.3.1 Cortex-A8处理器工作模式

Cortex-A8是基于ARMv7构架的处理器，共有8种工作模式：

处理器模式	模式标识符	备注
用户模式 (User)	usr	常程序执行模式
系统模式 (System)	sys	使用和用户模式相同的寄存器组，用于运行特权级操作系统任务
管理模式 (Supervisor)	svc	系统复位或软件中断时进入该模式，是供操作系统使用的一种保护模式
外部中断模式 (IRQ)	irq	低优先级中断发生时进入该模式，常用于普通的外部中断处理
快速中断模式 (FIQ)	fiq	高优先级中断发生时进入该模式，用于高速数据传输和通道处理
数据访问中止模式 (Abort)	abt	当存取异常时进入该模式，用于虚拟存储和存储保护
未定义指令中止模式 (Undefined)	und	当执行未定义指令时进入该模式，用于支持硬件协处理器的软件仿真
安全监控模式 (Monitor)	mon	可在安全模式和非安全模式下转换

异常模式

用户模式

非用户模式，或特权模式

处理器的运行模式可以通过软件控制进行切换，也可以通过外部中断或异常处理过程进行切换。大多数情况下，应用程序运行在用户模式下，应用程序不能访问受操作系统保护的系统资源，也不能直接进行处理器工作模式的切换。在需要进行工作模式切换时，应用程序可以产生异常处理，在异常处理过程中进行工作模式切换，由操作系统控制整个系统资源的使用。

用户模式

特权模式：是为了服务中断或异常，或访问受保护的资源，具有多系统资源的完全访问权限，可自由的切换工作模式。

系统模式：系统模式不能由任何异常进入，它有与用户模式完全相同的寄存器。系统模式供需要访问系统资源的操作系统任务使用，这样避免使用和异常模式相关的寄存器，保证在任何异常发生时都不会使任务的状态不可靠。

异常模式：除了可以通过程序切换进入外，还可以在发生特定的异常中断时进入。每一种异常模式都有一组专用的寄存器，以保证在进入异常模式时用户模式下的寄存器（保存着工作模式切换前的程序运行状态）不被破坏。

2.3.2 Cortex-A8处理器状态

Cortex-A8处理器是32位处理器，可执行32位ARM指令集指令，同时兼容16位Thumb-2指令集指令和数据类型。有3种工作状态，这些状态由程序状态寄存器（CPSR）的T位和J位控制与切换。

- ARM状态：执行32位的字对齐的ARM指令集指令，T位和J位为0；
- Thumb状态：执行16位或32位半字对齐的Thumb-2指令集指令，T位为1，J位为0；
- ThumbEE状态：执行为动态产生目标而设计的16位或32位半字对齐的Thumb-2指令集的变体，T位和J位为1，也称为Jazelle RCT技术。

ARM指令必须在ARM状态下执行；

Thumb指令也必须在Thumb状态下执行；

ARM处理器可以在两种状态下切换，只要遵循ATPCS调用规则，ARM子程序和Thumb子程序之间可以相互调用。

ARM状态和Thumb状态之间的切换并不影响处理器工作模式和寄存器组的内容。处理器复位后开始执行代码时，处于ARM状态。

处理器状态之间的切换：

（1）ARM状态和Thumb状态之间切换

执行**BX**和**BLX**指令时，将通用寄存器中所存储的目标地址值复制到程序寄存器**PC**，该目标地址值的最低一位为**1**则切换到**Thumb**状态。如果处理器在**Thumb**状态时发生异常（异常处理必须在**ARM**状态下），当异常处理返回时自动切换到**Thumb**状态。

执行**BX**和**BLX**指令时，将通用寄存器中所存储的目标地址值复制到程序寄存器**PC**，该目标地址值的最低一位为**0**则切换到**ARM**状态。处理器进行异常处理时，把**PC**的值放入异常模式链接寄存器中，从异常向量地址开始执行程序，系统自动进入**ARM**状态。

（2）Thumb状态和ThumbEE状态之间切换

使用**ENTERX**指令和**LEAVEX**指令。

2.4 Part Four

Cortex-A8存储器管理

若内存按字节编址，用存储容量为 $8K \times 8$ 比特的存储器芯片构成地址编号A0000H~DFFFFH的内存空间，则至少需要多少片。

- ▶ 本题考查内存容量的计算。
- ▶ 给定起、止地址码的内存容量 = 终止地址 - 起始地址 + 1。
- ▶ 将终止地址加1等于E0000H，再减去起始地址，即 $E0000H - A0000H = 40000H$ 。十六进制的 $(40000)_{16} = 2^{18}$ 。
- ▶ 组成内存存储器的芯片数量 = 内存存储器的容量 / 单个芯片的容量。
- ▶ $2^{18} / (8 * 2^{10}) = 2^{18} / 2^{13} = 2^5$

2.4.1 ARM的基本数据类型

Cortex-A8是32位处理器，支持多种数据类型：

- 字节（Byte）：8位；
- 半字（Half word）：16位；
- 字（Word）：32位；
- 双字（Double word）：64位。

当数据是无符号数时，二进制格式存储，数据范围为：

$0 \sim 2^N - 1$ ，其中N为数据类型长度；

当数据是有符号数时，二进制补码格式存储，数据范围为：

$-2^{N-1} \sim 2^{N-1} - 1$ ，其中N为数据类型长度。

ARM的体系结构将存储器看成是从0x00000000地址开始的按字节编码的线性存储结构，每个字节都有对应的地址编码。由于数据有不同的字节大小（1字节、2字节、4字节等），导致数据在存储器中存放不是连续的，这样降低了存储系统的效率，甚至引起数据读写错误。因此数据必须按照以下方式对齐：

- 以字为单位，按4字节对齐，地址最末两位为00。
- 以半字为单位，按2字节对齐，地址最末一位为0；
- 以字节为单位，按1字节对齐；

2.4.2 浮点数据类型

浮点运算使用在**ARM**硬件指令集中未定义的数据类型。在协处理器指令空间定义了一系列浮点指令，这些指令全部可以通过未定义指令异常（该异常收集所有硬件协处理器不接受的协处理器指令）在软件中实现，其中的一小部分也可以由浮点运算协处理器**FPA10**以硬件方式实现。

另外**ARM**公司还提供了**C**语言编写的浮点库作为**ARM**浮点指令集的替代方法（**Thumb**代码只能使用浮点指令集）。该库支持**IEEE**标准的单精度和双精度格式。**C**编译器有一个关键字标志来选择，它产生的代码与软件仿真（通过避免中断、译码和浮点指令仿真）相比既快又紧凑。

2.4.3 大/小端存储模式

Cortex-A8处理器支持大端（Big-endian）和小端（Little-endian）两种存储模式，同时还支持混合大小端模式（既有大端模式也有小端模式）和非对齐数据访问。可以通过硬件的方式设置（没有提供软件的方式）端模式。

大端模式是被存放字数据的高字节存储在存储系统的低地址中，而被存放的字数据的低字节则存放在存储系统的高地址中。

高对低
低对高

小端模式中，存储系统的低地址中存放的是被放字数据中的低字节内容，存储系统的高地址存放的是被存字数据中的高字节内容。

低对低
高对高

例如，一个32位的字数据
0x12345678

高地址	78
	56
	34
低地址	12

大端存储模式

高地址	12
	34
	56
低地址	78

小端存储模式

判断处理器使用大端还是小端最简单的方法可以使用C语言中的union，下面程序段中的IsBigEndian（）可以简单判断该处理器是否为大端模式。

```
typedef union
{
char chChar;
short shShort;
}UnEndian;
//该枚举体的内存分配如下，chChar和shShort的低地址字节重合
//如果是BigEndian则返回true
bool IsBigEndian（）
{
UnEndian test;
test.shShort =0x10;
//如果是大端模式，则上面的语句就该同时把chChar成员赋值成了0x10
if( test.chChar==0x10)
{
return false;
}
return true;
}
```

内存和I/O - 大小端

- 实例

- 变量A: word A=0x f6 73 4b cd, 在内存中的起始地址为0x b3 20 45 00

- 变量B: half word B=218 在内存中的起始地址为

问题: half word B=218与word C=218在内存中的存放方式有何不同? 请分大端和小端两种情况说明。

大端: 0xdddddd0	4b
	cd
	00
	da

小端: 0xdddddd0	73
	f6
	da
	00

2.4.4 寄存器组

Cortex-A8处理器共有40个32位寄存器，包括33个通用寄存器和7个状态寄存器。其中状态寄存器包括1个CPSR（Current Program Status Register，当前程序状态寄存器）和6个SPSR（Saved Program Status Register，备份程序状态寄存器）。

这些寄存器不能同时访问，在不同的处理器工作模式下只能够访问一组相应的寄存器组。


图2-7 ARM状态下的通用寄存器组

不分组的通用寄存器

系统和用户模式	快速中断模式	管理模式	数据访问中止模式	外部中断模式	未定义指令中止模式	安全监控模式
r0	r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7	r7
r8	r8_fiq	r8	r8	r8	r8	r8
r9	r9_fiq	r9	r9	r9	r9	r9
r10	r10_fiq	r10	r10	r10	r10	r10
r11	r11_fiq	r11	r11	r11	r11	r11
r12	r12_fiq	r12	r12	r12	r12	r12
r13	r13_fiq	r13_svc	r13_abt	r13_irq	r13_und	r13_mon
r14	r14_fiq	r14_svc	r14_abt	r14_irq	r14_und	r14_mon
r15	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

图2-8 ARM状态下的状态寄存器组

分组的通用寄存器

 = 特权模式

在特权模式下，特定模式下的寄存器阵列才是有效的。

1. 通用寄存器组

R0~R7是不分组的通用寄存器；

R8~R15是分组的通用寄存器；

在ARM状态下，任何时刻，16个数据寄存器R0~R15和1~2个状态寄存器是可访问的。在特权模式下，特定模式下的寄存器阵列才是有效的。

未分组的通用寄存器R0~R7用于保存数据和地址。在处理器的所有工作模式下，它们中的每一个都指向一个物理寄存器，且没有被系统用于特殊用途。在处理器工作模式切换时，由于使用的是相同的物理存储器，可能会破坏寄存器中的数据。

分组的通用寄存器R8~R15则具有不同的处理器工作模式决定访问的物理寄存器不同的特点。如图2-7所示，每个物理寄存器名字的形式为 **Rx_<mode>**，<mode>是模式标识符，每个模式标识符指示当前所处的工作模式。

- R8~R12寄存器分别对应两个不同的物理寄存器，分别是快速中断模式下的相应存储器和非快速中断模式下的相应存储器。
- R13、R14寄存器分别对应七个不同的物理寄存器，除了用户和系统模式共用一个物理寄存器外，其它六个分别是fiq、svc、abt、irq、und和mon模式下的不同物理寄存器。R13常作堆栈指针(SP: Stack Pointer); R14子程序链接寄存器(LR: Link Register)，该寄存器由ARM编译器自动使用。在执行BL和BLX指令时，R14保存返回地址。。

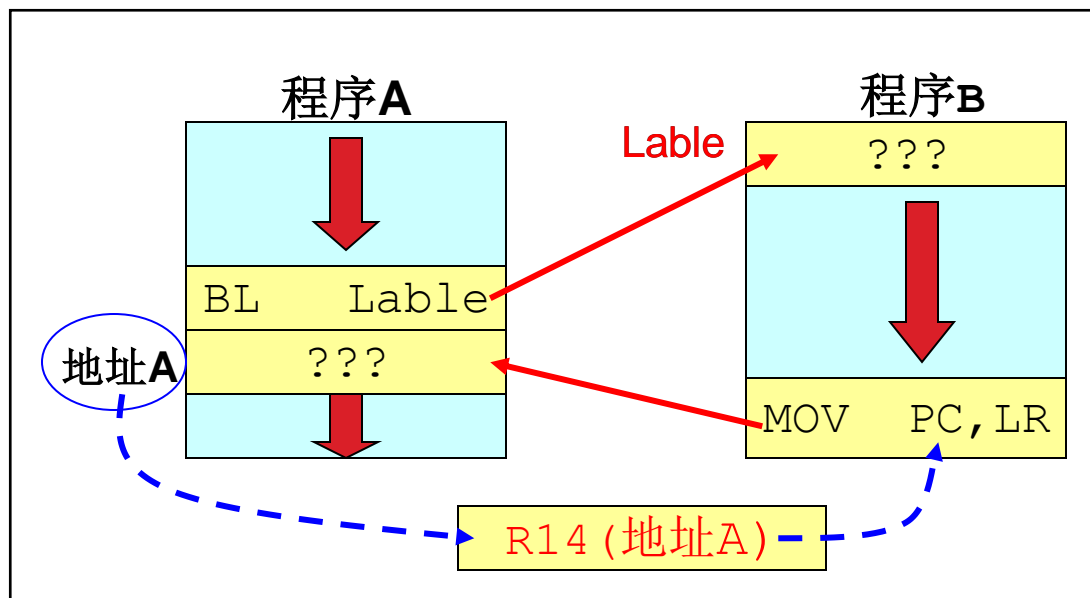
内部寄存器

▸ R14 (LR) 寄存器与子程序调用

操作流程

1. 程序A执行过程中调用程序B；

2. 程序跳转至标号Lable，执行程序B。同时硬件将“BL Lable”指令的下一条指令所在地址存入R14 (LR)；



3. 程序B执行最后，将R14寄存器的内容放入PC，返回程序A。

内部寄存器

▶ R14寄存器与异常发生

异常发生时，程序要跳转至异常服务程序，对返回地址的处理与子程序调用类似，都是由硬件完成的。区别在于有些异常有一个小常量的偏移。

内部寄存器

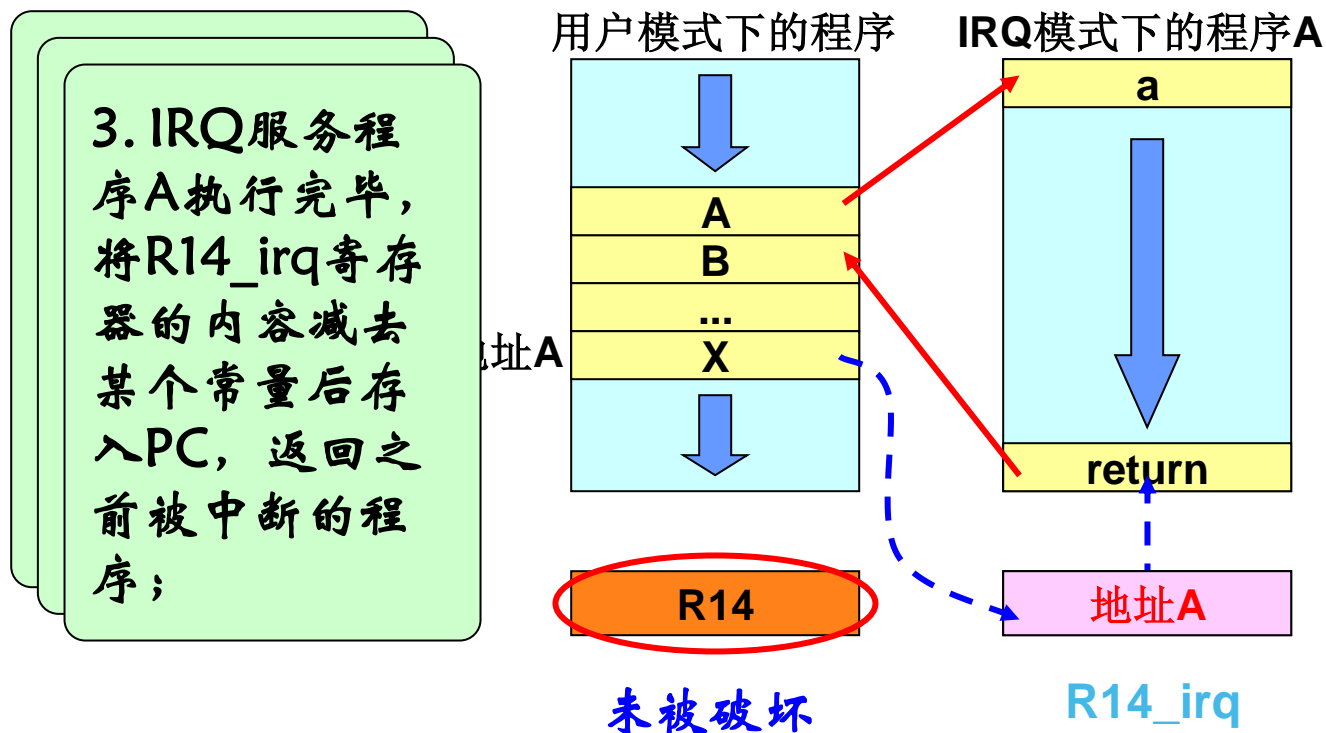
▶ R14寄存器注意要点

当发生异常嵌套时，这些异常之间可能会发生冲突。

- ▶ 例如：如果用户在用户模式下执行程序时发生了IRQ中断，用户模式寄存器不会被破坏。但是如果允许在IRQ模式下的中断处理程序重新使能IRQ中断，并且发生了嵌套的IRQ中断时，外部中断处理程序保存在R14_irq中的任何值都将被嵌套中断的返回地址所覆盖。

内部寄存器

R14寄存器注意要点:



内部寄存器

R14寄存器注意要点:

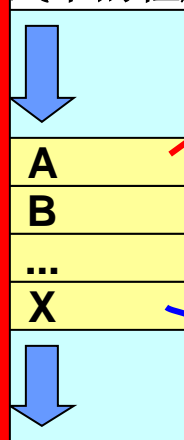
解决办法是确保R14的对应版本在发生中断嵌套时不再保存任何有意义的值（将R14入栈），或者切换到其它处理器模式下。

能正确返回；

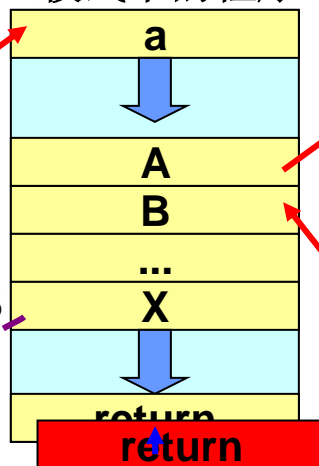
R14

未被破坏

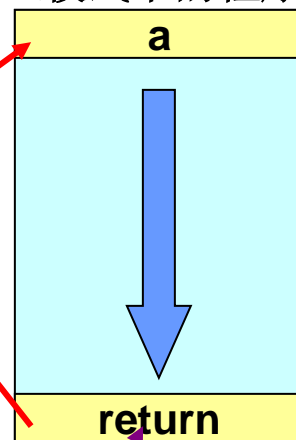
式下的程序



IRQ模式下的程序A



IRQ模式下的程序B



地址B

地址B

R14_irq 被破坏

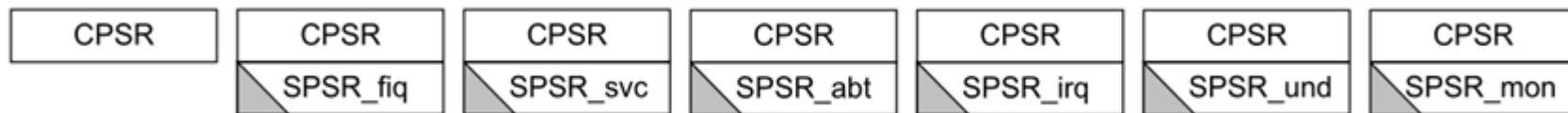
➤ **程序计数器R15 (PC)**，用于记录程序当前的运行地址。ARM处理器每执行一条指令，都会把PC增加4字节(Thumb模式为两个字节)。此外，相应的分支指令(如BL等)也会改变PC的值。在ARM状态下，PC字对齐；在Thumb和ThumbEE状态下，PC半字对齐。


➤ FIQ模式下有7个分组寄存器映射到R8~R14，即R8_fiq~R14_fiq,所以很多快速中断处理不需要保存任何寄存器。

2. 状态寄存器

ARM处理器有两类程序状态寄存器：1个当前程序状态寄存器CPSR和6个备份程序状态寄存器SPSR。它们的主要功能是：

- 保存最近执行的算术或逻辑运算的信息；
- 控制中断的允许或禁止；
- 设置处理器工作模式。

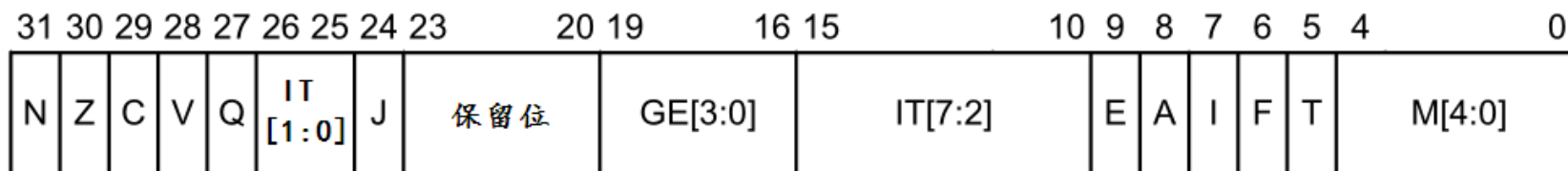


 = 特权模式

每一种处理器模式下使用专用的备份程序状态寄存器。当特定的中断或异常发生时，处理器切换到对应的工作模式下，该模式下的备份程序状态寄存器保存当前程序状态寄存器的内容。当异常处理程序返回时，再将其内容从备份程序状态寄存器回复到当前程序状态寄存器。

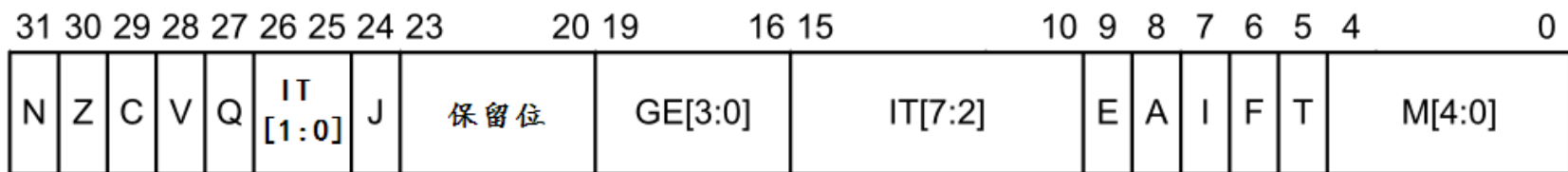
程序状态寄存器的格式如图2-9所示，32位寄存器会被分成四个域：

- 标志位域f (flag field), PSR[31:24];
- 状态域s (status field), PSR[23:16];
- 扩展域x (extend field), PSR[15:8];
- 控制域c (control field), PSR[7:0]。



（1）条件标志位：N、Z、C和V统称为条件标志位，这些标志位会根据程序中的算术和逻辑指令的执行结果修改。处理器则通过测试这些标志位来确定一条指令是否执行。

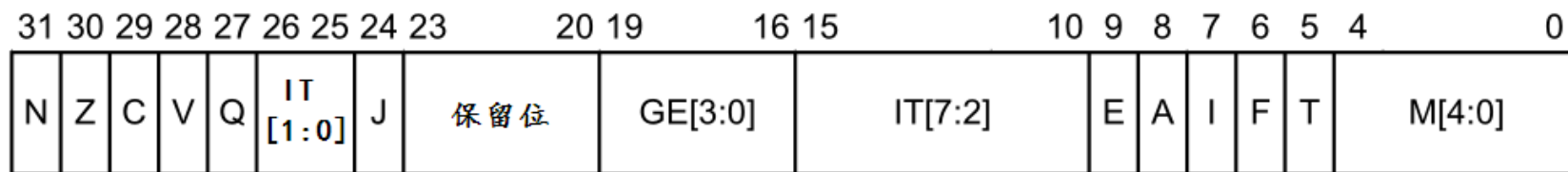
- N（Negative）：N=1表示运算的结果为负数，N=0表示结果为正数或零。
- Z（Zero）：Z=1表示运算的结果为零，Z=0表示运算的结果不为零。
- C（Carry）：在加法指令中，当结果产生了进位，C=1，其它情况C=0；在减法指令中，当运算发生了借位，C=1，其它情况C=0；在移位运算指令中，C被设置成被移位寄存器最后移出去的位。
- V（Overflow）：对于加/减法运算指令，当操作数和运算结果为二进制补码表示的带符号数时，V=1表示符号位溢出。



(2) **Q标志位**：在带有**DSP**指令扩展的**ARMv5**及以上版本中，**Q**标志位用于指示增强的**DSP**指令是否发生了溢出。**Q**标志位具有粘性，当因某条指令将其设置为**1**时，它将一直保持为**1**直到通过**MSR**指令写**CPSR**寄存器明确地将该位清**0**，不能根据**Q**标志位的状态来有条件地执行某条指令。

(3) **IT块**：**IT**块用于对**Thumb**指令集中**if-then-else**这一类语句块的控制。如果有**IT**块，则**IT[7: 5]**为当前**IT**块的基本条件码。在没有**IT**块处于活动状态时，该**3**位为**000**。**IT[4: 0]**表示条件执行指令的数量，不论指令的条件是基本条件码或是基本条件的逆条件码。在没有**IT**块处于活动状态时，该**5**位为**00000**。当处理器执行**IT**指令时，通过指令的条件和指令中**Then**、**Else**（**T**和**E**）参数来设置这些位。

(4) **J标志位**：用于表示处理器是否处于**ThumbEE**状态。**T=1**时，
 ➤ **J=0**，表示处理器处于**Thumb**状态。
 ➤ **J=1**，表示处理器处于**ThumbEE**状态。
 注意：**T=0**时，不能够设置**J=1**；当**T=0**时，**J=0**。不能通过**MSR**指令来改变**CPSR**的**J**标志位。



（5）**GE[3: 0]**位：该位用于表示在**SIMD**指令集中的大于、等于标志。在任何模式下可读可写。

（6）**E**标志位：该标志位控制存取操作的字节顺序。**0**表示小端操作，**1**表示大端操作。**ARM**和**Thumb**指令集都提供指令用于设置和清除**E**标志位。当使用**CFGEND0**信号复位时，**E**标志位将被初始化。

（7）**A**标志位：表示异步异常禁止。该位自动置为**1**，用于禁止不精确的数据中止。

（8）控制位：程序状态寄存器的低**8**位是控制位。当异常发生时，这些位的值将发生改变。在特权模式下，可通过软件编程来修改这些标志位的值。

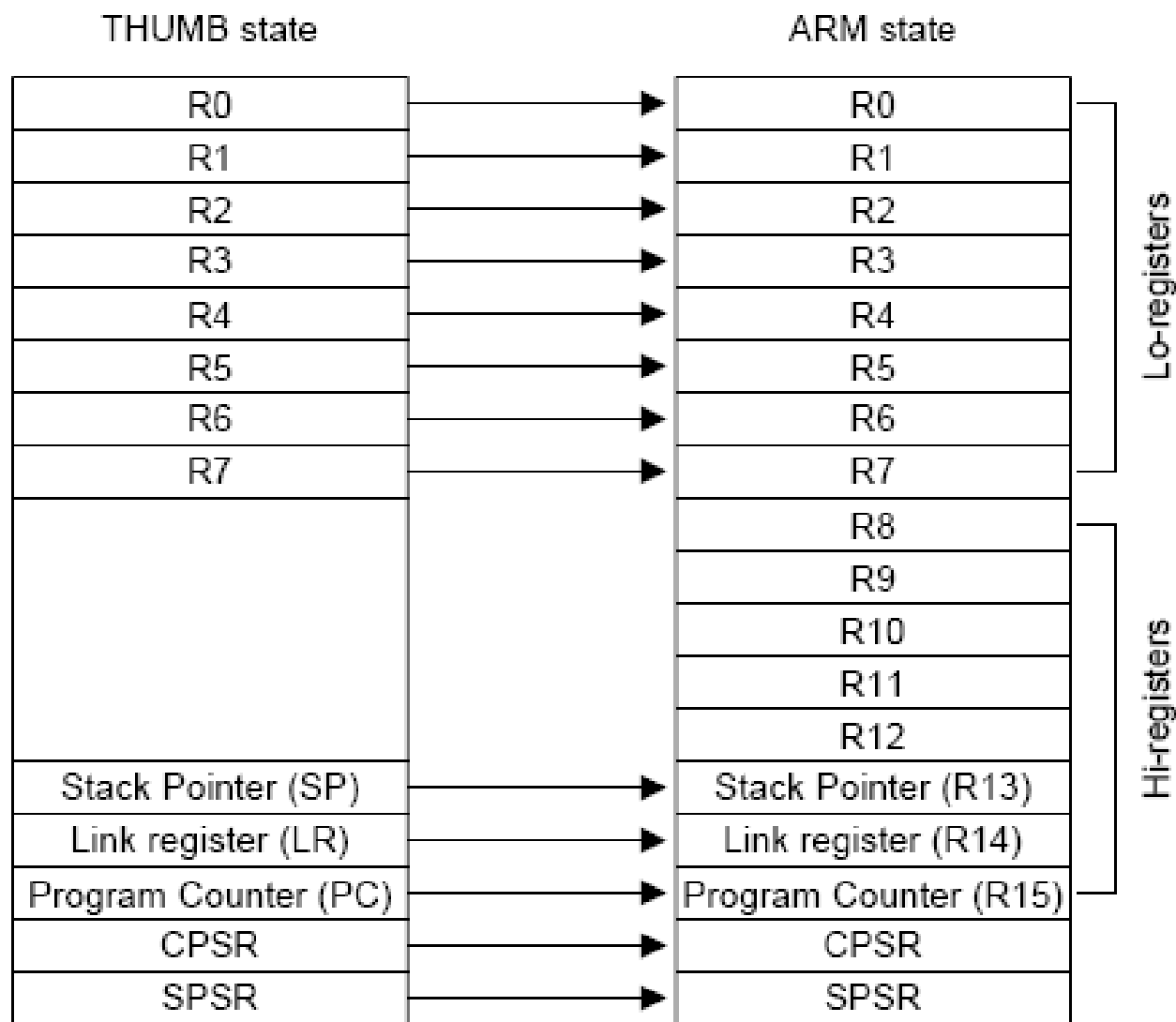
➤中断屏蔽位：**I=1**，**IRQ**中断被屏蔽；**F=1**，**FIQ**中断被屏蔽。

➤状态控制位：**T=0**，处理器处于**ARM**状态；**T=1**，处理器处于**Thumb**状态。

➤模式控制位：**M[4: 0]**为模式控制位，决定处理器的工作模式，如表2-3所示。

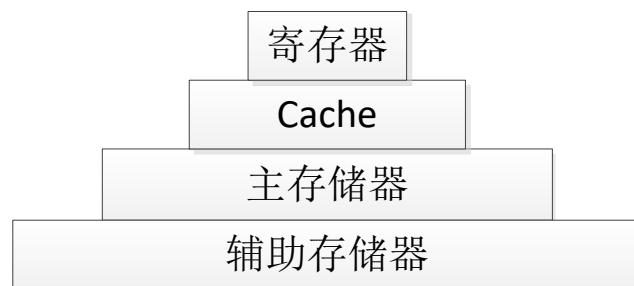
M[4: 0]	0b10000	0b10001	0b10010	0b10011	0b10111	0b11011	0b11111	0b10110
工作模式	User	FIQ	IRQ	Supervisor	Abort	Undefined	System	Secure Monitor

ARM State 与Thumb State寄存器关系



2.4.5 Cortex-A8存储系统

在ARM 嵌入式系统设计中，按照不同的存储容量、存取速度和价格，将存储器系统的层次结构分为4级：



寄存器包含在CPU内部，用于指令执行时的数据存放，如上节介绍的Cortex-A8处理器寄存器组。Cache是高速缓存，暂存CPU正在使用的指令和数据。主存储器是程序执行代码和数据的存放区，像DDR2 SDRAM存储芯片。辅助存储器类似PC机中的硬盘，在嵌入式系统中常采用FLASH芯片。

整个存储结构又可以被看成两个层次：主存-辅存层次和Cache-主存层次。

1.协处理器CP15

在ARM系统中，实现对存储系统的管理通常使用的是协处理器CP15，也被称为系统控制协处理器（**System Control Coprocessor**）。ARM处理器支持16个协处理器。

程序在执行过程中，每个协处理器忽略属于ARM处理器和其它协处理器的指令。当一个协处理器不能执行属于它的协处理器指令时，将产生一个未定义指令异常中断。

CP15负责完成大部分的存储器管理。当在一些没有标准存储管理的系统中，CP15是不存在的。针对CP15的指令将被视为未定义指令，指令的执行结果是不可预知的。

CP15有16个32位寄存器，编号0~15。某些编号的寄存器可能对应多个物理寄存器，在指令中指定特定的标志位来区分这些物理寄存器，类似于ARM中的寄存器。处于不同的处理器模式时，ARM某些寄存器可能不同。

2.内存管理单元MMU

面对一些复杂的、多任务的嵌入式应用时，常使用嵌入式操作系统来管理整个系统和任务的运行。高级的嵌入式操作系统都带有内存管理单元MMU

（Memory Management Unit）来管理每个任务各自的存储空间。

当应用程序的规模不断增大，超过了内存的增长速度，以至于内存存放不下应用程序时，需要采用了虚拟内存（Virtual Memory）技术。虚拟内存的基本思想就是程序、数据、堆栈的总大小可以超过物理内存的大小，操作系统把当前使用的部分保存在内存中，其它未使用的部分保存在外存上。

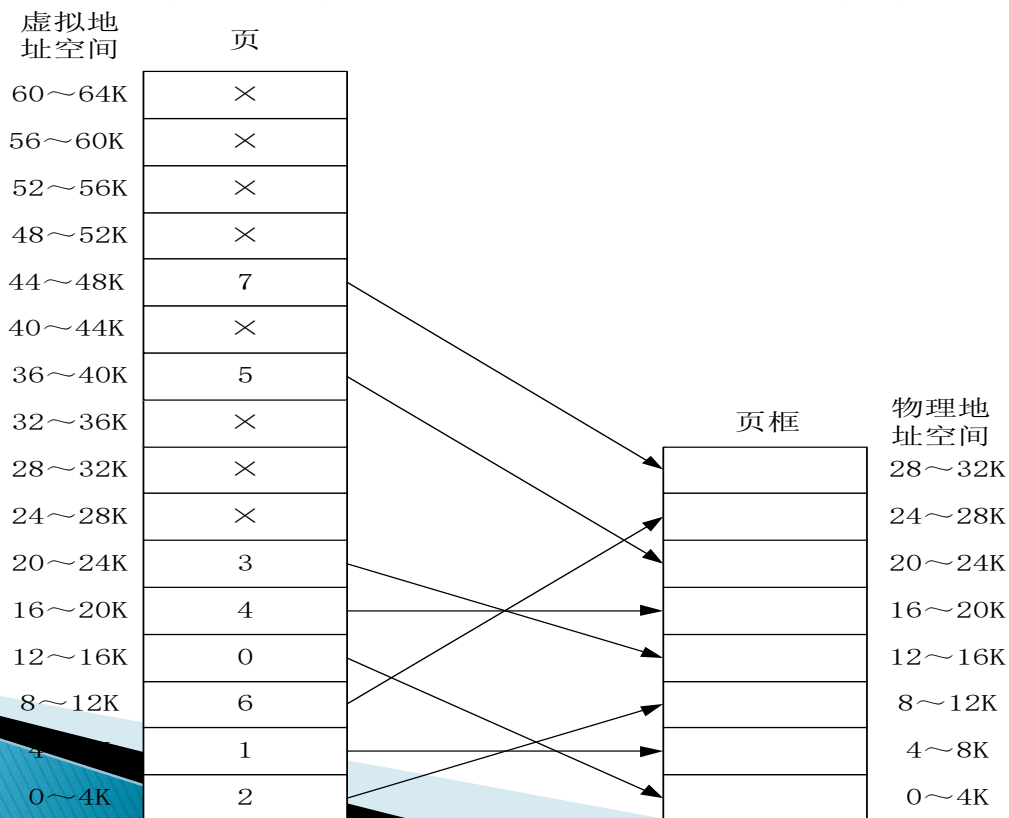
任何时候，计算机上都有一个程序能够产生的地址集合，称为地址范围。该范围由CPU的位数决定，如32位的Cortex-A8处理器S5PV210，它的地址范围是0~0xFFFF_FFFF（4GB）。这个范围就是程序能够产生的地址范围，这个地址范围就称为虚拟地址空间，该空间中的任一地址就是虚拟地址。

与虚拟地址和虚拟地址空间对应的就是物理地址和物理地址空间，大多数时候物理地址空间只是虚拟地址空间的一个子集。例如，内存为256M的32位嵌入式系统中，虚拟地址空间范围是0~0xFFFF_FFFF（4GB），而物理地址空间是0~0x0FFF_FFFF（256MB）。

虚拟地址由编译器和连接器在定位程序时分配；物理地址用来访问实际的主存储器硬件模块。在ARM中采用了页（Page）式虚拟存储管理方式。虚拟地址空间划分成称为页的单位，而相应的物理地址空间也被进行划分，单位是页帧（frame）。

页与页帧之间在MMU的调度下是如何进行映射：

虚拟地址0被送往MMU，MMU发现该地址在页0（0~4095）的范围内，页0所映射的页框为2（页框2的地址范围是8192~12287）。因此MMU将该虚拟地址转化为物理地址8192，并把地址8192送到内存地址总线上。内存对地址的映射过程并不清楚，它只是接收到一个对地址8192的访问请求并执行。



MMU的一个重要任务就是让每个任务都运行在各自的虚拟存储空间中。MMU做为转换器，将程序和数据的虚拟地址转换成实际的物理地址，也就是实现虚拟存储空间到物理存储空间的映射。

除此之外，MMU的其他重要功能还包括：

- （1）存储器访问权限的控制，提供硬件机制的内存访问授权。
- （2）设置虚拟存储空间缓冲的特性。

MMU中的虚拟地址转换成实际的物理地址的变换过程是通过两级页表实现的：

（1）一级页表中包含有以段为单位的地址变换条目以及指向二级页表的指针。一级页表实现的地址映射力度较大。以段为单位的地址变换过程只需要一级页表。

（2）二级页表中包含有以大页和小页为单位的地址变换条目。有一种类型的二级页表还包含有以极小页为单位的地址变换条目。以页为单位的地址变换过程需要二级页表。

3.高速缓冲存储器Cache

Cache是一种小容量、高速度的存储器，用于处理器与主存之间存放当前被使用的主存内容，以减少访问主存的等待时间。**Cache**通常和处理器核在同一个芯片上。对于程序员来说，**Cache**是透明的。

Cache一般和写缓存一起使用。写缓存是一个非常小的先进先出（**FIFO**）存储器，位于处理器核和主存之间。使用写缓存是为了将处理器核和**Cache**从较慢的主存写操作中解脱出来。当**CPU**向主存储器写入时，先将数据写入到写缓存区中，由于写缓存的速度很高，这种写入操作的速度也很高。写缓存在**CPU**空闲时，以较低的速度将数据写入到主存储器中相应的位置。

Cache放置数据的常用地址变换方法有直接映像、组相联映像、和全相联映射方式。

替换算法有随机法、近期最少使用法和循环法。

存储器写策略采用写直达法、通过缓存写直达法和写回法。

2.5

Part Five

Cortex-A8异常处理

异常是**ARM**处理器处理外部异步事件的一种方法，也称为中断。当处理器在正常执行程序的过程中，一个来自外部或内部的异常事件发生，处理器暂时中断当前程序的执行，跳转到相应的异常处理程序入口执行异常处理。在处理这个异常事件之前，处理器要保存当前处理器的状态和返回地址，以便异常处理程序结束后能返回原来的程序继续执行。若同时有多个异常发生，处理器将根据异常中断优先级来处理这些异常。

2.5.1 异常向量和优先级

在ARM体系结构中有7种异常中断。异常发生时，处理器会将PC寄存器设置为一个特定的存储器地址，这些特定的存储器地址称为异常向量。所有异常的异常向量被集中放在程序存储器的一个连续地址空间中，称为异常向量表。每个异常向量只占4个字节，异常向量处是一些跳转指令，跳转到对应的异常处理程序。

通常存储器地址映射地址0x00000000是为异常向量表保留的。但某些嵌入式系统中在系统配置使能时，低端的异常向量表可以选择映射到特定的高端地址0xFFFF0000处。一些嵌入式操作系统，如Linux和Windows CE就利用了这一特性。要说明的是Cortex-A8处理器支持通过设置协处理CP15的C12寄存器将异常向量表的首地址设置在任意地址。

表2-6列出了ARM的7种异常类型、异常发生后，处理器进入的异常模式、异常的优先级对应的异常向量地址。1优先级最高，6优先级最低。

异常类型	处理器模式	优先级	异常向量地址		说明
			低端	高端	
复位异常 (Rest)	管理模式	1	0x00000000	0xFFFF0000	RESET复位脚有效时进入该异常，常用在系统加电时和系统复位时
未定义指令异常 (Undefined Interrupt)	未定义指令模式	6	0x00000004	0xFFFF0004	ARM处理器或协处理器遇到不能处理的指令时产生该异常，可利用该异常进行软件仿真
软件中断异常 (SWI)	管理模式	6	0x00000008	0xFFFF0008	由SWI指令产生，可用于用户模式下的程序调用特权操作
指令预取中止异常 (Prefetch Abort)	未定义指令中止模式	5	0x0000000C	0xFFFF000C	当预取指令不存在或该地址不允许当前指令访问时产生该异常，常用于虚存和存储器保护
数据访问中止异常 (Data Abort)	数据访问中止模式	2	0x00000010	0xFFFF0010	数据访问指令的目标地址不存在或该地址不允许当前指令访问时产生该异常，常用于虚存和存储器保护
外部中断异常 (IRQ)	外部中断模式	4	0x00000018	0xFFFF0018	处理器的外部中断请求引脚有效，且CPSR的I位为0时，产生该异常，常用于系统外设请求
快速中断异常 (FIQ)	快速中断模式	3	0x0000001C	0xFFFF001C	该异常是为了支持数据传送或通道处理而设计的，处理器的快速中断请求引脚有效，且CPSR的F为0时，产生该异常。

在上表中可以看到，每一种异常都会导致内核进入一种特定的模式。此外，也可以通过编程改变**CPSR**，进入**ARM**处理器模式。要说明的是，用户模式和系统模式是仅有的不可通过异常进入的两种模式，也就是说，要进入这两种模式，必须通过编程改变**CPSR**。

当多个异常同时发生时，由系统根据不同异常的优先级按照从高到低的顺序处理。**7**种异常分成**6**个级别，**1**优先级最高，**6**优先级最低。其中，未定义指令异常和软件中断异常都依靠指令的特殊译码产生，这两者是互斥的，不可能同时产生。

2.5.2 异常相应过程

通常，一般异常（中断）响应大致可以分为以下几个步骤：

- （1）保护断点，即保存下一个将要执行的指令的地址，就是把这个地址送入堆栈；
- （2）寻找中断入口，根据不同的中断源所产生的中断，查找不同的入口地址；
- （3）执行中断处理程序；
- （4）中断返回，执行完中断指令后，就从中断处返回到主程序，继续执行。

具体到**ARM**处理器中，当异常发生后，除了复位异常会立即中止当前指令以外，其余异常都是在处理器完成当前指令后再执行异常处理程序。**ARM**处理器对异常的响应过程如下：

(1) 进入与特定的异常相应的运行模式。

(2) 将**CPSR**寄存器的值保存到将要执行的异常中断各自对应的**SPSR_mode**中，以实现处理器当前运行状态、中断屏蔽和各标志位的保护。

(3) 将引起异常指令的下一条指令的地址存入相应的链接寄存器**LR (R14_mode)**，以便程序在异常处理结束返回时能正确的返回到原来的程序处继续向下执行。若异常是从**ARM**态进入的，则链接寄存器**LR**保存的是下一条指令的地址（根据不同的异常类型，当前**PC+4**或**PC+8**）；若异常是从**Thumb**状态进入的，则将当前**PC**的偏移量值保存到**LR**中，这样异常处理程序就不需要确定异常是从何种状态进入的。

(4) 设置CPSR寄存器的低5位，使处理器进入相应的工作模式。设置I=1，以禁止IRQ中断；如果进入复位模式或FIQ模式，还要设置F=1以禁止FIQ中断。

(5) 根据异常类型，将表2-4中的向量地址强制复制给程序计数器PC，以便执行相应的异常处理程序。

以复位异常为例，比如存储器地址映射地址0x00000000是为异常向量表保留的情况下，则PC=0x00000000，如果异常向量表选择映射到特定的高端地址0xFFFF0000处，则PC = 0xffff0000。

每种异常模式对应两个寄存器R13_mode和R14_mode（mode为svc、irq、und、fiq或abt之一），分别存放堆栈指针和断点地址。

每种异常模式对应两个寄存器**R13_mode**和**R14_mode**（mode为svc、irq、und、fiq或abt之一），分别存放堆栈指针和断点地址。

举例

1. 整个地址空间的起始位置（地址0x00000000开始）有以下指令

	对应地址
b SYS_RST_HANDLER ;	0x00000000
b UDF_INS_HANDLER ;	0x00000004
b SWI_SVC_HANDLER ;	0x00000008
b INS_ABT_HANDLER ;	0x0000000c
b DAT_ABT_HANDLER ;	0x00000010
b . ;	
b IRQ_SVC_HANDLER ;	0x00000018
b FIQ_SVC_HANDLER ;	0x0000001c

➤ 一旦发生外部中断请求，处理器首先自动保存当前状态，进入外部中断模式 **PC->R14 CPSR->SPSR**

➤ 接着执行地址0x00000018处的指令，即b IRQ_SVC_HANDLER
跳转到标号IRQ_SVC_HANDLER处开始执行

举例

2. IRQ_SVC_HANDLER处的代码为:

IRQ_SVC_HANDLER

```
sub    lr, lr, #4  
stmfd  sp!, {r0-r3, lr}
```

```
ldr    r0, =IRQ_SVC_Vector
```

```
ldr    pc, [r0]
```

处理器将通用寄存器和返回地址压入堆栈

IRQ_SVC_Vector为外部中断请求的中断向量

跳转到外部中断请求的中断服务程序中

2.5.3 异常返回过程

复位异常发生后，由于系统自动从0x00000000开始重新执行程序，因此复位异常处理程序执行完后无须返回。其它异常处理完后必须返回到原来程序的断点处继续执行。

ARM处理器从异常处理程序中返回的过程如下：

- （1）恢复原来被保存的用户寄存器。
- （2）将SPSR_mode寄存器的值复制到CPSR中，以恢复被中断的程序工作状态。
- （3）根据异常类型将PC寄存器值恢复成断点地址，以执行原来被中断打断的程序。
- （4）清除CPSR中的中断屏蔽标志位I和F，开放外部中断和快速中断。

当返回地址保存在当前模式的R14_mode中时，从不同的模式返回，所用的指令有所不同。

异 常	返回地址	说明
复位	—	复位没有定义LR
数据中止	LR-8	指向导致数据中止异常的指令
FIQ	LR-4	指向发生异常时正在执行的指令
IRQ	LR-4	指向发生异常时正在执行的指令
预取指令中止	LR-4	指向导致预取指令异常的那条指令
SWI	LR	执行SWI指令的下一条指令
未定义指令	LR	指向未定义指令的下一条指令

下面是不同异常处理之后返回原程序的方法：

(1) FIQ和IRQ的返回指令为

SUB PC, R14_FIQ, #4

SUB PC, R14_IRQ, #4

FIQ和IRQ必须返回前一条指令，以便执行因为进入异常而未执行的指令。所以该返回指令将寄存器R14_FIQ/R14_IRQ的值减4后复制到程序计数器PC中，以实现从异常处理程序中返回，同时将SPSR_fiq/SPSR_irq寄存器的值复制到当前程序状态字寄存器CPSR中。

(2) ABORT的返回指令为

SUB PC, R14_abt, #4 指令预取中止返回

SUB PC, R14_abt, #8 数据中止返回

该指令从R14_abt、SPSR_abt恢复PC和CPSR的值，以实现从异常处理程序中返回。

指令预取中止必须返回前面第一条指令，以便执行在初次请求访问时造成存储器故障的指令。

数据中止必须返回前面第二条指令，以便重新执行因为进入异常而被占据指令之前的数据传输指令。

(3) SWI/未定义指令的返回指令为

MOVS PC, R14_svc

MOVS PC, R14_und

该指令从R14_svc/ R14_und和SPSR_svc/ SPSR_und恢复PC和CPSR的值，并返回到SWI/未定义指令的下一条指令。

如果异常处理程序把返回地址复制到堆栈中（当发送相同的异常嵌套时，为了能够再次进入中断，SPSR也必须和PC一样被保存）可以使用一条多寄存器传送指令来恢复用户寄存器并实现返回，即

LDMFD R13!, {R0-R3, PC}^

其中，寄存器列表后面的“^”表示从堆栈中装载PC的同时，也将CPSR恢复。

2.5.4 Cortex-A8处理器 S5PC100中断机制

S5PC100集成了3个向量中断控制器(VIC)，支持94个中断源。

一个向量中断控制器（Vector Interrupt Controller）是用来处理多个中断源的外围设备，通常包含以下几个特性：

- （1）确定请求服务的中断源以及确定中断处理程序的地址。
- （2）为每个中断源分配一个中断请求输入端口。为每个中断请求分配一个中断请求输出端口，以能连接到处理器的VIC端口。
- （3）可以用软件屏蔽掉任意制定的中断源的中断。
- （4）可以为每个中断设置优先级。

S5PC100中断控制器的特点主要有：

- 灵活的硬件中断优先级以及可编程的中断优先级设置；
- 支持硬件上的优先级屏蔽和编程上的优先级屏蔽；
- 内置**IRQ/FIQ**/软件中断产生器、用于调试方案的寄存器和原始中断状态寄存器/中断源请求状态寄存器；
- 支持特权模式下的限制性存取数据。

Cortex-A8提供了2种中断模式，即**FIQ**模式和**IRQ**模式。所有的中断源在中断请求时都要确定使用哪一种中断模式。

当处理器收到来自片内外设和外部中断请求引脚的多个中断请求时，**S5PC100**的中断控制器在中断仲裁过程后向**S5PC100**处理器内核请求**FIQ**或**IRQ**中断。中断仲裁过程依靠处理器的硬件优先级逻辑，在处理器这边会跳转到中断异常处理例程中，执行异常处理程序，这个时候**VICADDRESS**寄存器的值就是仲裁后中断源对应的（**ISR**）中断处理程序的入口地址。

S5PC100的中断控制器的最主要任务是在有多个中断发生时，选择其中一个中断通过**IRQ**或**FIQ**向**CPU**内核发出中断请求。实际上，最初**CPU**内核只有**FIQ**和**IRQ**两种中断，其他中断都是各个芯片厂家在设计芯片时，通过加入一个中断控制器来扩展定义的，这些中断根据中断的优先级高低来进行处理，更符合实际应用系统中要求提供多个中断源的要求，除此之外，向量中断控制器比以前的中断方式更加灵活，方便，把判断的任务留给了硬件，使得中断编程更为简洁。比如**PL192**核心**VIC**就可以通过**VICVECTADDROUT[31:0]**这个端口获取当前中断的中断服务程序**ISR**入口，这比**ARM9**系列处理器采用的强制跳转至**0x00000018**或者**0xffff0018**的策略有了很大的进步。但是处理器的**VIC**端口不支持读 **FIQ** 的向量地址。

2.6

Part Six

本章小结

本章对**ARM**处理器和相应体系结构的演变以及主要**ARM**处理器产品进行了介绍。详细说明了**Cortex-A8**处理器的工作模式和工作状态、存储器组织和异常处理流程。随着**ARM**处理器的不断改进，其体系结构也日趋复杂，但**ARM**处理器的性能对嵌入式系统有着至关重要的影响。嵌入式设计人员必须熟悉所采用的**ARM**处理器的结构和性能，更多更详细的资料可到**ARM**公司官网上获取和阅读。