很多人在写简单的裸机代码或分析uboot时，常常遇到adr ldr指令。却分不清这2者的区别，今天就来谈谈adr与ldr指令。

参照韦老师的代码和Makefile写了**test\_adr.S：**

.text

.globl \_start

\_start:  
    ldr r0, test  
    adr r0, test  
    ldr r0, =test  
    nop

test:

nop

**Makefile：**

all:test\_adr.S

       arm-linux-gcc -c -o test\_adr.o test\_adr.S  
         arm-linux-ld -Ttext 0x00000000 -gtest\_adr.o -o test\_adr\_elf  
        arm-linux-objcopy -O binary -S test\_adr\_elf test\_adr.bin  
        arm-linux-objdump -D -m arm test\_adr\_elf test\_adr.dis

clean:

        rm -ftest\_adr.dis test\_adr.bin test\_adr\_elf \*.o

反汇编test\_adr.S得到**test\_adr.dis：**

test\_adr\_elf:  
file format elf32-littlearm

Disassembly of section .text:

00000000 \_start:

0: e59f0008 ldr r0, [pc, #8]; 10 test

4: e28f0004 add r0, pc, #4; 0x4

8: e59f0004 ldr r0, [pc, #4]; 14.text+0x14

c: e1a00000 nop (mov r0,r0)

00000010 test:

10:e1a00000 nop (mov r0,r0)

14:00000010 andeq r0, r0, r0, lsl r0

很显然，ldr获取的是内存的值（至于这个内存存的是数据还是地址，不是问题重点），像指针一样间接寻址（看到了[]符号咯），而adr是得到一个与PC有关的值，必定是个地址。

韦老师举了个例子：

**adr r0, \_start，r0就是\_start对应指令当前的地址**

对于“\_start对应指令当前的地址”，我理解了很久，终于想清楚，比如在uboot中，\_start标号对应的指令（即b reset）的链接地址是0x33f80000确凿无疑。

如果从NOR Flash启动，b reset被烧在NOR Flash 0地址，那么b reset相对于此时的PC来说，它的地址就是0。

如果u-boot被直接下载到SDRAM的0x33f80000处运行，那么b reset自然处在SDRAM的0x33f80000。

**所谓“当前”---是以运行时的PC为参照。**

下面基于以上理解，分析**test\_adr.dis**

00000000 \_start:

0: e59f0008 ldr r0, [pc, #8]; 10 test

4: e28f0004 add r0, pc, #4; 0x4

8: e59f0004 ldr r0, [pc, #4]; 14.text+0x14

c: e1a00000 nop (mov r0,r0)

00000010 test:

10:e1a00000 nop (mov r0,r0)

14:00000010 andeq r0, r0, r0, lsl r0

1、先分析第一条指令ldr r0,test被编译成ldr  
r0, [pc, #8]，即到当前PC+8的存储器取值，运行第一条指令时，PC其实已经是8了（流水线决定的）。

那么8+8等于0x10，所以r0等于e1a00000，此指令的作用就是读取test地址处存放的值。由于此处放了一条nop，即得到nop的机器码。

2、第二条adr r0,test被编译成add r0, pc, #4  
这显然是依赖程序执行到此处的PC值。**ADR是小范围地址读取伪指令，会将基于PC 相对偏移的地址值读取到寄存器中，**此指令在4地址，PC是4+8=0xc再加4，于是r0=0x10。

从结果上来看，test自身的值（标号值），被读到了r0，这个值是以PC为参考的，也就是test对应的指令（第二个nop）当前的地址。**r0=（标号test的地址与此指令的距离差）+（此指令的地址）=（（0x10-0x4=12）+（4））=16=0x10。**

假如在0x30000000以上运行，r0=（（12）+（0x30000004））= 0x30000010。

3、ldr r0,=test被编译成两个字，一个指令，一个文字池。执行到这里PC=8, 8+8+4=0x14，所以在14地址取值，编译器在14地址处放了0x00000010，0x00000010是test的值，假如在Makefile指定连接地址是0x30000000，那么编译器放在这里的就是0x30000010，可见，这个值是编译时确定的。

最后一行andeq r0, r0, r0, lsl r0大概是编译器的机械动作，把一个数字翻译成了指令。

**总结**

ADR是小范围的地址读取伪指令，它将基于PC 相对偏移的地址值读取到寄存器中。而ldr获取的是内存的值，像指针一样间接寻址。