浙江工业大学

实验报告

课程：嵌入式系统 A

第一次实验

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 9 日

实验 1：ADS1.2 集成开发环境练习

一、 实验目的

了解 ADS1.2 集成开发环境的使用方法。

二、 实验内容

1. 建立一个新的工程。
2. 建立一个 C 源文件，并添加到工程中。
3. 建立编译链接控制选项。
4. 编译链接工程。

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 ADS，如图 1-1 所示。

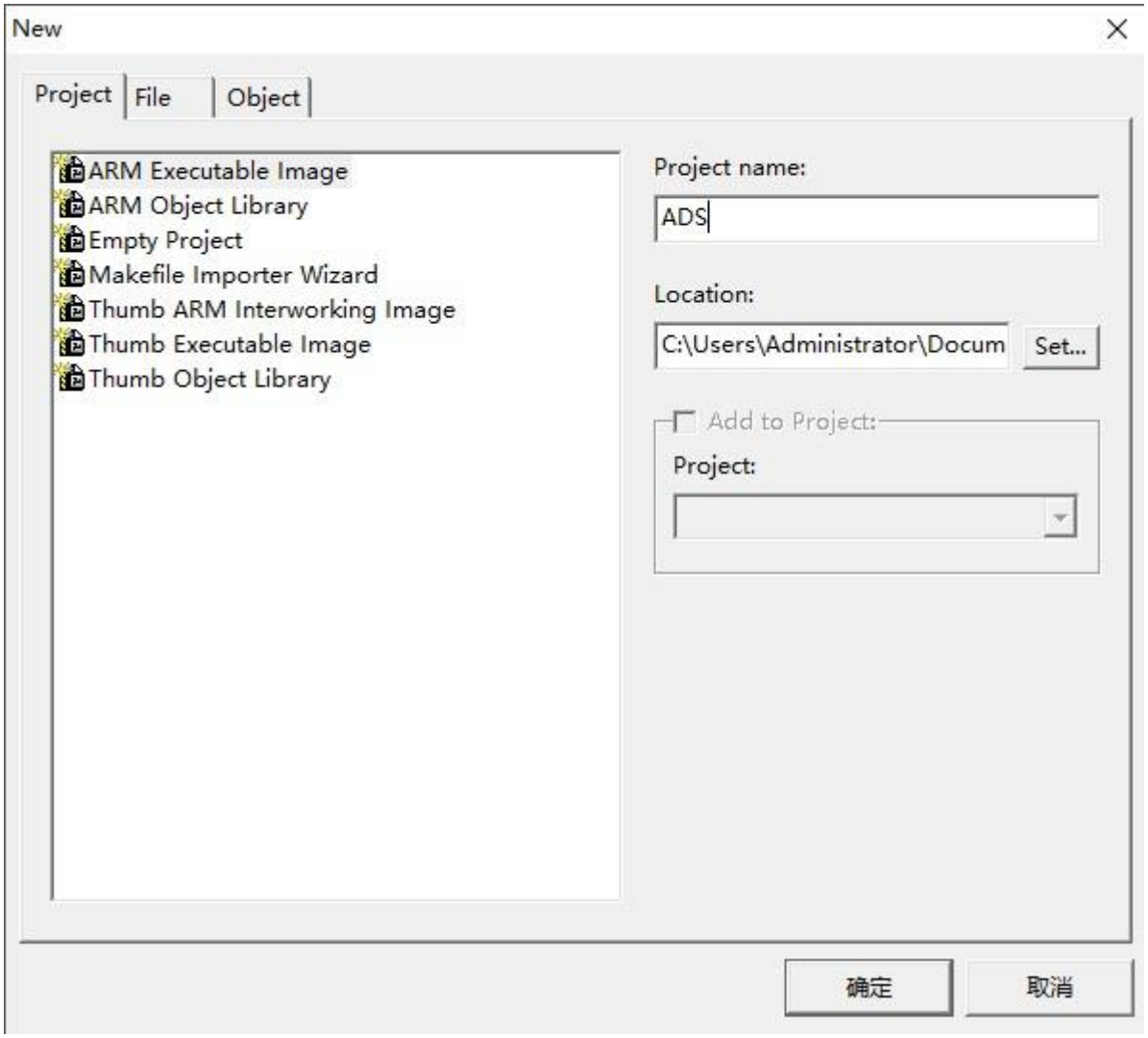

图 1-1 简历 ARM 指令代码的工程

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 TEST1.S，直接添加到项目中，如图 1-2 所示。输入代码并保存，添加了 TEST1.S 的工程管理窗口如图 1-3 所示。

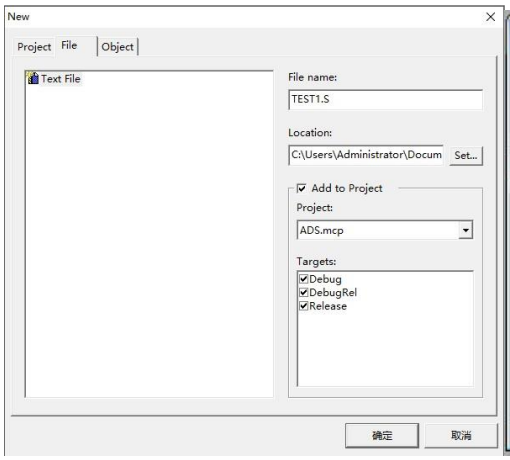

图 1-2 新建文件 TEST1.S

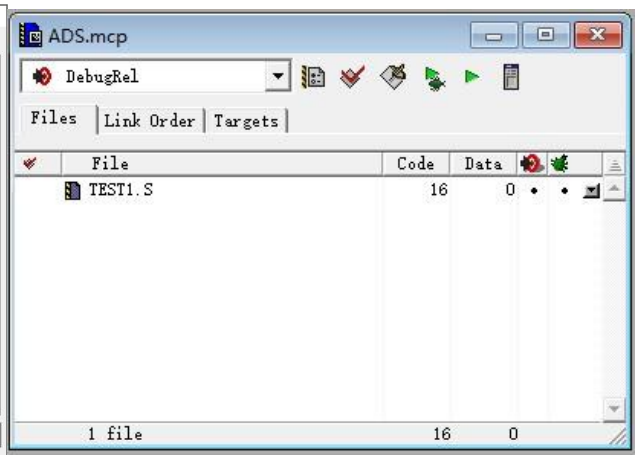

图 1-3 添加了 TEST1.S 的工程管理窗口

步骤 4：设置地址

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，如图 1-4 所示；在 Options 选项卡中设置调试入口地址，如图 1-5 所示。

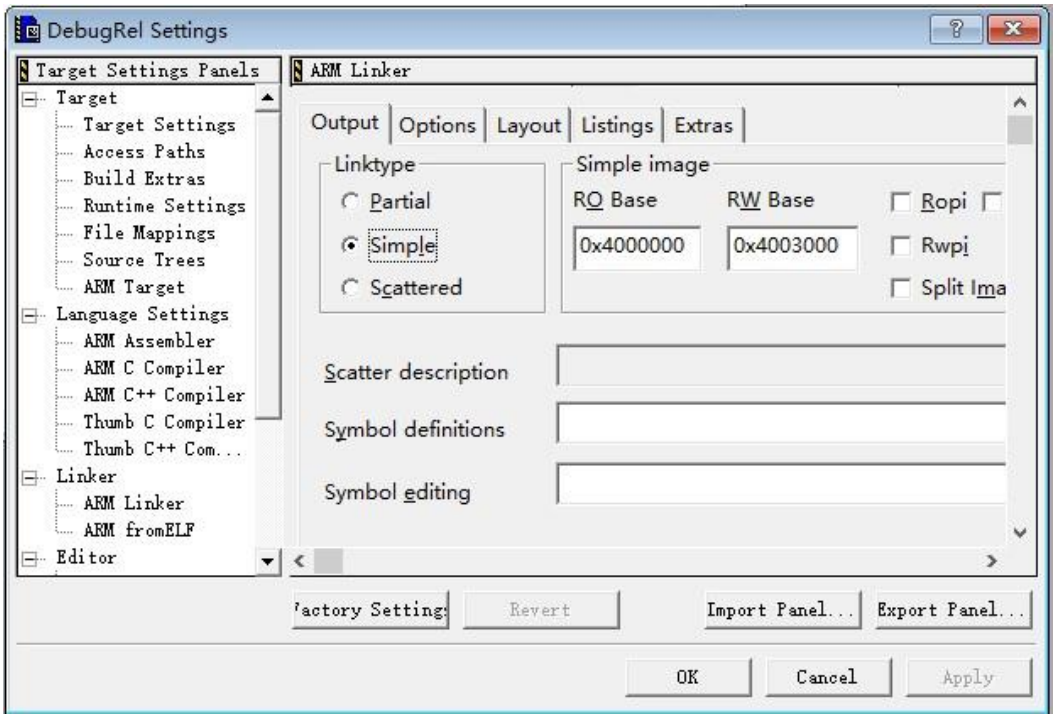

图 1-4 工程链接地址设置

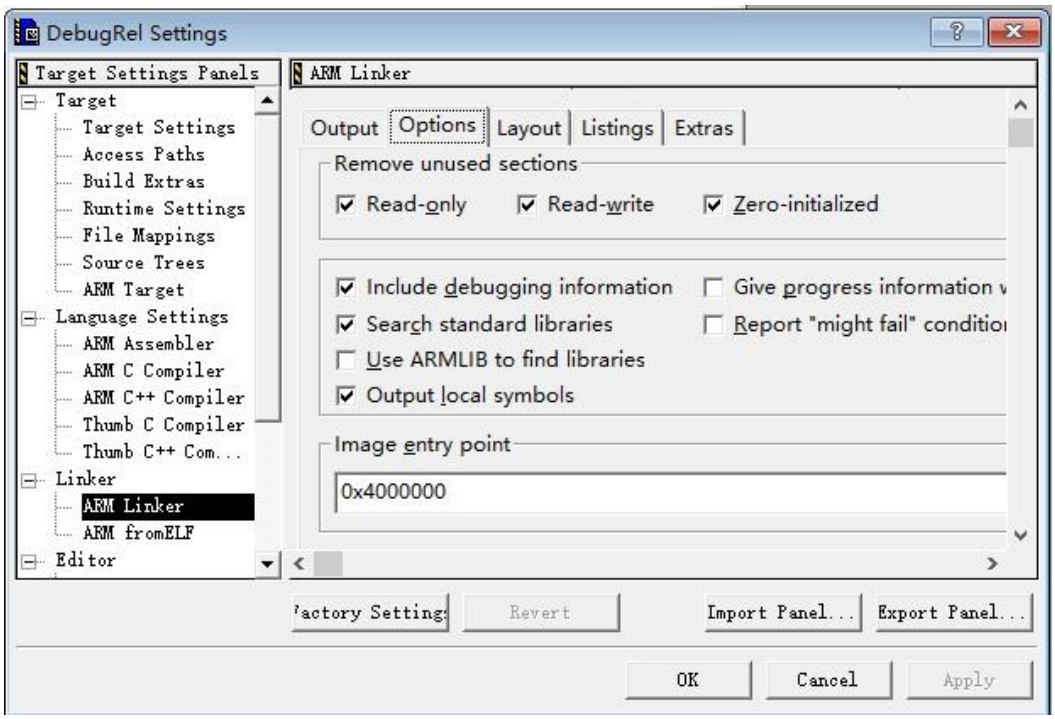

图 1-5 工程调试入口地址设置

步骤 5：编译工程

选择 Project→Make 命令，将编译、链接整个工程。如图 1-6 所示编译错误与警告对话框。

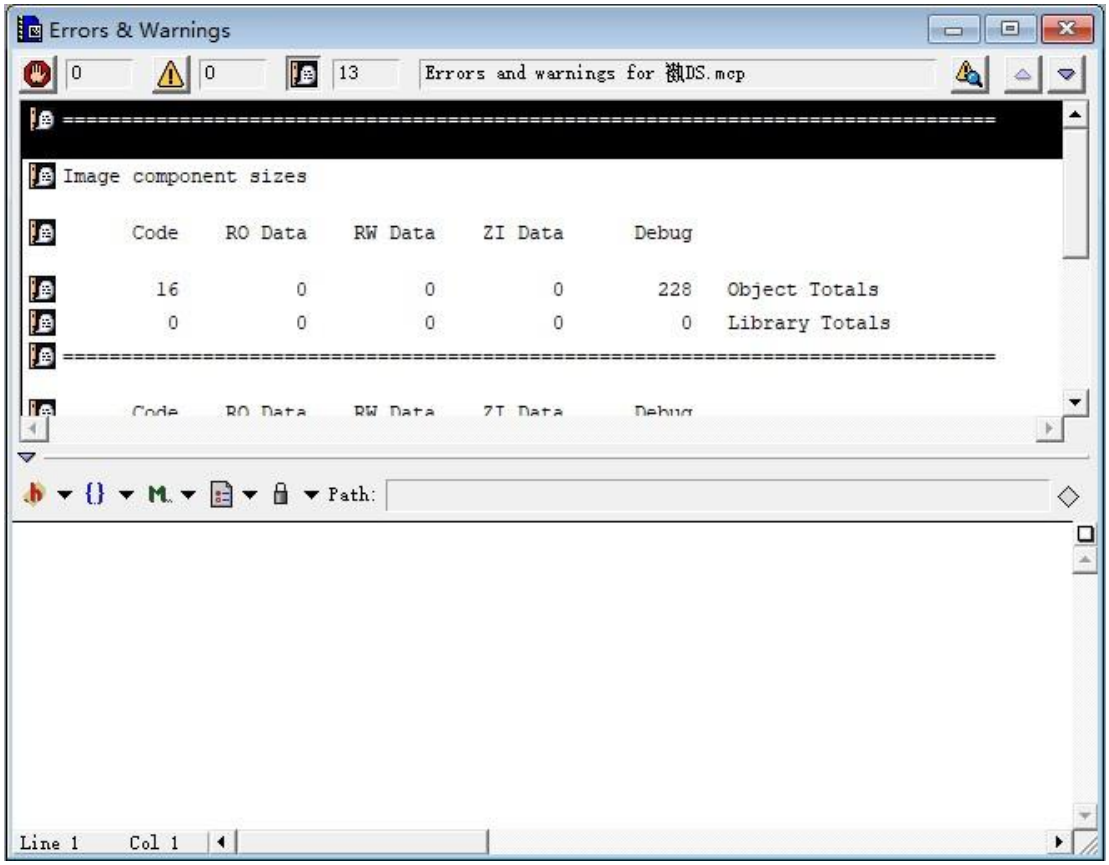

图 1-6 编译错误和警告对话框

步骤 6：单步运行程序

选择 Project→Debug 命令或 F5 将打开 AXD 进行调试。

1) 起始状态，如图 1-7 所示。

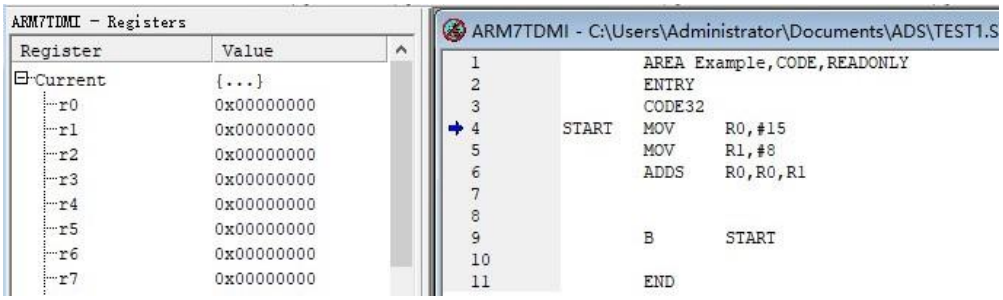

图 1-7 起始状态

2) 写入 R0，如图 1-8 所示。

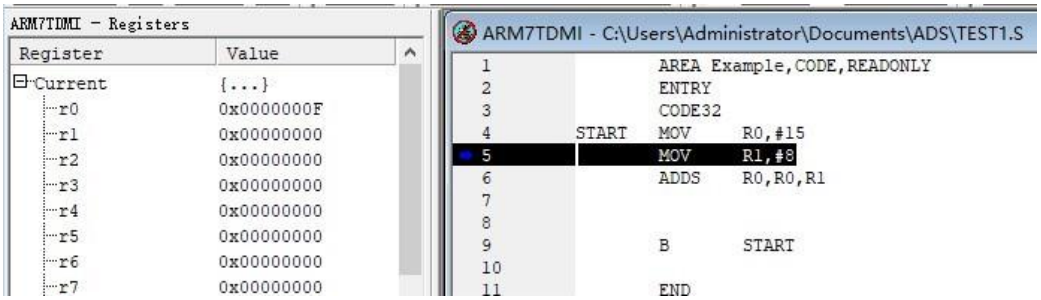

图 1-8 写入 R0

3) 写入 R1，如图 1-9 所示。

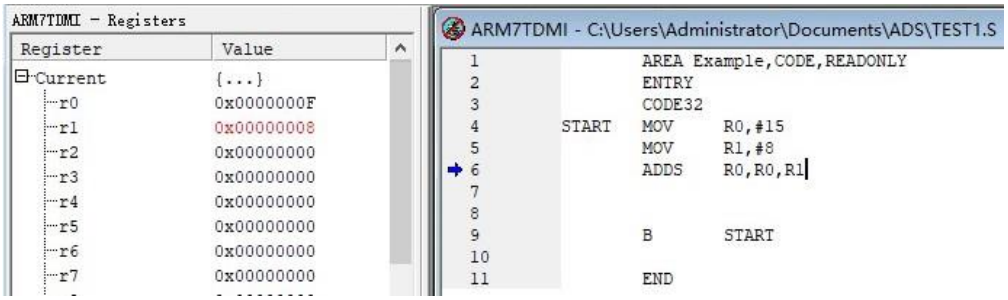

图 1-9 写入 R1

4) 将相加后的值写入 R0，如图 1-10 所示。

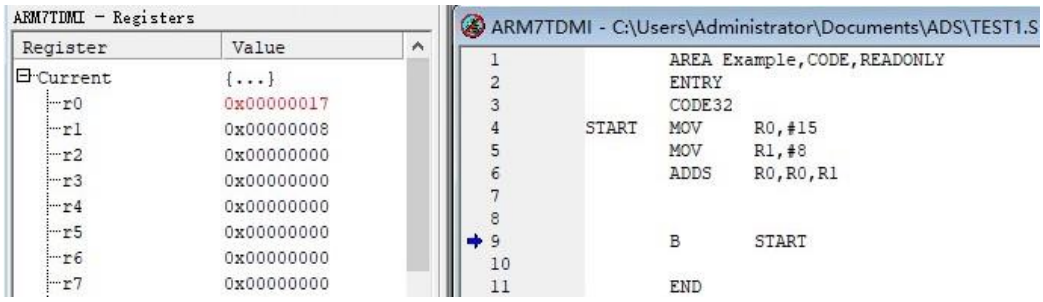

图 1-10 相加后的值写入 R0

步骤 7：强行重新编译工程

若想重新编译程序，选择 Project→Remove Object Code 命令，删除工程重点*.obj 文件，即可重新 Make 编译，如图 1-11 所示。

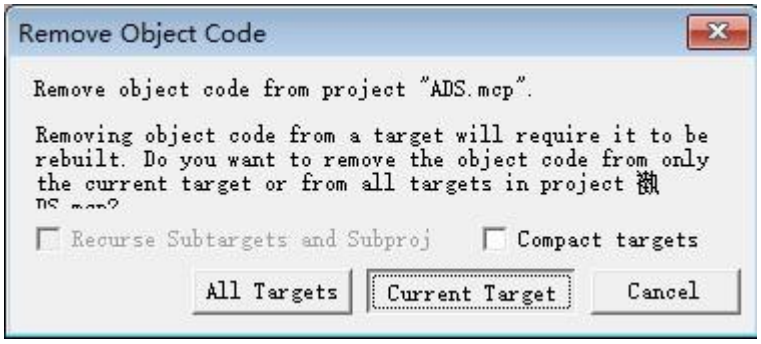

图 1-11 强行重新编译工程

四、 心得与体会

ADS 全称是 ARM Developer Suite，是一款由 ARM 公司提供的专门用于 ARM 相关应用开发和调试的综合性软件，这次试验主要用到了 CodeWarrior 和 AXD Debugger 两部分，在软件的安装时参照了网络上多个版本的教程，再次感谢老师与前辈们的付出与努力。

由于目前使用的是 Win10，若在应用列表中未出现一系列程序，可在安装目录的 Bin 中将 IDE.exe 和 AXD.exe 发送快捷键至桌面即可。

出现“error starting external process, Process error code 87(0x57)参数错误”，运行 ADS1.2 需要设置兼容性，将上述两个模块均以兼容模式运行 Windows XP（Service Pack2），在此处不选“以管理员模式运行此程序”，实测若选中仍然出现问题。

进行 Debug 调试时打开了 AXD 程序，但不会 Load Image，此处需要手动设置。Options→Config Target 命令，选中 ARMUL，OK 即可。Target 中出现 ARM7TDMI，重新进行 Debug 将进入调试步骤。

五、 附录

```
        AREA Example,CODE,READONLY      ; Pseudo-instruction, code snippet name
        ENTRY                           ; Program entry point
        CODE32                          ; 32-bit ARM instructions
START   MOV     R0,#15                  ; START is a label,15->R0
        MOV     R1,#8                   ; 8->R1
        ADDS    R0,R0,R1                ; R0=R0+R1

        B       START                   ; Jump to the START label

        END                             ; End of the source program
```

实验 2：汇编指令实验 1

一、 实验目的

1. 了解 ADS1.2 集成开发环境及 ARMulator 仿真软件。
2. 掌握汇编指令的用法，并能编写简单的汇编指令。
3. 掌握指令的条件执行和使用 LDR/STR 指令完成存储器的访问。

二、 实验内容

1. 使用 LDR 指令读取 0x40003100 地址上的数据，将数据加 1。若结果小于 10，则使用 STR 指令把结果写回原地址；若结果大于等于 10，则把 0 写回原地址。
2. 使用 ADS1.2 仿真软件，单步、全速运行程序，设置断点，打开寄存器窗口（Processor Registers）监视 R0 和 R1 的值，打开存储器观察窗口（Memory）监视 0x40003100 地址上的值。

三、 实验步骤

步骤 1： 正确连接试验箱及 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 Instrustion1。

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 TEST2.S，直接添加到项目中，输入代码并保存。

步骤 4： 设置地址

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。

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 2-1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

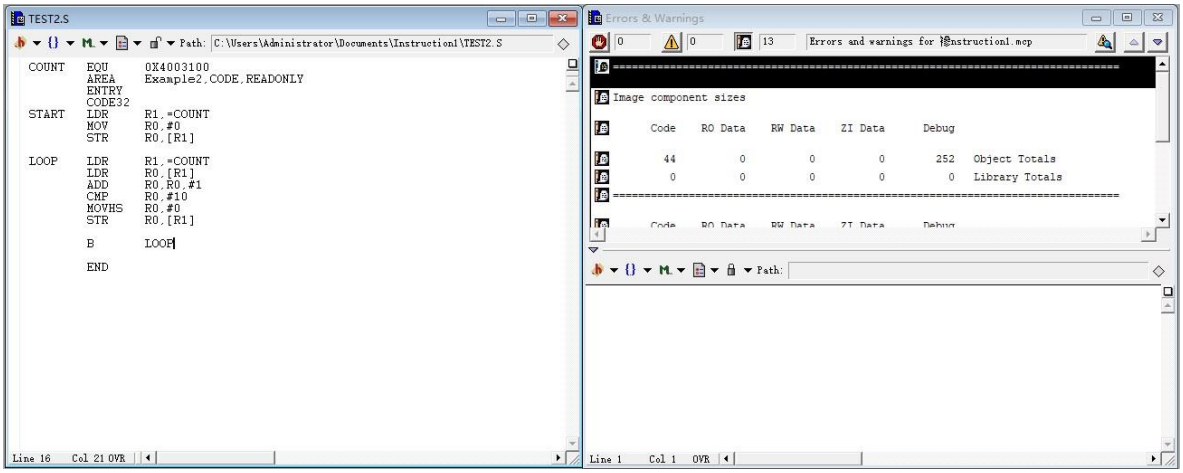

图 2-1 实验程序和错误警告对话框

步骤 6：观察监视寄存器和存储器上的值

打开寄存器窗口（Processor Registers），选择 Current 选项监视 R0 和 R1 的值。打开存储器观察窗口（Memory），设置观察地址为 0x4003100，显示方式 Size 为 32Bit，监视 0x4003100 地址上的值。

步骤 7：调试程序

可以单步运行程序，也可以设置/取消断电，或者全速运行程序，停止程序运行，调试时观察寄存器和 0x4003100 的地址上的值，运行结果如图 2-2 所示。

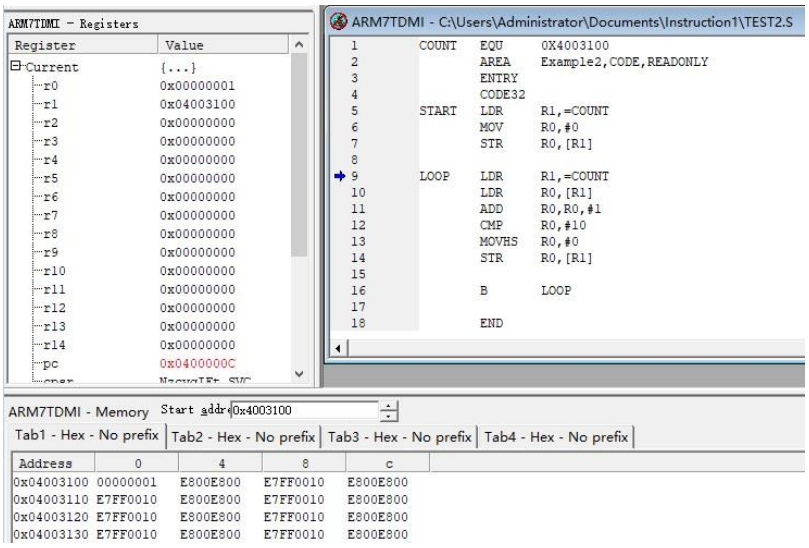

图 2-2 程序运行结果

四、 心得与体会

本程序是将 0x4003100 处的值加载给 R1，再将立即数 0 给 R0，之后将 R0 数据（此处为 0）存储到 R1 所在的地址（此处为 0x4003100），完成 START 部分。紧接着开始 LOOP，一直循环往复执行 R0=R0+1，CMP 和 MOVHS 两段代码是判断 R0 是否大于等于 10，如果大于等于 10 则执行 0→R0，否则不执行，即完成了需求 0→1→2→3→4→5→6→7→8→9→10→0→1……

五、 附录

```
COUNT   EQU     0X4003100               ; Define a name
        AREA    Example2,CODE,READONLY  ; Pseudo-instruction,code snippet name
        ENTRY                           ; Program entry point
        CODE32                          ; 32-bit ARM instructions
START   LDR     R1,=COUNT               ; Load 32-bit immediate data COUNT
        MOV     R0,#0                   ; 0->R0
        STR     R0,[R1]                 ; Store the data of the R0 register to the storage uni
t pointed to by R1 (offset is 0)

LOOP    LDR     R1,=COUNT               ; Repeat the initialization operation
        LDR     R0,[R1]                 ; Load R1 to R0
        ADD     R0,R0,#1                ; R0=R0+1
        CMP     R0,#10                  ; R0 is compared with 10, set the relevant flag bit
        MOVHS   R0,#0                   ; If >=, then 0->R0
        STR     R0,[R1]                 ; The repeat operation

        B       LOOP                    ; Jump to the LOOP label

        END                            ; End of the source program
```

实验 3：汇编指令实验 2

一、 实验目的

1. 了解 ARM 数据处理指令的使用方法。
2. 了解 ARM 指令的第二个操作数。

二、 实验内容

1. 使用 MOV 和 MVN 指令访问 ARM 通用寄存器。
2. 使用 ADD、SUB、ORR、CMP 和 TST 等指令完成数据加减运算及逻辑运算。

三、 实验步骤

步骤 1： 正确连接试验箱及 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 Instrustion2。

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 TEST3.S，直接添加到项目中，输入代码并保存。

步骤 4： 设置地址

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。

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 3-1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

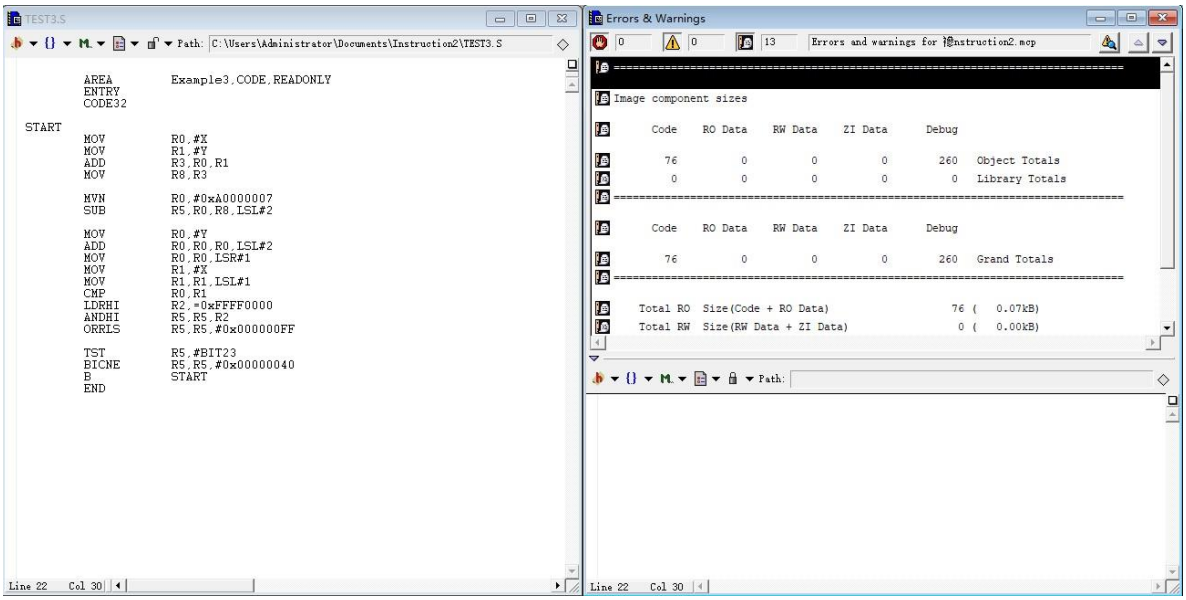

图 3-1 实验程序和错误警告对话框

步骤 6： 监视各寄存器的值

打开寄存器窗口（Processor Registers），选择 Current 选项监视各寄存器的值。

步骤 7：单步运行程序，观察寄存器值的变化，如图 3-2 所示。

tips：有变化的寄存器会以红色显示。

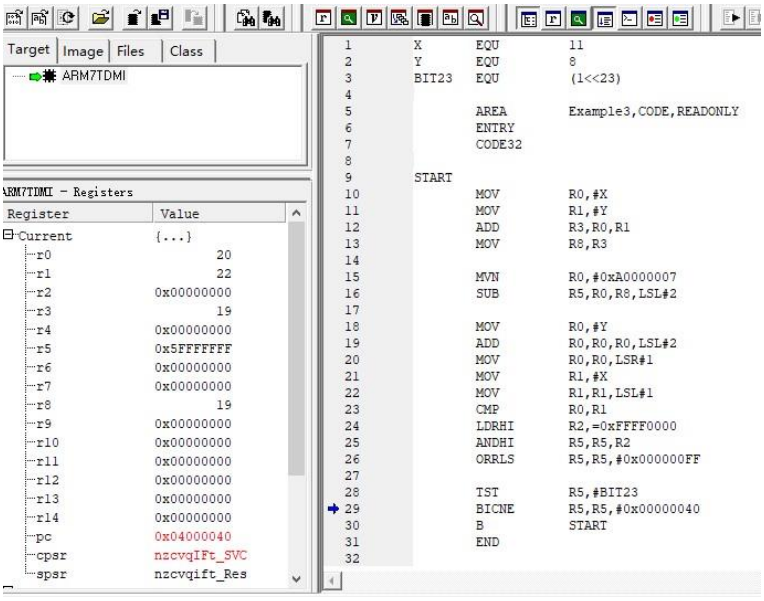

图 3-2 寄存器值的变化

四、 心得与体会

MVN R0,#0xA0000007 段代码是将#0xA0000007 取反后给 R0，A 取反是 5，0

取反是 F，7 取反是 8；经过这各实验后对 ADD、SUB、ORR、CMP 和 TST 等数据加减及逻辑运算有了更深刻的认识。

五、 附录

```
X       EQU         11                      ; Define a name X=11
Y       EQU         8                       ; Define a name Y=8
BIT23   EQU         (1<<23)                 ; Define a name BIT23=1<<23

        AREA        Example3,CODE,READONLY  ; Pseudo-instruction,code snippet name
        ENTRY                               ; Program entry point
        CODE32                              ; 32-bit ARM instructions

START
        MOV         R0,#X                   ; X(there is 11)->R0
        MOV         R1,#Y                   ; Y(there is 8)->R1
        ADD         R3,R0,R1                ; R3=R0+R1
        MOV         R8,R3                   ; R3->R8

        MVN         R0,#0xA0000007          ; Store the inversion result of #0xA0000007 in R0

        SUB         R5,R0,R8,LSL#2          ; R5=R0-R8<<2

        MOV         R0,#Y                   ; Y(there is 8)->R0
        ADD         R0,R0,R0,LSL#2          ; R0=R0+R0<<2=R0*5
        MOV         R0,R0,LSR#1             ; R0>>1 -> R0
        MOV         R1,#X                   ; X(there is 11)->R1
        MOV         R1,R1,LSL#1             ; R1<<1 -> R1
        CMP         R0,R1                   ; R0 is compared with R1, set NZCV bit
        LDRHI       R2,=0xFFFF0000          ; If R0>R1, then R2 load 0xFFFF0000R2
        ANDHI       R5,R5,R2                ; If R0>R1, then R5=R5&R2
        ORRLS       R5,R5,#0x000000FF       ; If R0<=R1(else),then R5=R5|#0x000000FF

        TST         R5,#BIT23               ; R0 and R1 Bitwise logical AND operation, set NZC
V bit
        BICNE       R5,R5,#0x00000040       ; If z=0,then Save the logical AND result of R5 an
d #x00000040 inverse code to R5
        B           START                   ; Jump to the START label
        END                                 ; End of the source program
```

实验 4：汇编指令实验 3

一、 实验目的

1. 掌握 ARM 乘法指令的使用方法。
2. 了解子程序编写及调用。

二、 实验内容

使用 STMFD／LDMFD、MUL 指令编写一个整数乘法的子程序，然后使用 BL 指令调用子程序计算 X^n 的值。

三、 实验步骤

步骤 1： 正确连接试验箱及 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 Instrustion3。

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 TEST4.S，直接添加到项目中，输入代码并保存。

步骤 4： 设置地址

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。

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

4-1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

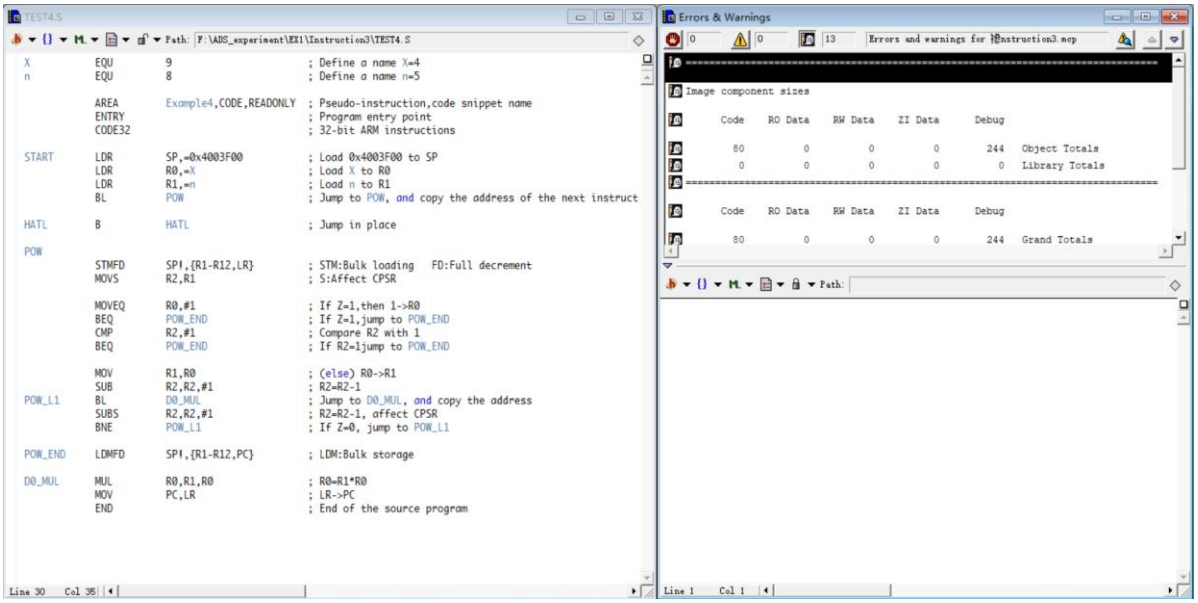

图 4-1　程序及错误和警告对话框

步骤 6： 监视寄存器的值

打开寄存器窗口(Processor Registers),选择 Current 项监视寄存器 R0、R1、R13(SP)和 R14(LR)的值，如图 4-2 所示。

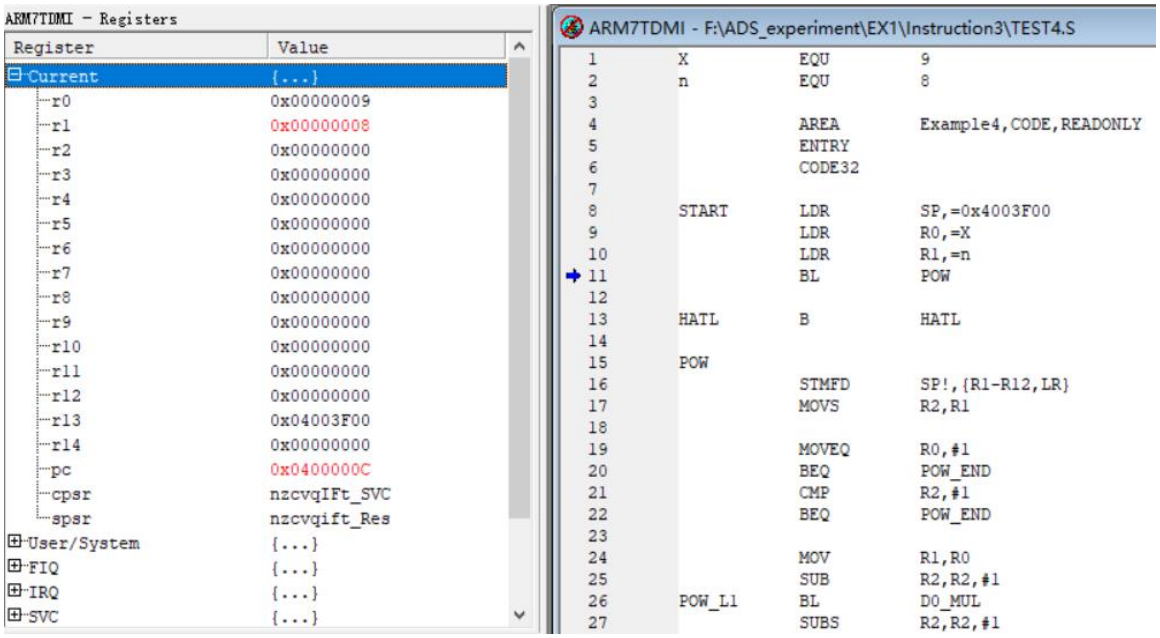

图 4-2 写入 X 与 n 值后的截图

步骤 7：观察存储器的值

打开存储器观察窗口（Memory），设置观察地址为 0 x 4003EA0，显示方式 Size

为 32bit，监视从 0x4003F00 起始的满递减堆栈区，如图 4-3 所示。

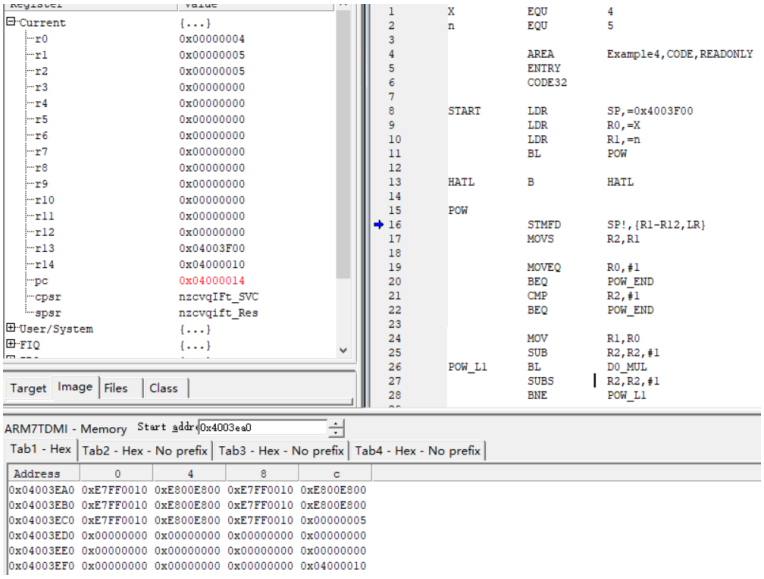

图 4-3 将 R1~R12 及 R14 的值写入

步骤 8：单步运行程序，观察寄存器值的变化

单步运行程序，跟踪程序执行的流程，观察寄存器值的变化和堆栈区的数据变化，判断执行结果是否正确。

步骤 9：调试程序

调试程序时，改变参数 X 和 n 来测试程序，观察是否得到正确的结果。例如，先复位程序（选择 File→Reload Current Image 命令），接着单步执行到 BL POW 指令，在寄存器窗口中将 R0 和 R1 的值进行修改，然后继续运行程序。两种结果如图 4-2 和图 4-3 所示。

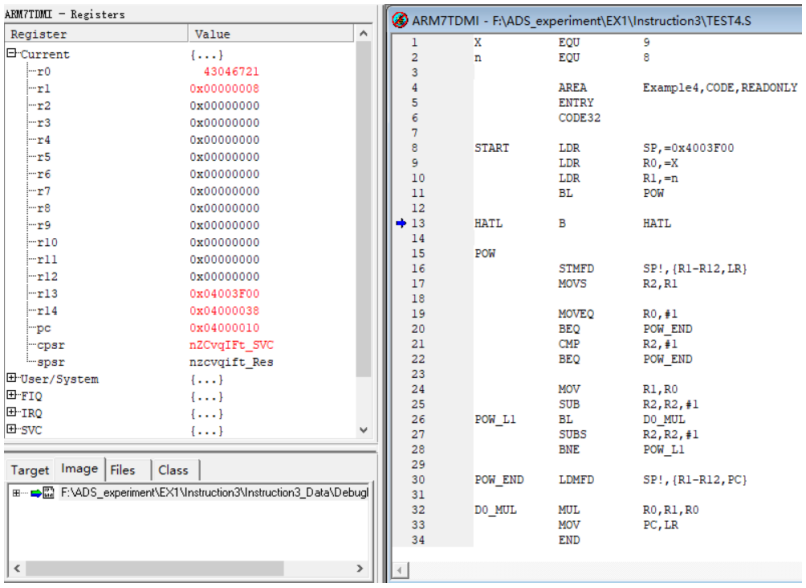

图 4-4 $X^n = 9^8 = 43046721$

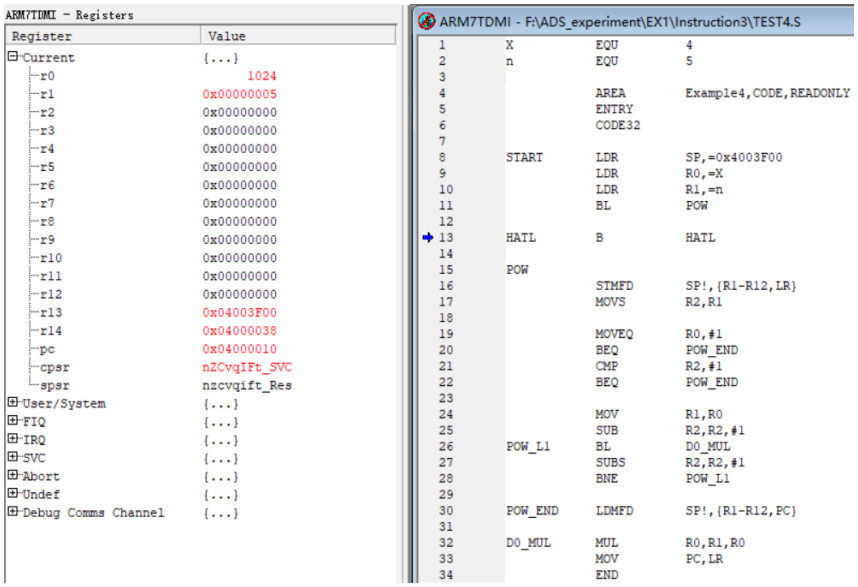

图 4-5 $X^n = 4^5 = 1024$

四、 心得与体会

先将 X 和 n 定义赋值，初始化加载 SP 指针为 0x4003F00，用 BL 跳转指令会在 R14（LR）存储下一条指令的地址（即指令 HATL 的地址 0x04000010）以方便幂运算结束后返回，用满递减堆栈存储 R0~R12 和 LR 的值，由于是使用满递减堆栈进行现场保护的，故最后的栈顶在 0x04003F00，！表示 SP 的值将更新，最后更新为 SP=0x04003ECC。

MOVS R2, R1 中的 S 表示将影响 CPSR 中的值，对下面的几条判断指令产生影响。MOVEQ R0, #1 和 BEQ POW_END 均在检查 Z=1，若 R1=0 即 n=0，此时 Z=1，运算结果 R0 将直接赋 1，并且跳转到 POW_END 程序段准备恢复现场后结束运算程序；若 Z≠0，将继续向下运行 CMP R2, #1 即比较 n 和 1 的值， 此时结果为 Z=0 表示运算结果非 0，C=1 表示减法运算 R2-#0 时未产生借位。BEQ POW_END 接上一段 CMP 指令，若 Z=1 则跳转至 POW_END。

向下运行 MOV R1, R0，将 R0 即未进行幂运算前的 X 保存至 R1，此时幂次 n 已经由前指令 MOVS R2, R1 转存至 R0，故不会产生影响。SUB R2, R2, #1 将幂次-1，表示准备开始进行下面的幂运算。

POW_L1 和 DO_MUL 为幂运算的主要程序。在 POW_L1 中，先 BL 跳转至 DO_MUL，L 表示将复制下一条指令（此处的 SUBS 指令）的地址到链接寄存器 R14（LR）以方便返回 POW_L1。MUL R0, R1, R0 进行一次 R0=R0×R1，即结果=结果×底数；MOV PC, LR 将 BL 时候存下来的 LR 给 PC 寄存器，因此可以返回到 BL 跳转指令的下一条指令继续进行下一级幂运算。BL 跳转返回后下一条指令 SUBS R2, R2, #1，表示 R2=R2-1 即进行了一次幂运算后幂次-1 并且影响 CPSR，BNE POW_L1 判定 Z=0，若 Z=0 则上述运算结果非 0，幂运算还未结束，继续进行下一次幂运算跳转至 POW_L1;若 Z=1 则上述运算结果为 0，幂运算结束，不进行跳转，将继续执行下一条指令 LDMFD SP!, {R1-R12, PC}恢复现场，恢复 LR=0x04000010，跳转回 HATL B HATL 后结束此程序。

五、 附录

```
X           EQU         4                       ; Define a name X=4
n           EQU         5                       ; Define a name n=5

            AREA        Example4,CODE,READONLY  ; Pseudo-instruction,code snippet name
            ENTRY                               ; Program entry point
            CODE32                              ; 32-bit ARM instructions

START       LDR         SP,=0x4003F00           ; Load 0x4003F00 to SP
            LDR         R0,=X                   ; Load X to R0
            LDR         R1,=n                   ; Load n to R1
            BL          POW                     ; Jump to POW, and copy the address of the nex
t instruction to the R14

HATL        B           HATL                    ; Jump in place

POW
            STMFD       SP!,{R1-R12,LR}         ; STM:Bulk loading   FD:Full decrement
            MOVS        R2,R1                   ; S:Affect CPSR

            MOVEQ       R0,#1                   ; If Z=1,then 1->R0
            BEQ         POW_END                 ; If Z=1,jump to POW_END
            CMP         R2,#1                   ; Compare R2 with 1
            BEQ         POW_END                 ; If R2=1jump to POW_END

            MOV         R1,R0                   ; (else) R0->R1
            SUB         R2,R2,#1                ; R2=R2-1
POW_L1      BL          D0_MUL                  ; Jump to D0_MUL, and copy the address
            SUBS        R2,R2,#1                ; R2=R2-1, affect CPSR
            BNE         POW_L1                  ; If Z=0, jump to POW_L1

POW_END     LDMFD       SP!,{R1-R12,PC}         ; LDM:Bulk storage

D0_MUL      MUL         R0,R1,R0                ; R0=R1*R0
            MOV         PC,LR                   ; LR->PC
            END                                 ; End of the source program
```

实验 5：ARM 微控制器工作模式实验

一、 实验目的

1. 掌握使用 MRS/MSR 指令实现 ARM 微控制器工作模式切换的方法。
2. 了解在各个工作模式下的寄存器。

二、 实验内容

1. MRS/MSR 指令切换工作模式，并初始化各种模式下的对战执政。
2. 观察 ARM 微控制器在各种模式下寄存器的区别。

三、 实验步骤

步骤 1： 正确连接试验箱及 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 MODE。

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 TEST7.S，直接添加到项目中，输入代码并保存。

步骤 4： 设置地址

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。

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 5-1 所示；然后选择 Project→Debug 命令，启动 AXD 进行软件仿真调试。

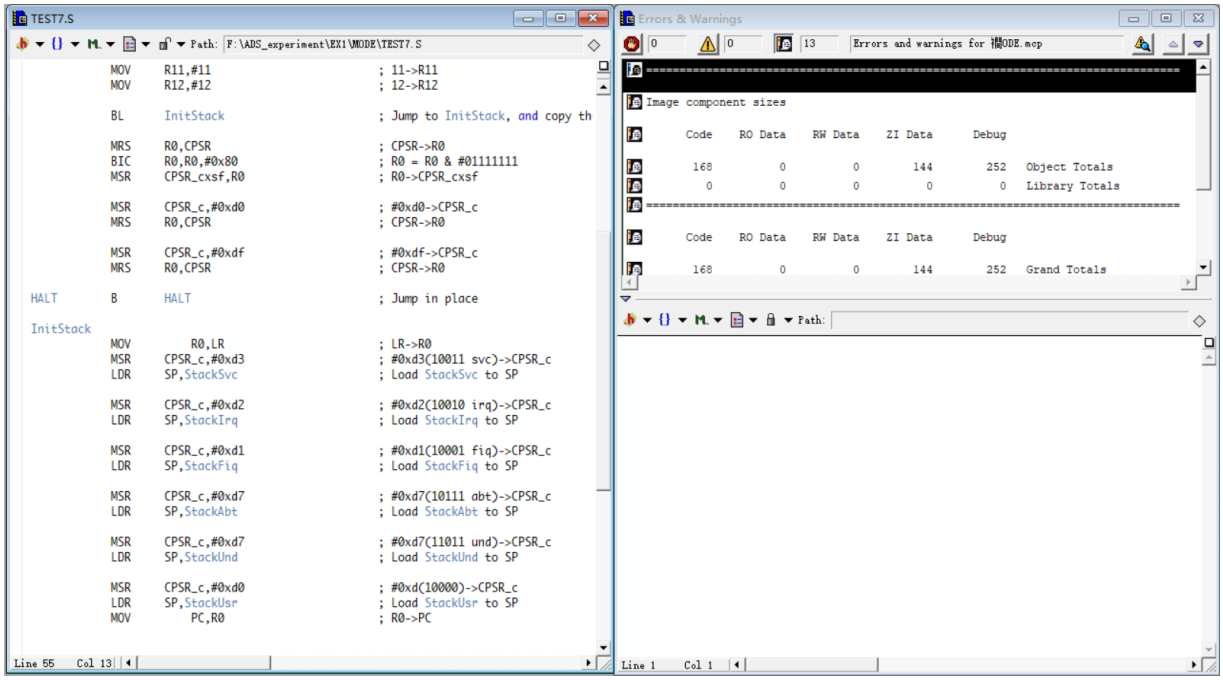

图 5-1 实验程序和错误警告对话框

步骤 6： 单步运行程序

打开寄存器窗口（Processor Registers），选择 Current 项监视各寄存器的值。单步运行程序，注意观察 CPSR、SPSR、R13（SP）、R14（LR）和 R15（PC）寄存器，寄存器更新示意例图如图 5-2 所示。

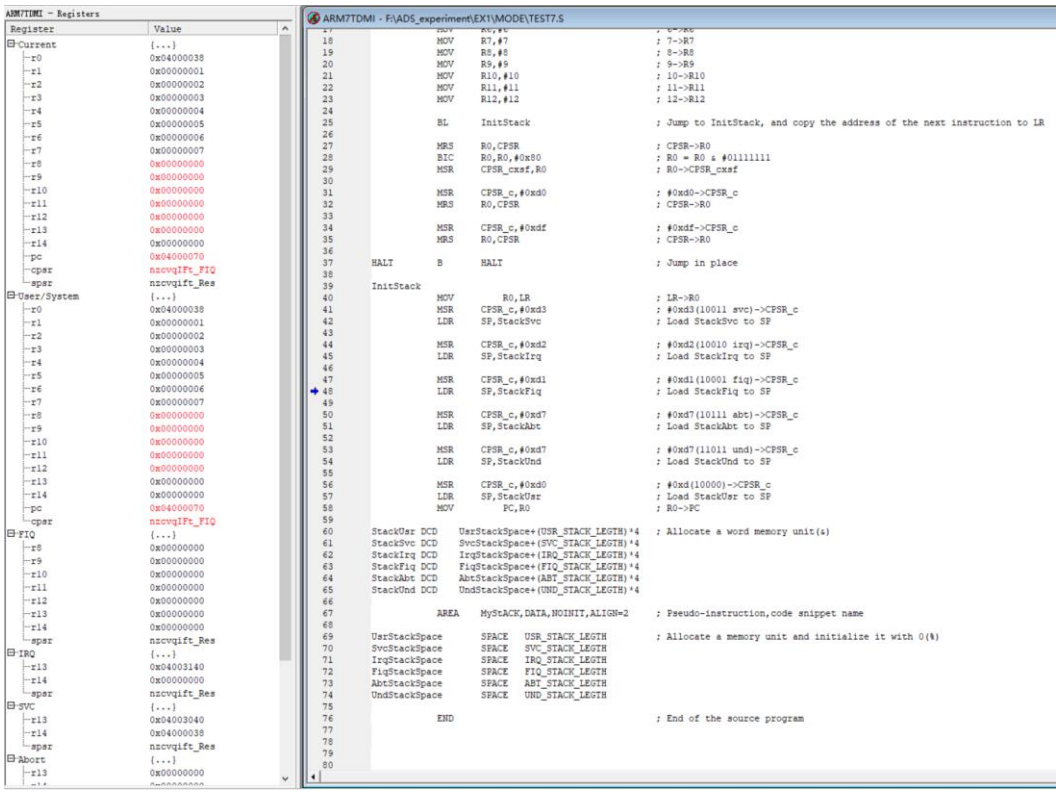

图 5-2 寄存器更新

四、 心得与体会

标志位 NZCVQ 的条件码标志 N、Z、C、V 和 Q 可以打开 cpsr 或者 spsr 后的 Value 查看，显示为大写字母，表示该位为 1；显示为小写字母，表示该位为 0。Q 标志位在 ARM 体系结构 v5 及以上版本的 E 变量中才有效。

标志位 IFT 为 IRQ 中断禁止位 I、FIQ 终端禁止位 F 和 ARM 微控制器状态位 T，显示为大写字母，表示该位为 1；显示为小写字母，表示该位为 0。T 标志位在 ATM 体系结构 v4 及以上版本的 T 变量中才有效。

工作模式知识 ARM 微控制器当前的工作模式，包括 User（用户模式）、FIQ（FIQ 中断模式）、IRQ（IRQ 中断模式）、SVC（SVC 管理模式）、Abort（中止模式）、Undef（未定义模式）和 SYS（系统模式）。

M[4:0]	Mode
0b10000	Usr
0b10001	FIQ
0b10010	IRQ
0b10011	Svc
0b10111	Abt
0b11011	Und
0b11111	Sys

表 5-1 CPSR 最低五位与对应工作模式

五、 附录

```
; Define the stack length of different modes
USR_STACK_LEGTH         EQU     64
SVC_STACK_LEGTH         EQU     0
FIQ_STACK_LEGTH         EQU     16
IRQ_STACK_LEGTH         EQU     64
ABT_STACK_LEGTH         EQU     0
UND_STACK_LEGTH         EQU     0

            AREA    Example7,CODE,READONLY          ; Pseudo-instruction,code snippet name
            ENTRY                                   ; Program entry point
```

```
            CODE32                              ; 32-bit ARM instructions
START       MOV     R0,#0                       ; 0->R0
            MOV     R1,#1                       ; 1->R1
            MOV     R2,#2                       ; 2->R2
            MOV     R3,#3                       ; 3->R3
            MOV     R4,#4                       ; 4->R4
            MOV     R5,#5                       ; 5->R5
            MOV     R6,#6                       ; 6->R6
            MOV     R7,#7                       ; 7->R7
            MOV     R8,#8                       ; 8->R8
            MOV     R9,#9                       ; 9->R9
            MOV     R10,#10                     ; 10->R10
            MOV     R11,#11                     ; 11->R11
            MOV     R12,#12                     ; 12->R12
; Jump to InitStack, and copy the address of the next instruction to LR
            BL      InitStack

            MRS     R0,CPSR                     ; CPSR->R0
            BIC     R0,R0,#0x80                 ; R0 = R0 & #01111111
            MSR     CPSR_cxsf,R0                ; R0->CPSR_cxsf

            MSR     CPSR_c,#0xd0                ; #0xd0->CPSR_c
            MRS     R0,CPSR                     ; CPSR->R0

            MSR     CPSR_c,#0xdf                ; #0xdf->CPSR_c
            MRS     R0,CPSR                     ; CPSR->R0

HALT        B       HALT                        ; Jump in place

InitStack
            MOV         R0,LR                   ; LR->R0
            MSR     CPSR_c,#0xd3                ; #0xd3(10011 svc)->CPSR_c
            LDR     SP,StackSvc                 ; Load StackSvc to SP

            MSR     CPSR_c,#0xd2                ; #0xd2(10010 irq)->CPSR_c
            LDR     SP,StackIrq                 ; Load StackIrq to SP

            MSR     CPSR_c,#0xd1                ; #0xd1(10001 fiq)->CPSR_c
            LDR     SP,StackFiq                 ; Load StackFiq to SP

            MSR     CPSR_c,#0xd7                ; #0xd7(10111 abt)->CPSR_c
            LDR     SP,StackAbt                 ; Load StackAbt to SP

            MSR     CPSR_c,#0xd7                ; #0xd7(11011 und)->CPSR_c
```

```
            LDR     SP,StackUnd                     ; Load StackUnd to SP

            MSR     CPSR_c,#0xd0                    ; #0xd(10000)->CPSR_c
            LDR     SP,StackUsr                     ; Load StackUsr to SP
            MOV         PC,R0                       ; R0->PC

StackUsr DCD    UsrStackSpace+(USR_STACK_LEGTH)*4   ; Allocate a word memory unit(&)
StackSvc DCD    SvcStackSpace+(SVC_STACK_LEGTH)*4
StackIrq DCD    IrqStackSpace+(IRQ_STACK_LEGTH)*4
StackFiq DCD    FiqStackSpace+(FIQ_STACK_LEGTH)*4
StackAbt DCD    AbtStackSpace+(ABT_STACK_LEGTH)*4
StackUnd DCD    UndStackSpace+(UND_STACK_LEGTH)*4

            AREA    MyStACK,DATA,NOINIT,ALIGN=2     ; Pseudo-instruction,code snippet name

UsrStackSpace       SPACE   USR_STACK_LEGTH         ; Allocate a memory unit and initialize it
 with 0(%)
SvcStackSpace       SPACE   SVC_STACK_LEGTH
IrqStackSpace       SPACE   IRQ_STACK_LEGTH
FiqStackSpace       SPACE   FIQ_STACK_LEGTH
AbtStackSpace       SPACE   ABT_STACK_LEGTH
UndStackSpace       SPACE   UND_STACK_LEGTH

            END                                     ; End of the source program
```