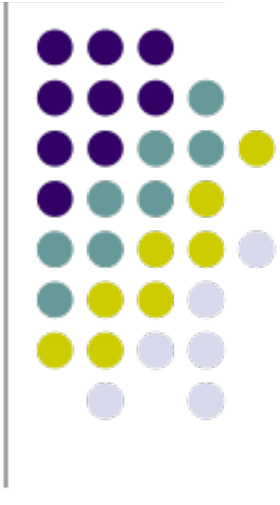
《嵌入式系统基础教程》

第2版

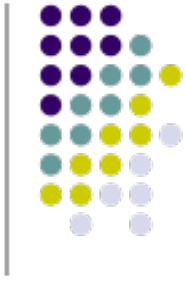
20XX年X季 第7讲



XXXX大学计算机系

XXXX主讲

教学主要内容



- 第5章
- **ARM处理器的指令系统**
- **ARM指令集的编码格式和语法**
- **ARM处理器的寻址方式**
- ARN指令分类说明

第5章 ARM处理器指令集

- 本章介绍以下内容：
  - ARM处理器指令系统的主要特征
  - ARM与x86的指令系统比较
  - ARM指令集的编码格式
  - ARM指令的一般语法格式
  - ARM指令的执行条件
  - **第2操作数<Operand2>说明**
  - ARM处理器的寻址方式
  - **ARM指令分类说明**
  - **单个ARM指令的用法举例**

教学知识点

- 本章主要介绍以下内容：
  - **ARM指令集的基本特点**
    - **与Thumb指令集的区别**
    - **与x86处理器的区别**
    - **ARM指令格式**
  - **ARM寻址方式**
  - ARM指令集分类详解

5.1 ARM处理器的指令集基本特点

- 指令集的异同点
  - ARM、Thumb、x86
- ARM指令集的语法
  - ARM指令集的编码格式
- 指令条件码表
- 第2操作数

ARM指令集和Thumb指令集的共同点

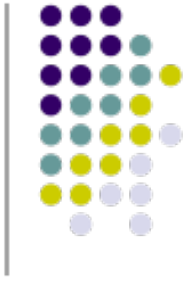
- ARM指令集和Thumb指令集具有以下共同点：

  1.较多的寄存器，可以用于多种用途。

  2.对存储器的访问只能通过Load/Store指令。

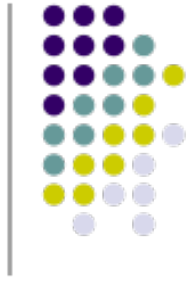
- 两种指令集的差异特征在下页给出

ARM指令集和Thumb指令集的不同点



项目	ARM指令	Thumb指令
指令工作标志	CPSR的T位=0	CPSR的T位=1
操作数寻址方式	大多数指令为3地址	大多数指令为2地址
指令长度	32位	16位
内核指令	58条	30条
条件执行	大多数指令	只有分支指令
数据处理指令	访问桶形移位器和ALU	独立的桶形移位器和ALU指令
寄存器使用	15个通用寄存器+PC	8个通用低寄存器+7个高寄存器+PC
程序状态寄存器	特权模式下可读可写	不能直接访问
异常处理	能够全盘处理	不能处理

ARM指令集与x86指令集的主要不同点

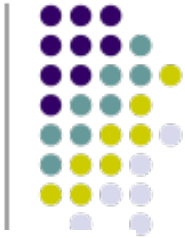


ARM指令集	x86指令集
规整指令格式	非规整指令格式
即：正交指令格式	即：非正交指令格式
三地址指令	二地址指令
由指令的附加位决定运算完毕后是否改变状态标志	指令隐含决定运算完毕后是否改变状态标志
状态标志位只有4位	状态标志位有6位
有两种指令密度	单一指令密度
无整数除法指令	有整数除法指令
大多数ARM指令都可以条件执行	专用条件判断指令进行程序分支
Load/Store访存体系结构	运算指令能够访问存储器

5.2 ARM指令集的编码格式和语法

- 参看ARM指令集编码格式PDF文件
- 教材图29_ARM指令集编码结构图

ARM指令集的语法



- 一条典型的ARM指令语法如下所示：

<opcode> {<cond>} {S} <Rd>, <Rn> {,<Operand2>}

其中：

<opcode > 是指令助记符，决定了指令的操作。

例如：ADD表示算术加操作指令。

{<cond>} 是指令执行的条件，可选项。

{S} 决定指令的操作是否影响CPSR的值，可选项。

<Rd> 表示目标寄存器，必有项。

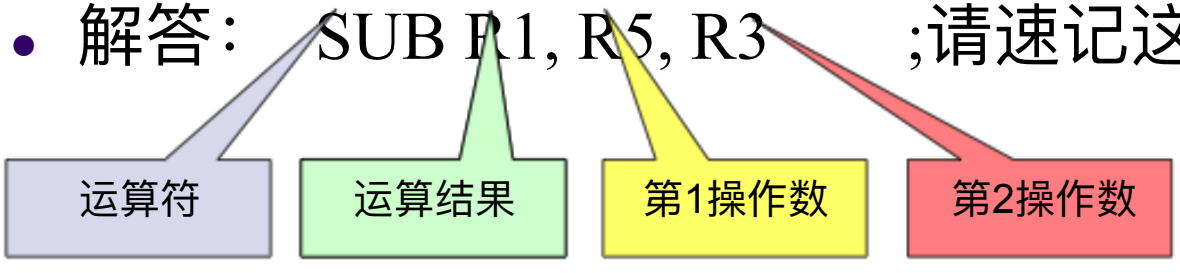
<Rn> 表示包含第1个操作数的寄存器，当仅需要一个源操作数时可省略。

<Operand2> 表示第2个操作数，可选项。

第2操作数有两种格式：#immed_8r，Rm{, Shift}

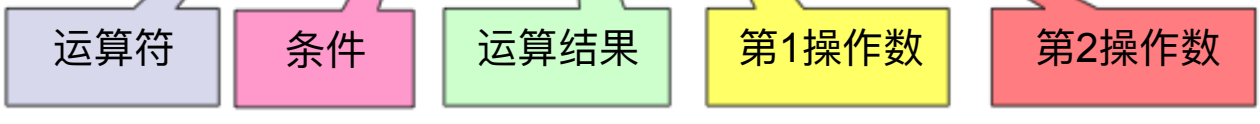
ARM指令的语法成分分析-1

- 假定被减数存放在R5中，减数存放在R3中，R5-R3的差数存放在R1里，请给出这个计算的ARM指令。
- 解答：　SUB R1, R5, R3　　;请速记这条指令



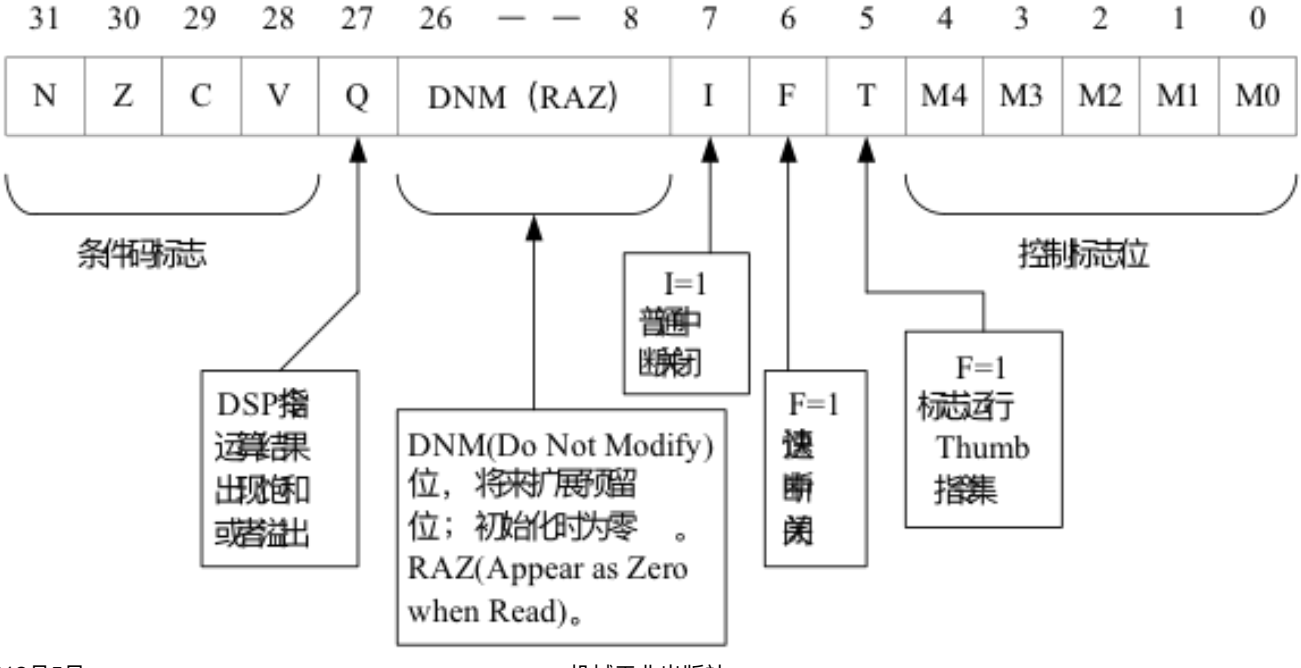
ARM指令的语法成分分析-2

- 假定被减数存放在R5中，减数存放在R3中，R5-R3的差数存放在R1里，如果要求反映前面的运算结果的条件是CC，请给出这个计算的ARM指令。
- 解答：　**SUB**CC　R1, R5, R3　　;请速记这条指令



ARM处理器的CPSR寄存器和SPSR寄存器的位定义格式图解

- 参看升级版基础教程第3.2.4节



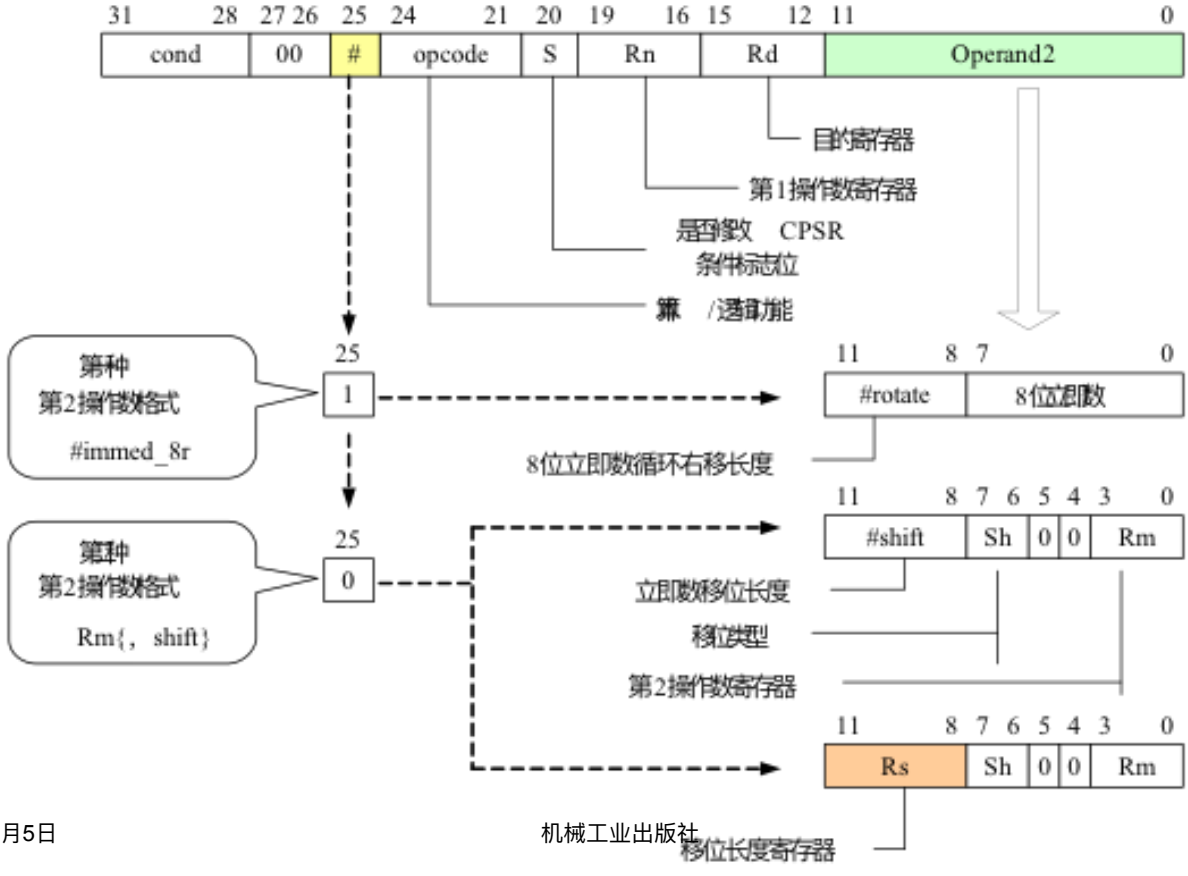
指令条件码表（1）

条件码助记符	标志	含义
EQ	Z＝1	相等
NE	Z＝0	不相等
CS/HS	C＝1	无符号数大于或等于
CC/LO	C＝0	无符号数小于
MI	N＝1	负数（minus）
PL	N＝0	正数或零
VS	V＝1	上溢出
VC	V＝0	没有上溢出

指令条件码表（2）

条件码助记符	标志	含义
HI	C＝1，Z＝0	无符号数大于
LS	C＝0，Z＝1	无符号数小于或等于
GE	N＝V	有符号数大于或等于
LT	N!＝V	有符号数小于
GT	Z=0，N=V	有符号数大于
LE	Z＝1，N!=V	有符号数小于或等于
AL	任何	无条件执行(指令默认条件)
NV	ARMv3之前	该指令从不执行

ARM数据处理指令中第2操作数的编码格式图解

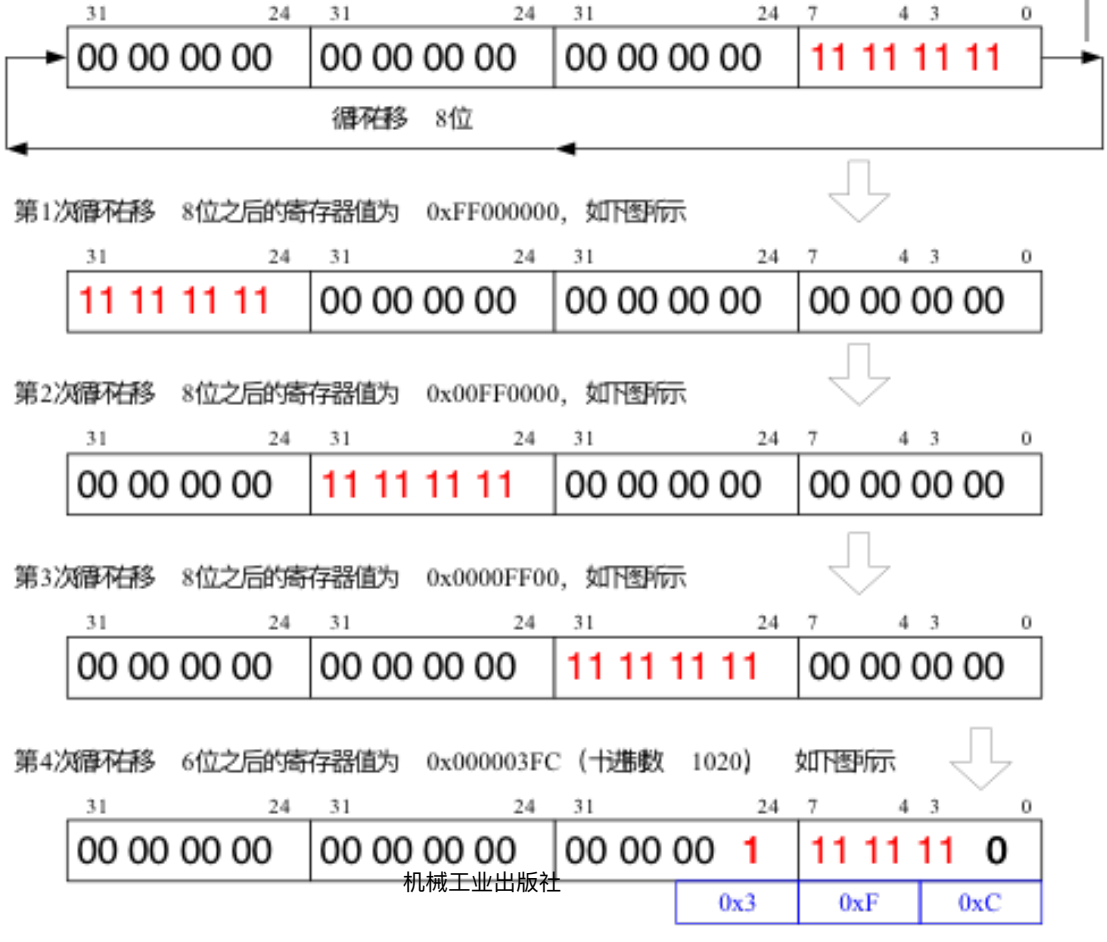


两种灵活的第2操作数

- **立即数型**
  - **格式： #<32位立即数>**
  - 也写成#immed_8r
  - #<32位立即数>是取值为数字常量的表达式，并不是所有的32位立即数都是有效的。
  - 有效的立即数很少。它必须由一个8位的立即数循环右移偶数位得到。原因是32位ARM指令中条件码和操作码等占用了一些必要的指令码位，32位立即数无法直接编码在指令中。
  - 举例：
    - ADD r3, r7, #1020 ;#immed_8r型第2操作数,
      ;1020是0xFF循环右移30位后生成的32位立即数
      ;推导：1020=0x3FC=0x000003FC

立即数1020内部汇编处理图解

- 立即数1020是0xFF循环右移30次获得的



总循环右移次数30次

灵活的第2操作数（续1）

- 数据处理指令中留给Operand2操作数的编码空间只有12位，需要利用这12位产生32位的立即数。其方法是：把指令最低8位（bit[7:0]）立即数循环右移偶数次，循环右移次数由2*bit[11:8] （bit[11:8]是Operand2的高4位）指定。
- 例如：MOV R4, #0x8000000A

;其中的立即数#0x8000000A是由8位的0xA8循环右移0x4位得到。

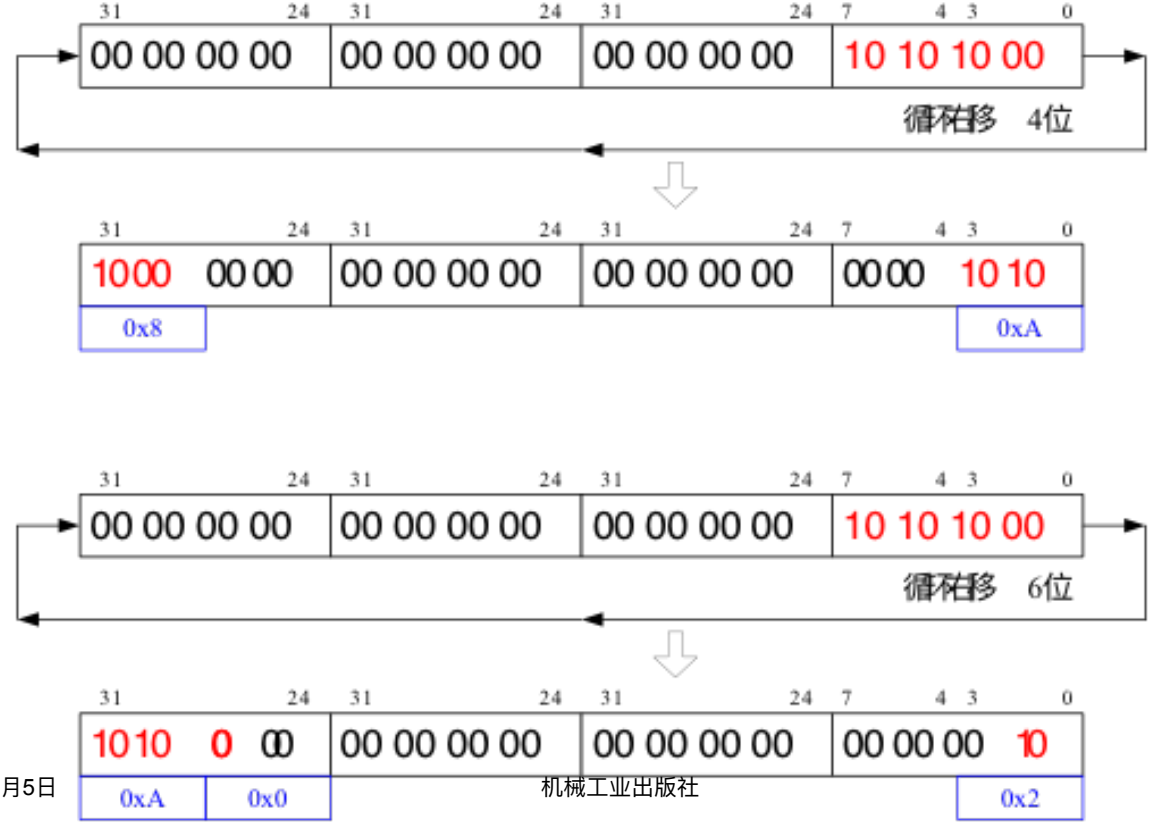
- 又例如：MOV R4, #0xA0000002

;其中的立即数#0xA0000002是由8位的0xA8循环右移0x6位得到。

两个立即数的产生图解

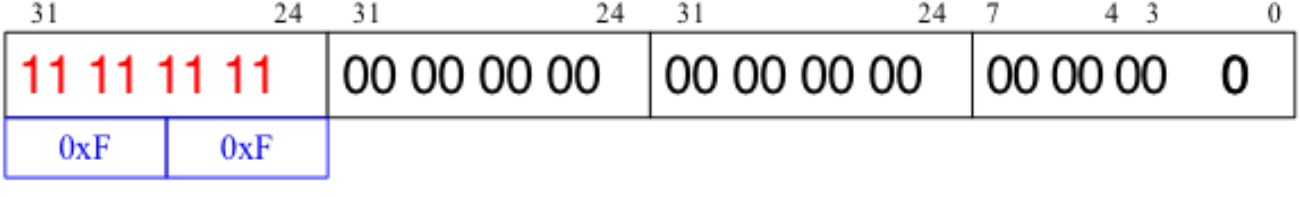
- 立即数#0x8000000A的产生图解
- 立即数#0xA0000002的产生图解

由单字节0xA8右移偶数次产生



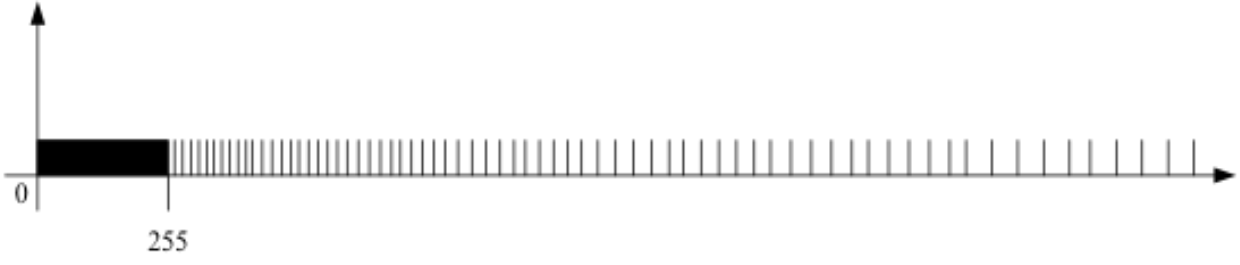
立即数型的第2操作数有多少?

- 在32位寄存器中，右移偶数次的种数一共是16个（0/2/4/6/8/…../24/26/28/30），即2^4种。
- 单字节的取值种数为256种，即2^8种。
- 于是，符合语法的立即数型第2操作数的总个数是2^12种，即4096个。
- 最大的立即数型第2操作数是0xFF左移24位
  - 相当于0xFF循环右移8位
  - 也就是0xFF000000，对应的十进制数是4278190080，参看图解



数轴上立即数型第2操作数的表示

- 下面的图解给出了立即数型第2操作数在整数数轴上的分布
- 说明：
  - 0-255是密集分布的，没有间隔
  - 256-1020之间，每一个相邻4。
  - 最大的立即数型第2操作数是4278190080。

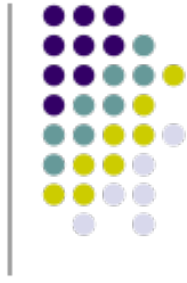


立即数型第2操作数在整数数轴上的表示

灵活的第2操作数（续2）

- **寄存器移位型**
- **格式：Rm{, <shift>}**
- Rm是第2操作数寄存器，可对它进行移位或循环移位。<shift>用来指定移位类型（LSL，LSR，ASR，ROR或RRX）和移位位数。其中移位位数有两种表示方式，一种是5位立即数（#shift），另外一种是位移量寄存器Rs的值。参看下面的例子。例子中的R1是Rm寄存器。
- ADD R5, R3, R1, LSL #2　;R5←R3+R1*4
- ADD R5, R3, R1, LSL R4　;R5←R3+R1*2R4 ;R4是Rs寄存器，Rs用于计算左移次数

详解第2操作数 # immed_8r



- 该常数必须对应8位位图，即常数是由一个8位的常数循环右移位偶数位得到。例如：
- 合法常量：0x3FC、0、0xF0000000、200、0xF0000001。
- 非法常量：0x1FE、511、0xFFFF、0x1010、0xF0000010。
- 常数表达式应用举例：

```
MOV   R0，#1        ;R0=1
AND   R1，R2，#0x0F  ;R2∩0x0F，结果保存在R1
LDR   R0，[R1]，#－4  :后索引偏移
SUB   R4，R2，#D4000002
                    ;该立即数是0xBE循环右移6位
                    ;课堂练习此第2操作数
```

详解第2操作数的Rm寄存器（1）

- RM寄存器通常是存放第2操作数的寄存器

<opcode>{<cond>}{S} <Rd>, <Rn> {,RM{, shift}}

- 举例：
  - SUB R1，R1，R2 ;R1－R2→R1
  - MOV PC，R0 ;PC←R0，程序跳转到指定地址
  - LDR R0，[R1]，－R2
    ;读取R1地址上的存储器单元内容并存入R0，
    ;且R1＝R1－R2，后索引偏移
  - AND R0，R5，R2
    ;R2中存放的是第2操作数，shift为空
    ;该数据属于寄存器方式的第2操作数

详解第2操作数的Rm寄存器（2）

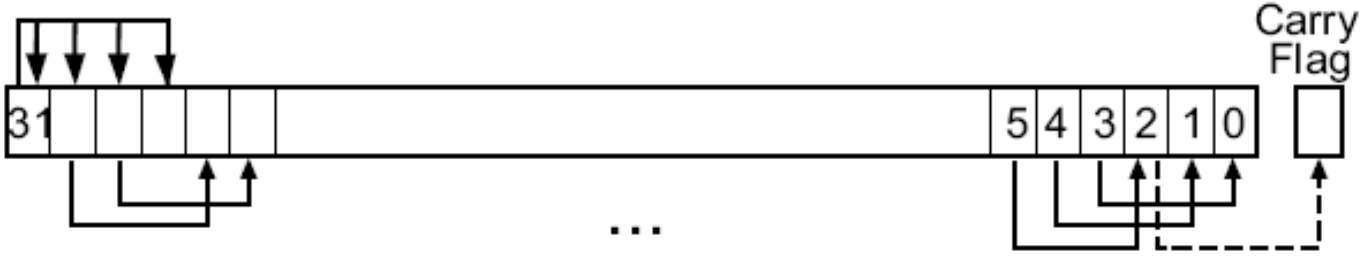
```
ADD    R0, R0, R0, LSL #2    ;执行结果R0=5*R0
ADD    R5, R3, R1, LSL #2    ;R5←R3+R1*4
ADD    R5, R3, R1, LSL R4    ;R5←R3+R1*2R4
```

寄存器移位方式生成的第2操作数 Rm{, shift}

- 将寄存器的移位结果作为操作数，但Rm值保存不变，移位方法如下：
- ASR # n ;算术右移n位(1≤n≤32)。
- LSL # n ;逻辑左移n位(1≤n≤31)。
- LSR # n ;逻辑右移n位(1≤n≤32)。
- ROR # n ;循环右移n位(1≤n≤31)。
- RRX ;带扩展的循环右移1位。

算术右移n位 ASR #n

- 算术右移n位将把寄存器Rm中的左边32-n位移动n位，复制到右边的32-n位，并且拷贝寄存器的b[31]到寄存器的左边n位。保持符号位不变。
- 程序员可以使用ASR #n操作符把寄存器的值除以2的n次方，使得结果的舍入值趋于负无穷大。
- 下图给出了ASR #3的图解。



逻辑右移n位　LSR #n

- 逻辑右移n位将把寄存器Rm中的左边32-n位移动n位，复制到右边的32-n位，并且把结果寄存器左边n位置为0。程序员可以使用LSR #n操作符把寄存器的值除以2的n次方，如果结果值被认为是一个无符号整数。
- 下图给出了寄存器Rm中的数值被逻辑右移3位的结果。

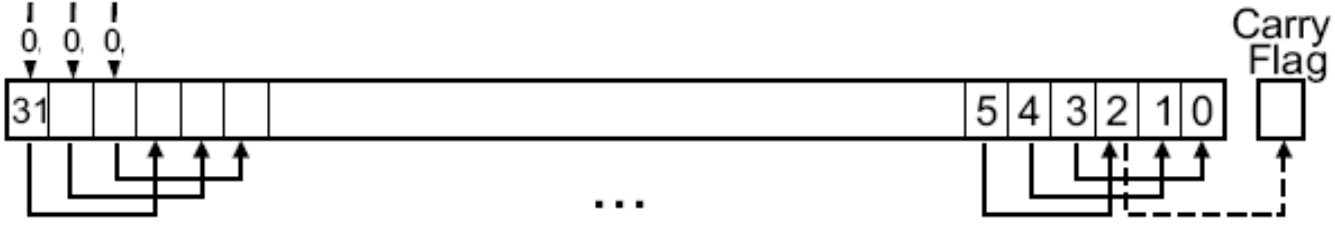


Figure 3-2 LSR #3

逻辑左移n位 LSL #n

- 逻辑左移n位将把寄存器Rm中的右边32-n位移动n位，复制到该寄存器左边的32-n位，并且把结果寄存器右边n位置为0。
- 程序员可以使用LSL #n操作符把寄存器的值乘以2的n次方，如果结果值被认为是一个无符号整数或者带符号的2的补码。这种操作对于溢出将不给出警告

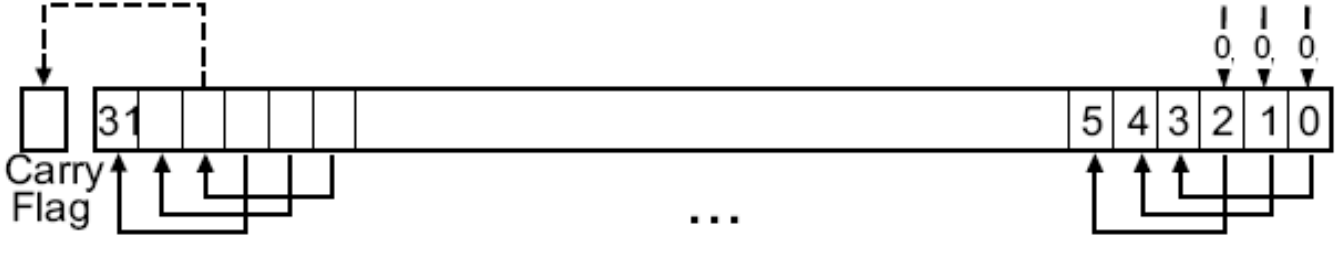


Figure 3-3 LSL #3

LSL操作时的标志位取值

- 当执行LSLS #n指令时，
- 或者

  在MOVS、MVNS、ANDS、OPRS、ORNS、EORS、BICS、TEQ或者TST指令中的第2操作数使用了ROR #n操作符（n不等于0），标志位将被更新为最后的左移出去位，即RM寄存器的bit[32-n]。

循环右移n位 ROR #n

- 循环右移n位将把寄存器Rm中的左边32-n位向右移动n位，复制到该寄存器右边的32-n位，并且把结果寄存器右边的n位移动到结果寄存器左边的n位。
- 下图给出了ROR #3的执行情况图解。

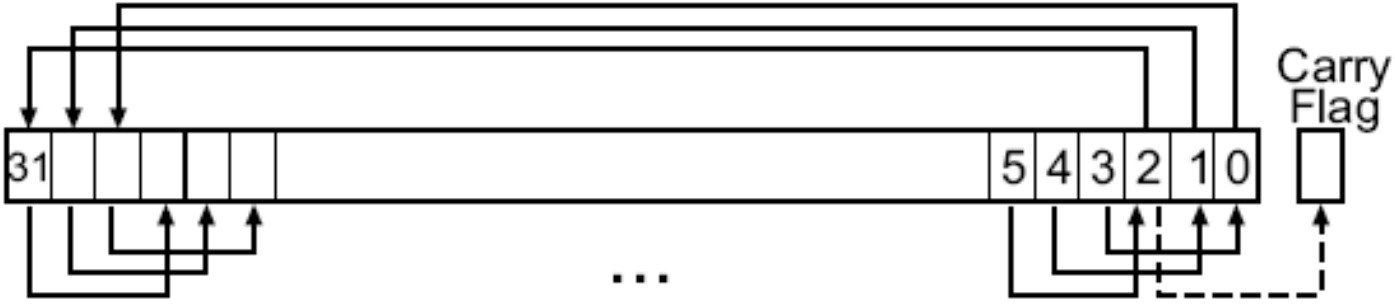


Figure 3-4 ROR #3

ROR操作时的标志位取值

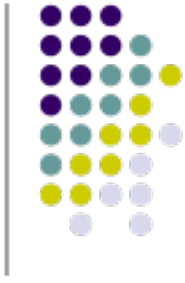
- 在以下两种情况下，标志位将被更新为最后的循环右移位，即Rm寄存器的bit[n-1]。
- 执行RORS指令
- 在MOVS、MVNS、ANDS、OPRS、ORNS、EORS、BICS、TEQ或者TST指令中的第2操作数使用了ROR #n操作符

桶型移位器移位操作：Type Rs

- 其中，**Type**为ASR、LSL、LSR和ROR中的一种；**Rs**为偏移量寄存器，最低8位有效。若其值大于或等于32，则第2个操作数的结果为0(ASR、LSR例外)。
- 例如

```
MOVS    R3, R1, LSL #7  ;R3←R1*128
```

寄存器位移方式生成第2操作数应用举例



- ADD R1, R1, R1, LSL # 3

;R1=R1*9，因为R1← R1+R1*8。

- SUB R1, R1, R2, LSR # 2

;R1＝R1－R2÷4,

;因为R2右移2位相当于R2除以4。

- EOR R11, R12, R3, ASR #5

;R11= R12⊕(R3÷32)

;第2操作数是R3的内容除以32

寄存器位移方式生成第2操作数应用举例（续）

- MOVS　R4, R4, LSR #32

;C标志更新为R4的位[31]，R4清零。

;参看本课程教材第119页

- R15为处理器的程序计数器PC，一般不要对其进行操作，而且有些指令是不允许使用R15的，如UMULL指令。

5.3 ARM处理器寻址方式

- 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，以下列出：

– 寄存器寻址 - 立即寻址
– 寄存器偏移寻址 - 寄存器间接寻址
– 基址寻址 - 多寄存器寻址
– 堆栈寻址 - 相对寻址

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
；将十六进制立即数0xFF000装入R0寄存器
- 立即数要以“#”号为前缀，16进制数值时以“0x”表示。

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 ；读取R2的值送到R1
- MOV R0，R0 ；R0=R0，相当于无操作
- SUB R0，R1，R2 ；R0←R1-R2
；将R1的值减去R2的值，结果保存到R0
- ADD R0, R1, R2 ；R0←R1+R2
；这条指令将两个寄存器（R1和R2）的内容相加，结果放入第3个寄存器R0中。必须注意写操作数的顺序：第1个是结果寄存器，然后是第一操作数寄存器，最后是第二操作数寄存器。

寄存器偏移寻址

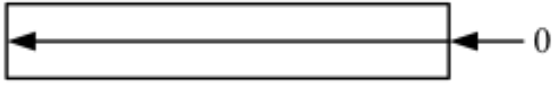
- 寄存器偏移寻址是ARM指令集特有的寻址方式。当第2作数是寄存器偏移方式时，第2个寄存器操作数在与第1操作数结合之前，选择进行移位操作。
- 寄存器偏移寻址指令举例如下：
- MOV R0, R2, LSL #3
;R2的值左移3位，结果放入R0，即R0＝R2×8
- ANDS R1, R1, R2, LSL R3
;R2的值左移R3位，然后与R1相“与”，结果放入R1，并且影响标志位。
- SUB R11, R12, R3, ASR #5
;R12—R3÷32，然后存入R11。

寄存器偏移寻址（续）

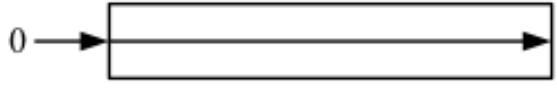
- 可采用的移位操作如下：
- **LSL**：逻辑左移(Logical Shift Left)，低端空出位补0。
- **LSR**：逻辑右移(Logical Shift Right)，高端空出位补0。
- **ASR**：算术右移(Arithmetic Shift Right)，移位过程中保持符号位不变，即若源操作数为正数，则字的高端空出的位补0；否则补1。
- **ROR**：循环右移(Rotate Right)，由字低端移出的位填入字高端空出的位。
- **RRX**：带扩展的循环右移(Rotate Right extended by 1 place)，操作数右移1位，高端空出的位用原C标志值填充。如果指定后缀“S”，则将Rm原值的位[0]移到进位标志。

移位操作示意图

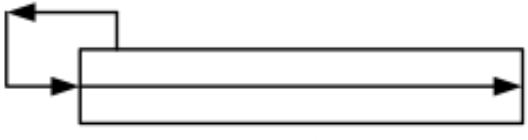
- 各种移位操作如下图所示：



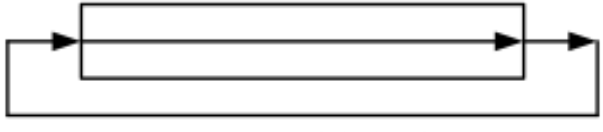
(a) LSL移位操作



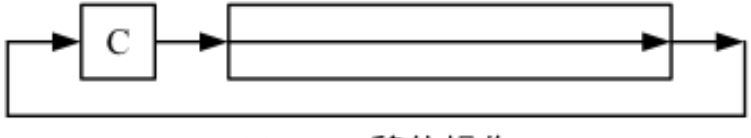
(b) LSR移位操作



(c) ASR移位操作



(d) ROR移位操作



(e) RRX移位操作

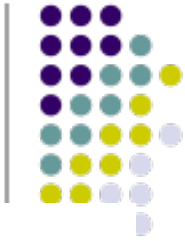
寄存器间接寻址

- 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]

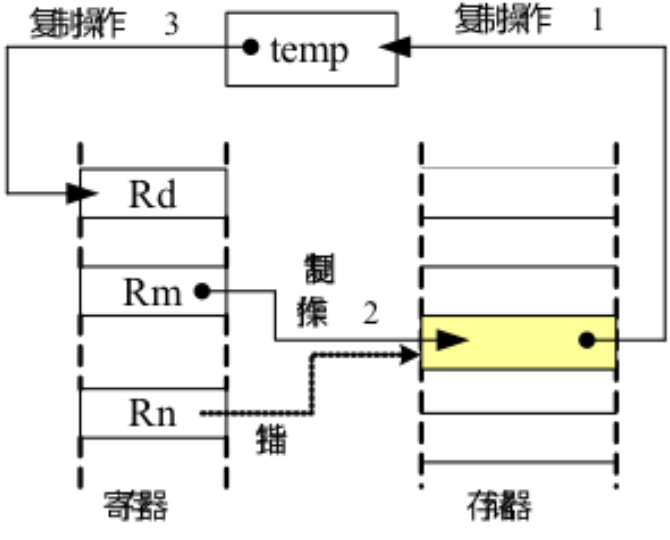
  将寄存器R1的值与R2指定的存储单元的内容交换

交换指令

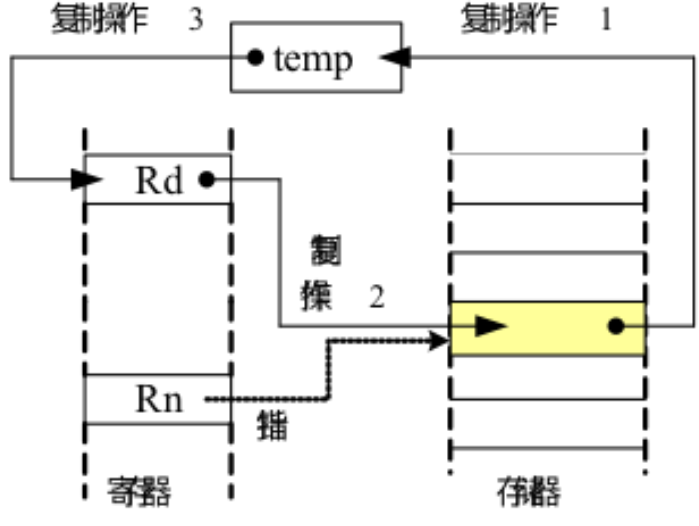


- 交换指令是Load/Store指令的特例。
- 目前，在ARM指令集里，设计了**SWP**指令用于在内存和寄存器之间交换数据。
- 该指令能够将存储单元和寄存器中的字或者无符号字节相交换。即交换数据的读取和存入组合在一条指令中。
- 通常把这2个数据传输结合成一个不能够被外部存储器访问分隔开的基本操作，因此SWP指令操作的是一个原子操作。
- 人们把SWP指令称为信号量指令。

SWP指令操作图解



(a)SWP指令的一般性操作



(b)当Rm同 Rd时，SWP指令的操作

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相加，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。基址寻址指令举例如下：
- 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]!

  ;[R0－4]←[R1]，R0=R0－4，符号“!”表明指令在完成数据传送后应该更新基址寄存器，否则不更新；属前索引。

基址寻址指令举例

- LDR R1，[R0，R3，LSL #1]
;将R0＋R3×2地址上的存储单元的内容读出，存入R1。
- LDR R0，[R1，R2，LSL #2]!
;将内存起始地址为R1+R2*4的字数据读取到R0中，
;同时修改R1，使得：R1=R1+R2*4。
- LDR R0, [R1, R2]!
;以R1+R2值为地址，访问内存。将该位置的字数据读
;取到R0中，同时修改R1，使得： R1=R1+R2。
;属于前索引指令

ARM指令LDR和STR的变址模式

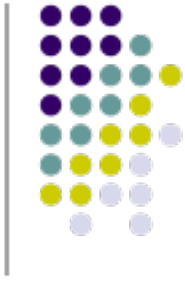


表 4-20　LDR/STR 指令的变址模式

变址模式	数据	基址寄存器	指令举例
回写前变址	mem[base+offset]	基址寄存器加偏移量	LDR r0, [r1, #4]!
前变址	mem[base+offset]	不变	LDR r0, [r1, #4]
后变址	mem[base]	基址寄存器加偏移量	LDR r0, [r1], #4

后变址：写入后变址

多寄存器寻址

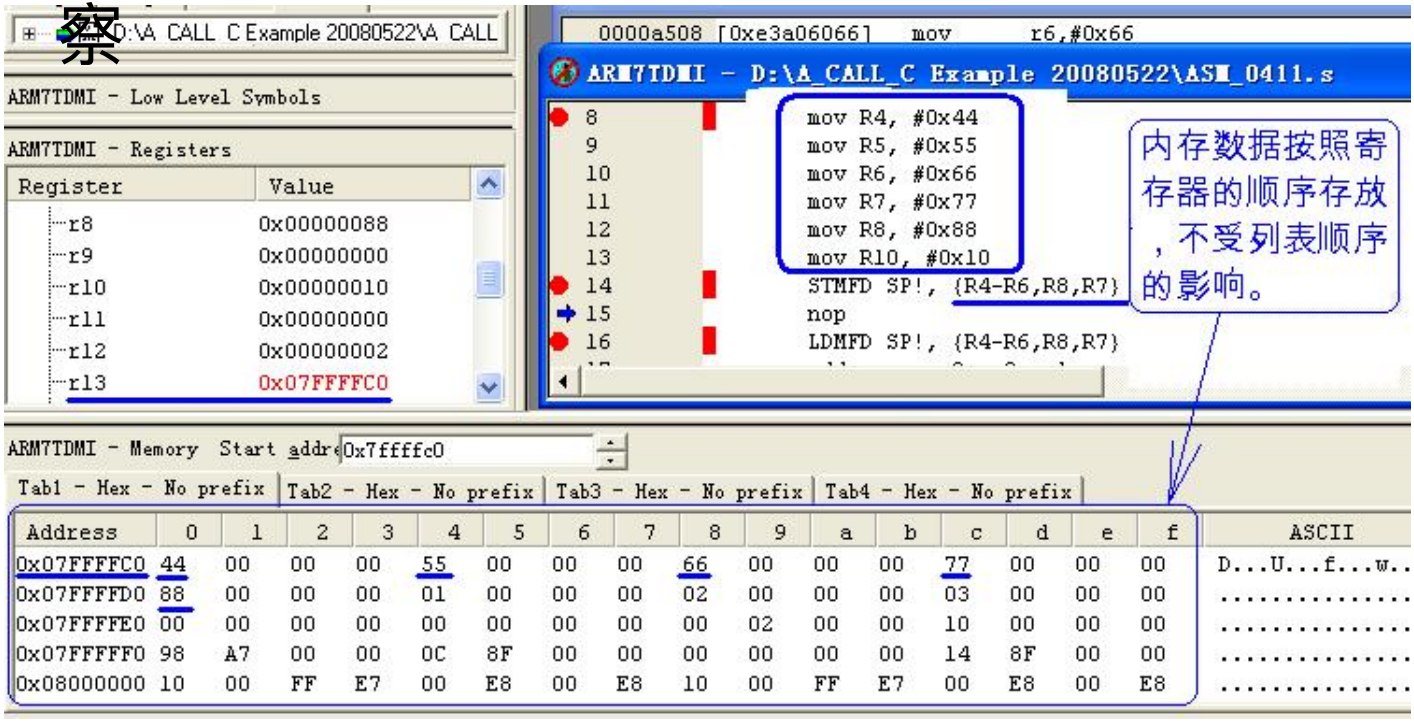
- 多寄存器寻址即是一次可传送几个寄存器值，允许一条指令传送16个寄存器的任何子集或所有寄存器。多寄存器寻址指令举例如下：
- LDMIA R1!，{R2－R7，R12}
；将R1指向的单元中的数据读出到R2~R7、R12中
；(R1自动增加)
- STMIA R0!，{R2－R7，R12}
；将寄存器R2~R7、R12的值保存到R0指向的存储单元中，
；(R0自动增加)
- 使用多寄存器寻址指令时，寄存器子集的顺序是按由小到大的顺序排列，连续的寄存器可用“－”连接；否则用“，”分隔书写。

多寄存器寻址（续1）

- 多寄存器寻址指令举例
- LDMIA R1!, {R0, R2, R5} ;
  ;R0←[R1]
  ;R2←[R1+4]
  ;R5←[R1+8]
  ;R1保持自动增值
  ;寄存器列表{R0, R2, R5}与{R2, R0, R5}等效
- 多寄存器指令的执行顺序与寄存器列表次序无关，而与寄存器的序号保持一致。

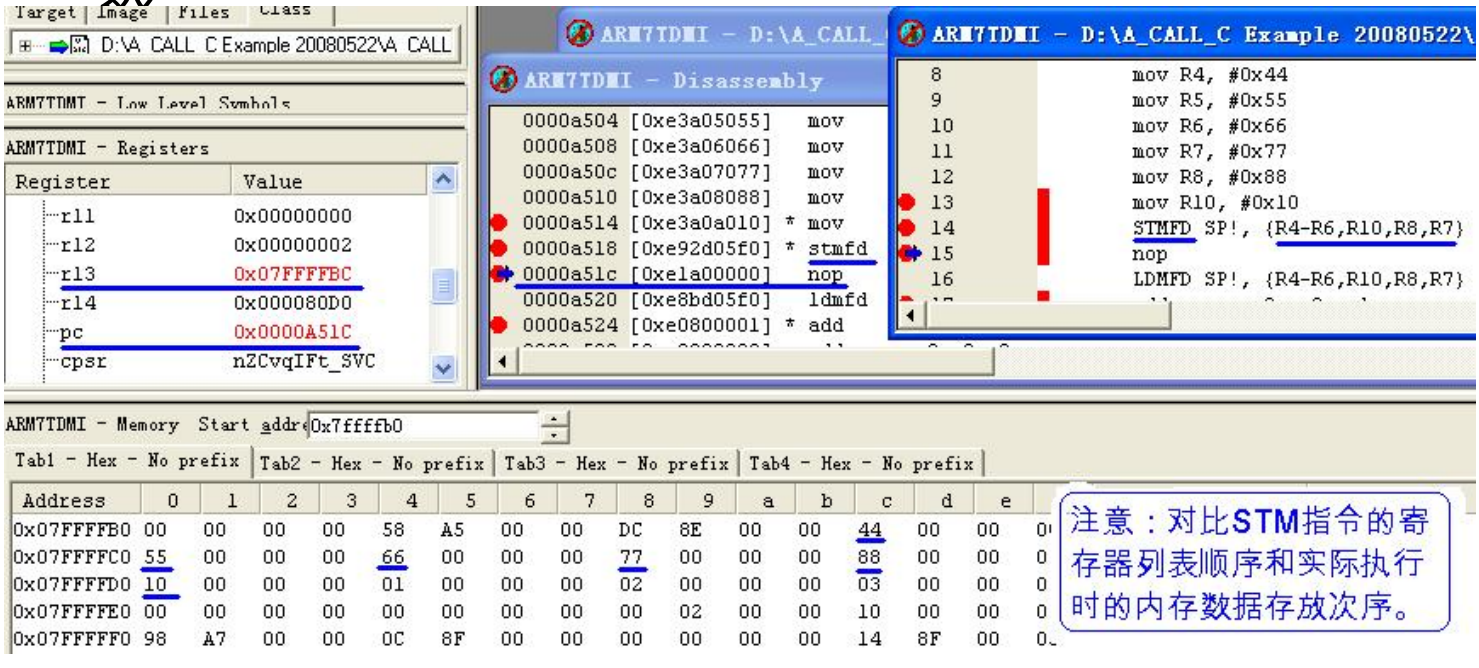
多寄存器指令的执行顺序举例1

- 通过ADS集成开发环境的AXD调试器窗口观察



多寄存器指令的执行顺序举例2

- 通过ADS集成开发环境的AXD调试器窗口观察



多寄存器寻址（续2）

- 下面是多寄存器传送指令STM举例如下：
- STMIA R0!，{R1—R7}
;将R1~R7的数据保存到存储器中。存储指针在保存第一
;个值之后增加，增长方向为向上增长
- STMIB R0!，{R1—R7}
;将R1~R7的数据保存到存储器中。存储指针在保存第一
;个值之前增加，增长方向为向上增长
- STMDA R0!, {R1—R7}
;将R1~R7的数据保存到存储器中。存储指针在保存第一
;个值之后增加，增长方向为向下增长
- STMDB R0!，{R1—R7}
;将R1~R7的数据保存到存储器中。存储指针在保存第一
;个值之前增加，增长方向为向下增长

堆栈寻址

- 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。
- 满堆栈
  - 堆栈指针指向最后压入的堆栈的有效数据项
- 空堆栈
  - 堆栈指针指向下一个待压入数据的空位置

堆栈寻址（续1）

- 有4种类型的堆栈组合
- 满递增：堆栈通过增大存储器的地址向上增长，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：堆栈通过增大存储器的地址向上增长，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：堆栈通过减小存储器的地址向下增长，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：堆栈通过减小存储器的地址向下增长，堆栈指针指向堆栈下的第一个空位置。指令如LDMED、STMED等。

堆栈寻址（续2）

- 堆栈寻址指令举例如下：
- STMFD SP!，{R1—R7，LR}
；将R1~R7、LR入栈（push），满递减堆栈。
- LDMFD SP!，{R1—R7，LR}
；数据出栈（pop），放入R1~R7、LR寄存器。
；满递减堆栈

多寄存器传送指令映射表

- STM =将寄存器内容存入内存单元（堆栈操作：入栈）
- LDM=将内存单元内容存入寄存器（堆栈操作：出栈）

传送与地址变化的关系 \ 地址变化的方向	向上生长		向下生长	
	满	空	满	空
地址增加在传送之前	STMIB STMFA			LDMIB LDMED
地址增加在传送之后		STMIA STMEA	LDMIA LDMFD	
地址减小在传送之前		LDMDB LDMEA	STMDB STMFD	
地址减小在传送之后	LDMDA LDMFA			STMDA STMED

多寄存器传送指令说明

- 数据块传送：
  - I =向地址增大方向处理数据传送(Increment)
  - D =向地址减小方向处理数据传送(Decrement)
  - A =先传送数据后改变地址(after)
  - B =先改变地址后传送数据(before)
- 堆栈操作：
  - F =满栈顶指针(full)
  - E =空栈顶指针(empty)
  - A =堆栈向高地址方向增长(ascending stack)
  - D =堆栈向低地址方向增长(decending stack)

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 ;保存子程序返回地址
；调用到SUBR1子程序
- BEQ LOOP
；条件跳转到LOOP标号处
…
- LOOP MOV R6，#1
…
- SUBR1
…

相对寻址举例

```
        BL  SUBR          ;转移到

        ADD  R1, R2, R4
        ……
        ……...
SUBR    …..               ;子程序入口
        …..
        MOV PC, R14       ;返回
```

; R14也就是LR

5.4 ARM指令集分类详解

- ARM指令集大致分为6类：分支指令、Load/Store指令、数据处理指令、程序状态寄存器指令、异常中断指令、协处理器指令。以下分别介绍其中的主要指令。

5.4.1 分支指令

- ARM有两种方法可以实现程序分支转移。
  - 跳转指令
  - 所谓的长跳转
    - 直接向PC寄存器（R15）中写入目标地址。
  - ARM跳转指令有以下4种：
    - ① B 分支指令，语法
      B{cond} label
    - ② BL 带链接分支指令
      语法：BL{cond} label
    - ③ BX 分支并可选地交换指令集
      语法：BX{cond} Rm
    - ④ BLX 带链接分支并可选择地交换指令集。
      语法：**BLX{cond} label | Rm**

BL指令举例

- BL指令的意义：Branch and Link
- 示例：

```
…….. ……..
bl MyPro    ;调用子程序MyPro
……..
MyPro          ;子程序MyPro本体
……..
……..
mov PC, LR    ;将R14的值送入R15，返回
```

BX指令使用举例

- 通过使用BX指令可以让ARM处理器内核工作状态在ARM状态和Thumb状态之间进行切换。
- 参看下例：

```
;从ARM状态转变为Thumb状态
Sub_Pro    LDR    R0, =Sub_Rout+1
           BX     R0
;从Thumb 状态转变为ARM状态
Sub_Rout   LDR    R0, =Sub_Pro
           BX     R0
```

长跳转

- 直接向PC寄存器写入目标地址值，可以实现4GB地址空间中的任意跳转。
- 示例：
- 以下的两条指令实现了4GB地址空间中的子程序调用。
- MOV LR, PC
  ;保存返回地址
- MOV R15, #0x00110000
  ;无条件转向绝对地址0x110000
  ;此32位立即数地址应满足单字节循环右移偶数次

5.4.2 Load/Store指令

- Load/Store指令用于在存储器和处理器之间传输数据。Load用于把内存中的数据装载到寄存器，Store指令用于把寄存器中的数据存入内存。
- 共有3种类型的Load/Store指令：
  - 单寄存器传输指令
  - 多寄存器传输指令
  - 交换指令

单寄存器传送指令

助记码	操作	指令描述
LDR	把一个字装入一个寄存器	Rd←mem32[address]
STR	从一个寄存器保存一个字	Rd→mem32[address]
LDRB	把一个字节装入一个寄存器	Rd←mem8[address]
STRB	从一个寄存器保存一个字节	Rd→mem8[address]
LDRH	把一个半字装入一个寄存器	Rd←mem16[address]
STRH	从一个寄存器保存一个半字	Rd→mem16[address]
LDRSB	把一个有符号字节装入寄存器	Rd←符号扩展(mem8[address])
LDRSH	把一个有符号半字装入寄存器	Rd←符号扩展(mem16[address])

Load/Store指令变址模式

- 变址模式有四种：零偏移、前变址、后变址、回写前变址 。

变址模式	数据	基址寄存器	指令举例
零偏移	mem[base]	直接基址寄存器寻址	LDR r0, [r1]
回写前变址	mem[base+offset]	基址寄存器加偏移量	LDR r0, [r1, #4]!
前变址	mem[base+offset]	不变	LDR r0, [r1, #4]
后变址	mem[base]	基址寄存器加偏移量	LDR r0, [r1], #4

单寄存器传送指令举例

- LDR R2，[R3，#0x0C]

读取R3＋0x0C地址上的一个字数据内容，放入R2。属前变址。

- STR R1，[R0，#－4]!

[R0－4]←[R1]，R0=R0－4，符号“!”表明指令在完成数据传送后应该更新基址寄存器，否则不更新；属回写前变址。

- LDR R1，[R0，R3，LSL #1]

将R0＋R3×2地址上的存储单元的内容读出，存入R1。

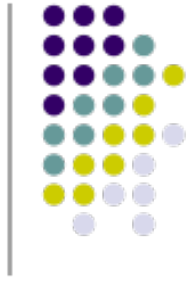
5.4.3 数据处理指令

- ARM数据处理指令大致分为以下6种类型。
  - 数据传送指令
  - 算术运算指令
  - 逻辑运算指令
  - 比较指令
  - 测试指令

ARM数据处理指令

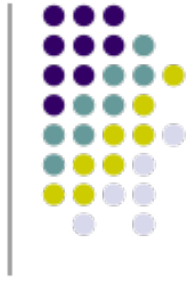
- ARM数据处理指令大致可分为3类：
  - 数据传送指令(如MOV、MVN)；
  - 算术逻辑运算指令(如ADD、SUB、AND)；
  - 比较指令(如CMP、TST)。
- 参见下面的表格
  - 数据处理指令只能对寄存器的内容进行操作。所有ARM数据处理指令均可选择使用S后缀，以影响状态标志。
  - 比较指令CMP、CMN、TST和TEQ不需要后缀S，它们会直接影响状态标志。

ARM数据处理指令集



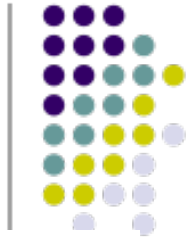
助记符	说 明	操 作	条件码位置
MOV Rd，operand2	数据传送指令	Rd←operand2	MOV{cond}{S}
MVN Rd，operand2	数据非传送指令	Rd←(~operand2)	MVN{cond}{S}
ADD Rd，Rn，operand2	加法运算指令	Rd←Rn＋operand2	ADD{cond}{S}
SUB Rd，Rn，operand2	减法运算指令	Rd←Rn－operand2	SUB{cond}{S}
RSB Rd，Rn，operand2	逆向减法指令	Rd←operand2－Rn	RSB{cond}{S}
ADC Rd，Rn，operand2	带进位加法指令	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数据处理指令集（续）



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^operand2	EOR{cond}{S}
BIC Rd，Rn，operand2	位清除指令	Rd←Rn&(~operand2)	BIC{cond}{S}
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}

5.4.4 乘法指令



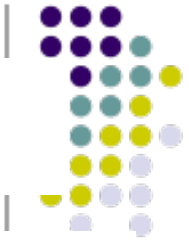
- ARM7TDMI(-S)具有32×32乘法指令、32×32乘加指令，32×32结果为64位的乘/乘加指令。ARM乘法指令如下表所列。

助记符	说 明	操 作	条件码
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}

ARM乘法指令MUL、MLA

- MUL, MLA, and MLS
  - Multiply, Multiply-Accumulate, and Multiply-Subtract, with signed or unsigned 32-bit operands, giving the least significant 32 bits of the result.
- Syntax
  - MUL{S}{cond} Rd, Rm, Rs
  - MLA{S}{cond} Rd, Rm, Rs, Rn
  - MLS{cond} Rd, Rm, Rs, Rn

课堂练习*：单条ARM指令的用法



- 试写出求解下面计算题的单条ARM指令
  - R4值同R7值的相加和存入R10寄存器
  - 如果条件码为EQ，则完成R2和R3的32位无符号乘法计算，结果存入R9。
  - R5值减去R3值，差数存入R8，运算结果影响CPSR的标志位。
  - 判断R4寄存器同R9寄存器是否相等。
  - 将R0寄存器与立即数0xFF做异或运算，结果存入R5。

5.4.5 前导零计数指令

- 在ARM v5及以上版本中含有一条特别的算术指令CLZ，用于计算操作数中前导0的个数。
- CLZ语法

  CLZ{<cond>} Rd，Rm
- 说明：CLZ（Count Leading Zeros）指令对Rm中值的前导零个数进行计数，结果放到Rd中。若源寄存器全为0，则结果为32。若位[31]为1，则结果为0。
- 举例

  下面的两条指令可以实现将寄存器R5中的数规范化。

```
CLZ    R1, R5
MOVS    R5, R5, LSL, R1
```

5.4.6 程序状态寄存器指令

- 读状态寄存器指令MRS
- 写状态寄存器指令MSR
- 指令举例
- 开中断与关中断

读状态寄存器指令MRS

- 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。
- 指令格式如下：
  - MRS{cond} Rd，psr
- 其中：
  - Rd 目标寄存器。Rd不允许为R15。
  - psr CPSR或SPSR。
- 指令举例如下：
  - MRS R1，CPSR ；将CPSR状态寄存器读取，保存到R1中。
  - MRS R2，SPSR ；将SPSR状态寄存器读取，保存到R2中。

写状态寄存器指令MSR

- 在ARM处理器中，只有MSR指令可以直接设置状态寄存器CPSR或SPSR。
- 指令格式如下：
  - MSR{cond} psr_fields，#immed_8r
  - MSR{cond} psr_fields，Rm
- 其中：
  - psr　　CPSR或SPSR。
  - fields　　指定传送的区域。

写状态寄存器指令MSR（续）

- fields可以是以下的一种或多种；(字母必须为小写)；c 控制域屏蔽字节 (psr[7…0])；x 扩展域屏蔽字节 (psr[15…8])； s 状态域屏蔽字节 (psr[23…16])；f 标志域屏蔽字节 (psr[31…24])。
- immed_8r　要传送到状态寄存器指定域的立即数，8位。
- Rm　　要传送到状态寄存器指定域的数据的源寄存器。

MSR指令举例

- MSR指令举例如下：

```
MSR  CPSR_c，#0xD3
；CPSR[7…0]=0xD3，即切换到管理模式，0b110 10011
MSR  CPSR_cxsf，R3
；CPSR=R3
```

使能IRQ中断（开中断）

```
ENABLE_IRQ
   MRS      R0, CPSR
   BIC      R0, R0, #0x80
   MSR      CPSR_c, R0
   MOV      PC, LR
```

I位=0 开中断

禁能IRQ中断（关中断）

```
DISABLE_IRQ
  MRS  R0  CPSR
  ORR  R0,  R0, #0x80
  MSR  CPSR_c,  R0
  MOV  PC,  LR
```

I位=1 关中断

MSR指令说明

- 程序中不能通过MSR指令直接修改CPSR中的T控制位来实现ARM状态/Thumb状态的切换，必须使用BX指令完成处理器状态的切换(因为BX指令属分支指令，它会打断流水线状态，实现处理器状态切换)。
- MRS与MSR配合使用，实现CPSR或SPSR寄存器的读—修改—写操作，可用来进行处理器模式切换、允许/禁止IRQ/FIQ中断等设置，如下面的程序清单所示。

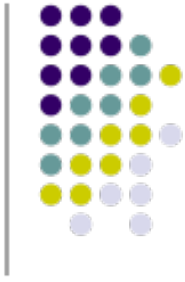
堆栈指令初始化

```
INITSTACK
    MOV  R0，LR
    ；保存返回地址
    MSR  CPSR_c，#0xD3
    LDR  SP，StackSvc
    ;设置管理模式堆栈，M[4:0]=0b10011
    MSR  CPSR_c，#0xD2
    LDR  SP，StackIrq
    ；设置中断模式堆栈，M[4:0]=0b10010
    MOV  PC，R0
```

5.4.7 软中断指令SWI

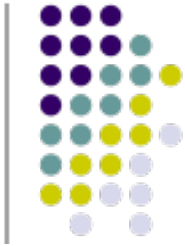
- SWI指令用于产生软中断，从而实现从用户模式变换到管理模式，CPSR保存到管理模式的SPSR中，执行转移到SWI向量。在其它模式下也可使用SWI指令，处理器同样地切换到管理模式。
- 指令格式如下：
- SWI{cond} immed_24 // Thumb指令是 immed_8
- 其中： immed_24是24位立即数，值为0~16,777,215之间的整数。立即数用于指定指令请求的具体SWI服务。
- 指令举例如下：
- SWI 0 ；软中断，中断立即数为0
- SWI 0x123456 ；软中断，中断立即数为0x123456

获得SWI指令的立即数



- 在SWI异常中断处理程序中，取出SWI立即数的步骤为：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；
- 接着读出指令，分解出立即数。
- 程序清单如下所示。

获得SWI指令的立即数（续）



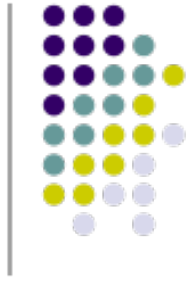
```
T_bit     EQU    0x20
SWI_Handler
  STMFD SP!，{R0－R3，R12，LR}   ；现场保护
  MRS  R0，SPSR          ；读取SPSR
  STMFD  SP!，{R0}          ；保存SPSR
  TST     R0，#T_bit       ；测试T标志位，CPSR第M5位
    ；T=1表明执行Thumb指令，参看讲义上集91页
  LDREQH R0，[LR,#－2]     ；若是Thumb指令，则读取指令码(16位)
  BICEQ  R0，R0,#0xFF00     ；取得Thumb指令的8位立即数
  LDRNE R0，[LR,#－4]        ；若是ARM指令，则读取指令码(32位)
  BICNE   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
```

5.4.8 ARM协处理器指令

- ARM支持协处理器操作。协处理器控制通过协处理器命令实现。

助记符	说 明	操 作	条件码
CDP coproc,opcode1,CRd,CRn, CRm{,opcode2}	协处理器数据操作指令	取决于协处理器	CDP{cond}
LDC{1} coproc,CRd,<地址>	协处理器数据读取指令	取决于协处理器	LDC{cond}{L}
STC{1} coproc,CRd,<地址>	协处理器数据写入指令	取决于协处理器	STC{cond}{L}

ARM协处理器指令（续）



助记符	说 明	操 作	条件码
MCR coproc,opcode1, Rd,CRn,CRm{,opcode2}	ARM寄存器到协处理器寄存器的数据传送指令	取决于协处理器	MCR{cond}
MRC coproc,opcode1, Rd,CRn,CRm{,opcode2)	协处理器寄存器到ARM寄存器的数据传送指令	取决于协处理器	MRC{cond}

5.4.9 ARM伪指令

- ARM伪指令不是ARM指令集中的指令，只是为了编程方便编译器定义了伪指令。可以像其它ARM指令一样使用伪指令，但在编译时这些指令将被等效的ARM指令代替。
- ARM伪指令有4条，分别为ADR伪指令、ADRL伪指令、LDR伪指令和NOP伪指令。

ADR伪指令

- 小范围的地址读取伪指令
- 该指令将基于PC的地址值或者基于寄存器的地址值读取到寄存器中
- 语法：
  - ADR {<cond>} register, expr
    - 其中，register为目标寄存器。expr为基于PC或者基于寄存器的地址表达式，其取值范围如下：
    - 当地址值不是字对齐时，其取值范围为-255~255。
    - 当地址值是字对齐时，其取值范围为-1020~1020。
    - 当地址值是16字节对齐时，其取值范围将更大。

ADR伪指令使用举例

- 下面是一个使用ADR伪指令的例子：
- start　MOV　R0，#1000
- 　　ADR　　R4，start
- ；案例ARM处理器是三级流水线，PC值为当前指令地址值加8字节
- ；因此本ADR伪指令将被编译器替换成机器指令
- ；SUB R4, PC, #0xC

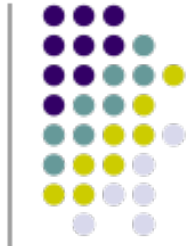
ADRL伪指令

- 中等范围的地址读取伪指令。该指令将基于PC或基于寄存器的地址值读取到寄存器中。ADRL伪指令比ADR伪指令可以读取更大范围的地址。
- ADRL伪指令在汇编时被编译器替换成两条指令。

ADRL伪指令语法

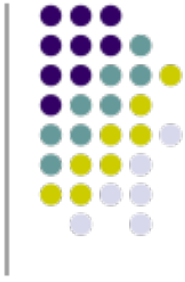
- 语法：
- ADRL {<cond>} register, expr
- 其中，register为目标寄存器。expr为基于PC或者基于寄存器的地址表达式，其取值范围如下：
- 当地址值不是字对齐时
  - 其取值范围为-64KB~64KB。
- 当地址值是字对齐时
  - 其取值范围为-256KB~256KB。
- 当地址值是16字节对齐时
  - 其取值范围将更大。

ADRL指示符的代码范例



```
            AREA    adrlabel, CODE,READONLY
            ENTRY                           ; Mark first instruction to execute
Start
            BL      func                    ; Branch to subroutine
stop        MOV     r0, #0x18               ; angel_SWIreason_ReportException
            LDR     r1, =0x20026            ; ADP_Stopped_ApplicationExit
            SWI     0x123456                ; ARM semihosting SWI
            LTORG                           ; Create a literal pool
func        ADR     r0, Start               ; => SUB r0, PC, #offset to Start
            ADR     r1, DataArea            ; => ADD r1, PC, #offset to DataArea
            ; ADR   r2, DataArea+4300       ; This would fail because the offset
                                            ; cannot be expressed by operand2
                                            ; of an ADD
            ADRL    r2, DataArea+4300       ; => ADD r2, PC, #offset1
                                            ;    ADD r2, r2, #offset2
            MOV     pc, lr                  ; Return
DataArea    SPACE   8000                    ; Starting at the current location,
                                            ; clears a 8000 byte area of memory
                                            ; to zero
            END
```

空操作伪指令NOP



- NOP伪指令在汇编时将会被替代成ARM中的空操作，比如可能为“MOV R0，R0"指令等。
- 伪指令格式如下：
- NOP

NOP指令的用法

- NOP可用于延时操作，如下面的程序清单所示。
- 软件延时程序清单

```
    …
DELAY1
    NOP
    NOP
    NOP
    SUBS    R1, R1，#1
    BNE     DELAY1
    …
```

大范围地址读取伪指令LDR

- LDR伪指令用于加载32位的立即数或一个地址值到指定寄存器。
- 在汇编编译源程序时，LDR伪指令被编译器替换成一条合适的指令。
- 若加载的常数未超出MOV或MVN的范围，则使用MOV或MVN指令代替该LDR伪指令；
- 否则汇编器将常量放入文字池，并使用一条程序相对偏移的LDR指令从文字池读出常量。
- 与ARM存储器访问指令的LDR相比，伪指令的LDR的参数有“＝”符号。

伪指令LDR格式

- 伪指令格式如下：
- LDR{cond} register，＝expr/label－expr
- 其中：
- register　　加载的目标寄存器。
- expr　　32位立即数。
- Label－expr　基于PC的地址表达式或外部表达式。

伪指令LDR举例

- LDR伪指令举例如下：

```
LDR   R0，＝0x12345678
；加载32位立即数0x12345678
LDR   R0，=DATA_BUF＋60
；加载DATA_BUF地址＋60
…
LTORG
；声明文字池
…
```

加载32位立即数程序举例

- 伪指令LDR常用于加载芯片外围功能部件的寄存器地址(32位立即数)，以实现各种控制操作，如下面的程序清单所示。

```
…
LDR  R0，=IOPIN  ;加载寄存器IOPIN的地址
LDR  R1，[R0]    ;读取IOPIN寄存器的值
…
LDR  R0，＝IOSET
LDR  R1，=0x00500500
STR  R1，[R0]    ;IOSET=0x00500500
```

自习题布置

- 题1，使用两种类型的第2操作数，分别编写3条ARM指令，并且说明这些指令中的第2操作数的形成方法。
- 题2，如何辨别LDR指令是ARM机器指令，还是伪指令。请你各举出3条数据传送LDR指令的例子和3条LDR伪指令的例子。
- 题3，LDR和STR指令有前变址、后变址和惠写前变址三种变址模式，请你举例说明之。

第7讲结束

谢谢!