实验时间：2014年10月16日 实验地点：主楼916机房

实验二 内核结构和汇编指令

院系 电子工程系

班级 无13班

姓名 蔡杨

学号 2011011040

**实验报告**

**【实验目的】**

1. 通过观察内核寄存器读懂汇编指令、理解结构

2. 学习汇编例程，掌握基本的计算和循环指令

3. 学习开发中的程序设计，编写滤波算法的代码和非并行计算的汇编代码

**【实验材料】**

1. 集成开发环境（公司的）

2. 实验用源程序

*a*) 汇编指令，见实验指导书附件

*b*) 点乘源程序

*c*) 点乘汇编源程序

d) 滤波例程

3. 阅读内容

*a*) 课程课件《内核结构与汇编指令》

*b*) 实验指导书及其附件——《处理器内核基础知识》

*c*) 软件集成环境的帮助系统

**【实验内容】**

**任务一 解读汇编指令**

运行测试程序，观察内核寄存器和相关内容的变化

**实验步骤**

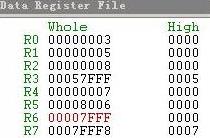
(1) 找到源代码

(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

[问题 1.1] 加法计算中“”和“”的选项对计算结果有什么影响？

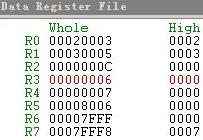
答：实验结果如下图所示



代表，即饱和。表示对加法进行不饱和处理，即不考虑符号位，结果可以进位到第位，即结果可以超过；而则代表对加法进行饱和处理，即考虑符号位，结果不可以进位到第位，即结果不可以超过。

[问题 1.2] 在乘累加（）计算中，“”选项意味着什么？

答：在乘累加（）计算中，“”选项意味着当前乘法为整数乘法，如果不指明选项则将默认执行小数乘法，小数乘法的结果会自动左移一位。

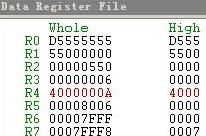


[问题 1.3] 在移位计算中，“”的执行结果和理论计算的结果一样吗？

答：执行结果如下图所示，与理论计算的结果不一致。

将写作二进制为

表达式表示将右移位，理论结果为

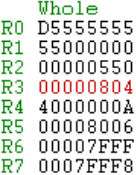


而实际结果则为

这是由于执行过程中会在寄存器中存放，因此相当于结尾处多出一位（执行时默认为），而原来右数第位的还未移入结果中，因此发生了与理论结果不一致的问题。

[问题1.4] 对比指令“”执行前后的结果，解释该指令含义？

答：执行前后各寄存器的值如下图所示

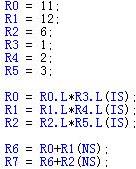
 

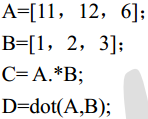
由此我们可以知道

这个指令代表从右至左第位起，取出连续，以零扩展的方式扩展并将结果存储在寄存器中。的信息决定了从中截取的位置和长度。

[问题 1.5] 在中，添加一段程序，完成下列用语句描述的计算，不用循环。

答：所添加的程序如下所示





**任务二 学习计算点乘的代码**

**实验步骤**

(1) 建立工程，导入语言程序, 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。代码的路径为

答：导入程序后，编译发生错误。将函数第一行的改为之后，编译成功。

(2) 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。

(3) 重新单步运行程序，观察状态寄存器的状态变化。

*a*) 菜单

*b*) 记录的变化情况并分析。

(4) 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。

答：使用进行运行时间开销统计，结果为

[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：共有四种方法可以观察到计算结果。

(1) 在菜单栏中点击，输入变量名即可查看其数值。这种方法便捷直观；

(2) 直接观察结果变量所对应寄存器。这种方法不够直观，遇到指针数据将很难追踪；

(3) 利用输出到文件中，再利用其他软件分析查看。 这种方法可以将大量数据同时处理（如数组），还可将其导入其他软件进行更详细的分析；

(4) 直接将计算结果输出。这种方法会影响程序的运行速度，也不便于实时追踪。

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

答：各个事件描述如下

程序运行的初始状态； 进入函数内部的循环； 执行完循环内部的乘法； 函数结束后返回主函数； 进入循环体； 打印； 再次进入循环体； 第二次打印； 第三次进入循环体； 第三次打印； 跳出循环体，程序结束

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

是操作标识。一些可以从观察中推断出来的操作标识列举如下

初始状态——00001025；循环满足循环——00000022；跳出循环——00001005；打印结果——00002062。此外，声明、赋值、加法等操作也会导致比特位的状态变化。

**任务三 对比分析计算点乘的和汇编代码**

对比实现点乘计算的和汇编代码（）的执行效率

**实验步骤**

(1) 建立工程，将

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。

[注意] 在建立新的工程时，需要选择“Add an LDF and startup code”。 此时如果简单导入下的源程序而不添加文件， 会出现错误，是编译器错误吗？下面是错误原因分析和解决建议：

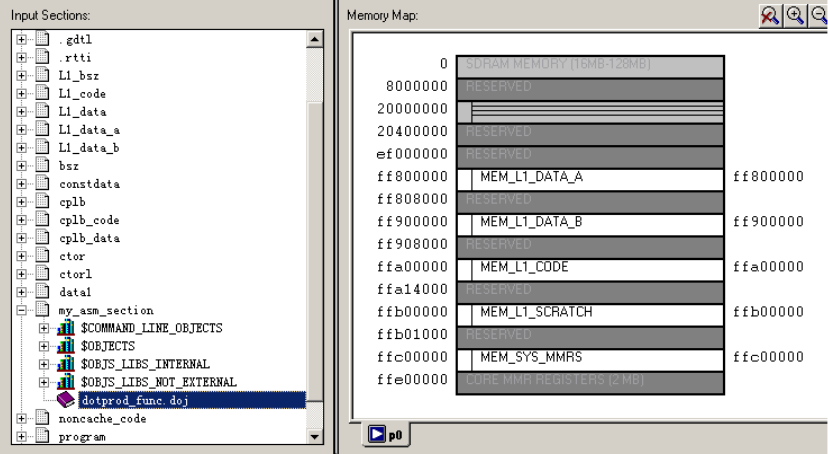
[原因] 主要是由于链接器不知道汇编语言形成的函数体在什么位置，并且相关的代码段没有被指定到可执行的内存区。

[解决] 添加文件。右击工程名，弹出菜单选择，在列表中选择

然后选择

之后系统会自动添加一些启动文件和一个文件。

[可能的新错误] 重新编译，依然会出现错误！主要是没有把函数编译的对象手动映射到相关的区段。双击文件，在弹出的窗口中寻找, 会发现有一个红叉，表示此段对象依然没有被映射。此时要手动将相关的函数对象映射到 (拖拽 )到一个空闲的内存段。 将打红叉的拖动到，之后发现红叉会消失。 保存文件并重新编译工程，此时将会成功。



(2) 重新单步运行程序，观察的状态变化，并且和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，主要集中在菜单下面。

(4) 重新运行程序，记录执行效率

[重点观察] 和循环控制寄存器。

[问题 3.1] 在这个实验中，的值改变了么？什么时候改变的？

答：这个实验中，的值没有发生改变，始终为。

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

答：(1) 数据地址计算单元中的寄存器发生了变化。

(2) 刚刚进入函数内部时



执行语句后



第一次进入循环体时



以后每次进入循环体



最后循环体结束时



(3) 在汇编代码下，被赋值为的首地址，每一次循环都执行一次操作。每一次进入循环时对索引值进行保存，实现循环的缓冲。

[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：(1) 有组循环控制寄存器

(2) 在程序运行中有赋值

(3) 对应点乘运算的循环次数，和对应两个首地址。

[问题 3.4] 在“”和“”的执行效率对比中，你得到了什么结论和启发？

答：仍然使用进行运行时间开销统计，结果为

也可以使用进行观察

|  |  |
| --- | --- |
|  |  |
|  |  |

与代码的开销相比，汇编程序的效率更高，运行速度更快。

[问题 3.5] 对比编译后汇编指令代码和代码，观察分析的效率更高的原因。

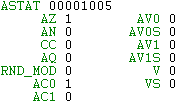
答：编译后得到的汇编指令代码是由解释生成的，采用汇编器的默认实现，存在许多冗余代码。而是经过人工优化得到的，比解释生成的代码短了很多。

例如，在中循环是用系统的语句实现，而解释语句则用汇编器默认实现。相比之下，的效率要提高很多。

[问题 3.6] 观察的状态变化

答：由于其他部分代码和代码完全相同，因此我们仅仅考察内部。

初始状态下



执行语句时出现过的值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

三个状态值发生了变化。这些变化分别表示了在函数调用中和没有声明为；没有进位；而时，表示循环不满足条件，退出循环。

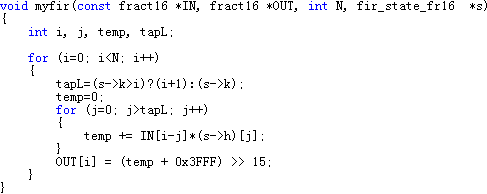
**任务四 用语言开发滤波程序**

**实验步骤**

(1) 参考目录中“”

对“”的调用方式，自己编写实现滤波的语言程序。

答：书写的代码如下



(2) 新建立一个工程，加载“”和相关头文件，分别调用和自己编写的“”，调试运行程序，记录执行效率和滤波输出结果。

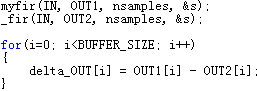
[问题 4.1] 通过对比和原始的执行效率

答：如下图所示

|  |  |
| --- | --- |
|  |  |
|  |  |

可见语言文件的执行效率要比汇编语言的文件低很多，甚至比之前使用到的语言的运行效率还要低。

[问题 4.2] 导出滤波结果数据，以的输出为基准，分析的计算误差，用误差曲线（波形）表示。说明你是如何控制计算精度的。



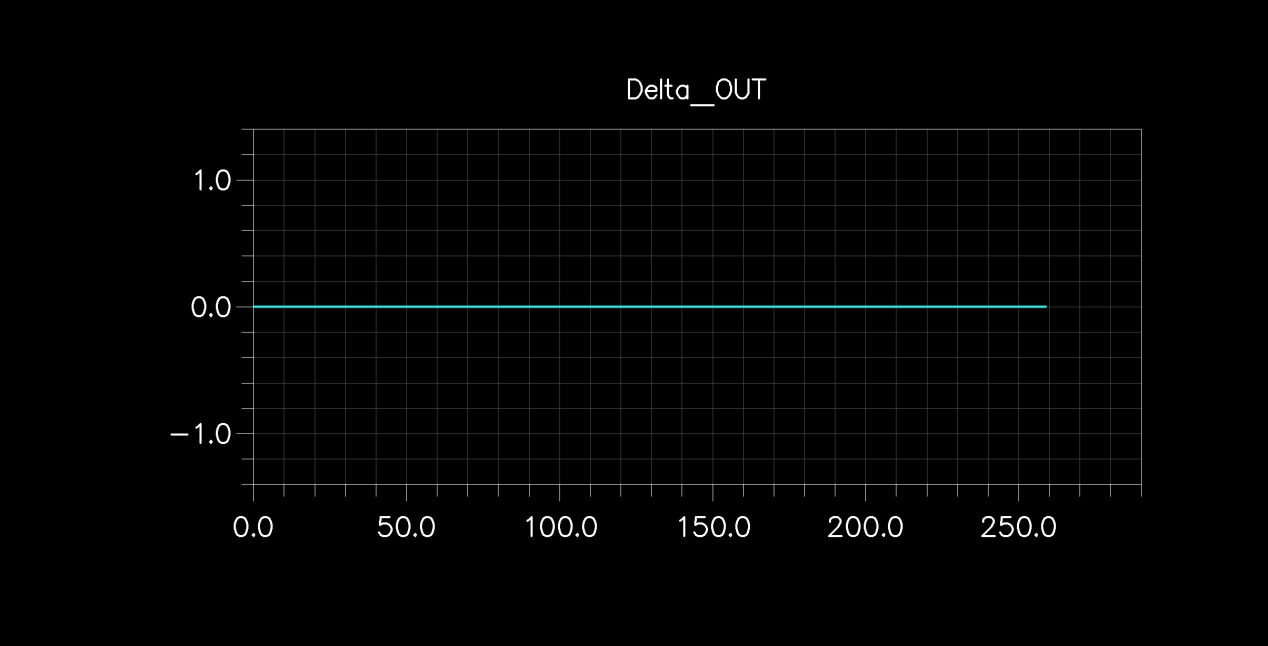
答：分别在主程序中调用语言和汇编语言函数，

对两者求差来显示误差。

首先观察两个函数各自的输出结果

|  |  |
| --- | --- |
|  |  |
|  |  |

进一步通过作差的方法验证两者的波形完全相同



因此误差曲线恒为零，即利用进行计算时没有误差。

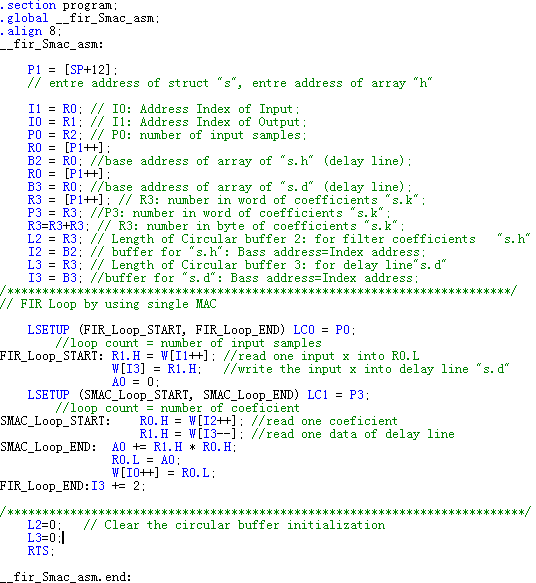
差错控制的方法采用的是课堂上讲授的滤波器设计方法。

**任务五 用汇编语言开发滤波程序**

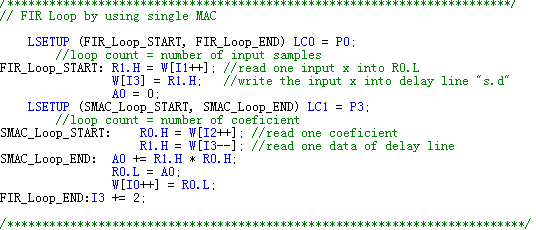
**实验步骤**

(1) 参考目录中“”对“”的调用方式， 自己编写实现滤波的汇编语言程序。

答：自编的文件内容如下



关键部分代码



(2) 新建立一个工程，加载“”和相关头文件，调用自己编写的“”，调试运行程序，记录执行效率和滤波输出结果。

[问题 5.1] 通过对比和原始的执行效率

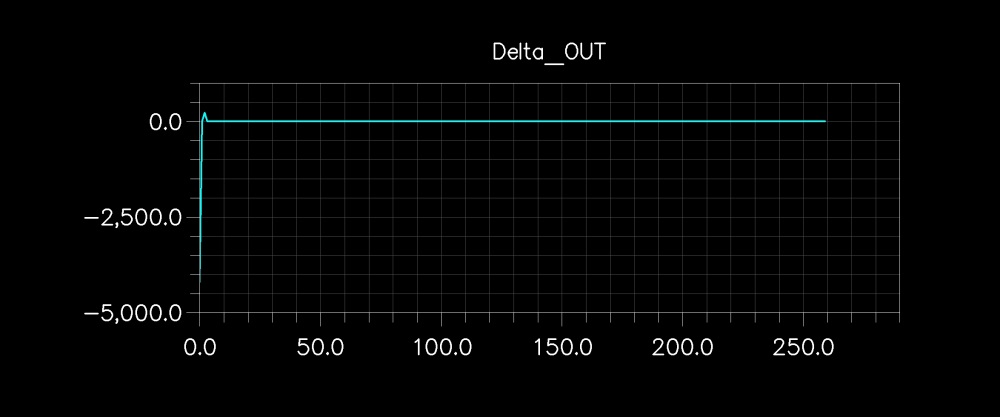
答：如下图所示

|  |  |
| --- | --- |
|  |  |
|  |  |

由于自编的文件采用的是串行处理，因此其执行效率仍然低于原来的文件。

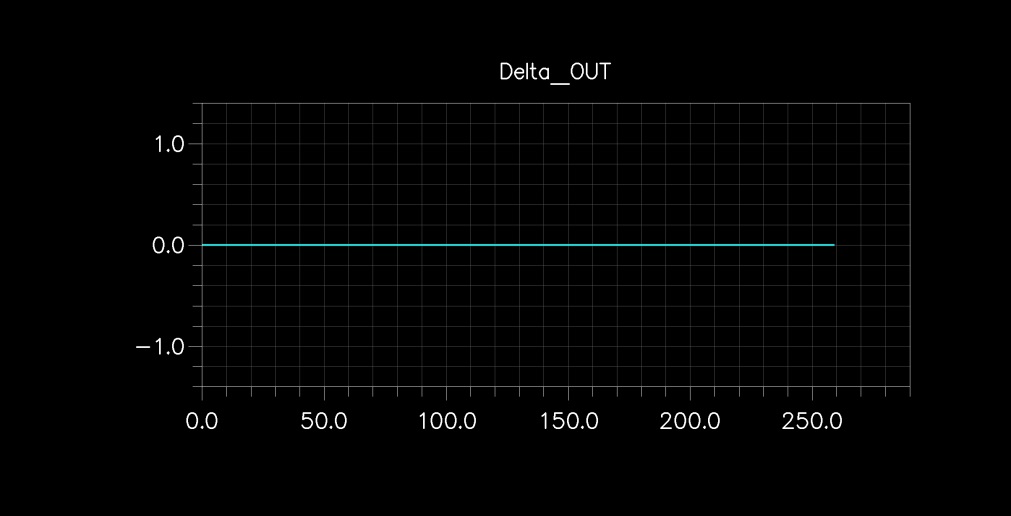
[问题 5.2] 导出滤波结果数据，以的输出为基准，分析的计算误差，用误差曲线（波形）表示。说明你采用的误差控制措施。

答：完全类似于 [问题4.2] 中的措施，我们得到了误差曲线如下图所示



这个结果说明，自编的汇编语言并不能完全替代原来的汇编语言文件，问题集中在前几个计算的结果中。分析其原因，不难发现，初始状态下延迟线中的数据并不为零。然而我们则认为其中的数据为零，因此在前面的几位会出现一定的误差。一旦参与运算的数据个数达到了滤波器的长度，计算的结果就正确了。因此出现之前几位非零的情形。

解决方案为在进行计算之前，首先把延迟线中的系数清零一下，而后再进行计算。采用这个方法进行计算以后，发现误差曲线严格为零，此时的汇编文件间才是可替代的。



**【实验小结】**

(1) 为什么定点小数乘法结果需要左移一个比特位？

答：对于两个有符号小数的乘积而言，乘积的结果中包含了两位符号位。小数点的真实位置不再位于结果的第一位以后，而是位于第二位以后。为了统一这个表示，我们需要将结果左移一位，这样就得到了仅有一位符号位的计算结果。

(2) 实验中遇到的问题和解决办法

答：今天的实验花费了我很长的时间，周围的同学也都反映实验内容比较难。虽然上机以前经过了充分的复习，但还是在任务一就出现了问题。主要的原因在于对于汇编语言、十六进制的表示不够熟悉，没能很快发现不同的结果的区别。但在完成任务一以后，我对这部分的内容有了更加深刻的认识，同时也信心大增，为后面的实验步骤打下良好的基础。

另一个问题发生在任务四的代码书写中。实际上，由于课件上包含了这部分的内容，但我在完成任务时，还是有点不知所以，照着课件上的代码内容照搬照套。导致在助教验收时竟说不出其中的代码的含义。我主动要求独立思考以后再找助教验收，通过复习课件、上课笔记、请教周围同学等途径终于弄明白了代码的含义，顺利完成了验收。

最后谈谈任务五。任务五中如果直接调用函数，会发现前面几位的结果并不正确，原因已经在之前分析过了。弄明白原因以后，我想到了相应的解决方案，就是在计算以前需要对延迟线中的系数进行清零。因此我们在执行完第一次滤波以后需要再次调用函数进行初始化，这样才能得到正确的结果。我尝试了自己的这种解决方案，终于得到了和预期结果相同的结果。

(3) 实验体会和建议

答：今天是第二次的系列实验。与第一次的实验感觉差别很大，因为上一次实验完全只是根据实验指导书进行操作的过程，而这一次实验则需要独立思考、发现并解决问题。由于软件使用的基础不够扎实，在一些问题上还是浪费了比较多的时间。然而，在询问助教、与周围同学进行讨论的过程中，整个实验还是进行得比较顺利的。

在整个实验过程中，我认为任务五的难度最大，也最耗费时间。归根结底，是因为我的汇编语言的编程能力依然有待提高。虽然在十分熟悉的语言环境下能够很好地缓解压力，但我深知如果希望成为一名熟练的编程者，依然需要大量的练习。

虽然今天的实验内容比较艰辛，甚至有一些同学花费了将近个小时。但这并不能冲淡我对这门课的热情。我发现这门实验课的确十分实用，熟练掌握以后将会大有裨益。我会在不断学习新知识的过程中加深对已学过的知识的理解，学好这门课。

最后，诚挚感谢老师和助教在实验过程中予以我们小组的大力支持！

2014年10月20日