电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2022130102013

姓 名 成棋伟

（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： 成棋伟 学号：2022130102013**

**指导教师： 兰刚**

**实验地点： 西304 实验时间：2025.04.22**

1. 实验名称：ARM主程序调用ARM/C语言子程序
2. 实验学时：4
3. 实验目的：
4. 了解ARM应用程序框架。
5. 了解ARM汇编程序函数和C语言程序函数相互调用时，遵循的ATPCS标准；
6. 了解和掌握ARM汇编程序调用C语言程序函数的基本方法；
7. 了解和掌握ARM汇编程序调用C语言程序函数的参数传递过程。
8. 实验原理：
9. **ARM工程**

由于C语言便于理解，有大量的支持库，所以它是当前ARM程序设计所使用的主要编程语言。

对硬件系统的初始化、CPU状态设定、中断使能、主频设定 以及RAM控制参数初始化等C程序力所不能及的底层操作，还是要由汇编语言程序 来完成。

ARM工程 的各种源文件之间的关系，以及最后形成可执行文件的过程如下图1所示：

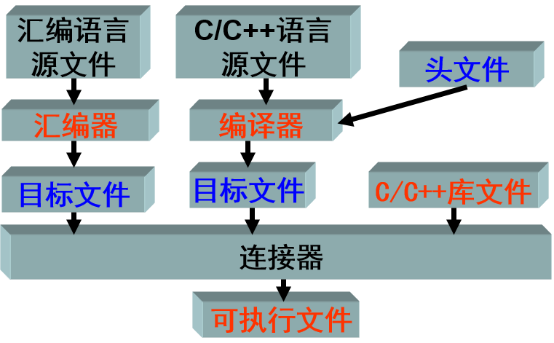


图1 汇编语言和C语言混合编译链接示意图

在应用系统的程序设计中，若所有的编程任务均用汇编语言 来完成，其工作量是可想而知的，这样做也不利于系统升级或应用软件移植。

通常汇编语言部分完成系统硬件的初始化；高级语言部分完成用户的应用。

执行时，首先执行初始化部分，然后再跳转到C/C++部分。整个程序结构显得清晰明了，容易理解。为方便工程开发，ARM公司的开发环境ARM ADS为用户提供了一个可以选用的应用程序框架。该框架把为用户程序做准备工作的程序分成了： 启动代码 和 应用程序初始化 两部分。

用于硬件初始化的汇编语言部分叫做 启动代码；用于应用程序初始化的C部分叫做初始化部分。整个程序如下图2所示：

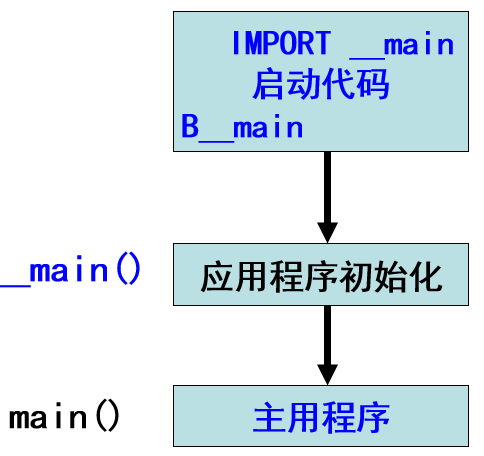


图2 ARM应用程序框架

1. **过程调用标准ATPCS**

在ARM工程中，C程序调用汇编函数和汇编程序调用C函数是经常发生的事情。为此人们制定了ARM-Thumb过程调用标准ATPCS（ARM-Thumb Procedure Call Standard）。

1. ATPCS规定，ARM的数据堆栈为FD型堆栈，即递减满堆栈。
2. ATPCS标准规定，对于参数个数不多于4的函数，编译器必须按参数在列表中的顺序，自左向右为它们分配寄存器R0~R3。其中函数返回时，R0还被用来存放函数的返回值。
3. 如果函数的参数多于4个，那么多余的参数则按自右向左的顺序压入数据堆栈，即参数入栈顺序与参数顺序相反。
4. 根据ATPCS的C语言程序调用汇编函数，参数由左向右依次传递给寄存器R0~R3的规则。
5. **子程序的调用与返回**

人们把可以多次反复调用的、能完成指定功能的程序段称为“子程序”。把调用子程序的程序称为“主程序”。

为进行识别，子程序的 第1条指令 之前必须赋予一个 标号，以便其他程序可以用这个标号调用子程序。

在 ARM 汇编语言程序中，主程序一般通过 BL 指令来调用子程序。该指令在执行时完成如下操作：将子程序的返回地址存放在连接寄存器 LR 中，同时将程序计数器 PC 指向子程序的入口点。

为使子程序执行完毕能 返回 主程序的调用处，子程序末尾 处应有 MOV、LDMFD 等指令，并在指令中将返回地址重新复制到 PC 中。

在调用子程序的同时，也可以使用 R0~R3 来进行 参数的传递 和从子程序返回 运算结果。

1. 实验内容：
2. 编写一个宏strcopy实现字符串拷贝功能（字符串以0做为结束符），该宏具有两个个输入参数：$DEST和$SRC，$SRC是源字符串的在内存中的首地址，$DEST是目的字符串在内存中的首地址。
3. 编写一个C语言子程序Proc\_C，C语言子程序的参数个数为6个分别是i1、i2、i3、i4、i5、i6，C言子程序实现的功能为：（i1+i2+i3+i4）\*i5-i6，观察参数是如何传递的。
4. 编写一个ARM汇编语言子程序Proc\_Arm，通过R0传入一个参数i，在汇编子程序Proc\_Arm中调用C函数Proc\_C()，以实现下面的功能： int Proc\_C(int i) {return –Proc\_C(i, 2\*i, 3\*i, 4\*i,5\*i, 6\*i)}。返回结果送至标号为val的内存字单元中保存。
5. 在汇编主程序中，使用strcopy宏实现两次字符串的拷贝功能，要求在代码段给两个字符串分配存储空间，拷贝到RAM区指定的地址单元。
6. 在汇编主程序中，调用Proc\_Arm子程序，传入参数为1。
7. 分析通过反汇编得到的C程序的ARM指令代码段，了解参数传递过程。
8. 实验器材（设备、元器件）：
9. PC机一台；
10. Keil MDK-ARM uVision4开发工具。
11. 实验步骤：
12. 打开Keil MDK-ARM uVision4开发工具；
13. 新建一个工程文件；
14. 在新建的工程文件中，添加新的源程序文件
15. 编写代码
16. 选择“Build target”菜单对编写好的工程文件进行编译链接。
17. 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。
18. 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）
19. **ARM指令主程序调用ARM指令子程序；**
20. **程序代码**

程序代码如代码1所示：

代码1 ARM指令主程序调用ARM指令子程序

|  |
| --- |
| PRESERVE8  AREA TEST,CODE,READONLY  ENTRY  BL func ;跳转至arm子程序    LDR R1,=END1 ;跳转至END1即END  BX R1  func ;arm子程序  MOV R1,#0X12  MOV R2,#0X12  BX LR  END1  END |

1. **运行过程及结果界面截图**

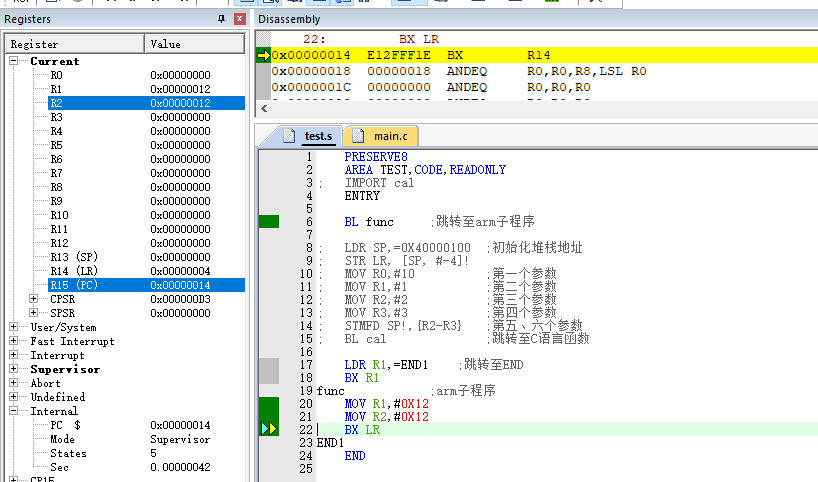


图1 ARM指令主程序调用ARM指令子程序运行结果截图

1. **实验结果分析**

由图1可见，子程序成功调用，向R1 R2传入0X12的数据

1. **实验结论**

ARM汇编可通过调用子程序的方式实现临时运行一段提前准备好的程序，再返回到主程序继续执行

1. **ARM指令主程序调用C语言子程序；**
2. **程序代码**

ARM指令主程序如代码2所示

代码2 RM指令主程序调用C语言子程序的RM指令主程序

|  |
| --- |
| PRESERVE8  AREA TEST,CODE,READONLY  IMPORT cal  ENTRY    LDR SP,=0X40000100 ;初始化堆栈地址  STR LR, [SP, #-4]!  MOV R0,#10 ;第一个参数  MOV R1,#1 ;第二个参数  MOV R2,#2 ;第三个参数  MOV R3,#3 ;第四个参数  STMFD SP!,{R2-R3} ;第五、六个参数  BL cal ;跳转至C语言函数    LDR R1,=END1 ;跳转至END  BX R1  END1  END |

C语言子程序代码如代码3所示

代码3 RM指令主程序调用C语言子程序的C语言子程序

|  |
| --- |
| int cal(int i1,int i2,int i3,int i4,int i5,int i6){  return (i1+i2+i3+i4)\*i5-i6;  }//计算函数 |

1. **运行过程及结果界面截图**

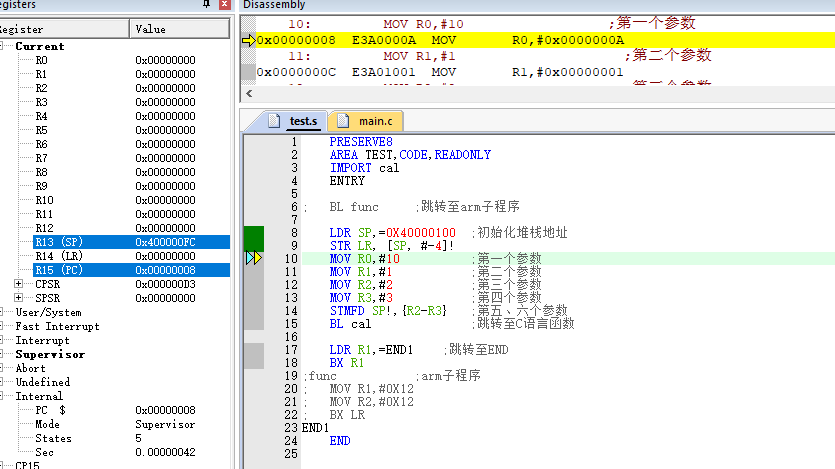


图2 程序执行完STR LR, [SP, #-4]!的界面截图

图3是程序即将调用C子程序的截图。

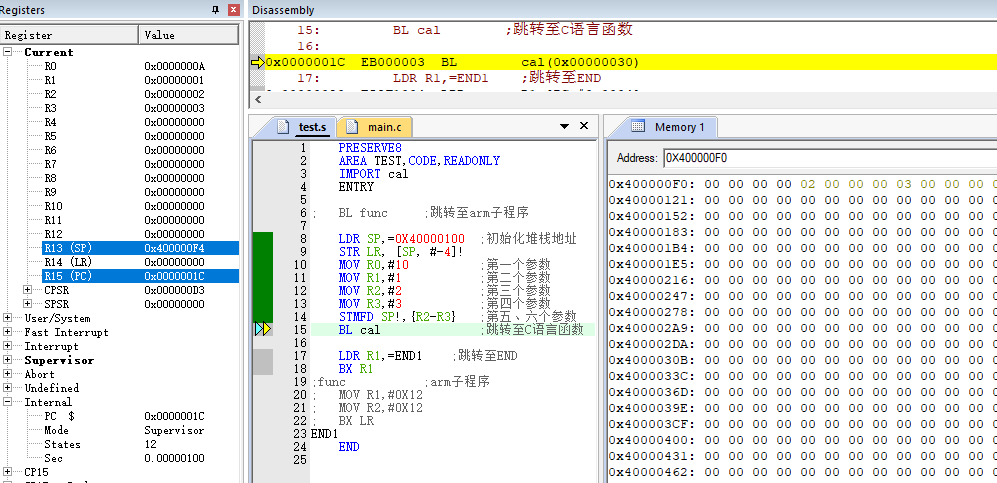


图3 程序即将调用C子程序的截图

图4是程序跟踪进入C语言子程序的截图。

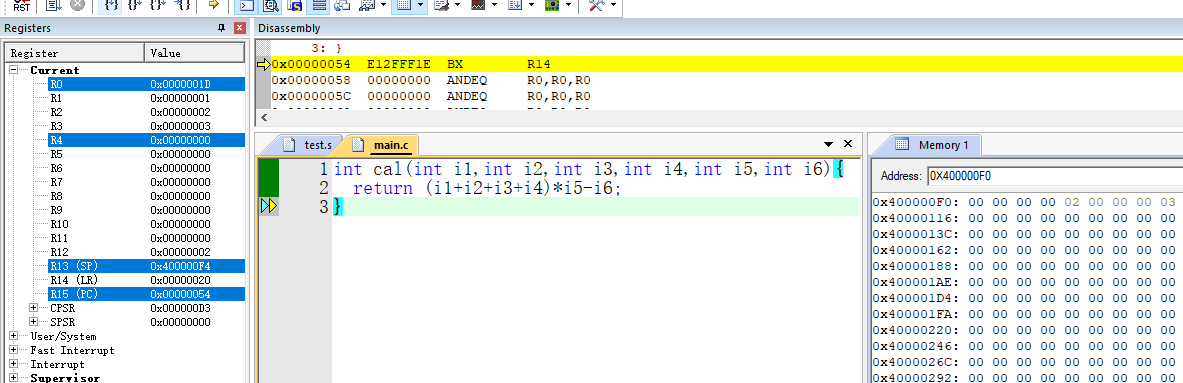


图4 程序跟踪进入C语言子程序的截图

1. **实验结论**

由图3可见，调用C语言子程序前将R0-R3以及堆栈数据设置好以便直接作为参数传入子函数，图4可见C语言子程序成功调用并计算出 (10+1+2+3)\*2-3=29=0X1D结果返回给R0。

ARM汇编调用C语言子程序时，前四个参数分别从R0、R1、R2、R3传入，本实验中为10、1、2、3，后续参数按满递减堆栈方式，SP指针从低地址到高地址，参数从左到右依次传入，本实验中为2、3，最终计算结果为(10+1+2+3)\*2-3=29=0X1D。

1. 总结及心得体会：

通过本实验的学习，我掌握了ARM汇编调用ARM子程序以及ARM汇编调用C语言子程序的方法，实践了解了参数的传递过程，同时对堆栈有了进一步的认识。

思考题

1. 观察调用C语言函数前SP寄存器的值以及C函数返回后，SP寄存器的值，看其是否有变化？

由图3图4可以看出，调用函数cal前后，sp的值保持为0x400000f4，前后一致。

原因：调用者负责清理通过堆栈传递的参数。然而，在此代码中，调用后未显式调整SP，程序直接结束，未进一步操作堆栈，SP的值在调用前后未发生改变。

1. 说明ARM汇编程序中，参数多余4个（比如为6个）时，参数是如何压栈传递的？通过C语言对应的汇编程序，分析C语言子程序是如何取得相关传入参数的？

超过4个时，前四个参数通过寄存器R0-R3传递，第五和第六个参数通过堆栈传递。在汇编代码中，调用者使用STMFD SP!, {R2-R3}将参数压栈，其中R2对应第五个参数（i5），R3对应第六个参数（i6）。由于堆栈是向下增长的，STMFD指令将R2存入较高地址（参数i5），R3存入较低地址（参数i6）。在C函数中，通过堆栈指针的偏移访问这些参数：参数i5位于[SP]，i6位于[SP, #4]。尽管存储顺序看似逆序，但根据AAPCS，参数按顺序放置，编译器能正确解析偏移量以获取参数值。

1. 对本实验过程及方法、手段的改进建议：

根据实验过程，提出改建建议。

本实验中不存在函数调用嵌套的问题，故STR LR, [SP, #-4]!指令的调用显得比较多余，建议重新考虑其应用场景。

报告评分：

指导教师签字：

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：成棋伟 学号：2022130102013**

**指导教师： 兰刚**

**实验地点： 西304 实验时间： 2025.04.29**

1. 实验名称：C语言主程序调用ARM子程序
2. 实验学时：4
3. 实验目的：
4. 了解ARM应用程序框架。
5. 了解ARM汇编程序函数和C语言程序函数相互调用时，遵循的ATPCS标准；
6. 了解和掌握C语言程序调用ARM语言程序函数的基本方法；
7. 了解和掌握C语言程序调用ARM汇编程序函数的参数传递过程；
8. 掌握内联汇编和嵌入式汇编的编程方法。
9. 实验原理：
10. **ARM工程**

由于C语言便于理解，有大量的支持库，所以它是当前ARM程序设计所使用的主要编程语言。

对硬件系统的初始化、CPU状态设定、中断使能、主频设定 以及RAM控制参数初始化等C程序力所不能及的底层操作，还是要由汇编语言程序 来完成。

ARM工程 的各种源文件之间的关系，以及最后形成可执行文件的过程如下图1所示：

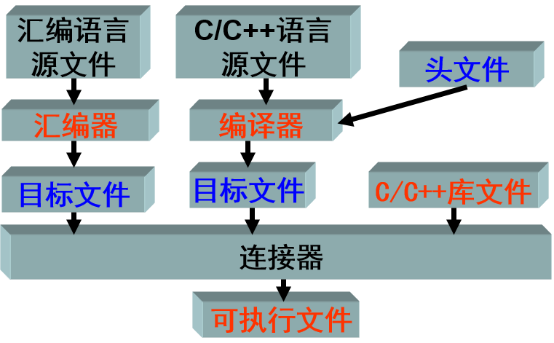


图1 汇编语言和C语言混合编译链接示意图

在应用系统的程序设计中，若所有的编程任务均用汇编语言 来完成，其工作量是可想而知的，这样做也不利于系统升级或应用软件移植。

通常汇编语言部分完成系统硬件的初始化；高级语言部分完成用户的应用。

执行时，首先执行初始化部分，然后再跳转到C/C++部分。整个程序结构显得清晰明了，容易理解。为方便工程开发，ARM公司的开发环境ARM ADS为用户提供了一个可以选用的应用程序框架。该框架把为用户程序做准备工作的程序分成了： 启动代码 和 应用程序初始化 两部分。

用于硬件初始化的汇编语言部分叫做 启动代码；用于应用程序初始化的C部分叫做初始化部分。整个程序如下图2所示：

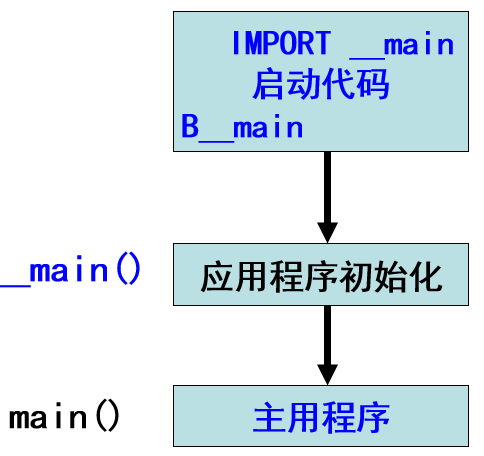


图2 ARM应用程序框架

1. **过程调用标准ATPCS**

在ARM工程中，C程序调用汇编函数和汇编程序调用C函数是经常发生的事情。为此人们制定了ARM-Thumb过程调用标准ATPCS（ARM-Thumb Procedure Call Standard）。

1. ATPCS规定，ARM的数据堆栈为FD型堆栈，即递减满堆栈。
2. ATPCS标准规定，对于参数个数不多于4的函数，编译器必须按参数在列表中的顺序，自左向右为它们分配寄存器R0~R3。其中函数返回时，R0还被用来存放函数的返回值。
3. 如果函数的参数多于4个，那么多余的参数则按自右向左的顺序压入数据堆栈，即参数入栈顺序与参数顺序相反。
4. 根据ATPCS的C语言程序调用汇编函数，参数由左向右依次传递给寄存器R0~R3的规则。
5. **C/C++语言和汇编语言的混合编程之:** **内联汇编和嵌入式汇编**

除了上面介绍的函数调用方法之外，ARM 编译器 armcc 中含有的内嵌汇编器还允许在 C 程序中 内联或嵌入式汇编代码，以提高程序的效率。

所谓内联汇编程序，就是在 C 程序中直接编写汇编程序段而形成一个语句块，这个语句块可以使用除了 BX 和 BLX之外的全部ARM指令来编写，从而可以使程序实现一些不能从C获得的底层功能。

其格式为：

\_\_asm

{

汇编语句块

}

内联汇编 与 真实汇编 之间有很大区别，会受到很多限制。

（1）它不支持 Thumb 指令；除了程序状态寄存器 PSR 之外，不能直接访问其他任何物理寄存器等；

（2）如果在内联汇编程序指令中出现了以某个寄存器名称命名的操作数，那么它被叫做虚拟寄存器，而不是实际的物理寄存器。编译器在生成和优化代码的过程中，会给每个虚拟寄存器分配实际的物理寄存器，但这个物理寄存器可能与在指令中指定的不同。唯一的一个例外就是状态寄存器 PSR ，任何对 PSR 的引用总是执行指向物理 PSR；

（3）在内联汇编代码中不能使用寄存器 PC（R15）、LR（R14）和SP（R13），任何试图使用这些寄存器的操作都会导致出现错误消息；

（4）虽然内联汇编代码可以更改处理器模式，但更改处理器模式会禁止使用 C 操作数或对已编译 C 代码的调用，直到将处理器模式恢复为原设置之后。

嵌入式汇编

嵌入式汇编程序是一个编写在C程序外的单独汇编程序，该程序段可以像函数那样被 C 程序调用。

与内联汇编不同，嵌入式汇编具有真实汇编的所有特性，数据交换符合 ATPCS 标准，同时支持 ARM 和Thumb，所以它可以对目标处理器进行不受限制的低级访问。但是不能直接引用 C/C++ 的变量。

用 \_ \_asm 声明的嵌入式汇编程序，像 C 函数那样，可以有参数和返回值。定义一个嵌入式汇编函数的语法格式为：

\_\_asm return–type function–name(parameter-list)

{

汇编程序段

}

灵活地使用内联汇编和嵌入式汇编，有助于提高程序效率。

1. 实验内容：
2. C语言主程序调用ARM指令子程序，程序的参数个数要求至少6个，ARM子程序实现的功能为：（i1+i2+i3+i4）\*i5-i6，并返回计算结果。观察参数是如何传递的。
3. 在C语言的main函数中，使用内联汇编编写一个函数，使得c语言中定义的3个int类型的变量(a,b,tmp)有如下的关系：tmp =(a+b)\*8。
4. 以嵌入式汇编的方式，编写一个子程序，实现字节序的转换。该子程序的函数的格式要求如下：int convertNum(int i);要求把输入的整数i（32位，四个字节顺序为ABCD），转换为另一种字节序（转换后的字节序为DCBA），返回结果为转换后的int值。在主程序中调用该子程序，并验证其结果的正确性。
5. 实验器材（设备、元器件）：
6. PC机一台；
7. Keil MDK-ARM uVision4开发工具。
8. 实验步骤：
9. 打开Keil MDK-ARM uVision4开发工具；
10. 新建一个工程文件；
11. 在新建的工程文件中，添加新的源程序文件
12. 编写代码
13. 选择“Build target”菜单对编写好的工程文件进行编译链接。
14. 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。
15. 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）
16. **程序代码**

C语言代码如代码4所示。

代码4 C语言主程序调用ARM指令子程序的C语言程序部分

|  |
| --- |
| extern int func(int i1,int i2,int i3,int i4,int i5,int i6);  \_\_asm int convertNum(int i){  MOV R1,#0 //寄存器清零  MOV R2,#0  MOV R3,#0  MOV R4,#0  AND R1,R0,#0XFF //0-7位放入R1  MOV R0,R0,LSR #8 //右移8位  AND R2,R0,#0XFF //8-15位放入R2  MOV R0,R0,LSR #8 //右移8位  AND R3,R0,#0XFF //16-23位放入R3  MOV R0,R0,LSR #8 //右移8位  AND R4,R0,#0XFF //24-31位放入R4  MOV R0,R0,LSR #8 //右移8位    ORR R0,R0,R1 //R1放入低8位  MOV R0,R0,LSL #8 //左移8位  ORR R0,R0,R2 //R2放入低8位  MOV R0,R0,LSL #8 //左移8位  ORR R0,R0,R3 //R3放入低8位  MOV R0,R0,LSL #8 //左移8位  ORR R0,R0,R4 //R4放入低8位  BX LR  }  int main(){  int i = func(1,2,3,4,5,6);  int a = 1,b=2,tmp=0;  \_\_asm{  ADD a,a,b  MOV b,8  MUL tmp,a,b  }  i = convertNum(4);  return 0;  } |

汇编程序代码如代码5所示。

代码5 C语言主程序调用ARM指令子程序的汇编程序部分

|  |
| --- |
| AREA TEST, CODE, READONLY  ENTRY  EXPORT func  func  STMFD SP!,{R4,R5} ;将R4、R5保存  MOV R4,#8  ADD SP,SP,R4 ;SP指针回到传入参数处  ADD R4,R0,R1 ;i1+i2->R4  ADD R5,R2,R3 ;i3+i4->R5  ADD R0,R4,R5 ;i1+i2+i3+i4->R0  LDMFD SP!,{R1,R2} ;i5->R1,i6->R2  MOV R4,#16  SUB SP,SP,R4 ;SP指针回到R4、R5处  LDMFD SP!,{R4,R5} ;恢复R4、R5  MUL R3,R0,R1 ;(i1+i2+i3+i4)\*i5->R3  SUB R0,R3,R2 ;(i1+i2+i3+i4)\*i5-i6->R0    BX LR  END |

1. **运行过程及结果界面截图**

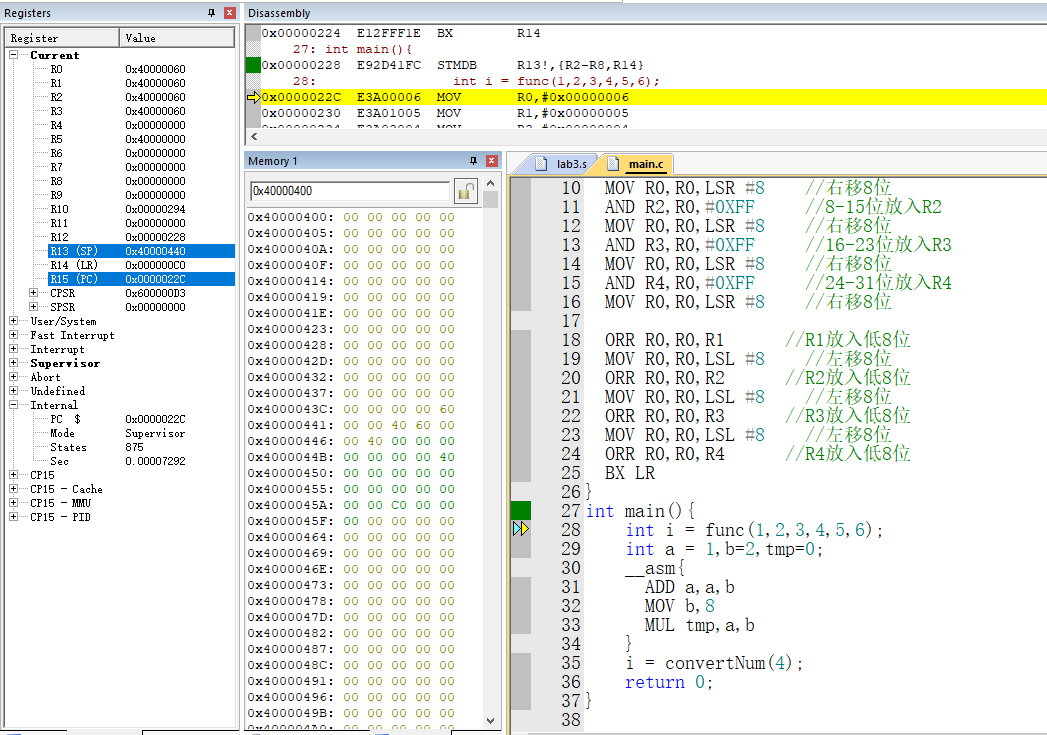


图5 程序即将调用汇编子程序Call\_Arm

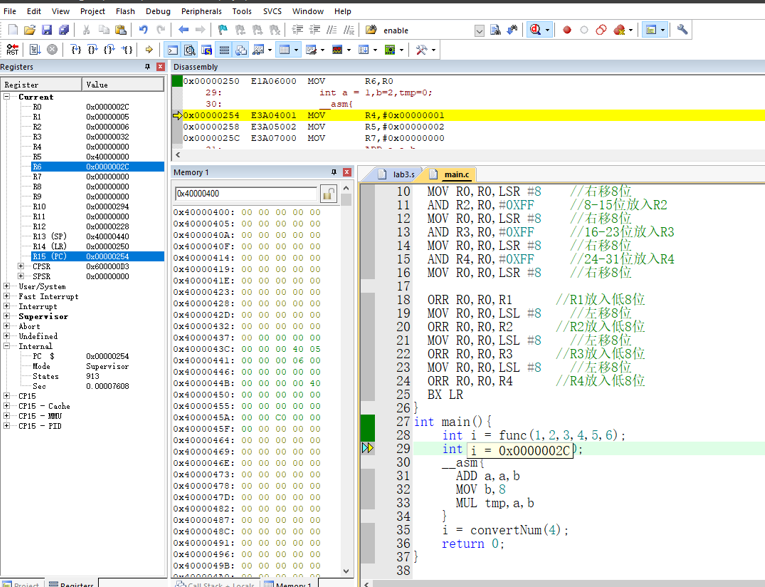


图6 程序从汇编子程序Call\_Arm中返回

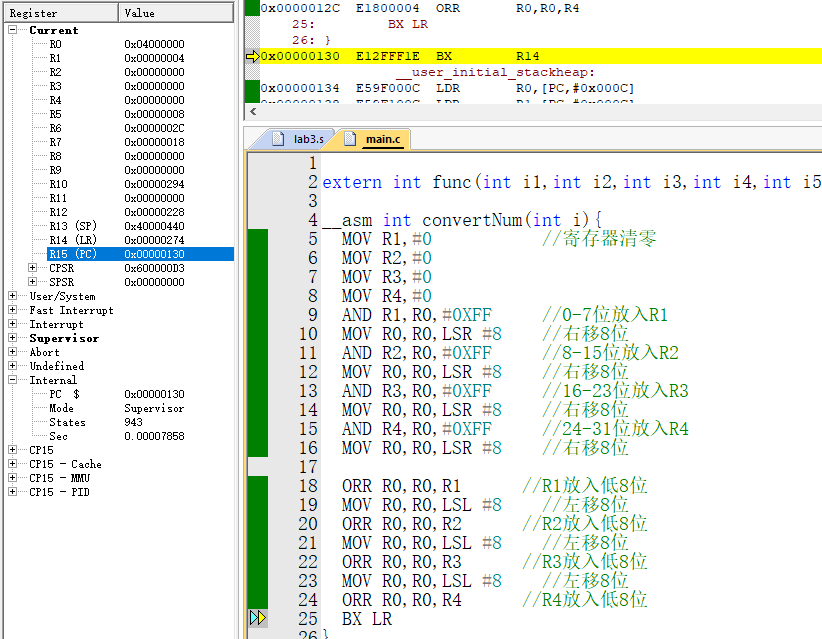


图7 程序跟踪进入汇编子程序Call\_Arm中

**3、实验结论**

图4可看出汇编子程序成功计算传入参数(1+2+3+4)\*5-6=44=0X2C，图5可看出结果成功从R0返回给变量i，图6可看出内联汇编成功计算(a+b)\*8并传入tmp，图7可看出嵌入式汇编子程序成功将传入参数4实现字节序反转。

C语言调用ARM汇编函数的参数传递规则与ARM汇编调用C语言子程序恰好相反，R0-R3对应从左往右前4个参数，剩余参数从右往左按地址从高到底压入堆栈。

内联汇编无法直接使用寄存器，若出现寄存器名称则会被编译器认为是一个未定义的变量，但是ADD、MUL等指令可直接对变量进行计算，不需先传入寄存器。

嵌入式汇编类似于加了\_\_asm前缀的普通函数，使用汇编语言编写，与在汇编文件内编写并在C文件中调用效果相同。

1. 总结及心得体会：

本次实验我掌握了C语言调用ARM汇编子程序以及内联汇编、嵌入式汇编的基本使用方法，对汇编语言的使用有了进一步理解。同时我在同学的帮助下发现，Option for Target中C/C++一栏Optimization选项用来控制编译优化程度，如果优化程度过高则会导致内联汇编部分不被编译从而不执行。

**思考题**

通过实验，查看C语言对应的汇编程序段，分析C语言调用ARM子程序时，在输入参数多余4个（比如是6个）的情况下，C语言程序汇编后对应的汇编程序段是如何实现参数传递的？结果是如何返回的？ARM汇编子程序是如何取得相关传入参数的？

参数传递：前4个参数通过 R0-R3 传递，其余参数按 从右向左顺序 压栈。

堆栈中参数的地址为 SP+0（第5个参数）、SP+4（第6个参数）等。

结果通过R0寄存器返回。

堆栈中的参数通过 SP偏移量 访问，需根据子程序入口处的压栈操作调整偏移量。

1. 对本实验过程及方法、手段的改进建议：

本实验过程清晰有条理，方法恰当，可以有效验证C语言中使用的ARM内联汇编和嵌入式汇编，暂无改进建议。

报告评分：

指导教师签字：