嵌入式系统原理

The Principle of Embedded System

合肥工业大学·计算机与信息学院

2024年4月

Instruction System (也称为指令集，机器语言)，指计算机所能执行的全部指令(功能)的集合。从系统结构角度看，它实现了软件和硬件的交互联系，是表征计算机性能的重要因素。

第三章 ARM指令集与汇编程序设计

- **3.1 ARM寻址方式**
- 3.2 ARM指令集
- 3.3 ARM汇编程序设计
- 3.4 Thumb指令集

几个概念

- 指令：计算机控制各功能部件协调动作的命令。
- 指令系统(集)：微处理器所能执行的全部指令的集合。
  - ✓不同的微处理器拥有不同的指令系统。同等条件下，指令系统越强，则构成的计算机系统的功能就越强。
  - ✓指令由硬件直接识别并执行。指令系统中指令数量愈多，硬件的结构也愈复杂。
- 机器语言指令(指令机器码)：能被微处理器直接识别和执行二进制编码，是指令在计算机内部的表示形式。
- 汇编语言指令——机器指令的符号化表示形式。

指令格式

- 汇编语言指令：由操作码和操作数两部分组成。

操作码	操作数1 … 操作数n

指令的一般格式

- ✓操作码：指示指令要执行的具体操作。用助记符（一般为英文字母缩写）表示——指令助记符。
- ✓操作数：指出指令执行过程中的操作对象。可以用符号或符号地址表示。

寻址方式

- 微处理器根据指令中由操作数(地址码字段)给出的地址信息，来寻找真实物理地址的方式。
- ARM处理器具有8种基本寻址方式：

(1) 寄存器寻址　　(2) 立即寻址

(3) 寄存器偏移寻址　　(4) 寄存器间接寻址

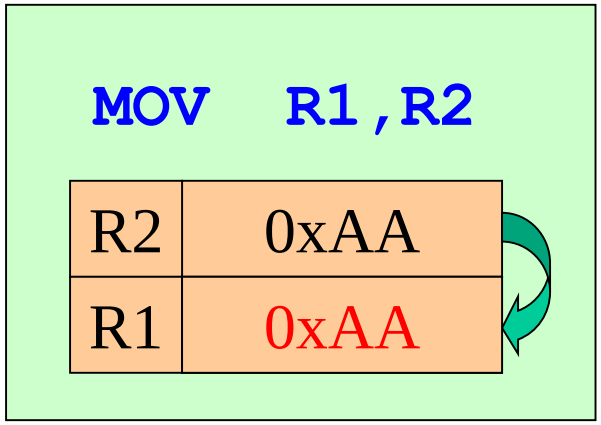
(5) 基址寻址　　(6) 多寄存器寻址

(7) 堆栈寻址　　(8) 相对寻址

(1) 寄存器寻址

- 指令中操作数指出的是寄存器编号，指令执行时直接取出寄存器的值来操作。
- 举例：

```
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
```

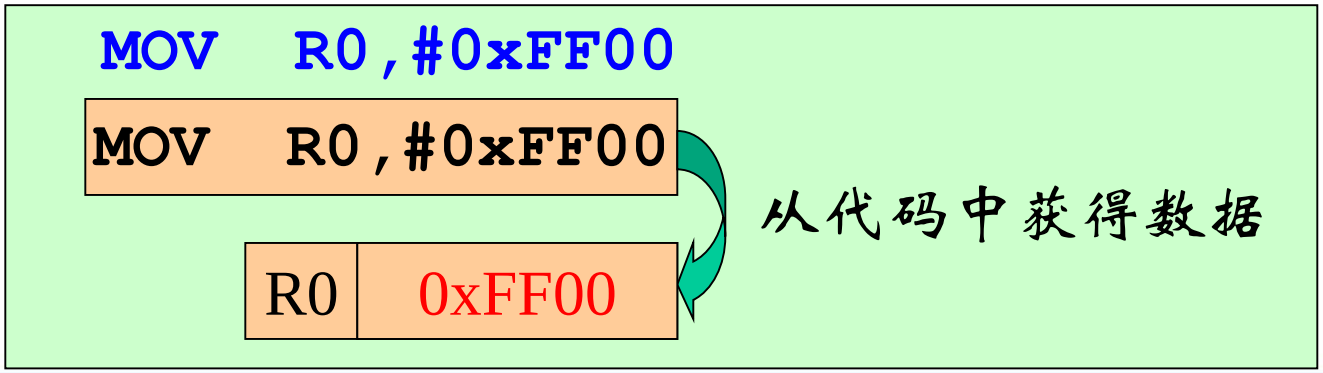


(2) 立即寻址

- 指令中的操作数就是数据本身，即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数(故称为立即数)。

- 举例：

```
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
```

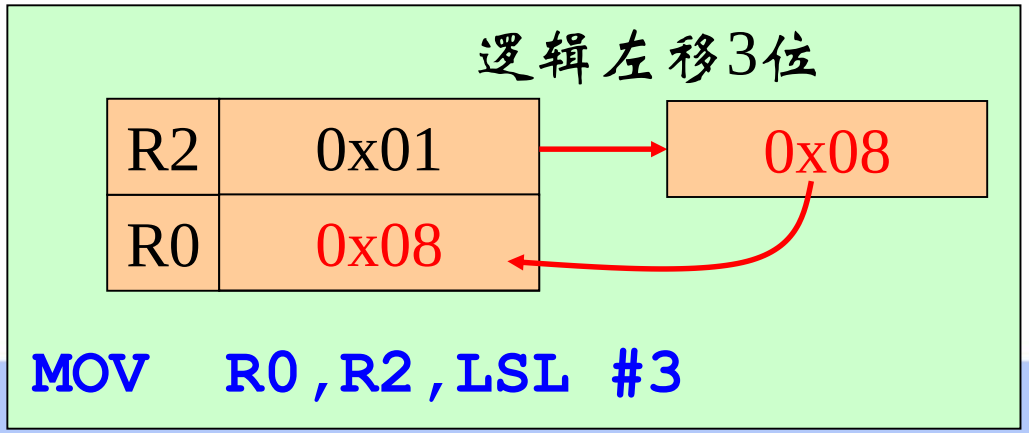


(3) 寄存器偏移寻址

- ARM指令集特有的寻址方式。当指令中某个操作数是该寻址方式时，则对其操作前，首先执行移位操作。

- 举例：

```
MOV R0,R2,LSL #3          ;R2的值左移3位，结果放入R0，
                          ;即R0=R2×8
ANDS R1,R1,R2,LSL  R3     ;R2的值左移R3位，然后和R1相
                          ;“与”操作，结果放入R1
```



ARM支持的移位操作

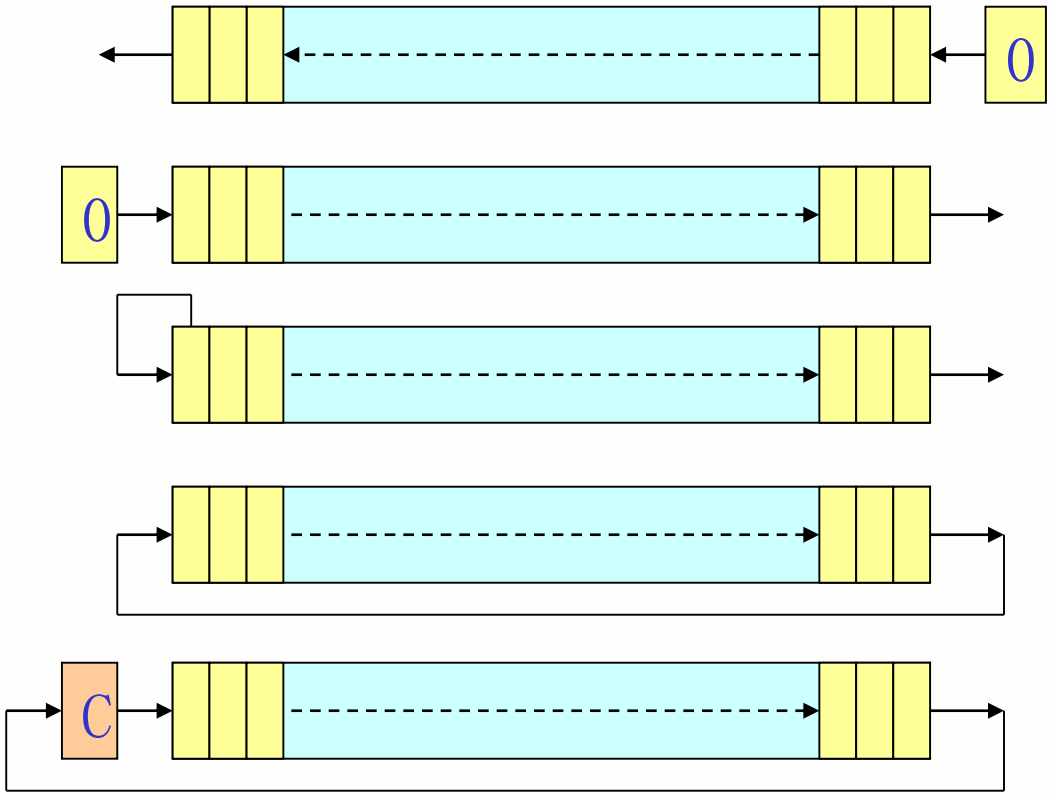
LSL移位操作：
(逻辑左移)

LSR移位操作：
(逻辑右移)

ASR移位操作：
(算术右移)

ROR移位操作：
(循环右移)

RRX移位操作：
(带进位位的循环右移)

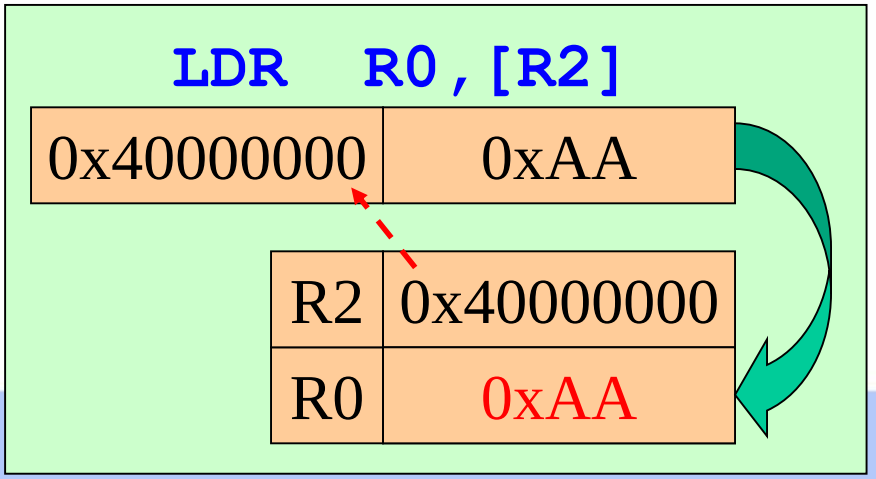


(4) 寄存器间接寻址

- 指令中的操作数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。

- 举例：

```
LDR   R1,[R2]        ;将R2指向的存储单元中的数据读出，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
```

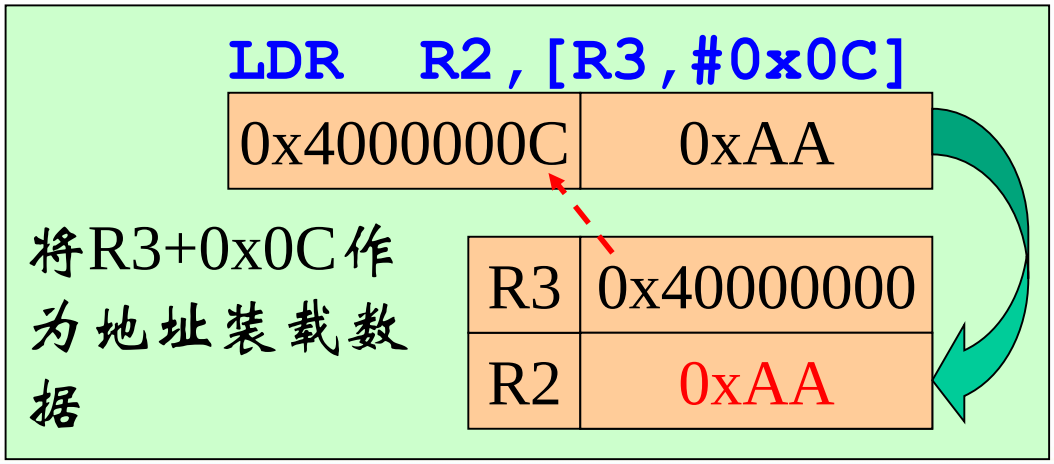


(5) 基址寻址

- 将基址寄存器的内容与指令中给出的偏移量相加，形成操作数的有效地址。
- 举例：

```
LDR   R2,[R3,#0x0C]   ;读取R3+0x0C地址的存储单元的内容，放入R2
STR   R1,[R0,#-4]!    ;把R1的值保存到R0-4地址指定的存储单元
```

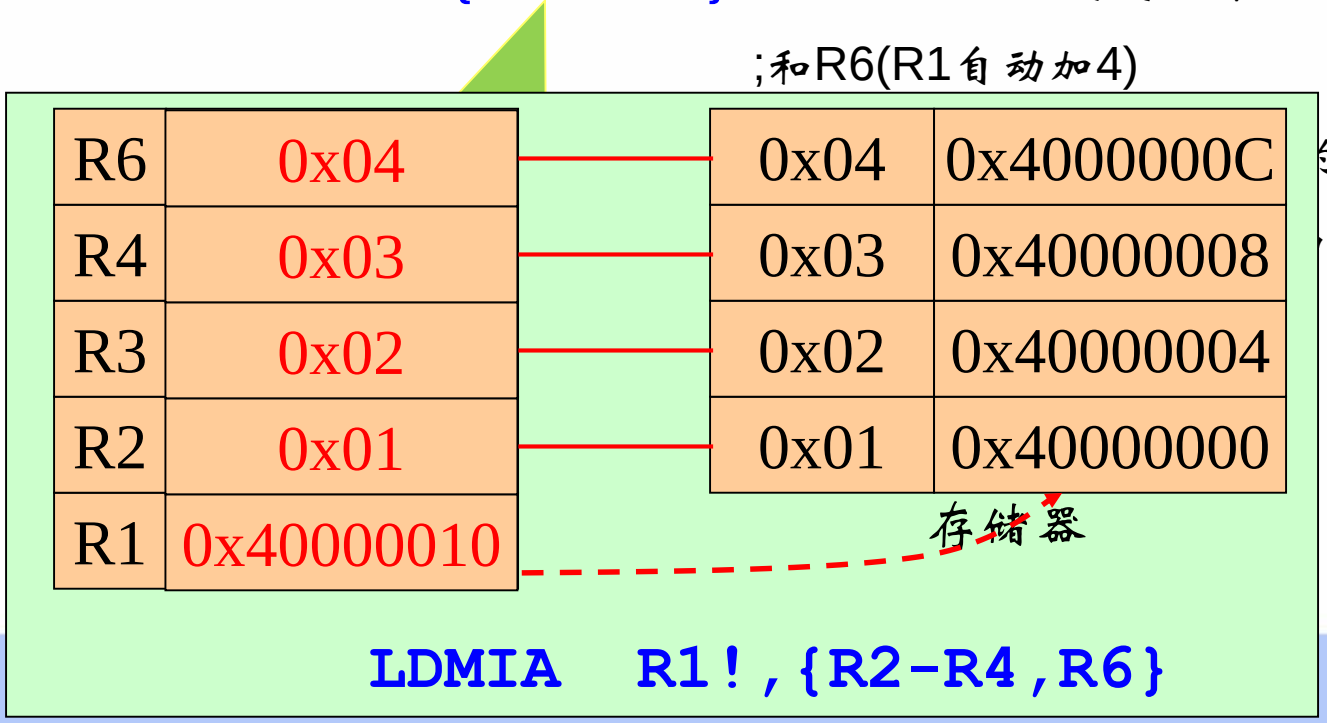
区别：寄存器偏移寻址



(6) 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。
- 举例：

```
LDMIA  R1!,{R2-R4,R6}  ;将R1指向的顺序存储单元中的数据读出到R2～R4
                       ;和R6(R1自动加4)
```



的值保存到R0指向的

4)

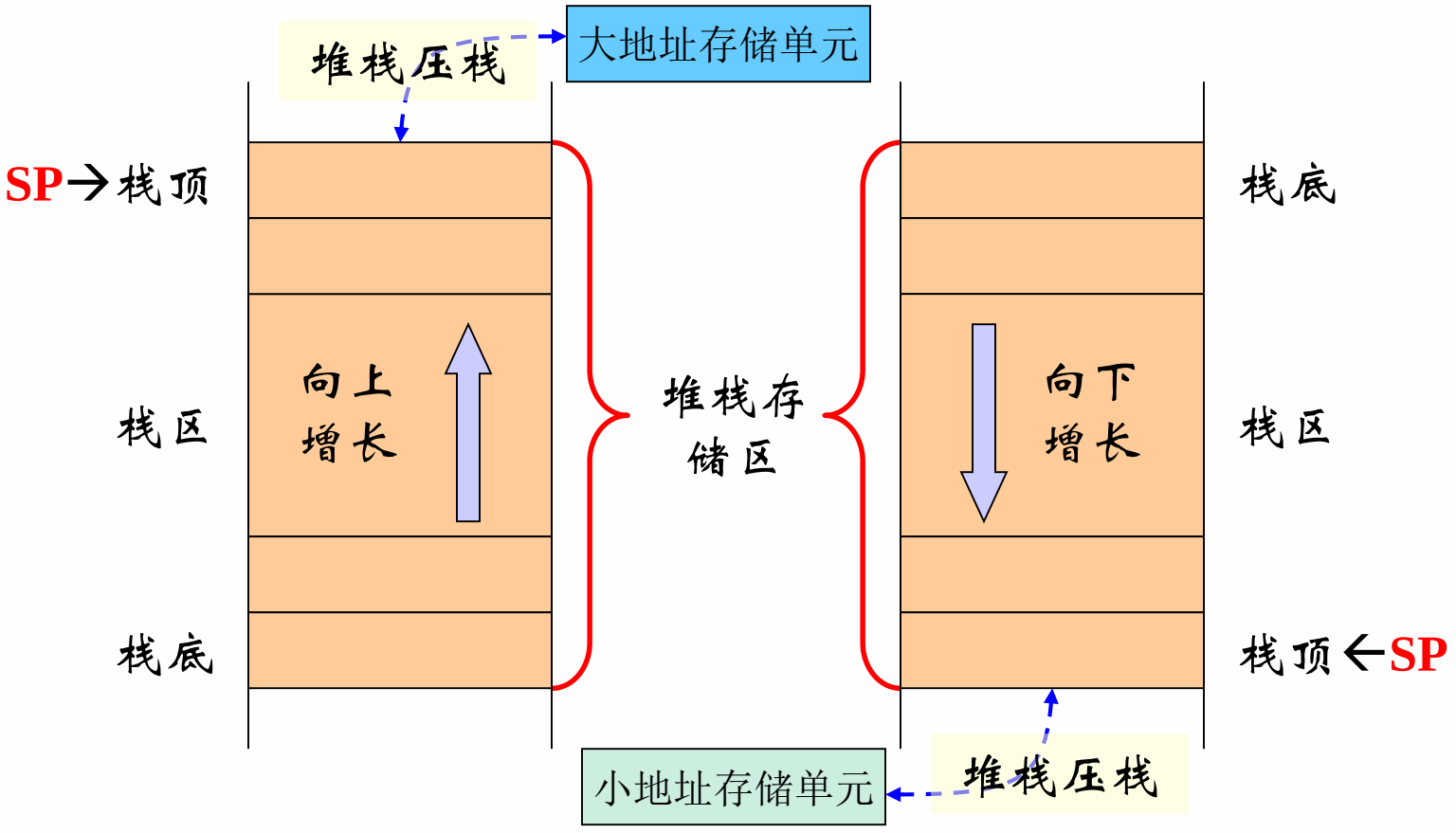
(7) 堆栈寻址

- 堆栈是存储器中一个按特定顺序(先进后出 or 后进先出)进行存取的区域。

- 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(即堆栈)，指针所指向的存储单元称为堆栈的栈顶。

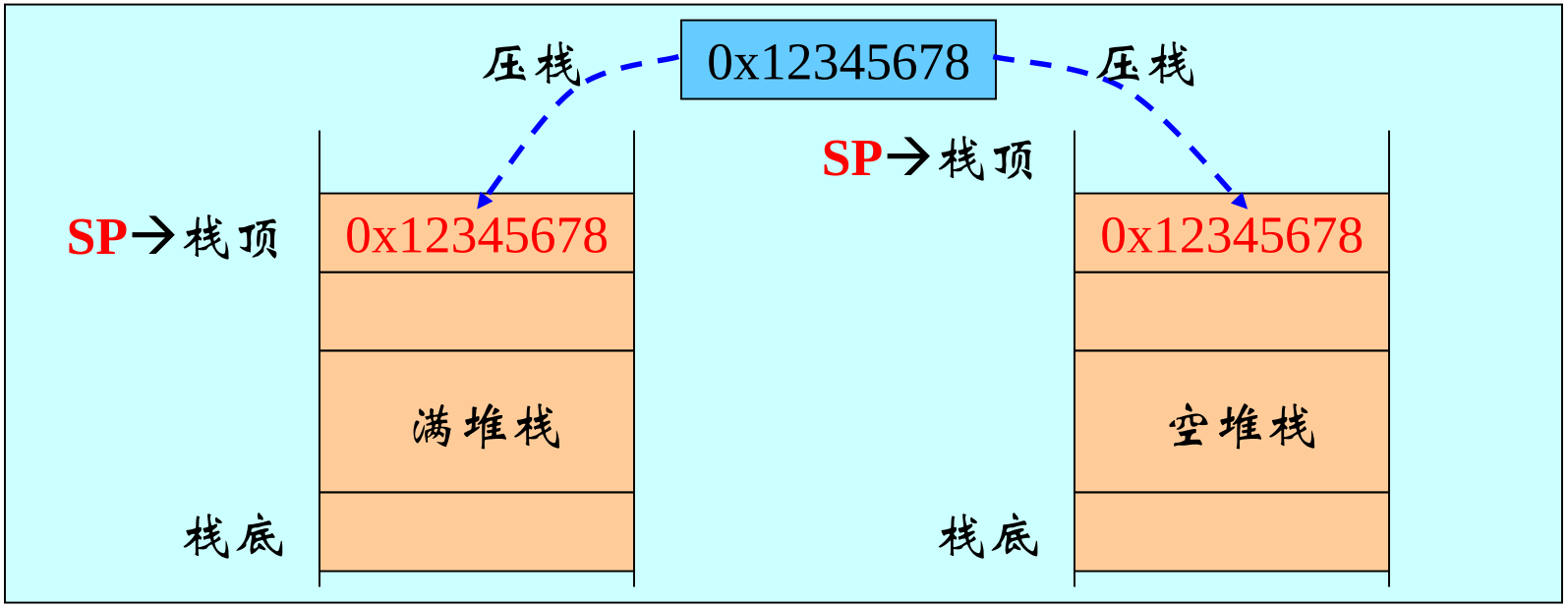
- 分类1：
  - ✓向上生长：向高地址方向生长，称为递增堆栈。
  - ✓向下生长：向低地址方向生长，称为递减堆栈。

(7) 堆栈寻址


堆栈压栈
大地址存储单元
SP→栈顶
栈区
栈底
向上
增长
堆栈存
储区
向下
增长
栈底
栈区
栈顶←SP
小地址存储单元
堆栈压栈

(7) 堆栈寻址

- 分类2：
  - ✓满堆栈：堆栈指针指向最后压入堆栈的有效数据项；
  - ✓空堆栈：堆栈指针指向下一个待压入数据的空位置。



(7) 堆栈寻址

➢两种分类组合→四种类型的堆栈：

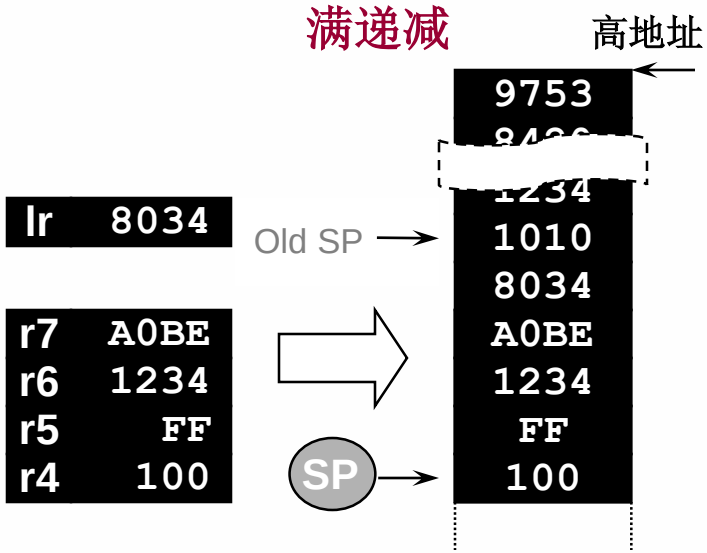
✓**满递增**：堆栈向上增长，堆栈指针指向内含有效数据项的最高地址。指令如LDMFA、STMFA等；

✓**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；

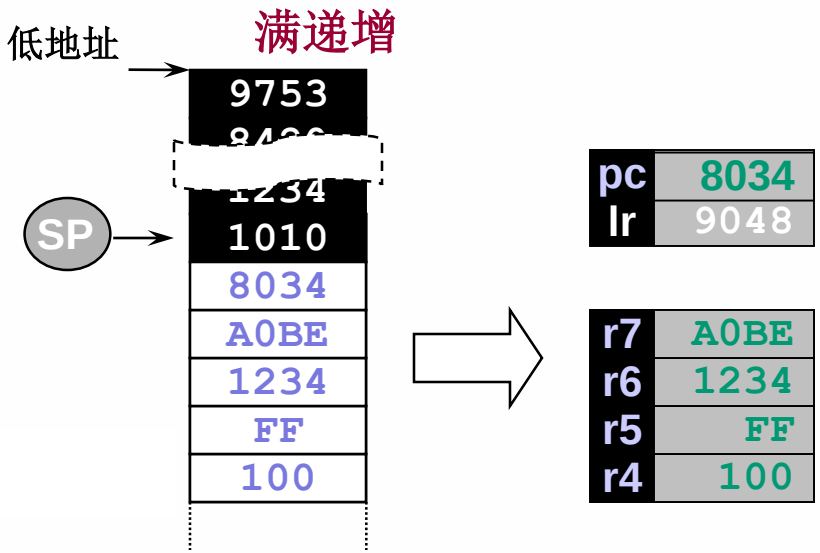
✓**满递减**：堆栈向下增长，堆栈指针指向内含有效数据项的最低地址。指令如LDMFD、STMFD等；

✓**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。

STMFD sp!,{r4-r7,lr}



LDMFA sp!,{r4-r7,pc}



注意入栈顺序！

(8) 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，指令中的操作数作为偏移量，两者相加后得到的地址即为操作数的有效地址。

- 举例：

```
        BL    SUBRl      ；调用SUBRl子程序
        . . .
SUBR1   …
        MOV PC,R14       ；返回
```

- 3.1 ARM寻址方式
- **3.2 ARM指令集**
- 3.3 ARM汇编程序设计
- 3.4 Thumb指令集

ARM微处理器的指令集

- RISC→加载/存储(Load/Store)型：
  - 指令集仅能处理寄存器中的数据，而且处理结果都要放回寄存器中。
  - 对存储器的访问需要通过专门的加载/存储指令来完成。
- ARM7TDMI(-S)的指令集，包括：
  - ARM指令集
  - Thumb指令集

ARM指令集与Thumb指令集的关系

Thumb指令集具有灵活、小巧的特点

ARM指令集支持ARM核所有的特性，具有高效、快速的特点

(1) 指令基本格式

(2) 分支指令(跳转指令)

(3) 数据处理指令

(4) 乘法指令

(5) 存储器访问指令

(6) 杂项指令

(7) 伪指令

ARM指令的基本格式

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

其中：<>号内的项是必须的，{}号内的项是可选的。

opcode：指令助记符；

cond：执行条件，可选，例如EQ、NE等。如果没有，则使用默认条件AL（即无条件执行）；

S：是否影响CPSR寄存器的值，可选；

Rd：目标寄存器；　　**Rn**：第1个操作数的寄存器；

operand2：第2个操作数，可选。

举例

寄存器寻址

寄存器间接寻址

- LDR R0，[R1] ;读取R1地址上的存储器单元内容，执行条件AL(无条件)

相对寻址

- BEQ D1 ;分支指令，执行条件EQ，即相等则跳转到D1

立即寻址

- ADDS R1，R1，#1 ;加法指令，R1+1→R1，影响CPSR寄存器(S)

- SUBNES R1，R1，#0x10 ;条件执行减法运算(NE)，
  ;R1-0x10→R1，影响CPSR寄存器(S)

条件码cond

- 可以实现高效的逻辑操作，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码，则默认为无条件(AL)执行。

指令条件码表

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)

标志影响位S

- 默认情况下，数据处理指令不影响CPSR的条件码标志位(N、Z、C、V)，但可以选择通过添加“S”来影响。
- 特殊：比较指令不需要添加“S”就可改变相应的标志位。
- 举例：

```
loop
  …
  SUBS  r1,r1,#1   ← R1减1，并设置标志位
  BNE   loop       ← 如果 Z标志清零(R1≠0)则跳转
```

1. 分支指令（跳转指令）
2. 数据处理指令
3. 乘法指令
4. 存储器访问指令
5. 杂项指令
6. 伪指令

1. 分支指令（跳转指令）

- 用于实现程序流程的跳转。
- 在ARM程序中有两种方法可以实现程序流程的跳转：
  - ✓ 使用专门的跳转指令。——可以实现向前或向后32MB的地址空间的跳转
  - ✓ 直接向程序计数器PC写入跳转地址值。——可以实现在4GB的地址空间的跳转
- 在跳转之前结合使用指令**MOV LR，PC**，可以保存将来的返回地址值，从而实现在4GB地址空间的子程序调用。

分支(跳转)指令包括以下4条指令：

— B 跳转指令

— BL 带返回的跳转指令

— BX 带状态切换的跳转指令

— BLX 带返回和状态切换的跳转指令

助记符	说明	操作	条件码位置
B label	跳转指令	PC←label	B{cond}
BL label	带返回的跳转指令	LR←PC-4，PC←label	BL{cond}
BX Rm	带状态切换的跳转指令	PC←Rm，切换处理器状态	BX{cond}
BLX Rm	带返回和状态切换的跳转指令	LR←PC-4，PC←Rm， 切换处理器状态	BLX{cond}

2. 数据处理指令

数据处理指令
- 数据传送指令
- 算术/逻辑运算指令
- 比较指令

➢数据处理指令只能对寄存器的内容进行操作，而不能对内存中的数据进行操作。所有ARM数据处理指令均可选择使用S后缀，并影响状态标志。

➢比较指令即使不使用S后缀，也会直接影响状态标志。

ARM数据处理指令----数据传送指令

助记符	说明	操作	条件码位置
MOV Rd,operand2	数据传送	Rd←operand2	MOV{cond}{S}
MVN Rd,operand2	数据非传送	Rd←(~operand2)	MVN{cond}{S}

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←Rn-operand2	CMP{cond}
CMN Rn, operand2	负数比较指令	标志N、Z、C、V←Rn+operand2	CMN{cond}
TST Rn, operand2	位测试指令	标志N、Z、C、V←Rn & operand2	TST{cond}
TEQ Rn, operand2	相等测试指令	标志N、Z、C、V←Rn ^ operand2	TEQ{cond}

ARM数据处理指令----算术运算指令

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

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←Rn&operand2	AND{cond}{S}
ORR Rd, Rn, operand2	逻辑或操作指令	Rd←Rn|operand2	ORR{cond}{S}
EOR Rd, Rn, operand2	逻辑异或操作指令	Rd←Rn^ perand2	EOR{cond}{S}
BIC Rd, Rn, operand2	位清除指令	Rd←Rn&(~operand2)	BIC{cond}{S}

3. 乘法指令

- ARM7TDMI具有三种乘法指令，分别为：
  - 32×32位乘法指令；
  - 32× 32位乘加指令；
  - 32× 32位结果为64位的乘/乘加指令。

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←Rm*Rs (Rd≠Rm)	MUL{cond}{S}
MLA Rd, Rm, Rs, Rn	32位乘加指令	Rd←Rm*Rs+Rn (Rd≠Rm)	MLA{cond}{S}
UMULL RdLo, RdHi, Rm, Rs	64位无符号乘法指令	(RdLo, RdHi) ←Rm*Rs	UMULL{cond}{S}
UMLAL RdLo, RdHi, Rm, Rs	64位无符号乘加指令	(RdLo, RdHi) ←Rm*Rs+(RdLo, RdHi)	UMLAL{cond}{S}
SMULL RdLo, RdHi, Rm, Rs	64位有符号乘法指令	(RdLo, RdHi) ←Rm*Rs	SMULL{cond}{S}
SMLAL RdLo, RdHi, Rm, Rs	64位有符号乘加指令	(RdLo, RdHi) ←Rm*Rs+(RdLo, RdHi)	SMLAL{cond}{S}

4. 存储器访问指令

存储器访问指令
- 单寄存器加载/存储指令
- 多寄存器加载/存储指令
- 寄存器和存储器交换指令

➢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。

➢使用单寄存器加载指令加载数据到PC寄存器，可实现程序的跳转功能。

存储器访问指令----单寄存器加载指令

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

存储器访问指令----单寄存器存储指令

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

(1)字和无符号字节加载/存储指令

➢LDR指令用于从内存中读取一个字或字节数据，存入寄存器中；STR指令用于将寄存器中的一个字或字节数据保存到内存。

➢指令格式如下：

（T表示以用户模式访问）

```
LDR{cond}{T}     Rd,<地址>    ;读出一个字（32位）
STR{cond}{T}     Rd,<地址>    ;存入一个字（32位）
LDR{cond}B{T}    Rd,<地址>    ;读出一个字节，高24位补0
STR{cond}B{T}    Rd,<地址>    ;存入一个字节，高24位不变
```

(2)半字和有符号字节加载/存储指令

- 这类LDR/STR指令可加载有符号半字或字节，可加载/存储无符号半字。

- 指令格式如下：

```
LDR{cond}SB   Rd,<地址>  ;读出一个带符号字节，高24位补齐符号位
LDR{cond}SH   Rd,<地址>  ;读出一个带符号半字，高16位补齐符号位
LDR{cond}H    Rd,<地址>  ;读出一个无符号半字，高16位补0
STR{cond}H    Rd,<地址>  ;写入半字，高16位不变
```

指令举例

```
LDRSB   R1，[R0，R3]   ；将R0+R3地址上的字节数据读
                       ；到R1，高24位用符号位扩展

LDRSH   R1，[R9]       ；将R9地址上的半字数据读出到R1，
                       ；高16位用符号位扩展

LDRH    R6,[R2]，#2    ；将R2地址上的半字数据读出到R6，
                       ；高16位用零扩展，R2=R2+2

STRH    R1，[R0，#2]!  ；将R1的数据保存到R0+2地址中
                       ；只存储低2字节数据，R0=R0+2
```

load/store指令应用举例：链表搜索操作

- 每个链表元素包括两个字：第1个字中包含一个字节数据；第2个字中包含指向下一个链表元素的指针，当这个指针为0时表示链表结束。
- 执行前R0指向链表的头元素，R1中存放要搜索的数据；执行后R0指向第1个匹配元素；若没有匹配元素，R0为0。

```
search
    CMP       R0,#0        ;R0指针是否为空
    LDRNEB    R2,[R0]      ;读取当前元素中的字节数据
    CMPNE     R1,R2        ;判断数据是否为搜索的数据
    LDRNE     R0,[R0,#4]   ;如果不是,指针R0指向下一个元素
    BNE       search       ;跳转到search执行
    MOV       PC,LR        ;搜索完成，程序返回
```

存储器访问指令----多寄存器加载/存储指令

- LDM和STM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。
- 指令格式：

```
LDM{cond}<模式>   Rn{!},reglist{^}
STM{cond}<模式>   Rn{!},reglist{^}
```

LDM和STM的主要用途是现场保护、数据复制、参数传递等。

多寄存器加载/存储指令的8种模式

模式	说明	模式	说明
IA	每次传送后地址加4	FD	满递减堆栈
IB	每次传送前地址加4	ED	空递减堆栈
DA	每次传送后地址减4	FA	满递增堆栈
DB	每次传送前地址减4	EA	空递增堆栈
数据块传送操作		堆栈操作	

- 进行数据复制时，先设置好源数据指针和目标指针，然后使用多寄存器寻址指令LDMIA/STMIA、LDMIB/STMIB、LDMDA/STMDA、LDMDB/STMDB进行读取和存储 。
- 进行堆栈操作时，要先设置堆栈指针（SP），然后使用堆栈寻址指令STMFD/LDMFD、STMED/LDMED、STMFA/LDMFA和STMEA/LDMEA实现堆栈操作。

关于加载/存储指令的补充说明

- 指令格式中，寄存器Rn为基址寄存器，装有传送数据的初始地址，Rn不允许为R15。
- 后缀“!”：表示最后的地址写回到Rn中。——更新Rn
- 寄存器列表reglist：可包含多于一个寄存器或包含寄存器范围，使用“，”分开，如{R1，R2，R6-R9}，寄存器按由小到大排列。
- 后缀“^”：不允许在用户模式或系统模式下使用。
  - 在LDM指令且寄存器列表中包含有PC时，除了正常的多寄存器传送外，还将SPSR拷贝到CPSR中，用于异常处理返回。
  - 若寄存器列表不包含PC，则加载/存储的是用户模式的寄存器，而不是当前(异常)模式的寄存器。

指令举例

```
LDMIA    R0!, {R3 - R9}     ; 加载R0指向地址上的多字数据
                            ; 保存到R3～R9中，R0值更新
STMIA    R1!, {R3 - R9}     ; 将R3～R9的数据存储到R1指
                            ; 向的地址上，R1值更新
STMFD    SP!, {R0 - R7, LR}  ; 现场保存，将R0～R7、
                             ; LR入栈
LDMFD    SP!, {R0 - R7, PC}  ; 恢复现场，异常处理返回
```

存储器访问指令----寄存器与存储器交换指令

- SWP指令用于将一个内存单元(地址在寄存器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。
- 可用于实现信号量操作。
- 指令格式：

SWP{cond}{B}　Rd,Rm,[Rn]

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

指令举例

```
SWP    R1,R1,[R0]    ;将R1的内容与R0指向的存储单元的内
                     ;容进行交换

SWPB   R1,R2,[R0]    ;将R0指向的存储单元内的容读取一字
                     ;节数据到R1中(高24位清零)，并将R2
                     ;的低8位数据(最低字节）写入到该存
                     ;储单元中
```

5. 杂项指令

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

杂项指令----读状态寄存器指令

指令格式：

注意：在ARM处理器中，只有MRS指令可以将状态寄存器CPSR或SPSR读出到通用寄存器中。

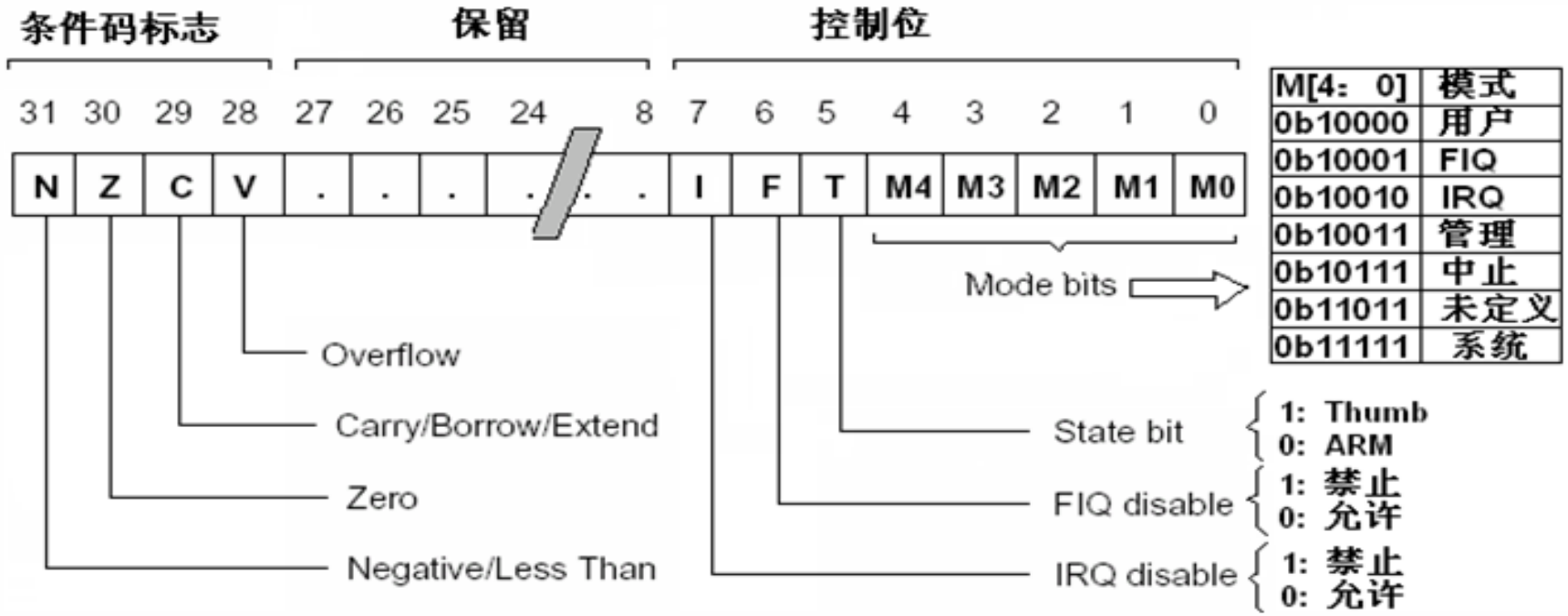
```
MRS {cond}    Rd，psr
```

其中：Rd　目标寄存器,不允许为R15。
　　psr　CPSR或SPSR。

指令举例

```
MRS    R1,CPSR   ；读取CPSR状态寄存器内容，保存到R1中
MRS    R2,SPSR   ；读取SPSR状态寄存器内容，保存到R2中
```

CPSR/SPSR寄存器格式：



M[4: 0]	模式
0b10000	用户
0b10001	FIQ
0b10010	IRQ
0b10011	管理
0b10111	中止
0b11011	未定义
0b11111	系统

指令举例

11010011b

```
MSR     CPSR_c, #0xD3      ; CPSR[7…0]=0xD3，即切换
                           ; 到管理模式
MSR     CPSR_cxsf, R3      ; CPSR=R3
```

★MSR与MRS配合使用：可以对CPSR或SPSR寄存器的读-修改-写操作，从而实现切换处理器模式、允许/禁止IRQ/FIQ中断等。

应用示例1：

```
;子程序：使能IRQ中断
ENABLE_IRQ
    MRS     R0, CPSR          (1)
    BIC     R0, R0,#0x80      (2)
    MSR     CPSR_c,R0         (3)
    MOV     PC,LR             (4)
```

1.将CPSR寄存器内容读出到R0；

2.修改对应于CPSR中的I控制位；

应用示例2：

```
;子程序：禁能IRQ中断
DISABLE_IRQ
    MRS     R0, CPSR          (1)
    ORR     R0, R0,#0x80      (2)
    MSR     CPSR_c,R0         (3)
    MOV     PC,LR             (4)
```

3.将修改后的值写回CPSR寄存器的对应控制域；

4.返回上一层函数；

杂项指令----软中断指令

- SWI指令用于产生异常中断，从而实现用户模式到管理模式的切换，用于在用户模式下对操作系统中特权模式的程序的调用。
- 执行流程：将处理器置于svc模式，并将CPSR保存到SPSR_svc中，然后程序跳转到SWI异常处理程序入口(异常向量地址为0x08)。
- 指令格式：

SWI{cond} immed_24

指令举例

```
SWI    0                ; 软中断，中断立即数为0
SWI    0x123456         ; 软中断，中断立即数为0x123456
```

SWI指令说明

- 用途：主要用于用户程序调用操作系统的API 。
- 两种主要参数传递方法
  - ✓ 第一种： 指令中的24bit立即数指定API号，其它参数通过寄存器传递。

    核心思想：在SWI异常处理子程序中执行LDR R0，[LR,#-4]，把产生SWI异常的SWI指令(如：SWI 0x98)装进R0寄存器。由于SWI指令的低24位保存了指令的操作数(如：0x98)，所以再执行BIC R0，R0，#0xFF000000语句，就可以获得*immed_24* 操作数的实际内容。

步骤

- 首先确定引起软中断的SWI指令是ARM指令还是Thumb指令，这可通过对SPSR访问得到；
- 然后取得该SWI指令的地址，这可通过访问LR寄存器得到；
- 接着读出该SWI指令，分解出立即数。

```
SWI_Handler
    STMFD   SP!, {R0-R3, R12, LR}    ; 现场保护
    MRS     R0, SPSR                 ; 读取SPSR
    STMFD   SP!, {R0}                ; 保存SPSR
    TST     R0, #0x20                ; 测试T标志位
    LDRNEH  R0, [LR,#-2]             ; 若是Thumb指令，读取指令码(16位)
    BICNE   R0, R0, #0xFF00          ; 取得Thumb指令的8位立即数
    LDREQ   R0, [LR,#-4]             ; 若是ARM指令，读取指令码(32位)
    BICEQ   R0, R0, #0xFF000000      ; 取得ARM指令的24位立即数
        ...
    LDMFD   SP!, {R0-R3, R12, PC}^ ; SWI异常中断返回
```

SWI指令说明

- 两种主要参数传递方法
  - ✓ 第二种：忽略指令中的24bit立即数，由R0指定API号，其它参数通过其它寄存器传递。

立即数无效，任何值都可以

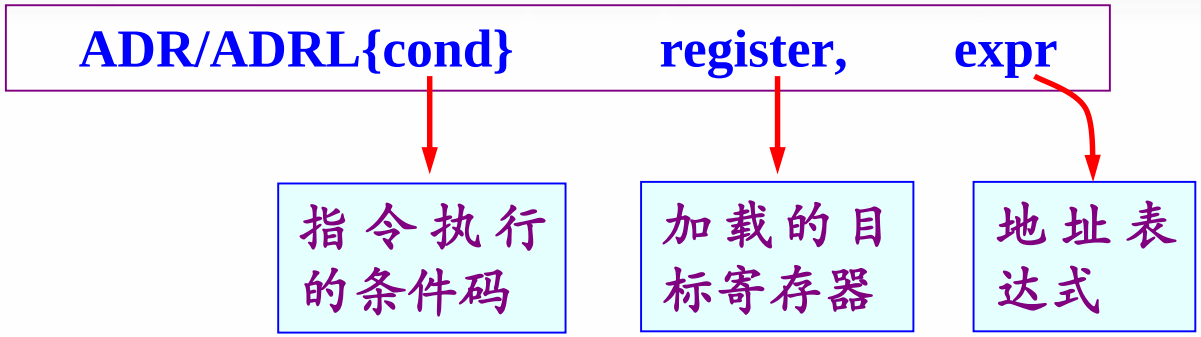
```
MOV     R0,#12    ;调用12号软中断
MOV     R1,#34    ;设置子功能号为34
SWI     0
```

6. 伪指令

- “伪”：不属于ARM指令集中的指令，为了编程方便而定义。
- 可以像其它ARM指令一样使用，但在编译时将被等效的一条或多条ARM指令所代替。

常用ARM伪指令
- ADR
- ADRL
- LDR
- EQU
- NOP

ADR/ADRL伪指令格式 ——小/中等范围的地址读取



➢功能：将基于PC相对偏移的地址值或基于寄存器相对偏移的地址值读取到寄存器中。

✓ADR：当地址值是字节对齐时，expr的取值范围为：-255~255；字对齐时，取值范围为-1020～1020；

✓ADRL：当地址值是字节对齐时，expr的取值范围为：-64K~64K；字对齐时，取值范围为-256K～256K。

❖ ADR举例：

应用示例（源程序）：

```
    ...
    ADR    R0,Delay
    ...
Delay
    MOV    R0,r14
    ...
```

使用伪指令将程序标号Delay的地址存入R0

编译后的反汇编代码：

```
    ...
0x20  ADD    r0,pc,#0x3c
    ...
    ...
0x64  MOV    r0,r14
    ...
```

ADR伪指令被汇编成一条指令

❖ ADRL举例：

应用示例（源程序）：

```
    ...
    ADRL    R0,Delay
    ...
Delay
    MOV     R0,r14
    ...
```

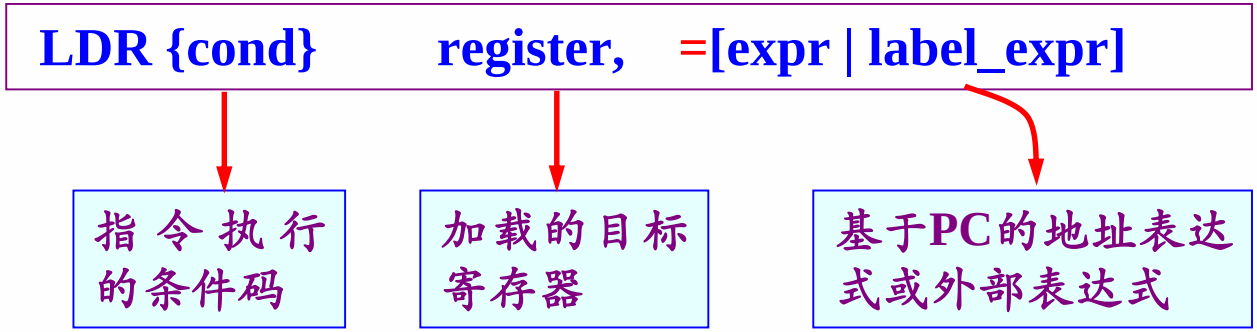
使用伪指令将程序标号Delay的地址存入R0

编译后的反汇编代码：

```
    ...
0x20  ADD     r0,pc,#40
0x24  ADD     r0,r0,#FF00
    ...
0xFF68 MOV    r0,r14
    ...
```

ADRL伪指令被汇编成两条指令

LDR伪指令格式——加载32位立即数，或一个地址值到指定寄存器



注意：

- 从指令位置到文字池的偏移量必须小于4KB；
- 与ARM指令的LDR相比，伪指令的LDR的参数有=号。

❖LDR伪指令举例：

```
LDR R2, =0xFF0              ;MOV R2, #0xFF0
LDR R0, =0xFF000000         ;MOV R0, #0xFF000000
LDR R1, =0xFFFFFFFE         ;MVN R1, #0x00000001
```

```
...
LDR    R1,=InitStack
...
InitStack
MOV    R0, LR
...
```

使用伪指令将程序标号InitStack的地址存入R1

EQU伪指令格式——将一个数值或寄存器名赋给一个指定的符号名

```
name  EQU  expr {,type}
```

其中：

- name：expr定义的符号名称；
- expr：基于寄存器的地址值、程序中的标号、32位的地址常量或者32位的常量；
- type：当expr为32位常量时，可以使用type指示expr数据的类型，取值为CODE32、CODE16和DATA。

❖EQU伪指令举例：

```
abcd  EQU 2                ;定义abcd符号的值为2

abcd  EQU label+16         ;定义abcd符号的值为(label+16)
abcd  EQU 0x1c,CODE32      ;定义abcd符号的值为绝对地址
                           ;值0x1c,而且此处为ARM指令
```

NOP伪指令格式——空操作

```
        MOV    R1,#0x1234
Delay
        NOP                    ;空操作，编译时被替换为类似
        NOP                    ; MOV R0, R0这样的无用指令
        NOP
        SUBS   R1,R1,#1        ;循环次数减一
        BNE    Delay           ;如果循环没结束，跳转Delay继续
        MOV    PC,LR           ;子程序返回
```

- 3.1 ARM寻址方式
- 3.2 ARM指令集
- **3.3 ARM汇编程序设计**
- 3.4 Thumb指令集

```
;文件名：TEST1.S
;功能：实现两个寄存器相加
        AREA    Example1,CODE,READONLY  ;声明代码段Example1
        ENTRY                           ;标识程序入口
        CODE32                          ;声明32位ARM指令
START   MOV     R0,#0                   ;设置参数
        MOV     R1,#10
LOOP    BL      ADD_SUB                 ;调用子程序ADD_SUB
        B       LOOP                    ;跳转到LOOP
ADD_SUB
        ADDS    R0,R0,R1                ;R0 = R0 + R1
        MOV     PC,LR                   ;子程序返回
        END                             ;文件结束
```

例1： 实现乘法的指令段

MOV R0,R0,LSL #n ;R0=R0<<n； R0= R0*2^n

ADD R0,R0,R0,LSL #n ;R0=R0+R0*2^n= R0*(2^{n+1})

RSB R0,R0,R0,LSL #n ;R0=R0*2^n-R0= R0*(2^{n-1})

例2：64位数据运算

➢假设R0和R1存放一个64位数据，R0中存放数据的低32位；R2和R3中存放另一个64位数据，R2中存放数据的低32位。

① 两个64位数据的加法运算，结果保存到R0和R1中。

```
ADDS  R0,R0,R2   ;低32位相加，设置CPSR的C标志位。
ADC   R1,R1,R3   ;高32位的带位相加
```

② 两个64位数据的减法运算，结果保存到R0和R1中。

```
SUBS  R0,R0,R2   ;低32位相减，设置CPSR的C标志位。
SBC   R1,R1,R3   ;高32位的带位相减
```

③ 两个64位数据的比较操作，并设置CPSR中的条件标志位。

```
CMP     R1,R3    ;比较高32位
CMPEQ   R0,R2    ;如果高32位相等，比较低32位
```

例3：转换内存中数据存储方式

- 将寄存器**R0**中的数据存储方式转换成另一种存储方式。指令执行前**R0**中数据存储方式为：**R0=A,B,C,D**；指令执行后**R0**中数据存储方式为：**R0=D,C,B,A**。

```
EOR   R1,R0,R0,  ROR   #16    ;R1=A^C,B^D,C^A,D^B
BIC   R1,R1,#0xFF0000         ;R1=A^C,0,C^A,D^B
MOV   R0,R0,ROR  #8           ;R0=D,A,B,C
EOR   R0,R0,R1,LSR  #8        ;R0=D,C,B,A
```

例4：子程序的调用

➢ **BL指令**在执行跳转操作的同时保存下一条指令的地址，用于从被调用的子程序中返回。

```
        ……
        BL   function     ;调用子程序function
        ……                ;子程序结束后，程序将返回到这里执行
        ……
function                  ;子程序的程序体
        ……
        MOV  PC,LR        ;子程序中的返回语句
```

例5：条件执行

- 实现类似于C语言中的if-else功能的代码段。

■ C语言代码为：

```
int   gcb (int a,int b)
{
  while (a!=b)
    {   if (a>b)   a=a-b;
        else       b=b-a;
    }
  return a;
}
```

对应的ARM代码段。（代码执行前R0中存放a，R1中存放b；代码执行后R0中存放最大公约数。

```
gcb
CMP     R0,R1       ;比较a和b的大小
SUBGT   R0,R0,R1    ;if(a>b)   a=a-b
SUBLT   R1,R1,R0    ;if(b>a)   b=b-a
BNE     gcb         ;if(a!=b)跳转到gcb继续执行
MOV     PC,LR       ;子程序结束，返回
```

例6：循环语句

- 下面代码段实现了程序循环执行。

```
        MOV    R0,#loopcount    ;初始化循环次数
loop                            ;循环体
        ……
        SUBS   R0,R0,#1         ;循环计数器减1，设置条件标志
        BNE    loop             ;循环计数器不为0,跳到loop继续执行
        ……                      ;循环计数器为0，程序继续执行
```

- 3.1 ARM寻址方式
- 3.2 ARM指令集
- 3.3 ARM汇编程序设计
- **3.4 Thumb指令集**

Thumb指令

- Thumb指令集可以看作是ARM指令压缩形式的子集。
  - ✓是为减小代码量而提出的；
  - ✓具有16位的代码密度。
- Thumb指令体系不完整，只支持通用功能。
- 必要时仍需要使用ARM指令。
  - ✓如：进入异常时。

主要区别

- 只有B指令可以条件执行，其它都不能条件执行；
- 分支指令的跳转范围有更多限制；
- 单寄存器访问指令，只能操作R0～R7；
- LDM和STM指令可对R0～R7的任何子集进行操作。

The End!

本章要点

- ARM指令系统，要求达到“简单应用”层次。
  - 熟练掌握八种基本寻址方式。
  - 认识指令的结构，通过例子熟悉常用ARM指令的格式、功能和使用方法。
  - 在读懂汇编程序的基础上，初步编写简单的程序。

课后作业

一、假设初始时寄存器R0=0x8000，R1=0x01，R2=0x10，R3=0x20，存储器内容为空，且采用小端格式。试分别分析顺序执行下列指令后，寄存器R0、R1和R2的内容是什么？

(1) **STMIA R0 , {R1, R2, R3}**

(2) **LDMIB R0!, {R1, R2}**

R0=?

R1=?

R2=?