**汇编语言设计对ARM\_Cortex\_M0+上LED灯的中断控制**

**第一步**，寄存器变化记录分析：

①RCC.IOPENR 0x0000 0005

②GPIOC.MODER 0xF3FF FFFF

③EXTI.EXTICR4 0x000 0200

④EXTI.IMR1 0xFFF8 2000

⑤EXTI.EMR1 0xFFF8 2000

⑥EXTI.FTSR1 0x0000 2000

⑦GPIOA.MODER 0xEBFF F7FF

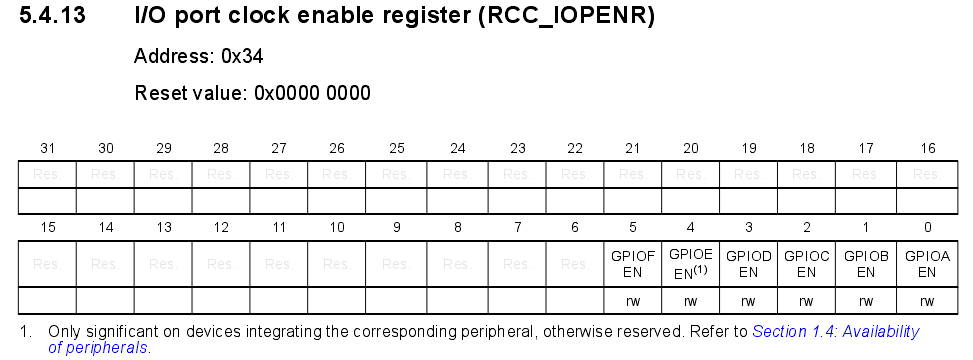
**第二步**，器件手册找寄存器地址：

ece45465660c19f08ae1aecd2703dfd

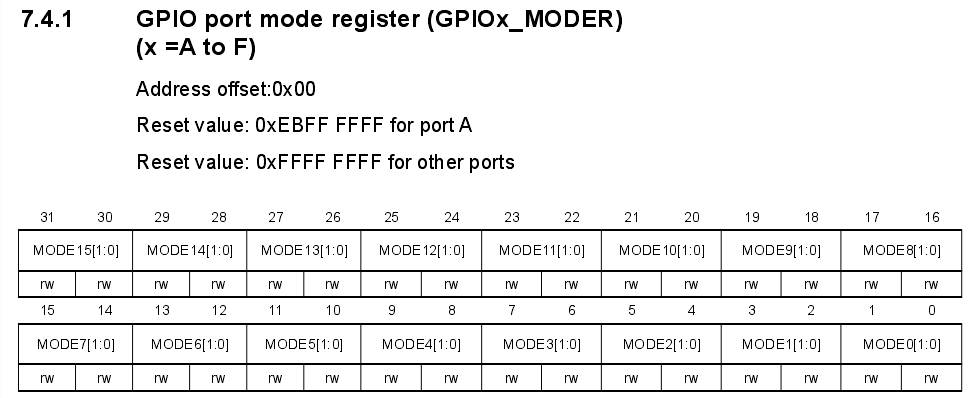
828ecc57cf078d25f14c4d6c28ea5db



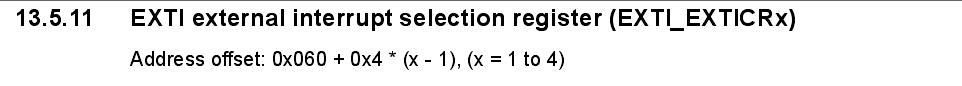
562d5c56a00b234f5c644e25385f386

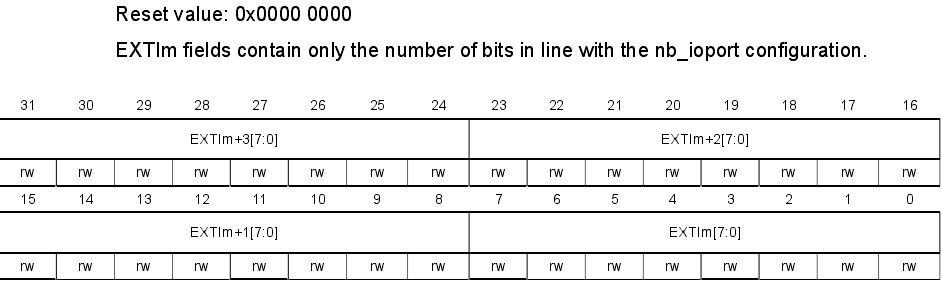


**RCC基址0x4002 1000 IOPENR偏移地址0x34 输入数据0x0000 0005**

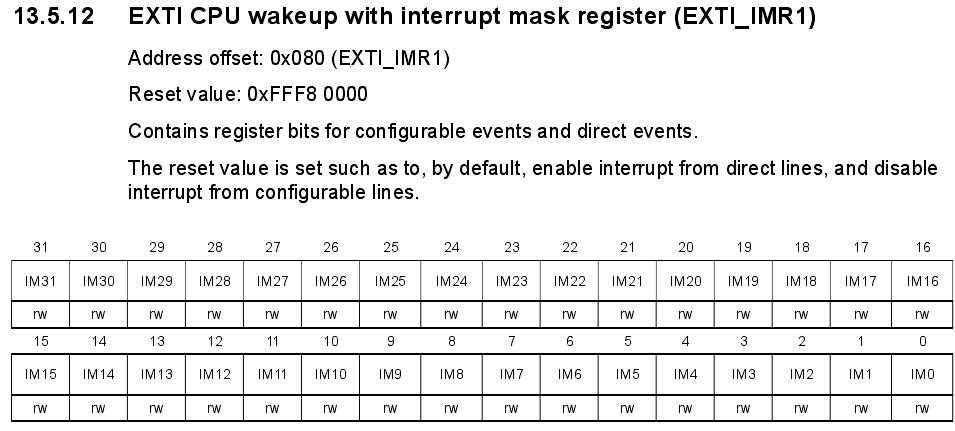


**GPIOC基址0x5000 0800 MODER偏移地址0x00 输入数据0xF3FF FFFF**

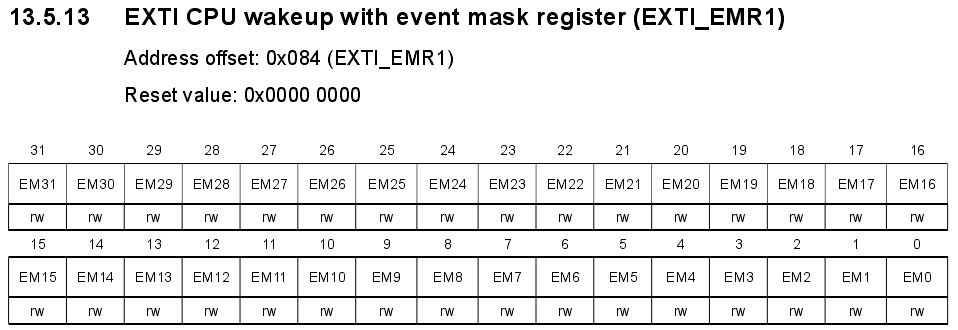




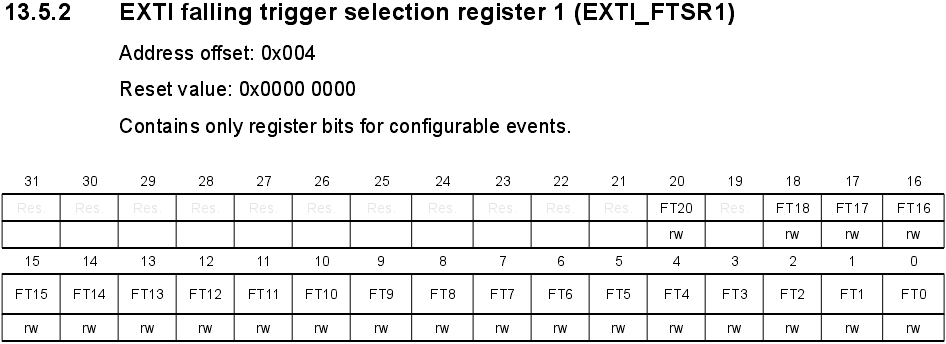
**EXTI基址0x4002 1800 EXTICR4偏移地址0x6C 输入数据0x000 0200**



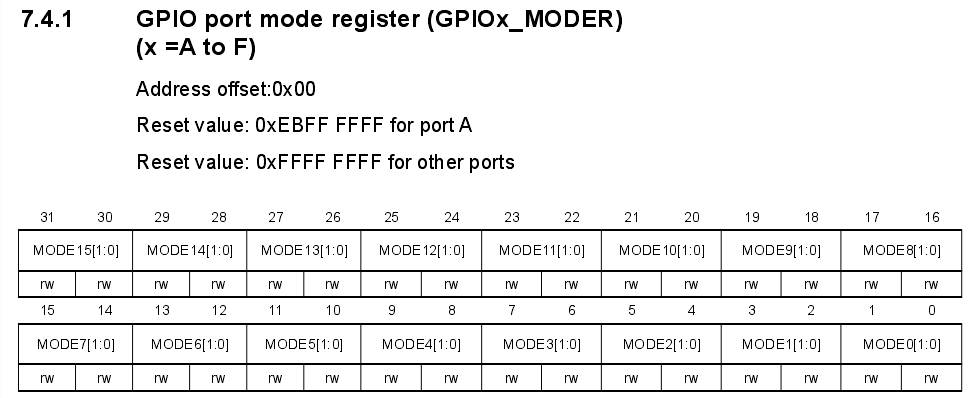
**EXTI基址0x4002 1800 IMR1偏移地址0x80 输入数据0xFFF8 2000**



**EXTI基址0x4002 1800 EMR1偏移地址0x84 输入数据0xFFF8 2000**



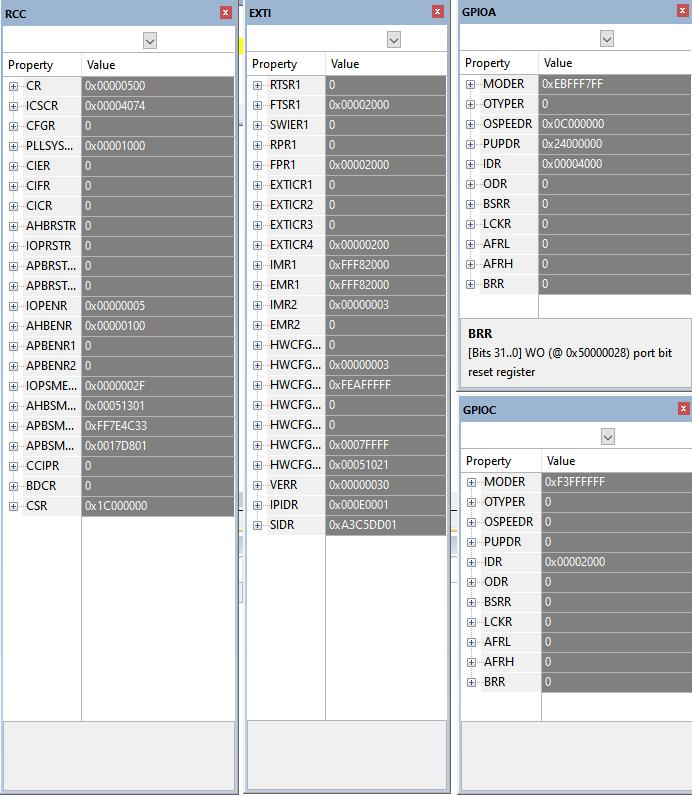
**EXTI基址0x4002 1800 FTSR1偏移地址0x04 输入数据0x0000 2000**



**GPIOA基址0x5000 0000 MODER偏移地址0x00 输入数据0xEBFF F7FF**

**第三步**：在启动引导代码中打断向C语言主函数main的跳转

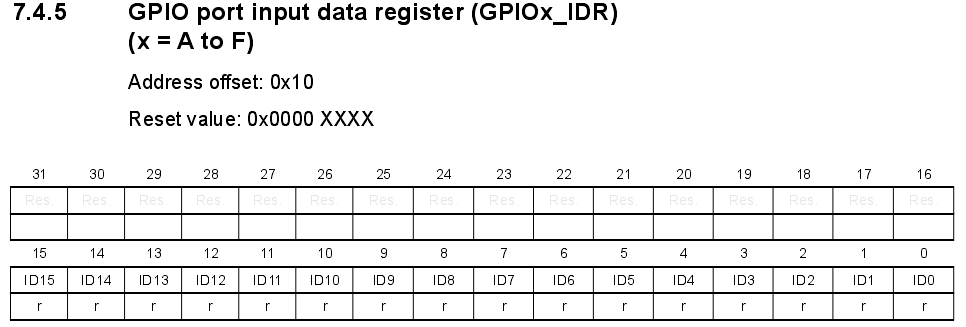
**第四步**：在启动引导代码下续写汇编语言实现上述寄存器的初始化



如上图所示，各个寄存器初始化之后的值均达到理想值。验证无误，可进行下一步。

**第五步**：在初始化之后添加设计代码。

注意到，当中断产生时，寄存器GPIOC.IDR的值为0x0000 0000；当中断恢复时，该寄存器值为0x0000 2000。故，可以读取该寄存器的值来判断中断按键是否按下。初步设计，按键按下LD4灯亮起。

已知GPOIC基址为0x5000 0800。且由上图可知，GPOIC.IDR寄存器偏移地址为0x10。

**第六步**：成果总结

实现了中断控制逻辑——按下即亮，松开即灭。代码如下：

MOVS R1, #0x40 ;RCC.IOPENR initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #12

LSLS R1, R3 ;Base address #4002 1000

MOVS R2, #0x34 ;Address offset #0x34

MOVS R0, #0x05 ;Set value #0x0000 0005

STR R0, [R1,R2]

MOVS R1, #0x50 ;GOIOC.MODER initialize

MOVS R3, #16

LSLS R1, R3

MOVS R3, #0x08

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3 ;Base address #5000 0800

MOVS R2, #0x00 ;Address offset #0x00

MOVS R0, #0xF3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3 ;Set value #0xF3FF FFFF

STR R0, [R1,R2]

MOVS R1, #0x40 ;EXTI.EXTICR4 initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3

MOVS R3, #0x80

ADDS R1, R3

MOVS R3, #4

LSLS R1, R3 ;Base address #4002 1800

MOVS R2, #0x6C ;Address offset #0x6C

MOVS R0, #0x02

MOVS R3, #8

LSLS R0, R3 ;Set value #0x0000 0200

STR R0, [R1,R2]

MOVS R1, #0x40 ;EXTI.IMR1 initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3

MOVS R3, #0x80

ADDS R1, R3

MOVS R3, #4

LSLS R1, R3 ;Base address #4002 1800

MOVS R2, #0x80 ;Address offset #0x80

MOVS R0, #0xFF

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xF8

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0x20

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3 ;Set value #0xFFF8 2000

STR R0, [R1,R2]

MOVS R1, #0x40 ;EXTI.EMR1 initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3

MOVS R3, #0x80

ADDS R1, R3

MOVS R3, #4

LSLS R1, R3 ;Base address #4002 1800

MOVS R2, #0x84 ;Address offset #0x84

MOVS R0, #0xFF

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xF8

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0x20

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3 ;Set value #0xFFF8 2000

STR R0, [R1,R2]

MOVS R1, #0x40 ;EXTI.FTSR1 initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3

MOVS R3, #0x80

ADDS R1, R3

MOVS R3, #4

LSLS R1, R3 ;Base address #4002 1800

MOVS R2, #0x04 ;Address offset #0x04

MOVS R0, #0x20

MOVS R3, #8

LSLS R0, R3 ;Set value #0x0000 2000

STR R0, [R1,R2]

MOVS R1, #0x00000050 ;GPIOA.MODER initialize

MOVS R3, #24

LSLS R1, R3 ;Base address #50000000

MOVS R2, #0x00 ;Address offset #0x00

MOVS R0, #0xEB

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xF7

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3 ;Set value #0xEBFFF7FF

STR R0, [R1,R2]

LEBEL ;Big cycle

MOVS R1, #0x50 ;read GOIOC.IDR

MOVS R3, #16

LSLS R1, R3

MOVS R3, #0x08

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3 ;Base address #5000 0800

MOVS R2, #0x10 ;Address offset #0x10

LDR R0, [R1,R2]

CMP R0, #0x00

BEQ LEBEL1 ;The interrupt

MOVS R1, #0x50 ;LD4 don't light

MOVS R3, #24

LSLS R1, R3

MOVS R2, #0x14

MOVS R0, #0x00

STR R0, [R1,R2]

B LEBEL

LEBEL1

MOVS R1, #0x50 ;LD4 light

MOVS R3, #24

LSLS R1, R3

MOVS R2, #0x14

MOVS R0, #0x20

STR R0, [R1,R2]

B LEBEL

**第七步**：代码优化，去掉冗余部分。

MOVS R1, #0x40 ;RCC.IOPENR initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #12

LSLS R1, R3 ;Base address #4002 1000

MOVS R2, #0x34 ;Address offset #0x34

MOVS R0, #0x05 ;Set value #0x0000 0005

STR R0, [R1,R2]

MOVS R1, #0x50 ;GOIOC.MODER initialize

MOVS R3, #16

LSLS R1, R3

MOVS R3, #0x08

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3 ;Base address #5000 0800

MOVS R2, #0x00 ;Address offset #0x00

MOVS R0, #0xF3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3 ;Set value #0xF3FF FFFF

STR R0, [R1,R2]

MOVS R1, #0x40 ;EXTI.EXTICR4 initialize

MOVS R3, #12

LSLS R1, R3

MOVS R3, #0x21

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3

MOVS R3, #0x80

ADDS R1, R3

MOVS R3, #4

LSLS R1, R3 ;Base address #4002 1800

MOVS R2, #0x6C ;Address offset #0x6C

MOVS R0, #0x02

MOVS R3, #8

LSLS R0, R3 ;Set value #0x0000 0200

STR R0, [R1,R2]

;EXTI.IMR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x80 ;Address offset #0x80

MOVS R0, #0xFF

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xF8

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0x20

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3 ;Set value #0xFFF8 2000

STR R0, [R1,R2]

;EXTI.EMR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x84 ;Address offset #0x84

;Set value #0xFFF8 2000,don't need to modify it

STR R0, [R1,R2]

;EXTI.FTSR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x04 ;Address offset #0x04

MOVS R0, #0x20

MOVS R3, #8

LSLS R0, R3 ;Set value #0x0000 2000

STR R0, [R1,R2]

MOVS R1, #0x00000050 ;GPIOA.MODER initialize

MOVS R3, #24

LSLS R1, R3 ;Base address #50000000

MOVS R2, #0x00 ;Address offset #0x00

MOVS R0, #0xEB

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xF7

ADDS R0, R3

MOVS R3, #8

LSLS R0, R3

MOVS R3, #0xFF

ADDS R0, R3 ;Set value #0xEBFFF7FF

STR R0, [R1,R2]

LEBEL ;Big cycle

MOVS R1, #0x50 ;read GOIOC.IDR

MOVS R3, #16

LSLS R1, R3

MOVS R3, #0x08

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3 ;Base address #5000 0800

MOVS R2, #0x10 ;Address offset #0x10

LDR R0, [R1,R2]

CMP R0, #0x00

BEQ LEBEL1 ;The interrupt

MOVS R1, #0x50 ;LD4 don't light

MOVS R3, #24

LSLS R1, R3 ;Base address #5000 0000

MOVS R2, #0x14 ;Address offset #0x14

MOVS R0, #0x00

STR R0, [R1,R2]

B LEBEL

LEBEL1

MOVS R1, #0x50 ;LD4 light

MOVS R3, #24

LSLS R1, R3 ;Base address #5000 0000

MOVS R2, #0x14 ;Address offset #0x14

MOVS R0, #0x20

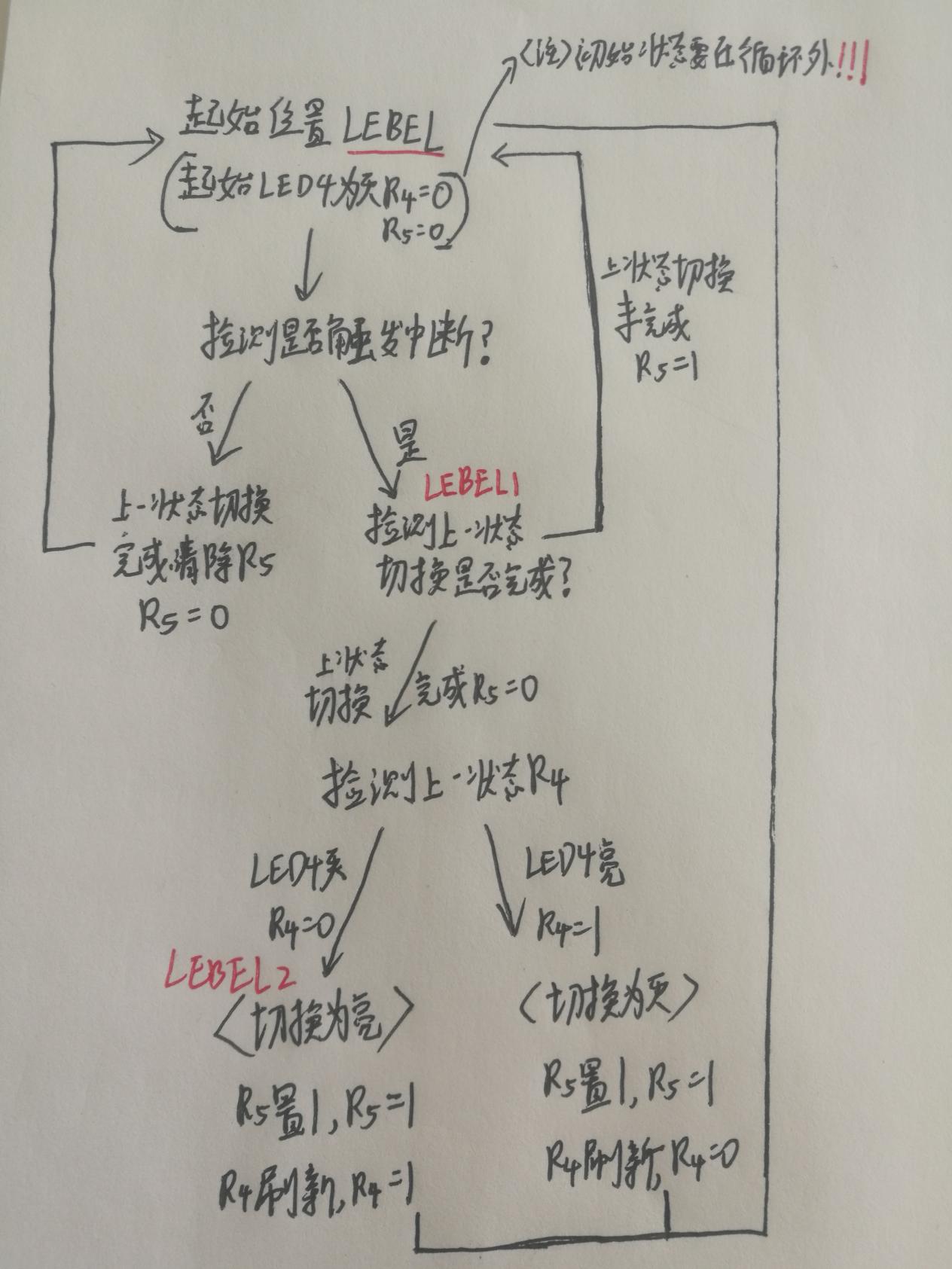
STR R0, [R1,R2]

B LEBEL

**第八步**：成果拓展：实现中断控制逻辑——按一次切换一种状态。

补充说明:寄存器R1存储寻址所需的基址。寄存器R2存储寻址所需的偏移地址。寄存器R0存储要往存储器中存入的数据或者从存储器中取出的数据。寄存器R3存储辅助数据，辅助移位，累加等。寄存器R4存储亮还是灭（0灭1亮，寄存器R5存储上一次状态切换是否完成（避免长时间按下中断状态一直切换）。

思路：寄存器初始化不变。具体的中断控制逻辑部分设计思路如下图所示：



中断控制逻辑部分代码如下：

MOVS R4, #0x00 ;LD4 state don't light

MOVS R5, #0x00 ;Set LD4 accomplish

LEBEL ;Big cycle

MOVS R1, #0x50 ;read GOIOC.IDR

MOVS R3, #16

LSLS R1, R3

MOVS R3, #0x08

ADDS R1, R3

MOVS R3, #8

LSLS R1, R3 ;Base address #5000 0800

MOVS R2, #0x10 ;Address offset #0x10

LDR R0, [R1,R2]

CMP R0, #0x00

BEQ LEBEL1 ;The interrupt

MOVS R5, #0x00

B LEBEL

LEBEL1

CMP R5, #0x01

BEQ LEBEL ;Set LD4 not accomplish

CMP R4, #0x00

BEQ LEBEL2 ;LD4 last state is don't light

MOVS R1, #0x50 ;SET LD4 don't light

MOVS R3, #24

LSLS R1, R3 ;Base address #5000 0000

MOVS R2, #0x14 ;Address offset #0x14

MOVS R0, #0x00

STR R0, [R1,R2]

MOVS R4, #0x00 ;LD4 state don't light

MOVS R5, #0x01 ;Set LD4 not accomplish

B LEBEL

LEBEL2

MOVS R1, #0x50 ;Set LD4 light

MOVS R3, #24

LSLS R1, R3 ;Base address #5000 0000

MOVS R2, #0x14 ;Address offset #0x14

MOVS R0, #0x20

STR R0, [R1,R2]

MOVS R4, #0x01 ;LD4 state don't light

MOVS R5, #0x01 ;Set LD4 not accomplish

B LEBEL

**第九步**：反思总结

以上实现的终究是轮循和GPIO的输入输出不是真正的中断。

用C语言实现中断，但是汇编没能实现问题原因分析：

可能原因一：汇编中我关于中断写的内容都不对。

可能原因二：汇编初始化漏掉了一些相关设置。

以下实验方法：①用调试器查看相关寄存器的值，②观察其编译下载之后的情况，③添加断点并查看中断标志位的变化。

实验一：在EXTI4\_15\_IRQHandler函数入口处添加中断。当按下中断按键，中断标志位立即置一。执行完EXTI\_GetITStatus函数判断中断来源并不影响标志位。当执行完函数EXTI\_ClearITPendingBit中的EXTI->FPR1 = EXTI\_Line时释放中断标志位。当执行完函数HAL\_GPIO\_TogglePin时，LD4的显示状态反转。【**符合预期**】结果：找到了中断标志位寄存器，及其运作方式。

实验二：将EXTI\_GetITStatus与EXTI\_ClearITPendingBit函数全部注释掉。编译下载之后的情况：初始状态为灭，当按下中断按键LD4变亮但是亮度不足。[推测其不停执行中断所以不断反转亮度仅一半]断点调试情况：当中断按键按下后FPR1对应位始终被拉高不能恢复，不断进入中断。【**符合预期**】结果：引申——如果汇编程序的中断服务程序不对该标志位做任何操作，它因该是处于此状态。

实验三：只注释掉EXTI\_GetITStatus函数。编译下载之后的情况：所有程序正常执行[推测只是不判断中断的来源了，例如4-15号中断都可以触发它]。断点调试情况：FPR1正常被拉高，正常复位。【**符合预期**】结果：引申——如果汇编程序的中断服务程序释放标志位但不判断其来源，它因该是处于此状态。

实验四：注释掉C语言中的的中断服务程序入口函数EXTI4\_15\_IRQHandler，相当于只是成功初始化了中断相关寄存器，并没有对中断做任何操作。编译下载之后的情况：按下中断按键无任何反应。断点调试情况：按下中断按键后，对应的中断标志位依旧被拉高不恢复。【**符合预期**】结果：汇编程序出错很大可能是中断送不进去。

实验五：注释掉C语言中的的中断服务程序入口函数EXTI4\_15\_IRQHandler，并改写启动引导代码中关于中断的部分。编译下载之后的情况：按下中断按键后，LD4灯的状态从灭转为50%亮度。【**符合预期**】结果：汇编程序对中断的相关寄存器初始化有问题，中断送不进去。

实验六：改写启动引导代码中中断向量表下的中断服务程序，注其中断标志位寄存器是写一清零。实现按下中断按键反转，并能跳出中断的过程。

最终解决：中断管脚PC13必须设置GPIOC的工作模式！！！

**第十步**：最终代码

Stack\_Size EQU 0x400 ;EQR相当于C中的define

AREA STACK, NOINIT, READWRITE, ALIGN=3 ;AREA汇编一个新的代码段数据段

Stack\_Mem SPACE Stack\_Size ;SPACE分配内存空间

\_\_initial\_sp

; <h> Heap Configuration

; <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>

; </h>

Heap\_Size EQU 0x200

AREA HEAP, NOINIT, READWRITE, ALIGN=3

\_\_heap\_base

Heap\_Mem SPACE Heap\_Size

\_\_heap\_limit

PRESERVE8 ;当前文件堆栈需按照8字节对齐

THUMB

; Vector Table Mapped to Address 0 at Reset

AREA RESET, DATA, READONLY

EXPORT \_\_Vectors ;EXPORT声明一个标号具有全局性，可被外部文件引用

EXPORT \_\_Vectors\_End

EXPORT \_\_Vectors\_Size

\_\_Vectors DCD \_\_initial\_sp ; Top of Stack DCD以字为单位分配内存

DCD Reset\_Handler ; Reset Handler

DCD NMI\_Handler ; NMI Handler

DCD HardFault\_Handler ; Hard Fault Handler

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD SVC\_Handler ; SVCall Handler

DCD 0 ; Reserved

DCD 0 ; Reserved

DCD PendSV\_Handler ; PendSV Handler

DCD SysTick\_Handler ; SysTick Handler

; External Interrupts

DCD WWDG\_IRQHandler ; Window Watchdog

DCD PVD\_IRQHandler ; PVD through EXTI Line detect

DCD RTC\_TAMP\_IRQHandler ; RTC through EXTI Line

DCD FLASH\_IRQHandler ; FLASH

DCD RCC\_IRQHandler ; RCC

DCD EXTI0\_1\_IRQHandler ; EXTI Line 0 and 1

DCD EXTI2\_3\_IRQHandler ; EXTI Line 2 and 3

DCD EXTI4\_15\_IRQHandler ; EXTI Line 4 to 15 中断向量表定义中断

\_\_Vectors\_End

\_\_Vectors\_Size EQU \_\_Vectors\_End - \_\_Vectors

AREA |.text|, CODE, READONLY

; Reset handler routine

Reset\_Handler PROC

EXPORT Reset\_Handler [WEAK]

;RCC.IOPENR initialize

LDR R1, =0x40021000;Base address #4002 1000

MOVS R2, #0x34 ;Address offset #0x34

MOVS R0, #0x05 ;Set value #0x0000 0005

STR R0, [R1,R2]

;GOIOC.MODER initialize

LDR R1, =0x50000800;Base address #5000 0800

MOVS R2, #0x00 ;Address offset #0x00

LDR R0, =0xF3FFFFFF;Set value #0xF3FF FFFF

STR R0, [R1,R2]

;EXTI.EXTICR4 initialize

LDR R1, =0x40021800;Base address #4002 1800

MOVS R2, #0x6C ;Address offset #0x6C

LDR R0, =0x00000200;Set value #0x0000 0200

STR R0, [R1,R2]

;EXTI.IMR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x80 ;Address offset #0x80

LDR R0, =0xFFF82000;Set value #0xFFF8 2000

STR R0, [R1,R2]

;EXTI.EMR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x84 ;Address offset #0x84

;Set value #0xFFF8 2000,don't need to modify it

STR R0, [R1,R2]

;EXTI.FTSR1 initialize

;Base address #4002 1800,don't need to modify it

MOVS R2, #0x04 ;Address offset #0x04

LDR R0, =0x00002000;Set value #0x0000 2000

STR R0, [R1,R2]

;GPIOA.MODER initialize

LDR R1, =0x50000000 ;Base address #50000000

MOVS R2, #0x00 ;Address offset #0x00

LDR R0, =0xEBFFF7FF;Set value #0xEBFFF7FF

STR R0, [R1,R2]

;NVIC.ISER initialize

LDR R1, =0xE000E100

MOVS R2, #0x00

MOVS R0, #0x80

STR R0, [R1,R2]

LDR R1, =0x50000000 ;light

MOVS R2, #0x14

MOVS R0, #0x20

STR R0,[R1,R2]

LEBEL5

B LEBEL5

ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI\_Handler PROC

EXPORT NMI\_Handler [WEAK]

B .

ENDP

HardFault\_Handler\

PROC

EXPORT HardFault\_Handler [WEAK]

B .

ENDP

SVC\_Handler PROC

EXPORT SVC\_Handler [WEAK]

B .

ENDP

PendSV\_Handler PROC

EXPORT PendSV\_Handler [WEAK]

B .

ENDP

SysTick\_Handler PROC

EXPORT SysTick\_Handler [WEAK]

B .

ENDP

WWDG\_IRQHandler PROC

EXPORT WWDG\_IRQHandler [WEAK]

B .

ENDP

PVD\_IRQHandler PROC

EXPORT PVD\_IRQHandler [WEAK]

B .

ENDP

RTC\_TAMP\_IRQHandler PROC

EXPORT RTC\_TAMP\_IRQHandler [WEAK]

B .

ENDP

FLASH\_IRQHandler PROC

EXPORT FLASH\_IRQHandler [WEAK]

B .

ENDP

RCC\_IRQHandler PROC

EXPORT RCC\_IRQHandler [WEAK]

B .

ENDP

EXTI0\_1\_IRQHandler PROC

EXPORT EXTI0\_1\_IRQHandler [WEAK]

B .

ENDP

EXTI2\_3\_IRQHandler PROC

EXPORT EXTI2\_3\_IRQHandler [WEAK]

B .

ENDP

EXTI4\_15\_IRQHandler PROC

EXPORT EXTI4\_15\_IRQHandler [WEAK]

PUSH {LR}

;清除中断标志位————写一清零

LDR R1, =0x40021800;Base address #4002 1800

MOVS R2, #0x010 ;Address offset #0x6C

LDR R0, =0x00002000;Set value #0x0000 2000

STR R0, [R1,R2]

LDR R1, =0x50000000

MOVS R2, #0x14

LDR R0, [R1,R2] ;GPIOA.ODR

CMP R0, #0x20

BEQ LEBEL1 ;如果亮跳转到LEBEL1

MOVS R0, #0x20 ;不亮则变亮

STR R0, [R1,R2]

B LEBEL ;跳出中断

LEBEL1

MOVS R0,#0x00 ;变灭

STR R0, [R1,R2]

LEBEL

POP {PC}

ENDP

ALIGN ;ALIGN四字节对齐

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; User Stack and Heap initialization

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF :DEF:\_\_MICROLIB

EXPORT \_\_initial\_sp

EXPORT \_\_heap\_base

EXPORT \_\_heap\_limit

ELSE

IMPORT \_\_use\_two\_region\_memory

EXPORT \_\_user\_initial\_stackheap

\_\_user\_initial\_stackheap

LDR R0, = Heap\_Mem

LDR R1, =(Stack\_Mem + Stack\_Size)

LDR R2, = (Heap\_Mem + Heap\_Size)

LDR R3, = Stack\_Mem

BX LR

ALIGN

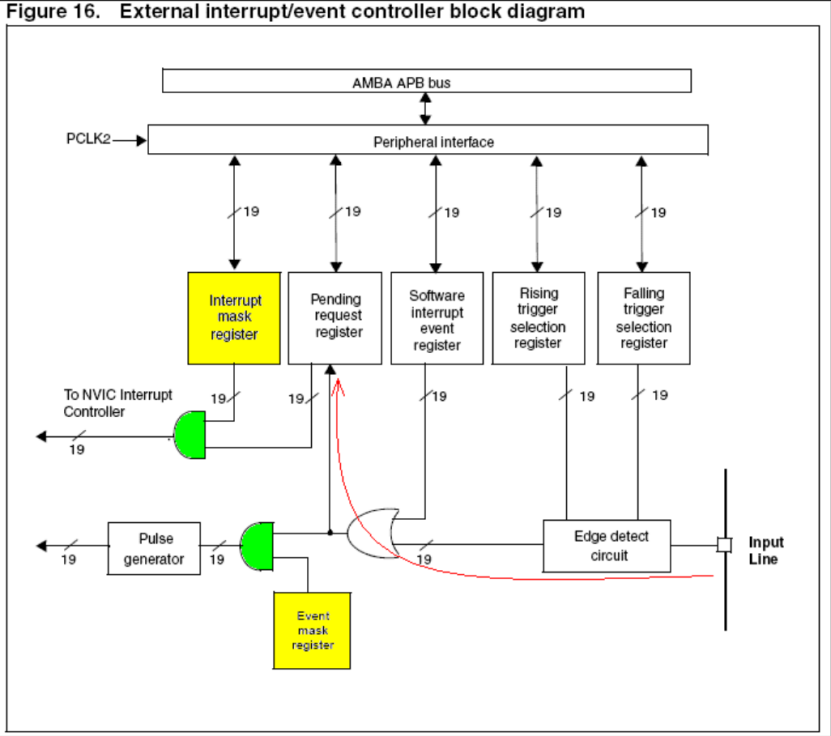
ENDIF

END

**第十一步**：关于中断的其他理解

中断标志位寄存器，不能通过手动将其置一来强行触发中断。其清除的时候还是写一清零，即给对应的标志位赋值一就会清零。

中断和事件，中断屏蔽寄存器与事件屏蔽寄存器之间的关系，如下图：



Cortex\_M0中断使能寄存器、中断屏蔽寄存器和事件屏蔽寄存器设置：中断使能的时候，寄存器各位对应的是中断编号，比如外部中断0和1是同一个编号、2和3也是同一个编号、4到15都是同一个编号。在STM32CubeMX的图形化配置界面，中断使能4到15号都是同一个√。中断屏蔽寄存器和事件屏蔽寄存器：两个寄存器的相对位一一对应，且没有共用，比如第13位对应的就是外部13号中断。

**ARM不同于X86，其内部配置的NVIC中断管理器自动入栈出栈，用汇编写中断服务程序时不需要再单独编写入栈出栈过程。**

**汇编语言伪指令LDR可以给寄存器直接赋值32位立即数，没有位数限制。可完美代替用MOVS、LSLS和ADD组合的基本指令，降低程序编写的复杂度。（编译之后没区别）**