目录

ARM 汇编伪指令

在 ARM 汇编语言程序里，有一些特殊指令助记符，这些助记符与指令系统的助记符不同，没有相对应的操作码，通常称这些特殊指令助记符为伪指令，他们所完成的操作称为伪操作。伪指令在源程序中的作用是为完成汇编程序作各种准备工作的，这些伪指令仅在汇编过程中起作用，一旦汇编结束，伪指令的使命就完成。

在 ARM 的汇编程序中，有如下几种伪指令：符号定义伪指令、数据定义伪指令、汇编控制伪指令、宏指令以及其他伪指令。

符号定义（ Symbol Definition ）伪指令

符号定义伪指令用于定义 ARM 汇编程序中的变量、对变量赋值以及定义寄存器的别名等操作。

一.常见的符号定义伪指令有如下几种：

— 用于定义全局变量的 GBLA 、 GBLL 和 GBLS 。

— 用于定义局部变量的 LCLA 、 LCLL 和 LCLS 。

— 用于对变量赋值的 SETA 、 SETL 、 SETS 。

— 为通用寄存器列表定义名称的 RLIST 。

1、 GBLA、GBLL 和 GBLS

语法格式：

GBLA （ GBLL 或 GBLS ） 全局变量名

GBLA 、 GBLL 和 GBLS 伪指令用于定义一个 ARM 程序中的全局变量，并将其初始化。其中：

GBLA 伪指令用于定义一个全局的数字变量，并初始化为 0 ；

GBLL 伪指令用于定义一个全局的逻辑变量，并初始化为 F （假）；

GBLS 伪指令用于定义一个全局的字符串变量，并初始化为空；

由于以上三条伪指令用于定义全局变量，因此在整个程序范围内变量名必须唯一。

使用示例：

GBLA Test1 ； 定义一个全局的数字变量，变量名为 Test1

Test1 SETA 0xaa ； 将该变量赋值为 0xaa

GBLL Test2 ； 定义一个全局的逻辑变量，变量名为 Test2

Test2 SETL {TRUE} ； 将该变量赋值为真

GBLS Test3 ； 定义一个全局的字符串变量，变量名为 Test3

Test3 SETS “ Testing ” ； 将该变量赋值为 “ Testing ”

2、 LCLA、LCLL 和 LCLS

语法格式：

LCLA （ LCLL 或 LCLS ） 局部变量名

LCLA 、 LCLL 和 LCLS 伪指令用于定义一个 ARM 程序中的局部变量，并将其初始化。其中：

LCLA 伪指令用于定义一个局部的数字变量，并初始化为 0 ；

LCLL 伪指令用于定义一个局部的逻辑变量，并初始化为 F （假）；

LCLS 伪指令用于定义一个局部的字符串变量，并初始化为空；

以上三条伪指令用于声明局部变量，在其作用范围内变量名必须唯一。

使用示例：

LCLA Test4 ； 声明一个局部的数字变量，变量名为 Test4

Test3 SETA 0xaa ； 将该变量赋值为 0xaa

LCLL Test5 ； 声明一个局部的逻辑变量，变量名为 Test5

Test4 SETL {TRUE} ； 将该变量赋值为真

LCLS Test6 ； 定义一个局部的字符串变量，变量名为 Test6

Test6 SETS “ Testing ” ； 将该变量赋值为 “ Testing ”

3、 SETA、SETL 和 SETS

语法格式：

变量名 SETA （ SETL 或 SETS ） 表达式

伪指令 SETA 、 SETL 、 SETS 用于给一个已经定义的全局变量或局部变量赋值。

SETA 伪指令用于给一个数学变量赋值；

SETL 伪指令用于给一个逻辑变量赋值；

SETS 伪指令用于给一个字符串变量赋值；

其中，变量名为已经定义过的全局变量或局部变量，表达式为将要赋给变量的值。

使用示例：

LCLA Test3 ； 声明一个局部的数字变量，变量名为 Test3

Test3 SETA 0xaa ； 将该变量赋值为 0xaa

LCLL Test4 ； 声明一个局部的逻辑变量，变量名为 Test4

Test4 SETL {TRUE} ； 将该变量赋值为真

4 、 RLIST

语法格式：

名称 RLIST { 寄存器列表 }

RLIST 伪指令可用于对一个通用寄存器列表定义名称，使用该伪指令定义的名称可在 ARM 指令 LDM/STM 中使用。在 LDM/STM 指令中，列表中的寄存器访问次序为根据寄存器的编号由低到高，而与列表中的寄存器排列次序无关。

使用示例：

RegList RLIST {R0-R5 ， R8 ， R10} ；将寄存器列表名称定义为 RegList ，可在 ARM 指令 LDM/STM 中通过该名称访问寄存器列表。

数据定义（ Data Definition ）伪指令

数据定义伪指令一般用于为特定的数据分配存储单元，同时可完成已分配存储单元的初始化。

二.常见的数据定义伪指令有如下几种：

— DCB 用于分配一片连续的字节存储单元并用指定的数据初始化。

— DCW （ DCWU ） 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。

— DCD （ DCDU ） 用于分配一片连续的字存储单元并用指定的数据初始化。

— DCFD （ DCFDU ）用于为双精度的浮点数分配一片连续的字存储单元并用指定的数据初始化。

— DCFS （ DCFSU ） 用于为单精度的浮点数分配一片连续的字存储单元并用指定的数据初 始化。

— DCQ （ DCQU ） 用于分配一片以 8 字节为单位的连续的存储单元并用指定的数据初始化。

— SPACE 用于分配一片连续的存储单元

— MAP 用于定义一个结构化的内存表首地址

— FIELD 用于定义一个结构化的内存表的数据域

1、 DCB

语法格式：

标号 DCB 表达式

DCB 伪指令用于分配一片连续的字节存储单元并用伪指令中指定的表达式初始化。其中，表达式可以为 0 ～ 255 的数字或字符串。 DCB 也可用 “ = ” 代替。

使用示例：

Str DCB “ This is a test ！ ” ；分配一片连续的字节存储单元并初始化。

2、 DCW（或 DCWU）

语法格式：

标号 DCW （或 DCWU ） 表达式

DCW （或 DCWU ）伪指令用于分配一片连续的半字存储单元并用伪指令中指定的表达式初始化。

其中，表达式可以为程序标号或数字表达式。。

用 DCW 分配的字存储单元是半字对齐的，而用 DCWU 分配的字存储单元并不严格半字对齐。

使用示例：

DataTest DCW 1 ， 2 ， 3 ；分配一片连续的半字存储单元并初始化。

3、 DCD（或 DCDU）

语法格式：

标号 DCD （或 DCDU ） 表达式

DCD （或 DCDU ）伪指令用于分配一片连续的字存储单元并用伪指令中指定的表达式初始化。其中，表达式可以为程序标号或数字表达式。 DCD 也可用 “ & ” 代替。

用 DCD 分配的字存储单元是字对齐的，而用 DCDU 分配的字存储单元并不严格字对齐。

使用示例：

DataTest DCD 4 ， 5 ， 6 ；分配一片连续的字存储单元并初始化。

4、 DCFD（或 DCFDU）

语法格式：

标号 DCFD （或 DCFDU ） 表达式

DCFD （或 DCFDU ）伪指令用于为双精度的浮点数分配一片连续的字存储单元并用伪指令中指定的表达式初始化。每个双精度的浮点数占据两个字单元。用 DCFD 分配的字存储单元是字对齐的，而用 DCFDU 分配的字存储单元并不严格字对齐。

使用示例：

FDataTest DCFD 2E115 ， -5E7 ；分配一片连续的字存储单元并初始化为指定的双精度数。

5、 DCFS（或 DCFSU）

语法格式：

标号 DCFS （或 DCFSU ） 表达式

DCFS （或 DCFSU ）伪指令用于为单精度的浮点数分配一片连续的字存储单元并用伪指令中指定的表达式初始化。每个单精度的浮点数占据一个字单元。 用 DCFS 分配的字存储单元是字对齐的，而用 DCFSU 分配的字存储单元并不严格字对齐。

使用示例：

FDataTest DCFS 2E5 ， -5E － 7 ；分配一片连续的字存储单元并初始化为指定的单精度数。

6、 DCQ(或 DCQU）

语法格式：

标号 DCQ （或 DCQU ） 表达式

DCQ （或 DCQU ）伪指令用于分配一片以 8 个字节为单位的连续存储区域并用伪指令中指定的表达式初始化。

用 DCQ 分配的存储单元是字对齐的，而用 DCQU 分配的存储单元并不严格字对齐。

使用示例：

DataTest DCQ 100 ；分配一片连续的存储单元并初始化为指定的值。

7、 SPACE

语法格式：

标号 SPACE 表达式

SPACE 伪指令用于分配一片连续的存储区域并初始化为 0 。其中，表达式为要分配的字节数。

SPACE 也可用 “ % ” 代替。

使用示例：

DataSpace SPACE 100 ；分配连续 100 字节的存储单元并初始化为 0 。

8、 MAP

语法格式：

MAP 表达式 { ，基址寄存器 }

MAP 伪指令用于定义一个结构化的内存表的首地址。 MAP 也可用 “ ^ ” 代替。

表达式可以为程序中的标号或数学表达式，基址寄存器为可选项，当基址寄存器选项不存在时，表达式的值即为内存表的首地址，当该选项存在时，内存表的首地址为表达式的值与基址寄存器的和。

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。

使用示例：

MAP 0x100 ， R0 ；定义结构化内存表首地址的值为 0x100 ＋ R0 。

9、 FILED

语法格式：

标号 FIELD 表达式

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

表达式的值为当前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。

注意 MAP 和 FIELD 伪指令仅用于定义数据结构，并不实际分配存储单元。

使用示例：

MAP 0x100 ；定义结构化内存表首地址的值为 0x100 。

A FIELD 16 ；定义 A 的长度为 16 字节，位置为 0x100

B FIELD 32 ；定义 B 的长度为 32 字节，位置为 0x110

S FIELD 256 ；定义 S 的长度为 256 字节，位置为 0x130

汇编控制（ Assembly Control ）伪指令

三. 常用的汇编控制伪指令

汇编控制伪指令用于控制汇编程序的执行流程，常用的汇编控制伪指令包括以下几条：

— IF 、 ELSE 、 ENDIF

— WHILE 、 WEND

— MACRO 、 MEND

— MEXIT

1、 IF、ELSE、ENDIF

语法格式：

IF 逻辑表达式

指令序列 1

ELSE

指令序列 2

ENDIF

IF 、 ELSE 、 ENDIF 伪指令能根据条件的成立与否决定是否执行某个指令序列。当 IF 后面的逻辑表达式为真，则执行指令序列 1 ，否则执行指令序列 2 。其中， ELSE 及指令序列 2 可以没有，此时，当 IF 后面的逻辑表达式为真，则执行指令序列 1 ，否则继续执行后面的指令。

IF 、 ELSE 、 ENDIF 伪指令可以嵌套使用。

使用示例：

GBLL Test ； 声明一个全局的逻辑变量，变量名为 Test……

IF Test = TRUE

指令序列 1

ELSE

指令序列 2

ENDIF

2、 WHILE、WEND

语法格式：

WHILE 逻辑表达式

指令序列

WEND

WHILE 、 WEND 伪指令能根据条件的成立与否决定是否循环执行某个指令序列。当 WHILE 后面的逻辑表达式为真，则执行指令序列，该指令序列执行完毕后，再判断逻辑表达式的值，若为真则继续执行，一直到逻辑表达式的值为假。

WHILE 、 WEND 伪指令可以嵌套使用。

使用示例：

GBLA Counter ； 声明一个全局的数学变量，变量名为 Counter

Counter SETA 3 ； 由变量 Counter 控制循环次数

……

WHILE Counter < 10

指令序列

WEND

3、 MACRO、MEND

语法格式：

$ 标号 宏名 $ 参数 1 ， $ 参数 2 ，……

指令序列

MEND

MACRO 、 MEND 伪指令可以将一段代码定义为一个整体，称为宏指令，然后就可以在程序中通过宏指令多次调用该段代码。其中， $ 标号在宏指令被展开时，标号会被替换为用户定义的符号， 宏指令可以使用一个或多个参数，当宏指令被展开时，这些参数被相应的值替换。

宏指令的使用方式和功能与子程序有些相似，子程序可以提供模块化的程序设计、节省存储空间并提高运行速度。但在使用子程序结构时需要保护现场，从而增加了系统的开销，因此，在代码较短且需要传递的参数较多时，可以使用宏指令代替子程序。

包含在 MACRO 和 MEND 之间的指令序列称为宏定义体，在宏定义体的第一行应声明宏的原型（包含宏名、所需的参数），然后就可以在汇编程序中通过宏名来调用该指令序列。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代替程序中的宏调用，并将实际参数的值传递给宏定义中的形式参数。

MACRO 、 MEND 伪指令可以嵌套使用。

4、 MEXIT

语法格式：

MEXIT

MEXIT 用于从宏定义中跳转出去。

四.其他常用的伪指令

还有一些其他的伪指令，在汇编程序中经常会被使用，包括以下几条：

— AREA

— ALIGN

— CODE16 、 CODE32

— ENTRY

— END

— EQU

— EXPORT （或 GLOBAL ）

— IMPORT

— EXTERN

— GET （或 INCLUDE ）

— INCBIN

— RN

— ROUT

1、 AREA

语法格式：

AREA 段名 属性 1 ，属性 2 ，……

AREA 伪指令用于定义一个代码段或数据段。其中，段名若以数字开头，则该段名需用 “ | ” 括起来，如 |1_test| 。

属性字段表示该代码段（或数据段）的相关属性，多个属性用逗号分隔。常用的属性如下：

— CODE 属性：用于定义代码段，默认为 READONLY 。

— DATA 属性：用于定义数据段，默认为 READWRITE 。

— READONLY 属性：指定本段为只读，代码段默认为 READONLY 。

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

— ALIGN 属性：使用方式为 ALIGN 表达式。在默认时， ELF （可执行连接文件）的代码段和数据段是按字对齐的，表达式的取值范围为 0 ～ 31 ，相应的对齐方式为 2 表达式次方。

— COMMON 属性：该属性定义一个通用的段，不包含任何的用户代码和数据。各源文件中同名的 COMMON 段共享同一段存储单元。

一个汇编语言程序至少要包含一个段，当程序太长时，也可以将程序分为多个代码段和数据段。

使用示例：

AREA Init ， CODE ， READONLY

该伪指令定义了一个代码段，段名为 Init ，属性为只读

2、 ALIGN

语法格式：

ALIGN { 表达式 { ，偏移量 }}

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其方式 | 。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。若未指定表达式，则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 的表达式次幂＋偏移量。

使用示例：

AREA Init ， CODE ， READONLY ， ALIEN ＝ 3 ；指定后面的指令为 8 字节对齐。

指令序列

END

3、 CODE16、CODE32

语法格式：

CODE16 （或 CODE32 ）

CODE16 伪指令通知编译器，其后的指令序列为 16 位的 Thumb 指令。

CODE32 伪指令通知编译器，其后的指令序列为 32 位的 ARM 指令。

若在汇编源程序中同时包含 ARM 指令和 Thumb 指令时，可用 CODE16 伪指令通知编译器其后的指令序列为 16 位的 Thumb 指令， CODE32 伪指令通知编译器其后的指令序列为 32 位的 ARM 指令。因此，在使用 ARM 指令和 Thumb 指令混合编程的代码里，可用这两条伪指令进行切换，但注意他们只通知编译器其后指令的类型，并不能对处理器进行状态的切换。

使用示例：

AREA Init ， CODE ， READONLY

……

CODE32 ；通知编译器其后的指令为 32 位的 ARM 指令

LDR R0 ，＝ NEXT ＋ 1 ；将跳转地址放入寄存器 R0

BX R0 ；程序跳转到新的位置执行，并将处理器切换到 Thumb 工作状态

……

CODE16 ；通知编译器其后的指令为 16 位的 Thumb 指令

NEXT LDR R3，＝0x3FF

……

END ；程序结束

4、 ENTRY

语法格式：

ENTRY

ENTRY 伪指令用于指定汇编程序的入口点。在一个完整的汇编程序中至少要有一个 ENTRY （也可以有多个，当有多个 ENTRY 时，程序的真正入口点由链接器指定），但在一个源文件里最多只能有一个 ENTRY （可以没有）。

使用示例：

AREA Init ， CODE ， READONLY

ENTRY ；指定应用程序的入口点

……

5、 END

语法格式：

END

END 伪指令用于通知编译器已经到了源程序的结尾。

使用示例：

AREA Init ， CODE ， READONLY

……

END ；指定应用程序的结尾

6、 EQU

语法格式：

名称 EQU 表达式 { ，类型 }

EQU 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，类似于 C 语言中的＃ define 。

其中 EQU 可用 “ * ” 代替。

名称为 EQU 伪指令定义的字符名称，当表达式为 32 位的常量时，可以指定表达式的数据类型，可以有以下三种类型：

CODE16 、 CODE32 和 DATA

使用示例：

Test EQU 50 ；定义标号 Test 的值为 50

Addr EQU 0x55 ， CODE32 ；定义 Addr 的值为 0x55 ，且该处为 32 位的 ARM 指令。

7、 EXPORT（或 GLOBAL）

语法格式：

EXPORT 标号 {[WEAK]}

EXPORT 伪指令用于在程序中声明一个全局的标号，该标号可在其他的文件中引用。EXPORT 可用 GLOBAL 代替。标号在程序中区分大小写， [WEAK] 选项声明其他的同名标号优先于该标号被引用。

使用示例：

AREA Init ， CODE ， READONLY

EXPORT Stest ；声明一个可全局引用的标号 Stest……

END

8、 IMPORT

语法格式：

IMPORT 标号 {[WEAK]}

IMPORT 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，但要在当前源文件中引用，而且无论当前源文件是否引用该标号，该标号均会被加入到当前源文件的符号表中。

标号在程序中区分大小写， [WEAK] 选项表示当所有的源文件都没有定义这样一个标号时，编译器也不给出错误信息，在多数情况下将该标号置为 0 ，若该标号为 B 或 BL 指令引用，则将 B 或 BL 指令置为 NOP 操作。

使用示例：

AREA Init ， CODE ， READONLY

IMPORT Main ；通知编译器当前文件要引用标号 Main，但 Main 在其他源文件中定义……

END

9、 EXTERN

语法格式：

EXTERN 标号 {[WEAK]}

EXTERN 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，但要在当前源文件中引用，如果当前源文件实际并未引用该标号，该标号就不会被加入到当前源文件的符号表中。标号在程序中区分大小写， [WEAK] 选项表示当所有的源文件都没有定义这样一个标号时，编译器也不给出错误信息，在多数情况下将该标号置为 0 ，若该标号为 B 或 BL 指令引用，则将 B 或 BL 指令置为 NOP 操作。

使用示例：

AREA Init ， CODE ， READONLY

EXTERN Main ；通知编译器当前文件要引用标号 Main，但 Main 在其他源文件中定义……

END

10、 GET（或 INCLUDE）

语法格式：

GET 文件名

GET 伪指令用于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进行汇编处理。可以使用 INCLUDE 代替 GET 。

汇编程序中常用的方法是在某源文件中定义一些宏指令，用 EQU 定义常量的符号名称，用 MAP 和 FIELD 定义结构化的数据类型，然后用 GET 伪指令将这个源文件包含到其他的源文件中。使用方法与 C 语言中的 “ include ” 相似。

GET 伪指令只能用于包含源文件，包含目标文件需要使用 INCBIN 伪指令

使用示例：

AREA Init ， CODE ， READONLY

GET a1.s ；通知编译器当前源文件包含源文件 a1.s

GE T C：\a2.s ；通知编译器当前源文件包含源文件 C：\ a2.s ……

END

11、 INCBIN

语法格式：

INCBIN 文件名

INCBIN 伪指令用于将一个目标文件或数据文件包含到当前的源文件中，被包含的文件不作任何变动的存放在当前文件中，编译器从其后开始继续处理。

使用示例：

AREA Init ， CODE ， READONLY

INCBIN a1.dat ；通知编译器当前源文件包含文件 a1.dat

INCBIN C：\a2.txt ；通知编译器当前源文件包含文件 C：\a2.txt……

END

12、 RN

语法格式：

名称 RN 表达式

RN 伪指令用于给一个寄存器定义一个别名。采用这种方式可以方便程序员记忆该寄存器的功能。其中，名称为给寄存器定义的别名，表达式为寄存器的编码。

使用示例：

Temp RN R0 ；将 R0 定义一个别名 Temp

13、 ROUT

语法格式：

{ 名称 } ROUT

ROUT 伪指令用于给一个局部变量定义作用范围。在程序中未使用该伪指令时，局部变量的作用范围为所在的 AREA ，而使用 ROUT 后，局部变量的作为范围为当前 ROUT 和下一个 ROUT 之间。