江苏师范大学 电气工程及自动化学院
JIANGSU NORMAL UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING & AUTOMATION





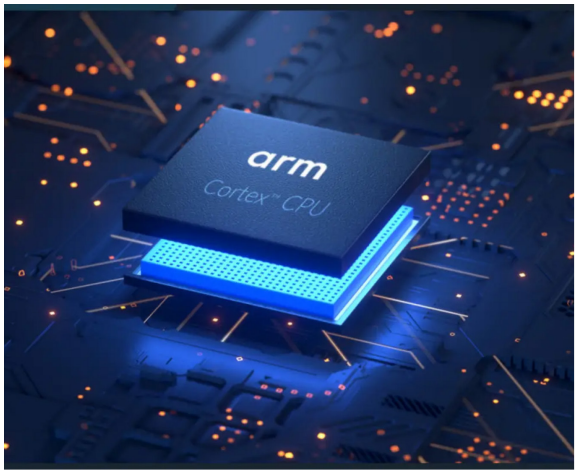
第2章 ARM体系结构

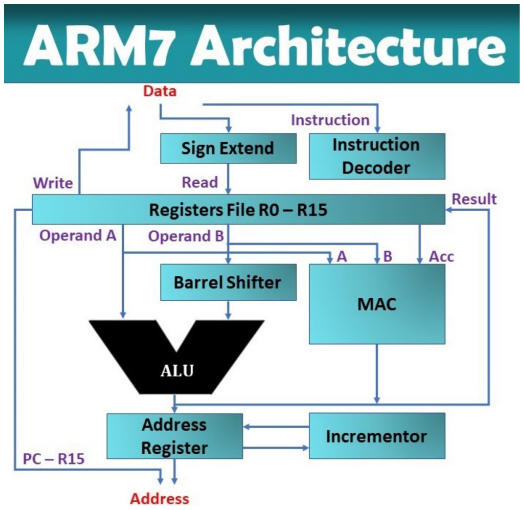
李灿 | 12#407A | lic@jsnu.edu.cn | https://sslic.cn/es

ARM

- **ARM——Advanced RISC Machines**
  - ✧ 公司的名字（ARM公司1991年成立于英国剑桥）
  - ✧ 一类微处理器的通称（ARM处理器）
  - ✧ 一种技术（ARM公司出售芯片设计技术的授权）







ARM

■ ARM公司的Chipless模式

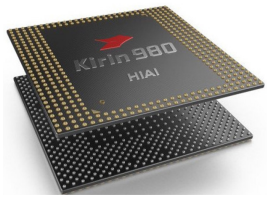
世界各大半导体生产商从ARM公司购买其设计的ARM微处理器核

根据各自不同的应用领域，加入适当的外围电路，从而形成自己的ARM微处理器芯片进入市场

基于ARM技术的微处理器应用约占据了32位RISC微处理器75%以上的市场份额

ARM技术已经渗入到我们生活的各个方面

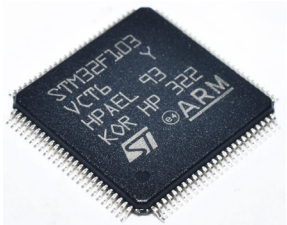
我国的华为海思、中兴集成电路、大唐电讯、中芯国际、上海华虹，紫光展锐，以及国外的一些公司如德州仪器、意法半导体、Philips、Intel、Qualcomm、三星等都推出了基于ARM核的处理器













ARM：合作模式


ATAP Partners
WIN-FINITY
BARCO SILEX
SOTA
DNP
Infinite Technology Corporation
SIEMENS
NSW
STEPMIND
Think
MacroTech Research, Inc.
COMIT SYSTEMS
YOGITECH
ARCADIA
SIDSA
TOPPAN
WIPRO
SEODU INCHIF
ZAIQ
HOYA
INI CORE
MAZ
FARADAY
SCIworx
nordic
TALITY
parthus
SYNOPSYS
FLEXTRONICS Semiconductor
Tools Partners
EPI
ASHLING
CoWare
Virtio
Green Hills
INNOVEDA.
Computex
YOKOGAWA
ADS
Tektronix
WindRiver
Sophia systems
Axis systems
Verisity
Aptix
Agilent Technologies
COGENT
Aliant
ALCATEL
IKOS SYSTEMS
SYNOPSYS
Mentor Graphics
LAUTERBACH
cadence
QUALCOMM
ZTEIC
GOODRICH
MITSUBISHI ELECTRIC
FUJITSU
ERICSSON
OKI
MOTOROLA
ADMtek
NEC
NEOMAGIC
TOSHIBA
intersil
AMI
MICRONAS
PHILIPS
UMC
EPSON
Agilent Technologies
Chartered
SANYO
ROHM
ANALOG DEVICES
ALCATEL
ARM
in Partnership
TEXAS INSTRUMENTS
AKM
CIRRUS LOGIC
Panasonic
3Com
LSI LOGIC
agere systems
ZEEVO
intel
HYUNDAI
BROADCOM
SHARP
YAMAHA
IBM
Virata
SONY
Triscend.
ALTERA
cogency
PRAIRIECOMM
Infineon
ZARLINK
ArrayComm
KAWASAKI MICRO
Silicon Wave
RESONEXT
National Semiconductor
FIRMWARE SYSTEMS
realogy
eSOL
jmi
QNX
LINEO
Accelerated Technology Inc
ACCESS
PRECISE
US Software
GEOWORKS
Tao Systems
KADAK Products Ltd.
Microsoft
JAVA
OSE systems
FirmWorks
AXE
Eonic
symbian
Sun
WindRiver
MICROWARE
ETNOTEAM
RTOS Partners
Embedded System Products
LYNUXWORKS
CMX COMPANY
interniche technologies, inc.
Microsoft
EMBLAZE Cellular
FRONTIER
Packet Video
INTERTRUST
ERICSSON
ZI corporation
liquid audio
Bluetooth
symbian
QNX
Dolby
Symmetricom
CPS
JAVA
ASAHI CHEMICAL INDUSTRY CO.,LTD.
Software Partners

ARM：特点

- 采用RISC架构的ARM微处理器一般具有如下特点：
  - 小体积、低功耗、低成本、高性能
  - 32位/16位双指令集（ARM+Thumb）
  - 全球众多的合作伙伴

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

- ✧ 计算机体系结构
- ✧ RISC技术
- ✧ 流水线技术

2.1.1 计算机体系结构

- 计算机的体系结构：遵循冯•诺依曼提出的“存储程序”模式

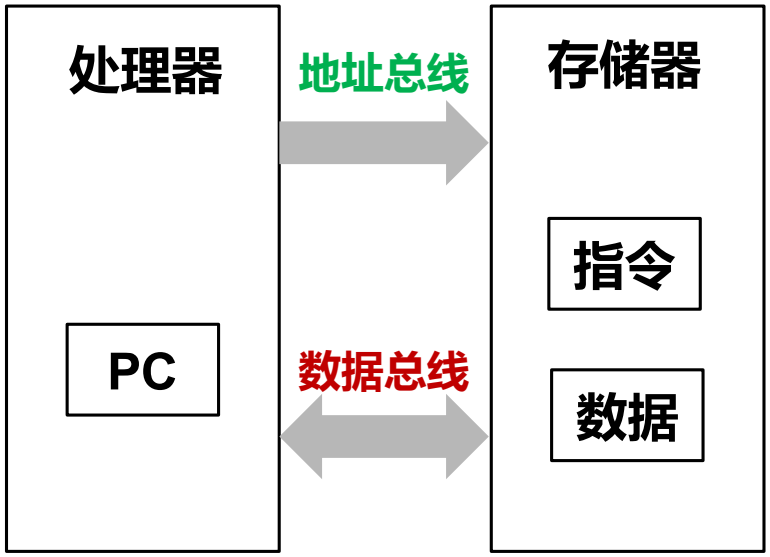


图2.1 冯•诺依曼体系结构

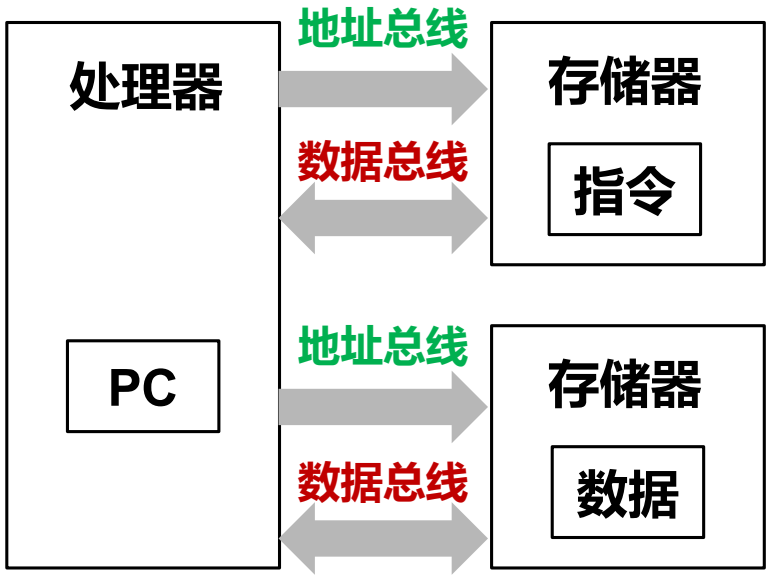


图2.2 哈佛体系结构

2.1.1 计算机体系结构

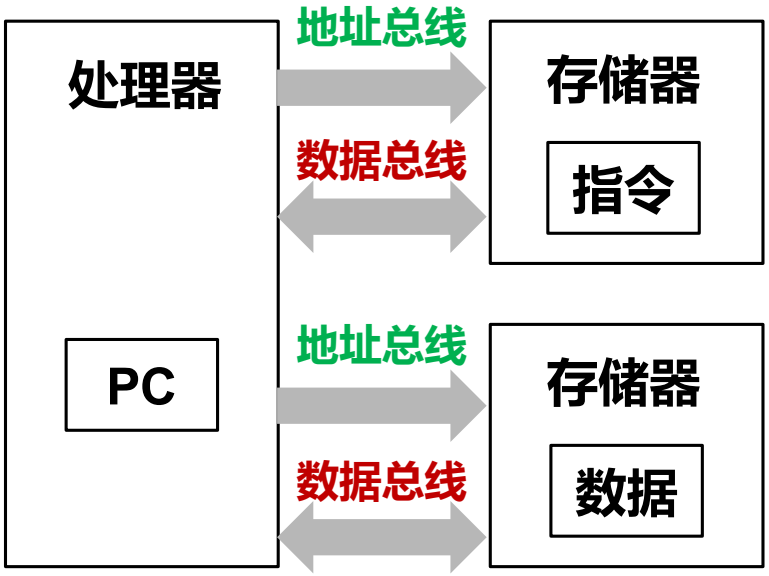


图2.2 哈佛体系结构

- 哈佛体系结构的特点：
  - 程序存储器与数据存储器分开
  - 提供了较大的存储器带宽
  - 适合于数字信号处理
  - 大多数DSP都是哈佛结构

2.1.2 RISC技术

- **复杂指令集计算机**
  - CISC（Complex Instruction Set Computer）
  - 偏重于由硬件执行指令
  - 主要趋势是增加指令集复杂度
- **精简指令集计算机**
  - RISC（Reduced Instruction Set Computer）
  - 目标：设计一套能在高时钟频率下周期执行、简单有效的指令集
  - 中心思想是精简指令集的复杂度：
    简化指令实现的硬件设计，硬件只执行很有限的最常用的那部分指令，大部分复杂的操作则由简单指令合成。
  - 获益：大幅提高计算机的性价比，以ARM为代表，成功商业化！

2.1.2 RISC技术

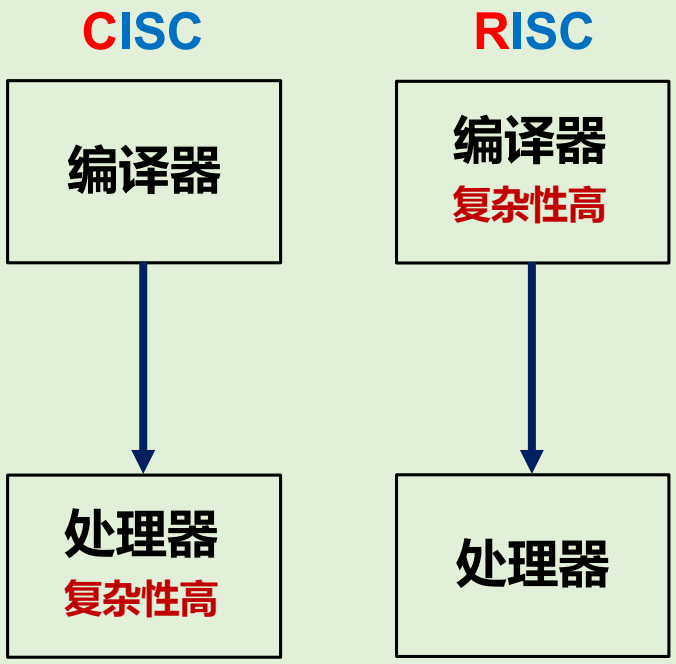


图2.3 CISC 和 RISC

注意：CISC和RISC各有各有优势，界限也并非绝对！

- **RISC的体系结构的特点：**
  - ✧ 使用大量通用寄存器组
    每个寄存器都可存放数据或地址
  - ✧ 采用简单寻址模式
  - ✧ 采用固定长度指令格式
    简化指令解码
  - ✧ 采用流水线技术

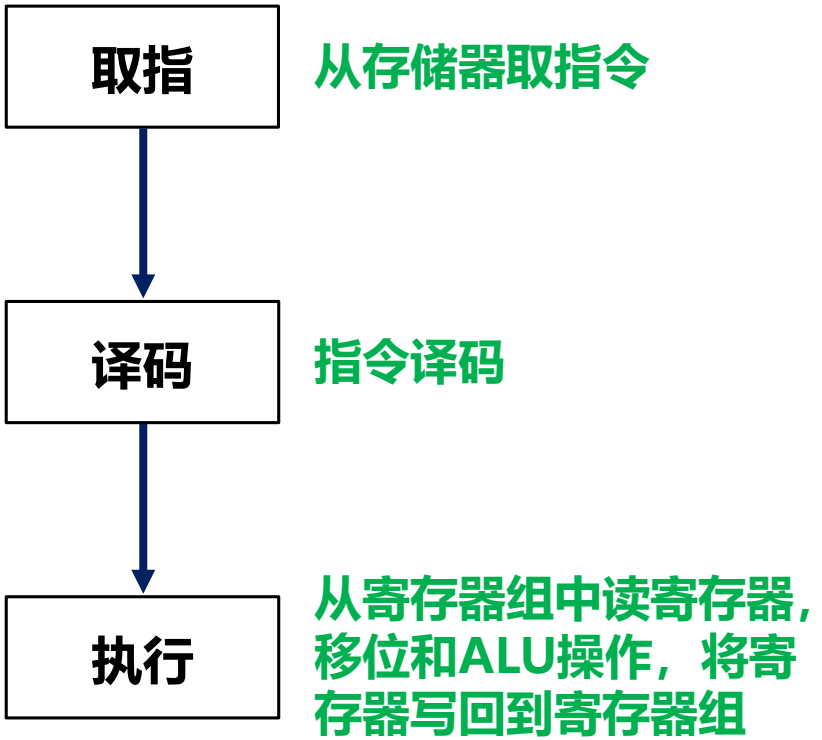
✧ 在Load/Store结构中：处理器只处理在寄存器中的数据，而不直接处理存储器中的数据，以提高指令的执行效率。一般通过Load/Store指令完成数据在寄存器和存储器之间的传送。

2.1.3 流水线技术

- **流水线：** RISC处理器执行指令时采用的机制
- **原因：**
  - ✧ 处理器的指令变得越来越复杂，往往需要多个时钟周期才能实现
  - ✧ 充分利用处理器执行多周期指令过程中处于空闲状态的总线资源
- **优点：** 处理器执行多周期指令时，流水线可以同时使用这些指令时钟周期向指令队列中取指令
- **缺点：** 执行跳转指令需丢弃流水线中所有预取指令；执行第1条指令前要填充流水线；某些段之间可能产生数据相关
- **典型：**
  - ✧ ARM7的内核一般都为三级流水线
  - ✧ ARM9的内核把流水线增加到了五级
  - ✧ ARM10中的内核流水线到达了六级

2.1.3 流水线技术

■ 三级流水线与功能段划分



可在取下一条指令的同时译码和执行其他指令，加快执行速度

图2.4 指令流水线功能段

2.1.3 流水线技术

■ 举例

```
ADD   R0, R1, R2
SUB   R3, R4, #2
CMP   R5, R1
```

取指	译码	执行	存储	写

图2.6 五级流水线

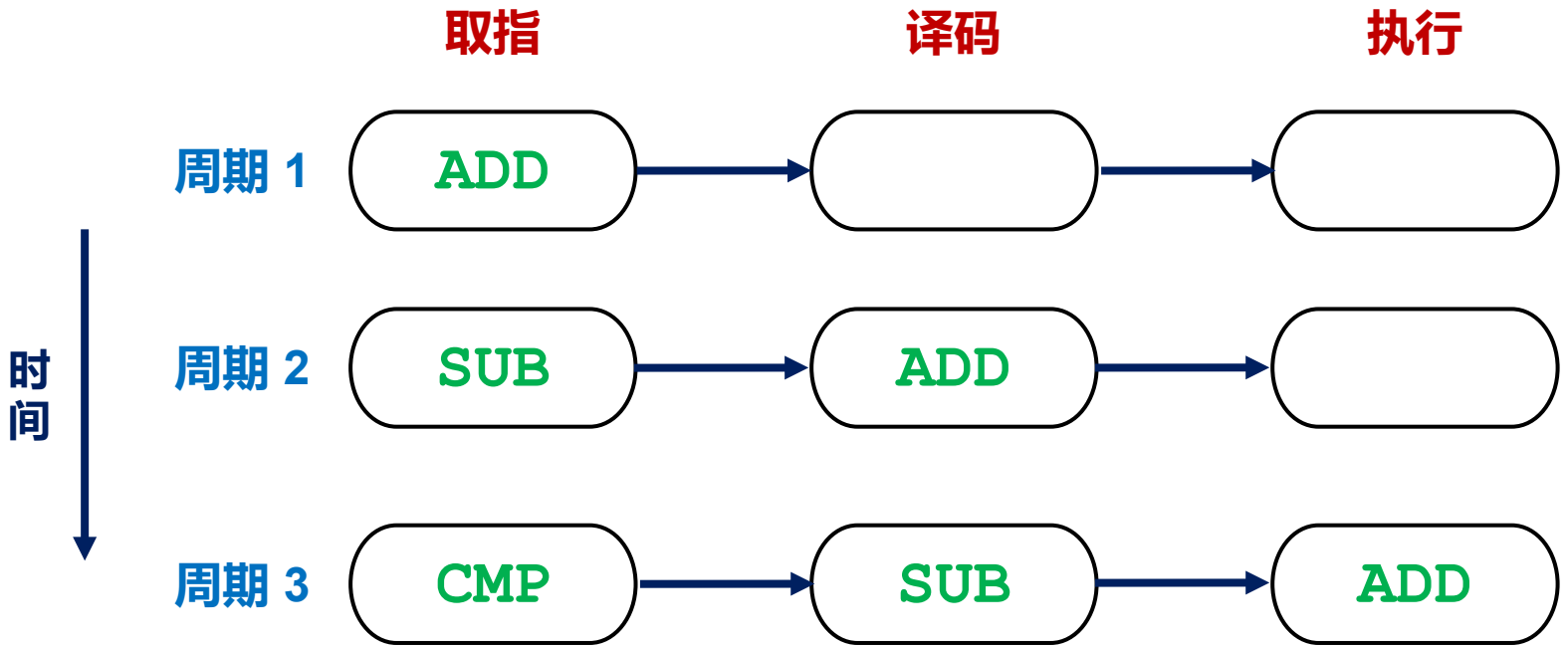


图2.5 三级流水线

章节内容

- RISC技术和流水线技术
- **ARM体系结构简介**
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

- ✧ ARM体系结构演变
- ✧ ARM体系结构特征
- ✧ ARM体系的变种
- ✧ ARM系列
- ✧ ARM存储数据类型

2.2 ARM体系结构简介

ARM核的命名规则

ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{S}

- x：系列
  - x=7：AMR7处理器核
  - x=9：ARM9处理器核
- y：存储管理/保护单元
  - y=2：带cache和MMU
  - y=3：物理地址标记的cache和MMU
  - y=4：cache和MPU
  - y=6：写缓冲但无cache大小
- z：Cache
  - z=0：标准cache大小
  - z=2：缩小的cache
  - z=6：包含紧密耦合SRAM（TCM）

2.2 ARM体系结构简介

ARM核的命名规则

ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{S}

- T：支持Thumb指令集，ARM V6及以后的内核都自动包含T
- D：支持片上Debug
- M：内嵌硬件乘法器
- I：嵌入式ICE(In-Circuit Emulator)，支持片上断点和调试点
- E：DSP运算的增强指令
- J：含有Java加速器Jazelle，速度快于Java虚拟机（8倍）
- F：ARM核通过向量浮点（VFP）结构支持硬件浮点
- S：可综合版本，意味着处理器内核是以源代码形式提供的
  这种源代码形式又可以被编译成一种易于EDA工具使用的形式

2.2.1 ARM体系结构演变

- **ARM有9个版本的体系架构（内核版本）**
  - V1~V6版本：经典版本
  - Cortex版本：V7-2004，V8-2011，V9-2021
- **体系结构的版本:**
  - 代表了一套指令集定义和相应的功能框架
  - V8以前所有的体系结构保持了良好的向下兼容性

2.2.1 ARM体系结构演变

- **V4T版本：开始用于商业授权，增加16位Thumb指令集**
  - ✧ **ARM7TDMI，1994**
    - 采用冯·诺依曼结构，三级流水线，可提供0.9MIPS/MHz的性能
    - 该内核不带有MMU和MPU
    - 用于嵌入式控制或简单的应用系统
  - ✧ **ARM922T，2000**
    - 采用哈佛结构，五级流水线，可提供1.1MIPS/MHz的性能
    - 该内核具有全性能的MMU，带有独立的指令和数据Cache
    - 可以应用于丰富的多媒体应用系统

2.2.1 ARM体系结构演变

- V5版本（V5TEJ）：代表CPU是ARM946E和ARM926EJ
  - ✧ 增加DSP常见运算指令，后者还增加对Java的支持
- V6版本
  - ✧ 采用一些更新的增强型技术
    - Thumb-2指令：功耗更低、性能更高和编码更紧凑
      比Thumb技术性能提高25%，比ARM技术存储空间减少26%
    - IEM (Intelligent Energy Manager)技术
      平均减少25%以上的处理器功耗；
    - TrustZone技术：新的软硬件结合的安全解决方案
  - ✧ 代表CPU：三款ARM11处理器
    - ARM1136J-S，2002
    - ARM1156JZ-S，2003
    - ARM1176JZ-S ，2003

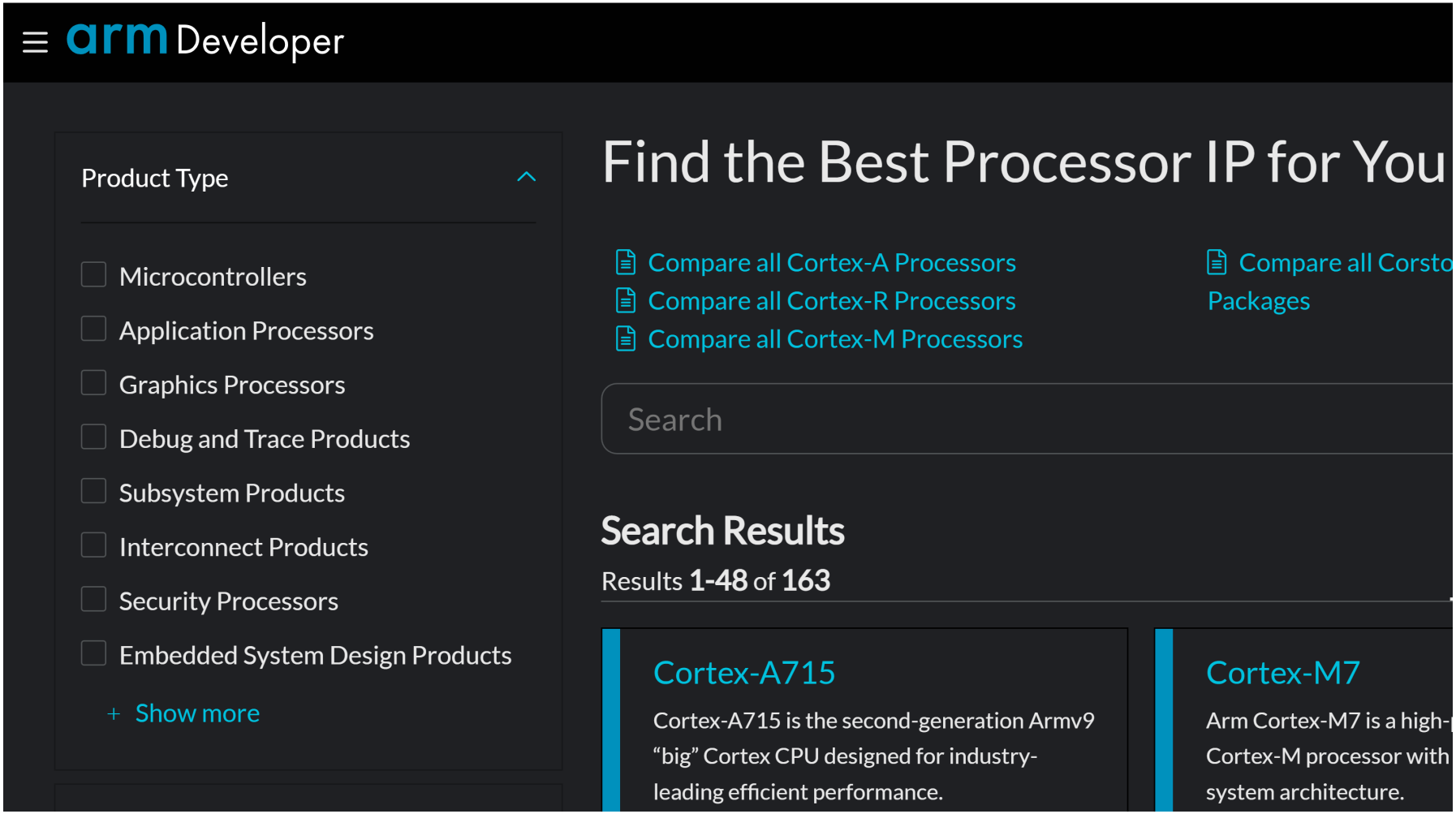
2.2.1 ARM体系结构演变

表．V1-V6版本ARM体系结构与处理器的对应关系

架构	处理器家族（经典系列）
ARMv1	ARM1
ARMv2	ARM2、ARM3
ARMv3	ARM6、ARM7
ARMv4	StrongARM、ARM7TDMI、ARM8、ARM9TDMI
ARMv5	ARM7EJ、ARM9E、ARM10E、XScale
ARMv6	ARM11、Cortex-M0/0+、Cortex-M1（后出）

2.2.1 ARM体系结构演变

- Cortex版本Cortex-A/R/M



2.2.1 ARM体系结构演变

- Cortex版本Cortex-A/R/M/SC
  - ✧ Cortex-A：Application
    - 面向计算要求高、运行丰富操作系统及提供交互媒体和图形体验的应用领域
    - 如：智能手机、平板电脑、汽车娱乐系统、数字电视等
  - ✧ Cortex-R：Real Time
    - 面向需要运行实时操作的系统应用
    - 如：汽车制动系统、大容量存储控制器等深层嵌入式实时应用
  - ✧ Cortex-M：Microcontroller
    - 面向微控制器领域，主要针对成本和功耗敏感的应用
    - 如：智能测量、人机接口设备、汽车和工业控制系统、家用电器、消费性产品和医疗器械等

2.2.1 ARM体系结构演变

表. ARM Cortex系列处理器对应关系

系列	架构	处理器家族
Cortex-A	ARMv7-A	A5, A7, A8, A9, A15, A17
	ARMv8-A	A32, A34, A35, A53, A57, A72, A
	ARMv8.2-A	A55, A65, A65AE, A75, A76, A76
	ARMv9-A	A510, A710, A715, X2, X3
Cortex-R	ARMv7-R	R4, R5, R7, R8
	ARMv8-R	R52, R52+, R82
Cortex-M	ARMv6-M	M0, M0+, M1
	ARMv7-M	M3, M4, M7
	ARMv8-M	M23, M33, M35P, M55, M85

32-bit	
Year	Core
2004	Cortex-M3
2007	Cortex-M1
2009	Cortex-M0
2010	Cortex-M4
2012	Cortex-M0+
2014	Cortex-M7
2016	Cortex-M23
2016	Cortex-M33
2018	Cortex-M35P
2020	Cortex-M55
2022	Cortex-M85

注：V7为32位（兼容V6以下的版本）；V8中32位和64位并存，32位可以向下兼容；V9则为64位，向下兼容V8的64位。

2.2.2* ARM体系结构的特征

- **ARM内核不是一个纯粹的RISC体系结构，这是为了使它能够更好地适应嵌入式系统。追求有效的系统性能，代码密度高和低功耗**
- **区别于RISC：**
  - 一些特定指令的周期数可变——并不是所有的ARM指令都是单周期的
  - 内嵌桶形移位器产生更为复杂的指令
  - 引入Thumb 16位指令集，代码密度可以提高约30%
  - 条件执行——只有当某个特定条件满足时指令才会被执行
    - 可以减少分支指令的数目，从而改善性能，提高代码密度
  - 增强指令（DSP）

2.2.3* ARM体系的变种

- **T变种**：Thumb指令集
- M变种：长乘法指令
- E变种：增强型DSP指令
- J变种：Java加速器Jazelle
- SIMD变种：媒体功能扩展

2.2.4* ARM处理器系列

- ARM7系列
- ARM9系列
- ARM9E系列
- ARM10E系列
- ARM11系列
- SecurCore系列
- Intel公司的StrongARM
- Intel公司的XScale
- Cortex系列

2.2.5 ARM存储数据类型

- ARM体系结构中的三种数据类型：
  - 字（Word）：长度为32位
  - 半字（Half-Word）：长度为16位
  - 字节（Byte）：长度为8位
- 几点说明：
  - 无符号数据类型，N位二进制表示$0 \sim 2^N - 1$的非负整数
  - 有符号数据类型，N位二进制表示$-2^{N-1} \sim 2^{N-1} - 1$的整数
  - 大多数数据操作，如ADD指令，都是以字为单位的
  - Load/store指令可以对上述三种类型数据进行操作
    - Load指令对半字或字节操作时，系统自动进行零扩展或符号位扩展
  - ARM指令是字对齐，占一个字长
    Thumb指令是半字对齐，占半个字长

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- **ARM处理器工作状态**
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

- ✧ **两种工作状态**
- ✧ **工作状态的切换**

2.3.1 两种工作状态

- ARM处理器有两种工作状态：
  - ARM状态：处理器执行32位的字对齐的ARM指令
  - Thumb状态：处理器执行16位的半字对齐的Thumb指令

- 工作状态的转变不影响处理器工作模式和相应寄存器中的内容

2.3.2 工作状态的切换

- ARMv4中使用指令BX
  - 指令格式

```
BX   {<cond>} <Rm>
```

    - <cond>：指令的条件码，忽略时无条件执行
    - <Rm>：寄存器中为跳转的目标地址
      <Rm>中bit[0]=0时为ARM指令， bit[0]=1时为Thumb指令
- ARMv5后增加指令BLX、LDR、LDM
- 注意：
  - ARM微处理器在复位、上电、发生异常时处于ARM状态
  - 如果处理器在Thumb状态进入异常，则当异常处理返回时，仍然处在Thumb状态

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- **ARM处理器工作模式**
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

2.4 ARM处理器工作模式

处理器工作模式	描述
用户 (usr)	**正常程序执行模式** 大部分应用程序运行于该模式，不能访问某些被保护的系统资源 不能直接切换到其他模式，除非异常发生
系统 (sys)	**运行操作系统的特权任务** 通常需要访问所有系统资源，使用usr模式的寄存器组 具有直接切换到其他模式等特权
快中断 (fiq)	**支持高速数据传输及通道处理** FIQ异常响应时进入此模式，高优先级
中断 (irq)	**用于通用中断处理** IRQ异常响应时进入此模式，低优先级
管理 (svc)	**操作系统保护模式** 系统复位和软件中断响应时进入此模式
中止 (abt)	**用于支持虚拟内存和/或存储器保护** 在ARM7TDMI没有大用处
未定义 (und)	**支持硬件协处理器的软件仿真** 未定义指令异常响应时进入此模式

改变工作模式?

- ✧ 软件
- ✧ 外部中断
- ✧ 异常处理

2.4 ARM处理器工作模式

处理器工作模式	描述
用户 (usr)	**正常程序执行模式** 大部分应用程序运行于该模式，不能访问某些被保护的系统资源 不能直接切换到其他模式，除非异常发生
系统 (sys)	**特权模式：** ✧ 该模式下可以访问所有系统资源 ✧ 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 ✧ 特权模式可以自由切换处理器模式，而用户模式不能直接切换到其他模式
快中断 (fiq)	
中断 (irq)	
管理 (svc)	
中止 (abt)	
未定义 (und)	

2.4 ARM处理器工作模式

处理器工作模式	描述
用户 (usr)	**正常程序执行模式** 大部分应用程序运行于该模式，不能访问某些被保护的系统资源 不能直接切换到其他模式，除非异常发生
系统 (sys)	**运行操作系统的特权任务** 通常需要访问所有系统资源，使用usr模式的寄存器组 具有直接切换到其他模式等特权
快中断 (fiq)	**异常模式：** ✧ 通过程序切换进入或者特定的异常进入 ✧ 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
中断 (irq)	
管理 (svc)	
中止 (abt)	
未定义 (und)	

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- **ARM处理器寄存器组织**
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

2.5 ARM处理器寄存器组织

- ARM处理器共有37个寄存器
  - ✧ 31个通用寄存器
    - 包括程序计数器PC
    - 这些寄存器是32位
  - ✧ 6个状态寄存器
    - 这些寄存器都是32位的，但只使用了其中的12位

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？
取决于微处理器的工作状态及具体的运行模式

寄存器安排成部分重叠的组，每种处理器模式使用不同的寄存器组

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10	R10						R10_fiq
	R11	R11						R11_fiq
	R12	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(PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10	R10						R10_fiq
	R11	R11						R11_fiq
	R12	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 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

31个通用寄存器

6个状态寄存器

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10	R10						R10_fiq
	R11	R11						R11_fiq
	R12	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 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

未分组寄存器
不同处理器模式都使用相同的32位物理寄存器

分组寄存器
同一个寄存器名对应多个物理寄存器，与当前处理器模式有关。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。

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10	R10						R10_fiq
	R11	R11						R11_fiq
	R12	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 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	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。

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0							
	R1							
	R2							
	R3							
	R4							
	R5							
	R6							
	R7							
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10	R10						R10_fiq
	R11	R11						R11_fiq
	R12	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 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

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

好处：发生FIQ中断后，可以加速FIQ的处理速度

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8							
	R9							
	R10							
	R11							
	R12							
	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 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

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

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7							
	R8							
	R9							
	R10							
	R11							
	R12							
	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(PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

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，只是习惯上都这样使用
- ✧ 在Thumb指令集中，存在使用R13的指令

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6							
	R7							
	R8							
	R9							
	R10							
	R11							
	R12							
	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(PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R14 又称作（子程序）链接寄存器（LR），两种作用：

- ✧ 每种模式各自的R14用于存放当前子程序返回地址
- ✧ 当异常中断发生时，对应模式的R14被设置为异常模式将要返回的地址（有些异常有一个小的固定偏移量）

另外，R14也可作为通用寄存器使用

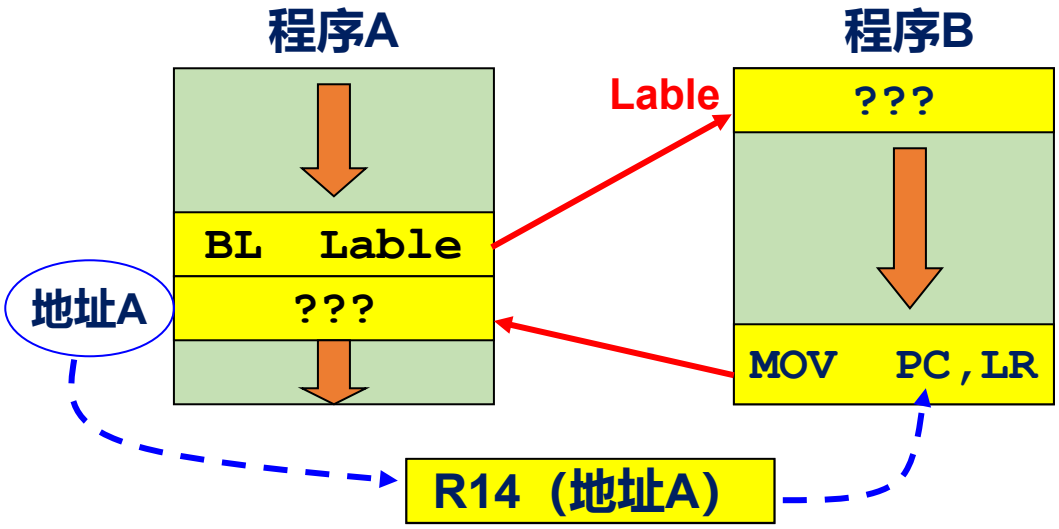
R14（LR）寄存器与子程序调用

操作流程

① 程序A执行过程中调用程序B

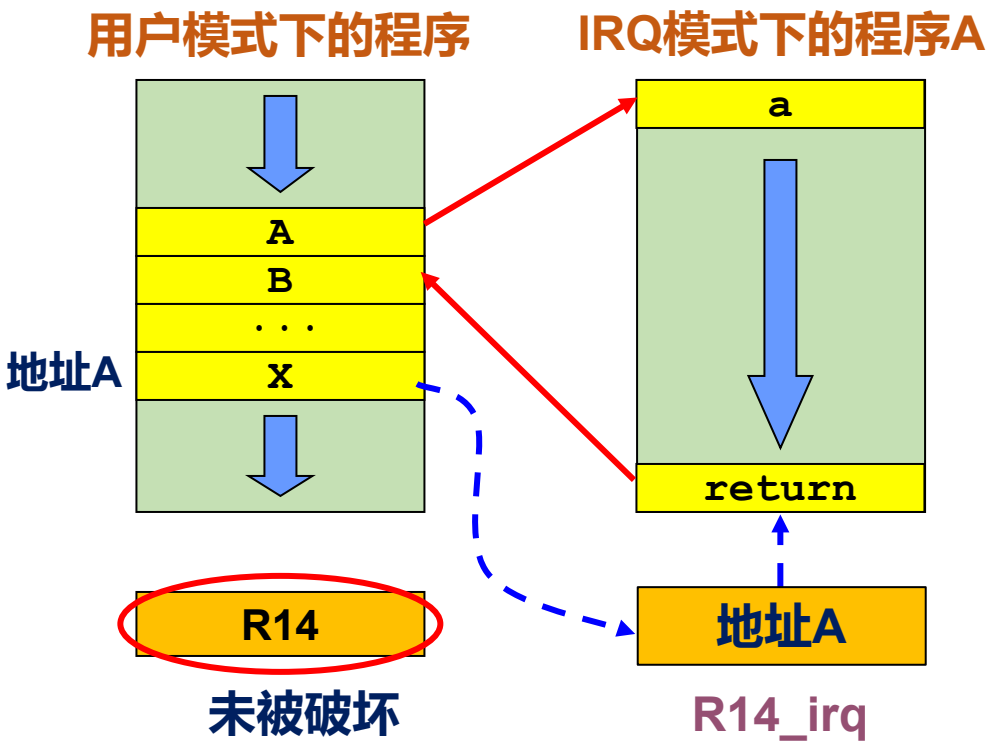
② 程序跳转至标号Lable，执行程序B，**同时**硬件将“BL Lable”指令的**下一条指令所在地址**存入R14（LR）

③ 程序B执行最后，将R14的值放入程序计数器PC，返回程序A

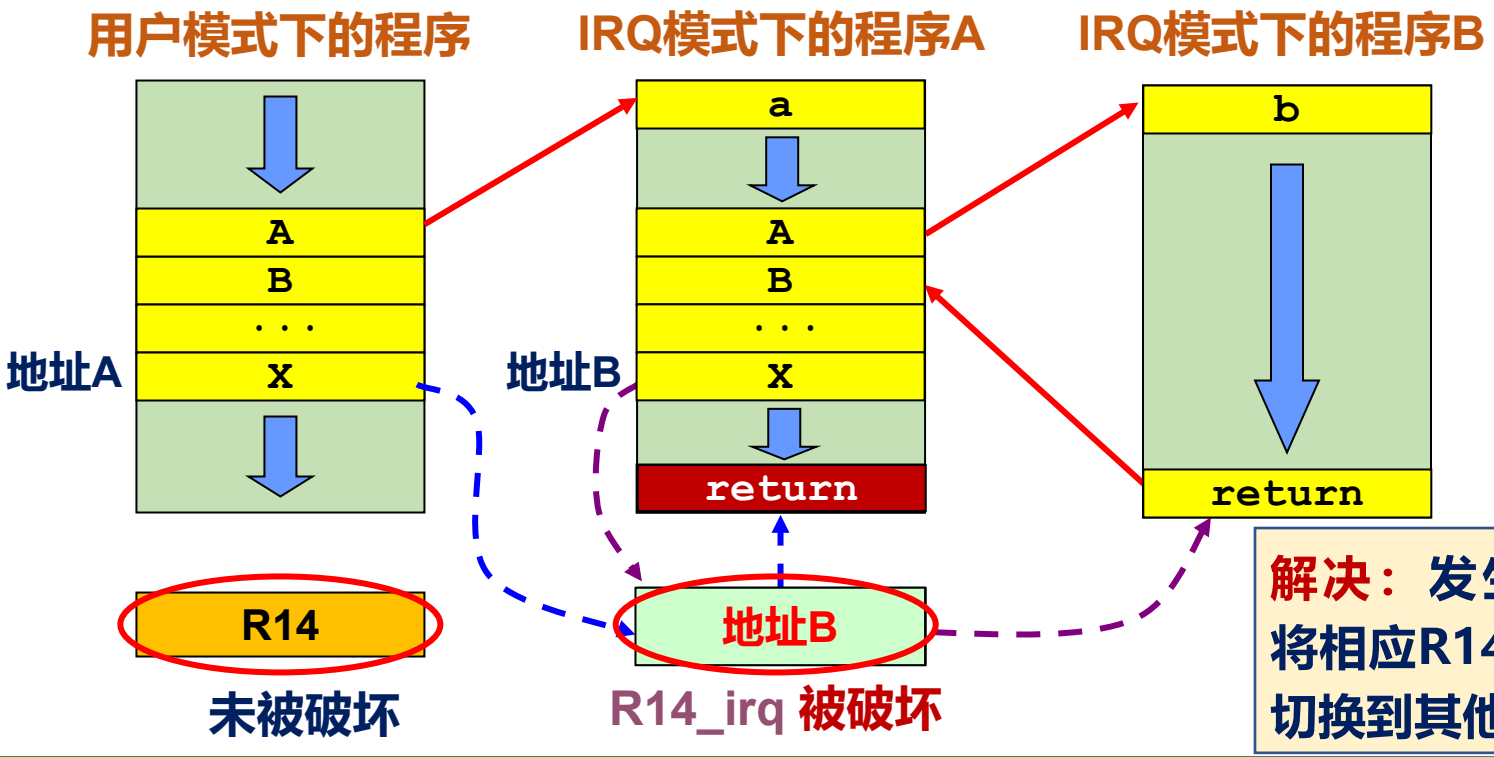


异常发生时，程序要跳转至异常服务程序，对返回地址的处理**与子程序调用类似**，都是由硬件完成的。**区别**在于有些异常有一个小常量的偏移。

- **当发生异常嵌套时，这些异常之间可能会发生冲突**
  - ① 执行用户模式下的程序
  - ② 发生IRQ中断，硬件将某个地址存入IRQ模式下的R14_irq寄存器，用户模式下的R14没有被破坏
  - ③ IRQ服务程序A执行完毕，将R14_irq寄存器的内容减去某个常量后存入PC，返回之前被中断的程序



- **当发生异常嵌套时，这些异常之间可能会发生冲突**
  - ④ 如果在IRQ处理程序中打开IRQ中断，并且再次发生IRQ中断
  - ⑤ 硬件将返回地址保存在R14_irq寄存器中，**原来保存的返回地址将被覆盖，造成错误**
  - ⑥ 在程序B返回到程序A，然后再返回到用户模式下被中断的程序时，发生错误，**不能正确返回**



解决：发生中断嵌套时将相应R14_irq入栈，或切换到其他处理器模式

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	R8	R8						R8_fiq
	R9	R9						R9_fiq
	R10							
	R11							
	R12							
	R13 (SP)							
	R14 (LR)							
	R15 (PC)	R15 (PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

R15 为程序计数器（PC）

- 指向正在取指的地址
- 可以认为是一个通用寄存器，但使用时有许多与指令相关的限制或特殊情况

2.5 ARM处理器寄存器组织

R15 为程序计数器（PC）

- ✧ **指向正在取指的地址**
- ✧ **可以认为是一个通用寄存器，但使用时有许多与指令相关的限制或特殊情况**

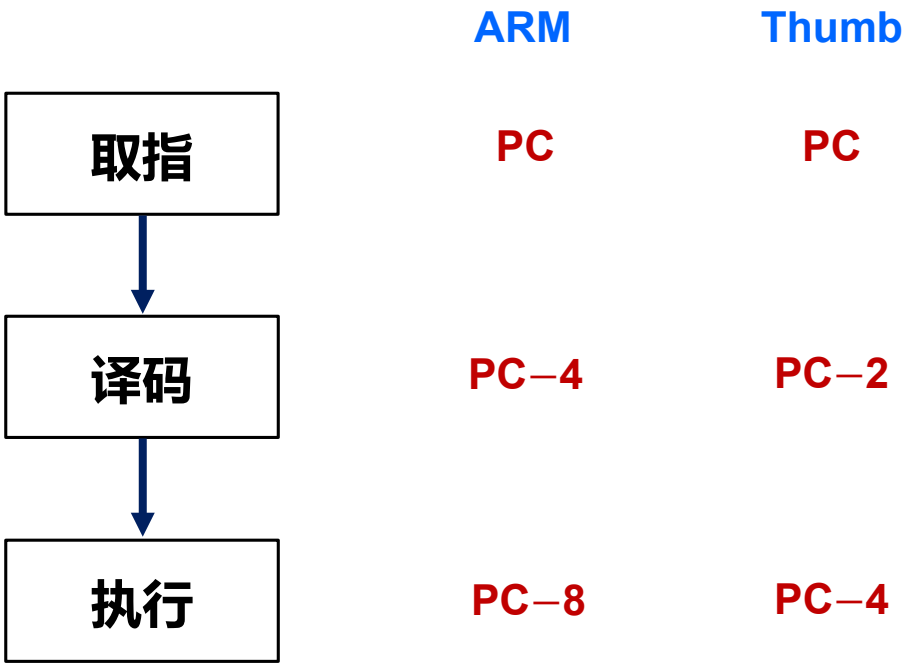


图2.4 指令流水线功能段

2.5 ARM处理器寄存器组织

■ 读R15的限制

✧ 正常操作时，从R15读取的值是处理器正在取指的地址

- 即当前正在执行指令的地址加上8个字节（两条ARM指令的长度）

地址	程序代码	流水线状态
PC-8	LDR R0,PC	正在执行
PC-4	???	正在译码
PC	???	正在取指

✧ ARM指令总是以字=4字节为单位，所以R15最低两位总是为0

- 字对齐：被访问的“字”必须存放在存储器紧邻的4个字节单元，并且首字节地址必须能被4整除——详见2.7.1节

2.5 ARM处理器寄存器组织

■ 写R15的限制

✧ 正常操作时，写入R15 的值被当作一个指令地址

- 程序从这个地址处继续执行（相当于**执行一次无条件跳转**）
- 在ARM结构V4及以上版本中，写入R15的值的**最低两位为0**
  如果不是，结果将**不可预测**

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（物理寄存器 = 37个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0							
	R1							
	R2							
	R3							
	R4							
	R5							
	R6							
	R7							
	R8							
	R9							
	R10							
	R11							
	R12							
	R13 (SP)							
	R14 (LR)							
	R15 (PC)							
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

CPSR 为程序状态寄存器，可用于所有模式

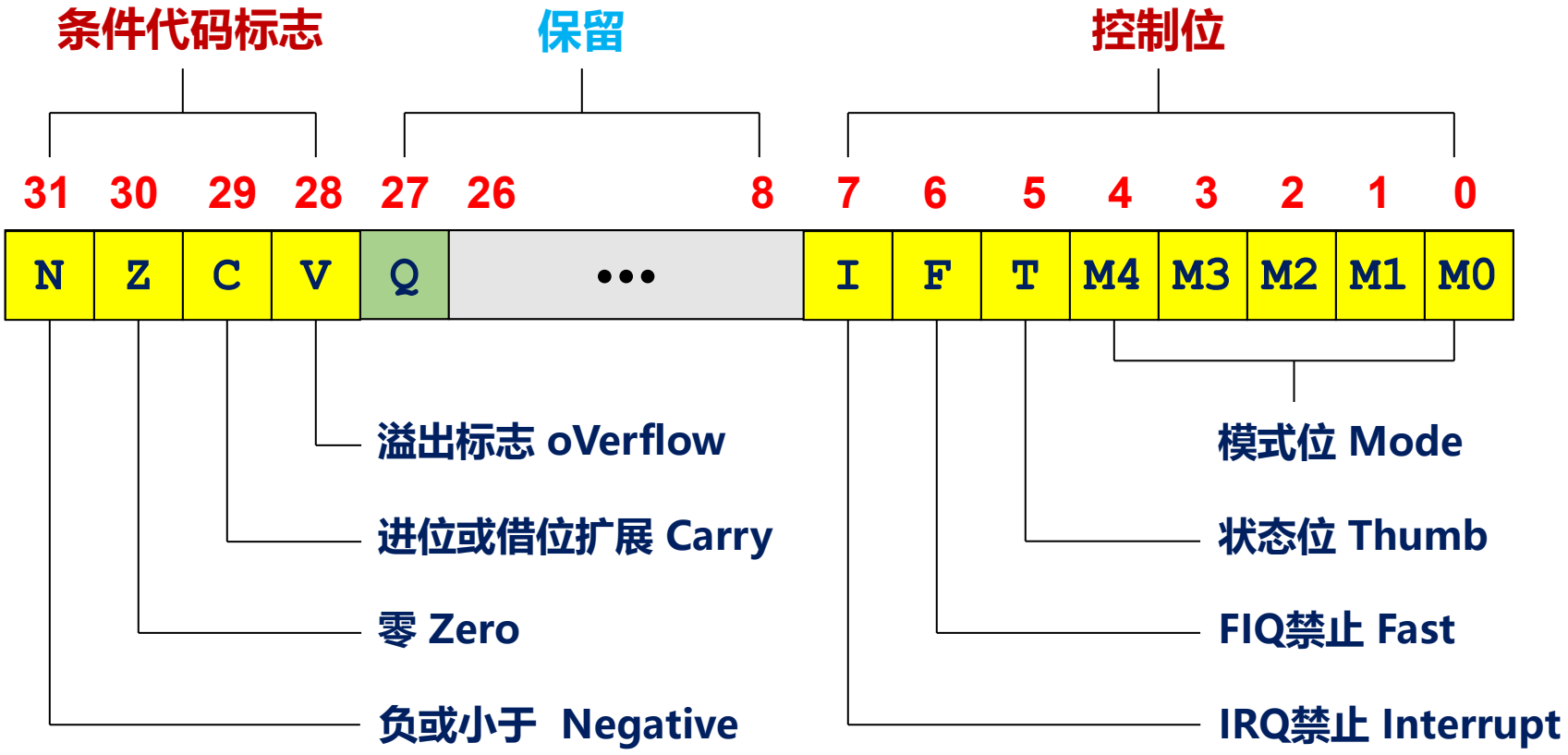
- 在异常模式中，“程序状态保存寄存器（SPSR）”可以被访问
- 每种异常都有自己的SPSR，在进入异常时它保存CPSR的当前值，异常退出时可通过它恢复CPSR

CPSR 反映了当前处理器的状态，包含：

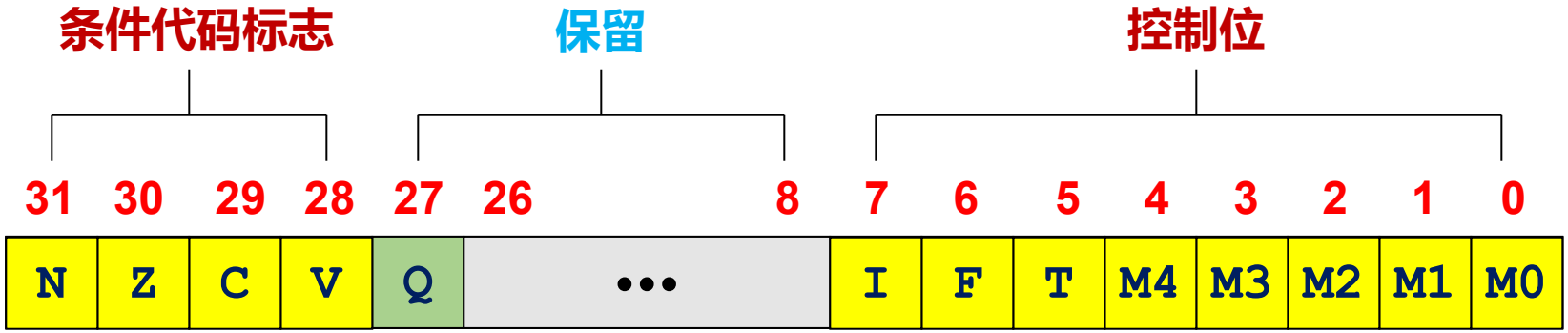
- 4个条件代码标志：负(N)、零(Z)、进位(C)、溢出(V)
- 2个中断禁止位，分别控制一种类型的中断
- 5个对当前处理器模式进行编码的位
- 1个用于指示当前执行指令（ARM or Thumb）的位

2.5 ARM处理器寄存器组织

CPSR寄存器的格式



2.5 ARM处理器寄存器组织

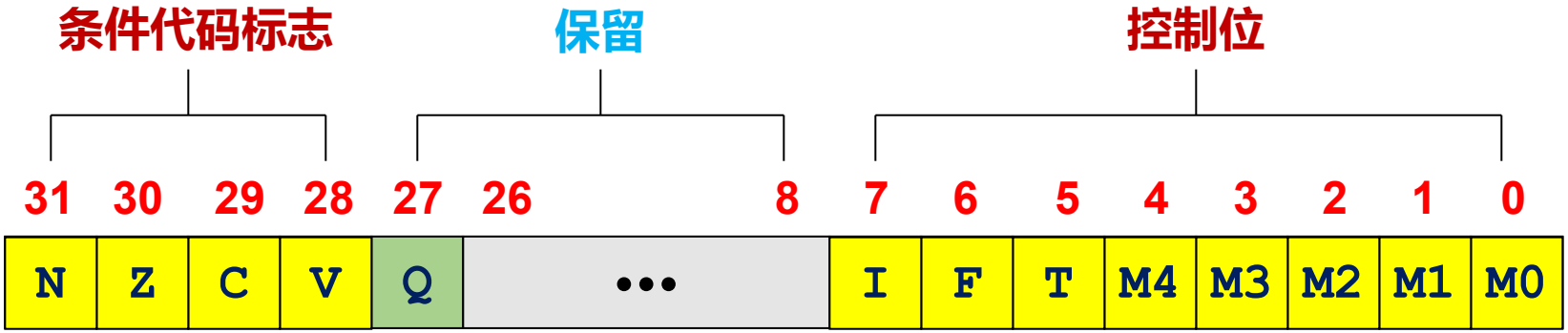


- 条件代码标志
  - 大多数“数值处理指令”可以选择是否影响条件代码标志位
    - 通常如果指令带S后缀，则该指令的执行会影响条件代码标志
    - 有一些指令的执行总是会影响条件代码标志
  - 算术操作、逻辑操作、MSR或者LDM指令可以对条件代码标志位位进行设置
    - 所有ARM指令都可按条件来执行
    - Thumb指令中只有分支指令可按条件执行

2.5 ARM处理器寄存器组织

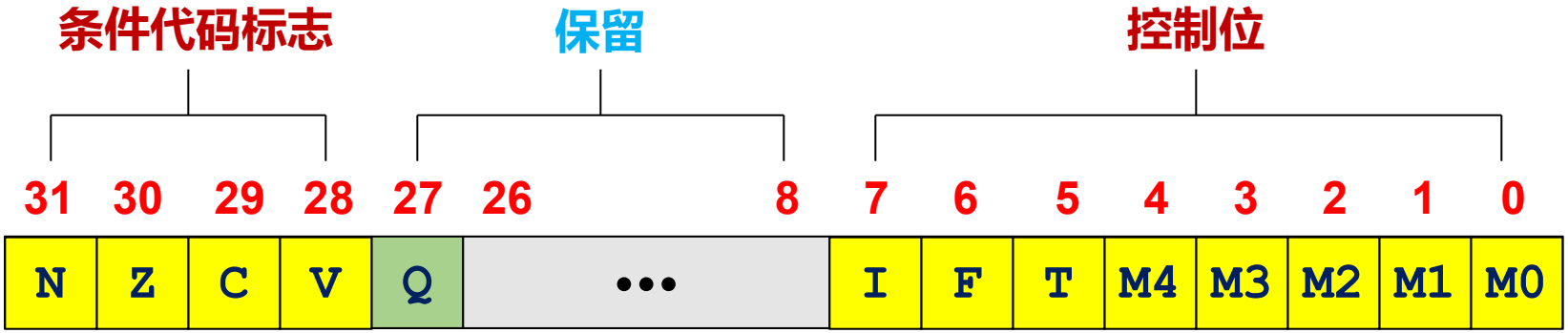
条件码标志位	含义
N	反映运算结果的最高位，若为补码表示的有符号数运算： ✧ N = 1，结果为负数 ✧ N = 0，结果为正数
Z	反映计算结果是否非零，或CMP指令中比较两个数是否相等： ✧ Z = 1，运算结果为0 ✧ Z = 0，运算结果不为0
C	包含3种方法： ✧ 在加法指令中(包括CMN指令)，最高位进位时C = 1 ✧ 在减法指令中(包括CMP指令)，最高位借位时C = 0 ✧ 在包含移位操作的非加法/减法指令中，C为从最高位最后移出的值，其他指令下C通常不改变
V	加法/减法运算指令中，若结果为补码表示的有符号数运算： ✧ V = 1，符号位溢出

2.5 ARM处理器寄存器组织



- 控制位 7~5
  - ✧ 中断禁止位包括第7位I，第6位F：
    - I = 1，IRQ中断被禁止
    - F = 1，FIQ中断被禁止
  - ✧ 标志位T反映正在操作的状态：
    - T = 1，处理器在Thumb状态下运行
    - T = 0，处理器在ARM状态下运行

2.5 ARM处理器寄存器组织



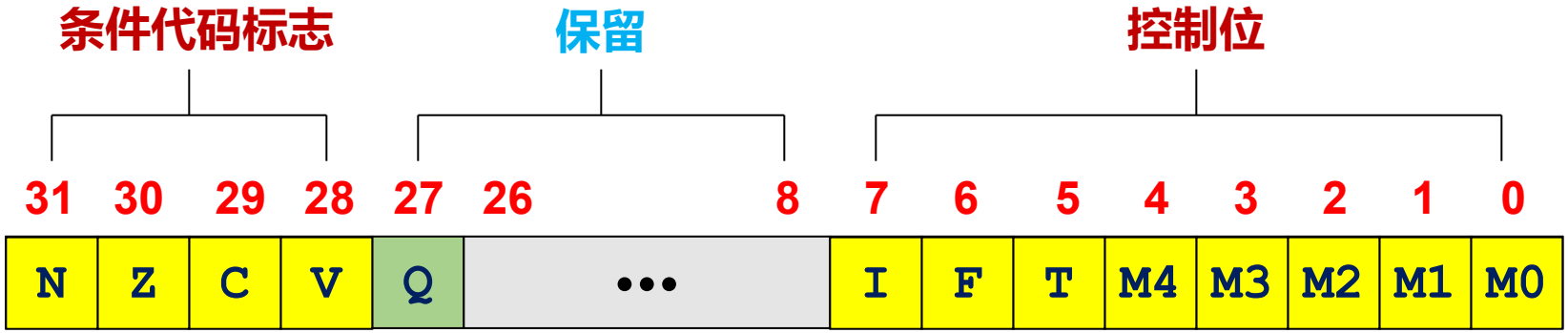
- **控制位 M4~M0**
  - ✧ 决定处理器的操作模式
  - ✧ 注意：
    - 不是所有模式位的组合都定义了有效的处理器模式（这里是7种）
      如果使用了错误的设置，将引起一个无法恢复的错误

2.5 ARM处理器寄存器组织

表2.4 CPSR模式位设置表

M[4:0]	模式	可访问的寄存器(ARM)
10000	用户	PC, CPSR, R0~R14
10001	快中断	PC, CPSR, SPSR_fiq, R14_fiq~R8_fiq, R0~R7
10010	中断	PC, CPSR, SPSR_irq, R14_irq~R13_irq, R0~R12
10011	管理	PC, CPSR, SPSR_svc, R14_svc~R13_svc, R0~R12
10111	中止	PC, CPSR, SPSR_abt, R14_abt~R13_abt, R0~R12
11011	未定义	PC, CPSR, SPSR_und, R14_und~R13_und, R0~R12
11111	系统	PC, CPSR, R0~R14

2.5 ARM处理器寄存器组织



- **保留位 27~8**
  - ✧ CPSR中的保留位被保留将来使用
    - 为了提高程序的可移植性
      当改变CPSR标志和控制位时，**不要改变这些保留位**
    - 要确保程序的运行不受保留位的值影响
      将来的处理器可能会将这些位设置为1或者0

2.5 ARM处理器寄存器组织

- Thumb状态下的寄存器是ARM状态下寄存器的子集
  - Thumb状态R0～R7与ARM状态R0～R7相同
  - Thumb状态CPSR/SPSR与ARM状态CPSR/SPSR相同
  - Thumb状态SP映射到ARM状态R13
  - Thumb状态LR映射到ARM状态R14
  - Thumb状态PC映射到ARM状态PC（R15）

2.5 ARM处理器寄存器组织

表2.5 Thumb状态下寄存器组织

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（访问物理寄存器 = 27个）						
		用户 usr	系统 sys	管理 svc	中止 abt	未定义 und	中断 irq	快中断 fiq
通用寄存器和程序计数器	R0	R0						
	R1	R1						
	R2	R2						
	R3	R3						
	R4	R4						
	R5	R5						
	R6	R6						
	R7	R7						
	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(PC)						
状态寄存器	CPSR	CPSR						
	SPSR	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- **ARM异常**
- ARM存储器
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

- ✧ ARM异常概述
- ✧ ARM异常处理
- ✧ ARM异常向量表
- ✧ ARM异常优先级
- ✧ 异常中断使用的寄存器

2.6 ARM异常

- ARM体系中有三种方式控制程序的执行流程

① 程序正常执行

- 按顺序执行，每执行一条ARM指令，PC的值加4个字节（1个字）

② 执行跳转指令

B	执行跳转操作
BL	执行跳转操作，保存子程序返回地址
BX	执行跳转操作，根据目标地址最低位，可切换到Thumb状态
BLX	执行跳转操作，BL与BX功能之并

- 程序跳转到特定的地址标号处执行，或者跳转到特定子程序执行

③ 异常中断

- 执行完当前指令后，跳转到相应的异常中断处理程序处执行
- 异常中断处理程序执行完成后，返回发生中断的指令的下一条处执行
- 进入异常中断时，要保存现场；退出异常中断时，要恢复现场

2.6.1 ARM异常概述

- 异常可通过内部或者外部源产生，并引起处理器处理一个事件
  - ✧ 外部中断或者试图执行未定义指令都会引起异常
  - ✧ 只要正常的程序流被暂时中止，处理器就进入异常模式
    - 在处理异常之前，ARM内核保存当前的处理器状态（CPSR→SPSR）
    - 在处理程序结束时，可以恢复执行原来的程序（SPSR→CPSR）
  - ✧ 同一时刻出现多个异常时，根据优先级顺序选择相应的异常中断
    ——详见2.6.4节

表2.6 ARM体系结构支持的异常类型

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，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，或地址不允许当前指令访问，产生数据中止异常
外部中断请求IRQ	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
快速中断请求FIQ	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

3.6.2 ARM异常处理

- **对异常响应**
  - ✧ 当任何一个异常发生并得到响应时，ARM内核自动完成动作：
    - ① 将下一条指令的地址存入相应链接寄存器LR，以便程序在异常处理完成后能从正确的位置重新开始执行
    - ② 将CPSR的值复制到相应的SPSR中
    - ③ 设置适当的CPSR位，包括：改变处理器状态进入ARM状态，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
    - ④ 设置PC使其从相应的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处

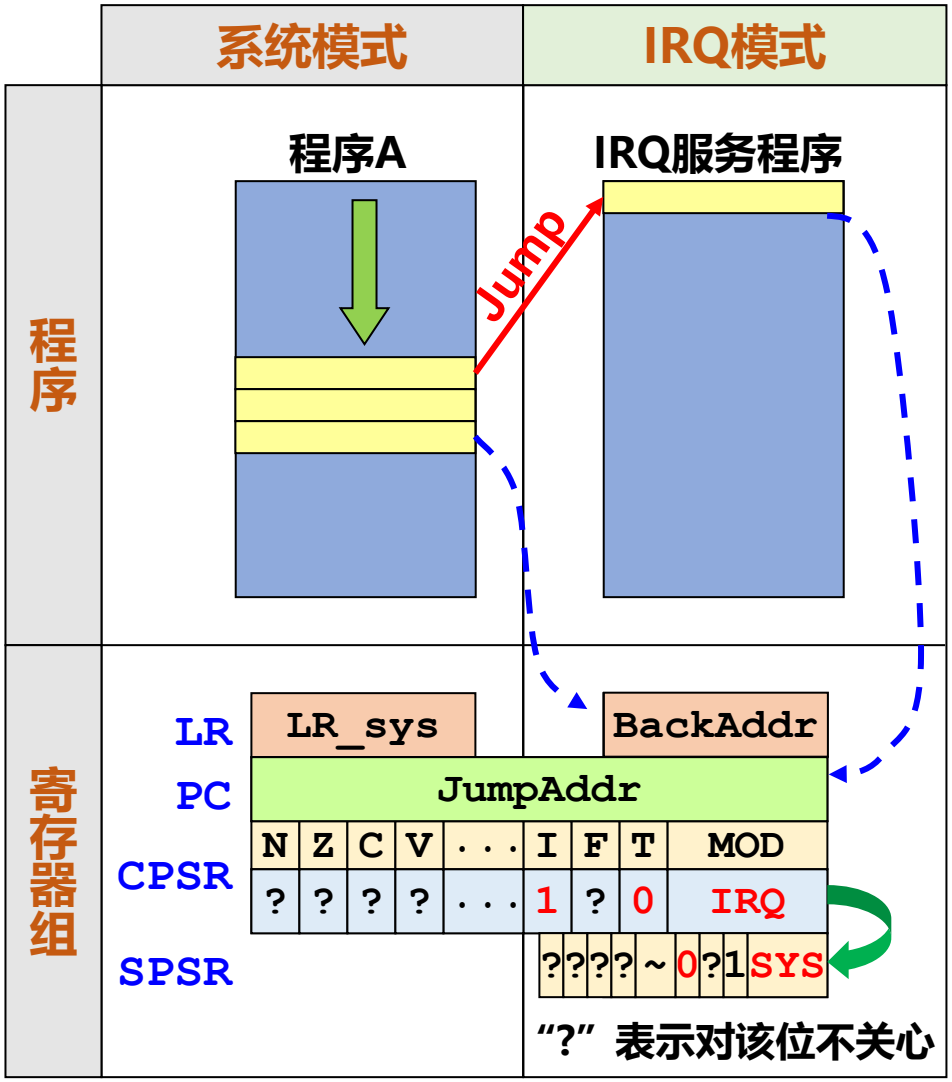


图. 进入异常的过程

- 程序在系统模式下运行用户程序，假定当前处理器状态为Thumb状态、允许IRQ中断
- 用户程序运行时发生IRQ中断，硬件完成以下动作：
  - 将CPSR寄存器内容存入IRQ模式的SPSR寄存器
  - 置位I位（禁止IRQ中断）
  - 清零T位（进入ARM状态）
  - 设置MOD位，切换处理器模式至IRQ模式
  - 将下一条指令的地址存入IRQ模式的LR寄存器
  - 将跳转地址存入PC，实现跳转

3.6.2 ARM异常处理

- 从异常返回
  - 异常处理完毕后，通过以下操作完成从异常中断处理程序中返回：
    ① 将链接寄存器LR的值减去相应的偏移量后送到PC中
    ② 将SPSR的值复制回CPSR中
    ③ 清除中断禁止位

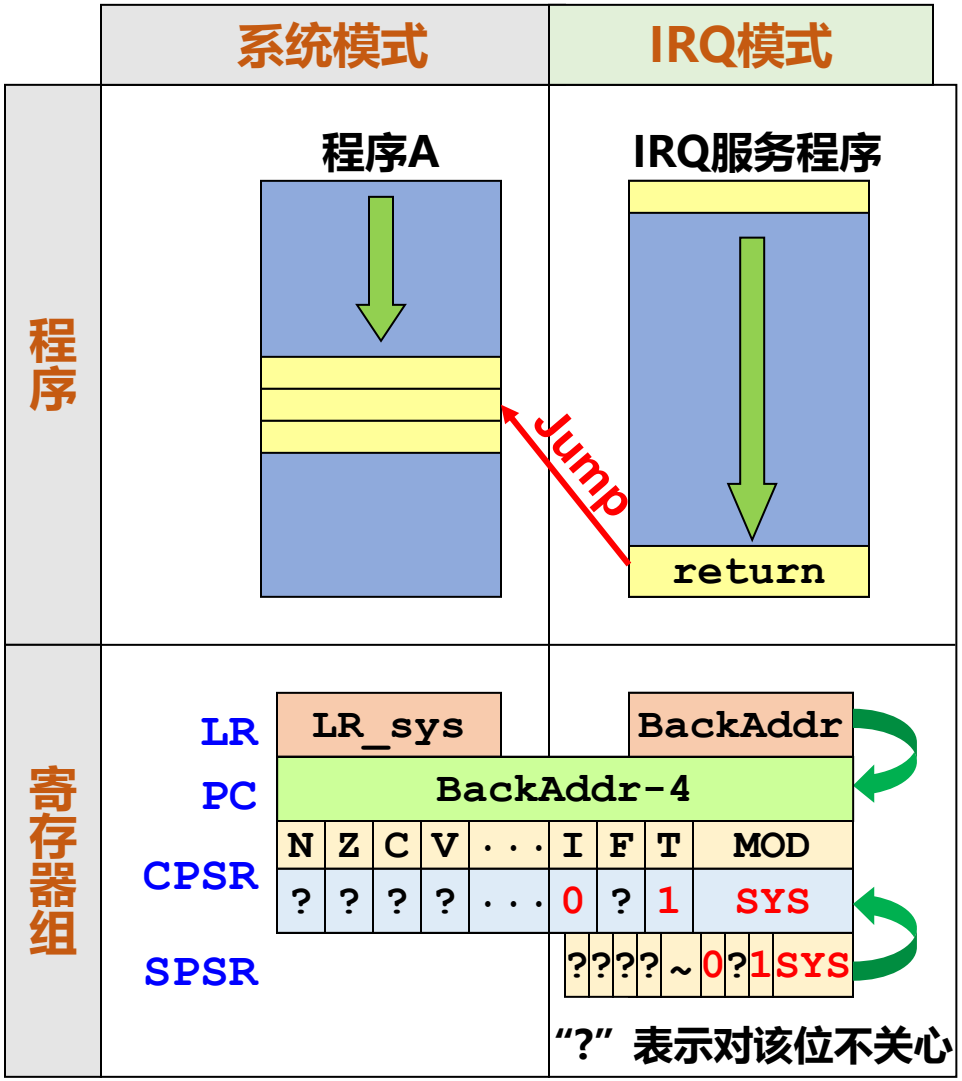


图. 退出异常的过程

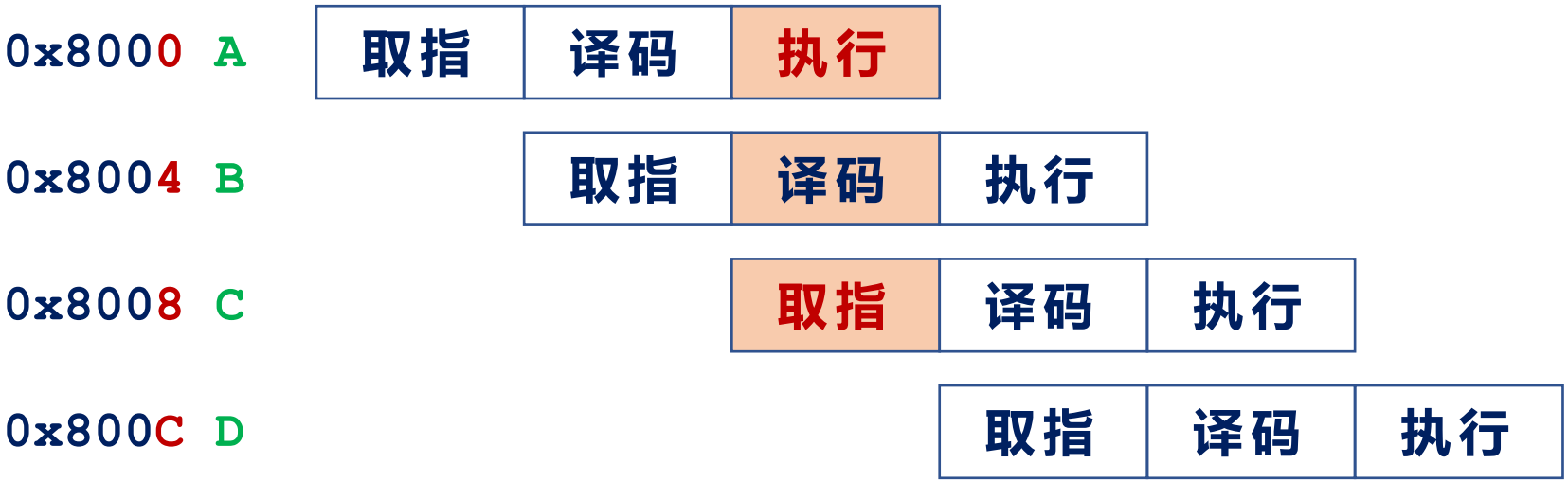
- 在异常处理结束后，异常处理程序完成以下动作：
  - 将SPSR寄存器的值复制回CPSR寄存器
  - 将LR寄存的值减去一个常量(IRQ/FRQ为4)后复制到PC寄存器，跳转到被中断的用户程序
  - 清除中断禁止位（确认I=0）

3.6.2 ARM异常处理

- 在ARM体系结构里，PC值指向当前执行指令的地址加8处
  - 当执行指令A（地址0x8000）时，PC等于指令C的地址（0x8008）
  - 假如指令A是“BL”指令，执行时会把PC（=0x8008）保存到LR寄存器里面，但是接着处理器会马上对LR进行一个自动调整动作：

    `LR = LR - 0x4`

    - 作用：最终保证LR里面的是指令B的地址，当从BL返回时，LR里面刚好是正确的返回地址（因为从异常返回后要继续执行A的下一条指令B）



3.6.2 ARM异常处理

表2.7 异常返回指令表

异 常	返回指令
软件中断	`MOVS PC, LR ; R14_svc`
未定义指令	`MOVS PC, LR ; R14_und`
快速中断 FIQ	`SUBS PC, LR, #4 ; R14_fiq`
外部中断 IRQ	`SUBS PC, LR, #4 ; R14_irq`
中止（预取指令）	`SUBS PC, LR, #4 ; R14_abt`
中止（数据）	`SUBS PC, LR, #8 ; R14_abt`
复位	NA（不需要返回）

3.6.3 ARM异常向量表

- 异常向量表
  - ✧ 异常产生后，系统会强制跳到固定地址开始执行程序
    - 这些固定地址称为异常向量表
  - ✧ 异常向量表可以存放在存储地址的低端或者高端，通常存放低端
  - ✧ 异常中断向量表的大小为32字节
    - 每个异常中断占据4个字节的大小，总共有8个异常中断
  - ✧ 4个字节 = 1个字，说明每个异常中断对应的中断向量表中的4个字节空间中存放了一条ARM指令
    - 即，跳转指令或者向PC寄存器中赋值的数据访问指令
    - 通过这两种指令，程序将跳转到相应的异常中断处理程序处执行

3.6.3 ARM异常向量表

表2.8 异常向量表（8条 = 32字节）

异常	异常向量地址（通常）	高向量地址
复位	0x0000,0000	0xFFFF,0000
未定义指令	0x0000,0004	0xFFFF,0004
软件中断	0x0000,0008	0xFFFF,0008
预取指令中止	0x0000,000C	0xFFFF,000C
数据中止	0x0000,0010	0xFFFF,0010
保留	0x0000,0014	0xFFFF,0014
外部中断 IRQ	0x0000,0018	0xFFFF,0018
快速中断 FIQ	0x0000,001C	0xFFFF,001C

3.6.4 ARM异常优先级

- 多个异常同时发生时，系统需要按照一定顺序来处理这些异常中断
  - ✧ 优先级高的先执行
  - ✧ 软件中断由SWI指令产生，和未定义指令优先级相同

表2.9 异常优先级

优先级	异常
1（最高）	复位
2	数据中止
3	快速中断 FIQ
4	外部中断 IRQ
5	预取指令中止
6（最低）	未定义指令，SWI

3.6.5 ARM异常中断使用的寄存器

表2.10 异常对应的处理器工作模式

异 常	模式
复位	管理模式
未定义指令	未定义模式
软件中断	管理模式
预取指令中止	中止模式
数据中止	中止模式
保留	保留
外部中断 IRQ	外部中断模式
快速中断 FIQ	快速中断模式

- 异常中断发生时，系统会自动切换到相应的处理器工作模式，各个异常都对应固定的处理器模式
- 异常中断使用的寄存器与处理器工作模式使用的寄存器相对应

表. ARM异常对应关系表

异常向量地址	异常类型	处理器模式	优先级
0x0000,0000	复位	管理模式	1（最高）
0x0000,0004	未定义指令	未定义模式	6
0x0000,0008	软件中断	管理模式	6
0x0000,000C	预取指令中止	中止模式	5
0x0000,0010	数据中止	中止模式	2
0x0000,0014	保留	保留	保留
0x0000,0018	外部中断 IRQ	外部中断模式	4
0x0000,001C	快速中断 FIQ	快速中断模式	3

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- **ARM存储器**
- ARM总线技术
- ARM存储系统
- 基于JTAG的调试系统

- ARM体系存储空间
- ARM存储器格式
- 非对齐存储访问操作

2.7.1 ARM体系存储空间

- ARM体系使用2^{32}个字节的单一、线性地址空间
  - 这些字节单元的地址为无符号32位数值
  - 取值范围为$0 \sim 2^{32}-1$
- ARM的地址空间可看作由2^{30}个32位的字组成
  - 每个字的地址是字对齐的，所以字的地址可被4整除
  - 字对齐的地址是A的字包含：A，$A+1$，$A+2$，$A+3$ 四个字节
- ARM的地址空间也可看作由2^{31}个32位的半字组成
  - 每个半字的地址是半字对齐的，所以字的地址可被2整除
  - 半字对齐的地址是A的半字包含：A，$A+1$ 两个字节

2.7.1 ARM体系存储空间

- ARM体系结构中的地址表示（32位）：
  - ✧ 二进制0000~1111对应十六进制的0~F
    - 每4位一组，32位二进制可分为8组，对应8个十六进制数
  - ✧ 地址表示为：0x00000000~0xFFFFFFFF
    - 字对齐时：0x00000000~0xFFFFFFFC
      二进制末2位为0，每个字地址对应4个字节（4个基本存储单元）
    - 半字对齐时：0x00000000~0xFFFFFFFE
      二进制末1位为0，每个半字地址对应2个字节（2个基本存储单元）

2.7.2 ARM存储格式

- ARM体系结构可以用两种格式来存储数据
  - 小端存储格式（Little Endian）
  - 大端存储格式（Big Endian）

2.7.2 ARM存储格式

- ARM体系结构可以用两种格式来存储数据
  - ✧ 小端存储格式（Little Endian）
    - 低字节内容保存在低地址空间，高字节内容保存在高地址空间
  - ✧ 大端存储格式（Big Endian）



图2.11 小端格式

2.7.2 ARM存储格式

- ARM体系结构可以用两种格式来存储数据
  - ✧ 小端存储格式（Little Endian）
  - ✧ 大端存储格式（Big Endian）
    - 高字节内容保存在低地址空间，低字节内容保存在高地址空间



图2.11 大端格式

2.7.2 ARM存储格式

- **例：将数据0xA9876543存入0x0000为起始地址的存储空间**
  - ✧ 该数据为32位（= 1个字 = 4个字节），需要存入4个存储单元

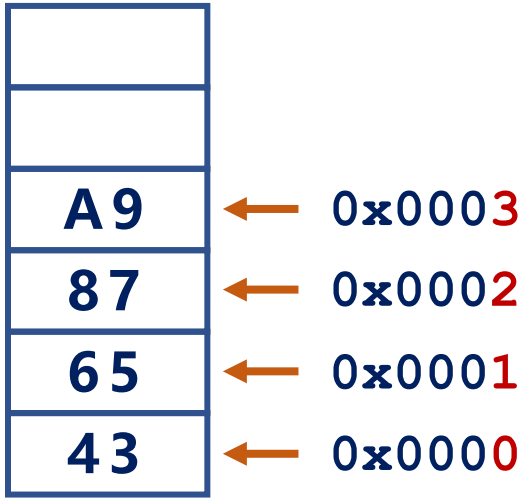


图. 小端格式

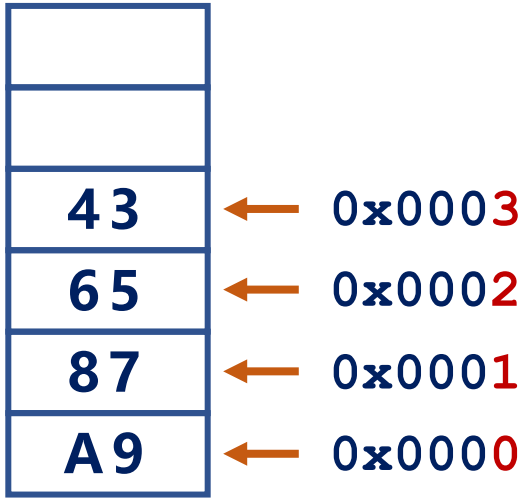


图. 大端格式

数据0xA9876543存储后，作为字的地址是0x0000，
作为半字的地址是0x0000和0x0002

2.7.3 非对齐存储访问操作

- 处理器在对半字和字进行操作时要注意地址对齐
  - 字操作时，字的地址末2位为0
  - 半字操作时，半字的地址末1位为0
  - 例如：对字(32位)进行操作时，要对齐字的起始地址：
    - 只能把地址指针对准地址0x0000，0x0004，0x0008等处
    - 不能把地址指向0x0001，0x0002，0x0003等中间部分
- 非对齐的取指或者数据访问操作，会引起读取错误，产生无法预测的结果

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- **ARM总线技术**
- ARM存储系统
- 基于JTAG的调试系统

2.8 ARM总线技术

- ARM公司于1996提出高级微控制器总线结构AMBA
  - ✧ 作用：使片上不同宏单元的连接实现标准化
- AMBA规范定义了三种总线：
  - ✧ AHB（Advanced High-performance Bus）
    - 用于连接高性能、高时钟工作频率的系统模块
    - 支持突发数据传输方式及单个数据传输方式
      所有时序参考同一个时钟沿
  - ✧ ASB（Advanced System Bus）
    - 用于连接高性能系统模块
    - 支持突发数据传输模式
  - ✧ APB（Advanced Peripheral Bus）
    - 一个简单接口，支持低性能的外围接口

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- **ARM存储系统**
- 基于JTAG的调试系统

- ✧ **高速缓冲存储器Cache和紧耦合存储器TCM**
- ✧ **存储管理**

2.9 ARM存储系统

- 根据不同的嵌入式应用系统需要，ARM存储系统的体系结构差别很大
  - ✧ 最简单的存储系统使用平板式的地址映射机制，地址空间的分配方式固定（类似一些简单的单片机系统）
  - ✧ 复杂的嵌入式系统可能包含多种类型的存储器件
    必须采用一些相应的存储技术
    - 如：Flash、ROM、SRAM、SDRAM 等

2.9.1 高速缓冲存储器和紧耦合存储器

- **高速缓冲存储器Cache**
  - ✧ 一种容量小，速度快的存储器阵列
  - ✧ 位于主存和处理器内核之间，保存最近一段时间处理器涉及的主存块内容
    - 处理器尽量从Cache中读取数据，而无需访问主存（慢速存储器）
    - 带有Cache的ARM内核可能采用了冯•诺依曼结构或哈佛结构
- **紧耦合存储器TCM** (Tightly Coupled Memory)
  - ✧ 一种快速SRAM
  - ✧ 紧挨内核，并且保证取指或数据操作的时钟周期数——实时响应
  - ✧ TCM位于存储器的地址映射中，可作为快速存储器使用

2.9.2 存储管理

- ARM内核的存储器管理硬件有三种不同类型：
  - 没有提供扩展，没有硬件保护（无保护）
  - 提供有限保护的**存储器保护单元**（MPU）
  - 提供全面保护的**存储器管理单元**（MMU）

① **无保护存储器**

- 只能提供非常有限的灵活性
- 通常用于小的、简单的嵌入式系统
  这种系统由于其应用特点而不要求存储器保护

② **存储器保护单元MPU**

- MPU使用**区域**（Region）来管理系统保护
- 保护的是：**存储器系统**和**外围设备**

2.9.2 存储管理

③ 存储管理单元MMU

- ✧ MMU提供的一个关键服务：
  - 使各个任务作为各自独立的程序在其自己的私有存储空间中运行
- ✧ MMU主要完成以下工作：
  - 虚拟存储空间到物理存储空间的映射
  - 存储器访问权限的控制
  - 设置虚拟存储空间的缓冲特性
- ✧ 管理的手段：页表（Translate Table）
  - 是一个位于内存中的表，系统通常有一个寄存器来保存页表的基地址
  - 表的每一行对应于虚拟存储空间的一个页，该行包含了该虚拟内存页对应的物理内存页的地址、该页的访问权限和该页的缓冲特性等
  - 从虚拟地址到物理地址的变换过程其实就是查询页表的过程

名称		描述
RAM 易失	SRAM	静态随机存储器，不需要刷新电路，速度非常快，集成度较低，价格昂贵，主要应用于缓存或者类似缓存的场合，如TCM
	DRAM	动态随机存储器，需要刷新电路，速度较快，集成度较高，应用于主存
	SDRAM	同步动态随机存储器，需要刷新电路，集成度非常高，速度比DRAM快，一般的嵌入式产品里面的主存都是用的SDRAM
ROM 非易失	PROM	可编程只读存储器，只能写一次
	EPROM	可擦除可编程只读存储器，在芯片的上面有个窗口，通过紫外线的照射来擦除数据
	EEPROM	电可擦除可编程只读存储器，比EPROM先进点，可以用电来擦除数据
Flash 电擦写	Nand 与非	采用串行接口，CPU从里面读取数据的速度很慢，用Nand做闪存时，必须把Nand里面的数据先读到主存，然后CPU才能够执行。大量应用于固态硬盘SSD和U盘等（SLC、MLC、TLC、QLC、3D NAND）
	Nor 或非	采用并行接口，读取速度比Nand快很多倍，程序可以直接在Nor里面运行。但其擦除速度比较慢，集成度低，成本高。一般用在代码量小的嵌入式产品方面

章节内容

- RISC技术和流水线技术
- ARM体系结构简介
- ARM处理器工作状态
- ARM处理器工作模式
- ARM处理器寄存器组织
- ARM异常
- ARM存储器
- ARM总线技术
- ARM存储系统
- **基于JTAG的调试系统**

2.10 基于JTAG的调试系统

- 基于JTAG的调试系统结构一般包括3部分：
  - 位于主机上的调试器，如ARM公司的AXD调试器
  - 包括硬件嵌入式调试部件的目标系统
  - 在主机和目标系统之间进行协议分析、转换的模块
    如JTAG仿真器



图2.12 基于JTAG的调试系统结构

本章小结

RISC技术和流水线技术

ARM体系结构的发展和特征

ARM处理器工作状态和工作模式

ARM处理器寄存器组织

ARM异常处理和存储器（技术）

ARM总线技术和调试系统简介