一般嵌入式开发流程就是先建立一个工程，再编写源文件，然后进行编译，把所有的 *.s 文件和 *.c 文件编译成一个 *.o 文件，再对目标文件进行链接和定位，编译成功后会生成一个 *.hex 文件和调试文件，接下来要进行调试，如果成功的话，就可以将它固化到 flash 里面去。

启动代码是用来初始化电路以及用来为高级语言写的软件作好运行前准备的一小段汇编语言，是任何处理器上电复位时的程序运行入口点。

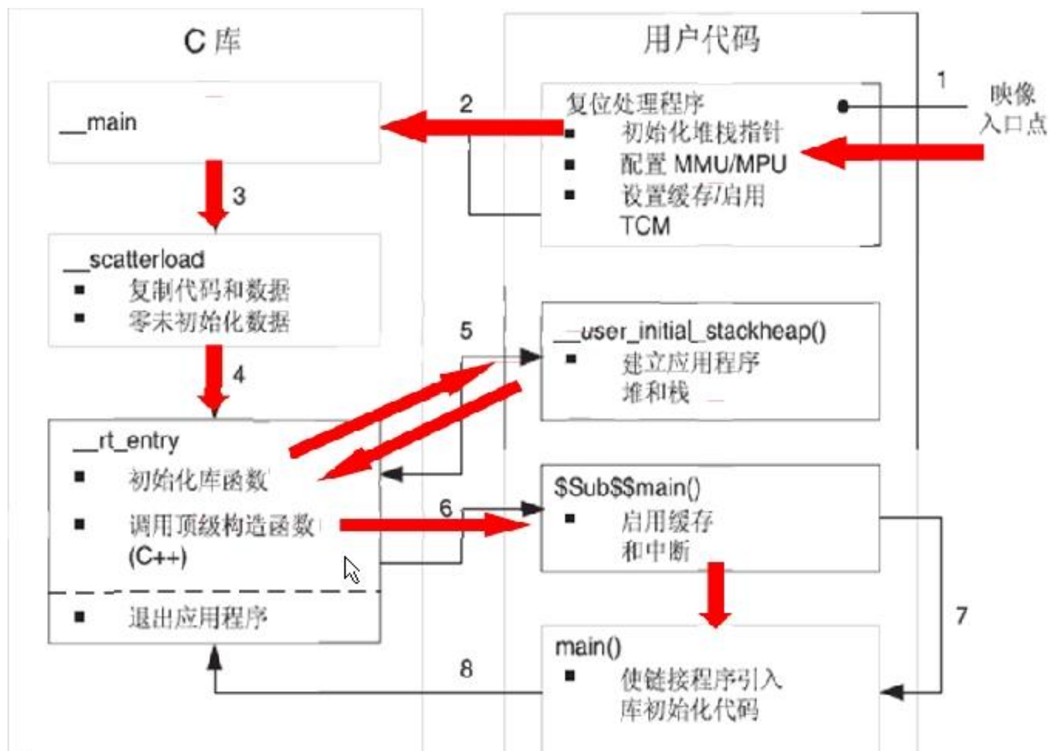
比如，刚上电的过程中，PC 机会对系统的一个运行频率进行锁定在一个固定的值，这个设计频率的过程就是在汇编源代码中进行的，也就是在启动代码中进行的。与此同时，设置完后，程序开始运行，注意，程序是在内存中运行的。这个时候，就需要把一些源文件从 flash 里面 copy 到内存中，又要对它们进行初始化读写，这又有频率的设置。这些都是初始化。

初始化完成后，我们又要设置一些堆栈，要跳到 C 语言的 main 函数里面运行。这就需要堆栈。对普通的 ARM CPU 有这样一个要求：在绝对地址为零的地方要放置一个异常向量表，但并不是所有的 ARM CPU 都留有这个一个空间，这就需要用到映射的功能。我们可以将其它地方的一些空间映射到绝对地址里面。当发生异常时，ARM 核来读取异常中断表的时候，它会使用映射之后的那个表，这个就可以接着往下执行，否则在绝对地址零的地方找不到任何信息，程序就会死掉。这些运行的环境全部建立好后，程序就会跳转到我们的 main 函数里面。

总之，启动代码，就是对最小系统的初始化。 包括晶振，CPU 频率等。

启动代码的最小系统是： 异常向量表的初始化 – 存储区分配 – 初始化堆栈 – 高级语言入口函数调用 – main()函数。

程序的启动过程：



以下面这个例子为例，编译完后， DEBUG 后，我们可以看到，光标指向绝对地址为零的地方，这里存放的就是一个异常向量表。

```
0x00000000  E59FF018  LDR     PC,[PC,#0x0018]
0x00000004  E59FF018  LDR     PC,[PC,#0x0018]
0x00000008  E59FF018  LDR     PC,[PC,#0x0018]
0x0000000C  E59FF018  LDR     PC,[PC,#0x0018]
0x00000010  E59FF018  LDR     PC,[PC,#0x0018]
0x00000014  E1A00000  NOP
0x00000018  E51FFFF0  LDR     PC,[PC,#-0x0FF0]
0x0000001C  E59FF018  LDR     PC,[PC,#0x0018]
```

它对应在 startup.s 里的源文件如下：

```
158 Vectors         LDR     PC, Reset_Addr
159                 LDR     PC, Undef_Addr
160                 LDR     PC, SWI_Addr
161                 LDR     PC, PAbt_Addr
162                 LDR     PC, DAbt_Addr
163                 NOP                              ; Reserved Vector
164 ;               LDR     PC, IRQ_Addr
165                 LDR     PC, [PC, #-0x0FF0]       ; Vector from VicVectAddr
166                 LDR     PC, FIQ_Addr
```

单步运行后，马上跳转到初始化 CPU 的频率。即初始化锁相环，将其锁在一个固定的频率。具体代码如下：

```
; Setup PLL

        IF      PLL_SETUP <> 0

        LDR     R0, =PLL_BASE

        MOV     R1, #0xAA

        MOV     R2, #0x55


;  Configure and Enable PLL

        MOV     R3, #PLLCFG_Val

        STR     R3, [R0, #PLLCFG_OFS]

        MOV     R3, #PLLCON_PLLE

        STR     R3, [R0, #PLLCON_OFS]

        STR     R1, [R0, #PLLFEED_OFS]

        STR     R2, [R0, #PLLFEED_OFS]


;  Wait until PLL Locked

PLL_Loop        LDR     R3, [R0, #PLLSTAT_OFS]
```

```
ANDS    R3, R3, #PLLSTAT_PLOCK
BEQ     PLL_Loop

;  Switch to PLL Clock
MOV     R3, #(PLLCON_PLLE:OR:PLLCON_PLLC)
STR     R3, [R0, #PLLCON_OFS]
STR     R1, [R0, #PLLFEED_OFS]
STR     R2, [R0, #PLLFEED_OFS]
ENDIF   ; PLL_SETUP
```

然后再初始化每一种模式的堆栈，再进行单步运行的时候，下面我们可以看到，它自动跳转到 main（）函数：

```
; Enter the C code

IMPORT  __main
LDR     R0, =__main
BX      R0

IF      :DEF:__MICROLIB

EXPORT  __heap_base
EXPORT  __heap_limit
```

ELSE

这个时候，程序会运行各种 scatterload 函数，将我们的堆栈、全局变量等内容拷贝到内存中去。拷贝完后，就正式跳转到我们的 main() 函数中来执行了。

```
0x00000294  E8BD8070  LDMIA     R13!,{R4-R6,PC}
    51: int main (void) {
    52:   unsigned int n;
    53:
0x00000298  E92D4010  STMDB     R13!,{R4,R14}
    54:   IODIR1 = 0x00FF0000;                    /* P1.16..23
0x0000029C  E3A008FF  MOV       R0,#0x00FF0000
0x000002A0  E59F105C  LDR       R1,[PC,#0x005C]
0x000002A4  E5810018  STR       R0,[R1,#0x0018]
    55:   ADCR   = 0x002E0401;                    /* Setup A/D:
```

这就是启动代码执行的全过程，呵呵，平时我们看到以为只是执行 main() 函数就行了，是不是没有想到在执行 main() 函数后还有这么多学问呢？