**指令与伪指令的区别：**

指令是在执行阶段发挥作用的，由CPU（Intel、AMD等）来执行。

伪指令是在编译阶段发挥作用的，由汇编器（MASM、TASM等）来解释。

汇编指令帮助文档

Keil ->help->ARM Compiler 5 User’s Guides ->Assembler User Guide->Directives Reference

Keil ->help->ARM Compiler 5 User’s Guides ->Assembler User Guide->ARM and Thumb Instructions

**常用汇编指令总结：**

**EQU**： 给数字常量取一个符号名，相当于 C 语言中的 define

**AREA:** 用于定义一个代码段或是数据段。语法格式：

AREA sectionname{,attr}{,attr}…attribute

 sectionname是将要指定的节名。

可以为节选择任何名称。但是，以数字开始的名称必须包含在竖杠内，否则会产生一个缺失节名错误。 例如，|1\_D[AT](http://ddwj.net.ru/catalog.asp?cate=3" \o "单片机)aArea|。有些名称是习惯性的名称。 例如，|.text| 用于表示由 C编译器生成的代码节，或以某种方式与 C 库关联的代码节

其中：sectionname为所定义的段的名称；attr为该段的属性，具有的属性如表4-4所列。

表4-4 attr段的属性含义

|  |  |
| --- | --- |
| 伪操作 | 功能 |
| CODE | 定义代码段 |
| DATA | 定义数据段 |
| READONLY | 指定本段为只读，代码段的默认属性 |
| READWRITE | 指定本段为可读可写，数据段的默认属性 |
| ALIGN | 指定段的对齐方式为2^expession。expression的取值为0～31 |
| COMMON | 指定一个通用段。该段不包含任何用户代码和数据 |
| NOINIT | 指定此数据段仅仅保留了内存单元，而没有将各初始值写入内存单元，或者将各个内存单元值初始化为0 |

#### **SPACE和DCD**的功能类似，SPACE申请一片内存空间，DCD申请一个字(32bit)的内存空间。

#### SPACE和DCD的区别在于，SPACE申请空间但不赋初值，DCD申请一个字的空间，并赋初值。

FILL指令，申请一片内存空间并赋初值。

#### **REQUIRE8** 指令指定当前文件要求堆栈八字节对齐。 它设置 REQ8 生成属性以通知链接器。

#### **PRESERVE8**指令指定当前文件保持堆栈八字节对齐。 它设置 PRES8 编译属性以通知链接器。

**EXPORT** 声明一个标号具有全局属性，可被外部的文件使用

**DCD** 以字为单位分配内存，要求 4 字节对齐，并要求初始化这些内存

**PROC**  定义子程序，与 ENDP 成对使用，表示子程序结束

**WEAK**  弱定义，如果外部文件声明了一个标号，则优先使用外部文件定义的

标号，如果外部文件没有定义也不出错。要注意的是：这个不是 ARM

的指令，是编译器的，这里放在一起只是为了方便。

**IMPORT** 声明标号来自外部文件，跟 C 语言中的 EXTERN 关键字类似

**B** 跳转到一个标号 B .表示一直在当前指令处原地跳转，也就是死循环。

**ALIGN**  编译器对指令或者数据的存放地址进行对齐，一般需要跟一个立即

数，缺省表示 4 字节对齐。要注意的是：这个不是 ARM 的指令，是

编译器的，这里放在一起只是为了方便。

**END**  到达文件的末尾，文件结束

**MRS** 状态寄存器到通用寄存器的传送指令。

(move special to register)

**MSR** 通用寄存器到状态寄存器的传送指令。

(move register to special)

**CBZ** :Compare and Branch on Zero

**CBNZ** :Compare and Branch on Non-Zero

**[LDR指令和LDR伪指令区别](http://www.cnblogs.com/simonshi/archive/2010/04/23/1718687.html)**

ARM指令集中，LDR通常都是作加载指令，但是它也可以作伪指令。  
初学者一般不会注意到它们的区别，其实在嵌入式开发过程中，这两条指令时非常常用的！我们应该了解他们的区别。

LDR伪指令的形式是“LDR Rn,=expr”。作用是装在一个32bit常数和一个地址到寄存器。  
下面举一个例子来说明它的用法。

COUNT EQU       0x56000054  
LDR       R1,=COUNT  
MOV       R0,#0  
STR       R0,[R1]

COUNT是我们定义的一个变量，地址为0x56000054。

LDR       R1,=COUNT  是将COUNT这个变量的地址，也就是0x56000054放到R1中。  
MOV      R0,#0是将立即数0放到R0中。

STR   R0,[R1]  是一个典型的存储指令，将R0中的值放到以R1中的值为地址的存储单元去。  
实际就是将0放到地址为0x56000054的存储单元中去。  
可见这三条指令是为了完成对变量COUNT赋值。

再举一个例子  
LDR       R1,=COUNT   ；这条伪指令，是将COUNT的地址赋给R1  
LDR       R0,[R1]   ；将COUNT的值赋给R0  
   
ARM是RISC结构，数据从内存到CPU之间的移动只能通过LDR/STR指令来完成。   
比如想把数据从内存中某处读取到寄存器中，只能使用ldr   
比如：   
ldr r0, 0x12345678 ；就是把0x12345678这个地址中的值存放到r0中。

而mov不能干这个活，mov只能在寄存器之间移动数据，或者把立即数移动到寄存器中，这个和x86这种CISC架构的芯片区别最大的地方。   
x86中没有ldr这种指令，因为x86的mov指令可以将数据从内存中移动到寄存器中。   
MOV是从一个寄存器或者移位的寄存器或者立即数的值传递到另外一个寄存器。  
从本质上是寄存器到寄存器的传递，为什么会有立即数，其实也是有限制的立即数，不是所有立即数都可以传递的  
这个立即数要符合一个8位数循环右移偶数位的取值。  
原因是，MOV本身就是一条32bit指令，除了指令码本身，它不可能再带一个可以表示32bit的数字，所以用了其中的12bit来表示立即数，其中4bit表示移位的位数(循环右移，且数值x2)，8bit用来表示要移位的一个基数。

另外还有一个就是ldr伪指令，虽然ldr伪指令和ARM的ldr指令很像，但是作用不太一样。ldr伪指令可以在立即数前加上=，以表示把一个地址写到某寄存器中，比如：   
ldr r0, =0x12345678   
这样，就把0x12345678这个地址写到r0中了。所以，ldr伪指令和mov是比较相似的。只不过mov指令限制了立即数的长度为8位，也就是不能超过512。而ldr伪指令没有这个限制。如果使用ldr伪指令时，后面跟的立即数没有超过8位，那么在实际汇编的时候该ldr伪指令是被转换为 mov指令的。

**LDR[伪指令]**  加载一个立即数或者一个地址值到一个寄存器。举例：LDR Rd, = label，

如果 label 是立即数，那 Rd 等于立即数，如果 label 是一个标识符，比如

指针，那存到 Rd 的就是 label 这个标识符的地址

**LDR** 从存储器中加载字到一个寄存器中

**LDRH** (Load Register Halfword )从存储器中加载半字到一个寄存器中

**LDRB**  (Load Register Byte)从存储器中加载字节到一个寄存器中

**STR**  把一个寄存器按字存储到存储器中

**STRH**  (Store Register Halfword) 把一个寄存器存器的低半字存储到存储器中

**STRB**  (Store Register Byte) 把一个寄存器的低字节存储到存储器中

**LDMIA** (Load Multiple registers )加载多个字，并且在加载后自增基址寄存器

**STMIA (**Store Multiple registers ) 存储多个字，并且在存储后自增基址寄存器

**IA** Increment address After each transfer. This is the default, and can be omitted.

**ORR**  按位或

**BX**  (Branch and exchange instruction set.) 直接跳转到由寄存器给定的地址

**BLX**  Branch with Link and exchange instruction set

跳转到由寄存器 REG 给出的的地址，并根据 REG 的 LSB 切换处理器状

态 ， 还 要 把 转 移 前 的 下 条 指 令 地 址 保 存 到 LR 。 ARM(LSB=0) ，

Thumb(LSB=1)。Cortex-M3 只在 Thumb 中运行，就必须保证 reg 的

LSB=1，否则一个 fault 打过来