**《微控制器原理、接口与应用》**

**2019年冬季学期课堂作业（2020.12.8）**

# 1、简述ARM920T的工作模式，简述ARM920T寄存器，并对比说明ARM状态下寄存器集和Thumb状态下寄存器集。

ARM920T有7种工作模式,除了用户模式外，其他模式可以自由地访问系统资源和改变模式，称为特权模式, FIQ，IRQ、超级用户、异常中止和未定义这5种模式称为异常模式，分别对应特定的异常出现时进入的模式每种特权模式都有某些附加的通用或状态寄存器，减少模式切换时对现场的保护.

**ARM920T寄存器:**

ARM 共有 37 个 32 b 的寄存器 其中 3 1个是通用寄存器 . 6 个是状态寄存器 。 但在同一时间，对程序员来说并不是所有的寄存器都可见，在某一时刻寄存器是否可见(可被访问 ) ，是由处理器当前的工作状态和工作模式决定的。

通用寄存器可以分为三类， 即未分组寄存器、分组寄存器和程序计数器PC。RO~R7 是未分组寄存器。 这意味着在处理器的所有工作模式下， RO~ R7 所对应的物理寄存器地址是相同的，没有体系结构隐含的特殊用途.

R8~ R14 是分组寄存器,它们所访问的物理寄存器取决于当前处理器的工作模式 。寄存器 R8~ R12 有两组物理寄存器 : 一组为 FIQ 模式 另 一组为除 FIQ 以外的其他模式。 寄存器 R8~ R12没有任何指定的特殊用途 。

寄存器 R 15 用做程序计数器 PC,R 1 5 用于控制程序中指令执行的顺序 。 正常运行时 .PC 指向 CPU 执行的下一条指令 。 每次取指后 PC 的值会自动修改以指向下一条指令 . 从而保证指令按一定的顺序执行 。

Thumb 状态下寄存器是 ARM 状态下寄存器的一个子集。程序员可以直接操作 8 个通用寄存器 R0~R7 。 同样可以操作程序计数器 ( PC) 、堆栈指针寄存器 (SP ) 、链接寄存器( LR) 和 CPSR 。 它们都是各个特权模式下的私有寄存器 。

在 Thumb 状态下访问高地址寄存器:在 Thumb 状态下寄存器 R8~R15( 高地址寄存 器)不是标准寄存器集 。 但是，汇编语言的程序员可以访问它们并用它们进行快速暂存 。 向 R8~ R15 写入或读出数据，可以采用 MOV 指令的某个变型 ， 从 RO~ R7(低地址寄存器)的 某个寄存器传送数据到高地址寄存器，或者从高地址寄存器传送到低地址寄存器 。 还可以 采用 CMP 和 ADD 指令，将高地址寄存器的值与低地址寄存器的值进行比较或相加 。

ARM 和 Thumb 状态寄存器之间的关系如下:

|  |  |
| --- | --- |
| **ARM** | **Thumb状态寄存器** |
| **R0-R7** | **R0-R7** |
| **CPSR,SPSRs** | **CPSR,SPSRs** |
| **SP** | **R13** |
| **LR** | **R14** |
| **PC** | **PC(R15)** |

# 2、简述ARM处理器寻址方式。

1.立即数寻址, **操作数本身包含在指令中**，只要取出指令也就取到了操作数。这个操作数叫做立即数，对应的寻址方式叫做立即寻址

2.寄存器寻址, 就是**利用寄存器中的数值作为操作数**，也称为**寄存器直接寻址**

3.寄存器间接寻址, 就是把**寄存器中的值作为地址**，再通过这个地址去取得操作数，**操作数本身存放在存储器中**。

4.寄存器移位寻址,也叫偏移寻址, 这是ARM指令集特有的寻址方式，**它是在寄存器寻址得到操作数后再进行移位操作**，得到最终的操作数。

5.基址变址寻址, 寄存器基址变址寻址又称为**基址变址寻址**, 它**将寄存器（该寄存器一般称作基址寄存器）中的值与指令中给出的地址偏移量相加，从而得到一个地址，通过这个地址取得操作数**。

6.多寄存器寻址, 可以**一次完成多个寄存器值的传送**

7.堆栈寻址, 堆栈是一种数据结构，按先进后出（First In Last Out，FILO）的方式工作，使用堆栈指针（Stack Pointer, SP）指示当前的操作位置，堆栈指针总是指向栈顶。有四种堆栈类型.

8. 相对寻址,由PC提供基准地址, 指令中的地址码字段作为偏移量, 两者相加后得到的地址即为操作数的有效地址

# 3、分类列出ARM伪指令，并简单说明该伪指令功能。

* + **(1) 符号定义伪指令**

(1) 全局变量声明: GBLA、GBLL和GBLS

(2) 局部变量声明: LCLA、LCLL和LCLS

(3) 变量赋值: SETA、SETL和SETS

(4) 为一个通用寄存器列表定义名称: RLIST

(5) 为一个协处理器的寄存器定义名称: CN

(6) 为一个协处理器定义名称: CP

(7) 为一个VFP寄存器定义名称: DN和SN

(8) 为一个FPA浮点寄存器定义名称: FN

* + **(2) 数据定义伪指令,用于数据表定义、文字池定义、数据空间分配等**

(1) 声明一个文字池: LTORG 。

(2) 定义一个结构化的内存表的首地址: MAP

(3) 定义结构化内存表中的一个数据域: FIELD

(4) 分配一块内存空间，并用0初始化: SPACE

(5) 分配一段字节的内存单元，并用指定的数据初始化: DCB

(6) 分配一段字的内存单元，并用指定的数据初始化: DCD和DCDU

(7) 分配一段字的内存单元，将每个单元的内容初始化为此单元相对于静态基址寄存器的偏移量: DCDO

(8) 分配一段双字的内存单元，并用双精度浮点数据初始化: DCFD和DCFDU

(9) 分配一段字的内存单元，并用单精度浮点数据初始化: DCFS和DCFSU

(10) 指定内存单元存放的是代码，而不是数据: DCI

(11) 分配一段双字的内存单元，并用64位整数数据初始化: DCQ和DCQU

(12) 分配一段半字的内存单元，并用指定的数据初始化: DCW和DCWU

* + **(3) 报告伪指令**

(1) 断言错误: ASSERT

(2) 汇编诊断信息显示: INFO

(3) 设置列表选项: OPT

(4) 插入标题: TTL和SUBT

* + **(4) 汇编控制伪指令,** **用于条件汇编、宏定义、重复汇编控制等**

(1) 条件汇编控制: IF ELSE 和ENDIF

(2) 宏定义: MACRO和MEND

(3) 重复汇编: WHILE及WEND

* + **(5) 杂项伪指令,**

(1) 条件汇编控制: IF ELSE 和ENDIF

(2) 宏定义: MACRO和MEND

(3) 重复汇编: WHILE及WEND

(1) 边界对齐: ALIGN

(2) 段定义: AREA

(3) 指令集定义: CODE16和CODE32

(4) 汇编结束: END

(5) 程序入口: ENTRY

(6) 常量定义: EQU

(7) 声明一个符号可以被其他文件引用: EXPORT和GLOBAL

(8) 声明一个外部符号: IMPORT和EXTERN

(9) 包含文件: GET和INCLUDE

(10) 包含不被汇编的文件: INCBIN

(11) 保留符号表中的局部符号: KEEP

(12) 禁止浮点指令: NOFP

(13) 指示两段之间的依赖关系: REQUIRE

(14) 堆栈8字节对准: REQUIRE8和PRESERVE8

(15) 给特定的寄存器命名RN

(16) 标记局部标号使用范围的界限: ROUT

**4、基于ARM920T处理，编写一段ARM状态的汇编指令，功能是将一个由5位十进制数字字符组成的字符串，转化为整数的代码，例如将字符串“12345”转换为数值12345。**

**用法:** **extern int stoi(char \* str);**

**char a[] = "12345";**

**stoi(a);**

**汇编代码如下**

    area stoi, code, readonly

    global stoi

num equ 4

hed equ 10

start

    mov r3,#0           ;int n = 0 ;

    mov r7,#10

    mov r2,#0           ;iterator

loop1

    ldrb r5,[r0,r2]      ; \*str char need use ldrb instead of ldr

    cmp r5,#0           ;

    bgt judge           ;if \*str != '\0'

    b stop              ;

judge

    cmp r5,#48          ; 0 ascii = 48

    bge step

    b  stop             ;\*str< 0 ,return n in r0

step

    cmp r5,#58          ; 9 asccii = 57

    bge  stop           ; return n in r0

    sub r5,r5,#48          ;

    mul r4,r3,r7          ;r3\*10

    mov r3,r4

    add r3,r3,r5           ; n = n\*10 + (\*str-'0')

    add r2,r2,#1      ;str ++

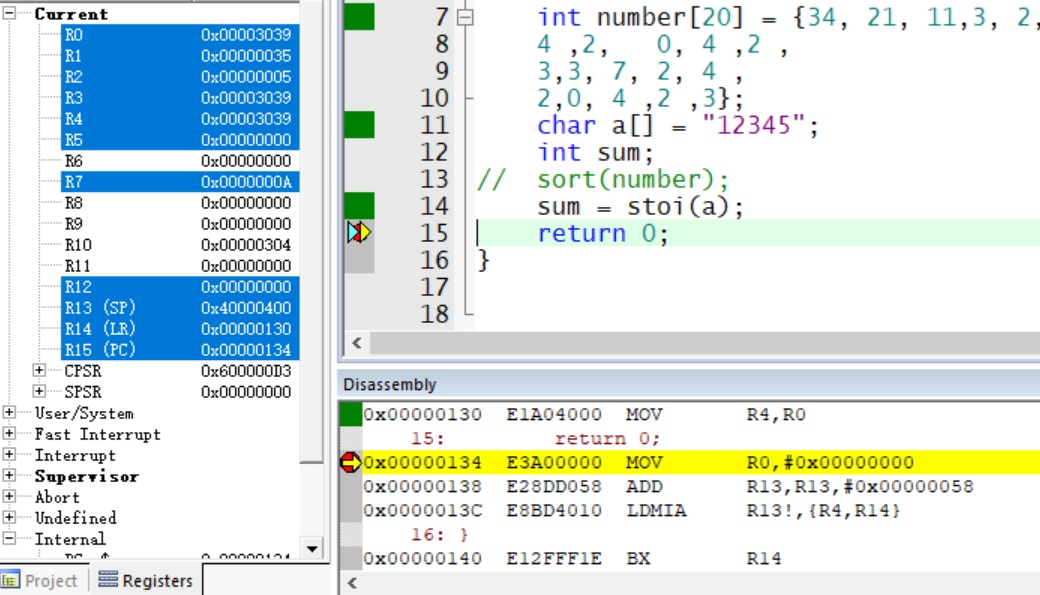
    b loop1

stop

    mov r0,r3 ;

    bx lr

    end





**5、编写一个函数，实现20个无符号32位整数数组的冒泡排序算法.**

    area sort, code, readonly

    global sort

num equ 4

rr1 equ 2

start

init

    mov r6,#1              ;   r6 set init value as  1  ,r6 = j ,in loop

    mov r5,#20              ;  r5 set init value as 20  ,r5 = i ,out loop

loop1

    sub r5,r5,#1          ;r5--

    cmp r5,#0               ; if r5<= 0

    bls stop                ; if r5<= 0 then stop loop

init2

    mov r6,#1             ;r6 =1  ,r6 = j ,in loop

loop2

    ldr r3,[r0,r6, lsl #2]          ;number[j]

    add r6,r6,#-1

    ldr r4,[r0,r6, lsl #2]          ;number[j-1]

    add r6,r6,#1

    cmp r4,r3               ;

    bge swap                ;number[j-1] >=number[j]

    b con               ;

swap

    str r4,[r0, r6, lsl #2]      ;str number[j-1]

    sub r6,r6,#1

    str r3,[r0, r6, lsl #2]     ;str number[j]

    add r6,r6,#1

con

    add r6,r6,#1

    cmp r6,r5

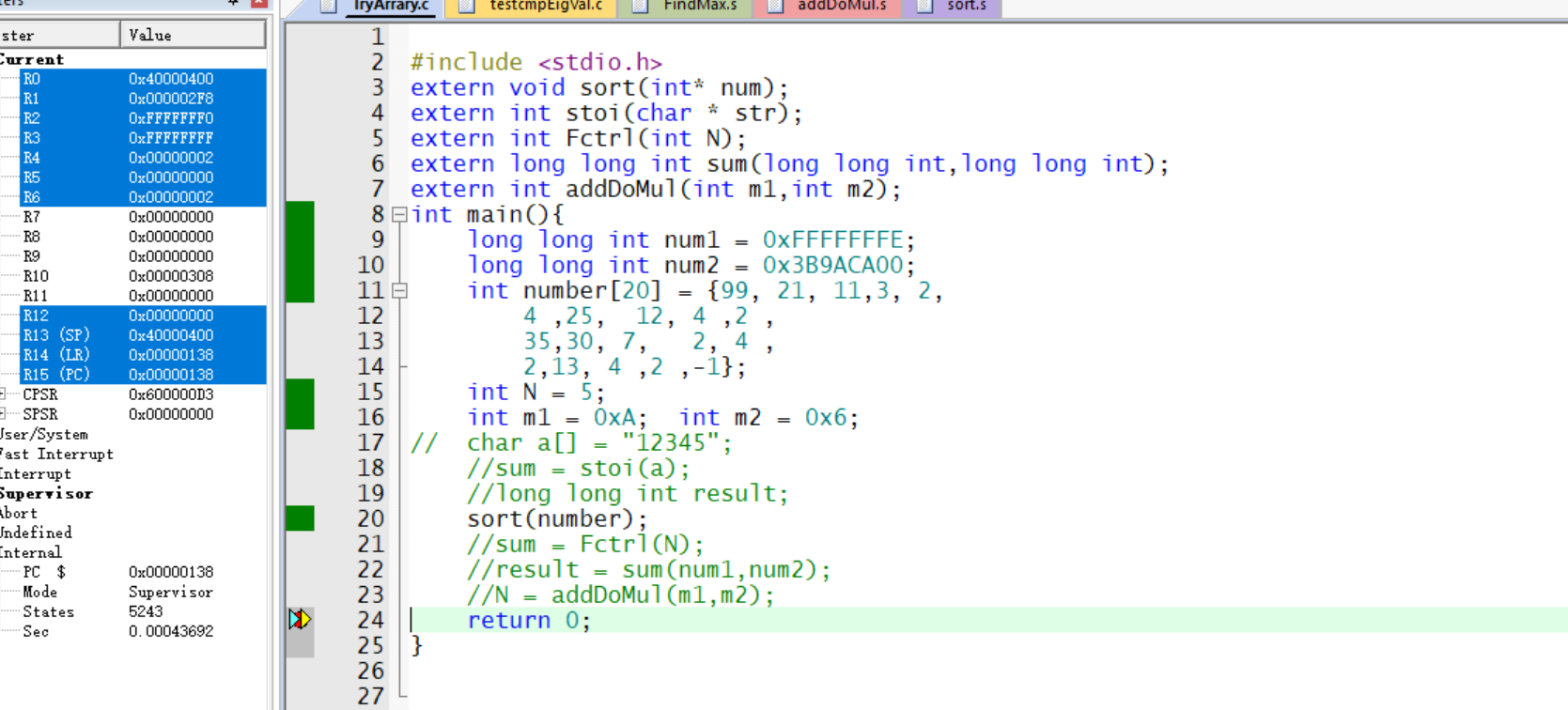
    ble loop2

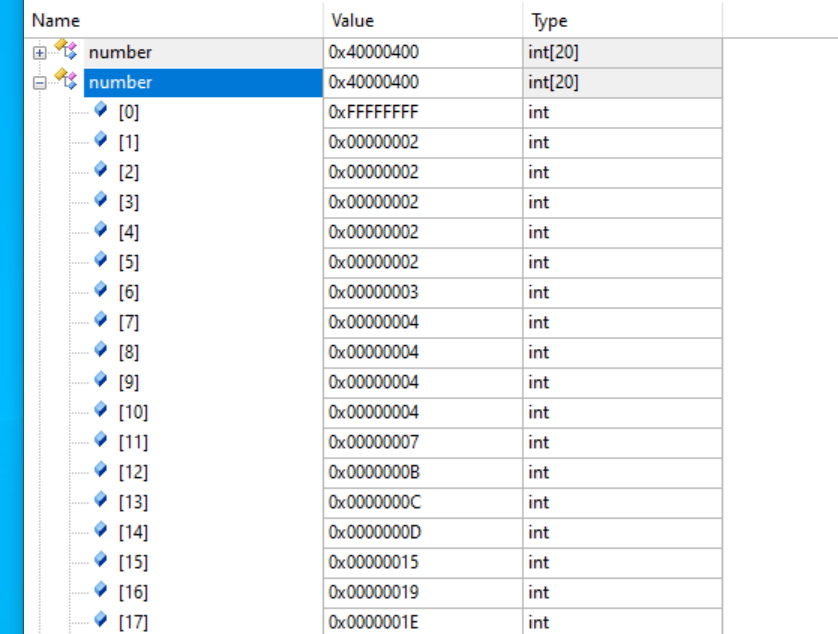
    b loop1

stop

    bx lr

    end

**排序完毕**



**6、用移位指令和加法指令, 实现两个32位无符号整数的乘法函数.**

如果是7\*8的话，很容易通过左移3个二进制位实现。而7\*7就必须分解  
7\*（4+2+1)即7\*4 + 7\*2 + 7\*1,显然7\*7的乘积＝7左移2位+7左移1位+7左移0位。

    area addDoMul, code, readonly

    global addDoMul

start

    mov r5,#0           ; r5 =sum

loop1

    TST r1,#0x01

    beq con             ; if lowest bit is 0, eq is 1 , continue without add r0 into r5

    add r5,r5,r0        ; 7\*1

    mov r0, r0, lsl #1  ;    r0\*2

    mov r1,r1,lsr #1    ;

    cmp r1,#0

    bgt loop1

    b stop

con

    mov r0, r0, lsl #1  ; r0 still need \*2 , 7\*（4+2+1) is 7\*4 + 7\*2 + 7\*1

    mov r1,r1,lsr #1    ;

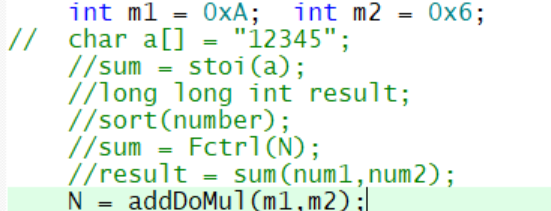
    b loop1

stop

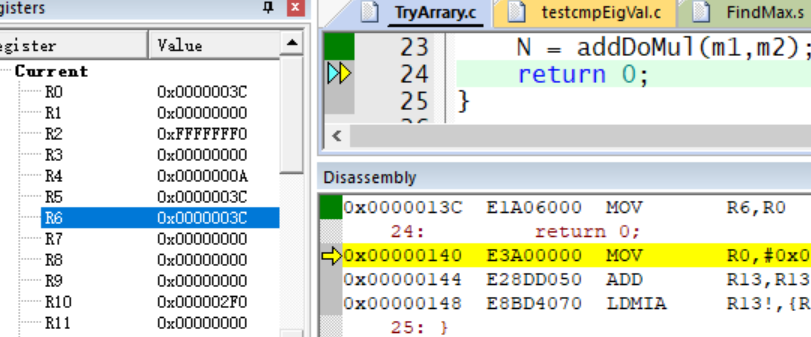
    mov r0,r5

    bx lr

    end



0x1010 \*0x0110 = 0x003C



**7、用汇编程序编写函数，实现如下功能： 1！+2！+3！+ ... + N!, 其中N作为参数存储在R0寄存器（建议先编写函数实现N！，然后再实现题目要求的函数）。**

    AREA Fctrl,CODE,READONLY     ; declare Fctrl

    global  Fctrl

    CODE32        ; declare 32 b ARM

START

    MOV  R6,  #0  ; save sum

    MOV  R7, r0   ;  R7  from N begin

loop2   ;this is  FOR i--

    MOV  R8 , R7    ; init low bit , such as calculate 4!, R7 give 4 to R8

    SUB  R0,R8,#1    ; init R0 = n-1

Loop   ; for calculate i!

    mul  R8 , R0 , R8  ; n\*n(-1)

    SUB  R0 , R0 , #1  ;    R0 from 9 to 0 loop

    cmp  r0,#0

    bgt  Loop    ;if not 0 continue loop ,dont' need cmp is ok . SUBS S will influence CPSR,  compare with zero

    ADD  R6, R8,R6 ;  SUM =SUM + I!

    SUBS R7, R7 ,#1 ;

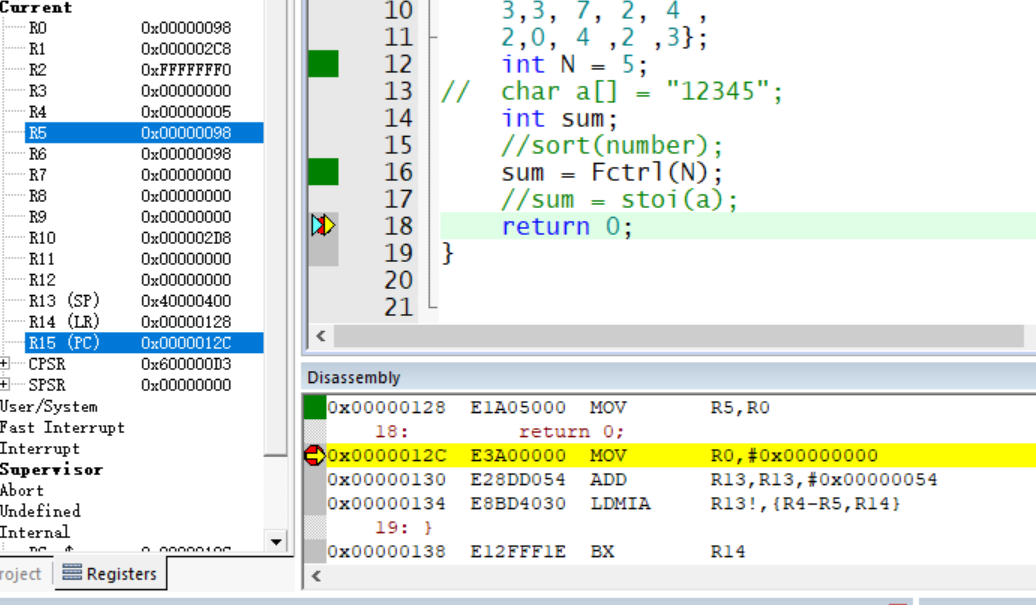
    BNE   loop2

Stop

    mov r0,r6 ;

    bx lr

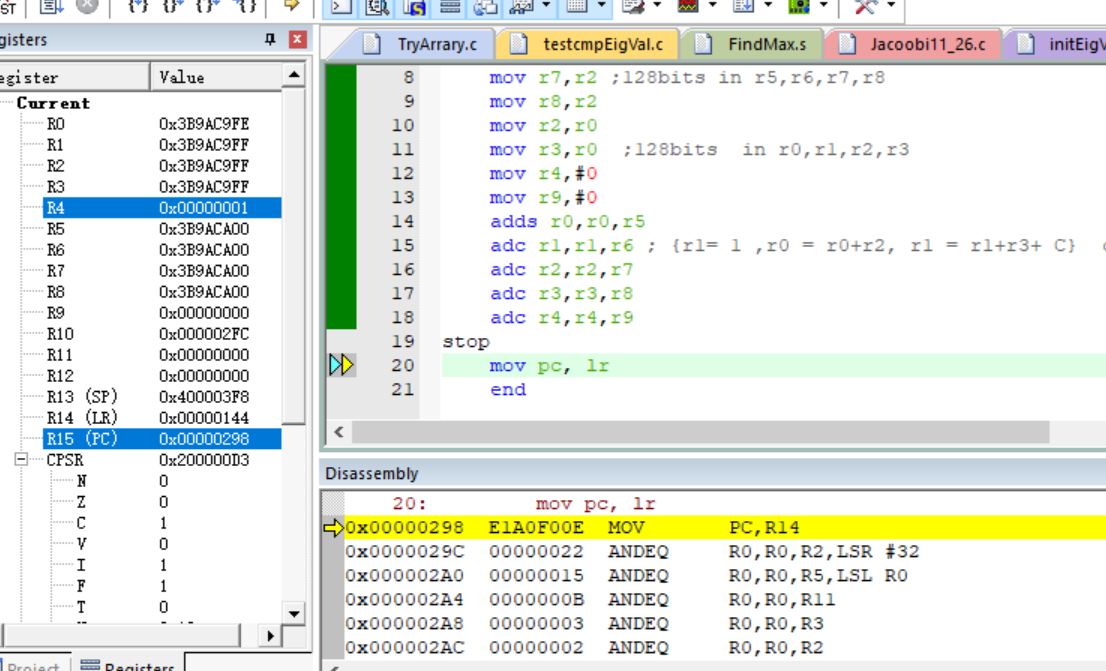
    end



**8、汇编代码编程实现,两个128bit无符号整数的加法函数**

adc可以 改变cpsr的C标志位

一个个寄存器相加就可以了



    area sum, code, readonly

    global sum

start

    mov r1,r0

    mov r5,r2

    mov r6,r2

    mov r7,r2 ;128bits in r5,r6,r7,r8

    mov r8,r2

    mov r2,r0

    mov r3,r0  ;128bits  in r0,r1,r2,r3

    mov r4,#0

    mov r9,#0

    adds r0,r0,r5

    adc r1,r1,r6 ; {r1= 1 ,r0 = r0+r2, r1 = r1+r3+ C}  could command adc  change the C in CPSR?

    adc r2,r2,r7

    adc r3,r3,r8

    adc r4,r4,r9

stop

    mov pc, lr

    end

**9、简述ARM处理器的异常中断。**

**异常中断有六种,如下表**

|  |  |
| --- | --- |
| **复位** | **nRESET 信号为低,ARM920T停止执行, 从增加的字地址处取指令,**  **nRESET 信号变高,ARM920T保存当前PC和CPSR, 设置对应位.**  **强制PC 从0x0000 0000处取得下一条指令.**  **恢复为arm状态开始执行** |
| **未定义指令** | **产生未定义指令陷阱,**  **完成处理后,**  **执行 MOVS PC,R14\_und 退出异常. 恢复CPSR,返回下一条指令** |
| **软件中断SWI** | **进入超级用户模式,**  **通过MOV PC,R14\_svc退出异常处理, 恢复CPSR和PC,返回下一条指令** |
| **Abort** | **异常中止表示当前存储访问不能完成，通过外部的ABORT输入信号来告知处理器有两种类型的异常中止**  **★预取指令异常中断，指令预取时产生**  **★数据异常中断，数据访问时产生**  **Abort机制使得页面虚拟存储器机制得以实现: 当某个地址的数据无效(缺页)，MMU(存储器管理单元)将产生Abort信号.** |
| **IRQ中断** | **IRQ(中断请求)异常是由nIRQ输入低电平引发的普通中断。IRQ 中断相对FIQ中断来说优先级低，当一个FIQ 中断同时进入时它 将被屏蔽。IRQ也可以通过设置CPRS中的I标志来禁止，但只能 在特权模式下进行** |
| **FIQ中断** | **FIQ(快速中断请求)异常通常是用来支持数据传输和通道操作，在**  **ARM状态下，它具有足够多的私有寄存器，从而减少进入中断前的 “context switch”工作。FIQ中断是由外部设备通过拉低nFIQ引脚触发的。FIRQ也可以通过设置CPRS中的F标志来禁止，但只能在特权模式下进行** |

**异常中断**

响应过程: 1.

★将下条指令的地址自动保存到相应的链接(R14)寄存器中

★复制CPSR到相应的SPSR

★根据异常类型强制改变CPSR模式位的值

★PC从相关的异常向量来取得异常处理程序入口地址

★这时也可能设置中断禁止标志，以防止不可控的异常嵌套发生。当处理器处于Thumb状态时发生了异常，将自动地切换到ARM状态.

返回过程: 复制程序状态字从SPSR到CPSR 

如果在进入中断时设置了中断禁止标志，则清除它 

将链接寄存器的值调整后(由指令执行的硬件微体系结构特殊性导致) 赋给PC返回