

ARM 与嵌入式技术

实验报告

专业班级：通信工程

姓名：****

学号：*****

实验日期：2012 年 6 月 7 日

指导老师：*****

一、实验目的

1. 学习使用 Embest IDE 开发环境及 ARM 软件模拟器；
2. 掌握简单 ARM 汇编指令, 进一步加强对嵌入式的熟悉和了解。

二、实验设备

硬件: PC 机

软件: Embest IDE 开发环境

三、实验内容

例 3: 实现 64 位加法运算, 要求【R1:R0】+【R3:R2】, 结果放回【R1:R0】中;

例 2: 编写程序将 R2 的高 8 位传送到 R3 的低 8 位 (不考虑 R3 的其它位);

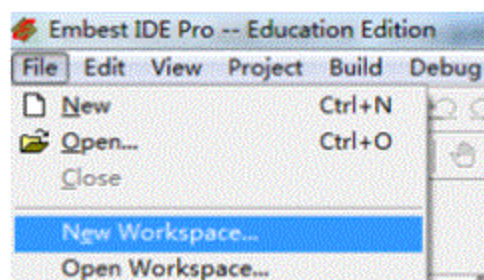
例 7: 编写一段程序计算 10! ;

例 8: 串拷贝 (R1 指向源数据串的首地址, R0 指向目的数据串的首地址)。

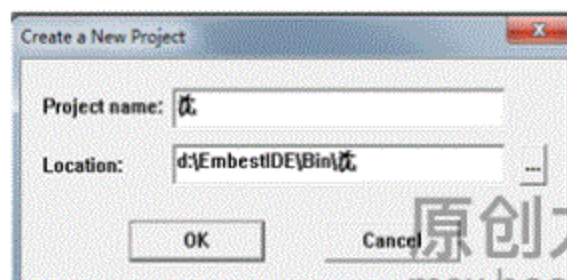
四、实验步骤

1) 新建工程:

运行 Embest IDE 集成开发环境, 选择菜单项 File → New Workspace, 如图一, 系统弹出一个对话框, 键入文件名“沈”, 如图二, 点击 OK 按钮。将创建一个新工程, 并同时创建一个与工程名相同的工作区。此时在工作区窗口将打开该工作区和工程。



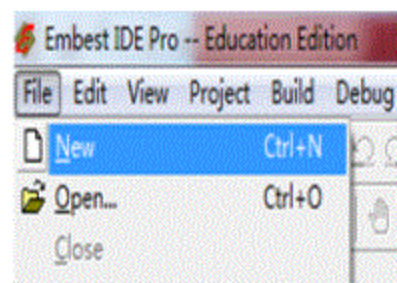
图一



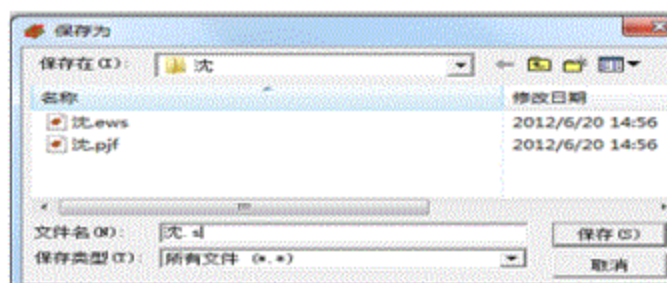
图二

2) 建立源文件:

点击菜单项 File → New, 如图三, 系统弹出一个新的文本编辑窗, 输入源文件代码。编辑完后, 保存文件“沈.s”后缀, 如图四。



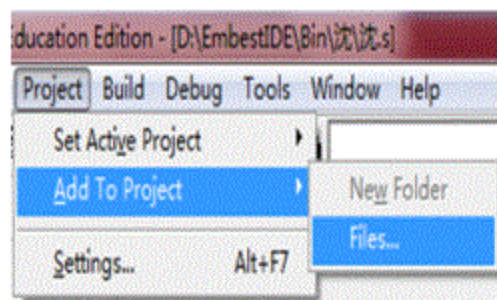
图三



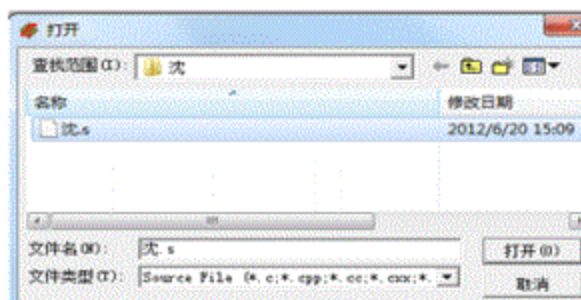
图四

3) 添加源文件:

选择菜单项 Project → Add To Project → Files , 在工程目录下选择刚才建立的源文件.s 后缀文件, 如图五, 图六。



图五



图六

4) 基本配置:

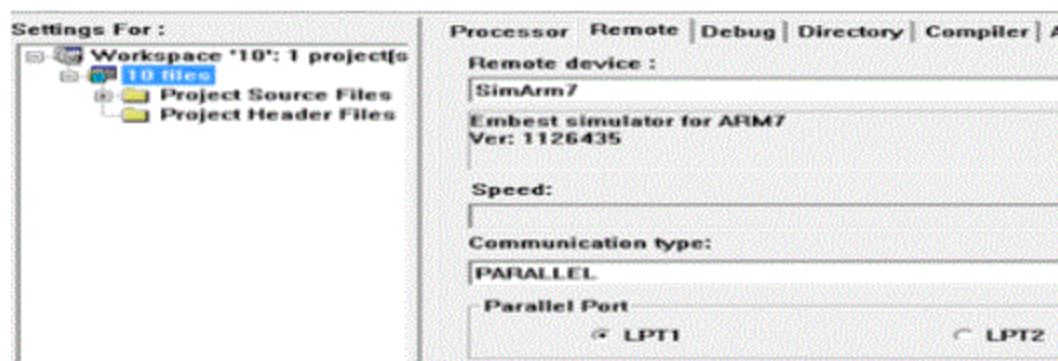
选择菜单项 Project → Settings, 弹出工程设置对话框。在工程设置对话框中。

① 选择Processor 设置对话框, 按照图七所示, 进行配置:



图七

② 选择Remote设置对话框, 按照下图八所示, 进行配置:



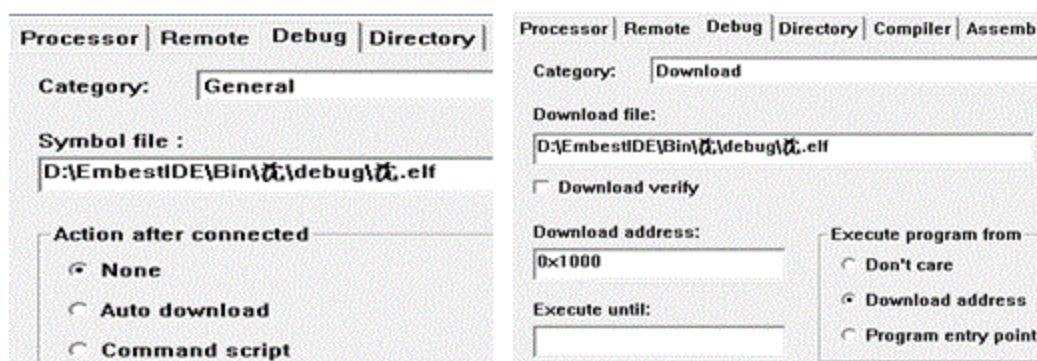
图八

③ 选择 最右边一个进行编译, 显示如图九, 则编译成功。

```
-----Building project: 沈-----  
arm-elf-as -gdwarf2 D:\EmbestIDE\Bin\沈\沈.s -o.\debug\沈.o  
arm-elf-ld -o.\debug\沈.elf .\debug\沈.o  
Command(s) successfully executed.
```

图九

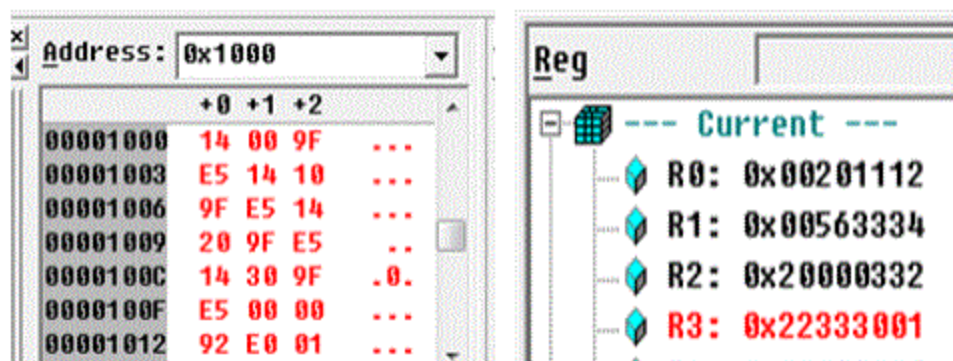
④ 选择Project → Settings → Debug设置对话框，按照图十所示，进行配置：



图十

⑤ 选择    最右边一个进行编译，显示如图九，则编译成功。

5) 选择菜单项Debug → Remote Connect 进行连接软件仿真器，将存储器地址改为0x1000，如图十一，执行Debug → Download 命令下载程序，并打开寄存器窗口。打开memory 窗口，按F10进行单步跟踪，观察寄存器的数据变化并分析。



图十一

五、各实验的参考程序及运行结果

实验一：（例 3）实现 64 位加法运算，要求【R1:R0】+【R3:R2】，结果放回【R1:R0】中；

1.程序代码如下：

```
.global _start
.text
_start:
mov R0,#11          /*R0=11*/
mov R1,#22          /*R1=22*/
mov R2,#33          /*R2=33*/
mov R3,#44          /*R3=44*/
```


ADDS R0,R0,R2

/*R0 等于低 32 位相加，并影响标志位*/

ADC R1,R1,R3

/*R1 等于高 32 位相加，并加上低位进位*/

stop:

b stop

.end

2. 分析调试:

① download 下载:

0x00001000	mov	r0, #11
0x00001004	mov	r1, #22
0x00001008	mov	r2, #33
0x0000100c	mov	r3, #44
0x00001010	adds	r0, r0, r2
0x00001014	adc	r1, r1, r3
0x00001018	b	0x1018
0x0000101c	swinv	0x00ffffff
0x00001020	swinv	0x00ffffff

Address: 0x1000			
	+0	+1	+2
00001000	0B	00	A0
00001003	E3	16	10
00001006	A0	E3	21
00001009	20	A0	E3
0000100C	2C	30	A0
0000100F	E3	02	00
00001012	90	E0	03

② 读入数据:

0x00001000	mov	r0, #11
0x00001004	mov	r1, #22
0x00001008	mov	r2, #33
0x0000100c	mov	r3, #44
0x00001010	adds	r0, r0, r2
0x00001014	adc	r1, r1, r3
0x00001018	b	0x1018
0x0000101c	swinv	0x00ffffff

Reg	
--- Current ---	
R0:	0x0000000b
R1:	0x00000016
R2:	0x00000021
R3:	0x0000002c
R4:	0x00000000

③ r0+r2→r0 (低 32 位):

0x00001000	mov	r0, #11
0x00001004	mov	r1, #22
0x00001008	mov	r2, #33
0x0000100c	mov	r3, #44
0x00001010	adds	r0, r0, r2
0x00001014	adc	r1, r1, r3
0x00001018	b	0x1018
0x0000101c	swinv	0x00ffffff

Reg	
--- Current ---	
R0:	0x0000002c
R1:	0x00000016
R2:	0x00000021
R3:	0x0000002c
R4:	0x00000000

④ r1+r3→r1 (带进位的加法):

0x00001000	mov	r0, #11
0x00001004	mov	r1, #22
0x00001008	mov	r2, #33
0x0000100c	mov	r3, #44
0x00001010	adds	r0, r0, r2
0x00001014	adc	r1, r1, r3
0x00001018	b	0x1018
0x0000101c	swinv	0x00ffffff

Reg	
--- Current ---	
R0:	0x0000002c
R1:	0x00000042
R2:	0x00000021
R3:	0x0000002c
R4:	0x00000000

原创力文档

max.book118.com

预览与源文档一致, 下载高清无水印

实验二：（例 2）编写程序将 R2 高 8 位传送到 R3 的低 8 位（不考虑 R3 的其它位）；

1.程序代码如下：

```
.global _start
_start:
    ldr r2,=0x
    ldr r3,=0xabcd1200
    and r2,r2,#0xff    /*保留 R2 的高 8 位,屏蔽低 24 位*/
    and r3,r3,#0xfffff0 /*保留 R3 的高 24 位,屏蔽低 8 位*/
    orr r3,r3,r2,lsr #24 /*将 R2 的高 8 位传送到 R3 的低 8 位*/
stop:
    b stop
.end
```

2.分析调试：

①download 下载：

Address	Value
0x00001000	10 20 9F ..
0x00001004	E5 10 30 ..0
0x00001008	9F E5 FF ..
0x0000100C	24 02 E2 \$..
0x00001010	FF 30 C3 -0.
0x00001014	E3 22 3C -"<
0x00001018	83 E1 FE ...

②保留 r2 的高 8 位，屏蔽低 24 位：

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x23453401
R3	0xabcd1200
R4	0x00000000

③保留 r3 的高 24 位，屏蔽低 8 位：

Register	Value
R0	0x00000000
R1	0x00000000
R2	0x23000000
R3	0xabcd1200

④将 R2 的高 8 位传送到 R3 的低 8 位：

0x00001000	ldr	r2, [pc, #10] ; 0x1018	 --- Current --- R0: 0x00000000 R1: 0x00000000 R2: 0x23000000 R3: 0xabcd1223 R4: 0x00000000
0x00001004	ldr	r3, [pc, #10] ; 0x101c	
0x00001008	and	r2, r2, #-16777216	
0x0000100c	bic	r3, r3, #255	
0x00001010	orr	r3, r3, r2, lsr #24	
0x00001014	b	0x1014	
0x00001018	cnpcs	r5, #16777216	
0x0000101c	blge	0xff345824	

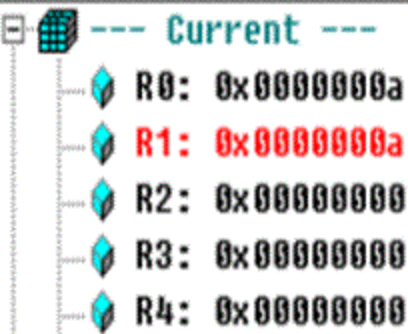
实验三：（例 7）编写一段程序计算 10！

1.程序代码如下：

```
.global _start
.text
.equ num,10
_start:
    mov r0,#num
    mov r1,r0
s1:
    subs r1,r1,#1          /*把 r1-1 放入 r1*/
    mul  r0,r1,r0          /*r0*r1 放入 r0*/
    cmp  r1,#1            /*比较 R1 与 1 的大小*/
    beq  stop
    bne  s1
stop:
    b stop
.end
```

2.分析调试：

① download 下载：

0x00001000	mov	r0, #10	 --- Current --- R0: 0x0000000a R1: 0x0000000a R2: 0x00000000 R3: 0x00000000 R4: 0x00000000
0x00001004	mov	r1, r0	
0x00001008	subs	r1, r1, #1	
0x0000100c	mul	r0, r1, r0	
0x00001010	cmp	r1, #1	
0x00001014	beq	0x101c	
0x00001018	bne	0x1008	
0x0000101c	b	0x101c	
0x00001020	swinu	0x00ffffff	

② 第一次执行 S1, $r1=10-1=9, 10*9=90$, 换成 16 进制是 5a。

0x00001000	mov	r0, #10
0x00001004	mov	r1, r0
0x00001008	subs	r1, r1, #1
0x0000100c	mul	r0, r1, r0
→ 0x00001010	cmp	r1, #1
0x00001014	beq	0x101c
0x00001018	bne	0x1008
0x0000101c	b	0x101c
0x00001020	swinu	0x00ffffff

--- Current ---

R0: 0x0000005a

R1: 0x00000009

R2: 0x00000000

R3: 0x00000000

③ 第二次执行 S1, $r1=9-1=8, 10*9*8=720$, 换成 16 进制是 2d0。

0x00001000	mov	r0, #10
0x00001004	mov	r1, r0
0x00001008	subs	r1, r1, #1
0x0000100c	mul	r0, r1, r0
→ 0x00001010	cmp	r1, #1
0x00001014	beq	0x101c
0x00001018	bne	0x1008
0x0000101c	b	0x101c
0x00001020	swinu	0x00ffffff

--- Current ---

R0: 0x000002d0

R1: 0x00000008

R2: 0x00000000

R3: 0x00000000

④ 依次执行 S1, 到 $r1=1$, 停止, $10*9*8*...*1=$, 换成 16 进制是 375f00。

0x00001000	mov	r0, #10
0x00001004	mov	r1, r0
0x00001008	subs	r1, r1, #1
0x0000100c	mul	r0, r1, r0
→ 0x00001010	cmp	r1, #1
0x00001014	beq	0x101c
0x00001018	bne	0x1008
0x0000101c	b	0x101c
0x00001020	swinu	0x00ffffff

--- Current ---

R0: 0x00375f00

R1: 0x00000001

R2: 0x00000000

R3: 0x00000000

实验四：（例 8）串拷贝（R1 指向源数据串首地址，R0 指向目的数据串的首地址）。

1. 程序代码如下：

```
.global _start
.text
.EQU NUM,8
_start:
    LDR R0,srcstr    /*指向源数据串 R0*/
    LDR R1,dststr    /*指向目标数据串 R1*/
    mov R3,#NUM      /*R3=8*/
```



```

mov LR,PC          /*返回*/
B strcopy          /*调用串拷贝子程序*/
stop: b stop
strcopy:
    LDRB R2,[R0],#1 /*装载字节同时更新地址*/
    STRB R2,[R1],#1 /*存储字节同时更新地址*/
    SUBS R3,R3,#1
    CMP R3,#0      /*判断是否结束*/
    BNE strcopy    /*不是,则继续*/
    MOV PC,LR      /*返回*/

.data
srcstr: .long 1,2,3,4,5,6,7,0 /*定义源数据串*/
dststr: .long 5,3,2,1,4,6,8,0 /*定义目的字符串*/

```

2.分析调试:

①单步跟踪后的结果及存储器的结果显示:

The screenshot shows a debugger window with two panes. The left pane displays assembly instructions with their addresses and comments. The right pane shows a memory dump starting at address 0x1000.

Address	Instruction	Comment
0x00001000	ldr r0, [pc, #28]	; 0x1030
0x00001004	ldr r1, [pc, #28]	; 0x1034
0x00001008	mov r3, #8	
0x0000100c	mov lr, pc	
0x00001010	b 0x1018	
0x00001014	b 0x1014	

Address	Value	Comment
00001000	28 00 9F E5 28 10	(...(.
00001006	9F E5 08 30 A0 E3	...0..
0000100C	0F E0 A0 E1 00 00	
00001012	00 EA FE FF FF EA	
00001018	01 20 00 E4 01 20	...
0000101E	C1 E4 01 30 53 E2	...0S.
00001024	00 00 53 E3 FA FF	..S...

②寄存器的结果显示:

The screenshot shows a debugger window with a register list. The registers R0 through R3 are listed on the left, and R14 through R15, SP, LR, PC, CPSR, and SPSR are listed on the right.

Register	Value
R0	0x00008140
R1	0x00008160
R2	0xffffffff
R3	0x00000000
R14	0x00001014
R15	0x00001014
SP	0x00000000
LR	0x00001014
PC	0x00001014
CPSR	0x600000d3
SPSR	0x00000000

六、实验心得

今天在实验室里,学习使用 Embest IDE 开发环境及 ARM 软件模拟器,掌握简单 ARM 汇编指令,进一步加强了对嵌入式的熟悉和了解。郑老师在兢兢业业的向我们传授实践知识的同时也向我们提问相关理论问题,让我们在学习的过程加深对实践和理论两者之间的联系,知道每一个步骤的发生的原因及产生相应的结果,即对实验的来龙去脉有了更清楚的认识,为今后的学习打下了一定的基础。相信在接下来的实验中,我们会在郑老师的引导下,做起实验来更能得心应手,轻车熟驾!嵌入式相关资料,欢迎下载!