## [ARM寻址方式](http://www.cnblogs.com/laojie4321/archive/2012/04/05/2432957.html)

所谓寻址方式就是**处理器根据指令中给出的地址信息来寻找物理地址的方式**。

**ARM处理器的寻址方式**

目前ARM处理器支持**9种**寻址方式，分别是**立即数寻址**、**寄存器寻址**、**寄存器偏移寻址**、**寄存器间接寻址**、**基址变址寻址**、**多寄存器寻址**、**相对寻址**、**堆栈寻址**和**块拷贝寻址**。

**1. 立即数寻址**

也叫**立即寻址**，是一种特殊的寻址方式，**操作数本身包含在指令中**，只要取出指令也就取到了操作数。这个操作数叫做立即数，对应的寻址方式叫做立即寻址。例如：

MOV R0,#64     ；R0  ← 64

ADD R0, R0, #1  ;  R0   ← R0 + 1

SUB R0, R0, #0X3D  ;  R0   ← R0 – 0X3D

在立即数寻址中，要求立即数**以“#”为前缀**，对于以十六进制表示的立即数，还要求在“#”后加上“0X”或“&”或”0x“。

在ARM处理器中，立即数必须对应8位位图格式，即立即数是一个在16位或32位的寄存器中的8bit常数，经循环移动**偶数位**得到。合法的立即数必须能够找到得到它的那个常数，否则这个立即数就是非法的。

例如：0X80是合法的，它可以通过0X80向左或向右移动0位得到，由于8位的常数都可以由其自身移动0位得到，因此**8位的立即数都是合法的**。

0X03F8也是合法的，把它写成二进制形式为：0011 1111 1000，可以看出如果使用0XFE这个8位的常数在16位寄存器中循环左移2位就可以得到0X03F8。

判断一个立即数是否合法可以用以下的办法：即对于这个立即数进行循左移或右移操作，看看经过移动**偶数位**后，**是否可以得到一个不大于0XFF的立即数（即不超过8位的立即数）**，如果可以得到，这个立即数就是合法的，否则就是非法的。象0X1010、0X1FA、0X1FF都是不合法的。

**2. 寄存器寻址**

寄存器寻址就是**利用寄存器中的数值作为操作数**，也称为**寄存器直接寻址**。

例如：ADD R0，R1， R2         ；R0  ← R1 + R2

该指令的执行效果是将寄存器R1和R2的内容相加，其结果存放在寄存器R0中。

这种寻址方式是各类微处理器经常采用的一种方式，也是执行效率较高的寻址方式。

**3. 寄存器间接寻址**

寄存器间接寻址就是把**寄存器中的值作为地址**，再通过这个地址去取得操作数，**操作数本身存放在存储器中**。

例如：

LDR R0，[R1]

；R0 ←[R1]，以寄存器R1的值作为操作数的地址，把取得操作数传送到R0中

ADD R0，R1，[R2]

；R0 ←R1 + [R2]，以寄存器R2的值作为操作数的地址，取得操作数后与R1相加，结果存入寄存器R0中。

**4. 寄存器偏移寻址**

这是ARM指令集特有的寻址方式，**它是在寄存器寻址得到操作数后再进行移位操作**，得到最终的操作数。

例如：

MOV R0，R2，LSL  #3   ；R0 ← R2 \* 8 ，R2的值左移3位，结果赋给R0。

MOV R0，R2，LSL  R1 ；R2的值左移R1位，结果放入R0。

可采用的移位操作如下：

LSL：逻辑左移（Logical Shift Left），寄存器中字的低端空出的位补0。

LSR：逻辑右移（Logical Shift Right），寄存器中字的高端空出的位补0。

ASL：算术左移（Arithmetic Shift Left），和逻辑左移LSL相同。

ASR：算术右移（Arithmetic Shift Right），移位过程中符号位不变，即如果源操作数是正数，则字的高端空出的位补0，否则补１。

ROR：循环右移（Rotate Right），由字的低端移出的位填入字的高端空出的位。

RRX：带扩展的循环右移（Rotate Right eXtended），操作数右移一位，高端空出的位用进位标志C的值来填充，低端移出的位填入进位标志位。

**5. 寄存器基址变址寻址**

寄存器基址变址寻址又称为**基址变址寻址**，它是在寄存器间接寻址的基础上扩展来的。它**将寄存器（该寄存器一般称作基址寄存器）中的值与指令中给出的地址偏移量相加，从而得到一个地址，通过这个地址取得操作数**。

例如：

LDR R0，[R1，#4]

；R0 ←[R1 + 4]，将R1的内容加上4形成操作数的地址，取得的操作数存入寄存器R0中。

LDR R0，[R1，#4]**！**

；R0 ←[R1 + 4]、R1 ←R1 + 4，将R1的内容加上4形成操作数的地址，取得的操作数存入寄存器R0中，然后，R1的内容自增4个字节。**其中！表示指令执行完毕把最后的数据地址写到R1**。

LDR R0，[R1，R2]

；R0 ←[R1 + R2]，将寄存器R1的内容加上寄存器R2的内容形成操作数的地址，取得的操作数存入寄存器R0中。

STR R0， [R1，#－4]

；R0→[R1 －4]，将R1中的数值减4作为地址，把R0中的数据存放到这个地址中。

LDR R0，[R1]，#4    ；R0 ←[R1]、R1 ←R1+4

**6. 多寄存器寻址**

这种寻址方式可以**一次完成多个寄存器值的传送**。例如：

LDM**IA**  R0，{R1，R2，R3，R4}

LDM：Load Data from Memory to Register.

；R1←[R0]，R2←[R0+4]，R3←[R0+8]，R4←[R0+12]

该指令的后缀IA表示在每次执行完加载/存储操作后，R0按字长度增加，因此，指令可将连续存储单元的值传送到R1~R4。

LDM**IA**  R0，{R1－R4}  ；功能同上。

使用多寄存器寻址指令时，寄存器子集的顺序如果由小到大的顺序排列，可以使用“－”连接，否则，用“，”分隔书写。

**7. 相对寻址**

相对寻址是一种特殊的基址寻址，特殊性是它把程序计数器PC中的当前值作为基地址，语句中的地址标号作为偏移量，将两者相加之后得到操作数的地址。

BL   NEXT  ；相对寻址，跳转到NEXT处执行。

……

……

NEXT

……

**8. 堆栈寻址**

堆栈是一种数据结构，按先进后出（First In Last Out，FILO）的方式工作，使用堆栈指针（Stack Pointer, SP）指示当前的操作位置，堆栈指针总是指向栈顶。

根据堆栈的生成方式不同，可以把堆栈分为递增堆栈和递减堆栈两种类型。如下图所示：

l  **递增堆栈**：向堆栈写入数据时，堆栈由低地址向高地址生长。

l  **递减堆栈**：向堆栈写入数据时，堆栈由高地址向低地址生长。

同时，根据堆栈指针（SP）指向的位置，又可以把堆栈分为满堆栈（Full Stack）和空堆栈（Empty Stack）两种类型。

l  **满堆栈**（Full Stack）：堆栈指针指向**最后压入堆栈的数据**。满堆栈在向堆栈存放数据时的操作是先移动SP指针，然后存放数据。在从堆栈取数据时，先取出数据，随后移动SP指针。这样保证了SP一直指向有效的数据。

l  **空堆栈**（Empty Stack）：堆栈指针SP指向下一个将要放入数据的**空位置**。空堆栈在向堆栈存放数据时的操作是先放数据，然后移动SP指针。在从堆栈取数据时，是先移动指针，再取数据。这种操作方式保证了堆栈指针一直指向一个空地址（没有有效数据的地址）。

上述两种堆栈类型的组合，可以得到四种基本的堆栈类型，即：

* 满递增堆栈（FA）：堆栈指针指向最后压入的数据，且由低地址向高地址生长。
* 满递减堆栈（FD）：堆栈指针指向最后压入的数据，且由高地址向低地址生长。
* 空递增堆栈（EA）：堆栈指针指向下一个将要压入数据的地址，且由低地址向高地址生长。
* 空递减堆栈（ED）：堆栈指针指向下一个将要压入数据的地址，且由高地址向低地址生长。

堆栈寻址举例如下：

STM**FD**  SP**！**，｛R1－R7, LR｝

STM: Store Data From register to Stack (Memory).

；将R1－R7, LR压入堆栈。满递减堆栈。

LDM**ED**  SP**！**，｛R1－R7, LR｝

；将堆栈中的数据取回到R1－R7, LR寄存器。空递减堆栈。

**9. 块拷贝寻址**

块拷贝寻址用于**寄存器数据的批量复制**，它实现从由基址寄存器所指示的一片连续存储器到寄存器列表所指示的多个寄存器传送数据。块拷贝寻址与堆栈寻址有所类似。两者的区别在于：堆栈寻址中数据的存取是面向堆栈的，块拷贝寻址中数据的存取**是面向寄存器指向的存储单元的**。

在块拷贝寻址方式中，基址寄存器传送一个数据后有4种增长方式，即：

1. IA：每次传送后地址增加4；（Increment After Operating）
2. IB：每次传送前的地址增加4；（Increment Before Operating）
3. DA：每次传送后地址减少4；（Decrement After Operating）
4. DB：每次传送前地址减少4。（Decrement Before Operating）

对于32位的ARM指令，每次地址的增加和减少的单位都是4 个字节单位。

例如：

STM**IA**  R0！，｛R1—R7｝

；将R1－R7的数据保存到R0指向的存储器中，存储器指针在保存第一个值**之后**增加4，向上增长。R0作为基址寄存器。

STM**IB**  R0！，｛R1—R7｝

；将R1－R7的数据保存到存储器中，存储器指针在保存第一个值**之前**增加4，向上增长。R0作为基址寄存器。

STM**DA**  R0！，｛R1—R7｝

；将R1－R7的数据保存到R0指向的存储器中，存储器指针在保存第一个值**之后**减少4，向下减少。R0作为基址寄存器。

STM**DB**  R0！，｛R1—R7｝

；将R1－R7的数据保存到存储器中，存储器指针在保存第一个值**之前**减少4，向下减少。R0作为基址寄存器。

ARM指令中｛！｝，为可选后缀，若选用该后缀，则当数据传送完毕之后，将最后的地址写入基址寄存器，否则基址寄存器的内容不改变。

基址寄存器不允许为R15,寄存器列表可以为R0～R15的任意组合。

｛^｝为可选后缀，当指令为LDM且寄存器列表中包含R15，选用该后缀时表示：除了正常的数据传送之外，还将SPSR复制到CPSR。同时，该后缀还表示传入或传出的是用户模式下的寄存器，而不是当前模式下的寄存器。

LDM**IA**  R0，{R1，R2，R3，R4}

与LDM**IA**  R0**！**，{R1，R2，R3，R4}的区别？

前一条指令，执行完毕之后，R0的值保持不变；后一条指令执行完毕之后，R0的值发生了变化。