# [ARM和NEON指令 very nice](http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/18263977)

2014-01-14 15:18 6810人阅读 [评论](http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/18263977" \l "comments)(2) [收藏](http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/javascript:void(0);" \o "收藏) [举报](http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/18263977" \l "report" \o "举报)

IMG_256 分类：

ARM&NEON（5） IMG_257

[转自：](http://blogs.arm.com/software-enablement/684-coding-for-neon-part-5-rearranging-vectors/" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

http://blog.csdn.net/chshplp\_liaoping/article/details/12752749

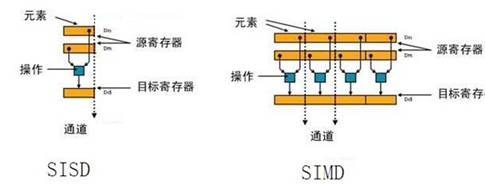
在移动平台上进行一些复杂算法的开发，一般需要用到指令集来进行加速。目前在移动上使用最多的是ARM芯片。

ARM是微处理器行业的一家知名企业，其芯片结构有：armv5、armv6、armv7和armv8系列。芯片类型有：arm7、arm9、arm11、cortex系列。指令集有：armv5、armv6和neon指令。关于ARM到知识参考：[http://baike.baidu.com/view/11200.htm](http://baike.baidu.com/view/11200.htm" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

最初的ARM指令集为通用计算型指令集，指令集都是针对单个数据进行计算，没有并行计算到功能。随着版本的更新，后面逐渐加入了一些复杂到指令以及并行计算到指令。而NEON指令是专门针对大规模到并行运算而设计的。

NEON 技术可加速多媒体和信号处理算法（如视频编码/解码、2D/3D 图形、游戏、音频和语音处理、图像处理技术、电话和声音合成），其性能至少为ARMv5 性能的3倍，为 [ARMv6](http://www.arm.com/zh/products/processors/instruction-set-architectures/index.php" \o "指令集架构" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank) SIMD性能的2倍。

关于SIMD和SISD：Single Instruction Multiple Data，单指令多数据流。反之SISD是单指令单数据。以加法指令为例，单指令单数据（[SISD](http://baike.baidu.com/view/2115849.htm" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)）的CPU对加法指令译码后，执行部件先访问内存，取得第一个[操作数](http://baike.baidu.com/view/420846.htm" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)；之后再一次访问内存，取得第二个操作数；随后才能进行求和运算。而在SIMD型的CPU中，指令译码后几个执行部件同时访问内存，一次性获得所有[操作数](http://baike.baidu.com/view/420846.htm" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)进行运算。这个特点使SIMD特别适合于多媒体应用等数据密集型运算。如下图所示：



如何才能快速到写出高效的指令代码？这就需要对各个指令比较熟悉，知道各个指令的使用规范和使用场合。

ARM指令有16个32位通用寄存器，为r0-r15，其中r13为堆栈指针寄存器，r15为指令计算寄存器。实际可以使用的寄存器只有14个。r0-r3一般作为函数参数使用，函数返回值放在r0中。若函数参数超过4个，超过到参数压入堆栈。

有效立即数的概念：每个立即数采用一个8位的常数(bit[7:0])循环右移偶数位而间接得到，其中循环右移的位数由一个4位二进制(bit[11:8] )的两倍表示。如果立即数记作<immediate> , 8位常数记作immed\_8 , 4位的循环右移值记作rotate\_imm ,有效的立即数是由一个8位的立即数循环右移偶数位得到，可以表示成：

<immediate>=immed\_8循环右移( 2×rotate\_imm)

如：mov r4 , #0x8000 000A    #0x8000 000A 由0xA8循环右移0x2位得到。

下面介绍一些比较常用到一些指令。

**内存访问指令：**

LDR和STR，有三种方式，比较容易搞混

LDR r0, [r1, #4]   r0 := mem[r1+4]   ，#4是直接偏移量，这时候只能在正负4Kb到范围内。也可以是寄存器偏移，用+/-表示。记住r1不进行偏移。

LDR r0, [r1, #4]!  r0 :=mem[r1+4]，r1 := r1 + 4，取值是取偏移量到值，并且r1进行偏移。

LDR r0, [r1], #4   r0 :=mem[r1] ，r1 := r1 +4，取值是取r1地方到值，取值后进行偏移。运算后自动加4，后变址。

另外：LDRB是无符号字节，SB是有符号字节，H无符号半字，SH有符号半字。

存储器和寄存器数据交换：SWP，SWPB

如SWP r0, r1, [r2]   r0 := mem[r2]，mem[r2] := r1

多寄存器数据传输：

LDMIA r1, {r0,r2,r5}  r0 = mem[r1], r2 = mem[r1+4], r5=mem[r1+8]

**通用数据处理指令**

第二操作数，常用到有LSR，LSL等，如mov r1, r2, lsl #2 将r2左移2位然后赋值到r1中。

常用到操作有ADD、SUB、AND、ORR、EOR、BIC、ORN，如果加上了S则会更新条件标记。

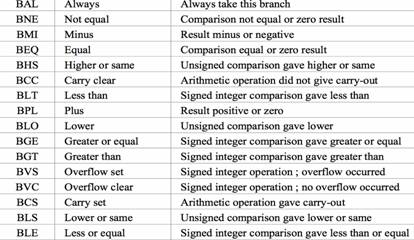
MOV移动，MVN取反移动。MOV可以是R寄存器，立即数以及接第二操作数。

REV：在字或半字内反转字节或位到顺序

MUL、MLA和MLS，乘法、乘加和乘减。MLA R1，R2，R3，R4表示R1=R2\*R3+R4，还有有符号和无符号乘法等。

**跳转指令**

B：无条件跳转，BL：带链接到跳转，BX跳转并交换指令集等。



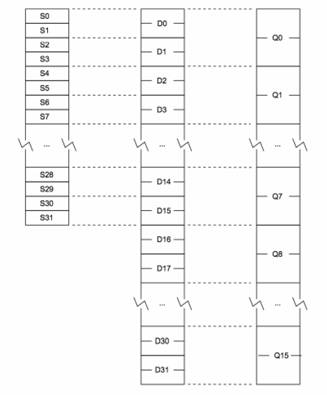
重点介绍一下NEON指令，目前使用较多。而且使用难度也较大，很多文档上都没有比较详细到介绍，也没有给出相应到例子或者图示。

一、NEON基本知识

**NEON的寄存器：**

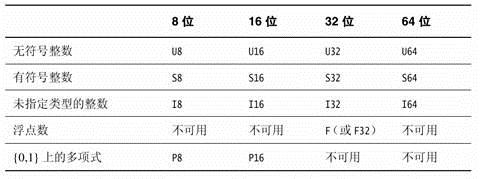
有16个128位四字到寄存器Q0-Q15，32个64位双子寄存器D0-D31，两个寄存器是重叠的，在使用到时候需要特别注意，不小心就会覆盖掉。如下图所示：

两个寄存器的关系：Qn =D2n和D2n+1，如Q8是d16和d17的组合。



**NEON的数据类型：**

注意数据类型针对到时操作数，而不是目标数，这点在写的时候要特别注意，很容易搞错，尤其是对那些长指令宽指令的时候，因为经常Q和D一起操作。



**NEON中的正常指令、宽指令、窄指令、饱和指令、长指令**

正常指令：生成大小相同且类型通常与操作数向量相同到结果向量

长指令：对双字向量操作数执行运算，生产四字向量到结果。所生成的元素一般是操作数元素宽度到两倍，并属于同一类型。L标记，如VMOVL。

宽指令：一个双字向量操作数和一个四字向量操作数执行运算，生成四字向量结果。W标记，如VADDW。

窄指令：四字向量操作数执行运算，并生成双字向量结果，所生成的元素一般是操作数元素宽度的一半。N标记，如VMOVN。

饱和指令：当超过数据类型指定到范围则自动限制在该范围内。Q标记，如VQSHRUN

**二、NEON指令**

NEON指令较多，下面主要介绍一些常见的指令用法。

**复制指令：**

VMOV：

两个arm寄存器和d之间

vmov d0, r0, r1：将r1的内容送到d0到低半部分，r0的内容送到d0到高半部分

vmov r0, r1, d0：将d0的低半部分送到r0，d0的高半部分内容送到r1

一个arm寄存器和d之间

vmov.U32 d0[0], r0：将r0的内容送到d0[0]中，d0[0]指d0到低32位

vmov.U32 r0, d0[0]：将d0[0]的内容送到r0中

立即数：

vmov.U16 d0, #1：将立即数1赋值给d0的每个16位

vmov.U32 q0, #1：将立即数1赋值给q0的每个32位

长指令：VMOVL：d赋值给q

vmovl.U16 q0, d0：将d0的每个16位数据赋值到q0的每个32位数据中

窄指令：VMOVN：q赋值给d

vmovn.I32 d0, q0：将q0的每32位数据赋值到q0的每16位数据中

饱和指令：VQMOVN等，饱和到指定的数据类型

 vqmovun.S32 d0, q0：将q0到每个32位移动到d0中到每个16位中，范围是0-65535

VDUP：

VDUP.8 d0, r0：将r0复制到d0中，8位

VDUP.16 q0, r0：将r0复制到q0中，16位

VDUP.32 q0, d2[0]：将d2的一半复制到q0中

VDUP.32 d0, d2[1]：将d2的一半复制到d0中

注意是vdup可以将r寄存器中的内容复制到整个neon寄存器中，不能将立即数进行vdup，立即数只能用vmov

**逻辑运算**：

VADD：按位与；VBIC：位清除；VEOR：按位异或；VORN：按位或非；VORR：按位或

**移位指令：**

VSHL：左移、VSHLL：左移扩展、VQSHL：左移饱和、VQSHLU：无符号左移饱和扩展

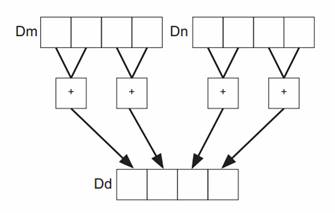
VSHR：右移、VSHRN：右移窄、VRSHR：右移舍入、VQSHRUN：无符号右移饱和舍入

**通用算术指令：**

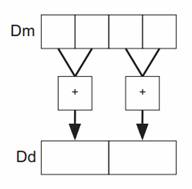
VABA：绝对值累加、VABD：绝对值相加、VABS：绝对值、VNEG：求反、VADD、VADDW、VADDL、VSUB、VSUBL、VSUBW：加减

VPADD：将两个向量的相邻元素相加

如VPADD.I16 {d2}, d0, d1



VPADDL：VPADDL.S16 d0, d1

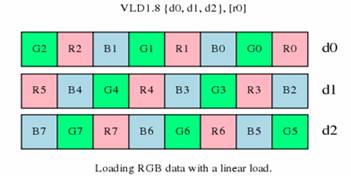


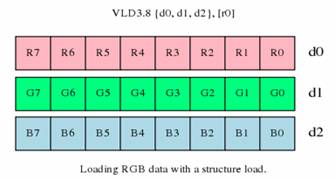
VMAX：最大值，VMIN：最小值

VMUL、VMULL、VMLA（乘加）、VMLS（乘减）、

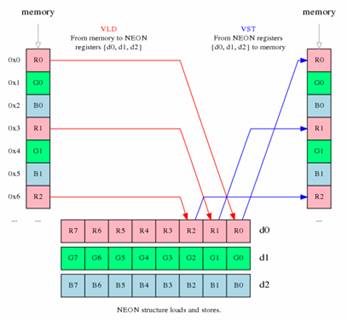
**加载存储指令：**

VLD和VST

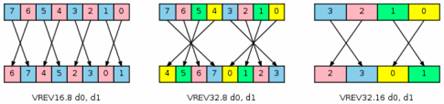




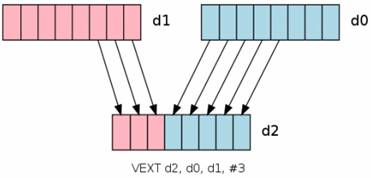
交叉存取的示意图：



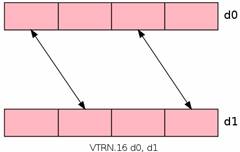
VREV反转元素指令：

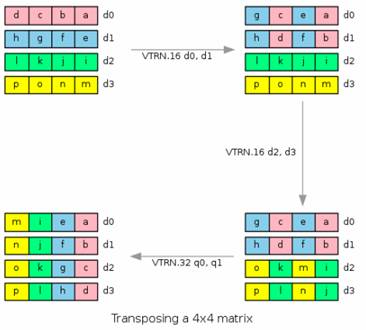


VEXT移位指令：

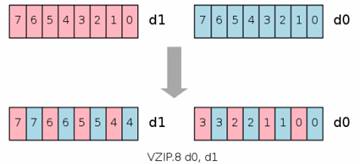


VTRN转置指令：可以用于矩阵的转置

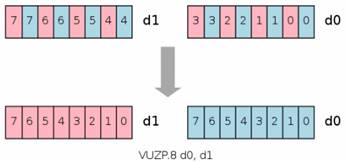




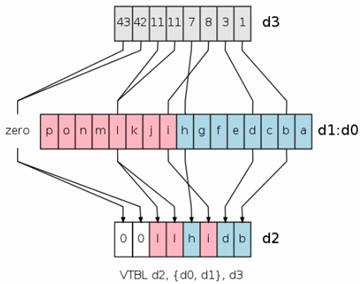
VZIP指令：压缩，类似交叉存取



VUZP指令：解压操作，类似交叉存取



VTBL查表指令：从d0，d1中查找d3中的索引值，如果找到则取出，没有找到则为0，存入d2中



三、需要注意的地方

    load数据的时候，第一次load会把数据放在cache里面，只要不超过cache的大小，下一次load同样数据的时候，则会比第一次load要快很多，会直接从cache中load数据，这样在汇编程序设计的时候是非常需要考虑的问题。

     如：求取一个图像的均值，8\*8的窗口，先行求和，然后列求和出来均值，这时候会有两个函数，数据会加载两遍，如果按照这样去优化的话则优化不了多少。如果换成上面这种思路，先做行16行，然后再做列，这样数据都在cache里面，做列的时候load数据会很快。

   在做neon乘法指令的时候会有大约2个clock的阻塞时间，如果你要立即使用乘法的结果，则就会阻塞在这里，在写neon指令的时候需要特别注意。乘法的结果不能立即使用，可以将一些其他的操作插入到乘法后面而不会有时间的消耗。

如：vmul.u16 q1, d3, d4

         vadd.u32 q1, q2, q3

此时直接使用乘法的结果q1则会阻塞，执行vadd需要再等待2个clock的时间

使用饱和指令的时候，如乘法饱和的时候，在做乘法后会再去做一次饱和，所以时间要比直接做乘法要慢。

如：  vmul.u16 q1, d3, d4

          vqmul.u32 q1, q2, q3

后一个的时间要比第一个的时间要久。

在对16位数据进行load或者store操作的时候，需要注意的是字节移位。比如是16位数据，则load 8个16位数据，如果指定寄存器进行偏移，此时需要特别注意。

例如：vld1.64 {d0}, [r0], r1

参考资料：

[http://blogs.arm.com/software-enablement/277-coding-for-neon-part-4-shifting-left-and-right/](http://blogs.arm.com/software-enablement/277-coding-for-neon-part-4-shifting-left-and-right/" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

[http://blogs.arm.com/software-enablement/161-coding-for-neon-part-1-load-and-stores/](http://blogs.arm.com/software-enablement/161-coding-for-neon-part-1-load-and-stores/" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

[http://blogs.arm.com/software-enablement/684-coding-for-neon-part-5-rearranging-vectors/](http://blogs.arm.com/software-enablement/684-coding-for-neon-part-5-rearranging-vectors/" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

[转自：http://blog.csdn.net/chshplp\_liaoping/article/details/12752749](http://blogs.arm.com/software-enablement/684-coding-for-neon-part-5-rearranging-vectors/" \t "http://blog.csdn.net/yxnyxnyxnyxnyxn/article/details/_blank)

# [关于neon](http://blog.csdn.net/ccjjnn19890720/article/details/7291228)

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)](http://blog.csdn.net/ccjjnn19890720/article/details/7291228" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/ccjjnn19890720/article/details/7291228" \o "展开)

这个学期的任务在开学的时候就已经定了下来.和老师聊过关于自己的想法,老师也表示了自己的想法,后来发现老师想的还是比我周到点.导师的意思是让我继续搞我的ARM板子,一直觉得搞这个没有搞PC机来的好.因为现在的板子的硬件已经提高了很多,可以说比以前的pc机都要强悍很多,在上面作开发让我觉得与在pc机上作开发没有什么区别.这样的思想在上个学期一直在我脑袋中存在呢,所以很多时候做事不是很积极的.但是后来导师说了,说你认为在上面做简单是因为你还没有深入进去,你还没了解它比较内涵的东西.那个时候的我觉得导师是忽悠我的,或者我真的只是想作一个软件开发者,而不喜欢自己去碰硬件,总是想和老师说这些.

大概老师是看穿我的思想了,在开学的时候就把任务给我安排好了.不能让我随便打酱油了,而且还和我说了我现在到底是往哪个方向发展.其实这个学期的任务看上去其实很简单的,就是把程序在板子能跑的更加快.那么要达到这个目的的最为主要的就是优化.导师的意思是说我要往底层发展,做一个硬件工程师,当然与那些画板子的工程师不一样的,我的人物就是提出自己的硬件要求和对程序进行最大和最深的优化.

一直作的都是C级别的优化,可是在上个学期的测试中发现这种级别的优化是很水的,提高的程度不是很客观.现在要往更为底层的发展.要接触硬件要接触汇编.这让我认清了自己的道路,而且本人对这个还是很感兴趣的.

接下来就要讲的就是**NEON**这个东西了.

一开始见到它还觉得它是一个可有可无的技术,后来看了很多的文章以后发现这个才是优化的突破口.

#### 1.什么是SIMD ####

其实这个术语我很久以前就已经听过了,可以一直只有表面的理解,不就是单指令多数据流吗.难道这个还能比MIMD还要牛逼吗,一直觉得这是一种可有可无的技术,就算就在今天我还是觉得这样的技术就是过渡性的技术.可是今天看了一篇文章以后才发现它都有用之处.   
通常我们进行多媒体处理的时候,很多的数据都是16位或者8位的,如果这些程序运行在32位的机器上,那么计算机有一部分的计算单元是没有工作的.所以这是一种浪费.为了更好的使用那些被浪费的资源.SIMD就应运而生了.SIMD这种技术就是使用一条指令，但对多个相同类型和尺寸的数据进行并行处理.就像我们现实生活中的好几个人都在做同一件事情那样,这样就可以将速度提升很多倍.

#### 2.什么是NEON

**ARMv7**体系结构介绍了增强性SIMD的扩展,它通过定义大量了在64位和128位的寄存器指令用来扩展了SIMD的概念.   
据文档上了解NEON就是将增强性SIMD的扩展使用在ARM处理器上的实现.也就是说NEON是指用一种特定的方式去实现的过程.现在ARM Cortex-A8的系列是具有着NEON.具有NEON技术的处理器都会配备了32个64位的寄存器和16个128位的寄存器,它们分别被标识为(D0-D31),(Q0-Q15)

NEON的指令集只是ARM和THUMB指令集中的子集,ARM和THUMB要管理所有程序流和同步的问题.NEON指令通常执行的就是：

* 内存的访问
* 在NEON寄存器和传统寄存器之间的数据拷贝
* 数据类型的转化
* 数据的处理

#### 3.关于NEON的编程模式

如何才能去优化NEON的代码呢？？

这就是怎么样去使自己编写的代码能整合进NEON的技术,这样编译器就可以在编译的过程中尽可能的使用NEON的处理方式.

要使用NEON,那就使用NEON本身支持的指令效果最好了,那么NEON本身提供了什么样的指令呢？？

就像上面所说那样,本身NEON指令提供的功能也很有限,主要在于数据的装载和存储,以及数据的处理.这应该是NEON指令的核心能力了.NEON的指令都是以V字母开头的.

VADD.I16 q0,q1,q2

这就是一个NEON的指令了,很明显的特点就是V开头,I主要用来表明是一个整型,16表示一个16位的整型,q0,q1,q2都是128位的寄存器(q打头的寄存器都是128位的).其实这个指令的意思就是让q1,q2中装载8个16位的数据,然后执行加法操作,最后放到q0中去.这么一个指令就完成了8次加法运算,也就是性能的提升.

像上面的指令,NEON还有很多.比如还提供比如RGB三个元素,分别取到3个不同的寄存器中,每一个寄存器中的元素要不都是R要不都是G要不都是B,这样一来就可以进行并行运算了.

既然知道了这些指令,那么使用到程序中去的话有几种手段呢？？

1. C语言级别的C语言级别的优化当然是对高级语言这个阶段来说的,对于NEON来说,这个级别的优化不是指那些很通用的优化手段,虽然那些能达到优化的作用,但是没有充分的利用NEON这的技术,所以很多时候你会发现,网上很多人说这个板子很厉害很厉害,可是你在上面开发的程序就是很慢很慢,这也就说明了你没有充分的利用其中的硬件资源.

C语言的优化可以分为两个手段:

* 内联函数

内联函数有可能会被误会成我们通常想的那样,主要是这个单词我翻译不正确(intrinsics).这样的函数可以被C和C++的程序所调用.看上去和别的函数没有很多的区别,最多也就名字比较古怪.但是其实当这个代码在被编译器编译的时候,它会被转化为有序的低级指令.这些指令就是NEON的指令了.所以这样就办到了在高级语言层次使用低级语言了.主要是很简单的可以使用.最为主要的就是程序员不用去接触汇编了,可以减小优化的难度.当然我可以说这样的优化效率没有使用汇编的来的高.

对于上面的这种技术其实就是ARM公司本身给你做好了一些函数,你就直接调用这些函数,这些函数在编译的时候就可以直接转化成NEON的汇编指令.为了支持这些内联的函数所以必须要包含头文件arm\_neon.h.

#include <arm\_neon.h>

uint32x4\_t double\_elements(uint32x4\_t input)

{

return(vaddq\_u32(input, input));

}

像vaddq\_u32这样的函数,gcc还提供很多很多,如果想知道更多可以查看[gcc\_neon](http://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html).   
当然使用了NEON技术以后必须还要通过在编译的时候加入-mfpu=neon才能起到效果的.整个编译的命令就是arm-none-linux-gnueabi-gcc -mfpu=neon intrinsic.c

* 汇编级别的

这个级别的优化是比较难的,但是也是最为有效的.通过对NEON指令的掌握然后对程序中消耗最大的那个部分进行汇编级别的改造,这样会有很大的区别.我昨天看了一篇文章上讲了,c级别的改造只提高性能1.5倍,但是汇编级别的就是7.5倍,从中可以看出之间的差距了吧.

* 使用库

ARM公司好像本身对NEON的技术做成了一个库(OPENMAX),里面都是ARM公司作了NEON作的优化程序.所以可以使用这些库函数来完成一些功能.可以在[ARM](http://blog.csdn.net/ccjjnn19890720/article/details/www.arm.com)

#### 4. 总结

NEON的知识点还有很多,不过核心的就是这么一些其他就是它的扩展.要具体问题具体分析,对于现在我在做cortex-A8我觉得这个技术应该对我的帮助会很大...接下来就试试这个技术吧.