

# [arm-汇编stmdb、ldmia、stmfd、ldmfd]



作者 放风筝的小小马 (/u/897e846b632f) [+ 关注](#)

2017.03.17 15:49 字数 1758 阅读 126 评论 0 喜欢 0

(/u/897e846b632f)

## STMFD

- ST - store
- M - Multiple
- F - FULL
- D - Descending

## LDMFD

- LD - Load
- M - Multiple
- F - FULL
- D - Descending

栈指针通常可以指向不同的位置。栈指针指向栈顶元素（即最后一个入栈的数据元素）时称为**FULL栈**；栈指针指向与栈顶元素相邻的一个可用书局单元时称为**EMPTY栈**。

数据栈的增长方向也可以不同。当数据栈向内存地址减小的方向增长时，称为**Descending栈**；当数据栈向内存地址增加的方向增长时，称为**Ascending栈**

综合上面两点，可以存在以下四种数据栈：

- FD - Full Descending
- ED - Empty Descending
- FA - Full Ascending
- EA - Empty Ascending

因此实际上存在下面这些批量load/save指令：

LDMFA, LDMFD, LDMEA, LDMED

STMED, STMEA, STMFD, STMFA

给定数据栈对应着的特定批量load/save指令，也决定了地址变化方式：

比如FD栈，对应的批量传送指令是LDMFD/STMFD，对应的地址变化方式是：

- **IA(事后递增方式)**

- **DB(事先递减方式)**

```
STMFD SP!, {R0~R7, LR}
start_address = sp - 9 * 4      //先压栈，然后增长sp
end_address = sp - 4
```



把寄存器r0~r7和LR共9个寄存器，存储到start\_address开始，到end\_address结束的栈中，并且修改SP的值（SP变小），相当于压栈。

```
LDMFD SP!, {R0~R7, LR}
start_address = SP
end_address = SP + 9 * 4
```

把堆栈从start\_address开始，到end\_address内的值恢复到寄存器R0, R1... R7和LR中，并修改SP的值(SP变大)，相当于出栈。

首先一句话说一下stmdb和ldmia指令的作用：

**stmdb和ldmia指令一般配对使用，stmdb用于将寄存器压栈，ldmia用于将寄存器弹出栈，作用是保存使用到的寄存器。**

ARM指令的多数据传输（STM、LDM）中，提到：多寄存器的Load和Store指令分为2组：一组用于数据的存储与读取，对应于IA、IB、DA、DB，一组用于堆栈操作，对应于FD、ED、FA、EA，两组中对应的指令含义相同。

即：

STMIB（地址先增而后完成操作）、STMFA（满递增堆栈）；  
 STMIA（完成操作而后地址递增）、STMEA（空递增堆栈）；  
 STMDB（地址先减而后完成操作）、STMFD（满递减堆栈）；  
 STMDA（完成操作而后地址递减）、STMED（空递减堆栈）。  
 上述各组2个指令含义相同只是适用场合不同，同理有：  
 LDMIB、LDMED；  
 LDMIA、LDMFD；  
 LDMDB、LDMEA；  
 LMDA、LDMFA。

IA模式表示：每次传送后地址+4；(After Increase)DB模式表示：每次传送前地址-4；(Before Decrease)多寄存器加载/存储指令共有8种模式（4个用与数据块的传输，4个用于栈操作）

**举例一：**

```
指令: stmdb sp!,{r0-r12,lr}
```

含义：sp = sp - 4，先压lr，sp = lr（即将lr中的内容放入sp所指的内存地址）。sp = sp - 4，再压r12，sp = r12。sp = sp - 4，再压r11，sp = r11.....sp = sp - 4，最后压r0，sp = r0。

如果想要将r0-r12和lr弹出，可以用ldmia指令：

```
指令: ldmia sp!,{r0-r12,lr}
```

**举例二：**

**STMIA R0!,{R1-R7}** 将 R1-R7 的数据保存到寄存器中，存储器指针在保存第一个值之后增加，增长方向为向上增长  
**STMDB R0!,{R1-R7}** 将 R1-R7 的数据保存到寄存器中，存储器指针在保存第一个值之前增加，增长方向为向下增长

STMIA，比如当前r0指向的内存地址是0x1000，STMIA R0!,{R1-R7} 就是 首先把r1存入0x1000,然后r2存入0x1004，然后r3存入0x1008，如果是32位的处理器就是每次加4



个字节，以此类推把 r1-r7按照递增的地址存入，这个r0!就是从r0的地址开始存的意思。STMDB则是地址从r0开始减少，依次存储。

## 第二部分代码说明：

先看个例子：

```
void test2(int a,int b,int c)
{
    int k=a,j=b,m=c;
}

GCC反汇编:
00000064 <test2>:
mov     ip, sp                //IP=SP;保存SP
stmdb   sp!, {fp, ip, lr, pc} //先对SP减4，再对fp, ip, lr, pc压栈。-----1
sub     fp, ip, #4            ; 0x4 //fp=ip-4; 此时fp指向栈里面的“fp”
sub     sp, sp, #24           ; 0x18 //分配空间
str     r0, [fp, #-28]        //
str     r1, [fp, #-32]        //
str     r2, [fp, #-36]        //参数压栈
ldr     r3, [fp, #-28]        //
str     r3, [fp, #-24]        //
ldr     r3, [fp, #-32]        //
str     r3, [fp, #-20]        //
ldr     r3, [fp, #-36]        //
str     r3, [fp, #-16]        //
sub     sp, fp, #12           ; 0xc //sp=fp-12; 此时sp指向栈里面的lr
ldmia   sp, {fp, sp, pc}      //弹栈pc=lr, sp=ip, fp=fp. 然后地址加4-----1
```

汇编基础：

```
stmdb   sp!, {fp, ip, lr, pc}
//sp=sp-4,sp=pc;先压PC
//sp=sp-4,sp=lr;再压lr
//sp=sp-4,sp=ip;再压ip
//sp=sp-4,sp=fp;再压fp

ldmia   sp, {fp, sp, pc}
//和stmdb成对使用，
//fp=sp,sp=sp+4;先弹fp
//sp=sp,sp=sp+4;先弹sp，此处的弹出不会影响sp，因为ldmia是一个机器周期执行完的。
//pc=sp,sp=sp+4;先弹pc
LDRH    R0, [R13, #0xC] //加载无符号半字数据，即低16位
LDRB    R0, [R13, #0x4] //加载一字节数据，即低8位。
```

**注意1：** R11=fp;R12=ip;R13=SP；R14=LR；R15=PC；R0,R1,R2用于传递参数和存放函数返回值。

**注意2：** 低地址的寄存器被压入低地址内存中，也就是说如果向下增长，高地址寄存器先压，向上增长测试低地址先压。

**注意3：** 根据“ARM-thumb 过程调用标准”：

1. r0-r3 用作传入函数参数，传出函数返回值。在子程序调用之间，可以将 r0-r3 用于任何用途。被调用函数在返回之前不必恢复 r0-r3。---如果调用函数需要再次使用 r0-r3 的内容，则它必须保留这些内容。
2. r4-r11 被用来存放函数的局部变量。如果被调用函数使用了这些寄存器，它在返回之前必须恢复这些寄存器的值。
3. r12 是内部调用暂时寄存器 ip。它在过程链接胶合代码（例如，交互操作胶合代码）中用于此角色。在过程调用之间，可以将它用于任何用途。被调用函数在返回之前不必恢复 r12。
4. 寄存器 r13 是栈指针 sp。它不能用于任何其它用途。sp 中存放的值在退出被调用函数时必须与进入时的值相同。
5. 寄存器 r14 是链接寄存器 lr。如果您保存了返回地址，则可以在调用之间将 r14 用于其它用途，程序返回时要恢复



6. 寄存器 r15 是程序计数器 PC。它不能用于任何其它用途。
7. 在中断程序中，所有的寄存器都必须保护，编译器会自动保护R4 ~ R11，所以一般你自己只要在程序的开头

```
sub lr,lr,#4
stmfd sp!,{r0-r3,r12,lr};
// 保护R0 ~ R3, R12,LR就可以了，除非你用汇编人为的去改变R4~R11的值。
( 具体去看UCOS os_cpu_a.S中的IRQ中断的代码 )
```

补充：

寄存器名字

Reg # APCS 意义

R0 a1 工作寄存器

R1 a2 "

R2 a3 "

R3 a4 "

R4 v1 必须保护

R5 v2 "

R6 v3 "

R7 v4 "

R8 v5 "

R9 v6 "

R10 sl 栈限制

R11 fp 栈指针

R12 ip

R13 sp 栈指针

R14 lr 连接寄存器

R15 pc 程序计数器

回溯结构

寄存器 fp (栈指针)应当是零或者是指向栈回溯结构的列表中的最后一个结构，提供了一种追溯程序的方式，来反向跟踪调用的函数。

回溯结构是:

地址高端



保存代码指针	[fp]	fp 指向这里
返回 lr 值	[fp, #-4]	
返回 sp 值	[fp, #-8]	
返回 fp 值	[fp, #-12]	指向下一个结构
[保存的 s1]		
[保存的 v6]		
[保存的 v5]		
[保存的 v4]		
[保存的 v3]		
[保存的 v2]		
[保存的 v1]		
[保存的 a4]		
[保存的 a3]		
[保存的 a2]		
[保存的 a1]		
[保存的 f7]		三个字
[保存的 f6]		三个字
[保存的 f5]		三个字
[保存的 f4]		三个字

pc 总是包含下一个要被执行的指令的位置。  
lr (总是)包含着退出时要装载到 pc 中的值。在 26-bit 位代码中它还包含着 PSR。  
sp 指向当前的栈块(chunk)限制，或它的上面。这是用于复制临时数据、寄存器和类似的东西到其中的地方。在 RISC OS 下，你有可选择的至少 256 字节来扩展它。  
fp 要是零，要么指向回溯结构的最当前的部分。

📄 嵌入式 (/nb/10099687)

举报文章 © 著作权归作者所有



放风筝的小小马 (/u/897e846b632f) ♂

写了 128797 字，被 16 人关注，获得了 30 个喜欢  
(/u/897e846b632f)

+ 关注

如果觉得我的文章对您有用，请随意赞赏。您的支持将鼓励我继续创作！

赞赏支持

🤍 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button) | 0



更多分享

(http://cwb.assets.jianshu.io/notes/images/1027656)



登录 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-comment-form) 后发表评论

评论



智慧如你，不想发表一点想法 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-nocomments-text)咩~

推荐阅读

更多精彩内容 > (/)

socket编程 (/p/08ac3a9f3e3e?utm\_campaign=maleskine&utm\_content=...

服务器端 步骤 创建一个socket套接字文件描述符 int serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP); 创建一个sockaddr\_in结构体的变量，并配置相应的参数，然后绑定 struct sockaddr\_in...

放风筝的小小马 (/u/897e846b632f?utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

一款不错的表单样式 (/p/53cbc07401c4?utm\_campai...

(/p/53cbc07401c4?utm\_campaign=maleskine&utm\_content=note&utm...

一款不错的表单样式主要学习表单的HTML结构 CSS样式：

放风筝的小小马 (/u/897e846b632f?utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

在简书学写作，我学会了撩汉子 (/p/db108ecdc5a9?ut...

(/p/db108ecdc5a9?utm\_campaign=maleskine&utm\_content=note&utm...

世间最好的爱情莫过于，我就喜欢你本来的样子。在男神面前，我也还是我。  
2017年9月11日 星期一 晴 01 去年的5月20日，我吃完最后一块生日蛋糕，然...

米那 (/u/546f95e0a658?utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

弗洛伊德教你如何快乐自律 (/p/f0e097dbca46?utm\_c...

(/p/f0e097dbca46?utm\_campaign=maleskine&utm\_content=note&utm...

一、你看过清晨四点的 \* \* 么？最近，朋友圈里总有人每天凌晨四五点在刷着这句话。琪琪就是其中一个。她加入了一个早起团，团里的朋友每天都在比...


伊沙贝 (/u/e6f93ca23584?utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

我成为高管后明白的这些道理，让你少走十年弯路 (/p/...

(/p/5c09446eea98?utm\_campaign=maleskine&utm\_content=note&utm...

最近有位读者私信我，问了下面的问题，这个问题其实很难回答，不是三言两语就可以讲清楚的。思考再三，我决定谈一谈我成为公司高管后的一些感想，也...

萌薇 (/u/c20b62e8e2ba?utm\_campaign=maleskine&utm\_content=user&utm\_medium=pc\_all\_hots&utm\_source=recommendation)

 登录/注册

为你个性化推荐内容

(/sign\_in?utm\_source=desktop&utm\_medium=note&utm\_campaign=maleskine&utm\_content=click-note-bottom-bind)

 下载简书App

随时随地发现和创作内容

(https://itunes.apple.com/app/apple-store/id1036775878?mt=8&utm\_source=desktop&utm\_medium=click-note-bottom-bind)