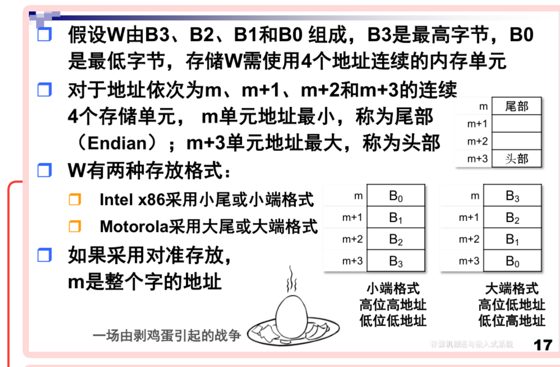
了解

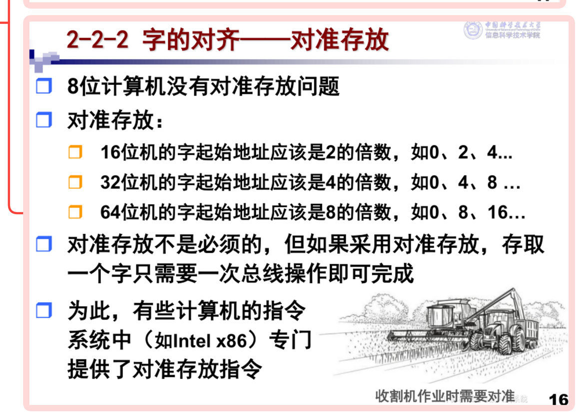
微处理器最小硬件系统概念
- 时钟源
- 下载接口
- 启动方式选择
- 复位电路
- 电源及滤波

嵌入式系统的交叉开发环境

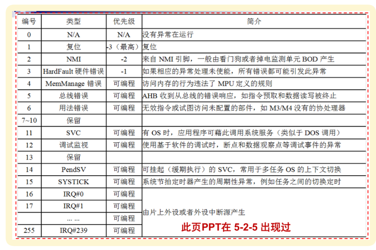
T32指令的汇编语法

小端和大端格式（基本概念）；字长与字的对齐



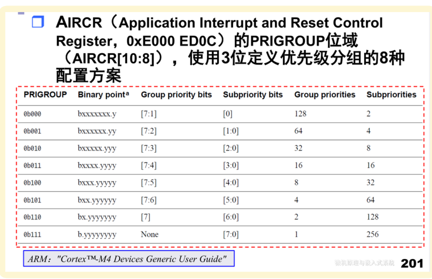


异常处理的基本过程，及异常优先级及优先级分组（概念）
- 异常类型（Exception types）

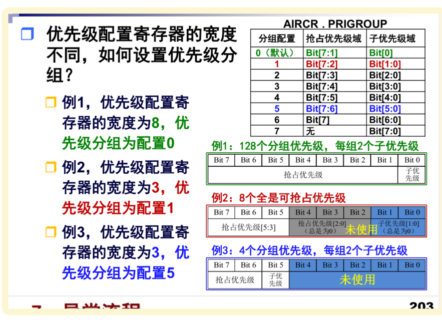


编号	类型	优先级	简介
0	N/A	N/A	没有异常在运行
1	复位	-3（最高）	复位
2	NMI	-2	来自 NMI 引脚，一般由看门狗或者掉电监测单元 BOD 产生
3	HardFault 硬件错误	-1	如果相应的异常处理未使能，所有错误都可能引发此异常
4	MemManage 错误	可编程	访问内存的行为违反了 MPU 定义的规则
5	总线错误	可编程	AHB 收到从总线的错误响应，如指令预取和数据读写被终止
6	用法错误	可编程	无效指令或试图访问未配置的部件，如 M0/M4 没有的协处理器
7~10	保留		
11	SVC	可编程	有 OS 时，应用程序可藉此调用系统服务（类似于 DOS 调用）
12	调试监视	可编程	使用基于软件的调试时，断点和数据观察点等调试事件的异常
13	保留		
14	PendSV	可编程	可挂起（缓期执行）的 SVC，常用于多任务 OS 的上下文切换
15	SYSTICK	可编程	系统节拍定时器产生的周期性异常，例如任务之间的切换定时
16	IRQ#0	可编程	由片上外设或者外设中断源产生
17	IRQ#1	可编程	
……	……	可编程	
255	IRQ#239	可编程	

- 2. 异常状态（Exception states）
  - 非激活状态 — Inactive — 异常既不在激活状态也不在挂起状态
  - 挂起状态 — Pending — 异常源发出了服务请求，正在等待处理
  - 激活状态 — Active — 正在接受处理器服务但未结束的异常（如果某异常处理程序被更高优先级的异常服务打断，则两个异常均处于激活状态）
  - 激活并挂起状态 — Active and pending — 异常正在接受处理器服务，而相同异常源又产生了异常请求
- 3. 异常处理程序（Exception handlers）
- 4. 异常向量表（Vector table） — 255个
- 5. 异常的优先级（Exception priorities） 6. 中断优先级分组（Interrupt priority grouping）
  - Cortex-M3/M4中，每一个中断都有一个8位的中断
  - 优先级的减少通过去除优先级配置寄存器的最低位（LSB）实现
  - 优先级寄存器3~8位又分为两部分（以8位为例）
    - 分组优先级（group priority），又称为抢占优先级（preempt priority）
    - 子优先级（subpriority）



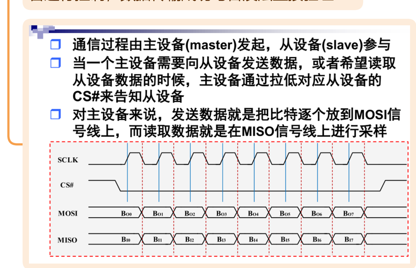
PRIGROUP	Binary point	Group priority bits	Subpriority bits	Group priorities	Subpriorities
0b000	bxxxxxxx.y	[7:1]	[0]	128	2
0b001	bxxxxxx.yy	[7:2]	[1:0]	64	4
0b010	bxxxxx.yyy	[7:3]	[2:0]	32	8
0b011	bxxxx.yyyy	[7:4]	[3:0]	16	16
0b100	bxxx.yyyyy	[7:5]	[4:0]	8	32
0b101	bxx.yyyyyy	[7:6]	[5:0]	4	64
0b110	bx.yyyyyyy	[7]	[6:0]	2	128
0b111	b.yyyyyyyy	None	[7:0]	1	256



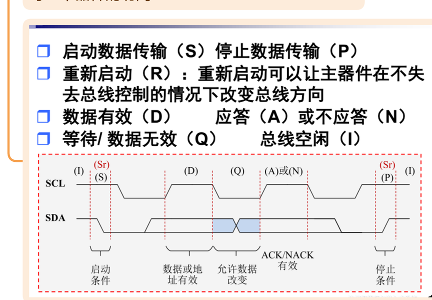
分组配置	抢占优先级域	子优先级域
0（默认）	Bit[7:1]	Bit[0]
1	Bit[7:2]	Bit[1:0]
2	Bit[7:3]	Bit[2:0]
3	Bit[7:4]	Bit[3:0]
4	Bit[7:5]	Bit[4:0]
5	Bit[7:6]	Bit[5:0]
6	Bit[7]	Bit[6:0]
7	无	Bit[7:0]

- 7. 异常流程（Exception entry and return）
  - 异常请求的接受
    - 处理器接受请求的条件
      - 处理器处于运行状态
      - 异常处于使能状态
      - 异常的优先级高于当前等级
      - 异常没有被屏蔽（如没有设置PRIMASK）
    - 注意，如果异常处理程序中出现了SVC指令，而该异常的优先级不低于SVC的优先级，就会触发硬件错误，从而进入硬件错误的处理程序
  - 异常进入流程
    - 异常进入流程包括如下操作
      - 多个寄存器的值和返回地址被压入当前使用的栈 — 若处理器处于线程模式且正在使用进程栈指针PSP，则PSP指向的堆栈区域就会用于该压栈过程，否则就会使用主栈指针MSP指向的堆栈区域
      - 从向量表中取出异常向量
      - 取出异常处理程序中的指令
      - 更新多个NVIC寄存器（后续介绍）和内核寄存器（PSR、LR、PC及SP）
    - 加快中断执行速度—压栈顺序
      - R0~R3、R12、LR、PC（返回地址）和 PSR共8个寄存器被压栈（注意：压栈顺序和栈帧结构不同）
        - 如果需要压栈保存FPU状态，则共有26字
        - 如果使能双字栈对齐，可能还会修改已入栈的PSR[9]
      - 压栈时为了尽快更新PC，首先压栈的是PC（返回地址）和PSR，出栈时为了尽快恢复处理器状态和返回主程序，出栈时也应该先出栈PSR和PC，咋办？ — 前述栈指针"…会在异常处理开始前自动调整"
  - 执行异常处理程序
    - 进入异常处理程序内部后，处理器进入处理模式，并运行于特权访问等级，栈操作使用MSP
    - 此过程中如果有更高优先级的异常产生，处理器会接受新的中断，当前正在执行的处理被更高优先级的处理抢占而进入挂起状态，此即异常嵌套
    - 若执行过程中产生的其他异常具有相同或更低的优先级，新产生的异常就会进入挂起状态，待当前异常处理完成后才可能被处理
    - 程序代码执行的返回指令会引起EXC_RETURN数值被加载到程序计数器中（PC），并触发异常返回机制
  - 异常返回
    - EXC_RETURN写入PC时，就会触发异常返回流程
    - 异常返回机制被触发后，进入异常期间被压入栈中的寄存器数值会被恢复到寄存器组中 — 因而多个NVIC寄存器和处理器内核中的寄存器（如PSR、SP和CONTROL）都会被更新
  - EXC_RETURN
    - 0xFFFFFFF1 — 返回至处理模式，使用MSP恢复非浮点状态信息
    - 0xFFFFFFF9 — 返回至线程模式，使用MSP恢复非浮点状态信息
    - 0xFFFFFFFD — 返回至线程模式，使用PSP恢复非浮点状态信息
    - 0xFFFFFFE1 — 返回至处理模式，使用MSP恢复浮点状态信息
    - 0xFFFFFFE9 — 返回至线程模式，使用MSP恢复浮点状态信息
    - 0xFFFFFFED — 返回至线程模式，使用PSP恢复浮点状态信息
- 加速中断处理速度
  - 咬尾中断
    - 中断#1正在服务，又出现中断#2，因优先级问题，中断#2被挂起
    - 优化
      - 中断#1结束后，读取中断#2的中断向量，立即为中断#2服务
      - 减少了中断#2的时延
  - 晚到中断
    - 当优先级较低的中断#n刚刚被响应，正在进行压栈，但是尚未取中断
    - 优化
      - 仍然压栈保护断点
      - 但是取ISR#m的中断向量
      - 提前为中断#m服务

SPI、I2C接口原理（大致传输过程）
- SPI
  - 需要至少4根线，事实上3根也可以（单向传输时）
    - MISO (Master Input Slave Output)，主设备数据输入，从设备数据输出
    - MOSI (Master Output Slave Input)，主设备数据输出，从设备数据输入
    - SCLK (Serial Clock)，时钟信号，由主设备产生
    - CS (Chip Select)，从设备使能信号，由主设备控制
  - SPI上可以挂载一个主设备和多个从设备
  - 任何时刻，一个主设备只与一个从设备进行通信，通信的从设备CS为低电平（有效）
  - 在不使用CS信号时，SPI上只能有一个主设备与一个从设备
  - SPI的缺点是没有应答机制，传输过程全都由主设备进行控制，数据传输成功与否没法直接验证

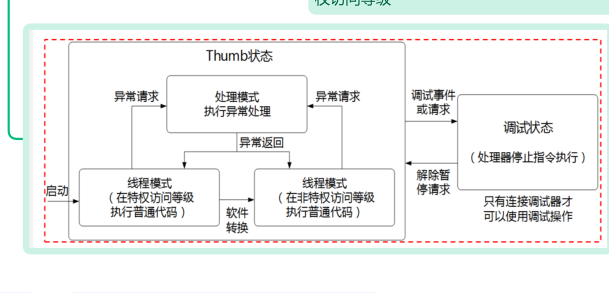


- I2C
  - 一条串行数据线SDA，一条串行时钟线SCL
  - 特点 — 同步方式、主从协议、半双工
  - I2C总线对发生在SDA信号线上的总线竞争进行仲裁，原理为: — 在检测到总线空闲(SCL和SDA均为高电平)后，拟使用总线的主机向SDA信号线发送数据(高电平或低电平)，每一位数据发送后随即检测SDA信号线电平是否与自身发送电平一致，若电平不符则竞争失败，自动关闭其输出
  - I2C总线标准对同类型的器件规定了一个固有的地址和可编程的地址，采用软件寻址方式，实现对每一个器件的访问 — 总线上每一个器件的地址必须是唯一的



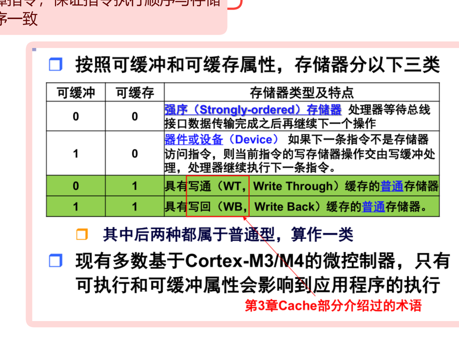
微架构的概念、哈佛结构的特点以及与冯·诺依曼结构的区别
- 计算机体系结构CA（Computer Architecture）
  - CA是对计算机系统的设计思想、逻辑特征、原理特征、结构特征和功能特征的一种抽象
  - 包括
    - 指令集体系结构ISA（Instruction Set Architecture）
    - 和微架构μarch（Microarchitecture）
    - 以及硬件实现（Hardware Implementation）
- 指令集体系结构ISA
  - ISA，描述软件如何使用硬件的一种规范和约定
  - 描述处理器指令及其功能、组织方式的规范称为指令集体系架构 ISA。
- 微架构μarch — ISA的硬件实现方式 — 即数字电路以何种方式来实现处理器的各种功能，包括运算器、控制器、流水线、超标量和存储系统结构等内容，也就是计算机的组织和实现技术
- 哈佛结构
  - 哈佛结构属于一种并行体系结构
  - 程序和数据存储在不同的存储空间中，每个存储器独立编址，使用两套总线单独访问
  - 可提高数据吞吐率，消除流水线上取指和取操作操作的资源相关，处理器性能高于冯·诺依曼结构
  - 哈佛结构较为复杂，实现难度较大，早期较少采用

Cortex-M3/M4处理器2种操作状态，2种操作模式，2种访问等级（切换原理）
- 操作状态（运行状态）
  - Thumb状态 — Thumb状态，执行Thumb指令的状态 — 由于 Cortex-M系列处理器不支持ARM指令集，所以没有ARM状态
  - 调试状态 — 调试状态，当处理器被暂停后就会进入调试状态 — 同时会停止指令执行。例如通过调试器或触发断点后
- 操作模式（运行模式）
  - 处理模式 — 处理模式（Handler mode），类似于经典处理器中的异常模式 — 执行的是中断服务程序（ISR），此时处理器具有特权访问等级
  - 线程模式 — 线程模式（Thread mode），除处理模式以外的模式
    - 特权线程模式 — 又分为特权线程模式（类似于经典处理器sys模式）
    - 非特权线程模式 — 和非特权线程模式（类似于经典处理器usr模式
- 特权与非特权的转换
  - 特权线程模式切换到非特权线程模式 — 系统启动后处于特权线程模式，在此模式下，可以通过对特殊寄存器CONTROL的写操作，将处理器从特权线程模式切换到非特权线程模式
  - 非特权线程模式切换到特权线程模式
    - 无论是特权还是非特权线程模式，在异常响应时都进入处理模式，并具有特权访问等级
    - 非特权线程模式可以利用这种机制，在处理模式中修改CONTROL寄存器，从异常返回时变成特权访问等级



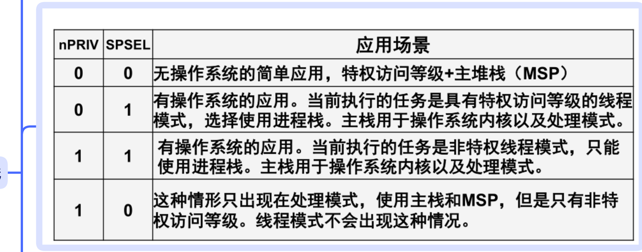
存储器屏障指令
- 存储器屏障指令可用于
  - 确保存储器访问顺序
  - 确保存储器访问和另外一个处理器操作之间的顺序
  - 确保系统配置发生在后序操作之前
- Cortex-M3/M4支持三种存储器屏障指令
  - DMB：数据存储器屏障 — 确保执行新的存储器操作之前，所有的存储器操作都已经完成
  - DSB：数据同步屏障 — 确保下一条指令执行前，所有的存储器访问都已经完成
  - ISB：指令屏障 — 清空流水线，确保在执行新的指令前，之前所有的指令都已经完成

存储器访问属性
- Cortex-M3/M4处理器的存储器访问属性
  - 可缓冲（Bufferable）
    - 存储器写操作可能无法在一个流水线周期内完成，如果配置了写缓存，剩下的工作交由写缓冲执行，处理器继续执行下一条指令
    - 写入加速
  - 可缓存（Cacheable）
    - 读存储器所得到的数据可被复制到缓存，下次再访问时可以从缓存中取出这个数值从而加快程序执行
    - 读取加速
  - 可执行（Executable） — 处理器可以从本存储器区域取出并执行程序代码
  - 可共享（Sharable）
    - 存储器区域的数据可被多个总线主设备共用
    - 存储器系统需要在不同总线主设备之间确保可共享存储器区域数据的一致性
- 可缓冲（Bufferable）的代价
  - 虽然存储器系统不会改变指令执行顺序，但是，并不能确保多条顺序执行指令中的存储器访问完成顺序与指令顺序一致
  - 使用存储器屏障指令，保证指令执行顺序与存储器访问结束顺序一致



可缓冲	可缓存	存储器类型及特点
0	0	强序（Strongly-ordered）存储器 处理器等待总线接口数据传输完成之后再继续下一个操作
1	0	器件或设备（Device） 如果下一条指令不是存储器访问指令，则当前指令的写存储器操作交由写缓冲处理，处理器继续执行下一条指令。
0	1	具有写通（WT，Write Through）缓存的普通存储器
1	1	具有写回（WB，Write Back）缓存的普通存储器。

堆栈的原理，Cortex-M3/M4处理器的堆栈模型（满递减）及双堆栈结构
- 堆栈是一种特殊的数据结构 — 一种只能在一端进行插入和删除操作的线型表
- 堆栈的数据存取操作按照后进先出(LIFO)的原则，并通过堆栈指针指示当前的操作位置
- 堆栈的作用
  - 保护断点，以便在异常/中断返回后，处理器能够从断点处继续运行 — 保护断点即，在异常/中断响应时，保存被中断程序的下一条指令的地址，以及处理器状态寄存器的内容
  - 保存现场，以便当异常返回或者函数或子程序处理结束时，可以恢复现场 — 异常/中断服务程序，或者正在执行的函数或者子程序，如果需要使用某些寄存器，可以使用堆栈保存这些原来的内容
  - 实现主程序与函数或者子程序之间的参数传递 — 函数或者过程调用有多种参数传递方式，堆栈传递是一种最安全的方式，并且对参数数量几乎没有限制
  - 用于存储局部变量
- 堆栈类型
  - 按照堆栈区在存储器中的地址增长方向
    - 递增栈（Ascending Stack） — 向堆栈写入数据时，堆栈区由低地址向高地址生长
    - 递减栈（Descending Stack） — 向堆栈写入数据时，堆栈区由高地址向低地址生长
  - 按照堆栈指针SP所指示的位置
    - 满堆栈（Full Stack） — 堆栈指针SP始终指向栈顶元素，也就是指向堆栈最后一个已使用的地址
    - 空堆栈（Empty Stack） — SP始终指向下一个将要放入元素的位置，也就是指向堆栈的第一个没有使用的地址或者空位置
- 4种基本堆栈类型
  - 满递增（FA）： — SP指向最后压入的数据，且由低地址向高地址生长
  - 满递减（FD）： — SP指向最后压入的数据，且由高地址向低地址生长
  - 空递增（EA）： — SP指向下一个可用空位置，且由低地址向高地址生长
  - 空递减（ED）： — SP指向下一个可用空位置，且由高地址向低地址生长
- Cortex-M系列处理器只能使用满递减（FD）类型
- 双堆栈
  - Cortex-M3/M4的双堆栈设计有MSP和PSP两个堆栈指针，分别服务于不同的操作模式和特权访问等级
  - CONTROL寄存器中nPRIV和SPSEL的不同组合，两个堆栈共有4种场景，其中前三种比较常见



nPRIV	SPSEL	应用场景
0	0	无操作系统的简单应用，特权访问等级+主堆栈（MSP）
0	1	有操作系统的应用。当前执行的任务是具有特权访问等级的线程模式，选择使用进程栈。主栈用于操作系统内核以及处理模式。
1	1	有操作系统的应用。当前执行的任务是非特权线程模式，只能使用进程栈。主栈用于操作系统内核以及处理模式。
1	0	这种情形只出现在处理模式，使用主栈和MSP，但是只有非特权访问等级。线程模式不会出现这种情况。

  - 如果使用双堆栈，应通过MPU在SRAM中建两个区域
    - 一个定义为特权级，其中一部分用作主栈存储区
    - 另一个定义为非特权级，其中一部分用于进程栈
  - 注意：Cortex-M3/M4的堆栈是满递减类型，因此两个栈指针的初始值应该是两个区域的最大地址
  - 如果采用双字对齐，栈顶应位于双字边界上，MSP和PSP最低3位为000

位段（位带）操作
- 位段（也称位带）操作：一次存储器操作只访问一个位
- Cortex-M3/M4在SRAM区和片上外设区，各有一个位段区和位段别名区
  - SRAM区域的最低1MB（0x2000 0000 ~ 0x200F FFFF）
  - 外设区域的最低1MB（0x4000 0000 ~ 0x400F FFFF）
- 位段区域中特定存储单元的某一位，映射为一个位段的别名地址（一个字）
- 一个字的32个比特被映射到32个位段别名地址（32个字）
- 位段操作的其他优点
  - 操作的原子性：操作过程不会被其它事务打断
  - 简化转移决断过程
- C程序实现位段操作
  - C编译器本身不支持位段操作
  - 但是可以编程实现位段操作
- 举例
  - 0x4000 0000 第0位 — 0x4200 0000
  - 0x4000 0000 第3位 — 0x4200 000C
  - 0x4000 1000 第3位 — 0x4202 000C — 0x1000 * 0x20 + 2x4200 0000 + 0x0C — 0x20 就是十进制的32，意思是扩展32倍大小