第4章 嵌入式程序设计基础

复习概要

48. 主要伪指令.

`DCB` `DCW` `DCD` `SPACE` `FIELD` `MAP` `AREA` `ENTRY` `END` `EQU` `IMPORT` `EXPORT` `ADR` `LDR` .

49. ARM 汇编程序基本结构. (P84.3)

`AREA` **伪指令**定义一个段,并说明所定义段的相关属性;
`ENTPY` **伪指令**标示程序的入口点;
接下来为语句段,执行主代码后,通过返回控制终止应用程序.
在程序的末尾为 `END` **伪指令**,该伪指令通知编译器停止对源文件的处理. 每一个汇编程序段都必须有一条 `END` 伪指令, 指示代码段的结束.

e.g.1.

```
    AREA example, CODE, READONLY    ;定义代码块为example
    CODE 32
    ENTRY                           ;程序入口
Start
    MOV R0, #40                     ;R0=40
    MOV R1, #16                     ;R1=16
    ADD R2, R0, R1                  ;R2=R0+R1
    MOV R0, #0x18                   ;传送到软件中断的参数
    LDR R1, =0x20026                ;传送到软件中断的参数
    SWI 0x123456                    ;通过软件中断指令返回
    END                             ;文件结束
```

e.g.2.

1. 程序框架

```
AREA    Jump, CODE, READONLY
CODE 32               ;ARM指令
ENTRY                 ;程序入口
start                 ;程序开始
    ......
stop
    MOV r0, #0x18   ;执行中止
    LDR r1, =0x20026
    SWI 0x123456    ;返回操作系统
    END             ;程序结束
```

2. 功能程序

```
DoAdd
    ADD r0, r1, r2
    MOV pc, lr      ;子程序返回
DoSub
    SUB r0, r1, r2
    MOV pc, lr      ;子程序返回
```

3. 功能程序地址表(散列表)

```
JumpTable
    DCD DoAdd   ;定义 DoAdd 地址
    DCD DoSub   ;定义 DoSub 地址
```

4. 根据编号计算转移地址,并转移

```
arithfunc
    CMP   r3, #num          ;num是功能程序个数,r3保存将转移的功能程序编号
    MOVHS pc, lr            ;HS无符号大于等于
    ADR   r4, JumpTable     ;装载地址表首地址
    LDR   pc, [r4,r3,LSL #2];跳转到相应功能程序入口地址
```

注意: 转移地址 `=r4+r3×4` , 表中使用 $4\mathrm{B}$ 保存功能程序地址.
完整汇编程序:

```
    AREA    Jump, CODE, READONLY
    CODE 32
num EQU     2           ;地址表中功能程序入口地址的个数
    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      ;散列表中需要转移的功能程序编号
    BL      arithfunc   ;调用子程序
stop
    MOV     r0, #0x18           ;执行中止
    LDR     r1, =0x20026
    SWI     0x123456            ;返回操作系统
arithfunc
    CMP     r3, #num            ;判断功能程序编号的有效性
    MOVHS   pc, lr              ;HS无符号大于等于
    ADR     r4, JumpTable       ;装载地址表首地址
    LDR     pc, [r4,r3,LSL #2]  ;跳转到相应功能程序入口地址
JumpTable
    DCD     DoAdd       ;定义 DoAdd 地址
    DCD     DoSub       ;定义 DoSub 地址
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                 ;程序结束
```

50. 数据变换应用.

e.g. $\mathrm{ABCD} \rightarrow \mathrm{DCBA}$.
方法1:

```
EOR R1, R0, R0, ROR #16 ;R1= A^C B^D C^A D^B
BIC R1, R1, #0xFF0000   ;R1= A^C 0   C^A D^B
MOV R0, R0, ROR #8      ;R0= D   A   B   C
EOR R0, R0, R1, LSR #8  ;R0= D   C   B   A
```

方法2:

```
MOV R2, #0xFF           ;R2=0xFF
ORR R2, R2, #0xFF0000   ;R2=0x00FF00FF
AND R1, R2, R0          ;R1= 0 B 0 D
AND R0, R2, R0, ROR #24 ;R0= 0 C 0 A
ORR R0, R0, R1, ROR #8  ;R0= D C B A
```

51. 求最大公约数.

```
R0=a
R1=b


gcd
    CMP     R0, R1      ;比较a和b大小
    SUBGT   R0, R0, R1  ;if(a>b) a=a-b
    SUBLT   R1, R1, R0  ;if(b>a) b=b-a
    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
```

52. 将串 1 中的字符数据拷贝到串 2, 按字节拷贝.

```
        AREA    StrCopy, CODE, READONLY
        ENTRY                   ;程序入口
start
        LDR     r1, =srcstr     ;初始串的指针
        LDR     r0, =dststr     ;结果串的指针
        BL      strcopy         ;调用子程序执行复制
stop
        MOV     r0, #0x18       ;执行中止
        LDR     r1, =0x20026
        SWI     0x123456
strcopy
        LDRB    r2, [r1], #1    ;加载并且更新源串指针
        STRB    r2, [r0], #1    ;存储且更新目的串指针
        CMP     r2, #0          ;为0,字符串结束
        BNE     strcopy
        MOV     pc, lr
        AREA    Strings, DATA, READWRITE
srcstr  DCB     "First string - source", 0
dststr  DCB     "Second string - destination", 0
        END
```

53. 将数据串 1 中的数据拷贝到串 2, 多字拷贝.

```
    AREA Block, CODE, READONLY  ;段定义
num EQU 20                      ;被拷贝的数据字数
    ENTRY                       ;程序入口
start
    LDR     r0, =src            ;r0=源串指针
    LDR     r1, =dst            ;r1=目的串指针
    MOV     r2, #num            ;r2=拷贝字数
    MOV     sp, #0x400          ;设置堆栈指针(r13)
blockcopy
    MOVS    r3, r2, LSR #3      ;字数/8
    BEQ     copywords           ;少于8个字,转
    STMFD   sp!,{r4-r11}        ;入栈保护
octcopy
    LDMIA   r0!,{r4-r11}        ;从源串加载8个字
    STMIA   r1!,{r4-r11}        ;放入目的串
    SUBS    r3, r3, #1          ;8倍字数减1
    BNE     octcopy             ;未完,继续
    LDMFD   sp!,{r4-r11}        ;出栈恢复
copywords
    ANDS    r2, r2, #7          ;字数%8
    BEQ     stop                ;为0,转
wordcopy
    LDR     r3, [r0], #4        ;从源串加载一个字且指针自增
    STR     r3, [r1], #4        ;存储到目的串
    SUBS    r2, r2, #1          ;字数减1
    BNE     wordcopy            ;未完,继续
stop
    MOV     r0, #0x18           ;执行中止
    LDR     r1, =0x20026
    SWI     0x123456
    AREA    BlockData, DATA, READWRITE
src DCD     1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4
dst DCD     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    END
```

54. APCS.

ARM Procedure Call Standard, **ARM** 过程调用标准,提供了紧凑的编写例程的一种机制.

55. APCS 参数的传递规则.

参数个数可变的子程序:
参数 ≤ 4, 使用 `R0~R3` 传递参数;
参数 > 4, 使用**数据栈**传递参数.
在传递参数时,将所有参数看做是存放在连续的内存单元中的字数据,然后依次将各名字数据传送到寄存器 `R0` `R1` `R2` `R3`; 如果参数 > 4, 将剩余的字数据传送到数据栈中,入栈的顺
序与参数顺序相反,即最后一个字数据先入栈.
参数个数固定的子程序:
对于浮点运算,各个浮点参数按顺序处理,第一个整数参数通过寄存器 `R0~R3` 来传递,其他参数通过数据栈传递.
要点: 如果参数 > 4, **多余**的参数将被压入堆栈.

56. 用 ARM 汇编语言进行程序设计.

教材复习题及课外练习题

教材 P84: 8,9,10

206. `LDR` 指令和 `LDR` 伪操作有什么不同? (P84.6)

`LDR` 伪指令: 是汇编指令,完成汇编阶段的数据传送,汇编后需要用相应的 **CPU** 指令替换. 用于将一个立即数读取到相应的寄存器中,需要用 `=` 来连接地址值. 如 `LDR R0,=0x4000`，表示 `R0=0x4000H`.
`LDR` 加载指令: 是 `CPU` 指令,完成实质性的数据传送. 用于从内存中加载数据到寄存器中. 如 `LDR R0,[R1,#4]`，表示 `R0=[R1+4]`.

207. 编写完整的汇编程序: 统计 20 个字数据中所有位中 1 的个数,如果为奇数则 `R0=1`，如果为偶数则 `R0=0`. (P60.8)

208. 编写完整的汇编程序: 将从存储器地址 `SRC` 开始的 `NUM` 个字的数据复制到存储器地址 `DST` 开始的存储器中. (P60.10)

209. 用 ARM 汇编代码实现以下 C 程序段.

```
if(x-y<3)
    x=0;
else
    y=0;
```

```
SUB     R0, R0, R1
CMP     R0, #3      ;比较
MOVLT   R0, #0
MOVGE   R1, #0
```

210. 阅读下列 C 内嵌 ARM 汇编程序段,并回答问题.

```
#include<stdio.h>
void my_strcpy(char *src,const char *dst){
    int ch;
    _asm{
        loop
        ldrb    ch, [src],  #1
        strb    ch, [dst],  #1
        cmp     ch, #0
        bne     loop
    }
}
int main(){
    const char*a="Hello world";
    char b[20];
    _asm{
        mov r0,a
        mov r1,b
        bl my_strcpy,{r0,r1}
    }
    printf("Original string: %s\n",a);
    printf("Copied string: %s\n",b);
    return 0;
}
```

指出程序的功能,并写出程序段输出的信息.
功能: 数据拷贝
输出信息:

```
Original string: Hello world
Copied string: Hello world
```

211. 在下面的程序段中,汇编语句至少有 1 处不规范的地方,找出并改正,同时解释标记为 1~4 的语句.

```
        AREA    INT,CODE, READONLY
        ENTRY
 start  LDR     R1, =SRCSTR
        LDR     R0, =dststr
        BL      strcopy             ;1
STOP:   B       STOP
strcopy
        ......
        MOV PC, LR                  ;2
        AREA Strings,DATA,READWRITE ;3
srcstr  DCB "First string - source",0
dststr  Space 100
        END                         ;4
```

start 要顶格书写;
=SRCSTR 要小写;
STOP: 应无 :;
Mov 应大小写一致.

1. 跳转到子程序 strcopy
2. 子程序返回
3. 定义一个数据段 Strings 读写属性
4. 结束汇编

212. 下面的内存表首地址为 0x40003100，请问数据域 a Y String 的地址是多少?

```
MAP     0x40003100
a       field   4
b       field   4
x       field   8
Y       field   8
String  field   16
```

0x40003100， 0x40003110， 0x40003118 .

213. STM / LDM 指令中用到寄存器 {R0-R3,R7,R6,R9}，现用一寄存器列表名 pblock 代替用到的寄存器,给出定义指令.

```
pblock RLIST {R0-R3,R7,R6,R9}
```

214. 在 C 语言与汇编程序混合编程中,子程序调用的 ATPCS 规定了哪些基本规则.简要说明寄存器使用规则.

基本规则有三个方面内容,分别是寄存器的使用规则及其相应的名字,数据栈的使用规则,参数传递规则.
寄存器的使用规则:

1. R0~R3 用于传递参数: 子程序通过寄存器 R0~R3 来传递参数,记作 A0~A3，被调用的子程序在返回前无需恢复寄存器 R0~R3 的内容.
2. R4~R11 用于保存局部变量: 记作 V1~V8，子程序进入时必须保存这些寄存器的值,在返回前必须恢复这些寄存器的值.
3. R12 用于子程序间 **scratch** 寄存器: 记作 IP，在子程序的连接代码段中经常会有这种使用规则.
4. R13 用于数据栈指针: 记做 SP，在子程序中 R13 不能用做其他用途. 寄存器 SP 在进入子程序时的值和退出子程序时的值必须相等.
5. R14 用于链接寄存器: 记作 LR，用于保存子程序的返回地址,如果在子程序中保存了返回地址,则 R14 可用作其它的用途.
6. R15 用于程序计数器: 记作 PC，它不能用作其他用途.
7. **ATPCS** 中的各寄存器在 **ARM** 编译器和汇编器中都是预定义的.

215. 下面的程序中,语句 IMPORT strhello 的作用是什么? 执行 main 过程后的结果是什么?

C 语言代码文件 `str.c`：

```
char*strhello="Hello world!\n\0";
```

汇编代码文件 `hello.s`：

```
    AREA    ||.text||, CODE, READONLY
main PROC
    STMFD   sp!,{lr}
    LDR     r0, =strtemp
    LDR     r0, [r0]
    BL      _printf
    LDMFD   sp!,{pc}
strtemp
    DCD strhello
        ENDP
        EXPORT main
        IMPORT strhello
        IMPORT _main
        IMPORT _main
        IMPORT _printf
        IMPORT ||Lib$$Request$$**ARM**lib||, WEAK
    END
```

汇编语言访问 **C** 变量; 打印输出字符串 `Hello world!`.

218. 编写一段 ARM 汇编程序: 循环累加队列 `rjarray` 中的所有元素,直到碰上零值元素,结果放在 `r4` 中.

```
AREA    total,CODE,READONLY
        ENTRY
Start   MOV     r4, #0
        ADR     r0, rjarray
LOOP    LDR     r1, [r0],   #4
        ADD     r4, r4,     r1
        CMP     r1, #0
        BNE     loop
stop    B       stop
rjarray
        DCD     0x11
        DCD     0x22
        ......
        DCD     0x0
        END
```

219. 编写子程序实现将存储器中起始地址 `S1` 开始的 6 个字数据移动到 `S2` 处(要求使用 `LDM` 和 `STM` 语句).

```
STMFD   SP!,{R0-R6,LR}
LDR     R6, =S1
LDMIA   R6!,{R0-R5}
LDR     R6, =S2
STMIA   R6!,{R0-R5}
LDMFD   SP!,{R0-R6,PC}
```

220. 用汇编语言实现以下 C 语言语句.

```
for(auto i=limit;i>=1;i--)
    fact*=i;
```

```
fact    MUL     R0, R1, R0
        SUBS    R1, R1, #1
        BNE     fact
```

221. DCB 用于分配一块字节单元并用伪指令中指定的表达式进行初始化.

222. 用 ARM 汇编语言编写完整的子程序,该程序从 NN 地址处连续读取 20 个字符,并将这 20 个字符复制到目的地址标号 MM 处.

```
AREA    ||.text||,CODE,READONLY
NN      DCB     "12345678901234567890"
MM      DCB     "12345678901234567890"
MY_SUB
        STMFD   R13!,   {R0-R3}
        LDR     R0,     =NN
        LDR     R1,     =MM
        MOV     R2,     #20
LOOP
        LDRB    R3,     [R0],   #1
        STRB    R3,     [R1],   #1
        SUBS    R2,     R2,     #1
        BNE     LOOP
        LDMFD   R13!,   {R0-R3}
        MOV     PC,     LR
        END
```

223. C 语言程序 可以 嵌套加入汇编程序模块.

225. ADR 和 ADRL 伪指令都是将基于 PC 的地址值或基于寄存器的地址值读取到寄存器中,二者的区别是什么?

ADR 小范围地址读取;
ADRL 中等范围地址读取,类似于 ADR , 但比 ADR 读取更大范围的地址.

227. 汇编语言编写的函数 strcopy 用于实现将字符串 s 拷贝到字符串 d . 下列用法正确的是:
C 语言首先声明 extern void strcopy(char*dnstr,const char*snstr); 然后调用函数 strcopy(d,s); 即可实现将字符串 s 拷贝到字符串 d .

A. **C 语言**直接调用函数 strcopy(d,s); 即可实现将字符串 s 拷贝到字符串 d .
B. **C 语言**首先声明 void strcopy(char*dnstr,const char*snstr); , 然后调用函数 strcopy(d,s); 即可实现将字符串 s 拷贝到字符串 d .
C. C 语言首先声明 extern void strcopy(char*dnstr,const char*snstr); 然后调用函数 strcopy(d,s); 即可实现将字符串 s 拷贝到字符串 d .
D. **C 语言**首先声明 void extern strcopy(char*dnstr,const char*snstr); 然后调用函数 strcopy(d,s); 即可实现将字符串 s 拷贝到字符串 d .
解析: Extern is a keyword in C programming language which is used to declare a global variable that is a variable without any memory assigned to it. It is used to declare variables and functions in header files. Extern can be used access variables across C files. Choose **C**.

224. 有 20 个有符号的字数据,依次存放在内存 BUFF 开始字单元中. 试用 ARM 汇编语言编写完整的程序(包括代码段/数据段),从中找出最大值/最小值,并分别放入内存字单元 MAX / MIN 中.

```
        AREA    Search, CODE, READONLY
        CODE 32
NUM     EQU 20              ;定义数据个数
        ENTRY
start
        LDR     R3, =BUFF   ;设置初始地址
        LDR     R4, =NUM    ;取数据个数
        LDR     R0, [R3]    ;R0存放最大数
        LDR     R1, [R3]    ;R1存放最小数
loop
        LDR     R2, [R3], #4;取比较数据
        CMP     R2, R0
        MOVGT   R0, R2      ;若取出的数大于R0中数据,则更新R0
        CMP     R2, R1
        CMPLT   R1, R2      ;若取出的数小于R1中数据,则更新R1
        SUBS    R4, #0x01   ;计数减1
        BNE     loop        ;计数未完,继续
        LDR     R3, =MAX
        STR     R0, [R3]
        LDR     R3, =MIN
        STR     R1, [R3]
stop
        MOV     R0, #0x18                   ;返回系统
        LDR     R1, 0x20026
        SWI         0x123456
        AREA    DefineData, DATA, READWRITE ;数据段
BUFF    DCD     23,34,64,34,...,98,0F5,39   ;定义20个有符号字数据
MAX     DCD     0                           ;最大值单元
MINDCD          0                           ;最小值单元
        END
```

228. 已知 `R0=a,R1=b`，用汇编语言实现

```
if((a!=0x10)&&(b!=0x30))
    a+=b;
```

```
AREA    Exp, CODE, READONLY
a   EQU 0x03
b   EQU 0x04
c   EQU 0x10
d   EQU 0x30
    ENTRY
    CODE 32
start
    LDR r0, =a
    LDR r1, =b
    LDR r2, =c
    LDR r3, =d
    CMP r0, r2      ;a!=0x10
    BEQ stop
    CMP r1, r3      ;b!=0x30
    BEQ stop
    ADD r0, r0, r1  ;a=a+b
stop
    MOV r0, #0x18
    LDR r1, =0x20026
    SWI 0x123456
    END
```

229. 编写汇编程序计算内存 `0x40003000` 开始的 20 个字节单元数据之和,如果和小于 100 则将这 20 个单元复制到内存 `0x40003020` 开始的地址处,否则将这 20 个单元清零.

```
AREA            Exp, CODE, READONLY
ADDR1   EQU     0x40003000
ADDR2   EQU     0x40003200
CNT     EQU     20
VALUE   EQU     100
        ENTRY
        CODE 32
start
        LDR     r0, =ADDR1
        LDR     r2, =CNT
        LDR     r3, =VALUE
        MOV     R4, #0
l0      LDRB    R5, [r0],   #1
        ADD     r4, r4,     r5
        SUBS    r2, r2,     #0x01
        BNE     l0
l1      CMP     r4, r3
        BCC     l3
        LDR     r0, =ADDR1
        LDR     r2, =CNT
        MOV     R4, #0
l2      strb    r4, [r0],   #1
        subs    r2, r2,     #1
        bne     l2
        b       stop
l3      LDR     r0, =ADDR1
        LDR     r1, =ADDR2
        LDR     r2, =CNT
l4      LDRB    r4, [r0],   #1
        STRB    r4, [r1],   #1
        subs    r2, r2,     #1
        bne     l4
stop
        MOV     r0, #0x18
        LDR     r1, =0x20026
        SWI     0x123456
        END
```

230. 说出下面函数的主要功能.

```
void my_strcopy(const char*src,char*dst){
    int ch;
    __asm{
        loop
            LDRB ch,[src],#1
            STRB ch,[dst],#1
            CMP  ch,#0
            BNE  loop
    }
}
```

字符串复制

232. 数组 `a` `b` 分别存放在 `0x4000` `0x5000` 为起始地址的区域内,将以下 C 语言程序转为完整的汇编程序.

```
for(auto i=1;i<8;i++){
    b[i]=a[i];
    if(b[i]==0)
        b[i]=-1;
}
```

```
    AREA    COPY1,CODE,READONLY     ;声明代码段COPY1
SS1 EQU     0x4000                  ;源地址
DD1 EQU     0x5000                  ;目的地址
NUM EQU     8                       ;字计数
    ENTRY                           ;标识程序入口
    CODE 32                          ;声明32位ARM指令
START
    LDR     R0, =SS1                ;设置源地址
    LDR     R1, =DD1                ;设置目标地址
    MOV     R2, #0                  ;设置字计数初始值为0
LOOP
    LDR     R3, [R0],   #4          ;取一个字源数据
    CMP     R3, #0                  ;字源数据与0比较
    MVNEQ   R3, #0                  ;为0,送-1
    STR     R3, [R1],   #4          ;送目标字单元
    ADD     R2, R2,     #1          ;字计数加1
    CMP     R2, #NUM                ;达到预定的NUM吗?
    BCC     LOOP                    ;未达到,继续
HALT
    B       HALT                    ;达到,暂停
END
```

234. 编写程序将 R1 的高 8 位数据转移到 R0 的低 8 位中,保存到地址 0x40003000 单元内.

```
AREA    ByteCopy, CODE, READONLY
COUNT   EQU 0x44332211
A1      EQU 0x40003000
        ENTRY
start
        LDR R1, =COUNT
        MOV R0, R1, LSR #24
        LDR R1, =A1
        STR R0, [R1]
        END
```

236. 修改下面嵌入式汇编的错误

```
__asm{
    MOV R0, x
    ADD y,  R0, x/y
}
```

```
__asm{
    mov var,x
    add y,  var,x/y
}
```

235. 编写汇编程序计算 $x!$ 的值.

```
X       EQU     9                           ;X=9
n       EQU     8                           ;n=8
        AREA    Example4, CODE, READONLY    ;声明代码段Example4
        ENTRY                               ;标识程序入口
        CODE 32                              ;声明32位ARM指令
START   LDR     SP, =0x40003F00             ;设置堆栈
                                    ;满递减堆栈,使用STMFD/LMDFD指令
        LDR     R0, =X
        LDR     R1, =n
        BL      POW                         ;调用子程序POW,返回值为R0
HALT    B       HALT

;名称        POW
;功能        整数乘方运算.
;入口参数    R0底数;R1指数
;出口参数    R0运算结果
;占用资源    R0,R1
;说明        本子程序不考虑溢出问题

POW
        STMFD   SP!,{R1-R12,LR} ;寄存器入栈保护
        MOVS    R2, R1          ;将指数值复制到R2,并影响条件码标志
        MOVEQ   R0, #1          ;若指数为0,则设置R0=1
        BEQ     POW_END         ;若指数为0,则返回
        CMP     R2, #1
        BEQ     POW_END         ;若指数为1,则返回.(此时R0没有被更改)
        MOV     R1, R0          ;设置DO_MUL子程序的入口参数R0,R1
        SUB     R2, R2, #1      ;计数器R2-=1
POW_L1  BL      DO_MUL          ;调用DO_MUL子程序,R0*=R1
        SUBS    R2, R2, #1      ;每循环一次,计数器R2-=1
        BNE     POW_L1          ;若计数器R2!=0,跳转到POW_L1
POW_END LDMFD   SP!,{R1-R12,PC} ;寄存器出栈,返回

;名称        DO_MUL
;功能        32位乘法运算
;入口参数    R0乘数;R1被乘数
;出口参数    R0计算结果
;占用资源    R0,R1
;说明        本子程序不会破坏R1

DO_MUL  MUL R0, R1, R0  ;R0*=R1
        MOV PC, LR      ;返回
        END
```

239. 用汇编程序实现如下功能: 将 R1 的高 16 位数据与低 16 位交换,保存到地址 0x40003000 内.

```
A1  EQU 0x40003000
start
    LDR R3, =A1
    MOV R0, R1, LSR #16
    AND R2, R1, #0xff
    ORR R0  R0  R2
    STR R0, [R3]
    END
```

238. 分析程序实现什么功能

```
num EQU 2
    MOV R0, #0
myfunc
    CMP R0, #num
    BCS DoError
    ADR R3, JumpTable
    LDR PC, [R3,R0,LSL#2]
JumpTable
    DCD DoAdd
    DCD DoSub
DoAdd
    ......
DoSub
    ......
DoError
    ......
```

查表或散转.

243. 以下程序片段的主要功能是什么,程序中有何错误,如果有请改正.

```
    CODE 32
A1  ADR R0,T1
    BX R0
    CODE16
T1  MOV R0,#10
    ......
    END
```

ARM 到 **THRUM** 状态切换.

有错误，`A1 ADR R0,T1` 应该改为 `A1 ADR R0,T1+1` .

246. 已知 32 位变量 `X Y` 存放在存储器的地址 `0x90010 0x90014` 中,要求实现 `Z=X+Y` , 其中 `Z` 的值存放在 `0x90018` 中.

```
AREA    EX4_41, CODE, READONLY
        ENTRY
        CODE 32
START   LDR R0, =0x90010    ;R0=&X
        LDR R1, [R0],   #4  ;R1=X
        LDR R2, [R0],   #4  ;R2=Y
        ADD R1, R1,     R2  ;R1+=R2
        STR R1, [R0]        ;[&Z]=R1
        B   START
        END
```

247. 已知 32 位有符号数 `X` 存放在存储器的地址 `0x90010` 中,要求实现 `X=abs(X)` .

```
AREA    EX4_42, CODE,READONLY
        ENTRY
        CODE 32
START   LDR     R1, =0x90010    ;R1=&X
        MOV     R0, #0          ;R0=0
        LDR     R2, [R1]        ;R2=X
        CMP     R2, #0          ;X与0比较,影响标志位
        SUBLT   R2, R0, R2      ;X<0执行R0-=R2
        STR     R2, [R1]        ;保存结果
        B       START
        END
```

248. 已知 32 位有符号数 `X` 存放在存储器的地址 `0x90010` 中,要求实现 `X=X<0?-1:X>0`.

```
AREA    EX4_43, CODE, READONLY
        ENTRY
        CODE 32
START   LDR R1, =0x90010    ;R1=&X
        LDR R2, [R1]        ;R2=X
        CMP R2, #0          ;与0比较,影响标志位
        BEQ ZERO            ;为0则跳转到ZERO
        BGT PLUS            ;大于0则跳转到PLUS处理
        MOV R0, #-1         ;否则小于0,R0=-1
        B   FINISH          ;跳转到结束
PLUS    MOV R0, #1          ;R0=0
        B   FINISH          ;跳转到结束
ZERO    MOV R0, #0          ;R0=0
FINISH  STR R0, [R1]        ;保存结果
        B   START
        END
```

250. 编制程序使 $S=1+\sum_{i=1}^{10} i(i+1)$.

```
AREA    EX4_45, CODE,READONLY
        ENTRY
        CODE 32
START   MOV R0, #1      ;R0用作累加,初值1,S
        MOV R1, #2      ;R1用作第一个乘数,初值2,i
REPEAT  ADD R2, R1, #1  ;R2用作第二个乘数,初值3,i+1
        MUL R3, R2, R1  ;R3=R1*R2,i*(i+1)
        ADD R0, R0, R3  ;R0+=R3
        ADD R1, R1, #1  ;R1+=1
        CMP R1, #10     ;比较i和10
        BLE REPEAT      ;未完则重复
        B START
        END
```

251. 编制程序,求两个数组 `DATA1` 和 `DATA2` 对应的数据之和,并把和数存入新数组 `SUM` 中,计算一直进行到两数之和为零时结束,并把新数组的长度存于 `R0` 中.

```
AREA    BlockData,DATA,READWRITE    ;定义数据段
DATA1   DCD 2,5,0,3,-4,5,0,10,9     ;数组 DATA1
DATA2   DCD 3,5,4,-2,0,8,3,-10,5    ;数组 DATA2
SUM     DCD 0,0,0,0,0,0,0,0,0       ;数组 SUM
AREA    Ex4_46, CODE, READONLY      ;定义代码段
        ENTRY
        CODE 32
START   LDR     R1, =DATA1          ;数组 DATA1 的首地址存入到 R1
        LDR     R2, =DATA2          ;数组 DATA2 的首地址存入到 R2
        LDR     R3, =SUM            ;数组 SUM 的首地址存入到 R3
        MOV     R0, #0              ;计数器 R0 的初始值置 0
LOOP    LDR     R4, [R1],   #04     ;取 DATA1 数组的一个数,同时修改地址指针
        LDR     R5, [R2],   #04     ;取 DATA1 数组的一个数,同时修改地址指针
        ADDS    R4, R4,     R5      ;相加并影响标志位
        ADD     R0, R0,     #1      ;计数器加 1
        STR     R4, [R3],   #04     ;保存结果到 SUM 中,同时修改地址指针
        BNE     LOOP                ;若相加的结果不为 0 则循环
        END
```

252. 在以 BUF 为首地址的字存储区中存放有 10 个无符号数 0xff,0x00,0x40,0x10,0x90,0x20,0x80,0x30,0x50,0x70，请将它们按从小到大的顺序排列在 BUF 存储区中.

```
N       EQU 10
AREA    EX4_47, CODE, READONLY
        ENTRY
        CODE 32
START   LDR R0, =BUF            ;指向数组的首地址
        MOV R1, #0              ;外循环计数器
        MOV R2, #0              ;内循环计数器
LOOPI   ADD R3, R0, R1, LSL #2  ;外循环首地址放入 R3
        MOV R4, R3              ;外循环首地址放入 R4
        ADD R2, R1, #1          ;内循环计数器初值
        MOV R5, R4              ;内循环下一地址初值
        LDR R6, [R4]            ;取内循环第一个值 R4
LOOPJ   ADD R5, R5, #4          ;内循环下一地址值
        LDR R7, [R5]            ;取出下一地址值 R7
        CMP R6, R7              ;比较
        BLT NEXT                ;小则取下一个
        SWP R7, R6, [R5]        ;大则交换,最小值 R6
        MOV R6, R7
NEXT    ADD R2, R2, #1          ;内循环计数
        CMP R2, #N              ;循环中止条件
        BLT LOOPJ               ;小于 N 则继续内循环,实现比较一轮
        SWP R7, R6, [R3]        ;否则内循环一轮结束
        ADD R1, R1, #1          ;外循环计数
        CMP R1, #N-1            ;处循环中止条件
        BLT LOOPI               ;小于 N-1 继续执行外循环
        B   START
AREA    BlockData, DATA, READWRITE
BUF     DCD 0x0ff,0x00,0x40,0x10,0x90,0x20,0x80,0x30,0x50,0x70
        END
```

255. 用完整 ARM 汇编语言编写程序: 使用 LDR 指令读取 0x40001000 上的数据,将数据加 3, 若结果 < 100 则使用 STR 指令把结果写回原地址,若结果大于等于 100, 则把 0 写回 原地址. 然后再次读取 0x40001000 上的数据,将数据 +1，判断结果是否 $<$ 100, 周而复始循环.

```
COUNT   EQU     0x40001000      ;定义一个变量,地址为0x40001000
AREA    Example2, CODE, READONLY;声明代码段 Example2
        ENTRY                   ;标识程序入口
        CODE 32                  ;声明32位ARM指令
START   LDR     R1, =COUNT      ;R1=COUNT
        MOV     R0, #0          ;R0=0
        STR     R0, [R1]        ;[R1]=R0,即设置[COUNT]为0
LOOP    LDR     R1, =COUNT
        LDR     R0, [R1]        ;R0=[R1]
        ADD     R0, R0,#3       ;R0+=1
        CMP     R0, #100        ;R0与100比较,影响条件码标志
        MOVHS   R0, #0          ;若R0>=100,则此指令执行:R0=0
        STR     R0, [R1]        ;[R1]=R0,即保存COUNT
        B       LOOP
        END
```

256. 用 ARM 汇编语言编写完整的子程序,该程序从 0x40001000 地址处连续读取 100 个字符,并将这 100 个字符复制到目的地址标号 DIST 处.

```
AREA    ||.text||, CODE, READONLY
DIST    DCB "123456789...01234567890"   ;100个字符
        CODE 32
        ENTRY
MY_SUB
        STMFD   R13!,   {R0-R3}
        MOV     R0,     =#0x40001000
        LDR     R1,     =DIST
        MOV     R2,     #100
LOOP
        SUBS    R2,     R2,     #1
        LDRB    R3,     [R0],   #1
        STRB    R3,     [R1],   #1
        BNE     LOOP
        LDMFD   R13!,   {R0-R3}
        MOV     PC,     LR
        END
```

257. 存储器从 `0x400000` 开始的 100 个单元中存放着 ASCII 码.编写程序,将其所有的小写字母转换成大写字母,对其它的 ASCII 码不做变换.

```
    MOV     R0, #0x400000
    MOV     R1, #0
LP
    LDRB    R2, [R0,R1]
    CMP     R2, #0x61
    BLO     NEXT
    CMP     R2, #0x7B   ;'z'=0x7a
    SUBLO   R2, R2, #0x20
    STRBLO  R2, [R0,R1]
NEXT
    ADD     R1, R1, #1
    CMP     R1, #100
    BNE     LP
```

258. 以下对伪指令的解释错误的是 `Baud EQU 2400` 为定义一个 16 位常量 Baud 值为 2400 .

A. `DCW 0x12` 在内存区域分配半字的内存空间并初始化为 `0x0012`

B. `CODE 32` 伪指令通知汇编器,其后的指令序列为 32 位的 **ARM** 指令

C. `Baud EQU 2400` 为定义一个 16 位常量 `Baud` 值为 `2400`

D. `EXTERN SUB1` 当前文件引用外部标号 `SUB1`

解析: 伪指令 `DCW` 用于分配一片连续的**半字**存储单元并用指定的数据初始化; `CODE 32` 通知编译器,其后的指令序列为 32 位的 **ARM** 指令; `EQU` 是等于伪指令,用于为程序中的常量/标号等定义一个等效的字符名称; `EXTERN` 是外部标号引用声明伪指令,用于通知编译器要使用的标号在其他的源文件中定义,但要在当前文件中引用. 故 **C** 项错误.