实验二 内核结构和汇编指令

王亭午

无210

2012011018

1. **实验目的**

**1.1通过观察内核寄存器读懂汇编指令、理解 DSP 结构**

**1.2学习汇编例程,掌握基本的计算和循环指令**

**1.3学习 DSP 开发中的程序设计,编写 FIR 滤波算法的 C 代码和非并行计算的汇编代码**

1. **实验材料**

集成开发环境 IDDE ( ADI 公司的 VisualDSP ++ 4.5 )。实验用源程序：汇编指令 core. asm ,见实验指导书附件。

点乘 C 源程序\Analog Device\VisualDSP\Blackfin\Examples\Tutorial\dot\_product\_c。点乘汇编源程序\Analog Device\VisualDSP 4.5\Blackfin\Examples\Tutorial\dot\_product\_asm。FIR 滤波例程\Analog Devices\VisualDSP 4.5\Blackfin\Examples\Tutorial\fir。阅读内容：课程课件《DSP内核结构与汇编指令》实验指导书及其附件《Blackfin 处理器内核基础知识》软件集成环境的帮助系统

1. **实验内容**

**3.1任务一：解读汇编指令**

运行core.asm测试程序,观察内核寄存器和相关内容的变化

**3.1.1实验步骤**

1. 找到源代码 core. asm。
2. 建立工程,导入源程序,编译测试。
3. 单步执行代码,观察各个内核寄存器的数据变化和汇编语言的对应关系。
4. 自己编写简单计算的汇编指令在 core.asm 中,添加一个程序段,完成下列用 Matlab 语句描述的计算,可以不用循环。

**答案**：这个的代码如下，可以用非常简单的方法来完成：

|  |
| --- |
| R4 = 0;  R0 = 11;  R1 = 1;  R5 = R1.l \* R0.l (is);  R0 = 12;  R1 = 2;  R6 = R1.l \* R0.l (is);  R0 = 6;  R1 = 3;  R7 = R1.l \* R0.l (is);    R4 = R5 + R7;  R4 = R4 + R6; |

在上面的代码中，R5,R6,R7分别是我们要求的C的三个值，R4是我们要求的D。逻辑很简单，就不细说了。

**3.1.2实验问题**

**问题 1.1 加法计算中NS和S的选项对计算结果有什么影响?**

**答案**：根据我们的课件可以知道，这个选项代表着不同的进位设置。S 代表 Saturation操作,需要考虑符号位，符号位不进位。NS则不考虑符号位，结果可以进位到最高。

**问题 1.2 在乘累加MAC计算中，IS选项意味着什么?**

**答案**：IS选项意味着当前乘法为整数乘法，不注明的时候默认是小数乘法，小数乘法的结果会自动左移一位（因为DSP中小数占一位，乘法后会多一位）。

**问题 1.3 在移位计算中，R4 = ROT R2 by −7的执行结果和理论计算的结果一样吗?**

**答案**：这个指令在课件中说的不清楚，所以用里面的来思考会出问题，但是只要知道这个指令的意思，就很好理解结果了。ROT指令会旋转7位，值得注意的是，这个指令会把CC位也旋转进来，而把多出来的一位放进CC位中。因此将 R2 写作二进制为0000 0000 0000 0000 0000 0101 0101 0000，将R2和CC（这个时候程序中CC是0）一起考虑，右移 7 位,理论结果为0100 0000 0000 0000 0000 0000 0000 1010 = 0xA000000A。和我们的预计一样。

**问题 1.4 对比指令R7 = EXTRACT(R2, R3. L)(Z)执行前后的结果,解释该指令含义?**

**答案**：这个指令看课件是不可能知道什么意思的。经过上网查询，结果如下：将R2从右至左第 8 位起，取出连续 4 bit，以零扩展的方式扩展并将结果存储在R7 寄存器中。起始位置和取出数量位置由R3的高低16位信息决定。最后的Z选项表示以零扩展的方式扩展。

**3.2任务二：学习计算点乘的 C 代码**

**3.2.1实验步骤**

1. 建立工程,导入 C 语言程序 dot\_product\_c , 编译加载执行程序。注意:如果出现错误，请按照第一次实验的经验自己调试并且解决。 C 代码的路径为\Analog Device\VisualDSP 4.5\Blackfin\Example\Tutorial\dot\_product\_c

**答案**：导入程序后，build发生错误，根据上面的提示将 main 函数第一行的itn i改为int i。

1. 单步执行程序,观察每一步执行完成之后的数组点乘结果的变化。
2. 重新单步运行程序,观察状态寄存器 ASTAT 的状态变化。菜单: Register->Core->Status->Arithmetic Status。记录 ASTAT 的变化情况并分析。
3. 重新全速运行程序,记录程序运行时间开销,用于和任务三的内容进行对比。

**3.2.2实验问题**

**问题 2.1有几种方法可以观察到计算结果?各有什么特点?**

**答案：**有以下几种方法：1. 在菜单栏中View→Debug Window→Expressions，然后在窗口输入变量名。这种方法便捷直观，可以一直保持监控。2. 直接观察结果变量所对应寄存器。这个基本很难做到。3. 我们也可以Dump memory到文件中。这个适合大量数据同时处理，不过比较麻烦。4.调用C语言直接将计算结果输出。这个最方便简单，不过需要修改代码。

**问题2.2在指令的运行中执行了什么操作,导致了哪些 ASTAT 比特位的状态变化?**

**答案：**各个事件的表格如下

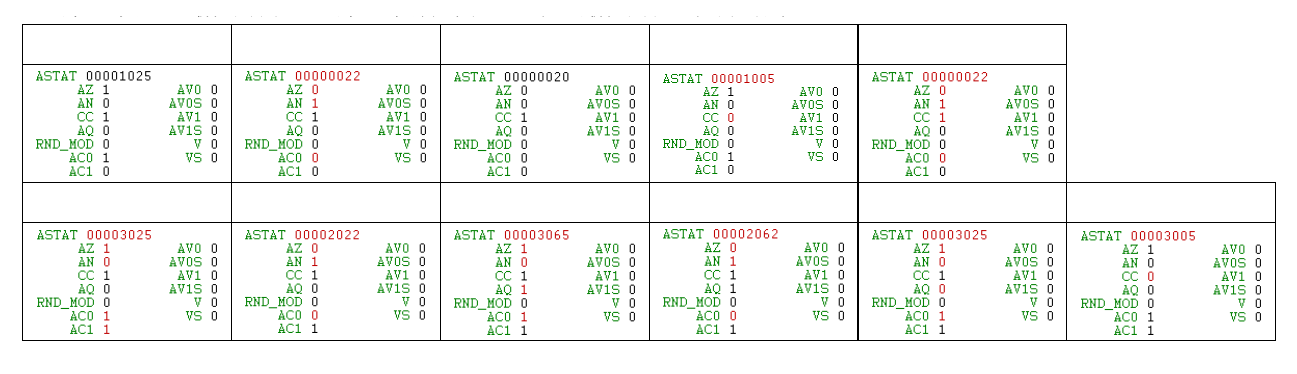


图1，ASTAT比特位状态变化

根据表格的顺序,每个图片的状态如下:

1.程序初始状态开始运行。2.调用一个函数，进入内部的循环。3.执行完循环内部的乘法。4.调用函数结束，出栈返回主函数。5.进入主函数循环体。6.调用C语言打印代码。7-10，重复三次打印循环。11.跳出循环体，主程序结束。

**3.3任务三：对比分析计算点乘的 C 和汇编代码**

对比实现点乘计算的C和汇编代码(dot\_product\_asm)的执行效率

**3.3.1实验步骤**

1. 建立工程, \Analog Device\VisualDSP 4.5\Blackfin\Example\Tutorial\ dot\_product\_asm下的源程序添加到工程中,编译执行程序。并且记录程序运行结果。
2. 重新单步运行程序,观察 ASTAT 的状态变化,并且和 C 语言的程序进行对比。
3. 重新单步运行程序,观察相关内核寄存器状态,主要集中在菜单 Register->Core下面。
4. 重新运行程序,记录执行效率

**3.3.2实验问题**

**问题 3.1 在这个实验中, accumulator 的值改变了么?什么时候改变的?**

**答案：**在我们的实验中，我们监视我们的accumulator变量，发现没有改变。

**问题 3.2 数据地址计算单元中的 DAG 寄存器发生变化了么?在什么情况下变化的? DAG 是怎样进行变址计算的?**

**答案：**DAG寄存器发生了变化**。**变化如下图：



图2，ASTAT比特位状态变化

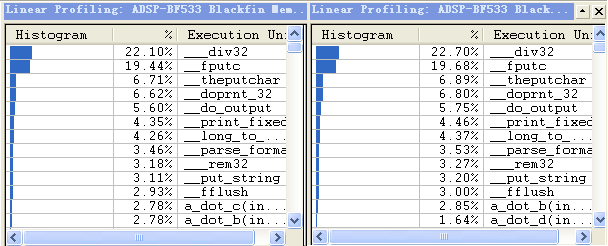
分别发生在我们的程序调用a\_dot\_c\_asm函数时；执行语句P1 = 19时；第一次进入循环体时；之后循环体技术增加I0 += 4时；退出循环体时。

**问题 3.3 有几组循环控制寄存器?在程序运行中是否有赋值?各是什么值?对应点乘运算的那几个数值?**

**答案：**我们有2组循环控制寄存器，他们在程序运行中有赋值。其中LC0是循环次数，而LT0和LT1对应运算时候的两个首地址。

**问题 3.4 在dot\_product\_c和dot\_product\_asm的执行效率对比中,你得到了什么结论和启发?**

**答案：**执行的效果图如下



运行时间分别为7s和8s，可以看得出，左图的会变代码运行速度最快。结论是：汇编代码速度快，但是编写复杂，c语言速度慢，但是编写方便。

**问题 3.5 对比 a\_dot\_c.c 编译后的汇编指令代码和 a\_dot\_c.asm 代码,观察分析a\_dot\_c.asm 效率高的原因。**

**答案：**编译得到的汇编指令代码当然无法直接和我们的汇编代码比速度，因为后者是经过通用的解释汇编器生成的，存在不最优的地方。在asm中人工生成的简洁的循环语句在C形成的汇编中可能会有很多复杂和多余的语句。

**3.4任务四：用 C 语言开发 FIR 滤波程序**

**3.4.1实验步骤**

1. 参考\Analog Device\VisualDSP 4.5\Blackfin\Example\Tutorial\fir 目录中fir\_test.c对fir.asm的调用方式,自己编写实现 FIR 滤波的C语言程序myfir.c。
2. 新建立一个工程,加载fir\_test.c和相关头文件，分别调用fir.asm和自己编写的myfir.c，调试运行程序,记录执行效率和滤波输出结果

**3.4.2实验问题**

**问题 4.1：通过 Tool->linear profiling 对比 myfir.c 和原始 fir.asm 的执行效率。**

**答案：**代码如下，我参考了课件上面的代码编写，运行的时间效率如图3。

|  |
| --- |
| **/\* Includes \*/**  #include "mds\_def.h"  #include "filter.h"  #include "fir\_coeff.h"  #include "fir\_input.h"  void my\_fir(const fract16\* IN, fract16\* OUT,int N,  fir\_state\_fr16\* s) {  **/\* the s contains**  **fract16 \*h; filter coefficients**  **fract16 \*d; start of delay line**  **int k; number of coefficients \*/**  **int i, j, temp, tapL;**  for (i = 0; i < N; i++) {  tapL = (s->k > i) ? (i + 1) : (s->k);  temp = 0;  for (j = 0; j < tapL; j++) {  temp += IN[i - j] \* (s->h)[j];  }  OUT[i] = (temp + 0x3FFF) >> 15;  }  return;  } |

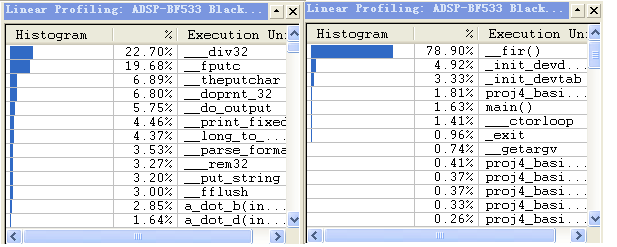


图3，汇编和C代码执行效率。

可以看出C语言的执行效率要比汇编语言的低很多，并且比之前使用到的C代码的效率还要低。

**问题 4.2：导出滤波结果数据,以 fir.asm 的输出为基准,分析 myfir.c 的计算误差,用误差曲线(波形)表示。说明你是如何控制计算精度的。**

**答案：**我们的代码如下：

|  |
| --- |
| fir\_init(s, h, delay, tapLength);    my\_fir(IN, MY\_OUT, nsamples, &s);  \_fir(IN, OUT, nsamples, &s);    for (i = 0; i < BUFFER\_SIZE; i++) {  DIFF[i] = OUT[i] - MY\_OUT[i];  } |

我们的结果如下，我们通过两个结果来作差，这样来计算精度：

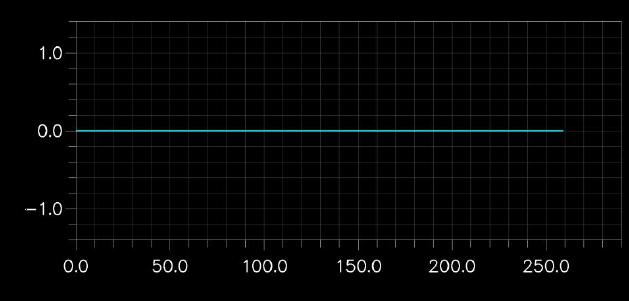


图5，输出误差结果图

可见两个的结果是一样的。

**3.5任务五：用汇编语言开发FIR滤波程序**

**3.5.1实验步骤**

1. 参考 \Analog Device\VisualDSP 4.5\Blackfin\Example\Tutorial\fir 目录中fir\_test. c对fir. asm 的调用方式, 自己编写实现FIR滤波的汇编语言程序fir\_Smac\_asm。
2. 新建立一个工程，加载fir\_test. c和相关头文件，调用自己编写的fir\_Smac\_asm. asm调试运行程序,记录执行效率和滤波输出结果。

**3.4.2实验问题**

**问题 5.1 通过 Tool → linear profiling 对比 fir\_Smac\_asm. asm 和原始 fir. asm 的执行效率**

**答案：**我们的代码如下：

|  |
| --- |
| LSETUP (lp\_output\_start, lp\_output\_end) LC0 = P0;  lp\_output\_start:  R1.H = W[I0++]; **// I0 is the input, read one input x into R0.L**  W[I3] = R1.H; **// I3 is the delay line**  A0 = 0;  LSETUP (lp\_coff\_start, lp\_coff\_end) LC1 = P3;  lp\_coff\_start:  R0.H = W[I2++];  R1.H = W[I3--];  lp\_coff\_end: A0 += R1.H \* R0.H; **// A0 is the sum of output[i]**  R0.L = A0;  W[I1++] = R0.L; **// write the result of output into I1**    lp\_output\_end: I3 += 2; |

在上面的代码中，lp\_output实现的是计算output的循环，lp\_coff实现的是计算每个output的时候的卷积循环。I0，I1，I2，I3分别是IN，OUT，滤波器Coefficient和delay line的地址。P0，P1分别是IN（也即OUT）和参数的长度。

在上面的一个非常重要的访问规则就是I2（对应滤波器参数访问）和I3（对应delay line上面的数据访问）的访问。在hardware loop中，它们两个的值在循环结束后，会恢复到原来的值,这是因为我们的参数是循环寻址的（见课件）。因此我们不需要在里面的循环中手动的恢复这两个寄存器。

因为我们的代码是8bit对齐的，因此在外层循环过后，我们需要设置I3 +=2，因为我们的delay line上的数据和我们的IN是一样的16bits。注意在W[I0++]中,自动是16bits对齐的。

然后和上一个任务一样，我们运行并且对比执行效率，结果如图6：

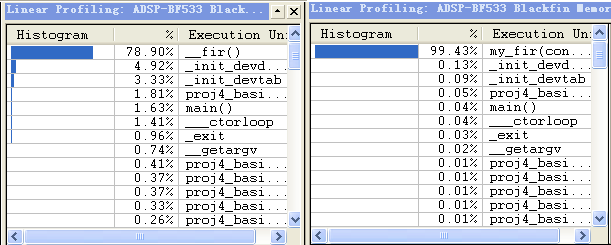


图6，两个汇编代码的效率对比

可以看到，两个汇编代码的效率差别还是很大的，系统自带的代码确实写的非常好，运用了大量的并行运算，冗余非常少。我们可以看出，汇编在时间节省上有很大的空间。

**问题 5.2 导出滤波结果数据，以fir. asm的输出为基准，分析fir\_Smac\_asm. asm的计算误差，用误差曲线(波形)表示。说明你采用的误差控制措施。**

**答案：**和上一个任务一样，我们采用类似的控制方法。结果如图：

图7，误差曲线图

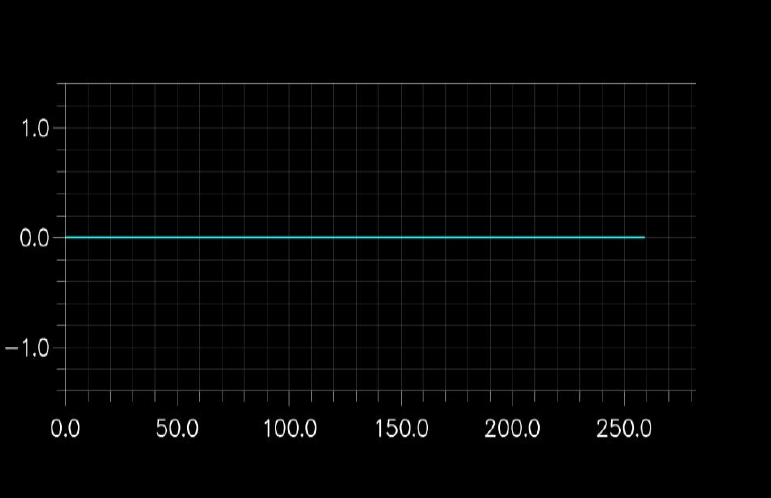
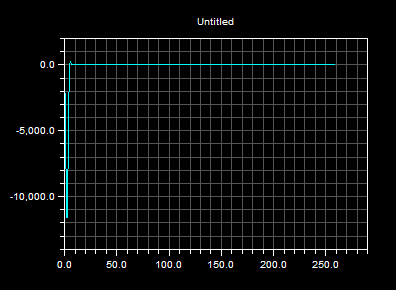


图8，修改后的误差曲线图

之所以在前几个会出现问题，是因为原来的汇编代码没有实现下面这几句代码：

|  |
| --- |
| tapL = (s->k > i) ? (i + 1) : (s->k);  temp = 0;  for (j = 0; j < tapL; j++) {  temp += IN[i - j] \* (s->h)[j];  } |

在计算前面几个系数的时候，我们实际上没有判断是否会出现我们对应的input的位置会不会已经是负数的情况。也就是说，我们的操作实际上访问了非法的地址。因此前几个会出现访问到我们的IN数组前面的地址的情况。当我们的output到达滤波器参数长度之后，这个时候我们的访问就是合法的了，因为我们的对应判断语句已经不会出现s->k > i的情况了。

所以我们有两种思路来解决这个问题：1. 在c代码中加入空白数据来填充前面的IN数据，这个时候前面的前k个结果虽然会错，但是我们在之后却知道这个数据是不安全的，去掉了他们。2.这个是系统的代码中的方法，就是在汇编语言中，在前面添加0。两个方法是等价的，不过考虑我们刚刚接触到了汇编代码，我认为第二种方法有点危险，因为我们在添加0的时候也访问了非法的地址。因此我们使用了第一种方法，在IN，OUT都添加滤波器参数长度个空数。得到的结果如图8，是正确的。

1. **总结问题**

**4.1为什么定点小数乘法结果需要左移一个比特位?**

**答案：**这是因为我们在进行小数乘法的时候，使用的是整数乘法的方法。但是我们需要考虑，这个时候我们实际上实在考虑小数。举个例子，两位小数的情况下我们计算0.12 × 0.15。我们使用12 × 15的方法来思考，得到的结果是180，于是在我们的直接结果上显示的是180，也就是1.80。这个时候我们实际上，少考虑了一个0.01的系数，所以我们需要将结果再移动两位，变成0.018（四舍五入0.02）。在DSP中，我们是1位小数，因此左移一位。

**4.2实验中遇到的问题和解决办法?**

**答案：**这一次的实验我个人觉得难度是非常高的。出现了以下几个问题。

1. 部分指令完全不知道什么意思。比如我们的EXTRACT指令，在课件里面没有，如果只是看我们手上的材料，那是不可能可以猜出来什么意思的。不少同学，包括我在内，都是很难会把R2分成高位和低位两个部分来考虑的，更不要说，我们很多同学一直都是用16进制在看的。16进制下，完全看不出任何所以然来。因此，这个问题，我是通过网络搜索来解决的。
2. 汇编难度太大。我们之前学的汇编和现在的汇编完全不是一个体系，可以说，我们是重新学一个新的汇编。同时我们这一回学的汇编，学的太浅太快，很多命令都是一张ppt不带任何例子就讲了7,8个，而且很多命令都没有学。比如，我们到现在都不知道在汇编中，如何实现一个判断的功能（我们之前的MIPS中，判断功能是很简单的）。因此在最后一个任务中，我们无法根据C代码来完成对应的汇编代码，因为那一个判断输出位置和滤波器参数长度的语句我们不知道该怎么实现。所以在实验中，我使用了在C代码中填充冗余数组的方法来完成。

此外，很多代码的效果，也是第一次接触。比如W[I0 ++]这一句代码，实际上I0移动了16个bit。这一点需要真正写，才会注意的到。

**4.3实验体会和建议?**

**答案：**实验体会就是，这次的实验难度很大。有些问题助教也不太清楚，整体来说整个实验都在不断的尝试和碰壁中度过。尤其是到最后都没有找到这个汇编的判断语句实现方法，觉得非常的遗憾。个人觉得老师应该在这个汇编上花更加多的时间来授课，一节课实在是有点短。同时也希望可以拿到关于DSP上汇编的推荐教材。