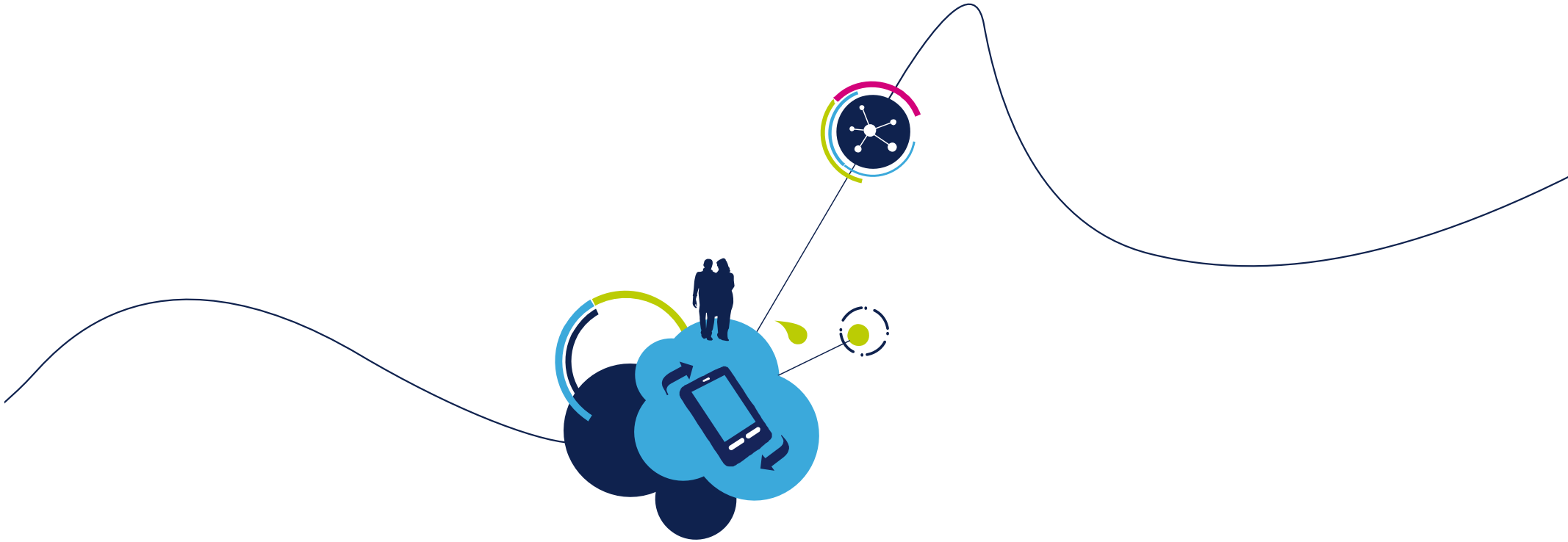
ARM Cortex-M系列内核及STM32相关介绍

life.augmented

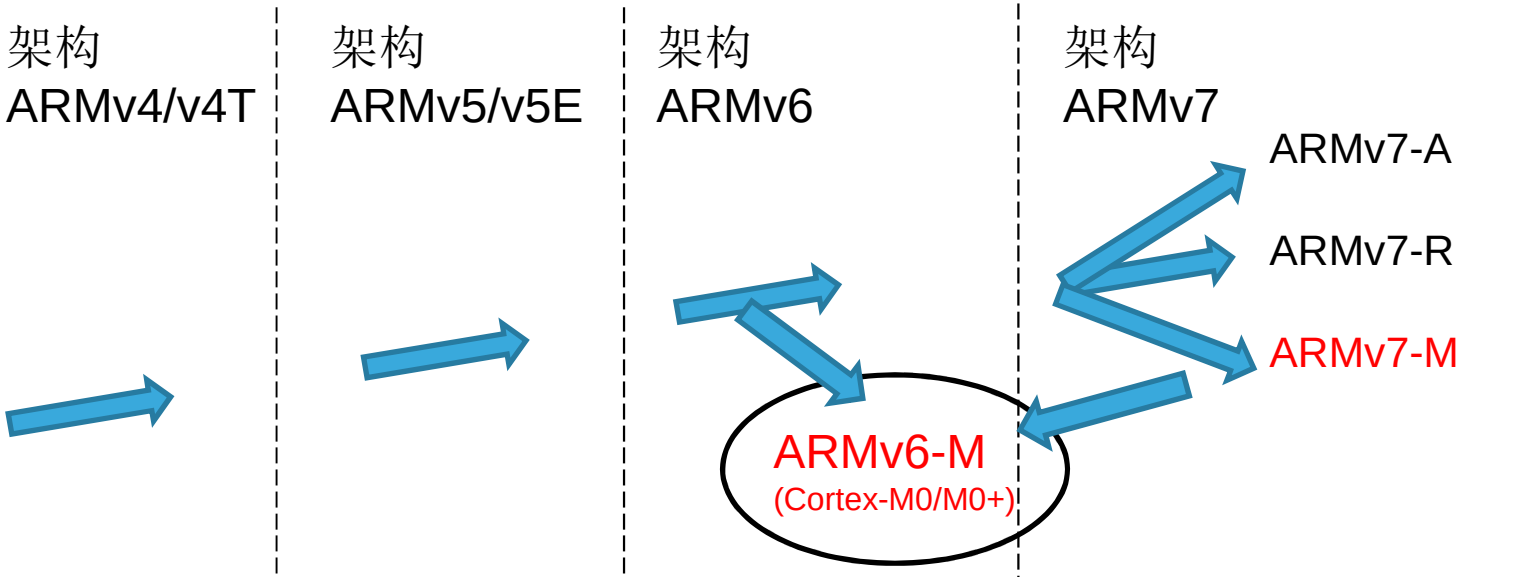
目录



ARM Cortex-M系列概述

life.augmented

ARM处理器架构

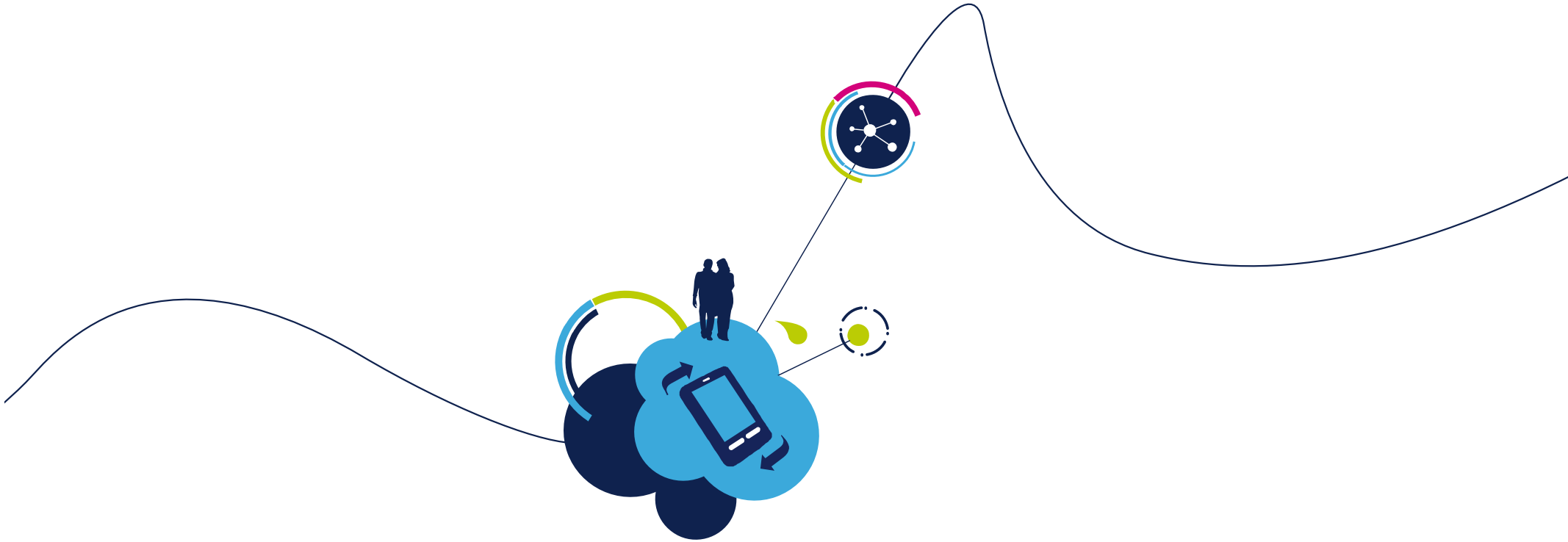

架构
ARMv4/v4T
架构
ARMv5/v5E
架构
ARMv6
架构
ARMv7
ARMv7-A
ARMv7-R
ARMv7-M
ARMv6-M
(Cortex-M0/M0+)
ARM7TDMI，
ARM920T，
ARM922T
ARM926EJ-S，
ARM946E，
ARM966E
ARM1136，
ARM1176，
ARM1156T-2
Cortex-A9，
Cortex-R4，
Cortex-M3,Cortex-M4

Cortex-M系列处理器

ARM Cortex-M0+	ARM Cortex-M0	ARM Cortex-M3	ARM Cortex-M4
8/16 位应用	8/16 位应用	16/32 位应用	32位/DSC应用
在M0的基础上进一步降低功耗，提高性能。	低能耗，低成本，适用低功耗微控制器和深度嵌入式应用	出色的计算性能以及对事件的优异系统响应能力。 适用于具有较高确定性的实时应用	高效的信号处理功能。适用需要有效且易于使用的控制和信号处理功能混合的数字信号控制市场

程序和开发工具都兼容

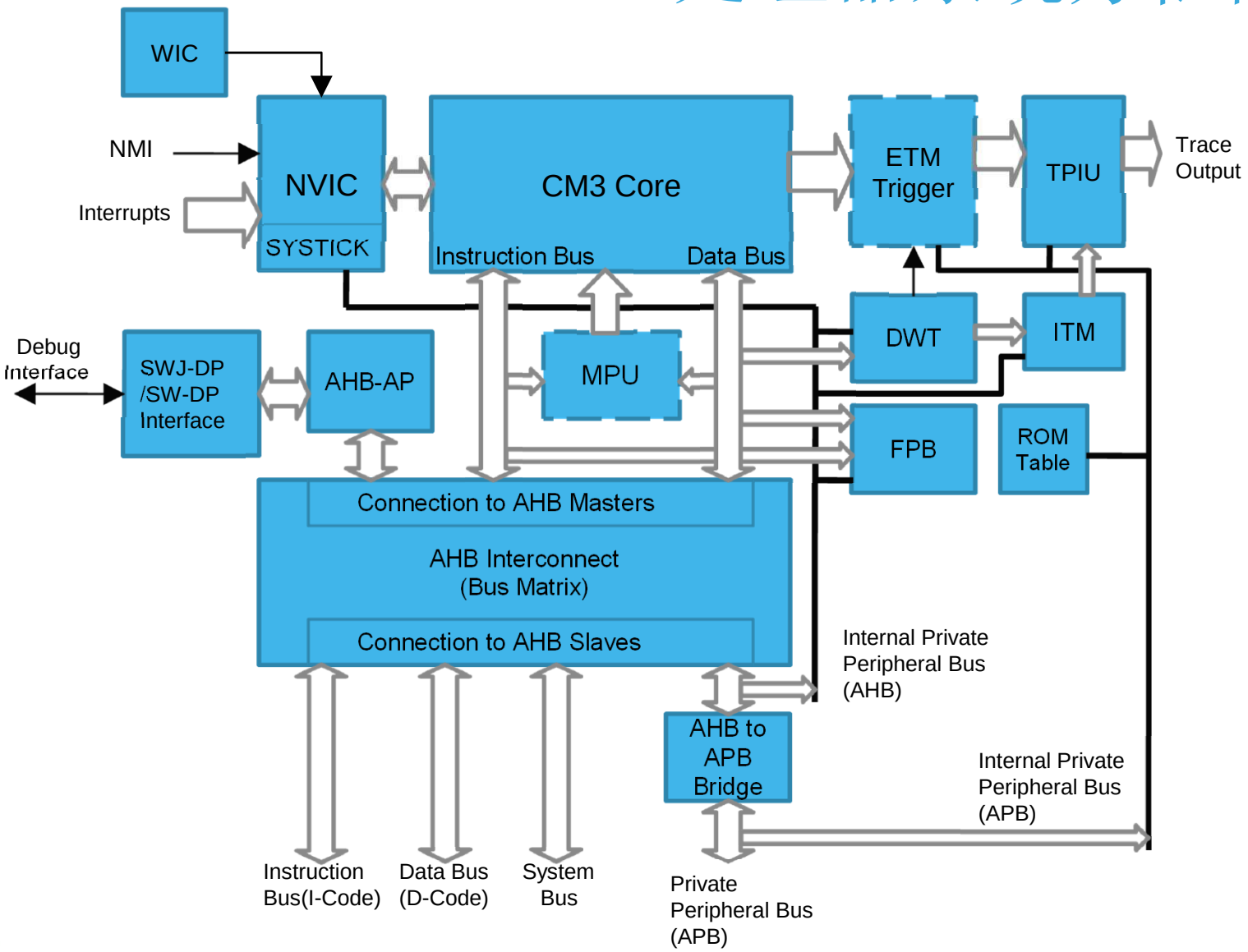
life.augmented



Cortex-M3介绍

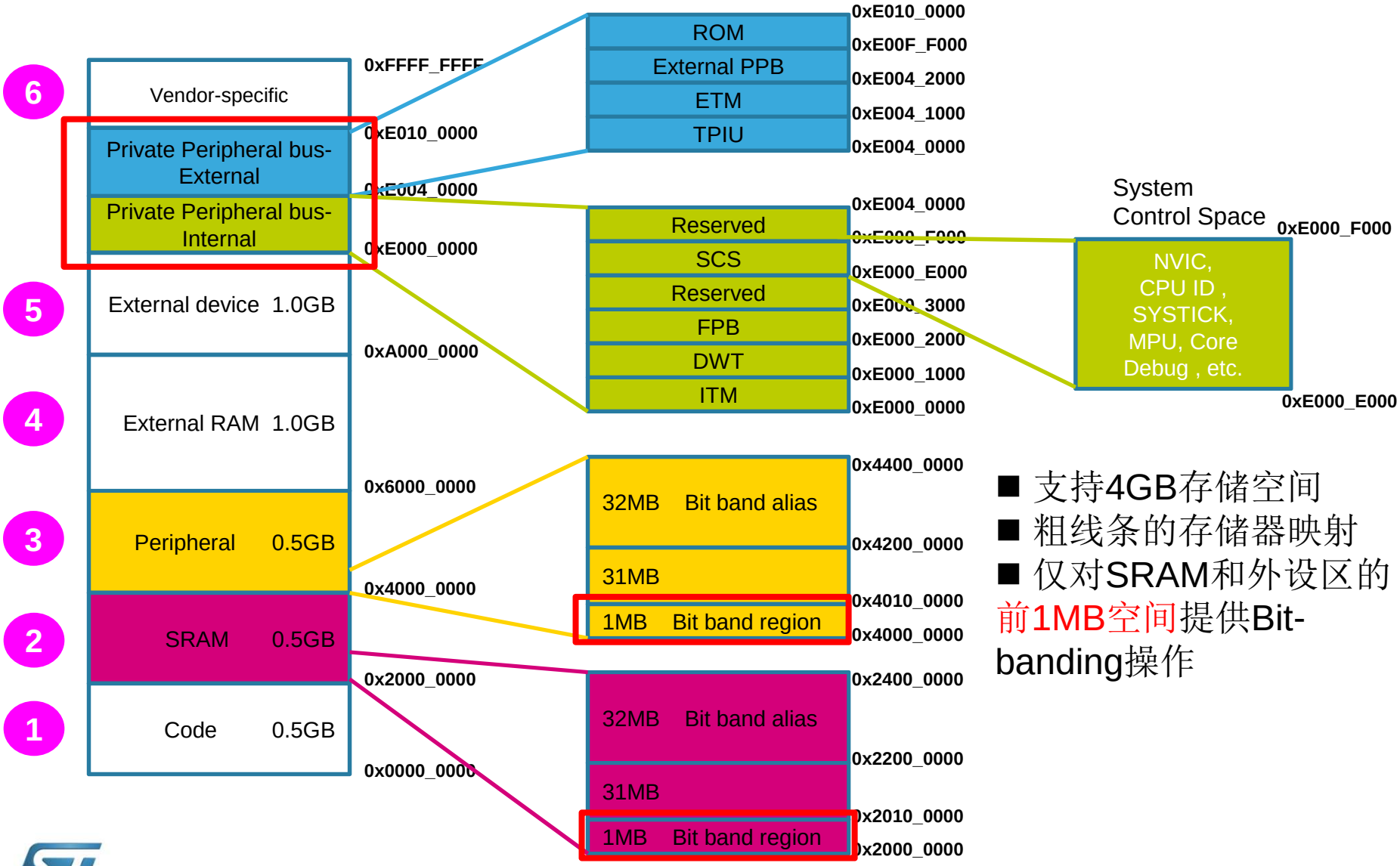
life.augmented

Cortex-M3处理器系统方框图



- 存储器系统
  - 存储器映射
  - STM32存储器重映射
  - 位带操作
  - STM32的GPIO位操作
- 编程模式
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

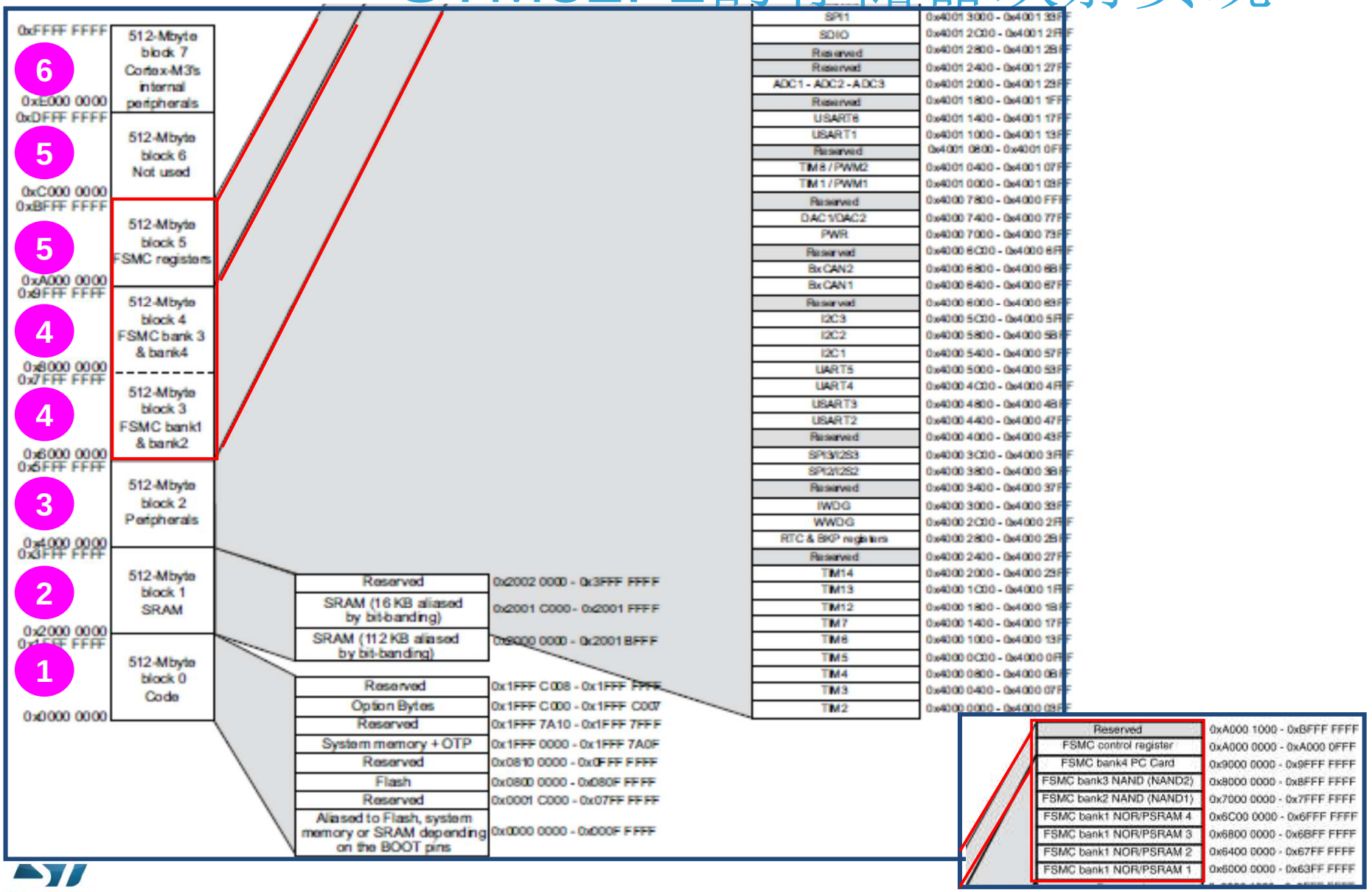
Cortex-M3存储器映射预定义



- 支持4GB存储空间
- 粗线条的存储器映射
- 仅对SRAM和外设区的前1MB空间提供Bit-banding操作

life.augmented

STM32F2的存储器映射实现



life.augmented

- 存储器系统
  - 存储器映射
  - STM32存储器重映射
  - 位带操作
  - STM32的GPIO位操作
- 编程模式
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

STM32存储器地址重映射
——通过Boot引脚配置(1/2)

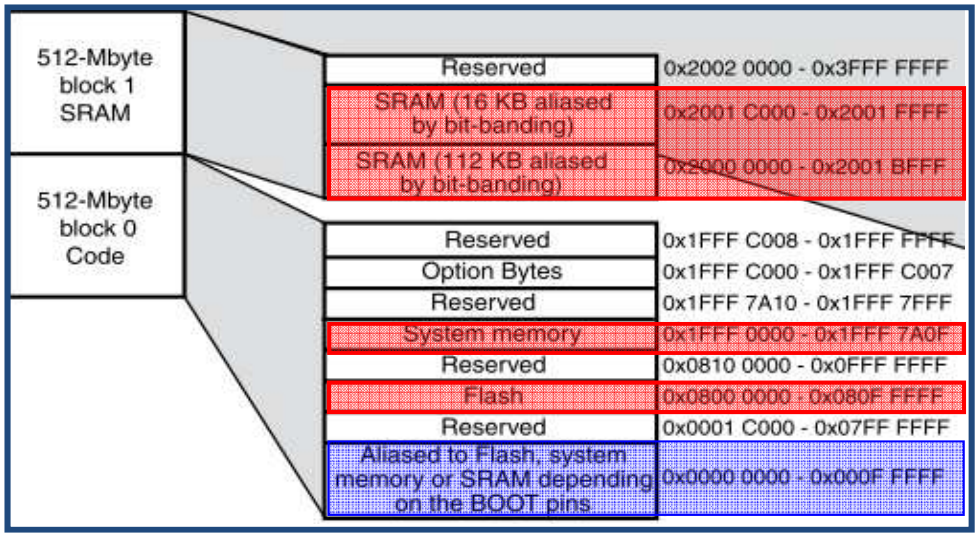
- 复位之后总是从地址0x04取复位代码的起始地址，以及从地址0x0取堆栈指针，开始执行复位程序

- 三块可用于启动的存储器的物理地址
  - 片上SRAM起始地址：0x2000,0000
  - 片上用户闪存起始地址：0x0800,0000
  - 片上系统闪存(bootloader)起始地址：0x1FFF,0000

- 由Boot引脚决定以上

三块物理存储器中的哪

块映射到0x0的起始地址
  - 见下页表格



life.augmented

STM32存储器地址重映射
——通过Boot引脚配置(2/2)

- BOOT引脚在复位后第四个SYSCLK上升沿采样锁定
  - BOOT0引脚是启动专用引脚（输入方向）
  - BOOT1和GPIO共享引脚
    - 采样确定启动空间后，即可被用户做GPIO使用
- 从待机模式(Standby)退出后，会重新采样BOOT引脚电平
  - 进入待机模式之前，用户要注意BOOT1的配置

启动模式选择引脚		启动模式
BOOT1(I/O PB2)	**BOOT0(I)**	
X	0	从片上用户闪存启动
0	1	从片上系统闪存启动，运行bootloader
1	1	从片上SRAM启动
1) 把BOOT电平配置好，复位系统，则可以激活启动代码的运行。 2) 在复位后第四个SYSCLK时钟的上升沿时刻，锁存两个BOOT引脚的电平，由此进入相应的启动空间		

STM32存储器地址重映射
——重映射寄存器

SYSCFG_MEMRMP (地址：0x4001_3800)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														MEM_MODE	
														rw	rw

Bits 31:2　保留

Bits 1:0　MEM_MODE: 设置存储器地址重映射模式，由软件置位和清除。复位之后，这两位的值自动根据BOOT引脚选择的启动模式进行设置。

00：片上用户闪存空间映射到0x0000_0000位置

01：片上系统闪存空间映射到0x0000_0000位置

10：FSMC Bank1(NOR/RSRAM 1和2)映射到0x0000_0000位置

11：片上SRAM（112KbB)映射到0x0000_0000位置

- 复位之后，BOOT引脚的值被拷贝到该寄存器
  - 唯一例外：当BOOT1=1/BOOT0=0从片上闪存启动时，寄存器值=0x00
- FSMC的映射到地址0x00，只能通过软件来使能
  - 通过I-Bus和D-Bus总线执行代码比映射前使用S-Bus效率更高
  - 重映射后，FSMC寄存器和外部存储器空间中未被重映射的区域就不能被访问了；退出重映射后，恢复访问允许。(*不同系列的芯片情况不一样，后面有具体说明）

life.augmented

STM32地址重映射特性

Addresses	Boot/Remap in main flash memory	Boot/Remap in embedded SRAM	Boot/Remap in system memory	Remap in FSMC
0x6400_0000 – 0x67FF_FFFF	FSMC Bank1 NOR/PSRAM 2	FSMC Bank1 NOR/PSRAM 2	FSMC Bank1 NOR/PSRAM 2	FSMC Bank1 NOR/PSRAM 2
0x6000_0000 – 0x63FF_FFFF	FSMC Bank1 NOR/PSRAM 1	FSMC Bank1 NOR/PSRAM 1	FSMC Bank1 NOR/PSRAM 1	FSMC Bank1 NOR/PSRAM 1
0x2001_C000 – 0x2001_FFFF	SRAM2(16KB)	SRAM2(16KB)	SRAM2(16KB)	SRAM2(16KB)
0x2000_0000 – 0x2001_BFFF	SRAM1(112KB)	SRAM1(112KB)	SRAM1(112KB)	SRAM1(112KB)
0x1FFF_0000 – 0x1FFF_77FF	System memory	System memory	System memory	System memory
0x0810_0000 – 0x0FFF_FFFF	Reserved	Reserved	Reserved	Reserved
0x0800_0000 – 0x080F_FFFF	Flash memory	Flash memory	Flash memory	Flash memory
0x0400_0000 – 0x07FF_FFFF	Reserved	Reserved	Reserved	FSMC Bank1 NOR/PSRAM 2 (Aliased)
0x0000_0000 – X03FF_FFFF	Flash(1MB) Aliased	SRAM1(112KB) Aliased	System memory(30KB) Aliased	FSMC Bank1 NOR/PSRAM 1 (Aliased)

可由BOOT引脚以及重映射寄存器配置

只能由重映射寄存器配置

* 只有Bank1*的前两个区域(region)能被重映射*

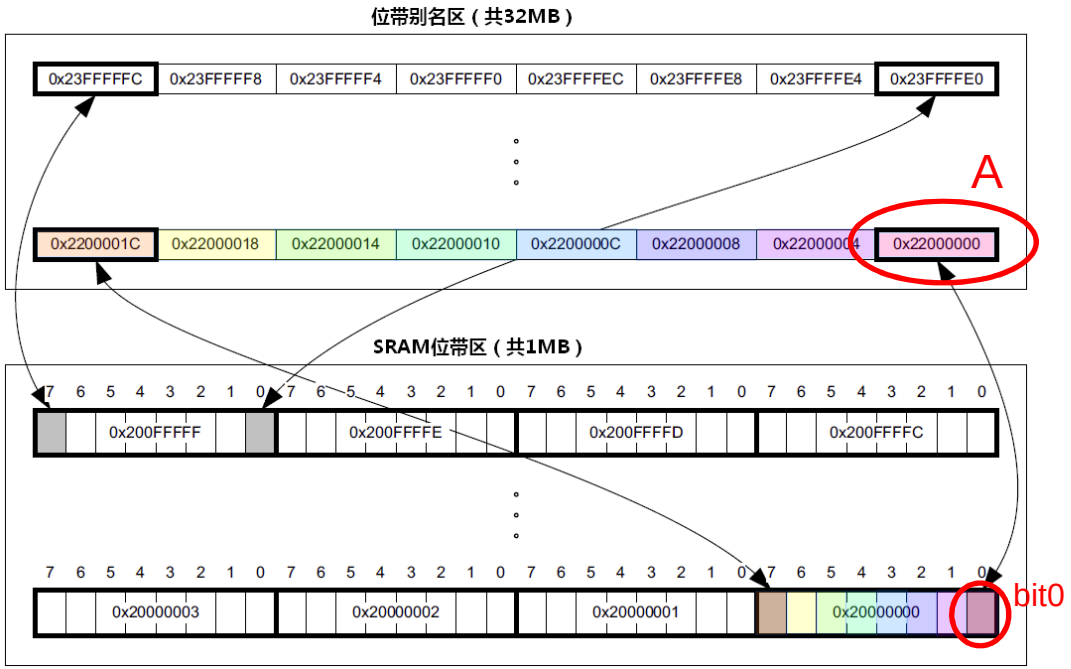
STM32各系列对FSMC地址重映射支持

	可重映射的范围	其他未被重映射的空间	FSMC寄存器
STM32F0xxxx	------	------	------
STM32F1xxxx	FSMC不支持重映射	-------	-------
STM32F2x5/2x7	Bank1 NOR/PSRAM 1 和2区域	不能被访问	不能被访问
STM32F3xxxx	------	------	------
STM32F405xx/407xx, STM32F415xx/417xx,STM32F42xxx, STM32F43xxx	Bank1 NOR/PSRAM 1 和2区域	可以访问	可以访问
STM32F401xx, STM32F411xx	------	------	------
STM32L0xxxx	------	------	------
STM32L1xxxx	Bank1 NOR/PSRAM 1 和2区域	可以访问	可以访问

- 存储器系统
  - 存储器映射
  - STM32存储器地址重映射
  - 位带操作
  - STM32的GPIO位操作
- 编程模式
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

位带操作

- Cortex-M3在SRAM区和片上外设区都开有一个1MB的“位带区”和32MB的“位带别名区”。位带别名区的每个字（只有LSB有效）对应位带区的一个比特位。
- 对位带别名区每个字的操作最终都变换成对位带区对应比特位的操作。
- 访问位带别名区必须字对齐，否则会产生不可预料的结果。
- 对位带别名区的访问操作，将原有的“读-改-写”做成一个硬件级别支持的原子操作，不能被中断打断。
- 在C语言中，使用位带功能时，要访问的变量必须用volatile来定义。因为C编译器并没有直接支持位带操作。



位带区与位带别名区的地址映射

- 计算位带区中某个比特位在位带别名区中的映射地址：

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

bit_band_base : 位带别名区的起始地址
byte_offset :包含目标比特位的字节在位带区的偏移值(字节数)
bit_number : 目标比特位在字节中的位置(0~7)

例如：

在SRAM的0x20004000地址定义一个长度为512字节的数组：

```
#pragma location=0x20004000
__root _no_init u8 Buffer[512];
```

数组首字节的BIT0对应的位带地址为：

0x22000000+(0x4000*32)+(0*4) = 0x22080000

数组第二个字节的BIT3对应的位带地址为：

0x22000000+(0x4001*32)+(3*4) = 0x2208002c

GPIOA的端口输出数据寄存器(ODR)地址0x40020014，对于PA.0来说控制其输出电平的比特位的位带操作地址为：

0x42000000+(0x20014*32)+(0*4) = 0x42400280

位带操作，使用例一

- 例1：将前页数组中的数据通过PA.0端口输出

不使用位带操作：

```
for (u16 cnt=0; cnt<512; cnt++)
    for (u8 num=0; num<8; num++)
        if ((Buffer[cnt]>>num)&0x01)
            GPIOA->BSRR = 1;
        else
            GPIOA->BRR = 1;
```

使用位带操作：

```
volatile U32 *pBuffer = ((u32*)0x22080000)

U16 cnt = 512*8;

While(cnt--)
{
    (*((u32*) 0x42400280)) = *pBuffer++;
}
```

位带操作，使用例二

- 例2：修改SPI控制器的BIT6，使能SPI

不使用位带操作：

U32 spi_ctrl = SPI1->CR1;

Spi_ctril = spi_ctrl | 0x00000040;

SPI1->CR1 = spi_ctril;

使用位带操作：

SPI1的CR1寄存器地址为0x40013000，因此CR1寄存器BIT6的位带地址为：
0x42000000 + 0x13000*32 + 6*4
= 0x42260018

对BIT6的置位操作为：

(*((u32*)0x42260018)) = 1;

使用位带操作注意事项

- 位带操作支持的地址范围：0x2000_0000 ~ 0x2010_0000, 和 0x4000_0000 ~ 0x4010_0000
- 超出位带支持地址范围的存储空间，不能使用位带操作。
  - 例如：STM32F3的GPIO寄存器的地址位于0x4800_0000 ~ 0x4800_17FF，超出了位带操作的范围，所以不支持位带操作。
- 位带操作是通过Cortex核内部的总线矩阵实现的,所以通过DMA访问SRAM不支持位带。

- 存储器系统
  - 存储器映射
  - STM32存储器地址重映射
  - 位带操作
  - STM32的GPIO位操作
- 编程模式
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

GPIO位操作及相关寄存器介绍
——ODR寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 保留

Bit3 15: 0 ODRy: 端口输出的值（y= 0…15)

可以通过软件读写。

- 可对ODR的每一位写0或1，控制对应端口的输出
- 非原子操作

GPIO位操作及相关寄存器介绍
——BSRR，BRR寄存器

BSRR寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bit16~Bit31:写1清零对应端口，写0无效
Bit0~Bit15: 写1置位对应端口，写0无效

- 在BSRR寄存器里同时对同一位做置位/清零操作，置位操作优先级高。
- 对BSRR寄存器的操作是“原子操作”
- STM32F1的BSRR/BRR寄存器只支持字操作
- STM32F2/STM32F4只有BSRR一个寄存器，支持字/半字/字节操作

BRR寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bit16~Bit31:保留
Bit0~Bit15: 写1清零对应端口，写0无效

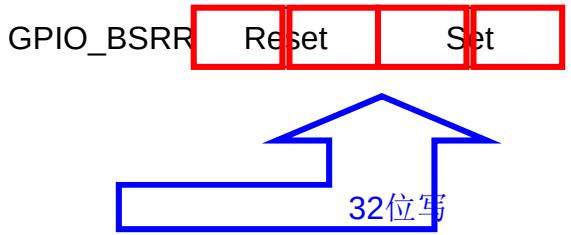
GPIO位操作及相关寄存器介绍
——BSRR,BRR寄存器应用举例

e.g. 同时同向toggle PA.0和PA.1

法一：*>> GPIOA_BSRR = 0x03*

>> GPIOA_BSRR = 0x03<<16

多运行一个移位的指令…效率影响…



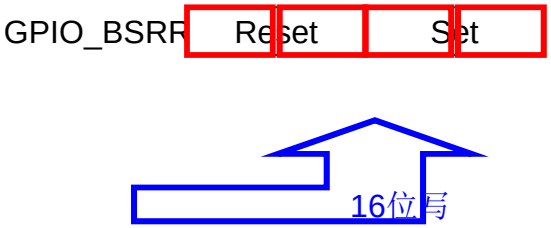
法二：利用寄存器可以半字访问的特点，STM32F1不适用

>> #define GPIOA_BitSet_ADDR = 0xXX

>> #define GPIOA_BitReset_ADDR = 0xXX + 2

*>> *(u16*)GPIOA_BitSet_ADDR = 0x03*

*>> *(u16*)GPIOA_BitReset_ADDR = 0x03*



法三：同时使用BSRR和BRR寄存器， STM32F2/STM32F4不适用

>> GPIOA_BSRR = 0x03

>> GPIOA_BRR = 0x03

e.g. 同时反向toggle PA.0和PA.1

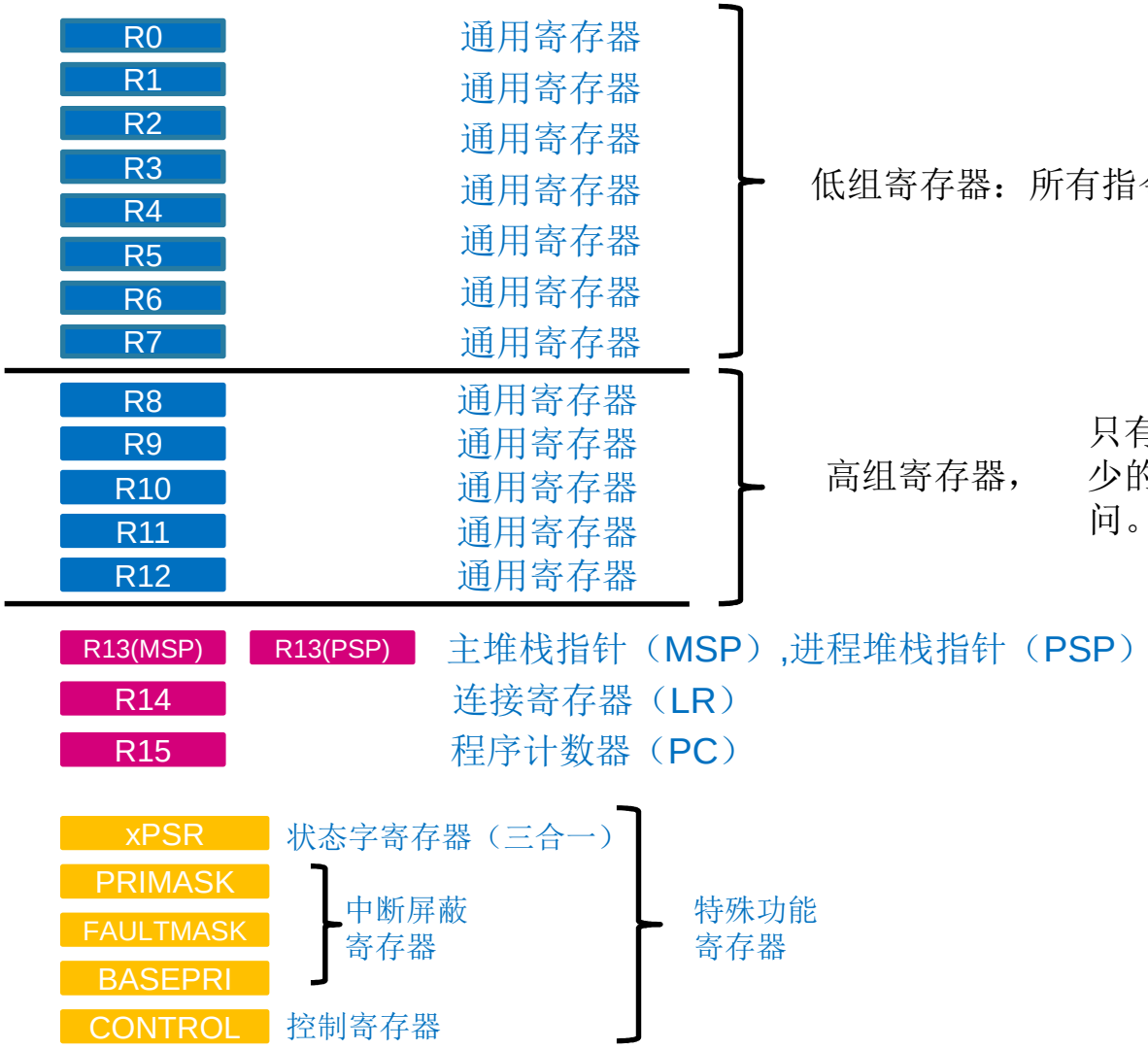
>> GPIOA_BSRR = 0x21

>> GPIOA_BSRR = 0x12

life.augmented

- 存储器系统
- 编程模式
  - 寄存器组
  - 特权操作和双堆栈
  - 流水线
  - 总线结构
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

寄存器组



- 主堆栈指针（MSP）：由OS内核，异常服务例程，以及所有需要特权访问的应用程序代码使用；
- 进程堆栈指针（PSP）：用于常规的应用程序代码。
- LR：保存子函数的返回地址和异常的返回值

- 存储器系统
- 编程模式
  - 寄存器组
  - 特权操作和双堆栈
  - 流水线
  - 总线结构
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

操作模式与特权级

- 两种操作模式----线程模式 VS Handler模式
- 两级特权级别----特权级 VS 非特权级

操作模式	执行代码类型	特权级别
线程模式	应用程序	特权级或者非特权级(CONTROL.0)
Handler模式	异常/中断服务程序	特权级

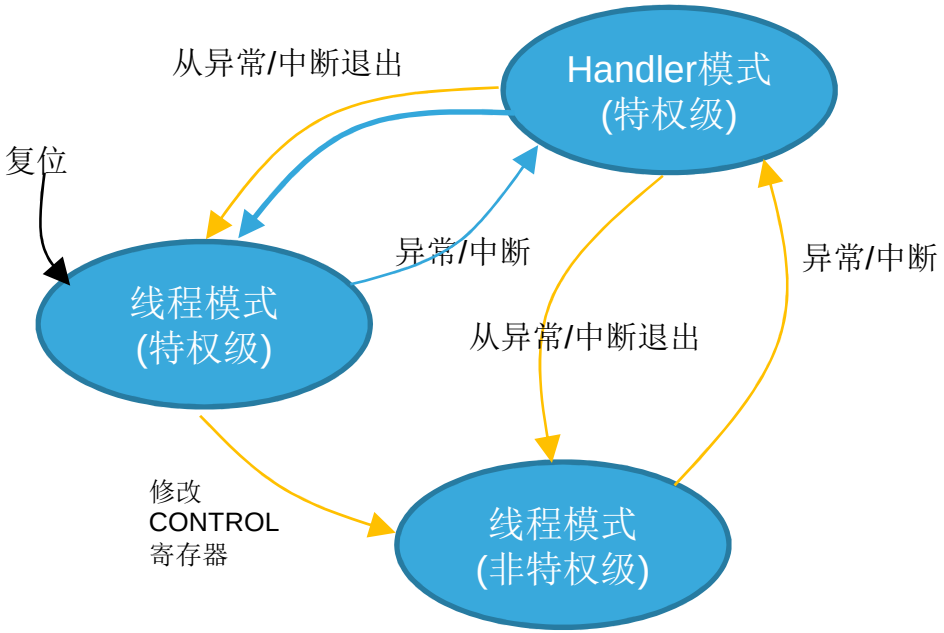
- 可以通过CONTROL[0]定义特权级别

位	功能
Bit[1]	0==选择主堆栈指针MSP(复位后的缺省值）； 1==选择进程堆栈指针PSP Handler模式下只允许使用MSP。
Bit[0]	0==特权级的线程模式； 1==用户级的线程模式 Handler模式永远是特权级的

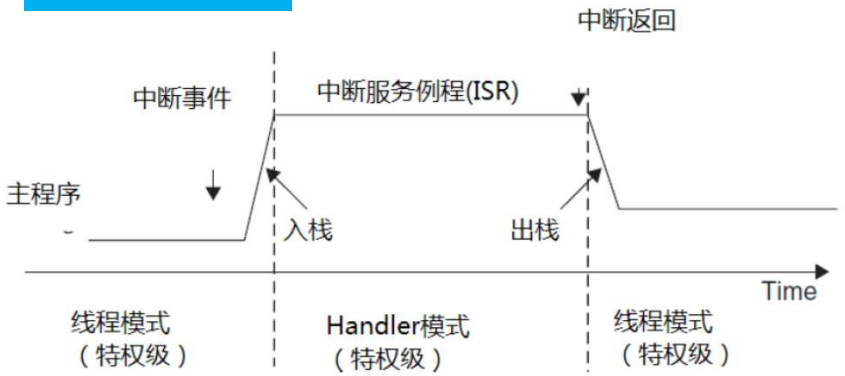
***非特权级线程模式下**

- **禁止对SCS的访问;**
- **禁止使用MRS/MSR访问除APSR之外的特殊功能寄存器**

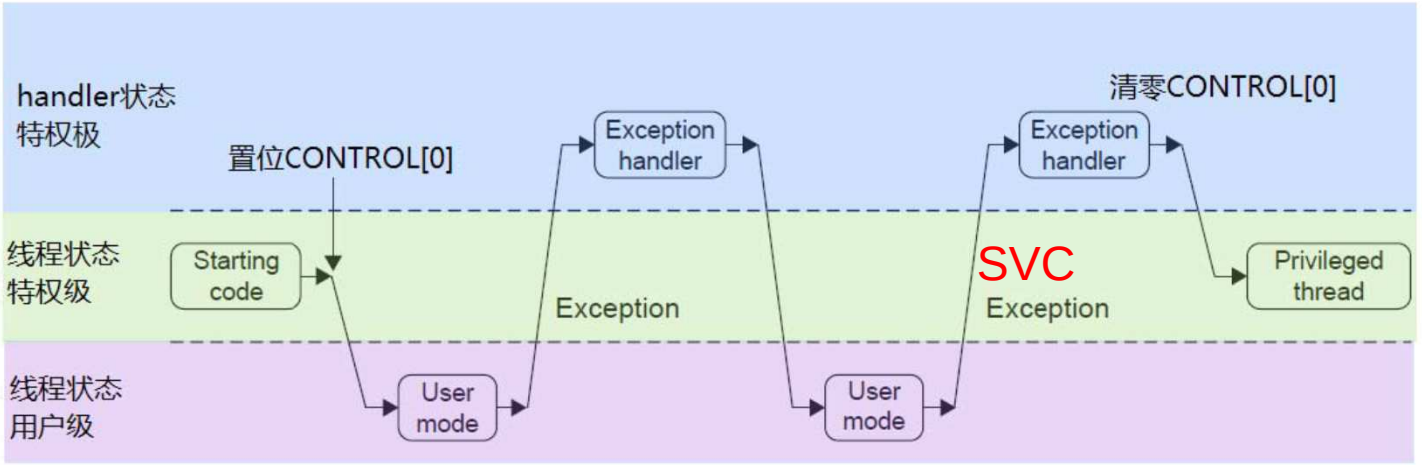
状态转换图



蓝色路径：



黄色路径：



双堆栈指针（MSP&PSP）

- CM3内核中有两个堆栈指针（MSP,PSP），但任何时刻只能用到其中一个（banked)
- 复位后默认使用MSP（复位后程序处于特权级）
- 通过SP访问到的是正在使用的那个指针（CONTROL.1寄存器中设置），可以通过MSR/MRS指令访问指定的指针

操作模式	执行代码类型	特权级	堆栈指针
线程模式	应用程序	特权级或者非特权级	MSP或者PSP (CONTROL.1)
Handler模式	异常服务程序	特权级	MSP

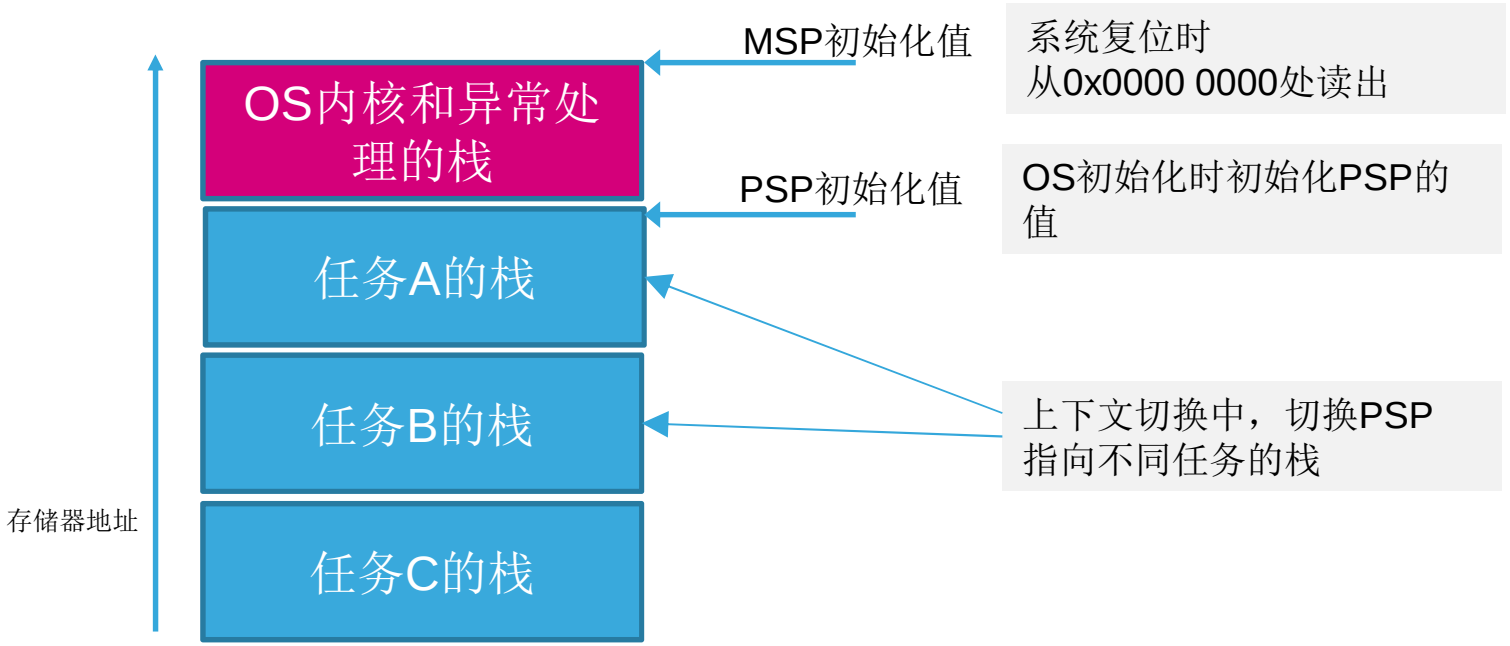
- 通过CONTROL寄存器选择使用哪个堆栈指针

位	功能
CONTROL[1]	0==选择主堆栈指针MSP(复位后的缺省值）； 1==选择进程堆栈指针PSP Handler模式下只允许使用MSP。
CONTROL[0]	0==特权级的线程模式； 1==用户级的线程模式 Handler模式永远是特权级的

双堆栈指针在OS中的应用

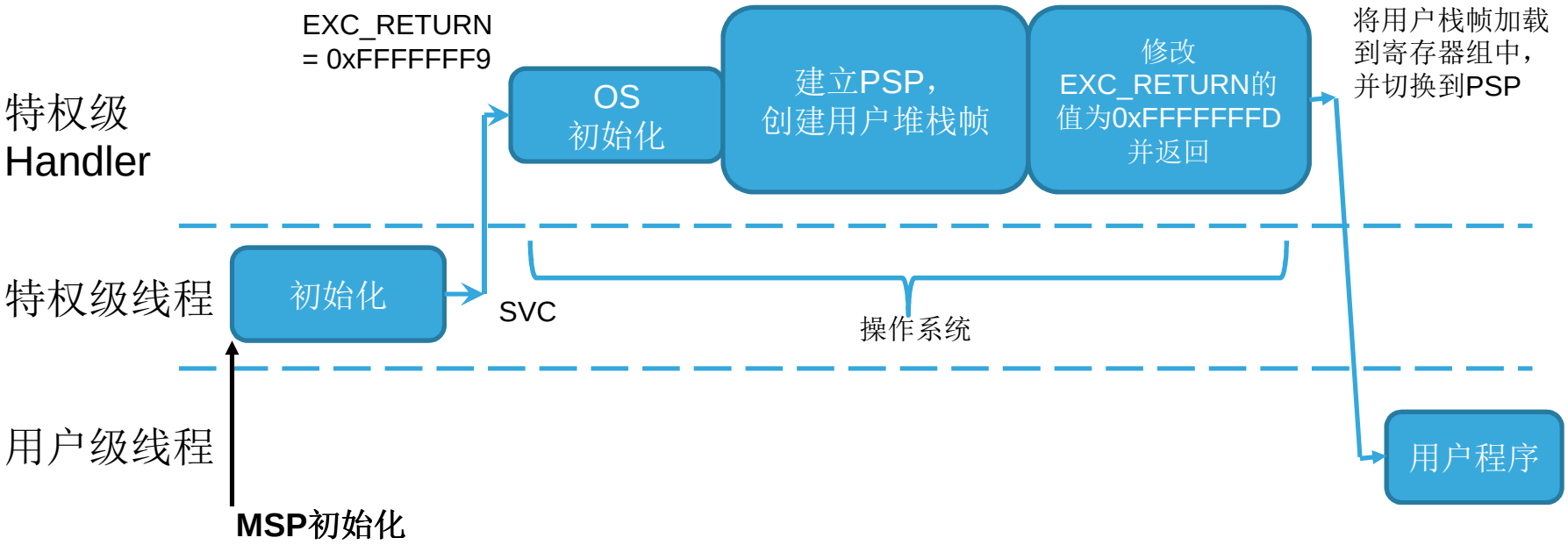
典型的OS环境中，MSP和PSP的用法如下：

- MSP，用于OS内核和异常处理
- PSP，用于应用任务



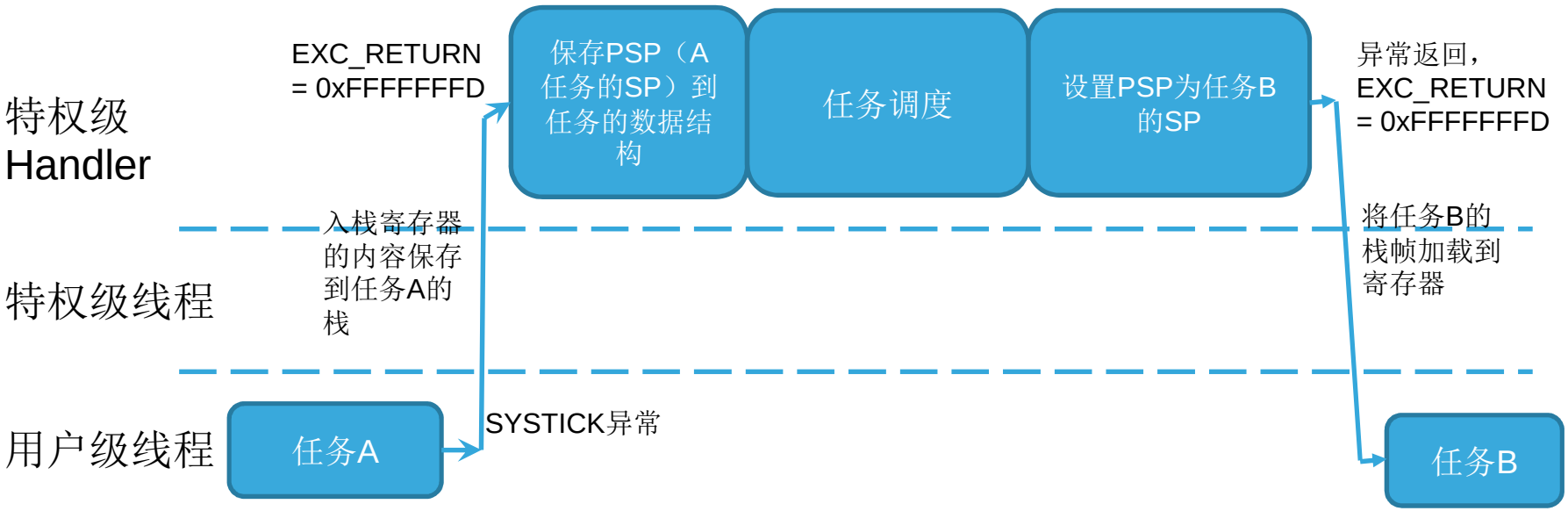
双堆栈指针的初始化

- 系统复位时 从0x0000 0000处读出MSP的初始值
- 在OS初始化时，对PSP进行初始化



PSP指针在不同任务间切换

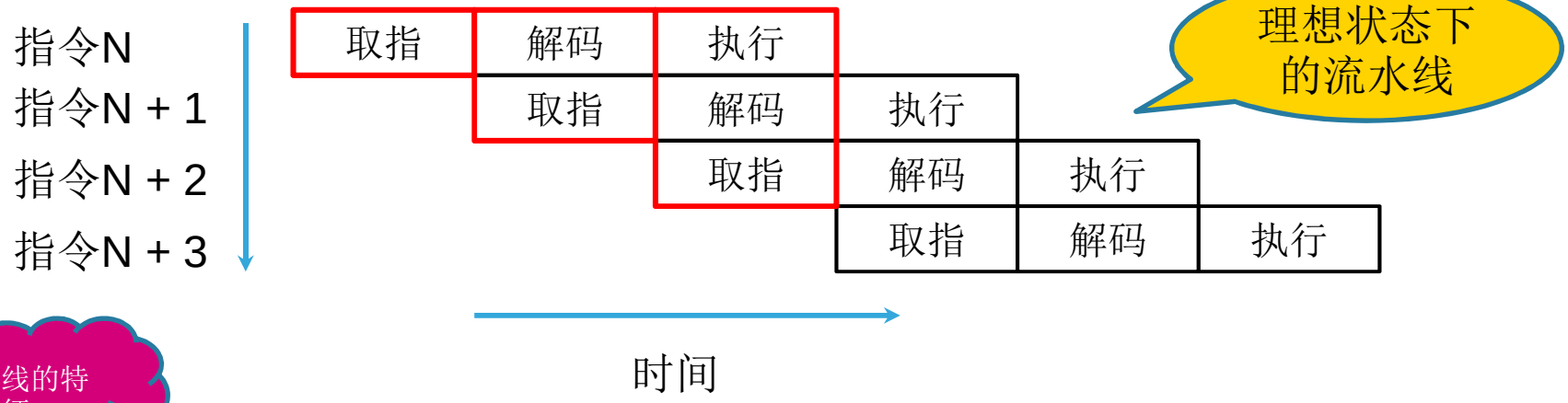
- 用任务A的SP执行入栈操作，并保存任务A的SP
- 设置PSP指向任务B的栈空间，用任务B的SP执行出栈，随后开始执行任务B



- 存储器系统
- 编程模式
  - 寄存器组
  - 特权操作和双堆栈
  - 流水线
  - 总线结构
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

流水线

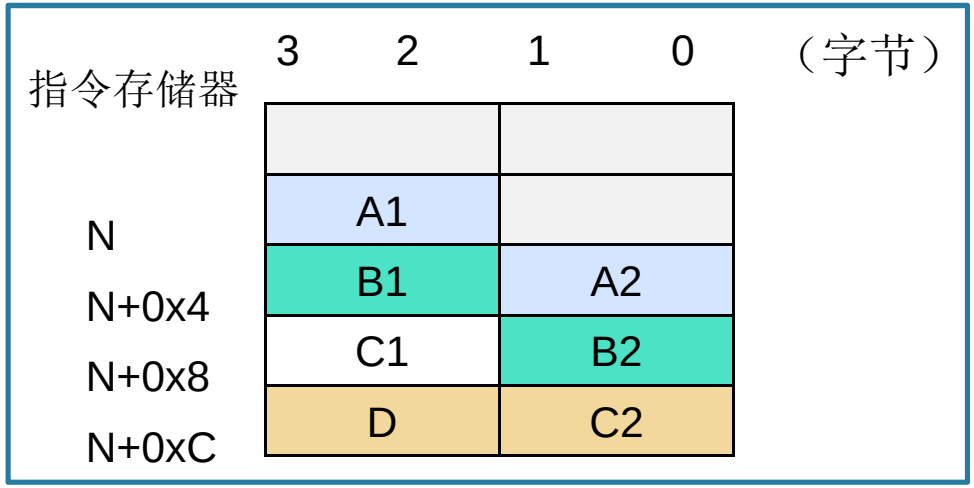
Cortex-M3使用一个3级流水线：取指，解码和执行



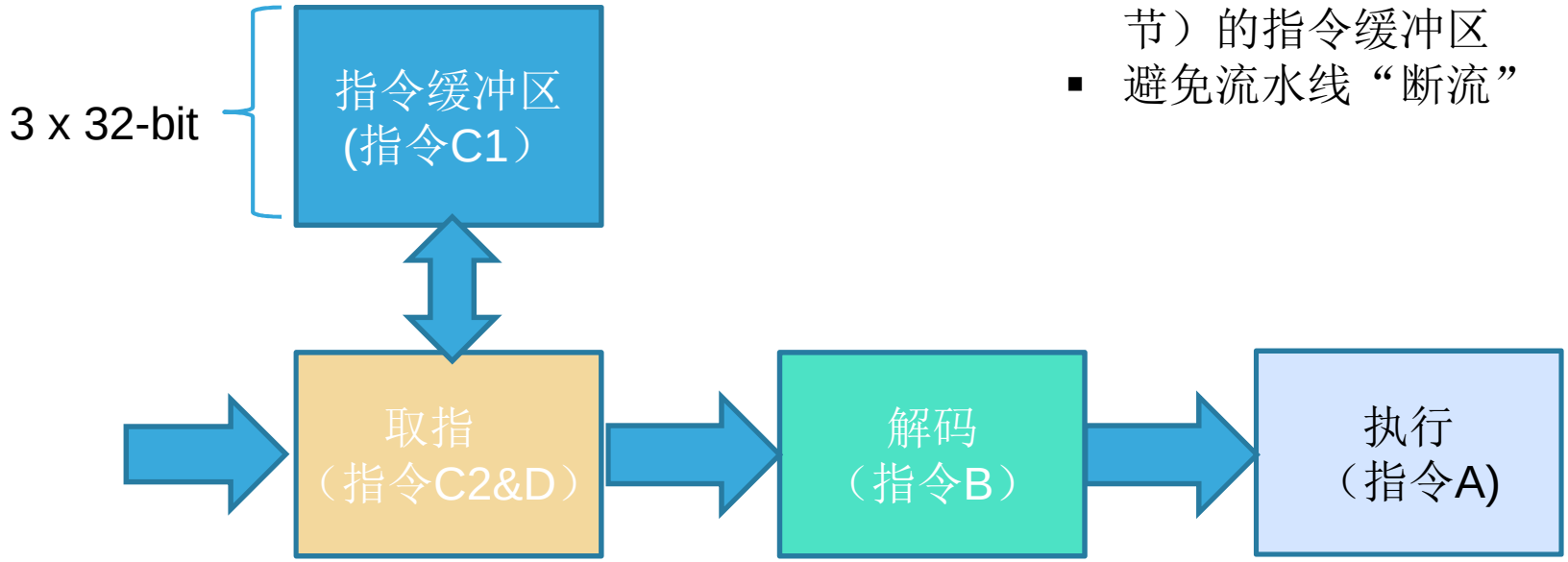
流水线的特征

- 执行一条分支指令或直接修改PC而发生跳转时，ARM内核有可能会清空流水线，而需要重新读取指令。
- 即使产生了一个中断，一条处于“执行”阶段的指令也将会完成。流水线里其他指令将会被放弃，而处理器将从向量表的适当入口开始填充流水线。
- 不论是执行16位指令还是32位指令，读取PC时，会返回当前指令地址+4的值。

CM3的预取指单元（PFU）



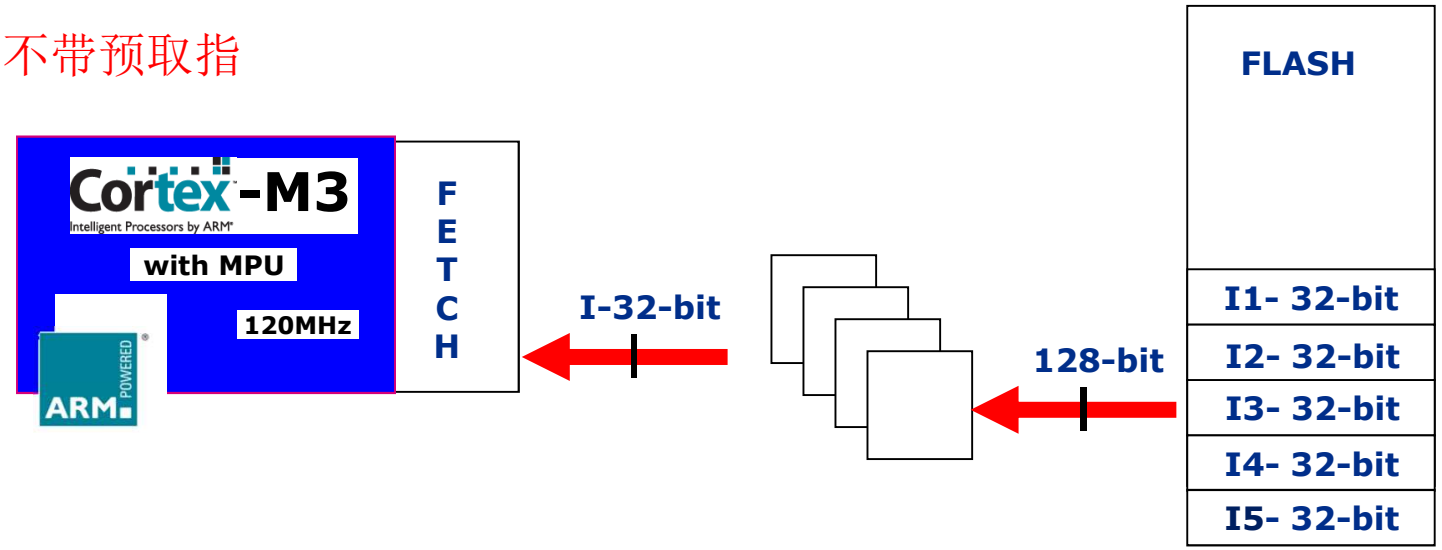
- CM3内核的预取指单元带有一个3个字长度（12字节）的指令缓冲区
- 避免流水线“断流”



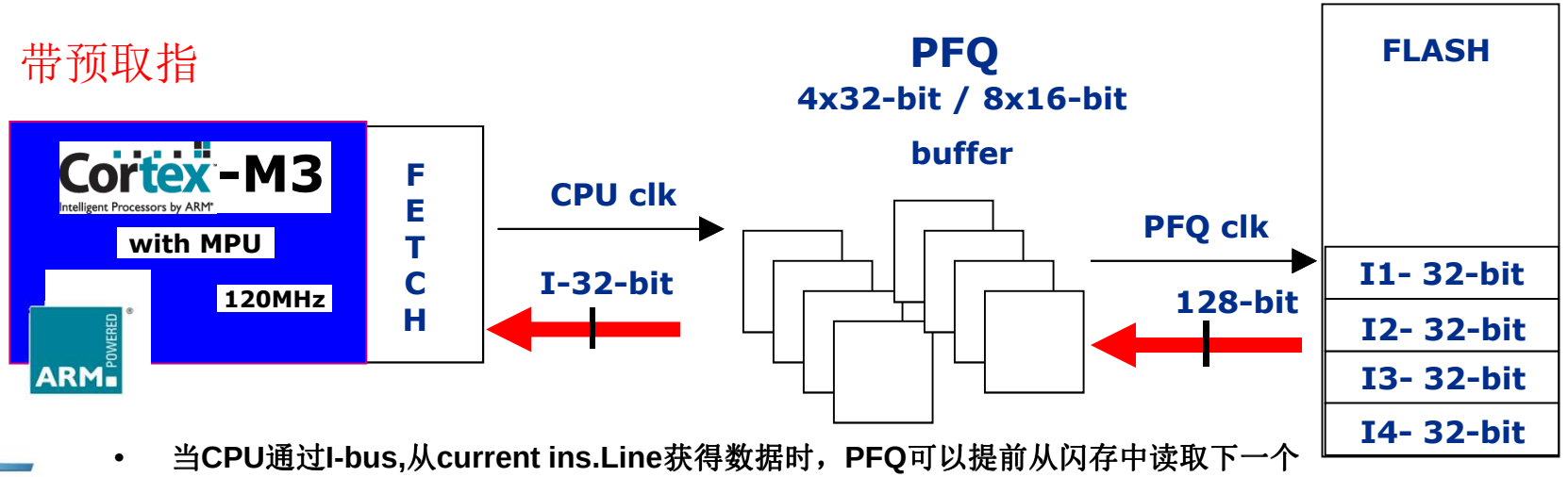
life.augmented

STM32闪存预取指

不带预取指



带预取指

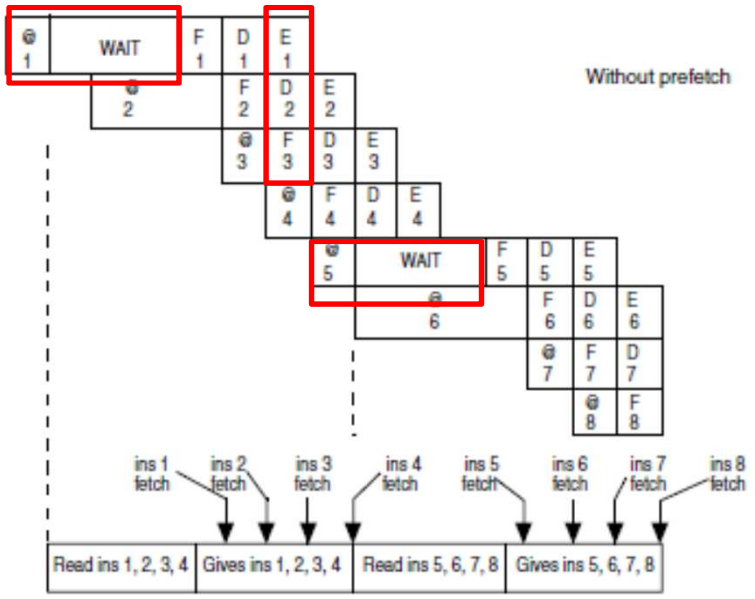


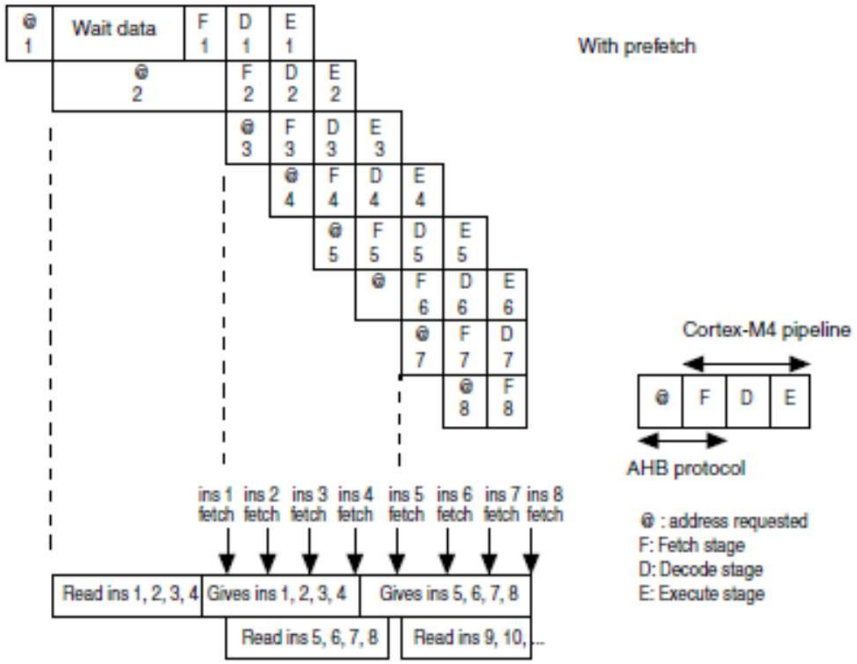
- 当CPU通过I-bus,从current ins.Line获得数据时，PFQ可以提前从闪存中读取下一个128bit数据，放在prefetch ins.Line
- 在执行顺序代码的情况下，CPU之后就无需等待了

life.augmented

预取指时的流水线示意图

- 假设顺序执行32位指令
- 读取FLASH的等待周期为3个周期
- 不带预取指和带预取指时流水线的运行情况比较

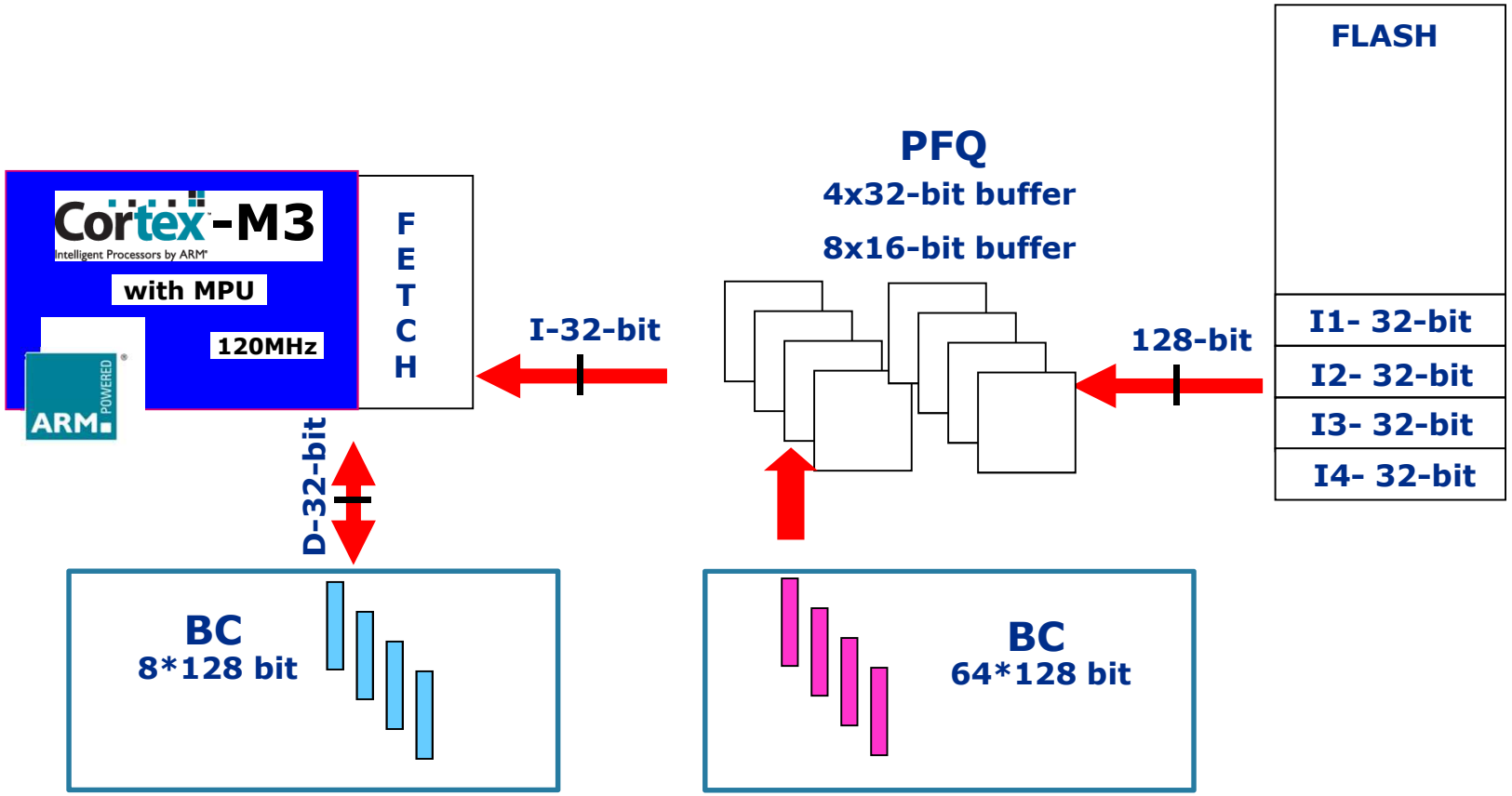




life.augmented

STM32片上闪存指令和数据缓存

- 如果指令发生跳转



储存经常用到的**literal pool**

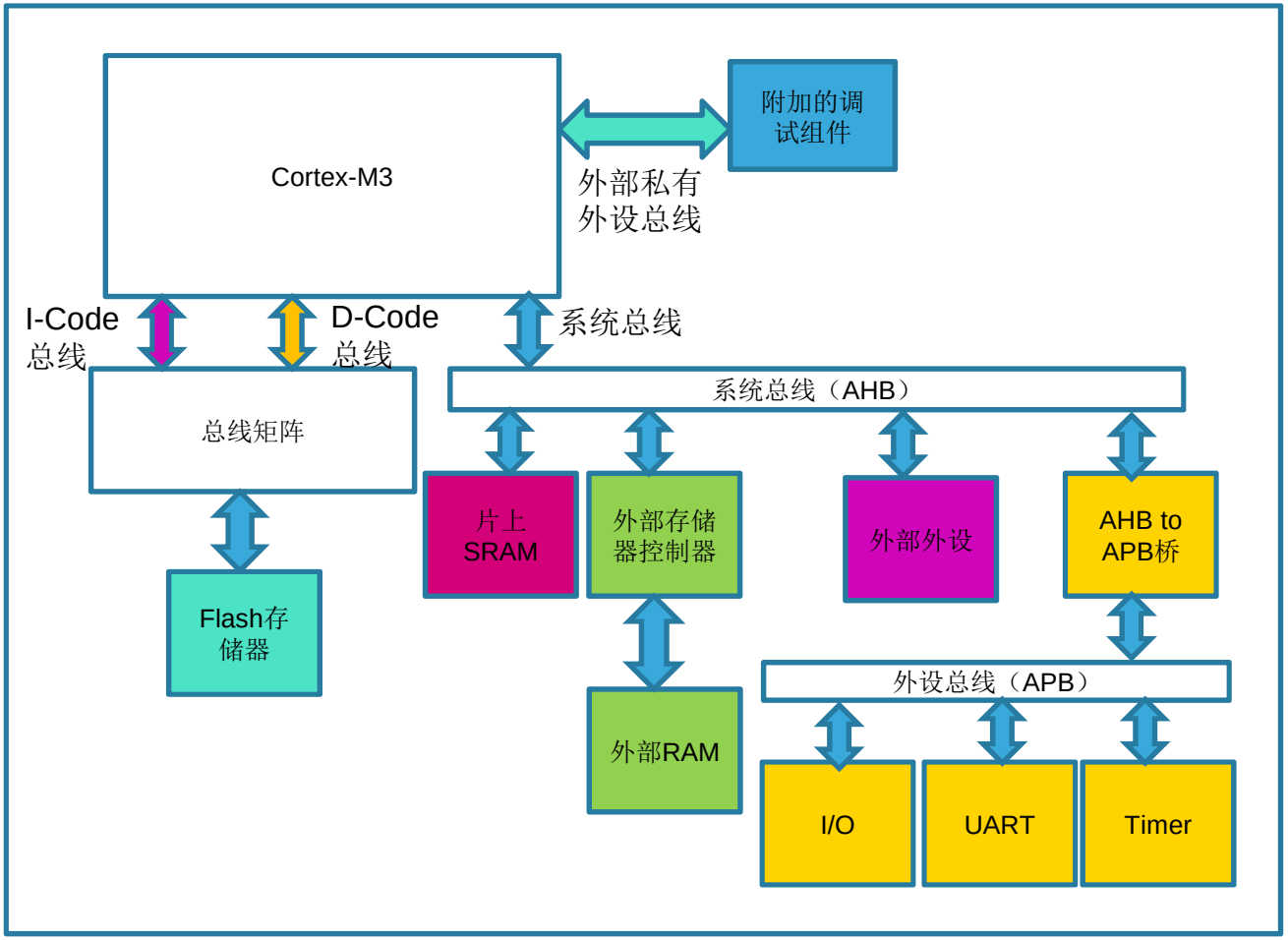
储存经常用到的指令，跳转发生又缓存击中的情况下，无需任何延迟地执行代码

- 存储器系统
- 编程模式
  - 寄存器组
  - 特权操作和双堆栈
  - 流水线
  - 总线结构
- 中断及其处理
- 低功耗模式
- 存储保护单元(MPU)

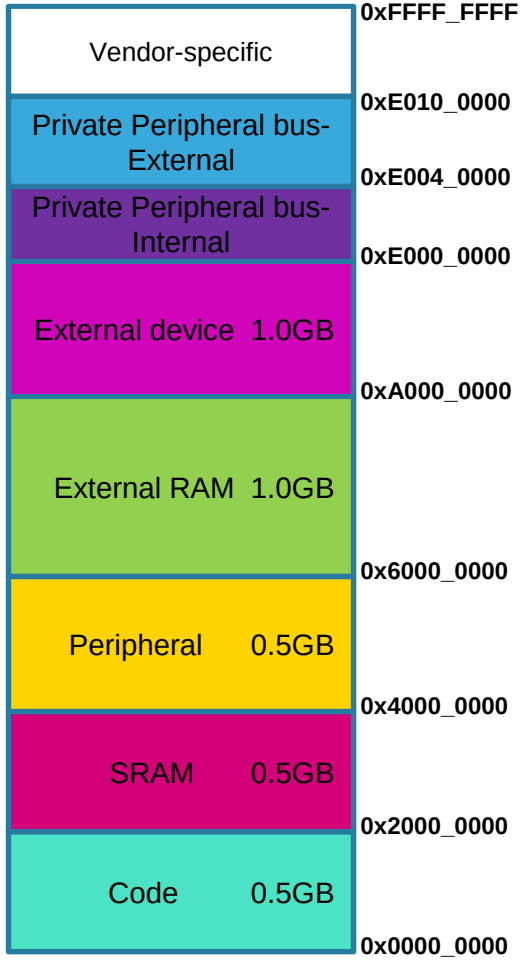
Cortex-M3总线接口

- I-Code 总线
  - 负责在0x0000_0000~0x1FFF_FFFF之间的取值操作。
  - 以字的长度执行
- D-Code总线
  - 负责在0x0000_0000~0x1FFF_FFFF之间的数据访问操作
- 系统总线
  - 负责在0x2000_0000~0xDFFF_FFFF和0xE010_0000~0xFFFF_FFFF之间的所有数据传送。
- 私有外设总线
  - 外部私有外设总线：0xE004_0000~0xE00F_FFFF,包括ETM,TPIU等。
  - 内部私有外设总线: 0xE000_0000~0xE003_FFFF，包括NVIC，MPU等

Cortex-M3总线连接



存储器空间划分



life.augmented

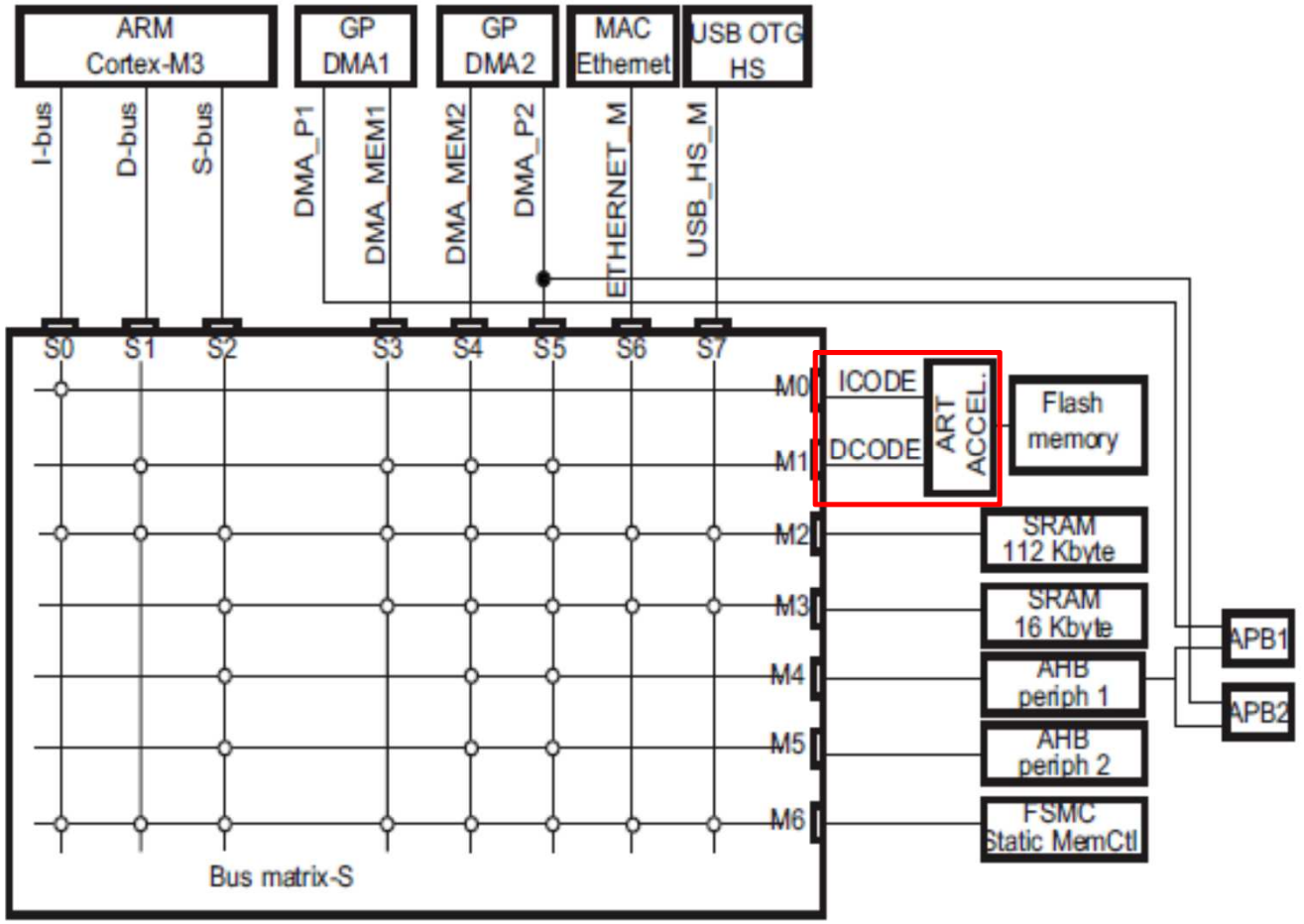
STM32的总线结构

8个AHB总线主设备

- Cortex-M3内核的指令总线(I-bus)
- Cortex-M3内核的数据总线(D-bus)
- Cortex-M3内核的系统总线
- DMA1和DAM2的存储器访问总线
- DMA2的外设总线
- 以太网DAM总线
- USB OTG HS DAM总线

7个AHB总线从设备

- 片上闪存的 指令和数据总线
- 主片上SRAM和副片上SRAM
- AHB1外设
- AHB2外设
- FSMC



life.augmented