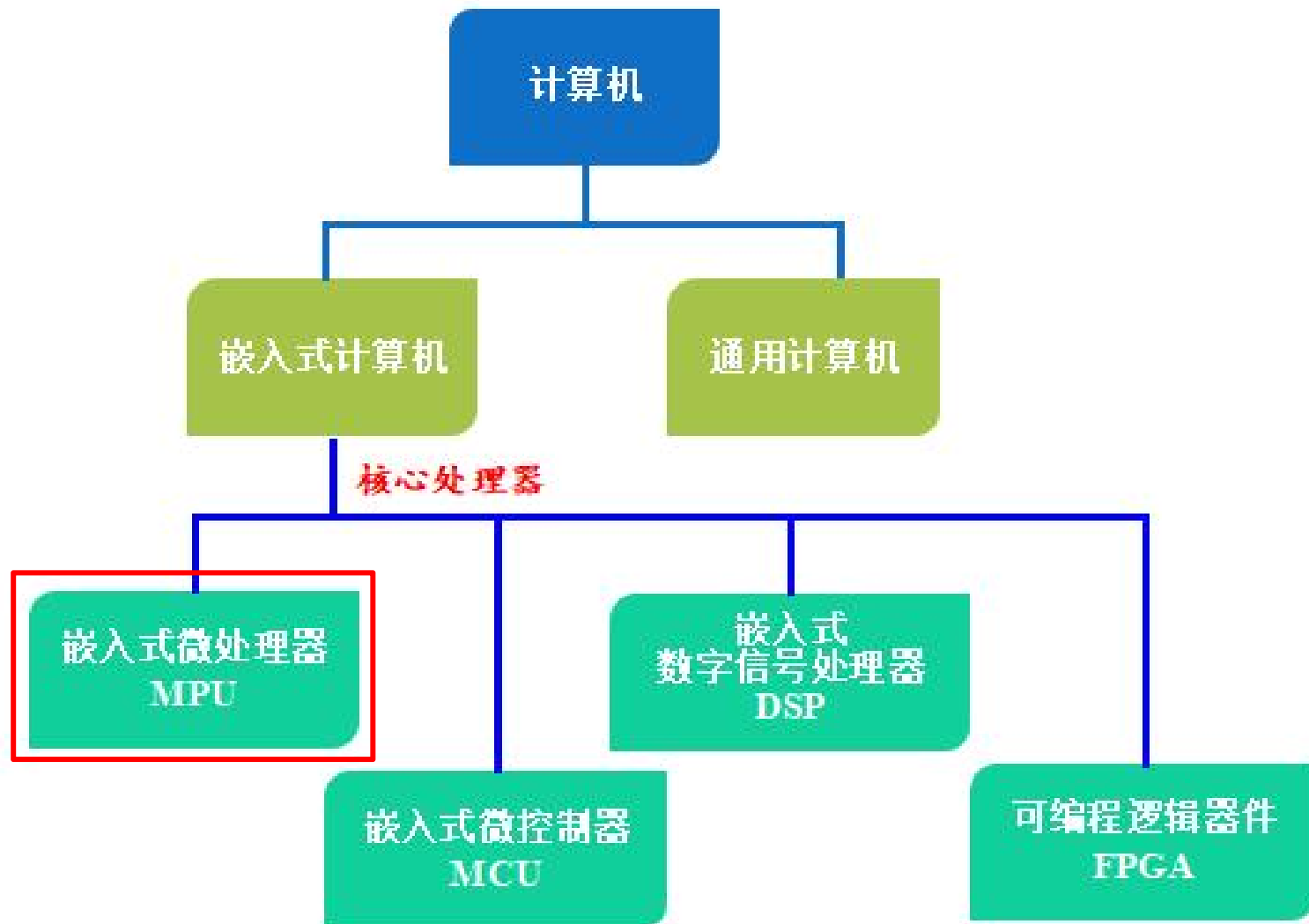




第二章 ARM 体系结构



微处理器的体系结构

- 微处理器的五种体系结构：

- ARM

The ARM logo, consisting of the lowercase letters "arm" in a blue, sans-serif font.

- MIPS

The MIPS Technologies logo, featuring the word "MIPS" in a large, blue, sans-serif font, with "TECHNOLOGIES" in a smaller, blue, sans-serif font below it.

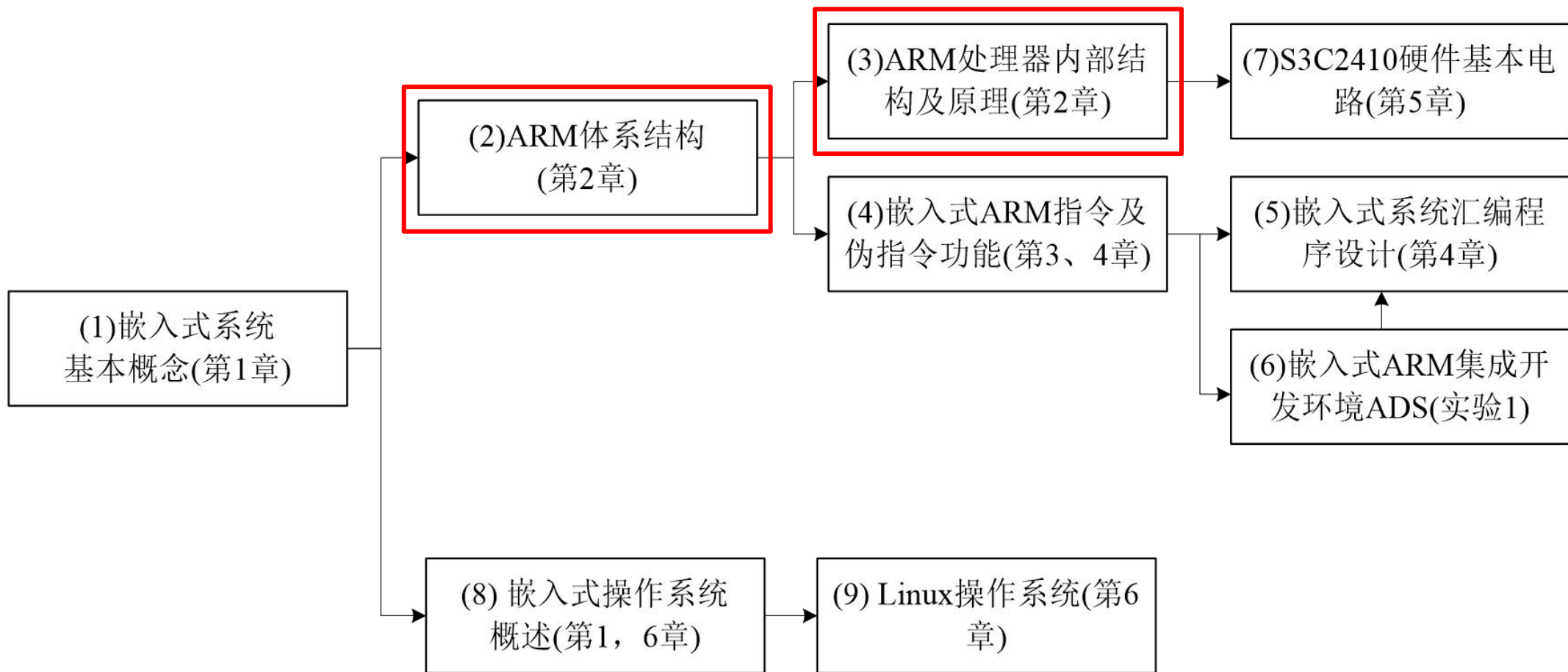
- POWER PC



MOTOROLA
摩托罗拉

- X86

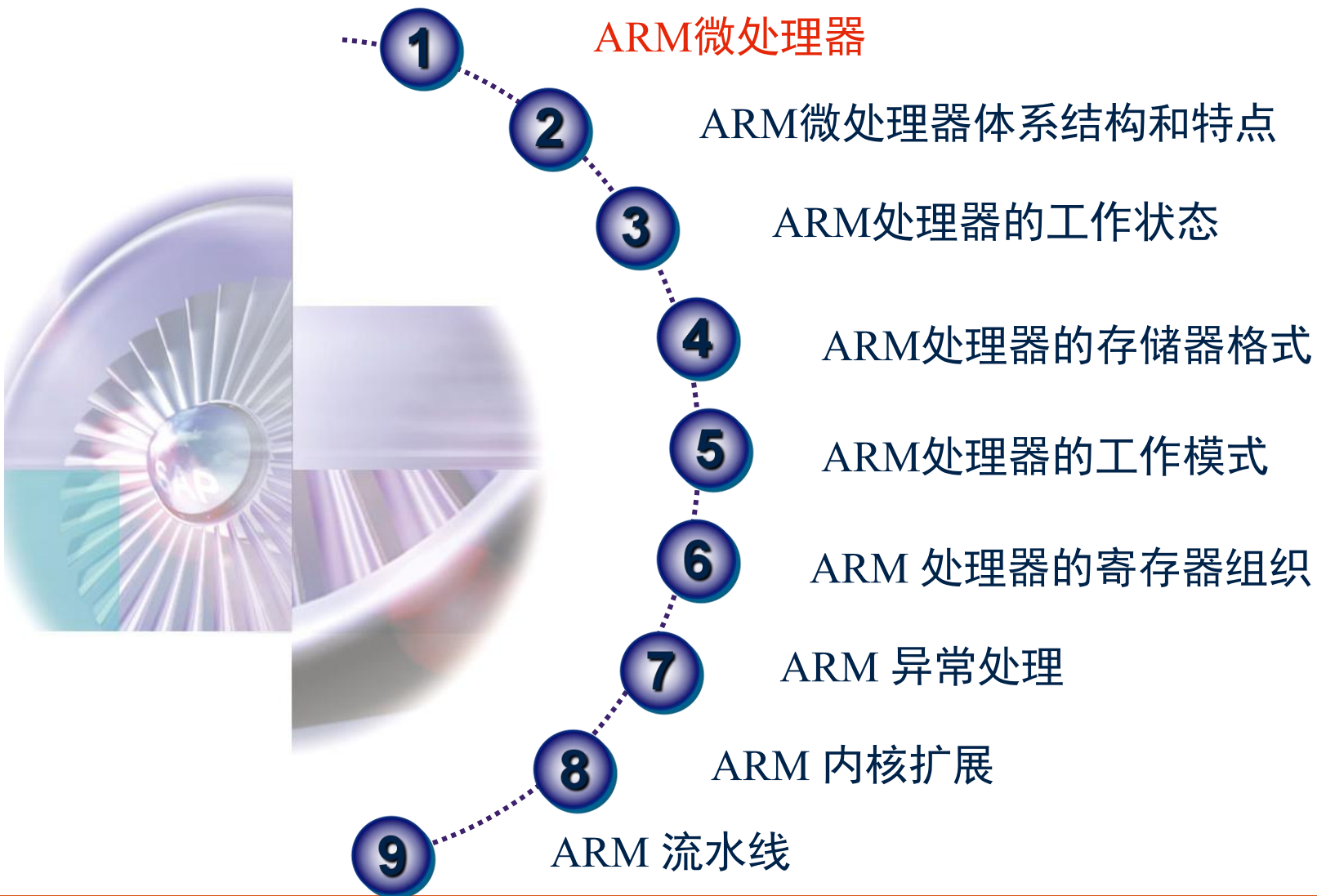
- SH系列



课堂理论部分

- ✓ ARM处理器的体系结构
- ✓ ARM处理器的工作状态
- ✓ ARM处理器的存储器格式
- ✓ ARM处理器的工作模式
- ✓ ARM处理器的异常处理

编程模型



- 1978, Cambridge Processing Unit: CPU
 - Christopher Curry和Herman Hauser
- 1979, Acorn Computers Ltd.
- 1985, RISC Computer, 32 bit, 6M处理器
 - Acorn RISC Machine(ARM)
 - Advanced RISC Machine(ARM)
- 1990, ARM Computers Ltd.
 - 苹果: 150万英镑
 - VLSI: 25万英镑
 - Acorn: 150万英镑(IP) + 12 engineers



- 成立于1990年11月
 - 前身为 Acorn计算机公司
 - Advance RISC Machine(ARM)
- 主要设计ARM系列RISC处理器内核
- 授权ARM内核给生产和销售半导体的合作伙伴
 - ARM 公司不生产芯片
 - IP(Intelligence Property)
- 另外也提供基于ARM架构的开发设计技术
 - 软件工具, 评估板, 调试工具, 应用软件,
 - 总线架构, 外围设备单元, 等等



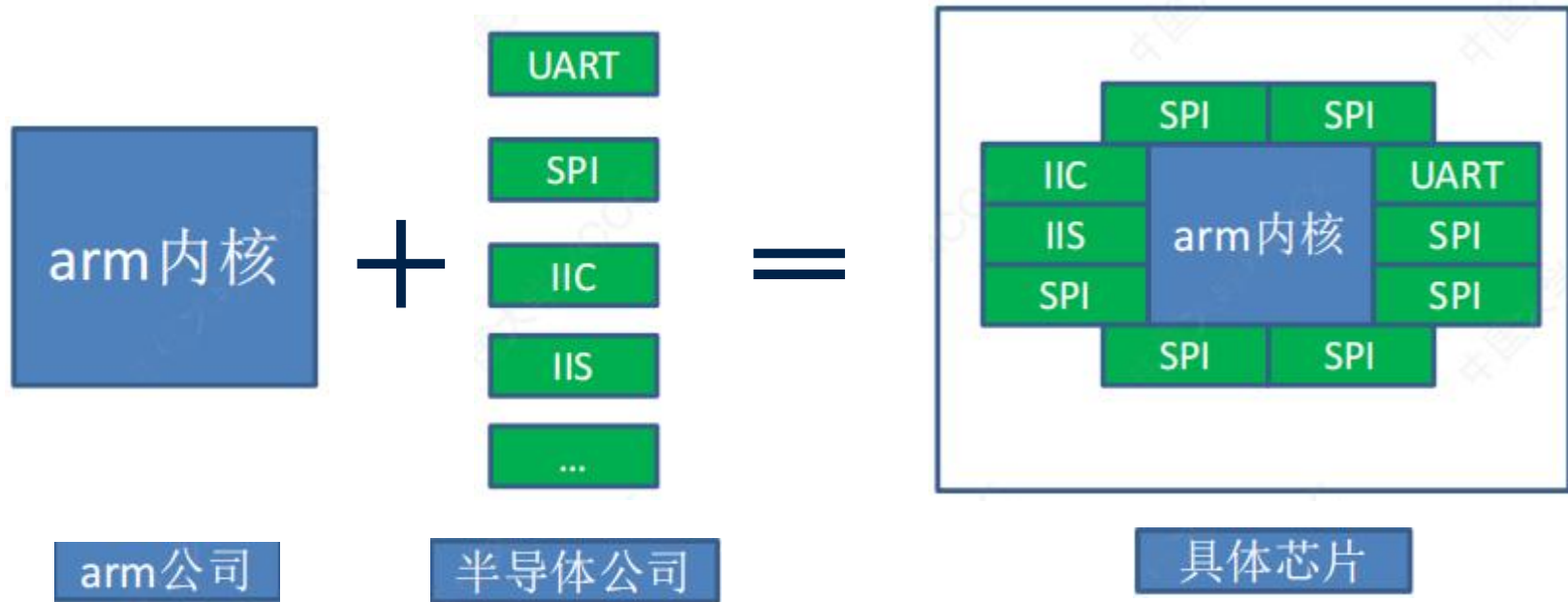


- 2016年7月被日本软银集团以310亿美元收购

ARM® → arm

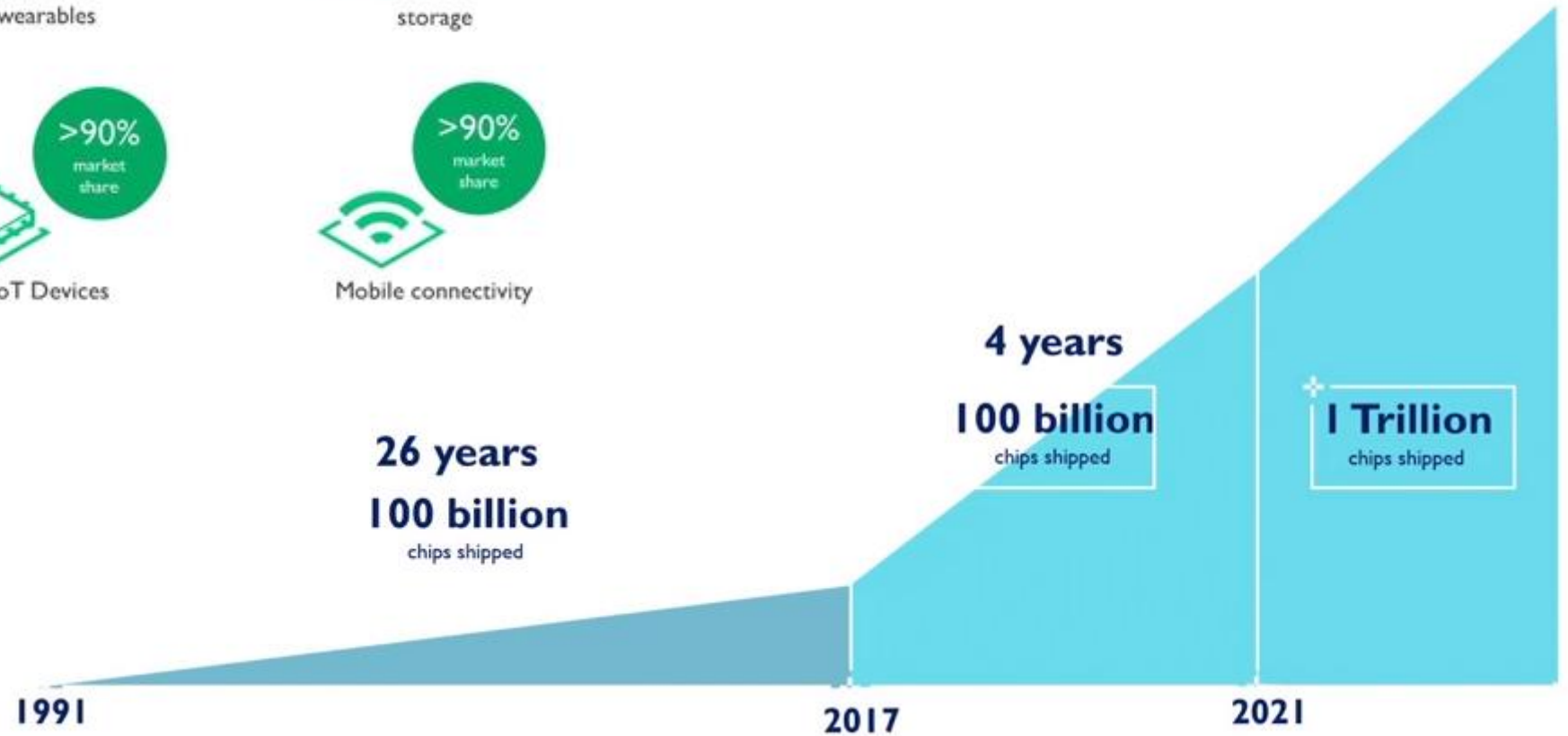
①ARM微处理器内核+②适当的外围电路=③ARM微处理器SOC芯片

- ARM公司设计①
- 由世界各大半导体生产商根据各自不同的应用领域加②
- 生成基于ARM内核的微处理器SOC芯片③



目前，全世界有几十家著名的半导体公司都使用ARM公司的授权





100+ billion

基于arm技术的芯片累计出货量

17.7 billion

2016年基于安谋技术的芯片出货量

1 trillion

软银收购arm:全球最大的物联网领域收购



80% +

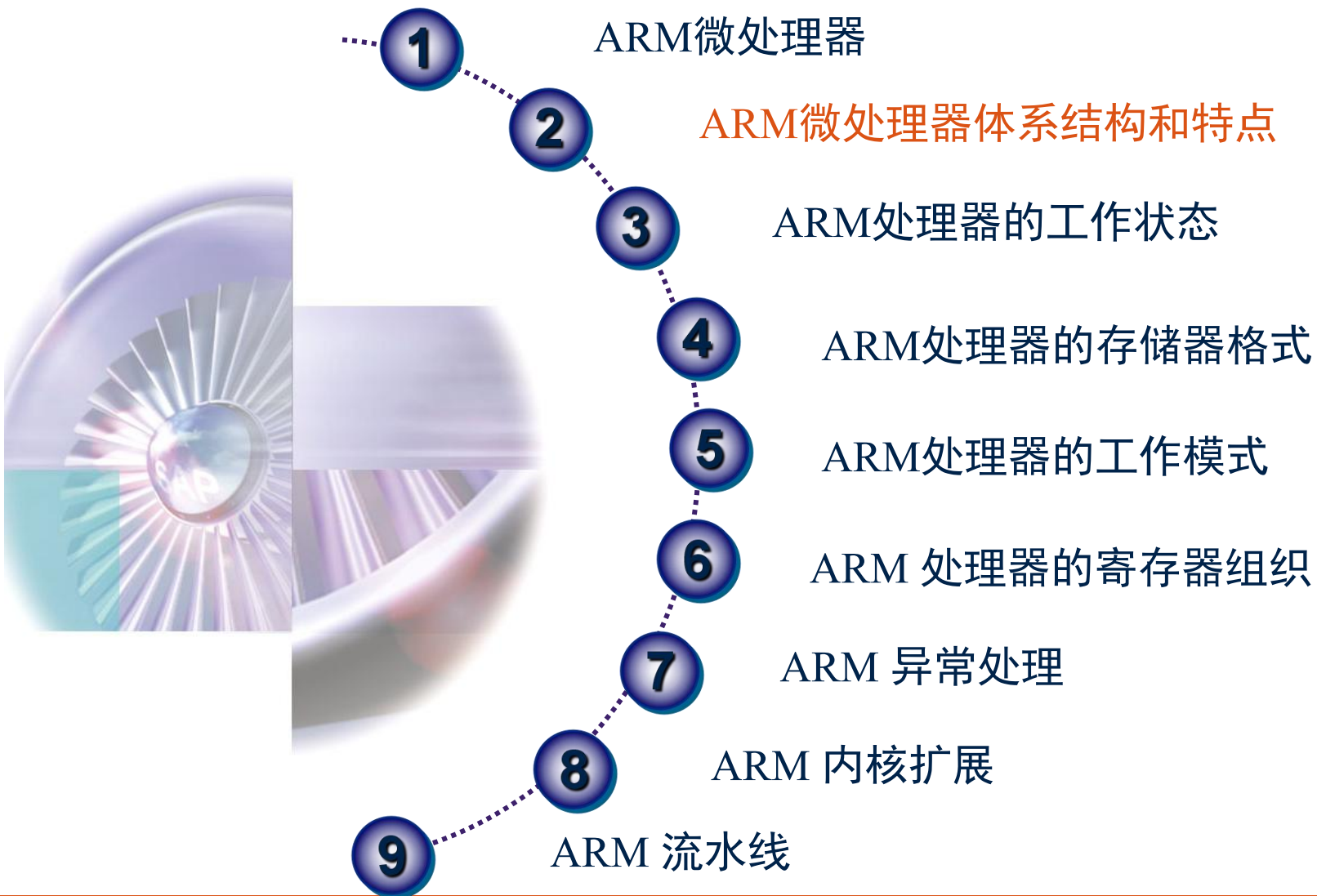
全球人口使用过基于arm处理器技术的产品

85% +

移动处理器使用arm技术

4,800+

全球雇员



■ **体系结构**就是指处理器的硬件和软件资源及其二者间的连接关系。

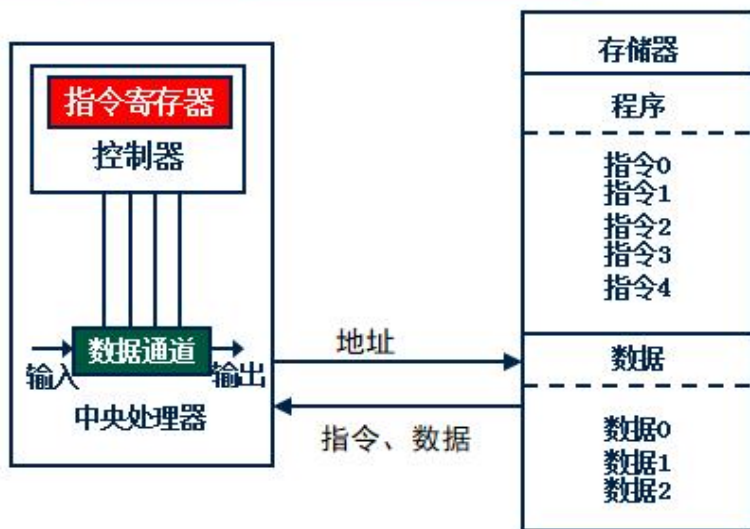
即**硬件结构**与**指令集**的组合。

■ 体系结构中的两个重要概念：

存储结构、指令集

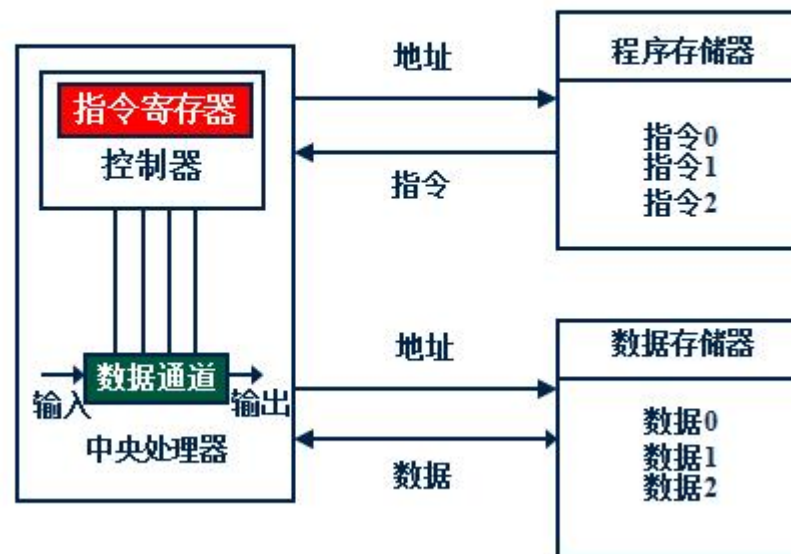
- 存储结构：2种结构，冯·诺依曼体系结构和哈佛体系结构
- 指令集：2种指令集，RISC和CISC

早期冯·诺依曼体系结构模型



- 指令和数据共一个存储空间；
- 经由同一条总线传输；
- 某些指令只能顺序执行。

哈佛体系结构



- 指令和数据独立存储；
- 经由互不相关的独立总线传输；
- 大多数指令都可以并行，提高运算速度。

例如：执行3条对存储器进行读写的指令



图 冯·诺曼结构处理器指令流的定时关系示意图



图 哈佛结构处理器指令流的定时关系示意图

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

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

例如: 计算 $a*=b$

CISC架构:

MUL ADDRA, ADDR B

RISC架构:

MOV A, ADDRA;

MOV B, ADDR B;

MUL A, B;

STR ADDRA, A。

类别	CISC	RISC
指令系统	指令数量很多	较少，通常少于100
执行时间	有些指令执行时间很长，如整块的存储器内容拷贝；或将多个寄存器的内容拷贝到存储器	没有较长执行时间的指令
编码长度	编码长度可变，1-15字节	编码长度固定，通常为4个字节
寻址方式	寻址方式多样	简单寻址
操作	可以对存储器和寄存器进行算术和逻辑操作	只能对寄存器执行算术和逻辑操作，Load/Store体系结构
编译	难以用优化编译器生成高效的目标代码程序	采用优化编译技术，生成高效的目标代码程序

--更适合通用机

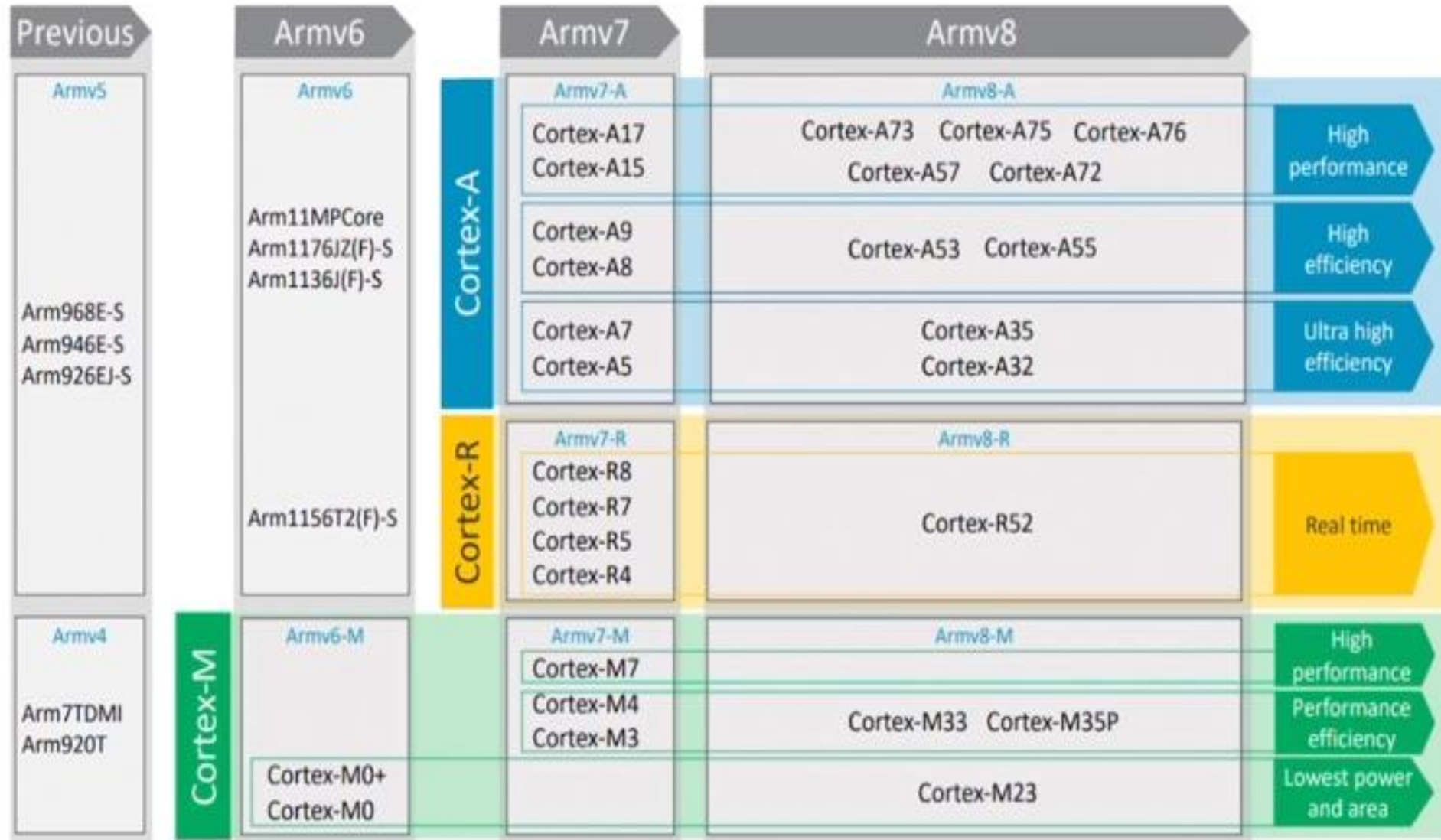
-----更适合专用机

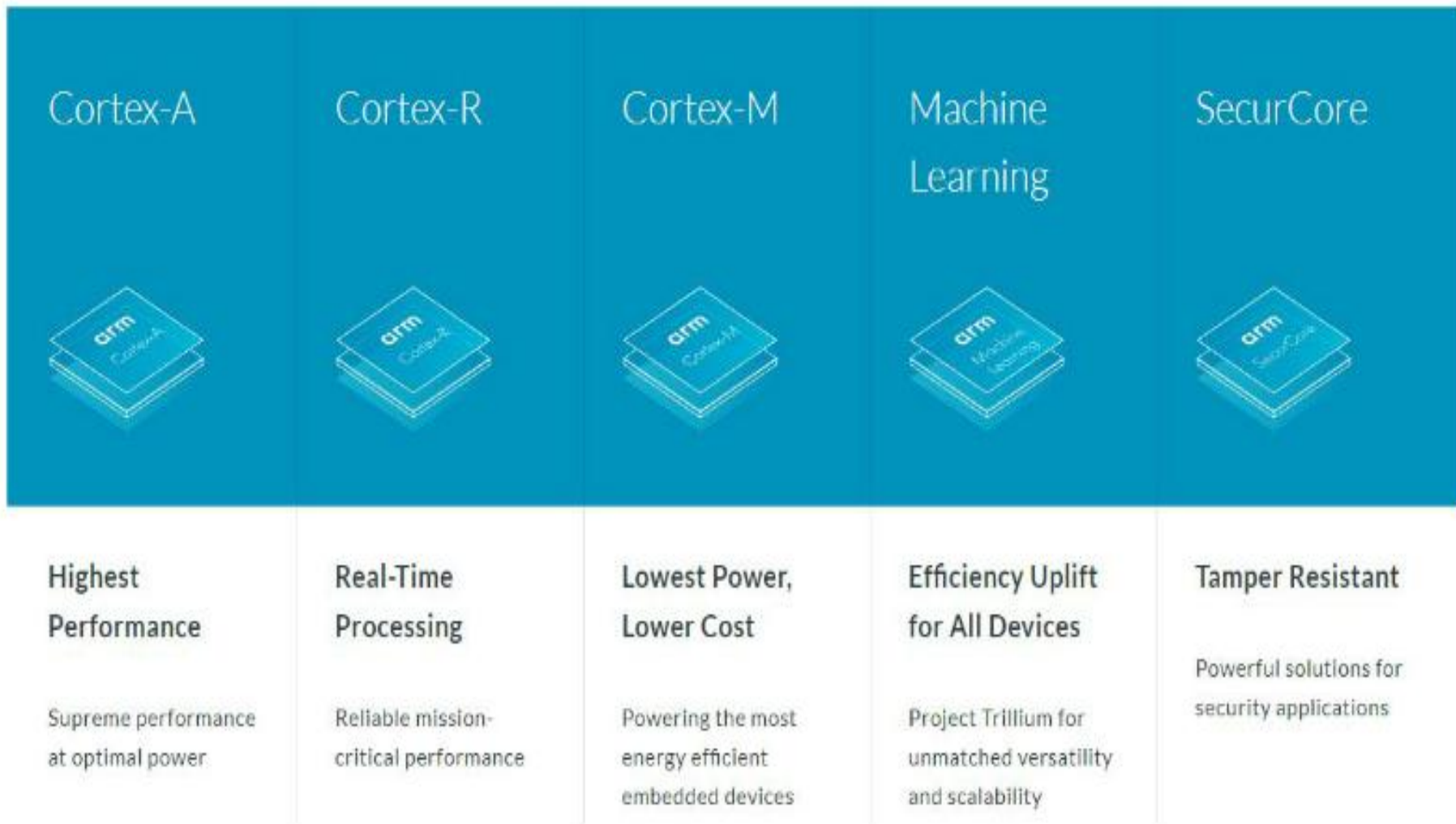
ARM微处理器采用RISC指令集，其特点：

- 低功耗、低成本、高性能；
- 大量使用寄存器，指令执行速度更快；
- 支持ARM/THUMB双指令集；
- 三/五级流水线；
- 绝大部分的操作就在寄存器中进行，通过LOAD/STORE的体系结构在内存和寄存器之间传递数据；
- 寻址方式简单；
- 指令长度固定。

- ARM体系结构共定义了9个版本，版本号分别为V1-V9。带来结构和功能的改善和提高。

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

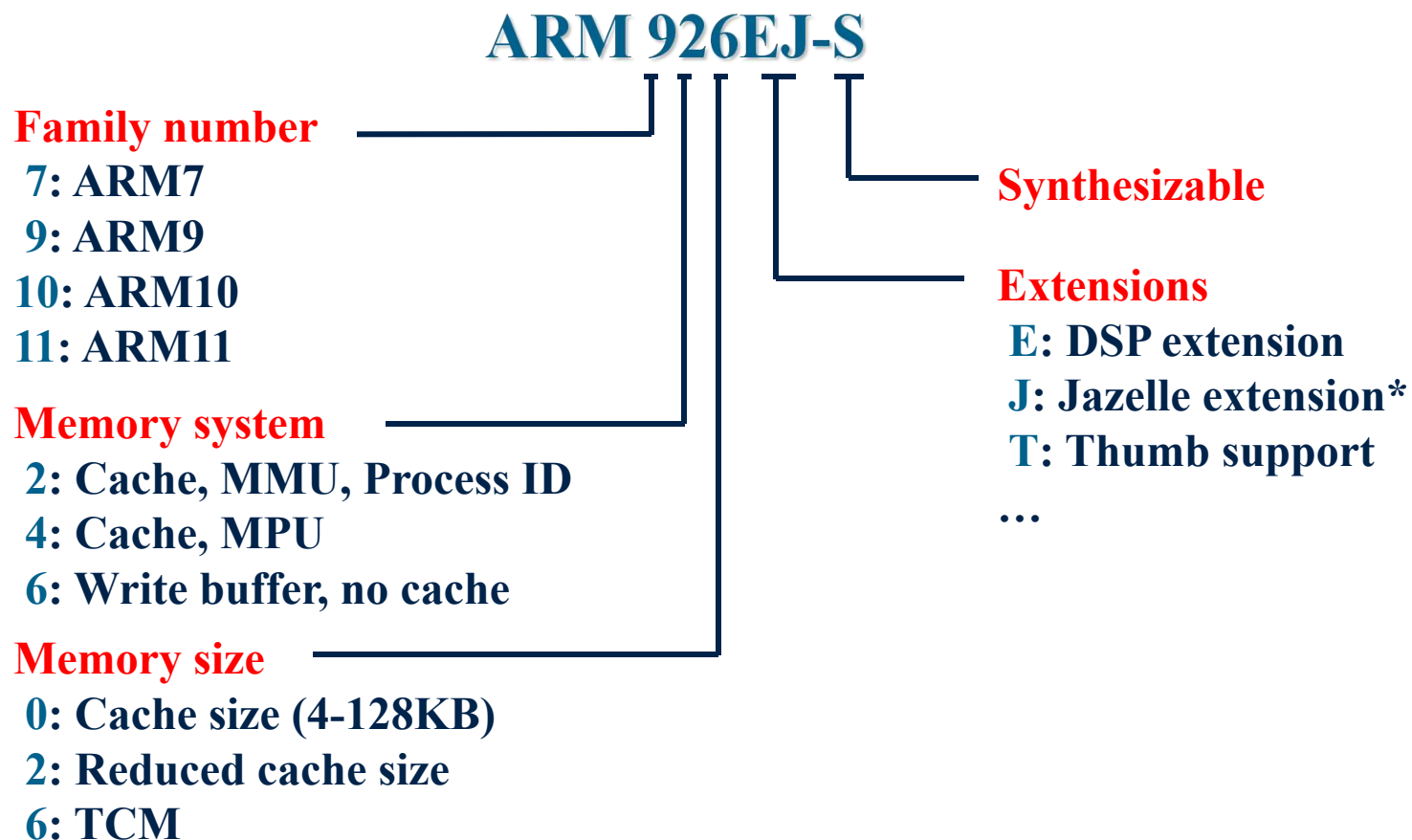




- ❑ **Version 1 – Acorn RISC, 26-bit address, never used in product**
- ❑ **Version 2 – multiplication and co-processor, atomic load/store**
- ❑ **Version 3 – 32-bit address, CPSR, and SPSR**
 - ❖ T – Thumb state: 16-bit instruction execution.
 - ❖ M – long multiply support ($32 \times 32 \Rightarrow 64$ or $32 \times 32 + 64 \Rightarrow 64$).
- ❑ **Version 4 – half-word load and store**
- ❑ **Version 5 – improved Thumb, additional instructions (count leading 0's, SW breakpoint, etc.)**
 - ❖ E – enhanced DSP instructions including saturated arithmetic operations and 16-bit multiply operations
 - ❖ J – support for new Java state, offering hardware and optimized software acceleration of bytecode execution.

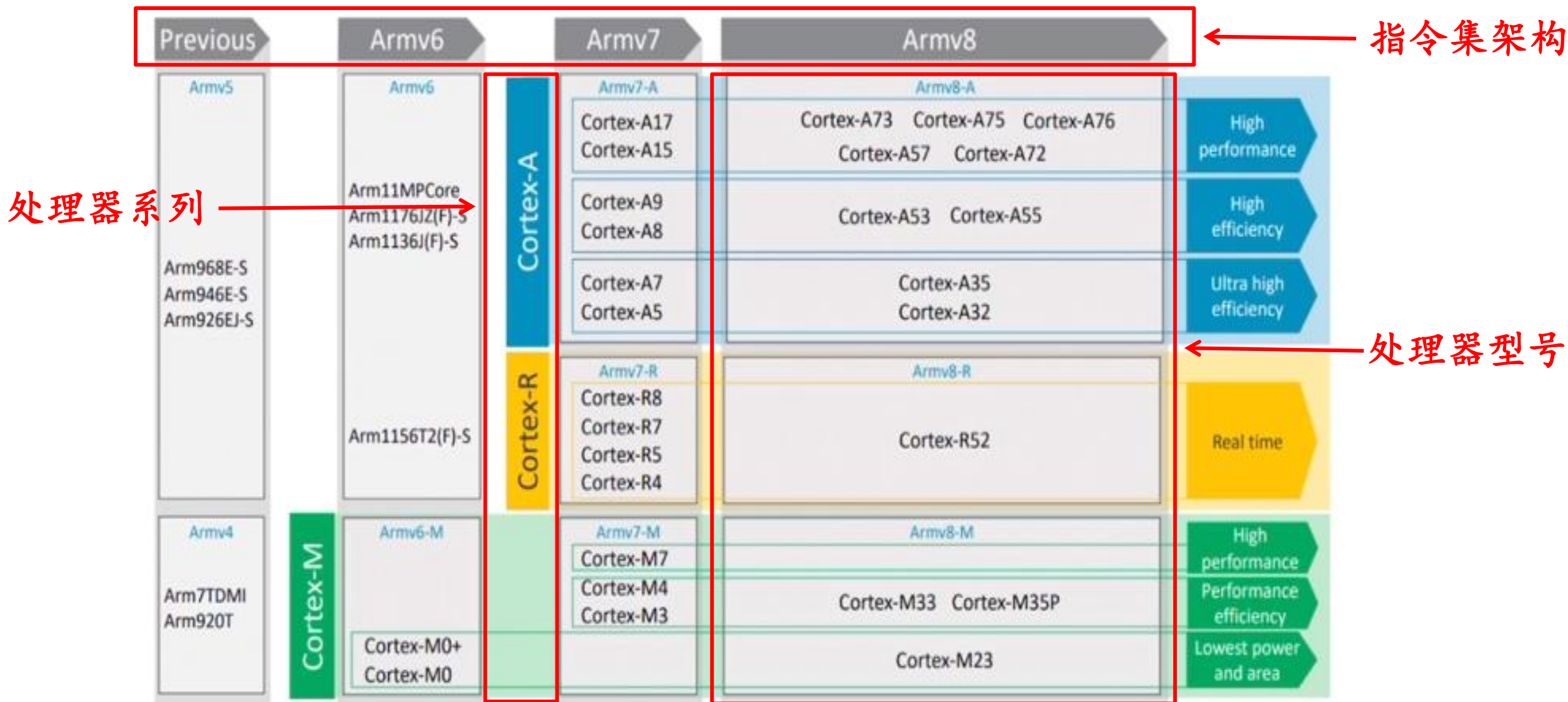
- ❑ **Version 6 – SIMD media processing, enhanced exception handling, overhaul of the memory system architecture**
- ❑ **Version 7 –**
 - ❖ Thumb-2 - variable length instruction set
 - ❖ TrustZone – two virtual processors backed by hardware based access control
 - ❖ Jazelle-RCT (ThumbEE)- supports ahead-of-time (AOT) and just-in-time (JIT) compilation
 - ❖ Neon – advanced 128-bit SIMD
 - ❖ Other extensions: MPE (multiprocessing), VE (virtualization), LPAE (large physical address), VFP (vector floating-point)
 - ❖ Profiles (A, M, R)
- ❑ **Version 8 – a fundamental change with AArch64 instruction set**

ARM920T什么意思？

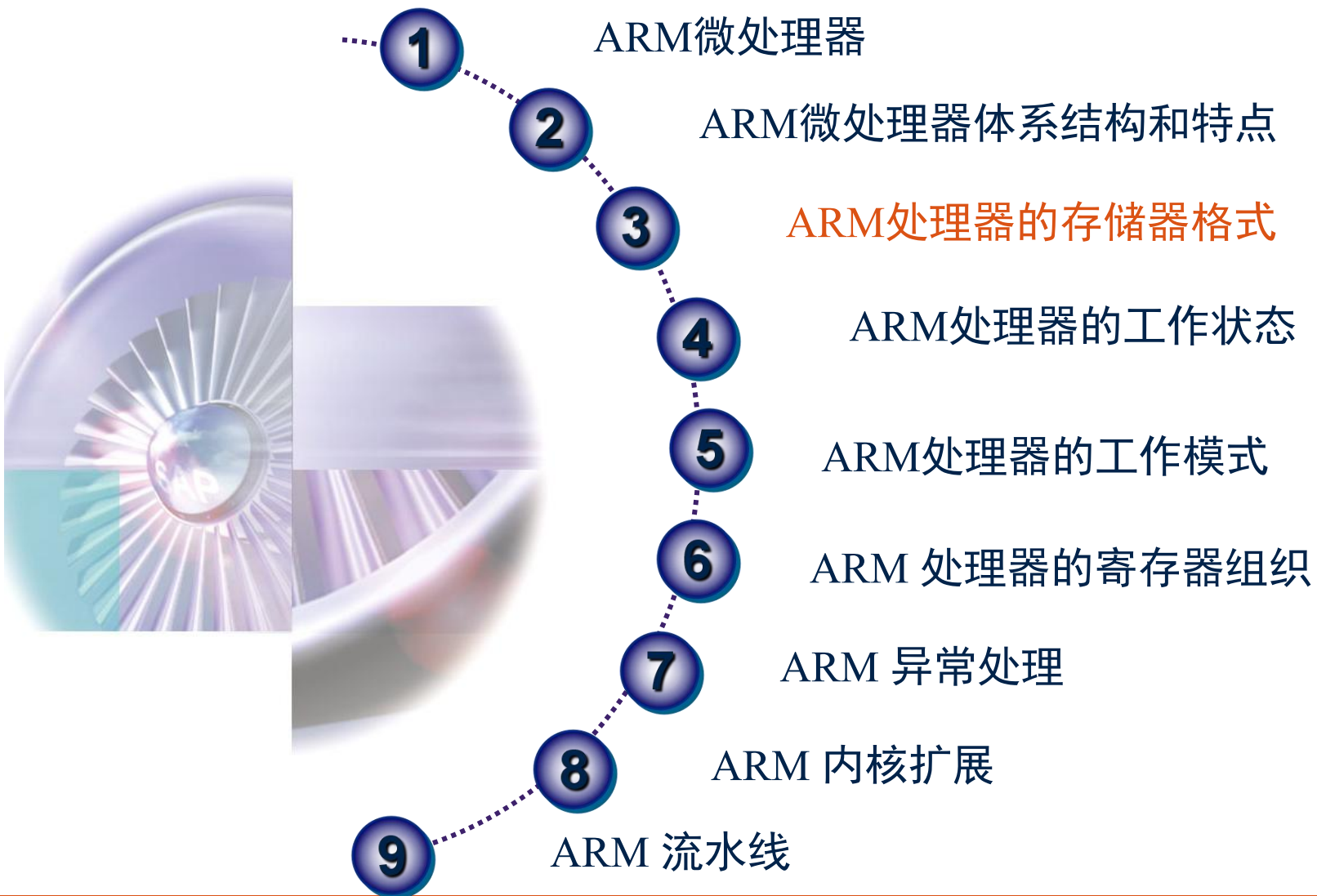


* jazelle 是 ARM 的 java extension

标志	含 义	说 明
T	支持Thumb指令集	Thumb指令集版本1: ARMv4T Thumb指令集版本2: ARMv5T Thumb-2: ARMv6T
D	片上调试	
M	支持长乘法	32位乘32位得到64位, 32位的乘加得到64位
I	Embedded ICE	In circuit emulation(支持片上断点和调试点)
E	DSP指令	增加了DSP算法处理器指令: 16位乘加指令, 饱和的带符号数的加减法, 双字数据操作, cache预取指令
J	Java加速器Jazelle	提高java代码的运行速度
S	可综合	提供VHDL或Verilog语言设计文件



- Intel
- Texas Instrument
- Samsung Semiconductor
- Freescale
- Philips Semiconductor
- Qualcomm
- Atmel
- Circus Logic
- 华为中心正购买ARM芯核，设计自主版权专用芯片



- 数据类型：字(Word)，半字(half-Word)，字节(Byte)
- 处理器工作状态：ARM，THUMB
- 存储器格式：大小端格式
- 工作模式：7种
- 寄存器组：31个通用寄存器和6个状态寄存器
- 异常处理

■ 数据类型:

- **字节型数据(Byte)**: 数据宽度为**8bits**;
- **半字数据类型(Halfword)**: 数据宽度为**16bits**, 存取时一般要求2字节对齐;
- **字数据类型(Word)**: 数据宽度为**32bits**, 存取时一般要求4字节对齐。

- 在v6架构之前，数据访问都必须把访问地址进行对齐，没有对齐的地址会导致“数据中止”异常。

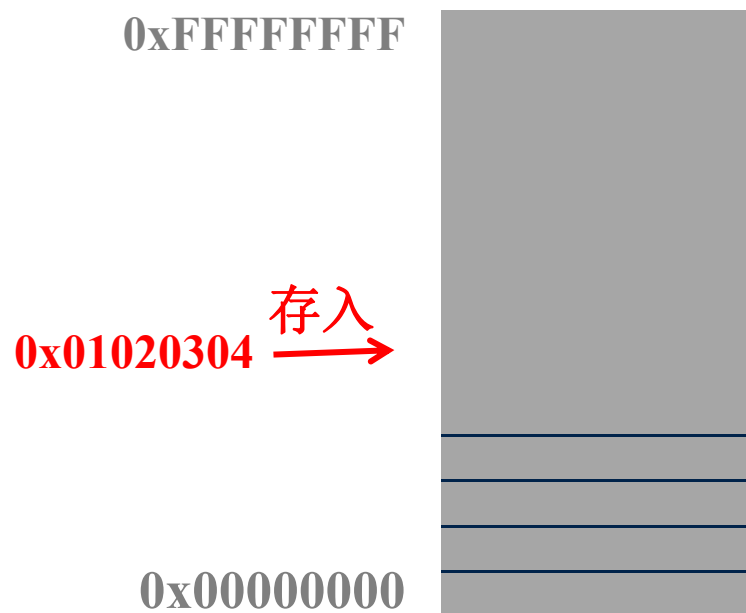
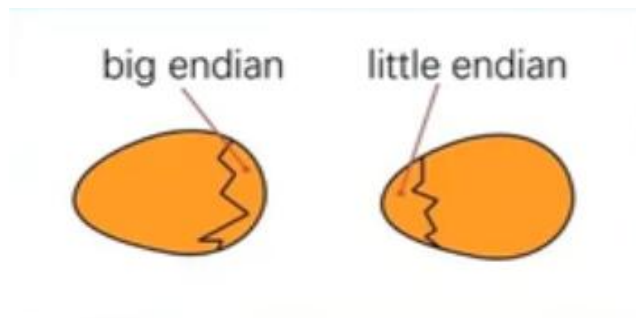


- ARM v6架构添加了新的硬件来支持未对齐的数据访问。

□ ARM体系结构所支持的最大寻址空间为**4GB** (2^{32} 字节)

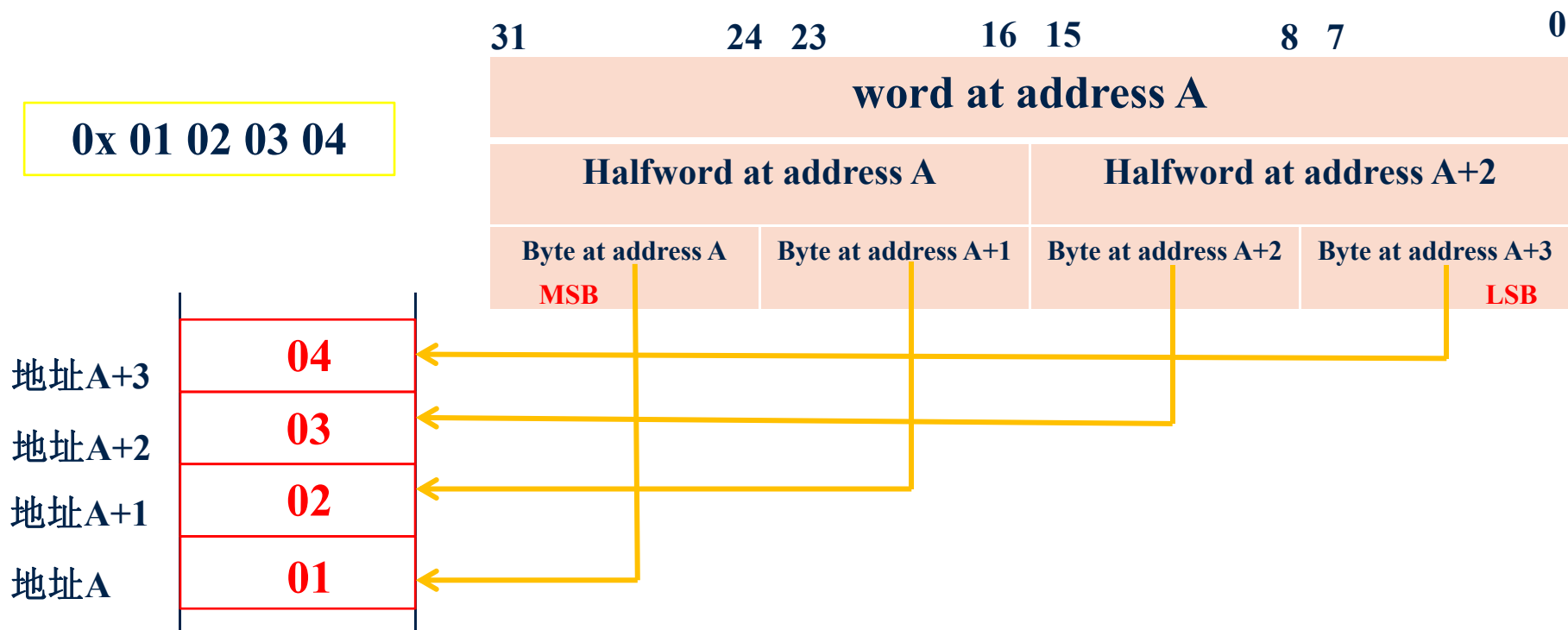
➤ 每个数据**4字节**

4个字节如何存放？

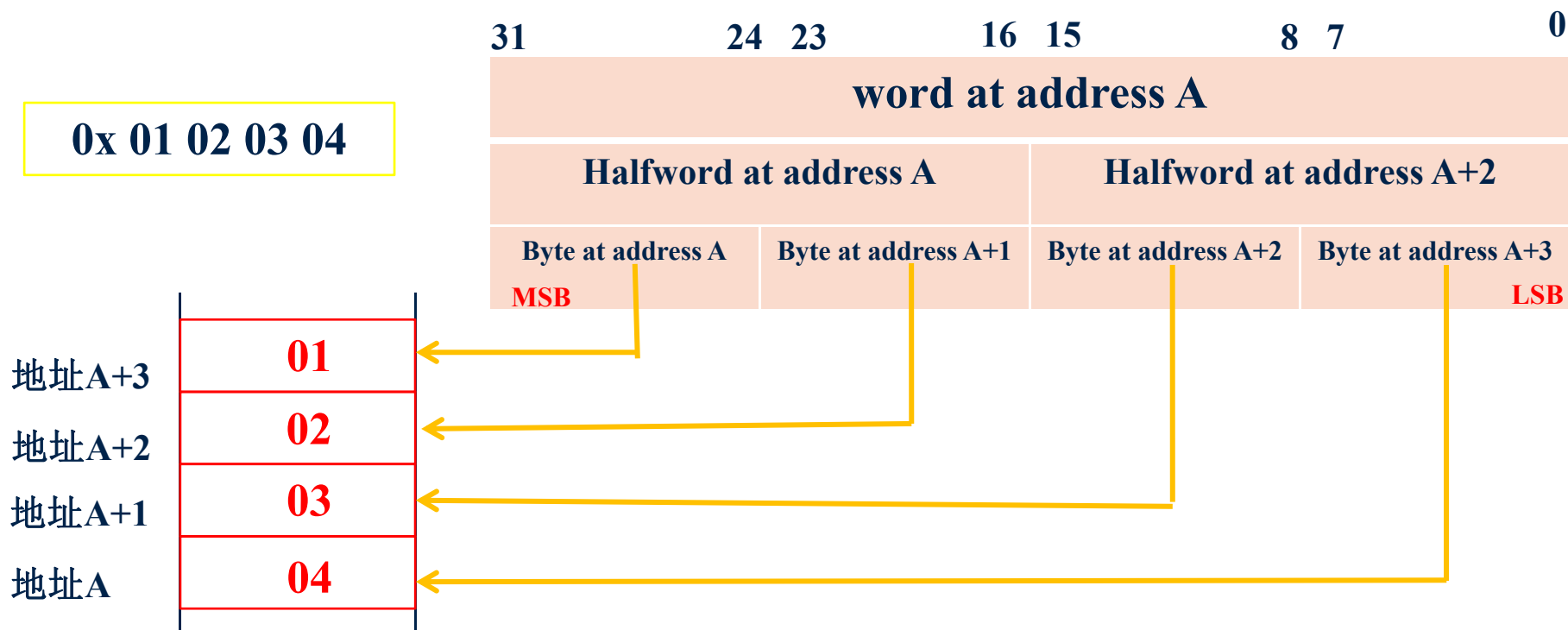


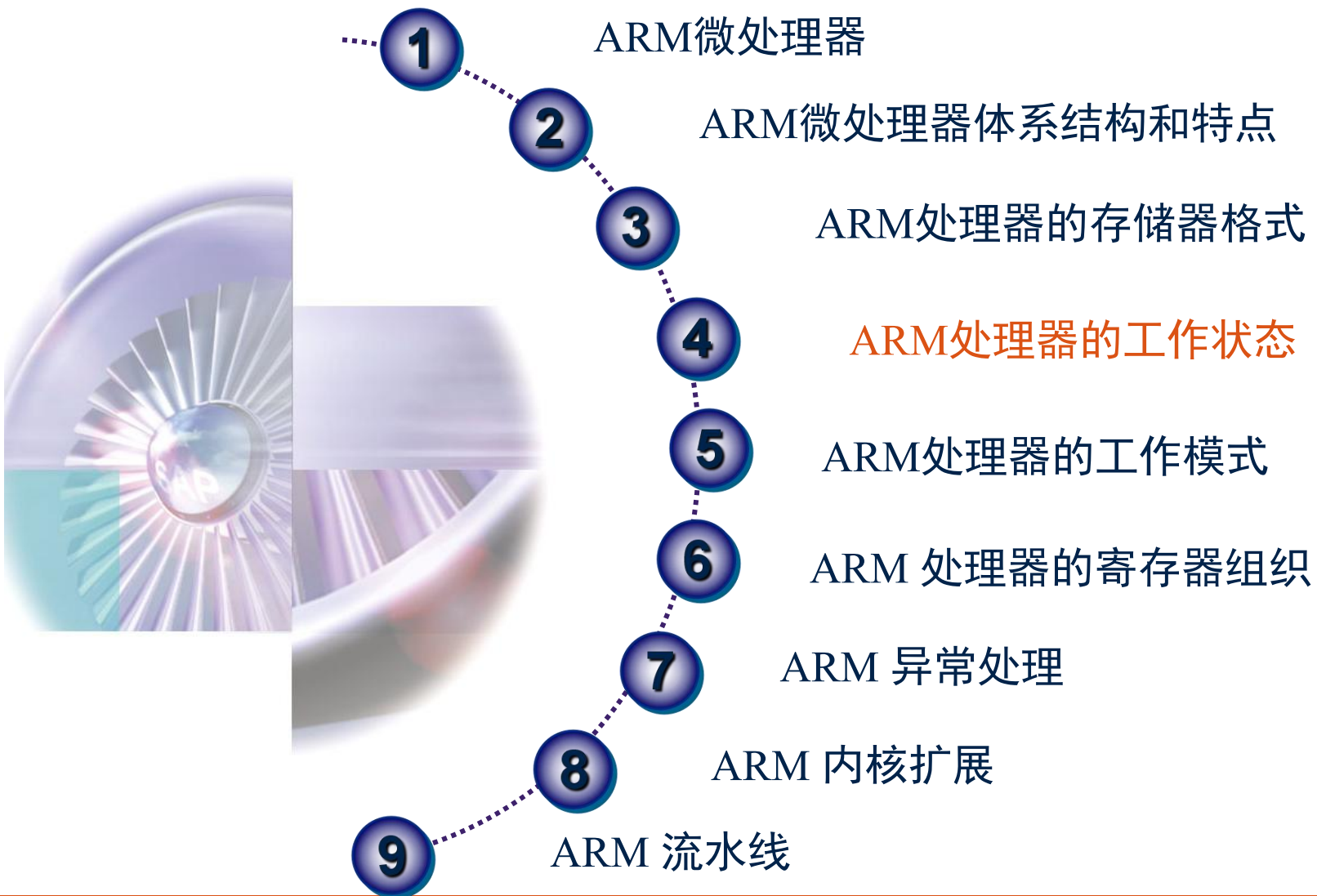
□ ARM体系结构可以用两种方法存储字数据(存储结构), 称之为**大端格式**和**小端格式**。

- ❑ 最高有效字节(**MSB**:Most Significant Byte)放内存**低**地址(A)。
- ❑ 最低有效字节(**LSB**:Least Significant Byte)放内存**高**地址(A+3)。
- ❑ 最低地址是该数据的地址(A)。



- ❑ 最**高**有效字节(**MSB**:Most Significant Byte)放内存**高**地址(A+3)。
- ❑ 最**低**有效字节(**LSB**:Least Significant Byte)放内存**低**地址(A)。
- ❑ 最低地址是该数据的地址(A)。





■ 大部分的ARM处理器都有两种指令集：

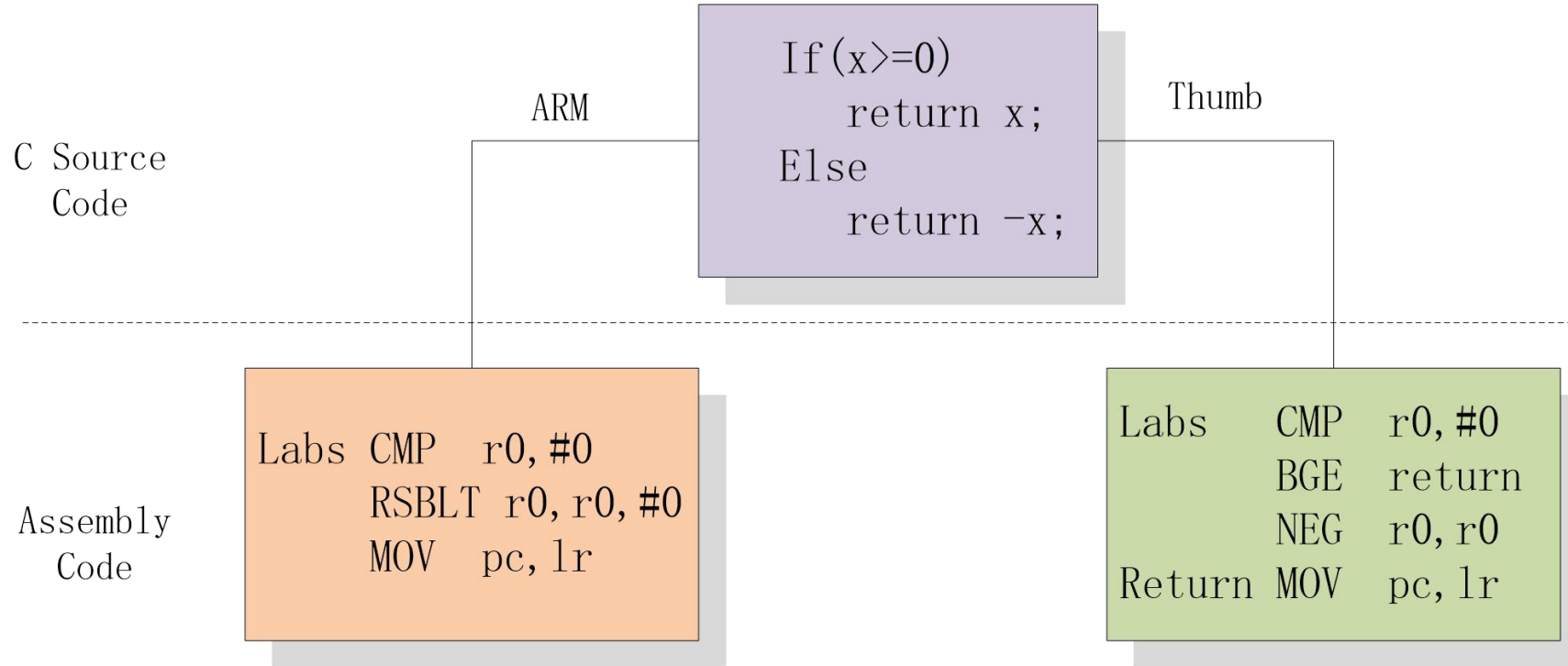
- ✓ 32位的ARM指令集
- ✓ 16位的Thumb指令集



■ ARM的两种工作状态：

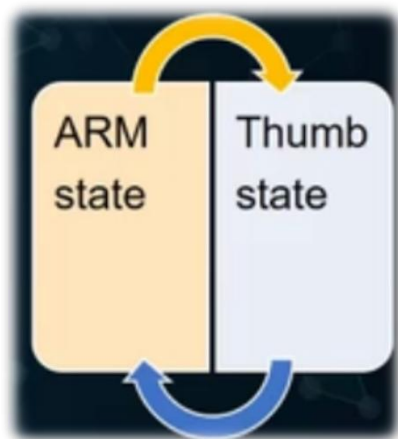
- ✓ **ARM状态：**此时处理器执行32位的ARM指令；
- ✓ **Thumb状态：**此时处理器执行16位的的Thumb指令。

- ❑ THUMB指令是ARM指令的子集，用于兼容16位总线系统
- ❑ 可以相互调用，只要遵循一定的调用规则
- ❑ Thumb指令与ARM指令的时间效率和空间效率关系为：
 - 存储空间约为ARM代码的60%~70%
 - 指令数比ARM代码多约30%~40%
 - 存储器为32位时ARM代码比Thumb代码快约40%
 - 存储器为16位时Thumb比ARM代码快约40~50%
 - 使用Thumb代码，存储器的功耗会降低约30%



	instruction	Size(Bytes)
ARM	3	12
Thumb	4	8

- ARM指令集和Thumb指令集均有切换处理器状态的指令，并可在两种工作状态之间切换；
- 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；
- 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。



；从Arm状态切换到Thumb状态

```
LDR    R0, =Lable+1
```

```
BX     R0
```

；从Thumb状态切换到ARM状态

```
LDR    R0, =Lable
```

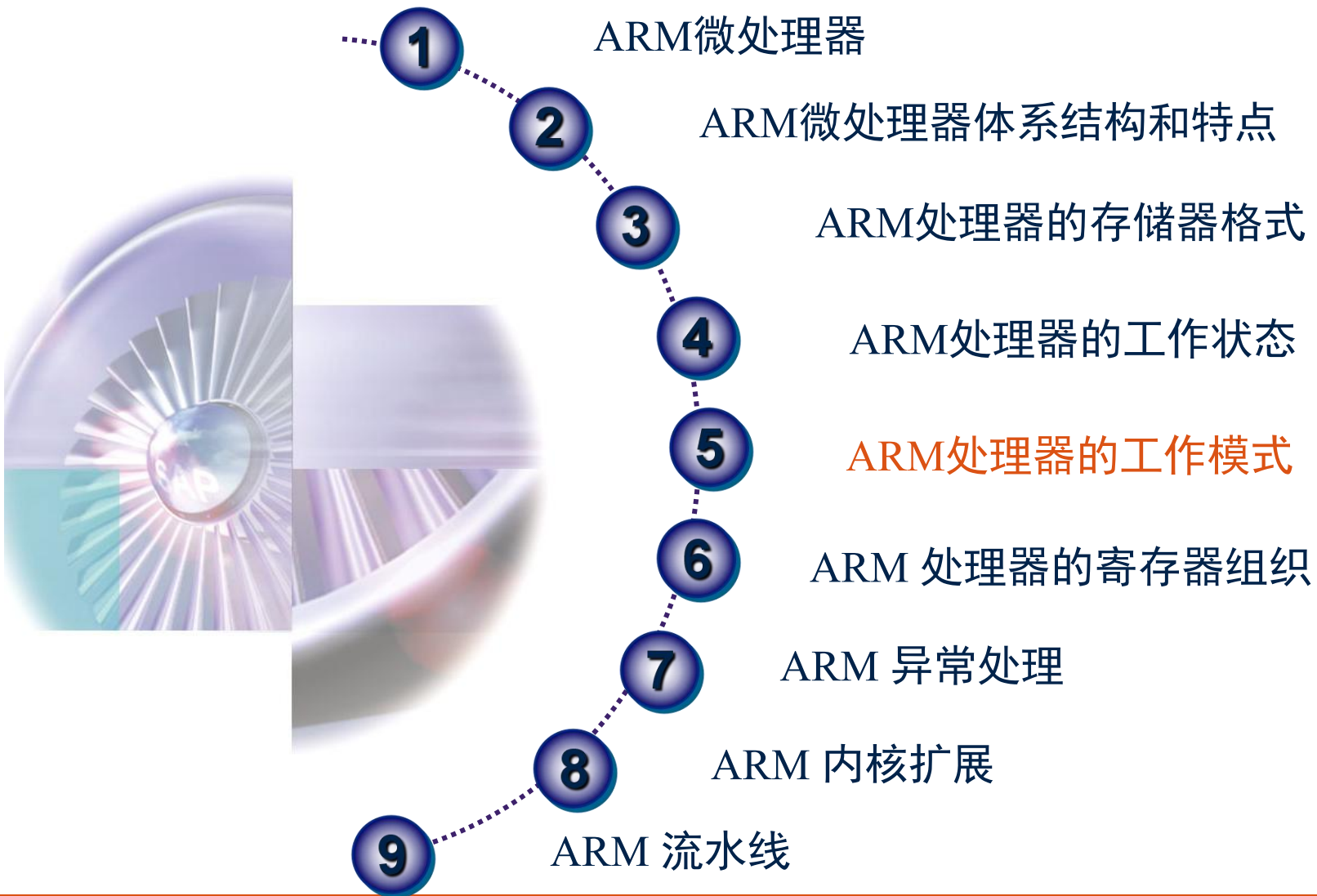
```
BX     R0
```

跳转地址标号

地址最低位为1，表示切换到Thumb状态

地址最低位为0，表示切换到ARM状态

- ❑ 在开始执行代码时，应该处于ARM状态。
- ❑ 在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。
- ❑ 在处理器进行异常处理时，无论处理器处于什么状态，都切换到ARM状态。
- ❑ 当处理器处于Thumb状态时发生异常（如IRQ、FIQ、Undef、Abort、SWI等），则异常处理返回时，自动切换到Thumb状态。



■ ARM 有7个基本工作模式:

- **User:** 用户模式。正常程序执行的模式。
 - 非特权模式，大部分任务执行在这种模式。
- **FIQ:** 快速中断模式。用于高速数据传输和通道处理。
 - 当一个高优先级(fast)中断产生时将会进入这种模式。
- **IRQ:** 外部中断模式。用于通常的中断处理。
 - 当一个低优先级(normal)中断产生时将会进入这种模式。
- **Supervisor:** 管理模式。供操作系统使用的一种保护模式。
 - 当复位或软中断指令执行时将会进入这种模式。
- **Abort:** 中止模式。用于虚拟存储及存储保护。
 - 当存取异常时将会进入这种模式。
- **Undef:** 未定义模式。用于硬件协处理器的软件仿真。
 - 当执行未定义指令时会进入这种模式。
- **System:** 系统模式。运行具有特权级的操作系统任务
 - 使用和用户模式相同寄存器集的特权模式。

模式	描述	分类
user	ARM处理器正常的程序执行状态	非特权模式
system	特权模式，与用户模式有相同寄存器	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	
Undefined	用来处理未定义的指令	
Abort	用来处理内存访问异常	
Supervisor	当复位或者软中断(SWI)指令被执行时进入	

 异常模式 非异常模式

❑ **usr:** 用户模式，也叫非特权模式，不能修改CPSR的控制域。

❑ **fiq:**

❑ **irq:**

❑ **svc:**

❑ **abt:**

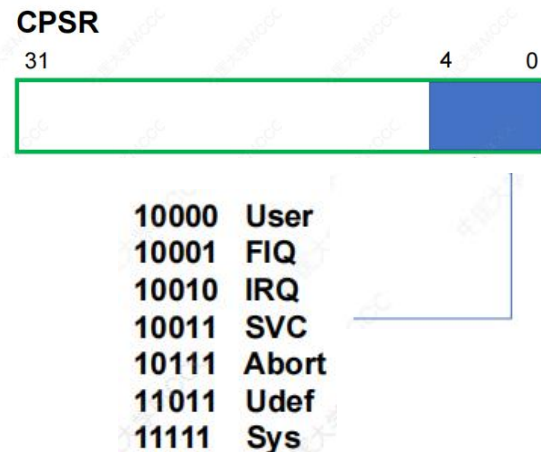
❑ **und:**

❑ **sys:**


异常模式

系统模式

特权模式：可访问处理器所有物理资源，处理中断、异常、读写CPSR等。



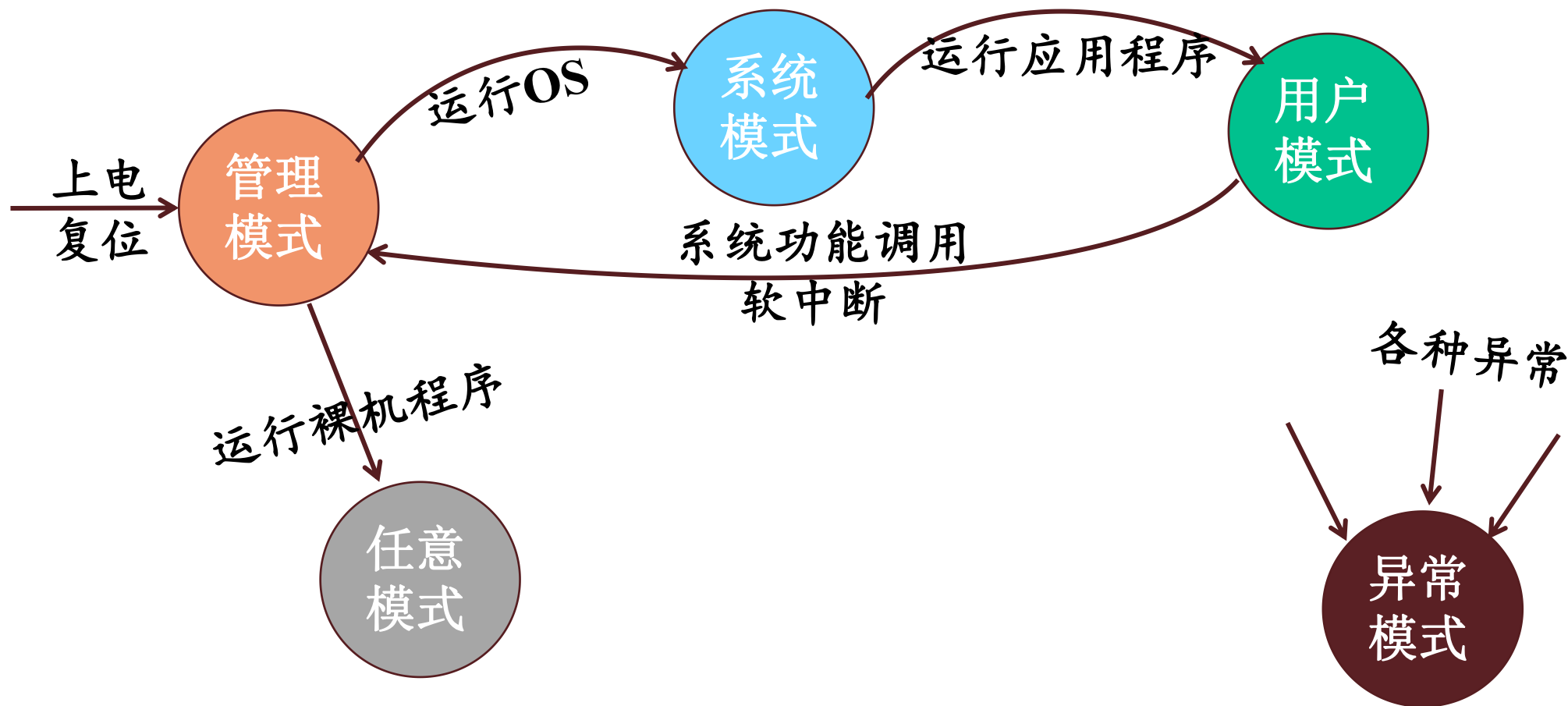
- ❑ 除用户模式外其他6种处理器模式称为特权模式。
- ❑ 特权模式下，程序可以访问所有的系统资源，也可以任意地进行处理器模式的切换。
- ❑ 特权模式中，除系统模式外，其他5种模式又称为异常模式
- ❑ 大多数的用户程序运行在用户模式下，此时，应用程序不能够访问一些受操作系统保护的系统资源，应用程序也不能直接进行处理器模式的切换。
- ❑ 用户模式下，当需要进行处理器模式切换时，应用程序可以产生异常处理，在异常处理中进行处理器模式的切换。

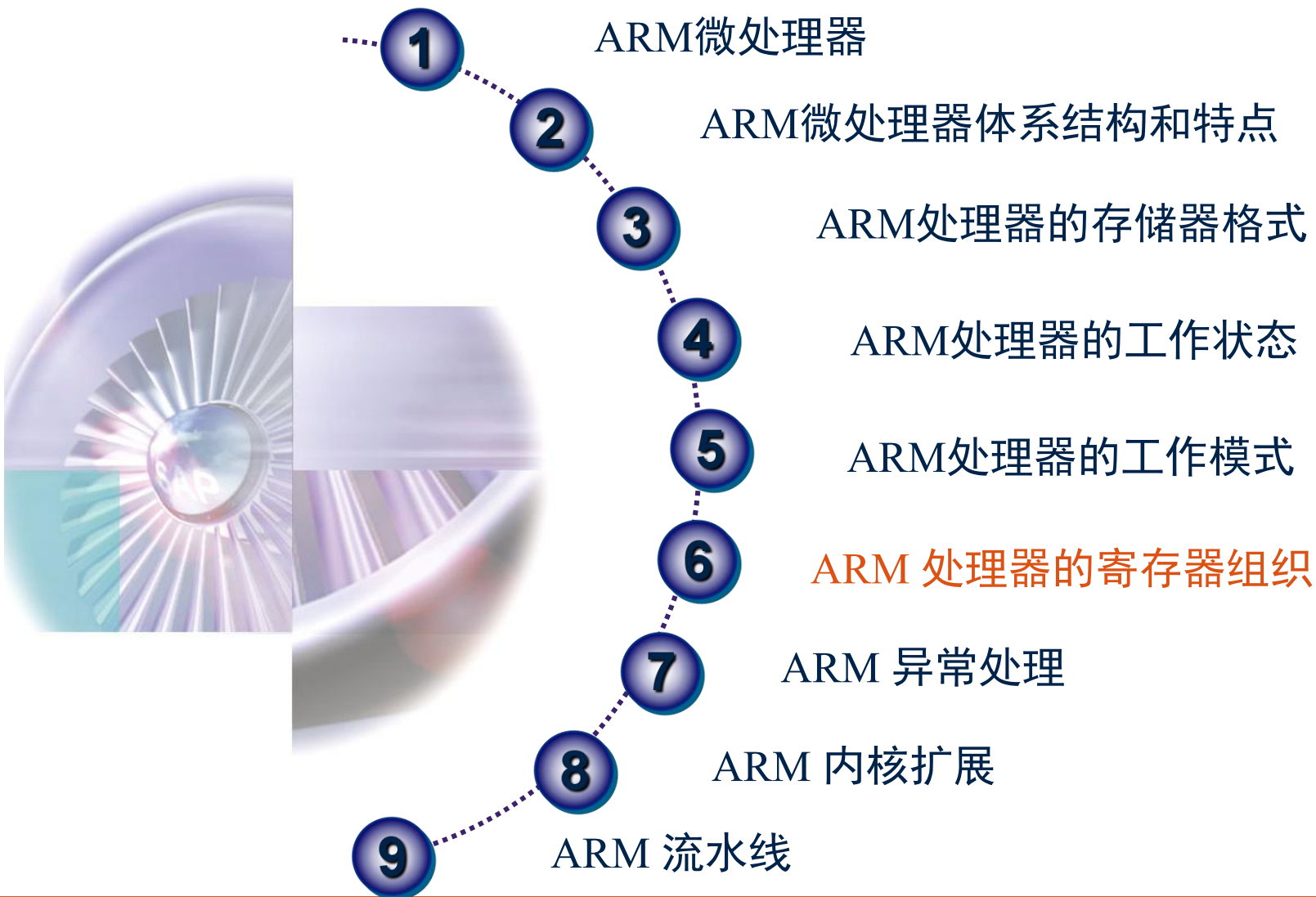
- ❑ 处理器模式可以通过软件进行切换，也可以通过外部中断或者异常处理过程进行切换。


主动切换 被动切换
- ❑ 当应用程序发生异常中断时，处理器进入相应的异常模式。在每一种异常模式下都有一组寄存器，供相应的异常处理程序使用。
- ❑ 系统模式并不是通过异常进入的，它和用户模式具有完全一样的寄存器。但是系统模式属于特权模式，可以访问所有的系统资源，也可以直接进行处理器模式切换。它主要供操作系统任务使用。

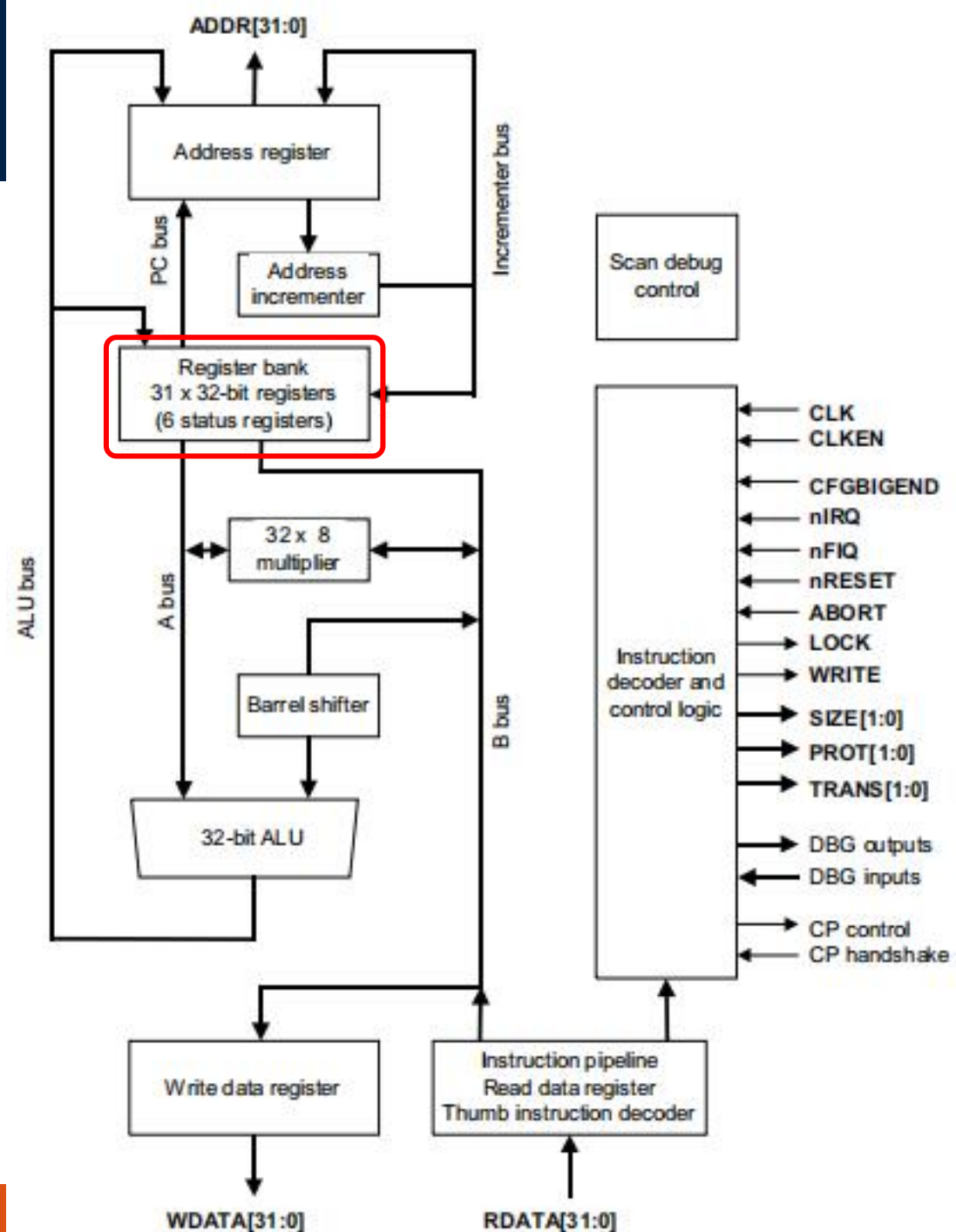
问题：在用户模式下，如何进入系统模式？

■ 处理器工作模式转换图





- 32 bit register
- load/store
- 31 general purpose register + 6 status register



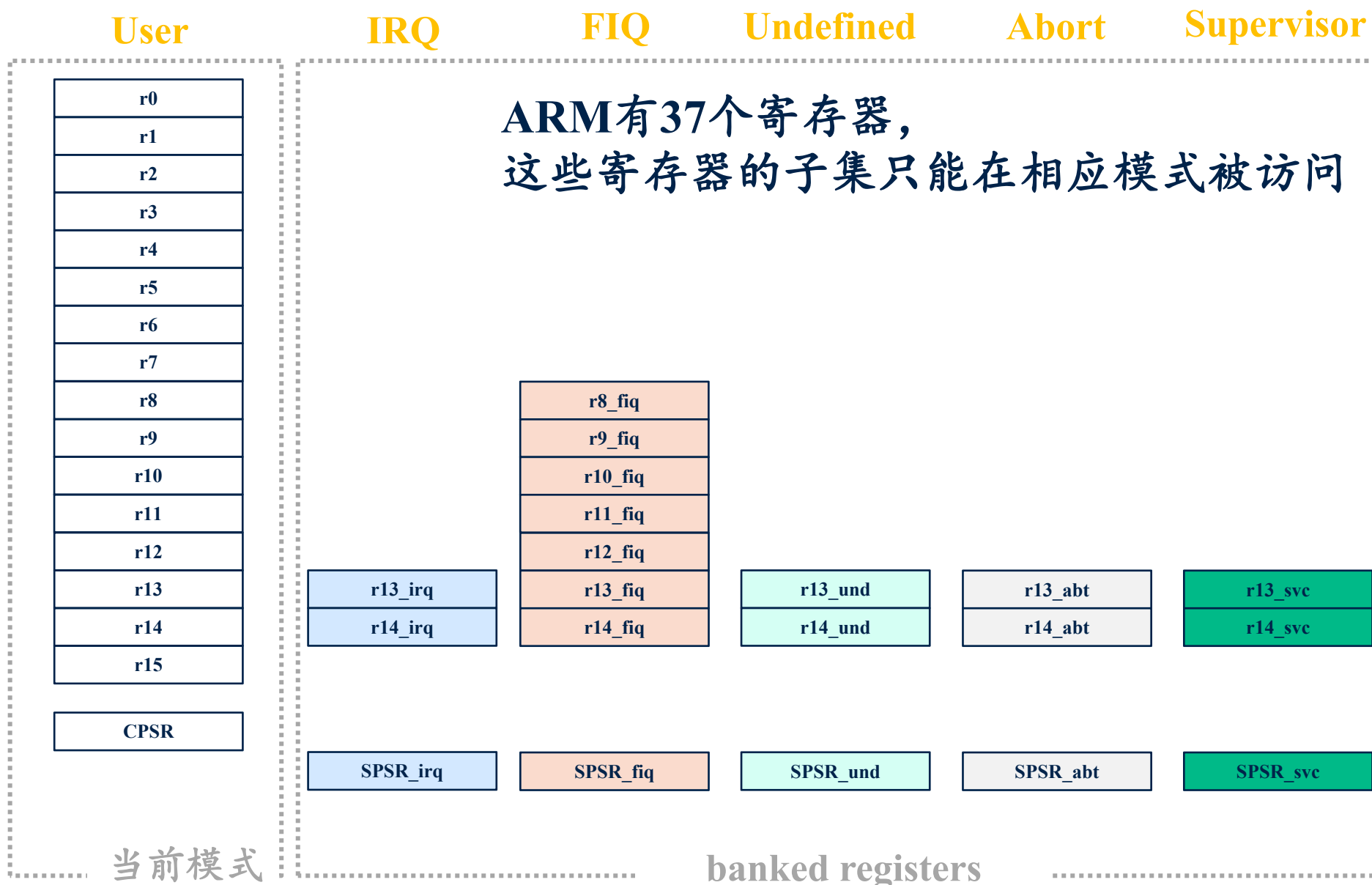
ARM state general registers and program counter

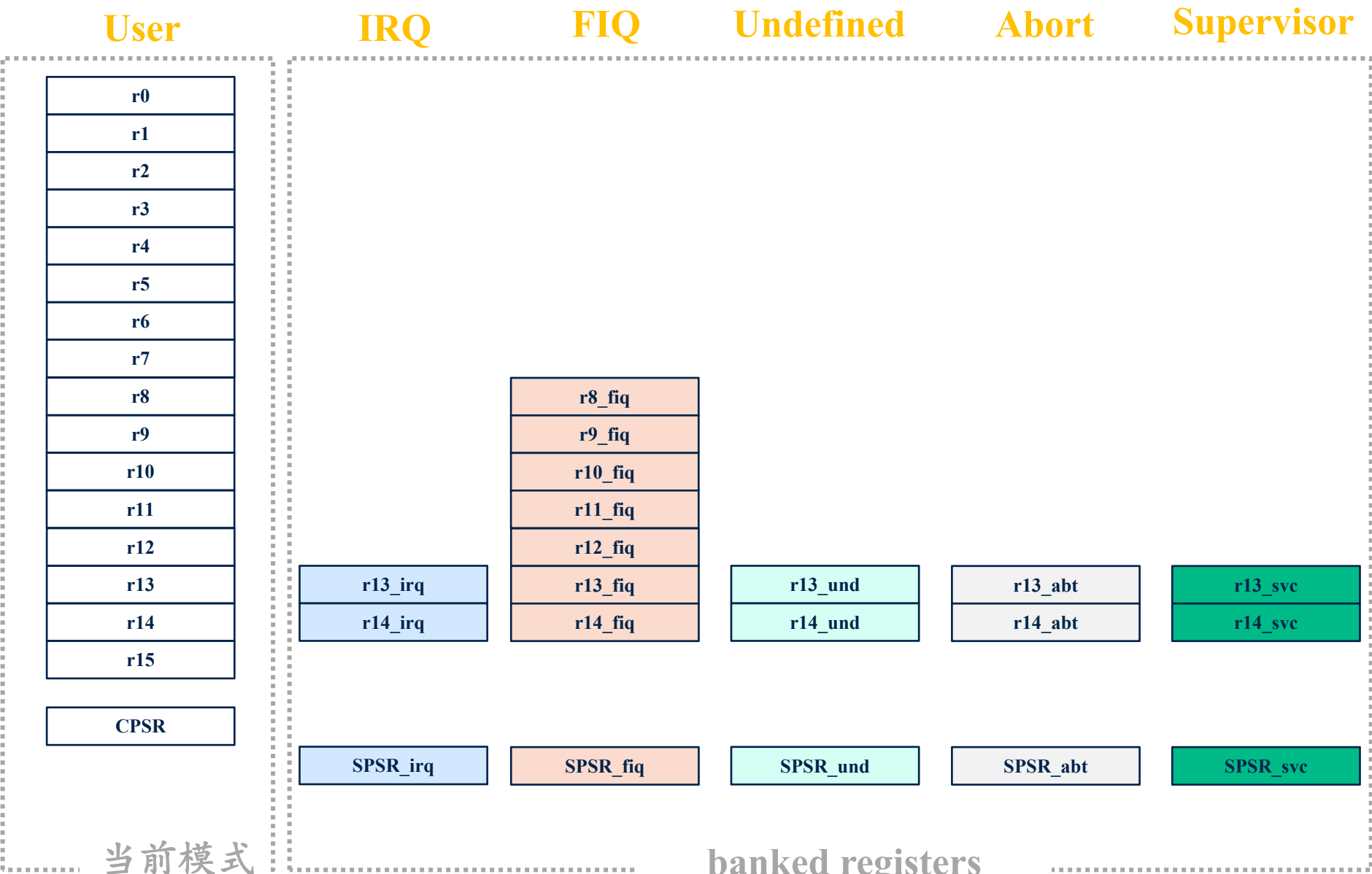
- r0~r15: 通用寄存器
 - ✓ r13: 栈指针
 - ✓ r14: 链接寄存器
 - ✓ r15: 指针寄存器
- CPSR: 当前程序状态寄存器
- SPSR: 保存的程序状态寄存器

System and User

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15 (PC)

CPSR






Thumb state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
SP	SP_fiq	SP_svc	SP_abt	SP_irq	SP_und
LR	LR_fiq	LR_svc	LR_abt	LR_irq	LR_und
PC	PC	PC	PC	PC	PC

Thumb state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

 = banked register

图：Thumb状态下的寄存器

- ARM9微处理器共有37个32位寄存器，其中31个为通用寄存器，6个为状态寄存器。
- 但是这些寄存器不能被同时访问，具体哪些寄存器是可编程访问的，取决微处理器的工作状态及具体的工作模式。
- 但在任何时候，通用寄存器R0~R14、程序计数器PC、一个或两个状态寄存器都是可访问的。

□ 通用寄存器：通用寄存器包括R0~R15，可以分为三类：

➤ 未分组寄存器R0~R7

➤ 分组寄存器R8~R14

➤ 程序计数器PC(R15)

□ 状态寄存器：包括 CPSR 和 SPSR 二类。

➤ 未分组寄存器R0~R7（无影子寄存器）

- 在所有的运行模式下，R0~R7所对应的物理寄存器都是相同的；
- 在工作模式切换时一般都需要对这几个寄存器进行保护。

ARM state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
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 state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

➤ 分组寄存器R8~R12（有影子寄存器）

- ❑ R8~R12：每个寄存器对应2个不同的物理寄存器（FIQ模式和非FIQ模式）
- ❑ 每次所访问的物理寄存器与处理器当前的运行模式有关；

➤ 当使用fiq模式时，访问寄存器R8_fiq~R12_fiq；

➤ 当使用非fiq模式时，访问寄存器R8~R12。



ARM state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
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 state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

➤ 分组寄存器R13~R14（有影子寄存器）

- ❑ R13、R14：每个寄存器对应6个不同的物理寄存器
- ❑ 其中一个为用户模式与系统模式共用，另外5个对应于其他5种不同的运行模式



ARM state general registers and program counter

System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
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 state program status registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

- ❑ R13在ARM指令中常用作堆栈指针sp，每一种工作模式（用户与系统模式共用）都有自己的独立的堆栈。
- ❑ 由于处理器的每种运行模式均有自己独立的物理寄存器R13。当程序的运行进入异常模式时，可以将需要保护的寄存器放入R13所指向的堆栈，而当程序从异常模式返回时，则从对应的堆栈中恢复。

初始化堆栈

```
17 ;Pre-defined constants 预定义6种工作模式
18 USERMODE EQU 0x10 ;用户模式
19 FIQMODE EQU 0x11 ;快速中断模式
20 IRQMODE EQU 0x12 ;中断模式
21 SVCMODE EQU 0x13 ;监管模式
22 ABORTMODE EQU 0x17 ;异常中断模式
23 UNDEFMODE EQU 0x1b ;未定义模式
24
25 MODEMASK EQU 0x1f ;模式掩码
26 NOINT EQU 0xc0 ;取消中断
```

```
511 mrs r0,cpsr
512 bic r0,r0,#MODEMASK
513 orr r1,r0,#UNDEFMODE|NOINT
514 msr cpsr,r1 ;UndefMode
515 ldr sp,=UndefStack ; UndefStack=0x33FF_5C00
```

```
517 orr r1,r0,#ABORTMODE|NOINT
518 msr cpsr,r1 ;AbortMode
519 ldr sp,=AbortStack ; AbortStack=0x33FF_6000
```

```
521 orr r1,r0,#IRQMODE|NOINT
522 msr cpsr,r1 ;IRQMode
523 ldr sp,=IRQStack ; IRQStack=0x33FF_7000
```

```
525 orr r1,r0,#FIQMODE|NOINT
526 msr cpsr,r1 ;FIQMode
527 ldr sp,=FIQStack ; FIQStack=0x33FF_8000
```

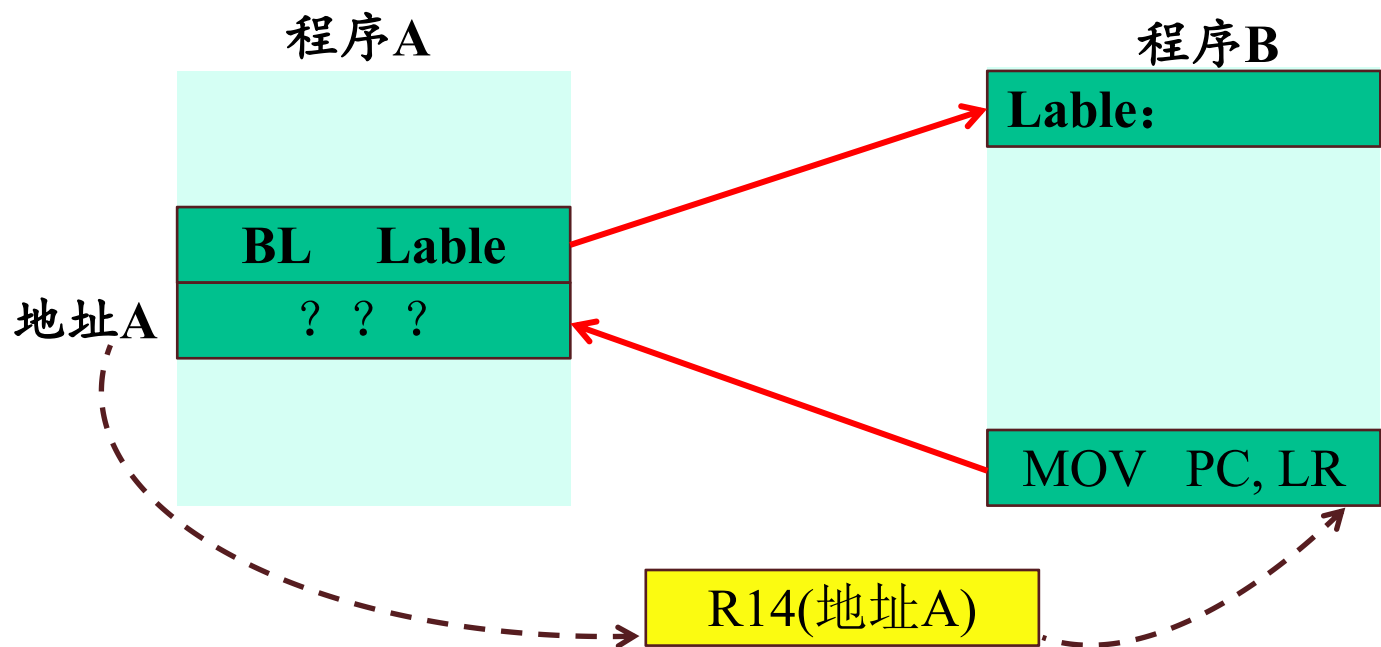
```
529 bic r0,r0,#MODEMASK|NOINT
530 orr r1,r0,#SVCMODE
531 msr cpsr,r1 ;SVCMode
532 ldr sp,=SVCStack ; SVCStack=0x33FF
```

；读cpsr 值送 r0
；r0低5位清0
；r0或上未定义模式取值和禁止FIQ和IRQ
中断的取值后，存r1中
；将r1传送给cpsr寄存器，使处理器处于
未定义模式，并禁止中断
；为该模式堆栈sp赋值

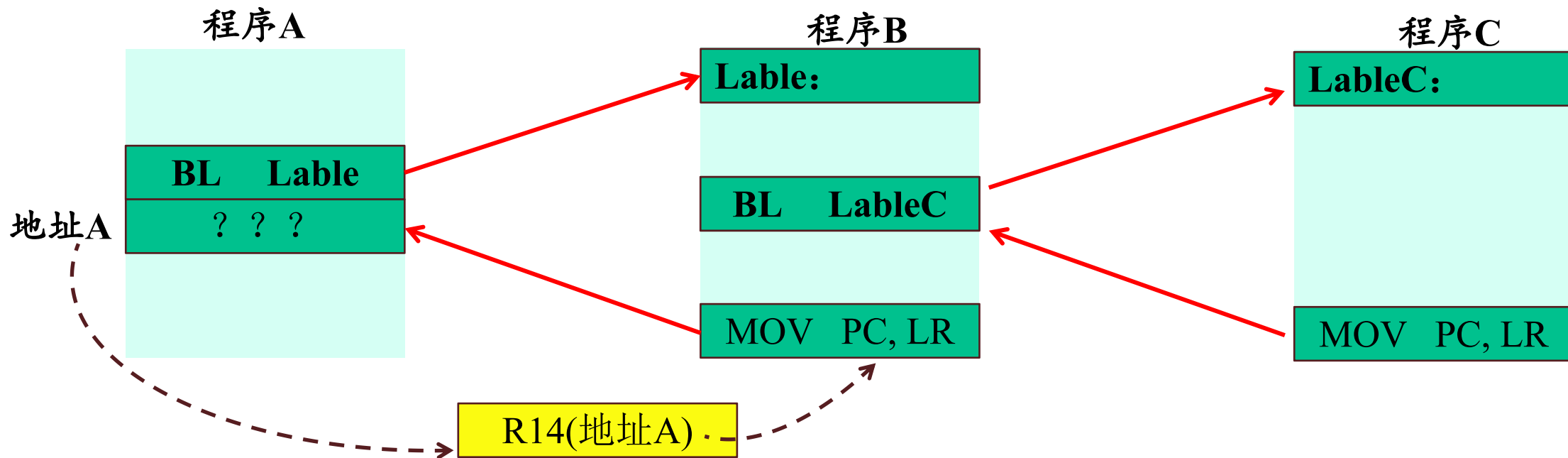
以下终止模式、FIQ、IRQ模式同上。

；r0低8位清0
；r0或上低5位赋值SVC管理模式取值
；cpsr赋值，使处理器处于SVC模式，并
开FIQ和IRQ中断
；为该模式堆栈sp赋值

- ❑ R14也称作子程序连接寄存器**LR**。子程序调用时，R14备份R15（程序计数器PC），**保存返回地址**。其他情况下，R14用作通用寄存器。
- ❑ 当用BL或BLX指令调用子程序时，将PC的当前值拷贝给R14，执行完子程序后，又将R14的值拷贝回PC，即可完成子程序的调用返回。



1. 程序A执行过程中调用程序B；
2. 程序跳转至标号Lable处执行程序B。同时将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。



子程序调用: **BL** **SUB1** /*调用子程序*/

.....

SUB1: **STMFD** SP!, {<regs>, LR} /*将R14存入堆栈*/

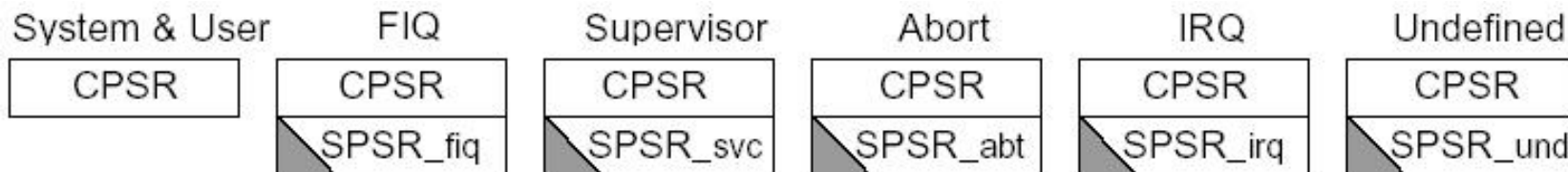
.....

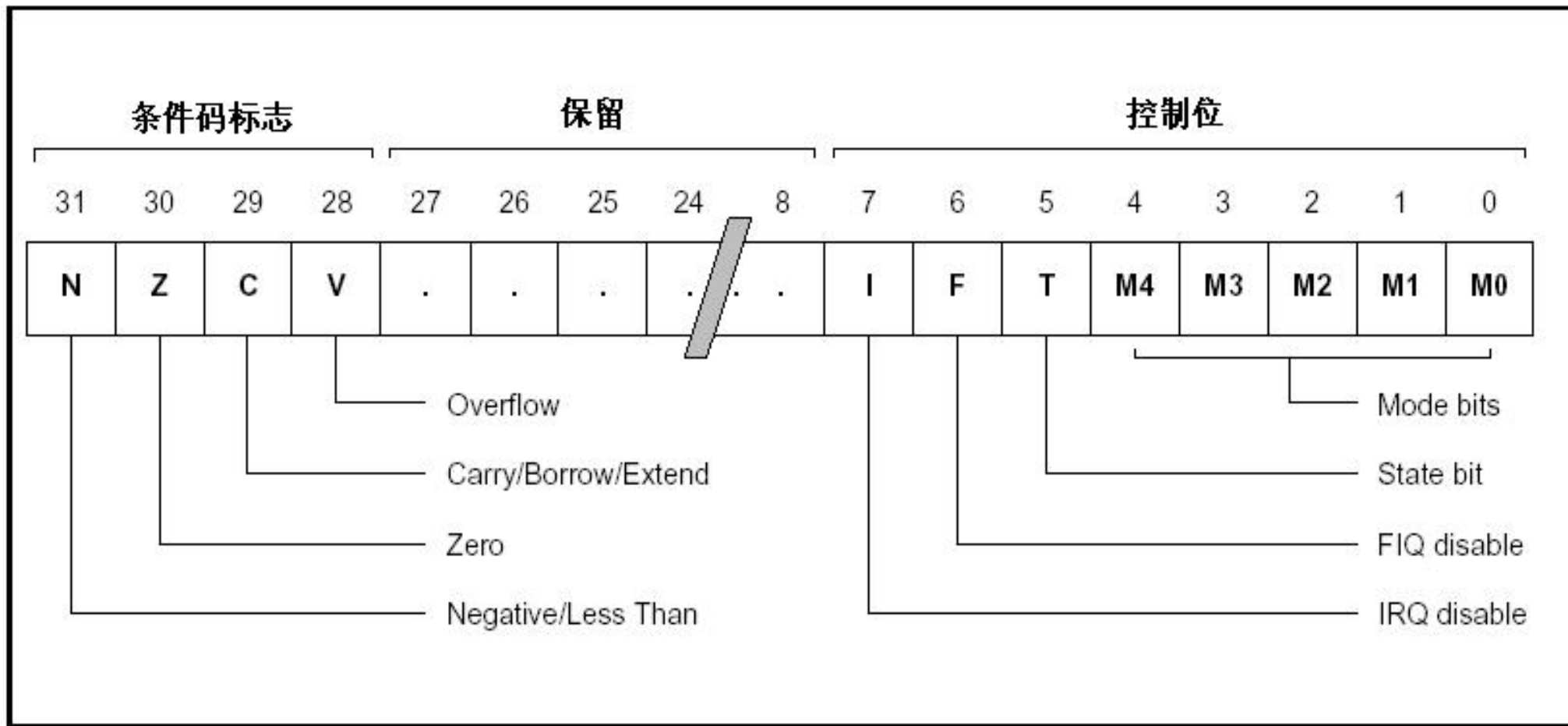
LDMFD SP!, {<regs>, PC} /*完成子程序返回 */

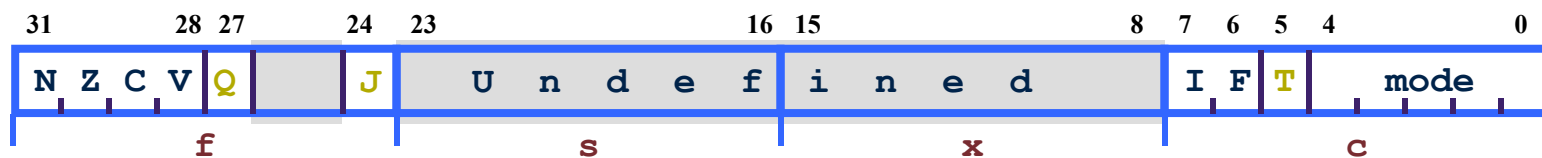
- 用作程序计数器 (PC) ;
- 当处理器执行在ARM状态:
 - ✓ 所有指令 32 bits 宽
 - ✓ 所有指令必须 word 对齐
 - ✓ 所以 **pc值**由bits [31:2]决定, bits [1:0] 为0 (所以指令不能halfword / byte对齐).
- 当处理器执行在Thumb状态:
 - ✓ 所有指令 16 bits 宽
 - ✓ 所有指令必须 halfword 对齐
 - ✓ 所以 **pc值**由bits [31:1]决定, bits [0] 位0 (所以指令不能 byte对齐).

- ❑ R15虽然也可用作通用寄存器，但一般不这么使用，因为对R15的使用有一些特殊的限制，当违反了这些限制时，程序的执行结果是未知的。
- ❑ 由于ARM体系结构采用了多级流水线技术，对于ARM指令集而言，PC总是指向当前正在执行的指令的下两条指令的地址，即PC的值为当前指令的地址值加8个字节。

- ❑ 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）。
- ❑ 异常发生时，SPSR用于保存CPSR的值，从异常退出时则可由SPSR来恢复CPSR。
- ❑ 由于用户模式和系统模式不属于异常模式，他们没有SPSR。







■ 条件位:

- N = 1-结果为负, 0-结果为正或0
- Z = 1-结果为0, 0-结果不为0
- C = 1-进位, 0-借位
- V = 1-结果溢出, 0-结果没溢出

■ Q 位:

- 仅ARM 5TE/J架构支持
- 指示增强型DSP指令是否溢出

■ J 位

- 仅ARM 5TE/J架构支持
- J = 1: 处理器处于Jazelle状态

■ 中断禁止位:

- I = 1: 禁止 IRQ.
- F = 1: 禁止 FIQ.

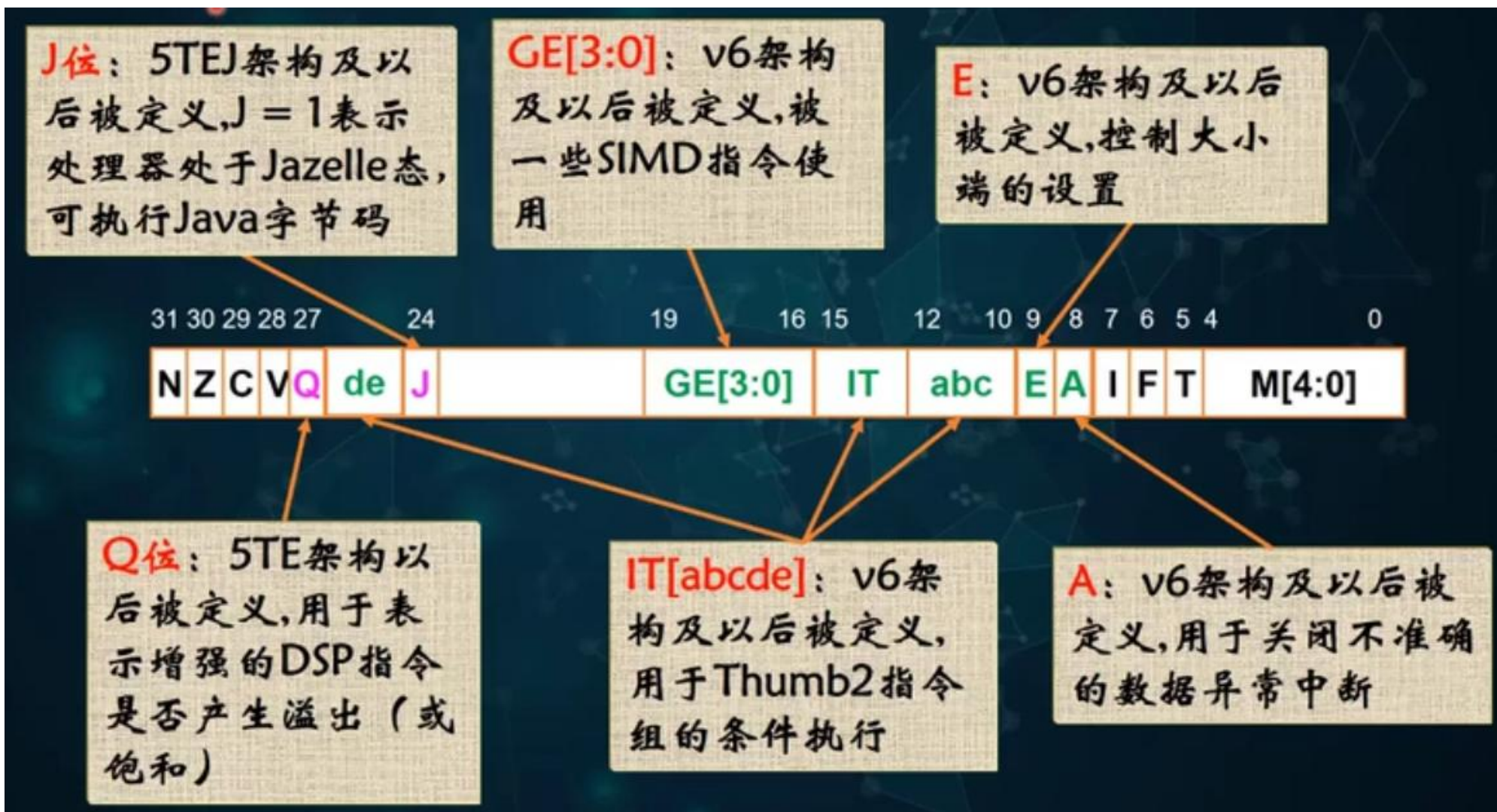
■ T 位:

- 仅ARM xT架构支持
- T = 0: 处理器处于 ARM 状态
- T = 1: 处理器处于 Thumb 状态

■ Mode位(处理器模式位):

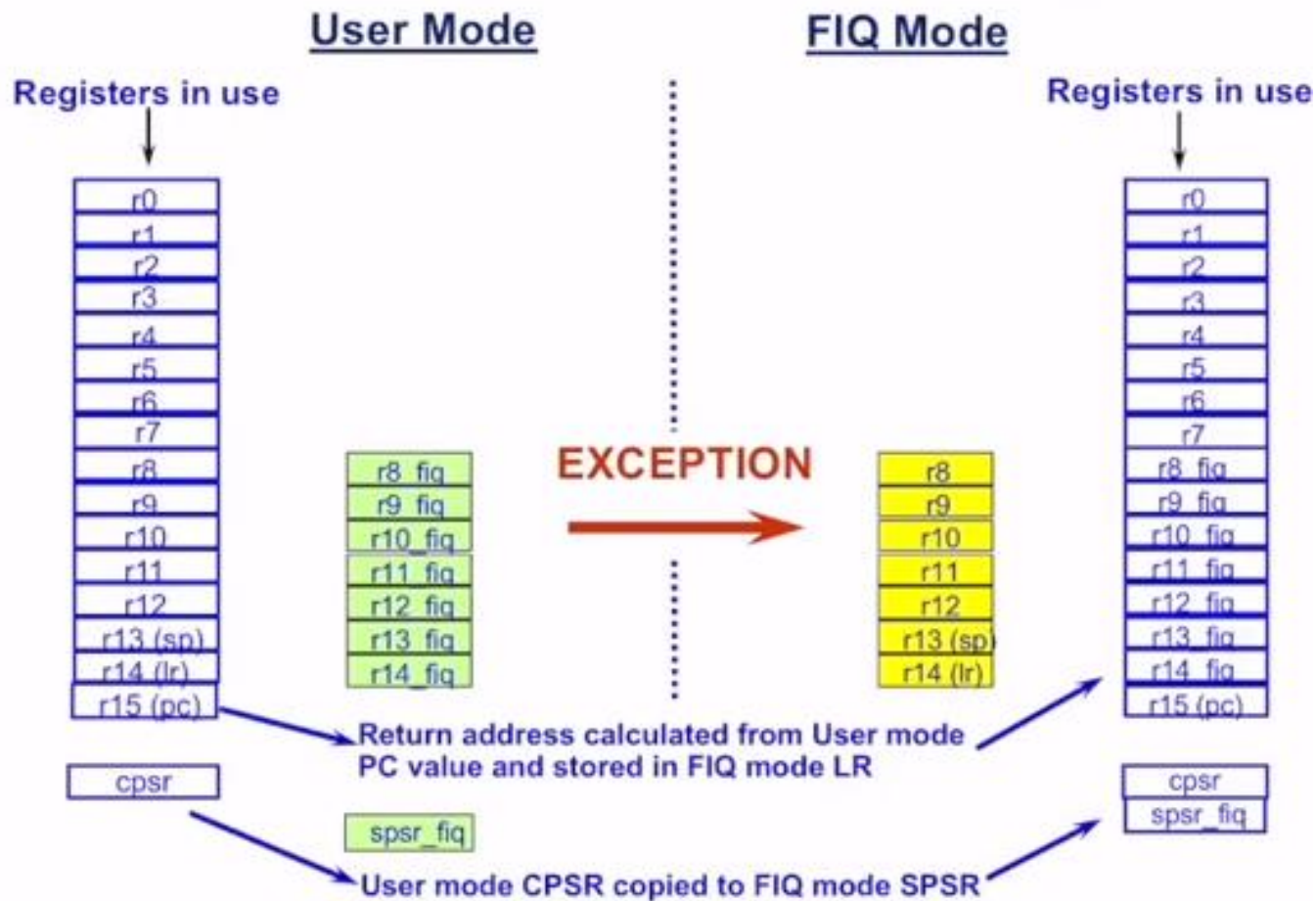
- 0b10000 User
- 0b10001 FIQ
- 0b10010 IRQ
- 0b10011 Supervisor
- 0b10111 Abort
- 0b11011 Undefined
- 0b11111 System

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	加法运算结果进位时，C=1，减法运算借位时，C=0； 移位操作的非加/减运算指令，C为移出的最后一位； 其他的非加/减运算指令，C的值通常不改变。
V	加/减法运算指令，V=1表示符号位溢出。 对于其他的非加/减运算指令，C的值通常不改变。
Q	在ARM v5及以上版本的E系列处理器中，Q标志指示DSP运算指令是否溢出。在其他版本中，Q标志位无定义。

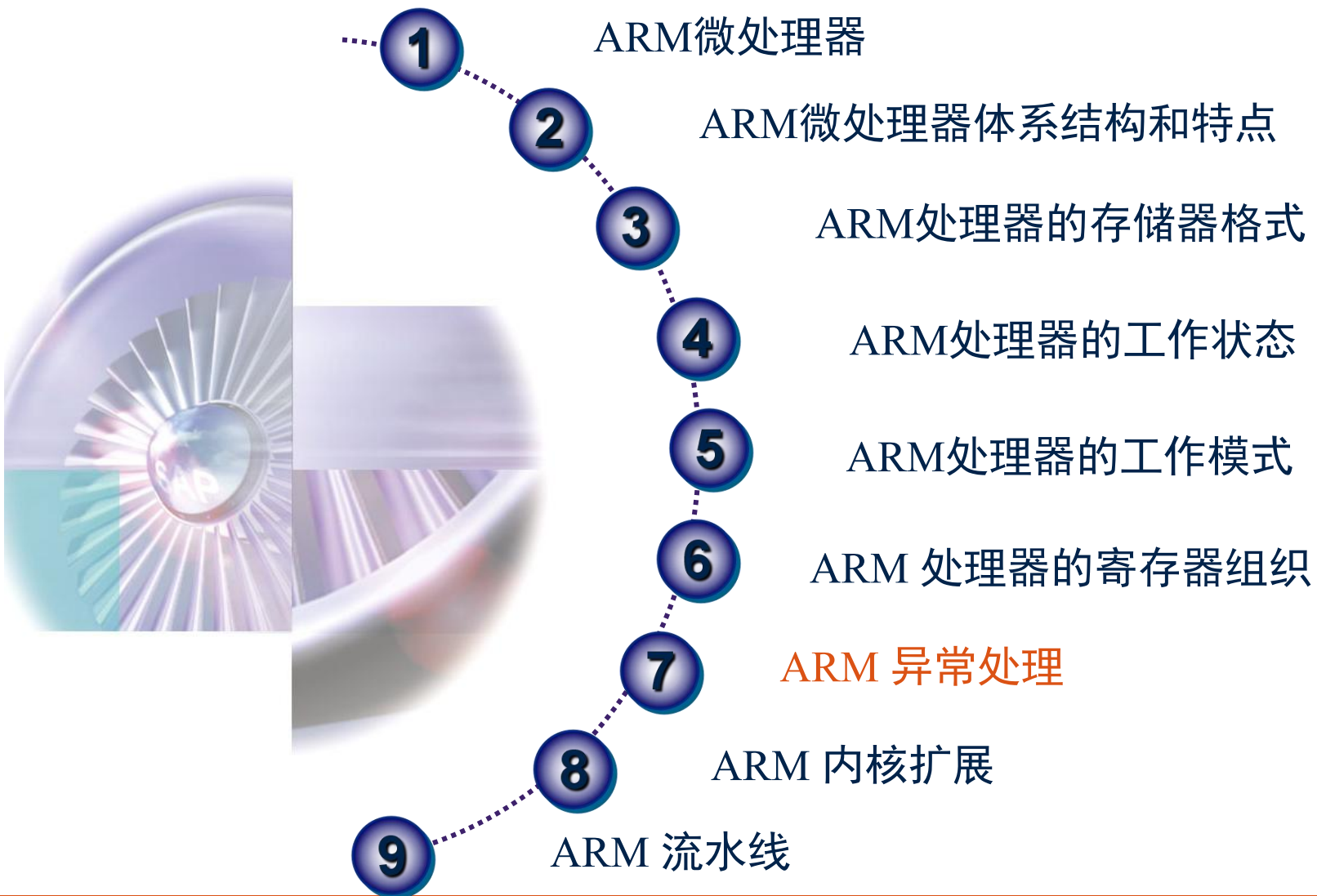


- R0-R15: 可以直接访问
- R0-R14: 通用寄存器
 - ✓ R0-R7: 未分组寄存器，所有模式下都是同一个物理寄存器。
 - ✓ R8-R14: 分组寄存器，不同模式下对应不同的物理寄存器
 - ✓ R13: 堆栈指针 (SP)
 - ✓ R14: 链接寄存器 (LR)
- R15: 程序计数器 (PC)
- CPSR: 当前程序状态寄存器
- 5个SPSR (程序状态保存寄存器)

从“用户模式”到“FIQ模式”的转换



提问：快速中断模式的“快速”体现在哪？？



什么是异常？

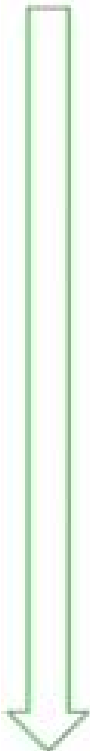
- **Exceptions** arise whenever the normal flow of a program has to be halted **temporarily**, for example to service an interrupt from a peripheral.
- **异常**：打断程序正常的执行过程。

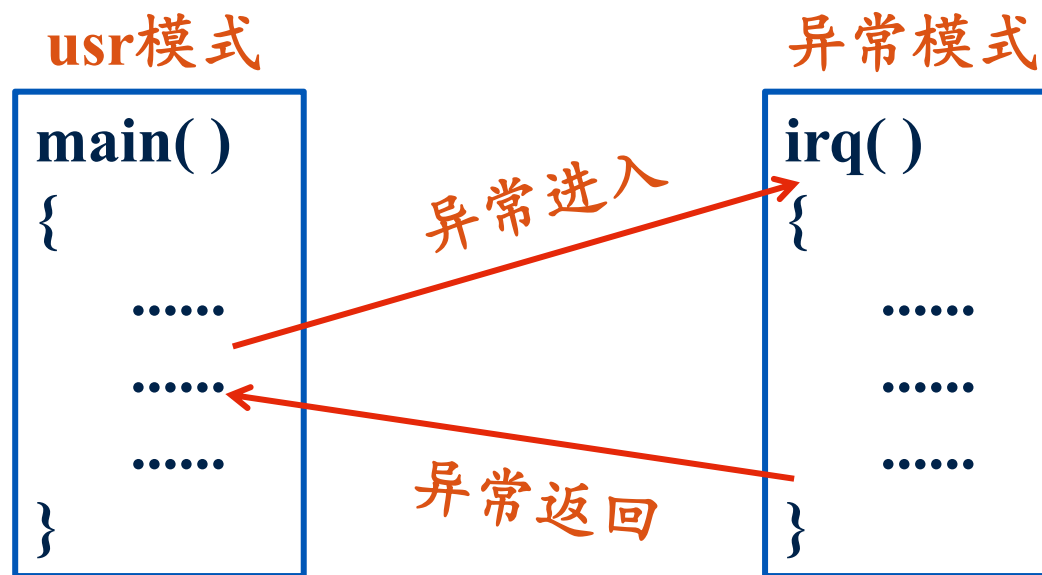
“中断”是异常的一种。

异常类型	具体含义	工作模式
复位	复位电平有效时，产生复位异常，程序跳转到复位处理程序处执行。	管 理
未定义指令	遇到不能处理的指令时，产生未定义指令异常。	未定义
软件中断	执行SWI指令产生，用于用户模式下的程序调用特权操作指令。	管 理
指令预取中止	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，产生指令预取中止异常。	中 止
数据预取中止	处理器数据访问的地址不存在，或该地址不允许访问时，产生数据中止异常。	中 止
IRQ	外部中断请求有效，且CPSR中的I位为0时，产生IRQ异常。	IRQ
FIQ	快速中断请求引脚有效，且CPSR中的F位为0时，产生FIQ异常。	FIQ

- ARM体系结构中的**异常**与**中断**有很大的相似之处，但异常与中断的概念并不完全等同。
- 在处理异常之前，当前处理器的状态**CPSR**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。

Priority

- 
1. *Reset (highest priority)*
 2. *Data Abort*
 3. *FIQ*
 4. *IRQ*
 5. *Prefetch Abort*
 6. *Undefined instruction*
 7. *SWI (lowest priority).*



- 1. 如何进入异常响应?
- 2. 如何退出异常响应?

- 当一个异常发生时，ARM微处理器会**自动执行**以下几步操作：
 - **CPSR保存**到相应异常模式下的**SPSR**；
 - **PC寄存器**的值保存到相应异常模式下的**LR(R14)**； **LR保存的是什么值？**
 - 将**CPSR**设置成相应的**异常模式**；
 - 设置**PC**值为相应**异常**处理程序的**入口地址**。

- 发生异常时，程序计数器PC会被强制设置为对应的异常向量，从而跳转到异常处理程序。
- 在异常向量表中的特定位置放置一条跳转指令，跳转到异常处理程序。

	⋮
0x0000001C	FIQ
0x00000018	IRQ
0x00000014	(Reserved)
0x00000010	Data Abort
0x0000000C	Prefetch Abort
0x00000008	Software Interrupt
0x00000004	Undefined Instruction
0x00000000	Reset

Vector Table

Vector table can be at
0xFFF0000 on ARM720T
and on ARM9/10 family devices

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

■实际编程中断向量表的使用:

cpu/arm920t/start.s:

```
00038:
00039: .globl _start
00040: start: b start_code
00041:     ldr pc, _undefined_instruction
00042:     ldr pc, _software_interrupt
00043:     ldr pc, _prefetch_abort
00044:     ldr pc, _data_abort
00045:     ldr pc, _not_used
00046:     ldr pc, _irq
00047:     ldr pc, _fiq
00048:
```

```
00103:
00104: /*
00105:  * the actual start code
00106:  */
00107:
00108: start_code:
00109:     /*
00110:      * set the cpu to SVC32 mode
00111:      */
00112:     mrs r0, cpsr
00113:     bic r0, r0, #0x1f
00114:     orr r0, r0, #0xd3
00115:     msr cpsr, r0
00116:
00117:     bl coloured_LED_init
00118:     bl red_LED_on
```

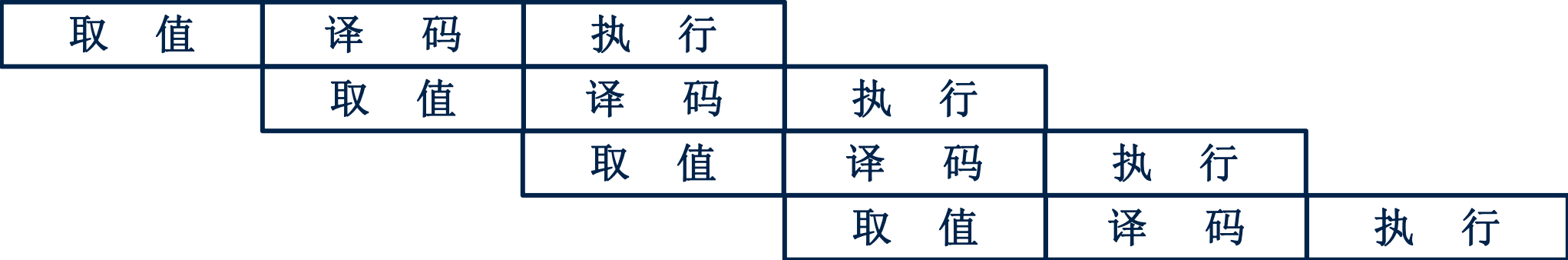
❑ 异常处理完毕之后，ARM微处理器会**自动执行**以下几步操作从异常返回：

- 从相应的异常模式的**SPSR寄存器**中复制回**CPSR的值**；
- 根据异常类型**恢复PC值**，以执行用户原来程序。
- 若在进入异常处理时设置了**中断禁止位**，要在此**清除**。

Note: 可以认为应用程序总是从复位异常处理程序开始执行的，因此复位异常处理程序不需要返回。

Table 2-3 Exception entry and exit

Exception or entry	Return instruction	Previous state		Notes
		ARM r14_x	Thumb r14_x	
BL	MOV PC, R14	PC + 4	PC + 2	Where the PC is the address of the BL, SWI, undefined instruction Fetch, or instruction that had the Prefetch Abort.
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	
Undefined instruction	MOVS PC, R14_und	PC + 4	PC + 2	
Prefetch Abort	SUBS PC, R14_abt, #4	PC + 4	PC + 4	
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	Where the PC is the address of the instruction that was not executed because the FIQ or IRQ took priority.
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	
Data Abort	SUBS PC, R14_abt, #8	PC + 8	PC + 8	Where the PC is the address of the Load or Store instruction that generated the Data Abort.
RESET	Not applicable	-	-	The value saved in r14_svc on reset is UNPREDICTABLE.



■ 当异常产生时, ARM core:

- 拷贝 CPSR 到 SPSR_<mode>
- 设置适当的 CPSR 位:
 - 改变处理器状态进入 ARM 态
 - 改变处理器模式进入相应的异常模式
 - 设置中断禁止位禁止相应中断 (如果需要)
- 保存返回地址到 LR_<mode>
- 设置 PC 为相应的异常向量

■ 返回时, 异常处理需要:

- 从 SPSR_<mode>恢复CPSR
- 从LR_<mode>恢复PC
- **Note:**这些操作只能在 ARM 态执行.

	⋮
0x1C	FIQ
0x18	IRQ
0x14	(Reserved)
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Software Interrupt
0x04	Undefined Instruction
0x00	Reset

Vector Table

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und



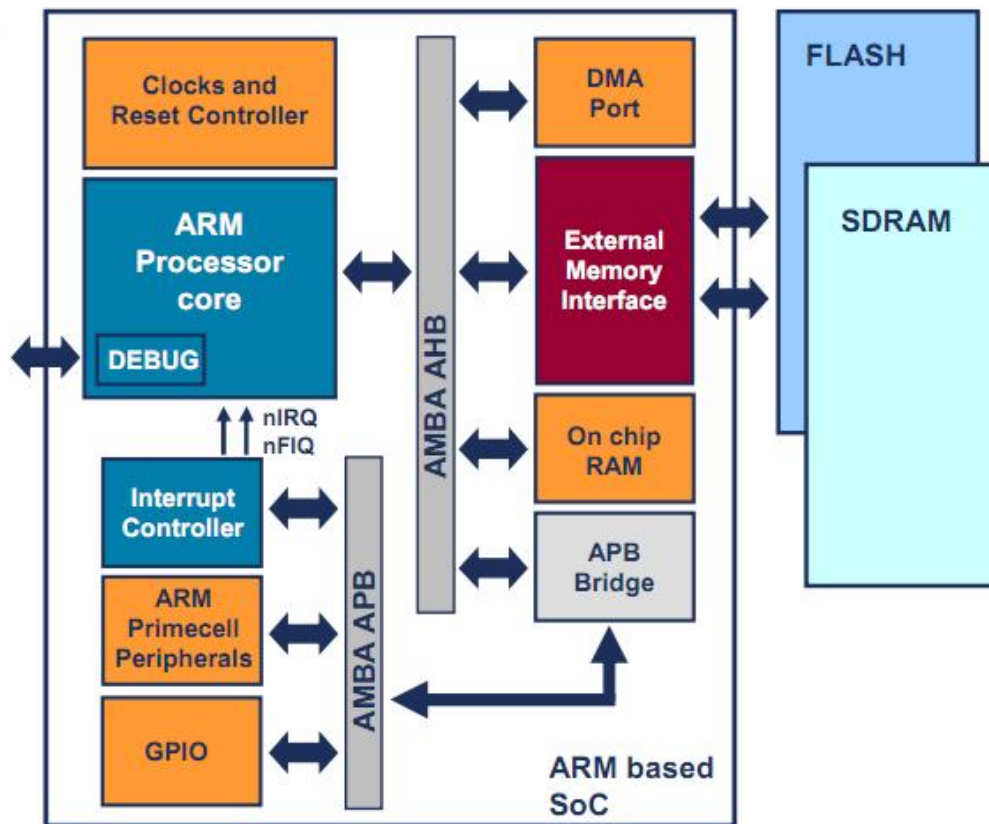
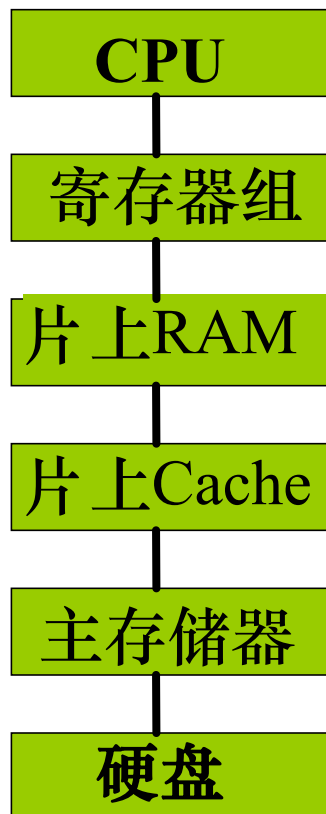
- 用来**改善系统性能、管理资源**以及提供额外的功能，为处理特殊的应用提供灵活性。是由**硬件实现的**。每个ARM系列都有不同程度的功能扩展。
- 有3种硬件扩展位于内核周围：
 - 1、**Cache和紧耦合存储器TCM**：都是位于内核和主存之间的快速存储器，提高访问速度。
 - 2、**存储管理**：存储设备多，用硬件实现其管理，保护系统免遭非法访问。
 - 3、**协处理器接口**：通过协处理器扩展指令集，扩展内核功能。

你尽管复习



考到了算我输

- ARM存储空间为多级复合存储器系统因此有必要实施某种策略来组织这些设备，并保护系统，避免一些应用非法访问硬件。可以使用**存储器管理硬件**来实现这些功能。



你尽管复习



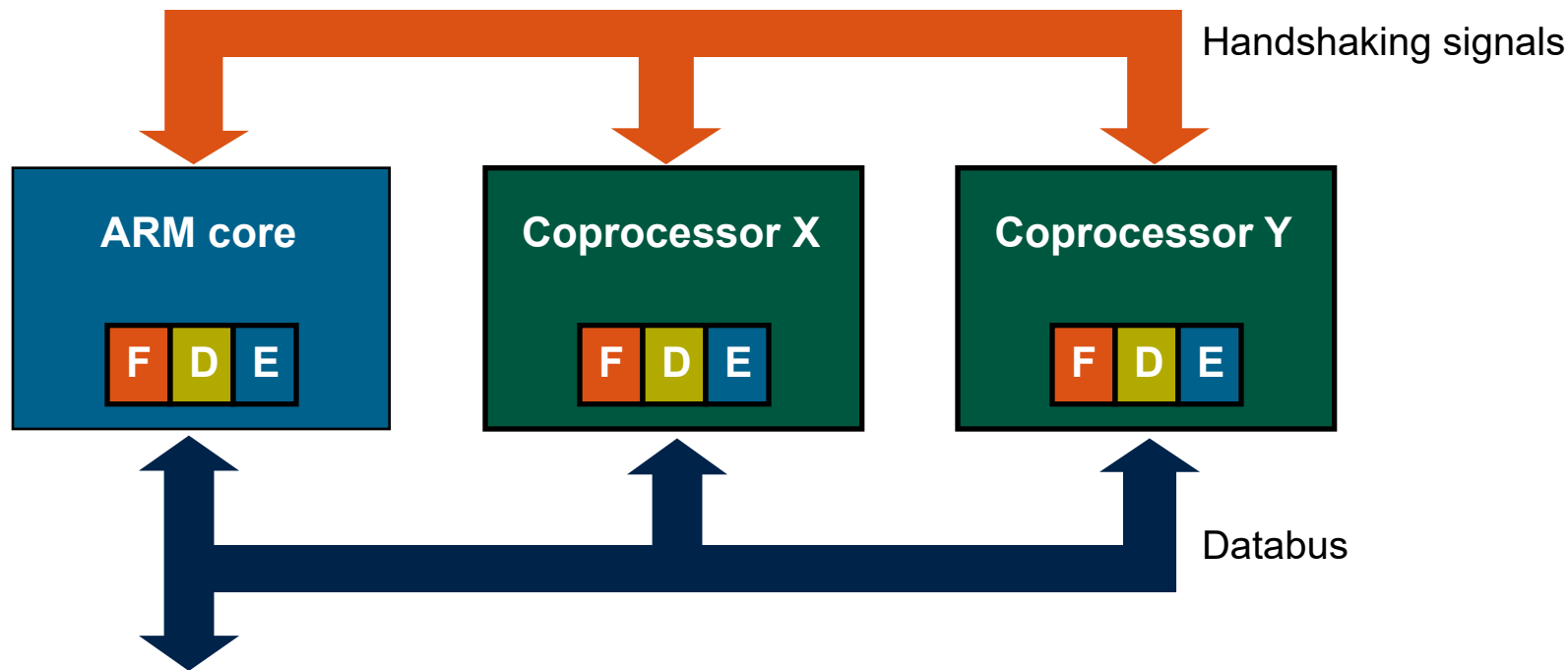
考到了算我输

- ARM内核的存储器管理硬件有3种不同类型，适应不同需要。
 - 1、无硬件保护
 - 2、提供有限保护 (MPU)
 - 3、提供全面保护的存储器管理单元 (MMU)
- 其中第三种提供全面保护的存储器管理单元 (MMU) 是ARM上应用最广泛的存储器管理硬件。具有如下特点：
 - 1、使用一组转化表，提供存储器控制
 - 2、这些表保存在主存里，提供虚拟地址和物理地址的映射和访问权限。
- 采用MMU可提高系统性能和运行速度。

你尽管复习



考到了算我输

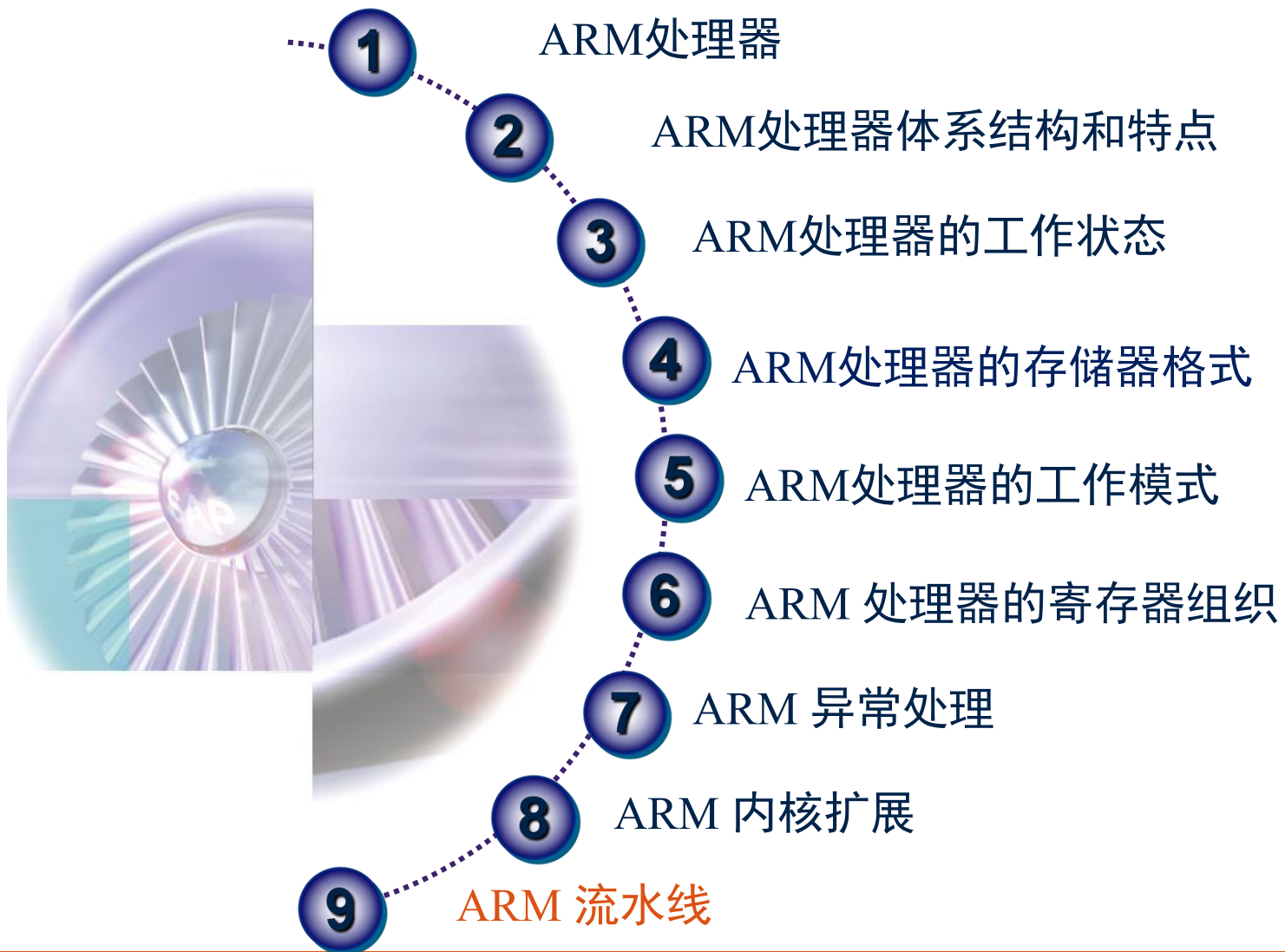


- 多达16个可定义协处理器,用唯一的ID来标示
- 扩充ARM指令集: 浮点运算等
- 通常用作ARM “internal functions” (例如: cp15通常 用作 ARM cache 控制器)
- 通常系统设计的时候最好用内存映射外设
 - 容易实现

你尽管复习



考到了算我输



- ✓ 流水线是RISC处理器执行指令时所采用的机制。使用流水线可以在取下一条指令的同时译码和执行其他指令，从而加快了执行速度。

三级流水线

- 下图显示了采用流水线技术后指令的执行过程：一条指令有3个时钟周期的执行时间，但吞吐量是每个周期1条指令。

I1 取指令	I1 译码	I1 执行		
	I2 取指令	I2 译码	I2 执行	
		I3 取指令	I3 译码	I3 执行

优秀的流水线结构

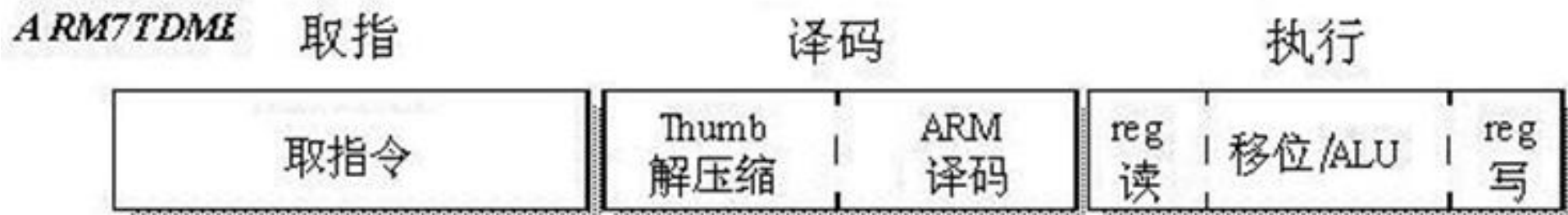
Cycle		1	2	3	4	5	6
Operation							
ADD	Fetch	Decode	Execute				
SUB		Fetch	Decode	Execute			
MOV			Fetch	Decode	Execute		
AND				Fetch	Decode	Execute	
ORR					Fetch	Decode	Execute
EOR						Fetch	Decode
CMP							Fetch
RSB							

ARM7的三级流水线

1取指：从程序存储器中取指令，放入指令流水线。(占用存储器访问操作)

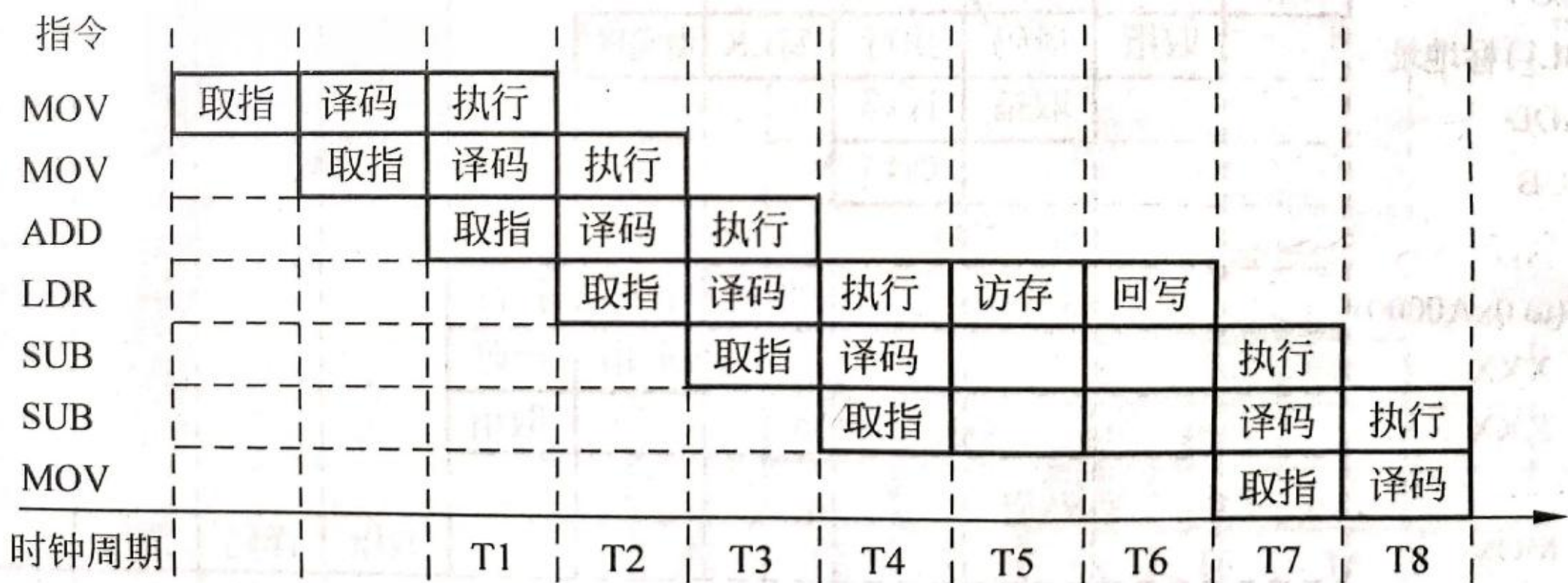
2译码：指令译码。(占用译码逻辑)

3执行：执行指令/读写REG。(占用ALU及寄存器)



ARM7采用的是冯诺依曼结构。

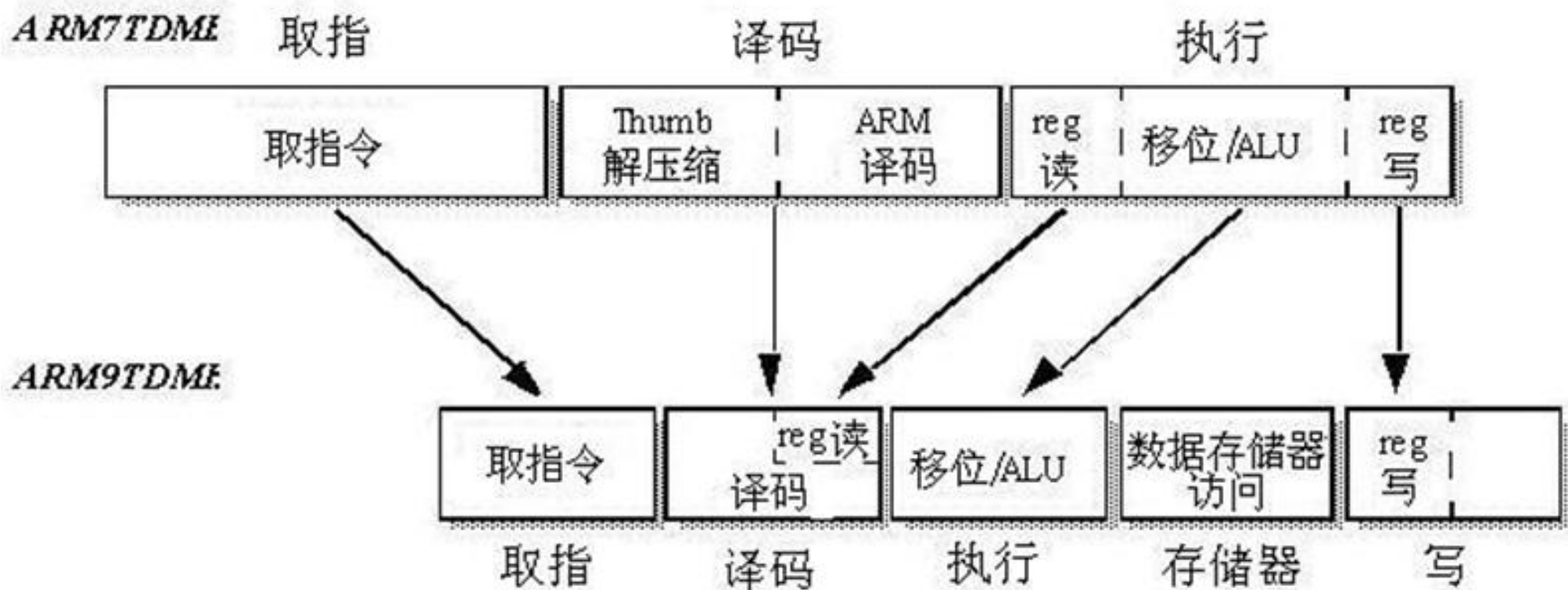
带存储器访问指令的流水线:



五级流水线

- ARM9具有5级流水线结构，指令分5个阶段运行，分别为：取值、译码、执行、存储器访问、寄存器写操作。

ARM9TDMI的五级流水线，采用哈佛结构



图：ARM7TDMI与ARM9TDMI流水线比较

ARM9带存储器访问的流水线结构



随着流水线深度（级数）的增加，每一段的工作量被削减了，这使得处理器可以工作在更高的频率，同时改进了处理器的性能。

- ✓ ARM处理器的体系结构：V1~V8
- ✓ ARM处理器的工作状态：ARM和Thumb
- ✓ ARM处理器的存储器格式：大端和小端格式
- ✓ ARM处理器的工作模式：7种工作模式
- ✓ ARM处理器的异常处理：异常的响应过程和返回过程
- ✓ ARM处理器的流水线：ARM7的3级流水和ARM9的5级流水

ARM[®]

THE ARCHITECTURE
FOR THE DIGITAL WORLD[™]