# 目录

# 指令和伪指令概念

## 指令

指令指的是 CPU 机器指令的助记符，是由 CPU 的指令集提供的，经过编译之后，会以二进制机器码的形式由 CPU 读取执行

## 伪指令

伪指令本质上不是指令，和 CPU 的机器指令没有任何关系，只是和指令一起写在代码中而已，是由编译器环境提供的，其目的是用于指导编译过程，伪指令经过编译后不会生成二进制机器码，仅仅在编译阶段有效果

## 1、指令编程风格

### ARM 官方风格

官方风格指令一般使用大写，例如：LDR R0,[R1]，Windows 中常使用这种风格

### GUN Linux 风格

指令一般使用小写字母，例如：ldr r0,[r1]，Linux 环境中常用这种风格

## 2、ARM 汇编特点

### LDR/STR 架构

1、采用 RISC 架构，CPU 本身不能直接读取内存，而需要把内存中的数据加载到 CPU 的通用寄存器中，才能被 CPU 处理
2、ldr（load register）将内存中的数据加载到通用寄存器
3、str（store register）将寄存器内容存入内存空间
4、ldr 和 str 组合，可以实现 ARM CPU 和内存的数据交换

### 8 种寻址方式

**寄存器寻址：**move r1,r2：把 r2 的值赋值到 r1 寄存器中

**立即寻址**：move r0,#0xFF00：把立即数 0xFF00 赋值给 r0 寄存器
**寄存器移位寻址**：move r0,r1,lsl #3：把 r1 左移三位(*8)之后的值赋值给 r0 寄存器
**寄存器间接寻址**：ldr r1,[r2]：寄存器有中括号，表示内存地址对应的数据，所以这里 r2 表示一个内存地址,[]表示取 r2 指针对应的数据，这句代码的意思是把 r2 对应的内存中的数据赋值给 r1
**基址变址寻址**：ldr r1,[r2,#4]：将指针 r2 的值(内存地址)+4 之后指向的数据赋值给 r1
**多寄存器寻址**：ldmia r1!,{r2 - r7,r12}：这种情况下，r1 是一个指针，里边存放的内存地址，然后以 r1 里边的内存地址为基地址，向后以此加 1 得到{}里的寄存器数量个内存地址，然后将刚才得到的这些内存地址指向的变量的值赋值给{}里的对应位置的寄存器，类似从内存中读取数组，然后把数组的元素依次赋值给这些寄存器
**堆栈寻址**：stmfd sp!,{r2 - r7,lr}：和多寄存器类似，区别是将栈 SP 中连续访问{}数量个字节，然后依次赋值给{}里的寄存器
**相对寻址**：beq flag:：flag:标号用于标记标号后面那句指令的地址，常用来表示入口点，函数名就是一个标号，C 语言中的 goto 就可以跳转到一个标号，在 ARM 汇编中用指令 b flag:就可以跳转到 flag:对应的标号处执行，和 beq flag:是一样的，其原理是相对于 PC 程序位置寄存器做一个偏移

# 指令后缀

ARM 中的指令可以带后缀，从而丰富该指令的功能，这种形式叫做指令族，常用的后缀有：
1、B(byte)：功能不变，操作长度变为 8 位(依赖 CPU 位数，以下相同)
2、H(Halfword)：功能不变，操作长度变为 16 位
3、S(signed)：功能不变，操作数变为有符号数
4、S(S 标识)：影响 CPSR 里的 NZCV 标识位，

举例：
ldr 指令族：ldrb,ldrh,ldrsb ldrsh，从内存中加载指定长度的数据
mov 指令族：movs r0,#0，结果是 0，赋值会影响 CPSR 的 NZCV 标识，将 Z 位置为 1

# 条件执行后缀

条件执行后缀用于限制该执行执行的，只有在符合条件之后才能够执行该指令

| 操作码 | 条件码助记符 | 标志 | 含义 | | 操作码 | 条件码助记符 | 标志 | 含义 |
|---|---|---|---|---|---|---|---|---|
| 0000 | EQ | Z=1 | 相等 | | 1000 | HI | C=1,Z=0 | 无符号数大于 |
| 0001 | NE | Z=0 | 不相等 | | 1001 | LS | C=0,Z=1 | 无符号数小于或等于 |
| 0010 | CS/HS | C=1 | 无符号数大于或等于 | | 1010 | GE | N=V | 有符号数大于或等于 |
| 0011 | CC/LO | C=0 | 无符号数小于 | | 1011 | LT | N!=V | 有符号数小于 |
| 0100 | MI | N=1 | 负数 | | 1100 | GT | Z=0,N=V | 有符号数大于 |
| 0101 | PL | N=0 | 正数或零 | | 1101 | LE | Z=1,N!=V | 有符号数小于或等于 |
| 0110 | VS | V=1 | 溢出 | | 1110 | AL | 任意 | 无条件执行（指令默认条件） |
| 0111 | VC | V=0 | 没有溢出 | | 1111 | NV | 任意 | 从不执行（不要使用） |

1、举例：moveq r0,r1，如果 eq 成立，执行 mov r0,r1，不成立则该条不执行，和 C 语言中的条件判断类似
2、条件后缀成立与否，不是取决于本条指令，而是取决于之前指令运行后的结果
3、条件后缀决定了本条指令是否执行，不会影响之前和之后指令
4、条件后缀和 CPSR 的 NZCV 位相关，例如，如果上一句代码执行的结果将 Z 置为 1，下一句带有 eq 条件后缀的语句就会被执行

## 多级指令流水线

1、多级流水线用于增加处理器处理指令的速度，
2、允许 CPU 同时异步的执行多条指令，而非上一条指令全部执行完毕之后才会执行下一条指令
3、多级可以简单那理解为把一条指令分为多个步骤来异步执行，例如：
1) CPU 把一条指令分为[取址，解码，执行]3 个步骤，则为 3 级指令流水线
2) 第一条指令进行取值操作
3) 第一条指令取值完毕，进入解码操作，第二条指令紧随其后就开始执行取值操作
4) 第一条指令解码完毕，进入执行操作，第二条指令紧接着进入解码操作，同时第三条指令进入取值操作
5) 第一条指令执行完毕，第二条指令进入执行操作，第三条指令进入解码操作，第四条指令进入取值操作，依次类推
4、可见，多级流水线可以提高同时执行指令的数量，从而加速指令执行
5、需要注意的是，PC 指向的是正在取值的指令，而非正在执行的指令，之间的差值就是流水线级数和单字节长度的乘积，在中断返回到 PC 的时候需要注意这个问题

# 3、ARM 指令

## 数据处理指令

### 1、数据传输指令

mov：move，在两个寄存器之间或者立即数和寄存器之间传递数据，将后一个寄存器上的值或者立即数赋值给前一个寄存器
例如：mov r1,r0
mov r1,#0xFF：将立即数 0xFF 赋值给寄存器 r1
mvn：和 mov 用法一致，区别是 mvn 会把后一个寄存器的值或者立即数按位取反后赋值给前一个寄存器
例如：mvn r0,#0xFF，则 r0 的值为 0xffffff00(32 位数据)

### 2、算术运算指令

add：加法运算
sub：减法运算
rsb：反减运算
adc： 带进位的加法运算
sbc： 带进位的减法运算
rsc：带进位的反减指令

**3、逻辑指令**
and：与操作
orr：或操作
eor：异或操作
bic：位清除操作
**4、比较指令**
cmp：比较大小
cmn：取反比较
tst：按位与运算
teq：按位异或运算
**5、乘法指令**
mvl： mla： umull： umlal： smull： smlal：
**6、前导 0 计数**
clz：统计一个数的二进制位前面有几个 0

## CPSR 访问指令

**mrs**
用于读取 CPSR 和 SPSR

**msr**
用于写 CPSR 和 SPSR

**CPSR 和 SPSR**

CPSR 是程序状态寄存器，整个 Soc 只有一个
SPSR 在五种异常模式下各有一个，用于从普通模式进入异常模式的时候，保存普通模式下的 CPSR，在返回普通模式时可以恢复原来的 CPSR

## 跳转分支指令

b 指令： 无条件直接跳转，没打算返回
bl 指令：跳转前把返回地址放入 lr 中，以便返回，常用在函数中
bx 指令：跳转同时切换到 ARM 模式，用于异常处理的跳转

## 内存访问指令

ldr：加载指定内存地址的数据到寄存器，按照字节访问
str：加载指定寄存器数据到内存地址中，按照字节访问
ldm：和 ldr 功能一样，一次多字节多寄存器访问
stm：和 str 功能一样，一次多字节多寄存器访问
swp：内存和寄存器互换指令，一边读一边写，例如：
swp r1,r2,[r0]：读取指针 r0 的数据到 r1 中，同时把 r2 的数据赋值给 r0 指针指向的变量

## 软中断指令

swi(software interrupt),在软件层模拟产生一个中断,这个中断会传送给 CPU,常用于实现系统调用

# 4、立即数

## 非法与合法

ARM 指令都是 32 为，除了指令标记和操作标记外，只能附带少位数的立即数，所以有非法与合法之分

非法立即数：
合法立即数：经过任意位数的移位后，非 0 部分可以用 8 位表示就是合法立即数

# 5、协处理器与指令

## 协处理器

协处理器属于 Soc 中另外一颗核心，用于协助主 CPU 实现某些功能，被主 CPU 调用来执行任务，协处理器和 MMU，Cache，TLB 有功能和管理上的联系
ARM 设计可以支持多达 16 个协处理器，但是一般只实现其中的 CP15

## 协处理器指令

mrc：读取 CP15 中的寄存器
mcr：向 CP15 中的寄存器写数据

指令格式：
mcr{<”cond”>}p15,<”opcode_1”>,<”Rd”>,<”Crn”>,<”Crm”>,{<”opcode_2”>}

- opcode_1：对于 CP15 永远为 0
- Rd：ARM 通用寄存器
- Crn：CP15 寄存器，取值范围 c0~c15
- Crm：CP15 寄存器，一般为 c0
- opcode_2：省略或者为 0

# 6、ldm,stm 和栈

## ldm,stm

ldr 与 str 只能访问 4 个字节，当数据较大的时候，就会明显的降低效率，这时就需要使用到 ldm 和 stm,ldm 与 stm 是大量的从寄存器与内存交换数据的方式，常用于在内存和寄存器之间大量读取和写入数据：

1、stmia sp {r0 - r12}：stm 表示进行批量数据操作，ia 的意思是将 r0 存入 SP 的内存地址处，然后 SP 内存地址+4(32 位)，将 r1 存入该地址，内存地址再+4，存入 r2，依次存到 r12，

2、这就是一个寄存器和内存交换大量数据的示例，在一个周期内完成了多个内存地址和多个寄存器的操作。

3、除了 ia 后缀，还有多个不同功能的后缀：
ia：increase after，后增加，表示每个操作的时候，先传输数据，后增加内存地址，

- ib：increase before，先增加，表示在每个操作的时候，先增加内存地址，再进行数据传输
- da：decrease after：和 ia 一样，差别在于减少地址
- db：decrease before：和 ib 一样，差别在于减少地址
- fd：full decrease：满递减堆栈，查看栈的描述
- ed：empty decrease：空递减堆栈
- fa：满递增堆栈
- ea：空递增堆栈

操作栈时使用相同的后缀就不会出错

## 堆栈(栈)

1) 空栈：栈指针指向空位，每次可以直接存入，然后栈指针(SP)递增或者递减 1 格，取的时候要递增或者递减 1 格才能取出
2) 满栈：SP 指向栈最后 1 格数据，存入的时候需要先移动 1 格才能存入，取的时候可以直接取出
3) 增栈：SP 向地址增加的方向移动
4) 减栈：SP 向地址减少的方向移动
5) 组合起来就可以成为空增/减栈，满增/减栈。四种形式

# 7、汇编中的符号

## ！号的作用

在汇编中常见”！”符号，究竟是用来干嘛的呢？
例如：

ldmia r0,{r2 - r3}：数据传输完毕之后指针 r0 的内存地址值还是第一次操作之前的值
ldmia r0!,{r2 - r3}：数据传输完毕之后指针 r0 的内存地址值是最后一次操作后的值

## ^号的作用

那么^号又是什么作用呢？
例如：

ldmfd sp!,{r0 - r6,pc}
ldmfd sp!,{r0 - r6,pc}^
作用：当目标寄存器中有 pc 时，会同时将 SPSR 写入到 CPSR，一般用于从异常返回。

# 8、伪指令

伪指令的作用用于指导编译过程，伪指令并不是指令，编译后并不会生成二进制机器码，伪指令是和具体的编译环境有关的，我们使用的 GNU 编译工具链，所以需要使用 GNU 下的汇编伪指令。

## GNU 汇编符号

- [@，#，//，/~/]：注释，和 C 语言的//是一样的
- :：冒号，在汇编中以冒号结尾的是标号，标号标记标号后面的指令的地址，
- .：点号，代表当前指令的地址，例如：[b .]指令会进入死循环
- #：#或者$号后面跟着数值，代表一个立即数(不区分进制)

## 伪指令

- .global _start：给_start 外部可链接属性，可以在外部文件中访问_start
- .section .text：指定当前段为代码段
- .ascii .byte .short .long .word .quad .float .string：定义各种类型的数据
- .align 4：以 16 字节对齐
- .balignl 16 0x3C：b 表示填充，align 表示对齐，l 表示 long，以 4 字节为单位填充，16 表示以 16 字节对齐，0x3C 是用来填充的原料
- .equ：宏定义
- .end：表示一个文件的结束
- .include：用于包含头文件
- .arm / .code32：声明以下的代码是 arm 指令
- .thumb / .code16：声明以下的代码是 thumb 指令

## 比较重要的伪指令

- nop：空操作，什么也不执行
- ldr：大范围地址加载指令，把内存地址加载到寄存器中，和 ldr 指令是有区别的
- ldr 指令附带立即数要考虑合法性问题，只能带合法立即数，带的立即数前缀#号
- ldr 伪指令，不用考虑立即数合法问题，带的立即数前面是一个=号，借助了编译器环境文字池的帮助，帮忙加载非法立即数
- ldr 可以加载的地址比较广，加载的地址和链接时给的地址有关，由连接脚本决定，在链接时确定，编译时会被 mov 或者以文字池方式处理
- adr：小范围地址加载指令，可加载的范围比较小
- adr：总是以 P C 为基准来表示地址，则该地址为相对地址，要在执行的时候才能确定，因此该指令和运行地址有关，可以用于检测程序当前运行地址在哪里，编译时会被 sub 或者 add 替代
- adrl：中等范围地址加载指令

# 大小端

大端模式，是指数据的高位，保存在内存的低地址中，而数据的低位，保存在内存的高地址中，这样的存储模式有点儿类似于把数据当作字符串顺序处理：地址由小向大增加，而数据从高位往低位放。

小端模式，是指数据的高位保存在内存的高地址中，而数据的低位保存在内存的低地址中，这种存储模式将地址的高低和数据位权有效地结合起来，高地址部分权值高，低地址部分权值低，和我们的逻辑方法一致。

例如：
16 位数 0x1122 在内存中存放方式：

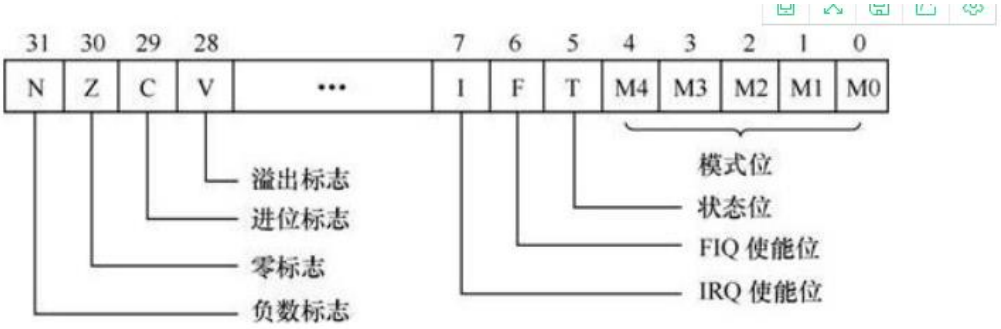
| 内存地址 | 小端模式 | 大端模式 |
| --- | --- | --- |
| 0x6000 | 0x22 | 0x11 |
| 0x6001 | 0x11 | 0x22 |

32 位数 0x11223366 在内存中存放方式：

| 内存地址 | 小端模式 | 大端模式 |
| --- | --- | --- |
| 0x6000 | 0x66 | 0x11 |
| 0x6000 | 0x33 | 0x22 |
| 0x6000 | 0x22 | 0x33 |
| 0x6000 | 0x11 | 0x66 |

# CPSR 程序状态寄存器

CPSR 程序状态寄存器标识各种状态的工作模式




这个状态寄存器的 32 位可以分为四个 8 位的域

| | |
|---|---|
| Bit[31:24] | 条件标志域，用 f表示 |
| Bit[23:16] | 状态位域，用 s表示 |
| Bit[15:8] | 扩展位域，用 x表示 |
| Bit[7:0] | 控制位域，用 c表示 |

条件码的说明：高 4 位是 4 个状态位 NZCV：是负数、0、进借位、溢出
控制位前三位的说明：6-7 位：I、F、T：分别代表有无中断、快速中断、运行在 ARM 还是 THUMB 态

# 指令详解

## 数据加载/存储指令

### 【LDR 和 STR】

## LDR 和 STR 使用 1

（1）LDR：L 表示 LOAD　（从内存中装载到寄存器中）
例如：LDR R1, [R2]
方向：R1<—[R2]
就是把 R2 所指向的存储单元的内容的值（即一个 memory 地址内的值），读取到 R1 中（一个 register）
（2）STR：S 表示 STORE （ 从寄存器中保存到内存）
例如：STR R1, [R2]
方向：R1—>[R2]
就是把寄存器 R1 中的内容“保存”到 R2 所指向的存储的单元中（一个 memory 地址）。

## LDR 和 STR 使用 2

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
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。

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。
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。
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。
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。

## 【批量数据加载/存储指令指令】

## STM 和 LDM 使用 1

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】
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。

## STM 和 LDM 使用 2

（在实际使用中，和 sp 一起使用更多）
STM 和 LDM 的主要用途是现场保护、数据复制、参数传递等，其模式有 8 种，如下：
前面 4 种用于数据块的传输，后面 4 种用于堆栈操作
I：指地址是增加的；
D：指地址是减少的；
A（after）:地址在操作之后增加或减少
B（before）:地址在操作之前增加或减少

（1）IA 每次传送后地址加 4 -- Inc After
（2）IB 每次传送前地址加 4 -- Inc Before
（3）DA 每次传送后地址减 4 -- Dec After
（4）DB 每次传送前地址减 4 -- Dec Before
（5）FD 满递减堆栈
（6）FA 满递增堆栈
（7）ED 空递减堆栈
（8）EA 空递增堆栈

**（3）LDM**：L 的含义仍然是 LOAD，但是指令运行是从左到右运行的。该指令是将内存中堆栈内的数据，批量的赋值给寄存器，即是出栈操作；其中堆栈指针一般对应于 SP（ SP 是寄存器 R13），实际用到的却是 R13 中的内存地址，只是该指令没有写为[R13]
例如：LDMFD SP! ,{R0, R1, R2}
可理解为：LDMFD [SP]!,{R0, R1, R2}
方向：[sp] —>{R0, R1, R2}
即为：把 sp 指向的 3 个连续地址段（3*4=12 字节（将 r0,r1,r2 在 ARM 中作为是 32 位来计算））中的数据拷贝到 r0,r1,r2 这 3 个寄存器中去
**（4）STM**：S 的含义仍然是 STORE，与 LDM 是配对使用的，其指令运行是将堆栈指针写在左边，而把寄存器组写在右边。
例如：STMFD SP!,{R0}
可理解为：STMFD [SP]!,{R0}
方向：[sp]<—{R0}
即为：把 R0 保存到堆栈（sp 指向的地址）中

# 数据处理指令

## 【数据传送指令】

【MOV 指令】

定义：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
格式：MOV{条件}{S} 目的寄存器，源操作数
注　：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，当后缀有 S 时，这些指令根据结果更新标志 N 和 Z，在计算 Operand2 时更新标志 C，不影响标志 V

【MVN 指令】

定义：传送和 mov 指令相似，不同点在将数据取反后传送
格式：MVN{条件}{S} 目的寄存器，源操作数

## 【算术运算指令】

【ADD 指令】
定义：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
格式：ADD{条件}{S} 目的寄存器，操作数 1，操作数 2

【带进位加法 ADC 指令】

定义：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
格式：ADC{条件}{S} 目的寄存器，操作数 1，操作数 2

【SUB 减法指令】

格式：SUB{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。

【带借位减法 SBC 指令】

格式：SBC{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。
注：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

## 【比较指令】

【直接比较 CMP 指令】

格式：CMP{条件} 操作数 1，操作数 2
定义：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。
注 ： 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。

【负数比较 CMN 指令】

格式：CMN{条件} 操作数 1，操作数 2
定义：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。
注 ：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。

## 【逻辑运算指令】

【逻辑与 AND 指令】
格式：AND{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。
注 :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。

【逻辑或 ORR 指令】
格式：ORR{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。
注 ：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。
【逻辑异或 EOR 指令】
格式：EOR{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。
注：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。
【位清零 BIC 指令】
格式：BIC{条件}{S} 目的寄存器，操作数 1，操作数 2
定义：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。
注 ：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。

## 【测试指令】

【位测试 TST 指令】
格式：TST{条件} 操作数 1，操作数 2
定义：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。
【位测试 TEQ 指令】
格式：TEQ{条件} 操作数 1，操作数 2
定义：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。

### 【跳转分支指令】

b 指令： 无条件直接跳转，没打算返回
bl 指令：跳转前把返回地址放入 lr 中，以便返回，常用在函数中
bx 指令：跳转同时切换到 ARM 模式，用于异常处理的跳转

## 访问指令

### 【状态寄存器访问指令】

ARM 微处理器支持程序状态寄存器访问指令，用于在程序状态寄存器和通用寄存器之间传送数据，程序状态寄存器访问指令包括以下两条。
【MRS 指令】
格式：MRS{条件} 通用寄存器，程序状态寄存器（CPSR 或 SPSR）
功能：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两种情况。
① 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。
② 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。例如：

```
MRS R0, CPSR   ; 传送 CPSR 的内容到 R0
MRS R0, SPSR   ; 传送 SPSR 的内容到 R0
```

【MSR 指令】
格式：MSR 程序状态寄存器(CPSR 或 SPSR)_<域>，操作数
功能：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。例如：

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

### 【协处理器指令】

ARM 微处理器可支持多达 16 个协处理器，用于各种协处理操作，在程序执行的过程中，每个协处理器只执行针对自身的协处理指令，忽略 ARM 处理器和其他协处理器的指令。

ARM 的协处理器指令主要用于 ARM 处理器初始化 ARM 协处理器的数据处理操作，以及在 ARM 处理器的寄存器和协处理器的寄存器之间传送数据，在 ARM 协处理器的寄存器和存储器之间传送数据。

【CDP 指令】
格式：CDP{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2
功能：CDP 指令用于 ARM 处理器通知 ARM 协处理器执行特定的操作，若协处理器不能成功完成特定的操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器和源寄存器均为协处理器的寄存器，指令不涉及 ARM 处理器的寄存器和存储器。例如：

```
CDP   P3,2,C12,C10,C3,4          ;该指令完成协处理器 P3 的初始化
```

【LDC 指令】
格式：LDC{条件}{L} 协处理器编码，目的寄存器，[源寄存器]
功能：LDC 指令用于将源寄存器所指向的存储器中的字数据传送到目的寄存器中，若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。例如：

```
LDC   P3,C4,[R0]          ;将 ARM 处理器的寄存器 R0 所指向的存储器中的字数据传送到协处理器 P3 的寄存器 C4 中
```

【STC 指令】
格式：STC{条件}{L} 协处理器编码，源寄存器，[目的寄存器]
功能：STC 指令用于将源寄存器中的字数据传送到目的寄存器所指向的存储器中，若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。例如：

```
STC   P3,C4,[R0]          ;将协处理器 P3 的寄存器 C4 中的字数据传送到 ARM 处理器的寄存器 R0 所指向的存储器中
```

【MCR 指令】
格式：MCR{条件} 协处理器编码，协处理器操作码 1，源寄存器，目的寄存器 1，目的寄存器 2，协处理器操作码 2
功能：MCR 指令用于将 ARM 处理器寄存器中的数据传送到协处理器寄存器中，若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，源寄存器为 ARM 处理器的寄存器，目的寄存器 1 和目的寄存器 2 均为协处理器的寄存器。例如：

```
MCR   P3,3,R0,C4,C5,6       ;该指令将 ARM 处理器寄存器 R0 中的数据传送到协处理器 P3 的寄存器 C4 和 C5 中
```

【MRC 指令】
格式：MRC{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2
功能：MRC 指令用于将协处理器寄存器中的数据传送到 ARM 处理器寄存器中，若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器为 ARM 处理器的寄存器，源寄存器 1 和源寄存器 2 均为协处理器的寄存器。例如：

```
MRC   P3,3,R0,C4,C5,6       ;该指令将协处理器 P3 的寄存器中的数据传送到 ARM 处理器寄存器中
```