【pc 嵌入式系统的定义】

【国内普遍认同定义】以特定应用为中心、以计算机技术为基础，软硬件可裁剪，适应应用系统对功能、可靠性、功耗、成本、体积等严格约束的专用计算机系统。

【微软定义为】完成某一特定功能、或是使用某一特定嵌入式应用软件的计算机或计算装置。嵌入式系统，是区别与第一类常见的计算机的第二类计算设备，它是嵌入式到其他系统中的。

冯诺依曼和哈佛结构的区别

【冯·诺依曼结构】也称普林斯顿结构，是一种将程序指令存储器和数据存储器合并在一起的存储器结构。特点：数据，指令存放在统一的存储器中；程序当作数据；存储系统的利用；单一的存储器接口（ARM7）。

【哈佛结构】是一种将程序指令存储和数据存储分开的存储器结构。特点：数据，指令存放在不同的存储器中；高性能实现具有优势；带宽增加。（例 ATMEL 公司的 AVR 系列和安谋公司的 ARM9、ARM10 和 ARM11 以及 M3。）

嵌入式微处理器原理-储存计算

处理器是执行存储器指令的有限状态机。每一条指令规定了状态的变化，并指定随后执行哪一条指令

存储程序处理器：把指令和数据存放在存储器中

*计算机处理器：不同时间运行不同的程序

*嵌入式微处理器：不同时间运行固定的程序，可用于不同的应用范围

嵌入式系统软件架构

复杂嵌入式系统：硬件+操作系统+中间件+应用程序

简单嵌入式系统：硬件+应用软件

嵌入式系统硬件架构（最小系统包含的内容）

电源电路、时钟（晶振）电路、复位电路、JTAG 测试接口、存储器、处理器。

SoC 的基本概念，什么是 IP

System on chip 把一个处理器和其他外围支持设备集成到一块芯片上（降低耗电量、减少体积、丰富系统功能、提高速度、节省成本），IP 复用是 Soc 的基础。

IP 知识产权，我们把一个特定功能设置成一个模块，并提供接口方便其他模块调用的这种功能模块成为 IP（核）（是设计好并经过验证的集成电路功能单元）

处理器的一般架构，数据通路+控制逻辑

CPU 由运算器和控制器组成　　运算器：执行所有算数/逻辑运算

运算器的组成：#算术逻辑单元（ALU）#累加寄存器#状态标志寄存器#数据缓冲寄存器

控制器的组成：#指令部件（程序计数器/指令寄存器/指令译码器）#时序部件（脉冲源/节拍信号发生器/启停控制逻辑）#微操作信号发生器#中断控制逻辑

【数据通路】（CPU 内部各部件之间传送数据的通路）并行的传送、存储、处理多位二进制数的部件都属于数据通路：累加器；程序计数器；ALU；指令，数据寄存器……

【控制逻辑】：对当前指令译码，产生数据通路的控制信号，实现数据通路中的数据输入和输出。指令部件/时序部件\微操作信号发生器\断控制逻辑

处理器流水线的概念，ARM7，ARM9 的流水线级别

通过增加指令吞吐率来改进性能，无助于减少单个任务的处理延时；多个不同任务同时操作，使用不同资源；加速比=流水线级数；速率受限于最慢流水级：（资源、数据、控制）相关将导致流水线暂停。ARM7 三级流水线（取指-译码-执行），ARM9 五级流水线（取指-译码-执行-访存-回写）

ARM 微处理器体系结构的发展史

#Version4（ARM7，9）THUMB 指令集；增加了特权模式；PC 距离当前指令为 2 个字，三级流水线；这是第一个正式结构；#Version5 改进了 ARM，THUMB 的交互执行；增加了软件断点；加强了对协处理器的支持；#Version6 增加了并行处理能力；增加了 SIMD 指令（音视频）　ARM 指令集：32 位，THUMB 指令集：16 位，Jazelle 指令集：8 位

ARM RISC 处理器内核的特点

#一个大的、统一的寄存器文件；#通过 Load/Store 在寄存器和内存中进行数据传递，数据处理的操作只针对寄存器的内容，而不直接对存储器进行操作。#简单的寻址模式，所有 Load/Store 的地址都只由寄存器内容和指令域决定#统一和固定长度的指令域，简化了指令的译码. #此外，ARM 体系结构还提供#每一条数据处理指令都对算术逻辑单元（ALU）和移位器控制，以实现对 ALU 和移位器的最大利用。#多寄存器装载和存储指令实现最大数据吞吐量#所有指令的条件执行实现了最快速的代码执行#地址自动增加和自动减少的寻址模式实现了程序循环的优化

ARM7 三级流水线的概念

#【取指】从存储器中取出指令并放进指令流。#【译码】指令译码，数据控制信号将准备下一个周期。这一阶段，指令拥有译码逻辑，但不拥有数据通路。#【执行】指令拥有数据通路；寄存器组被读，一个操作数被移位，ALU 产生的结果被写回目标寄存器。

ARM7 TDMI 内核的主要组成部分（结构图），数据通路所包含的部分

①数据通路和运算单元：{#寄存器组//#地址寄存器#读数据寄存器#写数据寄存器//#乘法器#桶型移位器#32 位 ALU#地址累加器//#A BUS#B BUS#ALU BUS# INC BUS}

【寄存器组】用来存储处理器的状态。它有两个读端口和一个写端口，而 R15（PC 程序寄存器）则有三个读端口和一个写端口。

【桶形移位器】能使一个操作数移动或循环任意位。LSL（逻辑左移）（低位补 0，算术乘 2），LSR（逻辑右移），ASR（算术右移），ROR（循环右移），RRX（带扩展的循环右移）。

【ALU】完成指令集所需要的算术和逻辑功能。

【地址寄存器与累加器】选择或使用已有的存储器地址，在需要时产生下一个地址。

【数据寄存器】用来暂存输入或传出存储器的数据

②指令译码器和控制逻辑：分析指令的操作码是什么，以决定操作的性质和方法，根据条件的逻辑关系决定最后措施的控制。

内存映射的概念

如何划分不同的内存和 I/O 设备之间的地址空间(memory Map)。I/O 端口编址方法：存储器映射编制（I/O 端口和内存单元统一编址）（ARM 使用）；I/O 映射编址（分开编址）

两种处理器指令系统 RISC 和 CISC

【CISC(复杂指令集)处理器】20 世纪 70-80 年代，由于存储设备昂贵而缓慢，所以设计一条指令就完成一项任务，成为处理器设计的目标，复杂的 CISC 处理器指令多种多样，很容易就超过上千种指令#优点：减少了完成给定功能的指令数量，降低了存储空间访问次数#缺点：处理器本身非常复杂，控制和指令解码单元迟缓，芯片面积增加，功耗大

【RISC(精简指令集)处理器】随着微电子技术的发展，处理器的速度越来越快，存储器也越来越便宜，人们发现如果把指令集精简为少数必须的指令，处理器 简单而快速#优点：处理器结构简单，功耗低，面积小，#缺点：执行一个任务所需的指令增加，但是随着处理器性能提高，这个缺陷被弥补.

核心思想：由于运行速度的提高，可以把硬片的复杂性转移到语言编译器中去，硬件部分尽可能简单而快速；

印第安序（大端和小端）

ARM 体系中，字=32 位=4 字节=2 半字。数据 0x12345678；#处理器在存放多字节的数据时候，一般将低位字节存放在低地址单元，譬如 X86 系列的处理器；这种方法成为小印第安序(低端优先)；

在嵌入式系统中被广泛应用的 PowerPC 处理器中，则把低位字节放在高地址单元，这种方法成为大印第安序(高端优先)；0x78563421

ARM 微处理器的 7 种工作模式及其对应寄存器

#User 用户模式：处理器正常的程序执行状态（程序不能访问有些受保护的资源）；#FIQ 快速中断模式 ：用于高速数据传输或通道处理；#IRQ 中断模式 ：用于通用的中断处理；#Supervisor 管理模式：系统使用的保护模式；#Abort 数据访问中止模式：当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护；#Undef 未定义指令中止模式：当未定义的指令执行时进入该模式，可用于支持硬件协处理器的软件仿真；#System 系统模式 ：运行具有特权的操作系统任务（与 User 模式一样，但可以不受限制的访问任何资源）#在用户模式：应用程序运行在用户模式。应用程序可以产生异常处理，在异常处理过程中进行模式的切换。系统模式使用用户模式的寄存器组，但可以访问所有的系统资源。主要供操作系统任务使用。#异常模式：快速中断模式、外部中断模式、管理模式、中止模式、未定义模式。常用于处理中断或异常，以及需要访问受保护的系统资源等情况。#特权模式：绿色的+系统模式#对应寄存器：R0-R7 是共用的；FIQ 有自己的 R8-R12，其他模式共享 R8-R12；用户和系统模式共享 SP（R13）和 LR（R14），其余有各自的；R15（PC）和 CPSR 是共用的；除了 User 和系统模式（无），其他模式都有各自的 SPSR

31 个通用寄存器：程序计数器、堆栈及其他通用寄存器//6 状态寄存器//同时看到 17 个各模式寄存器 R0-R7 R15（pc）R16(CPSR)七种模式共用相同；R8-R12 除 FIQ 外共用；R13（sp） R14（lr）七种都不同，独自有自己的；SPSR：user 没有，其余六个有各自的

ARM 微处理器每个寄存器的含义和特征

#在汇编语言中寄存器 R0～R12 为保存数据或地址值的通用寄存器（32 位）。它们是完全通用的寄存器，不会被体系结构作为特殊用途，并且可用于任何使用通用寄存器的指令。#寄存器 R13 常作为堆栈指针（SP）。在 ARM 指令集当中，没有以特殊方式使用 R13 的指令或其它功能，只是习惯上都这样使用。但是在 Thumb 指令集中存在使用 R13 的指令。#R14 为链接寄存器（LR），在结构上有两个特殊功能：在每种模式下，模式自身的 R14 版本用于保存子程序返回地址；当发生异常时，将 R14 对应的异常模式版本设置为异常返回地址（有些异常有一个小的固定偏移量）。#寄存器 R15 为程序计数器（PC），它指向正在取指的地址。可以认为它是一个通用寄存器，但是对于它的使用有许多与指令相关的限制或特殊情况。如果 R15 使用的方式超出了这些限制，那么结果将是不可预测的。#1 个专用的当前【程序状态寄存器 】CPSR 。条件标志；中断使能标志；当前处理器的模式；#其它的一些状态和控制标志【程序状态保存寄存器】SPSR，每一种异常模式对应一个 SPSR 异常模式发生的时候用来存储当前程序的 CPSR。

ARM7 响应异常的一般步骤

ARM 处理器发生异常的时候：尽量完成当前指令（除了复位异常中止当前指令），然后脱离当前的指令序列处理器异常；间接和外部事件异常将占据当前序列中的指令；直接异常按照顺序执行

当异常产生时，ARM core：#拷贝 CPSR 到 SPSR_<mode>#设置适当的 CPSR 位：改变处理器状态进入 ARM 态；改变处理器模式进入相应的异常模式；设置中断禁止位禁止相应中断（如需要）#保存返回地址到 LR_<mode>#设置 PC 为相应的异常向量（将跳转地址存入 PC，实现跳转）。返回时，异常处理需要：#从 SPSR_<mode>恢复 CPSR#从 LR_<mode>恢复 PC（LR 减去偏移量存入 PC）#Note:这些操作只能在 ARM 态执行.

ARM 响应异常的一般步骤：1、保存处理器当前状态，中断屏蔽位及各条件标志位。这是通过将当前程序寄存器 CPSR 的内容保存到将要执行的异常对应的 SPSR 寄存器中实现的。各异常有自己的物理 SPSR 寄存器。2、设置当前程序状态寄存器 CPSR 中的相应位，禁止 IRQ 中断，当进入 FIQ 模式时，禁止 FIQ 中断。3、将寄存器 LR_mode 设置成返回地址。4、将程序计数器值（PC）设置成该异常的异常向量地址，从而跳转到相应的异常处理程序处执行。

上述响应过程用伪码描述为：R14_<Exception_Mode> = Return Address(保存返回地址);SPSR_<Exception_Mode> = CPSR(保存当前状态);CPSR[4:0] = Exception Mode Number(设置进入异常模式);CPSR[5] = 0(当运行于 ARM 工作状态时);If <Exception_Mode>== Reset or FIQ then(当响应 FIQ 异常时,禁止新的 FIQ 异常);CPSR[6] = 1 ;CPSR[7] = 1;PC = Exception Vector Address

ARM7 异常处理中寄存器的使用

在异常发生时处理器模式的改变意味着异常处理程序至少需要访问下列寄存器：#堆栈指针寄存器（SP_<mode>）#连接寄存器（LR_<mode>）#程序状态保存寄存器（SPSR_<mode>）#在 FIQ 异常处理中，5 个其他的通用寄存器（r8_FIQ 到 r12_FIQ）#其他的寄存器可以和发生异常之前的模式共用

异常处理程序必须确保其他的寄存器在退出异常处理程序时恢复到进入异常之前的值。这个可以通过在进入异常时把工作寄存器的值压入堆栈，在退出异常时再弹出堆栈来实现。

每种异常的特点以及它的返回方式和返回地址的计算

ARM 的 7 种异常：复位异常，SWI 异常，未定义指令异常，数据访问中止异常，指令预取中止异常，FIQ，IRQ #SWI 和未定义指令异常是由当前执行的指令产生的 ，当 SWI 和未定义指令异常产生时，程序计数器 PC 的值还未更新，它指向当前指令后面第 2 条指令 MOV PC，LR #当 IRQ 和 FIQ 异常产生时，程序计数器 PC 的值已经更新，它指向当前指令后面第 3 条指令。SUBS PC，LR ，#4 #当发生指令预取中止异常时，程序要返回到该有问题的指令处，重新读取并执行该指令。因此指令预取中止异常处理程序返回到产生该指令预取中止异常的指令处 SUBS PC，LR，#4 #数据访问中止异常是由数据访问指令产生的，返回时需要重新执行，当数据访问中止异常产生时，程序计数器 PC 的值已经更新，它指向当前指令后面的第 3 条指令。SUBS　PC，LR，#8

ARM7 启动后进入的模式和 R15 的值

系统进入管理模式、ARM 状态、PC（R15）指向 0x00000000 地址处

ARM 指令集的特点

#所有的 ARM 指令都是 32 位宽，在存储器中以 4 字节的边界对齐，THUMB 指令 16 位宽，2 字节边界对齐；#LOAD/STORE 架构，包含非常强大的多寄存器 Load 和 Store 指令；#指令可条件执行。

ARM 指令

● ARM 条件执行指令

EQ（相等），NE（不相等），LS（无符号数小于或等于），GE（带符号数大于或等于），LT（带符号数小于），GT（带符号数大于）。

● ARM 数据处理指令

[数据传送：MOV Rd，operand；Rd=operand]，[数据反传送：MVN Rd，operand；Rd=～operand]，[加法运算：ADD Rd，Rn，operand；Rd=Rn+operand]，[减法运算：SUB Rd，Rn，operand；Rd=Rn－operand]，[逆向减法：RSB Rd，Rn，operand；Rd=operand－Rn]，[逻辑与：AND]，[逻辑或：ORR]，[逻辑异或：EOR]，[位清除：BIC Rd，Rn，operand；Rd=Rn& (～operand)]，[比较：CMP Rn，operand；NCZVRn-operand]，[位测试：TST Rn，operand；NCZVRn&operand]，[相等测试：TEQ Rn，operand；NCZVRn^operand]，[32 位乘法：MUL Rd，Rm，Rs；Rd=Rm*Rs]。

● 数据处理指令第二个操作数的规则

若操作数是寄存器，可以选择移位。移位的值：5 bit 整数或另外一个寄存器的值

若操作数是立即数，则可以是 8 位立即数，范围是 0-255（立即数是一个 8 位的常数循环右移偶数位得到的），也可以是 32 位常数载入寄存器以后使用

● ARM 的跳转指令

[分支指令：B label；PClabel]，[带链接分支：BL label；LRPC-4,PClabel]，[带状态切换分支：BX Rm；PCRm，切换处理器状态]。

● LOAD/STORE 指令，寄存器和内存之间的操作

ARM 内核通过 LOAD 和 STORE 指令在寄存器和内存中进行数据传递，而数据处理的操作只针对寄存器的内容，而不直接对存储器进行操作。[LDR Rd，addressing；Rdaddressing]（存储器加载到寄存器），[STR Rd，addressing]（寄存器存到存储器）。

● 批量加载和存储指令（LDM/STM）（储存器和堆栈的操作方式）

LDM/STM 允许在存储器和 16 个寄存器之间传送数据：#寄存器的传送顺序不能任意指定#低地址的内容总是传送到低寄存器。在 ARM 中，往堆栈中保存数据的指令是 STMFD;[STMFD（Push）批量存储]，[LDMFD （Pop）批量加载]。例：STMFD sp!，{r4-r7，lr}；现场保存，将 r4－r7，lr 入栈；LDMFD sp!，{r4-r7，pc}^；恢复现场，异常处理返回。ARM 的堆栈操作由块传送指令来实现 STMFD（Push） 批量存储一满递减堆栈

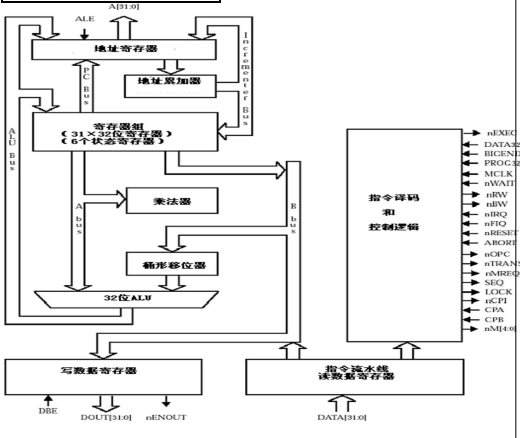
LDMFD（Pop）批量加载一满递减堆栈

其他 ARM7 指令

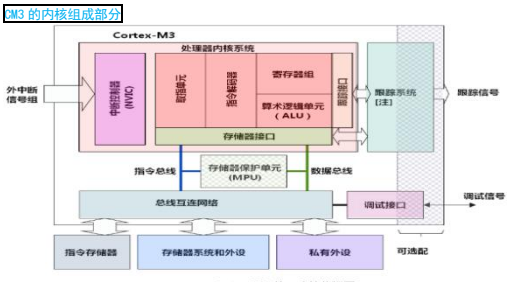
状态寄存器访问指令（MSR 状态寄存器写指令， 在 ARM 处理器中，只有 MSR 指令可以对状态寄存器 CPSR 和 SPSR 进行写操作；MRS 状态寄存器读指令，在 ARM 处理器中，只有 MRS 指令可以对状态寄存器 CPSR 和 SPSR 进行读操作）

基于 ARM 指令的简单编程

ARM7 TDMI 内核的主要组成部分



CM3 的内核组成部分



Cortex-M3 的一个简化视图

SWI 指令，SWI 指令异常的汇编处理程序内容

SWI 指令用于产生 SWI 异常，使得 CPU 模式变换到管理模式，并且将 CPSR 保存到管理模式的 SPSR 中，然后程序跳转到 SWI 异常入口。不影响条件码标志。指令格式：[SWI｛条件｝ 24 位的立即数]，用于产生软件中断，以便用户程序能调用操作系统的系统例程。例：SWI 0x02；该指令调用编号为 02 的系统例程。

SWI 指令（整个过程目的：在 SWI 异常中断处理程序中，根据是 ARM 状态还是 Thumb 状态来提取 SWI 指令中相应的立即数，然后进入 C 处理程序，执行完毕后再从 SWI 异常中退出返回主程序。）

该指令主要用于用户程序调用操作系统的系统服务，操作系统在 SWI 异常处理程序中进行相应的系统服务。

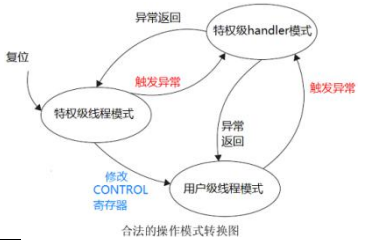
```
SWI_Handler
    STMFD   SP!, {R0-R3, R12, LR}   ; 现场保护
    MRS     R0, SPSR                ; 读取 SPSR
    STMFD   SP!, {R0}               ; 保存 SPSR
    TST     R0, #0x20               ; 测试 T 标志位
    LDRNEH  R0, [LR,#-2]            ; 若是 Thumb 指令，读取指令码(16 位)
    BICNE   R0, R0, #0xFF00         ; 取得 Thumb 指令的 8 位立即数（低 8 位）
    LDREQ   R0, [LR,#-4]            ; 若是 ARM 指令，读取指令码(32 位)
    BICEQ   R0, R0, #0xFF000000     ; 取得 ARM 指令的 24 位立即数（低 23 位）
    ...
    LDMFD   SP!, {R0-R3, R12, PC}^  ; SWI 异常中断返回
```

CM3 的两种模式和特权级别及其切换方式（有图）

线程模式和处理模式；访问级别：特权级和用户级

当处理器处在线程状态下时，既可以使用特权级也可以使用用户级；另一方面，handler 模式总是特权级的。在复位后，处理器进入线程模式＋特权级。

［线程模式］：在复位或异常返回时处理器进入线程模式；特权和用户（非特权）代码能够在线程模式下运行；MSP（主堆栈）和 PSP（进程堆栈）均可用。［处理模式］：出现异常时处理器进入处理模式；在处理模式中，所有代码都是特权访问的；始终使用 MSP。



合法的操作模式转换图

CM3 寄存器组成

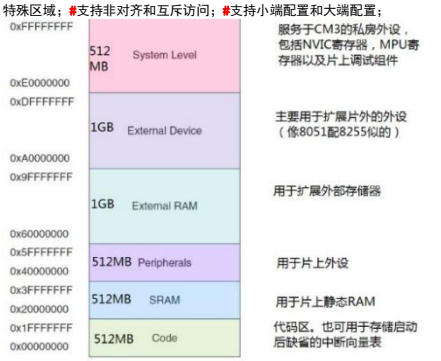
CM3 拥有通用寄存器 R0-R15 以及一些特殊功能寄存器。R0-R12 是最“通用目的”的，但是绝大多数的 16 位指令只能使用 R0-R7（低组寄存器），而 32 位的 Thumb-2 指令则可以访问所有通用寄存器。R13 有两个，分别是 MSP（主堆栈指针），PSP（进程堆栈指针），R14 是连接寄存器（LR）、R15 PC；特殊功能寄存器有预定义的功能，而且必须通过专用的指令来访问。特殊功能寄存器有#程序状态寄存器组（PSRs 或曰 xPSR）#中断屏蔽寄存器组（PRIMASK，FAULTMASK 以及 BASEPRI）#控制寄存器（CONTROL）

CM3 的流水线

带分支预测的 3 级流水线#【取指】从存储器装载一条指令并放进指令流/PC 值总是指向正在去取指的指令 #【译码】识别将要被执行的指令/在译码时进行分支预测，遇跳转指令也不会打断流水线#【执行】处理指令并将结果写回寄存器。

CM3 的存储器空间分布约定

#支持 4GB 存储空间；#存储器映射是预定义的；#支持“位绑定”（bit-band）操作——特殊区域；#支持非对齐和互斥访问；#支持小端配置和大端配置；



CM3 位带存储的原理

位带操作就是（1）对 32MB SRAM 别名区的访问映射为对 1MB SRAM 的 bit-band 区的访问.（2）对 32MB 外设别名区的访问映射为对 1MB 外设 bit-band 区的访问，bit_word_addr = bit_band_alias_base + (byte_offset ×32) + (bit_number×4)/1 别名存储器区中字的地址，它映射到某个目标位。/2 是别名区的基址，即起始地址。/3 是包含目标位的字节在位带区里的序号。/4 是位带区目标位所在字节中的位置（0-7）。

CM3 NVIC 的特点

Cortex-M3 处理器和嵌套向量中断控制器（NVIC）对所有异常按优先级进行排序并处理。所有异常都在处理模式中操作。#出现异常时，自动将处理器状态保存到堆栈中，并在中断服务程序（ISR）结束时自动从堆栈中恢复。在状态保存的同时取出向量快速地进入中断。#处理器支持末尾连锁（tail-chaining）中断技术，它能够在没有多余的状态保存和恢复指令的情况下执行背对背中断（back-to-back interrupt）。以下特性可使能高效的低延迟异常处理：##Cortex-M3 与 NVIC 之间采用紧耦合接口，通过该接口可以及早地对中断和高优先级的迟来中断进行处理。##异常数目可配置为 1～240。

异常优先级的数目可配置为 1～8 位（1～256 级）。##异常优先级可动态重新设置。##处理模式和线程模式具有独立的堆栈和特权等级。##ISR 调用采用 C/C++标准 ARM 体系结构过程调用标准（AAPCS）；##可屏蔽优先级以支持临界区。

NVIC 与处理器内核是紧密耦合的，这样可简化低延迟的异常处理。NVIC 主要特性包括：#外部中断可配置为 1～240 个 #优先级的位可配置为 3～8 位 #支持电平和边沿中断 #中断优先级可动态地重新配置 #优先级分组 #支持末尾连锁（tail-chaining）中断技术 #处理器状态在进入中断时自动保存，中断返回时自动恢复，不需要多余的指令。

功能：#可嵌套中断支持#向量中断支持#动态优先级调整支持（软件可在运行时期更改中断优先级。若某 ISR 中修改了自己所对应的中断优先级且这个中断又有新的实例处于挂起中，也不会自己打断自己，从而无重入风险）#中断延迟大大缩短（M3 为了缩短中断延迟，引入自动现场保护和恢复等措施，缩短中断嵌套时 ISR 间延迟）#中断可屏蔽。既可屏蔽优先级低于某阈值的中断/异常（设置 BASEPRI 寄存器），也可全体封杀（设置 PRIMASK 和 FAULTMASK 寄存器）。是为了让时间苛求的任务能在时限到来前完成，而不被干扰。

CM3 的异常类型#Cortex-M3 处理器将复位、不可屏蔽中断 NMI、外部中断 IRQ、故障都统一为异常。

根据触发源的不同，一般将异常分为同步异常和异步异常。同步异常是指与 CPU 当前执行的指令密切相关、造成 CPU 正常运行状态被中止的系统事件（或称内部事件），如指令未定义、指令预取中止、数据访问中止等。异步异常则是由于外部事件的触发而产生的，与 CPU 当前执行的指令无关，故被称为异步异常。复位即属于异步异常。

CM3 的中断响应序列　　#1、保护现场：依次把 xPSR，PC，LR，R12 及 R3 - R0 等 8 个寄存器内容由硬件自动压入当前堆栈/更新堆栈指针 SP（MSP/PSP）和 PSR 状态寄存器，如 IPSR 段的中断号/链接寄存器 LR 自动更新为 EXC_RETURN（特权级+工作模式）2、从向量表中找出对应的服务程序入口地址，更新程序计数器 PC //同时，NVIC 也会更新相关寄存器。例如，新响应异常的挂起位将被清除，同时其活动位将被置位。#故障（fault）是指令执行时由于错误的条件所导致的异常。同步故障是指当指令产生错误时就同时报告错误。异步故障则是指当指令产生错误时无法保证同时报告错误。总线 faults 存储器管理 faults 用法 faults 硬 fault

当 CM3 开始响应一个中断时，三个动作#入栈：把 8 个寄存器的值压入栈 r0-r3，r12，lr，pc，xPSR#取向量：从向量表中找出对应的服务程序入口地址#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

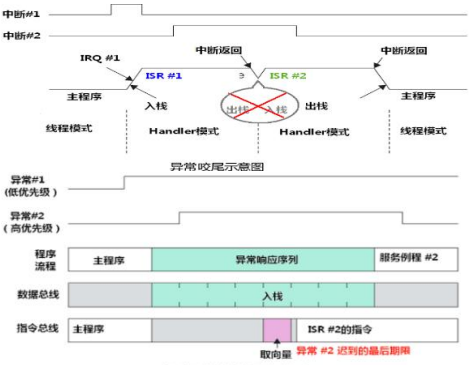
异常进入步骤：#8 个寄存器压栈#从向量表中读 SP#更新 PC#加载流水线#更新 LR

CM3 的咬尾中断和晚到异常机制及其图示（有图）　　执行了一次入栈 / 出栈操作------咬尾

咬尾中断（缩短延时）是指高优先级终端服务程序在执行过程中发生了低优先级中断，那么在高优先级中断服务程序执行完毕后直接去执行低优先级中断服务程序，低优先级中断服务程序执行完毕后才将高优先级中断压入堆栈的 8 个寄存器数据弹出，这中间就减少了高优先级中断出栈以及低优先级入栈的过程，另外还会有一种情况，低优先级中断先发生，而在低优先级中断压栈的过程中有发生了高优先级中断，这时高优先级中断就会抢占低优先级中断，如果高优先级中断再压栈然后执行，再出栈，低优先级中断执行再出栈，这样对这 8 个寄存器又是重复入栈出栈 2 次，做了无用功，cortex 内核

采用晚到中断机制避免和种问题发生，所谓晚到中断是指在低优先级中断压栈的过程中又发生了高优先级的中断，那么这个压栈过程就算是这个高优先级红短的压栈，压栈之后执行高优先级中断。CM3 的中断处理还有另一个机制，它强调了优先级的作用，这就是“晚到的异常处理”。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。

咬尾中断即继续使用上一个异常已经 PUSH 好的系统现场，在本次异常完成后才执行现场恢复。晚到的异常处理强调优先级的作用。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。



晚到异常的处理模式图

交叉编译的概念

【宿主机】开发机器(包含开发用软件编辑器,调试器 ,编译器,汇编器……)【目标机】程序运行的机器　宿主机=目标机 称为本地编译 。“交叉编译”是指 宿主机和目标机是不同的系统；由一个平台生成另外一个平台的二进制镜像文件的过程

嵌入式系统调试的需求

Run Control（运行控制）#Set watchpoints on data accesses（观察感兴趣的数据）#Set breakpoints on instructions（设置指令断点）#Single step through code（单步调试）State Control（状态控制）#Processor state（处理器状态）Read and write register values #System state（系统状态）Access to system memory||Download code Execution History（执行过程）#Execution trace information（跟踪程序信息）#Memory access history（存储器访问历史数据）

能够获取处理器数据总线、地址总线，处理器状态等信息，以便在选中的数据点（ watchpoints）和指令点（breakpoints）进入调试状态；能够获取处理器内核寄存器的信息；能够插入指令运行；能够访问存储器的内容；记录指令/数据的执行和访问记录

软件断点和硬件断点的概念

A hardware breakpoint：当程序再约定地址的取出指令时候，触发断点。A software breakpoint：软件断点是约定的一条指令被取指的时候触发程序断点（这条指令可以在任何地址）。

不同级别的调试模拟器

调试有四种基本方法：#模拟调试（Simulator）：指令集模拟器　操作系统模拟器 #软件调试（Debugger）#全仿真调试（Emulator）-在线调试#JTAG 调试（JTAG Debugger）

JTAG 在线调试的原理 JTAG 扫描链一共有四种操作：挂起、捕获、移位和更新；

当芯片处于调试状态时，BSR 将芯片与外围隔离。通过 BSR 可实现对芯片管脚的观察与控制。对于输入管脚，可通过对 BSR 的操作，将信号（数据）加载至管脚。对于输出管脚，可通过 BSR 将管脚上的输出信号“捕获”。BSR 可以相互连接起来，在芯片周围形成一个边界扫描链。

ARM 集成开发环境的构成（编译器，编辑器，连接器，调试器）

编辑器：用于编写 C 或 C++代码或者汇编代码的软件，一般类似文本编辑器。编译器：依据指令集架构 ISA 规定，把我们书写的高级语言代码或汇编语言代码翻译成机器可以识别的 0、1 数据（俗称机器码）。链接器（Linker）是把不同部分的代码和数据，收集、组合成为一个可加载、可执行的文件。调试器：调试器的工作原理是基于中央处理器的异常机制，对运行的代码进行断点设置，从而观察内存中的信息、反汇编信息、堆栈信息、寄存器信息。

ARM 编译器的概念/连接器的 2 个作用

【编译器】编译器就是将“高级语言”翻译为“机器语言（低级语言）”的程序

【链接器】链接编译后得到的目标文件和相应的运行时库，生成可执行的映像文件；也可以把一部分目标文件链接为新的目标文件；指定代码/数据在内存中的位置；生成被链接文件的调试信息和相互间的引用信息；

【链接器】功能：［分组信息］决定如何将各个输入端组织为输出段和域，链接器按照下面的顺序组织输入段：只读代码段，只读数据段，可读写代码段，非零初始化的数据段；［定位信息］决定各个域在存储空间中的位置，两种方法：对于比较简单的映像文件采用命令行的方式，ADS；对于比较复杂的使用专门的配置文件。

ARM 中段的概念(RO RW ZI)

段的概念：文件是计算机操作的基本单位，而段则是链接器操作的基本单位。一个文件中可以包含一个或多个段。对于链接器来说，它不关心有多少个输入文件，而只关心有多少个输入段。链接器的输入段其来源可以有两种：一种是来自源文件中的段；另一种是来自库中的段，这些段有三种属性：只读(RO)段、可读写(RW)段、初始化为零(ZI)段。RO 段最前面，其后是 RW 段，最后 ZI 段同一段中，代码段在数据段前面。链接器的输出是一个可执行的映像文件，包含一个或多个段，这些在镜像文件中的段叫输出段，也有三种属性：只读(RO)段、可读写(RW)段、初始化为零(ZI)段。

ARM 映像文件的组织方式

ARM 映像文件是一个层次性结构的文件，包括了域（region）、输出段(output section)和输入段(input section)。映像文件由一个或多个域组成，每个域可以包含一个、两个或三个输出段，每个输出段包含一个或多个输入段，每个输入段都可以包含代码和数据。映像文件可以有两种地址：加载地址和执行地址。加载地址就是文件在存储器中的存储地址，执行地址就是文件在运行时的地址，文件加载的存储区叫加载域，文件运行的存储区叫执行域，例如，为了提高速度，要把执行的程序从 ROM 存储区移到高速缓冲区之后再执行，此时，加载地址就不再是执行地址了。

映像文件的段(输入/输出段)，域（加载/运行域）的概念

目标文件经过链接器链接生成的文件称为映像文件.文件中程序之间的位置关系与实际存储时的地址关系是对应的，程序代码是实际储存后的代码的一个“映像”.映像文件由一个或多个域组成#映像文件中的域，就是存放映像文件的一个存储区。一个映像文件会用几个存储区，主要看映像文件的结构和目标存储器的组成。如果映像文件不大，一般情况下自用一个存储区；如果系统比较复杂，而文件又很大，一般把只读属性的代码和可读/写属性的数据分开存储。#每个域可以包含一个、两个或三个输出段，主要看域的存放位置和对输出段的要求，也和存储器的特性有关。#每个输出段包含一个或多个输入段：链接器把属性相同的输入段按照一定的顺序组织在一起，形成输出段。属性相同的输入段，它们在系统中的作用相近，这样组成的输出段便于存储和管理。#每个输入段都都可以包含代码和数据：输入段可以包含代码和数据，如在一个代码段后面定义一个数据缓冲区。但是这样的代码和数据只能有相同的属性，一般定义为只读属性。所以，如果数据需要可读/写，就不能和代码处于同一段内。

域看起来有些多余，实际上更为重要的是用域来描述输出区运行前和运行时在存储系统上的位置。所以，域分为装载域和运行域。装载域描述运行前输出段在 rom/ram 中的分布状态，运行域描述运行时输出段在 rom/ram 中的分布状态。大多数情况下，映像文件在执行前把它装载到 rom 中，而当运行时域里 的有些输出段必须复制到 ram 中，程序才能正常运行，所以，在装载和运行时，有些段处在不同的位置（地址空间）。

什么是 Scatterloading

在实际的嵌入式系统中，ADS 提供的缺省存储器映射是不能满足要求的。#用户的目标硬件通常有多个存储器设备位于不同的位置#这些存储器设备在程序装载和运行时可能还有不同的配置

【Scattertoading（分散装载】#可以通过一个文本文件来指定一段代码或数据在加载和运行时在存储器中的不同位置。这个文本文件 scatterfile 在命令行中由-scatter 开关指定，例如：armlink_scatterscat.scffile1.ofile2.O. 在 scatterfile 中可以为每一个代码或数据区在装载和执行时指定不同的存储区域地址，Scatlertoading 的存储区块可以分成二种类型：#装载区：当系统启动或加载时应用程序的存放区。#执行区：系统启动后，应用程序进行执行和数据访问的存储器区域，系统在实时运行时可以有一个或多个执行块。

ARM7 系统启动的一般过程

启动程序一般流程如下：1）堆和栈的初始化；　2）向量表定义；3）地址重映射及中断向量表的转移；　4）设置系统时钟频率；5）中断寄存器的初始化；6）进入主程序。

AMBA 总线（AHB 和 APB 的概念）

AMBA 总线是由 ARM 公司提出的一种开放性的片上总线标准，它独立于处理器和工艺技术，具有高速度低功耗等特点，受到集成电路设计行业的广泛欢迎。

一个以 AMBA 架构的 SoC，一般包含高性能的系统总线 AHB 和低功耗的外设总线 APB。系统总线（AHB）负责连接如嵌入式处理器、DMA 控制器、片上存储器和其他外设接口，或者其他需求高带宽的元件。而外设总线（APB）则用来连接系统周边的外部设备，其协议相对 AHB 较为简单。AHB 与 APB 之间通过桥接器（Bridge）互联。

EMI 和存储器—NOR/NAND_FLASH 的区别

EMI 控制器（EXTERNAL MEMORY INTERFACE）支持 SRAM、SDRAM 、ROM、NOR FLASH 及 NAND FLASH。芯片的外部存储接口模块提供了对这些外部存储器的读写接口，并且可以通过配置相关寄存器，灵活的实现对不同外部存储器的操作.同时 EMI 也支持地址的 REMAP 功能，即两个逻辑地址指向同一个物理地址.

NOR 技术（亦称为 Linear 技术）闪速存储器是最早出现的 Flash Memory，目前仍是多数供应商支持的技术架构。它源于传统的 EPROM 器件；在擦除和编程操作较少而直接执行代码的场合，尤其是纯代码存储的应用中广泛使用，如 PC 的 BIOS 固件、移动电话、硬盘驱动器控制存储器等。

NOR 技术 Flash Memory 具有以下特点：#程序和数据可存放在同一芯片上，拥有独立的数据总线和地址总线，能快速随机读取，允许系统直接从 Flash 中读取代码执行，而无需先将代码下载至 RAM 中再执行；#可以单字节或单字编程，但不能单字节擦除，必须以块为单位或对整片执行擦除操作，在对存储器进行重新编程之前需要对块或整片进行预编程和擦除操作。

由于 NOR 技术 Flash Memory 的擦除和编程速度较慢，而块尺寸又较大，因此擦除和编程操作所花费的时间很长，在纯数据存储和文件存储的应用中，NOR 技术显得力不从心。

NAND 技术 Flash Memory 具有以下特点：#以页为单位进行读和编程操作，1 页为 256 或 512B（字节）；以块为单位进行擦除操作，1 块为 4K、8K 或 16KB。具有快编程和快擦除的功能，其块擦除时间是 2ms；而 NOR 技术的块擦除时间达到几百 ms。#数据、地址采用同一总线，实现串行读取。随机读取速度慢且不能按字节随机编程。#芯片尺寸小，引脚少，是位成本（bit cost）最低的固态存储器，将很快突破每兆字节 1 美元的价格限制。#芯片包含有失效块，其数目最大可达到 3~35 块（取决于存储器密度）。失效块不会影响有效块的性能，但设计者需要将失效块在地址映射表中屏蔽起来。

性能比较：NOR 读取速度稍快;NAND 擦除和写入速度远远大于 NOR flash;擦除 Nor 器件时是以 64~128KB 的块进行的，执行一个写入/擦除操作的时间为 1~5s；擦除 Nand 器件是以 8~32KB 的块进行的，执行相同的操作最多只需要 4ms。

接口差别：Nor Flash 带有 SRAM 接口，有足够的地址引脚来寻址，可以很容易地存取其内容的每一字节。用作程序存储器；Nand 器件使用复杂的 I/O 口来串行地存取数据，各个产品或厂商的方法可能各不相同。Nand 的读和写操作采用固定大小的块，这一点有点像硬盘管理此类操作。

容量和成本:NOR 小 ，1~16MB,成本高，存储代码;NAND 非常大，Nand Flash 的单元尺寸几乎是 Nor 器件的一半。由于生产过程更为简单，Nand 结构可以在给定的尺寸内提供更高的容量。16MB~512B 以上，存储数据

可靠性和耐用性:NOR 可擦写 10 万次;NAND 可擦写 100 万次;存在位反转和坏区的问题，需要进行 EDC/ECC 算法校验和坏区标识管理

易用性:NOR 可以非常直接地使用基于 Nor 的闪存，像 SRAM 存储器那样连接，并可以在上面直接运行代码;NAND 由于需要 I/O 接口，Nand 要复杂得多。各种 Nand 器件的存取方法因厂家而异。在使用 Nand 器件时，必须先写入驱动程序，才能继续执行其它操作。

通信接口 UART，RS232 电平，SPI，IIC，RS485 差分方式

UART 是通用异步串行通信接口的总称，UART 允许在串行链路上进行全双工的通信，输出/输入的电平为 TTL 电平。一般来说，全双工 UART 定义了一个串行发送引脚(TXD)和一个串行接收引脚(RXD)，可以在同一时刻发送和接收数据.【RS232】是美国电子工业协会(EIA)制定的串行通讯标准，又称 RS-232-C 。RS232 是一个全双工的通讯标准，它可以同时进行数据接收和发送的工作. 传输距离 25 米，最大传输速度 115200BPS 逻辑高电平：－3V~15V，逻辑低电平 ＋3V~15V；RS422/485 差分双绞线传输信号异步串口#RS232 的电平信号是相对于本地地线而言#RS422/485 的信号是平衡传输方式，双绞线之间的相对电平代表信号#传输距离远 1200 米#传输速度快 10Mb/S 采用差分信号负逻辑，-2V~~-6V 表示“0”，+2V~~+6V 表示“1”；I2C BUS（Inter IC BUS）是 Philips 推出的芯片间串行传输总线，它以 2 根连线实现了完善的全双工同步数据传送，可以极方便地构成多机系统和外围器件扩展系统。I2C 总线采用了器件地址的硬件设置方法，通过软件寻址完全避免了器件的片选线寻址方法，从而使硬件系统具有最简单而灵活的扩展方法。SPI（Serial Peripheral Interface——串行外设接口）总线系统是一种同步串行外设接口，允许 MCU 与各种外围设备以串行方式进行通信、数据交换。外围设备包 FLASH、RAM、A/D 转换器、网络控制器、MCU 等。SPI 系统可直接与各个厂家生产的多种标准外围器件直接接口，一般使用 4 条线：串行时钟线 SCK、主机输入/从机输出数据线 MISO、主机输出/从机输入数据线 MOSI 和低电平有效的从机选择线 SSEL（有的 SPI 接口芯片带有中断信号线 INT，有的 SPI 接口芯片没有主机输出/从机输入数据线 MOSI）。

音频接口 IIS 和 AC97

I2S（Inter—IC Sound)总线是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准，该总线专责于音频设备之间的数据传输，广泛应用于各种多媒体系统。将音频数字化，其实就是将声音数字化。最常见的方式是透过脉冲编码调制 PCM。从 1996 年 Inter 联合 AD，YAMAHA、NS 和 Creative 一起推出 Audio Codec ‘97 规范开始，主要是声卡，在 AC ‘97 规范下，主板集成声卡电路变得很简单，成本也很低廉

Watchdog，pwm，timer，rtc 原理

Watchdog：[作用]提供防止系统失败的一种保证措施，通过软件间隔的对 WATCHDONG 进行服务，确保系统工作正常。［特征］提供 reset 系统；提供 watchdog 中断；

PWM：脉宽调制模块(PWM)可以产生脉宽调制信号#录音重放模式：可以运用存储的声音文件数据驱动 speaker，为了发出声音，必须使用跟你录下声音的相同频率，对于不变的频率，它能产生占空比变化的脉冲，脉冲的宽度必须跟某个指定的采样声音的模拟电压成比例#当 PWM 工作在发音模式时，PWM 能够对一个单一频率发出连续的音调。用户可以利用该功能播放 midi 等简单音乐。#如果在 PWM 输出加上一个低通滤波器，那么它就能输出具有不同脉宽的频率。

TIMER：［作用］TIMER 是片内集成的通用定时器，能够向系统提供定时中断，也可以通过外部时钟进行定时计数。［特征］多少个通道：4 个，10 个；多少位：16 位/32 位/64 位。［计数模式］重启模式：由用户自定义周期大小，每个时钟（内部或者外部）计数一次，到时产生中断然后重新装载；自由模式：到时产生中断，重新装载固定周期大小（0xffff_ffff）。［外部事件捕获］外部特定边沿时候，记录计数器的当前值并产生中断。

RTC：［作用］提供系统时钟；产生连续的周期中断；提供定时功能。［特征］提供年，月，日，时，分，秒计时功能；提供 1/256 秒—1 秒软件可配置的连续中断(即采样中断)；可设置周期中断，有分中断和秒中断；提供定时中断，精确到分钟（一般用作闹铃）；提供闰年判断机制。

电源管理设计，线性稳压电源和开关电源的原理和比较

嵌入式系统 硬件设计中电源系统非常重要，常用的两种稳压电源器件是交流稳压电源和直流稳压电源。

待机模式的设计；卸载计算密集任务到专用硬件；动态电源管理(Dynamic Power Management)；动态电压/频率调整(Dynamic Voltage/Frequency Scaling)；时钟门控(Clock Gating)技术；线性电压稳压器[优点]：简单;输出纹波电压低;出色的 line 和负载稳压;对负载和 line 的变化响应迅速;电磁干扰（EMI）低.[缺点]：效率低;如果需冷却设备，则要求较大的空间.O 开关电压稳压器[优点]：效率高（降低了冷却所需的空间需求）;能够处理较高的电源密度;可用于传递单个或多个输出电压，大于或小于生成的输出电压。[缺点]：输出纹波电压高；瞬时恢复时间较慢；产生电磁干扰（EMI）

可编程逻辑器件（FPGA，CPLD）的主要单元构成

基于乘积项(Product-Term)的 PLD 结构,称为【CPLD】三部分构成：宏单元（Marocell），可编程连线（PIA）和 I/O 控制块。宏单元是 PLD 的基本结构，由它来实现基本的逻辑功能。

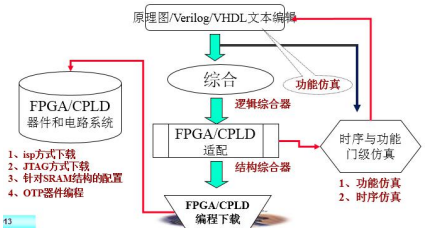
基于查找表 LUT(Look-Up-Table)的结构称为【FPGA】。A,B,C,D 由 FPGA 芯片的管脚输入后进入可编程连线，然后作为地址线连到到 LUT，LUT 中已经事先写入了所有可能的逻辑结果，通过地址查找到相应的数据然后输出，这样组合逻辑就实现了。

PCB 设计流程

原理图设计基本流程：●设置图纸 ●装载元件库 ●元件布局 ●电路布线 ●元件封装与序号 ●报表输出

印刷电路板(PCB)设计基本流程：●规划电路板和环境参数 ●引入网络表 ●元件布局和调整 ●布线规则设置 ●自动布线和手工调整 ●报表输出 ●存盘与打印

FPGA 的设计流程



OS 基本概念

操作系统是计算机系统中软件技术含量最大、附加值最高的部分，是软件（子）系统的核心，是软件的基础运行平台。1.操作系统实际上是一个计算机系统中硬、软件资源的总指挥部 。2.操作系统的性能高低，决定了整个计算机的潜在硬件性能能否发挥出来。3.操作系统本身的安全可靠程度，决定了整个计算机系统的安全性和可靠性。

进程和线程/任务

进程是可并发执行的、具有独立功能的程序在一个数据集合上的运行过程，是操作系统进行资源分配和保护的基本单位。一个进程可以简单的认为是一个程序在系统内的唯一执行，一个进程应该包括：程序的代码；程序的数据；PC 的值，用来指示下一条将运行的指令；一组通用的寄存器的当前值，堆、栈；一组系统资源（如打开的文件）。总之，进程包含了正在运行的一个程序的所有状态信息。动态性：程序的运行状态在变，PC、寄存器、堆和栈等；独立性：是一个独立的实体，是计算机系统资源的使用单位。每个进程都有“自己”的 PC 和内部状态，运行时独立于其他的进程（逻辑 PC 和物理 PC）；并发性：从宏观上看各个进程是同时独立运行的。进程包括它的指令代码和数据。也包括程序计数器（PC）和 CPU 中所有的寄存器，还包括存放在进程堆栈中的临时数据、返回地址及变量。线程（Thread）实体之间可以并发地执行；实体之间共享相同的地址空间；进程=线程+资源平台。线程和进程主要区别：地址空间（对数据是否有保护），线程和线程之间可以共享地址，但进程不可以一个线程也可做一个任务。任务是程序的动态表现，在操作系统中体现为线程，是程序的依次执行过程。[单道作业-多道作业（并发）-OS（多进程）-OS(多进程+多线程）-embedded OS(单进程+多线程）-多任务]

任务概念(堆栈，TCB，无限循环，共享内存)

任务是程序的动态表现，在操作系统中体现为线程，是程序的一次执行过程。

程序是静止的，存在于 ROM、硬盘等外部设备。任务是运动的，存在于内存中，有睡眠、就绪、运行、阻塞、挂起等多种状态。

相同的程序的多次执行是可以的，就形成了多个优先级不同的任务，每一个都是独立的。

在实时系统中，把应用程序的设计过程分割为多个任务，每个任务都有自己的优先级，在操作系统的调度下协调运行。

一个任务，也称作一个线程，是一个简单的程序，嵌入式操作系统分配资源的基本单位.该程序可以认为 CPU 完全只属于该程序自己。

#任务（TASK）就是一个拥有自己堆栈的函数；#每个任务都是一个无限的循环；#每个任务被赋予一定的优先级；#每个任务认为自己独自享用 CPU 寄存器和自己的栈空间；#任务通过控制块 TCB 描述；#每个任务都处在若干个状态下；

任务调度的概念，非抢占式和抢占式调度

多任务操作系统的核心工作就是任务调度。所谓调度，就是通过一个算法在多个任务中确定该运行的任务，做这项工作的函数就叫做调度器。

μC/OS_II 进行任务调度的思想是 “近似地每时每刻总是让优先级最高的就绪任务处于运行状态” 。为了保证这一点，它在系统或用户任务调用系统函数及执行中断服务程序结束时总是调用调度器，来确定应该运行的任务并运行它 。μC/OS_II 进行任务调度的依据就是任务就绪表.基本的调度算法：有先来先服务 FCFS,最短周期优先 SBF；优先级法 Priority；轮转法 Round-Robin。

非抢占内核要求每个任务自我放弃 CPU 的所有权。#异步事件还是由中断服务来处理。#中断服务可以使一个高优先级的任务由挂起状态变为就绪状态。#中断服务以后控制权还是回到原来被中断了的那个任务。优点：#不可剥夺型内核的一个优点是响应中断快问题：#不可剥夺型内核的任务级响应时间是不确定的，不知道什么时候最高优先级的任务才能拿到 CPU 的控制权，完全取决于应用程序什么时候释放 CPU。

抢占最高优先级的任务一旦就绪，总能得到 CPU 的控制权#当一个运行着的任务使一个比它优先级高的任务进入了就绪态#（时钟机制）当前任务的 CPU 使用权就被剥夺了，或者说被挂起了#那个高优先级的任务立刻得到了 CPU 的控制权。

如果是中断服务子程序使一个高优先级的任务进入就绪态#中断完成时，中断了的任务被挂起，#优先级高的那个任务开始运行。

抢占式内核总是让就绪态的高优先级的任务先运行，中断服务程序可以抢占 CPU，到中断服务完成时,内核让此时优先级最高的任务运行（不一定是那个被中断了的任务）。

任务级系统响应时间得到了最优化

可重入函数的概念

可重入函数主要用于多任务环境中，一个可重入的函数简单来说就是可以被中断的函数，也就是说，可以在这个函数执行的任何时刻中断它，转入 OS 调度下去执行另外一段代码，而返回控制时不会出现什么错误

#可重入型函数可以被一个以上的任务调用，而不必担心数据的破坏。#可重入型函数任何时候都可以被中断，一段时间以后又可以运行，而相应数据不会丢失。#可重入型函数或者只使用局部变量，即变量保存在 CPU 寄存器中或堆栈中。如果使用全局变量，则要对全局变量予以保护。

任务同步和通信的几种机制的原理（信号量 2 种，常用同步和通信机制，邮箱和消息）

任务间的同步是指，异步环境下的一组并发执行任务因各自的执行结果互为对方的执行条件，因而任务之间需要互发信号，以使各任务按一定的速度执行。# 任务同步也常常使用信号量。与任务间的通信不同，信号量的使用不再作为一种互斥机制，而是代表某个特定的事件是否发生。# 任务的同步分为单向同步和多向同步。

任务间的通信有 2 个途径：#共享数据结构#消息机制////使共享数据满足互斥：#关中断#使用测试并置位指令#禁止做任务切换#利用信号量和其他方式

【信号量】用于对共享资源的竞争性访问实现任务同步；信号量通常有二进制信号量和计数器信号量两种形式。用于对共享资源的竞争性访问，信号像是一把钥匙，任务要运行下去，得先拿到这把钥匙。如果信号已被别的任务占用，该任务只得被挂起，直到信号被当前使用者释放。 二进制信号量（信号 0，1）计数器信号量（0~~255）

任务间另一种通信方式是使用消息机制。任务可以通过内核提供的系统服务向另一个任务发消息。消息机制包括：消息邮箱和消息队列。

【消息邮箱】消息通常是内存空间的一个数据结构，通常是一个指针型变量。#任务或中断服务子程序通过内核服务，可以把一则消息放到邮箱里去；同样的，一个或多个任务通过内核服务可以接收这则消息。#内核一般提供以下邮箱服务：邮箱内消息内容的初始化；将消息放入邮箱（POST）；等待消息进入邮箱（PEND）；从邮箱中得到消息；

【消息队列】消息队列实际上是邮箱阵列，在消息队列中允许存放多个消息。对消息队列的操作和对消息邮箱的操作基本相同。#消息队列服务包括：消息队列初始化；放一则消息到队列中去（POST）；等待一则消息的到来（PEND）；从队列中等到消息；。

系统时钟的管理和定时器

实时内核提供时间，定时管理：【在定时方面】创建；停止；复位；删除；启动软件定时器；【在时间方面】#设置系统时间，使应用能设置当前的系统时间和日期#获得系统时间，以日历，系统启动以来的 tick 数等来获得系统时间#维护系统时基，处理定时事件。

时钟管理一般具有以下功能：郊吃日历时间，任务有限等待的计时，软件定时器的定时管理和维持系统时间片轮转调度。实时内核的时间、定时管理一系统时钟为基础，系统时钟一般定义为整数或长整数，提供给应用程序所有和时间有关的服务。定时器的初始化工作主要包含以下内容：初始化定时器相关的寄存器；设置 tick 的间隔时间，使定时器每隔一个确定的时间产生一个时钟中断；挂接系统时钟中断处理程序。在时间方面，内核通常提供以下功能：设置系统时间：使应用能够设置当前系统的日期和时间。获得系统时间：以日历时间、系统启动以来所经历的 tick 数等形式获得当前的系统时间。维护系统时基、处理定时事件。在定时方面，内核通常提供以下功能：创建软件定时器；启动软件定时器；使用软件定时器停止计时；复位软件定时器；删除软件定时 i

程序使用 ARM 汇编指令描述 ARM 处理器响应 IRQ 中断请求完成的工作内容

```
MRS     R0, CPSR                STMFD     SP!, {R0-R12}
MSR     SPSR_IRQ, R0            BL        C_IRQ_HANDLE
BIC     R0, R0, #0XBF           LDMFD     SP!, {R0-R12}
ORR     R0, R0, #0B10010010     MSR       R0, SPSR_IRQ
MRS     CPSR , R0               MRS       CPSR,  R0
SUB     LR , PC,#4              SUB       PC,  LR,  #4
MOV     PC, #0X00000018
```

分析

```
T_bit    EQU     0x20
EXPORT SWI_HANDER
SWI_Hander
    STMFD       sp!, {r0-r12, lr}
    TST         r0, #T_bit
    LDRNEH      r0, [lr, #-2]
    BICNE       r0, r0, #0xff00
    LDREQ       r0, [lr, #-4]
    BICEQ       r0, r0, #0xff000000
    BL          C_SWI_Handler
    LDMFD       sp!, {r0-r12,pc}^
END
```

”；伪指令，定义 T_bit 值为 0x20 的常量，实际上用来检测 Thumb 指令标志位

；伪指令来声明一个可全局引用的标号 SWI_HANDER

；进入中断程序 SWI_Hander

；进入中断前保护现场，压栈保存 CPSR 的值，R1-R12 的数据以及 LR 的地址值

；判断 CPSR[5]是否为 1 来说明是否为 Thumb 状态

；若为 Thumb 状态，提取 SWI 指令中相应的低 8 位立即数

；若为 ARM 状态，提取 SWI 指令中相应的低 24 位立即数

；跳转到 C_SWI_Hander 这个 C 处理程序执行

；SWI 异常中断返回，由堆栈弹出进入中断前 CPSR 的值（即 R0），R1-R12 的数据以及 LR 的地址值

；中断程序结束