1、ABI和EABI

ABI(Application Binary Interface), EABI(Embeded application Binary Interface)，即编译器将C代码编译成汇编代码时使用的一种规则。一般包括

1. C类型的表示（int, short, long, union…）
2. 调用约定，包括如何传递参数和返回值；使用寄存器和堆栈。

2、ARM浮点运算

（1）硬浮点

编译器将代码直接编译成硬件浮点协处理器(浮点运算单元FPU)能识别的指令，这些指令在执行的时候ARM核直接把它转给协处理器执行。FPU 通常有一套额外的寄存器来完成浮点参数传递和运算。使用实际的硬件浮点运算单元(FPU)会带来性能的提升。

使用硬浮点时，需要给编译器传递-mfpu和-mfloat-abi参数，让编译器编译出硬件浮点单元（fpu）处理器能识别的指令

（2）软浮点

编译器把浮点运算转成浮点运算的函数调用和库函数调用，没有FPU的指令调用，也没有浮点寄存器的参数传递。浮点参数的传递也是通过ARM寄存器或者堆栈完成。如果系统没有任何浮点处理器单元，使用hard-float就会产生非法指令和异常。因而一般的系统镜像都采用软浮点以兼容没有VFP的处理器。

3、编译时参数：-march=armv7-a -mfloat-abi=softfp -mfpu=neon-vfpv4

（1）-march=armv7-a：指定编译时arm架构（即代码要运行的架构）。

arm处理器每个核使用不同的arm体系结构版本，每个版本对应的指令集也可能不同；编译优化时可能根据架构指令集不同进行不同的优化。

（2）-mfloat-abi=softfp：soft/softfp/hard

soft（软浮点）：表明不是用FPU硬件，使用GCC整数算术库来模拟浮点运算

softfp（硬浮点）：表明要使用FPU硬件来做浮点运算，函数的参数传递到整数寄存器（r0-r3）中，然后再传递到FPU。目的是为了生成的代码采用兼容软浮点调用接口(即使用-mfloat-abi=soft时的调用接口)，这样带来的好处是：兼容性和灵活性。实际也可以这样应用：库可以采用-mfloat-abi=soft编译，而关键的应用程序可以采用-mfloat-abi=softfp来编译。

hard（硬浮点）：表明要使用FPU硬件来做浮点运算，并且函数的参数直接传递到FPU的寄存器（s0、d0）。这样要求所有库和应用程序必须采用这同一个参数来编译，否则连接时会出现接口不兼容错误

用-mfloat-abi=soft编译的app或者库，在用-mfloat-abi=softfp编译的OS中是可以跑的；

用-mfloat-abi=softfp编译的app或者库，在用-mfloat-abi=soft编译的OS中，如果SoC中没有FPU，那么是不能跑的；

-mfloat-abi=softfp/soft与-mfloat-abi=hard，是互不兼容的。

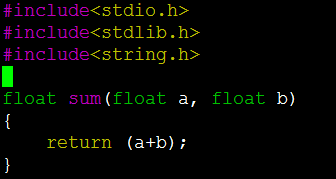
Armv7系列基本都有硬浮点，所以一般都选择softfp的方式。

（3）-mfpu=neon-vfpv4

参数-mfpu就是用来指定要产生那种硬件浮点运算指令。常用的有vfpv3，vfpv4，neon等，hi3536 A17支持的是neon+vfpv4相结合的结构。

4、编译参数添加与否对比

（1）源码

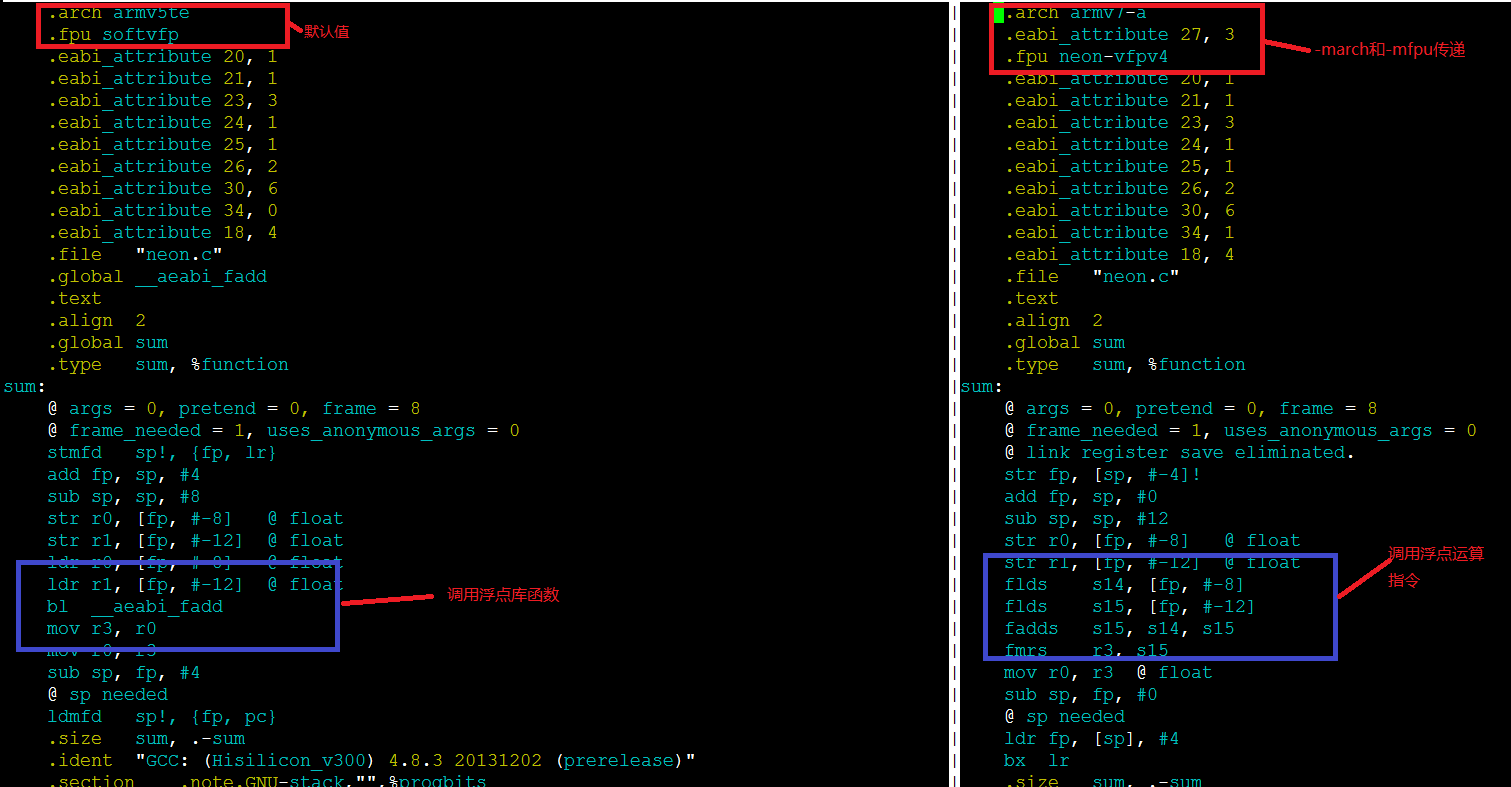


（2）编译成汇编

arm-hisiv300-linux-gcc -S -o neon1.S neon.c

arm-hisiv300-linux-gcc -S -march=armv7-a -mfloat-abi=softfp -mfpu=neon-vfpv4 -o neon2.S neon.c

（3）汇编对比



左图为不传递编译参数，右图传递编译参数。

从图中可知arm-hisiv300-linux-gcc默认是armv5te和softfp的方式，走的是软浮点。按理neon指令是vxxx而这里是fxxx，这是可能是gcc符号没更新，使用objdump可以看到vxxx。