江苏中科龙梦科技有限公司

龙芯 SIMD 指令使用手册

修 订 记 录

项 次	修订日期	版 本	修订内容	修订者	审 核
1	2009-02-04	0.1	初版发行		

DISCLAIMER

THIS DOCUMENTATION IS PROVIDED FOR USE WITH LEMOTE PRODUCTS. NO LICENSE TO LEMOTE PROPERTY RIGHTS IS GRANTED. LEMOTE ASSUMES NO LIABILITY, PROVIDES NO WARRANTY EITHER EXPRESSED OR IMPLIED RELATING TO THE USAGE, OR INTELLECTUAL PROPERTY RIGHT INFRINGEMENT EXCEPT AS PROVIDED FOR BY LEMOTE TERMS AND CONDITIONS OF SALE.
LEMOTE PRODUCTS ARE NOT DESIGNED FOR AND SHOULD NOT BE USED IN ANY MEDICAL OR LIFE SUSTAINING OR SUPPORTING EQUIPMENT.
ALL INFORMATION IN THIS DOCUMENT SHOULD BE TREATED AS PRELIMINARY. LEMOTE MAY MAKE CHANGES TO THIS DOCUMENT WITHOUT NOTICE. ANYONE RELYING ON THIS DOCUMENTATION SHOULD CONTACT LEMOTE FOR THE CURRENT DOCUMENTATION AND ERRATA.

JIANGSU LEMOTE TECHNOLOGY CORPORATION LIMITED

MENGLAN INDUSTRIAL PARK,YUSHAN,CHANGSHU CITY,JIANGSU PROVINCE,CHINA

Tel: 0512-52308661
Fax: 0512-52308688
Http: //www.lemote.com

目录

一．SIMD简介

SIMD (Single Instruction Multiple Data，单指令多数据流),能在一条指令内完成多对数据运算,实质上是通过并行技术,来提高处理器的吞吐量。在软件优化中，使用 SIMD 发挥程序的并行性非常有效和必要,尤其在数字图像处理中这种技术得到了广泛的应用。

二．SIMD指令分类

根据 SIMD 的实现方式，SIMD 指令可分为 3 类：

1. mmx 指令。通过使用浮点寄存器的 mmx 指令实现，一般用于整数型（8,16,32 位）数据的处理。所有的龙芯 2F 支持的 mmx 指令见附录 A
2. 双单精度指令（paired-single,简称 ps）。特征是 opcode 以.ps 结尾，用于浮点型数据的处理（一次进行两对 float 数据的运算）。具体见附录 B
3. 普通指令。比如在不存在溢出问题的情况下，两个 short 数组相加，先分别 ld 到通用寄存器，再使用一条 dadd 指令就实现了 4 次加法。

可见，SIMD 不必非要用 mmx 指令实现，它只是一种思想而已。

三．SIMD主要参考文档

龙芯 2F 的用户手册

mips64 指令集手册

comcat 写的关于龙芯多媒体指令的文档

四．SIMD汇编程序编写方法

汇编语言作为一种低级语言，由于其移植性、可读性和可维护性的限制一般较少使用，但在十分强调效率或者和底层打交道的情形下，汇编语言仍是重要手段。

汇编语言在使用中有两种形式：

1. 完整的汇编文件（文件名以 s 或 S 结尾）
2. c 代码中内嵌汇编

完整汇编文件的典型例子有 pmon 源码的 start.S 和 mplayer 源码下的 libvo/godson_memcpy.S（大 S 表示需要使用 c 的预处理）。平时一般使用内嵌汇编的方式使用汇编指令，因为这样比较方便，不需要修改 makefile 文件，也便于调试。在处理得当的情况下和完全用汇编写效果相同（可通过反汇编可执行文件看出，反汇编命令为 objdump -mmips:loongson_2f -d 文件名）。开始阶段多看反汇编后的代码对程序汇编编写非常有好处，可以了解汇编器常见的优化方法，另外还可以知道内嵌汇编代码如何与 c 代码的上下文结合，对寄存器使用有指导意义，比如你将会发现内嵌代码破坏部的寄存器在什么情况下不需要入栈保存等。

龙芯内嵌汇编的方法与在 x86 下没有实质区别，以龙芯 2 F 为例，一般框架如下：

```
__asm__(
```

```
".set noreorder\n"
".set arch=loongson2f\n"
。。。。。。。。//具体的汇编程序
".set noreorder\n"
:输出部
:输入部
:破坏部
);
```

其中的.set 语句都不是死板的，使用时可灵活使用。稍后会有一个简单的例子。

注意如果开头使用了.set noreorder,则输出部前的.set noreorder 不可省略，否则会出现不可预料的结果。

一般破坏部要使用 memory 这个标记，具体功能可网上查询。

在 o32 系统中，只能使用 16 个偶数号的 64 位浮点寄存器（$f0,$f2,$f4...),如果要使用$f1 这种寄存器，则只能当 32 位来使用。硬件上，龙芯完全支持 32 个 64 位浮点寄存器。

五．使用SIMD指令的例子

这里举个简单的例子，两个长度均为 LEN 的 float 型数组 a, b 求和,结果放回 a 数组。为比较 c 版本和内嵌汇编版本的时间差，这个操作循环 100 次，代码中假定的常量 LEN 为非负数。

c 代码 test-c.c ：

```
#define     LEN    1000000
void main()
{
      float a[LEN] __attribute__((aligned(8))) ;//设定 8 字节地址对齐
      float b[LEN] __attribute__((aligned(8))) ;
      int j,i;

      for(j = 0 ; j < LEN ; j++){    //初始化
            a[j]    = 1.2345 + j;
            b[j]    = 6.5432 + j;
      }

      for(i=0;i<100;i++)
            for( j = 0 ; j < LEN ; j++){
                  a[j] = a[j] + b[j];
            }
}
```

内嵌汇编的代码 test-asm.c ：

```
#define    LEN    1000000
void main()
{
//不对齐的话，下面的 ldc 1 会产生不对齐的异常，提示总线错误
   float a[LEN] __attribute__((aligned(8))) ;
   float b[LEN] __attribute__((aligned(8))) ;
     int j , i;

      for(j = 0 ; j < LEN ; j++){    //  初始化
            a[j]    = 1.2345 + j;
            b[j]    = 6.5432 + j;
      }
for(i=0;i < 100 ;i++)
      __asm__(
            ".set noreorder\n"
            ".set arch=loongson2f\n"
            "1:\n"
            "slti $8,%2,2\n"    //  判断待操作的数据是否小于 2
            "bnez $8,2f\n"
            "addiu %2,-2\n"    //利用延时槽
            "ldc1 $f0,0(%0)\n"//在龙芯的世界里，字 32 位，双字 64 位
            "ldc1 $f2,0(%1)\n"//ldc1 读取两个 float，利用 64 位的总线宽度
            "add.ps $f0,$f0,$f2\n"// add.ps 指令并行做了两对 float 的加法
            "sdc1 $f0,0(%0)\n"
            "addiu %0,8\n"
```

```
            "b 1b\n"
            "addiu %1,8\n" //  延时槽
            "2:\n"              //程序执行到此表示待操作的数据小于 2,为 0 或 1
            "beq %2,$0,3f\n"
            "nop\n"          //这个延时槽没有合适的指令填充,用 nop 填充
            "lwc1 $f0,0(%0)\n"
            "lwc1 $f2,0(%1)\n"
            "add    $f0,$f0,$f2\n"
            "swc1 $f0,0(%0)\n"
            "3:\n"
            ".set reorder\n"
            ::"r"(a),"r"(b),"r"(LEN)
            :"$8","$f0","$f2","memory"
        );
    }
```

以上的汇编代码对于初学者可能有点复杂，其中有个新概念，延时槽（delay slot），具体的解释和使用注意事项可参看《see mips run》第一章末的讲述。

以下输出是在我的龙芯 2 F 福珑 6003 上运行的结果：

先编译

```
RAYS-b0f748fa:/tmp# gcc test-c.c -o test-c
test-c.c: In function ‘main’:
test-c.c:3: warning: return type of ‘main’ is not ‘int’
RAYS-b0f748fa:/tmp# gcc test-asm.c -o test-asm
test-asm.c: In function ‘main’:
test-asm.c:3: warning: return type of ‘main’ is not ‘int’
```

再看执行时间

```
RAYS-b0f748fa:/tmp# time ./test-c
real      0m7.929s
user      0m7.890s
sys       0m0.035s

RAYS-b0f748fa:/tmp# time ./test-asm
real      0m3.763s
user      0m3.703s
sys       0m0.055s
```

以上是个极端的例子。实际工作中，这种情形较少，一般都需要经过一定的转化来使程序适于用 SIMD 指令来实现。这两个文件如果在编译时加了 O2 的优化选项，即编译命令为

```
gcc test-c.c -o test-c -O2
gcc test-asm.c -o test-asm -O2
```

结果是：

```
RAYS-b0f748fa:/tmp# time ./test-c
real    0m0.004s
user    0m0.000s
sys     0m0.004s

RAYS-b0f748fa:/tmp# time ./test-asm
real    0m0.093s
user    0m0.070s
sys     0m0.023s
```

这次结果是 c 版本的快很多，如果你感到困惑，那么反汇编一把。并将可执行文件 test-c 反汇编,结果重定向到 temp 文件中。

```
RAYS-b0f748fa:/tmp# objdump -mmips:loongson_2f -d test-c > temp
```

在这 temp 文件中搜索 main，发现 main 编译后只有两句代码（红字标明的代码）。

```
  400658:   2484082c    addiu   a0,a0,2092
  40065c:   03e00008    jr  ra
  400660:   00000000    nop
    ...

00400670 <main>:
  400670:   03e00008    jr  ra
  400674:   00000000    nop
    ...

00400680 <__libc_csu_fini>:
  400680:   03e00008    jr  ra
  400684:   00000000    nop
```

可见编译器还是很聪明的。 以上只是个例子，在mplayer中有许多地方使用了simd的功能，当然一般都是混杂的，不会有这么纯粹的simd,典型的如mp3lib这个库的龙芯版代码。mplayer的源码可通过git协议从http://dev.lemote.com 下载。

六．编译工具

除了 gcc，编译龙芯的汇编程序需要有龙芯支持的 binutils，如果系统自带的汇编器未加上龙芯补丁，则某些指令不识别，只要从源上 apt-get 安装上 binutils 即可。可以编写个简单的内嵌程序测试一下，假如能识别.set arch=loongson2f 的一般就行了。极少数的 2F 手册上列出的指令仍没得到最新汇编器的支持，比如 madd.ps 这条指令（也许过些天就支持了也没准）。最新的工具链版本可 lomote 的 deb 源中获得 deb http://dev.lemote.com/debian-loongson loongson contrib main non-free,apt-get install binutils 即可

七．程序常见问题及调试

暂时在内嵌汇编中使用 gdb 调试不是很方便，调试也往往成为代码编写任务的瓶颈。根据个人经验，汇编程序编写人员初期需要特别注意的几个常见问题是：

1. 32 位和 64 位指令的混用。比如 sll 只针对 32 位，dsll 针对 64 位
2. 类型扩展相关问题。包括短类型向长类型转换，比如

char->short,float->double 等。

3. 符号问题。使用指令时要注意，类型转换时也要注意，比如 lw 和 lwu 这一对要严格区分
4. 数组下标问题。比如 int 型数组相邻元素之间地址相差为 4，short 型数组相邻元素之间地址相差为 2，load 和 store 时切记。
5. 延时槽问题。延时槽不建议存放会影响跳转方向的指令，不允许放跳转指令。
6. 一些限制，比如 dsll 的常数域不能超过+31，那么要将一个寄存器左移 40 位的话就要使用 dsll32 这种指令，通常超过范围的常数使用会在编译时警告，这种警告不可不理会。

以上这些都是我开始写汇编时曾遇到过的问题。

在本文例子的编写中，由于键入错误，我开始将"addiu %0,8\n"写成了"addu %0,8\n"，这种错误不会得到任何的警告，需格外注意。汇编时几个常见的错误提示为:

1.总线错误

2.段错误

3.error: invalid 'asm': operand number out of range

4.非法指令

除了第 3 个是在编译时报外，其余都是运行是报错。

在前面的例子编写中，如果将"ldc1 $f2,0(%1)\n"写成了"ldc1 $f2,2(%1)\n",这样 ldc1 的那个地址就不是 8 字节对齐了，就会出现总线错误，注意 ld 指令的地址未对齐不会报总线错误，那种异常由内

核处理。

如果循环条件"addiu %2,-2\n" 这句写忘了，运行时报段错误,表示访问了无权限访问的地址。

如果将"addiu %2,-2\n" 的%2 改成%3,编译时就报错误 3，因为传入的参数只有 3 个，没有%3（表示第四个参数）。

如果将代码开头的初始化部分注释掉，会报非法指令，因为浮点有 ieee 标准，不是任何数据都可以解析成正确的浮点数。非法指令也有可能是由内存分配失败却继续执行、特权指令或某些 cpu 不能识别的指令导致的。比如以下代码，可以编译，但运行时报非法指令。

```
  float a[2] __attribute__((aligned(8))),b[2],c[2];
 a[0]=a[1]=12.2;
 b[0]=b[1]=12.2;
 c[0]=c[1]=12.2;
__asm__(
        ".set arch=loongson2f\n"
        ".set noreorder\n"
        "ldc1 $f0,0(%0)\n"
        "ldc1 $f2,0(%1)\n"
        "ldc1 $f4,0(%2)\n"
        "madd.ps $f4,$f0,$f2\n"
        ".set reorder\n"
        ::"r"(a),"r"(b),"r"(c)
        :"$f0","$f2","memory"
        );
```

就是由于汇编器翻译后的机器码不能为 2F cpu 所识别导致，这种错误极少，待汇编器升级后这种错误可以得到消除。所以遇到非法指令要小心处理，浮点例外参见 2F 用户手册第 7 章浮点异常一节。

打印寄存器的值是调试的最基本方法，具体就是在内嵌汇编输入部传入一个临时变量的地址，store 某个寄存器到那个地址，再打印

临时变量即可，在代码不长的情形下，还是很有用的。

八．性能分析

在实际情况中，使用嵌入汇编一般的目的是提高程序效率。性能分析的最简单方法是 top 看一个程序优化前后 cpu 占用比，使用 oprofile 的精度更高，一般能准确统计函数的 cpu 占用情况，但个人经验是 oprofile 的 opannotate 这个命令精度值得怀疑（oprofile 的手册也提醒过这个问题）。一般 opcontrol 和 opreport 两个命令结合就能满足基本需求。更多 oprofile 的使用说明可参阅 oprofile 的官方手册。龙芯版的 oprofile 可从 dev.lemote.com 的 apt 源中下载。

附录A　多媒体指令列表

punpcklbh/punpcklhw/punpcklwd

punpckhbh/punpckhhw/punpckhwd

packsshb/packsswh

packushb

pcmpgtb/pcmpgth/pcmpgtw

pcmpeqb/pcmpeqh/pcmpeqw

pmaxsh

pmaxub

pminsh

pminub

paddb/paddh/paddw/paddd

paddsb/paddsh

paddusb/paddush

psubb/psubh/psubw/psubd

psubsb/psubsh

psubusb/psubush

pmullh

pmulhh

pmuluw

pmulhuh

pmaddhw

pandn

psrlh/psrlw

psrah/psraw

psllh/psllw

pavgb/pavgh

Pshufh

pmovmskb

pextrh

pinsrh_0/pinsrh_1/pinsrh_2/pinsrh_3

pasubub

biadd

or

xor

nor

and

srl

dsrl

sra

dsra

sll

dsll

add

addu

dadd

sub

subu

dsub

seq

seq1

slt

sltu

sle

sleu

附录B　双单精度指令列表

ADD.ps

MADD.ps

MSUB.ps

NMADD.ps

NMSUB.ps

SUB.ps

NEG.ps

ABS.ps

C.F.ps

C.UN.ps

C.EQ.ps

C.UEQ.ps

C.OLT.ps

C.ULT.ps

C.OLE.ps

C.ULE.ps

C.SF.ps

C.NGLE.ps

C.SEQ.ps

C.NGL.ps

C.LT.ps

C.NGE.ps

C.LE.ps

C.NGT.ps

MUL.ps

MOV.ps