





中断／异常的响应序列

当CM3开始响应一个中断时，会在它看不见的体内奔涌起三股暗流：

🡺 入栈： 把8个寄存器的值压入栈

🡺 取向量：从向量表中找出对应的服务程序入口地址

🡺 更新寄存器 : 选择堆栈指针MSP/PSP，更新堆栈指针SP，更新连接寄存器LR，更新程序计数器PC



表 3.3 Cortex‐M3 的 CONTROL 寄存器

位 功能

CONTROL[1] 堆栈指针选择

0=选择主堆栈指针 MSP（复位后缺省值）

1=选择进程堆栈指针 PSP

在线程或基础级（没有在响应异常——译注），可以使用 PSP。在 handler 模式下，

只允许使用 MSP，所以此时不得往该位写 1。

CONTROL[0]

0=特权级的线程模式

1=用户级的线程模式

Handler 模式永远都是特权级的

程序状态寄存器（PSRs 或曰 PSR）

程序状态寄存器在其内部又被分为三个子状态寄存器：

z 应用程序 PSR（ APSR）

z 中断号 PSR（ IPSR）

z 执行 PSR（ EPSR）

===============================================================

表 4.2 16 位数据操作指令

名字 功能

ADC 带进位加法

ADD 加法

AND 按位与（原文为逻辑与，有误——译注）。这里的按位与和 C 的”&”功能相同

ASR 算术右移

BIC 按位清 0（把一个数跟另一个无符号数的反码按位与）

CMN 负向比较（把一个数跟另一个数据的二进制补码相比较）

CMP 比较（比较两个数并且更新标志）

CPY 把一个寄存器的值拷贝到另一个寄存器中

EOR 近位异或

LSL 逻辑左移（如无其它说明，所有移位操作都可以一次移动多格——译注）

LSR 逻辑右移

MOV 寄存器加载数据，既能用于寄存器间的传输，也能用于加载立即数

MUL 乘法

MVN 加载一个数的 NOT 值（取到逻辑反的值）

NEG 取二进制补码

ORR 按位或（原文为逻辑或，有误——译注）

ROR 圆圈右移

SBC 带借位的减法

SUB 减法

TST 测试（执行按位与操作，并且根据结果更新 Z）

REV 在一个 32 位寄存器中反转字节序

REVH 把一个 32 位寄存器分成两个 16 位数，在每个 16 位数中反转字节序

REVSH 把一个 32 位寄存器的低 16 位半字进行字节反转，然后带符号扩展到 32 位

SXTB 带符号扩展一个字节到 32 位

SXTH 带符号扩展一个半字到 32 位

UXTB 无符号扩展一个字节到 32 位

UXTH 无符号扩展一个半字到 32 位

表 4.3 16 位转移指令

名字 功能

B 无条件转移

B<cond> 条件转移

BL 转移并连接。用于呼叫一个子程序，返回地址被存储在 LR 中

CBZ 比较，如果结果为 0 就转移（只能跳到后面的指令——译注）

CBNZ 比较，如果结果非 0 就转移（只能跳到后面的指令——译注）

IT If‐Then

表 4.4 16 位存储器数据传送指令

名字 功能

LDR 从存储器中加载字到一个寄存器中

LDRH 从存储器中加载半字到一个寄存器中

LDRB 从存储器中加载字节到一个寄存器中

LDRSH 从存储器中加载半字，再经过带符号扩展后存储一个寄存器中

LDRSB 从存储器中加载字节，再经过带符号扩展后存储一个寄存器中

STR 把一个寄存器按字存储到存储器中

STRH 把一个寄存器存器的低半字存储到存储器中

STRB 把一个寄存器的低字节存储到存储器中

LDMIA 加载多个字，并且在加载后自增基址寄存器

STMIA 加载多个字，并且在加载后自增基址寄存器

PUSH 压入多个寄存器到栈中

POP 从栈中弹出多个值到寄存器中

表 4.5 其它 16 位指令

名字 功能

SVC 系统服务调用

BKPT 断点指令。如果调试被使能，则进入调试状态（停机）。或者如果调试监视器异常被

使能，则调用一个调试异常，否则调用一个 fault 异常

NOP 无操作

CPSIE 使能 PRIMASK(CPSIE i)/ FAULTMASK(CPSIE f)——清 0 相应的位

CPSID 除能 PRIMASK(CPSID i)/ FAULTMASK(CPSID f)——置位相应的位

表 4.6 32 位数据操作指令

名字 功能

ADC 带进位加法

ADD 加法

ADDW 宽加法（可以加 12 位立即数）

AND 按位与（原文是逻辑与，有误——译注）

ASR 算术右移

BIC 位清零（把一个数按位取反后，与另一个数逻辑与）

BFC 位段清零

BFI 位段插入

CMN 负向比较（把一个数和另一个数的二进制补码比较，并更新标志位）

CMP 比较两个数并更新标志位

CLZ 计算前导零的数目

EOR 按位异或

LSL 逻辑左移

LSR 逻辑右移

MLA 乘加

MLS 乘减

MOVW 把 16 位立即数放到寄存器的底 16 位， 高 16 位清 0

MOV 加载 16 位立即数到寄存器（其实汇编器会产生 MOVW——译注）

MOVT 把 16 位立即数放到寄存器的高 16 位， 低 16 位不影响

MVN 移动一个数的补码

MUL 乘法

ORR 按位或（原文为逻辑或，有误——译注）

ORN 把源操作数按位取反后，再执行按位或（原文为逻辑或，有误——译注）

RBIT 位反转（把一个 32 位整数先用 2 进制表达，再旋转 180 度——译注）

REV 对一个 32 位整数做按字节反转

REVH/REV16 对一个 32 位整数的高低半字都执行字节反转

REVSH 对一个 32 位整数的低半字执行字节反转，再带符号扩展成 32 位数

ROR 圆圈右移

RRX 带进位的逻辑右移一格（最高位用 C 填充，且不影响 C 的值——译注）

SFBX 从一个 32 位整数中提取任意的位段，并且带符号扩展成 32 位整数

SDIV 带符号除法

SMLAL 带符号长乘加（两个带符号的 32 位整数相乘得到 64 位的带符号积，再把积加到另一个带符号 64 位整数中）

SMULL 带符号长乘法（两个带符号的 32 位整数相乘得到 64 位的带符号积）

SSAT 带符号的饱和运算

SBC 带借位的减法

SUB 减法

SUBW 宽减法，可以减 12 位立即数

SXTB 字节带符号扩展到 32 位数

TEQ 测试是否相等（对两个数执行异或，更新标志但不存储结果）

TST 测试（对两个数执行按位与，更新 Z 标志但不存储结果）

UBFX 无符号位段提取

UDIV 无符号除法

UMLAL 无符号长乘加（两个无符号的 32 位整数相乘得到 64 位的无符号积，再把积加到另一

个无符号 64 位整数中）

UMULL 无符号长乘法（两个无符号的 32 位整数相乘得到 64 位的无符号积）

USAT 无符号饱和操作（但是源操作数是带符号的——译注）

UXTB 字节被无符号扩展到 32 位（高 24 位清 0——译注）

UXTH 半字被无符号扩展到 32 位（高 16 位清 0——译注）

表 4.7 32 位存储器数据传送指令

名字 功能

LDR 加载字到寄存器

LDRB 加载字节到寄存器

LDRH 加载半字到寄存器

LDRSH 加载半字到寄存器，再带符号扩展到 32 位

LDM 从一片连续的地址空间中加载多个字到若干寄存器

LDRD 从连续的地址空间加载双字（ 64 位整数）到 2 个寄存器

STR 存储寄存器中的字

STRB 存储寄存器中的低字节

STRH 存储寄存器中的低半字

STM 存储若干寄存器中的字到一片连续的地址空间中

STRD 存储 2 个寄存器组成的双字到连续的地址空间中

PUSH 把若干寄存器的值压入堆栈中

POP 从堆栈中弹出若干的寄存器的值

表 4.8 32 位转移指令

名字 功能

B 无条件转移

BL 转移并连接（呼叫子程序）

TBB 以字节为单位的查表转移。从一个字节数组中选一个 8 位前向跳转地址并转移

TBH 以半字为单位的查表转移。从一个半字数组中选一个 16 位前向跳转的地址并转移

表 4.9 其它 32 位指令

LDREX 加载字到寄存器，并且在内核中标明一段地址进入了互斥访问状态

LDREXH 加载半字到寄存器，并且在内核中标明一段地址进入了互斥访问状态

LDREXB 加载字节到寄存器，并且在内核中标明一段地址进入了互斥访问状态

STREX 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的字

STREXH 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的半字

STREXB 检查将要写入的地址是否已进入了互斥访问状态，如果是则存储寄存器的字节

CLREX 在本地的处理上清除互斥访问状态的标记（先前由 LDREX/LDREXH/LDREXB 做的标记）

MRS 加载特殊功能寄存器的值到通用寄存器

MSR 存储通用寄存器的值到特殊功能寄存器

NOP 无操作

SEV 发送事件

WFE 休眠并且在发生事件时被唤醒

WFI 休眠并且在发生中断时被唤醒

ISB 指令同步隔离（与流水线和 MPU 等有关——译注）

DSB 数据同步隔离（与流水线、 MPU 和 cache 等有关——译注）

DMB 数据存储隔离（与流水线、 MPU 和 cache 等有关——译注）

0xE0000000 至 0xE00FFFFF











