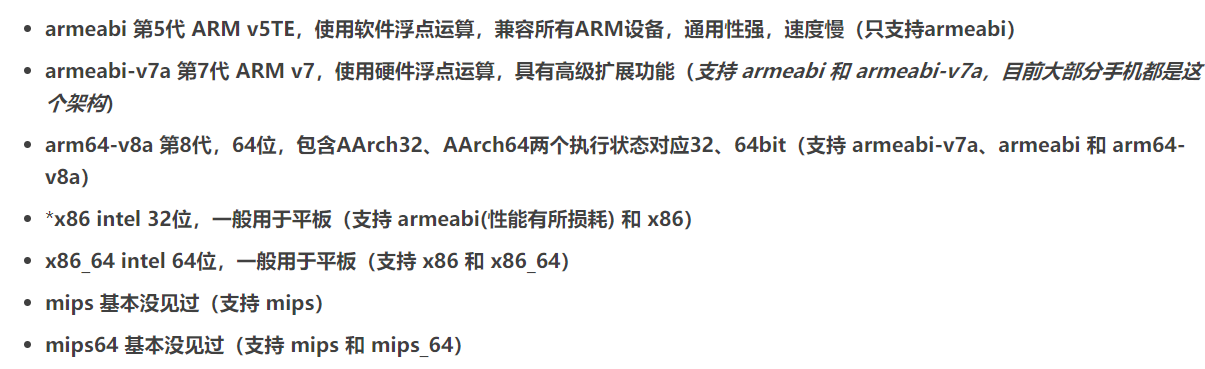
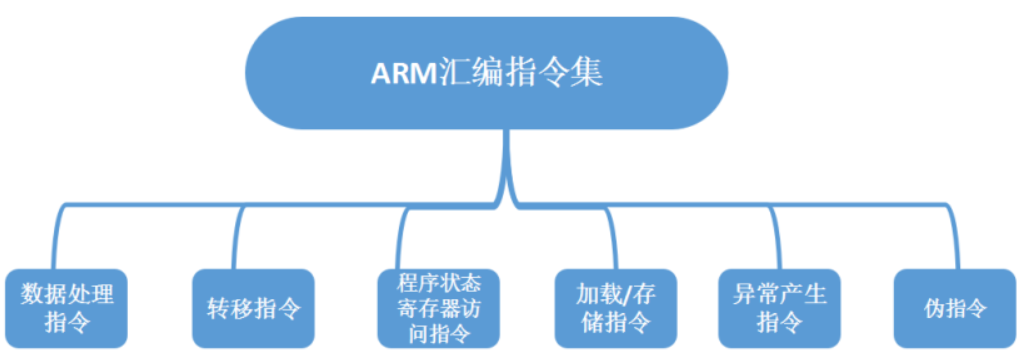
# 1 Android设备支持的CPU架构





# 2 ARM汇编指令集



## 一、汇编数据处理指令

### 1.数据传送指令

#### 【MOV指令】

它的传送指令只能是把一个寄存器的值(要能用立即数表示)赋给另一个寄存器，或者将一个常量赋给寄存器，将后边的量赋给前边的量。

MOV指令的格式为：

MOV{条件}{S} 目的寄存器，源操作数

复制代码

MOV指令中，条件缺省时指令无条件执行；S选项决定指令的操作是否影响CPSR中条件标志位的值，当没有S时指令不更新CPSR中条件标志位的值。

指令示例：

MOV R1，R0 ；将寄存器R0的值传送到寄存器R1

MOV PC，R14 ；将寄存器R14的值传送到PC，常用于子程序返回

MOV R1，R0，LSL＃3 ；将寄存器R0的值左移3位后传送到R1（即乘8）

MOVS PC, R14 ；将寄存器R14的值传送到PC中，返回到调用代码并恢复标志位

#### 【MVN指令】

除了MOV指令外，还有数据取反传送指令MVN。

MVN指令的格式为：

MVN{条件}{S} 目的寄存器，源操作数

复制代码

MVN指令可完成从另一个寄存器、被移位的寄存器、或将一个立即数加载到目的寄存器。与MOV指令不同之处是在传送之前按位被取反了，即把一个被取反的值传送到目的寄存器中。其中S决定指令的操作是否影响CPSR中条件标志位的值，当没有S时指令不更新CPSR中条件标志位的值。

指令示例：

MVN R0，＃0 ；将立即数0取反传送到寄存器R0中，完成后R0=-1（有符号位取反）

### 2.算术运算指令

#### 【加法指令】：ADD

ADD指令的格式为：

ADD{条件}{S} 目的寄存器，操作数1，操作数2

复制代码

ADD指令用于把两个操作数相加，并将结果存放到目的寄存器中。操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。

指令示例：

ADD R0，R1，R2 ； R0 = R1 + R2

ADD R0，R1，#256 ； R0 = R1 + 256

ADD R0，R2，R3，LSL#1 ； R0 = R2 + (R3 << 1)

#### 【带进位的加法指令】：ADC

ADC指令的格式为：

ADC{条件}{S} 目的寄存器，操作数1，操作数2

ADC指令用于把两个操作数相加，再加上CPSR中的C条件标志位的值，并将结果存放到目的寄存器中。

它使用一个进位标志位，这样就可以做比32位大的数的加法，注意不要忘记设置S后缀来更改进位标志。操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。

以下指令序列完成两个128位数（此处应为两个四字数相加）的加法，第一个数由高到低存放在寄存器R7～R4，第二个数由高到低存放在寄存器R11～R8，运算结果由高到低存放在寄存器R3～R0。

指令示例：

ADDS R0，R4，R8 ； 加低端的字，R0=R4+R8

ADCS R1，R5，R9 ； 加第二个字，带进位,R1=R5+R9

ADCS R2，R6，R10 ； 加第三个字，带进位,R2=R6+R10

ADC R3，R7，R11 ； 加第四个字，带进位,R3=R7+R11

#### 【减法指令】：SUB

SUB指令的格式为：

SUB{条件}{S} 目的寄存器，操作数1，操作数2

SUB指令用于把操作数1减去操作数2，并将结果存放到目的寄存器中。操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。该指令可用于有符号数或无符号数的减法运算。

指令示例：

SUB R0，R1，R2 ； R0 = R1 - R2

SUB R0，R1，#256 ； R0 = R1 - 256

SUB R0，R2，R3，LSL#1 ； R0 = R2 - (R3 << 1)

#### 【带借位减法指令】：SBC

SBC指令的格式为：

SBC{条件}{S} 目的寄存器，操作数1，操作数2

BC指令用于把操作数1减去操作数2，再减去CPSR中的C条件标志位的反码，并将结果存放到目的寄存器中。

操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。该指令使用进位标志来表示借位，这样就可以做大于32位的减法，注意不要忘记设置S后缀来更改进位标志。该指令可用于有符号数或无符号数的减法运算。

指令示例：

SUBS R0，R1，R2 ；R0 = R1 - R2 - ！C，并根据结果设置CPSR的进位标志位

### 3.比较指令

#### 【直接比较指令】：CMP

CMP指令的格式为：

CMP{条件} 操作数1，操作数2

CMP指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行比较，同时更新CPSR中条件标志位的值。

该指令进行一次减法运算，但不存储结果，只更改条件标志位。标志位表示的是操作数1与操作数2的关系(大、小、相等)，例如，当操作数1大于操作操作数2，则此后的有GT 后缀的指令将可以执行。

指令示例：

CMP R1，R0 ；将寄存器R1的值与寄存器R0的值相减，并根据结果设置CPSR的标志位

CMP R1，＃100 ；将寄存器R1的值与立即数100相减，并根据结果设置CPSR的标志位

#### 【负数比较指令】：CMN

CMN指令的格式为：

CMN{条件} 操作数1，操作数2

CMN指令用于把一个寄存器的内容和另一个寄存器的内容或立即数取反后进行比较，同时更新CPSR中条件标志位的值。

该指令实际完成操作数1和操作数2相加，并根据结果更改条件标志位。

指令示例：

CMN R1，R0 ；将寄存器R1的值与寄存器R0的值相加，并根据结果设置CPSR的标志位

CMN R1，＃100 ；将寄存器R1的值与立即数100相加，并根据结果设置CPSR的标志位

### 4.逻辑运算指令

#### 【逻辑与指令】：AND

AND指令的格式为：

AND{条件}{S} 目的寄存器，操作数1，操作数2

AND指令用于在两个操作数上进行逻辑与运算，并把结果放置到目的寄存器中。

操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于屏蔽操作数1的某些位。

指令示例：

AND R0，R0，＃3 ； 该指令保持R0的0、1位，其余位清零。

#### 【逻辑或指令】：ORR

ORR指令的格式为：

ORR{条件}{S} 目的寄存器，操作数1，操作数2

ORR指令用于在两个操作数上进行逻辑或运算，并把结果放置到目的寄存器中。

操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于设置操作数1的某些位。

指令示例：

ORR R0，R0，＃3 ； 该指令设置R0的0、1位，其余位保持不变。

#### 【逻辑异或指令】：EOR

EOR指令的格式为：

EOR{条件}{S} 目的寄存器，操作数1，操作数2

EOR指令用于在两个操作数上进行逻辑异或运算，并把结果放置到目的寄存器中。

操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于反转操作数1的某些位。

指令示例：

EOR R0，R0，＃3 ； 该指令反转R0的0、1位，其余位保持不变。

#### 【位清零指令】：BIC

BIC指令的格式为：

BIC{条件}{S} 目的寄存器，操作数1，操作数2

BIC指令用于清除操作数1的某些位，并把结果放置到目的寄存器中。操作数1应是一个寄存器，操作数2可以是一个寄存器，被移位的寄存器，或一个立即数。

操作数2为32位的掩码，如果在掩码中设置了某一位，则清除这一位。未设置的掩码位保持不变。

指令示例：

BIC R0，R0，＃％1011 ； 该指令清除 R0 中的位 0、1、和 3，其余的位保持不变。

### 5.测试指令

#### 【位测试指令】：TST

TST指令的格式为：

TST{条件} 操作数1，操作数2

TST指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行按位的与运算，并根据运算结果更新CPSR中条件标志位的值。

操作数1是要测试的数据，而操作数2是一个位掩码，该指令一般用来检测是否设置了特定的位。

指令示例：

TST R1，＃％1 ；用于测试在寄存器R1中是否设置了最低位（％表示二进制数）

TST R1，＃0xffe ；将寄存器R1的值与立即数0xffe按位与，并根据结果设置CPSR的标志位

#### 【位测试指令】：TEQ

TEQ指令的格式为：

TEQ{条件} 操作数1，操作数2

TEQ指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行按位的异或运算，并根据运算结果更新CPSR中条件标志位的值。该指令通常用于比较操作数1和操作数2是否相等。

指令示例：

TEQ R1，R2 ；将寄存器R1的值与寄存器R2的值按位异或，并根据结果设置CPSR的标志位

### 6.乘法指令



## 二、汇编转移指令

【跳转指令】

跳转指令用于实现程序流程的跳转，在ARM程序中有两种方法可以实现程序流程的跳转：

— 使用专门的跳转指令。

— 直接向程序计数器PC写入跳转地址值。

通过向程序计数器PC写入跳转地址值，可以实现在4GB的地址空间中的任意跳转，在跳转之前结合使用MOV LR，PC等类似指令，可以保存将来的返回地址值，从而实现在4GB连续的线性地址空间的子程序调用。

ARM指令集中的跳转指令可以完成从当前指令向前或向后的32MB的地址空间的跳转，包括以下4条指令：

— B 跳转指令

— BL 带返回的跳转指令

— BLX 带返回和状态切换的跳转指令

— BX 带状态切换的跳转指令

### 1.【B指令】

B指令的格式为：

B{条件} 目标地址

B指令是最简单的跳转指令。一旦遇到一个 B 指令，ARM 处理器将立即跳转到给定的目标地址，从那里继续执行。注意存储在跳转指令中的实际值是相对当前PC值的一个偏移量，而不是一个绝对地址，它的值由汇编器来计算（参考寻址方式中的相对寻址）。

它是 24 位有符号数，左移两位后有符号扩展为 32 位，表示的有效偏移为 26 位(前后32MB的地址空间)。

以下指令：

B Label ；程序无条件跳转到标号Label处执行

CMP R1，＃0 ；当CPSR寄存器中的Z条件码置位时，程序跳转到标号Label处执行

BEQ Label

### 2.【BL指令】

BL指令的格式为：

BL{条件} 目标地址

BL 是另一个跳转指令，但跳转之前，会在寄存器R14中保存PC的当前内容，因此，可以通过将R14 的内容重新加载到PC中，来返回到跳转指令之后的那个指令处执行。

该指令是实现子程序调用的一个基本但常用的手段。

以下指令：

BL Label ；当程序无条件跳转到标号Label处执行时，同时将当前的PC值保存到R14中

### 3.【BLX指令】

BLX指令的格式为：

BLX 目标地址

BLX指令从ARM指令集跳转到指令中所指定的目标地址，并将处理器的工作状态有ARM状态切换到Thumb状态，该指令同时将PC的当前内容保存到寄存器R14中。

因此，当子程序使用Thumb指令集，而调用者使用ARM指令集时，可以通过BLX指令实现子程序的调用和处理器工作状态的切换。同时，子程序的返回可以通过将寄存器R14值复制到PC中来完成。

### 4.【BX指令】

BX指令的格式为：

BX{条件} 目标地址

BX指令跳转到指令中所指定的目标地址，目标地址处的指令既可以是ARM指令，也可以是Thumb指令。

## 三、汇编程序状态寄存器访问指令

### 1.【MRS指令】

MRS指令的格式为：

MRS{条件} 通用寄存器，程序状态寄存器（CPSR或SPSR）

MRS指令用于将程序状态寄存器的内容传送到通用寄存器中。该指令一般用在以下几种情况：

－ 当需要改变程序状态寄存器的内容时，可用MRS将程序状态寄存器的内容读入通用寄存器，修改后再写回程序状态寄存器。

－ 当在异常处理或进程切换时，需要保存程序状态寄存器的值，可先用该指令读出程序状态寄存器的值，然后保存。

指令示例：

MRS R0，CPSR ；传送CPSR的内容到R0

MRS R0，SPSR ；传送SPSR的内容到R0

### 2.【MSR指令】

MSR指令的格式为：

MSR{条件} 程序状态寄存器（CPSR或SPSR）\_<域>，操作数

MSR指令用于将操作数的内容传送到程序状态寄存器的特定域中。其中，操作数可以为通用寄存器或立即数。<域>用于设置程序状态寄存器中需要操作的位，32位的程序状态寄存器可分为4个域：

位[31：24]为条件标志位域，用f表示；

位[23：16]为状态位域，用s表示；

位[15：8]为扩展位域，用x表示；

位[7：0]为控制位域，用c表示；

该指令通常用于恢复或改变程序状态寄存器的内容，在使用时，一般要在MSR指令中指明将要操作的域。

指令示例：

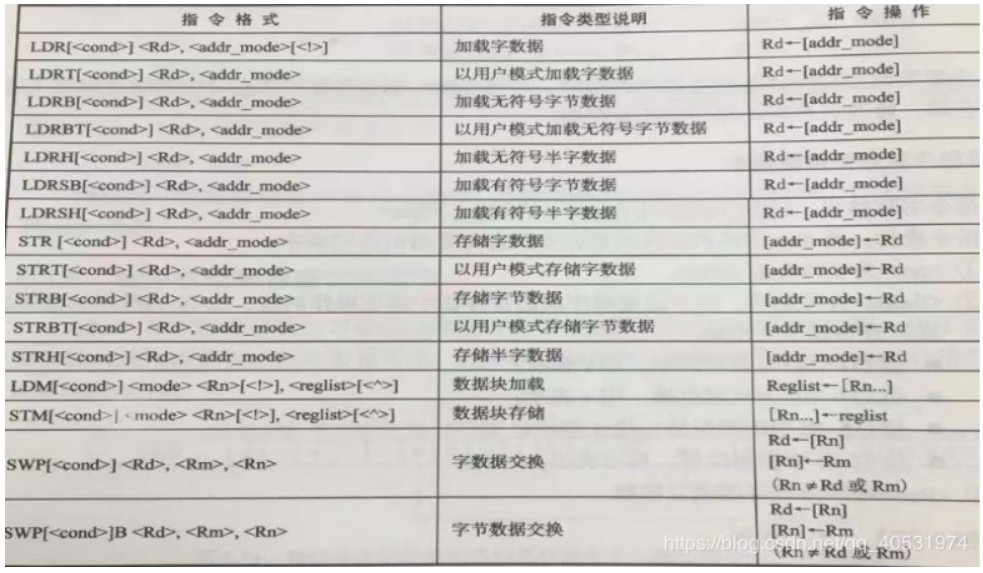
MSR CPSR，R0 ；传送R0的内容到CPSR

MSR SPSR，R0 ；传送R0的内容到SPSR

MSR CPSR\_c，R0 ；传送R0的内容到SPSR，但仅仅修改CPSR中的控制位域

## 四、汇编加载/存储指令

ARM微处理器支持加载/存储指令用于在寄存器和存储器之间传送数据，加载指令用于将存储器中的数据传送到寄存器，存储指令则完成相反的操作。常用的加载存储指令如下：



### 1.【LDR指令】

LDR指令的格式为：

LDR{条件} 目的寄存器，<存储器地址>

LDR指令用于从存储器中将一个32位的字数据传送到目的寄存器中。该指令通常用于从存储器中读取32位的字数据到通用寄存器，然后对数据进行处理。

当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDR R0，[R1] ；将存储器地址为R1的字数据读入寄存器R0。

LDR R0，[R1，R2] ；将存储器地址为R1+R2的字数据读入寄存器R0。

LDR R0，[R1，＃8] ；将存储器地址为R1+8的字数据读入寄存器R0。

LDR R0，[R1，R2] ！ ；将存储器地址为R1+R2的字数据读入寄存器R0，并将新地址R1＋R2写入R1。

LDR R0，[R1，＃8] ！ ；将存储器地址为R1+8的字数据读入寄存器R0，并将新地址R1＋8写入R1。

LDR R0，[R1]，R2 ；将存储器地址为R1的字数据读入寄存器R0，并将新地址R1＋R2写入R1。

LDR R0，[R1，R2，LSL＃2]！ ；将存储器地址为R1＋R2×4的字数据读入寄存器R0，并将新地址R1＋R2×4写入R1。

LDR R0，[R1]，R2，LSL＃2 ；将存储器地址为R1的字数据读入寄存器R0，并将新地址R1＋R2×4写入R1。

### 2.【LDRB指令】

LDRB指令的格式为：

LDR{条件}B 目的寄存器，<存储器地址>

LDRB指令用于从存储器中将一个8位的字节数据传送到目的寄存器中，同时将寄存器的高24位清零。

该指令通常用于从存储器中读取8位的字节数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDRB R0，[R1] ；将存储器地址为R1的字节数据读入寄存器R0，并将R0的高24位清零。

LDRB R0，[R1，＃8] ；将存储器地址为R1＋8的字节数据读入寄存器R0，并将R0的高24位清零。

### 3.【LDRH指令】

LDRH指令的格式为：

LDR{条件}H 目的寄存器，<存储器地址>

LDRH指令用于从存储器中将一个16位的半字数据传送到目的寄存器中，同时将寄存器的高16位清零。

该指令通常用于从存储器中读取16位的半字数据到通用寄存器，然后对数据进行处理。当程序计数器PC作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

LDRH R0，[R1] ；将存储器地址为R1的半字数据读入寄存器R0，并将R0的高16位清零。

LDRH R0，[R1，＃8] ；将存储器地址为R1＋8的半字数据读入寄存器R0，并将R0的高16位清零。

LDRH R0，[R1，R2] ；将存储器地址为R1＋R2的半字数据读入寄存器R0，并将R0的高16位清零。

### 4.【STR指令】

STR指令的格式为：

STR{条件} 源寄存器，<存储器地址>

STR指令用于从源寄存器中将一个32位的字数据传送到存储器中。该指令在程序设计中比较常用，且寻址方式灵活多样，使用方式可参考指令LDR。

指令示例：

STR R0，[R1]，＃8 ；将R0中的字数据写入以R1为地址的存储器中，并将新地址R1＋8写入R1。

STR R0，[R1，＃8] ；将R0中的字数据写入以R1＋8为地址的存储器中。

### 5.【STRB指令】

STRB指令的格式为：

STR{条件}B 源寄存器，<存储器地址>

STRB指令用于从源寄存器中将一个8位的字节数据传送到存储器中。该字节数据为源寄存器中的低8位。

指令示例：

STRB R0，[R1] ；将寄存器R0中的字节数据写入以R1为地址的存储器中。

STRB R0，[R1，＃8] ；将寄存器R0中的字节数据写入以R1＋8为地址的存储器中。

### 6.【STRH指令】

STRH指令的格式为：

STR{条件}H 源寄存器，<存储器地址>

STRH指令用于从源寄存器中将一个16位的半字数据传送到存储器中。该半字数据为源寄存器中的低16位。

指令示例：

STRH R0，[R1] ；将寄存器R0中的半字数据写入以R1为地址的存储器中。

STRH R0，[R1，＃8] ；将寄存器R0中的半字数据写入以R1＋8为地址的存储器中。

### 7.【LDM（或STM）指令】

ARM微处理器所支持批量数据加载/存储指令可以一次在一片连续的存储器单元和多个寄存器之间传送数据，**批量加载**指令用于将一片连续的存储器中的数据传送到多个寄存器，批量数据存储指令则完成相反的操作。常用的加载存储指令如下：

— LDM 批量数据加载指令

— STM 批量数据存储指令

LDM（或STM）指令的格式为：

LDM（或STM）{条件}{类型} 基址寄存器{！}，寄存器列表{∧}

LDM（或STM）指令用于从由基址寄存器所指示的一片连续存储器到寄存器列表所指示的多个寄存器之间传送数据，该指令的常见用途是将多个寄存器的内容入栈或出栈。其中，{类型}为以下几种情况：

IA 每次传送后地址加1；

IB 每次传送前地址加1；

DA 每次传送后地址减1；

DB 每次传送前地址减1；

FD 满递减堆栈；

ED 空递减堆栈；

FA 满递增堆栈；

EA 空递增堆栈；

{！}为可选后缀，若选用该后缀，则当数据传送完毕之后，将最后的地址写入基址寄存器，否则基址寄存器的内容不改变。

基址寄存器不允许为R15，寄存器列表可以为R0～R15的任意组合。

{∧}为可选后缀，当指令为LDM且寄存器列表中包含R15，选用该后缀时表示：除了正常的数据传送之外，还将SPSR复制到CPSR。

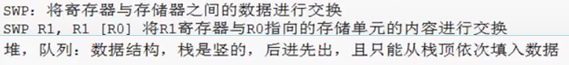
同时，该后缀还表示传入或传出的是用户模式下的寄存器，而不是当前模式下的寄存器。

指令示例：

STMFD R13!，{R0，R4-R12，LR} ；将寄存器列表中的寄存器（R0，R4到R12，LR）存入堆栈。

LDMFD R13!，{R0，R4-R12，PC} ；将堆栈内容恢复到寄存器（R0，R4到R12，LR）。

### 8. 【SWP指令】



## 五、汇编异常产生指令

### 1.【SWI指令】

SWI指令的格式为：

SWI{条件} 24位的立即数

SWI指令用于产生软件中断，以便用户程序能调用操作系统的系统例程。操作系统在SWI的异常处理程序中提供相应的系统服务，指令中24位的立即数指定用户程序调用系统例程的类型，相关参数通过通用寄存器传递，当指令中24位的立即数被忽略时，用户程序调用系统例程的类型由通用寄存器R0的内容决定，同时，参数通过其他通用寄存器传递。

指令示例：

SWI 0x02 ；该指令调用操作系统编号位02的系统例程。

### 2.【BKPT指令】

BKPT指令的格式为：

BKPT 16位的立即数

BKPT指令产生软件断点中断，可用于程序的调试。

## 六、汇编伪代码

### 1.【AREA】

一个汇编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，因此在汇编程序的开头，我们一般的语句会用到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。

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

....

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

### 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

## 参考链接

<https://blog.csdn.net/qq_40531974/article/details/83897559>