ARM 程序示例

ARM 汇编

C内嵌/调用汇编

- 参数传递规则见3-46

字符串复制，C调用汇编

C中声明 extern void strcopy(char *d,const char *s);

C中调用 strcopy(dst_str, src_str);

- 汇编文件scopy.s

```
    AREA    SCopy, CODE, READONLY
    EXPORT  strcopy             ;标号可供外部使用
strcopy                         ; r0 目的串指针，r1 源串指针
```

extern void strcopy(char *d, const char *s)

参数传递：从左～右→ 从r0～ r3，多于4个参数时，多于部分压入堆栈。

```
    LDRB   r2, [r1],#1          ; 读取字节并修改地址
    STRB   r2, [r0],#1          ; 存储字节并修改地址
    CMP    r2, #0               ; 检查读取的字节是否为0
    BNE    strcopy              ; 不为0继续
    MOV    pc,lr                ; 否则返回
    END
```

字符串复制

```
#include <stdio.h>
void my_strcpy (const char *src，  char *dest) //声明一个函数
{
    char ch;                          //声明一个字符型变量。
    __asm                             //调用关键词__asm。
    {
    LOOP                              //循环入口。
        LDRB ch，[src]，#1            // ARM指令，ch←[src]， src ←src +1。
        STRB ch，[dest]，#1           // ARM指令， [dest]←ch，[dest]←[dest]+1。
        CMP ch， #0                   //比较ch是否为零，判断字符串是否结束。
        BNE LOOP                      // NE为不相等条件， ch是否为零，否则循环。
    }
}
```

注意：教材此处有2个问题，（1）C和ARM对字母大小写敏感；（2）语句解释错误。

汇编内嵌/访问C

1. 使用IMPORT伪指令声明该全局变量；
2. 使用LDR指令读取该全局变量的内存地址（一般位于程序的数据缓冲池中）；
3. 根据该全局变量的数据类型，使用相应的LDR指令读取其值，使用相应的STR指令修改其值。
    1. 无符号char类型通过LDRB/STRB读写；
    2. 无符号short类型通过LDRH/STRH读写；
    3. int类型通过LDR/STR读写；
    4. 有符号char类型通过LDRSB读取；
    5. 有符号char类型通过STRB写入；
    6. 有符号short类型通过LDRH读取；
    7. 有符号short类型通过STRH写入；
    8. 小于8个字的结构型变量通过LDM/STM读写；
    9. 对于结构型变量的数据成员，可以使用相应的LDR/STR指令访问，此时需要知道该成员变量在结构体中的偏移量。

例子 IMPORT globvl 访问全局变量

```
AREA globals, CODE, READONLY
EXPORT asmsub                       ;导出汇编函数使之可被其它文件中的代码调用
IMPORT globvl                       ;声明C中的全局变量globvl
asmsub                                      ;汇编函数名（标记）
LDR r1, =globvl                     ;将内存地址读入到寄存器r1中
LDR r0, \[r1]                       ;将globvl的值读入到寄存器r0中
ADD r0, r0, #2
STR r0, \[r1]                       ;修改后再将值赋予变量
MOV pc, lr
END
```

例子 IMPORT g 调用C函数

```
int g(int a, int b, int c, int d, int e)
{
    return a+b+c+d+e;
}
```

```
;汇编调用C程序g()计算5个整数i,2i,3i,4i,5i的和，前4个整数由r0至r3传递，第5个由数据栈传递
EXPORT
AREA f, CODE, READONLY
IMPORT g                            ;伪指令IMPORT声明C程序g()
STR lr, [sp, #-4]!      ;
ADD r1, r0, r0          ;假设进入程序f时，r0的值为i,r1的值为2*i
ADD r2, r1, r0          ;r2的值为3*i
ADD r3, r1, r2          ;r3的值为5*i
STR r3, [sp, #-4]!      ;第5个参数5*i通过栈传递
ADD r3, r1, r1          ;r4的值为4*i
BL  g                               ;调用C程序g()
ADD sp, sp, #4          ;调整数据栈指针，准备返回
LDR pc, [sp], #4        ; 返回
END
```

用汇编程序实现C程序功能

```
char Sendbuf[256];
unsigned char  SendLen;
unsigned char pack(unsigned char CMD,char *buf,unsigned char buflen)
{
 unsigned char i,sum=0;
 if ((buflen<=0)||(buflen>255)) return 0;
```

```
 SendLen=0;
 Sendbuf[SendLen++]=0xAA;
 Sendbuf[SendLen++]=CMD;
 Sendbuf[SendLen++]=buflen;
 for (i=0;i<buflen;i++) Sendbuf[SendLen++]=buf[i];
 for (i=0;i<SendLen;i++) sum+=Sendbuf[i];
 Sendbuf[SendLen++]=sum;
 return 0;
}
```

```
AREA HomeWork2,CODE,READONLY
 ENTRY
 EXPORT pack
 ;r0==>CMD    r1==>buf  r2==buflen 入口参数
pack  LDR R4,=addrSendBuf
      LDR R4,[R4]          ;R4===>SendBuf数组起始
      MOV R5,#0            ;R5=SendLen
  ;Head 0xAA
      LDR R6,=0xAA
      STRB R6,[R4],#1
      ADD R5,R5,#1        ;缓冲区字节数
  ;CMD
      STRB R0,[R4],#1
      ADD R5,R5,#1
  ;buflen
      STRB R2,[R4],#1
      ADD R5,R5,#1
  ;buf
LOOP  LDRB R6,[R1],#1
      STRB R6,[R4],#1
      ADD R5,R5,#1
      SUBS R2,R2,#1
      BNE LOOP
  ;CheckSum
      LDR R7,=addrSendBuf
      LDR R7,[R7]          ;R7===>SendBuf
      MOV R8,R5            ;前4部分数据字节数
      MOV R9,#0            ;CheckSum
LOOP1 LDRB R10,[R7],#1
      ADD R9,R9,R10
      SUBS R8,R8,#1
      BNE LOOP1
      STRB R9,[R4],#1
      ADD R5,R5,#1
      ;返回R0
      LDR R7= addrSendLen
```

```
      LDRB R7,[R7]
      STRB R5,[R7]     ;SendLen
      MOV R0,R5        ;return value
      MOV PC,LR

 AREA HomeWorkD,DATA,READWRITE
 addrSendBuf   DCD  SendBuf
 addrSendLen   DCD  SendLen
 IMPORT SendBuf                    ;C变量,直接用就可以了
 IMPORT Sendlen                    ;C变量
 END
```

第三章程序设计题

1. 从有100变量的数组BUFF中，找MAX、MIN

```
area sdata,data,readwrite
MAX DCD 0
MIN DCD 0

area serach,code,readonly
CODE 32
BUFF DCD 1,2,...,100
NUM EQU 100
ENTRY
start:
        LDR R3,=BUFF;设置起始地址
        LDR R4,NUM;取数据个数
        LDR R0,[R3];最大数
        LDR R1,[R3];最小数
loop:
        LDR R2,[R3],#4;取一个数，地址自增
        CMP R2,R0
        MOVGT R0,R2;更新最大数
        CMP R2,R1
        CMPLT R1,R2;更新最小数
        SUBS R4,R4,#1;计数减1
        BNE loop

        LDR R3,=MAX
        STR R0,[R3]
        LDR R3,=MIN
        STR R1,[R3]
stop:;返回系统
        MOV R0,#0x18
```

```
        LDR R1,0x20026
        SWI 0x123456
END
```

2. 实现1+2+…+100

```
AREA ADATA,DATA,READWRITE
sum DCD 0

AREA ADDP,CODE,READONLY
CODE 32
ENTRY
start:
        MOV R0,#0
        MOV R1,#1
loop:
        ADD R0,R0,R1
        ADD R1,R1,#1
        CMP R1,#101
        BLT loop

        LDR R2,=sum
        STR R0,[R2]
stop:;返回系统
        MOV R0,#0x18
        LDR R1,0x20026
        SWI 0x123456
END
```

3. 实现128位add和sub子程序

```
; 加法函数
add128:
    ADDS R0, R0, R4    ; 低32位相加
    ADCS R1, R1, R5    ; 加上进位并与下一对32位相加
    ADCS R2, R2, R6    ; 加上进位并与下一对32位相加
    ADCS R3, R3, R7    ; 加上进位并与最高的32位相加
    MOV PC, LR         ; 返回到调用者
; 减法函数
sub128:
    SUBS R0, R0, R4    ; 相减低32位
    SBCS R1, R1, R5    ; 减去借位并与下一对32位相减
    SBCS R2, R2, R6    ; 减去借位并与下一对32位相减
    SBCS R3, R3, R7    ; 减去借位并与最高的32位相减
```

```
    MOV PC, LR          ; 返回到调用者
; 测试程序
test:
    ; 测试add128
    MOV R0, #1          ; a的低32位
    MOV R1, #2          ; a的第二个32位
    MOV R2, #3          ; a的第三个32位
    MOV R3, #4          ; a的最高32位
    MOV R4, #5          ; b的低32位
    MOV R5, #6          ; b的第二个32位
    MOV R6, #7          ; b的第三个32位
    MOV R7, #8          ; b的最高32位
    BL add128           ; 调用add128
    ...
    BL sub128           ; 调用sub128
    END
```

4. 交换R1的高16位与低16位

```
    ROR R1, R1, #16
    LDR R0, =0x40003000
    STR R1, [R0]
```

5. 找出R0和R1的最大公约数

```
AREA gcd, CODE, READONLY

ENTRY
        ;R0,R1两数
gcd
  CMP       R0, R1            ; 比较a和b大小
  SUBGT  R0, R0, R1  ; if (a>b) a=a-b (if a= = b do nothing)
  SUBLT  R1, R1, R0    ; if (b>a) b=b-a (if a= = b do nothing)
  BNE       gcd                 ; if (a!=b) then 跳转到gcd处继续执行
  MOV     PC, LR            ; 子程序结束，返回
END
```

6. 字符串复制

```
AREA StrCopy, CODE, READONLY
ENTRY
start:
        LDR R0,=src
```

```
        LDR R1,=dst
        BL strcopy
stop:
        MOV,#0x18
        LDR,R1=0x20026
        SWI,0x123456

strcopy:
        LDRB r2,[r0],#1
        STRB r2,[r1],#1
        CMP r2,#0
        BNE strcopy
        MOV PC,LR

AERA  Strings,DATA,READWRITE
        src DCB "First String",0
        dst DCB "Dst String",0
END
```

8. 编写汇编程序实现求 N!

```
AREA Factorial, CODE, READONLY
;参数：R0 = N  返回结果：R0 = N!
FactorialProc
    MOV R1, #1 ; 初始化结果为1
FactorialLoop
    CMP R0, #0 ; 检查N是否为0
    BEQ FactorialEnd ; 如果N为0，跳转到FactorialEnd
    MUL R1, R1, R0 ; 计算阶乘，R1 = R1 * N
    SUB R0, R0, #1 ; 将N减1
    B FactorialLoop ; 跳转回FactorialLoop
FactorialEnd
    MOV R0, R1 ; 将结果复制到R0
    BX LR ; 返回

AREA TestFactorial, CODE, READONLY
ENTRY
    MOV R0, #5 ; 测试5的阶乘
    BL FactorialProc ; 调用阶乘子程序
    ;此时，R0包含5的阶乘，即120
    BX LR ; 返回

END
```

9. 编写汇编程序，将十六进制数0x34A3087C转化为十六进制数的ASCII字符串，结果通过控制台显示

```
AREA HexToStr, CODE, READONLY
ENTRY

start:
    LDR R0, =0x34AE087C ; 要转换的十六进制数
    LDR R1, =output ; 输出字符串的地址
    BL HexToStr ; 转换十六进制数为ASCII字符串
    LDR R0, =output ; 输出字符串的地址
    BL _print ; 打印字符串
    B exit ; 结束

; 参数：R0 = 十六进制数，R1 = 输出字符串的地址
HexToStr:
    PUSH {R4-R7, LR} ; 保存寄存器
    MOV R4, #8 ; 需要转换的十六进制数字数量
HexToStrLoop:
    MOV R5, R0, LSR #28 ; 提取最高4位
    CMP R5, #9 ; 检查是否大于9
    ADDLE R5, R5, #'0' ; 如果小于或等于9，转换为'0'-'9'
    ADDGT R5, R5, #('A'-10) ; 如果大于9，转换为'A'-'F'
    STRB R5, [R1], #1 ; 将ASCII字符存储在字符串中并递增地址
    MOV R0, R0, LSL #4 ; 左移4位以准备下一个十六进制数字
    SUBS R4, R4, #1 ; 减少剩余的十六进制数字数量
    BNE HexToStrLoop ; 如果还有剩余的数字，继续循环
    MOV R5, #0 ; 在字符串末尾添加null字符
    STRB R5, [R1]
    POP {R4-R7, LR} ; 恢复寄存器
    BX LR ; 返回

AREA Data, DATA, READWRITE
output DCB 9 DUP(0) ; 输出字符串，预分配9个字节

END
```

11. 拷贝连续600个字节；方法1：8个字节为单位进行拷贝；方法2：1个字节为单位

```
AREA CopyMem, CODE, READONLY
ENTRY

start:
    ; 方法一：以8个字为单位进行拷贝，不足8个字的以字节为单位进行拷贝
    LDR R0, =0x40003000 ; 源地址
```

```
    LDR R1, =0x40003600 ; 目标地址
    LDR R2, =600 ; 字节数
        STMFD sp!,{r3-r10}
CopyLoop1:
    CMP R2, #8 ; 检查是否有至少8个字节剩余
    BLT CopyLast1 ; 如果没有，跳转到最后的字节拷贝
    LDMIA R0!, {R3-R10} ; 加载8个字并递增源地址
    STMIA R1!, {R3-R10} ; 存储8个字并递增目标地址
    SUBS R2, R2, #8 ; 减少剩余的字节数
    BNE CopyLoop1 ; 如果还有剩余的字节，继续循环

CopyLast1:
    CMP R2, #0 ; 检查是否有剩余的字节
    BEQ exit ; 如果没有，结束
    LDRB R3, [R0], #1 ; 加载一个字节并递增源地址
    STRB R3, [R1], #1 ; 存储一个字节并递增目标地址
    SUBS R2, R2, #1 ; 减少剩余的字节数
    BNE CopyLast1 ; 如果还有剩余的字节，继续循环

        LDMFD sp!,{r3-r10}
    ; 方法二：仅以字节为单位进行拷贝
    LDR R0, =0x40003000 ; 源地址
    LDR R1, =0x40003600 ; 目标地址
    LDR R2, =600 ; 字节数

CopyLoop2:
    LDRB R3, [R0], #1 ; 加载一个字节并递增源地址
    STRB R3, [R1], #1 ; 存储一个字节并递增目标地址
    SUBS R2, R2, #1 ; 减少剩余的字节数
    BNE CopyLoop2 ; 如果还有剩余的字节，继续循环

exit:
    BX LR ; 返回

END
```

16. 排序，折半查找

```
;首地址保存在r0寄存器中，数组长度保存在r1寄存器中。
BubbleSort
    STMFD sp!, {r4-r6, lr}
    SUB r4, r1, #1
OuterLoop
    MOV r5, r4
InnerLoop
    LDR r2, [r0, r5, LSL #2]
```

```
    SUB r3, r5, #1
    LDR r6, [r0, r3, LSL #2]
    CMP r2, r6
    BLE NoSwap
    STR r2, [r0, r3, LSL #2]
    STR r6, [r0, r5, LSL #2]
NoSwap
    SUBS r5, r5, #1
    BPL InnerLoop
    SUBS r4, r4, #1
    BPL OuterLoop
    LDMFD sp!, {r4-r6, pc}

;假设数组已经排序，搜索键保存在r2寄存器中。如果找到，r0将被设置为数组中的位置，否则r0将被设置为-1。
BinarySearch
    STMFD sp!, {r4-r6, lr}
    MOV r4, #0;r4=头
    SUB r5, r1, #1;r5=尾-1
SearchLoop
    ADD r1, r4, r5
    MOV r1, r1, ASR #1;mid/2
    LDR r3, [r0, r1, LSL #2];r3=a[mid/2]
    CMP r2, r3;
    BEQ Found;找到
    BLT Lower;头~mid
    ADD r4, r1, #1;mid+1~尾，r4=头=mid+1
    B Continue
Lower
    MOV r5, r1;r5=尾=mid
Continue
    CMP r4, r5;比较头尾，是否结束
    BLE SearchLoop
    MOV r0, #-1;失败
    B End
Found
    MOV r0, r1
End
    LDMFD sp!, {r4-r6, pc}

;测试程序
start
    LDR r0, =array
    LDR r1, =5
    BL BubbleSort
    MOV r2, #15
    BL BinarySearch
```

```
    B end

array   DCD 20, 15, 10, 5, 0

end
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x123456
```

19. 将以0x40003100为起始地址的连续1000个字节单元数据写入文件

```c
#include <stdio.h>
#define START_ADDR 0x40003200
#define BYTE_COUNT 1000
int main() {
    FILE *file = fopen("MemoryData.dat", "wb");
    if (file == NULL) {
        printf("无法打开文件\n");
        return 1;
    }
    unsigned char *mem_ptr = (unsigned char *)START_ADDR;
    for (int i = 0; i < BYTE_COUNT; i++) {
        fputc(*(mem_ptr + i), file);
    }
    fclose(file);
    printf("内存数据已写入文件\n");
    return 0;
}
```

23. bin to bcd 千位

```
bin_to_bcd:
        MOV R2,#0 ;R2存结果
        MOV R1,#0 ;R1保存每一位的计数
LoopK:
        CMP R0,#1000 ;和千位比较
        ADDGT R1,R1,#1
        SUBS R0,R0,#1000
        ORRLT R2,R2,R1,LSL#12 ;将计数值移到对应位，注意百位和十位移动8、4位
        MOVLT R1,#0 ;清零计数值
        BLT LoopB
        BGT LoopK
;下面百位十位个位略去
```

```
end:
        MOV R0,R2
        MOV PC,LR ;返回
```

38. 查表法跳转，根据r0的值完成跳转，稍作修改即可

```
AREA    Jump, CODE, READONLY
        CODE32
num       EQU     2                ; 跳转表的入口数目
        ENTRY                      ; 程序入口
start
  MOV       r0, #0            ; 传递给子程序的参数
  MOV       r1, #3    ; 传递给子程序的参数
  MOV       r2, #2    ; 传递给子程序的参数
  MOV       R3, #0            ; 调用散列表中的子程序序号
  BL        arithfunc          ; 调用计算地址的子程序
stop
  MOV       r0, #0x18          ; 执行中止
  LDR       r1, =0x20026
  SWI       0x123456
arithfunc
  CMP       r3, #num          ; 比较参数
  MOVHS     pc, lr             ; HS 无符号大于
  ADR       r4, JumpTable      ; 装载地址表首地址
  LDR       pc, [r4, r3, LSL#2]          ; 跳转到相应子程序入口地址处
JumpTable
  DCD      DoAdd
  DCD      DoSub
DoAdd
          ADD       r0, r1, r2         ; =0时的操作
          MOV       pc, lr             ; 返回
DoSub
          SUB       r0, r1, r2         ; =1时的操作
          MOV       pc, lr              ; 返回
          END                         ; 程序结尾
```

41. 存入64个字节数据然后相加

```
#include <stdint.h>
uint64_t add100(unsigned char* addr)
{
        int n=0;
        for (i=0;i<=99;i++)
                n=n+*(addr+i);
```

```
        return n;
}
```

```
AREA ADD100,CODE,READONLY
IMPORT add100
ENTRY
start:
        MOV R0,#1;
LOOP:
        LDR R1,=0X4000F000
        STR R0,[R1],#1
        ADD R0,R0,#1
        CMP R0,#101
        BNE LOOP
add100:
        LDR R0=0x4000F000
        BL add100
stop:
        MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x123456
END
```