清華大學
Tsinghua University

《嵌入式系统》

3. ARM

清華大學
Tsinghua University

指令集体系结构

RISC 和 CISC

Great Microprocessors of the Past and Present (V 13.4.0)
http://www.cpushack.com/CPU/cpu.html

Instruction Set Architecture (ISA)

- ISA（指令集体系结构）规定了**如何使用硬件**，可执行的指令集合，包括**指令格式，操作种类**，指令接受的**操作数类型**，操作数所存放的寄存器组和结构，包括每个**寄存器名称、编号、长度和用途等**，操作数所存放的**存储空间大小和编址方式，操作数存储方式（大端/小端），寻址方式，指令执行的控制方式**，包括程序计数器，条件码定义等。

ISA类型：

CISC：Complex Instruction Set Computer：x86，IBM370

RISC：Reduced Instruction Set Computer：Sparc

VLIW: Very long instruction word: IA-64, TMS320C6000 DSP

EPIC：Explicitly Parallel Instruction Computing: Itanium

CISC的背景和特点

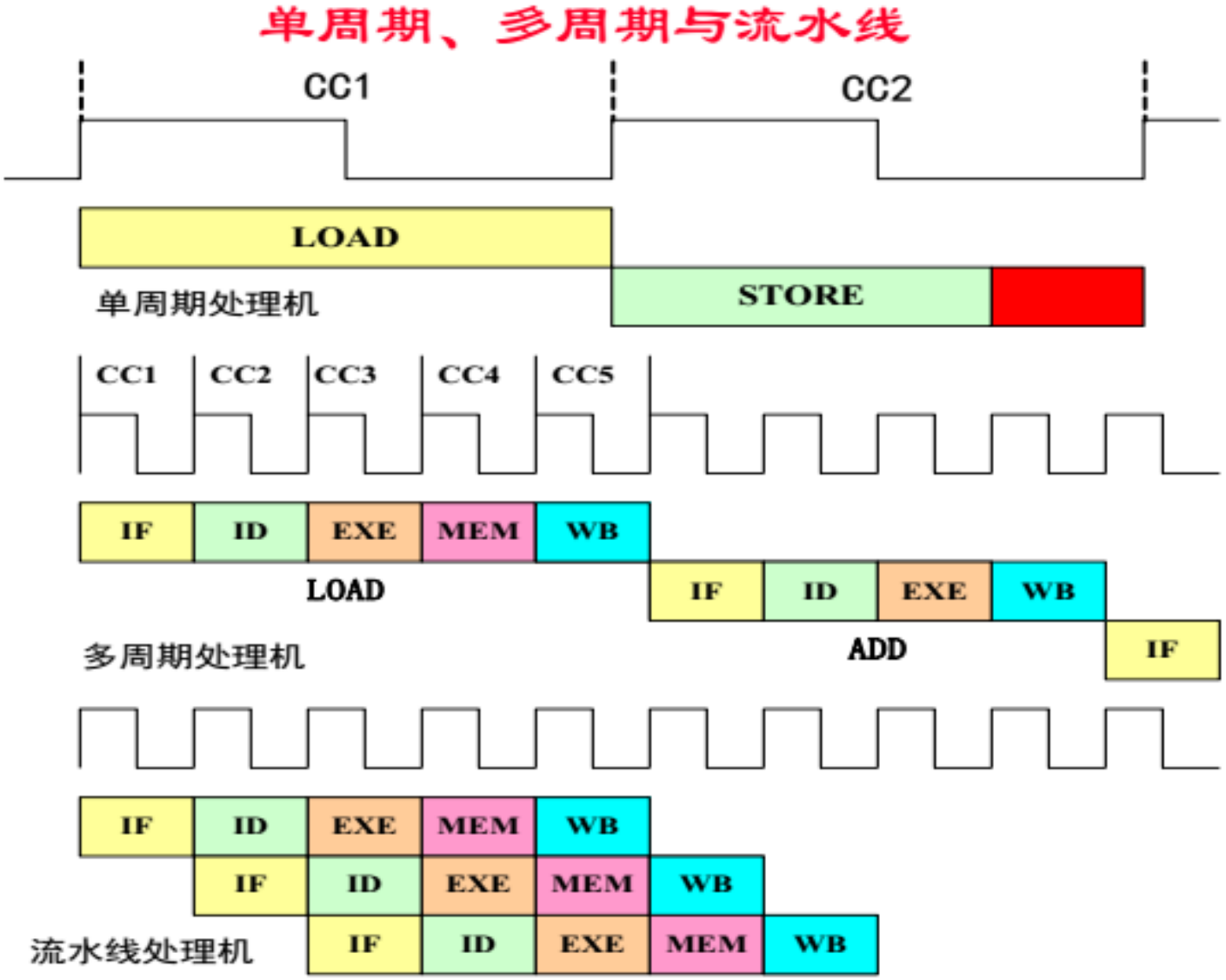
- 背景：硬件比软件快；存储资源紧缺。
- 增强指令功能，设置一些功能复杂的指令，把原来由软件实现的功能改用硬件（微程序）指令系统实现。采用微程序控制，执行每条指令均需完成一个微指令序列。
- 为节省存储空间，强调高代码密度。指令格式不固定，指令可长可短，操作数可多可少。
- 寻址方式复杂多样，操作数可来自寄存器或存储器。
- 指令复杂，CPI（clock per instruction）大。

CISC的主要缺点

- 复杂指令的控制逻辑不规整，不适于VLSI工艺，后者利于实现规整的硬联逻辑，而非微程序。 微程序的使用制约了CPU速度提高。
- 不利于先进指令级并行技术的采用：流水线技术
- 指令使用频度不均衡：高频度使用的指令占据了CPU执行时间的大部分，扩充的复杂指令往往是低频度指令，例如：双精度右移SHRD(Shift Right Double)

assignment:	47%
if:	23%
call:	15%
loop:	6%
goto:	3%
Other:	6~7%

流水线技术（Pipeline）



RISC的提出与发展

- Load/Store结构：CPU只对寄存器中数据进行运算。
- RISC思想最早由IBM提出，IBM801处理器是首个RISC处理器，它有120条指令。
- 1980年，Berkeley的Patterson（2017图灵奖）和Dizel提出RISC名词，并研制了RISC-I, II实验样机。
- 1981年Stanford的Hennessy （2017图灵奖）研制 MIPS芯片。

RISC基本设计思想

- 精简指令集：保留最基本的指令，去掉复杂、使用频度不高的指令；
- 采用Load/Store结构，统一存储器访问方式；
- 采用多级指令流水线结构以减小CPI；

$$CPUtime = Instr\_Count * CPI * Clock\_cycle;$$

- 多寄存器；
  - 有别于CISC处理器往往有一个功能强大但是复杂的寄存器，处理几乎所有运算；RISC处理器中有几十个（典型的数字是32个）寄存器，每个（或多数）寄存器都能进行全部指令的运算。

一条RISC指令的5个时钟周期

- **IF（取指）**：根据PC（程序计数器）中所存储的内存地址，在内存中找到该地址所指向的指令，并将该指令存储在寄存器中。同时，PC指向下一条指令。
- **ID （译码）**：操作从IF阶段获取来的指令。将指令解码，最终找到指令所需要的寄存器中存储的数据。如果该指令只是一条跳转指令，那么在这一阶段需要根据跳转指令的意义对获取的值进行比较，如果比较结果为true则执行跳转，如果比较结果为false则不执行跳转，继续下一条指令的执行；如果指令需要对指令中某些位进行填充，也在ID阶段完成，比如对高四位进行填充以满足指令结果是32位；计算可能跳转的指令的地址。

一条RISC指令的5个时钟周期（续）

- **EX（执行）**：ALU对ID阶段的结果进行计算。在ID阶段已经获得了指令计算所需要的寄存器的值，那么在EX阶段需要根据指令的意义对这些寄存器的值进行计算。计算根据指令的不同变得不同。主要有三种类型的ALU计算：1. ALU根据ID中补充的地址，对有效的地址单元进行计算，最终得到所需要的内存的地址；2. 根据指令的意义，对从寄存器中获取的值，进行操作，比如对两个寄存器的值进行相加；3. 根据寄存器的值以及补充的值，计算出立即数的结果。
- **MEM（访存）**：如果当前指令是Load指令，那么，根据EX计算出的内存地址，从内存中获取对应的值；如果当前指令是store，那么，根据EX计算出的内存地址和寄存器的值，将寄存器的值存入该内存地址中。其他的指令一般不会涉及内存的访问。
- **WB（写回）**：将计算出来的最终的寄存器的值写入到register file（寄存器文件）中。这部操作包括从内存中获取的值以及通过算术运算得到的结果。

典型的RISC处理器

- SUN公司的SPARC(1987)
- MIPS公司的MIPS(1986)
- IBM、Motorola公司的PowerPC
- HP公司的PA-RISC
- DEC、Compaq公司的Alpha AXP
- IBM RS6000(1990): 第一台Superscalar RISC机

CISC与RISC的对比

类别	CISC	RISC
指令系统	指令数量多	指令数量少，通常少于100
执行时间	有些指令执行时间长，如将多个寄存器的内容拷贝到存储器	简单指令仅需要1个指令周期（5个时钟周期）
编码长度	编码长度可变，1-15字节； 不利于流水线实现	编码长度固定，通常为4个字节； 方便流水线实现
寻址方式	寻址方式多样	寻址方式较为简单
内存访问	Register/Memory体系结构，许多指令可直接操作内存	Load/Store体系结构，只能对寄存器进行算术和逻辑操作
编译器及硬件	编译器简单； 处理器硬件(如微程序)复杂；	编译器复杂； 处理器硬件简单；

参考文章：Analysis:x86 Vs PPC和CISC vs RISC

ARM


优酷
ARM Powered®
Journey Home

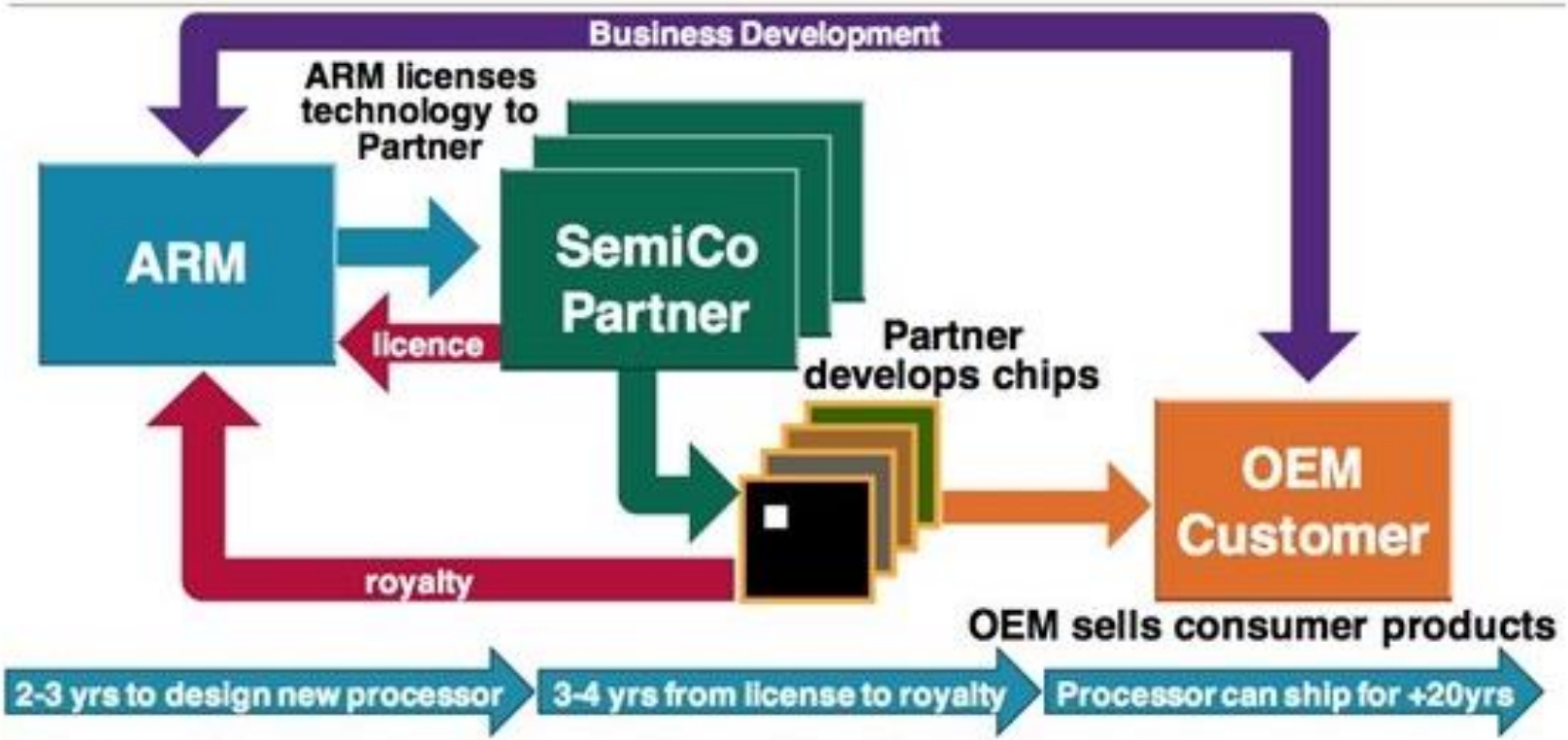
ARM公司

- ARM公司成立于1990多年，总部在英国剑桥，全称 **Advanced RISC Machines**。
- 在1999年因移动电话火爆市场，其32位RISC处理器占市场份额超过了50%。
- ARM公司商业模式的强大之处在于其价格合理，全世界范围有超过100个合作伙伴，包括半导体工业的著名公司：TI、意法半导体、Atmel、Philips、Intel、三星等
- ARM公司专注于设计，其内核耗电少、成本低、功能强，**特有16/32位双指令集**。ARM已成为移动通信、手持计算、多媒体数字消费等嵌入式解决方案的RISC标准。

ARM的发展策略

- ARM的合作伙伴分为三种：
  - EDA（电子设计自动化）伙伴计划，融合了ARM在线系统设计、可重复使用IP生成和IP模型等方面的专长与领先EDA工具厂商的专业特长，开发SoC产品。
  - ARM技术共享计划，为ARM与外部设计服务公司合作关系的扩展和规范化结果，现在已经有2900多工程师成员。
  - 制造商计划，使新兴市场的OEM能分享ARM处理器技术，用于设计和制造先进的SoC解决方案。

ARM 商业模式



- Innovative business model yields high margins
  - Upfront license fee – flexible licensing models
  - Ongoing royalties – typically based on percentage of chip price
  - Technology suitable for multiple applications – can ship for decades

ARM的三种IP授权方式

- **处理器授权**
  - 授权合作厂商使用ARM设计好的CPU或者GPU处理器，对方不能改变原有设计，但可以根据自己的需要使用。
- **处理器优化包授权**
  - 处理器授权的高级形式，当用户驾驭不了ARM处理器，那么ARM也可以出售优化后的处理器，这样用户就能在特定工艺下设计、生产出性能有保证的处理器。
- **架构授权**
  - ARM会授权对方使用自己的架构，对方可以根据自己的需要来设计处理器，这些处理器跟ARM自己设计的处理器是兼容的，但是有用户自己的实现方式。

ARM--不靠卖处理器挣钱的公司

- 收入来源：
  - 技术授权费：50%，每笔100万-1000万美元，也可能更高
  - 版税提成：33%，按比例收取(典型的)1-2%
  - 其他软件工具和技术支持：<17%
- 毛利率高达94%，运营利润在45%左右。
- 移动设备市场份额的90%以上，数字家电市场份额的80%以上。

阅读材料：ARM架构与ARM处理器对应关系

序号	ARM微处理器核心	体系结构版本
1	ARM1	v1
2	ARM2	v2
3	ARM2As，ARM3	v2a
4	ARM6，ARM600，ARM610，ARM7，ARM700，ARM710	v3
5	StrongARM，ARM8，ARM810	v4
6	ARM7TDMI，ARM710T，ARM720T，ARM740T，ARM9TDMI，ARM920T，ARM940T	v4T
7	ARM9E-S，ARM10TDMI，ARM1020E	v5TE
8	ARM1136J(F)-S，ARM1176JZ(F)-S，ARM11 MPCore	v6
9	ARM1156T2(F)-S	v6T2
10	ARM Cortex-M，ARM Cortex-R，ARM Cortex-A	v7

V1版架构

该版架构只在原型机ARM1出现过,其基本性能：

- 基本的数据处理指令(无乘法)
- 字节、半字和字的LOAD/STORE指令
- 转移指令,包括子程序调用及链接指令
- 软件中断（SWI）指令
- 寻址空间：64M字节(26位)

V2版架构

该版架构对V1版进行了扩展，如ARM2与ARM3 (V2a版)架构，增加了以下功能：

- 乘法和乘加指令
- 支持协处理器操作指令
- 快速中断模式
- SWP/SWPB的最基本存储器与寄存器交换指令
- 寻址空间：64M字节（还是26位）

V3版架构

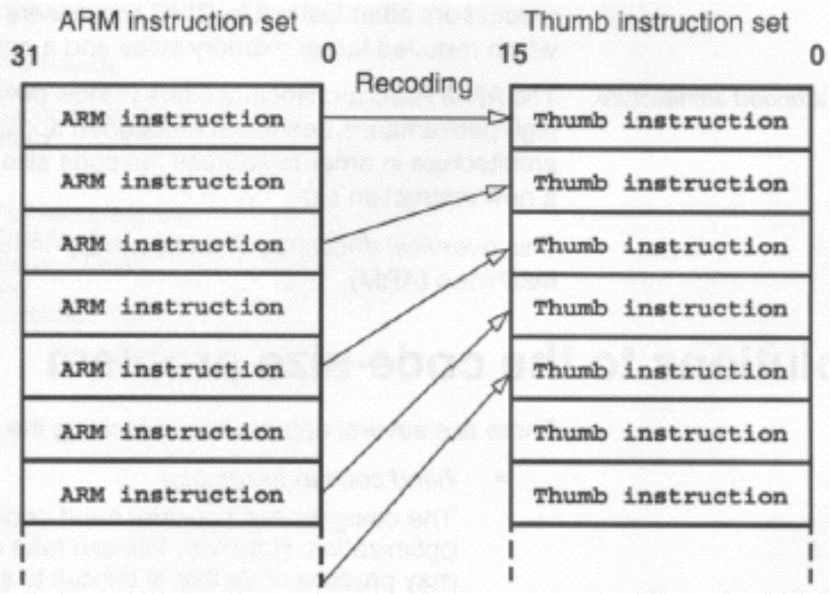
- 把寻址空间增至32位(4G字节)
- 增加了当前程序状态寄存器CPSR (Current Program Status Register)，存储当前运行程序的状态量，如指令集结构、系统工作模式等
- 增加了程序状态保存寄存器SPSR (Saved Program Status Register)，用于被异常中断时保护现场
- 增加了中止(Abort)和未定义二种处理器模式。ARM6就采用该版架构。指令集变化如下：
  - 增加了MRS/MSR指令，以访问新增的CPSR/SPSR寄存器
  - 增加了从异常处理返回的指令功能。

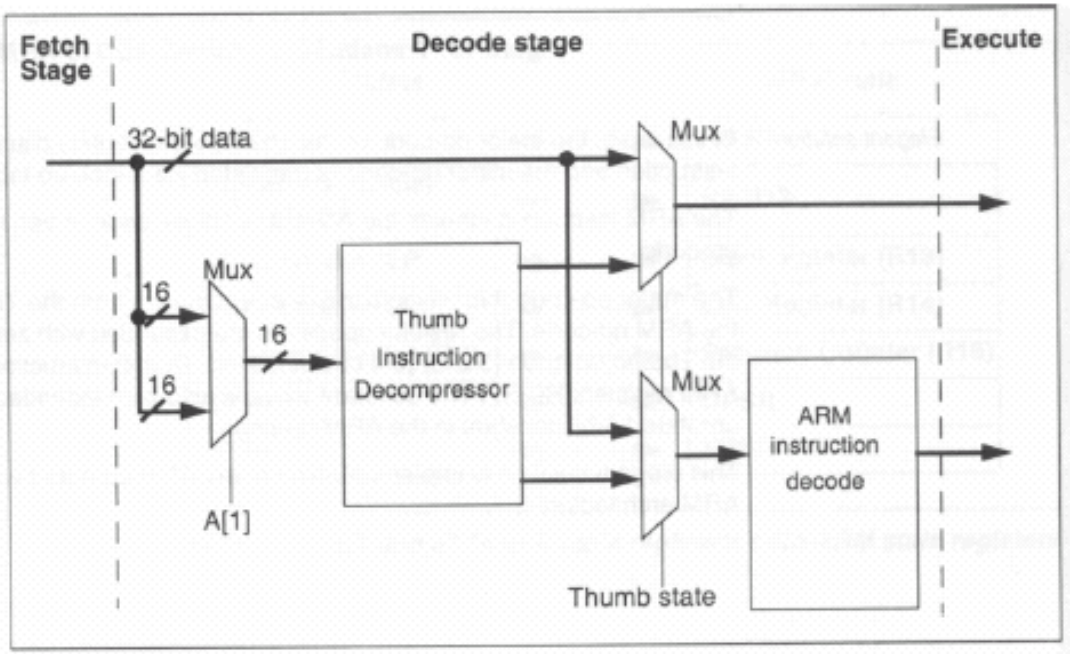
V4版架构

- V4版架构是曾经应用最广的ARM体系结构，对V3版架构进行了进一步扩充。
- 引进了16位的Thumb指令集，使ARM使用更加灵活。
- ARM7、ARM8、ARM9和StrongARM都采用该版架构。指令集中增加了以下功能：
  - 符号化和非符号化半字及符号化字节的存/取指令
  - 增加了16位Thumb指令集
  - 完善了软件中断SWI指令的功能
  - 处理器系统模式引进特权方式时使用用户寄存器操作
  - 把一些未使用的指令空间捕获为未定义指令

THUMB指令集(‘T’)

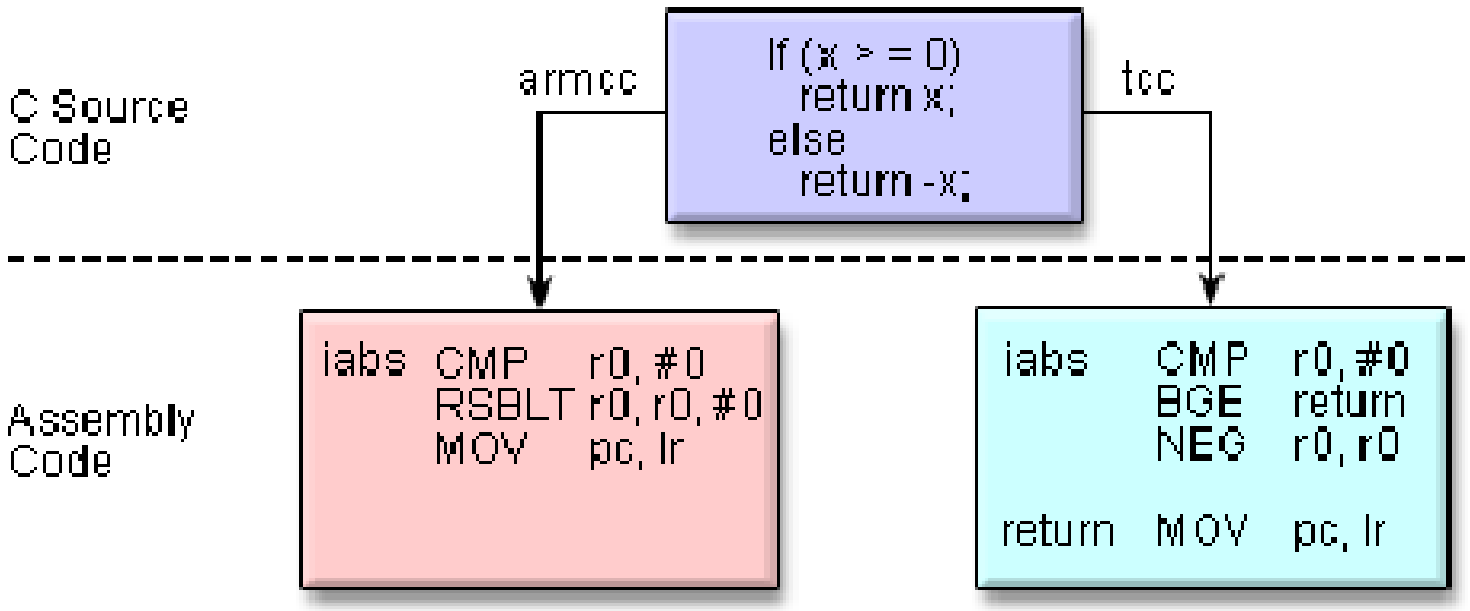
THUMB 指令集: 32位ARM指令集的子集，按16位指令重新编码





- 代码尺寸小，存储效率高(up to 40 % compression)
- 简化设计

ARM vs Thumb



Comparison

Code	Instruction	Size (Bytes)	Normalized
ARM	3	12	1.0
Thumb	4	8	0.67

清華大學 Tsinghua University

处理器的操作状态

- ARM7TDMI 处理器有两种操作状态:
  - ARM - 32-bit, 按字排列的ARM指令集
  - Thumb -16-bit, 按半字排列的Thumb指令集
- ARM7TDMI 核的操作状态可能通过BX指令(分支交换指令)在ARM状态和Thumb状态之间切换

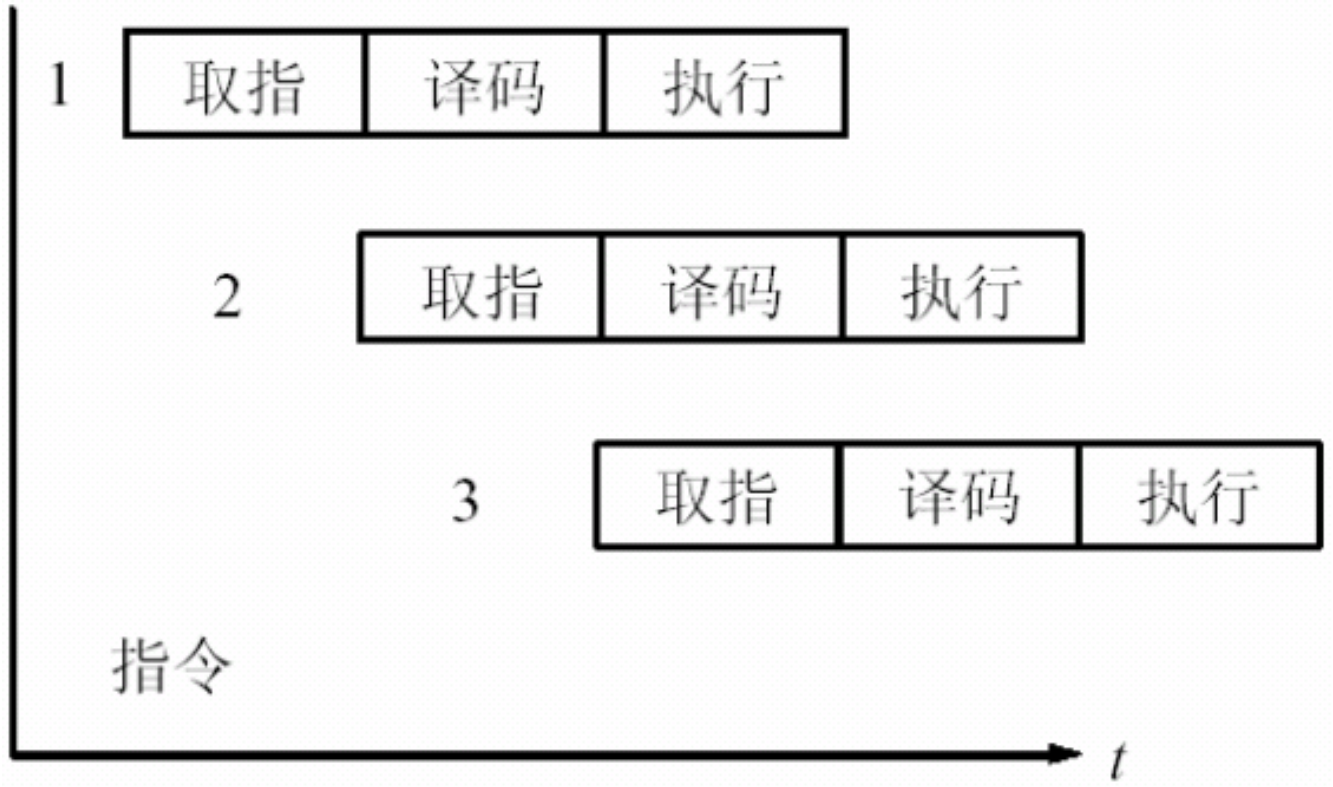
从ARM状态切换到Thumb状态：

```
LDR R0,=Label+1
BX R0
```

从Thumb状态切换到ARM状态：

```
LDR R0,=Label
BX R0
```

ARM7TDMI的3段流水线操作



注:程序计数器PC指向正在取指的指令而不是正在执行的指令。

V5版架构

- 在V4版基本上增加了一些新的指令,ARM10和XScale都采用该版架构,这些新增指令有:
  - 带有链接和交换的转移BLX指令
  - 计数前导零CLZ指令，提高整数除法和重点优先级排队操作的效率
  - BRKT中断指令（软中断）
  - 增加了信号处理指令(V5TE版)
  - 为协处理器增加更多可选择的指令

V6版架构

- 在芯片功耗和多媒体处理上有所突破。
  - 增加多媒体扩展（SIMD变种）功能；
  - 在语音、视频、图像上的处理能力提高4倍；
  - 使用先进工艺使得芯片的功耗进一步降低；
  - 该版本可以广泛应用于移动电话、PDA等手持设备；
  - 改进了的混合端(Endian)与不对齐数据支持， 使得小端系统支持大端；

小端和大端

- 小端： Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
- 大端： Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

0x12345678

小端模式：低地址 ----------> 高地址
0x78 | 0x56 | 0x34 | 0x12

大端模式：低地址 ----------> 高地址
0x12 | 0x34 | 0x56 | 0x78

V7架构

- 定义了三大系列：
  - “A”：面向尖端的基于虚拟内存的操作系统和用户应用，Cortex-A
  - “R”：针对实时系统，Cortex-R
  - “M”：对微控制器和低成本应用提供优化，Cortex-M。

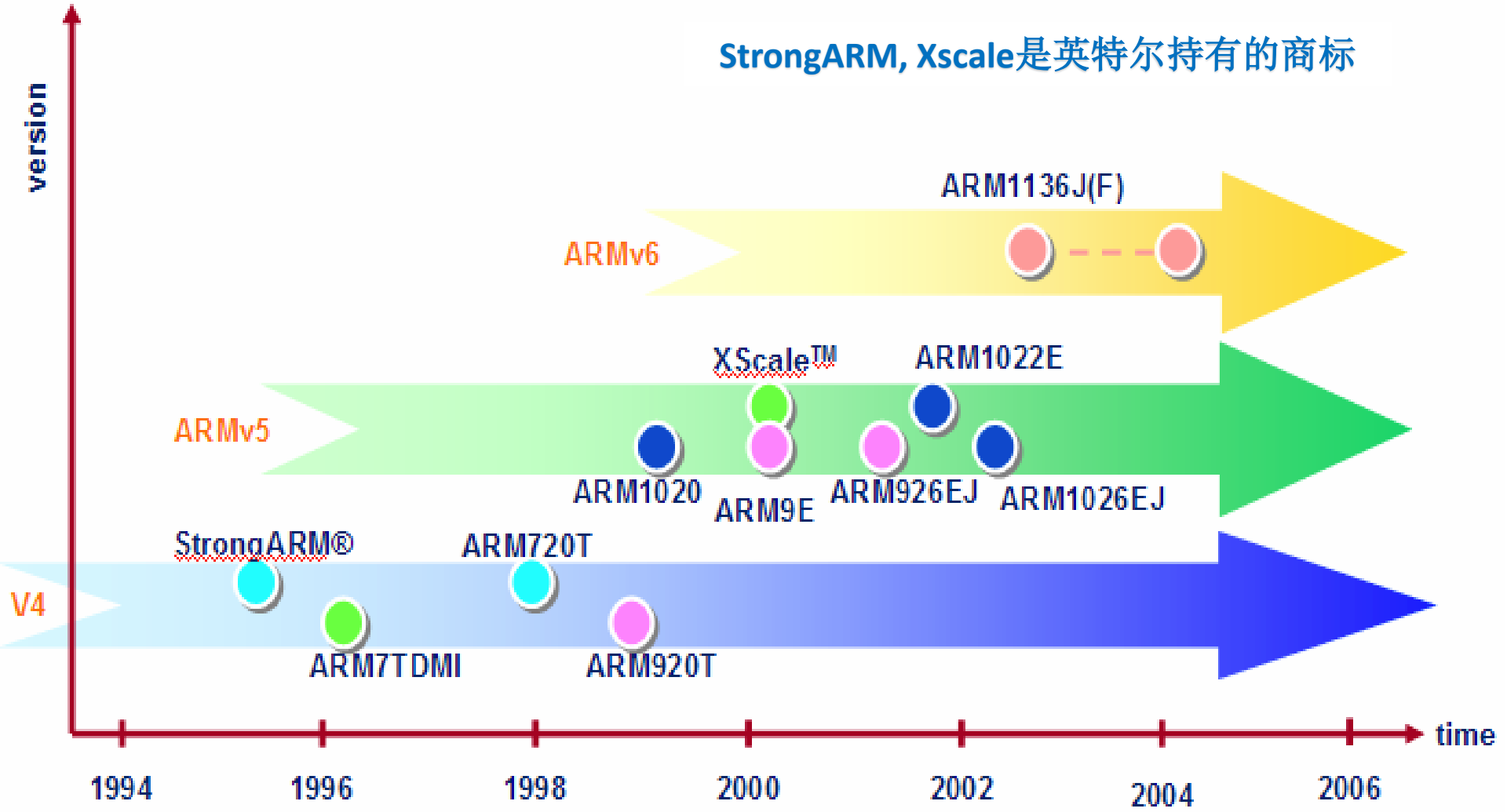
ARM内核命名规则（V7以前）

- 命名规则如下“ARM [x] [y] [z] [T] [D] [M] [I] [E] [J] [F] [-S]”， [x]表示系列号， [y]表示内存存储管理和保护单元， [z]表示含有高速缓存。

后缀变量	功能描述
T	Thumb指令集， Thumb指令长度为16位， 目前有两个版本， Thumb1用于ARM V4的T变种， Thumb2用于ARM V5以上的版本
D	含有JTAG调试， 支持片上调试
M	内嵌硬件乘法器(Multiplier)， 提供用于进行长乘法操作的ARM指令， 产生全64位结果
I	内嵌是在线测试宏单元(Embedded ICE Macrocell)硬件部件， 提供片上断点和调试点支持。
E	增强型DSP指令， 增加了几条16位乘法和加法指令， 加减法指令可以完成饱和带符号算数运算
J	Java加速器Jazelle， 与普通的Java虚拟机相比， Jazelle是Java代码运行速度提高了8倍， 而功耗降低了80%
F	向量浮点单元
S	可综合版本， 以源代码形式提供的， 可以被EDA工具使用

ARM结构体系和处理器家族的演变发展

StrongARM, Xscale是英特尔持有的商标



ARM处理器

- ARM7系列，3级流水线，系统主时钟为20MHz到133MHz，适用于价位低、功耗低的消费类应用。无MMU，可运行uCLinux。
- ARM9系列，5级流水线，系统主时钟为100MHz到233MHz，提供全性能的MMU (Memory Management Unit)，支持Windows CE、Linux等主流操作系统。
- ARM9E系列，使用单一的处理器内核，支持VFP9浮点处理协处理器，提供了微控制器、DSP、Java应用系统的解决方案。

ARM处理器

- ARM10E系列，支持VFP10浮点处理协处理器，时钟频率则可以高达400MHz，性能大幅提升，并功耗极低。
- ARM11系列，8级流水线，时钟频率可达350MHz到1GHz，媒体处理能力强，功耗低。
- Cortex主要分为三个系列
  - Cortex-A 面向高性能，13级流水线，多核
  - Cortex-R面向高实时，处理能力强，低能耗
  - Cortex-M面向微控制器，低延迟/能耗/价格

如何对ARM处理器进行选型（1）

- 操作系统需求
  - ARM720T/ARM920T/ARM922T/ARM946T:有MMU,支持Linux,WinCE
  - ARM7TDMI:没MMU，uCLinux、VxWorks、uC/OS
- 系统时钟控制器
  - ARM7:20~133 MHZ
  - ARM9:100~233 MHZ
  - ARM10:700 MHZ
  - Cirrus Logic的EP7312: 只有一个主时钟频率，不能顾及UART和音频时钟的准确性。
  - Philips的SAA7750: 分别为CPU核、UART/DSP/音频提供同频率时钟。

如何对ARM处理器进行选型（2）

- USB接口
  - USB控制器
  - USB Host
  - USB Slave
- GPIO
  - 数量的多少
  - 是否和地址线、数据线、串口线等引线复用
- 中断控制器
  - SAA7550: 所有GPIO都可以设置成FIQ或IRQ，可以选择上升沿、下降沿、高电平、低电平四种中断方式，使得IRDA、键盘等任务可以作为背景程序运行。

如何对ARM处理器进行选型（3）

- Cirrus Logic的EP7312：只有四个外部中断源，每个中断源只能是低电平或高电平中断，需用轮询方式。

- IIS(Integrate Interface of Sound)接口
  - 便于设计音频应用产品。

- nWAIT信号
  - 与GAL芯片结合实现符合PCMCIA标准的WLAN卡和Bluetooth卡的接口，成本比PCMCIA专业控制芯片低。
  - 可以扩展外部DSP协处理器。

- LCD控制器
  - 设计手持式显示记录设备时，选用内置LCD控制器的ARM芯片较为合适。如S1C2410

如何对ARM处理器进行选型（4）

- 电源管理
  - 低功耗模式
  - 睡眠模式
- DMA控制器
  - 可以快速和外部设备交换数据，如硬盘等。
- 封装
  - 主要有QFP/TQFP/PQFP/LQFP/BGA/LBGA等形式。
  - BGA封装芯片面积小，但需要专用焊接设备，无法用双面板完成PCB布线，需要多层PCB板布线。

如何选择多芯核结构ARM芯片

- 多ARM核 （ARM + ARM）
  - Portal Player公司的PP5002： 集成两个ARM7TDMI,用于MP3。
  - MinSpeed公司多款通信芯片：集成2~4个ARM7TDMI内核。
- ARM + DSP
  - ARM公司的Piccolo DSP芯核
  - OAK公司的16位定点DSP芯核
  - TI公司的TMS320C5000系列DSP芯核
  - Motorola公司的56K DSP核
- ARM + FPGA

主要的ARM芯片提供商

- Intel
- Texas Instrument
- Samsung Semiconductor
- Freescale
- Philips Semiconductor
- Qualcomm
- Atmel
- Cirrus Logic
- 华为、中兴购买ARM芯核，设计自主版权专用芯片

Q:这是那种IP授权?

ARM9处理器架构 (1)

- ARM9系列处理器共有37个寄存器
  - 31个通用寄存器，6个专用状态寄存器（为不同工作模式设立）
- ARM总共有7种不同的处理器模式
  - 用户模式(User)，这是程序正常执行的模式
  - 特权模式
    - 系统模式(System)，用于运行特权级的操作系统任务
    - 异常模式
      - 快中断模式(FIQ)，用于高速数据传输和通道处理
      - 外部中断模式(IRQ)，用于普通的外部中断请求处理
      - 管理模式(Supervisor)，这是供操作系统使用的保护模式
      - 数据访问中止模式(Abort)，用于虚拟存储和存储保护
      - 未定义模式(Undef)，用于支持硬件协处理器软件仿真

清華大學
Tsinghua University

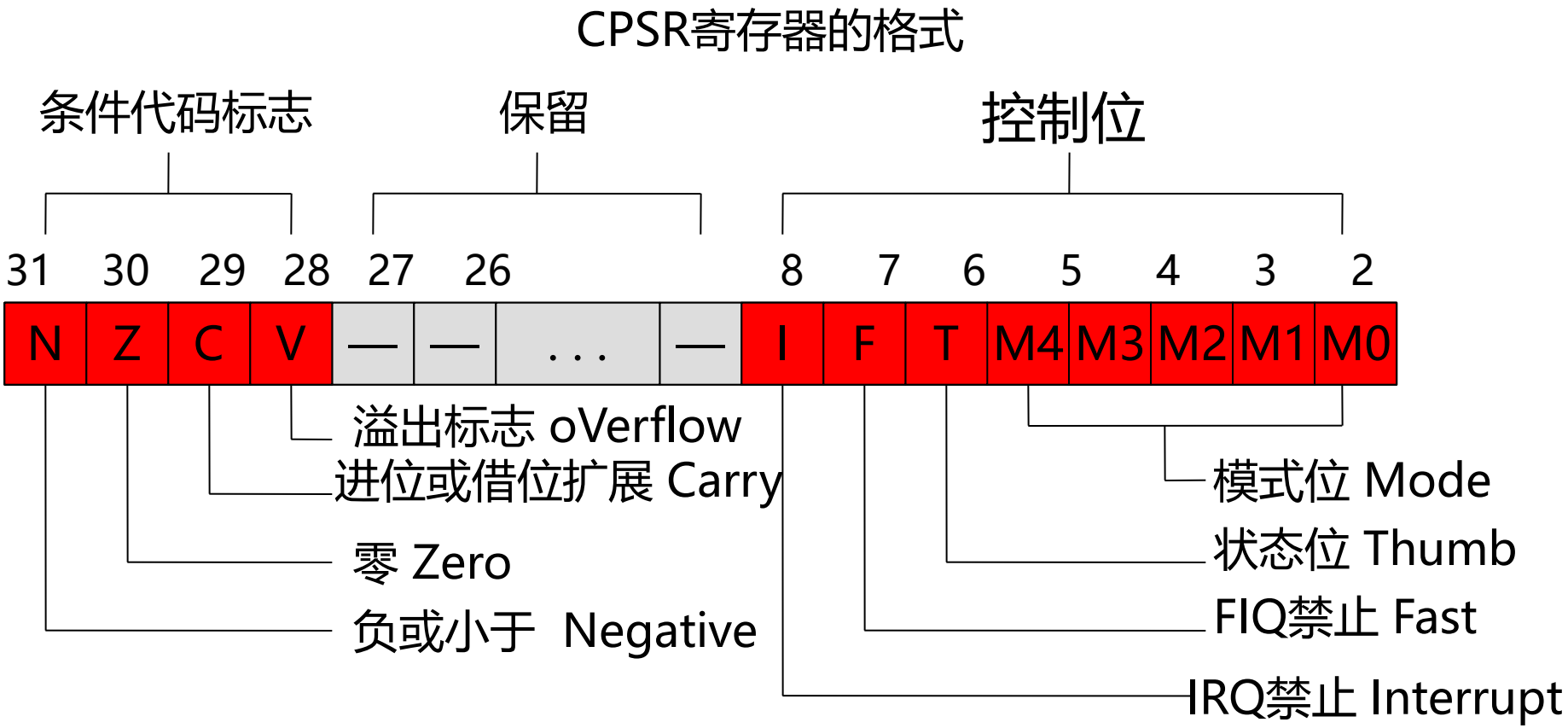
ARM体系的主要特征

- Load/Store体系结构
- 大量的寄存器，都可用于多种用途
- 三地址指令（两个源操作数寄存器和结果寄存器独立设定）
- 每条指令都可以条件执行，包含非常强大的多寄存器Load和Store指令
- 能在单时钟周期内完成1项普通移位操作和1项普通ALU操作
- 通过协处理器指令集来扩展ARM指令集，包括在编程模式下增加了新的寄存器和数据类型
- 在Thumb体系结构中以高密度16位压缩形式表示指令集

寄存器分配：通用寄存器

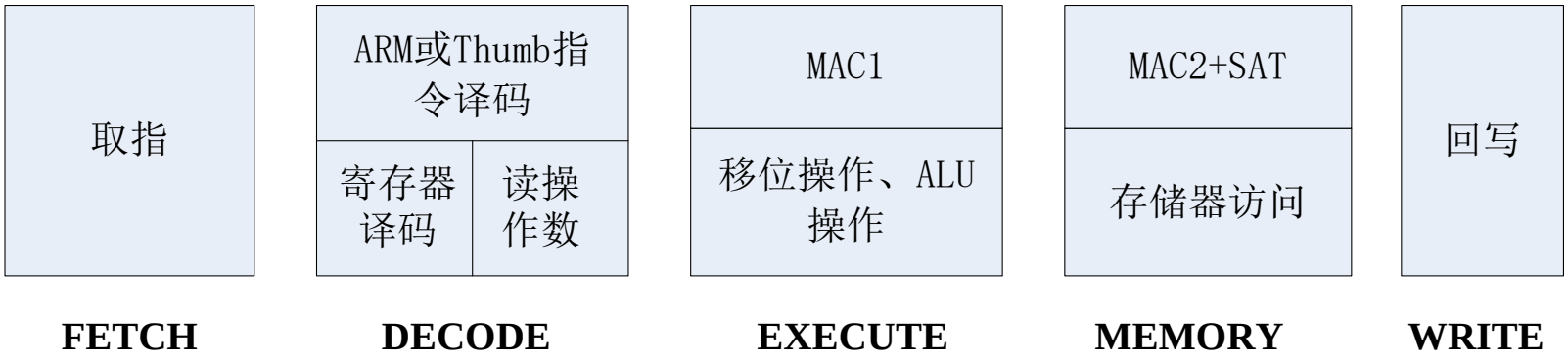
- R0 - R15：可以直接访问
- R0 - R14：通用寄存器
  - 0-7：未分组寄存器：所有模式下都是同一个物理寄存器
  - 8-14：分组寄存器
    - R8～R12，每个寄存器对应两个不同的物理寄存器
  - R13：堆栈指针(sp)
    - 通常每种处理器模式都有单独的堆栈
    - 每个寄存器对应6个不同的物理寄存器
  - R14：链接寄存器(Link Register，LR)
    - R15的备份，子程序的返回地址，异常返回地址
    - 不需要存放在堆栈中，函数调用速度快
    - 每个寄存器对应6个不同的物理寄存器
- R15：程序计数器(PC)
- R16，就是CPSR
  - 当前程序状态寄存器，包括代码标志状态和当前模式位
- 5个SPSRs(程序状态保存寄存器)
  - 当异常发生时保存CPSR状态

ARM的程序状态寄存器（CPSR）简介



ARM9处理器架构（2）

- 流水线技术
  - ARM9采用5级流水线技术，分别是FETCH、DECODE、EXECUTE、MEMORY、WRITE



- ARM9采用了哈佛体系结构

ARM9指令集分类

- ARM指令集属于Load/Store型指令。
- ARM9指令集分为6类
  - 跳转指令
  - 数据处理指令
  - 存储器访问指令
  - 协处理器指令
  - 杂项指令
  - 饱和算术指令

ARM体系结构的主要特征 - 三地址指令

ARM是三地址指令格式，指令的基本格式如下：

<opcode> {<cond>} {S} <Rd> ,<Rn>{,<operand2>}

其中<>号内的项是必须的，{}号内的项是可选的。各项的说明如下：

opcode：指令助记符； cond：执行条件；

S：是否影响CPSR寄存器的值；

Rd：目标寄存器； Rn：第1个操作数的寄存器；

operand2：第2个操作数；

例：

指令语法	目标寄存器(Rd)	源寄存器1(Rn)	源寄存器2(Rm)
ADD r3, r1, r2	r3	r1	r2

ARM体系结构的主要特征 - 条件执行

❑ARM指令集——条件码

ARM指令的基本格式如下：

<opcode> {<cond>} {S} <Rd> ,<Rn>{,<operand2>}

使用条件码“cond”可以实现高效的逻辑操作(节省跳转和条件语句)，提高代码效率。

所有的ARM指令都可以条件执行，而Thumb指令只有B（跳转）指令具有条件执行 功能。如果指令不标明条件代码，将默认为无条件（AL）执行。

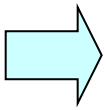
ARM体系结构的主要特征 - 条件执行

- ARM指令集——条件码

示例：

C代码：

```
If(a > b)
  a++;
Else
  b++;
```



对应的汇编代码：

```
CMP    R0,R1      ;R0（a）与R1（b）比较
ADDHI  R0,R0,#1   ;若R0>R1，则R0=R0+1
ADDLS  R1,R1,#1   ;若R0≤R1，则R1=R1+1
```

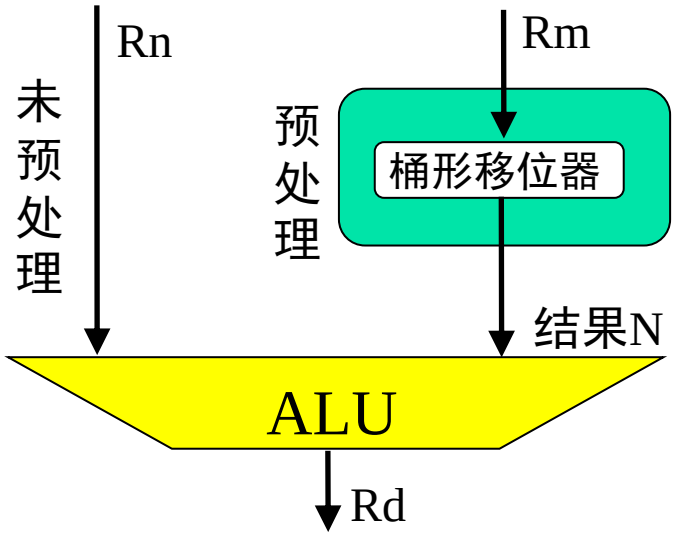
ARM体系结构的主要特征 - 单指令移位操作

- ARM指令集对第2个操作数，提供单指令的移位操作功能

第2个操作Rm如果是下面的形式，

Rm，shift

则表示使用的是寄存器移位方式：将寄存器的移位结果作为操作数（移位操作不消耗额外的时间），但Rm值保持不变，移位方法如下：



寄存器移位方式举例：

```
ADD   R1,R1,R1,LSL #3     ;R1=R1+(R1<<3)
SUB   R1,R1,R2,LSR R3     ;R1=R1-(R2>>R3)
```

ARM处理器寻址方式

❑寻址方式分类

寻址方式是根据指令中给出的地址码字段来实现寻找真实操作数地址的方式。ARM有7种基本寻址方式。

1.寄存器寻址；　　2.立即寻址；

3.寄存器移位寻址；　　4.寄存器间接寻址；

5.基址寻址；　　6.相对寻址。

7.多寄存器寻址；

ARM处理器寻址方式

❑寻址方式分类——寄存器寻址

操作数的值在寄存器中，指令中的地址码字段指出的是寄存器编号，指令执行时直接取出寄存器值来操作。寄存器寻址指令举例如下：

```
MOV  R1,R2        ;R1 := R2
SUB  R0,R1,R2     ;R0 := R1 - R2
```

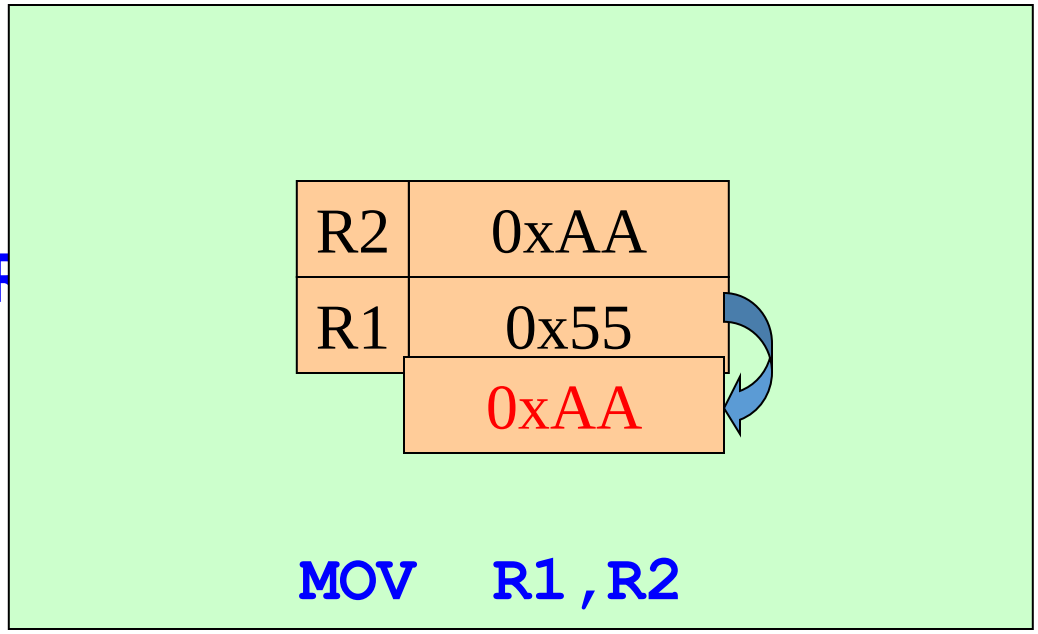
ARM处理器寻址方式

❑寻址方式分类——寄存器寻址

操作数的值在寄存器中，指令中的地址码字段指出的是寄存器编号，指令执行时直接取出寄存器值来操作。寄存器寻址指令举例如下：

MOV R1,R2

SUB R0,R1,R



ARM处理器寻址方式

❑寻址方式分类——立即寻址

立即寻址指令中数据就包含在指令当中，取出指令也就取出了可以立即使用的操作数(这样的数称为立即数)。立即寻址指令举例如下：

```
SUBS R0,R0,#1      ;R0--，并且影响标志位
MOV  R0,#0xFF000   ;将立即数0xFF000装入R0寄存器
```

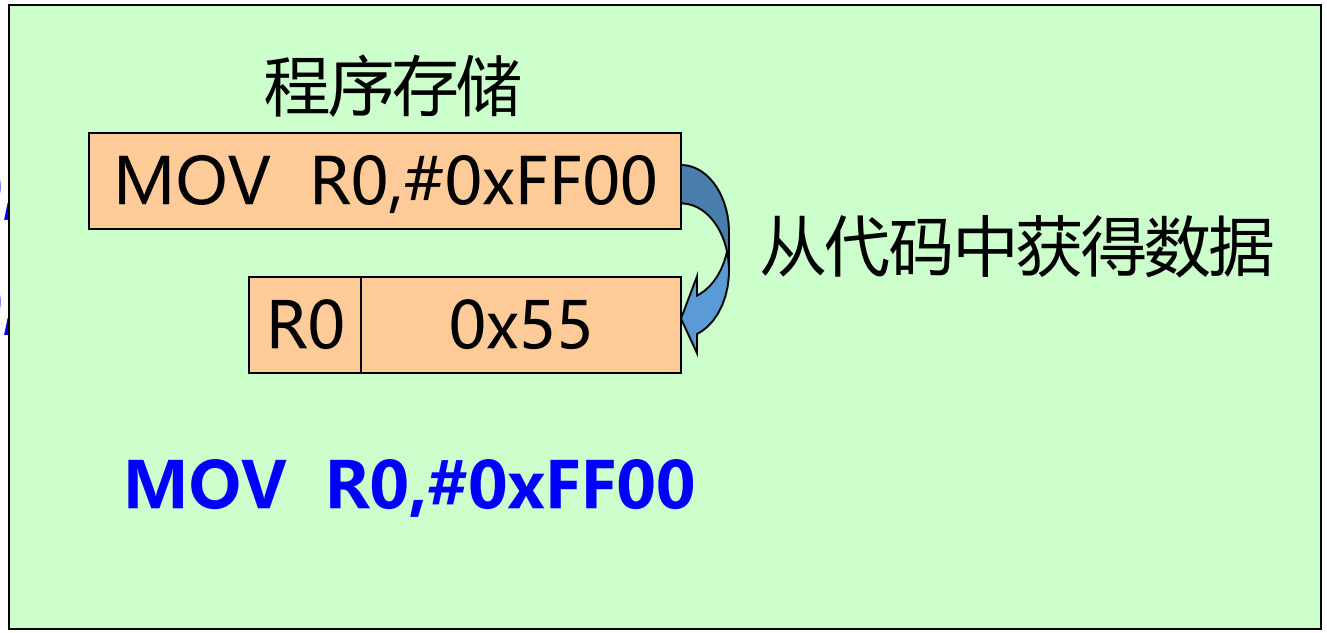
ARM处理器寻址方式

❑寻址方式分类——立即寻址

立即寻址指令中数据就包含在指令当中，取出指令也就取出了可以立即使用的操作数(这样的数称为立即数)。立即寻址指令举例如下：

SUBS R0,

MOV R0,



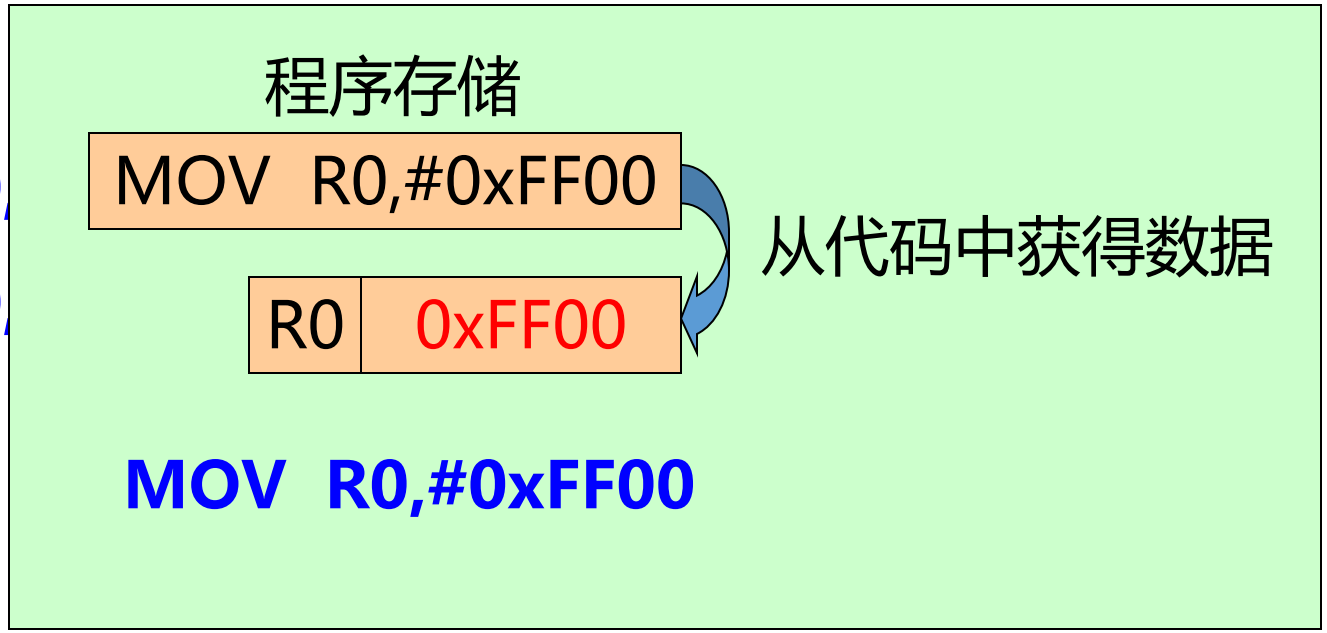
ARM处理器寻址方式

❑寻址方式分类——立即寻址

立即寻址指令中数据就包含在指令当中，取出指令也就取出了可以立即使用的操作数(这样的数称为立即数)。立即寻址指令举例如下：

SUBS R0,

MOV R0,



ARM处理器寻址方式

❑寻址方式分类——寄存器移位寻址

寄存器移位寻址是ARM指令集特有的寻址方式：第2个寄存器操作数先进行移位操作。寄存器移位寻址指令举例如下：

```
MOV  R0, R2, LSL #3          ; R0 := R2×8
AND  R1, R1, R2,LSR R3       ; R1 &= (R2>>R3)
```

ARM处理器寻址方式

❑寻址方式分类——寄存器移位寻址

寄存器移位寻址是ARM指令集特有的寻址方式：第2个寄存器操作数先进行移位操作，寄存器移位寻址指令举例如下：

MOV R0,

AND R1,

逻辑左移3位

R2	0x01
R0	0x55

MOV R0, R2, LSL #3

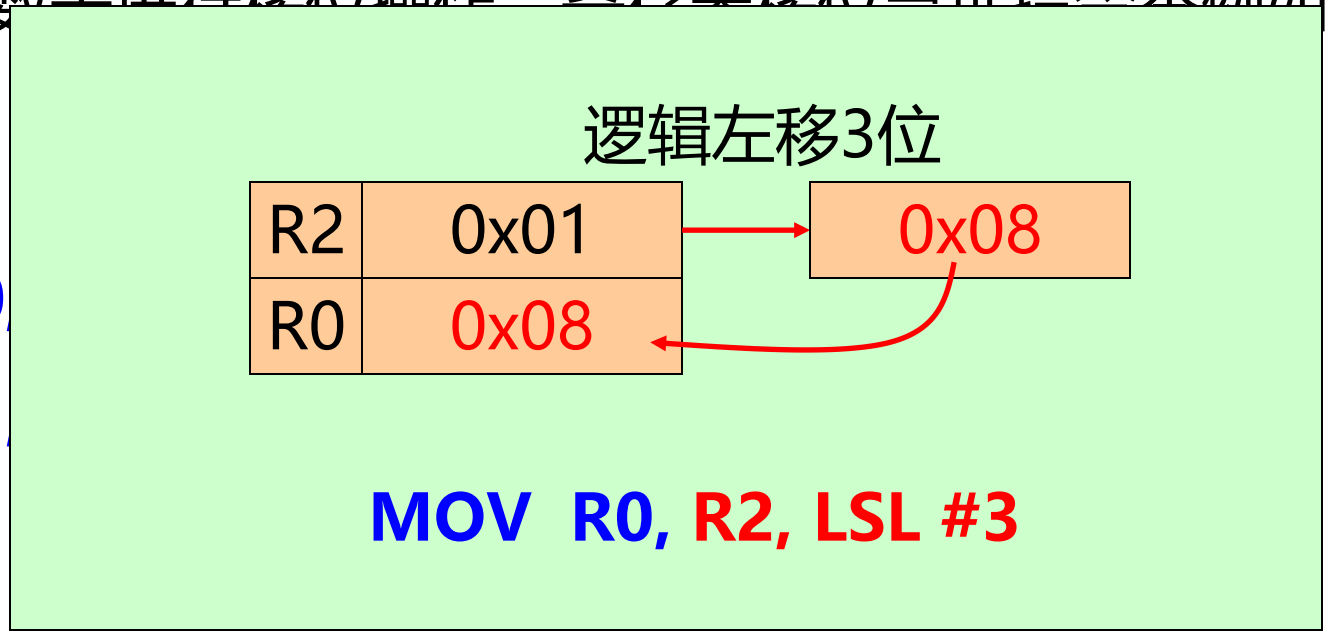
ARM处理器寻址方式

❑寻址方式分类——寄存器移位寻址

寄存器移位寻址是ARM指令集特有的寻址方式：第2个寄存器操作数先进行移位操作，寄存器移位寻址指令举例如下：

MOV R0,

AND R1,



ARM处理器寻址方式

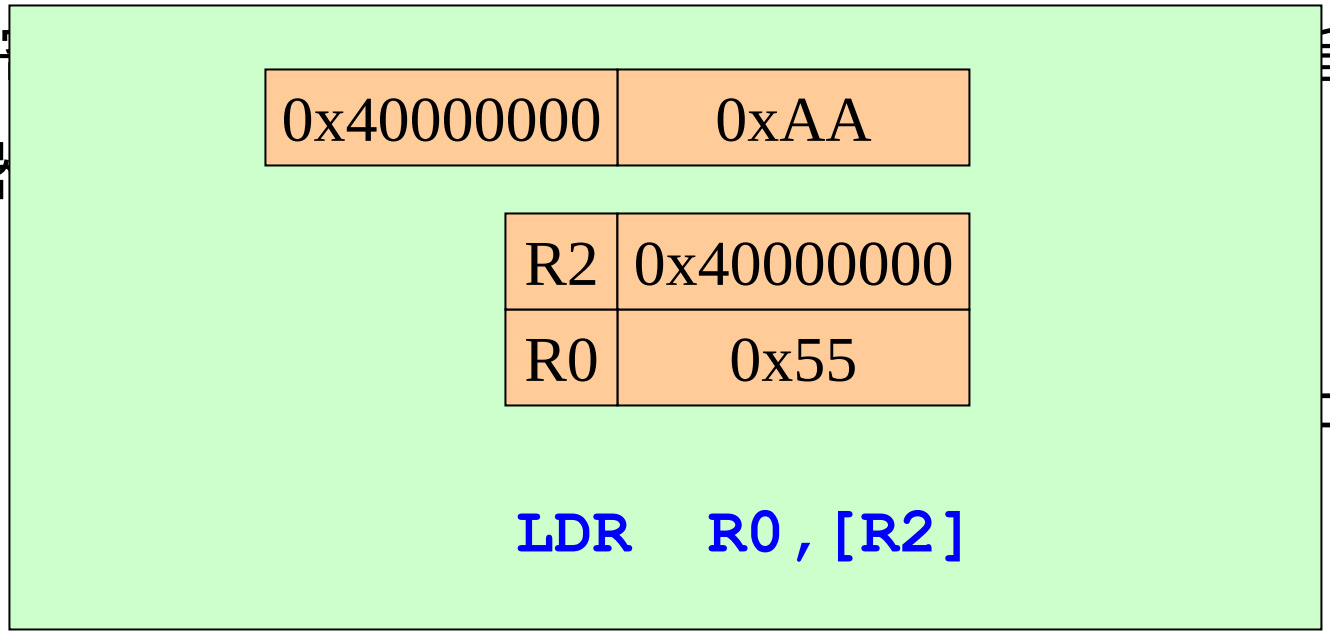
❑寻址方式分类——寄存器间接寻址

寄存器间接寻址指令中寄存器为操作数的地址指针。寄存器间接寻址指令举例如下：

```
LDR   R1,[R2]      ;将R2指向的存储单元的数据读出
                   ;保存在R1中
```

ARM处理器寻址方式

❑寻址方式分类——寄存器间接寻址



寄存器

指针。

LDR

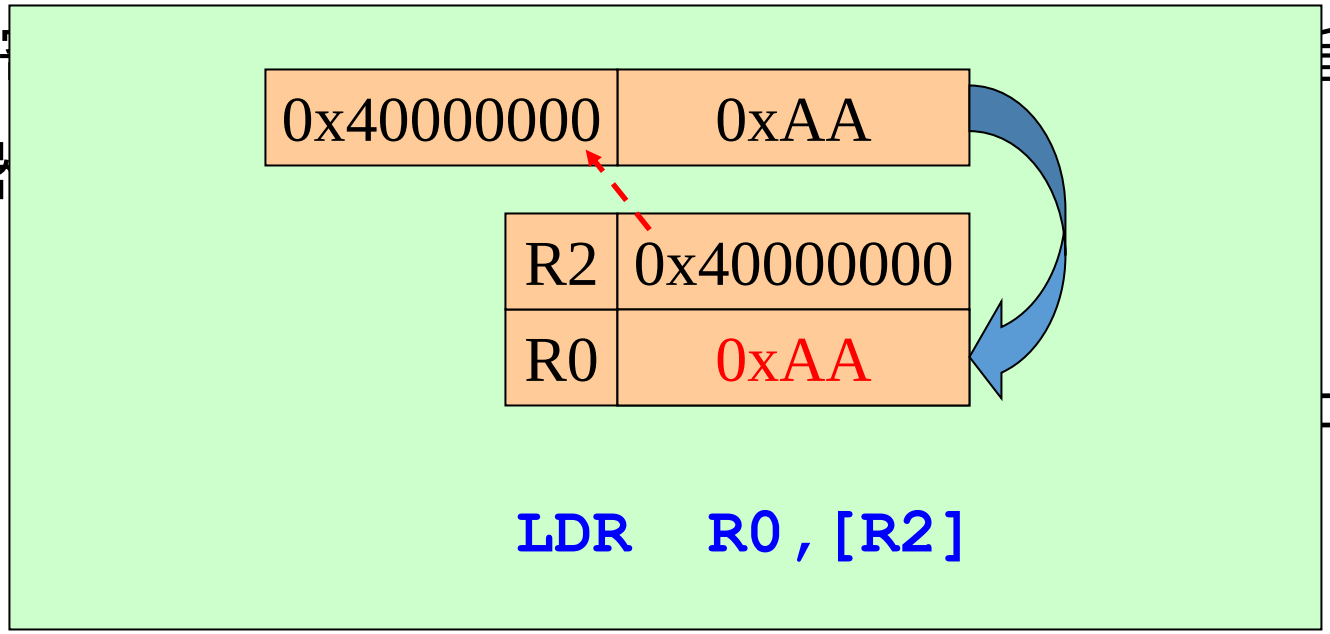
出

ARM处理器寻址方式

❑寻址方式分类——寄存器间接寻址

寄存器

LDR



ARM处理器寻址方式

❑寻址方式分类——相对寻址

可以看做是寄存器变址寻址方式的一个特例，此时包含基地址的寄存器特指程序计数器PC，通过PC值与指令中的偏移量结合，生成有效的操作数地址。一般这种方式用于指令跳转：

```
BL Label                    ;跳转到Label标签处
```

堆栈操作也可以归为一种寻址方式，称为堆栈寻址：

```
STMFD SP!, {R1-R7, LR} ;将R1-R7和LR压栈，一般用于保护现场
```

ARM处理器寻址方式

❑寻址方式分类——基址变址寻址

基址变址寻址将基址寄存器的内容与指令中给出的偏移量（<4K）相加/减，形成操作数的有效地址。寄存器间接寻址相当于偏移量为0的基址加偏移寻址。

基址寻址指令举例如下(前索引寻址)：

```
LDR     R2,[R3,#0x0C]     ;读取R3+0x0C地址上的存储单元
                          ;的内容，放入R2
STR     R1,[R0,#-4]!      ;先R0=R0-4，然后把R1的值保存
                          ;到R0指向的存储单元
LDR   R0，[R1,R2]          ;R0=[R1+R2]
```

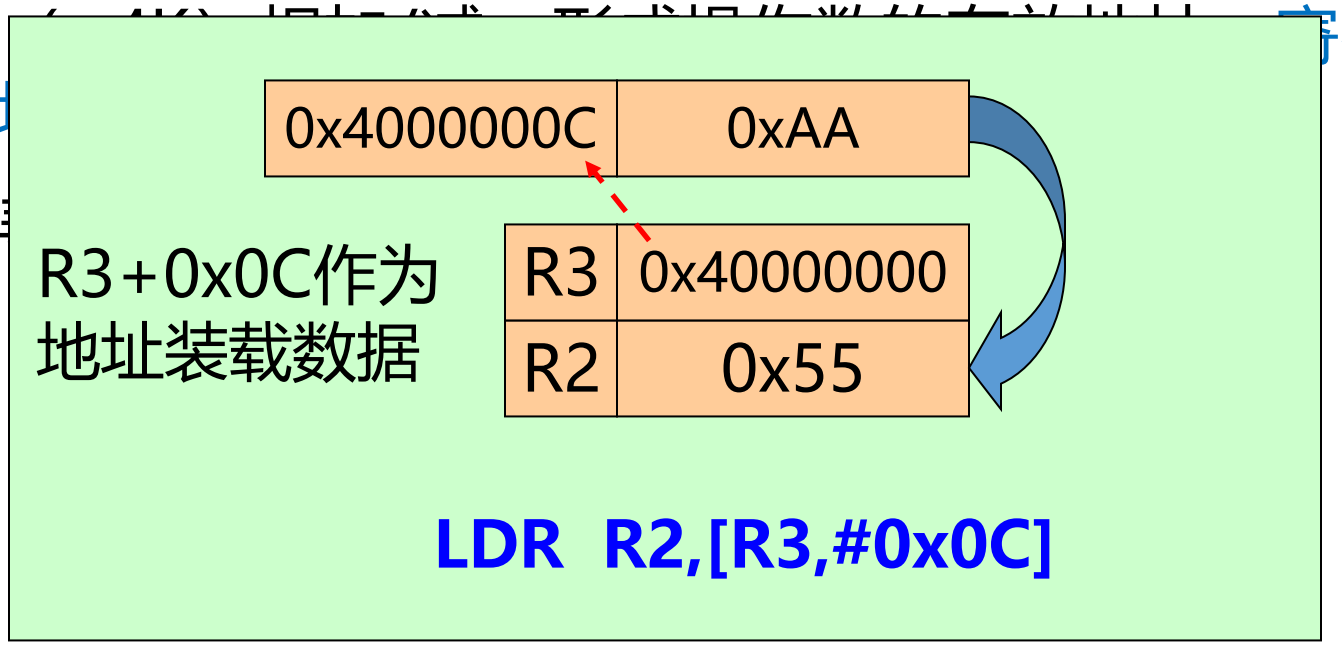
ARM处理器寻址方式

❑寻址方式分类——基址变址寻址

基址变址寻址将基址寄存器的内容与指令中给出的偏移量

寄存器间接寻址

基址



LDR

STR

LDR R0，[R1,R2] ;R0=[R1+R2]

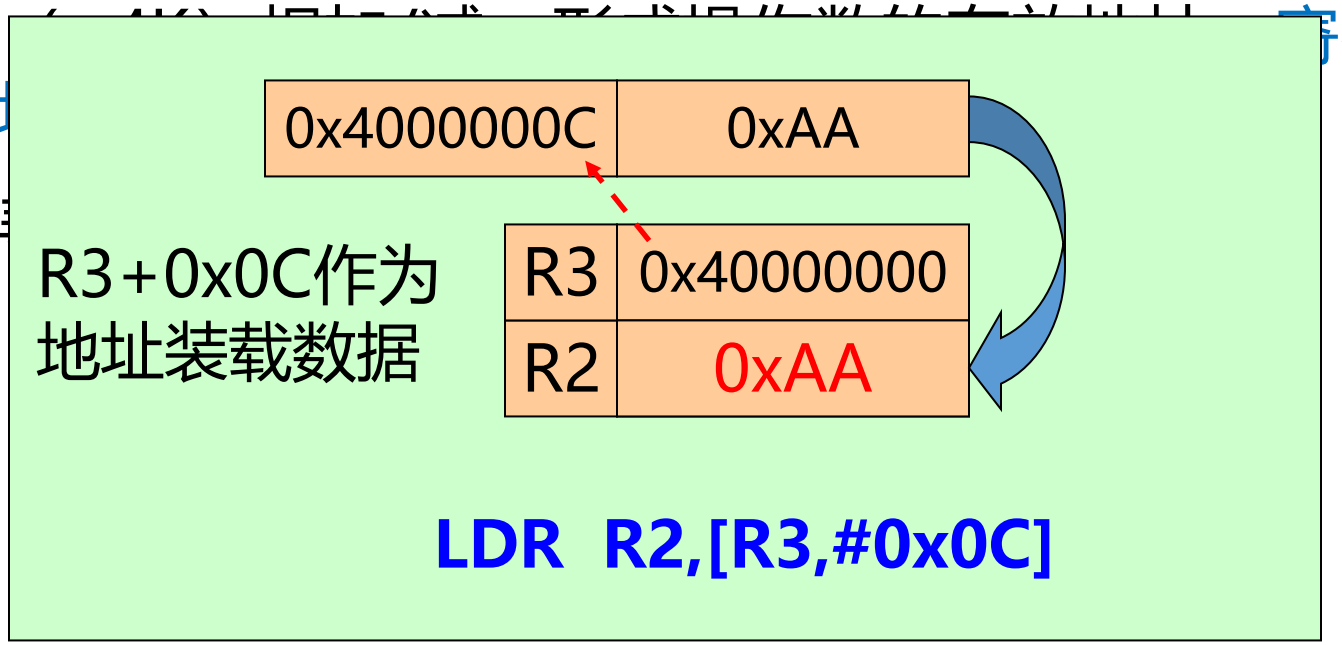
ARM处理器寻址方式

❑寻址方式分类——基址变址寻址

基址变址寻址将基址寄存器的内容与指令中给出的偏移量

寄存器间接寻址

基址



LDR

STR

LDR R0，[R1,R2] ;R0=[R1+R2]

ARM处理器寻址方式

❑寻址方式分类——多寄存器寻址

允许一条指令传送16个寄存器的任何子集或所有寄存器。多寄存器寻址指令举例如下：

```
LDMIA  R1!,{R2-R4,R6}        ;将R1指向的单元中的数据装入
                             ;R2～R4、R6中(R1自动加4)
STMIA  R0!,{R2-R7,R12}       ;将寄存器R2～R7、R12的值保
                             ;存到R0指向的存储; 单元中
                             ;(R0自动加7)
```

IA/IB/DA/DB

- **IA:**（Increase After） 每次传送后地址加4，其中的寄存器**从左到右执行**，例如:STMIA R0,{R1,LR} 先存R1，再存LR
- **IB:**（Increase Before）每次传送前地址加4，其中的寄存器**从左到右执行。**
- **DA:**（Decrease After）每次传送后地址减4，其中的寄存器**从右到左执行**，例如:STMDA R0,{R1,LR} 先存LR，再存R1
- **DB:** （Decrease Before）每次传送前地址减4，其中的寄存器**从右到左执行。**

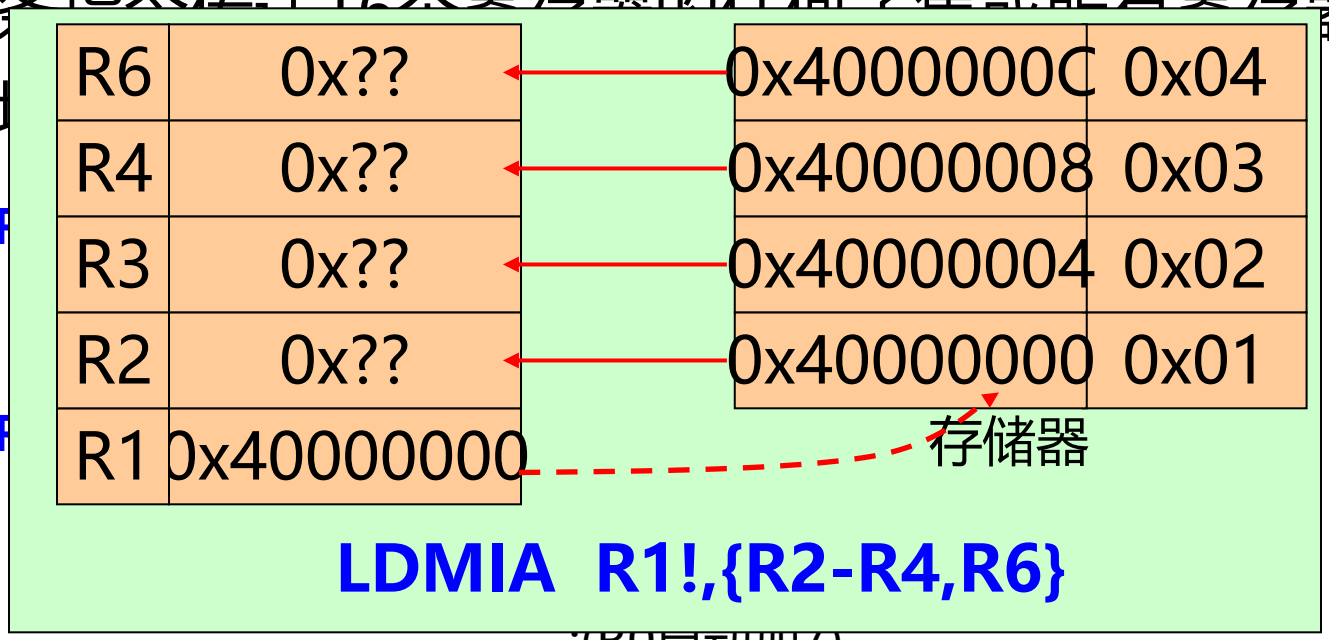
ARM处理器寻址方式

❑寻址方式分类——多寄存器寻址

允许一条指令传送16个寄存器的任何子集或所有寄存器。多寄存器寻址

LDMIA R

STMIA R



;(R0自动加7)

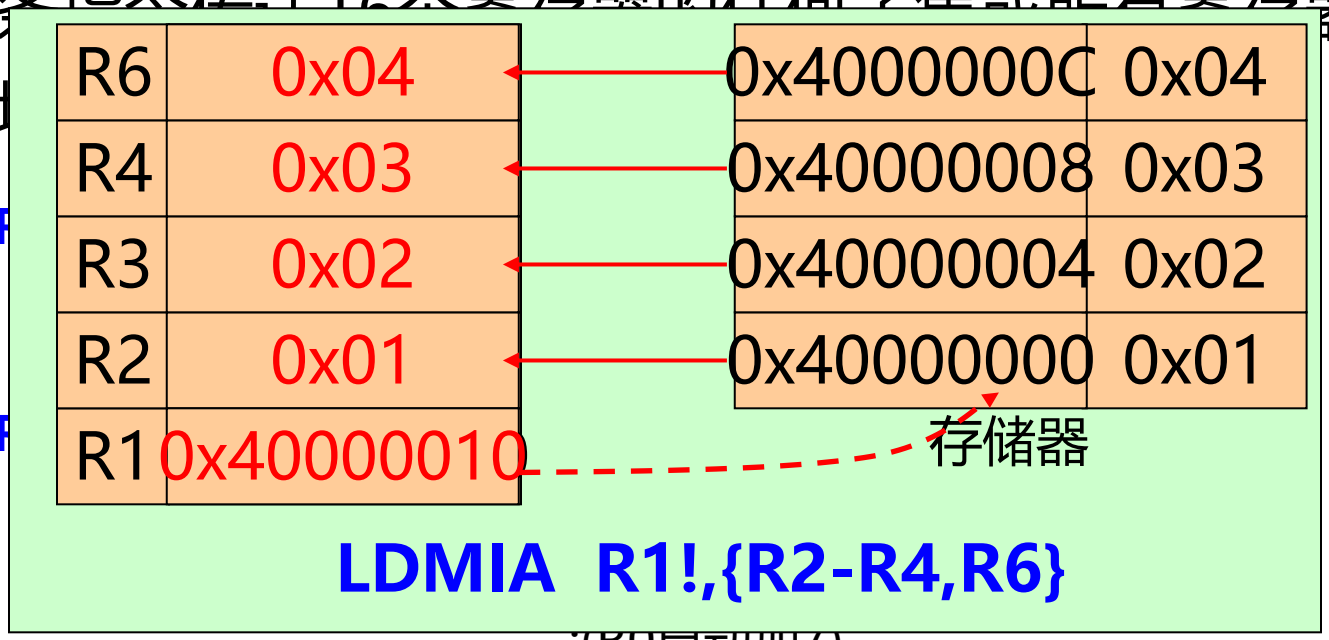
ARM处理器寻址方式

❑寻址方式分类——多寄存器寻址

允许一条指令传送16个寄存器的任何子集或所有寄存器。多寄存器寻址

LDMIA R

STMIA R



;(R0自动加7)

ARM存储器访问指令—多寄存器存取

助记符	说明	操作	条件码位置
LDM{mode} Rn{!},reglist	多寄存器加载	reglist←[Rn...]，Rn回写等	LDM{cond}{mode}
STM{mode} Rn{!},reglist	多寄存器存储	[Rn...]←reglist,Rn回写等	STM{cond}{mode}

多寄存器加载/存储指令可以实现在一组寄存器和一块连续的内存单元之间传输数据。LDM为加载多个寄存器；STM为存储多个寄存器。允许一条指令传送16个寄存器的任何子集或所有寄存器。它们主要用于现场保护、数据复制、常数传递等。

ARM存储器访问指令—寄存器和存储器交换指令

助记符	说明	操作	条件码位置
SWP Rd,Rm,[Rn]	寄存器和存储器字数据交换	Rd←[Rn] ， [Rn]←Rm (Rn≠Rd或Rm)	SWP{cond}
SWPB Rd,Rm,[Rn]	寄存器和存储器字节数据交换	Rd←[Rn] ， [Rn]←Rm (Rn≠Rd或Rm)	SWP{cond}B

SWP指令用于将一个内存单元(该单元地址放在寄存器Rn中)的内容读取到一个寄存器Rd中，同时将另一个寄存器Rm的内容写入到该内存单元中。使用SWP可实现信号量操作。

指令格式如下：

SWP{cond}{B} Rd,Rm,[Rn]

其中，B为可选后缀，若有B，则交换字节，否则交换32位字；Rd用于保存从存储器中读入的数据；Rm的数据用于存储到存储器中；Rn为要进行数据交换的存储器地址，Rn不能与Rd和Rm相同。

ARM子程序调用

```
BL SUBR                 ; branch to SUBR
…                       ; return to here
…
SUBR
…                       ; subroutine entry point
MOV pc, r14             ; return
```

转跳时返回地址存放在r14中，子程序返回时只要把原先保存在r14的地址装入pc即可（没有类似RET的语句）

注意：
如果子程序要使用r14的话，需要保存存在r14里的返回地址。

ARM指令集—分支指令

助记符	说明	操作	条件码位置
B label	分支指令	PC←label	B{cond}
BL label	带链接的分支指令	LR←PC-4，PC←label	BL{cond}
BX Rm	带状态切换的分支指令	PC←Rm，切换处理器状态	BX{cond}
BLX label	带链接和状态的分支		

带状态切换的分支指令——BX指令，该指令可以根据跳转地址（Rm）的最低位来切换处理器状态。其跳转范围限制在当前指令的±32M字节地址内(ARM指令为字对齐，最低2位地址固定为0)。指令格式如下：

BX{cond}　Rm

跳转地址 Rm[0]	跳转后	
	CPSR标志T位	处理器状态
0	0	ARM
1	1	Thumb

ARM子程序嵌套调用

子程序里再调用子程序时会改变r14，因此遇到多层程序调用时得保存r14

```
BL SUB1                          ; branch to SUB!
…
…

SUB1
STMFA r13!, {r0-r2, r14}         ; save regs
BL SUB2
…
…
LDMFA r13!, {r0-r2, pc}          ; return

SUB2
…
MOV pc, r14                      ; return
```


Thank you!