浙江工业大学

实验报告

课程：嵌入式系统 A

第二次实验

姓　　名 凌智城

学　　号 201806061211

专业班级 通信工程 1803 班

老　　师 黄国兴

学　　院 信息工程学院

提交日期 2021 年 5 月 12 日

实验 6：C 语言程序实验

一、 实验目的

了解使用 ADS1.2 编写 C 语言程序并进行调试。

二、 实验内容

编写一个汇编程序文件和一个 C 程序文件。汇编程序的功能是初始化堆栈指针和初始化 C 程序的运行环境，然后跳转到 C 程序运行，这就是一个简答的启动程序。C 程序使用加法运算来计算 1+2+3+…+(N-1)+N 的值(N>0)。

三、 实验步骤

步骤 1：正确连接实验箱及 PC 机

将 Multi-ICE 的一端与 PC 机 USB 口正确链接，另一端与开发板正确连接。将开发板的电源线正确连接，插上电源、串口线，先不要打开开发板的电源开关。

步骤 2：建立工程

启动 CodeWarrior for ARM Developer Suite，选择 File->New 命令，使用 ARM Executable Image 工程模板建立一个工程，工程名称为 ProgramC。

步骤 3：在工程中创建一个新文件

选择 File→New 命令，建立一个新文件 Test.c 和 Startup.S，直接添加到项目中，输入代码并保存。

步骤 4：设置地址和起始代码段

选择 Edit→DebugRel Setting 命令，在 DebugRel Setting 的对话框的左侧选择 ARM Linker 选项，然后再 Output 选项卡中设置链接地址，RO Base 为 0x4000000，RW Base 为 0x4003000；在 Options 选项卡中设置调试入口地址 Image entry point 为 0x4000000；在 Layout 选项卡中设置位于开始位置的起始代码段，如图 6-1 所示。

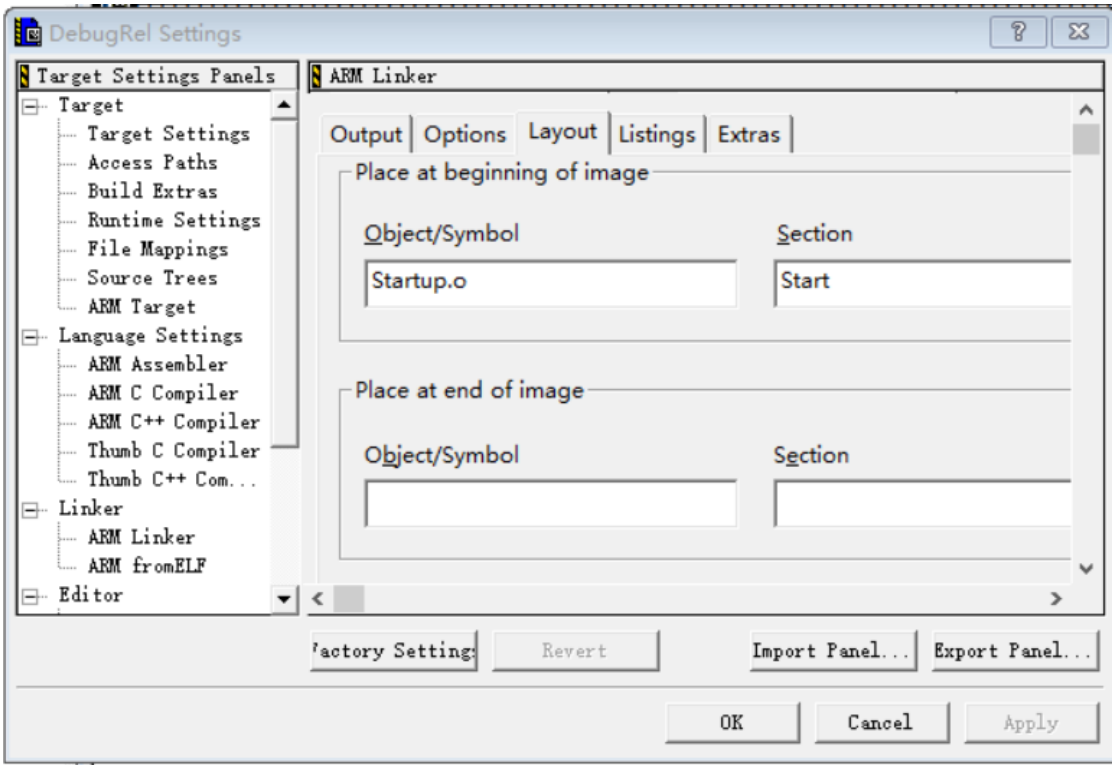

图 6-1 ARM Linker 选项

步骤 5：编译工程并仿真调试

选择 Project→Make 命令，将编译、链接整个工程，代码及错误和警告对话框如图 6-2 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

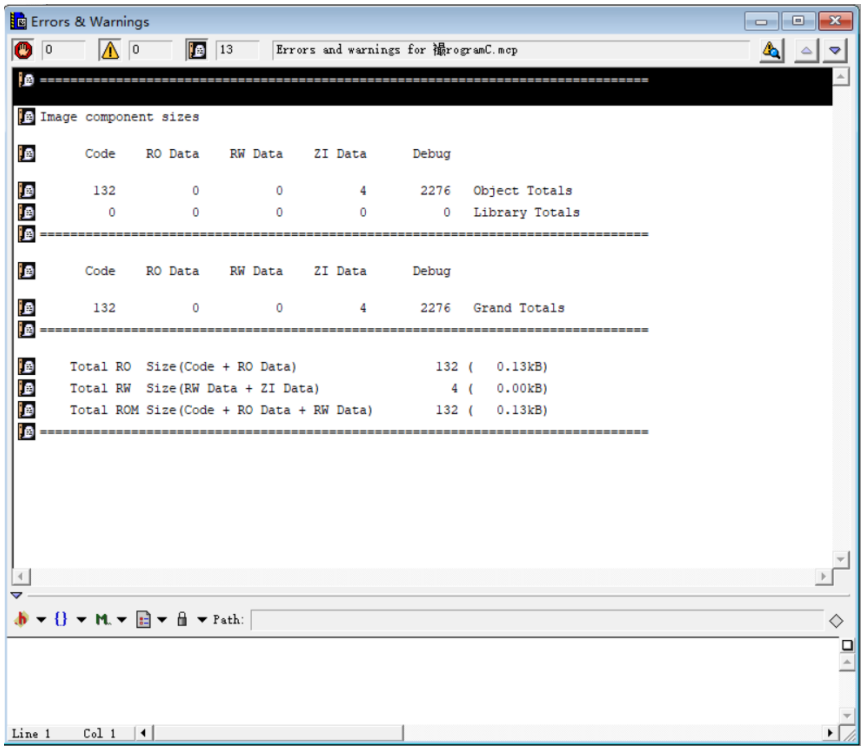

图 6-2 映像文件数据

步骤 6：设置断点并运行程序

在 Startup.S 的 B Main 处设置断点，然后元素运行程序，如图 6-3 所示。

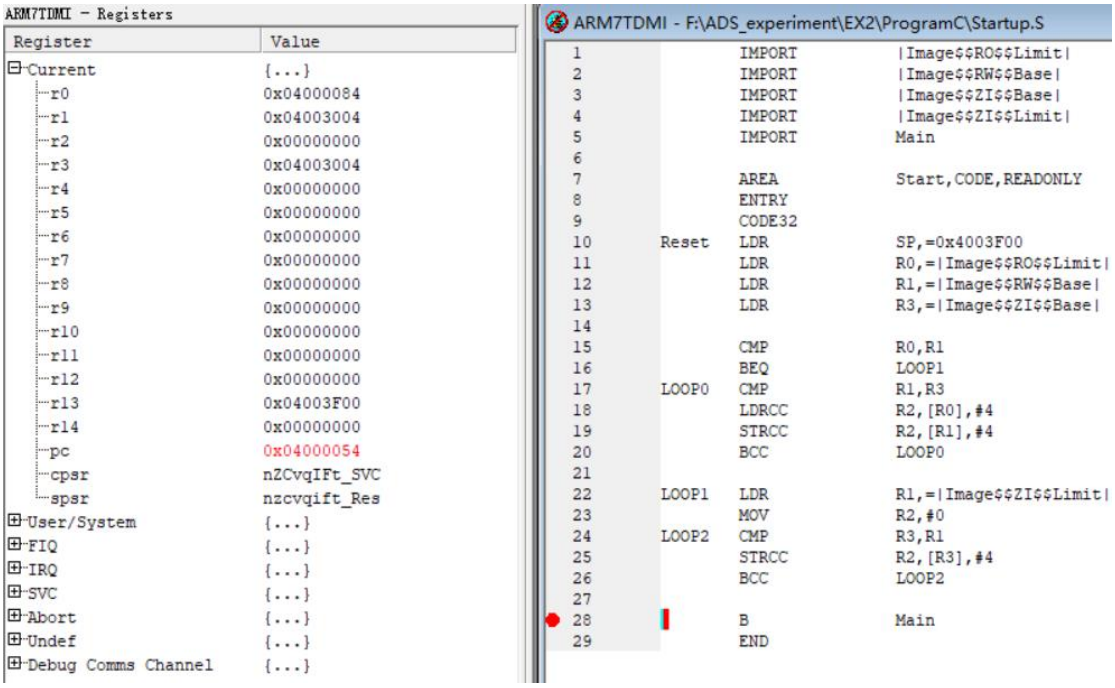

图 6-3 程序全速运行图

步骤 7：单步运行程序

程序在断点处停止。单步运行程序，判断程序是否跳转到 C 程序运行，如图 6-4 所示。

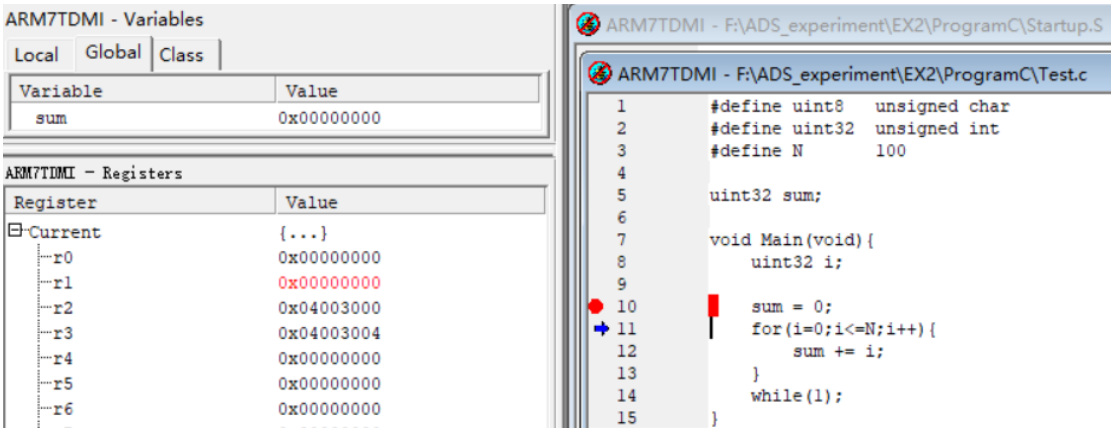

图 6-4 程序单步运行图

步骤 8：观察全局变量的值，判断程序的运算结果是否正确

选择 Processor View→Variables 命令，打开变量观察窗口，观察全局变量的值，单步或全速运行程序，判断程序的运行结果是否正确（也可以选择在 while(1) 处设置断点后再全速运行），如图 6-5 所示。

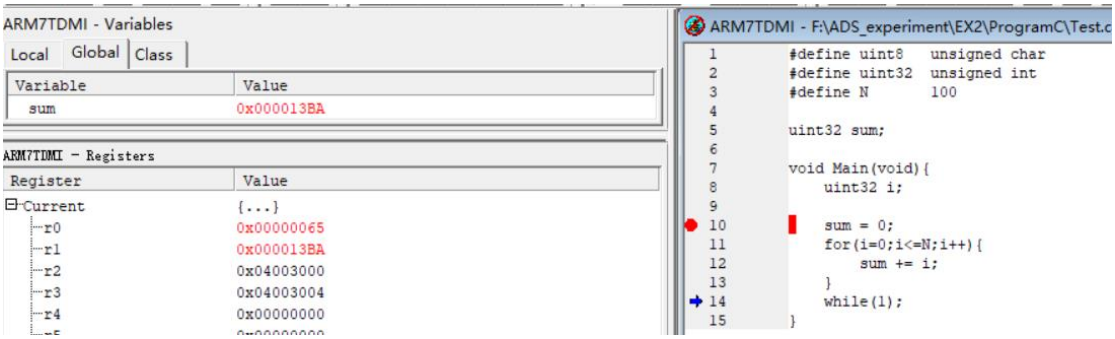

图 6-5 程序变量运行图（0x000013BA=5050）

四、 心得与体会

1. RO（readonly）是程序中的指令和常量，RW（read/write）是程序中的已初始化变量，ZI（zero）是程序中的未初始化的变量。
2. Total ROM Size（Code + RO Data + RW Data）
3. |Image$$RO$$Base| = RO Base

表示 RO 输出段运行时起始地址，也可以说是程序代码存放的起始地址。

4. |Image$$RO$$Limit| = RO Base + Total RO Size（Code + RO Data）

表示 RO 输出段运行时存储区域界限。

5. |Image$$RW$$Base| = RW Base

表示 RW 输出段运行时起始地址，而不一定是加载时的存放地址，因为 RW 输出段在加载时可能是在 ROM 中并紧跟着 RO 输出段存放的，当程序运行时才移动到 RAM 起始地址为|Image$$RW$$Base|的区域，由 RW Base 这个参数指定；未指定的话，默认紧跟 RO 输出段，那么|Image$$RW$$Base|＝|Image$$RO$$Limit|。

6. |Image$$RW$$Limit| = RW Base + Total RW Size（RW Data + ZI Data）

表示 RW 输出段运行时存储区域界限。

7. |Image$$ZI$$Base| = |Image$$RW$$Base| + RW Data

表示 ZI 输出段运行时起始地址。

8. |Image$$ZI$$Limit| = |Image$$ZI$$Base| + ZI Data

表示 ZI 输出段运行时存储区域界限。

9. 汇编程序调用 C 程序需要先 IMPORT［函数名］

五、 附录

Startup.S

```
        IMPORT          |Image$$RO$$Limit|      ; 引入其他源文件的定义，区分大小写
        IMPORT          |Image$$RW$$Base|
        IMPORT          |Image$$RW$$Limit|
        IMPORT          |Image$$ZI$$Base|
        IMPORT          |Image$$ZI$$Limit|
        IMPORT          Main                    ; 引入 C 程序中 Main 函数的定义

        AREA            Start,CODE,READONLY     ; 声明一段只读代码，名字为 Start
        ENTRY                                   ; 指定程序的入口点
        CODE32                                  ; 32 位的 ARM 指令

Reset   LDR             SP,=0x4003F00           ; 将#0x04003F00 加载到 SP
        LDR             R0,=|Image$$RO$$Limit|  ; 将#0x04000084 加载到 R0
        LDR             R1,=|Image$$RW$$Base|   ; 将#0x04003000 加载到 R1
        LDR             R3,=|Image$$ZI$$Base|   ; 将#0x04003000 加载到 R3

        CMP             R0,R1                   ; 减法比较 R0 和 R1 的值,R0-R1 为负数
        BEQ             LOOP1                   ; 上条指令 N=1,Z=0,此处不跳转
LOOP0   CMP             R1,R3                   ; 减法比较 R1 和 R3 的值,R1-R3=0
        LDRCC           R2,[R0],#4              ; 上条指令 Z=1,N=0,C=1,CC 条件是 C=0 不执行
        STRCC           R2,[R1],#4              ; 保证 Image$$RW$$Base≥Image$$ZI$$Base
        BCC             LOOP0                   ; 同上条

LOOP1   LDR             R1,=|Image$$ZI$$Limit|  ; 将#0x04003004 加载到 R3
        MOV             R2,#0                   ; 0->R2
LOOP2   CMP             R3,R1                   ; 减法比较 R3 和 R1 的值,R3-R1<0，产生借
位,N=1,Z=0,C=0
        STRCC           R2,[R3],#4              ; CC 条件为 C=0 即 R3 比 R1 小，则 R3+4 写回 R3
        BCC             LOOP2                   ; 满足 CC 条件，再来一遍 LOOP2 知道 R3=R1

        B               Main                    ; 跳转到 C 程序的 Main 函数继续执行
        END                                     ; 指示本源程序结束
```

Test.c

```
#define uint8   unsigned char   // 宏定义
#define uint32  unsigned int    // 宏定义
#define N       100             // 宏定义
```

```
uint32 sum;                     // 声明全局变量

void Main(void){
    uint32 i;                   // 声明无符 int 型 i

    sum = 0;
    for(i=0;i<=N;i++){          // for 循环实现累加
        sum += i;
    }
    while(1);                   // 原地等待
}
```

调用 C 语言实现两数相减

一、 实验目的

掌握在 C 语言程序中条用汇编程序，了解 ATPCS 基本规则

二、 实验内容

在 C 程序中调用汇编子程序，实现两个整数的减法运算。汇编子程序的原型为 uint32 Sub(uint32 x, uint32 y)。

其中，uint32 已定义为 unsigned int。

三、 实验步骤

步骤 1：正确连接实验箱及 PC 机

将 Multi-ICE 的一端与 PC 机 USB 口正确链接，另一端与开发板正确连接。将开发板的电源线正确连接，插上电源、串口线，先不要打开开发板的电源开关。

步骤 2：建立工程

启动 CodeWarrior for ARM Developer Suite，选择 File->New 命令，使用 ARM Executable Image 工程模板建立一个工程，工程名称为 ProgramC2。

步骤 3：在工程中创建一个新文件

选择 File→New 命令，建立一个新文件 Startup.S、Sub.S 和 Test.c，设置直接添加到项目中，输入代码并保存，其中 Startup.S 程序同上一实验。

步骤 4：设置地址和起始代码段

选择 Edit→DebugRel Setting 命令，在 DebugRel Setting 的对话框的左侧选择 ARM Linker 选项，然后再 Output 选项卡中设置链接地址，RO Base 为 0x4000000，RW Base 为 0x4003000；在 Options 选项卡中设置调试入口地址 Image entry point 为 0x4000000；在 Layout 选项卡中设置位于开始位置的起始代码段设置为 Startup.o 的 Start 段。

步骤 5：编译工程并仿真调试

选择 Project→Make 命令，将编译、链接整个工程，代码及错误和警告对话框如图 1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

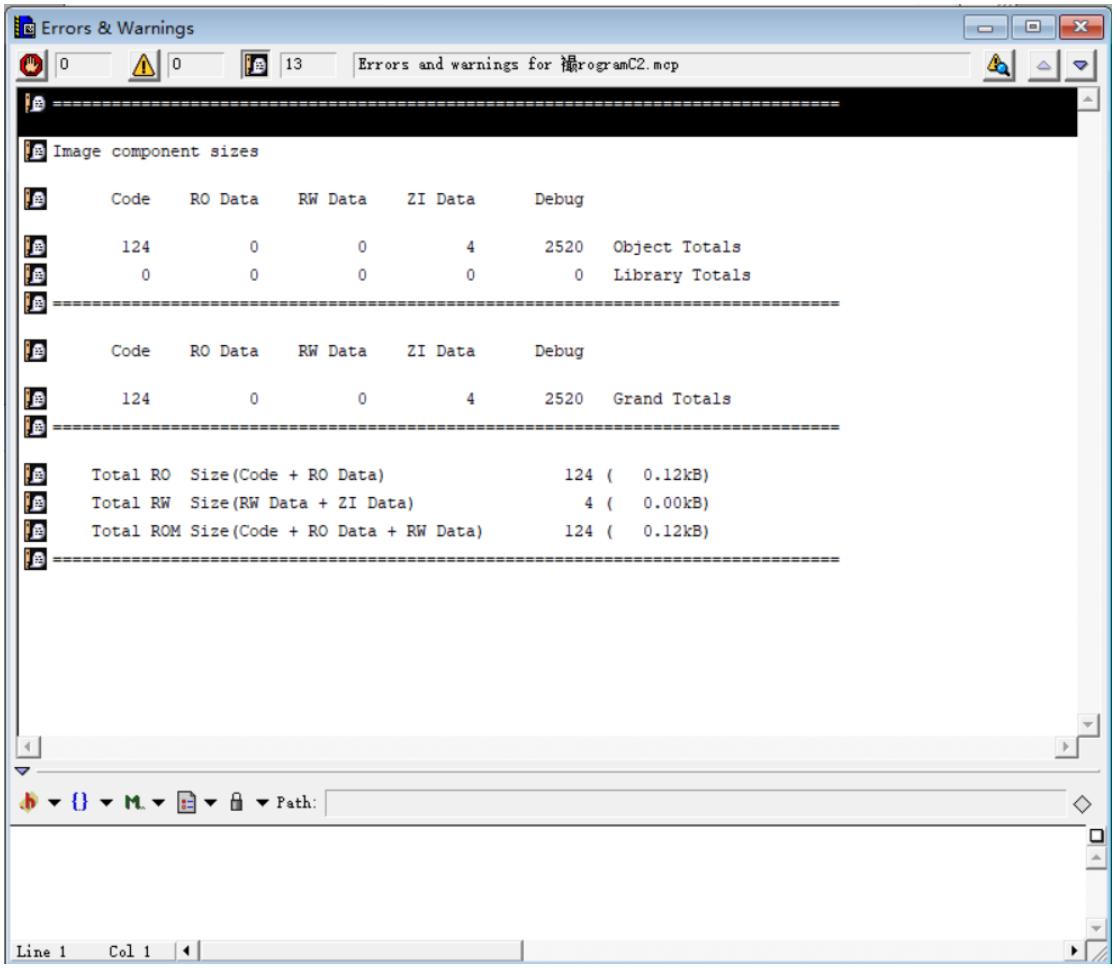

图 1 映像文件数据

步骤 6：设置断点并运行程序

在 Startup.o 的 B Main 处设置断点，然后全速运行程序。单步运行跳至 Test.c 文件，运行至 sum=Sub()处，再单步运行，观察程序是否跳转到汇编程序 Sub.S，如图 2 所示。

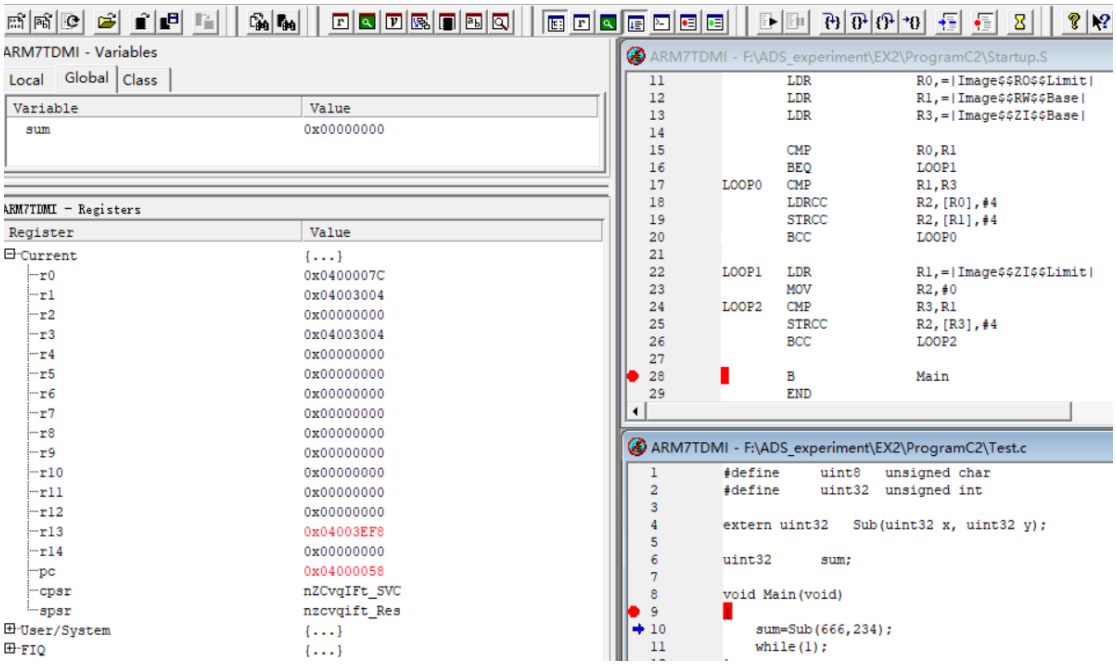

图 2 程序运行图

步骤 7：观察全局变量的值，判断程序的运算结果是否正确

选择 Processor View→Variables 命令，打开变量观察窗口，观察全局变量的值，单步运行程序，判断程序的运算结果是否正确，如图 3、图 4 所示。

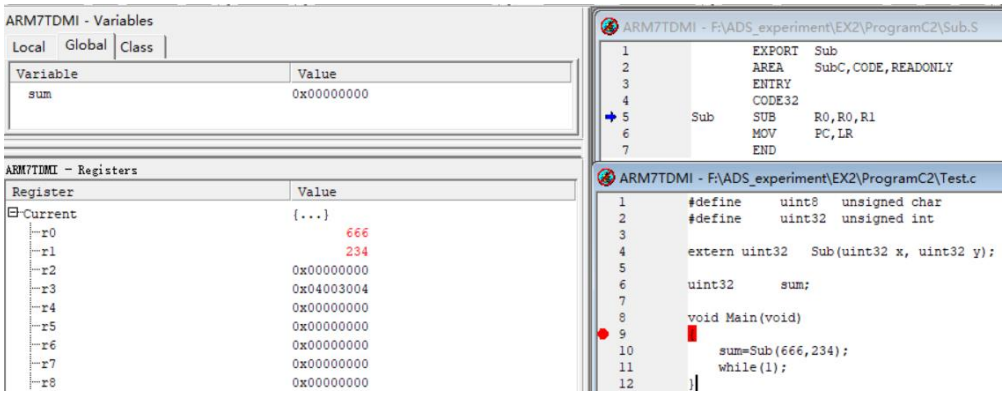

图 3 程序变量运行图 1

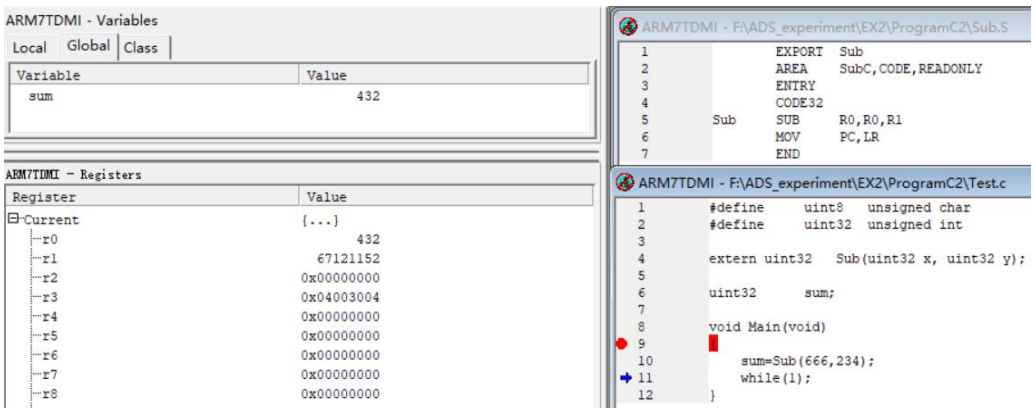

图 4 程序变量运行图 2

四、 心得与体会

在从 C 语言程序跳转到汇编语言程序时，会自动将 C 语言程序的下一条指令地址存放在在 R14（LR），在结束汇编语言程序的时候记得要将 LR 传送给 PC，以保证程序能正确返回。测试数据 666-234=432 正确。

通过本次实验熟悉和掌握 ARM 汇编指令中的常用指令及 ADS1.2 开发环境的使用方法，对 ARM 汇编语言和 C 语言混合编程的方式等知识点有了更为直观和充分的了解，对使用 ADS1.2 进行 ARM 软件编程开发有了更为深入的了解 。

五、 附录

Startup.S

```
        IMPORT          |Image$$RO$$Limit|      ; 引入其他源文件的定义，区分大小写
        IMPORT          |Image$$RW$$Base|
        IMPORT          |Image$$RW$$Limit|
        IMPORT          |Image$$ZI$$Base|
        IMPORT          |Image$$ZI$$Limit|
        IMPORT          Main                    ; 引入 C 程序中 Main 函数的定义

        AREA            Start,CODE,READONLY     ; 声明一段只读代码，名字为 Start
        ENTRY                                   ; 指定程序的入口点
        CODE32                                  ; 32 位的 ARM 指令

Reset   LDR             SP,=0x4003F00           ; 将#0x04003F00 加载到 SP
        LDR             R0,=|Image$$RO$$Limit|  ; 将#0x04000080 加载到 R0
        LDR             R1,=|Image$$RW$$Base|   ; 将#0x04003000 加载到 R1
        LDR             R3,=|Image$$ZI$$Base|   ; 将#0x04003000 加载到 R3

        CMP             R0,R1                   ; 减法比较 R0 和 R1 的值,R0-R1 为负数
        BEQ             LOOP1                   ; 上条指令 N=1,Z=0,此处不跳转
LOOP0   CMP             R1,R3                   ; 减法比较 R1 和 R3 的值,R1-R3=0
        LDRCC           R2,[R0],#4              ; 上条指令 Z=1,N=0,C=1,CC 条件是 C=0 不执行
        STRCC           R2,[R1],#4              ; 保证 Image$$RW$$Base≥Image$$ZI$$Base
        BCC             LOOP0                   ; 同上条

LOOP1   LDR             R1,=|Image$$ZI$$Limit|  ; 将#0x04003004 加载到 R3
        MOV             R2,#0                   ; 0->R2
```

```
LOOP2   CMP             R3,R1                   ; 减法比较 R3 和 R1 的值,R3-R1<0，产生借
位,N=1,Z=0,C=0
        STRCC           R2,[R3],#4              ; CC 条件为 C=0 即 R3 比 R1 小，则 R3+4 写回 R3
        BCC             LOOP2                   ; 满足 CC 条件，再来一遍 LOOP2 知道 R3=R1

        B               Main                    ; 跳转到 C 程序的 Main 函数继续执行
        END                                     ; 指示本源程序结束
```

Test.c

```c
#define     uint8   unsigned char   // 宏定义
#define     uint32  unsigned int    // 宏定义

extern uint32   Sub(uint32 x, uint32 y);    // 申明 uint32 型汇编程序 Add

uint32      sum;                    // 声明全局变量 sum

void Main(void)
{
    sum=Sub(666,234);               // 调用 Sub 汇编程序计算两数之差
    while(1);                       // 原地等待
}
```

Sub.S

```
        EXPORT  Sub                     ; 声明 Add 可以被其他文件引用，相当于全局
        AREA    SubC,CODE,READONLY      ; 声明一段只读代码，名字为 AddC
        ENTRY                           ; 指定程序的入口点
        CODE32                          ; 32 位的 ARM 指令
Sub     SUB     R0,R0,R1                ; R0=R0-R1
        MOV     PC,LR                   ; LR -> PC 用来返回程序
        END                             ; 指示本源程序结束
```

实验 7：C 语言程序实验

六、 实验目的

掌握在 C 语言程序中条用汇编程序，了解 ATPCS 基本规则

七、 实验内容

在 C 程序中调用汇编子程序，实现两个整数的加法运算。汇编子程序的原型为 uint32 Add(uint32 x, uint32 y)。

其中，uint32 已定义为 unsigned int。

八、 实验步骤

步骤 1：正确连接实验箱及 PC 机

将 Multi-ICE 的一端与 PC 机 USB 口正确链接，另一端与开发板正确连接。将开发板的电源线正确连接，插上电源、串口线，先不要打开开发板的电源开关。

步骤 2：建立工程

启动 CodeWarrior for ARM Developer Suite，选择 File->New 命令，使用 ARM Executable Image 工程模板建立一个工程，工程名称为 ProgramC1。

步骤 3：在工程中创建一个新文件

选择 File→New 命令，建立一个新文件 Startup.S、Add.S 和 Test.c，设置直接添加到项目中，输入代码并保存，其中 Startup.S 程序同上一实验。

步骤 4：设置地址和起始代码段

选择 Edit→DebugRel Setting 命令，在 DebugRel Setting 的对话框的左侧选择 ARM Linker 选项，然后再 Output 选项卡中设置链接地址，RO Base 为 0x4000000，RW Base 为 0x4003000；在 Options 选项卡中设置调试入口地址 Image entry point 为 0x4000000；在 Layout 选项卡中设置位于开始位置的起始代码段设置为 Startup.o 的 Start 段。

步骤 5：编译工程并仿真调试

选择 Project→Make 命令，将编译、链接整个工程，代码及错误和警告对话框如图 7-1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

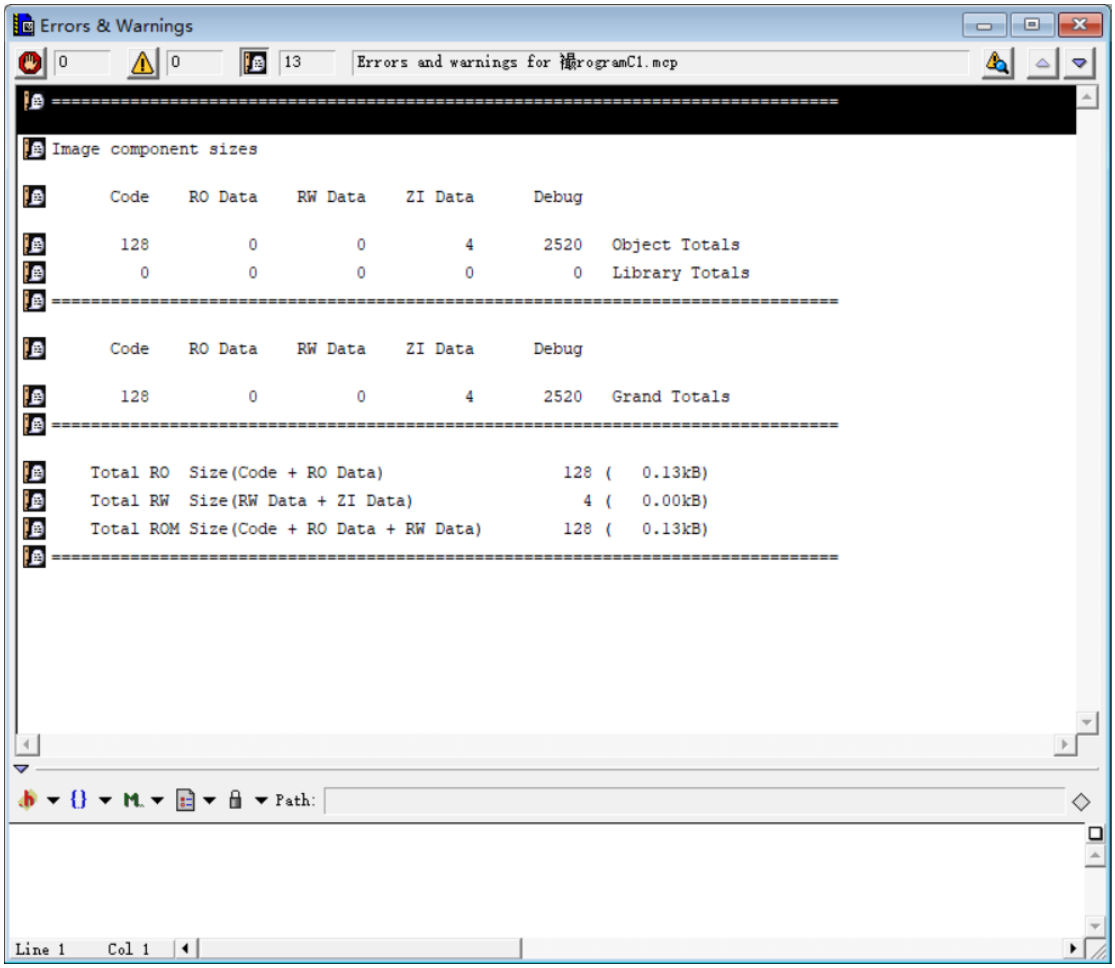

图 7-1 映像文件数据

步骤 6：设置断点并运行程序

在 Startup.o 的 B Main 处设置断点，然后全速运行程序。单步运行跳至 Test.c 文件，运行至 sum=Add()处，再单步运行，观察程序是否跳转到汇编程序 Add.S，如图 7-2 所示。

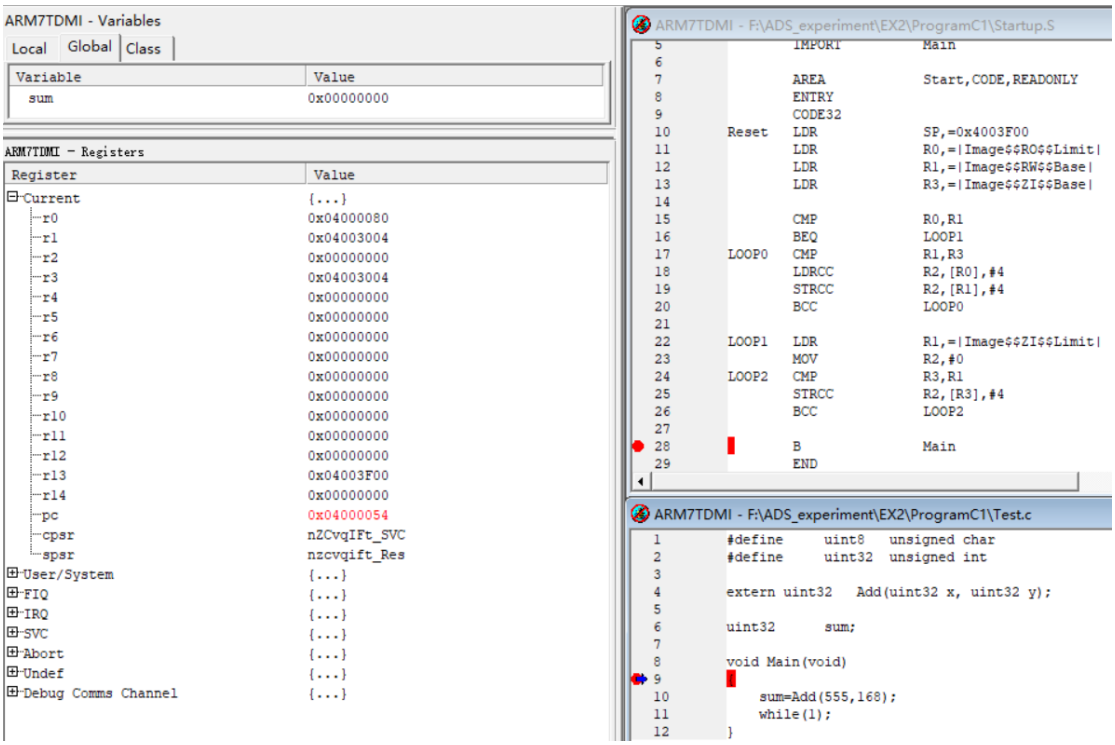

图 7-2 程序运行图

步骤 7：观察全局变量的值，判断程序的运算结果是否正确

选择 Processor View→Variables 命令，打开变量观察窗口，观察全局变量的值，单步运行程序，判断程序的运算结果是否正确，如图 7-3、图 7-4 所示。

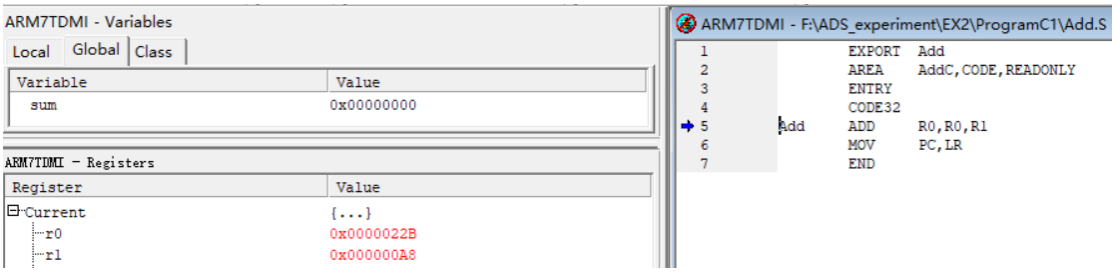

图 7-3 程序变量运行图 1

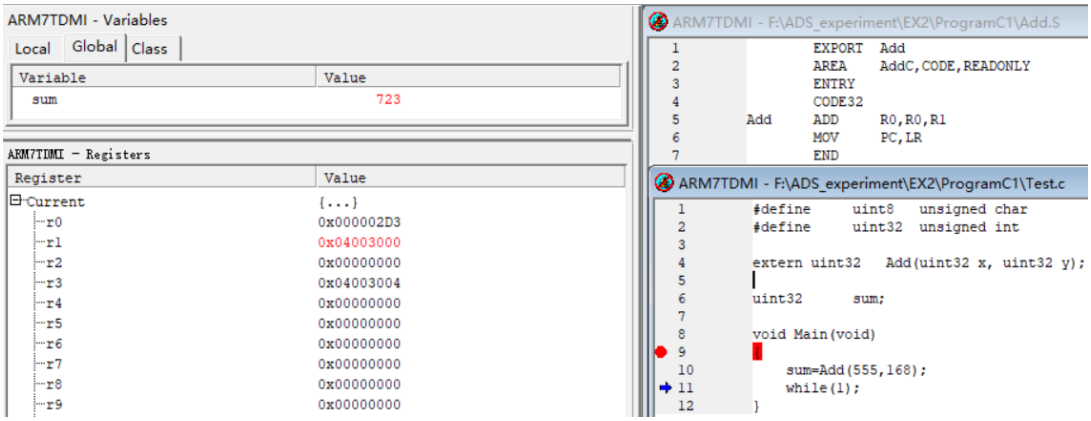

图 7-4 程序变量运行图 2

九、 心得与体会

在从 C 语言程序跳转到汇编语言程序时，会自动将 C 语言程序的下一条指令地址存放在在 R14（LR），在结束汇编语言程序的时候记得要将 LR 传送给 PC，以保证程序能正确返回。通过本次实验熟悉和掌握 ARM 汇编指令中的常用指令及 ADS1.2 开发环境的使用方法，对 ARM 汇编语言和 C 语言混合编程的方式等知识点有了更为直观和充分的了解，对使用 ADS1.2 进行 ARM 软件编程开发有了更为深入的了解 。

十、 附录

Startup.S

```
        IMPORT          |Image$$RO$$Limit|      ; 引入其他源文件的定义，区分大小写
        IMPORT          |Image$$RW$$Base|
        IMPORT          |Image$$RW$$Limit|
        IMPORT          |Image$$ZI$$Base|
        IMPORT          |Image$$ZI$$Limit|
        IMPORT          Main                    ; 引入 C 程序中 Main 函数的定义

        AREA            Start,CODE,READONLY     ; 声明一段只读代码，名字为 Start
        ENTRY                                   ; 指定程序的入口点
        CODE32                                  ; 32 位的 ARM 指令

Reset   LDR             SP,=0x4003F00           ; 将#0x04003F00 加载到 SP
        LDR             R0,=|Image$$RO$$Limit|  ; 将#0x04000080 加载到 R0
        LDR             R1,=|Image$$RW$$Base|   ; 将#0x04003000 加载到 R1
        LDR             R3,=|Image$$ZI$$Base|   ; 将#0x04003000 加载到 R3

        CMP             R0,R1                   ; 减法比较 R0 和 R1 的值,R0-R1 为负数
        BEQ             LOOP1                   ; 上条指令 N=1,Z=0,此处不跳转
LOOP0   CMP             R1,R3                   ; 减法比较 R1 和 R3 的值,R1-R3=0
        LDRCC           R2,[R0],#4              ; 上条指令 Z=1,N=0,C=1,CC 条件是 C=0 不执行
        STRCC           R2,[R1],#4              ; 保证 Image$$RW$$Base≥Image$$ZI$$Base
        BCC             LOOP0                   ; 同上条

LOOP1   LDR             R1,=|Image$$ZI$$Limit|  ; 将#0x04003004 加载到 R3
        MOV             R2,#0                   ; 0->R2
```

```
LOOP2   CMP             R3,R1                   ; 减法比较 R3 和 R1 的值,R3-R1<0，产生借
位,N=1,Z=0,C=0
        STRCC           R2,[R3],#4              ; CC 条件为 C=0 即 R3 比 R1 小，则 R3+4 写回 R3
        BCC             LOOP2                   ; 满足 CC 条件，再来一遍 LOOP2 知道 R3=R1

        B               Main                    ; 跳转到 C 程序的 Main 函数继续执行
        END                                     ; 指示本源程序结束
```

Test.c

```c
#define     uint8   unsigned char   // 宏定义
#define     uint32  unsigned int    // 宏定义

extern uint32   Add(uint32 x, uint32 y);    // 申明 uint32 型汇编程序 Add

uint32      sum;                    // 声明全局变量 sum

void Main(void)
{
    sum=Add(555,168);               // 调用 Add 汇编程序计算两数之和
    while(1);                       // 原地等待
}
```

Add.S

```
        EXPORT  Add                     ; 声明 Add 可以被其他文件引用，相当于全局
        AREA    AddC,CODE,READONLY      ; 声明一段只读代码，名字为 AddC
        ENTRY                           ; 指定程序的入口点
        CODE32                          ; 32 位的 ARM 指令
Add     ADD     R0,R0,R1                ; R0=R0+R1
        MOV     PC,LR                   ; LR -> PC 用来返回程序
        END                             ; 指示本源程序结束
```