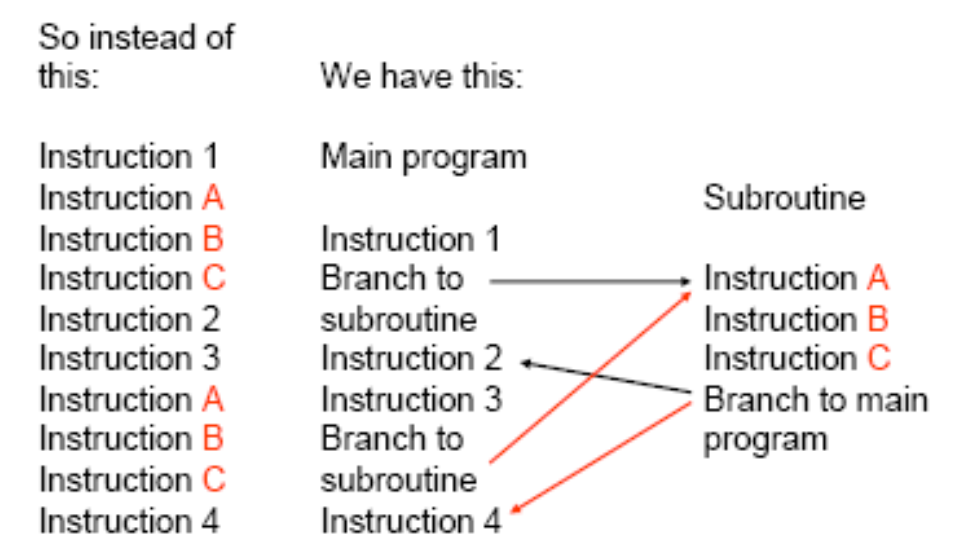
**Subroutine 子进程**

有很多进程会重复地调用某些指令，于是可以把这些指令放到一个特殊的数据结构中，子进程。

当主程序需要调用这些指令时，就branch到子进程，在子进程的最后，有一个branch跳回主程序。



在ARM架构中，r14被用来当做**link register。**

**link register** 记录了子进程要跳回的主程序指令地址。

mnemonic for r14 is lr

MOV pc, lr：将link register里的值存到program counter

如果要在一个子进程内调用另一个子进程：

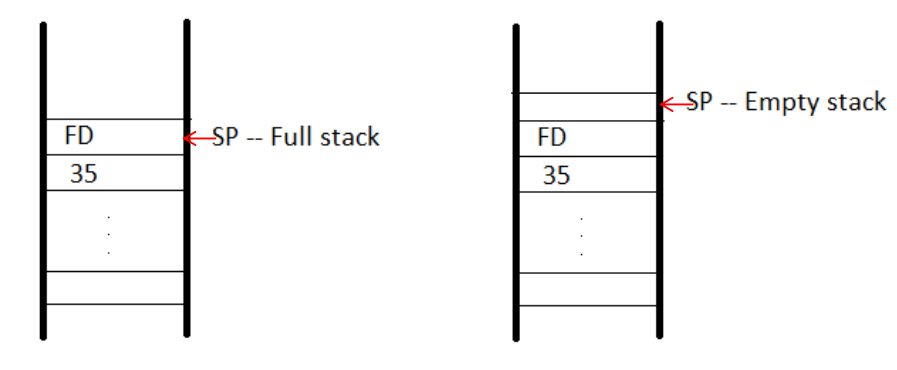
返回主程序的lr的值会被覆盖，所以需要把它在子进程调用另一个子进程之前存到另一个变量里

**Nest Subroutine & Stack**

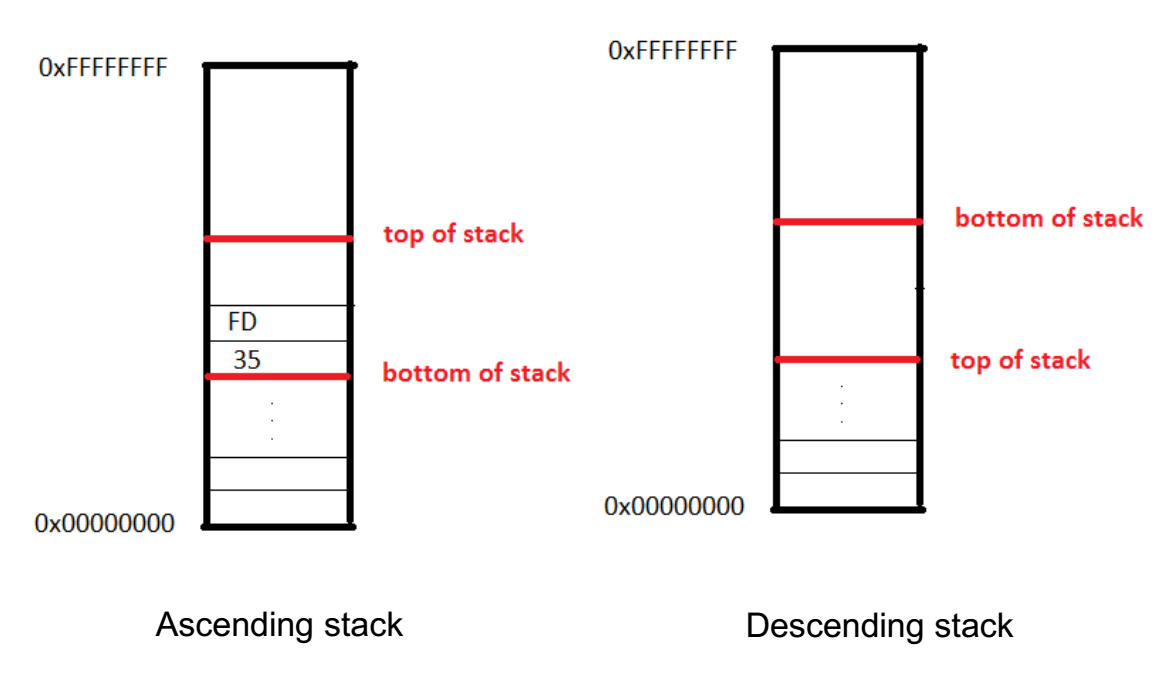
在一些程序里会存在子程序互相调用的情形，在这种时候link register就不能很好地记录return address，这时就需要用到stack

stack里储存的数据量是动态的，所以我们把栈底的地址固定，用stack pointer来记录栈头的位置。

stack pointer在stack满和空的状态时是不一样的。



栈的工作方式也有上升和下降的区别



ARM7架构中 r13被用来表示stack pointer， mnemonic 是sp

STMFD sp!, {lr}：push the link register value onto a full descending stack

LDMFD sp!, {lr}：pop a value from a full descending stack back into the link register

**子进程的一般流程**：

在一开始会调用 STMFD sp!, {lr}

在结束时调用 LDMFD sp!, {lr} ；MOV pc, lr；

也可以用：LDMFD sp!, {pc} 来代替上面这两个指令

对栈的push和pop可以由多个register在同一时间调用。

LDM and STM stand for load multiple and store multiple

E.g.

– to push registers r6, r7, r8 and r9 with the link register use: STMFD sp!,

{r6-r9, lr}

– and to pop the same registers use: LDMFD sp!, {r6-r9, pc}

**Barrel shifter**

MOV r1, r2, LSL #5

在r2被放入r1之前，将r2左移5位

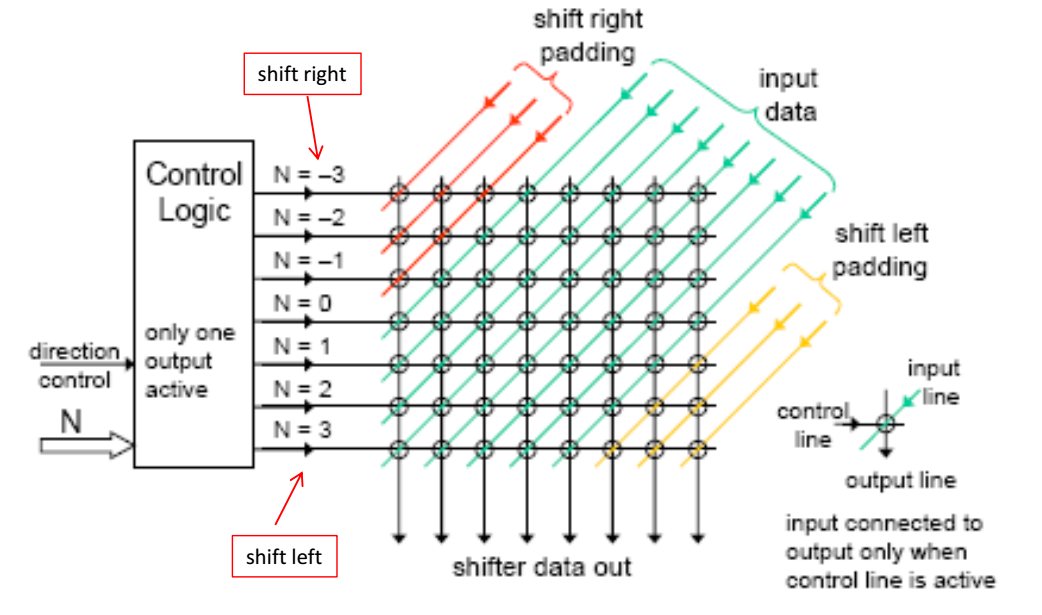
There are different types of

shift e.g.

• logical shift left (LSL),

• logical shift right (LSR)

• and arithmetic shift right (ASR)



ARM提供的五种shift：

LSL逻辑左移：用0填充右边

LSR逻辑右移：用0填充左边

ASR算术右移：用符号位填充左边R

Rotate right(ROR)：右移出去的比特会在左边出现

Rotate extended (RRX) - bits are shifted right one place only and the carry flag is shifted into the new most significant bit. The least significant bit is shifted into the carry flag only if the mnemonic specifies an S.（整体右移一位，用carry flag补最高位，如果mnemonic有S，则将原本最低位设为carry flag）