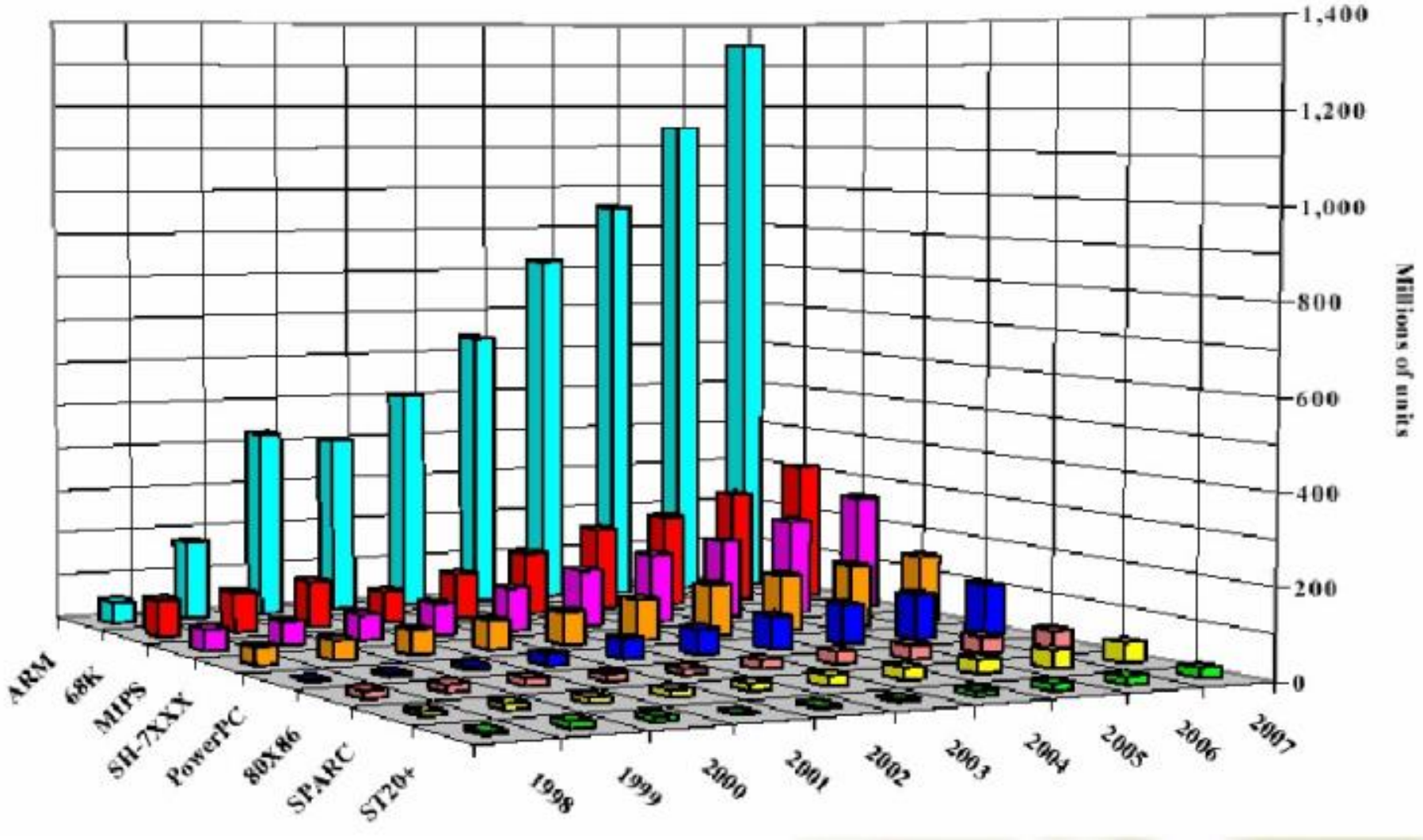
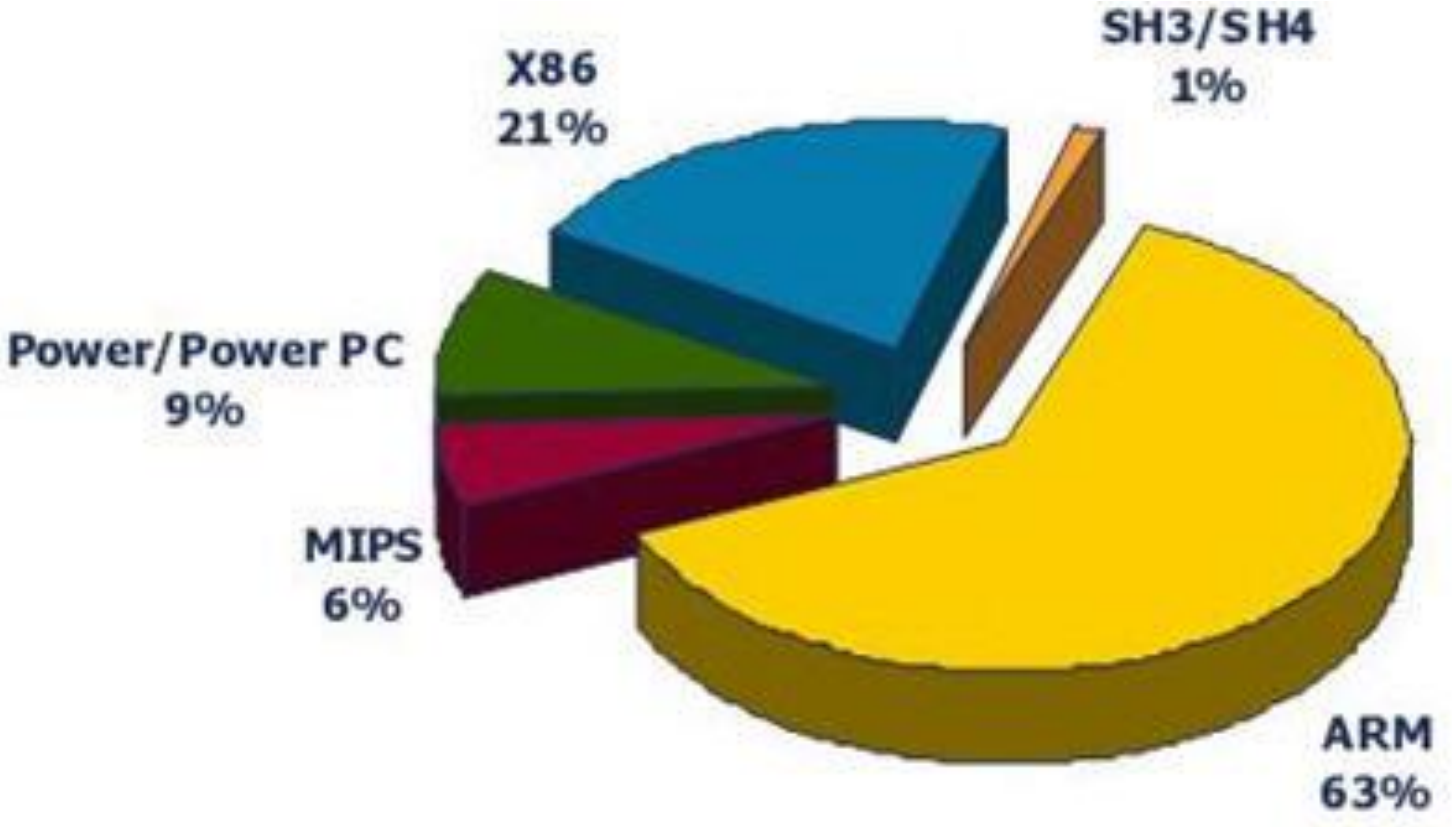
第2章 ARM微处理器概述

ARM处理器的使用情况



ARM处理器的使用情况



本章目标

- ARM微处理器的应用与选型
- ARM微处理器的数据类型和工作状态
- ARM微处理器工作模式
- ARM微处理器的寄存器

2.1 ARM微处理器概述

- 问题
  - ARM微处理器有哪些技术特点，其应用在哪些领域？
- 重点
  - ARM微处理器的技术特点。
- 内容
  - ARM微处理器的技术特点和其应用领域。

2.1.1 ARM微处理器的技术特点

任务：了解ARM微处理器的技术特点。

特点

- 功耗低、成本低、性能高
- 支持Thumb（16 位）/ARM（32 位）双指令集，能很好的兼容8 位/16 位器件；
- 大量使用寄存器，大多数数据操作都在寄存器中完成；
- 寻址方式灵活简单，执行效率高；
- 通过专用的载入和存储指令访问存储器；
- 指令长度固定。

采用的其他技术

- 为提高指令的执行效率，所有的指令都可以条件执行；
- 同一条数据处理指令中包含算术逻辑单元处理和移位处理；
- 使用地址自动增加（减少）来优化程序中的循环处理；
- 载入和存储指令可以批量传输数据，从而提高数据传输效率。

2.1.2 ARM微处理器的应用领域

任务：了解ARM微处理器的应用领域

- 工业控制领域：
  - 作为32位 的RISC 架构，基于ARM 核的微控制器芯片不但占据了高端微控制器市场的大部分市场份额，同时也逐渐向低端微控制器应用领域扩展，ARM 微控制器的低功耗、高性价比，向传统的8 位/16 位微控制器提出了挑战。
- 无线通讯领域：
  - 目前已有超过85%的无线通讯设备采用了ARM 技术，ARM 以其高性能和低成本，在该领域的地位日益巩固。

- 网络系统：
  - 随着宽带技术的推广，采用ARM 技术的ADSL 芯片正逐步获得竞争优势。此外，ARM 在语音及视频处理上行了优化，并获得广泛支持，也对DSP 的应用领域提出了挑战。
- 消费类电子产品：
  - ARM 技术在目前流行的数字音频播放器、数字机顶盒和游戏机中得到广泛采用。
- 成像和安全产品：
  - 现在流行的数码相机和打印机中绝大部分采用ARM 技术。手机中的32位SIM 智能卡也采用了ARM 技术。

第2章 ARM微处理器概述

2.2 ARM微处理器体系结构

- 问题：
  - 什么是RISC？
  - 什么是CISC？
  - ARM体系结构目前有哪些版本，有哪些变种？
  - ARM微处理器目前包括几个系列，每个系列的特点有哪些？
- 重点：
  - RISC和CISC的定义
  - ARM体系结构版本和目前的ARM微处理器系列及特点
- 内容：
  - RISC体系结构
  - ARM版本及其变种
  - ARM微处理器系列及特点

2.2.1 RISC体系结构

任务：掌握RISC和CISXC体系结构的含义，RISC体系结构的优点。

RISC和CISC

- RISC 精简指令集计算机
  - -Reduced instruction Set Computer
  - （1）确定指令系统时，只选择使用频度很高的那些指令，在此基础上增加少量能有效支持操作系统和高级语言实现及其他功能的最有用的指令，让指令的条数大大减少，一般不超过100条。
  - （2）大大减少指令系统可采用的寻址方式的种类，一般不超过两种。简化指令的格式，使之也限制在两种之内，并让全部指令都具有相同的长度。
  - （3）让所有指令都在一个机器周期内完成。
  - （4）扩大通用寄存器的个数，一般不少于 32 个寄存器，以尽可能减少访存操作，所有指令中只有存(STORE)、取(LOAD)指令才可访存，其他指令的操作一律都在寄存器间进行。

RISC和CISC

- RISC
  - （5）为提高指令执行速度，大多数指令都采用硬联控制实现，少数指令采用微程序实现。
  - （6）CPU硬件结构设计更为简单，包含较少的单元电路，因而面积小、功耗低。
  - （7）采用RISC结构的单片机数据线和指令线分离，即所谓的哈佛结构。
  - （8）计算机指令多为单字节，程序存储器的空间利用率大提高，有利于实现超小型化

RISC和CISC

- CISC 复杂指令集计算机
  - Complex Instruction Set Computer
  - （1）具有大量的指令和寻址方式，一般CISC计算机所含的指令数目至少300条以上，有的甚至超过500条。
  - （2）8/2原则：80%的程序只使用20%的指令。
  - （3）大多数程序只使用少量的指令就能够运行。
  - （4）CISC CPU包含有丰富的单元电路，因而功能强、面积大、功耗大。
  - （5）处理器在分析每一条指令之后执行一系列初级指令运算来完成 所需的功能。
  - （6）采用CISC结构的计算机数据线和指令线是分时利用的，即所谓的冯诺依曼结构

RISC和CISC

指标	RISC	CISC
指令集	一个周期执行一条指令，通过简单指令的组合实现复杂操作；指令长度固定	指令长度不固定，执行需要多个周期
流水线	流水线每周期前进一步	指令的执行需要调用微代码的一个微程序
寄存器	更多通用寄存器	用于特定目的的专用寄存器
LOAD/STORE结构	独立的LOAD和STORE指令完成 数据在寄存器和外部存储器之间的传输	处理器能够直接处理存储器中的数据

第2章 ARM微处理器概述

2.2.2 ARM体系结构版本

任务：了解ARM体系结构的各个版本及其特点

体系结构

- 体系结构定义了指令集（ISA）和基于这一体系结构下处理器的编码模型。基于同种体系结构可以有多种处理器，每个处理器性能不同，所面向的应用不同，每个处理器的实现都要遵循这一体系结构。
- ARM指令集体系结构，从最初开发至今已有了重大改进，而且将会不断完善和发展。为了精确表达每个ARM实现中所使用的指令集，到目前ARM体系结构共定义了8个版本，各版本特点如下：

Version 1（v1）

- 基本数据处理指令（不包括乘法指令）；
- 字节、字以及半字加载和存储；
- 软件中断指令；
- 分支指令；
- 26 位地址总线。

Version 2（v2）

- 该版本增加了下列指令：
- 乘法和乘加指令（Multiply & Multiply-accumulate）；
- 支持协处理器的指令；
- 对于FIQ模式，提供了额外的两个备份寄存器；
- SWP指令及SWPB指令；
- 26 位地址总线。

Version 3（v3）

- 该版本推出32位寻址能力，主要结构扩展变化为：
- 32 位地址总线，但除版本3G（版本3的一个变种）外其他版本是向前兼容的，支持26 位地址总线；
- 当前程序状态信息从原来的R15移到一个新的寄存器CPSR（Current Program Status Register，当前程序状态寄存器）中；
- 增加了SPSR（Saved Program Status Register，备份程序状态寄存器），用于在程序异常中断程序时，保存被中断程序的程序状态；
- 增加了两种处理器模式，使操作系统代码可以方便地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；
- 增加了指令MSR和MRS，用于访问CPSR和SPSR；
- 增加了从异常返回的指令。

Version 4（v4）--- ARM7，ARM9

- 该版本增加了下列指令：
- 半字加载和存储指令；
- 加载带符号的字节和半字数据的指令；
- 增加了T变种，可以使处理器状态切换到Thumb状态；
- 增加了处理器的特权模式。
- 该版本不再强制要求与以前的26位地址空间兼容。

Version 5（v5）--- ARM9E/ ARM10和XScale

- 提高了ARM指令集和Thumb指令集的混合使用的效率。
- 增加了前导零计数CLZ（Count Leading Zeros）指令，该指令可以使整数除法和中断优先级排队操作更为有效；
- 增加了软件断点（BKPT）指令；
- 更加严格地定义了乘法指令对条件标志位的影响。
- V5TE版增加了数字信号处理指令。

Version 6（v6）---ARM11

- 具备高性能定点DSP功能
- 引入全新的Jazelle技术。
- 支持SIMD（Single Instruction Multiple Data，单指令流多数据流）技术。
- 支持微处理器内核。

Version 7（v7）---cortex-a8/a9

- 被称作CorTex系列；定义了三大系列：“A”、“R”，“M”
- 架构采用了Thumb-2技术，它是在ARM的Thumb代码压缩技术的基础上发展出来的，并且保持了对已存ARM解决方案的完整的代码兼容性。
- 支持改良的浮点运算
- 支持改良的运行环境，来迎合不断增加的JIT（Just In Time）和DAC（Dynamic Adaptive Compilation）技术的使用。

Version 8（v8）

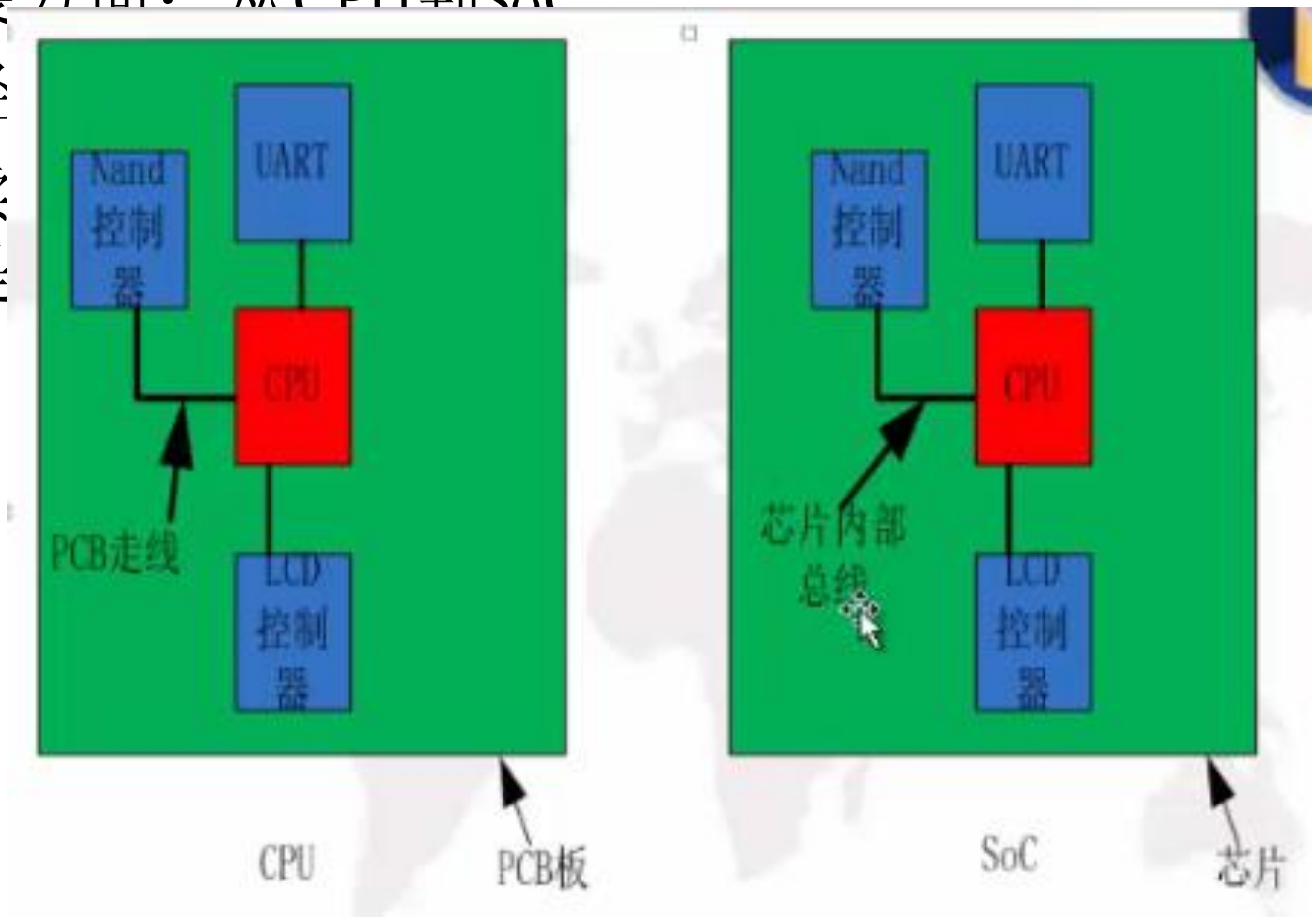
- 2011年11月，ARM公司发布了新一代处理器架构ARMV8，ARM的首个64位架构。
- 在继承了V7架构的基础上，可以选择64或32执行状态。64执行状态针对64位处理技术，引入了一个全新指令集A64，可以存取大虚拟地址空间。

ARM型号的发展过程

内核版本号	SoC版本号	芯片型号（三星）
ARMv1-ARMv3		
ARMv4	ARM7	S3C44B0
ARMv4	ARM9	S3C2440 S3C2410
ARMv5	ARM9 xScalse	
ARMv6	ARM11	S3C6410
ARMv7	Cortex-M Cortex-A Cortex-R	

M: microconctroller 微控制器就是单片机
A: application 应用级处理器 手机，电脑，平板
R: realtime 实时处理器 响应速度快，主要用在工业，航天

- SoC: system on chip
- 芯片发展方面：从CPU到SoC
- 现在已经
- ARM出卖
  种外设是



2.2.3 ARM体系结构的变种及版本命名格式

任务：了解ARM体系结构的各变种的特点及版本命名格式。

ARM体系结构的变种

- （1）T变种
  - T变种是支持Thumb指令集的ARM体系。Thumb指令集是将32位ARM指令集的一个子集重新编码而形成的一个指令集。Thumb指令的长度是16位。Thumb指令集可以得到比ARM指令集密度更高的代码，能够以16位的指令实现32位的指令功能，这对需要严格控制产品成本的设计是非常有意义的。目前Thumb指令集有两个版本：Thumb-1和Thumb-2。Thumb-1是ARM体系版本4的T变种，Thumb-2是ARM体系版本5的T变种。

ARM体系结构的变种

- （2）M变种（长乘法指令）
  - 长乘法指令是一种生成64位乘法结果的乘法指令。M变种增加了两条用于进行长乘法操作的ARM指令：一条用于完成32位整数乘以32位整数生成64位整数结果的长乘法操作；另一条用于完成32位整数乘以32位整数，再加上64位整数生成64位整数结果长乘加法操作。
  - 对于支持长乘法指令的ARM体系版本，用字母M来表示。M变种首先在ARM体系版本3中引入，在ARM体系版本4及其以后的版本中，M变种是系统的标准部分，所以字母M通常也不单独列出来。

ARM体系结构的变种

- （3）E变种
  - E变种增加一些附加指令用于增强处理器对一些典型的DSP算法的处理性能，主要包括：
  - 几条新的实现16位数据乘法和乘加操作的指令；
  - 实现饱和的带符号数的加减法操作的指令。所谓饱和的带符号数的加减法操作是在加减法操作溢出时，结果并不进行卷绕，而是使用最大的整数或最小的负数来表示；
  - 进行双字数据处理的指令，包括双字加载指令LDRD、双字存储指令STRD和协处理器的寄存器传输指令MCRR/MRRC；
  - cache预取指令PLD。

ARM体系结构的变种

- （4）J变种（Java加速器Jazelle）
  - ARM的Jazelle技术将Java的优势和先进的32位RISC芯片完美地结合在一起。Jazelle技术提供了Java加速功能，可以得到比普通Java虚拟机高得多的性能。与普通的Java虚拟机相比，Jazelle使Java代码运行速度提高了8倍，而功耗降低了80%。
  - Jazelle技术使得程序员可以在一个独立的处理器上同时运行Java应用程序、已经建立好的操作系统、中间件以及其他的应用程序。与使用协处理器和双处理器相比，使用单独的处理器可以在提供高性能的同时保证低功耗和低成本。
  - J变种首先在ARM体系版本4TEJ中使用，用字母J表示。

ARM体系结构的变种

- （5）SIMD变种
  - ARM媒体功能扩展SIMD技术极大地提高了嵌入式应用系统的音频和视频处理器能力，它可以使微处理器的音频和视频性能提高4倍。其主要特点如下：
  - 使处理器的音频和视频性能提高了2~4倍；
  - 可同时进行2个16位操作数或者4个8位操作数的运算；
  - 用户可自定义饱和运算的模式；
  - 可进行2个16位操作数的乘加/乘减运算及32位乘以32位的小数乘加运算；
  - 同时8/16位选择操作。

ARM体系结构版本的命名格式

ARMv5TExP

- 基本字符串ARMv；
- 基本字符串ARMv后是ARM指令集版本号，目前是数字1~6；
- ARM指令集版本号后是表示所含变种的字符；
- 最后使用的字符x表示排除某种功能。例如，在早期的一些E变种中，未包含双字加载指令LDRD、双字存储指令STRD、协处理器的寄存器传输指令MCRR/MRRC以及cache预取指令PLD。这种E变种记作ExP，其中x表示缺少，P代表上述的几种指令；

ARMv5TExP表示ARM体系结构的版本5，含T变种、E变种，未包含P。

第2章 ARM微处理器概述

2.2.4 ARM微处理器系列

任务：了解每一个ARM微处理器系列的特点和应用领域。

- 通用系列：

  ARM7 系列（低端的）

  ARM9 系列（低端的）

  ARM9E 系列（中端的）

  ARM10E 系列（中端的）

  ARM11系列（高端的）
- SecurCore 系列（安全系列）
- Intel 的Xscale
- Intel 的StrongARM
- Cortex系列

ARM7微处理器系列 特点

- 世界上最为广泛使用的CPU之一
- 特点：

极低的功耗：0.28mW/MHz，适合对功率要求较高的应用。

具有嵌入式ICE-RT逻辑，调试开发方便

能够提供0.9MIPS/MHz的三级流水线结构

支持操作系统：μC/OS-Ⅱ、μCLinux等。

性能最高可达130MIPS

- 应用领域：

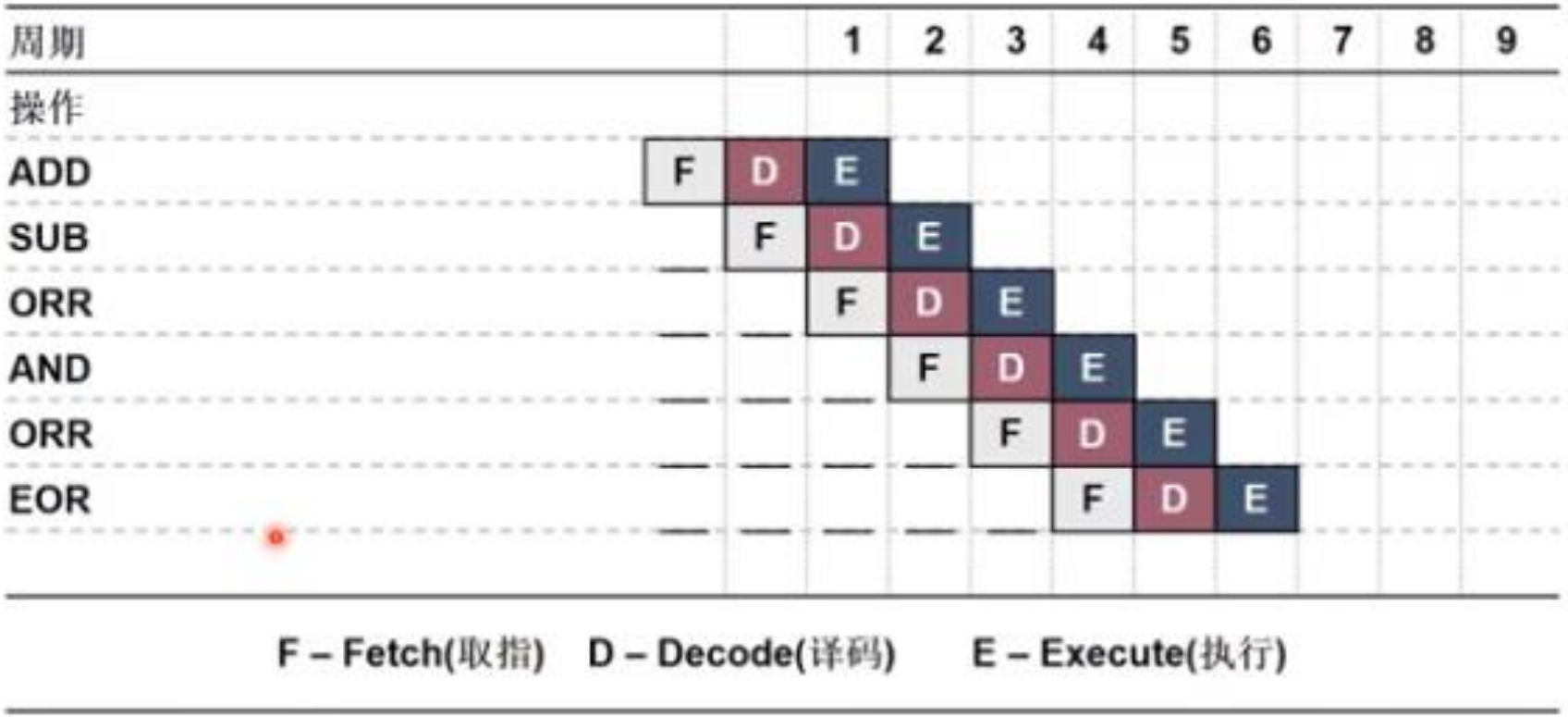
工业控制、网络和调制解调器（ADSL）

移动电话、消费电子等

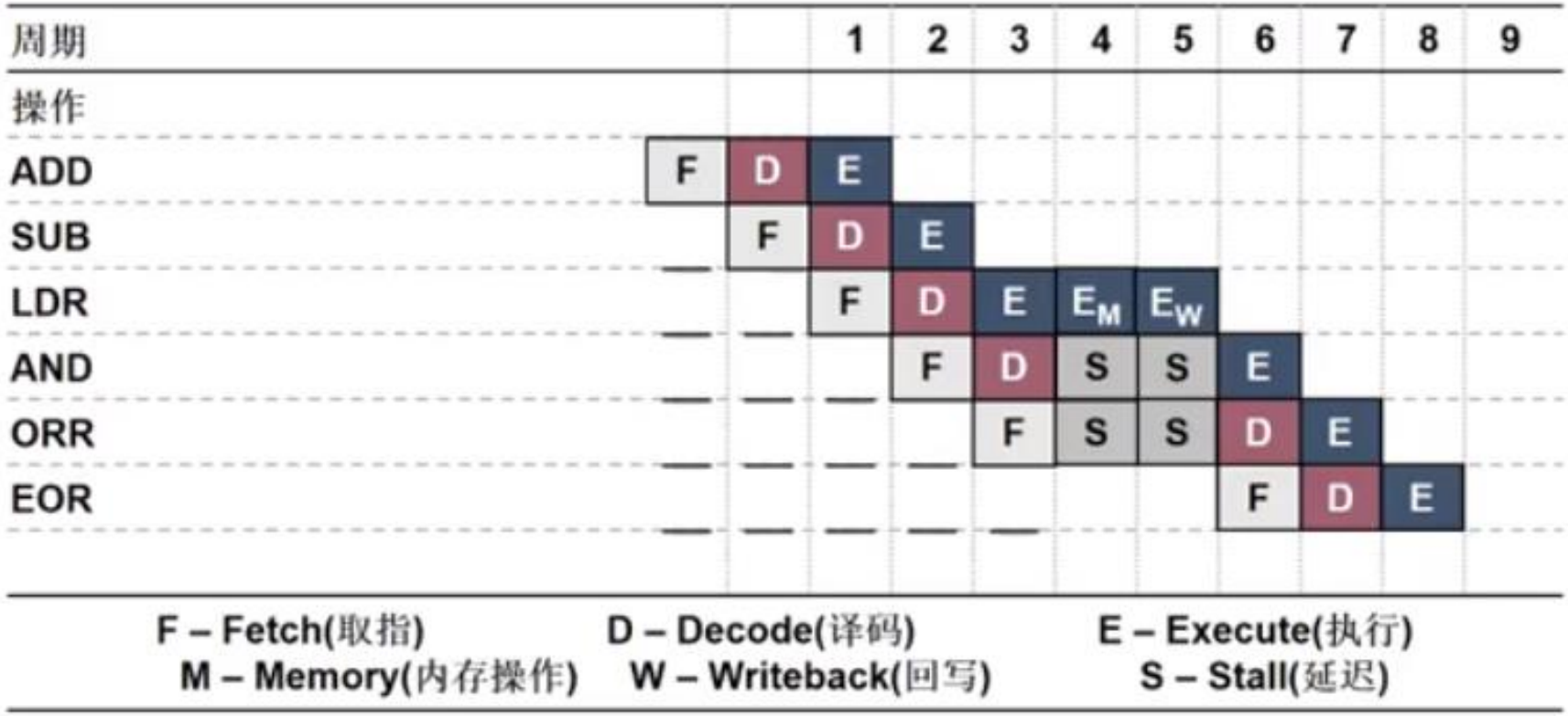
ARM7微处理器系列

- ARM7TDMI、ARM7TDMI-S、ARM720T
- ARM7TDMI属低端ARM处理器核
- ARM7TMDI的TDMI基本含义为：
- T：支持16位压缩指令集Thumb；
- D：支持片上Debug，使处理器能够停止以响应调试请求；
- M：内嵌硬件乘法器（Multiplier）；
- I：嵌入式ICE，支持片上断点和调试点。

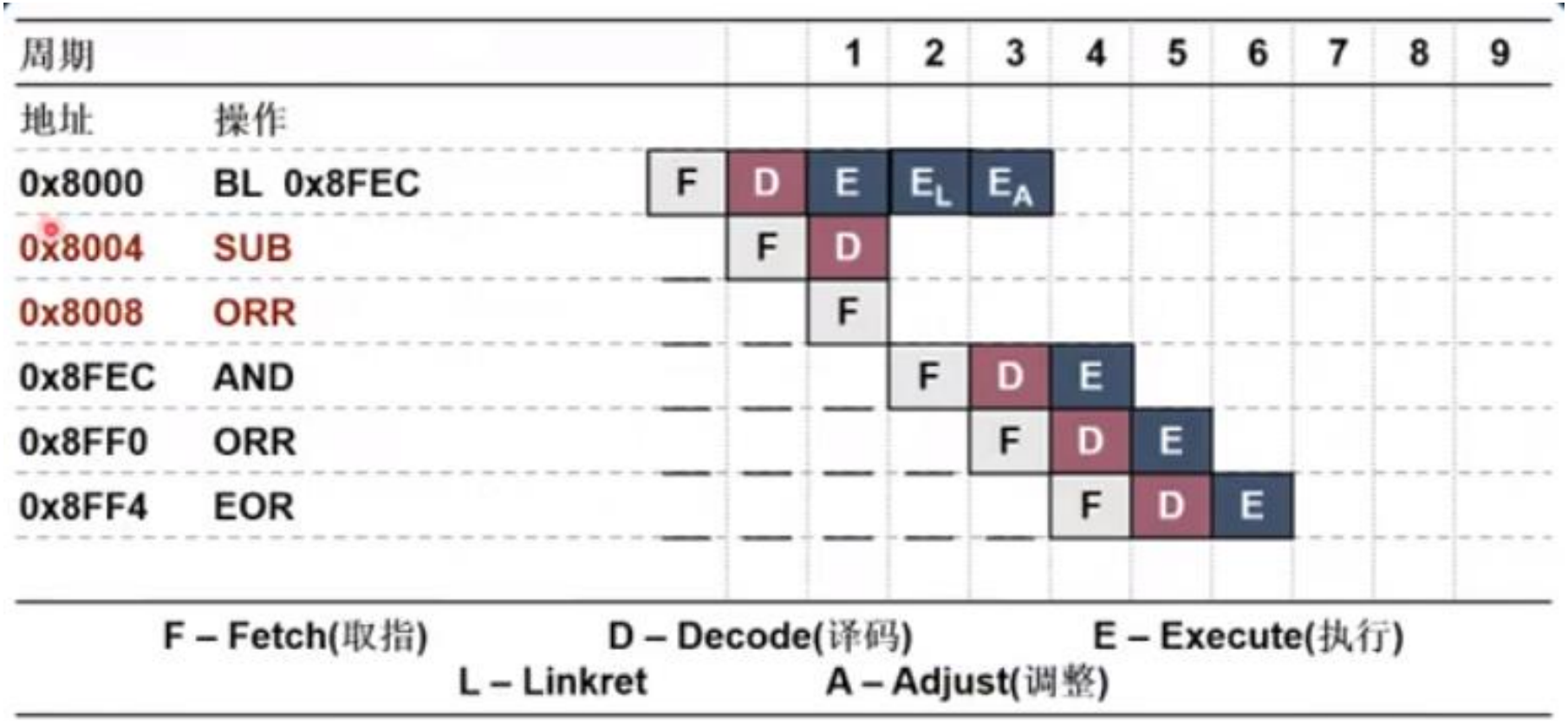
- ARM7TDMI使用3级流水线来提高对处理器指令流的速度，从而允许一些操作同时进行。
- 需要注意的是，PC指向的是将要被 预取的指令，而不是执行的指令。一般调试工具会向你隐藏这个细节。
- ARM　Thumb
- PC　　PC　　**取指**　从内存读取指令
- PC-4　　PC-2　　**解码**　对已取指令的寄存器进行解码
- PC-8　　PC-4　　**执行**　读寄存器的值，移位和运算操作，将结果回写寄存器



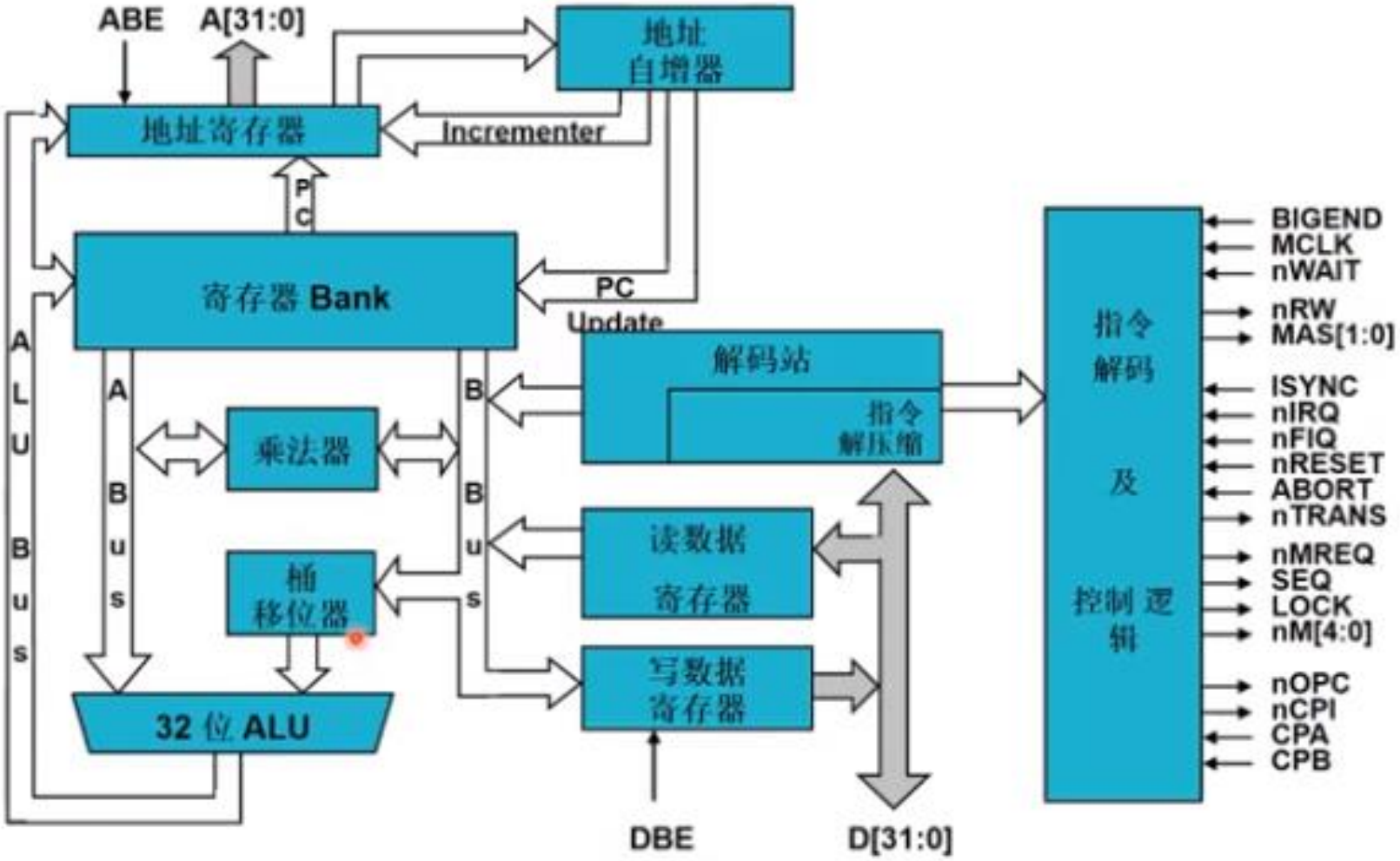
- 所有的操作都是在寄存器中进行
- 6个时钟周期执行6条指令
- Clock cycles per Instrution(CPI)=1



- 存在一个LDR的内存读取操作指令
- 6个时钟周期执行4条指令
- Clock cycles per Instrution(CPI)=1.5



分支指令的存在可能打断流水线的运行


ABE
A[31:0]
地址
自增器
地址寄存器
Incrementer
PC
寄存器 Bank
PC
Update
解码站
指令
解压缩
乘法器
桶
移位器
读数据
寄存器
写数据
寄存器
32 位 ALU
A
B
ALU Bus
A Bus
B Bus
DBE
D[31:0]
指令
解码
及
控制 逻
辑
BIGEND
MCLK
nWAIT
nRW
MAS[1:0]
ISYNC
nIRQ
nFIQ
nRESET
ABORT
nTRANS
nMREQ
SEQ
LOCK
nM[4:0]
nOPC
nCPI
CPA
CPB

ARM9微处理器系列

- 特点：
- 5 级整数流水线，指令执行效率更高；CPI约为1.5
- 提供1.1MIPS/MHz 的哈佛结构；
- 支持32 位ARM 指令集和16 位Thumb 指令集；
- 支持32 位的高速AMBA 总线接口；
- 全性能的MMU，支持Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统；
- MPU 支持实时操作系统；
- 支持数据cache 和指令cache，具有更高的指令和数据处理能力。（最好不用）
- 性能最高可达300MIPS。

ARM922T

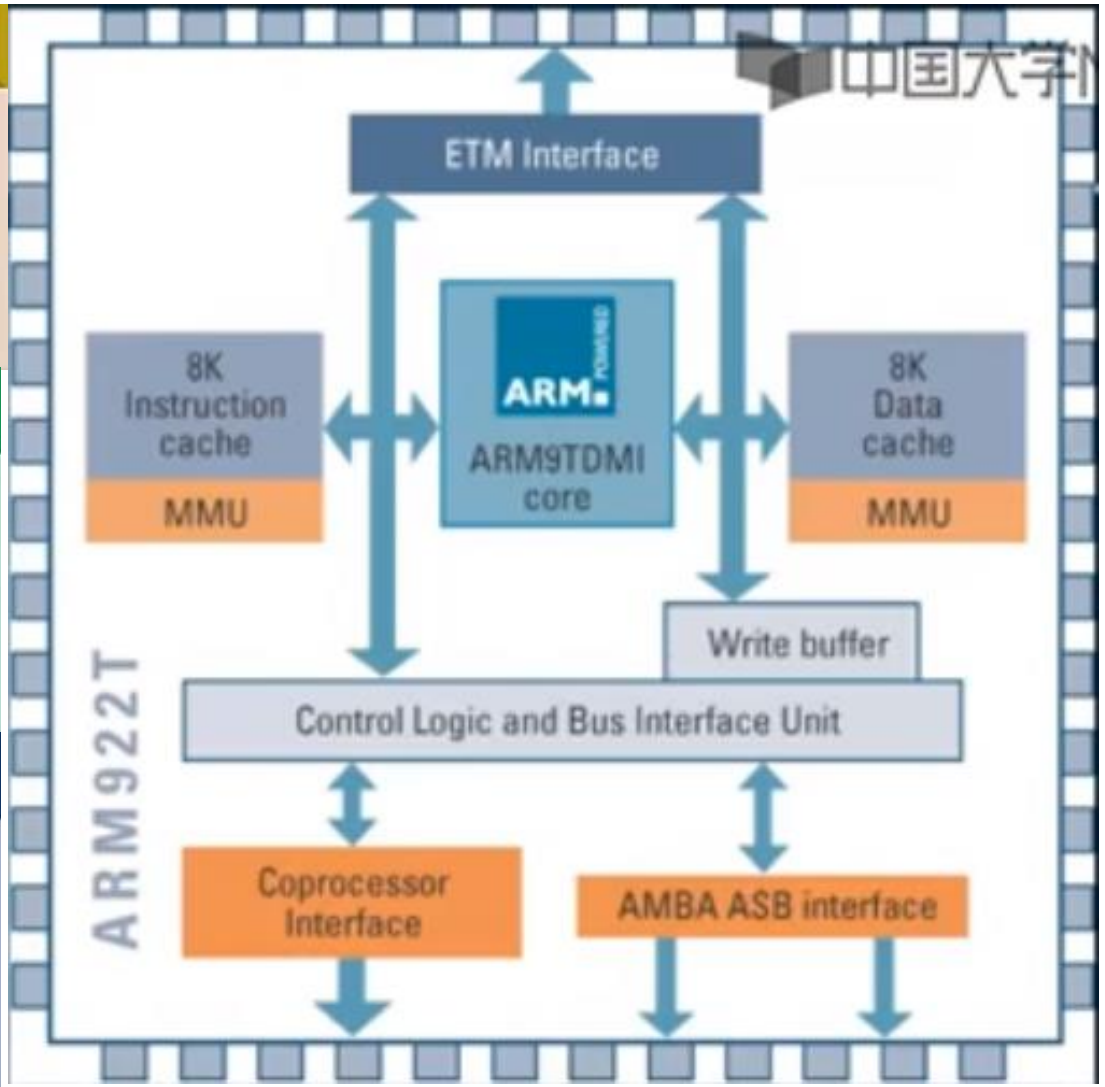
- 2 x 8K 高速缓存 (caches)
- 内存管理单元 (MMU)
- 写缓存 (Write Buffer)

ARM920T

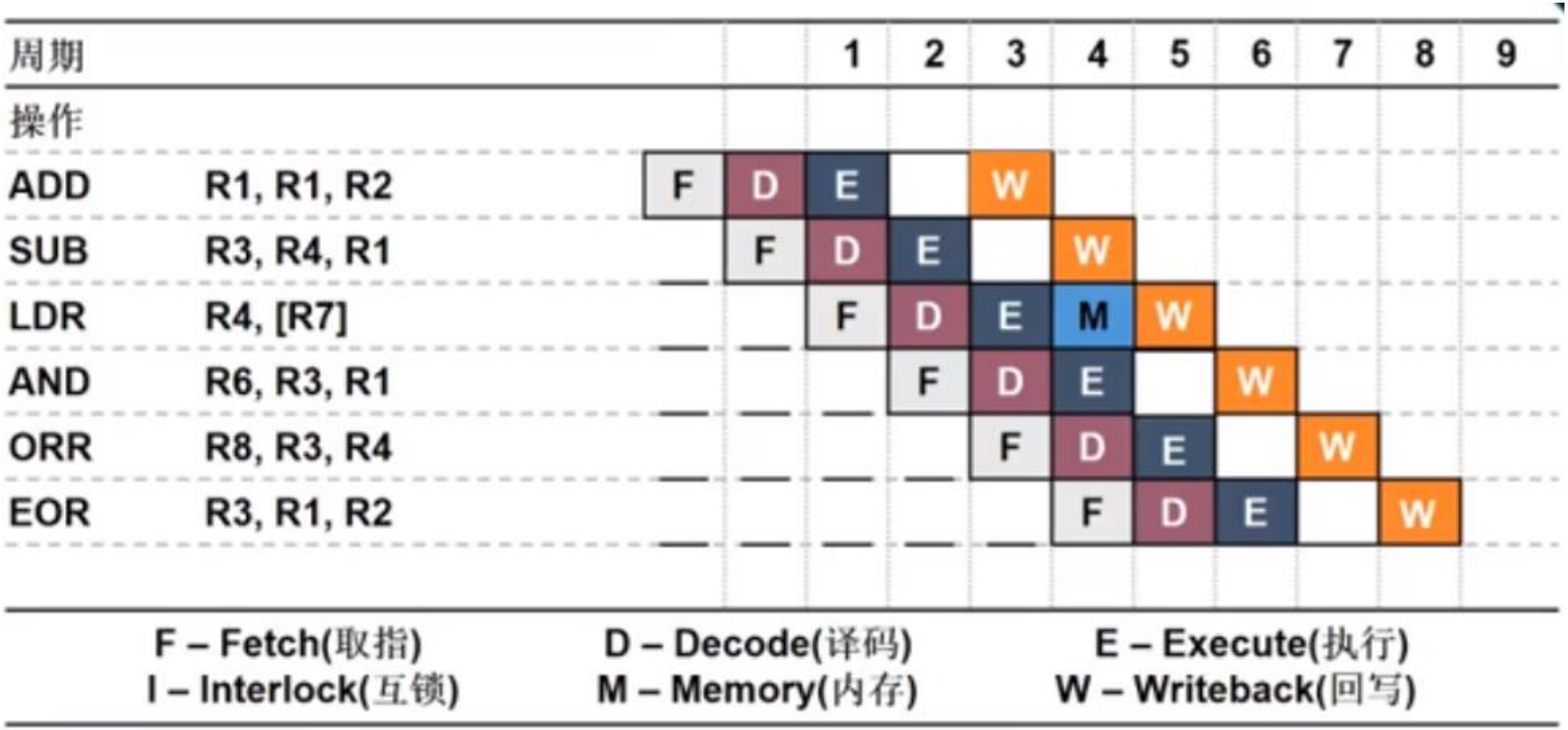
- 2 x 16K caches
- MMU
- 写缓冲

ARM940T

- 2x 4K caches
- MPU
- 写缓冲

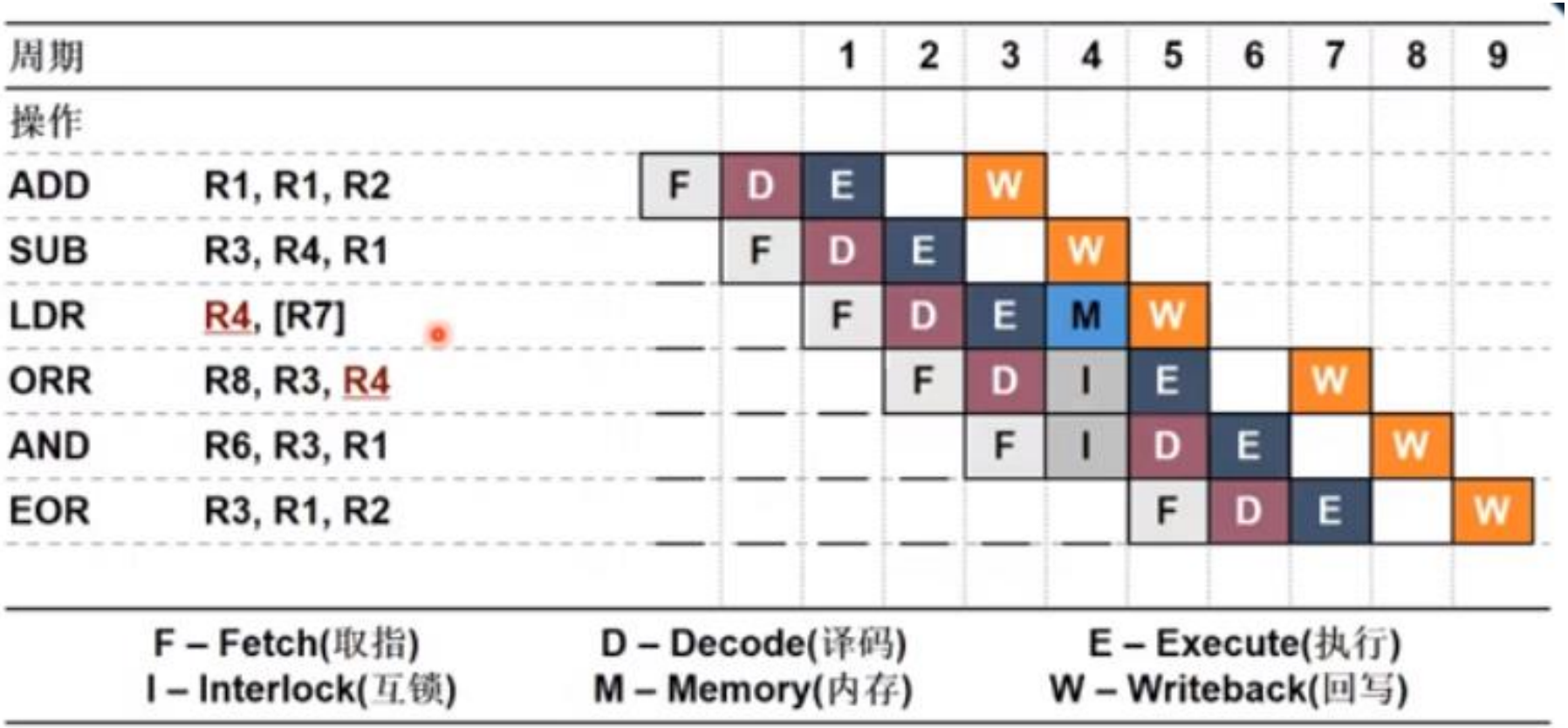


ARM9TDMI的理想流水线



LDR指令不会引起流水线互锁

6个时钟周期执行6条指令，CPI=1



紧接着LDR指令后用相同寄存器的数据操作会引起流水线互锁

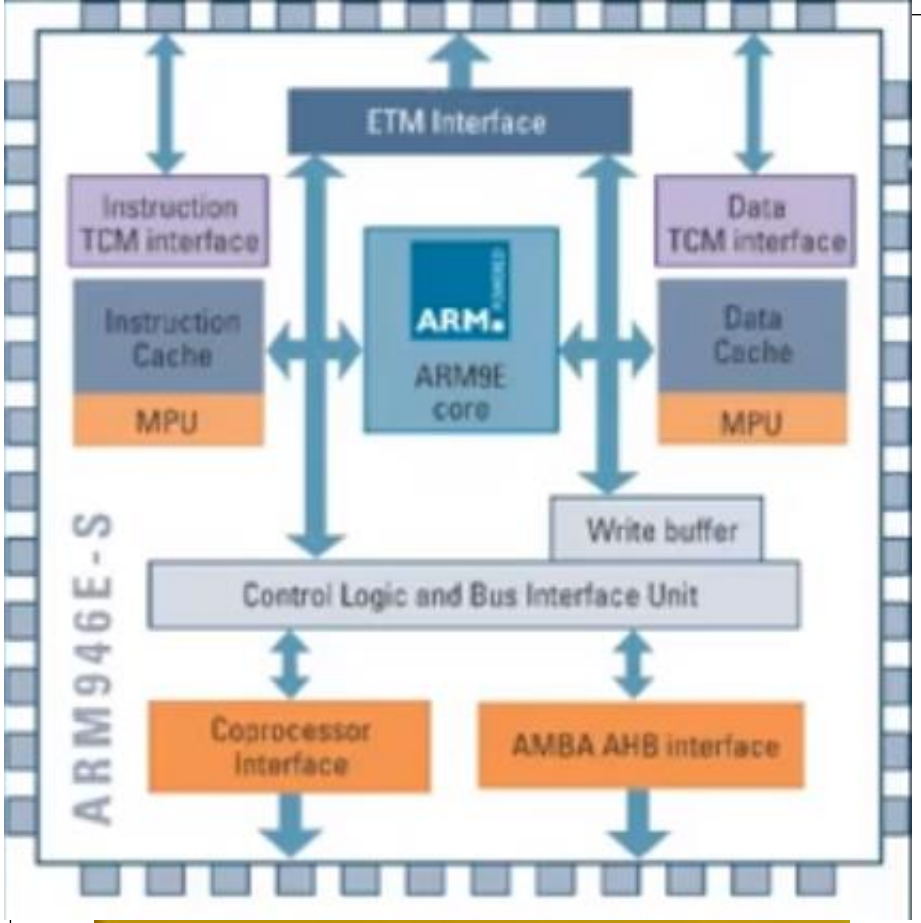
7个时钟周期执行了6条指令，CPI=1.2

ARM9微处理器系列 主要应用

- 无线设备
- 机顶盒
- 高端打印机、数字照相机、数字摄像机
- 汽车电子

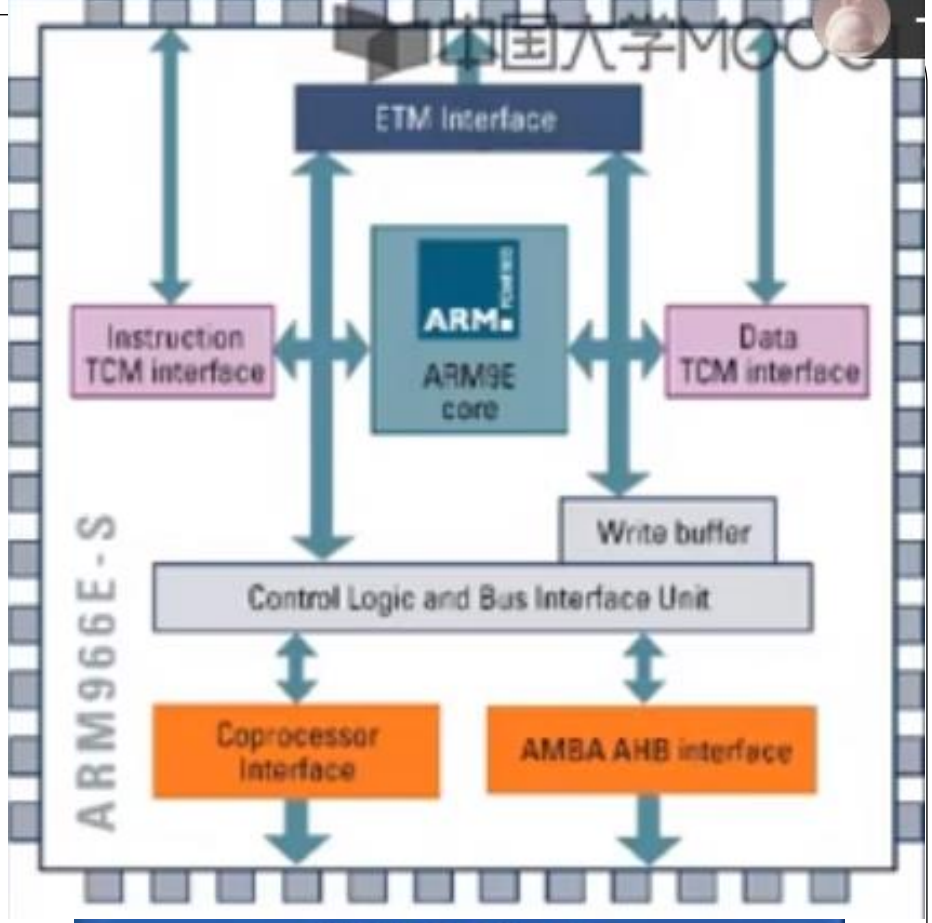
ARM9E微处理器系列

- 主要特点如下：
- 以ARM9TDMI为基础
- 支持DSP 指令集，适合于需要高速数字信号处理的场合；
- 5 级整数流水线，指令执行效率更高；
- 支持32 位ARM 指令集和16 位Thumb 指令集；
- 支持32 位的高速AMBA总线接口；
- 支持VFP9 浮点处理协处理器；
- 全性能的MMU，支持Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统；
- MPU 支持实时操作系统；
- 支持数据cache 和指令cache，具有更高的指令和数据处理能力；
- 主频最高可达300MIPS。



ARM926EJ-S / ARM946E-S

- 可配置的指令和数据cache
- 指令和数据的TCM接口
- ARM926EJ-S具有MMU
- ARM946E-S具有MPU



ARM966E-S

- 指令和数据的TCM接口
- 没有Cache，MMU和MPU

ARMIOE微处理器系列

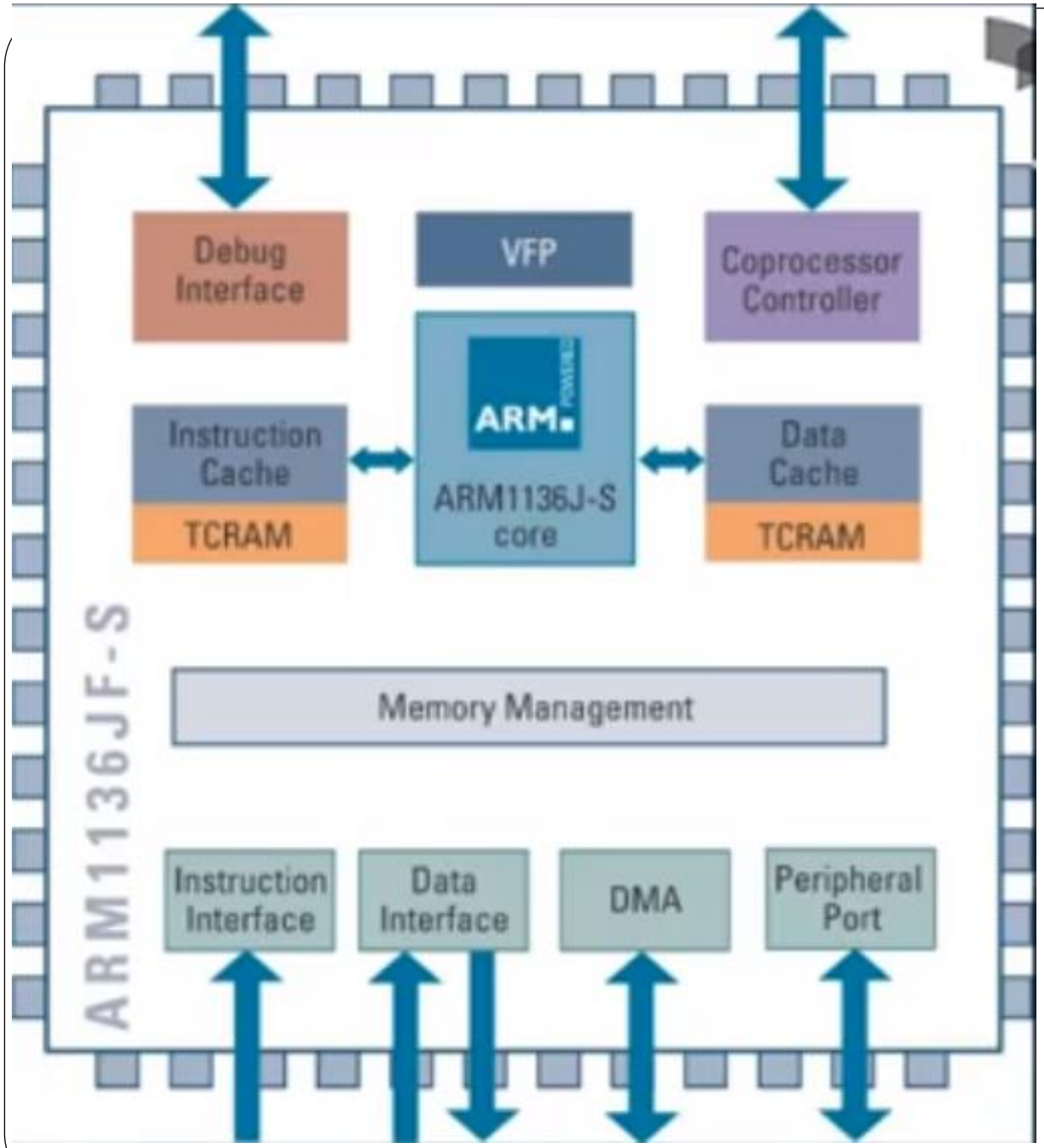
- 主要特点如下：
- 支持DSP 指令集，适合于需要高速数字信号处理的场合；
- 6 级整数流水线，指令执行效率更高；
- 支持32 位ARM 指令集和16 位Thumb 指令集；
- 支持32 位的高速AMBA 总线接口；
- 支持VFP10 浮点处理协处理器；
- 全性能的MMU，支持Windows CE、Linux、Palm OS 等多种主流嵌入式操作系统；
- 支持数据cache 和指令cache，具有更高的指令和数据处理能力；
- 主频最高可达400MIPS；
- 内嵌并行读/写操作部件。

ARMIOE微处理器系列 主要应用

- 手持设备
- 工业控制
- 数字消费品、汽车电子
- 图像处理设备

ARM11微处理器系列

- 高性能：
- 8 级整数流水线，指令执行效率更高；
- 增强的ARMv6体系结构
- 与同等的ARM10相比较，在同样的时钟频率下，性能提高了近50%。
- 时钟频率达到500-750MHz。
- 低功耗：
- 采用了两种先进的节能方式，使其功耗极低
- 0.6mW/MHz(0.13μm，1.2V)



ARM1136JF-S

- MMU

ARM1156T2(F)-S

- MPU
- 支持Thumb2指令集

ARM1176JZ(F)-S

- 和ARM1136JF-S相同
- 支持TrustZone

- arm系列从arm11开始，以后的就命名为cortex，并且性能上大幅度提升。
- 从cortex开始，分为三个系列，a系列，r系列，m系列。

- m系列偏向于控制方面，不带MMU（带MMU，芯片面积增加30%左右），不加载或者加载简单嵌入式操作系统。低成本，有取代arm7的倾向。
- a系列主要应用在人机互动要求较高的场合，比如pda，手机，平板电脑等。a系列类似于cpu，与arm9和arm11相对应，带操作系统，带MMU。
- r系列，是实时控制，不带MMU。主要应用在对实时性要求高的场合。

- arm7和m3，m4是同一类型。这三个里面，arm7是最早的arm产品。m3是cortex m系列的过渡品，其低端市场被cortex m0的高端替代，其高端市场又被cortex m4的低端取代。现在m系列，是m4内核的。典型的芯片是st公司和飞思卡尔公司的。
- arm9 和cortex a8 是一个类型的。
- 智能手机，就是用的cortex a8，cortex a9 内核的芯片作为cpu。

- ARM Cortex-A9处理器隶属于Cortex-A系列，基于ARMv7-A架构，目前我们能见到的四核处理器大多都是属于Cortex-A9系列。
- Cortex-A9 处理器的设计旨在打造最先进的、高效率的、长度动态可变的、多指令执行超标量体系结构，提供采用乱序猜测方式执行的 8 阶段管道处理器，凭借范围广泛的消费类、网络、企业和移动应用中的前沿产品所需的功能，它可以提供史无前例的高性能和高能效。

- ARM Cortex-A9处理器隶属于Cortex-A系列，基于ARMv7-A架构，目前我们能见到的四核处理器大多都是属于Cortex-A9系列。
- Cortex-A9 处理器的设计旨在打造最先进的、高效率的、长度动态可变的、多指令执行超标量体系结构，提供采用乱序猜测方式执行的 8 阶段管道处理器，凭借范围广泛的消费类、网络、企业和移动应用中的前沿产品所需的功能，它可以提供史无前例的高性能和高能效。

SecurCore微处理器系列

- 特点：
- 支持32 位ARM 指令集和16 位Thumb 指令集；
- 提供面向智能卡的和低成本的存储保护单元；
- 带有灵活的保护单元，以确保操作系统和应用数据的安全；
- 采用软内核技术，防止外部对其进行扫描探测；
- 可集成用户自己的安全特性和其他协处理器。

SecurCore微处理器系列

- 电子商务
- 电子银行
- 电子政务
- 网络和认证系统
- SecurCoreSC100
- SecurCoreSC110
- SecurCoreSC200
- SecurCoreSC210

StrongARM微处理器系列

- 特点：
- 支持32 位ARM 指令集和16 位Thumb 指令集；
- 实现了ARMv4体系结构，并且增加了cache容量，支持虚拟存储器管理；
- 协处理器CP15包含一些寄存器，用于控制用配置cache、写缓存、虚拟存储器管理MMU、读缓存、断点以及一些时钟功能等；
- 具有MMU部件，进行虚拟地址转换成物理地址和控制存储器访问权限；
- 虚拟存储器管理部件分为指令存储器管理单元和数据存储器管理单元；

Intel的Xscale微处理器系列

- 处理速度是Intel StrongARM处理速度的2倍，其内部结构也有了相应的变化：
  - 数据cache和指令cache均达到32KB；
  - 微小数据cache的容量达到2KB；
  - 具有7级超级流水线；
  - 新增乘法/加法器MAC和特定的DSP型协处理器CPO，以提高对多媒体技术的支持；
  - 动态电源管理，使Xscale 微处理器的时钟频率可达1GHz、功耗达1.6W，并能达到1200MIPS；
  - 超低功耗与良好性能的组合使Intel Xscale 微处理器广泛用于因特网的接入设备。

第2章 ARM微处理器概述

2.3.1 ARM微处理器的数据类型

任务：掌握ARM微处理器的数据类型有哪些

ARM微处理器的数据类型

- 字、半字、字节
- 字（Word）：字的长度为32位
- 半字（Half-Word）：半字的长度为16位
- 字节（Byte）：字节的长度为8位
- ARM处理器要求自然对界
- 字需要4字节对齐：地址的低两位为0
- 半字需要2字节对齐：地址的最低位为0
- 字节则是任意地址对齐：

- 边界对齐（数据存储）
- 如果一个数据是从偶地址开始的连续存储，那么它就是半字对齐，否则就是非半字对齐。
- 如果一个数据是以能被4整除的地址开始的连续存储，那么它就是字对齐，否则就是非字对齐。
- 半字对齐地址：0x00000000
- 0x00000002
- 0x00000004 Bit0=0
- 字对齐地址：0x00000000
- 0x00000004
- 0x00000008 Bit1=0;Bit0=0

第2章 ARM微处理器概述

2.3.5 ARM体系中的存储模式

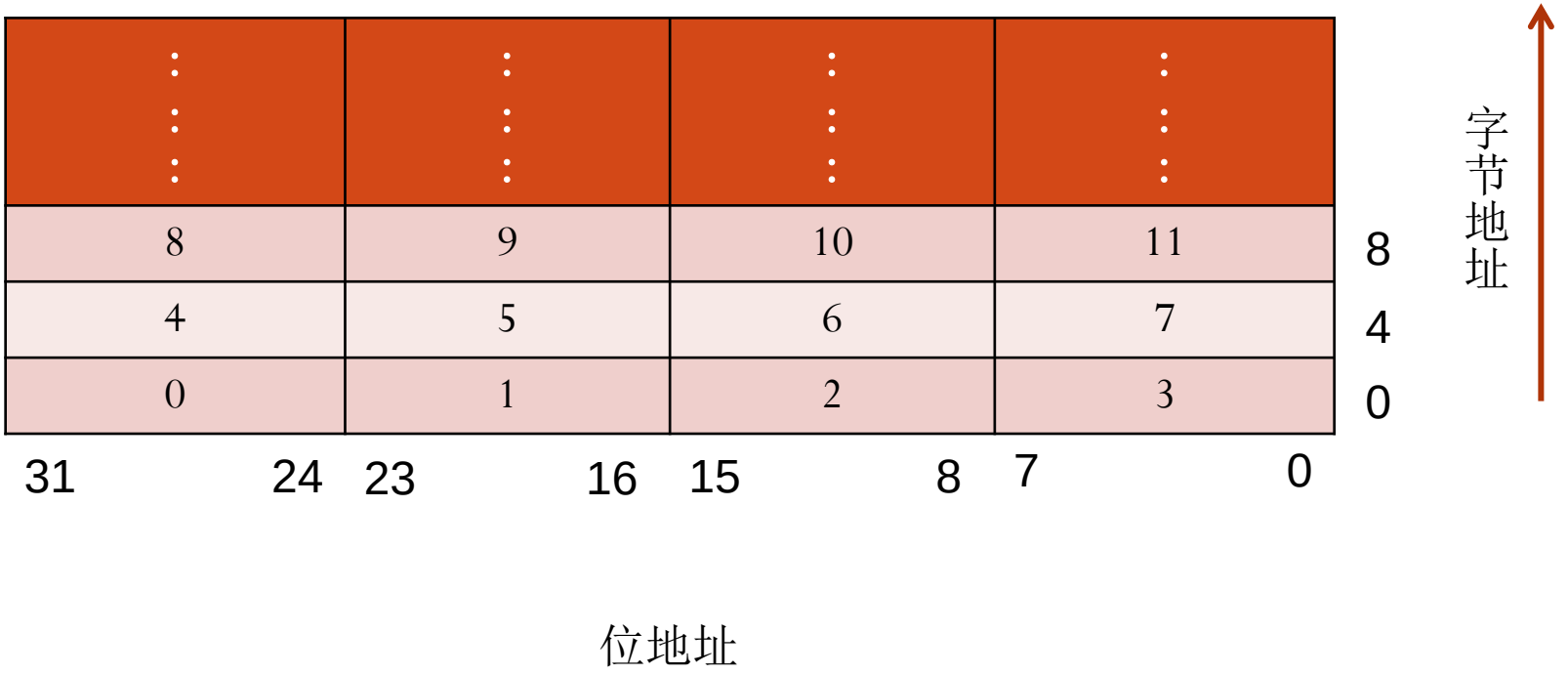
任务：掌握ARM体系中的两种存储模式：
大端模式和小端模式。

ARM体系结构的存储结构

- 从零地址开始的以字节为单位的线性组合
- 从零字节到三字节放置第一个存储的字数据，从第四个字节到第七个字节放置第二个存储的字数据，依次排列。
- 作为32位的处理器，最大寻址空间为4GB（2^{32}）
- 受到物理地址线的限制，一般实际嵌入式处理器最大寻址空间为几百兆。

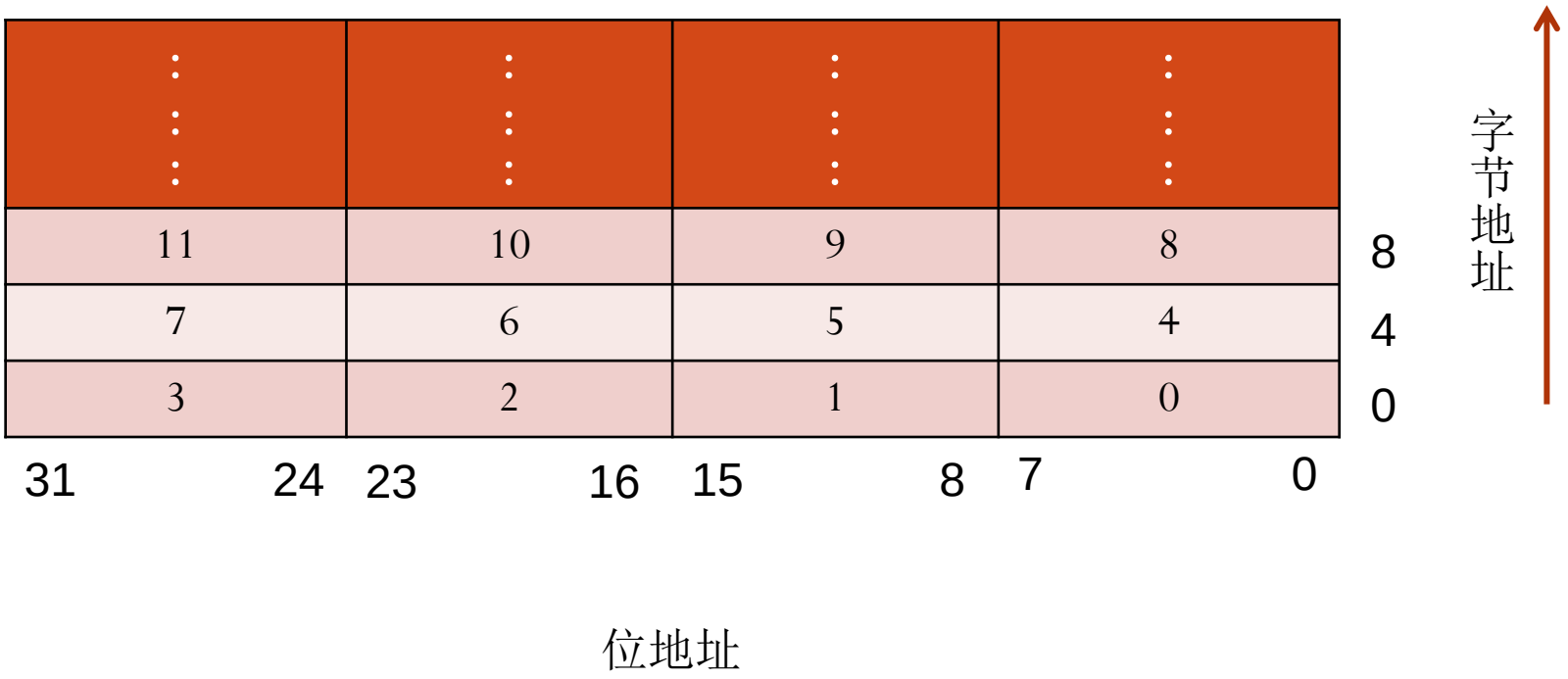
- ARM体系结构可以用两种方法存储字数据，称之为大端模式和小端模式。这两种模式是根据最低有效字节与相邻较高有效字节相比，是存放在较低的地址还是较高的地址来划分的。
- 小端模式：字的高位字节对应存储在高地址、低位字节对应存储在相邻的低地址。
- 大端模式：字的高位字节对应存储在低地址、低位字节对应存储在相邻的高地址。

大端格式（Big endian）



- 大端模式：字的高位字节对应存储在低地址、低位字节对应存储在相邻的高地址。

小端格式（Little endian）



- 小端模式：字的高位字节对应存储在高地址、低位字节对应存储在相邻的低地址。

例子：

- LDR R0，=0x11223344
- LDR R1, =0x00000100
- STR R0,[R1] //R0存储到R1所指的空间
- LDRB R2,[R1] //把一段内存区域加载到某个寄存器，仅加载了一个字节
- R2=?

LDR指令，用于从存储器中将一个32位的字数据传送到目的寄存器中。
LDR指令的形式：
LDR{条件} 目的寄存器Rn，<存储器地址>

LDR伪指令，使用以下项之一加载寄存器：
一个32位常数值
一个地址
LDR伪指令的形式：
LDR Rn， =EXPR； =而非#

- Struct{
- char a; 字节
- int b; 字
- char c; 字节
- short d;} 半字
- 问：在缺省（自然对界）情况下，上述结构需要多少存储空间（小端格式）？


d
c
b
a
12
8
4
0

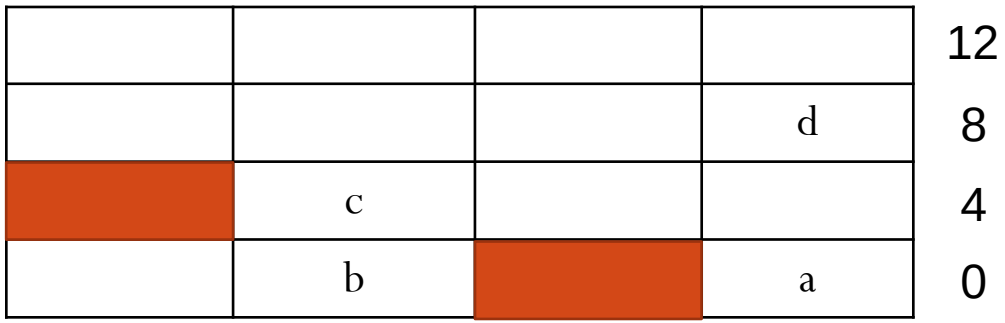
强制对界：
编译控制：
汇编align(1/2/4/8) unalign
C语言#pragma pack(1/2/4/8)

Struct{
  char a; 字节
  int b;   字
  char c;   字节
  short d;} 半字

强制字节对齐：

				12
				8
	d	c		4
		b	a	0

强制半字对齐



第2章 ARM微处理器概述

2.3.2 ARM微处理器的工作模式

任务：理解ARM微处理器的7种工作模式。

七种工作模式

- 用户模式（User）： ARM微处理器的正常运行模式，通常用来执行一般的应用程序；
- 快速中断模式（FIQ）：响应快速中断时的模式，由外部触发FIQ引脚，用于高速数据传输或通道处理；
- 外部中断模式（IRQ）： 响应一般的外部中断时的模式，由外部触发IRQ引脚；
- 管理模式（SVC）： 操作系统的保护模式；
- 中止模式(ABT)： 当数据或指令预取中止时进入该模式，可用于实现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软件仿真

七种工作模式

- 用户模式（User）： ARM微处理器的正常运行模式，通常用来执行一般的应用程序；
- （1）是用户程序的工作模式，
- （2）它运行在操作系统的状态
- （3）它没有权限去操作其它硬件资源，只能执行处理自己的数据。
- （4）也不能切换到其它模式下，要想访问硬件资源或切换到其它模式只能通过软中断（SWI）或产生异常。

七种工作模式

- 快速中断模式（FIQ）：响应快速中断时的模式，由外部触发FIQ引脚，用于高速数据传输或通道处理；
- （1）用来处理对时间要求比较紧急的中断请求
- （2）主要用于高速数据传输及通道处理中。

- 外部中断模式（IRQ）： 响应一般的外部中断时的模式，由外部触发IRQ引脚；
- （1）用于处理一般的中断请求
- （2）通常在硬件产生中断信号之后自动进入该模式
- （3）该模式为特权模式，可以自由访问系统硬件资源

七种工作模式

- 管理模式（SVC）： 操作系统的保护模式；
- （1）是CPU上电后默认模式
- （2）用来做系统的初始化
- （3）软中断处理也在该模式下，当用户模式下的用户程序请求使用硬件资源时通过软件中断进入该模式
- （4）中断模式会很快切换到管理模式下运行

- 中止模式(ABT)： 当数据或指令预取中止时进入该模式，可用于实现虚拟存储及存储保护；
- （1）用户程序访问非法地址，没有权限读取的内存地址时，进入该模式
- （2）Linux下编程时经常出现的segment fault通常都是在该模式下抛出返回的。

七种工作模式

- 系统模式（SYS）： 运行具有特权的操作系统任务；
- （1）是特权模式，不受用户模式的限制
- （2）用户模式和系统模式共用一套寄存 器
- （3）操作系统在该模式下可以方便的访问用户模式的寄存器，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。

- 未定义模式（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软件仿真
- CPU在指令的译码阶段不能识别该指令操作时，会进入未定义模式。

七种工作模式

- cortex-A9第8 种工作模式monitor
- 为了安全而扩展出的用于执行安全监控代码的模式
- 是一种特权模式

模式	描述	
Supervisor(SVC)	当复位或者软中断（SWI）指令被执行时进入	特权模式
FIQ	一个高优先级的快速中断产生时进入	
IRQ	一个低优先级的普通中断产生时进入	
Abort	用来处理内存访问异常	
Undefined	用来处理未定义的指令	
System	特权模式，与用户模式使用相同寄存器	
User	大多数应用程序/操作系统任务执行时的模式	非特权模式

- 特权模式：

除了用户模式以外，其余6种模式称之为特权模式（Privileged Modes)

当处理器运行在用户模式下时，某些被保护的系统资源是不能被访问的。

- 异常模式：

除去用户模式和系统模式以外的5种模式称为异常模式（Exception Modes）

常用于处理中断或异常，以及需要访问受修护的资源等情况。

异常源

异常源	描述
Reset	上电时执行
Undef	当流水线中的某个非法指令到达执行状态时执行
SWI	当一个软中断指令被执行完的时候执行
Prefetch	当一个指令被从内存中预取时，由于某种原因而失败，如果它能到达执行状态这个异常才会产生
Data	如果一个预取指令试图存取一个非法的内存单元，这时异常产生
IRQ	通常的中断
FIQ	快速中断

第2章 ARM微处理器概述

2.3.4 ARM微处理器的寄存器组织

任务：掌握ARM微处理器的寄存器组织。

寄存器

- 概念：
- 寄存器是处理器内部的存储器，没有地址
- 作用：
- 一般用于暂时存放参与运算的数据和运算结果
- 分类：
- 包括通用寄存器 、专用寄存器、控制寄存器

ARM微处理器37个32位寄存器

- 31个为通用寄存器。这些寄存器是32位的，分别是R0~R15、R13_svc、R14_svc、R13_abt、R14_abt、R13_und、R14_und、R13_irq、R14_irq、R13_fiq、R14_fiq;
- 6个为状态寄存器。状态寄存器也是32位的，但只使用了其中的12位。这些寄存器分别是CPSR、SPSR_svc、SPSR_abt、SPSR_und、SPSR_irq、SPSR_fiq。

ARM 状态下的通用寄存器和程序计数器					
SYS&User	**FIQ**	**SVC**	**ABT**	**IRQ**	**Undef**
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)
ARM状态下的程序状态寄存器					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und
◣分組寄存器					

图 2-1　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：
- 不分组寄存器R0～R7；
- 分组寄存器R8～R14；
- 程序计数器R15(PC)。

1.通用寄存器

- 不分组寄存器R0～R7；

在所有的工作模式下，每个未分组寄存器都指向对应的一个物理寄存器。

在中断或异常处理进行工作模式转换时，由于不同的处理器工作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。

1.通用寄存器

- 分组寄存器R8～R14；

R8-R12每个寄存器对应2个不同的物理寄存器

当使用FIQ模式时，访问R8_flq-R12_flq

当使用其他模式时，访问R8_usr-R12_usr

ARM 状态下的通用寄存器和程序计数器					
SYS&User	**FIQ**	**SVC**	**ABT**	**IRQ**	**Undef**
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)
ARM状态下的程序状态寄存器					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und
◣分組寄存器					

图 2-1　ARM 状态下的寄存器组织

1.通用寄存器

- R13、R14

  每个寄存器对应6个不同的物理寄存器

  其中一个寄存器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种不同的工作模式

- 不同物理寄存器的区分

  R13_<mode>　R14_<mode>

- Mode为：usr、flq、irq、svc、abt、und

  例如：R13_usr R13_flq R13_irq

  R14_svc R14_abt R14_und

ARM 状态下的通用寄存器和程序计数器					
SYS&User	**FIQ**	**SVC**	**ABT**	**IRQ**	**Undef**
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)
ARM状态下的程序状态寄存器					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und
◣分組寄存器					

图 2-1　ARM 状态下的寄存器组织

1.通用寄存器

- R13寄存器
- 常用作堆栈指针SP(Stack Pointer)，一种习惯用法。
- 也可会使用其他的寄存器作为堆栈指针
- 在Thumb指令中，某些指令强制使用R13作为堆栈指针
- 在应用程序初始化时，一般都要初始化每种模式下的R13，使其指向该工作模式的栈空间。

1.通用寄存器

- R14寄存器
- 也称链接寄存器LR(Link Register)
- 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
- BL label； 下一条指令地址->LR，label->PC
- 当发生中断或异常时，对应的分组寄存器R14_svc、R14_irq 、R14_flq、R14_abt和 R14_tmd用来保存R15的返回值。

1.通用寄存器

- R14寄存器常用情形
- 常用用法（子程序返回）

```
MOV  PC,LR
BX   LR
```

- 在子程序入口处使用以下指令将R14存入堆栈

```
STMFD  SP！，{<Regs>,LR}
```

使用以下指令可以完成子程序返回

```
LDMFD  SP！，{<Regs>,PC}
```

1.通用寄存器

- 程序计数器R15(PC)
- R15寄存器用作程序计数器（PC)
- 在ARM状态下，位【1：0】为0，位【31：2】用于保存PC
- 在Thumb状态下，位【0】为0，位【31：1】用于保存PC

1.通用寄存器

- 由于ARM体系结构采用了多级流水结构，对于ARM指令集而言，PC总是指向 当前执行指令的后面指令的地址。
- 比如arm7采用三级流水，那么执行ARM指令时，PC的值为当前执行指令的地址值加8个字节，即指向当前指令的后面两条指令的地址。

第2章 ARM微处理器概述

2.3.4 ARM微处理器的寄存器组织

任务：掌握ARM微处理器的寄存器组织。

2．程序状态寄存器

- ARM体系结构包含一个当前程序状态寄存器（CPSR）和五个备份的程序状态寄存器（SPSR）。
- CPSR:可在任何工作模式下被访问，它包括条件标志位，中断禁止位，当前处理器模式标志位，以及其他一些相关的控制和状态位。
- 异常模式下有一个专用的物理状态寄存器，称为SPSR（Saved Program Status Register)
- 当异常发生时，SPSR用于保存CPSR的当前值。从异常退出时则可由SPSR来恢复CPSR
- 用户模式和系统模式不属于异常模式，没有SPSR

2．程序状态寄存器

- ARM体系结构包含
  一个当前程序状态寄存器（CPSR）
  五个备份的程序状态寄存器（SPSR）。
  备份的程序状态寄存器用来进行异常处理
- 其功能包括：
- 保存ALU中的当前操作信息。
- 控制允许和禁止中断。
- 设置处理器的工作模式。

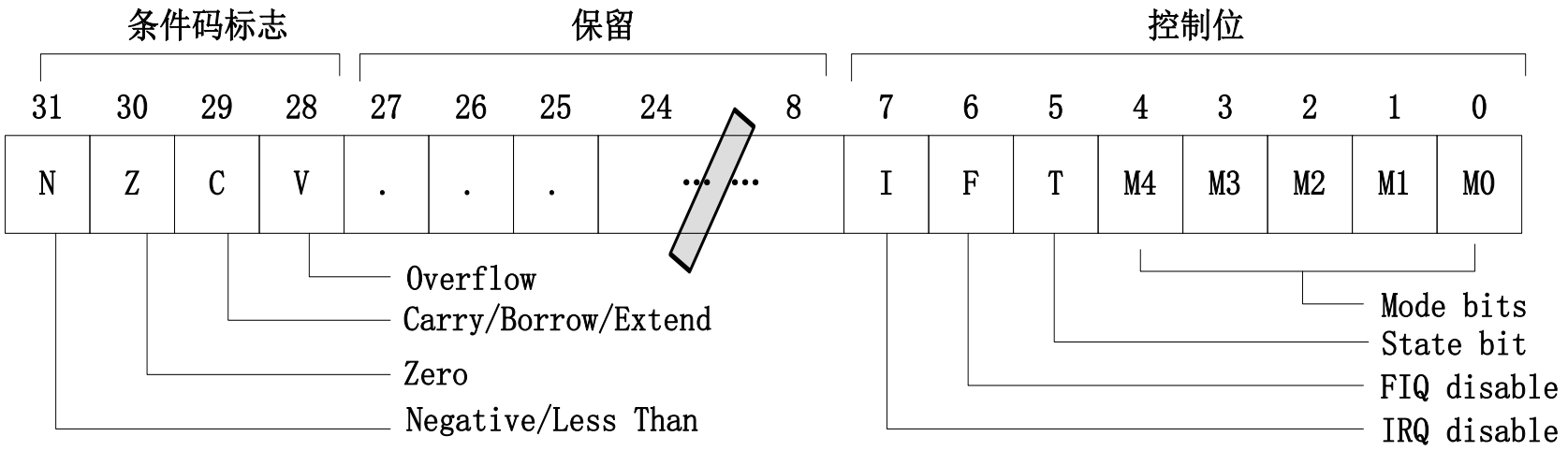


图2-2 程序状态寄存器格式

- 条件码标志（Condition Code Flags)
- N 、Z、C、V均为条件码标志位
- 它们的内容可被算术或逻辑运算的结果所改变，并且可以决定某条指令是否被执行
- 在ARM状态下，绝大多数的指令有条件执行的
- 在Thumb状态下，仅有分支指令B是有条件执行的

- 举例：C代码：　if(a>b) a++; else b++;
- Thumb代码：CMP　R0，R1
- 　　BHI　A_ADD
- 　　ADD　R1,R1,#1
- 　　B　　OVER
- A_ADD　ADD　R0,R0,#1
- OVER　　……
- ARM代码：　CMP　R0,R1
- 　　ADDHI　R0,R0,#1
- 　　ADDLS　R1,R1,#1

表2-1 条件码标志位的含义

标志位	含义
N	当用两个补码表示的带符号数进行运算时，N=1 表示运算的结果为负数；N=0 表示运算的结果为正数或零。
Z	Z=1 表示运算的结果为零； Z=0 表示运算的结果为非零。
C	可以有4种方法设置C的值： --加法运算（包括比较指令CMN）：当运算结果产生了进位时（无符号数溢出），C=1，否则C=0； --减法运算（包括比较指令CMP）：当运算时产生了借位（无符号数溢出），C=0，否则C=1； --对于包含移位操作的非加/减运算指令，C为移出值的最后一位； --对于其他的非加/减运算指令，C的值通常不受影响。
V	可以有2种方法设置V的值： --对于加/减运算指令，当操作数和运算结果为二进制的补码表示的带符号数时，V=1表示符号位溢出； --其他的指令通常不影响V位。
Q（27位）	在ARM V5及以上版本的E系列处理器中，用Q标志位指示增强的DSP运算指令是否发生了溢出；在其他版本的处理器中，Q标志位无定义。

表2-2 处理器的工作模式位M[4：0] 组合

M[4：0]	处理器模式	可访问的寄存器
0b10000	用户模式	PC,CPSR,R0-R14
0b10001	FIQ模式	PC,CPSR, SPSR_fiq, R14_fiq~R8_fiq,R7~R0
0b10010	IRQ模式	PC,CPSR, SPSR_irq, R14_irq, R13_irq,R12~R0
0b10011	管理模式	PC,CPSR, SPSR_svc, R14_svc, R13_svc, R12~R0
0b10111	中止模式	PC,CPSR, SPSR_abt, R14_abt, R13_abt,R12~R0
0b11011	未定义模式	PC,CPSR, SPSR_und, R14_und, R13_und, R12~R0
0b11111	系统模式	PC,CPSR(ARM v4及以上版本),R14~R0

Thumb状态下的寄存器组织

- Thumb状态下的寄存器集是ARM状态下寄存器集的一个子集
- 程序可以直接访问

8个通用寄存器(R7-R0)

程序计数器（PC）

堆栈指针（SP）

连接寄存器（LR）

CPSR

SPSR

第2章 ARM微处理器概述

2.3.3 ARM微处理器的工作状态

任务：了解ARM微处理器的两种工作状态：ARM和Thumb，了解ARM和Thumb的切换方式

工作状态

- 两种：
  - 一种为ARM状态，此时处理器执行32位的字对齐的ARM指令；
  - 另一种为Thumb状态，此时处理器执行16位的、半字对齐的Thumb指令。
- 当ARM微处理器执行32位的ARM指令集时，工作在ARM状态；
- 当ARM微处理器执行16位的Thumb指令集时，工作在Thumb状态。

工作状态转换方法(子程序调用)

- 在程序的执行过程中，处理器可以随时在两种工作状态之间切换。
- 处理器工作状态的转变并不影响处理器的工作模式和相应寄存器中的内容
- ARM微处理器在开始执行代码时总是处于ARM状态
- 也就是复位后进行ARM状态

工作状态转换方法(子程序调用)

- 进入Thumb状态：
- 当操作数寄存器的状态位（位0）为1时，可以采用执行BX指令的方法，使微处理器从ARM状态切换到Thumb状态。
- BX指令：带状态切换的跳转指令。
- BX R0；R0的最低位为1
- 此外，当处理器从Thumb状态进入异常（如FIQ、IRQ、Undef、Abort、SWI等）处理，在异常处理返回时，自动切换到Thumb状态。

工作状态转换方法(子程序调用)

- 进入ARM状态：
- 当操作数寄存器的状态位为0时，执行BX指令时可以使微处理器从Thumb状态切换到ARM状态。
- BX R0；R0的最低位为0
- 处理器工作在Thumb状态，如果发生异常并进入异常处理子程序，则进行时处理器自动从Thumb状态切换到ARM状态，
- 所有的异常处理都在ARM状态下执行。

范例：

从ARM状态切换到Thumb状态：

```
LDR  R0,=LABEL+1  ;因为LABEL对应的地址最低位（位0）为0，该地址加1后复制给R0
BX   R0      ; R0最低位（位0）是1
```

从Thumb状态切换到ARM状态：

```
LDR  R0,=LABEL    ; LABEL对应的地址最低位（位0）为0，该地址复制给R0
BX   R0           ; R0最低位（位0）是0
```

第2章 ARM微处理器概述

2.3.6 I/O端口的访问方式

任务：了解ARM体系中I/O端口的访问方式。

- 对于I/O端口的访问，ARM系统采用的标准方法是存储器映射方式（也叫统一编址方式）。该方式为每个I/O端口分配特定的存储器地址，当从这些地址读出或写入时，实际完成的是I/O功能，即对存储器映射的I/O地址进行读取操作时即是输入，而向存储器映射的I/O地址进行写入操作时即是输出。

2.3.7异常(Exceptions)

任务：掌握异常的定义，掌握各种异常类型的含义，掌握异常向量的定义，掌握当产生异常发生时ARM微处理器对异常中断的响应过程。

异常

- 当正常的程序执行流程发生暂时的停止时，称之为异常（Exception）
- 例如处理一个外部的中断请求
- 在处理异常之前，当前处理器的状态必须保留，这样当异常处理完成之后，当前程序可以继续执行
- 处理器允许多个异常同时发生，它们将会按固定的优先级进行处理
- 中断优先级
- 中断嵌套

1. 异常类型

表2-3 ARM体系结构支持异常类型

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，或该地址不允许当前指令访问，存储器会向处理器发出中止信号，但当预取的指令被执行时，才会产生指令预取中止异常。
数据中止	若处理器数据访问指令的地址不存在，或该地址不允许当前指令访问，产生数据中止异常。
IRQ（外部中断请求）	当处理器的外部中断请求引脚有效，且CPSR中的I位为0时，产生IRQ异常。系统的外设可通过该异常请求中断服务。
FIQ（快速中断请求）	当处理器的快速中断请求引脚有效，且CPSR中的F位为0时，产生FIQ异常

2．异常向量

表2-4 异常向量

异常类型	模式	正常向量	高向量地址
复位	管理	0x00000000	0xFFFF0000
未定义指令	未定义	0x00000004	0xFFFF0004
软件中断	管理	0x00000008	0xFFFF0008
指令预取中止	中止	0x0000000C	0xFFFF000C
数据中止	中止	0x00000010	0xFFFF0010
IRQ（外部中断请求）	IRQ	0x00000018	0xFFFF0018
FIQ（快速中断请求）	FIQ	0x0000001C	0xFFFF001C

当出现异常后，ARM处理器会执行以下操作：

- 将CPSR复制到相应的SPSR中
- 对CPSR进行设置

  根据异常类型，强制设置CPSR的工作模式位

  设置中断禁止位，以禁止中断发生

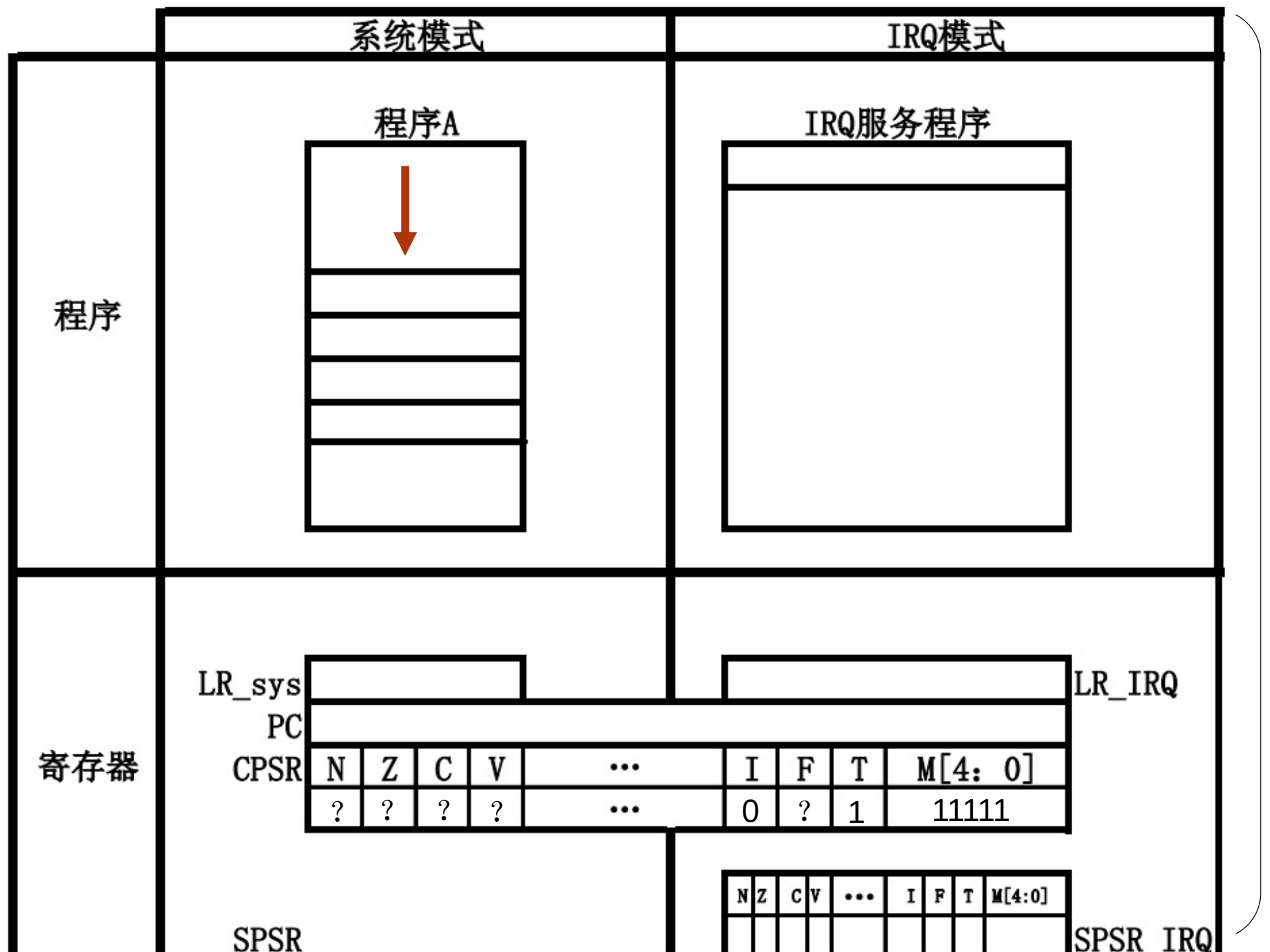
  如果处理器处于Thumb状态，则切换到ARM状态
- 将下一条指令的地址存入相应链接寄存LR

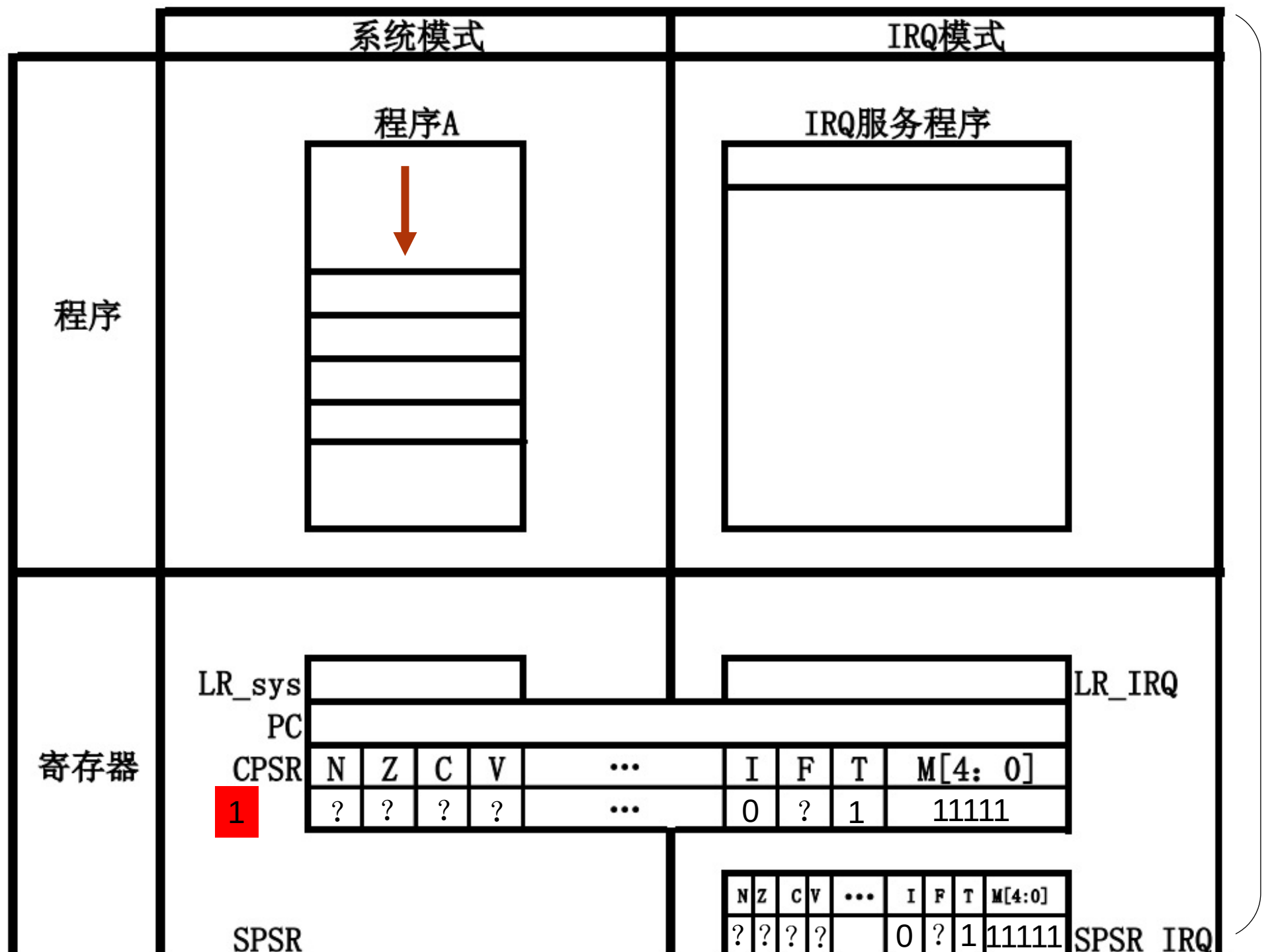
  LR中保存的是下一条指令的地址（当前执行指令地址+4或+8，与异常类型有关）
- 强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

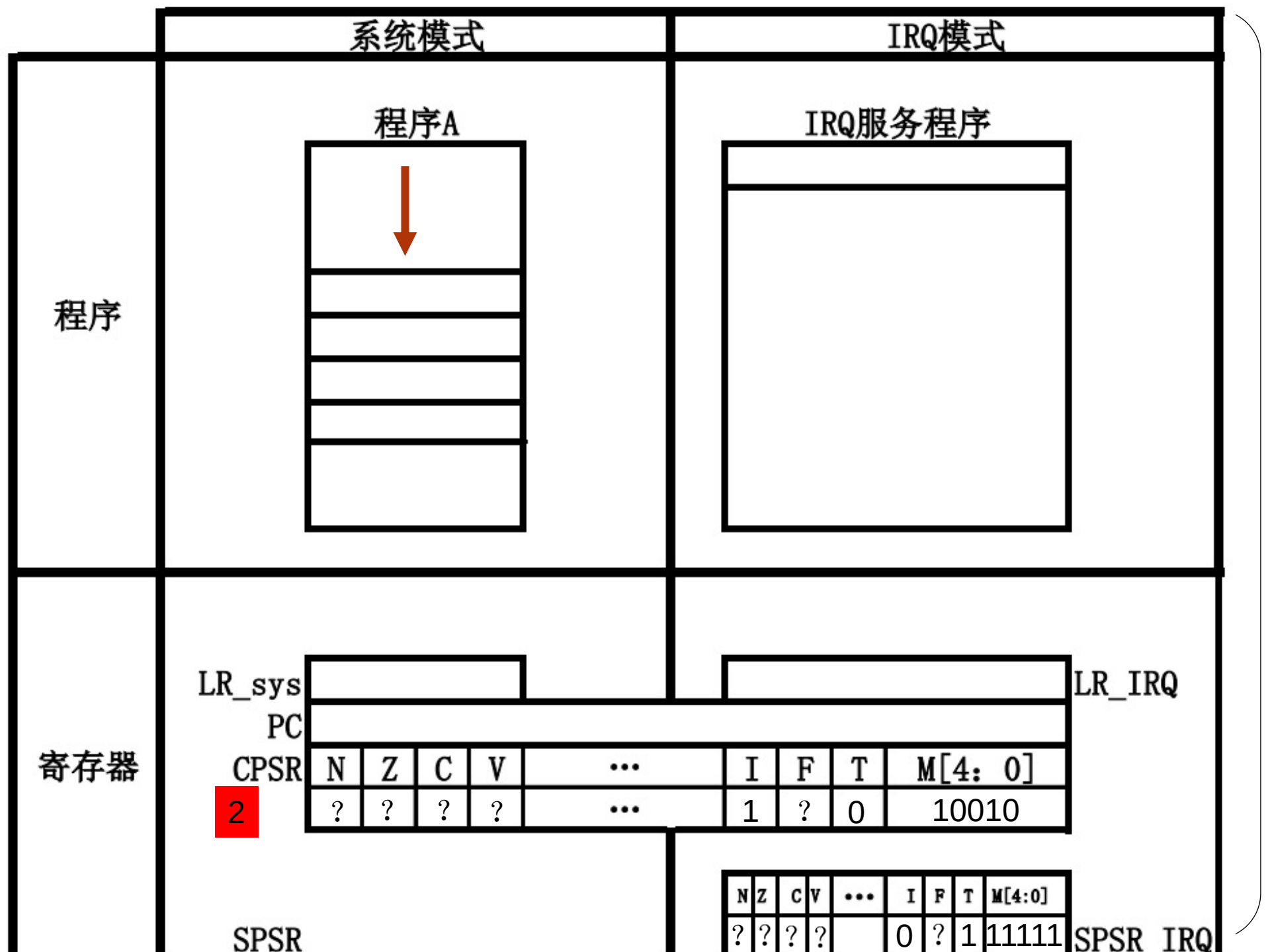
ARM处理器对异常的响应过程伪码描述（CPU自动完成）

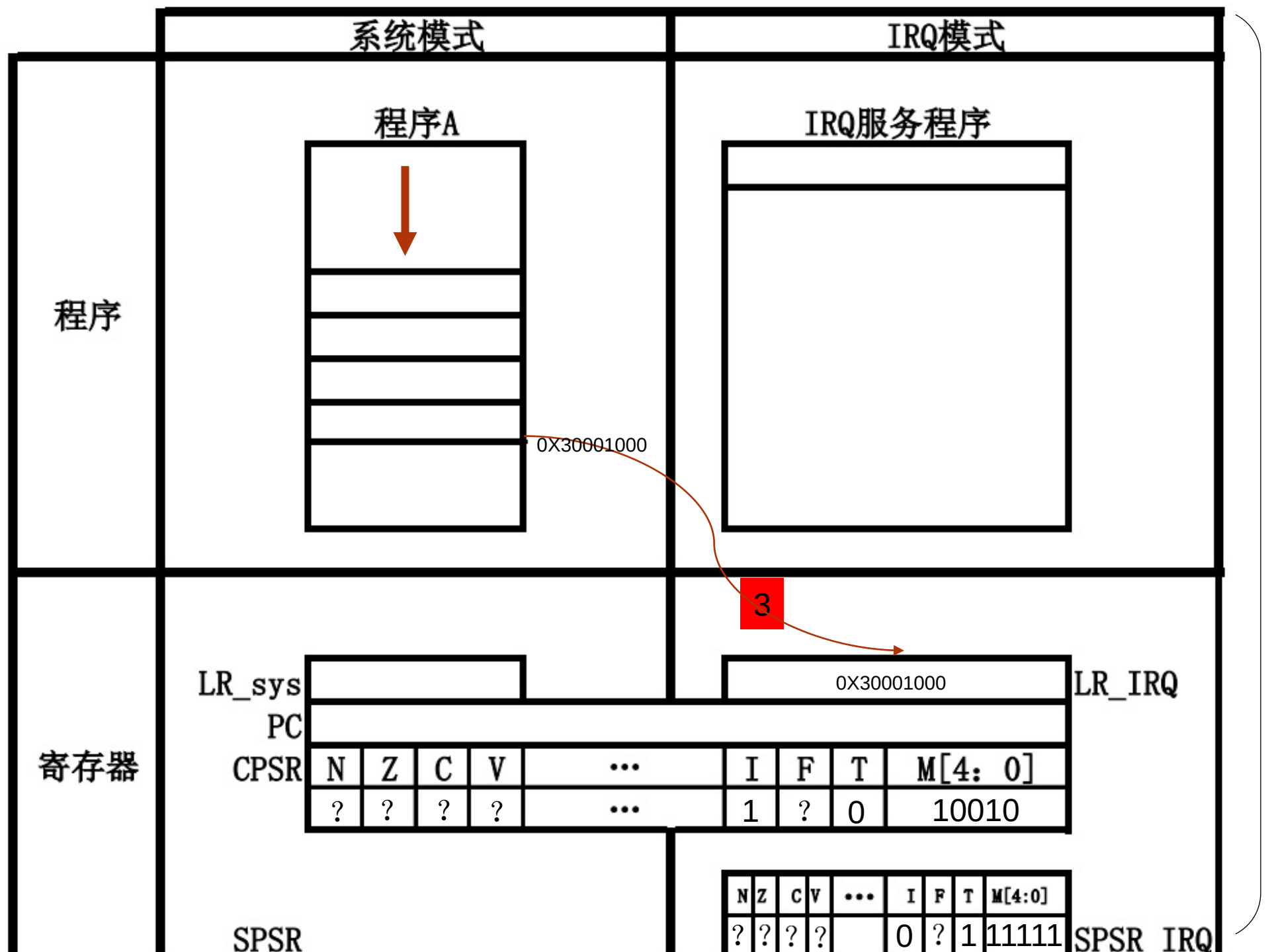
- 上述的处理器对异常中断的响应过程可以用如下的伪代码描述：
- SPSR_<exception_mode> = CPSR
- CPSR[4：0] = exception mode number
- CPSR[5] = 0 /*当运行于AMR状态时*/
- If <exception_mode> = = reset or FIQ then/*当相应FIQ异常中断时，禁止新的FIQ中断*/
- CPSR[6] = 1 /*禁止新的FIQ中断*/
- CPSR[7] = 1 /*禁止IRQ中断*/
- R14_<exception_mode> = return link
- PC = exception vector address

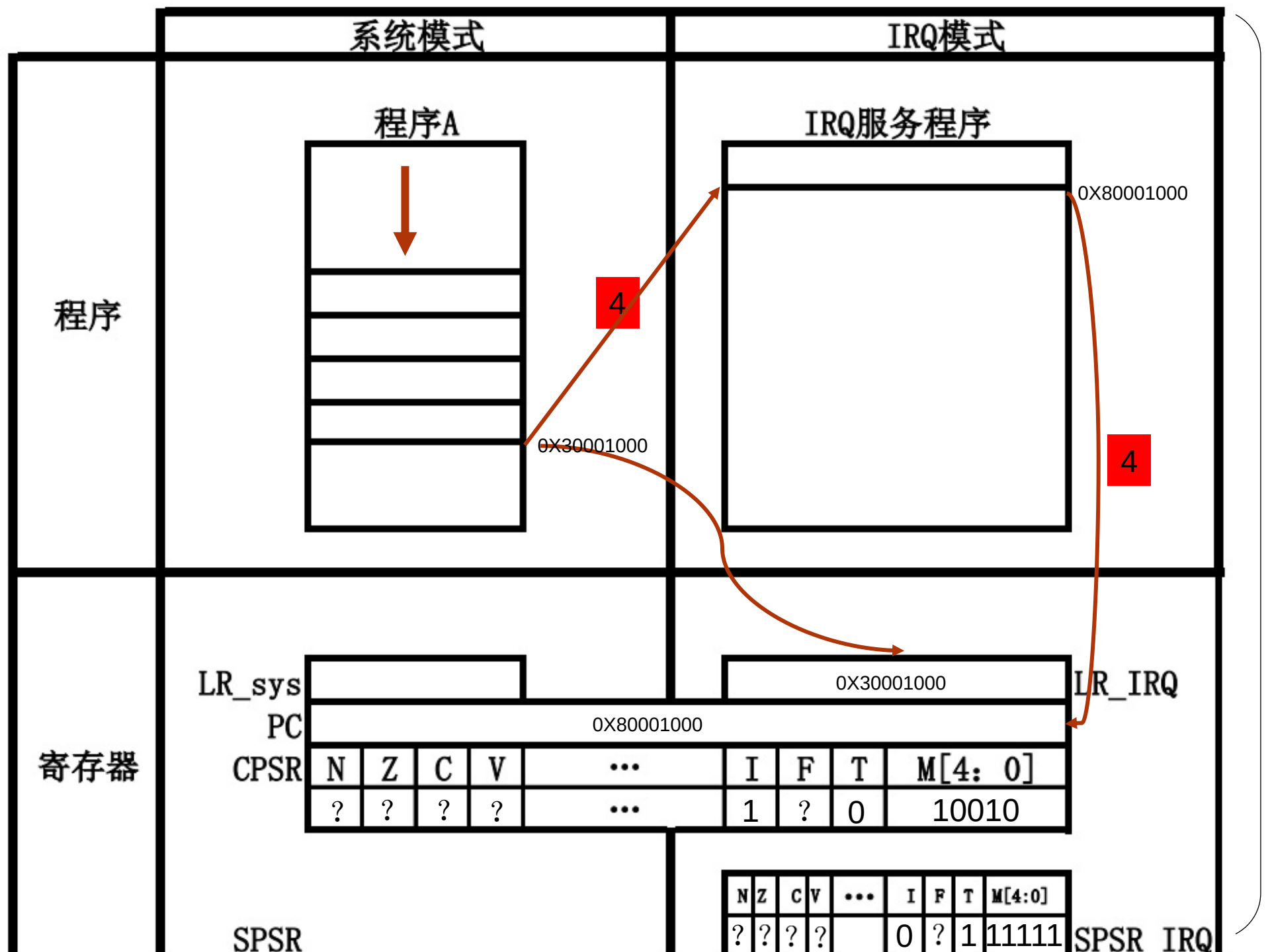
- 例：程序在系统模式下运行用户程序，假定当前处理器状态为Thumb状态，允许IRQ中断。


系统模式
IRQ模式
程序
程序A
IRQ服务程序
寄存器
LR_sys
LR_IRQ
PC
CPSR
N Z C V ••• I F T M[4: 0]
? ? ? ? ••• 0 ? 1 11111
SPSR
N Z C V ••• I F T M[4:0]
SPSR_IRQ


系统模式
IRQ模式
程序
程序A
IRQ服务程序
寄存器
LR_sys
LR_IRQ
PC
CPSR
N Z C V ••• I F T M[4: 0]
1
? ? ? ? ••• 0 ? 1 11111
N Z C V ••• I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR
SPSR_IRQ

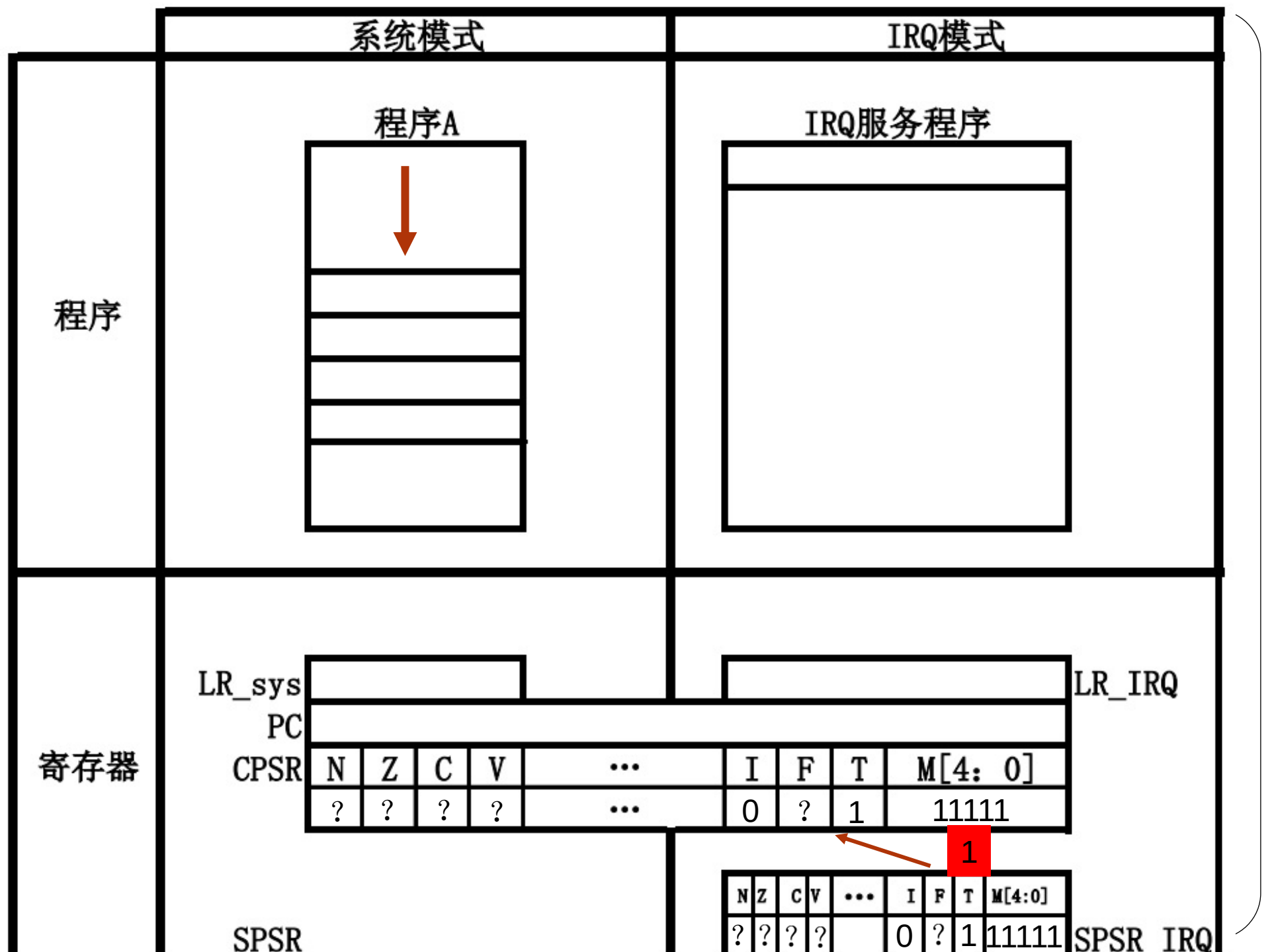

系统模式
IRQ模式
程序
程序A
IRQ服务程序
寄存器
LR_sys
LR_IRQ
PC
CPSR
N Z C V ••• I F T M[4: 0]
? ? ? ? ••• 1 ? 0 10010
2
SPSR
N Z C V ••• I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR_IRQ

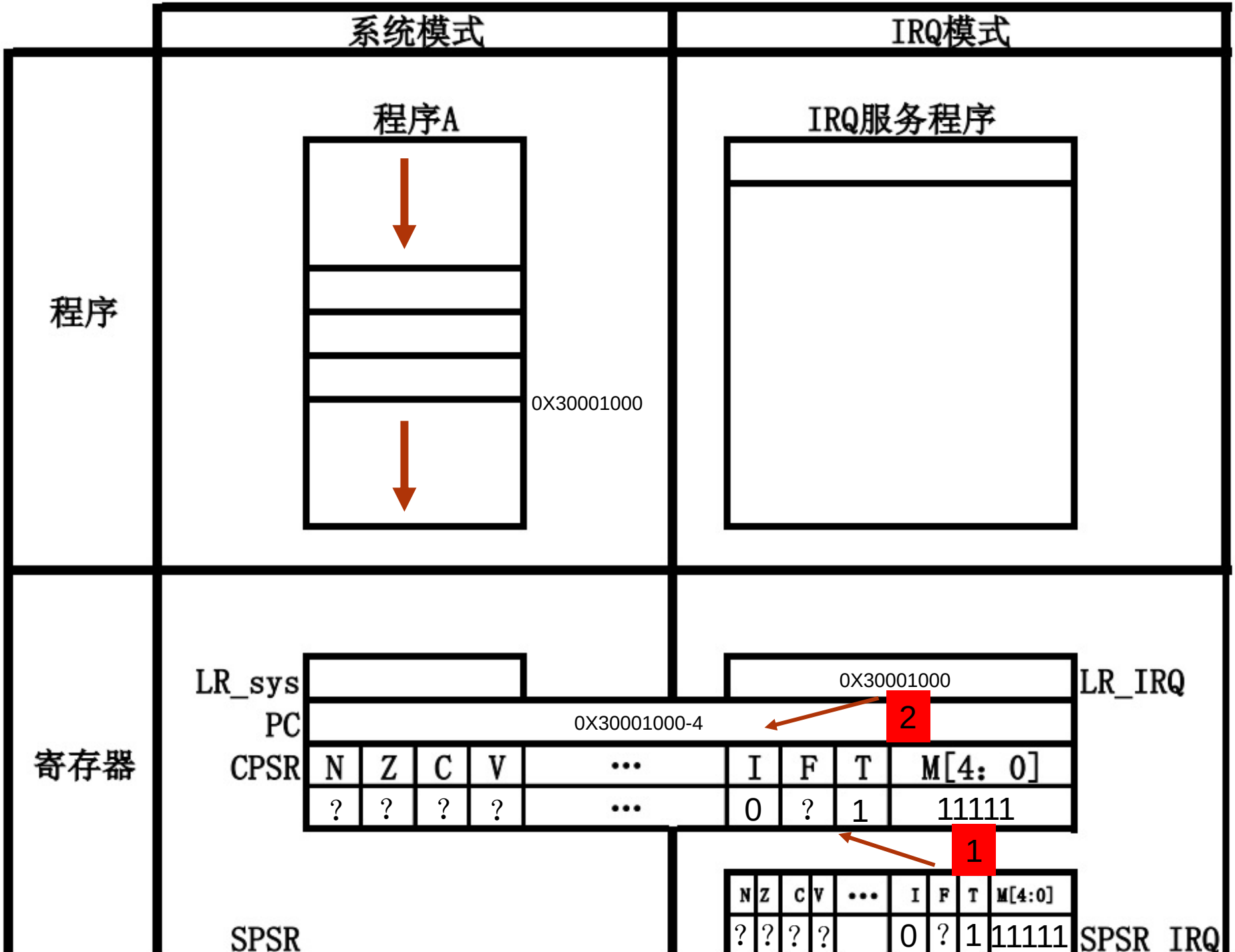

系统模式
IRQ模式
程序
程序A
IRQ服务程序
0X30001000
3
寄存器
LR_sys
0X30001000
LR_IRQ
PC
CPSR
N Z C V ... I F T M[4: 0]
? ? ? ? ... 1 ? 0 10010
SPSR
N Z C V ... I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR_IRQ


系统模式
IRQ模式
程序
寄存器
程序A
IRQ服务程序
0X80001000
4
0X30001000
4
LR_sys
0X30001000
LR_IRQ
PC
0X80001000
CPSR
N Z C V ••• I F T M[4: 0]
? ? ? ? ••• 1 ? 0 10010
N Z C V ••• I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR
SPSR_IRQ

4．从异常返回（编程完成）

- （1）将SPSR复制回CPSR中。
- （1）将连接寄存器LR的值减去相应的偏移量后送到PC中。
- （3）若在进入异常处理时设置了中断禁止位，要在此清除。


系统模式
IRQ模式
程序
程序A
IRQ服务程序
寄存器
LR_sys
LR_IRQ
PC
CPSR
N Z C V ••• I F T M[4: 0]
? ? ? ? ••• 0 ? 1 11111
1
N Z C V ••• I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR
SPSR_IRQ


系统模式
IRQ模式
程序
程序A
IRQ服务程序
0X30001000
寄存器
LR_sys
0X30001000
LR_IRQ
PC
0X30001000-4
2
CPSR
N Z C V ••• I F T M[4: 0]
? ? ? ? ••• 0 ? 1 11111
1
N Z C V ••• I F T M[4:0]
? ? ? ? 0 ? 1 11111
SPSR
SPSR_IRQ

复位的响应过程

- SPSR_svc = CPSR（不可预知）
- CPSR[4:0] = 0b10011
- CPSR[5] = 0　切换到ARM工作状态
- CPSR[6] = 1　禁止新的FIQ异常
- CPSR[7] = 1　禁止新的IRQ异常
- R14_svc = Return Link（不可预知）
- PC = 0x00000000

(Undefined Instruction)
未定义指令异常

- 当ARM处理器遇到不能处理的指令时，会产生未定义指令异常

采用这种机制，可以通过软件仿真扩展ARM或Thumb指令集

未定义指令异常的响应过程

- SPAR_und = CPSR
- CPSR[4:0] = 0B11011
- CPSR[5] = 0 ； 切换到ARM工作状态

CPSR[6] 保持不变

- CPSR[7] = 1 ； 禁止新的IRQ异常
- R14-und = Return Link
- PC = 0x00000004

未定义指令异常返回

- 在未定义指令处理程序中执行以下指令返回

MOVS PC,R14_und

恢复PC（从R14_und)和CPSR(从SPSR_und)的值，并返回到未定义指令后的下一条指令

指令加后缀“s”且目的寄存器为PC则自动复制

SoftwareInterrupt（软件中断）

- 软件中断指令（SWI）用于进入管理模式，常用于请求执行特定的管理功能

在ARM上写的操作系统典型的使用SWI来为编程者提供各种例程。

SWI的响应过程

- SPSRP_svc = CPSR
- CPSR[4:0] = 0b10011
- CPSR[5] = 0 ； 切换到ARM工作状态
- #CPSR[6]保持不变
- CPSR[7] = 1 ；禁止新的IRQ中断
- R14_svc = Return Link
- PC = 0x00000008

SoftwareInterrupt（软件中断）

- 在软件中断处理程序中执行以下指令返回

  MOVS PC,R14_svc

  恢复PC（从R14_svc）和CPSR（从SPSR_svc）的值，并返回到SWI的下一条指令
- 指令加后缀“S”且目的寄存器为PC时自动复制

中止响应

- ABORT（中止）
- 产生中止异常意味着对存储器的访问失败
- 中止异常包括两种类型：

指令预取中止，发生在指令预取时

数据中止：发生在数据访问时

中止响应过程

- SPSR_abt = CPSR
- CPSR[4:0] = 0b10111
- CPSR[5] = 0 ； 切换到ARM工作状态
- #CPSR[6]保持不变
- CPSR[7] = 1 ；禁止新的IRQ中断
- R14_abt = Return Link
- PC = 0x0000000C 指令中止/PC = 0x00000010 数据中止

中止响应返回

- 当确定中止原因后，Abort处理程序执行以下指令返回

SUBS PC,R14_abt,#4 ； 指令预取中止

SUBS PC,R14_abt,#8 ； 数据中止

恢复PC（从R14_abt）和CPSR（从SPSR_abt)的值，并重新执行产生中止的指令（返回当前指令）

IRQ（Interrupt Request)

- IRQ异常属于正常的中断请求，IRQ优先级低于FIQ

  若CPSR的I位置1，则禁止IRQ中断，若CPSR的I位清零，处理器会在指令执行完之前检查IRQ的输入

  注意只有在特权模式下才能改变I位的状态

IRQ的响应过程

- SPSR_irq = CPSR
- CPSR[4:0] = 0b10010
- CPSR[5] = 0 ； 切换到ARM工作状态
- #CPSR[6]保持不变
- CPSR[7] = 1 ；禁止新的IRQ中断
- R14_abt = Return Link
- PC = 0x00000018

IRQ（Interrupt Request)异常返回

- IRQ处理程序执行以下指令返回

SUBS PC,R14_IRQ.#4

该指令将寄存器R14_irq的值送去4后，复制到程序计算器PC中，同时将SPSR_irq寄存器的内容复制 到CPSR中，并返回到引起中断的下一条指令

FIQ（Fast Interrupt Request)

- FIQ异常是为了支持数据传输或者通道处理而设计的

  若将CPSR的F位置1，则禁止FIQ中断，若CPSR的F位清零，处理器会在指令执行完之前检查FIQ的输入

  注意只有在特权模式下才能改变F位的状态

FIQ的响应过程

- SPSR_fiq = CPSR
- CPSR[4:0] = 0b10001
- CPSR[5] = 0 ； 切换到ARM工作状态
- CPSR[6] = 1 ；禁止新的FIQ中断
- CPSR[7] = 1 ；禁止新的IRQ中断
- R14_abt = Return Link
- PC = 0x0000001C

FIQ（Interrupt Request)异常返回

- FIQ处理程序执行以下指令返回

SUBS PC,R14_ fiq.#4

该指令将寄存器R14_irq的值送去4后，复制到程序计算器PC中，同时将SPSR_fiq寄存器的内容复制 到CPSR中，并返回到引起中断的下一条指令

	返回指令	**R14_x先前的值**
BL(子程序调用)	MOV PC,R14	下一条指令地址
未定义指令	MOVS PC,R14_und	下一条指令地址
SWI	MOVS PC,R14_svc	下一条指令地址
指令预取	SUBS PC,R14_abt,#4	下一条指令地址
数据预取	SUBS PC,R14_abt,#8	下一条指令地址+4
IRQ	SUBS PC,R14_irq,#4	下一条指令地址+4
FIQ	SUBS PC,R14_flq,#4	下一条指令地址+4
复位	N/A	-

异常向量

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

ARM的CPU中断向量表

向量表

- Exception Vectors
- Mapped to Address 0
- Absolute addressing mode must be used
- Vectors LDR PC,Reset_Address
- LDR PC,Undef_Address
- LDR PC,SWI_Address
- LDR PC,PAbt_Address
- LDR PC,DAbt_Address
- NOP ;Reserved Vector
- LDR PC,IRQ_Address
- LDR PC,FIQ_Address

异常优先级（Exception Priorities)

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