

# 嵌入式系统

## An Introduction to Embedded System

### 第三课 **ARM**处理平台概述

# 课程大纲

## 系统结构内容补充

### ARM系统概述

### ARM指令集简介

# 计算机体系结构分类方法

## 体系结构分类

### 1、按计算规模（IEEE，1989）

- 个人计算机（PC）

- 工作站

- 小型机

- 中型机

- 小巨型机

- 巨型机

### 2、按指令流、数据流的多倍性分布（Flynn）

- 单指令/单数据流 (SISD)

- 单指令/多数据流 (SIMD)

- 多指令/单数据流 (MISD)

- 多指令/多数据流 (MIMD)

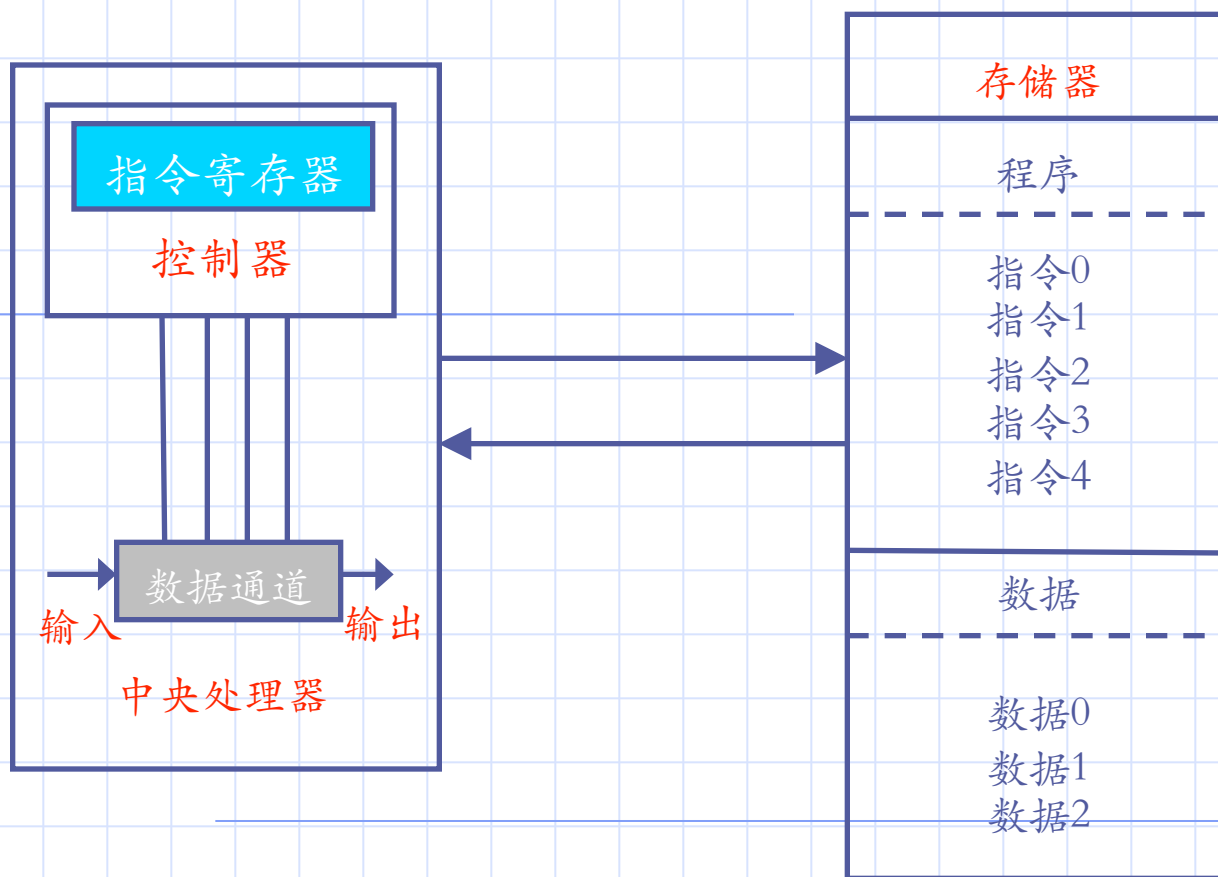
### 3、按指令/数据逻辑分布

- 普林斯顿体系结构

- 哈佛体系结构

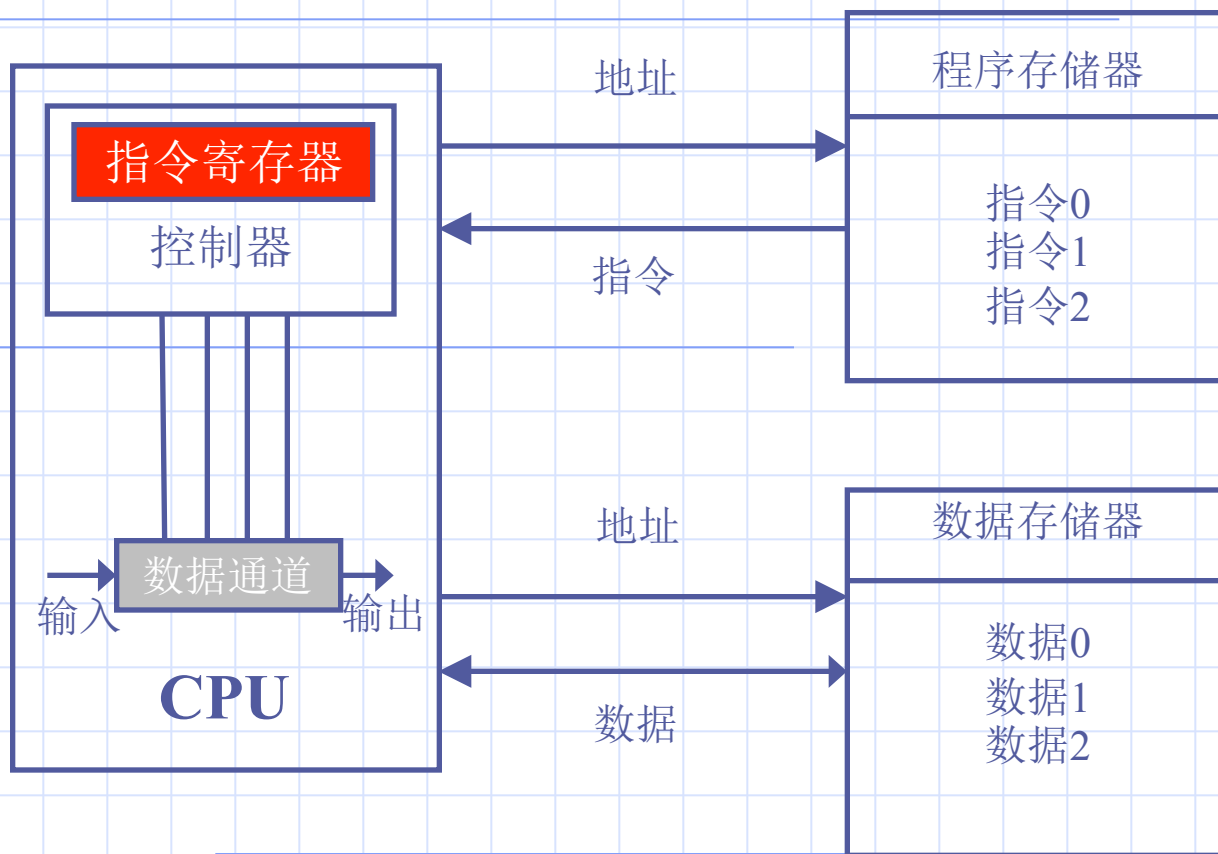
- 混合体系结构

# 普林斯顿体系结构（冯·诺依曼体系结构）



约翰·冯·诺依曼  
1903~1957

# 哈佛体系结构



# 流水线—提高CPU性能的方法

- 流水线(pipeline)方式是把一个重复的过程分解为若干子过程，每个子过程可以与其他子过程并行进行的工作方式。
- 采用流水线技术设计的微处理器，把每条指令分为若干个顺序的操作(如取指、译码、执行等)，每个操作分别由不同的处理部件(如取指部件、译码部件、执行部件等)来完成。
- 对于每个处理部件来说，每条指令的同类操作(如取指令)就像流水一样连续被加工处理。
- 超流水线**技术指CPU内部的流水线超过通常的5-6步以上，例如PENTIUM PRO的流水线就长达14步。流水线的步数越多，其完成一条指令的速度越快。

# 流水线竞争问题

- 无法进行理想的流水线，是因为存在竞争问题：
  - 结构竞争(structure hazard): 硬件资源冲突
  - 数据竞争(data hazard): 指令之间存在数据传递
  - 控制竞争(control hazard): 转移指令引起执行顺序改变
  
- 流水线竞争的后果：
  - 造成流水线停顿几个周期，从而降低流水线效率

# 解决结构竞争的方法—超标量技术

- **超标量(superscalar)**技术是指在CPU中有一条以上的流水线，并且每个时钟周期内可以完成一条以上的指令。
- Pentium处理器的流水线由分别称为“U流水”和“V流水”的两条指令流水线构成(双流水线结构)，其中每条流水线都拥有自己的地址生成逻辑、ALU及数据Cache接口。
- Pentium处理器可以在一个时钟周期内同时发送两条指令进入流水线。比相同频率的单条流水线结构(如 80486)性能提高了一倍。
- **总结**
  - ✓ 超标量技术采用的是指令空间并行性
  - ✓ 超级流水线技术采用的是指令时间并行性



# 解决数据竞争的方法—延迟槽调度技术（1/2）

■延迟槽（Delay Slots）是指令取数后至获得运算结果的延迟时钟周期数。

Instruction Type	Delay Slots	Read Cycles†	Write Cycles†	Branch Taken†
NOP (no operation)	0			
Store	0	i	i	
Single cycle	0	i	i	
Multiply (16 × 16)	1	i	i + 1	
Load	4	i	i, i + 4\$	
Branch	5	i‡		i + 5

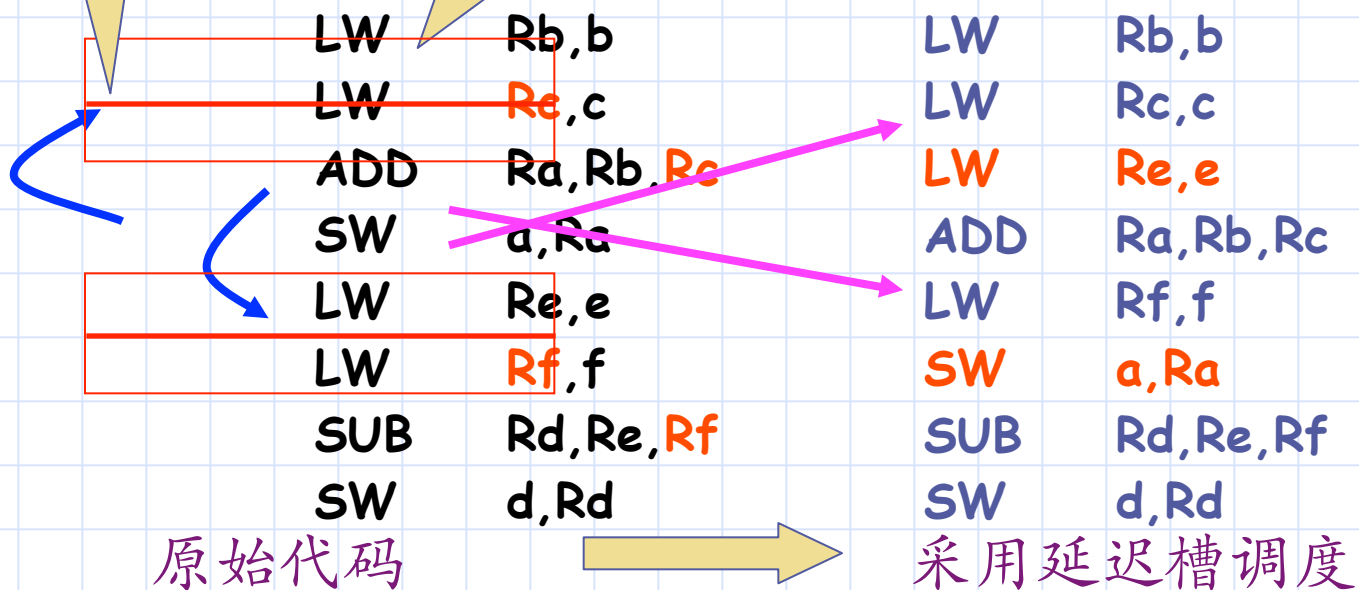
■延迟槽调度技术是指编译器采用指令重排（reorder）的方法，将延迟槽填满，从而减少流水线停顿。

TI C6000 DSP的延迟槽

# 解决数据竞争的方法—延迟槽调度技术 (2/2)

具有延迟槽

存在数据冲突

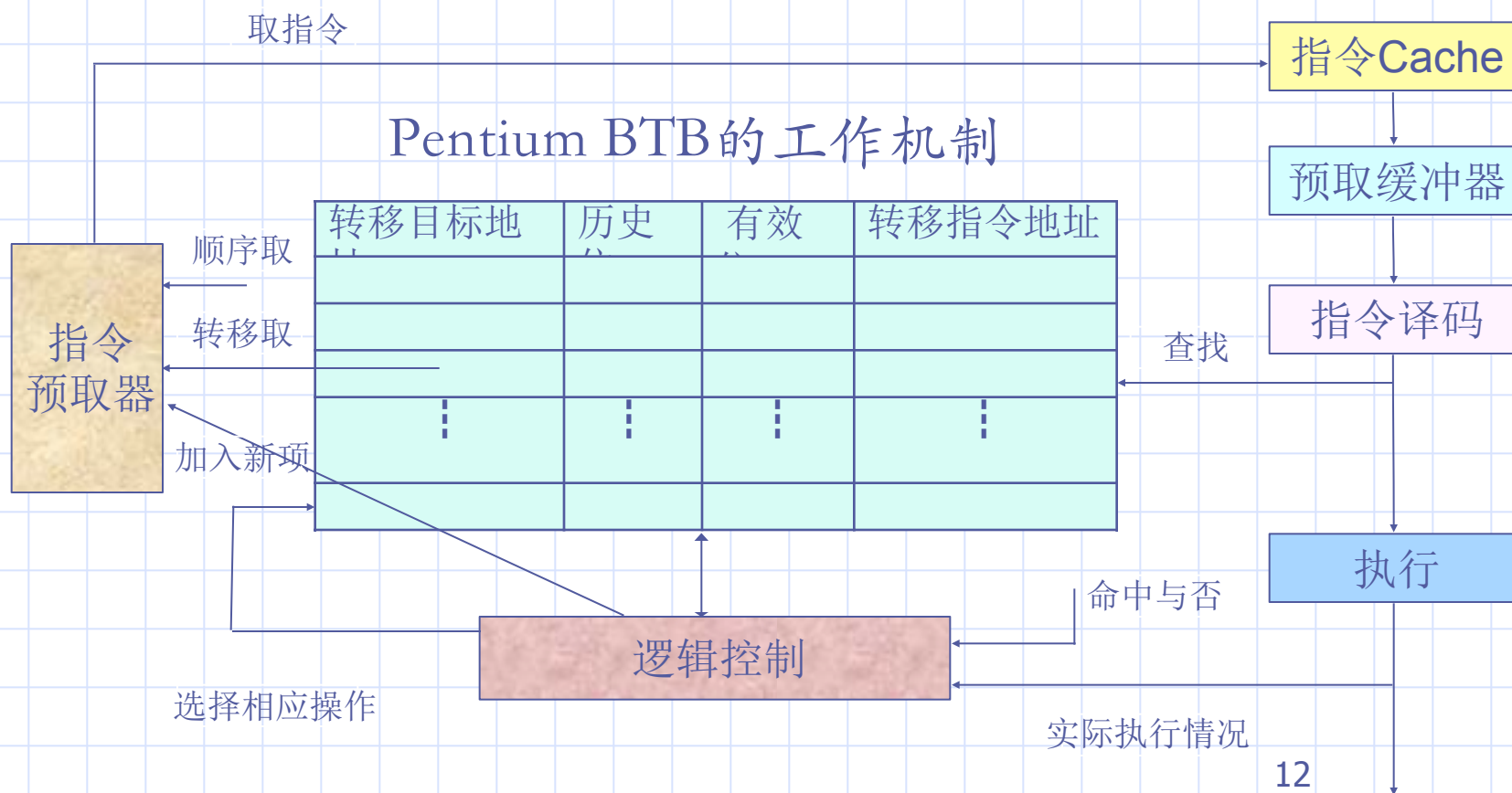


# 解决控制竞争的方法—转移预测技术（1/2）

- 正是由于计算机指令中具有能够改变程序流向的指令，才使得程序结构灵活多样，程序功能丰富多彩。
- 这类指令包括“无条件转移指令”及“条件转移指令”两大类，我们统称之为转移(branch)指令。
- 转移指令给处理器的流水线操作带来麻烦。
- 转移预测有“静态转移预测”及“动态转移预测”之分。
- 动态转移预测法(dynamic branch prediction)是依据一条转移指令过去的行为来预测该指令的将来行为。即处理器要有一个“不断学习”的过程。

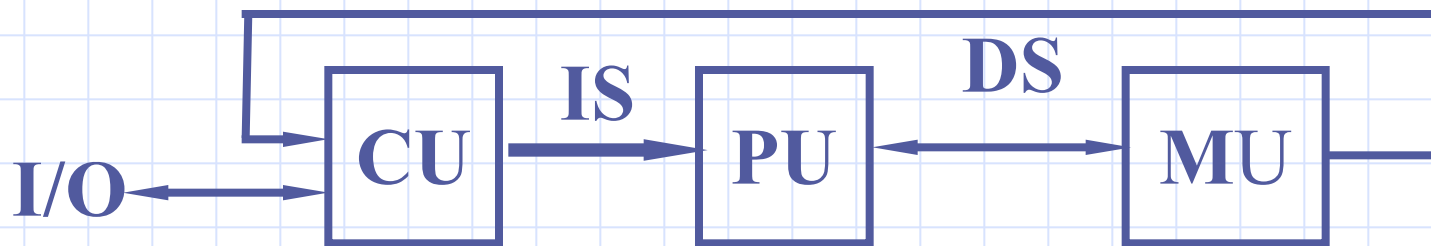
# 解决控制竞争的方法—转移预测技术（2/2）

- 采用“转移目标缓冲器”BTB(Branch Target Buffer)来动态预测程序的转移操作。



# Flynn分类—单指令/单数据流 **SISD**

传统的顺序计算机  
**IS**



CU: 控制部件

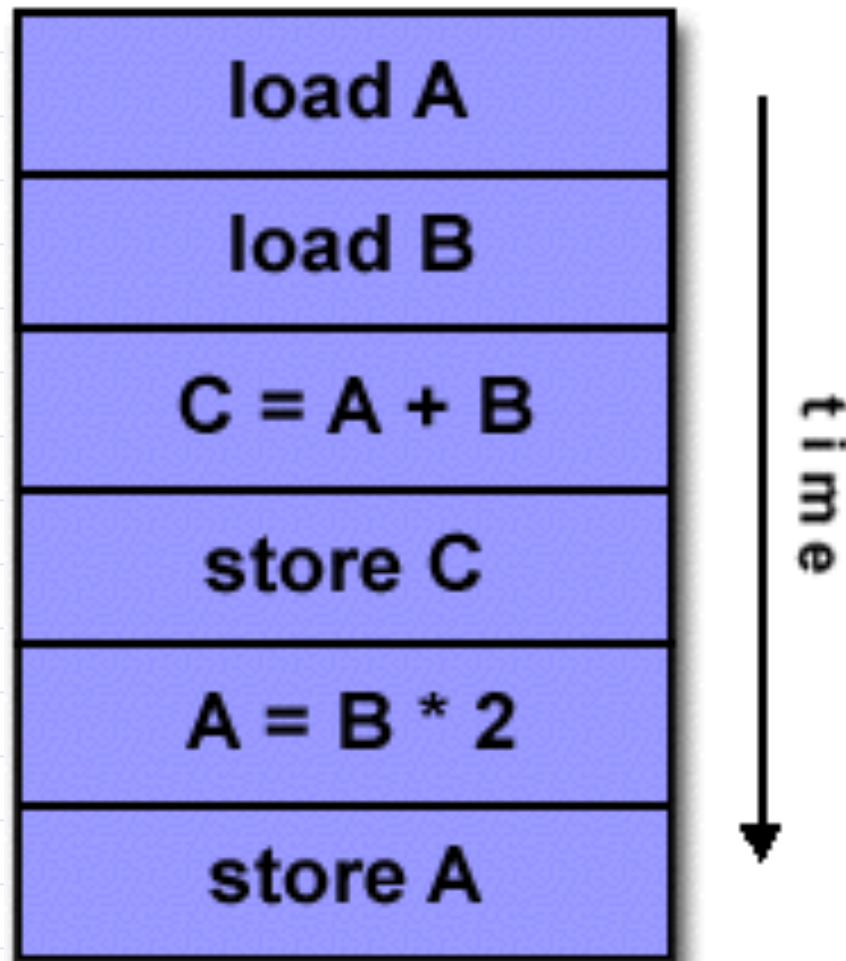
IS: 指令流

PU: 处理部件

DS: 数据流

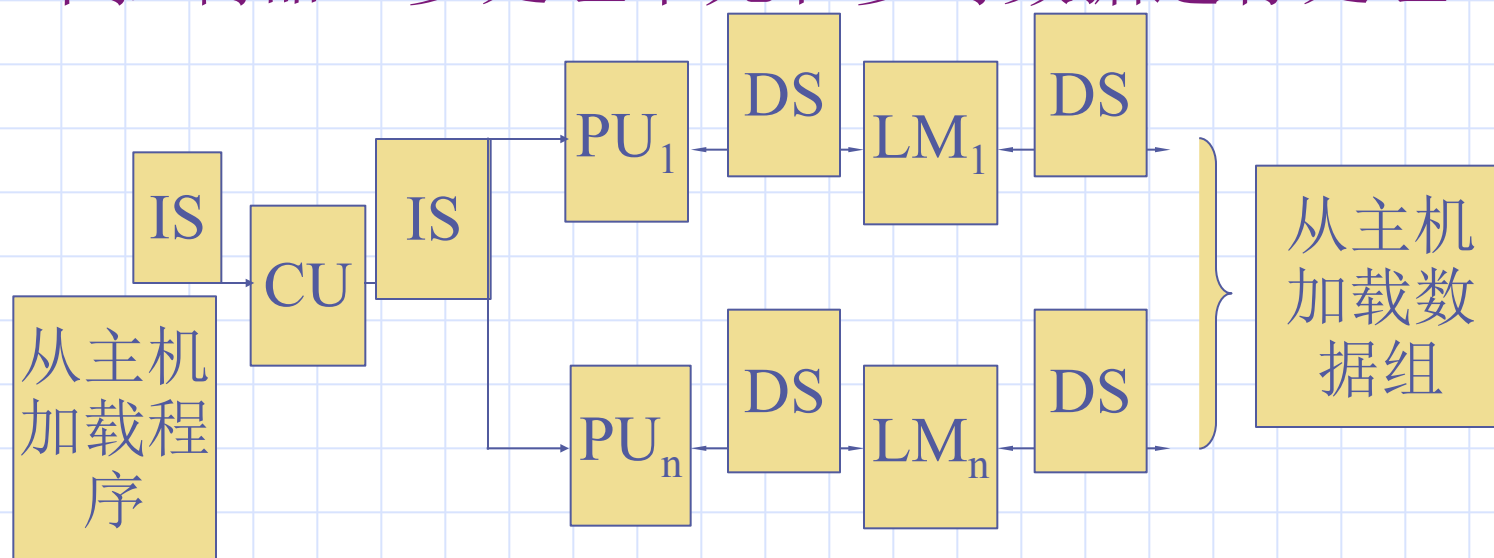
MU: 存储部件

# SISD



# Flynn分类—单指令/多数据流 SIMD

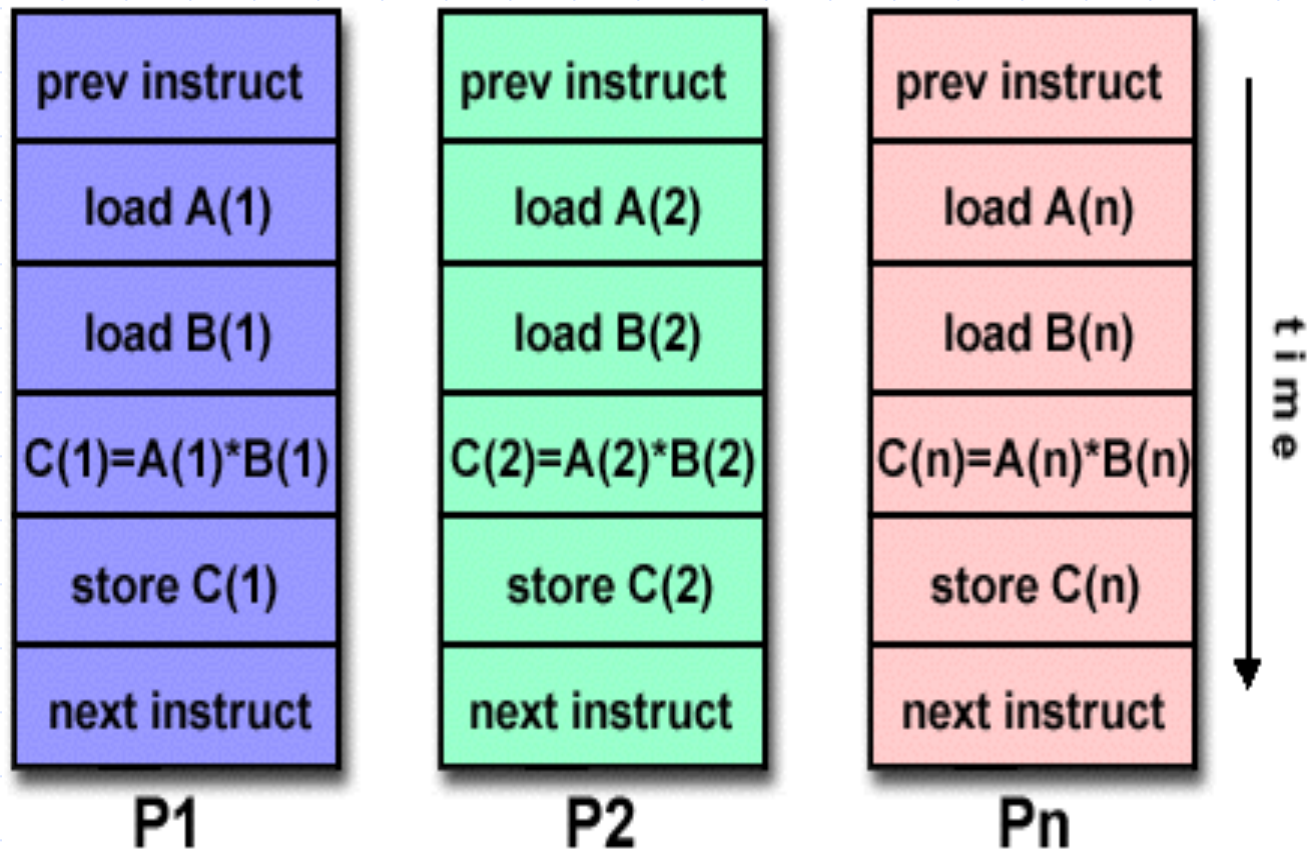
单控制器、多处理单元和多对数据进行处理



PU: 处理部件

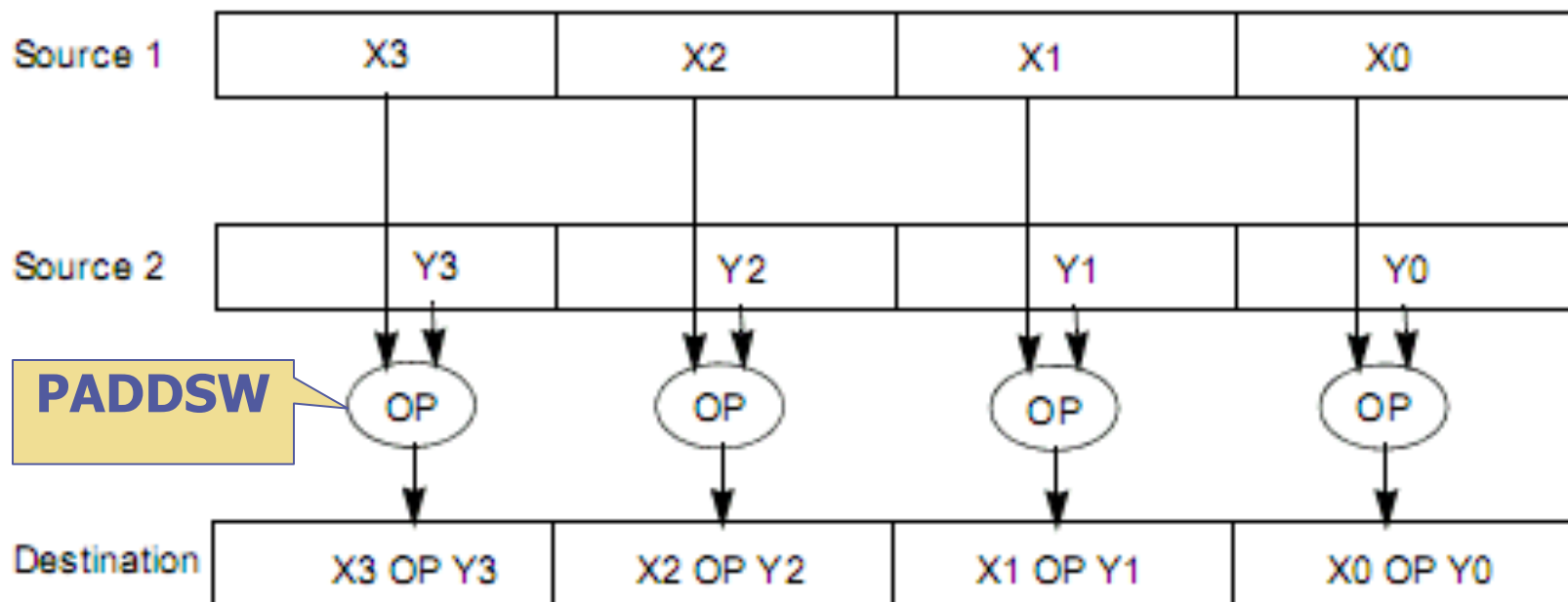
LM: 本地存储器

# SIMD



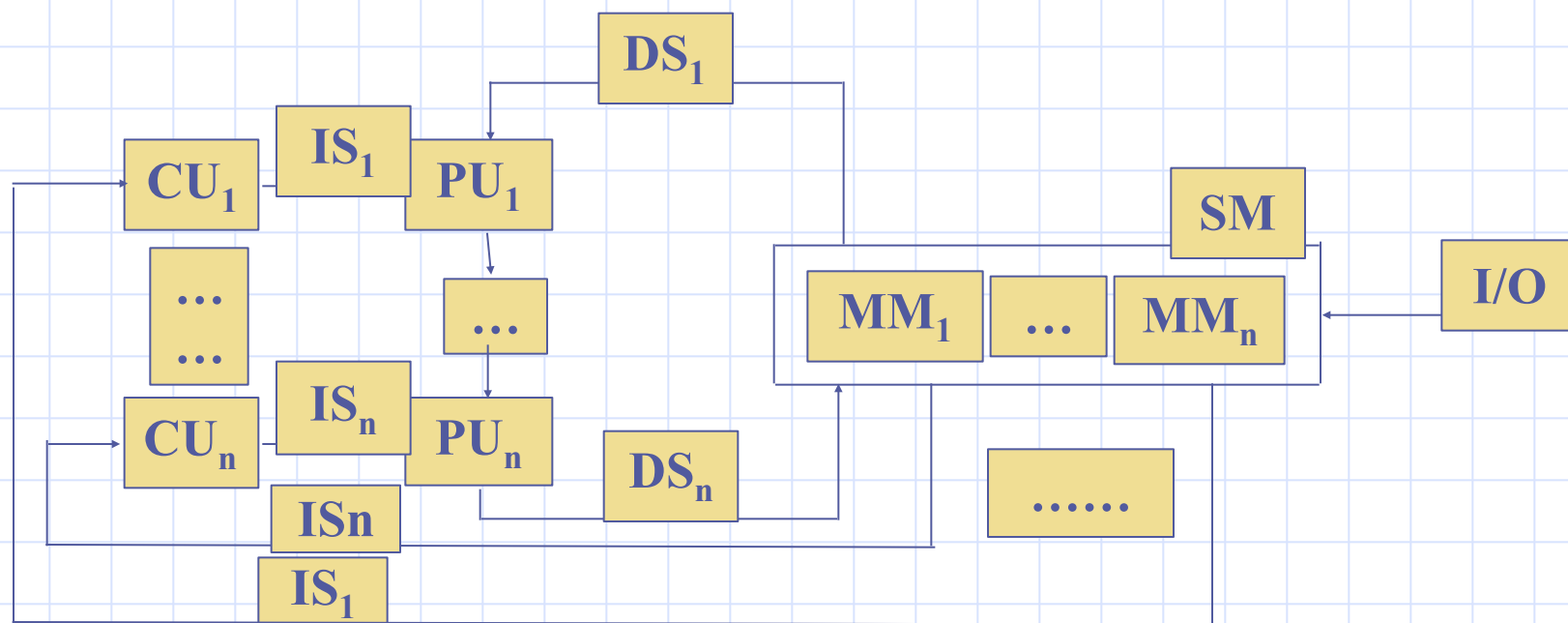


# SIMD实例：Intel MMX



# Flynn分类—多指令/单数据流 MISD

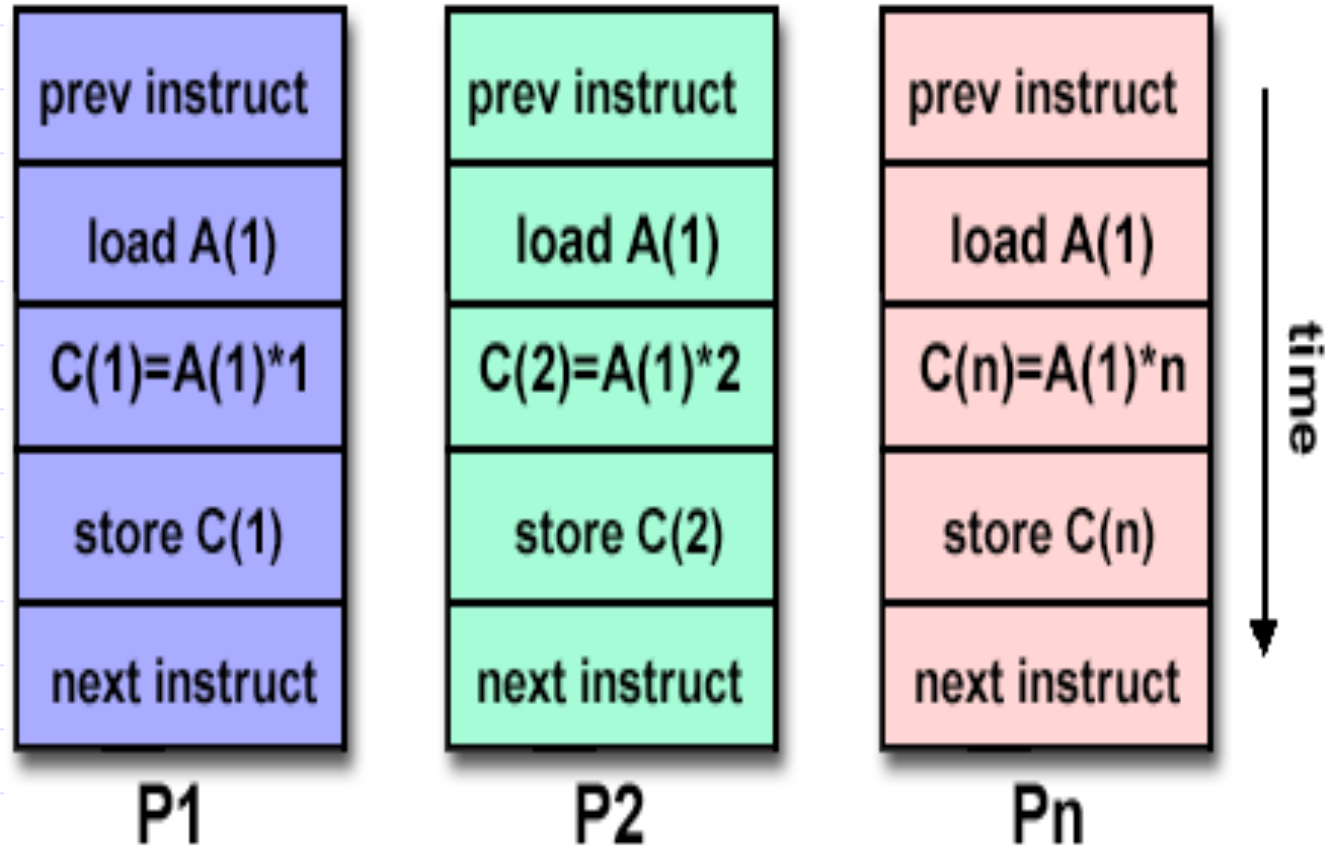
多个处理单元，对同一数据流进行处理



**MM:** 主存贮模块

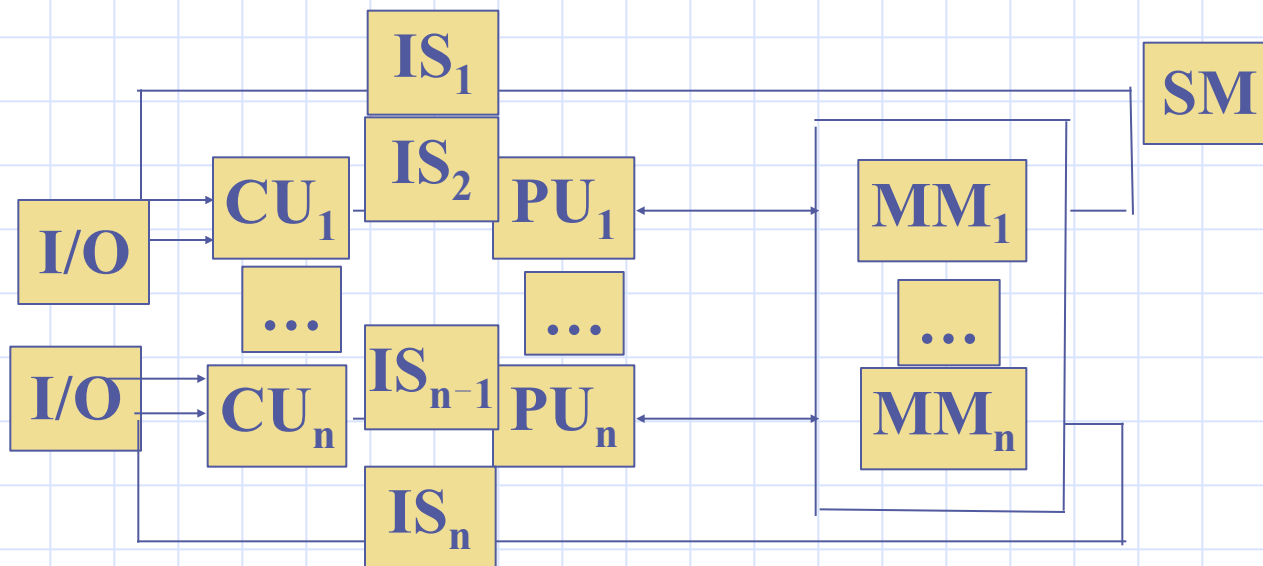
**SM:** 共享存储器

# MISD

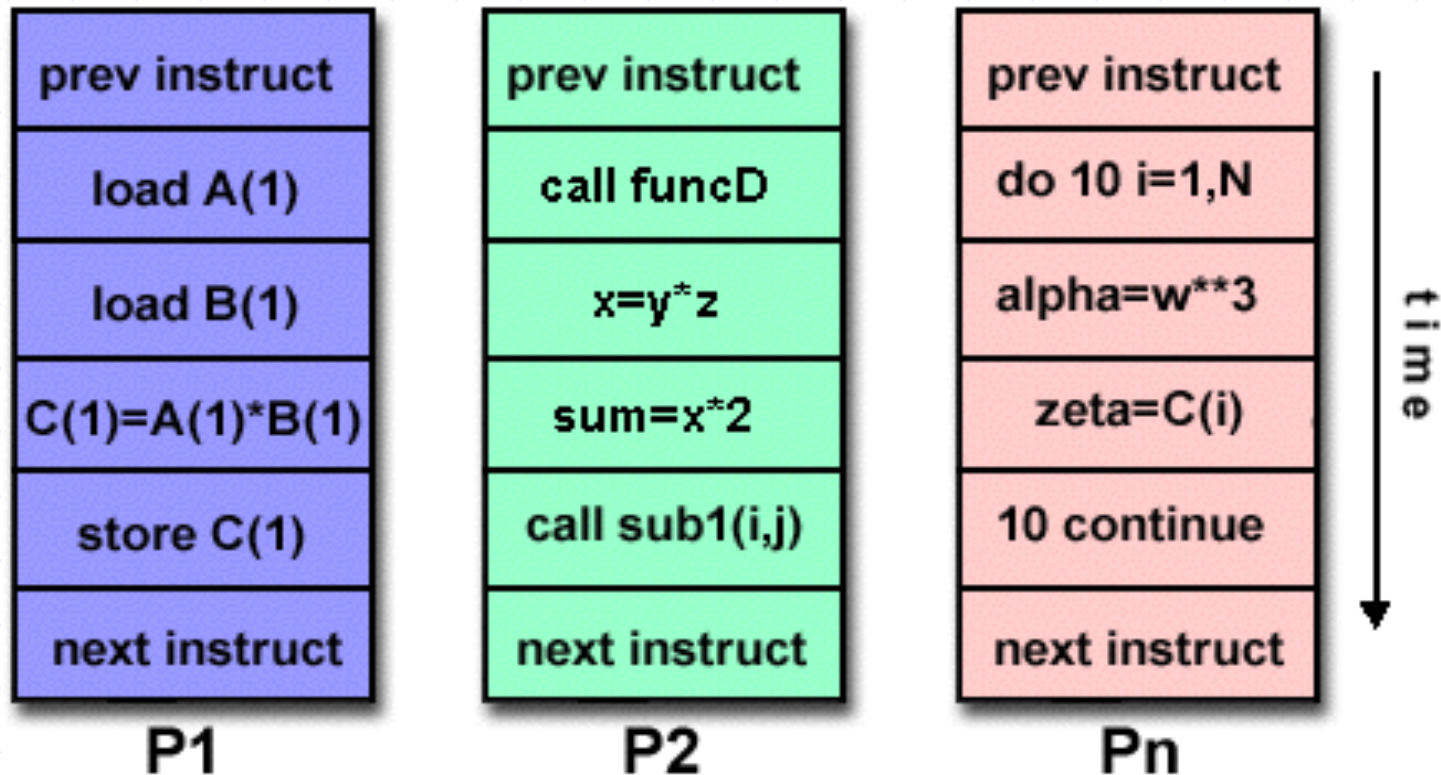


# Flynn分类—多指令/多数据流 **MIMD**

一般认为超标量计算机（**superscalar**）、超长指令字计算机（**VLIW**）可以作为此类计算机。

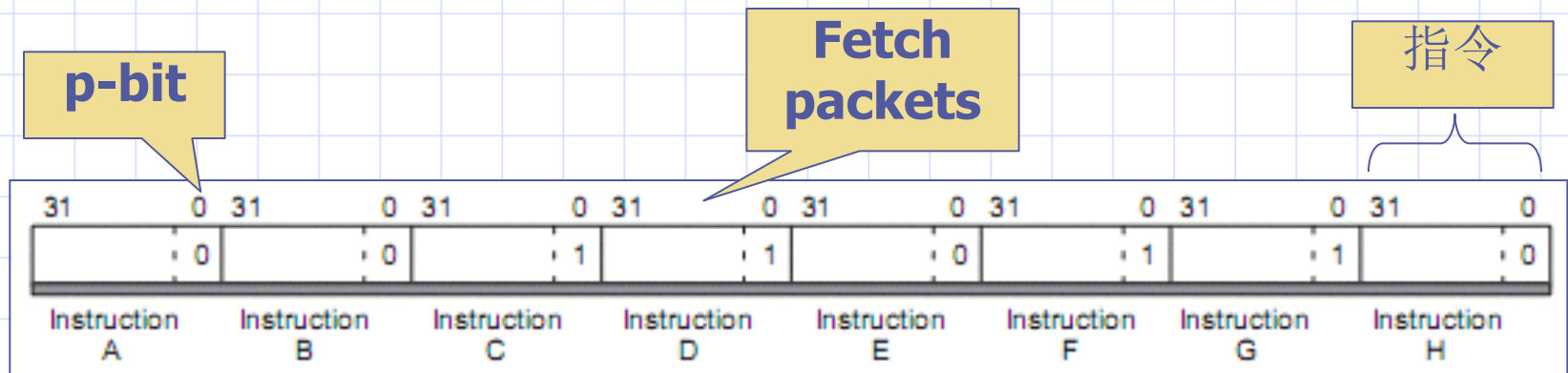


# MIMD



# MIMD实例： TI C6000 DSP(VLIW)

□ 超长指令字： Very long instruction word (VLIW)



Cycle/Execute Packet	Instructions		
1	A		
2	B		
3	C	D	E
4	F	G	H

执行顺序

# 指令集：CISC和RISC

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

具有大量的指令和寻址方式，指令长度可变

- 1、8/2原则：80%的程序只使用20%的指令
- 2、大多数程序只使用少量的指令就能运行



指令集  
精简的  
依据

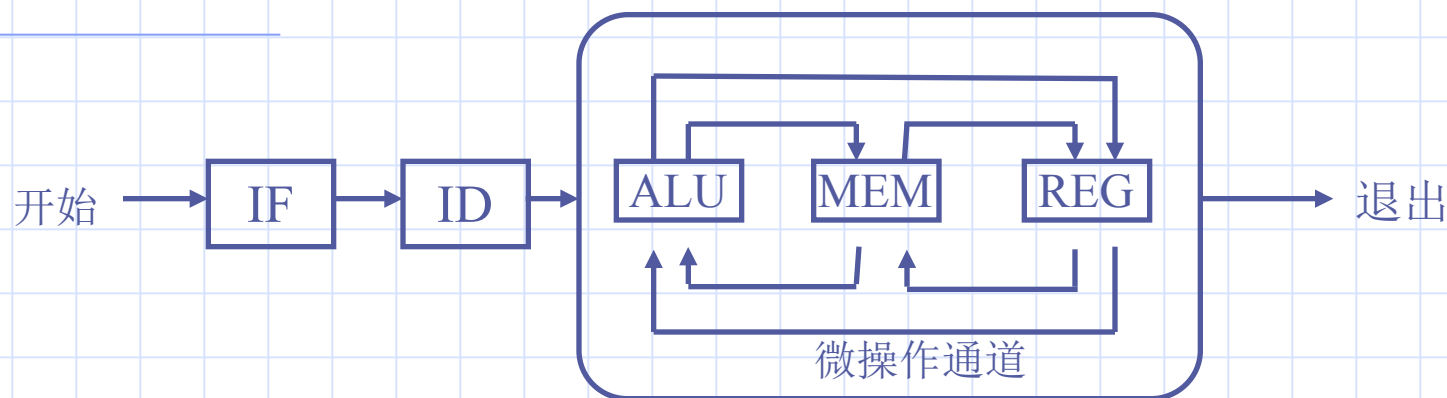
RISC：精简指令集 (Reduced Instruction Set Computer)

只包含最有用的指令，指令长度固定

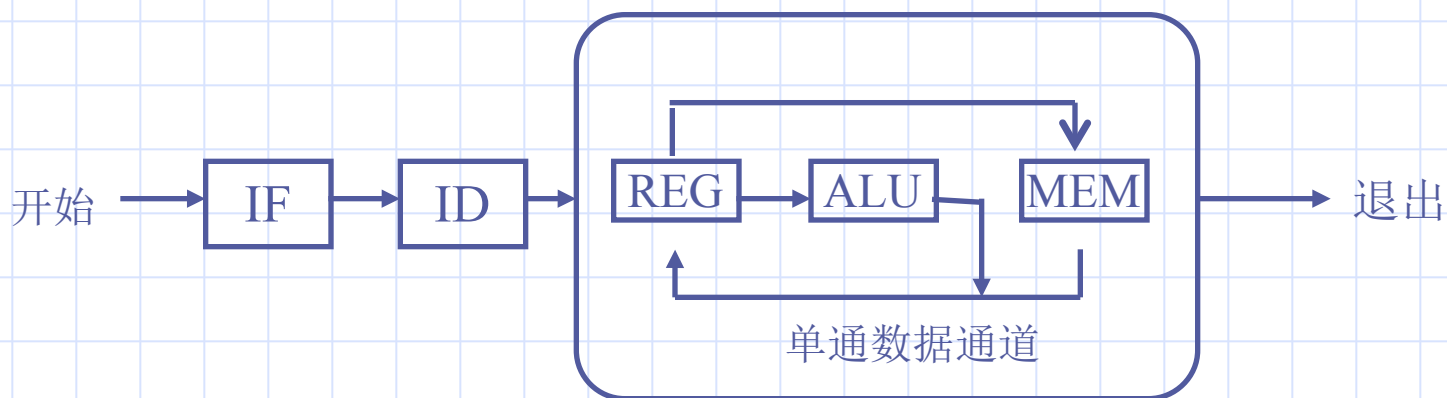
Load/Store结构，只采用寄存器寻址

使CPU硬件结构设计变得更为简单。

# 指令集：CISC与RISC的数据通道



CISC: 寻址方式复杂



RISC: Load/Store结构



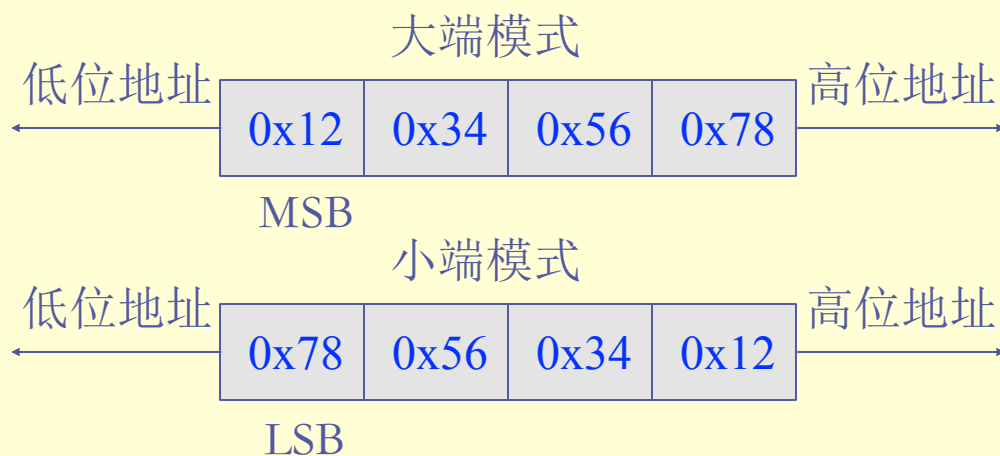
# CISC与RISC的对比

类别	CISC	RISC
指令系统	指令数量很多	较少，通常少于100
<p>■ RISC型处理器且有结构简单、处理速度快和处理功能强等优点，因而新型的嵌入式系统大多数都采用RISC型处理器作为核。</p> <p>■ ARM、PPC、MIPS、DSP等都是嵌入式系统常用的RISC型处理器。</p>		
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	只能对寄存器进行算术和逻辑操作，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

# 多字节数据存储——字节序(Byte Order)

**大端模式:** 1) 最高位字节 (MSB, Most Significant Bit) 保存在最低位地址 (RISC) 2) 字由最低位字节的字节地址寻址 — **网络字节序**

**小端模式:** 1) 最低位字节 (LSB, Least Significant Bit) 保存在最低位地址 (CISC) 2) 字由最低位字节的字节地址寻址



0x12345678字数据的大小端存储方式

# RISC系统举例—ARM (1/2)

## 指令格式—32位

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	8	7	5	4	3	0		
Cond		0 0		I	Opcode			S	Rn			Rd		Operand 2								Data Processing PSR Transfer
Cond		0 0 0 0 0 0						A	S	Rd			Rn		Rs		1 0 0 1		Rm		Multiply	
Cond		0 0 0 1 0					B	0 0		Rn			Rd		0 0 0 0		1 0 0 1		Rm		Single Data Swap	

## 寻址方式

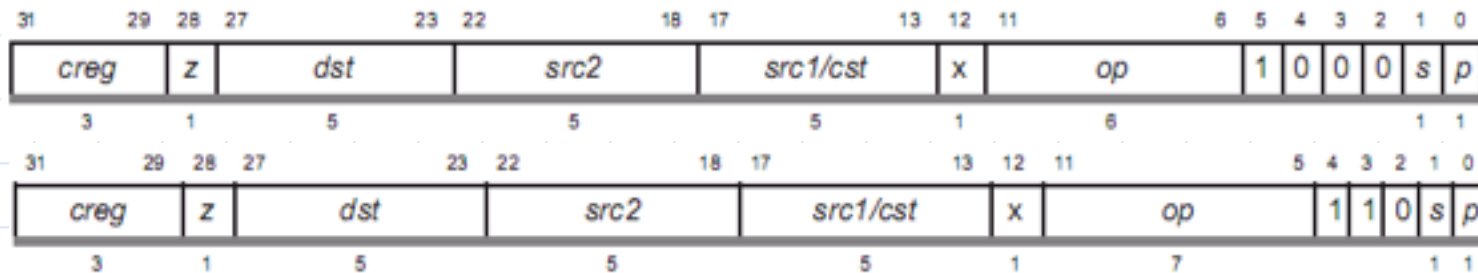
- |            |          |            |
|------------|----------|------------|
| 1. 寄存器寻址   | 2. 立即寻址  | 3. 寄存器移位寻址 |
| 4. 寄存器间接寻址 | 5. 基址寻址  | 6. 多寄存器寻址  |
| 7. 堆栈寻址    | 8. 块拷贝寻址 | 9. 相对寻址    |

## 处理器运行状态

- |       |        |        |       |
|-------|--------|--------|-------|
| 1. 用户 | 2. 中断  | 3. 快中断 | 4. 管理 |
| 5. 中止 | 6. 未定义 | 7. 系统  |       |

# RISC系统举例—TI C6000 DSP（2/2）

## 指令格式—32位



ADD

AND

## 寻址方式

1. 寄存器间接寻址
2. 寄存器相对寻址
3. 基址加变址寻址
4. 循环寻址

## 处理器运行状态

1. 系统

# CISC系统举例—Intel x86

## 指令格式—不定长（最长13字节）

Instruction Prefixes	Opcode	ModR/M	SIB	Displacement	Immediate
Up to four prefixes of 1 byte each (optional)	1-, 2-, or 3-byte opcode	1 byte (if required)	1 byte (if required)	Address displacement of 1, 2, or 4 bytes or none	Immediate data of 1, 2, or 4 bytes or none

## 寻址方式

1. 寄存器寻址
2. 立即寻址
3. 直接寻址
4. 寄存器间接寻址
5. 寄存器相对寻址
6. 基址加变址寻址
7. 相对基址加变址寻址

## 处理器运行状态

1. 实模式
2. 保护模式
3. 虚拟8086

# 课程大纲

 系统结构内容补充

 ARM系统概述

 ARM指令集简介

# ARM公司简介

ARM是Advanced RISC Machines的缩写，它是一家总部位于英国的微处理器行业知名企业，该企业设计了大量高性能、廉价、耗能低的RISC处理器。

**ARM** 公司的特点是只设计芯片，而不生产。它将技术授权给世界上许多著名的半导体、软件和OEM厂商，并提供服务。



# ARM应用概述

- 基于ARM技术的微处理器应用约占据了32位嵌入式微处理器50%以上的市场份额。
- 全球80%的GSM/3G手机、99%的CDMA手机以及绝大多数PDA产品均采用ARM体系的嵌入式处理器。
- 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Intel、三星、飞利浦。
- 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处理器的特点

ARM处理器的3大特点如下：

- ◆ 小体积、低功耗、成本低、高性能；
- ◆ 16位/32位双指令集；
- ◆ 全球众多的合作伙伴。

当前ARM体系结构的扩充包括：

- ◆ Thumb: 16位指令集，用以改善代码密度；
- ◆ DSP：用于DSP应用的算术运算指令集；
- ◆ Jazeller：允许直接执行Java代码。

ARM处理器系列提供的解决方案包括：

- ◆ 在无线、消费电子和图像应用方面的开放平台；
- ◆ 智能卡和SIM卡的安全应用。

# ARM系列产品表示

□ ARM系列产品丰富，以ARM7为例，内核ARM7TDMI表示为：

- ✓ ARM7： ARM系列具有32位整数运算核
- ✓ T： 内含16位压缩指令集Thumb
- ✓ D： 支持片内Debug调试
- ✓ M： 采用增强型乘法器(Multiplier)
- ✓ I： 内含嵌入式ICE宏单元

□ ARM系列产品的后缀提供了各种形式与功能的选择：

- ✓ -S： 可综合的软核Softcore
- ✓ -E： 具有DSP的功能
- ✓ -J： Jazeller， 允许直接执行Java字节码

# ARM架构

ARM架构自诞生至今, 已经发生了很大的演变, 至今已定义7种不同的版本:

- V1版架构
- V2版架构
- V3版架构
- V4版架构
- V5版架构
- V6版架构
- V7版架构

# 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
- 软件中断指令
- 寻址空间: 64M字节 ( $2^{26}$ )

# 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

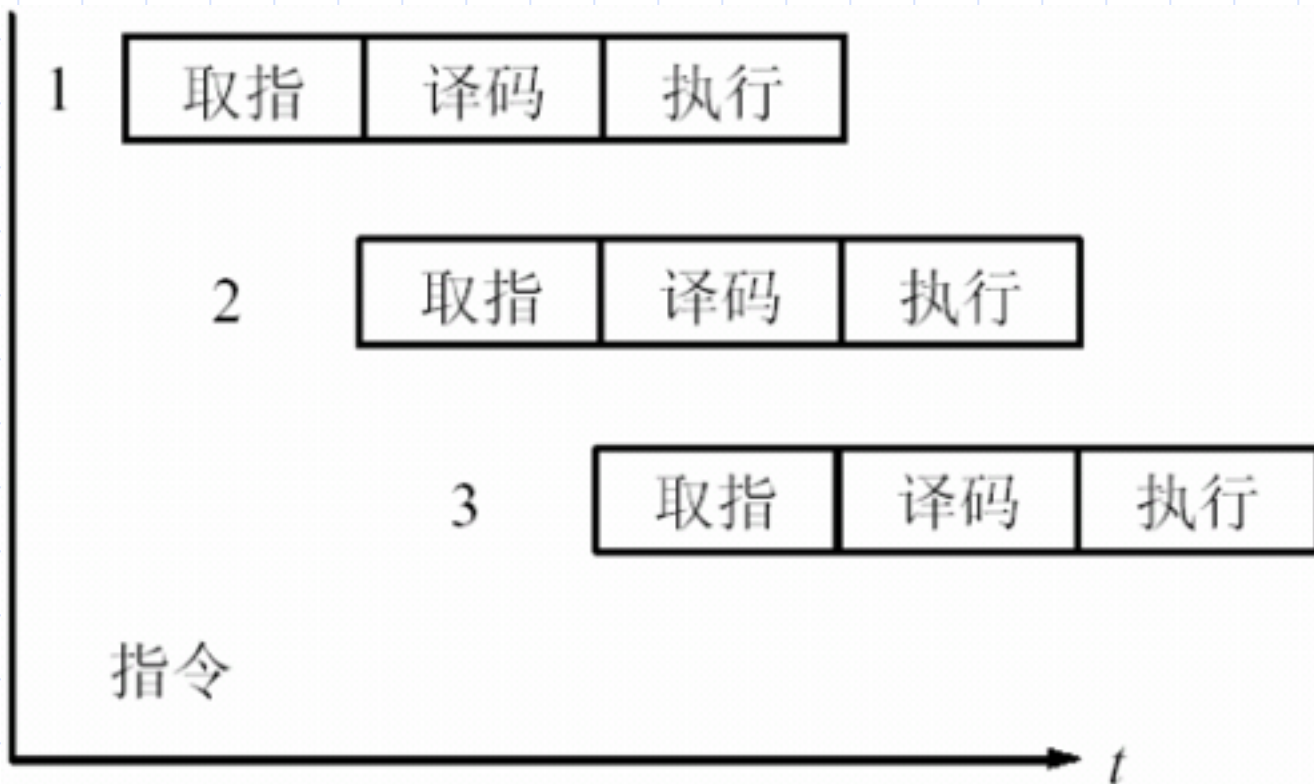
# V3版架构

- ❑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)和程序状态保存寄存器SPSR(Saved Program Status Register)。
- ❑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版架构是应用广泛的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（S3C2410）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

# ARM7 TDMI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
  - ✓ BRK中断指令
  - ✓ 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

- ARMv6，发布于2001年10月，是广泛流行的架构
- 形成ARM11系列嵌入式处理器
- 特点
  - 增强的Cache结构
    - 支持实地址Cache
    - 减少Cache的刷新和重载
    - 减少上下文切换的开销
  - 加强媒体处理能力，使MPEG4编码/解码/音频处理加快一倍
  - 低功耗：0.13um工艺，1.2v下，ARM11功耗0.4mW/MHz

## □ 增强的异常和中断处理

- 使实时任务的处理更加迅速

- 支持Unaligned和Mixed-endian数据访问

- 使数据共享、软件移植更简单，也有利于节省存储器空间

## □ 用户定制能力强

- 提供可综合版本和半定制硬核两种实现

- 客户根据自己的半导体工艺开发出各有特色的处理器内核

- 采用了易于综合的流水线结构

## □ 保持了100%的二进制向下兼容

## □ 支持多处理器系统MPCORE（由2-4个ARM11内核组成）

# V7版架构

- ARMv7，发布于2005年
- 超标量流水线，能够同时执行多条指令
- 可调尺寸的L2
- Thumb-2技术
  - 在Thumb代码压缩技术的基础上发展出来的，保持代码兼容性。
  - Thumb-2技术比纯32位代码少使用31%的内存，降低系统开销
  - 能够提供比Thumb技术的解决方案高出38%的性能表现
- NEON技术
  - 将DSP和媒体处理能力提高了近4倍
  - 对H. 264和MP3等媒体编解码提供加速
  - 支持改良的浮点运算，满足3D图形和游戏物理应用
- 支持改良运行环境，支持不断增加的JIT和DAC技术

# ARM的七种工作模式

处理器模式	说明	备注
用户	正常程序执行模式	不能直接切换到其它模式
系统 (sys)	运行操作系统的特权任务	与用户模式类似，但具有可以直接切换到其它模式等特权
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
中断	用于通用中断处理	IRQ异常响应时进入此模式
管理	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

## □ 模式切换的方法

- 软件控制
- 外部中断
- 异常处理

# ARM体系结构的主要特征

- 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操作数的移位操作
- 能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

# 体系结构和内存速度

## ■ Load/Store（内存读写）体系

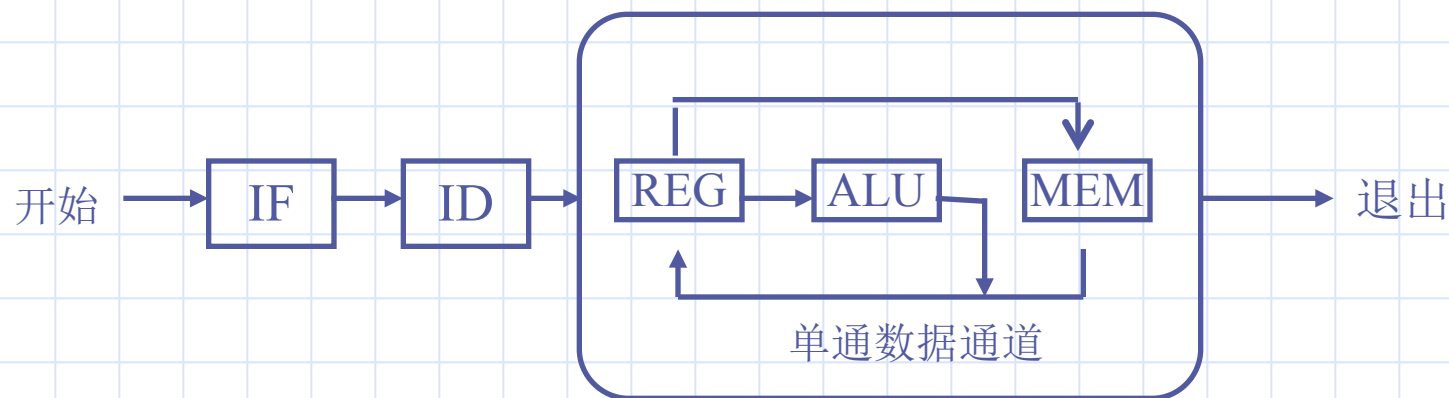
- 用来简化CPU设计而且能改善性能
  - 内存墙：CPU越来越比内存快
  - 内存访问拖慢了CPU，限制了编译器的优化能力
  - 修改指令集，使得大多数指令与内存无关
- 数据处理指令只能访问寄存器
  1. 把数据装入寄存器（load）
  2. 处理数据（计算）
  3. 把数据存回内存（store）
- 寄存器越多效率越高

## ■ 寄存器/存储器体系结构

- 数据处理指令可以访问内存或寄存器
- 相对于较慢的CPU速度（如低于50MHz），内存墙不是非常明显

# ARM体系结构的主要特征—Load/Store

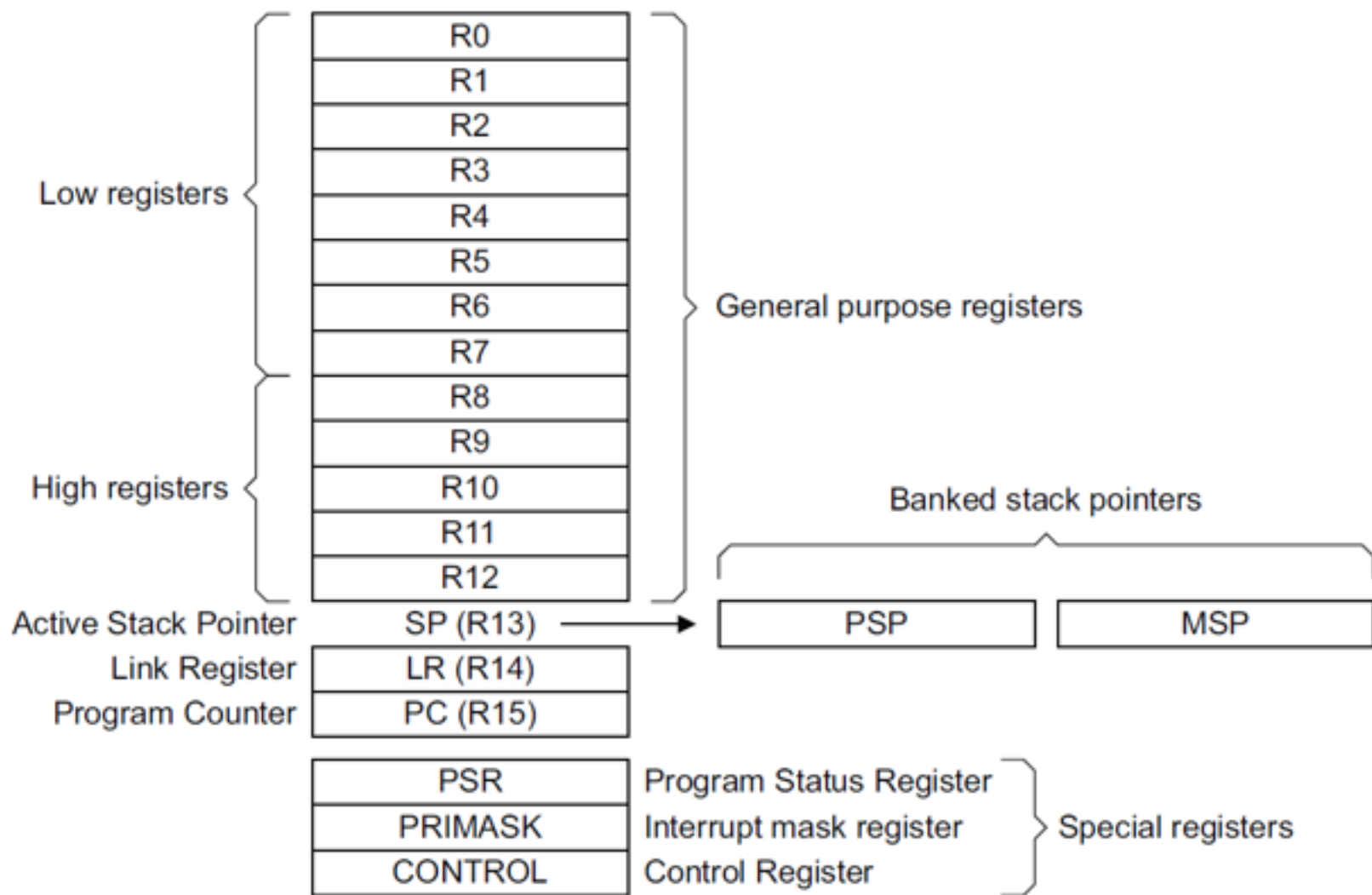
- 只对存放在寄存器的数据进行处理
- Load/Store操作：从存储器中读某个值，操作完后再将其放回存储器中
- 对于存储器中的数据，只能使用load/store指令进行存取



ARM: Load/Store结构



# ARM处理器核心寄存器



# ARM体系结构的主要特征—大量的寄存器

## ◆ 37个寄存器

- 31个通用32位寄存器
- 6个状态寄存器，R16—CPSR（当前程序状态寄存器），5个SPSR（程序状态保存寄存器），用于当异常发生时保存CPSR的状态
- 每一种处理器模式中，可见：15个通用寄存器(R0～R14，其中：R13—堆栈指针(sp)、R14—链接寄存器(lr))、R15：程序计数器PC，以及1～2个状态寄存器

## ◆ 可见的寄存器取决于处理器的模式

## ◆ 其它寄存器(the banked registers)在处理器状态切换时被屏蔽

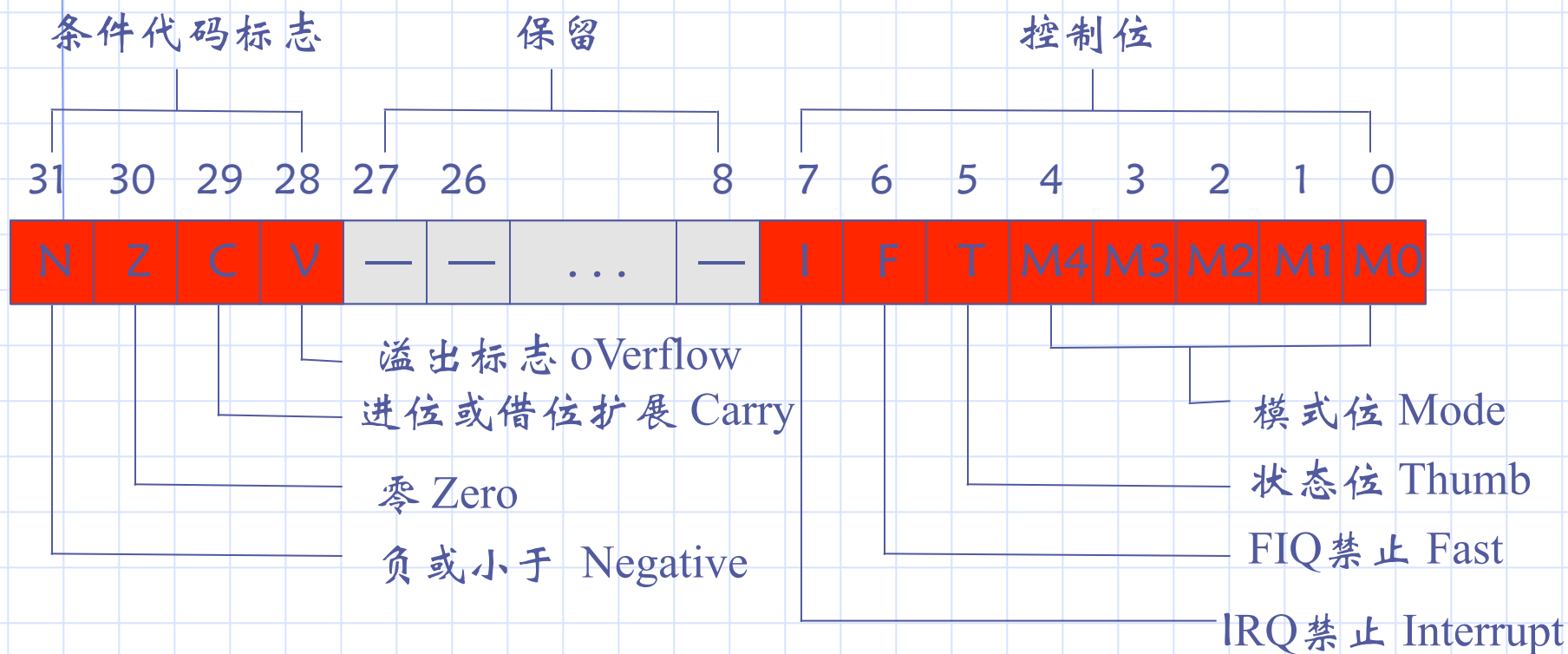
寄存器多的好处是什么呢？

# ARM体系结构的主要特征—大量的寄存器

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													
		用户		系统		管理		中止		未定义		中断		快中断	
通用寄存器	R0(a1)	R0													
	R1(a2)	R1													
	R2(a3)	R2													
	R3(a4)	R3													
	R4(v1)	R4													
	R5(v2)	R5													
	R6(v3)	R6													
	R7(v4)	R7													
	R8(v5)	R8												R8_fiq	
	R9(SB,v6)	R9												R9_fiq	
	R10(SL,v7)	R10												R10_fiq	
	R11(FP,v8)	R11												R11_fiq	
	R12(IP)	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													
状态寄存器	CPSR	CPSR													
	SPSR	无				SPSR_abt		SPSR_abt		SPSR_und		SPSR_irq		SPSR_fiq	

# ARM的程序状态寄存器（CPSR）简介

CPSR寄存器的格式



# ARM的程序状态寄存器（CPSR）简介

CPSR模式位设置表

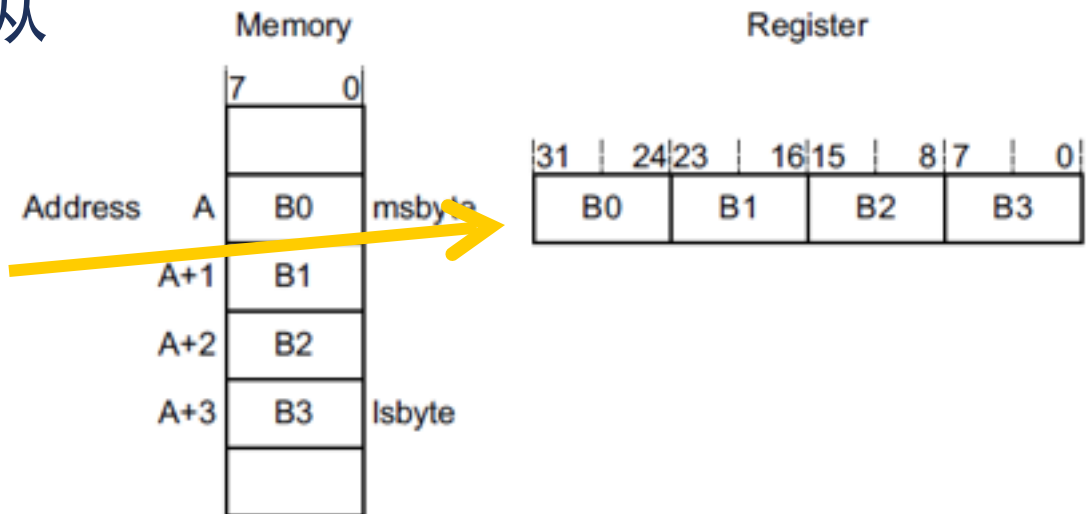
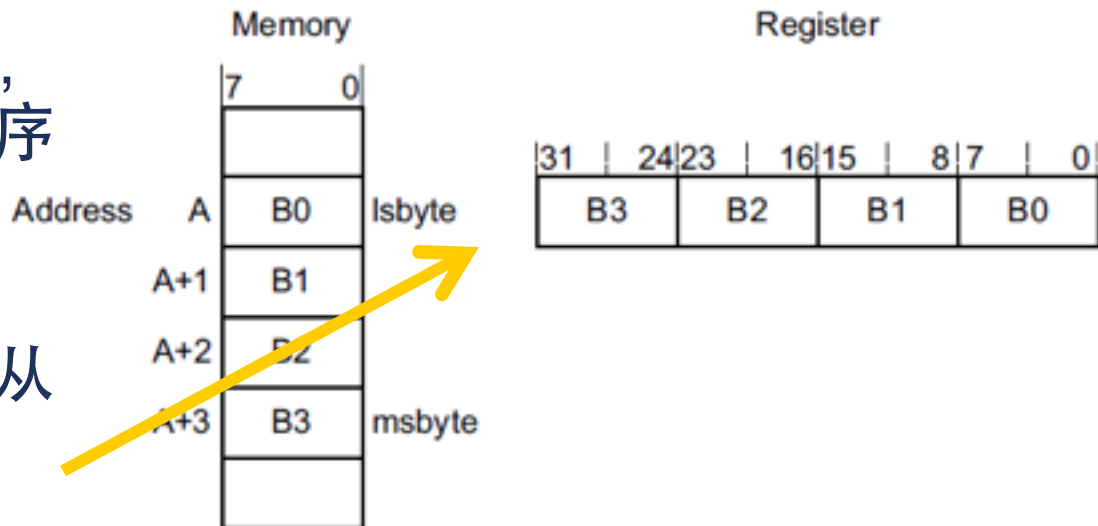
M[4:0]	模式	可见的Thumb状态寄存器	可见的ARM状态寄存器
10000	用户	R0~R7,SP,LR,PC,CPSR	R0~R14,PC, CPSR
10001	快中断	R0~R7,SP_fiq,LR_fiq,PC,CPSR, SPSR_fiq	R0~R7,R8_fiq~R14_fiq,PC, CPSR, SPSR_fiq
10010	中断	R0~R7,SP_irq,LR_irq,PC,CPSR, SPSR_fiq	R0~ R12,R13_irq,R14_irq,PC,
10011	管理	R0~R7,SP_svc,LR_svc,PC,CPSR, SPSR_svc	R0~R12,R13_svc,R14_svc, PC,CPSR, SPSR_svc
10111	中止	R0~R7,SP_abt,LR_abt,PC,CPSR, SPSR_abt	R0~ R12,R13_abt,R14_abt,PC, CPSR, SPSR_abt
11011	未定义	R0~ R7,SP_und,LR_und,PC,CPSR, SPSR_und	R0~R12,R13_und,R14_und, PC,CPSR,SPSR_und
11111	系统	R0~R7,SP,LR,PC,CPSR	R0~R14,PC, CPSR

# 大小端

- 对于一个多字节的值，这些字节以怎样的顺序存储？

- 小端（低位在前）：从最低位的字节开始

- 大端（高位在前）：从最高位的字节开始



# ARMv6-M的大小端

---

- 指令始终是小端的
- 对私有的外围部件总线的读和写 (**load/store**) 始终是小端的
- 数据：取决于芯片的具体实现，或是启动时的配置
  - Kinetis处理器是小端的

# 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：

<code>&lt;opcode&gt; {&lt;cond&gt;} {S}      &lt;Rd&gt; ,&lt;Rn&gt;{ ,&lt;operand2&gt;}</code>
--

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

操作码	条件助记符	标志	含义
0000	EQ	Z=1	相等
0001	NE	Z=0	不相等
0010	CS/HS	C=1	无符号数大于或等于
0011	CC/LO	C=0	无符号数小于
0100	MI	N=1	负数
0101	PL	N=0	正数或零
0110	VS	V=1	溢出
0111	VC	V=0	没有溢出
1000	HI	C=1,Z=0	无符号数大于
1001	LS	C=0,Z=1	无符号数小于或等于
1010	GE	N=V	有符号数大于或等于
1011	LT	N!=V	有符号数小于
1100	GT	Z=0,N=V	有符号数大于
1101	LE	Z=1,N!=V	有符号数小于或等于
1110	AL	任何	无条件执行 (指令默认条件)
1111	NV	任何	从不执行(不要使用)

# ARM体系结构的主要特征——条件执行

## ◆ARM指令集——条件码

示例：

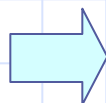
C代码：

```
If (a > b)
```

```
    a++;
```

```
Else
```

```
    b++;
```



对应的汇编代码：

```
CMP    R0,R1        ;R0 (a) 与R1 (b) 比较
```

```
ADDEHI 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

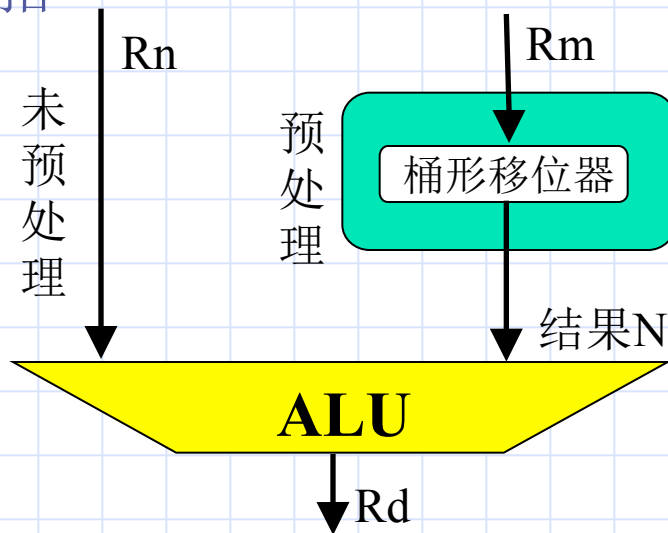
■Rm, shift——寄存器移位方式

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体系结构的主要特征—Thumb指令



16位Thumb指令集具有灵活、小巧的特点

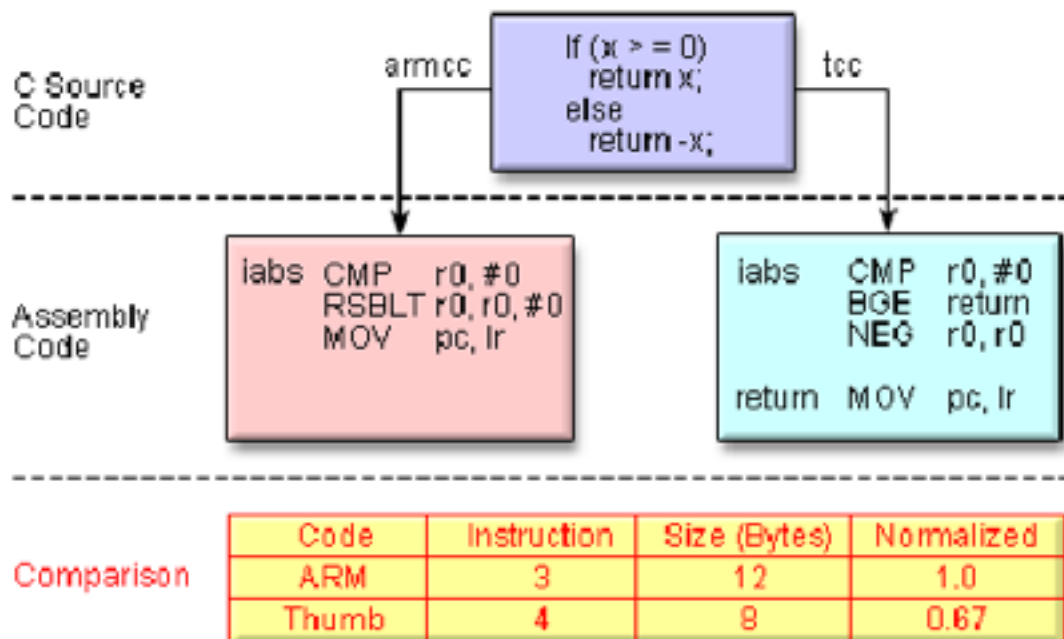
32位ARM指令集支持ARM核所有的特性，具有高效、快速的特点

# ARM、Thumb和 Thumb-2指令

- **ARM指令是为资源丰富、高性能计算系统而优化设计的**
  - 深度流水线处理器、高的时钟频率、宽的内存总线（如32位）
- **低端嵌入式计算系统是不同的**
  - 较慢的时钟频率、浅的流水线
  - 不同的成本因素——比如代码大小更为重要、比特和字节运算很重要
- **修改了ARM指令集来适应低端嵌入式计算**
  - 1995: Thumb指令集
    - 16位指令
    - 降低了内存需求但是也降低了性能
  - 2003: Thumb-2指令集
    - 增加了一些32位指令
    - 改进了速度，只需很小的内存额外开销
  - CPU根据是Thumb状态还是ARM状态来解码指令，状态由T位控制

# ARM体系结构的主要特征—Thumb指令

- 若对系统的性能有较高要求，应使用**32位**的存储系统和**ARM指令**
- 若对系统的成本及功耗有较高要求，则应使用**16位**的存储系统和**Thumb指令集**。
- 若两者结合使用，充分发挥其各自的优点，会取得更好的效果。



# 课程大纲

 系统结构内容补充

 ARM系统概述

 ARM指令集简介



# 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处理器具有8种基本寻址方式。

- 1.立即寻址;
- 2.寄存器寻址;
- 3.寄存器移位寻址;
- 4.寄存器间接寻址;
- 5.基址寻址;
- 6.多寄存器寻址;
- 7.堆栈寻址;
- 8.相对寻址。

# ARM处理器寻址方式

## ◆寻址方式分类——立即寻址

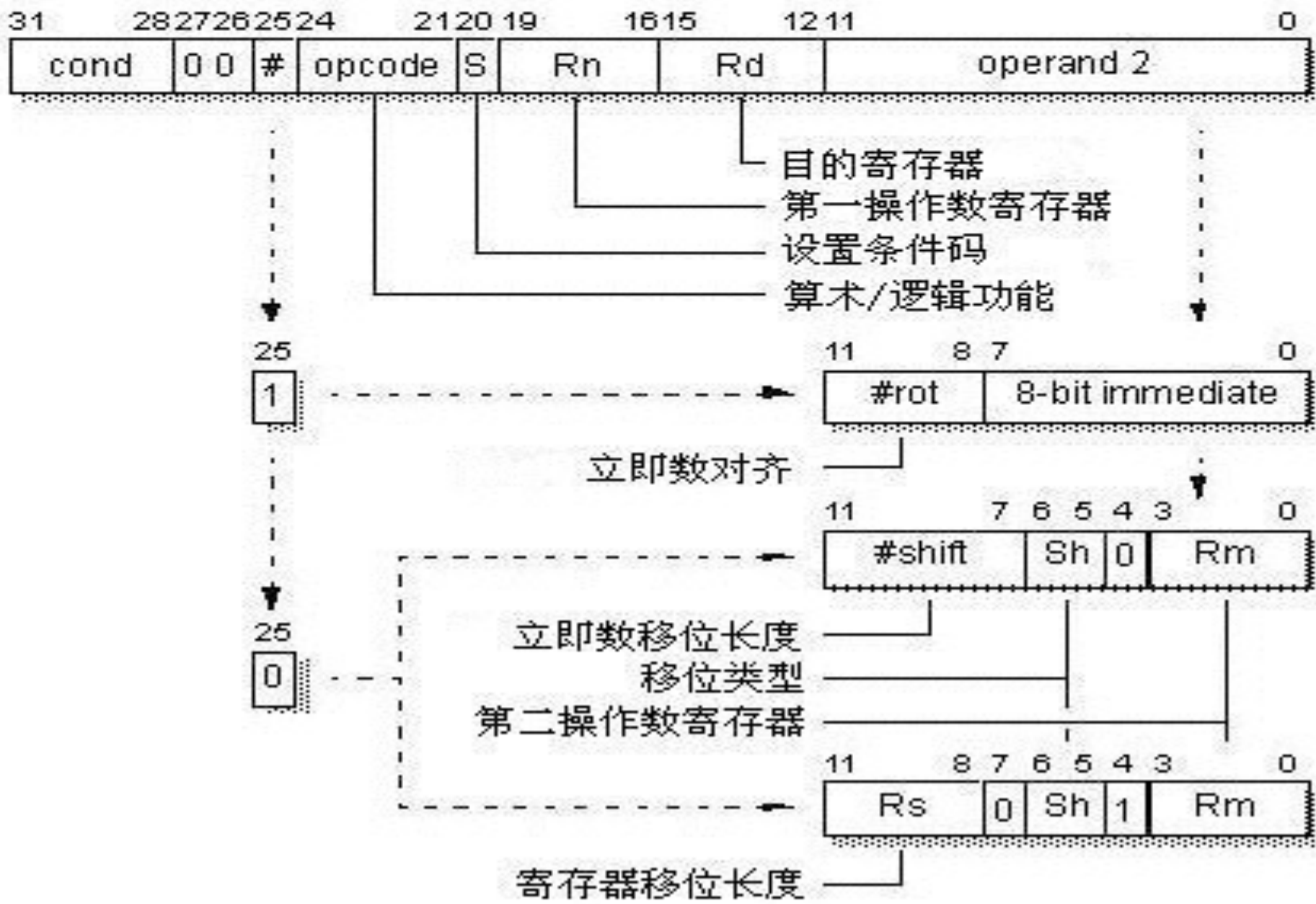
立即寻址指令中的操作码字段后面的地址码部分即是操作数本身，也就是说，**数据就包含在指令当中**，取出指令也就取出了可以立即使用的操作数(这样的数称为立即数)。立即寻址指令举例如下：

**SUBS     R0,R0,#1**                ;R0减1，结果放入R0，并且影响标志位

**MOV      R0,#0xFF000**        ;将立即数0xFF000装入R0寄存器

# 有效立即数问题

- ◆ 在ARM数据处理指令中，当参与操作的第二操作数为立即数型时，每个立即数都是采用一个8位的常数循环右移偶数位而间接得到。其中循环右移的位数由一个4位二进制的两倍表示，如果立即数记作<immediate>，8位常数记作immed\_8，4位的循环右移值记作rotate\_imm，有效的立即数是由一个8位的立即数循环右移偶数位得到。
- ◆ 因此有效立即数immediate可以表示成：
  - <immediate>=immed\_8循环右移



# 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** ;将R2的值存入R1

**SUB R0,R1,R2** ;将R1的值减去R2的值，结果保存到R0

# ARM处理器寻址方式

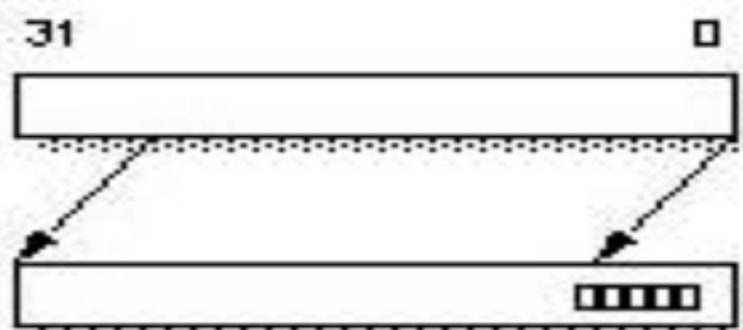
## ◆寻址方式分类——寄存器移位寻址

寄存器移位寻址是ARM指令集特有的寻址方式。当第2个操作数是寄存器移位方式时，第2个寄存器操作数在与第1个操作数结合之前，选择进行移位操作。寄存器移位寻址指令举例如下：

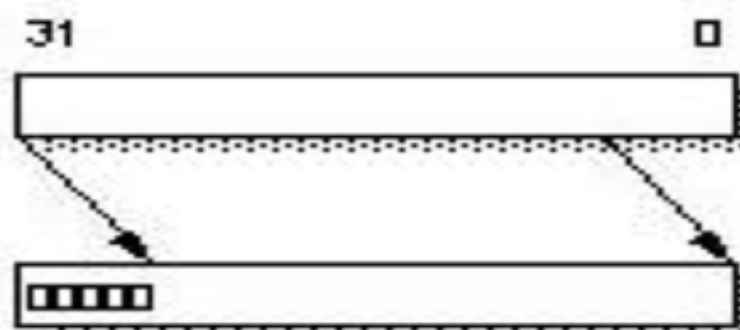
```
MOV    R0,R2,LSL #3           ;R2的值左移3位, 结果放入R0,
                                ;即是R0=R2×8
```

**ANDS R1,R1,R2,LSL R3** ;R2的值左移R3位,然后和R1相  
相 ;"与"操作,结果放入R1

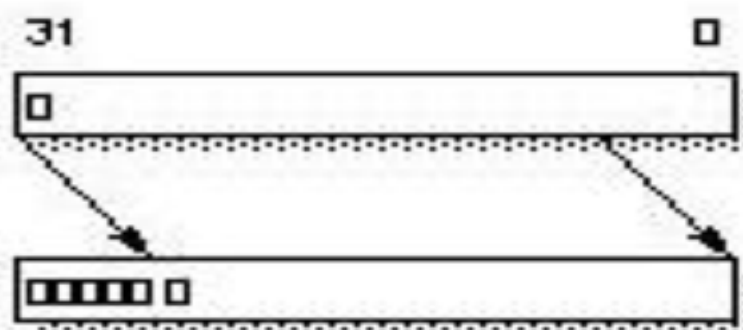
- ◆ LSL: 逻辑左移, 空出的最低有效位用0填充。
- ◆ LSR: 逻辑右移, 空出的最高有效位用0填充。
- ◆ ASL: 算术左移, 由于左移空出的有效位用0填充, 因此 它与LSL同义。
- ◆ ASR: 算术右移, 算术移位对象是带符号数, 移位过程中必须保持操作数的符号不变。如果源操作数是正数, 空出的最高有效位用0填充, 如果是负数用1填充。
- ◆ ROR: 循环右移, 移出的字的最低有效位依次填入空出的最高有效位。
- ◆ RRX: 带扩展的循环右移。将寄存器的内容循环右移1位, 空位用原来C标志位填充。



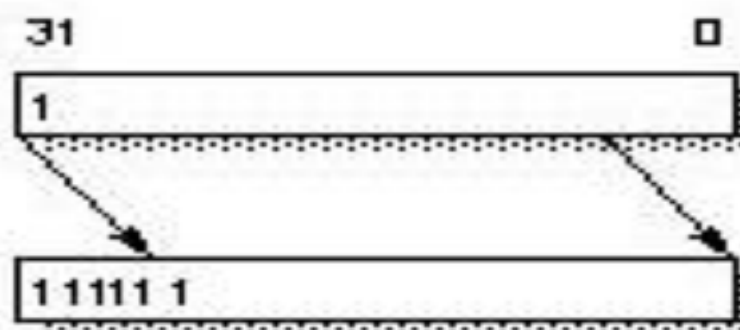
LSL #5



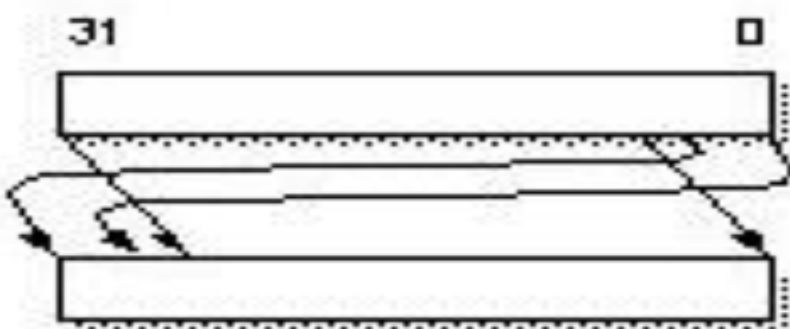
LSR #5



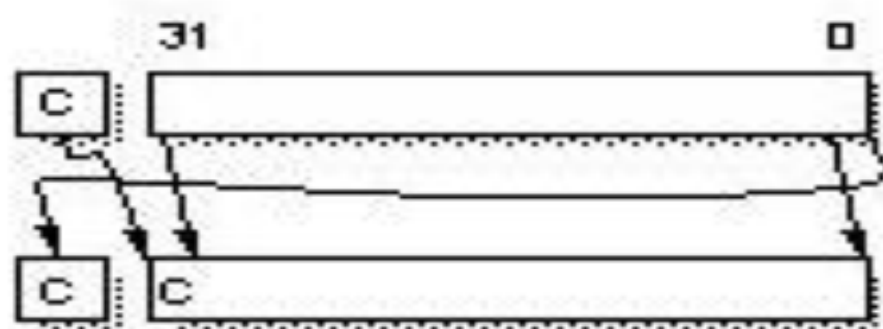
ASR #5 , positive operand



ASR #5 , negative operand



ROR #5



RRC



# 寄存器寻址

◆移位位数可以用立即数方式或者寄存器方式给出,如下所示:

- $\text{ADD } R3, R2, R1, \text{LSR } \#2$  ;  $R3 \leftarrow R2 + R1 \div 4$
- $\text{ADD } R3, R2, R1, \text{LSR } R4$ ;  $R3 \leftarrow R2 + R1 \div 2R4$

◆寄存器R1的内容分别逻辑右移2位、R4位（亦即 $R1 \div 4$ 、 $R1 \div 2R4$ ），再与寄存器R2的内容相加，结果放入R3中。

# ARM处理器寻址方式

## ◆寻址方式分类——寄存器间接寻址

寄存器间接寻址指令中的地址码给出的是一个通用寄存器的编号，所需的**操作数保存在寄存器**指定地址的存储单元中，即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

**SWP      R1, R1, [R2]**                  ;将寄存器R1的值和R2指定的存储

   ;单元的内容交换

**STR      R0, [R1]**                      /\* [R1] ← R0 \*/

# ARM处理器寻址方式

## ◆ 寻址方式分类——基址寻址

基址寻址就是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。基址寻址用于访问基址附近的存储单元, 常用于查表、数组操作、功能部件寄存器访问等。寄存器间接寻址是偏移量为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R0的值寄存 ; 到保存到R1指定的存储单元

# 基址加偏址寻址

## ◆前变址模式:

■ LDR R0, [R1, # 4] ; R0 ← [R1 + 4]

## ◆自动变址模式:

■ LDR R0, [R1, # 4]! ; R0 ← [R1 + 4]、R1 ← R1 + 4

## ◆后变址模式:

■ LDR R0, [R1], # 4 ;  
R0 ← [R1]、R1 ← R1 + 4

# 基址加偏址寻址

◆ 基址寄存器的地址偏移可以是一个立即数，也可以是另一个寄存器，并且在加到基址寄存器前还可以经过移位操作，如下所示：

- `LDR r0, [r1, r2]; r0 ← mem32[r1+r2]`
- `LDR r0, [r1, r2, LSL #2]; r0 ← [r1+r2*4]`

◆ 但常用的是立即数偏移的形式，地址偏移为寄存器形式的指令很少使用。

# ARM处理器寻址方式

## ◆寻址方式分类——堆栈寻址

堆栈是一个按特定顺序进行存取的存储区，操作顺序为“后进先出”。堆栈寻址是隐含的，它使用一个专门的寄存器(堆栈指针)指向一块存储区域(堆栈)，指针所指向的存储单元即是堆栈的栈顶。存储器堆栈可分为两种：

- 向上生长：向高地址方向生长，称为递增堆栈
- 向下生长：向低地址方向生长，称为递减堆栈

# ARM处理器寻址方式

## ◆寻址方式分类——堆栈寻址

根据堆栈指针指向是否为空，以及堆栈生长方向，可以组合出四种类型的堆栈方式：

- **满递增**：堆栈向上增长，堆栈指针指向内含有效数据项的最高地址。指令如LDMFA、STMFA等；
- **空递增**：堆栈向上增长，堆栈指针指向堆栈上的第一个空位置。指令如LDMEA、STMEA等；
- **满递减**：堆栈向下增长，堆栈指针指向内含有效数据项的最低地址。指令如LDMFD、STMFD等；
- **空递减**：堆栈向下增长，堆栈指针向堆栈下的第一个空位置。指令如LDMED、STMED等。

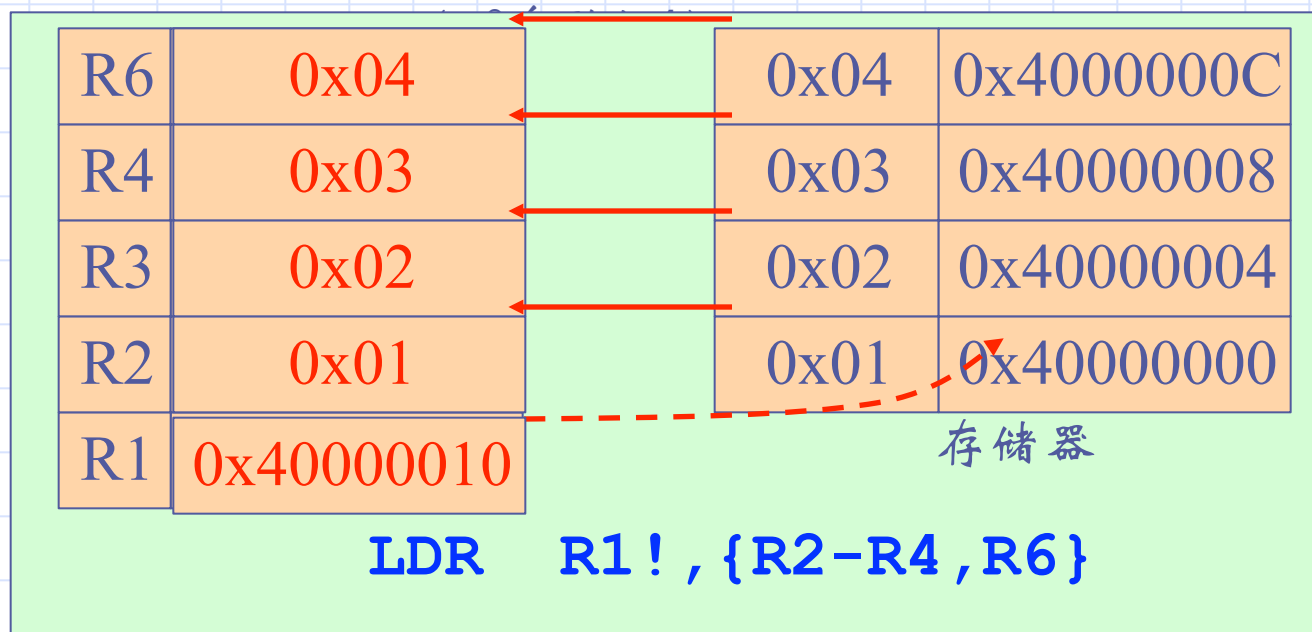
# ARM处理器寻址方式

## ◆ 寻址方式分类——多寄存器寻址

一次可传送几个寄存器值，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-R7, R12}** ; 将R1指向的单元中的数据读出到  
; R2~R7、R12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





# 多寄存器load和store指令的堆栈和块拷贝对照

		递增		递减	
		满	空	满	空
增值	先增	STM B STMFA			LDM B LDMED
	后增		STM A STMEA	LDM A LDMFD	
减值	先减		LDMDB LDMEA	STMDB STMFD	
	后减	LDMDA LDMFA			STMDA STMED

# ARM处理器寻址方式

## ◆ 寻址方式分类——相对寻址

相对寻址是基址寻址的一种变通。由程序计数器PC提供基准地址，指令中的地址码字段作为偏移量，两者相加后得到的地址即为操作数的有效地址。相对寻址指令举例如下：

	BL	SUBR1	;调用到SUBR1子程序
	BEQ	LOOP	;条件跳转到LOOP标号处
	...		
LOOP	MOV	R6, #1	
	...		
SUBR1	...		

# ARM指令集——指令格式

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	8	7	5	4	3	0		
Cond	0	0		I	Opcode			S	Rn			Rd			Operand 2							Data Processing PSR Transfer
Cond	0 0 0 0 0 0						A	S	Rd			Rn			Rs		1 0 0 1		Rm		Multiply	
Cond	0 0 0 1 0					B	0 0		Rn			Rd			0 0 0 0		1 0 0 1		Rm		Single Data Swap	
Cond	0	1		I	P	U	B	W	L	Rn			Rd			offset						Single Data Transfer
Cond	0	1	1	XXXXXXXXXXXXXXXXXXXXX															1	XXXX		Undefined
Cond	1	0	0	P	U	S	W	L	Rn			Register List									Block Data Transfer	
Cond	1	0	1	L	offset																Branch	
Cond	1	1	0	P	U	N	W	L	Rn			CRd		CP#		offset					Coproc Data Transfer	
Cond	1	1	1	0	CP Opc			CRn			CRd		CP#		CP		0	CRm		Coproc Data Operation		
Cond	1	1	1	0	CP Opc			L	CRn			Rd		CP#		CP		1	CRm		Coproc Register Transfer	
Cond	1	1	1	1	ignored by processor																Software Interrupt	

# op-code

操作码[24: 21]	助记符	意义	效果
0000	AND	逻辑位与	$Rd = Rn \text{ AND } Op2$
0001	EOR	逻辑位异或	$Rd = Rn \text{ EOR } Op2$
0010	SUB	减	$Rd = Rn - Op2$
0011	RSB	反向减	$Rd = Op2 - Rn$
0100	ADD	加	$Rd = Rn + Op2$
0101	ADC	带进位加	$Rd = Rn + Op2 + C$
0110	SBC	带进位减	$Rd = Rn - Op2 + C - 1$
0111	RSC	反向带进位减	$Rd = Op2 - Rn + C - 1$
1000	TST	测试	根据 $Rn \text{ AND } Op2$ 设置条件码
1001	TEQ	测试相等	根据 $Rn \text{ EOR } Op2$ 设置条件
1010	CMP	比较	根据 $Rn - Op2$ 设置条件码
1011	CMN	负数比较	根据 $Rn + Op2$ 设置条件码
1100	ORR	逻辑位或	$Rd = Rn \text{ OR } Op2$
1101	MOV	传送	$Rd = Op2$
1110	BIC	位清零	$Rd = Rn \text{ AND NOT } Op2$
1111	MVN	求反	$Rd = \text{NOT } Op2$

# ARM指令集介绍——基本指令格式

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	源寄存器1(Rn)	源寄存器2(Rm)
ADD r3,r1,r2	r3	r1	r2

# ARM指令分类

- 存储器访问指令
- 数据处理指令
- 乘法指令
- ARM分支指令
- 杂项指令

# ARM存储器访问指令——单寄存器加载

助记符	说明	操作	条件码位置
LDR Rd, addressing	加载字数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond}
LDRB Rd, addressing	加载无符号字节数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} B
LDRT Rd, addressing	以用户模式加载字数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} T
LDRBT Rd, addressing	以用户模式加载无符号字节数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} BT
LDRH Rd, addressing	加载无符号半字数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} H
LDRSB Rd, addressing	加载有符号字节数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} SB
LDRSH Rd, addressing	加载有符号半字数据	$Rd \leftarrow [addressing],$ addressing索引	LDR {cond} SH

# ARM存储器访问指令——单寄存器存储

助记符	说明	操作	条件码位置
STR Rd, addressing	存储字数据	[addressing] ← Rd, addressing索引	STR {cond}
STRB Rd, addressing	存储字节数据	[addressing] ← Rd, addressing索引	STR {cond} B
STRT Rd, addressing	以用户模式存储字数据	[addressing] ← Rd, addressing索引	STR {cond} T
STRBT Rd, addressing	以用户模式存储字节数据	[addressing] ← Rd, addressing索引	STR {cond} BT
STRH Rd, addressing	存储半字数据	[addressing] ← Rd, addressing索引	STR {cond} H

LDR/STR指令用于对内存变量的访问、内存缓冲区数据的访问、查表、外围部件的控制操作等。若使用LDR指令加载数据到PC寄存器，则实现程序跳转功能，这样也就实现了程序跳转。

所有单寄存器加载/存储指令可分为“字和无符号字节加载存储指令”和“半字和有符号字节加载存储指令”。



# 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 $\leftarrow$ [Rn...], Rn 回写等	LDM{cond} {mode}
STM{mode} Rn{!},reglist	多寄存器存储	[Rn...] $\leftarrow$ 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数据交换	$Rd \leftarrow [Rn], [Rn] \leftarrow Rm$ ( $Rn \neq Rd$ 或 $Rm$ )	SWP {cond}
SWPB Rd,Rm,Rn	寄存器和存储器字节数据交换	$Rd \leftarrow [Rn], [Rn] \leftarrow Rm$ ( $Rn \neq Rd$ 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, 若Rm与Rd相同, 则为寄存器与存储器内容进行互换; 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数据处理指令——数据传送

助记符		说明	操作	条件码位置
MOV	Rd, operand2	数据传送	$Rd \leftarrow \text{operand2}$	MOV {cond} {S}
MVN	Rd, operand2	数据非传送	$Rd \leftarrow (\sim \text{operand2})$	MVN {cond} {S}

注：当后缀S时，这些指令根据结果更新标志N和Z，在计算Operand2时更新标志C，不影响标志V。

# ARM数据处理指令——算术运算

助记符		说明	操作	条件码位置
ADD	Rd, Rn, operand2	加法运算指令	$Rd \leftarrow Rn + \text{operand2}$	ADD {cond} {S}
SUB	Rd, Rn, operand2	减法运算指令	$Rd \leftarrow Rn - \text{operand2}$	SUB {cond} {S}
RSB	Rd, Rn, operand2	逆向减法指令	$Rd \leftarrow \text{operand2} - Rn$	RSB {cond} {S}
ADC	Rd, Rn, operand2	带进位加法	$Rd \leftarrow Rn + \text{operand2} + \text{Carry}$	ADC {cond} {S}
SBC	Rd, Rn, operand2	带进位减法指令	$Rd \leftarrow Rn - \text{operand2} - (\text{NOT}) \text{Carry}$	SBC {cond} {S}
RSC	Rd, Rn, operand2	带进位逆向减法指令	$Rd \leftarrow \text{operand2} - Rn - (\text{NOT}) \text{Carry}$	RSC {cond} {S}

注：这些指令影响N，Z，C和V标志位。

# ARM数据处理指令——逻辑运算指令

助记符		说明	操作	条件码位置
AND	Rd, Rn, operand2	逻辑与操作指令	$Rd \leftarrow Rn \ \& \ operand2$	AND {cond} {S}
ORR	Rd, Rn, operand2	逻辑或操作指令	$Rd \leftarrow Rn \mid operand2$	ORR {cond} {S}
EOR	Rd, Rn, operand2	逻辑异或操作指令	$Rd \leftarrow Rn \wedge operand2$	EOR {cond} {S}
BIC	Rd, Rn, operand2	位清除指令	$Rd \leftarrow Rn \ \& \ (\sim operand2)$	BIC {cond} {S}

注：当后缀S时，这些指令根据结果更新标志N和Z，在计算Operand2时更新标志C，不影响标志V。

# ARM数据处理指令——比较指令

助记符		说明	操作	条件码位置
CMP	Rn, operand2	比较指令	标志N、Z、C、 $V \leftarrow Rn - \text{operand2}$	CMP {cond}
CMN	Rn, operand2	负数比较指令	标志N、Z、C、 $V \leftarrow Rn + \text{operand2}$	CMN {cond}
TST	Rn, operand2	位测试指令	标志N、Z、C、 $V \leftarrow Rn \ \& \ \text{operand2}$	TST {cond}
TEQ	Rn, operand2	相等测试指令	标志N、Z、C、 $V \leftarrow Rn \ \wedge \ \text{operand2}$	TEQ {cond}

注：这些指令影响N，Z，C和V标志位。

# ARM指令集——乘法指令

助记符		说明	操作	条件码位置
MUL	Rd, Rm, Rs	32位乘法指令	$Rd \leftarrow Rm * Rs$ ( $Rd \neq Rm$ )	MUL {cond} {S}
MLA	Rd, Rm, Rs, Rn	32位乘加指令	$Rd \leftarrow Rm * Rs + Rn$ ( $Rd \neq Rm$ )	MLA {cond} {S}
UMULL RdLo, RdHi, Rm, Rs		64位无符号乘法指令	$(RdLo, RdHi) \leftarrow Rm * Rs$	UMULL {cond} {S}
UMLAL RdLo, RdHi, Rm, Rs		64位无符号乘加指令	$(RdLo, RdHi) \leftarrow Rm * Rs +$ $(RdLo, RdHi)$	UMLAL {cond} {S}
SMULL RdLo, RdHi, Rm, Rs		64位有符号乘法指令	$(RdLo, RdHi) \leftarrow Rm * Rs$	SMULL {cond} {S}
SMLAL RdLo, RdHi, Rm, Rs		64位有符号乘加指令	$(RdLo, RdHi) \leftarrow Rm * Rs +$ $(RdLo, RdHi)$	SMLAL {cond} {S}

# 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	$PC \leftarrow \text{label}$	B{cond}
BL label	带链接的分支指令	$LR \leftarrow PC - 4, PC \leftarrow \text{label}$	BL{cond}
BX Rm	带状态切换的分支指令	$PC \leftarrow Rm$ , 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 $\pm 32M$ 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
0	0	ARM
1	1	Thumb



# ARM指令集——杂项指令

ARM指令集中有三条指令作为杂项指令，在实际应用中这三条指令非常重要。它们如下所示：

助记符	说明	操作	条件码位置
SWI <code>immed_24</code>	软中断指令	产生软中断，处理器进入管理模式	SWI{cond}
MRS <code>Rd,psr</code>	读状态寄存器指令	$Rd \leftarrow psr$ ，psr为CPSR或SPSR	MRS{cond}
MSR <code>psr_fields,</code> <code>Rd/#immed_8r</code>	写状态寄存器指令	$psr\_fields \leftarrow Rd/\#immed\_8r$ ，psr为CPSR或SPSR	MSR{cond}

# 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，需要存在里面的保存返回地址。

# 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
```

# ARM调用规范(ARM PROCEDURE CALL STANDARD)

## ◆ 使用栈的规范:

- 数据栈为FD（满递减）类型

## ◆ 在函数调用之间传递/返回参数

- 4个以下参数，由R0~R3传递，其中arg1—R0
- 大于4个参数时，通过堆栈传递
- 返回结果存在R0中
- 寄存器R4~R10用于局部变量、中间变量存储

# 混合编程

◆ 汇编语言和C/C++的混合编程通常有以下几种方式：

- ◆ ① 汇编程序中调用C程序
- ◆ ② C程序中调用汇编程序
- ◆ ③ C程序中内嵌汇编语句
- ◆ ④ 从汇编程序中访问C程序变量

```
/*C file, called by asmfile */  
int cFun(int a, int b, int c)  
{  
    return a + b + c;  
}
```

◆这里的参数传递是利用寄存器r0~r2。需要指出的是当函数的参数个数大于4时就要借助堆栈。

◆在汇编程序中使用EXPORT伪指令声明程序，使得本程序可以被其他的程序调用；在C语言中使用EXTERN关键词声明该汇编程序，这样就可以在C中使用该函数了。从C的角度，并不知道该函数的实现是用C还是汇编。

◆ 在C中内嵌的汇编指令支持大部分的ARM和Thumb指令，不过其使用与汇编文件中的指令有些不同，存在一些限制，主要有下面几个方面：

- 不能直接向PC寄存器赋值，程序跳转要使用B或者BL指令；
- 在使用物理寄存器时，不要使用过于复杂的C表达式，避免物理寄存器冲突；
- R12和R13可能被编译器用来存放中间编译结果；
- 一般不要直接指定物理寄存器，而让编译器进行分配；

◆ 内嵌汇编的标志是：\_asm或者asm关键字



◆在C程序中声明的全局变量可以被汇编程序通过地址间接访问，具体访问方法如下：

- 使用**IMPORT**伪指令声明该全局变量。
- 使用**LDR**指令读取该全局变量的内存地址，通常该全局变量的内存地址值存放在程序的数据缓冲区中。
- 根据该数据的类型，使用相应的**LDR**指令读取该全局变量的值，使用相应的**STR**指令修改该全局变量的值。

AREA global\_exp, CODE, READONLY

EXPORT asmsub

IMPORT globv ; 声明全局变量

asmsub

LDR r1, = globv ; 将内存地址读入到R1中

LDR r0, [r1] ; 将数据读入到R0中

ADD r0, r0, #2

STR r0, [r1] ; 修改 后再将值赋予变量

MOV pc, lr

END