电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2018091602007

姓 名 王乐卿

（实验）课程名称《ARM处理器体系结构及

应用》课程实验

理论教师 兰刚

实验教师 兰刚

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：王乐卿 学号：2018091602007 指导教师：兰刚**

**实验地点：家 实验时间：2020.5.26**

1. 实验名称：数字滤波
2. 实验学时：4
3. 实验目的：
4. 通过工程中常用的数字滤波编程实现，验培养学生分析问题、解决问题能力。
5. 通过该编程，进一步巩固和强化学生ARM汇编编程的能。
6. 实验原理：

**（1）中值滤波**

数字图像在其形成、传输记录的过程中往往会受到很多噪声的的污染，比如：椒盐噪声、高斯噪声等，为了抑制和消除这些随即产生的噪声而改善图像的质量，就需要去、对图像进行去滤波噪处理。

中值滤波是图像平滑的一种方法 它是一种非线性平滑滤波技术，在一定条件下可以克服线性滤波带来的图像细节的模糊问题，特别是针对被椒盐噪声污染的图像。中值滤波对脉冲噪声有良好的滤除作用，特别是在滤除噪声的同时，能够保护信号的边缘，使之不被模糊。这些优良特性是线性滤波方法所不具有的。此外，中值滤波的算法比较简单，也易于用硬件实现。所以，中值滤波方法一经提出后，便在数字信号处理领得到重要的应用。

中值滤波方法：对一个数字信号序列xj(-∞<j<∞)进行滤波处理时，首先要定义一个长度为奇数的L长窗口，L=2N+1，N为正整数。设在某一个时刻，窗口内的信号样本为x(i-N)，…，x(i)，…，x(i+N)，其中x(i)为位于窗口中心的信号样本值。对这L个信号样本值按从小到大的顺序排列后，其中值，在i处的样值，便定义为中值滤波的输出值。

**（2）均值滤波**

均值滤波也用于图像处理，均值滤波也称为线性滤波，其采用的主要方法为邻域平均法。线性滤波的基本原理是用均值代替原图像中的各个像素值，即对待处理的当前像素点（x，y），选择一个模板，该模板由其近邻的若干像素组成，求模板中所有像素的均值，再把该均值赋予当前像素点（x，y），作为处理后图像在该点上的灰度g（x，y），即g（x，y）=∑f（x，y）/m m为该模板中包含当前像素在内的像素总个数。

本事实验要求，将 N 个无符号数进行排序，去掉最大值和最小值，剩余的数求平均值，平均值即为本次滤波所得的结果。

**（3）中值滤波程序流程图**

中值滤波markdown代码：

|  |
| --- |
| ```flow  st=>start: 开始  oper1=>operation: 初始化：R0=N，排序数的个数  R2为原始数据的起始地址  R3为原始数据的结束地址  R4为原始数据倒数第2个地址  oper2=>operation: R5=R2，内循环的开始地址  oper3=>operation: 1、R5和R5+4进行比较，若R5>R5+4大于就交换，排序后为从小到大  2、修改R5=R5+4  oper4=>operation: R3=R3-4，此时R3已经保存最大值，不再进入循环比较  oper5=>operation: 找到中间那个数  oper6=>operation: R1=中间那个数  cond1=>condition: R5>=R4？  cond2=>condition: R3<=R2？  en=>end: 结束  st->oper1->oper2->oper3->cond1(yes)->oper4->cond2(yes)->oper5->oper6->en  cond1(no)->oper2  cond2(no)->oper3  ``` |

中值滤波的流程如图1所示：

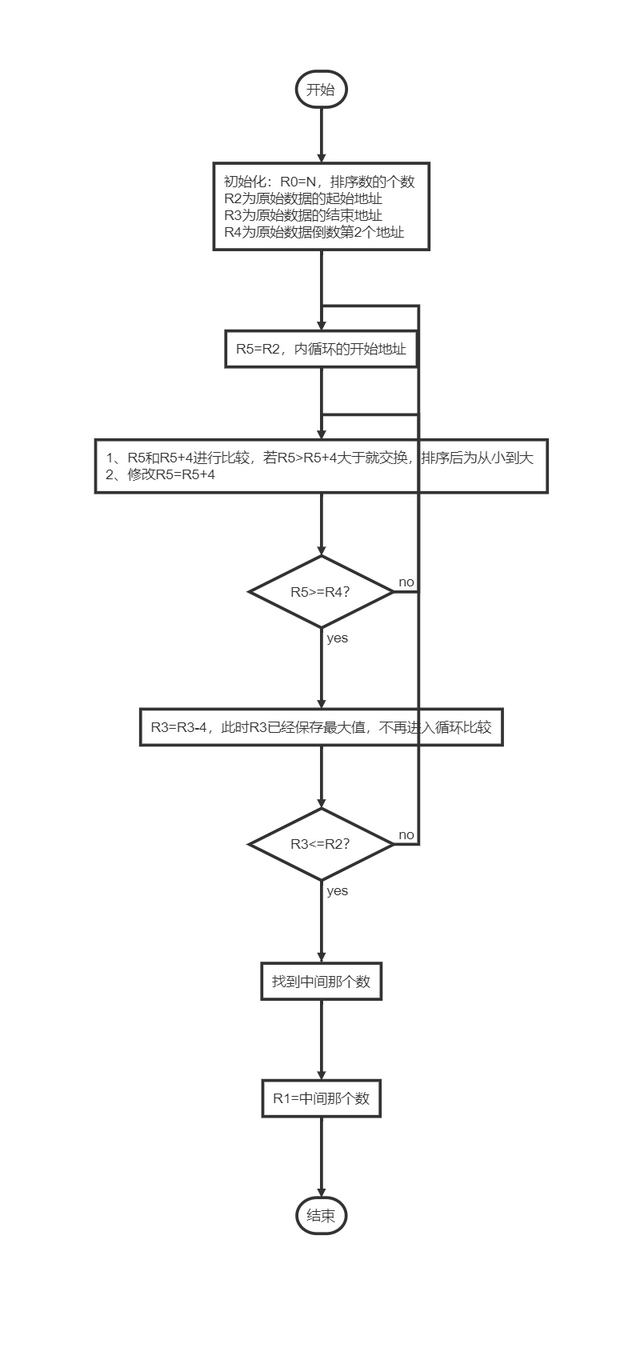


图1 中值滤波流程图

**（4）均值滤波程序流程图**

均值滤波流程图markdown代码：

|  |
| --- |
| ```flow  st=>start: 开始  oper1=>operation: 初始化：R0=N，排序数的个数  R2为原始数据的起始地址  R3为原始数据的结束地址  R4为原始数据倒数第2个地址  oper2=>operation: R5=R2，内循环的开始地址  oper3=>operation: 1、R5和R5+4进行比较，若R5>R5+4大于就交换，排序后为从小到大  2、修改R5=R5+4  oper4=>operation: R3=R3-4，此时R3已经保存最大值，不再进入循环比较  oper5=>operation: 去除最大值和最小值后剩余的数求平均  oper6=>operation: R1=平均值  cond1=>condition: R5>=R4？  cond2=>condition: R3<=R2？  en=>end: 结束  st->oper1->oper2->oper3->cond1(yes)->oper4->cond2(yes)->oper5->oper6->en  cond1(no)->oper2  cond2(no)->oper3  ``` |

均值滤波流程图如图2所示：

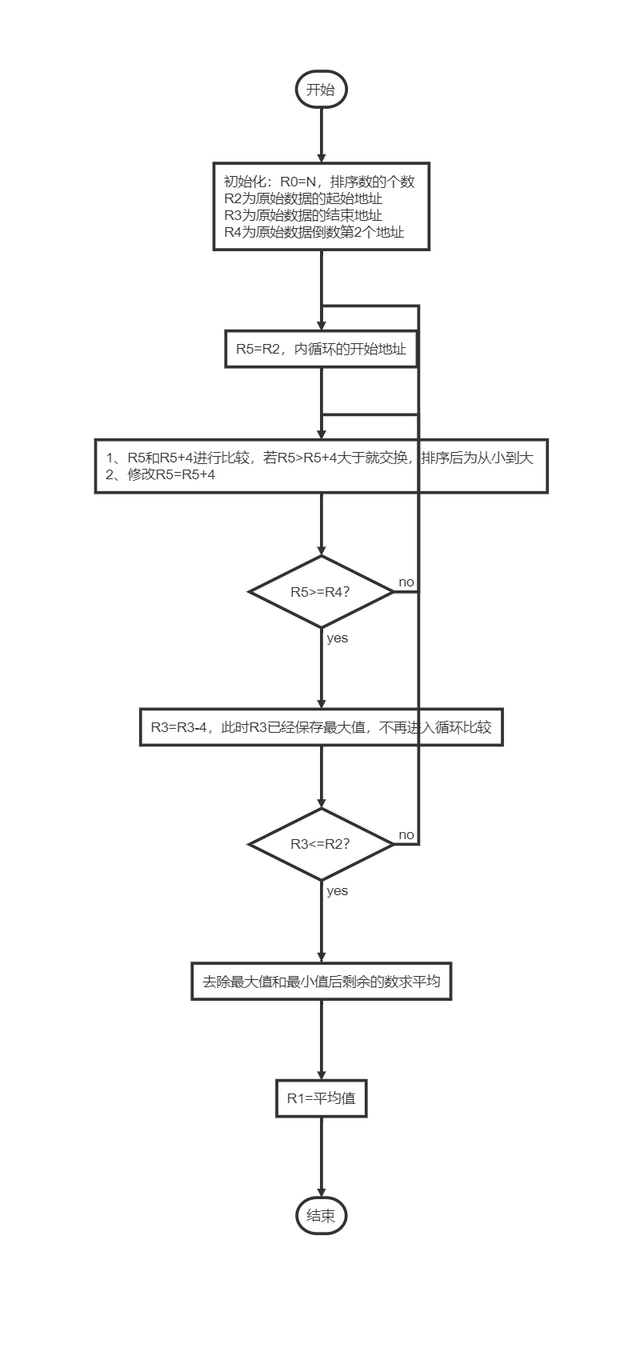


图2 均值滤波流程图

1. 实验内容：
2. 中值滤波及编程实验，要求如下：

* 关于 N（N为奇数）个数的值在程序中能任意、方便设置，并且放在 R0 中；
* 原始数据放在内存 0X40000000 开始的地址空间；
* 中值滤波的结果放在寄存器 R1 中；

1. 均值滤波编程实验，要求如下：

* 关于 N（N为偶数）个数的值在程序中能任意、方便设置，如 4、6、10、18 等，该值放在 R0 中；
* 原始数据放在内存中 0X40000000 开始的地址空间；
* 均值滤波的结果放在寄存器 R1 中；

1. 实验器材（设备、元器件）：
2. PC机一台；
3. Keil MDK-ARM uVision4开发工具。
4. 实验步骤：
5. 打开Keil MDK-ARM uVision4开发工具；
6. 新建一个工程文件；
7. 在新建的工程文件中，添加新的源程序文件
8. 编写代码
9. 选择“Build target”菜单对编写好的工程文件进行编译链接。
10. 点击““Start/Stop Debug Section””按键，对程序进行跟踪调试，在调试界面，单步执行，对CPU各寄存器的值的变化、以及相关内存的变化进行分析比较，判断程序的执行是否符合预期要求。
11. 实验结果与分析（含重要数据结果分析或核心代码流程分析）
12. **中值滤波**
13. **实验代码**

代码1中值滤波程序代码

|  |
| --- |
| ;本次使用冒泡算法！  AREA SORT,CODE,READONLY ;定义一个代码段  ENTRY  MOV R0,#9 ;排序数的个数，为奇数  LDR R2,=0x40000000 ;R2为原始数据的起始地址  SUB R1,R0,#1  MOV R4,#4  MLA R3,R1,R4,R2 ;R3为原始数据的结束地址 R3=R1\*R4+R2  SUB R4,R3,#4 ;R4为原始数据的倒数第二个数的地址  LOOP1 MOV R5,R2 ;重置R5=R2  LOOP2 LDR R6,[R5]  LDR R7,[R5,#4]  CMP R6,R7 ;比较交换，从小到大排序  STRHI R6,[R5,#4] ;将R6的值送入R5+4的地址中，如果R6>R7  STRHI R7,[R5]  ADD R5,R5,#4 ;修改内循环地址  CMP R5,R4 ;内循环结束比较  BLS LOOP2 ;一轮循环结束后，最大值已被冒泡至结束地址R3  SUB R3,R3,#4 ;修改外循环结束地址  SUB R4,R4,#4  CMP R2,R3 ;外循环结束比较  BLS LOOP1  LDR R2,=0x40000000  MOV R0,R0,LSR #1  MOV R4,#4  MLA R3,R0,R4,R2  LDR R1,[R3] ;找到中间那个数并赋给R1  MOV R0,#100  END |

1. **运行过程及结果界面截图**

**第一组数据：**

下图3是程序运行刚开始的界面。R0设置为7，总共有7个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x05、0x01、0x14、0x09、0x23、0x20、0x17。

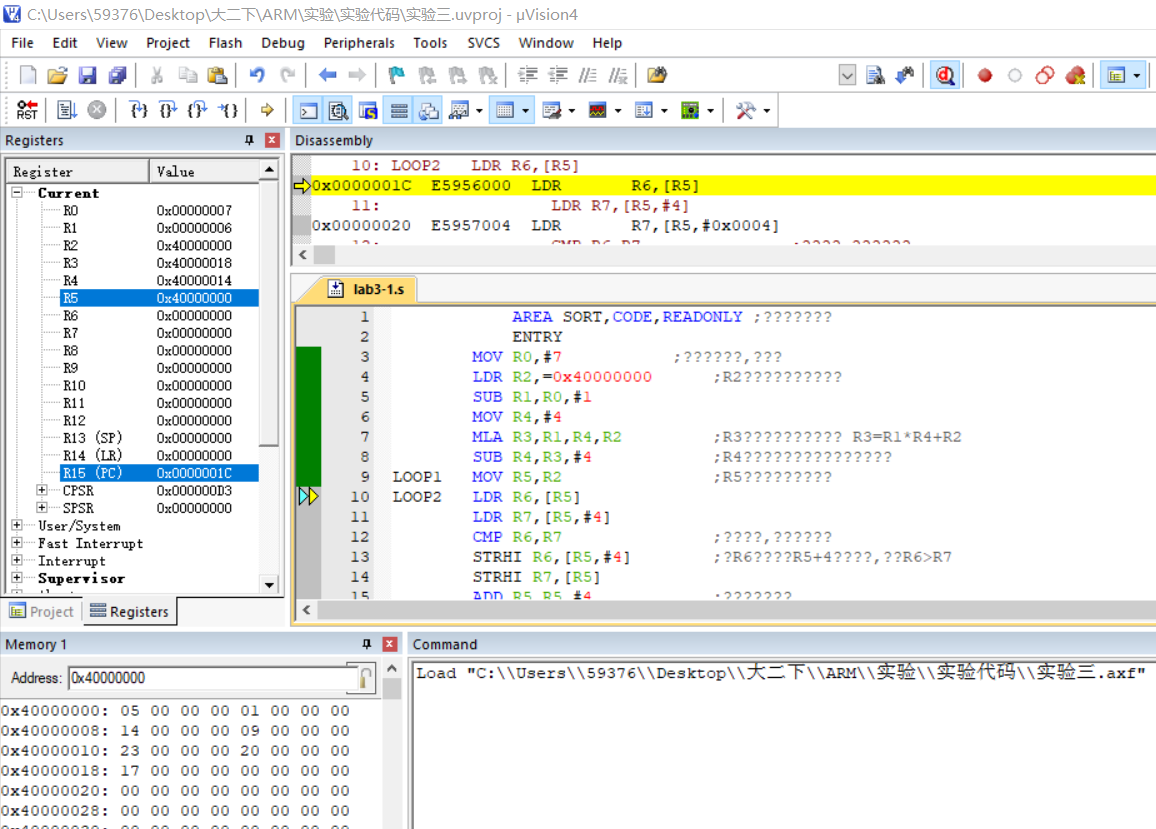


图3 数据一中值滤波程序刚开始运行时的界面截图

下图4是程序运行结束时的界面，可以看出从0x40000000开始的7个存储单元被排序了，运行结果存放于R1中，结果为0x14。

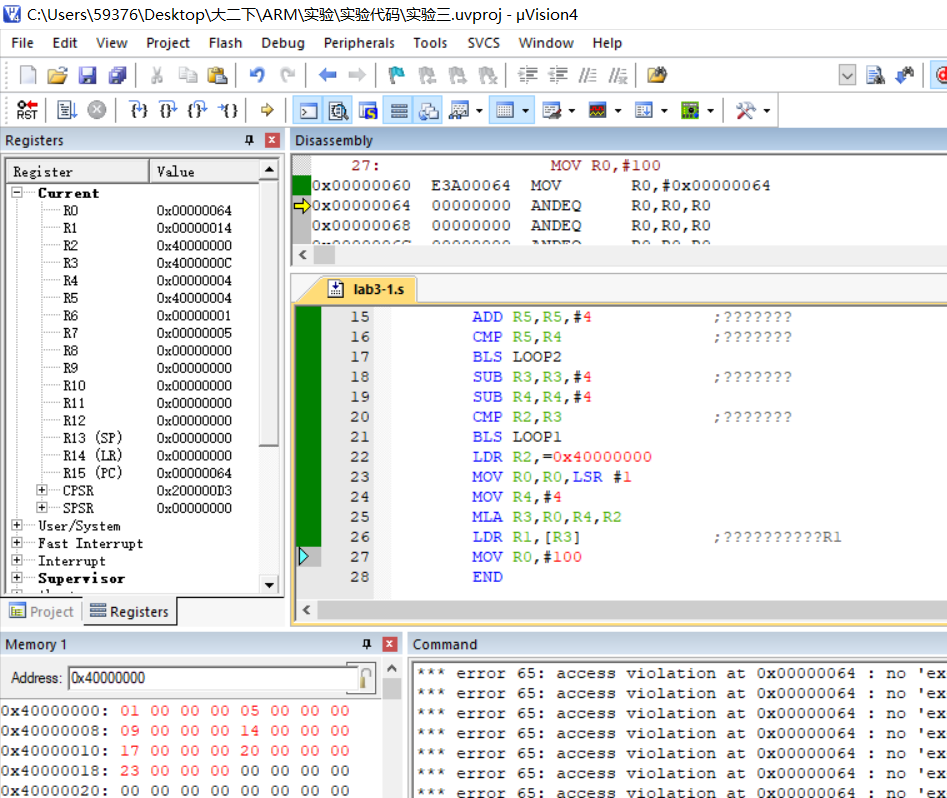


图4 数据一中值滤波程序运行结束时的界面截图

**第二组数据：**

下图5是程序运行刚开始的界面。R0设置为7，总共有7个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x60、0x69、0x30、0x79、0x20、0x32、0x65。

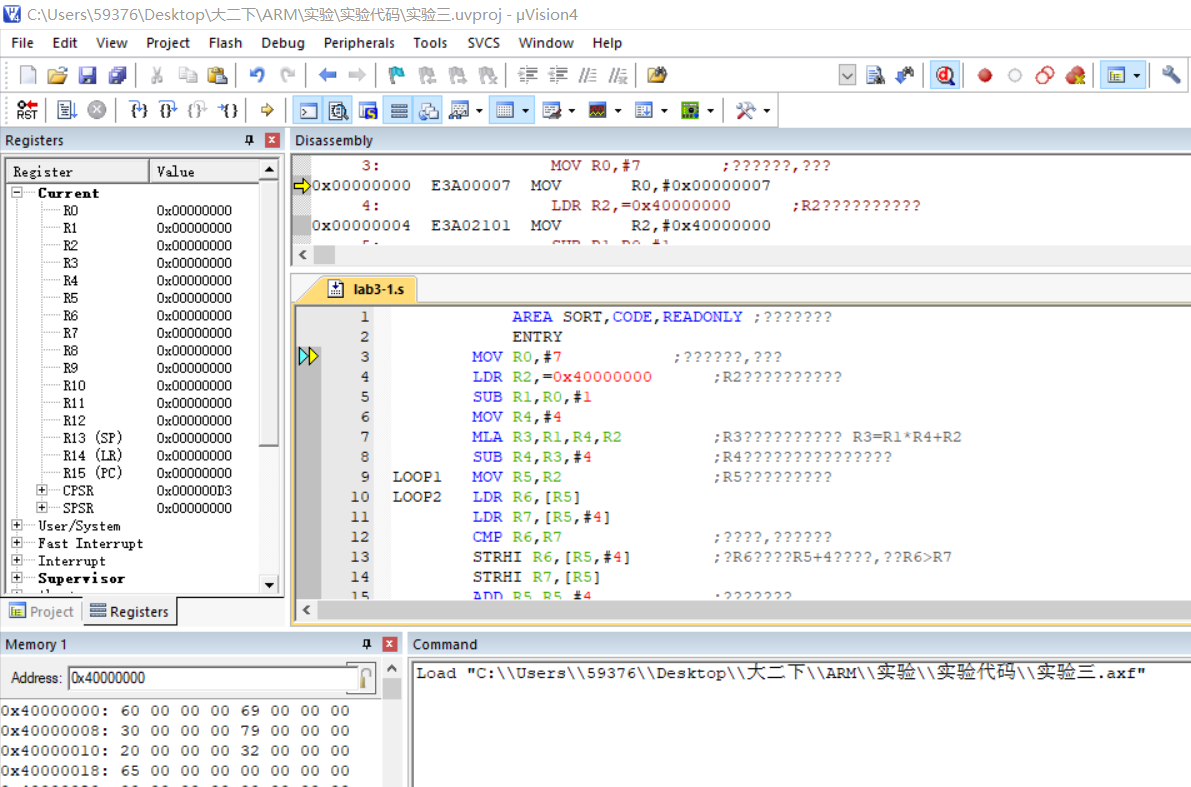


图5 数据二中值滤波程序刚开始运行时的界面截图

下图6是程序运行结束时的界面，可以看出从0x40000000开始的7个存储单元被排序了，运行结果存放于R1中，结果为0x60。

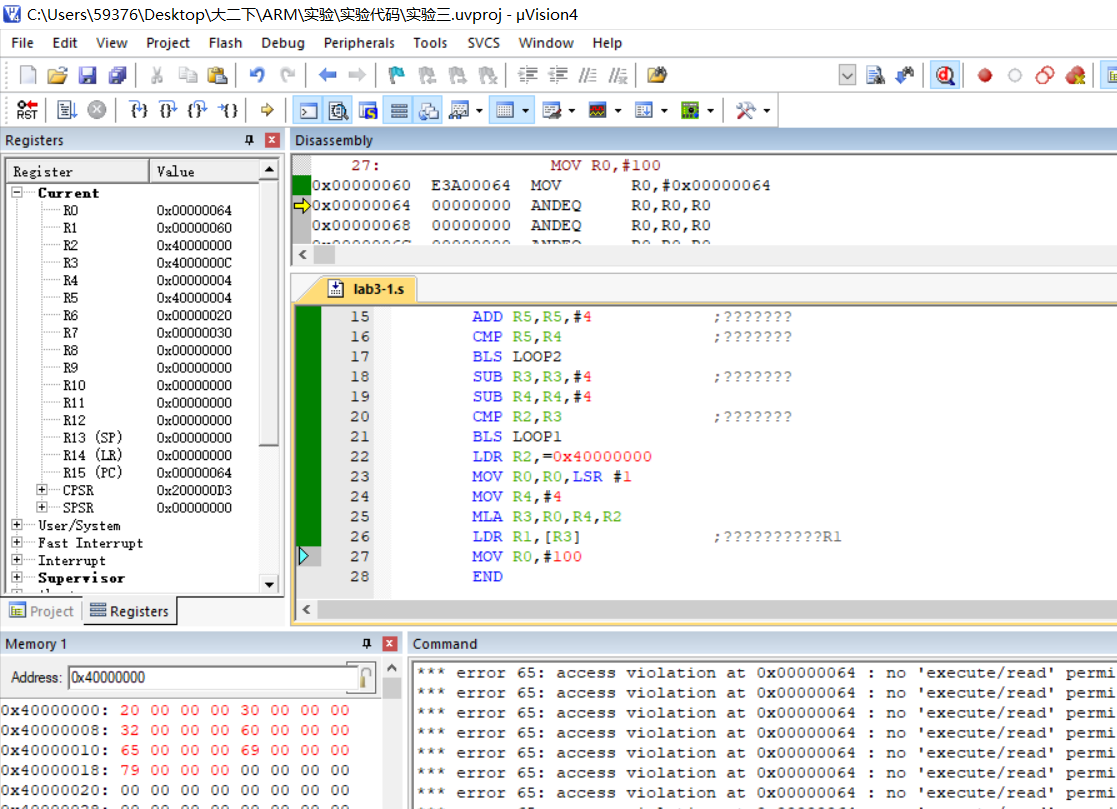


图6 数据二中值滤波程序运行结束时的界面截图

1. **实验结论**

**测式数据一：**0x05、0x01、0x14、0x09、0x23、0x20、0x17这七个数据中，最终排序为：0x01、0x05、0x09、0x14、0x17、0x20、0x23，且R1=0x14，程序运行结果符合预期要求。

**测式数据二：**0x60、0x69、0x30、0x79、0x20、0x32、0x65这七个数据中，最终排序为：0x20、0x30、0x32、0x60、0x65、0x69、0x79，且R1=0x60，程序运行结果符合预期要求。

**2、均值滤波**

1. **实验代码**

代码2 均值滤波程序代码

|  |
| --- |
| ;本代码程序段排序使用冒泡算法！  AREA SORT,CODE,READONLY ;定义一个代码段  ENTRY  MOV R0,#6 ;排序数的个数，为奇数  LDR R2,=0x40000000 ;R2为原始数据的起始地址  SUB R1,R0,#1  MOV R4,#4  MLA R3,R1,R4,R2 ;R3为原始数据的结束地址 R3=R1\*R4+R2  SUB R4,R3,#4 ;R4为原始数据的倒数第二个数的地址  LOOP1 MOV R5,R2 ;重置R5=R2  LOOP2 LDR R6,[R5]  LDR R7,[R5,#4]  CMP R6,R7 ;比较交换，从小到大排序  STRHI R6,[R5,#4] ;将R6的值送入R5+4的地址中，如果R6>R7  STRHI R7,[R5]  ADD R5,R5,#4 ;修改内循环地址  CMP R5,R4 ;内循环结束比较  BLS LOOP2 ;一轮循环结束后，最大值已被冒泡至结束地址R3  SUB R3,R3,#4 ;修改外循环结束地址  SUB R4,R4,#4  CMP R2,R3 ;外循环结束比较  BLS LOOP1    EOR R4,R4,R4  EOR R5,R5,R5 ;R4、R5为求和的结果，先要清零  LDR R2,=0x40000004 ;由于去掉最大值和最小值，求和从第2个数开始  SUB R1,R0,#2 ;R1为求和的总个数  LOOP3 LDR R6,[R2]  ADDS R4,R4,R6  ADC R5,R5,#0  ADD R2,R2,#4  SUB R1,R1,#1 ;修改计数器  CMP R1,#0  BNE LOOP3  MOV R1,R0 ;R1为排序的总个数  SUB R1,R1,#2 ;R2为求和的总个数  MOV R1,R1,LSR #1 ;下面程序为求平均值  LOOP4 MSR CPSR\_F,#0  MOV R5,R5,RRX  MOV R4,R4,RRX  SUB R1,R1,#1  CMP R1,#0  BNE LOOP4  MOV R1,R4 ;均值滤波的结果在R1中  MOV R0,#100 ;为了调试方便加的程序  END |

1. **运行过程及结果界面截图**

**第一组数据：**

下图7是程序运行刚开始的界面。R0设置为6，总共有6个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x0B，0x09，0x03，0x02，0x07，0x05。

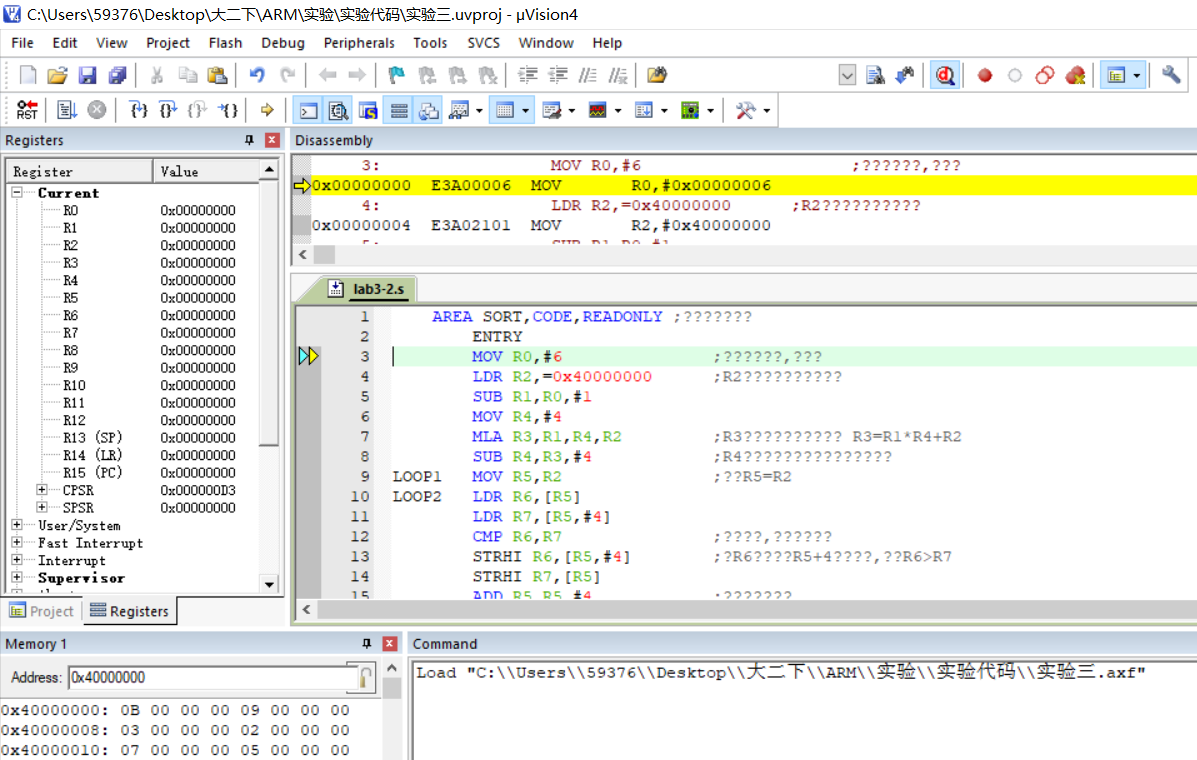


图7 均值滤波程序刚开始运行时的界面截图

下图8是程序运行结束时的界面，可以看出从0x40000000开始的6个存储单元被排序了，运行结果存放于R1中，结果为6。

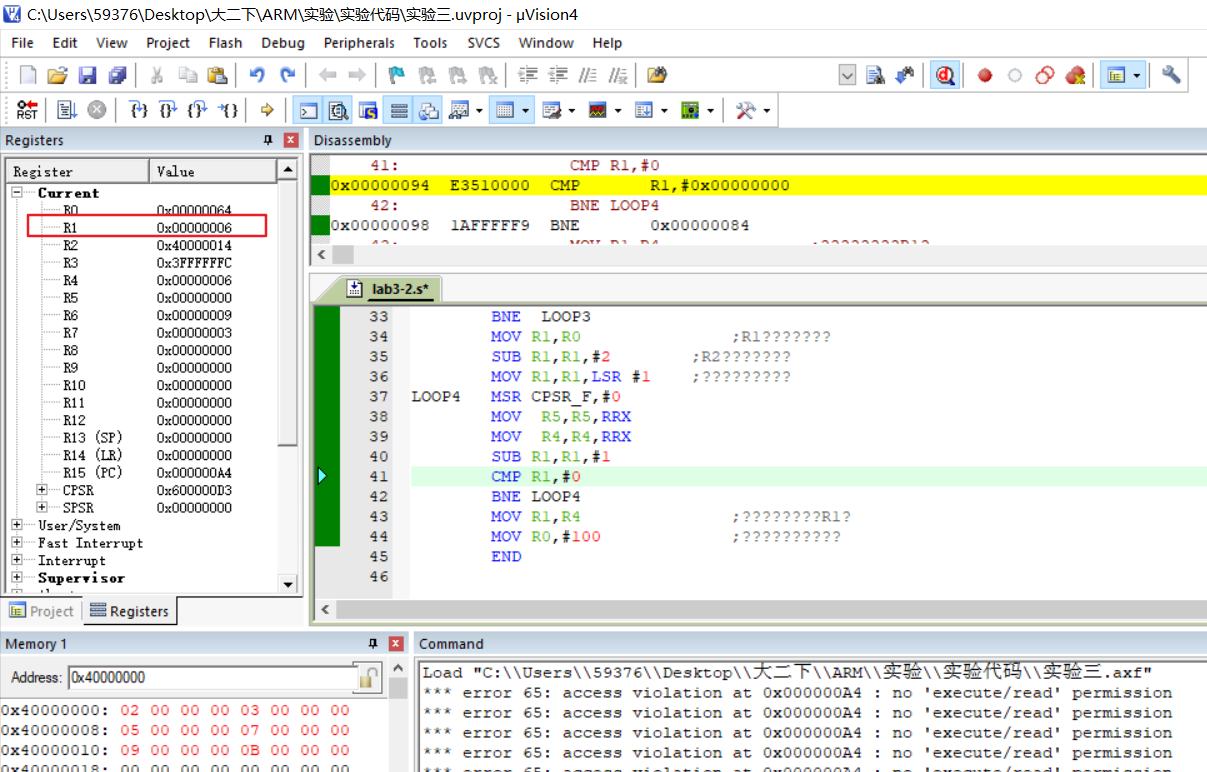


图8 均值滤波程序运行结束时的界面截图

**第二组数据：**

下图9是程序运行刚开始的界面。R0设置为6，总共有6个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x0C，0x08，0x05，0x04，0x03，0x01。

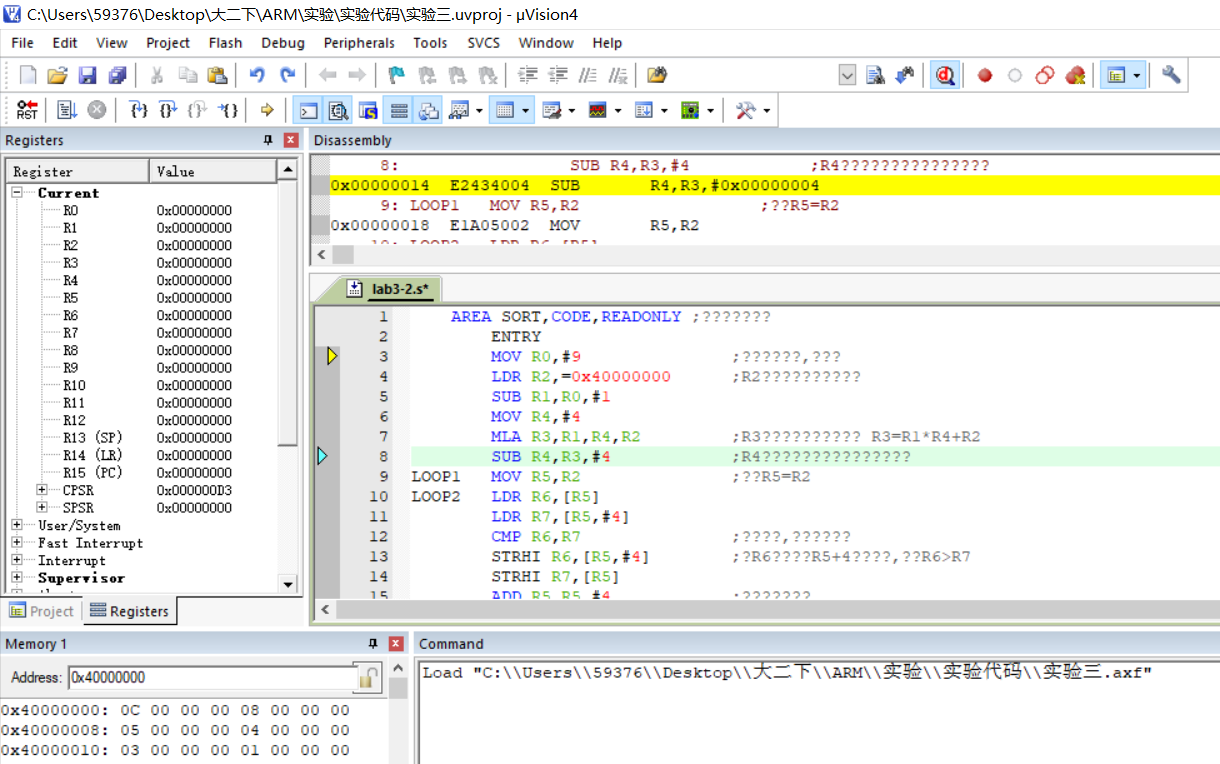


图9 均值滤波程序刚开始运行时的界面截图

下图10是程序运行结束时的界面，可以看出从0x40000000开始的6个存储单元被排序了，运行结果存放于R1中，结果为5。

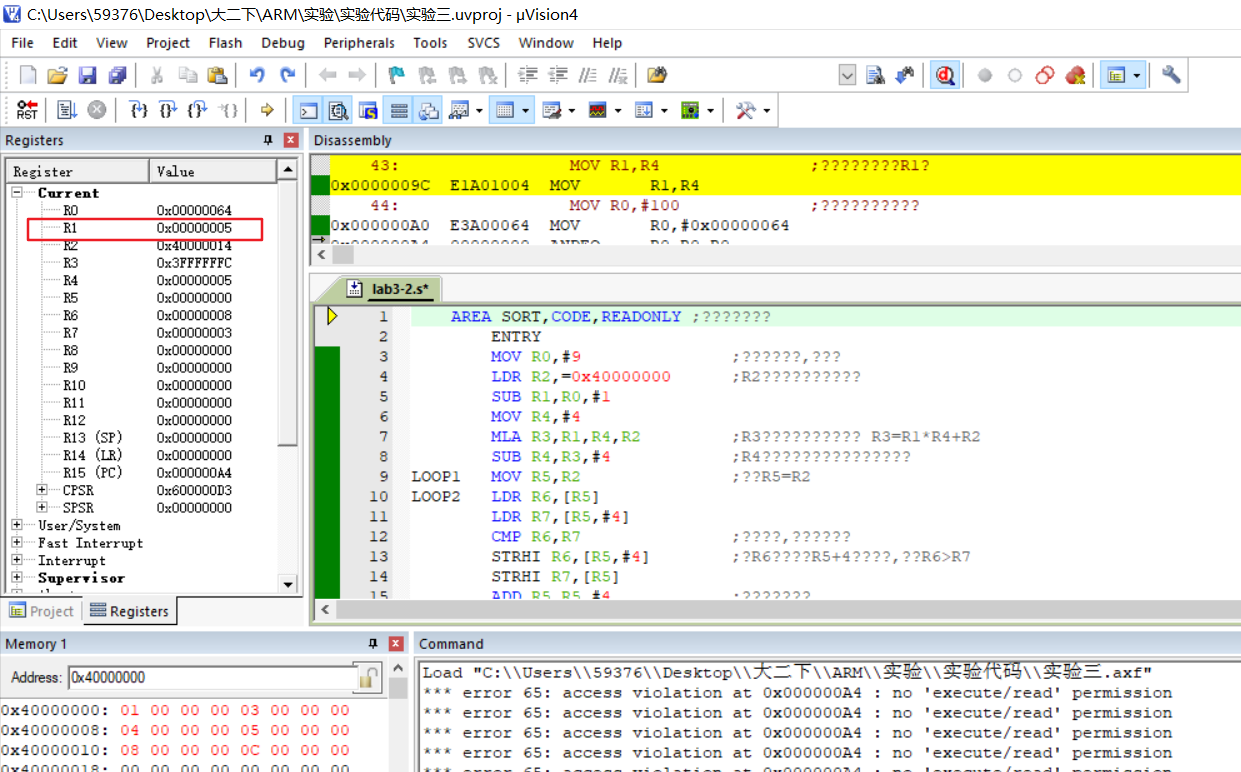


图10 均值滤波程序运行结束时的界面截图

1. **实验结论**

**测式数据一：**0x0B，0x09，0x03，0x02，0x07，0x05这六个数，去掉最大值B和最小值2后，其平均值为6，程序运行结果符合预期要求。

**测式数据二：**0x0C，0x08，0x05，0x04，0x03，0x01这六个数，去掉最大值C和最小值1后，其平均值为5，程序运行结果符合预期要求。

1. 总结及心得体会：

本次实验，通过工程中常用的数字滤波编程实现，提高了我分析问题、解决问题的能力。同时，通过ARM编程实战，进一步巩固和强化了我ARM汇编编程的能力。在本次实验中也遇到了很多问题，有逻辑上的，比如我试图将标准实验报告中代码的排序手法修改为冒泡排序，虽然其原理十分简单，但在具体使用ARM汇编实现的时候却出现了很多问题，ARM汇编编程及其讲究逻辑性和考虑的周到性，需要精确的考虑每一个内存块、每一次bl跳转！不过好在最后在努力许久之后成功实现了排序的冒泡算法实现。当然还有不少是软件上的问题，在查阅资料后都成功解决，我更加深刻的体会到了活用工具的能力是多么的重要。

1. 对本实验过程及方法、手段的改进建议：

本次实验编程难度较大，且实验指导中并未给出一定的提示，因此实验过程中较为困难，虽然代码的逻辑在看懂之后其实并不复杂，但一开始上手的时候还是觉得有点难以下手！希望可以增加更多注释，方便理解。

报告评分：

指导教师签字：