ARM汇编语言编程详解

硅谷芯微嵌入式学院 技术贡献

网址：www.threeway.cc

硅谷芯微 · 技术中心
www.threeway.cc

4.1 汇编语言

使用汇编语言编写程序，它的特点是程序执行速度快，程序代码生成量少，但汇编语言是一种不易学习的编程语言，并且可读性较差，这种语言属于低级语言。每一种汇编语言对应每一款芯片，使用这种语言需要对硬件有深刻的了解。在通常情况下，可以使用汇编语言编写驱动程序、需要严格计算执行时间的程序以及需要加速执行的程序。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.1 ARM汇编程序的格式（1）

先介绍一个例子来说明ARM汇编程序的格式。

例1 计算20+8，结果放入R0寄存器。

```
        AREA    Buf, DATA, READWRITE            ; 声明数据段Buf
Count       DCB     20                          ; 定义一个字节单元Count


        AREA    Example, CODE, READONLY         ; 声明代码段Example
            ENTRY                               ; 标识程序入口
            CODE32                              ; 声明32位ARM指令
START
            LDRB    R0, Count                   ; R0 = Count =20
            MOV     R1, #8                      ; R1 = 8
            ADD     R0, R0, R1                  ; R0 = R0 + R1
            B       START
        END
```

硅谷芯微 · 技术中心
www.threeway.cc

• 4.1.1 ARM汇编程序的格式（2）

例1中定义了两个段：数据段Buf和代码段Example。数据段中定义了字节单元Count，其中Count用来保存一个被加数；代码段中包含了所有源程序代码，程序中首先读取Count字节单元的内容，然后与立即数8相加，计算结果保存到R0中。

由例1可见，ARM汇编语言的源程序是分段的，由若干个段组成一个源程序。源程序的一般格式为：

```
        AREA    name1, attr        ；声明特定的段
标号            语句1              ；语句
                ...
                语句n

        AREA    name2, attr
标号            语句n+1
                ...
                语句n+m
        END                        ；结束符
```

旺谷芯微·技术中心
www.threeway.cc

- 4.1.1 ARM汇编程序的格式（3）

每一个段都有一个名字，并且段名是唯一的。每个段以符号AREA作为段的开始，以碰到下一个符号AREA作为该段的结束。段都有自己的属性，如是代码段（CODE）还是数据段（DATA），是只读（READONLY）还是可读写（READWRITE）？这些属性可以在attr栏中设定。

注意：符号AREA和END都不能顶格写，只有标号可以而且必须顶格写。

• 4.1.1.1 ARM汇编程序的书写格式（1）

ARM汇编源程序是由若干段组成的，而一个段又是由若干个语句行组成。语句就是完成一个动作的说明。源程序中的语句可以分为以下两种类型：

■ 指令性语句：汇编程序会把指令性语句翻译成机器代码，然后利用这些机器代码命令处理器执行某些操作。如由MOV、ADD等指令构造的语句。

■ 指示性语句：汇编程序并不把它们翻译成机器代码，只是用来指示、引导汇编程序在汇编时进行一些操作。如由ENTRY、AREA等指令构造的语句，我们也称这些指令为伪指令。

从例1可知，语句行的基本格式如下：

[标号]　　<指令>　　<操作数>　　　　　[;注释]

硅谷芯微·技术中心
www.threeway.cc

• 4.1.1.1 ARM汇编程序的书写格式（2）

在一条语句中，[ ]号中的内容是可选的。在书写ARM汇编程序时，需要注意以下3点：

■ 标号必须在一行的顶格书写，其后面不要加“：”，对于变量的设置、常量的定义，其标识符必须在一行的顶格书写；而所有指令均不能顶格书写。

■ 汇编器对标识符大小写敏感，书写标号及指令时字母大小写要一致。在ARM汇编程序时，一个ARM指令、伪指令、寄存器名可以全部为大写字母，也可以全部为小写字母，但不要大小写混合使用。

■ 注释使用“；”。注释内容由“；”开始到此行结束，注释可以在一行的顶格书写。

例2 某一段错误的汇编语言程序。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.1.1 ARM汇编程序的书写格式（3）

例2 某一段错误的汇编语言程序。

```
    START   MOV    R0, #1            ；标号START没有顶格写
ABC:        MOV    R1, #2            ；标号后不能带:
MOV         R2, #3                   ；指令不允许顶格书写
LOOP        Mov    R2, #3            ；指令中大小写混合
            B      loop              ；无法跳转到loop标号，只有LOOP标号
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.1.2 语句行的符号（1）

任何一个汇编源程序都是由符号组成的。符号分为两大类：指令助记符和用户定义符。指令助记符包括ARM指令、伪指令等，这些符号都是预先定义好的，且具备专用的目的和功能；用户定义符是由用户在编写汇编程序时自行定义的，只在本程序中有意义，不具备通用性。本节所讲的符号特指用户定义符，符号的命名需注意以下规则：

- 符号由大小写字母、数字以及下划线组成。
- 符号不能以数字开头（局部标号除外）。
- 符号区分大小写，且所有字符都是有意义的。
- 符号在其作用域范围内必须是唯一的。
- 符号不能与系统内部或系统预定义的符号同名。
- 符号不要以指令助记符、伪指令同名。

符号可以代表地址、数值、变量。当符号代表地址时又称为标号，符号代表某个特定数值时又称为符号常量，

硅谷芯微·技术中心
www.threeway.cc

• 4.1.1.2 语句行的符号（2）

符号代表变量时又称为变量名。所以符号有3个用途：标号、符号常量、变量名。

(1) 标号：

标号代表一个地址，段内标号的地址在汇编时确定，而段外标号的地址值在链接时确定。根据标号的生成方式可以分为以下3种：

■ 基于PC的标号：该标号是位于目标指令前的标号或程序中的数据定义伪指令前的标号。这种标号在汇编时将被处理成PC值加上或减去一个数字常量。它常用于表示跳转指令的目标地址，或者代码段中所嵌入的少量数据。

■ 基于寄存器的标号：该标号通常用MAP和FILED伪指令定义，也可以用于EQU伪指令定义。这种标号在汇编时被处理成寄存器的值加上或减去一个数字常量。它常用于访问位于数据段中的数据。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.1.2 语句行的符号（3）

■ 绝对地址：绝对地址是一个32位的数字量，可寻址的范围为0～232-1，可以直接寻址整个内存空间。

例3 标号举例。

```
        …                    ; …表示省略的程序
        B       START        ; 程序跳转到START标号处
        …
START                        ; START标号，这是一个基于PC的段内标号
        …
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.1.2 语句行的符号（4）

（2）符号常量：

在程序运行过程中，其值不能被改变的量称为常量。常量区分为3种不同的类型：

■ 数字常量：数字常量表示某个特定的数字，如0、5、-19、0xF8都是数字常量。同一个数字常量可以有十进制数、十六进制数等多种表达方式。

■ 字符常量：字符常量由一对单引号及中间字符串表示，标准C语言中的转义符也是也可使用。如果需要包含双引号或“$”，必须使用“”或“$$”代替。如‘H’是一个字符常量，“Hello World”是一个字符串常量。

■ 布尔常量：布尔常量由括号{}和逻辑值（TRUE、FALSE）表示。逻辑真为{TRUE}，逻辑假为{FALSE}。

为了程序书写的方便，可以用一个标识符来代表一个常量，称这个标识符为符号常量，即标识符形式的常量。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.1.2 语句行的符号（5）

例4 用EQU伪指令定义数字符号常量。

```
T_bit       EQU     0x20            ；定义数字常量T_bit，其值为0x20
PLLCON      EQU     0xE01FC080      ；定义PLLCON，值为0xE01FC080
```

例子中定义了两个数字符号常量：T_bit和PLLCON。所以程序中用到这两个符号常量时，在程序链接时就会被相应的值0x20、0xE01FC080所替代。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.1.2 语句行的符号（6）

（3）变量名：

变量是指存放在存储单元的操作数，并且它的值可以改变。变量名代表了一个变量，当程序中要用到变量时，只需要引用对应的变量名。

实际上，变量名是一个符号地址，当程序编译链接时，系统会给每一个变量名分配一个内存地址。在程序中从变量中取值，实际上是通过变量名找到相应的内存地址，从其存储单元中读取数据。

按照变量的作用范围可分为全局变量和局部变量；按照变量的数值类型可分为数字变量、字符变量和逻辑变量。根据两种类型的组合，变量共具有6种类型：全局数字变量、全局逻辑变量、全局字符串变量、局部数字变量、局部逻辑变量、局部字符串变量。具体对这些类型的变量如何声明、赋初值，将在4.1.2小节符号定义伪指令中详细介绍。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2 伪指令语句（1）

汇编语言程序由机器指令、伪指令和宏指令组成。伪指令不像机器指令那样在处理器运行期间由机器执行，而是在对源程序进行汇编期间由汇编工具处理的操作，它们可以完成如符号定义、数据定义、分配存储区、指示程序开始结束等功能。本小节只说明一些常用的伪指令。另外，还有一些伪指令可查看相关手册。

在前面的ARM指令集章节中，已经接触了几条常用的ARM伪指令，如ADR、ADRL、LDR、NOP等。把它们和指令集一起介绍是因为它们在汇编时会被合适的机器指令代替，实现真正机器指令操作。伪指令大概可分为以下6种类型：

- ARM伪指令，如ADR、LDR、NOP等，本节不再重复介绍。
- 符号定义伪指令。
- 段及段属性定义伪指令。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2 伪指令语句（2）
  - 数据定义伪指令。
  - 汇编控制伪指令。
  - 杂项伪指令。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.1 符号定义伪指令（1）

符号定义伪指令用于定义ARM汇编程序的常量、标号和变量，对变量进行赋值等操作。符号定义伪指令包括EQU伪指令、变量声明伪指令、变量赋值伪指令。变量的声明与赋值伪指令如表4.1所示。

表4.1 变量声明与赋值伪指令

变量类型	全局变量声明伪指令	局部变量声明伪指令	赋值伪指令
数字变量	GBLA	LCLA	SETA
逻辑变量	GBLL	LCLL	SETL
字符串变量	GBLS	LCLS	SETS

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.1 符号定义伪指令（2）

（1）EQU：

EQU用于将程序中的数字常量、标号、基于寄存器的值赋予一个等效的名称，这一点类似于C语言中的#define，可用“*”代替EQU。

指令格式如下：

```
name            EQU             expr {, type}
```

其中，name为要定义的常量的名称；expr可以为数字常量、程序中的标号、32位地址常量、寄存器的地址值等；type指示expr的数据类型，是可选项。

例5 EQU伪指令的使用。

```
T_bit       EQU      0x20                  ；定义常量T_bit = 0x20
PLLCON      EQU      0xE01F000C            ；定义PLLCON = 0xE01F000C
ABCD        EQU      label + 8             ；定义标号ABCD = label +8
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.1 符号定义伪指令（3）

（2）变量声明伪指令：

变量声明伪指令包括全局变量声明伪指令和局部变量声明伪指令。全局变量声明伪指令包括GBLA、GBLL、GBLS，局部变量声明伪指令包括LCLA、LCLL、LCLS。全局变量多用于程序体中，而局部变量用于宏定义体中。其中：

■ GBLA、LCLA伪指令用于声明一个数字变量，并将其初始化为0。

■ GBLL、LCLL伪指令用于声明一个逻辑变量，并将其初始化为{FALSE}。

■ GBLS、LCLS伪指令用于声明一个字符串变量，并将其初始化为空字符串。

伪指令格式如下：

```
[GB/LC]L[A/L/S]     variable
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.1 符号定义伪指令（4）

其中，[GB/LC]L[A/L/S]为变量声明伪指令，可以为6个变量声明伪指令（GBLA、GBLL、GBLS、LCLA、LCLL、LCLS）中的任一个。variable是定义的变量名，其数据类型和作用范围由变量声明伪指令来确定，但变量名在其作用内必须唯一。

例6 使用全局变量。

```
        GBLL    codebg          ；声明一个全局逻辑变量codebg
codebg  SETL    {TRUE}          ；设置变量为{TRUE}
        ...
```

例7 宏结构中使用局部变量。

```
MACRO                   ；声明一个宏
SENDDAT   $dat          ；宏的原型
LCLA      bitno         ；声明一个局部数字变量，在此宏中使用
...
bitno     SETA   8      ；设置变量bitno值为8
...
MEND
```

技术中心
eway.cc

• 4.1.2.1 符号定义伪指令（5）

（3）变量赋值伪指令：

变量赋值伪指令用于对已定义的全局变量或局部变量赋值，共有3条变量赋值伪指令：SETA、SETL、SETS。

■SETA伪指令用于给一个全局或局部的算术变量赋值。

■SETL伪指令用于给一个全局或局部的逻辑变量赋值。

■SETS伪指令用于给一个全局或局部的字符变量赋值。

指令格式如下：

```
Variable_a    SETA    expr_a
Variable_l    SETL    expr_l
Variable_s    SETS    expr_s
```

其中Variable_a、Variable_l、Variable_s就是前面全局变量或局部变量所定义的变量名。expr_a为赋值的常数；expr_l为逻辑值，即{TRUE}或{FALSE}；expr_s为赋值的字符串。

crossing 技术中心
www.threeway.cc

- 4.1.2.1 符号定义伪指令（6）

例8 给字符串变量赋值。

```
        GBLS    ErrStr              ；先声明字符串变量ErrStr
        …
ErrStr  SETS    "No, Semaphone"     ；给ErrStr变量赋值
        …
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.2 段及段属性定义伪指令（1）

由前面分析得知，汇编语言的源程序是分段的，并且每个段都有自己的属性，下面讲述段定义和段属性定义伪指令。该类指令介绍如下：

- AREA：定义一个段开始。
- END：整个文件结束。
- ALIGN：定义边界对齐方式。
- ENTRY：定义程序入口。
- CODE16：指明本段为16位Thumb代码。
- CODE32：指明本段为32位ARM代码。

例9 代码段的例子。

```
        AREA    Hello, CODE, READONLY          ; 声明代码段Hello, 只读
        ENTRY                                  ; 定义程序入口
        CODE32                                 ; 指明本段为32位 ARM代码
Start           MOV         R7, #10
                MOV         R6, #5
                ADD         R6, R6, R7         ; R6 = R6 + R7
                B           Start
        END
```

技术中心
www.threeway.cc

• 4.1.2.2 段及段属性定义伪指令（2）

例9中AREA伪指令定义了一个段，段名为Hello，段属性是只读的代码段。

ENTRY伪指令用于指定程序的入口点。一个程序（可以包含多个源文件）至少要有一个ENTRY，可以有多个ENTRY，但一个源文件中最多只有一个ENTRY。

CODE32伪指令指示汇编编译器后面的指令为32位的ARM指令。ARM9处理器支持两种指令集：Thumb指令集和ARM指令集。其中CODE16伪指令指示汇编编译器后面的指令为16位的Thumb指令，CODE32伪指令指示汇编编译器后面的指令为32位的ARM指令。CODE16和CODE32伪指令只是指示汇编编译器后面的指令的类型，伪指令本身并不进行程序状态切换。要用BX指令操作才能进行切换。

最后一条语句END伪指令用于告诉编译器已经到了源程序的结尾。每一个汇编文件均要使用一个END伪指令指示本源程序结束。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.2 段及段属性定义伪指令（3）

下面介绍稍微复杂的AREA伪指令和ALIGN伪指令。

（1）AREA伪指令：

AREA伪指令用于定义一个代码段或数据段。ARM汇编程序设计采用分段式设计，一个ARM源程序至少需要一个代码段，而大的程序可以包含多个代码段及数据段。

伪指令格式如下：

```
AREA    sectionname {, attr}{,attr}...
```

其中sectionname为所定义的代码段或数据段的名称。如果该名称是以数据开头的，则该名称必须用“|”括起来。attr为该代码段或数据段的属性。

在AREA伪指令中，各属性之间用逗号隔开，以下为段属性及相关说明：

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.2 段及段属性定义伪指令（4）

■ ALIGN=expr。默认的情况下，代码段和数据段是4字节对齐的，expr可以取0～31的数值，相应的对齐方式为2expr字节对齐。对于代码段，expr不能为0或1。

■ ASSOC=section。指定与本段相关的ELF段。任何时候链接section段也必须包括sectionname段。

■ CODE为定义代码段。属性默认为READONLY。

■ DATA为定义数据段。属性默认为READWRITE。

■ COMMON定义一个通用段。该段不包含任何用户代码和数据。链接器将其初始化为0。各源文件中同名的COMMON段共用同样的内存单元，链接器为其分配合适的尺寸。

■ NOINIT指定本数据段仅仅保留了内存单元，而没有将各个初始值写入内存单元，或者将内存单元值初始化为0。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.2 段及段属性定义伪指令（5）

■ READONLY指定本段为只读，代码段的默认属性为READONLY。

■ READWRITE指定本段为可读可写，数据段的默认属性为READWRITE。

■ 使用AREA伪指令将程序分为多个ELF格式的段，段名称可以相同，这时同名的段被放在同一个ELF段中。

例10 声明了代码段Example1，只读，并且4字节对齐。

```
    AREA    Example1, CODE, READONLY, ALIGN = 2
```

• 4.1.2.2 段及段属性定义伪指令（6）

（2）ALIGN伪指令：

ALIGN伪指令可通过添加填充字节的方式，使当前位置满足一定的对齐方式。

伪指令格式如下：

```
ALIGN    {表达式{，偏移量}}
```

其中，表达式的值用于指定对齐方式。可能的取值为2的幂，如1、2、4、8、16等，不能为0。如果伪指令中没有指定表达式，则编译器会将当前位置对齐到下一个字的位置。偏移量也为一个数字表达式，若使用该字段，则当前位置的对齐方式为：2的表达式次方+偏移量。

ALIGN=expr：对齐方式为2expr，如expr=3，则对齐方式为8字节对齐。表达式的取值范围为0～31。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（1）

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，同时也可完成已分配存储单元的初始化。该类伪指令有许多，这里只详细介绍如下常用的伪指令，感兴趣的可参考相关手册。

- ■ 声明一个文字池：LTORG。
- ■ 定义一个结构化的内存表的首地址：MAP。
- ■ 定义结构化内存表中的一个数据域：FIELD。
- ■ 分配一块内存空间，并用0初始化：SPACE。
- ■ 分配一段字节的内存单元，并用指定的数据初始化：DCB。
- ■ 分配一段半字的内存单元，并用指定的数据初始化：DCW。
- ■ 分配一段字的内存单元，并用指定的数据初始化：DCD。
- ■ 分配一段双字的内存单元，并用指定的数据初始化：DCQ。

www.threeway.cc

- 4.1.2.3 数据定义伪指令（2）

(1) LTORG：

LTORG用于声明一个文字池，在使用LDR伪指令时，要在适当的地址加入LTORG声明文字池，这样就会把要加载的数据保存到文字池内，再用ARM的加载指令读出数据（如果没有使用LTORG声明文字池，则汇编器会在程序末尾自动声明）。

伪指令格式如下：

```
        LTORG
```

例11 文字池举例。

```
LDR         R0, =0xFFFF5678     ；LDR伪指令，装载0xFFFF5678给R0
ADD         R1, R1, R0
MOV         PC, LR
LTORG                           ；声明文字池，此地址存储程序中用到的数据0xFFFF5678
…                               ；其他代码
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（3）

LTORG伪指令常放在无条件跳转指令之后，或者子程序返回指令之后，这样处理器就不会错误地将文字池中的数据当作指令来执行。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（4）

（2）MAP：

MAP伪指令用于定义一个结构化的内存表的首地址。此时内存表的位置计数器{VAR}设置为该地址值。{VAR}为汇编器的内置变量。MAP也可用“^”代替。

伪指令格式如下：

```
MAP            expr {, base_register}
```

其中，expr为程序中的标号或数字表达式。base_register（基址寄存器）为可选项，当base_register选项不存在时，expr的值即为内存表的首地址，当该选项存在时，内存表的首地址为expr的值与base_register的和。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.3 数据定义伪指令（5）

例12 MAP指令。

```
            MAP      0x00, R9          ；定义内存表的首地址为R9
Timer       FIELD    4                 ；定义数据域Timer，长度为4字节
Attrib      FIELD    4                 ；定义数据域Attrib，长度为4字节
String      FIELD    100               ；定义数据域String，长度为100字节
            …
            ADR      R9, DataStart     ；设置R9的值，即设置结构化的内存表地址
            LDR      R0, Attrib        ；相当于LDR, R0, [R9, #4]
            …
```

MAP伪指令通常与FIELD伪指令配合使用来定义结构化的内存表。但MAP、FIELD伪指令仅仅是定义数据结构，它们并不初始化内存单元的内容。MAP伪指令中的base_register寄存器的值对于其后所有的FIELD伪指令定义的数据域是默认使用的，直到遇到新的包含base_register项的MAP伪指令。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.3 数据定义伪指令（6）

（3）FIELD：

FIELD伪指令用于定义一个结构化内存表中的数据域。FIELD也可用"#"代替。

伪指令格式如下：

```
{label}          FIELD        expr
```

其中label为数据域标号，expr表示本数据域在内存表中所占的字节数。FIELD伪指令常与MAP伪指令配合使用来定义结构化的内存表。MAP伪指令定义内存表的首地址，FIELD伪指令定义内存表中的各个数据域，并可以为每个数据域指定一个标号供其他的指令引用。

例13 MAP和FIELD伪指令的使用。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.3 数据定义伪指令（7）

例13 MAP和FIELD伪指令的使用。

```
        MAP     0x40003000       ；定义内存表的首地址为0x0x40003000
Count1  FIELD   4                ；定义数据域Count1，长度为4字节
Count2  FIELD   4                ；定义数据域Count2，长度为4字节
Count3  FIELD   4                ；定义数据域Count3，长度为4字节
        ...
        LDR     R1, Count1       ；R1 ← [0x0x40003000 + 0x00]
        STR     R1, Count2       ；R1 → [0x0x40003000 + 0x04]
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.3 数据定义伪指令（8）

（4）SPACE：

SPACE用于分配一块内存单元，并用0初始化。%与SPACE同义。

伪指令格式如下：

```
{label}        SPACE        expr
```

其中，label为内存块起始地址标号，expr为所要分配的内存字节数。

例14 为Buf变量申请空间。

```
        AREA    DataRAM, DATA, READWRITE        ；声明数据段DataRAM
Buf             SPACE     1000                   ；分配1000字节空间
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（9）

（5）DCB、DCW、DCD、DCQ：

这4条伪指令都是用于分配一段内存单元，并对该内存单元初始化。唯一的区别是它们分配内存单元的大小不同。

这一类伪指令的格式是：

```
{label}    Mnemonic    Operand, ..., Operand
```

其中标号label字段是可有可无的，它表示分配的内存起始地址，作用与指令语句前的标号相同。Operand为操作数，即内存单元的初始化数据。

助记符（Mnemonic）字段说明所用伪指令的助记符，常用的有以下几种：

■ DCB分配一段字节的内存单元，其后的每个操作数都占有一个字节，操作数可以为-128～255的数值或字符串。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（10）

■ DCW分配一段半字的内存单元，其后的每个操作数都占有两个字节，操作数是16位二进制数，取值范围为-32768～65535。

■ DCD分配一段字的内存单元，其后的每个操作数都占有4个字节，操作数可以是32位的数字表达式，也可以是程序中的标号（因为程序中的标号代表地址，也是32位二进制数值）。

■ DCQ分配一段双字的内存单元，其后的每个操作数都占有8个字节。

例15 分配内存单元举例（操作数可以是常数，或者是表达式）。

硅谷芯微·技术中心
www.threeway.cc

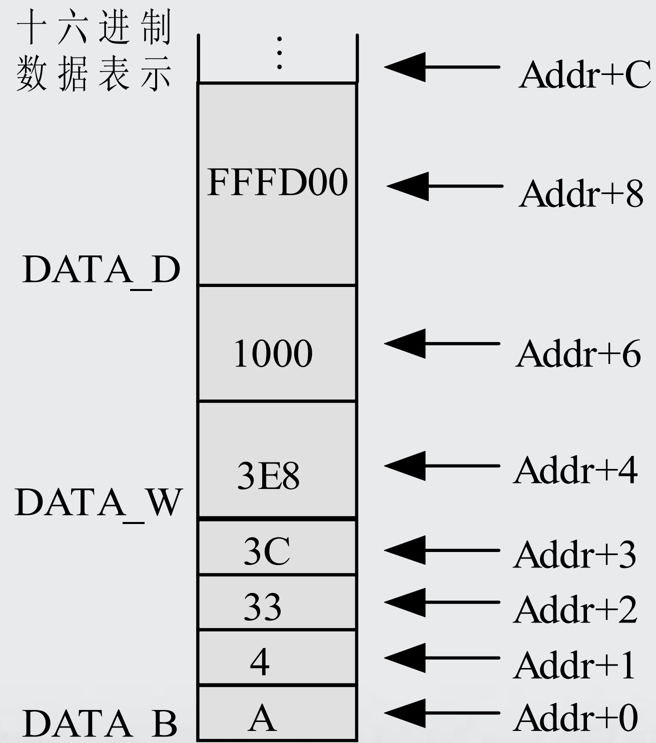
• 4.1.2.3 数据定义伪指令（11）

```
DATA_B      DCB      10, 4, 0x33, 3*20          ；分配字节类型内存单元
DATA_W      DCW      1000, 0x1000               ；分配半字类型内存单元
DATA_D      DCD      0xFFFD00                   ；分配字类型内存单元
```

汇编程序在汇编期间对存储器进行内存分配，分配结果如图4.1所示，其中Addr代表一个随机分配的内存地址。图4.1中保存的数据都用十六进制表示，其中DCB伪指令定义的每个数据占用一个字节空间，DCW伪指令定义的每个数据占用两个字节空间，DCD伪指令定义的每个数据占用4个字节空间。

- 4.1.2.3 数据定义伪指令（12）

图4.1 例15内存分配示意图



硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（13）

例16 分配内存单元举例（操作数也可以是字符串）。

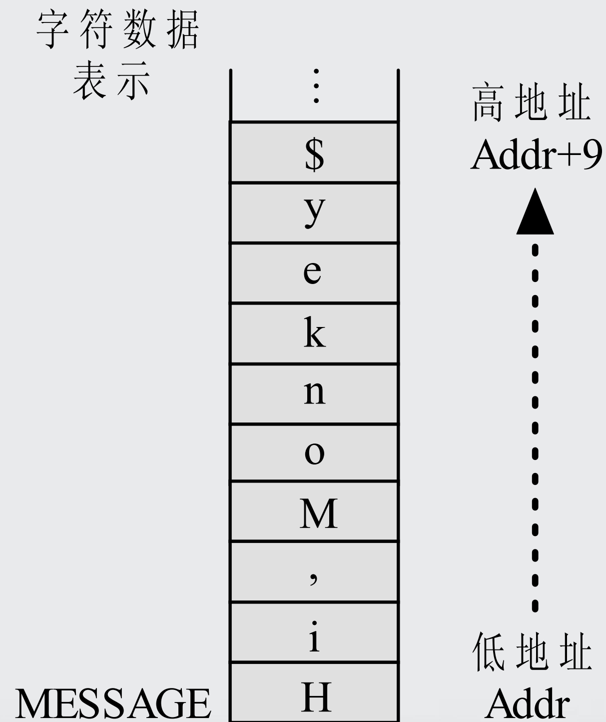
```
MESSAGE    DCB        "Hi, Monkey$"                    ；分配字节类型内存单元
```

例16的内存分配结果如图4.2所示，用了10个字节空间保存这个字符串，标号MESSAGE指向该内存块的第一个地址。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.3 数据定义伪指令（14）

图4.2 例16内存分配示意图



硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.3 数据定义伪指令（15）

例17 向量中断表（操作数还可以是程序中的标号）。

```
            LDR     PC, ResetAddr
            LDR     PC, UndefineAddr
            …
ResetAddr   DCD     Reset           ；ResetAddr变量保存Reset标号的地址
UndefineAddr DCD    Undefined       ；UndefineAddr变量保存Undefined标号的地址
            …
Reset
            …
Undefined
            …
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.4 汇编控制伪指令（1）

汇编控制伪指令用于条件汇编、宏定义、重复汇编控制等。该类伪指令如下：

■ 宏定义：MACRO和MEND。

■ 条件汇编控制：IF，ELSE和ENDIF。

■ 重复汇编：WHILE和WEND。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.4 汇编控制伪指令（2）

（1）宏定义伪指令MACRO和MEND：

宏定义伪指令包括MACRO、MEND、MEXIT。MACRO定义一个宏语句段的开始，MEND定义宏语句段的结束，MEXIT可以实现从宏程序段的跳出。用MACRO及MEND定义的一段代码称为宏定义体。在程序中就可以通过宏指令多次调用该代码段。指令格式如下：

```
MACRO
[$标号]        宏名[$参数]                ；参数可以有多个
               语句段
               MEXIT
               语句段
MEND
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.4 汇编控制伪指令（3）

宏定义指令格式说明如下：

■ 标号：是可选项。当宏定义被展开时，可被替换成相应的符号，通常为一个标号，在一个符号前使用$表示被汇编时将使用相应的值替代$后的符号。

■ 宏名：所定义的宏的名称。宏调用是通过调用宏的名称来实现的。

■ 参数：宏指令的参数。当宏指令被展开时将被替换成相应的值，类似于函数的形式参数。参数是可选项，可以有多个。

宏是一段功能完整的程序，能够实现一个特定的功能，在使用中可以把它视为一个子程序。在其他程序中可以调用宏来完成某个功能。调用宏是通过调用宏的名称来实现的。在源程序被编译时，汇编器将宏调用展开。用宏定义中的指令序列替换程序中宏名的调用，并将实际参数的值传递给宏定义中的参数。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.4 汇编控制伪指令（4）

宏与子程序的区别，在于调用宏时编译程序会在调用处插入宏的程序段，有多少次调用就会插入多少宏的程序段；而调用子程序不增加新的程序段。调用宏的好处是不占用传送参数的寄存器，不用保护现场，但如果多次调用宏，则无形中增加了代码量。

例18 宏应用举例。

```
MACRO                                   ; 宏定义开头
        MAX     $date, $time            ; 宏名为MAX，带两个参数date和time
        LDR     R1, = 0x1000            ; R1 = 0x1000（存储单元的首地址）
        LDR     R0, = $date             ; R0等于date参数值
        STR     R0, [R1], #04           ; R0 → [R1]，保存R0的内容，同时R1的地址 +4
        LDR     R2, = $time
        STR     R2, [R1]
MEND                                    ; 宏定义结束
        ...
        MAX     0x858, 12               ; 调用宏
        ADD     R3, R0, R2
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.4 汇编控制伪指令（5）

上例定义了MAX宏，宏语句段的功能是完成将两个参数date和time保存到起始地址为0x1000的内存单元中。倒数第二条语句是宏调用语句，编译后该条语句会展开。整个语句段编译后的程序如下：

```
LDR    R1, = 0x1000        ； R1 = 0x1000（存储单元的首地址）
LDR    R0, = 0x858         ； R0等于0x858
STR    R0, [R1], #04       ； R0 → [R1]，保存R0的内容，同时R1的地址 +4
LDR    R2, = 0xC           ； R2等于0xC
STR    R2, [R1]
ADD    R3, R0, R2
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.4 汇编控制伪指令（6）

（2）条件汇编控制伪指令IF、ELSE和ENDIF：

IF，ELSE和ENDIF伪指令能够根据条件把一段代码包括在汇编程序内或将其排除在程序之外。

[与IF同义，|与ELSE同义，]与ENDIF同义。

伪指令格式如下：

```
IF        logical_expr
          ；指令或伪指令代码段1
{
ELSE
          ；指令或伪指令代码段2
}
```

其中，logical_expr为用于控制的逻辑表达式。若条件成立，则代码段1落在汇编源程序中有效。若条件不成立，代码段1无效，同时若使用ELSE伪指令，代码段2有效。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.4 汇编控制伪指令（7）

例19 条件汇编控制举例。

```
...
IF {CONFIG} = 16
        BNE         __rt_udiv_1
        LDR         R0, =__rt_div0
        BX          R0
ELSE
        BEQ         __rt_div0
ENDIF
```

注意：IF，ELSE和ENDIF伪指令是可以嵌套使用的。

• 4.1.2.4 汇编控制伪指令（8）

（3）重复汇编伪指令WHILE和WEND：

WHILE和WEND伪指令用于根据条件重复汇编相同的或几乎相同的一段源程序。

伪指令格式如下：

```
WHILE       logical_expr
        ；指令或伪指令代码段
WEND
```

其中，logical_expr为用于控制的逻辑表达式。若条件成立，则代码段在汇编程序中有效，并不断重复这段代码直到条件不成立。

例20 重复汇编举例。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.4 汇编控制伪指令（9）

例20 重复汇编举例。

```
WHILE        no<5
no           SETA        no+1
             …
WEND
```

注意：WHILE和WEND伪指令是可以嵌套使用的。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.5 杂项伪指令（1）

杂项伪指令介绍如下：

- 导出伪指令：EXPORT、GLOBAL。
- 导入伪指令：IMPORT、EXTERN。
- 文件包含伪指令：GET、INCLUDE。

一个程序可以由多个汇编源文件组成，多个文件间会互相引用符号（变量或标号）。当在一个源文件中定义的一个符号希望其他文件引用时，则必须用导出伪指令定义这个符号；如果这个文件引用了外部定义的符号，则必须用导入伪指令定义这个符号。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.5 杂项伪指令（2）

（1）导出伪指令EXPORT和GLOBAL：

EXPORT声明一个符号可以被其他文件应用，相当于声明一个全局标号，可以被其他文件引用。GLOBAL与EXPORT相同。

伪指令格式如下：

```
EXPORT    标号  [, WEAK]
GLOBAL    标号  [, WEAK]
```

其中，标号为要声明的符号名称。[, WEAK]为可选项，声明其他文件有同名的标号时，则该同名标号优先于该标号被引用。

例21 定义全局标号InitStack和Vectors。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.5 杂项伪指令（3）

例21 定义全局标号InitStack和Vectors。

```
        EXPORT    InitStack, WEAK
        GLOBAL    Vectors
```

• 4.1.2.5 杂项伪指令（4）

（2）导入伪指令IMPORT和EXTERN：

IMPORT告诉编译器这个标号要在当前源文件中使用，但标号是在其他的源文件中定义的。不管当前源文件是否使用过该标号，这个标号都会加入到当前源文件的符号表中。EXTERN和IMPORT同义，都是声明一个外部符号。

伪指令格式如下：

```
IMPORT    标号  [, WEAK]
EXTERN    标号  [, WEAK]
```

其中，标号为要声明的符号名称。[, WEAK]为可选项，表示如果所有的源文件都没有找到这个标号的定义，编译器也不会提示错误信息。

硅谷芯微·技术中心
www.threeway.cc

- 4.1.2.5 杂项伪指令（5）

使用IMPORT或EXTERN声明外部标号时，若链接器在链接处理时不能解释该符号，而伪指令中没有[WEAK]选项，则链接器会报告错误；如伪指令中有[WEAK]选项，则链接器不会报告错误，而是进行下面的操作：

■ 如果该符号被B或者BL指令引用，则该符号被设置成下一条指令的地址，该B或BL指令相当于一条NOP指令。

■ 其他情况下该符号被设置为0。

例22 导入标号InitStack和Vectors。

```
IMPORT    InitStack
EXTERN    Vectors
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.2.5 杂项伪指令（6）

（3）文件包含伪指令GET和INCLUDE：

GET伪指令将一个源文件包含到当前的源文件中，并将被包含的源文件在当前位置展开进行汇编处理。INCLUDE与之同义。

伪指令格式如下：

```
GET        文件名
INCLUDE    文件名
```

我们通常这样使用这个伪指令：在某源文件中定义一些宏指令用MAP和FIELD定义结构化的数据类型，用EQU定义常量的符号名称，然后用GET/INCLUDE将这个源文件包含到其他的源文件中。这样的源文件类似于C语言中的头文件，GET/INCLUDE不能用于包含目标文件，包含目标文件则需要使用INCBIN伪指令。

例23 包含inc文件。

```
INCLUDE    Start.inc
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3 汇编语言程序设计及举例

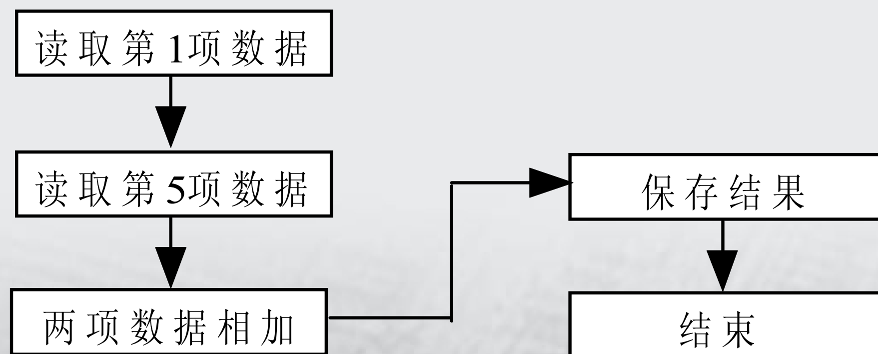
程序有顺序、分支、循环和子程序4种结构形式。顺序程序结构是指完全按顺序逐条执行的指令序列，这在程序段中是大量存在的，但作为完整的程序则很少见。本小节将介绍顺序、分支、循环和子程序这4种结构。

- 4.1.3.1 顺序程序设计（1）

最简单的程序是没有分支、没有循环的顺序运行程序。下面以一个算术运算程序进行说明和介绍。

例24 通过查表操作实现数组中的第1项数据和第5项数据相加，结果保存到数组中。程序中首先读取数组的第1项数据，然后读取第5项数据，之后将结果相加，最后保存结果，整个流程是顺序执行的，如图4.3所示。

图4.3 程序流程图



硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.1 顺序程序设计（2）

程序清单4.1 例24的程序

```
        AREA    Buf, DATA, READWRITE                         ; 定义数据段Buf
Array           DCD     0x11, 0x22, 0x33, 0x44               ; 定义12个字的数组Array
                DCD     0x55, 0x66, 0x77, 0x88
                DCD     0x00, 0x00, 0x00, 0x00
        AREA    Example, CODE, READONLY
        ENTRY
        CODE32
                LDR     R0, = Array                          ; 取得数组Array首地址
                LDR     R2, [R0]                             ; 装载数组第1项字数据给R2
                MOV     R1, #4
                LDR     R3, [R0, R1, LSL #2]                 ; 装载数组第5项字数据给R3
                ADD     R2, R2, R3                           ; R2 + R3 → R2
                MOV     R1, #8                               ; R1 = 8
                STR     R2, [R0, R1, LSL #2]                 ; 保存结果到数组第9项
        END
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.3.2 分支程序设计（1）

在一个实际的程序中，程序始终是顺序执行的情况并不多见，通常都会有各种分支。在ARM汇编程序中，条件后缀能实现程序分支。

例25 编写汇编程序实现C语言if else分支程序。C语言程序如下（分支流程图如图4.4所示）：

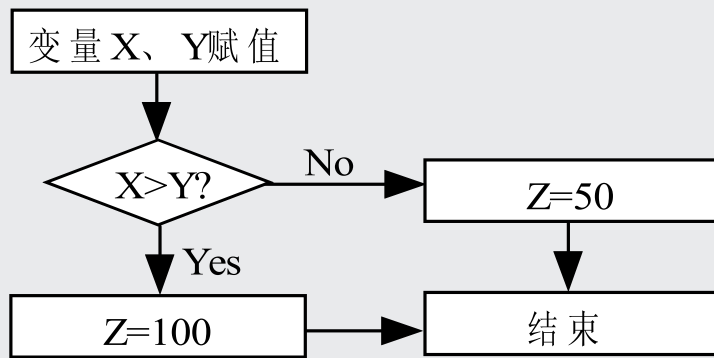
```
int        x=76;  //定义整型变量
int        y=88;
if( x>y )
           z=100;
      else
           z=50;
```

图4.4 分支流程图

硅谷芯微·技术中心
www.threeway.cc

- 4.1.3.2 分支程序设计（2）

图4.4 分支流程图



设R0保存变量x的值，R1保存变量y的值，R2保存变量z的值（x、y、z均为无符号整数），对应实现的汇编程序见程序清单4.2。

程序清单4.2 例25的汇编程序

硅谷芯微·技术中心
www.threeway.cc

- 4.1.3.2 分支程序设计（3）

程序清单4.2 例25的汇编程序

```
...
MOV      R0, #76          ；初始化R0的值
MOV      R1, #88          ；初始化R1的值
CMP      R0, R1           ；判断R0>R1?
MOVHI    R2, #100         ；R0>R1时，这条语句执行，则R2 = 100
MOVLS    R2, #50          ；R0<R1时，这条语句执行，则R2 = 50
...
```

B和BL指令可以实现分支，程序清单4.3展示了B指令实现汇编程序中常用的散转算法。

程序清单4.3 B指令实现散转功能

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.2 分支程序设计（4）

程序清单4.3 B指令实现散转功能

```
CMP      R0, #MAX INDEX        ；判断索引号是否超出最大索引值
ADDLO    PC, PC, R0, LSL #2    ；索引号若没有超出，则程序跳转到相应位置
BHI      ERROR                 ；若超出，则进行出错处理

B        FUN1                  ；跳到第1个程序
B        FUN2                  ；跳到第2个程序
B        FUN3                  ；跳到第3个程序
…
```

- 4.1.3.3 循环程序设计（1）

在程序中，往往要求某一段程序重复执行多次，这时就可以利用循环程序结构。一个循环结构由以下两部分组成。

■ 循环体：就是要求重复执行的程序段部分。

■ 循环结束条件：在循环程序中必须给出循环结束条件，否则程序就会进入死循环。常见的循环有计数循环和条件循环。计数循环是当循环了一定次数后就结束循环；条件循环是当循环条件为假时就结束循环。

在C语言中，for和while语句可以实现这两种循环。下面介绍如何用汇编语言实现这两种循环。

例26 编写汇编程序实现计数循环。

计数循环用C语言表达如下：

```
for( i = 0 ; i < 10 ; i++)    x++;
```

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.3 循环程序设计（2）

设R0为x，R2为i（i、x均为无符号整数），汇编语言程序段如下：

```
            ...
            MOV     R0, #0          ; 初始化R0 = 0
            MOV     R2, #0          ; 设置R2 = 0，R2控制循环次数
FOR         CMP     R2, #10         ; 判断R2<10?
            BCS     FOR_E           ; 若条件失败（即R2≥10），退出循环
            ADD     R0, R0, #1      ; 循环体，R0 = R0 +1
            ADD     R2, R2, #1      ; R2 = R2+1
            B       FOR
FOR_E       ...
```

例27 编写汇编程序实现条件循环。

条件循环用C语言表达如下：

```
while（ x <= y）
        x *= 2;
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.3.3 循环程序设计（3）

设x为R0，y为R1（x、y均为无符号整数），汇编语言程序段如下：

```
        ...
        MOV     R0, #1              ；初始化R0 = 1
        MOV     R1, #20             ；初始化R1 = 20
        B       W_2                 ；首先要判断条件
W_1     MOV     R0, R0, LSL #1      ；循环体，R0 *= 2
W_2     CMP     R0, R1              ；判断R0≤R1？，即x≤y?
        BLS     W_1                 ；若R0≤R1，继续循环
W_END
        ...
```

例28 编写循环语句实现数据块复制。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.3 循环程序设计（4）

例28 编写循环语句实现数据块复制。

```
        LDR     R0, =DATA_DST       ；指向数据目标地址
        LDR     R1, =DATA_SRC       ；指向数据源地址
        MOV     R10, #20            ；赋值数据个数20×N个字
                                    ；N为LDM指令操作数据个数
LOOP    LDMIA   R1!, {R2-R9}        ；从数据源读取8个字到R2～R9
        STMIA   R0!, {R2-R9}        ；将R2～R9的数据保存到目标地址
        SUBS    R10, R10, #1        ；R10-1，并改变程序状态寄存器
        BNE     LOOP
```

硅谷芯微·技术中心
www.threeway.cc

- 4.1.3.4 子程序设计（1）

在一个程序的不同部分往往要用到类似的程序段，这些程序段的功能和结构形式都相同，只是某些变量的赋值不同，此时就可以把这些程序段写成子程序形式，以便需要时可以调用它。

调用程序在调用子程序时，经常需要传送一些参数给子程序；子程序运行完后也经常要回送结果给调用程序。这种调用程序和子程序之间的信息传送称为参数传送。参数传送可以有以下两种方法：

- 当参数比较少时，可以通过寄存器传送参数。
- 当参数比较多时，可以通过内存块或堆栈传送参数。

子程序的正确执行是由子程序的正确调用及正确返回保证的。这就要求调用程序在调入子程序时必须保存正确的返回地址，即当前PC值（由于ARM流水线特性

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.4 子程序设计（2）

，可能要减去一个偏移量）。PC值可以保存在专用的链接寄存器R14中，也可以保存到堆栈中。根据这两种情况，可以在子程序中采用如下的返回语句：

```
        MOV     PC, LR              ；恢复PC的值
或      STMFD   SP!, {R0-R7,PC}     ；将PC值从堆栈中返回
```

使用堆栈来恢复处理器的状态时，注意STMFD与LDMFD要配合使用。

一般来讲，在ARM汇编语言程序中，子程序的调用是通过BL指令来实现的。该指令在执行时完成如下操作：将子程序的返回地址存放在链接寄存器LR中（针对流水线特性，已经减去偏移量了），同时将程序计数器PC指向子程序的入口点，当子程序执行完毕需要返回调用处时，只需要将存放在LR中的返回地址重新拷贝给程序计数器PC即可。

硅谷芯微·技术中心
www.threeway.cc

• 4.1.3.4 子程序设计（3）

下面看一段C代码。程序实现比较两个数，取出其最大值。实现求最大值功能的C语言代码见程序清单4.4的MAX函数，对应的汇编代码见程序清单4.5的MAX标号。在程序清单4.5中，子程序和调用主程序之间通过寄存器R0、R1、R2传递参数。

程序清单4.4 MAX函数和调用主程序

```
int   MAX( int  i, int  j )                          //声明子函数MAX
{
      if( i>j )              return ( i );
            else             return ( j );
}

Main(void)                                           //主函数
{
      int      a, b, c;
      a=19;                                          //给变量a赋初值
      b=20;                                          //给变量b赋初值
      c=MAX(a,b);                                    //调用MAX子函数，把最大值赋给c
}
```

术中心
www.threeway.cc

• 4.1.3.4 子程序设计（4）

程序清单4.5 MAX汇编子程序和调用主程序

```
X       EQU     19                                  ；定义X的值为19
N       EQU     20                                  ；定义N的值为20

        AREA    Example4, CODE, READONLY            ；声明代码段Example4
        ENTRY                                       ；标识程序入口
        CODE32                                      ；声明32位ARM指令
START           LDR         R0, =X                  ；给R0、R1赋初值
                LDR         R1, =N
                BL          MAX                     ；调用子程序MAX，返回值为R2
HALT            B           HALT                    ；死循环

MAX                                                 ；声明子程序MAX
                CMP         R0, R1                  ；比较R0与R1，R2等于最大值
                MOVHI       R2, R0
                MOVLS       R2, R1
                MOV         PC, LR                  ；返回语句
MAX_END

        END
```

硅谷芯微·技术中心
www.threeway.cc

谢谢大家

硅谷芯微嵌入式学院
技术奉献