# 1 ARM汇编指令集

## 1.1指令、伪指令

（汇编）指令：是机器码的助记符，经过汇编器编译后，由CPU执行。

（汇编）伪指令：用来指导指令执行，是汇编器的产物，最终不会生成机器码。

## 1.2有两种不同风格的ARM指令

1).ARM官方的ARM汇编风格：指令一般用大写，Windows中的IDE开发环境。

2).GNU风格的ARM汇编：指令一般用小写。

# 2 ARM汇编的特点

## 2.1 LDR/STR架构

1).ARM采用RISC架构，CPU本身不能直接读取内存，而需要先将内存中内容加载入CPU中通用寄存器中才能被CPU处理。

2).ldr（load register）指令将内存内容加载入通用寄存器。

3).str（store register）指令将寄存器内容存入内存空间中。

4).ldr/str组合用来实现 ARM CPU和内存数据交换。

## 2.2 八种寻址方式

1).寄存器寻址 mov r1, r2。

2).立即(立即数)寻址 mov r0, #0xFF00。

3).寄存器移位寻址 mov r0, r1, lsl #3。

4).寄存器间接寻址 ldr r1, [r2] 表示内存，内存地址存在r2这个寄存器中，把内存地址里的值给r1。

5).基址变址寻址ldr r1, [r2, #4]内存地址在r2+4里面。

6).多寄存器寻址 ldmia r1!, {r2-r7, r12}一次访问多个寄存器。

7).堆栈寻址 stmfd sp!, {r2-r7, lr}。

8).相对寻址 beq flag。

## 2.3 指令后缀

同一指令经常附带不同后缀，变成不同的指令。经常使用的后缀有：

B（byte）功能不变，操作长度变为8位

H（half word）功能不变，长度变为16位

S（signed）功能不变，操作数变为有符号

如 ldr ldrb ldrh ldrsb ldrsh

S（S标志）功能不变，影响CPSR标志位

如 mov和movs movs r0, #0

## 2.4 条件执行后缀

条件后缀是否成立取决于当前代码的前面的代码。条件后缀只影响当前代码的执行。

## 2.5 多级指令流水线

为增加处理器指令流的速度，ARM使用多级流水线.，下图为3级流水线工作原理示意图。（S5PV210使用13级流水线，ARM11为8级，ARM9为5级）

允许多个操作同时处理，而非顺序执行。

1).PC指向正被取指的指令，而非正在执行的指令

# 3 数据传输与跳转指令详解

## 3.1 数据处理指令

数据传输指令 mov mvn

算术指令 add sub rsb adc sbc rsc

逻辑指令 and orr eor bic

比较指令 cmp cmn tst teq

乘法指令 mvl mla umull umlal smull smlal

前导零计数 clz

## 3.2 cpsr访问指令

mrs & msr

mrs用来读psr，msr用来写psr

CPSR寄存器比较特殊，需要专门的指令访问，这就是mrs和msr。

## 3.3 跳转(分支)指令

b & bl & bx

b 直接跳转（就没打开算返回）

bl branch and link，跳转前把返回地址放入lr中，以便返回，以便用于函数调用

bx跳转同时切换到ARM模式，一般用于异常处理的跳转。

## 3.4 访存指令

ldr/str & ldm/stm & swp

单个字/半字/字节访问 ldr/str

多字批量访问 ldm/stm

swp r1, r2, [r0]

swp r1, r1, [r0]

## 3.5 软中断指令

swi（software interrupt）

软中断指令用来实现OS中系统调用

ARM汇编中的立即数

合法立即数与非法立即数

ARM指令都是32位，除了指令标记和操作标记外，本身只能附带很少位数的立即数。因此立即数有合法和非法之分。

合法立即数：经过任意位数的移位后非零部分可以用8位表示的即为合法立即数。

# 4 协处理器与协处理器指令集

## 4.1 协处理器cp15操作指令

mcr & mrc

mrc用于读取CP15中的寄存器

mcr用于写入CP15中的寄存器

## 4.2.arm寻址方式

1). 寄存器： MOV R1,R2 ; R2->R1

2). 立即数： SUBS R0,R1,#1; R0=R1-1

3). 寄存器移位：MOV R0,R2,LSL #3 ;R2左移三位->R0

4). 间接寻址： LDR R1,[R2] ; 装载R2指向的内存数值至R1

5). 基址寻址： LDR R2,[R3,#0x0F]；R3+0x0F作为地址，将所指向的值装入R2。R3的值不改变

6). 多寄存器寻址： LDMIA R1!,{R2-R7,R12}；将R1所指向的内存块依次装入{}中的寄存器STMIA R0!,{R3-R6,R10};将{}列出的寄存器里的值依次填入R0所指向的内存块。

7). 相对寻址： BL XXX ;跳转

BEQ XXX ;条件跳转

协处理器解析：

SoC内部另一处理核心，协助主CPU实现某些功能，被主CPU调用执行一定任务。

ARM设计上支持多达16个协处理器，但是一般SoC只实现其中的CP15.（cp：coprocessor）

协处理器和MMU、cache、TLB等处理有关，功能上和操作系统的虚拟地址映射、cache管理等有关。

MRC & MCR的使用方法

mcr{<cond>} p15, <opcode\_1>, <Rd>, <Crn>, <Crm>, {<opcode\_2>}

opcode\_1：对于cp15永远为0

Rd：ARM的普通寄存器

Crn：cp15的寄存器，合法值是c0～c15

Crm：cp15的寄存器，一般均设为c0

opcode\_2：一般省略或为。

ldm/stm与栈的处理

为什么需要多寄存器访问指令

ldr/str每周期只能访问4字节内存，如果需要批量读取、写入内存时太慢，解决方案是stm/ldm

ldm(load register mutiple)

stm（store register mutiple）

举例（uboot start.S 537行）

stmia sp, {r0 - r12}

将r0存入sp指向的内存处（假设为0x30001000）；然后地址+4（即指向0x30001004），将r1存入该地址；然后地址再+4（指向0x30001008），将r2存入该地址······直到r12内容放入（0x3001030），指令完成。

一个访存周期同时完成13个寄存器的读写

后缀的种类：

ia（increase after）先传输，再地址+4

ib（increase before）先地址+4，再传输

da（decrease after）先传输，再地址-4

db（decrease before）先地址-4，再传输

fd（full decrease）满递减堆栈

ed（empty decrease）空递减堆栈

fa（·······） 满递增堆栈

ea（·······）空递增堆栈

四种栈解析：

空栈：栈指针指向空位，每次存入时可以直接存入然后栈指针移动一格；而取出时需要先移动一格才能取出。

满栈：栈指针指向栈中最后一格数据，每次存入时需要先移动栈指针一格再存入；取出时可以直接取出，然后再移动栈指针。

增栈：栈指针移动时向地址增加的方向移动的栈。

减栈：栈指针移动时向地址减小的方向移动的栈。

!的作用:

ldmia r0, {r2 - r3}

ldmia r0！, {r2 - r3}

感叹号的作用就是r0的值在ldm过程中发生的增加或者减少最后写回到r0去，也就是说ldm时会改变r0的值。

^的作用:

ldmfd sp!, {r0 - r6, pc}

ldmfd sp!, {r0 - r6, pc}^

^的作用：在目标寄存器中有pc时，会同时将spsr写入到cpsr，一般用于从异常模式返回。

总结：批量读取或写入内存时要用ldm/stm指令。

各种后缀以理解为主，不需记忆，最常见的是stmia和stmfd。

谨记：操作栈时使用相同的后缀就不会出错，不管是满栈还是空栈、增栈还是减栈。

常用gun伪指令：

global \_start @ 给\_start外部链接属性

.section .text @ 指定当前段为代码段

.ascii .byte .short .long .word

.quad .float .string @ 定义数据

.align 4 @ 以4字节对齐

.balignl 16 0xabcdefgh @ 16字节对齐填充

.equ @ 类似于C中宏定义

偶尔会用到的gun伪指令

.end @标识文件结束

.include @ 头文件包含

.arm / .code32 @声明以下为arm指令

.thumb / .code16 @声明以下为thubm指令

重要的几个伪指令

ldr 大范围的地址加载指令

adr 小范围的地址加载指令

adrl 中等范围的地址加载指令

nop 空操作

ARM中有一个ldr指令，还有一个ldr伪指令

一般都使用ldr伪指令而不用ldr指令

adr与ldr

adr编译时会被1条sub或add指令替代，而ldr编译时会被一条mov指令替代或者文字池方式处理；

adr总是以PC为基准来表示地址，因此指令本身和运行地址有关，可以用来检测程序当前的运行地址在哪里

ldr加载的地址和链接时给定的地址有关，由链接脚本决定。