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：2020.5.12**

1. 实验名称：指令系统及寻址方式
2. 实验学时：4
3. 实验目的：
4. 熟悉Keil MDK开发环境，掌握开发平台MDK -ARM的使用；
5. 能正确建立ARM汇编程序工程文件，掌握汇编程序调试方法；
6. 熟练掌握ARM的指令系统以及指令寻址方式；
7. 熟练掌握ARM汇编语言的程序框架，能正确编写ARM汇编程序。
8. 实验原理：

**指令系统寻址方式**

1. **立即数寻址**：也叫立即寻址，操作数本身就在指令中给出，取出指令也就取到了操作数。这个操作数被称为立即数，对应的寻址方式也就叫做立即数寻址。
2. **寄存器寻址**：就是利用寄存器中的数值作为操作数，这种寻址方式是各类微处理器经常采用的一种方式，也是一种执行效率较高的寻址方式。
3. **寄存器移位寻址**：当第二操作数为寄存器型时，在执行寄存器寻址操作时，也可以对第二操作数寄存器进行移位，此时第二操作数形式为。
4. **寄存器间接寻址**：就是以寄存器中的值作为操作数的地址，而操作数本身存放在存储器中。
5. **基址变址寻址**：将基址寄存器的内容与指令中给出的地址偏移量相加，得到操作数所在的存储器的有效地址。变址寻址方式常用于访问某基地址附近的地址单元。（4K范围的偏移）。有三种加偏址的方式：前变址、自动变址和后变址寻址方式。
   1. 前变址模式：先基址+偏址，生成操作数地址，再做指令指定的操作。也叫前索引偏移。
   2. 自动变址模式：先基址+偏移，生成操作数地址，做指令指定的操作。
   3. 后变址模式：即先用基地址传数，然后修改基地址（基址+偏移），也叫后索引偏移。
6. **多寄存器寻址：**采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送。这种寻址方式是多寄存器传送指令LDM/STM的寻址方式，这种寻址方式中用一条指令最多可传送16个通用寄存器的值。连续的寄存器间用“-”连接，否则用“，”分隔。然后自动修改基址寄存器。
7. **堆栈寻址：**是对堆栈进行操作的寻址方式堆栈寻址是隐含的，它使用一个专门的寄存器（堆栈指针SP）指向一块存储区域(堆栈)。四种类型的堆栈工作方式：
   1. 满递增堆栈FA(Full Ascending)：堆栈指针指向最后压入的数据，且由低地址向高地址生长。
   2. 满递减堆栈FD(Full Descending ） ：堆栈指针指向最后压入的数据，且由高地址向低地址生长。
   3. 空递增堆栈EA(Empty Ascending)：堆栈指针指向下一个将要放入数据的空位置，且由低地址向高地址生长。
   4. 空递减堆栈ED(Empty Descending)：堆栈指针指向下一个将要放入数据的空位置，且由高地址向低地址生长。
8. **相对寻址：**以程序计数器PC的当前值为基地址，指令中的地址标号作为偏移量，将两者相加之后得到操作数的有效地址。
9. 实验内容：
10. 学习Keil MDK-ARM开发平台的的使用，包括新建一个工程、在建立的工程中编写相关程序。
11. 掌握对ARM汇编工程编译、调试的方法。
12. 对40多条常用的指令，使用不同的寻址方式进行测试，并通过查看寄存器、存储器、程序状态寄存器的内容，检查是否与期望一致。
13. 针对这40多条指令中，可以加S标记和条件的指令，使用S标记和不同的条件进行测试，并通过查看寄存器、存储器、程序状态寄存器的内容，检查是否与期望一致。
14. 实验器材（设备、元器件）：
15. PC机一台；
16. Keil MDK-ARM uVision4开发工具。
17. 实验步骤：
18. 打开Keil MDK-ARM uVision4开发工具；
19. 新建一个工程文件；
20. 在新建的工程文件中，添加新的源程序文件
21. 编写代码
22. 选择“Build target”菜单对编写好的工程文件进行编译链接。
23. 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。
24. 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）
25. **相关指令执行后，结果如下：**

**实验1-1：寻址方式指令系统（一）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **指令** | **寻址方式或后缀** | **具体执行指令** | **结果说明** |
| 1 | MOV | 立即数寻址+S条件后缀 | MOVS R2，#0x0 | R2 ← #0x0  并影响标志位：nZcv(注：字母大写，表示相关标志位为1) |
| 寄存器寻址+EQ条件后缀 | CMP R2，R1  MOVEQ R2，R1 | 如果R2=R1  则执行R2 ← R1，否则不执行 |
| 寄存器移位寻址 | MOV R0，#0x01  MOV R1，R0，LSL#3 | 将寄存器R0左移3位传送到寄存器R1。本例执行后，R1=0x08 |
| 2 | MVN | 寄存器寻址 | MOV R2,#15  MVN R1，R2 | Rl←R2取反,结果如下：  R1=0xFFFFFFF0,R2=0x0000000F |
| 寄存器移位寻址 | MVN R1,R2,LSR#4 | 将寄存器R0右4位取反，结果送入R1 |
| 基址变址寻址 | MVN R2,[R4,#5] | 将R4+4地址上的数据取反，结果送入R2 |
| 3 | AND | 立即数寻址  +S后缀 | ANDS R0，R0，#0x01 | R0＝R0&0x01  取出最低位数据 |
| 寄存器移位寻址 | ANDS R0，R0，R1,LSL#5 | R0＝R0&(R1左移5位) |
| 基址变址寻址 | ANDS R0，R0，[R1,#3] | R0＝R0&(R1+3地址上的数据)： |
| 4 | EOR | 立即数寻址 | EOR R1，R1，#0x0F | ；将R1的低4位取反 |
| 寄存器寻址  +S后缀 | EORS R0，R5，R3 | ；R0<-R5异或R3，；并影响标志位 |
| 5 | SUB | 立即数寻址  +S后缀 | MOV R0,#1  SUBS R0,R0,#1 | RO←R0-1,由于R0=1，且带有S后缀，故影响标志位：nZcv(注：字母大写，表示相关标志位为1) |
| 寄存器寻址  +GT移位后缀 | CMP R0,R1  SUBGT R0,R0,R1 | 如果R0>R1,则R0=R0-R1 |
| 6 | RSB | 立即数寻址 | RSBR3，R1，#0xFF00 | R3＝0xFF00-R1 |
| 寄存器移位寻址  +S后缀 | RSBSR1，R2，R2，LSL#2 | R1=R2<<2-R2  R1=R2×3) |
| 7 | ADD | 立即数寻址 | ADD R0,R0,#1 | RO←R0+1 |
| 寄存器寻址 | ADD R0,R1,R2 | R0←R1+R2 |
| 寄存器移位寻址 | ADD R3,R2,R1,LSR #2 | R3←R2+(R1右移2位) |
| 8 | ADC | 寄存器间接寻址 | ADC R0,R1,[R2] | R0←R1+[R2]+C |
| 寄存器寻址 | ADC R0,R1,R2 | R0←R1+R2+C |
| 9 | SBC | 寄存器寻址 | RSC R2,RO,R1 | R0-R1再减去CPSR中C条件标志位的反码，结果保存到R2中 |
| 寄存器间接寻址 | RSC R2,RO,[R1] | R0-[R1]再减去CPSR中C条件标志位的反码，结果保存到R2中 |
| 10 | RSC | 寄存器间接寻址 | RSC R2,RO,[R1] | [R1]-R0再减去CPSR中C条件标志位的反码，结果保存到R2中 |
| 寄存器寻址 | RSC R2,RO,R1 | R1-R0再减去CPSR中C条件标志位的反码，结果保存到R2中 |
| 11 | TST | 立即数寻址 | TST R0，#0x01 | 判断R0的最低位是否为0，更新CPSR的相应条件标志位 |
| 基址变址寻址 | TST R0,[R2,#OXOB] | RO与[R2,#OXOB]进行按位逻辑与操作，更新CPSR的相应条件标志位 |
| 12 | TEQ | 基址变址寻址 | TSQ R0,[R2,#OXOB] | 比较R0和[R2,#OXOB]是否相等，更新CPSR的相应条件标志位 |
| 寄存器寻址 | TEQ R0，R1 | 比较R0与R1是否相等，更新CPSR的相应条件标志位 |
| 13 | CMP | 基址变址寻址 | CMP R1，[R1,#2] | (R1)-(R1+2地址的数据)；根据结果设置CPSR的标志位。 |
| 寄存器寻址 | CMP R1，R0 | (R1)-(R0)；根据结果设置CPSR的标志位。 |
| 14 | CMN | 立即数寻址 | CMN R0，#1 | R0+1，判断R0是否为-1。 |
| 寄存器间接寻址 | CMN R1,[R0] | R1减去[R0],更新CPSR中的相应条件标志位 |
| 15 | ORR | 立即数寻址 | ORR R0，R0，#0x0F | 将R0的低4位置1 |
| 寄存器移位寻址 | MOV R3,#1  MOV R0,#1  ORR R3，R0，R3，LSL#8 | 将R3左移8位后将R0低8位送至R3  最终结果R3= 0x00000101 |
| 16 | BIC | 立即数寻址 | BIC R1，R1，#0x0F | 将R1的低4位清0，  其它位不变 |
| 寄存器寻址 | MOV R2,#0x0F  MOV R1,#0xFF  BIC R1，R1，R2 | 将R2的值与R1的值的反码按位逻辑与操作，保存到R1中 |
| 17 | MUL | 基址变址寻址 | MUL R0,R1,[R2,#3] | R0=R1\*[R2,#3] |
| 寄存器移位寻址 | MUL R0,R1,R2,LSL#3 | 将寄存器R2左移三位后与R1相乘，传到R0 |
| 18 | MLA | 寄存器寻址  +S后缀 | MLAS R0,R1,R2,R3 | R0←(R1)X(R2)+(R3),  并更新CPSR标志位 |
| 立即数寻址 | MLAS R0,R1,R2,OX0B | R0←(R1)X(R2)+OX0B,  并更新CPSR标志位 |
| 19 | UMULL | 寄存器寻址 | UMULL R0，R1，R5，R8 | (R1,R0)←R5×R8，低32位保存在R0，高32位保存在R1 |
| 寄存器间接寻址 | UMULL R0，R1，R5，[R8] | (R1,R0)←R5×[R8]，低32位保存在R0，高32位保存在R1 |
| 寄存器移位寻址 | UMULL R0，R1，R5，R8，LSL#2 | 将寄存器R8左移2位后与R5相乘, 低32位保存在R0，高32位保存在R1 |
| 20 | UMLAL | 基址变址寻址 | UMLAL R0，R1，R5，[R8,#2] | (R1,R0)←R5×[R8,#2]+(R1,R0), 低32位保存在R0，高32位保存在R1 |
| 寄存器寻址 | UMLAL R0，R1，R5，R8 | (R1,R0)←R5×R8+(R1,R0), 低32位保存在R0，高32位保存在R1 |
| 寄存器间接寻址 | UMLAL R0，R1，R5，[R8] | (R1,R0)←R5×[R8]+(R1,R0), 低32位保存在R0，高32位保存在R1 |
| 21 | SMULL | 寄存器间接寻址 | SMULL R2，R3，R7，[R6] | (R3,R2)←R7×[R6], 低32位保存在R2，高32位保存在R3 |
| 寄存器寻址 | SMULL R2，R3，R7，R6 | (R3,R2)←R7×[R6], 低32位保存在R2，高32位保存在R3 |
| 基址变址寻址 | SMULL R2，R3，R7，[R6,#5] | (R3,R2)←R7×[R6,#5], 低32位保存在R2，高32位保存在R3 |
| 22 | SMLAL | 寄存器移位寻址 | SMLAL R2，R3，R7，R6,LSL#6 | (R3,R2)←R7×(R6左移位六位)+(R3,R2), 低32位保存在R2，高32位保存在R3 |
| 基址变址寻址 | SMLAL R2，R3，R7，[R6,#3] | (R3,R2)←R7×[R6,#3]+(R3,R2), 低32位保存在R2，高32位保存在R3 |
| 寄存器寻址 | SMLAL R2，R3，R7，R6 | (R3,R2)←R7×R6+(R3,R2), 低32位保存在R2，高32位保存在R3 |

**实验1-2：寻址方式指令系统（二）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **指令** | **寻址方式或后缀** | **具体执行指令** | **结果说明** |
| 1 | LDR | 寄存器间接寻址 | LDR R4,=0x40000000  LDR R0,[R4] | RO←[R4]地址上的字数据 |
| 基址变址寻址 | LDR R1,=0x40000000  LDR R0,[R1,#4] | R0←[R1+4]地址上的字数据，先手动设置0x40000004数据位置为6，则最后运行后R0=6. |
| 2 | STR | 寄存器间接寻址 | STR R0,[R4] | [R4]作为地址←R0的字数据 |
| 基址变址寻址 | STR R0,[R1,#4] | [R1+4]作为地址←R0的字数据 |
| 3 | LDRB | 寄存器间接寻址 | LDRB R0,[R4] | RO的低8位数据中，高24位为0←[R4]地址上的字节数据 |
| 基址变址寻址 | LDRB R0,[R1,#4] | RO的低8位数据中，高24位为0←[R1+4]地址上的字节数据： |
| 4 | STRB | 寄存器间接寻址 | STRB R0,[R4] | [R4]作为地址←R0的字节数据 |
| 基址变址寻址  +T后缀 | STRBT R0,[R1,#4] | [R1+4]作为地址←R0的字节数据，处理器是在用户模式下 |
| 5 | LDRH | 寄存器间接寻址 | LDRH R0,[R4] | 加载[R4]上无符号半字数据到R0的低16位,高16位清零。 |
| 基址变址寻址  +T后缀 | LDRHT R0,[R1,#4] | 加载[R1+4]上无符号半字数据到R0的低16位,高16位清零。处理器是在用户模式下 |
| 6 | STRH | 寄存器间接寻址 | STRH R0,[R4] | 存储R0中的低16位半字数据到[R4]上 |
| 基址变址寻址  +T后缀 | STRHT R0,[R1,#4] | 存储R0中的低16位半字数据到[R1+4]上，处理器是在用户模式下 |
| 7 | LDMIA | 多寄存器寻址 | LDMIA R0!,{R1-R3} | R1←[R0]、R2←[R0+4]、R3←[R0+8]  由于存在！后缀，故R0←[R0+12] |
| 8 | LDMIB | 多寄存器寻址 | LDMIB R0！，{R1—R3} | R1←[R0+4]、R2←[R0+8]、R3←[R0+12]  由于存在！后缀，故R0←[R0+12] |
| 9 | LDMDA | 多寄存器寻址 | LDMDA R0!,{R1-R3} | R1←[R0]、R2←[R0-4]、R3←[R0-8]  由于存在！后缀，故R0←[R0-12] |
| 10 | LDMDB | 多寄存器寻址 | LDMDB R0!,{R1-R3} | R1←[R0-4]、R2←[R0-8]、R3←[R0-12]  由于存在！后缀，故R0←[R0-12] |
| 11 | LDMFA | 堆栈寻址 | LDMFA SP!，{R1-R3} | R1←[SP]、R2←[SP-4]、R3←[SP-8]  由于存在！后缀，故SP←[SP-12] |
| 12 | STMFA | 堆栈寻址 | STMFA SP!，{R0-R3} | R1→[SP]、R2→[SP-4]、R3→[SP-8]  由于存在！后缀，故SP←[SP-12] |
| 13 | LDMFD | 堆栈寻址+^后缀 | LDMFD SP!，{R0-R7，PC}^ | 递减满堆栈，恢复现场，包括CPSR，异常处理返回，SP值更新 |
| 14 | STMFD | 堆栈寻址 | STMFD sp!,{R0-R7,LR} | 现场保存，将R0～R7、LR入栈，SP值更新。 |
| 15 | LDMEA | 堆栈寻址 | LDMEA SP!，{R1-R3} | R1←[SP]、R2←[SP+4]、R3←[SP+8]  由于存在！后缀，故SP←[SP+12] |
| 16 | STMEA | 堆栈寻址 | STMEA SP!，{R0-R3} | R1→[SP]、R2→[SP+4]、R3→[SP+8]  由于存在！后缀，故SP←[SP+12] |
| 17 | LDMED | 堆栈寻址 | LDMED SP!，{R1-R3} | R1←[SP+4]、R2←[SP+8]、R3←[SP+12]  由于存在！后缀，故SP←[SP+12] |
| 18 | STMED | 堆栈寻址 | STMED SP!，{R0-R3} | R1→[SP+4]、R2→[SP+8]、R3→[SP+12]  由于存在！后缀，故SP←[SP+12] |
| 19 | SWP | 寄存器间接寻址 | SWP R1，R1，[R0] | 将R1的内容与R0指向的存储单元的内容进行交换。 |
| 基址变址寻址  +B后缀 | SWPB R1，R2，[R0，#5] | 将R0+5指向的存储单元的内容读取1字节数据到R1中(高24位清零)，并将R2的内容写入到该内存单元中(最低字节有效) |
| 20 | B | 立即数寻址 | B 0x1234 | 跳转到绝对地址Ox1234 |
| 相对寻址  +NE条件后缀 | WLQ CMP R0,R1  SUBGT R0,R0,R1  SUBLT R1,R1,R0  BNE WLQ  MOV PC,LR | ；比较a和b的大小  ；如果a>b,则a=a-b  ；如果a<b,则b=b-a  ；如果a!=b,则返回WLQ  ；如果a=b,则返回主程序 |
| 21 | BL | 立即数寻址 | BL 0x1234 | 跳转到绝对地址Ox1234,在跳转之前会把BL指令的下一条指令地址（断点地址）保存到连接寄存器LR（R14） |
| 相对寻址  +NE条件后缀 | WLQ CMP R0,R1  SUBGT R0,R0,R1  SUBLT R1,R1,R0  BLNE WLQ  MOV PC,LR | ；比较a和b的大小  ；如果a>b,则a=a-b  ；如果a<b,则b=b-a  ；如果a!=b,则返回WLQ  ；如果a=b,则返回主程序  在跳转之前会把BL指令的下一条指令地址（断点地址）保存到连接寄存器LR（R14） |
| 22 | BX | 寄存器寻址 | CODE32  ARM1 LDR R0，=Thumb1+#1  MOV LR,PC  BX R0  ADD R1,R2,#2  …  CODE16 Thumb1 ADD R1，R3,#1  BXLR | ；把Thumb地址赋给R0，末位置1  ；跳转，切换状态 |
| 23 | MRS | 寄存器寻址 | MRS R1，CPSR | R1<-CPSR |
| 23 | MSR | 立即数寻址 | MSR CPSR，#3 | 传送0b00010001到CPSR |
| 寄存器寻址 | MSR CPSR，R0 | 传送R0的内容到CPSR |

**2、部分指令测试界面截图**

下图图1为MVN R1，R2，我们先设置R2=#15，即0x0000000F，则执行MVN命令后进行取反，则R1= 0xFFFFFFF0，测试结果符合预期要求

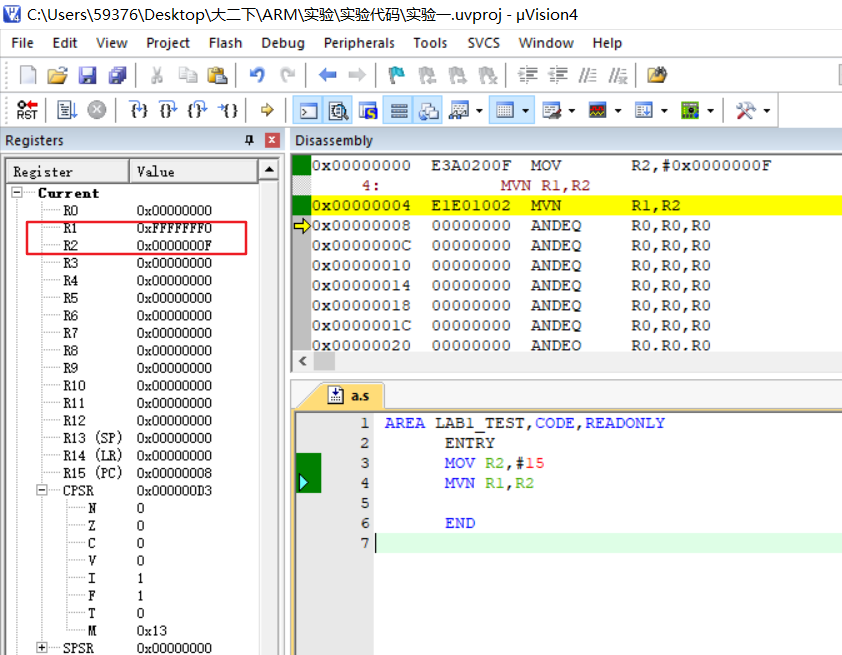


图1 MVN R1，R2 指令执行测试截图

下图图2为SUBS R0，#1测式结果图，先通过运行MOV R0,#1，设置R0=1，再运行SUBS R0，#1，最终得到R0=1。由于存在S后缀，故影响CPSR，可以看到，由于R0=1，故CPSR中Z被置于1了，指令符合预期要求。

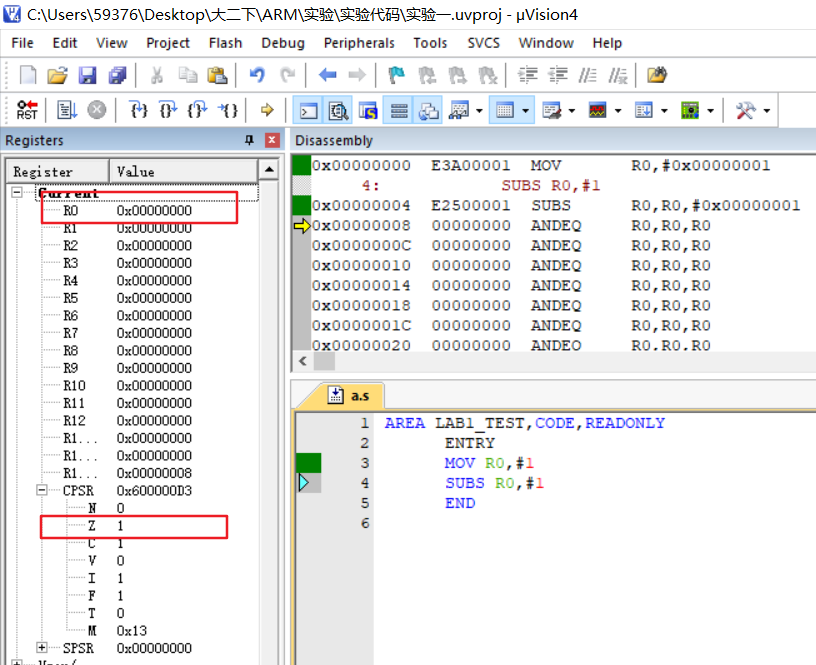


图2 SUBS R0，#1指令执行测试截图

下图图3为ORR R3,R0,R3,LSL#8测式结果图，先通过运行MOV R0,#1，MOV R3,#1设置R0=R3=1，然后将R3左移8位，将R0的低8位送到R3的低八位，得到结果R3= 0x00000101，符合指令预期要求。

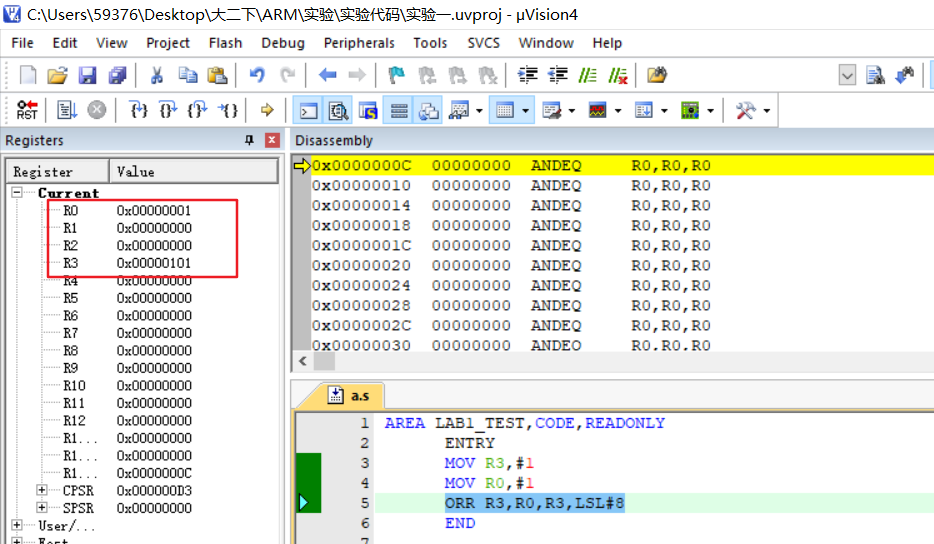


图3 ORR R3,R0,R3,LSL#8指令执行测试截图

下图图4为BIC R1，R1，R2测式结果图，先通过运行MOV R2,#0x0F，MOV R1,#0xFF将R2和R1设置到目标数值，运行测试指令，将R2的值与R1的值的反码按位逻辑与操作，保存到R1中，即清除低四位为0的操作，得到结果R3= 0x000000F0，符合指令预期要求。

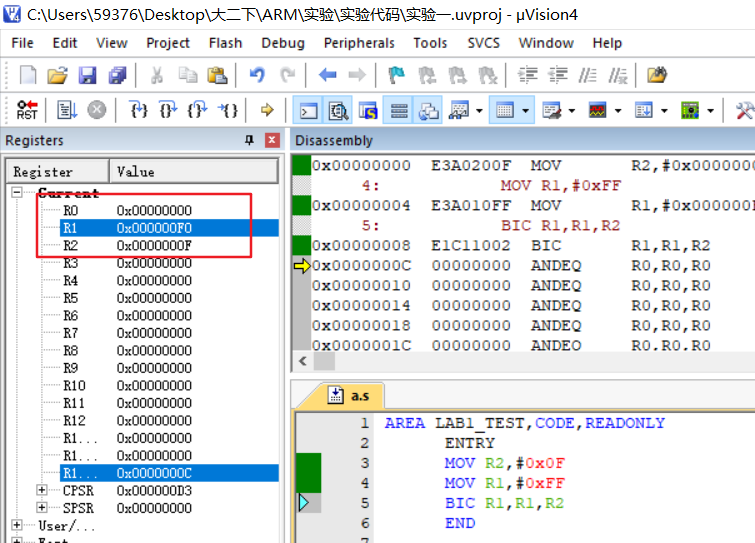


图4 BIC R1，R1，R2指令执行测试截图

下图图5为LDR R0,[R1,#4]测式结果图，先通过运行LDR R1,=0x40000000将R1设置到目标数值，在手动修改0x40000004的数据为6，然后运行LDR R0,[R1,#4]，可以得到R0=0x40000004，符合指令预期效果。

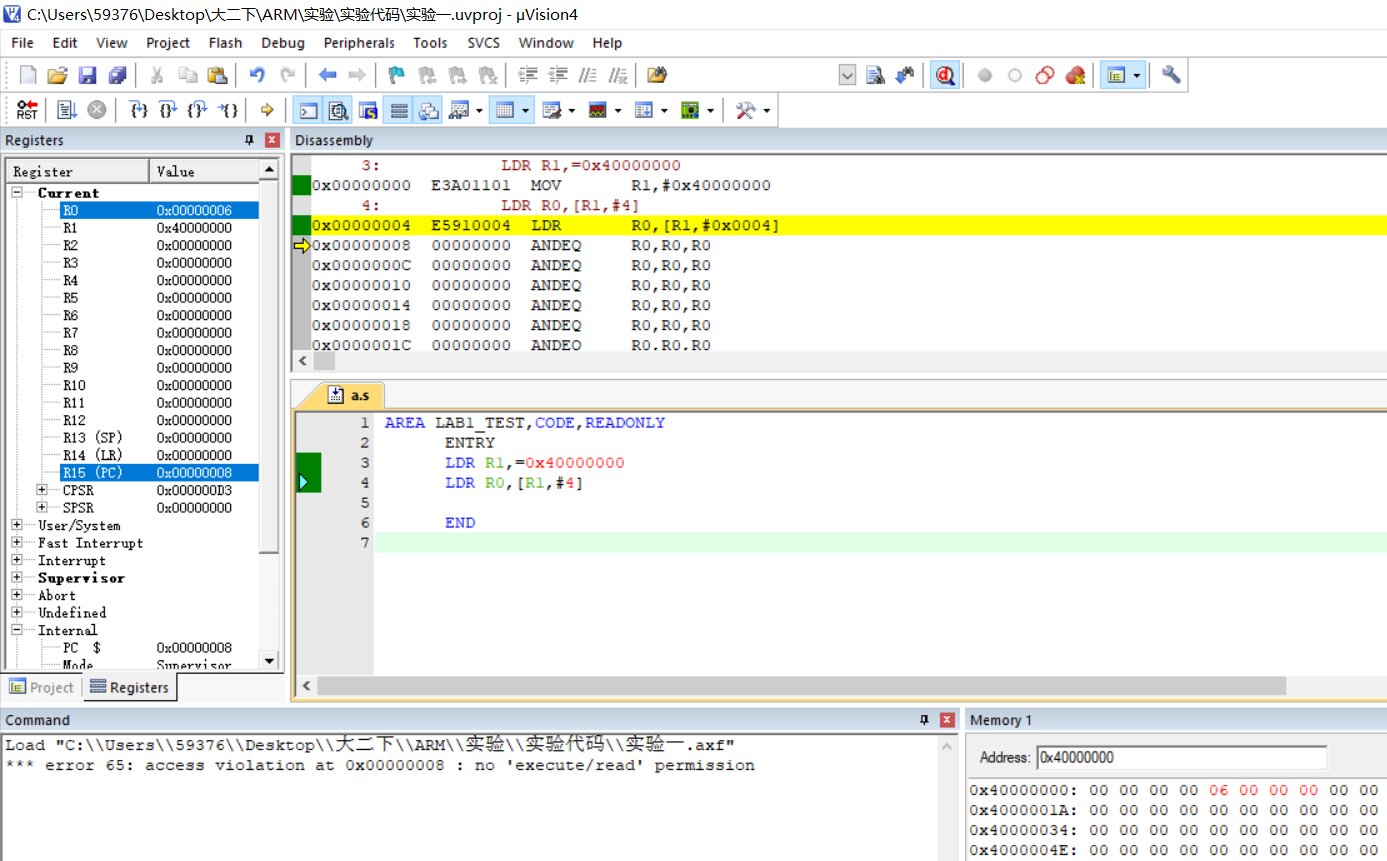


图5 LDR R0,[R1,#4]指令执行测试截图

下图图6为MSR CPSR,#3测式结果图，可以发现，该指令在Build Target时无法成功。查阅资料后，将代码改为MSR CPSR\_cxsf,#3，可通过Build Target，图7为MSR CPSR\_cxsf,#3测式结果图，可以看到CPSR=0x00000003，符合指令预期效果。

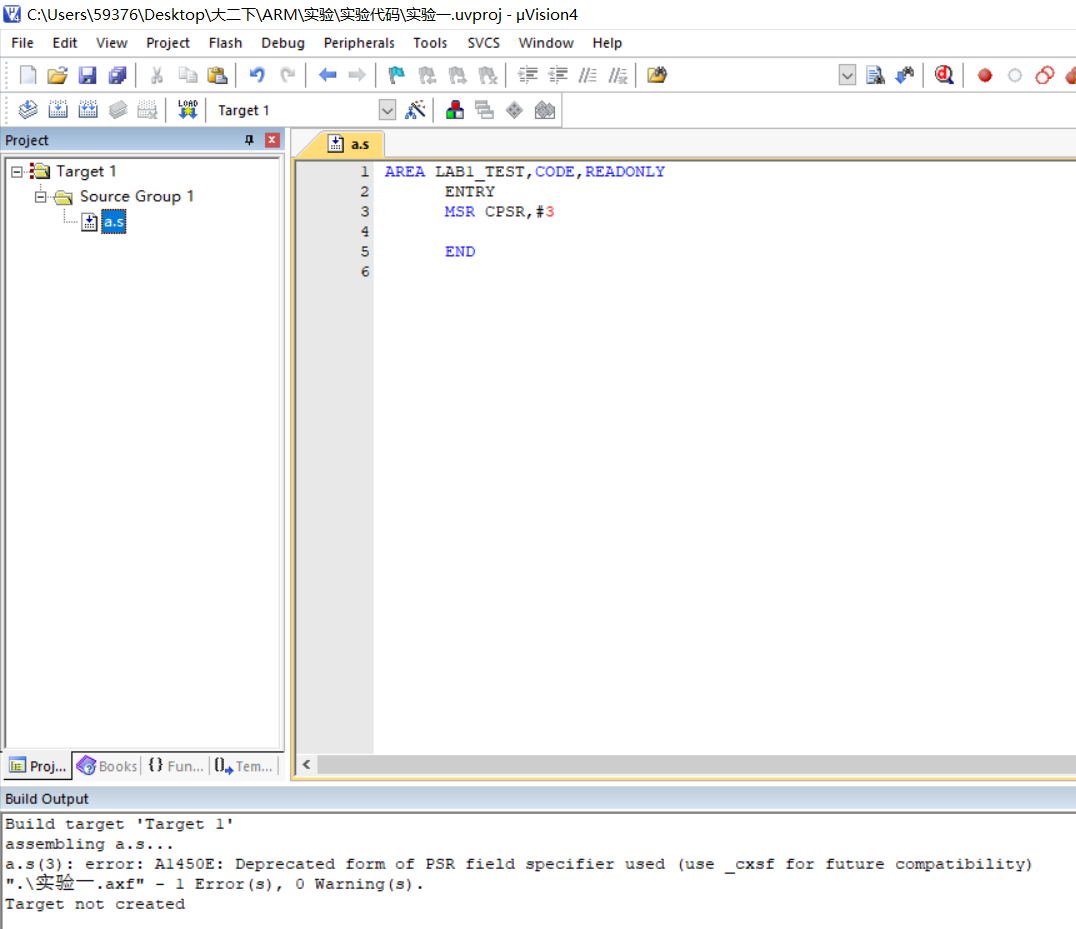


图6 MSR CPSR,#3Build Target 失败图

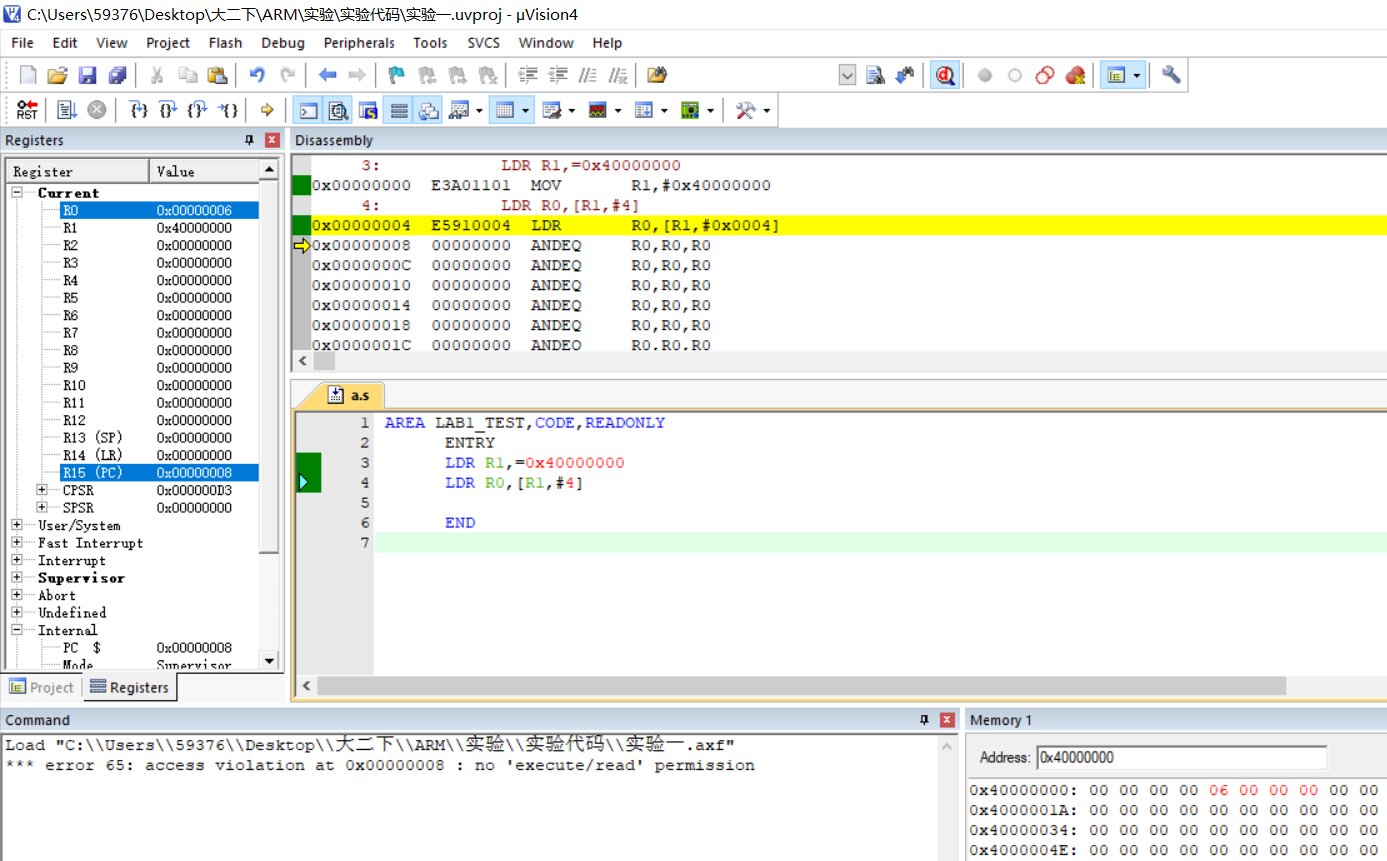


图7 MSR CPSR\_cxsf,#3指令执行测试截图

**3、实验结论**

通过测试可知，MSR指令与课本中预期结果不同，不同之处在于指令格式需要修改为：

|  |
| --- |
| MSR{cond}CPSR\_cxsf，#immed  MSR{cond}CPSR\_cxsf，Rm  MSR{cond}SPSR\_cxsf，#immed  MSR{cond}SPSR\_cxsf，Rm |

必须在CPSR或者SPSR后面加上\_cxsf才能在keil 4中build target成功。

BLX指令在keil 4中无法测式成功。

除MSR、BLX之外的所有指令都与课本预期相同。

1. 总结及心得体会：

通过本次实验，我熟悉了Keil MDK开发环境，学会了正确建立ARM汇编程序工程文件，掌握了汇编程序调试方法。更重要的是，在验证指令的过程中，我熟练掌握ARM的指令系统以及指令寻址方式和ARM汇编语言的程序框架，大大提高了编写ARM汇编程序的能力。

同时在验证指令的过程中，也发现了一些指令的运行结果有与书本不同的地方，比如MSR指令需要在CPSR或者SPSR后面加上\_cxsf才能在keil 4中build target成功，更加印证了实践出真知的道理。

1. 对本实验过程及方法、手段的改进建议：

本次实验重复性工作过于繁多，且设置有些不合理，有些指令无法找出多种寻址方式，希望能有所改进。

报告评分：

指导教师签字：