Principles and Practices of Microcontroller (Embedded System Design I) -STM32 Processor

Gang Chen (陈刚)

Associate Professor

Institute of Unmanned Systems

School of data and computer science

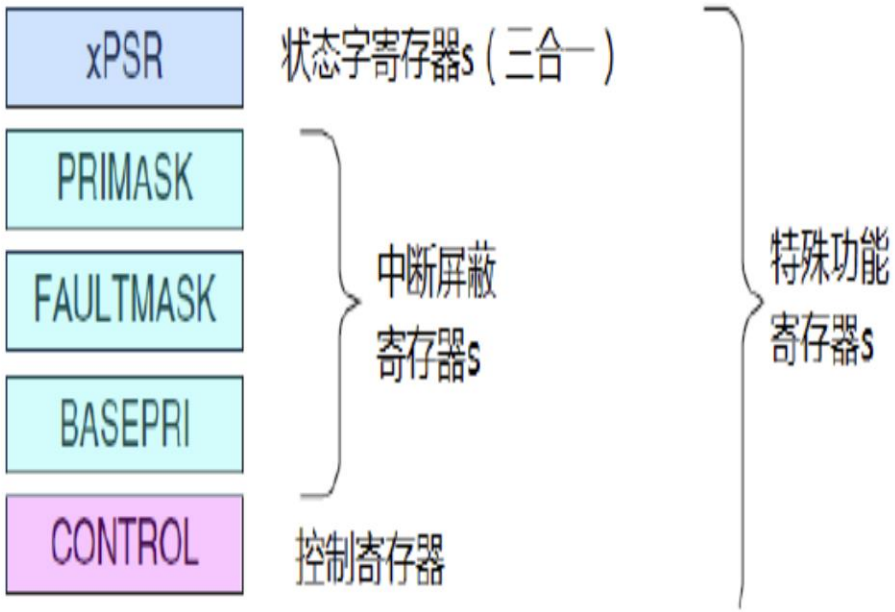
Sun Yat-Sen University

https://www.usilab.cn/team/chengang/

Cortex – M3 特性

- 特殊功能寄存器
- 复位序列
- 中断咬尾
- 晚到异常
- 位带操作
- 互斥访问

特殊功能寄存器



CM3特殊寄存器

xPSR: 记录ALU标志（0标志，进位标志，负数标志，溢出标志），执行状态，以及当前正服务的中断号。

PRIMASK： 除能所有中断，不可屏蔽NMI。

FAULTMASK：除能所有的fault, NMI依然不受影响，而且被除能的faults会“上访”。

BASEPRI： 除能所有优先级不高于某个具体数值的中断。

CONTROL：定义特权状态

PRIMASK, FAULTMASK 和BASEPRI 寄存器

寄存器名	描述
PRIMASK	一个1-bit 寄存器。当置位时，它允许NMI 和硬件默认异常；所有其他的中断和异常将被屏蔽。
FAULTMASK	一个1-bit 寄存器。当置位时，它只允许NMI，所有中断和默认异常处理被忽略。
BASEPRI	一个9位寄存器。它定义了屏蔽优先级。 当它置位时，所有同级的或低级的中断被忽略。

Cortex-M3 中断屏蔽寄存器

当访问PRIMASK, FAULTMASK, 和BASEPRI 寄存器时, MRS 和MSR指令被使用

例子:

```
MRS r0, BASEPRI ; Read BASEPRI register into R0
MRS r0, PRIMASK ; Read PRIMASK register into R0
MRS r0, FAULTMASK ; Read FAULTMASK register into R0
MSR BASEPRI, r0 ; Write R0 into BASEPRI register
MSR PRIMASK, r0 ; Write R0 into PRIMASK register
MSR FAULTMASK, r0 ; Write R0 into FAULTMASK register
```

注：在用户访问级，PRIMASK, FAULTMASK, 和BASEPRI 寄存器不能被置位。

3

控制寄存器

控制寄存器被用来定义特权级和堆栈指针的选择，该寄存器有两位。

Bit	Function
CONTROL[1]	堆栈状态:(当访问级别改变时自动改变) 1 = 进程堆栈(PSP) 被使用 (针对用户级) 0 = 默认堆栈 (MSP) 被使用(针对特权级)
CONTROL[0]	指定的访问级别: 0 = 特权的线程模式 1 = 用户状态的线程模式

Cortex-M3 控制寄存器

在CM3中, 在处理模式中CONTROL[1] 位总是0 (MSP)。但是, 在线程或基本级别，它可以为0或1。CONTRL[0] 位只在特权状态可写。

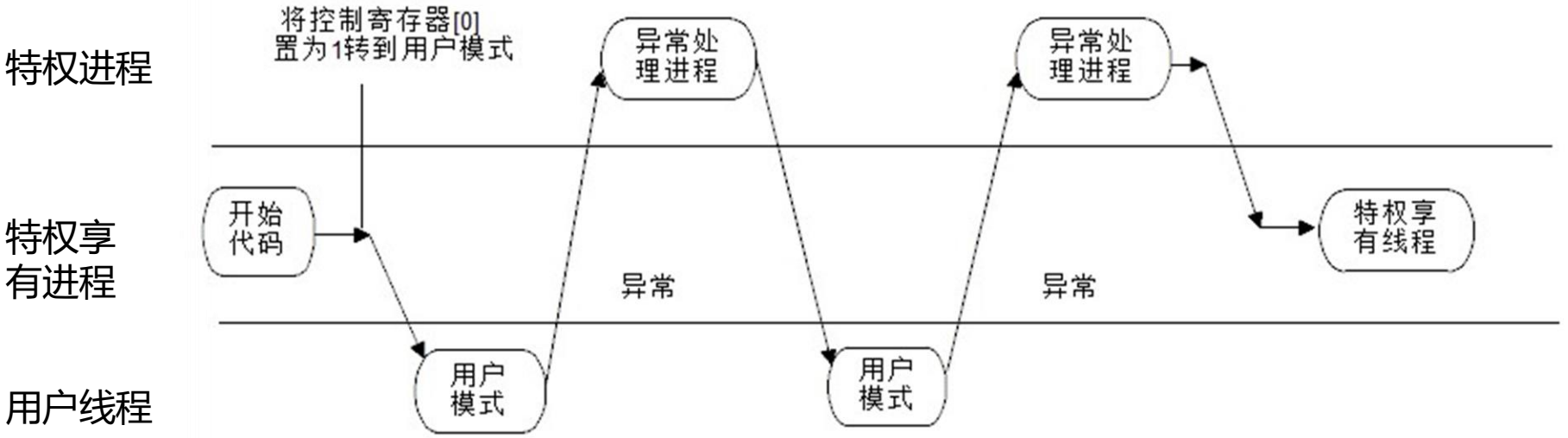
操作模式

CM3有两种模式和两种权限级别：

	特权	用户
当运行一个异常	处理器模式	
当运行主程序	线程模式	线程模式

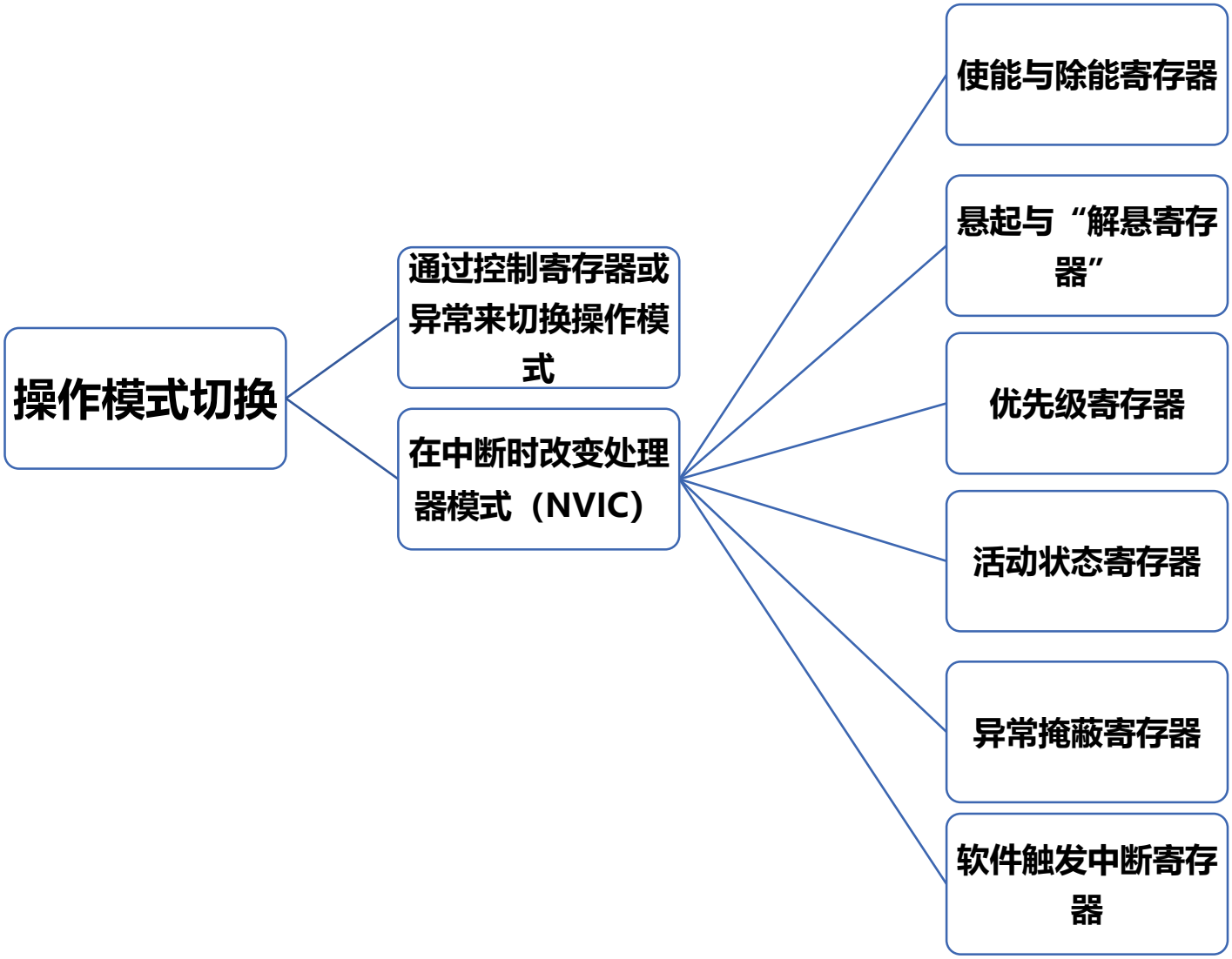
Cortex-M3 控制模式和特权关系表

1. 操作模式决定处理器运行正常程序或运行异常处理程序。

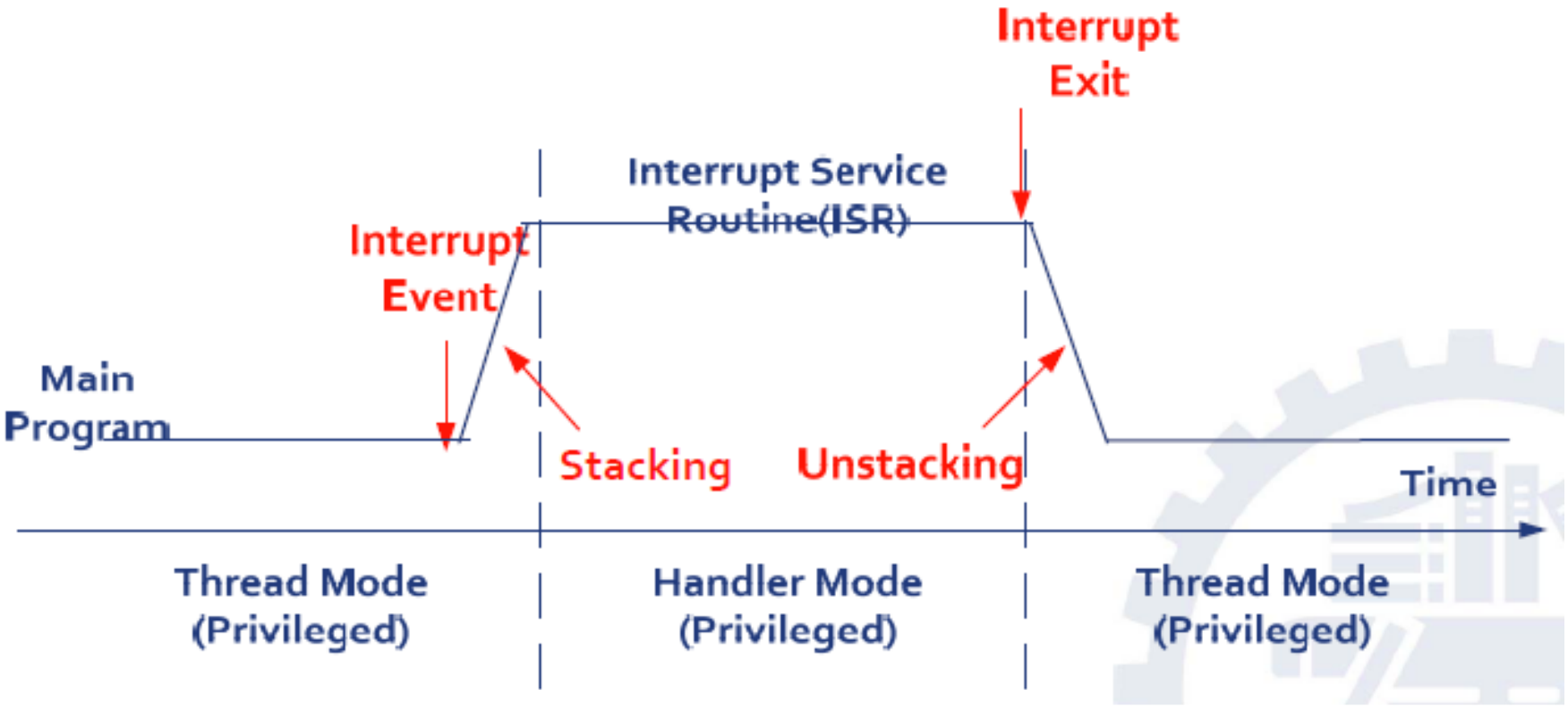


模式与特权转化图

1. 通过写Control register[0]=1，软件在特权访问级别可以使程序转换到用户访问级别。
2. 用户程序不能够通过写控制寄存器直接变回特权状态，它要经过一个异常处理程序设置Control register[0]=0使得处理器切换回特权访问级别。
3. ．特权级别提供了一种机制来保障访问存储器的关键区域，同时还提供了一个基本的安全模式。

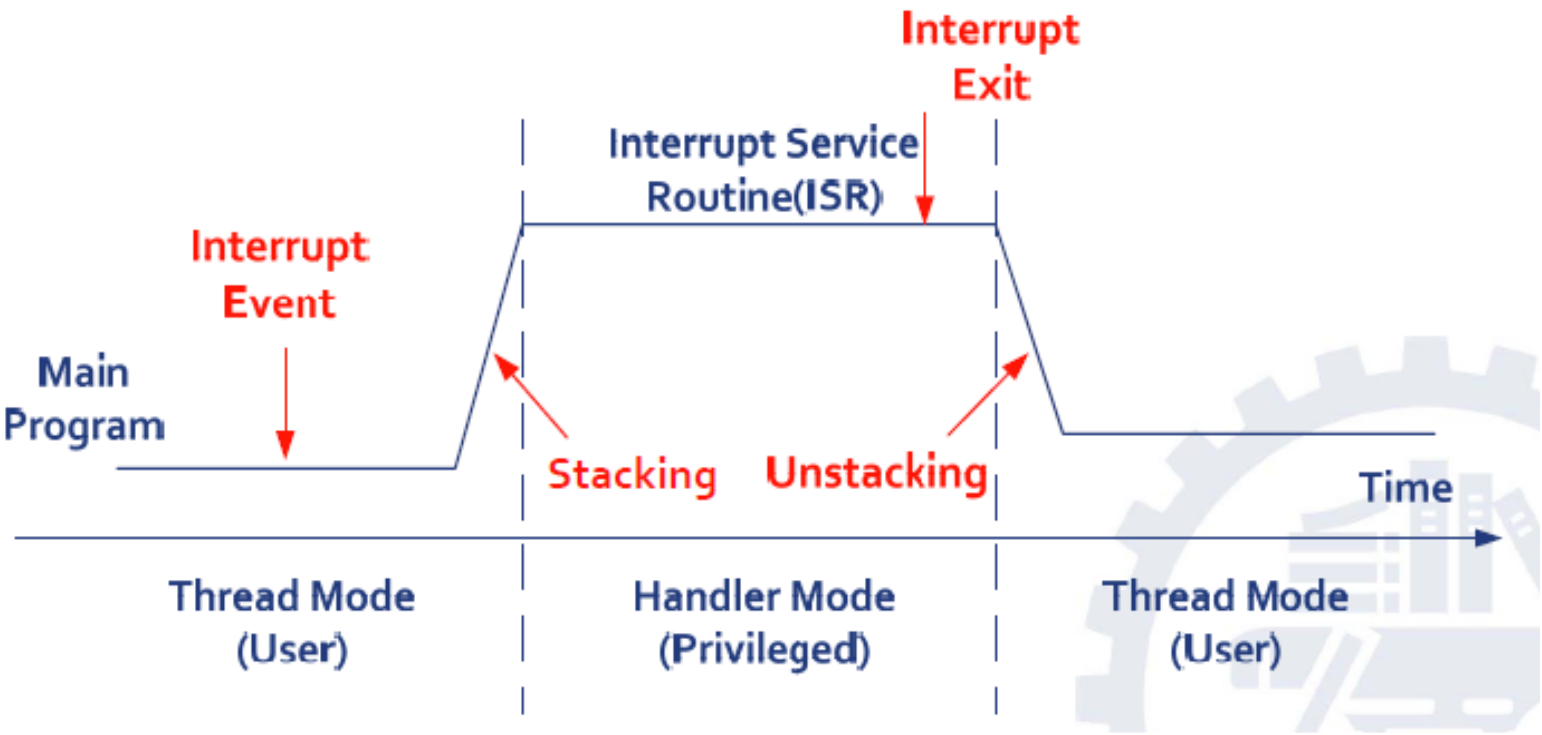

操作模式切换
通过控制寄存器或异常来切换操作模式
在中断时改变处理器模式（NVIC)
使能与除能寄存器
悬起与“解悬寄存器”
优先级寄存器
活动状态寄存器
异常掩蔽寄存器
软件触发中断寄存器

通过控制寄存器或异常来切换操作模式



由控制寄存器来定义处理器的模式和访问级别。当Control register[0] =0，异常发生时只有处理器的模式发生了变化。访问级别始终停留在特权状态。

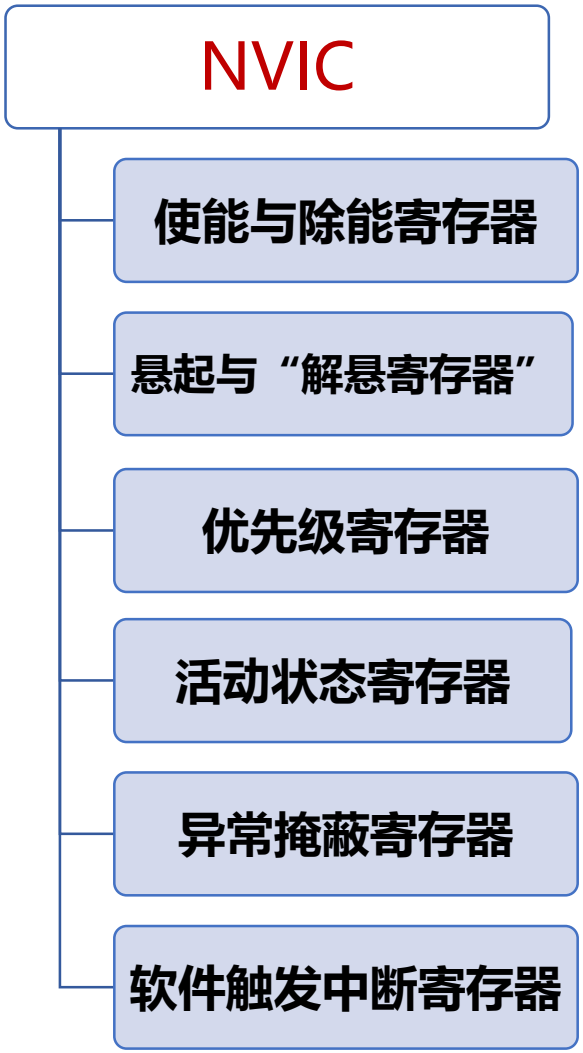
在中断时改变处理器模式



在中断时改变处理器模式

对于用户级别程序转换到特权状态, 需要在处理程序产生一个中断(例如, SVC, 或呼叫系统服务) 和写CONTROL[0]=0。

9



向量中断控制器（NVIC）：NVIC的寄存器以存储器映射的方式来访问，除了包含控制寄存器和中断处理的控制逻辑之外，NVIC 还包含了MPU的控制寄存器、SysTick 定时器以及调试控制。所有NVIC 的中断控制/状态寄存器都只能在特权级下访问。不过有一个例外——软件触发中断寄存器可以在用户级下访问以产生软件中断。所有的中断控制 / 状态寄存器均可按字 / 半字 / 字节的方式访问。

使能与除能寄存器

名称	类型	地址	复位值	描述
SETENA0	R/W	0xE000_E100	0	中断 0-31 的使能寄存器，共 32 个使能位 位[n]，中断#n 使能（异常号 16+n）
SETENA1	R/W	0xE000_E104	0	中断 32-63 的使能寄存器，共 32 个使能位
…	…	…	…	…
SETENA7	R/W	0xE000_E11C	0	中断 224-239 的使能寄存器，共 16 个使能位
CLRENA0	R/W	0xE000_E180	0	中断 0-31 的除能寄存器，共 32 个除能位 位[n]，中断#n 除能（异常号 16+n）
CLRENA1	R/W	0xE000_E184	0	中断 32-63 的除能寄存器，共 32 个除能位
…	…	…	…	…
CLRENA7	R/W	0xE000_E19C	0	中断 224-239 的除能寄存器，共 16 个除能位

SETENA与CLRENA寄存器说明

240对

使能中断，写1到SETENA

除能中断，写1到CLRENA

悬起与“解悬”寄存器

如果中断发生时，正在处理同级或高优先级异常，或者被掩蔽，则中断不能立即得到响应，此时中断被悬起。

名称	类型	地址	复位值	描述
SETPEND0	R/W	0xE000_E200	0	中断 0-31 的悬起寄存器，共 32 个悬起位 位[n]，中断#n 悬起（异常号 16+n）
SETPEND1	R/W	0xE000_E204	0	中断 32-63 的悬起寄存器，共 32 个悬起位
…	…	…	…	…
SETPEND7	R/W	0xE000_E21C	0	中断 224-239 的悬起寄存器，共 16 个悬起位
CLRPEND0	R/W	0xE000_E280	0	中断 0-31 的解悬寄存器，共 32 个解悬位 位[n]，中断#n 解悬（异常号 16+n）
CLRPEND1	R/W	0xE000_E284	0	中断 32-63 的解悬寄存器，共 32 个解悬位
…	…	…	…	…
CLRPEND7	R/W	0xE000_E29C	0	中断 224-239 的解悬寄存器，共 16 个解悬位

SETPEND与CLRPEND寄存器说明

优先级寄存器

名称	类型	地址	复位值	描述
PRI_0	R/W	0xE000_E400	0（8 位）	外中断#0 的优先级
PRI_1	R/W	0xE000_E401	0（8 位）	外中断#1 的优先级
…	…	…	…	…
PRI_239	R/W	0xE000_E4EF	0（8 位）	外中断#239 的优先级

中断优先级寄存器

地址	名称	类型	复位值	描述
0xE000_ED18	PRI_4			存储器管理 fault 的优先级
0xE000_ED19	PRI_5			总线 fault 的优先级
0xE000_ED1A	PRI_6			用法 fault 的优先级
0xE000_ED1B	-	-	-	-
0xE000_ED1C	-	-	-	-
0xE000_ED1D	-	-	-	-
0xE000_ED1E	-	-	-	-
0xE000_ED1F	PRI_11			SVC 优先级
0xE000_ED20	PRI_12			调试监视器的优先级
0xE000_ED21	-	-	-	-
0xE000_ED22	PRI_14			PendSV 的优先级
0xE000_ED23	PRI_15			SysTick 的优先级

系统异常优先级寄存器

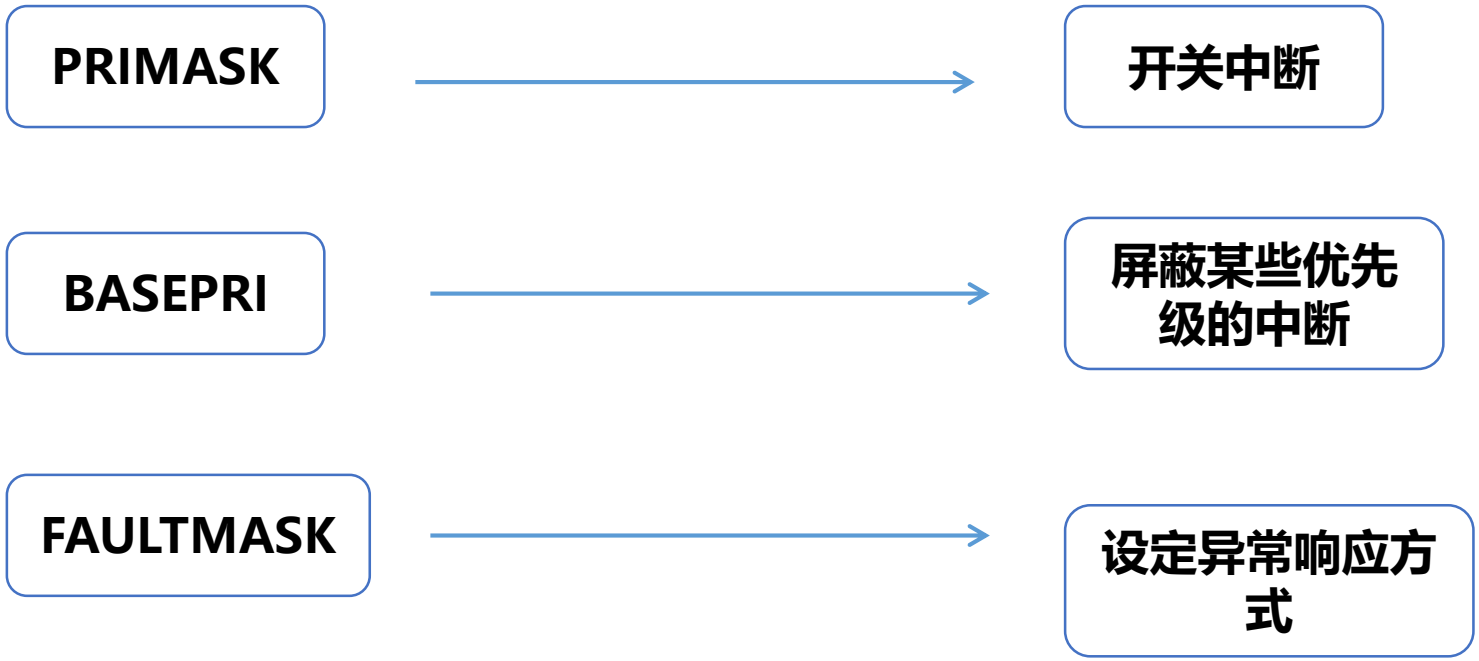
活动状态寄存器

ACTIVE寄存器说明

名称	类型	地址	复位值	描述
ACTIVE0	RO	0xE000_E300	0	中断 0-31 的活动状态寄存器，共 32 个状态位 位[n]，中断#n 活动状态（异常号 16+n）
ACTIVE1	RO	0xE000_E304	0	中断 32-63 的活动状态寄存器，共 32 个状态位
…	…	…	…	…
ACTIVE7	RO	0xE000_E31C	0	中断 224-239 的活动状态寄存器，共 16 个状态位

每个外部中断都有一个活动状态位。在处理器执行了其ISR 的第一条指令后，它的活动位就被置1，并且直到ISR 返回时才硬件清零。

异常掩蔽寄存器



PRIMASK使用例子

例： 关中断
MOV R0, #1
MSR PRIMASK, R0
例： 开中断
MOV R0, #0
MSR PRIMASK, R0

或

通过CPS指令快速完成上述功能：
CPSID i： 关中断
CPSIE i： 开中断

BASEPRI用法

- 如需要掩蔽所有优先级不高于0x60的中断：
- MOV R0, #0x60
- MSR BASEPRI, R0

- 如果需要取消BASEPRI 对中断的掩蔽：
- MOV R0, #0
- MSR BASEPRI, R0

- 注：也可使用BASEPRI_MAX 来访问BASEPRI

FAULTMASK用法

- Fault可以捕获非法内存方法和非法编程行为，Fault异常能够检测到以下情况：
  - 总线Fault（BUSFault)
  - 存储器管理Fault（MEMFault)
  - 用法Fault（USGFault)
  - 硬Fault

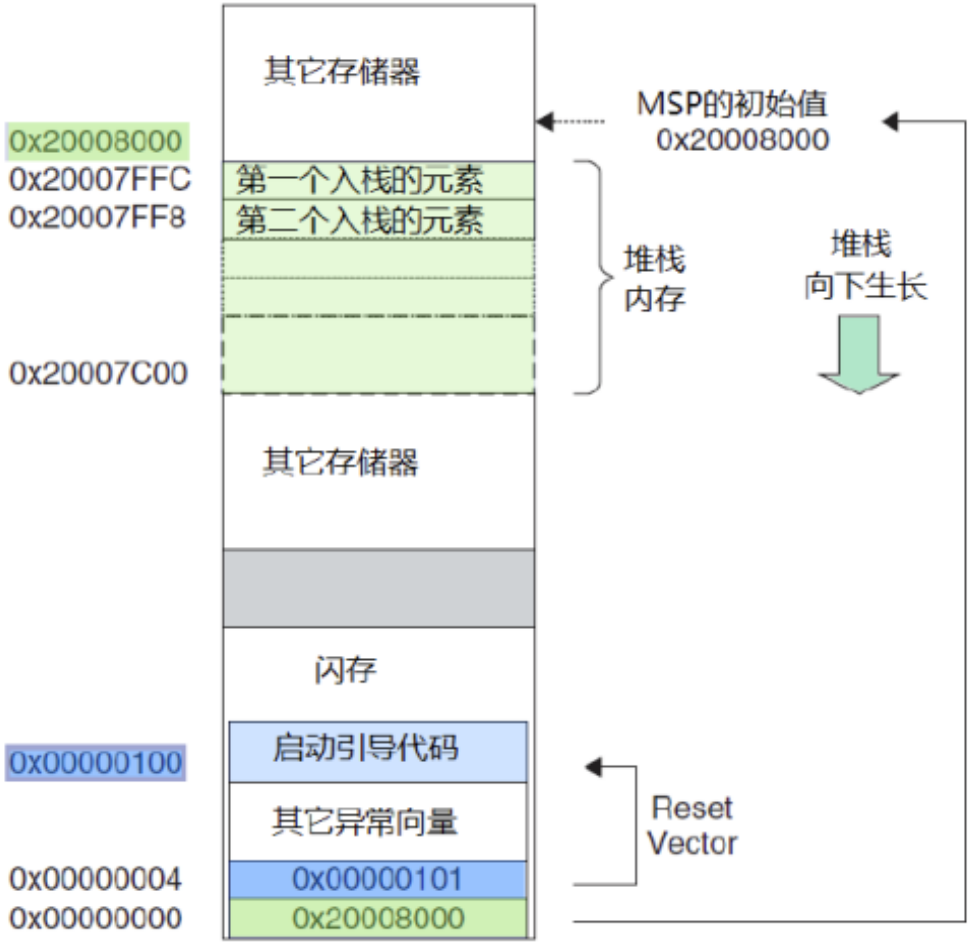
软件触发中断寄存器

软件触发中断寄存器STIR（地址：0xE000_EF00）

位段	名称	类型	复位值	描述
8:0	INTID	W	-	影响编号为 INTID 的外部中断，其悬起位被置位。 例如，写入 8，则悬起 IRQ #8

软件中断，包括手工产生的普通中断，能以多种方式产生。最简单的就是使用相应的SETPEND寄存器；而更专业更快捷的作法，则是通过使用软件触发中断寄存器STIR。

复位序列

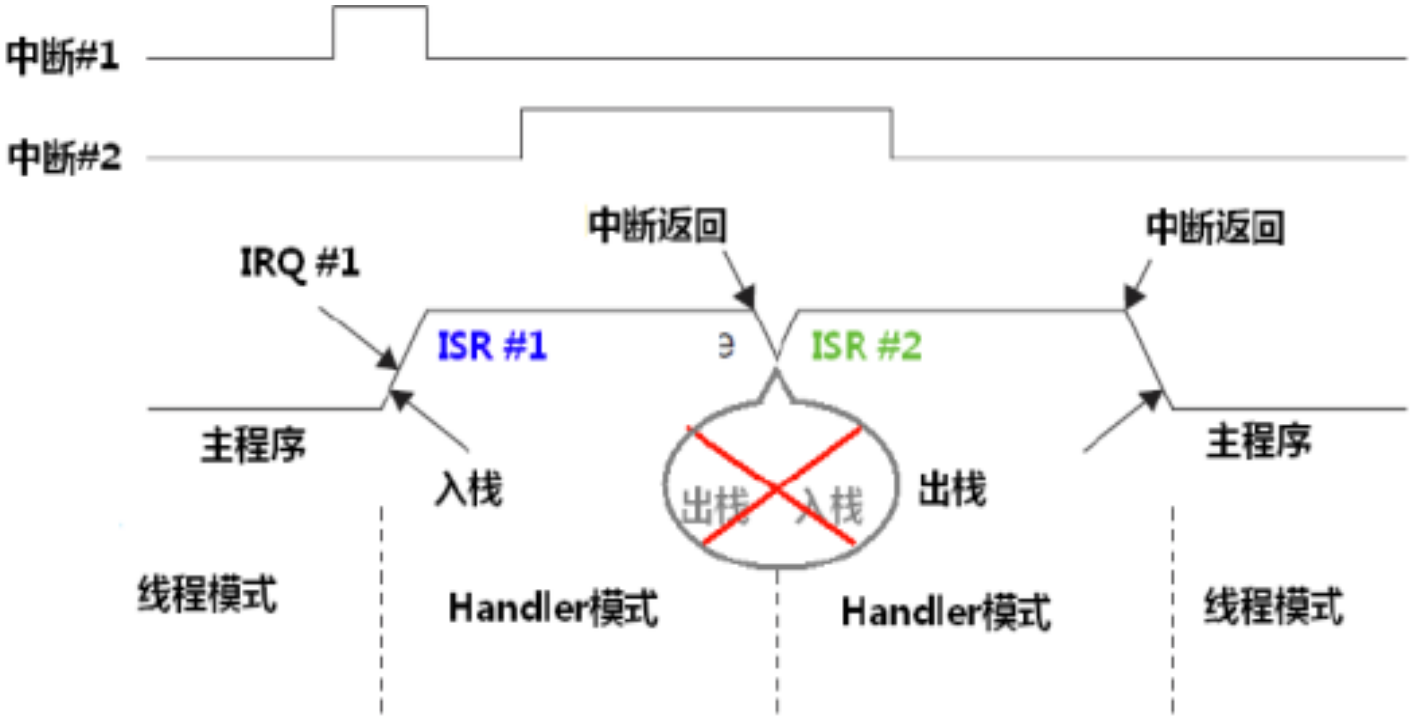


初始MSP及PC初始化的一个范例

1. 从地址 0x0000,0000 处取出MSP 的初始值。
2. 从地址 0x0000,0004 处取出PC 的初始值——这个值是复位向量，LSB 必须是1，然后从这个值所对应的地址处取指。

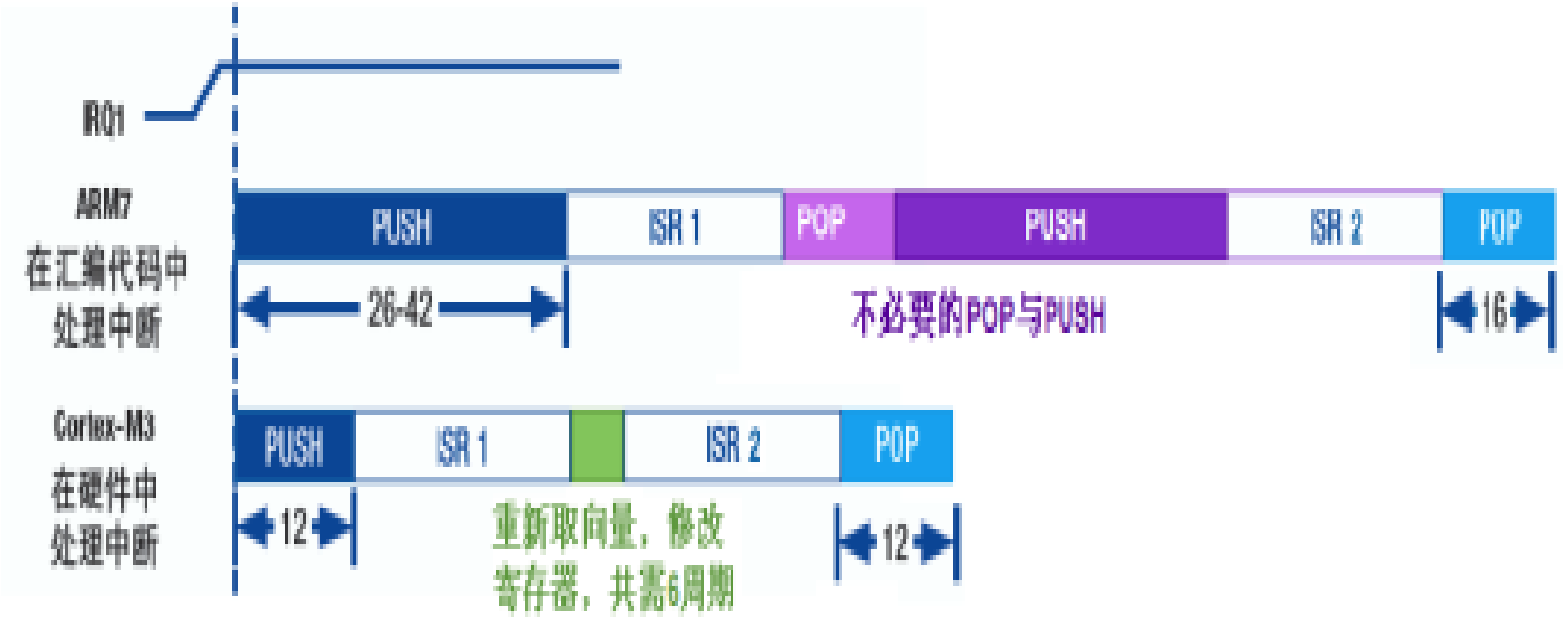
中断咬尾

当处理器在响应某异常时，如果又发生其它异常，但它们优先级不够高，则被阻塞。CM3不会POP这些寄存器，而是继续使用上一个异常已经PUSH好的成果，从而降低了频繁的进出栈消耗



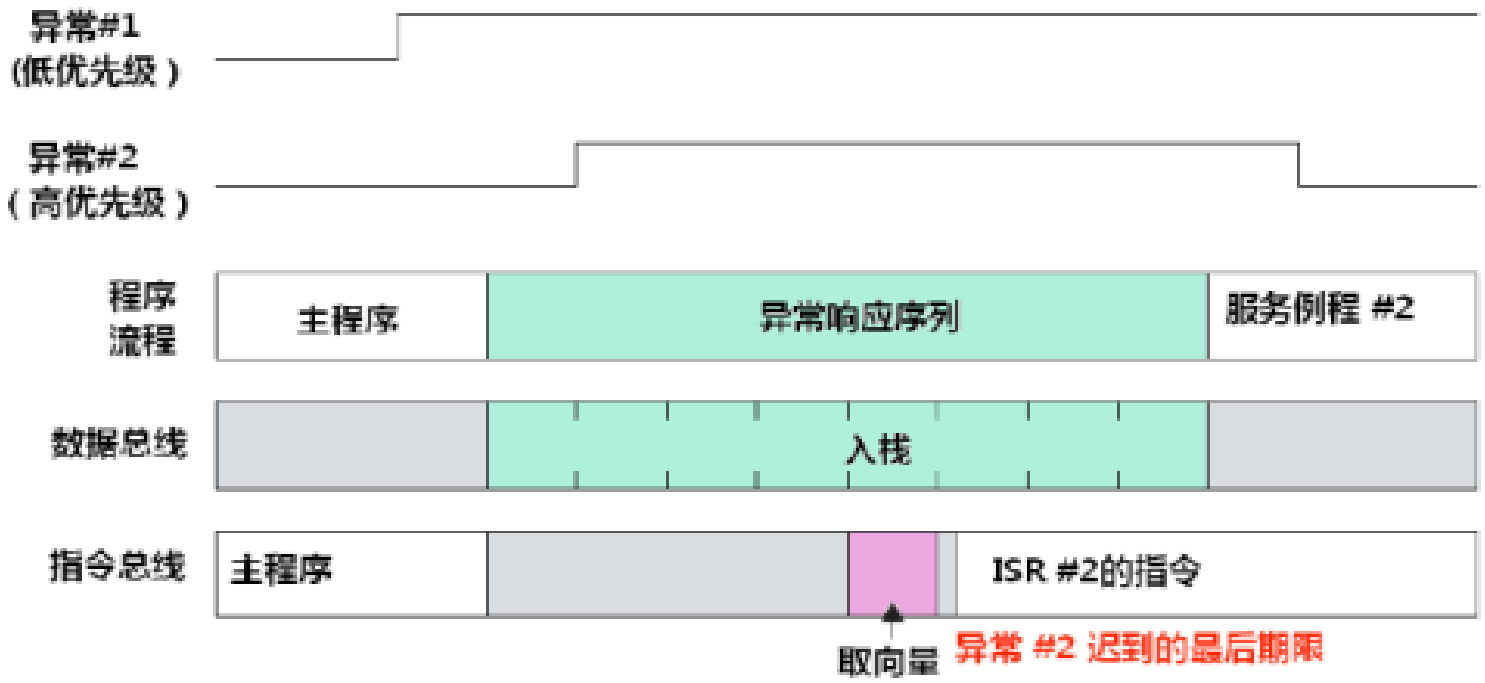
异常咬尾示意图

22


IRQ1
ARM7
在汇编代码中
处理中断
PUSH
ISR 1
POP
PUSH
ISR 2
POP
26-42
不必要的POP与PUSH
16
Cortex-M3
在硬件中
处理中断
PUSH
ISR 1
ISR 2
POP
12
重新取向量，修改
寄存器，共需6周期
12

晚到异常

当CM3对某异常的响应序列还处在早期：入栈的阶段，尚未执行其服务例程时，如果此时收到了高优先级异常的请求，则本次入栈就成了为高优先级中断所做的了——入栈后，将执行高优先级异常的服务例程。（执行完成之后，还会中断咬尾）



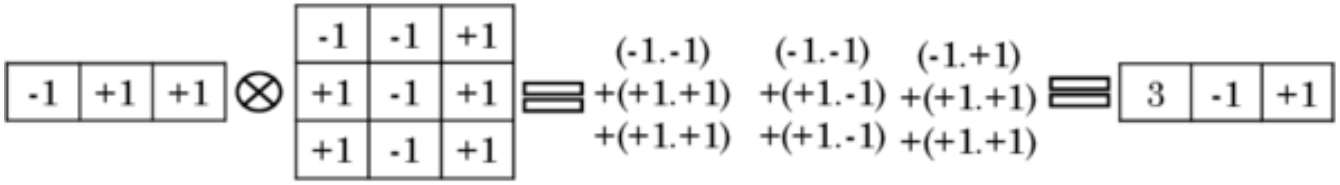
晚到异常的处理模式图

Technical Report (20%)

- **Topics**
  - Survey: How to implement BNN on MCU
    - https://www.tensorflow.org/lite/microcontrollers
    - Harvard Univ: http://www.eecs.harvard.edu/~htk/publication/2017-ewsn-mcdanel-teerapittayanon-kung.pdf
  - Light-weighted DNN
  - Applications: Deep learning on Wearable Devices
  - Applications: Deep learning on IoT devices
  - Survey: RISC-V and open-source ISA

- **3-4 pages with A4 format**

Implementing binarized MM using XNOR and BCNT



(a) An example of binarized MM

(b) Binarized MM using XNOR and BCNT. -1 is represented using 0.

BCNT= OneCount-ZeroCount

IN	Computation	OUT
000	-1-1-1= -3	101
001	-1-1+1 = -1	111
010	-1+1-1= -1	111
011	-1+1+1= +1	001
100	+1-1-1= -1	111
101	+1-1+1= +1	001
110	+1+1-1= +1	001
111	+1+1+1= +3	011

(c) BCNT using a lookup table (OUT is in 2's complement form)

2*BCNT(XNOR(A*B))-N

- For FPGA, XNOR gate can be implemented for BNN, avoiding float MM operation.
- However, the GPU implementations of BNN is still in a proof-of concept stage.

- a 和 b 分别是-1和+1的向量
- A和B分别是0和1的向量（0代表-1，1代表1）
- A=（a+1）/2； B=（b+1）/2
- a=2A-1; b=2B-1
- a*b=(2A-1)*(2B-1)=2(2AB-(A+B)+1)-1=2(AB+(1-A)(1-B))-1
- a*b=2(xnor(A,B))-1

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？

```
Popcount:
        MOV R0, # 0x2000_0010
        MOV R1, #0; >>计数
        MOV R2, #0; >>读外存次数，不大于4次
LOOP：  LDR  R4, [R0], #1;
        MOV R3, #0; >>右移次数
        LOOP1: AND R5, R4, #0x01;
        ADD R1, R1, R5
        LSR R4, R4, #1;
        ADD R3, #1;
        CMP R3, #32;
        BNE LOOP1;
        ADD R2, #1;
        CMP R2, 4
        BNE LOOP
        MOV  PC, LR
```

需要：128个循环，有没有更加高效的方法？

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？

如果我们能够构建一个查找表，输入一个字节x,我们对应x存在的值离线算出来。会形成一个256字节的查找表。假设这个查找表存在DATAPOP地址这个地方。

如何写这段程序。

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？

```
        Popcount_LOOKUP:
        MOV R0, # 0x2000_0010
        MOV R1, #0; >>计数
        MOV R2, #0; >>读外存次数，不大于16次
LOOP：  LDRB  R4, [R0], #1;
        MOV R3, #DATAPOP;
        LDRB R5, [R3, R4];
        ADD R1, R1, R5;
        ADD R2, #1;
        CMP R2, 16
        BNE LOOP
        MOV  PC, LR
```

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？

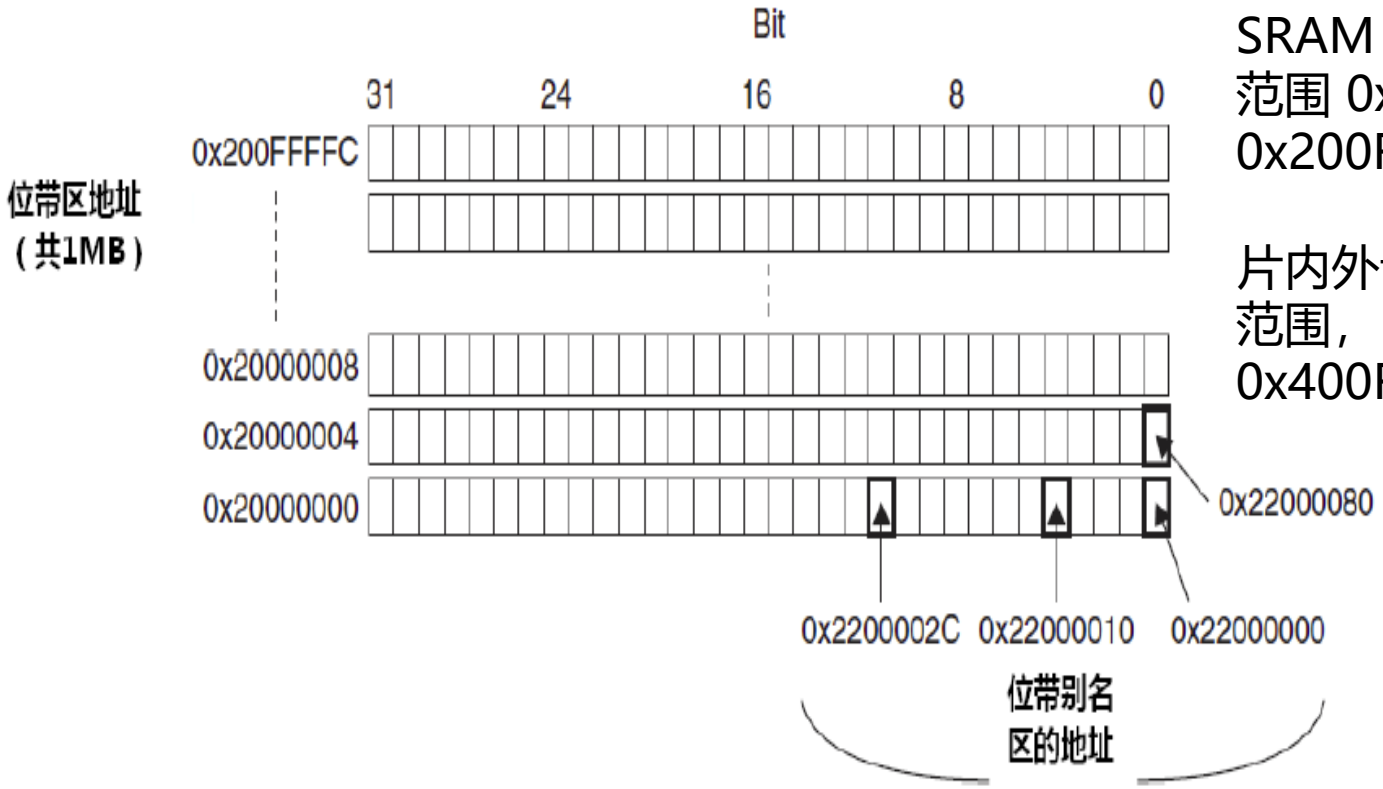
如果我们能够构建一个查找表，输入一个字节x,我们对应x存在的值离线算出来。会形成一个256字节的查找表。

假设这个查找表存在DATAPOP地址这个地方。

如何写这段程序。

位带操作

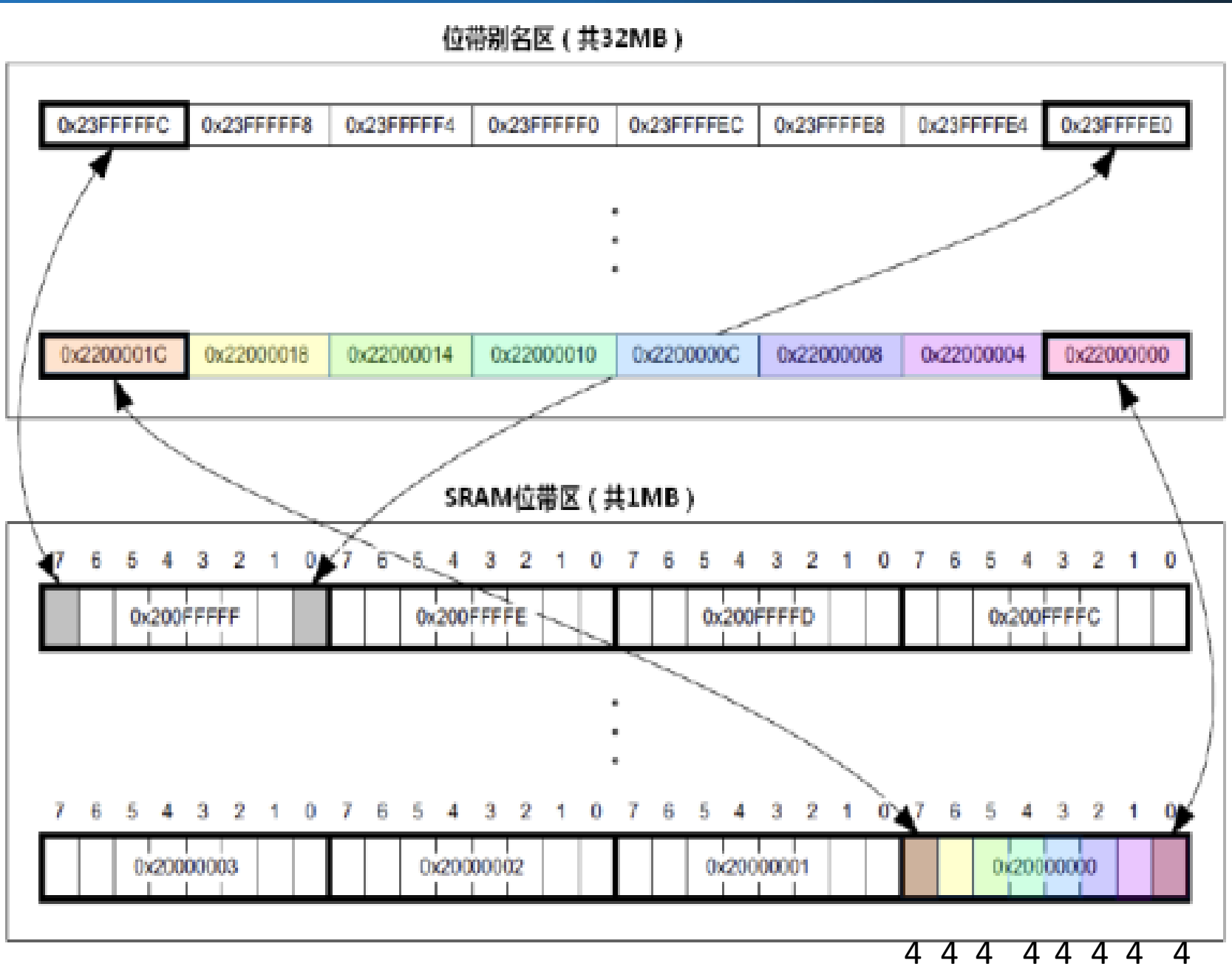
CM3 支持了位操作后，可以使用普通的加载/存储指令来对单一的比特进行读写。在 CM3 支持的位带中，有两个区中实现了位带。

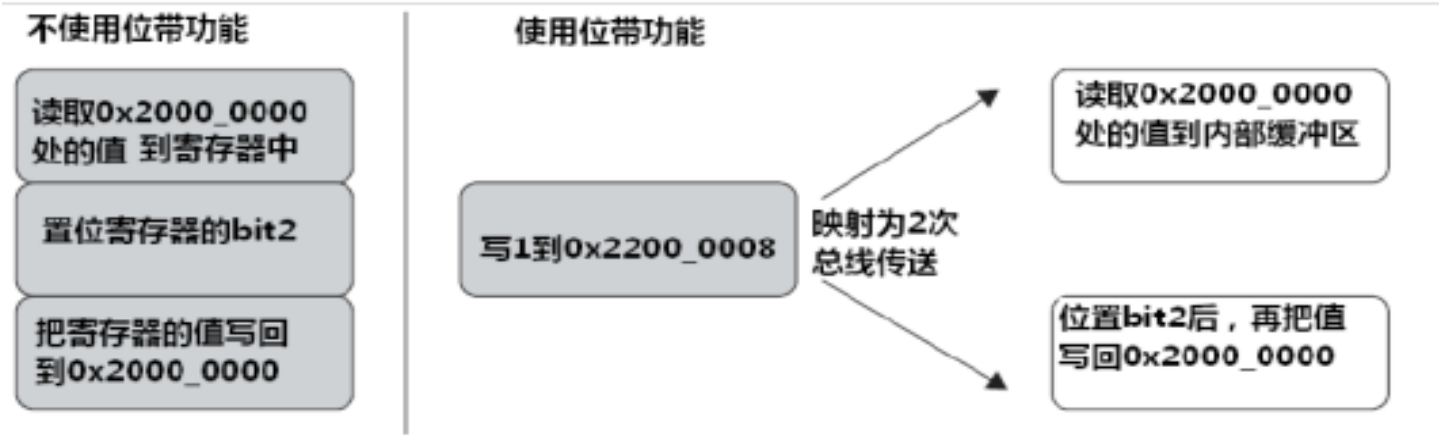


位带区与位带别名区

SRAM 区的最低 1MB 范围 0x20000000 - 0x200FFFFF

片内外设区的最低 1MB 范围， 0x40000000 - 0x400FFFFF


位带别名区（共32MB）
0x23FFFFFC
0x23FFFFF8
0x23FFFFF4
0x23FFFFF0
0x23FFFFEC
0x23FFFFE8
0x23FFFFE4
0x23FFFFE0
0x2200001C
0x22000018
0x22000014
0x22000010
0x2200000C
0x22000008
0x22000004
0x22000000
SRAM位带区（共1MB）
7 6 5 4 3 2 1 0
0x200FFFFF
0x200FFFFE
0x200FFFFD
0x200FFFFC
0x20000003
0x20000002
0x20000001
0x20000000
4 4 4 4 4 4 4 4



写数据到位带别名区

Without Bit-Band

```
LDR   R0,=0x20000000 ; Setup address
LDR   R1, [R0]       ; Read
ORR.W R1, #0x4       ; Modify bit
STR   R1, [R0] ; Write back result
```

With Bit-Band

```
LDR   R0,=0x22000008 ; Setup address
MOV   R1, #1         ; Setup data
STR   R1, [R0]       ; Write
```

从位带别名区读取比特

无位带

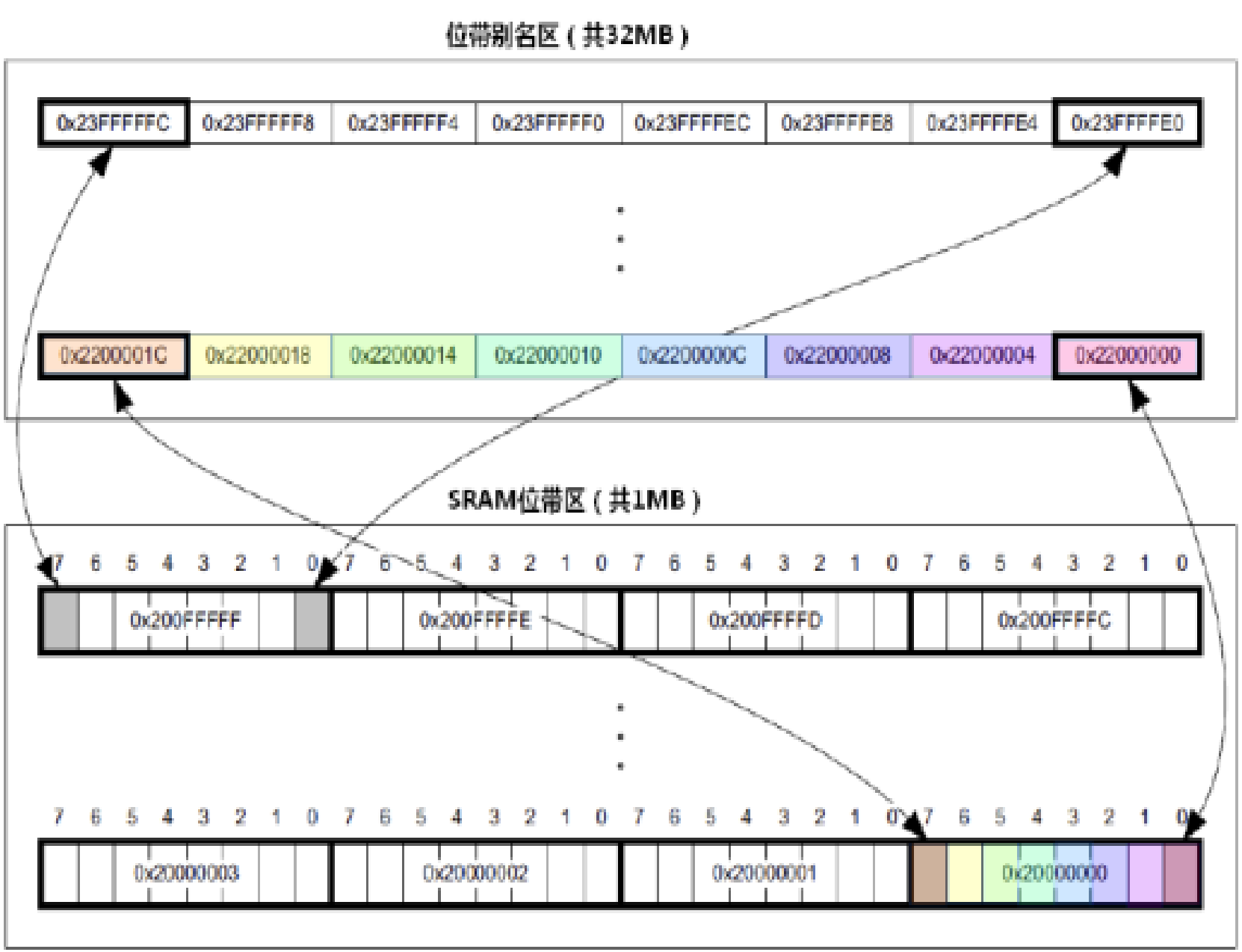
```
LDR     R0,=0x20000000 ; 建立地址
LDR     R1, [R0]       ; Read
UBFX.W  R1,R1, #2, #1  ; 提取bit2
```

有位带

```
LDR     R0,=0x22000008 ; 建立地址
LDR     R1, [R0]       ; Read
```

- **CM3 使用如下术语来表示位带存储的相关地址:**
  - 位带区: 支持位带操作的地址区。
  - 位带别名： 对别名地址的访问最终作用到位带区的访问上（注意：这中途有一个地址映射过程）。带区中的每个比特都映射到别名地址区的一个字 —— 这是只有 LSB 有效的字（位带别名区的字只有最低位有意义）。
- 对于SRAM中的某个比特，该比特在位带别名区的地址：
- AliasAddr=0x22000000+((A-0x20000000)*8+n)*4 = 0x22000000 + (A-0x20000000)*32 + n*4
- 对于片上外设位带区的某个比特， 该比特在位带别名区的地址：
- AliasAddr=0x42000000+((A-0x40000000)*8+n)*4 = 0x42000000 + (A-0x40000000)*32 + n*4

- A : 该比特所在的字节的地址，
- n: 0 <= n <= 7 ,
- *4: 表示一个字为 4 个字节，
- *8: 表示一个字节中有 8 个比特。



AliasAddr=0x22000000+((A-0x20000000)*8+n)*4

4 4 4 4 4 4 4 4

- 别名区中的每个字如何对应位带区的相应位:

- bit_word_addr = bit_band_base + (byte_offset x 32) + (bit_number × 4)

- bit_word_addr: 别名存储器区中字的地址;
- bit_band_base: 别名区的起始地址;
- byte_offset: 包含目标位的字节在位段里的序号;
- bit_number: 目标位所在位置（0 ~ 31）

在C语言中使用位带操作

注：使用位段功能时，要访问的变量必须用 volatile 来定义。因为 C 编译器并不知道同一个比特可以有两个地址。所以就要通过 volatile，使得编译器每次都如实地把新数值写入存储器。

在 C编译器中并没有直接支持位段操作，所以欲在 C中使用位段操作，最简单的做法就是#define 一个位段别名区的地址：

```
#define DEVICE_REG0 ((volatile unsigned long *) (0x40000000))
#define DEVICE_REG0_BIT0 ((volatile unsigned long *) (0x42000000))
#define DEVICE_REG0_BIT1 ((volatile unsigned long *) (0x42000004))
...
*DEVICE_REG0 = 0xab;                    //使用正常地址访问寄存器
*DEVICE_REG0_BIT1 = 0x1;
```

更简化版：

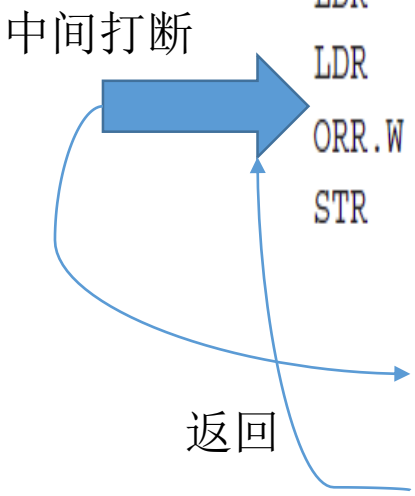
```
//把“位带地址+位序号” 转换成别名地址的宏
#define BITBAND(addr, bitnum)((addr &
0xF0000000)+0x2000000+((addr & 0xFFFFF)<<5)+(bitnum<<2))
//把该地址转换成一个指针
#define MEM_ADDR(addr)  *((volatile unsigned long *) (addr))
于是：
MEM_ADDR(DEVICE_REG0) = 0xAB;       //使用正常地址访问寄存器
MEM_ADDR(BITBAND(DEVICE_REG0,1)) = 0x1;    //使用位段别名地址
```


40

位带操作带来的好处

- 操作简单

- 原子操作，避免多线程访问的中间空挡
  - 原始位操作：需要三条指令



Without Bit-Band

```
LDR   R0,=0x20000000 ; Setup address
LDR   R1, [R0]       ; Read
ORR.W R1, #0x4       ; Modify bit
STR   R1, [R0] ; Write back result
```

With Bit-Band

```
LDR   R0,=0x22000008 ; Setup address
MOV   R1, #1         ; Setup data
STR   R1, [R0]       ; Write
```


Without Bit-Band

```
LDR   R0,=0x20000000 ; Setup address
LDR   R1, [R0]       ; Read
ORR.W R1, #0x4       ; Modify bit
STR   R1, [R0] ; Write back result
```

With Bit-Band

```
LDR   R0,=0x22000008 ; Setup address
MOV   R1, #1         ; Setup data
STR   R1, [R0]       ; Write
```

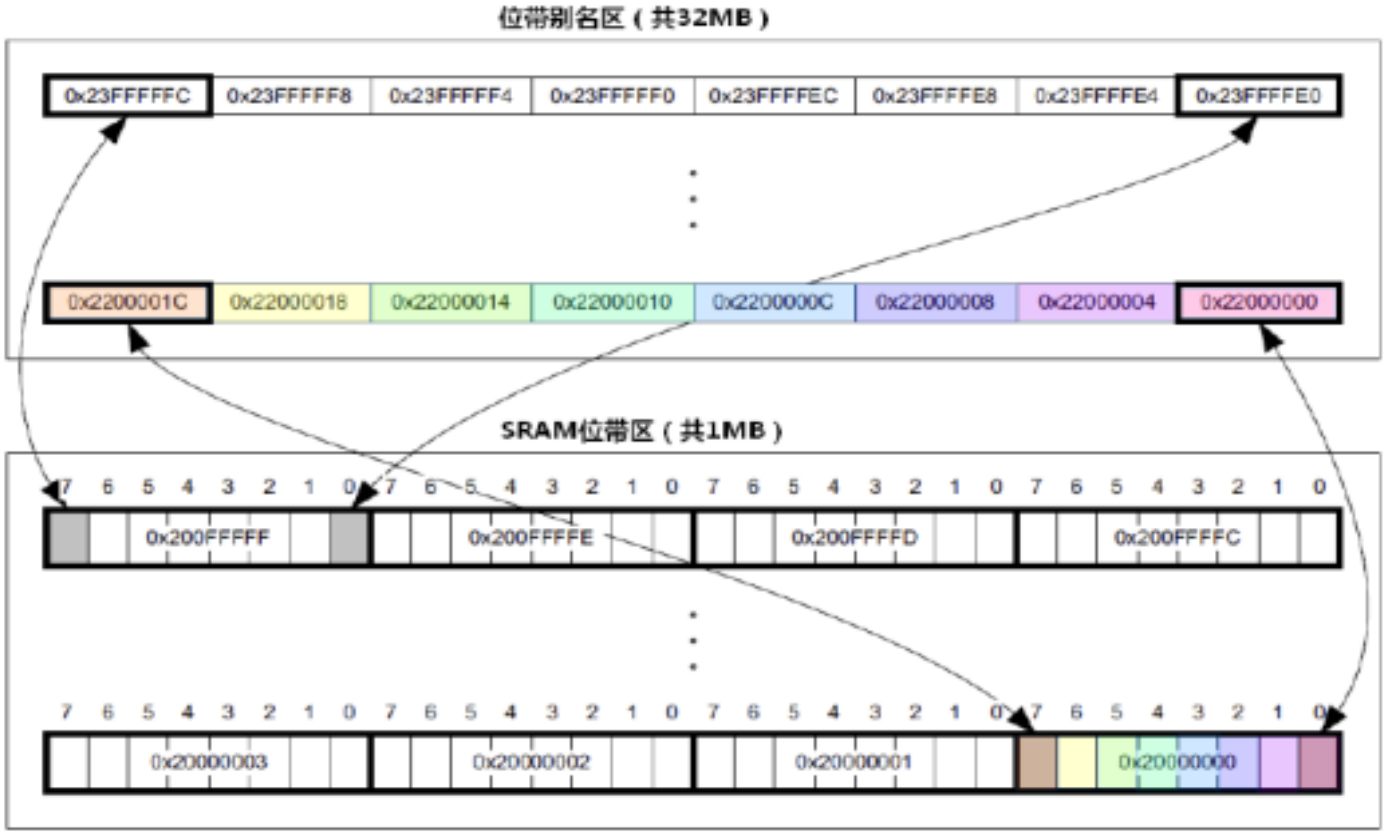
课程习题讲解

- 地址为0x2000_0010所对应的地址的第7位是位带地址是多少？

- Byte_offset=16
- N=7
- 0x2200_0000+16*32+7*4=?

课堂习题

Popcount函数: 从地址为0x2000_0010的地方读取2个64bit的数，统计这个128bit中1的个数？(用位带操作实现)



课堂习题

- 已知数组 A 包含 15 个互不相等的整数，数组 B 包含 20 个互不相等的整数。试编制 一个程序，把既在 A 中又在 B 中出现的整数存放于数组 C 中。

课堂习题

- 已知数组 A 包含 15 个互不相等的整数，数组 B 包含 20 个互不相等的整数。试编制一个程序，把既在 A 中又在 B 中出现的整数存放于数组 C 中。

```
COPY_EQUAL
MOV R0,#A;
MOV R1,#B;
MOV R6, #C
MOV R2,#0;
LOOP: LDR R3, [R0], #4;
        MOV R4, #0;
        LOOP1：  LDR R5, [R1], #4;
        ADD R4,#1;
        CMP R5,R3;
        STREQ R5, [R6],#4;
        CMP R4,#20；
        BNE LOOP1；
        ADD R2, #1;
        CMP R2,#15;
        BNE LOOP
        MOV PC, LR
```

课堂习题

- 试编写一个程序，求出首地址为 DATA 的 100D 字数组中的最小偶数，并把它存放在 MIN地址中