ADR是一条小范围的地址读取伪指令,它将基于PC的相对偏移的地址值读到目标寄存器中。格式：ADR register,exper。

编译源程序时，汇编器首先计算当前PC值（当前指令位置）到exper的距离,然后用一条ADD或者SUB指令替换这条伪指令，

例如:ADD register,PC,#offset\_to\_exper。

**注意**，标号exper与指令必须在同一代码段。

比如：adr r0, \_start ：//将指定地址赋到r0中

    .........

\_start:
    b \_start

r0的值为标号\_start与此指令的距离差 + PC值。

**ADRL**:

这是一条中等范围的地址读取伪指令，它将基于PC的相对偏移的地址值读到目标寄存器中。格式：ADRL register,exper。编译源程序时，汇编器会用两条合适的指令替换这条伪指令。

比如：

ADD register,PC,offset1

ADD register,register,offset2

与ADR相比，它能读取更大范围的地址。

**注意**，标号exper与指令必须在同一代码段。

接下来是**LDR**，首先要说两个家伙，他们都叫LDR。

**一个是LDR伪指令，一个是LDR指令**，名字相同却不是一个东西。

区分的方法就是看第二个参数，**如果有等号，就是伪指令**。

**LDR指令：**

例： ldr r0, 0x12345678

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。

**LDR伪指令：**

例1(立即数)： 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，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伪指令会被转换为mov指令。

例2(标号)： ldr r0, =\_start //将指定标号的值赋给r0

这里取得的是标号\_start的绝对地址，这个绝对地址（运行地址）是在链接的时候确定的。它要占用 2 个32bit的空间，一条是指令，另一条是文字池中存放\_start 的绝对地址。

**对比adr r0, \_start和 ldr r0, =\_start**

它们的目的一样，都是把标签的赋给r0，区别---左边是相对地址，右边绝对地址。目的一样，但结果不一定相同。结果是否相同，要看PC值是否和链接地址相同。