□计算模式

- **十五年周期定律：IBM前任CEO郭士纳（Louis Gerstner）提出一个重要的观点：计算模式每隔15年发生一次变革。**

❶1965~1980：大型机时代，以科学计算为主的系统计算模式

❷1980~1995：个人计算机时代，以信息处理为主的个人计算模式

❸1995~2010：互联网时代，以信息传输为主的协同计算模式

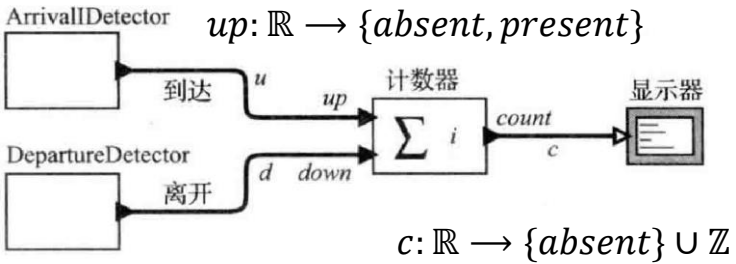
❹2010~ ：物联网、云计算时代，信息爆炸条件下的共享服务计算模式

趋势：集中——分布——集中

2. 云计算:一种无处不在、便捷且按需对一个共享的可配置计算资源（包括网络、服务器、存储、应用和服务）进行网络访问的模式，它能够通过最少量的管理以及服务提供商的互动实现计算资源的迅速供给和释放。从提供服务的层次可分为:基础设施即服务、平台及服务、软件即服务。云计算的三种模式：集中式的云、边缘的云以及被统称为雾计算的边缘节点。雾计算命名源自”雾是更贴近地面的云”这一名句。

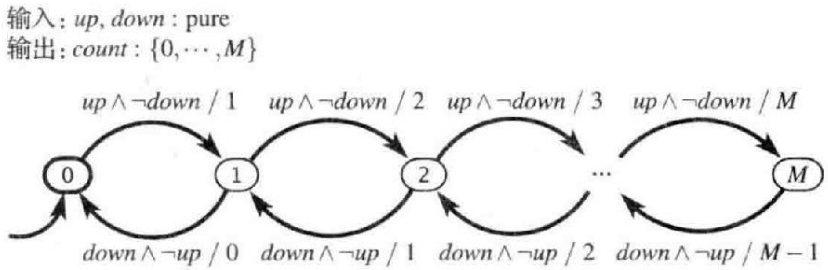
边缘计算：将计算资源和数据存储功能放置在接近数据源的位置，以减少数据传输延迟并提高实时性能的计算模型。
雾计算：一种介于云计算和边缘计算之间的计算模型，它将计算资源和数据存储功能放置在接近数据源的位置，但比边缘设备更远一些。

3.嵌入式系统
- 定义：以应用为中心，以计算机技术为基础，软件硬件可裁剪，适应应用系统对功能、可靠性、成本、体积、重量、功耗严格要求的专用计算机系统。简而言之，是含有处理器的专用软硬件系统
- 特点：嵌入专用、综合性强、设计高效可裁剪、程序固化、需要独立的开发系统、生命周期长、可靠性高、成本低、资源受限、功耗低等
- 组成：嵌入式微处理器、外围硬件设备、嵌入式操作系统、应用软件。



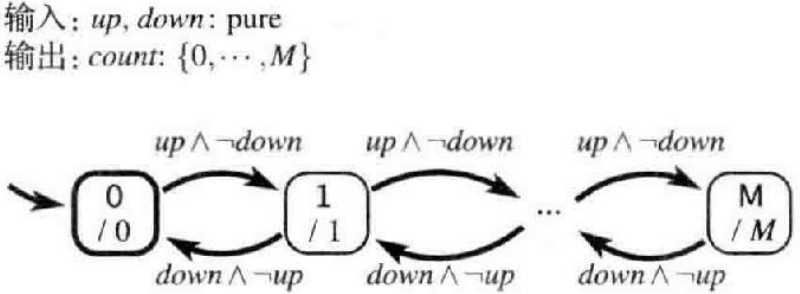
1. 纯信号（pure）：在任何时间$t \in \mathbb{R}$，输入信号$u(t)$要么不存在$(absent)$（即没有事件），要么存在$(present)$（即存在事件）
2. 数值型信号：以数值的形式存在或不存在
3. 状态：是系统在一个特定时间点上所具有的情形，编码了所有影响系统对当前输入作出反应的过去信息
4. 状态机：离散动态系统的模型，每个响应将一组输入估值映射到一组输出估值，且该映射可能依赖于它的当前状态。

有限状态机——米利型



监督条件/输出，输出由当前状态和当前输入确定,状态迁移时产生输出。

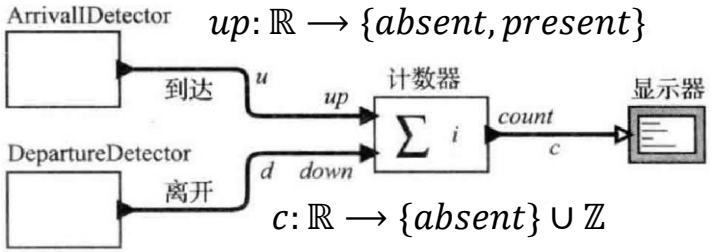
有限状态机——莫尔型

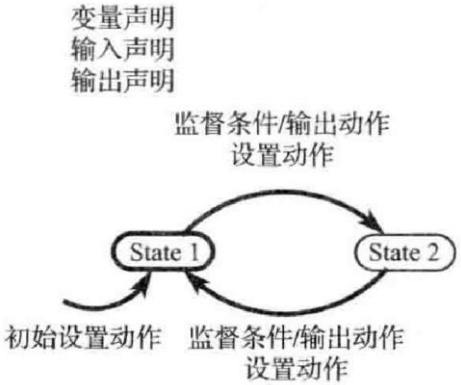


状态编号/输出，输出只由当前状态确定，在状态中产生输出

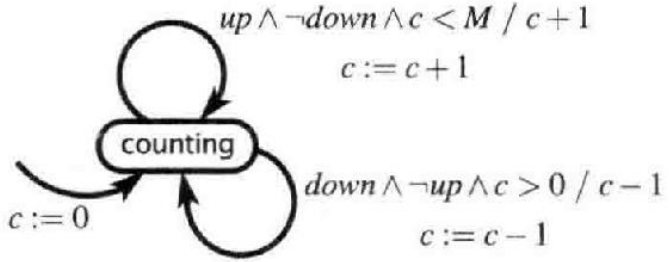
当状态数量扩大时，有限状态机的符号表示将不太适宜，**扩展状态机**(extended state machine)通过使用变量扩展有限状态机模型来解决这个问题，这些变量作为在状态间进行迁移操作的一部分，可读可写。

扩展自动机的一般表示方法：1）显示给出变量声明；2）初始阶段被声明的变量应该被初始化并显示在指向初始状态的迁移上；3）迁移的标注形式为 监督条件/输出动作
设置动作



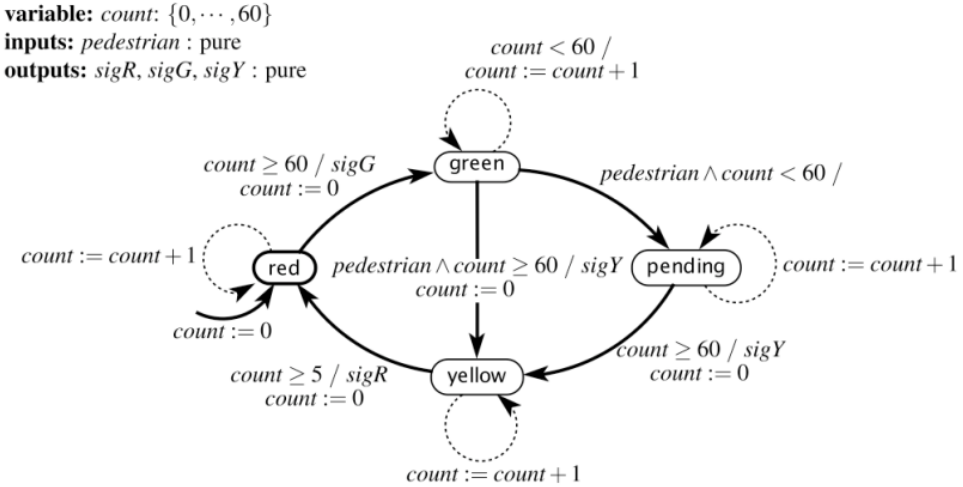


变量：$c: \{0, \cdots, M\}$
输入：$up, down$: pure
输出：$count$: $\{0, \cdots, M\}$



示例 3.9 图 3-10 给出了一个表示人行道交通灯的扩展状态机。这是一个时间触发的状态机，并假设每秒响应一次。状态机以红灯（red）状态开始，且用变量 count 来计数 60s。随后，其转换为绿灯（green）状态，将保持至纯输入 *pedestrian* 变为 *present*。这个输入是可以被生成的，如可以由行人按下“请求通过”按钮来触发。当 *pedestrian* 为 *present* 时，如果它已在绿灯状态保持至少 60s，那么状态机就转换至黄灯（yellow）状态。否则，它切换到挂起（pending）状态并在 60s 间隔的剩余时间内一直保持。这确保了一旦交通灯变为绿灯，其至少会保持 60s。在 60s 结束时，它将转为黄灯状态，并在转回红灯之前保持 5s 的黄灯。

状态机产生的输出是 *sigG*、*sigY*、*sigR*，分别为打开绿灯、黄灯与红灯。



红外传感器:
- 红外线的波长范围大致在0.76μm到1000μm的频谱范围之内。频率大致在4×10^{14}∽3×10^{11}Hz之间。红外分区：近红外区、中红外区、远红外区、极远红外区。这里所说的远近是指红外辐射在电磁波谱中与可见光的距离。
-红外线对管包括红外线发射管和红外线接收管。
- 红外发射管是一种发光二极管LED，发射的是红外线（近红外线），发射的红外光强度会随着电流的增大而增强，红外发对管工作于脉冲状态，因为脉动光（调制光）的有效传送距离与脉冲的峰值电流Ip成正比，只需尽量提高峰值Ip，就能增加红外光的发射距离。
- 红外线接收管是一种光敏二极管，内部具有红外光敏感特征的PN结，没有红外线照射时，反向电流很小（一般小于0.1微安），称为暗电流；有红外线照射时，携带能量的光子将能量传递给束缚电子，使其挣脱共价键形成自由电子和空穴对，这些载流子在反向电压作用下参与漂移运动，从而接收管导通形成光电流。
-应用：1）在遥控器中，发射管发射红外线信号，接收管接收并转换成电信号，实现信号的传递；2）在避障系统中，发射管发送红外线信号，当遇到障碍物时，红外线会被反射，接收管接收反射回来的红外线，通过比较接收到的光强变化，可以判断障碍物的存在和距离；3）作为一种电磁波，具有直线传播、反射、折射等特性，同时也具有较强的穿透能力，能够穿透某些不透明物质。因此，红外线对管在很多领域都有应用，如机器人避障、红外循迹小车等。

超声波传感器（**压电式超声波传感器**）：利用压电材料的压电效应来工作的。压电效应有正向压电效应和逆向压电效应。
- 超声波发送器:利用逆向压电效应制成——即在压电元件上施加电压，元件就变形（也称应变）引起空气振动产生超声波，超声波以疏密波形式传播，传送给超声波接收器。
- 超声波接收器:利用正向压电效应制成——即接收到的超声波促使接收器的振子随着相应频率进行振动，由于存在正向压电效应，就产生与超声波频率相同的高频电压
- 应用：1）无损探伤、测距离、B超成像

伺服电机：可以将输入的电压信号变换为轴上的角位移和角速度输出。在信号来到之前，转子静止不动；信号来到之后，转子立即转动；信号消失之后，转子又能即时自行停转。由于这种“伺服”性能，因而将这种控制性能较好的电动机称做伺服电动机。

步进电机：将电脉冲信号转换为直线位移或角位移的执行器

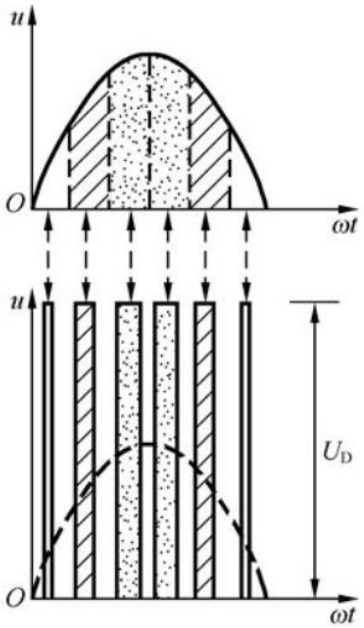
PWM原理：通过将有效的电信号分散成离散形式从而来降低电信号所传递的平均功率。根据面积等效法变脉冲的时间宽度，来等效的获得所需要合成的相应幅值和频率的波形

面积等效原理：冲量（脉冲的面积）相等而形状不同的窄脉冲加在具有惯性的环节上时，其效果基本相同，形基本相同

PWM频率：频率是指1秒钟内信号从高电平到低电平再回到高电平的次数(一个周期)

占空比：是一个脉冲周期内，高电平的时间与整个周期时间的比例

例子：高电平为5V 低电平则为0V，占空比为50%，高电平时间一半，低电平时间一半，在一定的频率下，就可以得到模拟的2.5V输出电压；75%的占空比，得到的电压就是3.75V

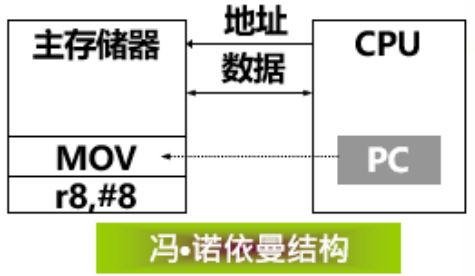


嵌入式处理器的分类：嵌入式微控制器（MCU）、嵌入式微处理器（MPU）、嵌入式DSP处理器、嵌入式片上系统（SOC）

- 嵌入式微处理器(MPU)的基础就是通用CPU,为了满足嵌入式应用的特殊要求，嵌入式微处理器在功能上和通用微处理器基本一样，但在工作温度、抗电磁干扰、可靠性、功耗等方面做了各种增强。目前主要嵌入式微处理器有MIPS、X86系列、ARM系列、PowerPC系列
- 嵌入式微控制器MCU，又称单片机，就是将整个计算机集成到一块芯片中
- 嵌入式DSP处理器,具有特殊设计：高效乘累加运算、超标量操作、指令流水线、高效数据存取、硬件重复循环、确定性操作（程序执行时间可预测）
- 嵌入式片上系统SoC,将很多功能模块集成到单个芯片上

计算机体系结构分类

1. 冯诺依曼结构（普林斯顿结构）：1）指令存储器与数据存储器一体化设计；2）指令地址与数据地址统一编码；3）高速运算时，储存传输通道有瓶颈
2. 哈佛结构：1）指令存储器与数据存储器一体化设计；2）存储地址独立编址、独立访问；3）四总线制提高吞吐率：程序的地址总线、数据总线，数据的地址总线、数据总线；4）取指与执行能并发



指令集分类：CISC与RISC

类别	CISC	RISC
指令系统	指令数量很多	较少，通常少于100
执行时间	有些指令执行时间很长	没有较长执行时间的指令
编码长度	编码长度可变，1~15字节	编码长度固定，通常为4个字节
寻址方式	寻址模式多样	较少的寻址模式
操作	可对mem和reg进行算逻操作	除Load/Store外只对reg进行算逻操作
编译	编译器简单，程序难以优化	编译器复杂，程序易于优化

信息存储的字节顺序：

- 大端存储：低地址中存放的是字数据的高字节
- 小端存储：低地址中存放的是字数据的低字节

ARM处理器

- **37个32位寄存器**：31个通用寄存器、6个状态寄存器。
- **工作状态**：ARM状态和Thumb状态
- **工作模式**：user（ARM处理器正常的程序执行状态）、system（运行具有特权的操作系统任务）、FIQ（用于高速数据传输或通道处理）、IRQ（用于通用的中断处理）、supervisor（操作系统使用的保护模式，复位、软中断调用（SWI））、abort（当数据或指令预取中止时进入该模式，可用于虚拟存储及存储保护）、undefined（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）
- **特权模式**：除用户模式以外，其余6种模式称之为特权模式。当处理器运行在用户模式下时，某些被保护的系统资源是不能被访问的。
- **异常模式**：除去用户模式和系统模式以外的5种又称为异常模式。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.

ARM状态下的寄存器组织

工作模式：	System&User	FIQ	Supervisor	Abort	IRQ	Undefined
未分组寄存器	R0	R0	R0	R0	R0	R0
	R1	R1	R1	R1	R1	R1
	R2	R2	R2	R2	R2	R2
	R3	R3	R3	R3	R3	R3
	R4	R4	R4	R4	R4	R4
	R5	R5	R5	R5	R5	R5
	R6	R6	R6	R6	R6	R6
	R7	R7	R7	R7	R7	R7
分组寄存器	R8	*R8_fiq*	R8	R8	R8	R8
	R9	*R9_fiq*	R9	R9	R9	R9
	R10	*R10_fiq*	R10	R10	R10	R10
	R11	*R11_fiq*	R11	R11	R11	R11
	R12	*R12_fiq*	R12	R12	R12	R12
堆栈指针	R13_usr	*R13_fiq*	*R13_svc*	*R13_abt*	*R13_irq*	*R13_und*
链接寄存器	R14_usr	*R14_fiq*	*R14_svc*	*R14_abt*	*R14_irq*	*R14_und*
程序计数器	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)	R15(PC)
当前程序状态寄存器	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
备份程序状态寄存器		*SPSR_fiq*	*SPSR_svc*	*SPSR_abt*	*SPSR_irq*	*SPSR_und*

Thumb状态下的寄存器组织

工作模式：	System&User	FIQ	Supervisor	Abort	IRQ	Undefined
未分组寄存器	R0	R0	R0	R0	R0	R0
	R1	R1	R1	R1	R1	R1
	R2	R2	R2	R2	R2	R2
	R3	R3	R3	R3	R3	R3
	R4	R4	R4	R4	R4	R4
	R5	R5	R5	R5	R5	R5
	R6	R6	R6	R6	R6	R6
	R7	R7	R7	R7	R7	R7
堆栈指针	SP_usr	*SP_fiq*	*SP_svc*	*SP_abt*	*SP_irq*	*SP_und*
链接寄存器	LR_usr	*LR_fiq*	*LR_svc*	*LR_abt*	*LR_irq*	*LR_und*
程序计数器	PC	PC	PC	PC	PC	PC
当前程序状态寄存器	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
备份程序状态寄存器		*SPSR_fiq*	*SPSR_svc*	*SPSR_abt*	*SPSR_irq*	*SPSR_und*

需要注意的几个通用寄存器：

R13寄存器：常用作堆栈指针SP（Stack Pointer），一种习惯用法。也可使用其他的寄存器作为堆栈指针。在Thumb指令集中，某些指令强制使用R13作为堆栈指针。在应用程序初始化时，一般都要初始化每种模式下的R13，使其指向该工作模式的栈空间。

R14寄存器:链接寄存器LR,当执行BL子程序调用指令时，R14中得到R15（程序计数器PC）的备份。BL Label,下一条指令地址→LR, Label→PC。当发生异常中断时，对应的分组寄存器R14_svc、R14_irq、R14_fiq、R14_abt和R14_und用来保存R15的返回值。其他情况下，R14用作通用寄存器。R14寄存器常用情形：1）**子程序返回**： MOV PC, LR； BX LR ； 2）**在子程序入口处使用以下指令将R14存入堆栈**：STMFD SP！,{<Regs>, LR}；**使用以下指令可以完成子程序返回**：LDMFD SP！,{<Regs>, PC}

R15寄存器：程序计数器（PC）。在ARM状态下，位[1:0]为0，位[31:2]用于保存PC；在Thumb状态下，位[0]为0，位[31:1]用于保存PC 。这跟每个状态下的指令长度有关， ARM状态下的指令长度位32位， Thumb状态下的指令长度为16位。

状态寄存器：

CPSR寄存器：可在任何工作模式下被访问，包括条件码标志位、中断禁止位、当前处理器模式标志位，以及其他一些相关的控制和状态位。

SPSR寄存器：备份状态寄存器，当异常发生时，SPSR用于保存CPSR的当前值，从异常退出时则可由SPSR来恢复CPSR。用户模式和系统模式不属于异常模式，没有SPSR。

所以，**六个状态寄存器由1个CPSR和6个不同异常模式下的SPSR组成**。

程序状态寄存器的功能：1）保存ALU中的当前操作信息；2）控制允许和禁止中断；3）设置处理器的工作模式

CPSR_f或SPSR_f								CPSR_s或SPSR_s								CPSR_x或SPSR_x								CPSR_c或SPSR_c							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
N	Z	C	V	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	I	F	T	M4	M3	M2	M1	M0
负数或小于	零	进位或借位或扩展	溢出	保留位																				IRQ 禁止 1-禁止 0－允许	FIQ 禁止 1-禁止 0－允许	状态位 0-ARM 1-Thumb	模式位 10000-0x0010-用户 10001-0x0011-快速中断 10010-0x0012-中断 10011-0x0013 管理 10111-0x0017-未定义 11111-0x001F-系统						

在ARM状态下，绝大多数的指令都是有条件执行的
在Thumb状态下，仅有分支指令B是有条件执行的

ARM微处理器的指令集是Load/Store型，它可以使用间接寻址等方式访问存储器。

指令格式：

- <opcode>{<cond>}{S} <Rd>, <Rn>{, operand2}
  - opcode: 指令助记符，如ADD, LDR, STR
  - *cond:* *执行条件，如NE, EQ*
  - *S:* *是否影响CPSR的值*
  - Rd: 目标寄存器
  - Rn: 第一操作数寄存器
  - *operand2:* *第二操作数*
    - *灵活使用第二操作数能够提高代码质量*
  - 例子：ADDEQS R0, R1, #0x3f

指令助记符

BL	**带返回的跳转指令**
BLX	**带返回和状态切换的跳转指令**
BX	**带状态切换的跳转指令**
SWI	**软件中断指令**

Thumb指令集：

- ARM体系结构支持16位的Thumb指令集
  - Thumb指令集是ARM指令集的一个子集，允许指令编码为16位的长度
  - 指令对应
    - 所有的Thumb指令都有对应的ARM指令
  - 编程模型对应
    - Thumb的编程模型对应于ARM的编程模型
  - 子程序互相调用
    - 在应用程序的编写过程中，只要遵循一定调用的规则，Thumb子程序和ARM子程序就可以互相调用
- Thumb指令的特性
  - Thumb指令集中的数据处理指令的操作数仍然是32位，指令地址也为32位
  - 大多数的Thumb指令是无条件执行的，而几乎所有的ARM指令都是有条件执行的
  - 大多数的Thumb数据处理指令的目的寄存器与其中一个源寄存器相同
  - Thumb指令的长度为16位，只用ARM指令一半的位数来实现同样的功能
    - 要实现特定的程序功能，所需的Thumb指令的条数较ARM指令多
- 空间效率、时间效率和功耗分析
  - 存储空间
    - Thumb代码约为ARM代码的60%～70%
    - Thumb代码使用指令数比ARM代码多约30%～40%
  - 访存速度
    - 使用32位存储器，ARM代码比Thumb代码快约40%
    - 使用16位存储器，Thumb代码比ARM代码快约40%～50%
  - 功耗分析
    - 使用Thumb代码，存储器功耗会降低约30%
- ARM指令集和Thumb指令集各有其优点
  - 对系统的性能有较高要求，应使用32位的存储系统和ARM指令集
  - 对系统的成本及功耗有较高要求，则应使用16位的存储系统和Thumb指令集
  - 若两者结合使用，充分发挥其各自的优点，会取得更好的效果

ARM指令集功能更全，性能更高。thumb指令集比ARM指令集指令密度要大。

工作模式的改变：系统调用、外部中断或异常
工作状态的改变：BX、BLX、异常

◆进入Thumb状态

■执行BX指令

– BX：带状态切换的跳转指令

– 当操作数寄存器的最低位[0]为1时，可以使微处理器从ARM状态切换到Thumb状态

– BX R0 ;R0的最低位[0]为1

■处理器工作在Thumb状态，如果发生异常并进入异常处理子程序，则异常处理完毕返回时，自动从ARM状态切换到Thumb状态

BLX Label

BLX：
下一条指令地址送往LR寄存器；
Label送往PC，并完成状态切换

◆进入ARM状态

■执行BX指令

– BX：带状态切换的跳转指令

– 当操作数寄存器的最低位[0]为0时，可以使微处理器从Thumb状态切换到ARM 状态

– BX R0 ;R0的最低位[0]为0

■处理器工作在Thumb状态，如果发生异常并进入异常处理子程序，则进入时处理器自动从Thumb状态切换到ARM状态

在程序的执行过程中，处理器可以随时在两种工作状态之间切换
处理器工作状态的转变并不影响处理器的工作模式和相应寄存器中的内容
ARM微处理器在开始执行代码时总是处于ARM状态，也就是说复位后进入ARM状态

ARM处理器的异常处理

ARM的中断向量表内存放的是响应异常和中断的转移指令而不是中断向量地址。ARM处理器发生异常时，完成当前指令后跳转到相应的异常中断处理程序入口执行异常中断处理。异常处理完毕后返回原来的程序断点继续执行原来的程序。

具体来讲：

1）当出现异常后，ARM处理器会执行以下操作

-将CPSR复制到相应的SPSR中

-对CPSR进行设置：根据异常类型，强制设置CPSR的工作模式位；设置中断禁止位，以禁止中断发生；如果处理器处于Thumb状态，则切换到ARM状态。

-将下一条指令的地址存入相应链接寄存器LR。LR中保存的是下一条指令的地址（当前执行指令地址＋4或＋8，与异常类型有关）

-强制PC从相关的异常向量地址取下一条指令执行，从而跳转到相应的异常处理程序处。

2）异常处理完毕之后，ARM微处理器会执行以下几步操作从异常返回

-将SPSR复制回CPSR中

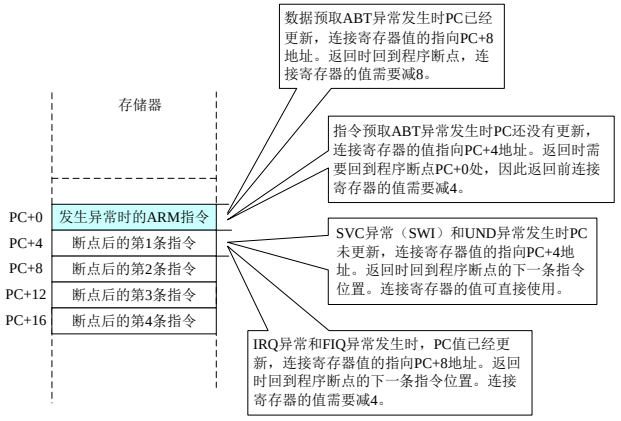
-将链接寄存器LR的值减去相应偏移量后送到PC中

注意：复位异常处理程序不需要返回

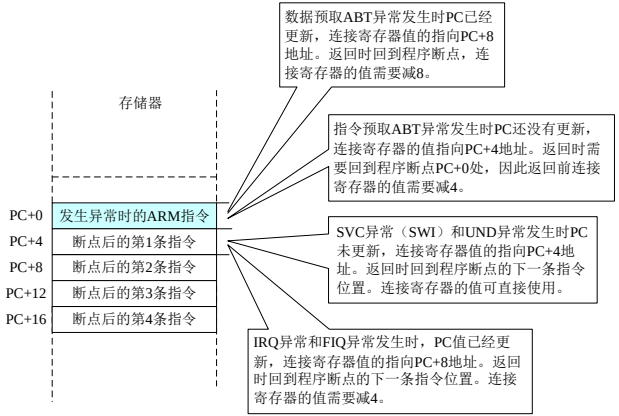
异常向量（**Exception Vectors**）

地 址	异 常	进入模式
0x0000,0000	复位	管理模式
0x0000,0004	未定义指令	未定义模式
0x0000,0008	软件中断	管理模式
0x0000,000C	中止（预取指令）	中止模式
0x0000,0010	中止（数据）	中止模式
0x0000,0014	保留	保留
0x0000,0018	IRQ	IRQ
0x0000,001C	FIQ	FIQ

中断向量号	异常类型	异常类型	PC是否已被更新	LR寄存器值	返回地址	返回时LR值传送返回到PC前调整
①	复位	SVC	清零	清零	不返回	
②	未定义指令	UND	未更新	X+4	X+4	不需要
③	软件中断SWI	SVC	未更新	X+4	X+4	不需要
④	指令预取中止	ABT	未更新	X+4	X	LR-4
⑤	数据访问中止	ABT	已经更新	X+8	X	LR-8
	保留					
⑦	外部中断请求,	IRQ	已经更新	X+8	X+4	LR-4
⑧	快速中断请求,	FIQ	已经更新	X+8	X+4	LR-4



中断向量号	异常类型	异常类型	PC是否已被更新	LR寄存器值	返回地址	返回时LR值传送返回到PC前调整
①	复位	SVC	清零	清零	不返回	
②	未定义指令	UND	未更新	X+4	X+4	不需要
③	软件中断SWI	SVC	未更新	X+4	X+4	不需要
④	指令预取中止	ABT	未更新	X+4	X	LR-4
⑤	数据访问中止	ABT	已经更新	X+8	X	LR-8
	保留					
⑦	外部中断请求,	IRQ	已经更新	X+8	X+4	LR-4
⑧	快速中断请求,	FIQ	已经更新	X+8	X+4	LR-4



◆在未定义指令处理程序中执行以下指令返回

■MOVS PC, R14_und

– 恢复PC（从R14_und）和CPSR（从SPSR_und）的值，并返回到未定义指令后的下一条指令

– 指令加后缀“S”且目的寄存器为PC则自动复制CPSR

◆在软件中断处理程序中执行以下指令返回

■MOVS PC，R14_svc

– 恢复PC（从R14_svc）和CPSR（从SPSR_svc）的值，并返回到SWI的下一条指令

– 指令加后缀“S”且目的寄存器为PC则自动复制CPSR

◆当确定中止原因后，Abort处理程序执行以下指令返回

■SUBS PC, R14_abt, #4 ; 指令预取中止

■SUBS PC, R14_abt, #8 ; 数据中止

– 恢复PC（从R14_abt）和CPSR（从SPSR_abt）的值，并重新执行产生中止的指令（返回当前指令）

◆IRQ处理程序执行以下指令返回

■SUBS PC , R14_irq , #4

– 该指令将寄存器R14_irq的值减去4后，复制到程序计数器PC中，同时将SPSR_irq寄存器的内容复制到CPSR中，并返回到引起中断的下一条指令

◆FIQ处理程序执行以下指令返回

■SUBS PC, R14_fiq , #4

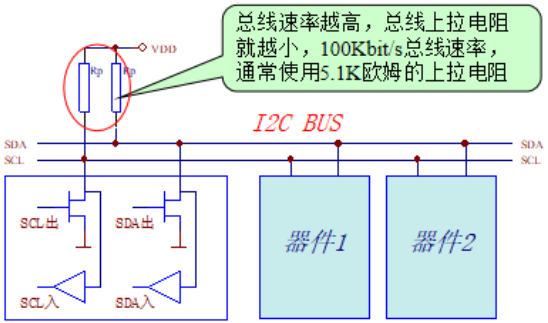
– 该指令将寄存器R14_fiq的值减去4后，复制到程序计数器PC中，同时将SPSR_fiq寄存器的内容复制到CPSR中，并返回到引起中断的下一条指令

各类半导体存储器的主要特点

类型	基本技术特点
FLASH	非易失、低成本、高密度、高速度、低功耗、高可靠性。闪速存储器是在EEPROM的基础上进化而来
ROM	成熟技术、高密度、可靠、低成本、不挥发、掩模耗时长、适合稳定编码的大规模生产。
SRAM	最快访问速度、高功耗、低密度、高成本。
EPROM	高密度、不挥发、擦除时必须用紫外线光照射。
EEPROM	电可擦除、可进行单字节的读/擦除/写、低可靠性、不挥发、高成本、低密度。数据保持时间最少10年。
DRAM	高密度、低成本、高速度、高功耗。

- SDRAM是在现有的标准DRAM中加入同步控制逻辑（一个状态机），利用一个单一的系统时钟同步所有地址、数据和控制信号，做到SDRAM的时钟频率与CPU前端总线时钟频率相同，实现存储器读写速度与CPU的处理速度保持一致。
- DDR是Double(Dual) Data Rate SDRAM的缩写（双倍数据速率）。嵌入式系统通常直接使用一颗SDRAM芯片作为主存储器。

嵌入式系统常用总线：I²C、SPI

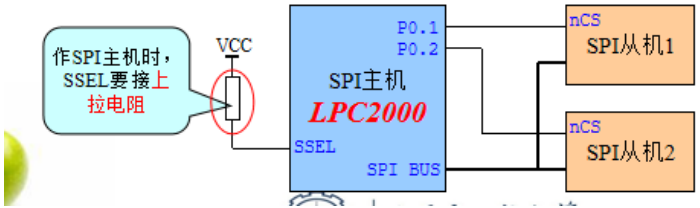


I²C总线，通过SDA（串行数据线）和SCL（串行时钟线）两根线在连到总线上的器件之间传送信息，并通过软件寻址识别每个器件，而不需要片选线。

SPI电气连接

使用SPI通信需要4个引脚，分别为：

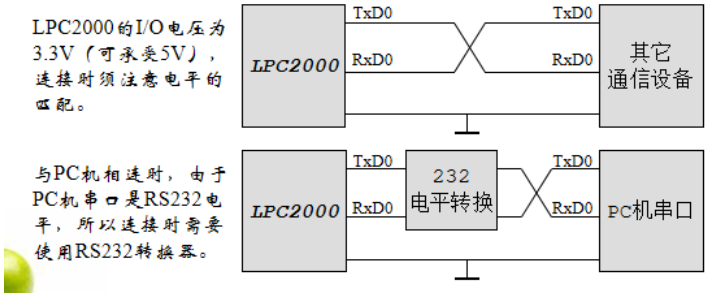
引脚名称	类型	描述
SCK	输入/输出	串行时钟，用于同步SPI接口间数据传输的时钟信号。该时钟信号总是由主机输出。
SSEL	输入	从机选择，SPI从机选择信号是一个低有效信号，用于指示被选择参与数据传输的从机。每个从机都有各自特定的从机选择输入信号。
MISO	输入/输出	主入从出，MISO信号是一个单向的信号，它将数据由从机传输到主机。
MOSI	输入/输出	主出从入，MOSI信号是一个单向的信号，它将数据从主机传输到从机。

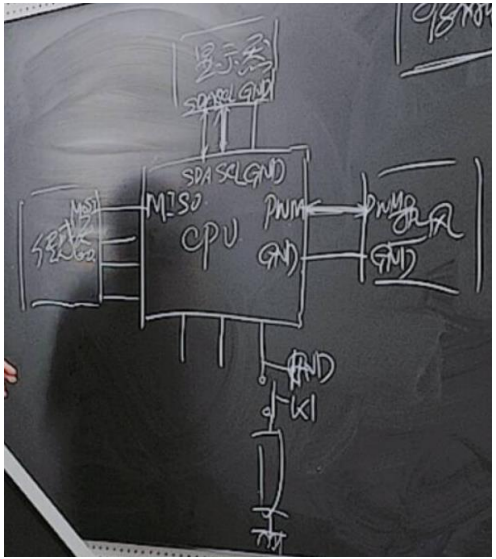


嵌入式系统常用接口：GPIO、UART、中断

使用UART0通信需要两个引脚，分别为：

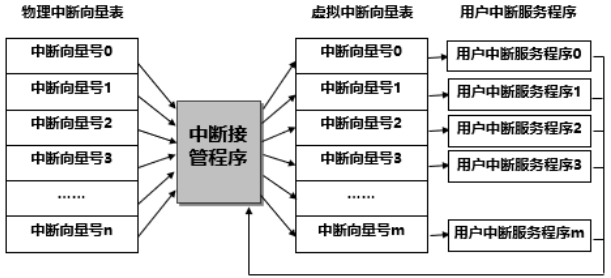
引脚名称	类型	描述
RxD0	输入	串行输入，接收数据
TxD0	输出	串行输出，发送数据





第四、五章的内容对最后一道设计题的解答有很大帮助,根据需求选择合适的处理器、存储器、外围硬件设备、各个硬件之间的如何连接，采取什么样的体系结构（哈佛还是冯诺依曼），采取什么样的操作系统（第七章）

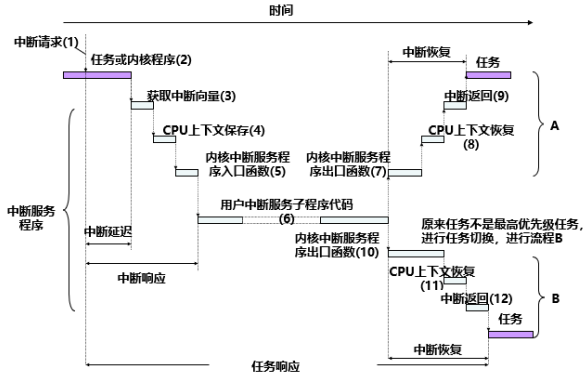
- 提高实时性能的因素：1）尽量采用硬件处理；2）优化微处理器的中断机制；3）采用简单的单线程循环程序；4）采用基于实时操作系统（ RTOS ）的复杂多线程操作系统
- 实时系统：对外来事件能在限定的响应时间内做出预定质量处理的计算机系统。
- 实时系统的分类：根据响应性能分类可分为硬实时系统和软实时系统。硬实时系统未能在时限内就某一事件做出响应而失败，并且该失败被认为是一种全面的系统失败。软实时系统存在时限指标，但是如果输出响应超过时限，一般不会造成严重后果。
- 实时系统的中断服务程序包括三个方面的内容：1）中断前导：保存中断现场，进入中断处理；2）用户中断服务程序：完成对中断的具体处理；3）中断后续：恢复中断现场，退出中断处理。
- 中断前导和中断后续通常由内核的中断接管程序来实现。硬件中断发生后，中断接管程序获得控制权并进行处理，然后才将控制权交给相应的用户中断服务程序。用户中断服务程序执行完成后，又回到中断接管程序。
- 中断接管程序处理中断前导：保存必要的寄存器，并根据情况在中断栈或是任务栈中设置堆栈的起始位置，然后调用用户中断服务程序。
- 中断接管程序处理中断后续：实现中断返回前需要处理的工作，主要包括恢复寄存器和堆栈，并从中断服务程序返回到被中断的程序。



特殊情况：

- 如果中断处理导致系统中出现比被中断任务具有更高优先级的就绪任务出现：把高优先级任务放入就绪队列，把被中断的任务从执行状态转变为就绪状态；完成用户中断服务程序后，在中断接管程序的中断后续处理中激活重调度程序，使高优先级任务能在中断处理工作完成后得到执行。
- 在允许中断嵌套的情况下，在执行中断服务程序的过程中，如果出现高优先级的中断：当前中断服务程序的执行将被打断，以执行高优先级中断的中断服务程序；当高优先级中断处理完成后，被打断的中断服务程序才又得到继续执行；发生中断嵌套时，如果需要进行任务调度，任务的调度将延迟到最外层中断处理结束时才能发生。

中断时序



抢占式调度内核的中断时序图

- 中断延迟时间:从中断发生到系统获知中断，并且开始执行中断服务程序所需要的最大滞后时间。
- 中断响应时间:从中断发生到开始执行用户中断服务程序的第一条指令之间的时间。 非抢占式系统的中断响应时间=中断延迟 ＋ 保存CPU内部寄存器的时间。抢占式系统的中断响应时间=中断延迟 ＋ 保存CPU内部寄存器的时间 ＋ 内核中断服务程序入口函数的执行时间。调用内核中断服务程序入口函数是为了通知内核即将进行中断服务，使得内核可以跟踪中断的嵌套，以便在解除中断嵌套后进行重调度
- 中断恢复时间： 用户中断服务程序结束后回到被中断代码之间的时间。非抢占式系统的中断恢复时间=恢复CPU内部寄存器的时间 ＋ 执行中断返回指令的时间。抢占式系统的中断恢复时间=内核中断服务程序出口函数执行时间＋ 恢复即将运行任务的CPU内部寄存器的时间 ＋ 执行中断返回指令的时间 。调用内核中断服务程序出口函数是为了判断是否脱离了所有的中断嵌套；如果脱离了嵌套，内核要判断是返回到原来被中断的任务，还是进入另外一个优先级最高的就绪任务。

用户的中断处理是由应用决定的，中断处理时间应该尽可能地短，
中断服务程序的作用：1）识别中断来源；2）从产生中断的设备取得数据或状态；3）通知真正处理该事件的那个任务来进行实际的中断处理工作。

后续嵌入式系统启动的内容看思考题的答案就行。链接：https://flowus.cn/share/d59f40ac-654f-41dc-aa99-938afd611ef3?code=8J4Q57【FlowUs 息流】ch6 嵌入式计算系统软件

这章的重点：嵌入式操作系统的特点、嵌入式操作系统的组成、可重入函数的定义和意义，任务状态切换、任务调度算法、恒定时间查找最高优先级任务的算法、优先级协议。

常用的嵌入式操作系统：嵌入式Linux、Windows CE、VxWorks、μc/OS、 HarmonyOS

嵌入式操作系统的特点：

- 可移植性：硬件平台的多样性和代码可重用性的要求，导致EOS要具有良好的可移植性。
- 强调实时性能
- 内核精简：所占空间小
- 抢占式内核：保证最高优先级任务能够立即得到执行，从而能够保证系统具有高度实时性能。通常把最高优先级指派给实时要求最高的任务。
- 使用可重入函数：抢占式内核的函数必须是可重入的。在抢占式内核控制之下，如果有两个以上任务需要调用同一个不可重入函数，则必须使之满足互斥条件。
- 可配置
- 可裁剪
- 高可靠性

嵌入式操作系统的组成：嵌入式内核、嵌入式TCP/IP网络系统、嵌入式文件系统

OpenHarmony的技术特点：可裁剪、跨终端、易开发

非抢占式内核VS抢占式内核：

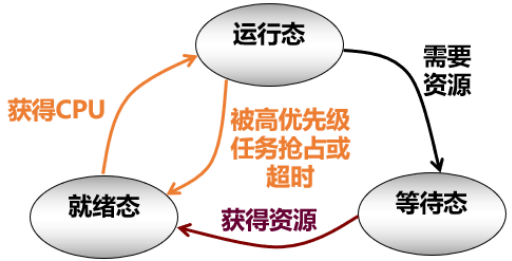
-非抢占式内核称为合作型多任务（进程）处理，要求每个任务在程序代码执行完毕后自我放弃CPU的所有权。非抢占式内核的最大缺陷在于响应时间。高优先级的任务已经进入就绪态，但还不能运行，直到当前运行的任务释放CPU为止。

-抢占式内核可以保证最高优先级的任务就绪必然得到CPU的控制权 。当一个运行着的任务使另一个比它优先级高的任务进入了就绪态，则当前任务的CPU控制权就会被抢占，那个高优先级的任务立刻获得CPU的控制权。

可重入函数

- 定义：如果一个函数的代码能够同时被多个任务并发地调用（分享），并且在调用该函数时这些任务之间不会产生数据干扰错误，那么这个函数就是可重入函数。
- 意义：任意时刻被中断后再继续运行不会丢失数据
- 可重入函数的特征：1）使用本地数据；2）不返回静态数据的指针，所有数据都由函数的调用者提供；3）不为连续的调用持有静态数据；4）通过制作全局数据的本地拷贝来保护全局数据；5）不调用任何不可重入函数
- 不可重入函数的特征：1）函数体内使用了静态的数据结构；2）函数体内调用了malloc()或者free()函数；3)函数体内调用了标准I/O函数

任务：任务（Task）通常为进程（Process）和线程（Thread）的统称。任务是调度的基本单位。
任务的内容：1）代码：一段可执行的程序；2）数据：程序所需要的相关数据（变量、工作空间、缓冲区等）；3）堆栈；4）程序执行的上下文环境。
任务上下文环境包括了实时内核管理任务以及处理器执行任务所需要的所有信息：1）任务优先级；2）任务的状态等实时内核所需要的信息；3）处理器的各种寄存器的内容（hardware context）：程序计数器、堆栈指针、通用寄存器等的内容。
任务的上下文环境通过任务控制块（TCB）来体现
任务的三个基本状态：等待、就绪、执行。
任务状态与变迁：



空闲任务： 当没有其他任务准备运行时执行空闲任务。空闲任务总是被设置为最低优先级。 空闲任务由操作系统创建。 应用软件永远无法删除空闲任务。
为节约内存，任务数量通常需要进行预先配置：按照配置的任务数量初始化任务控制块，一个任务对应一个初始的任务控制块，形成一个空闲任务控制块链。在任务创建时，实时内核从空闲任务控制块链中为任务分配一个任务控制块。随后对任务的操作，都是基于对应的任务控制块来进行的。当任务被删除后，对应的任务控制块又会被实时内核回收到空闲任务控制块链。

任务切换:**保存当前任务的上下文，并恢复需要执行的任务的上下文的过程。当前正在运行的任务的上下文就需要通过该任务的任务控制块保存起来；把需要投入运行的任务的上下文从对应的任务控制块中恢复出来。**

任务切换基本步骤:1)保存任务上下文环境;2)更新当前运行任务的控制块内容，将其状态改为就绪或等待状态;3)将任务控制块移到相应队列（就绪队列或等待队列）;4)选择另一个任务进行执行(调度);5)改变需投入运行任务的控制块内容，将其状态变为运行状态;6)恢复需投入运行任务的上下文环境

任务切换的时机：中断、自陷；运行任务时因缺乏资源而被阻塞；时间片轮转调度；高优先级任务处于就绪状态时。

- 当I/O中断发生的时候：如果I/O活动是一个或多个任务正在等待的事件，内核将把相应的处于等待状态的任务转换为就绪状态。同时，内核还将确定是否继续执行当前处于运行状态的任务，或是用高优先级的就绪任务抢占该任务。
- 自陷：由于执行任务中当前指令所引起，将导致实时内核处理相应的错误或异常事件，并根据事件类型，确定是否进行任务的切换
- 运行任务因缺乏资源而被阻塞：任务执行过程中进行I/O操作时（如打开文件），如果此前该文件已被其他任务打开，将导致当前任务处于等待状态，而不能继续执行
- 时间片轮转调度时：内核将在时钟中断处理程序中确定当前正在运行任务的执行时间是否已经超过了设定的时间片。如果超过了时间片，实时内核将停止当前任务的运行，把当前任务的状态变为就绪状态，并把另一个任务投入运行
- 高优先级任务处于就绪时：如果采用基于优先级的抢占式调度算法，将导致当前任务停止运行，使更高优先级的任务处于运行状态

任务切换的时间=**保存当前运行任务上下文的时间+选择下一个任务的调度时间+将要运行任务的上下文的恢复时间**

保存和恢复的时间取决于任务上下文的定义和处理器的速度。这两个内容一旦确定，保存和恢复的时间也就是确定的。

强实时内核要求调度过程所花费的时间是确定的，即不能随系统中就绪任务的数目而变化。然而普通操作系统的调度算法随着任务数量的增加，找出优先级最高的任务TCB的时间也会增加。

为提高实时内核的确定性，可采用一种被称为优先级位图的就绪任务处理算法。

位优先级算法：

数据结构：

char OSRdyGrp; #优先级就绪组

char OSRdyTbl[8];#优先级就绪表

char OSMapTbl[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

char OSUnMapTbl[256];#优先级判定表

```
INT8U  const OSUnMapTbl [ ] = {
    0,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x00 to 0x0F   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x10 to 0x1F   */
    5,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x20 to 0x2F   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x30 to 0x3F   */
    6,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x40 to 0x4F   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x50 to 0x5F   */
    5,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x60 to 0x6F   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x70 to 0x7F   */
    7,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x80 to 0x8F   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0x90 to 0x9F   */
    5,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xA0 to 0xAF   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xB0 to 0xBF   */
    6,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xC0 to 0xCF   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xD0 to 0xDF   */
    5,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xE0 to 0xEF   */
    4,  0,  1,  0,  2,  0,  1,  0,  3,  0,  1,  0,  2,  0,  1,  0,    /* 0xF0 to 0xFF   */
};
```

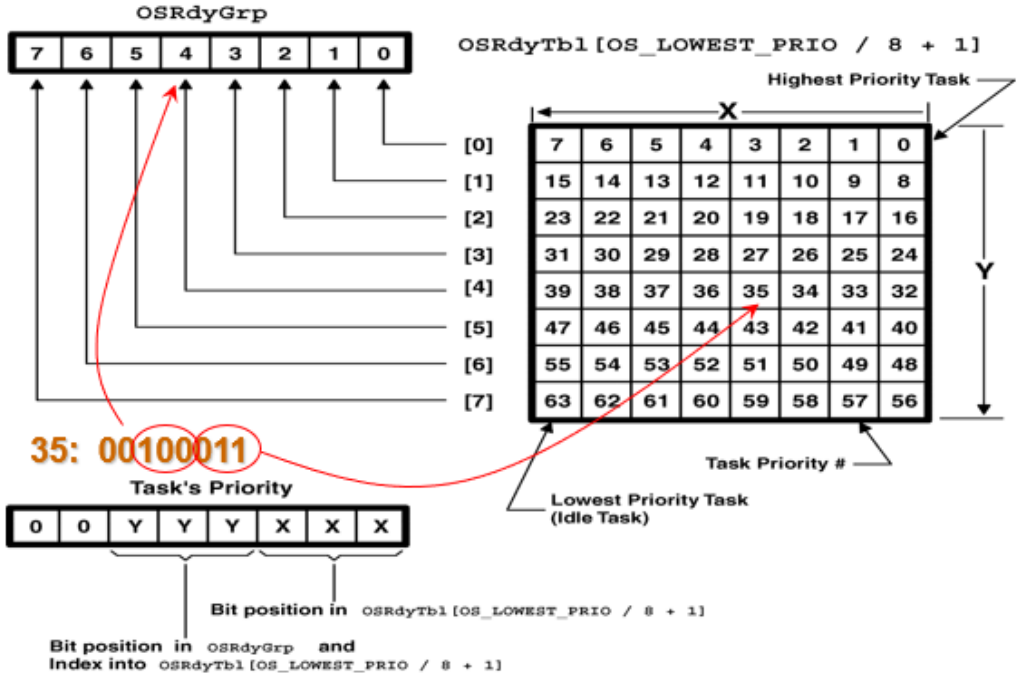
OSRdyGrp contains 0x68

OSRdyTbl[3] contains 0xE4

```
3 = OSUnMapTbl[ 0x68 ] ;
2 = OSUnMapTbl[ 0xE4 ] ;
26 = (3 << 3) + 2;
```

多任务参与时怎么选定最高优先级任务？以OSRdyGrp或是OSRdyTbl[]数组元素的值为索引，获取该值对应二进制表示中1出现的最低二进制位的序号（0-7）。

优先级判定表char OSUnMapTbl[256]



为提高实时内核的确定性，可采用一种被称为优先级位图的就绪任务处理算法。

位优先级算法：

1. 任务进入就绪态：

```
OSRdyGrp |= OSMapTbl[priority >> 3];
OSRdyTbl[priority >> 3] |= OSMapTbl[priority & 0x07];
```

2. 任务退出就绪态：

```
if((OSRdyTbl[priority >> 3] &= ~OSMapTbl[priority & 0x07]) = = 0)
        OSRdyGrp &= ~OSMapTbl[priority >> 3];
```

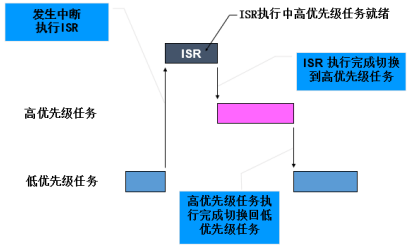
3. 多任务条件下，获取进入就绪态的最高优先级：

```
high3Bit = OSUnMapTbl[OSRdyGrp];
low3Bit = OSUnMapTbl[OSRdyTbl[high3Bit]];
priority = (high3Bit << 3) + low3Bit;
```

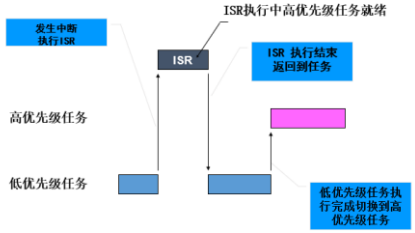
只需要将任务按优先级进行组织，以优先级为数组元素下标，通过TCB数组和位优先级算法得到的最高优先级即可找到进入就绪态的具有最高优先级任务的TCB。

调度算法：

1. 抢占式调度 /非抢占式调度。



抢占调度

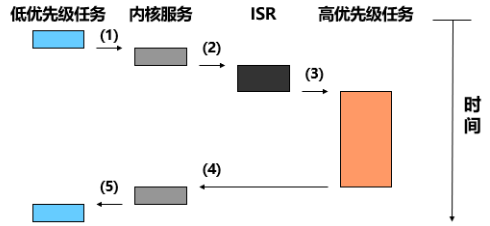


非抢占调度

内核可抢占与不可抢占：执行内核提供的系统调用过程中，是否可以被中断打断。

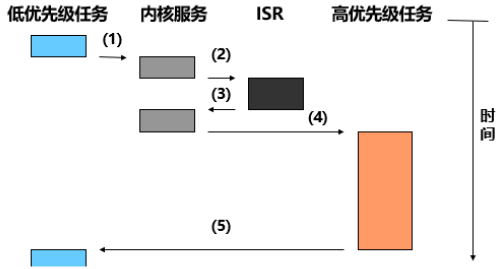
- 可抢占内核：即使正在执行的是内核服务函数，也能响应中断，并且中断服务程序退出时能进行任务重新调度。**如果有优先级更高的任务就绪，就立即让高优先级任务运行，不要求回到被中断的任务将未完成的系统调用执行完。**
- 不可抢占内核：不可抢占内核有两种情况1）内核服务函数不能被中断。2）能被中断但是不能进行任务重新调度。

(1)低优先级任务调用内核服务
(2)内核服务过程中系统发生中断，在允许中断的情况下，进入中断服务程序（ISR）
(3)中断服务程序完成后，内核调度新就绪的高优先级任务运行
(4)高优先级任务运行完成或者因为其它原因阻塞，系统回到先前被低优先级任务调用的、尚未完成的内核服务中
(5)内核服务完成，返回到低优先级任务中，低优先级任务继续执行



可抢占内核

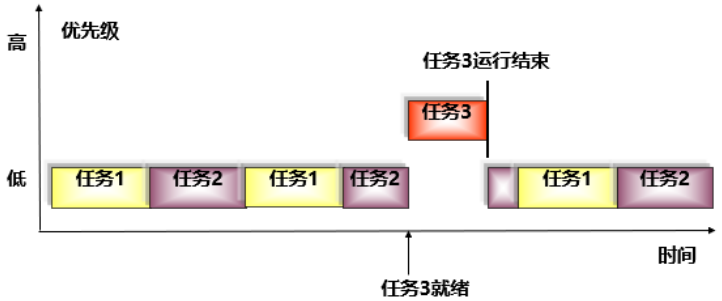
(1)低优先级任务调用内核服务
(2)内核服务过程中，系统发生中断，在允许中断的情况下，进入中断服务程序（ISR）
(3)中断服务程序完成后，回到内核服务中
(4)内核服务完成，进行任务调度，切换到新就绪的高优先级任务
(5)高优先级任务运行完成或者因为其它原因阻塞，内核调度低优先级任务，低优先级任务恢复执行



不可抢占内核（允许中断）

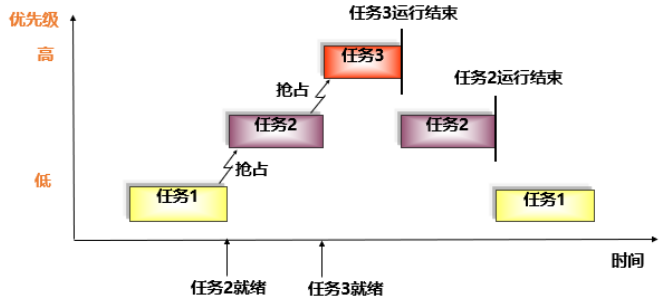
2. 基于优先级的可抢占调度：如果出现具有更高优先级的任务处于就绪状态时，当前任务将停止运行，把CPU的控制权交给具有更高优先级的任务，使更高优先级的任务得到执行。实时内核需要确保CPU总是被具有最高优先级的就绪任务所控制。

3. 时间片轮转调度：当有两个或多个就绪任务具有相同的优先级，且优先级最高时采用时间片轮转调度算法。



在时间片轮转调度方式下的任务运行情况

- 任务1和任务2具有相同的优先级，按照时间片轮转的方式轮流执行。
- 当高优先级任务3就绪后，正在执行的任务2被抢占，高优先级任务3得到执行。
- 当任务3完成运行后，任务2才重新在未完成的时间片内继续执行。
- 随后任务1和任务2又按照时间片轮转的方式执行。

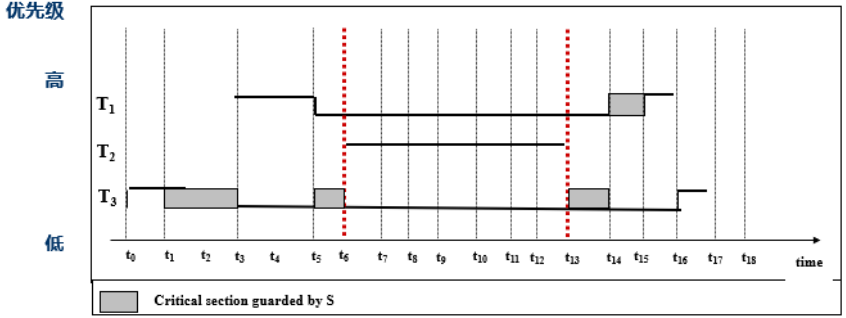


在可抢占调度方式下的任务运行情况

任务1被具有更高优先级的任务2所抢占，然后任务2又被任务3抢占。当任务3完成运行后，任务2继续执行。当任务2完成运行后，任务1才又得以继续执行。

理想情况下，高优先级任务就绪后，能够立即抢占低优先级任务而得到执行。但在有多个任务需要使用共享资源的情况下，可能会出现高优先级任务被低优先级任务阻塞，并等待低优先级任务执行的情况。

优先级反转：高优先级任务需要等待低优先级任务释放资源，而低优先级任务又正在等待中等优先级任务的现象。
优先级反转产生的原因：应用程序中的同步互斥机制采用信号量和锁来避免死锁或读取脏数据。直接应用这些同步互斥机制将导致系统中出现不定时间长度的优先级反转和任务可调度性比较低的情况。



- 假定T_1和T_3通过信号量S共享一个数据结构。
- 在时刻t_1，任务T_3获得信号量S，开始执行临界区代码。
- 在T_3执行临界区代码的过程中，高优先级任务T_1就绪，抢占任务T_3，并在随后试图使用共享数据，但该共享数据已被T_3通过信号量S加锁。在这种情况下，会期望具有最高优先级的任务T_1被阻塞的时间不超过任务T_3执行完整个临界区的时间。
- 但事实上，这种阻塞时间的长度是无法预知的。这主要是由于任务T_3还可能被具有中等优先级的任务T_2所阻塞，使得T_1也需要等待T_2和其他中等优先级的任务释放CPU资源。

解决优先级反转现象的常用协议为：
1. 优先级继承协议：
2. 优先级天花板协议

1. 优先级继承协议：

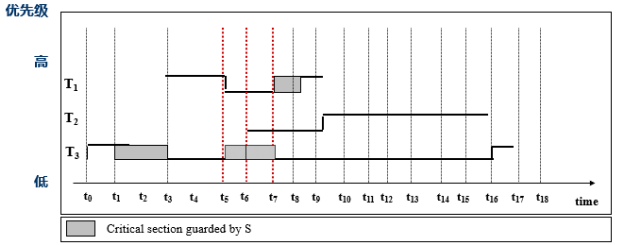
当一个任务阻塞了一个或多个高优先级任务时，该任务将不使用其原来的优先级，而使用被该任务所阻塞的所有任务的最高优先级作为其执行临界区的优先级。当该任务退出临界区时，又恢复到其最初的优先级。

采用优先级继承协议，系统运行前就能够确定任务的最大阻塞时间。

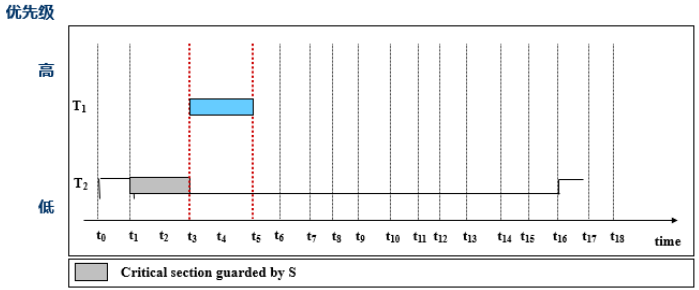
But，优先级继承协议存在以下两方面的问题：

1）优先级继承协议本身不能避免死锁的发生；

2）在优先级继承协议中，任务的阻塞时间虽然是有界的，但由于可能出现阻塞链，使得任务的阻塞时间可能会很长。

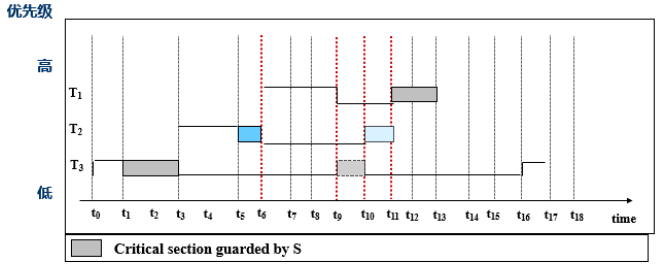


如果任务T_1被T_3阻塞，优先级继承协议要求任务T_3以任务T_1的优先级执行临界区。这样，任务T_3在执行临界区的时候，原来比T_3具有更高优先级的任务T_2就不能抢占T_3了。当T_3退出临界区时，T_3又恢复到其原来的低优先级，使任务T_1又成为最高优先级的任务。这样任务T_1会抢占任务T_3而继续获得CPU资源，而不会出现T_1无限期被任务T_2所阻塞的情形。



假定在时刻t_1，任务T_2获得信号量S_2，进入临界区。

在时刻t_3，任务T_2又试图获得信号量S_1，但一个高优先级任务T_1在这个时候就绪，抢占任务T_2并获得信号量S_1，接下来任务T_1又试图获得信号量S_2。这样就出现了死锁现象。



假定任务T_1需要顺序获得信号量S_1和S_2；

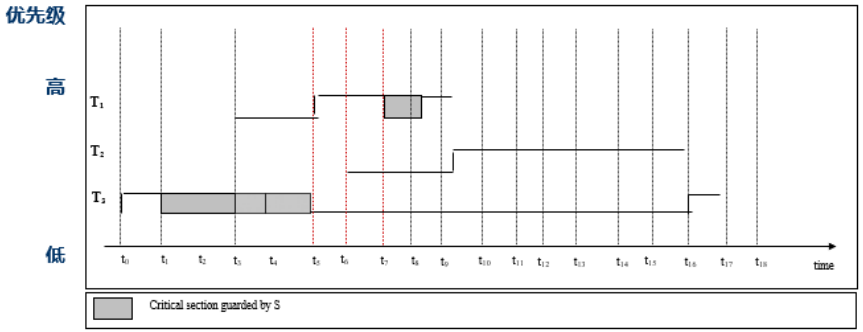
任务T_3在S_1控制的临界区中被T_2抢占，然后T_2进入S_2控制的临界区。

这个时候，任务T_1被激活而获得CPU资源，发现信号量S_1和S_2都分别被低优先级任务T_2和T_3加锁，使得T_1将被阻塞两个临界区，需要先等待任务T_3释放信号量S_1，然后等待任务T_2释放信号量S_2，这样就形成了关于任务T_1的阻塞链。

2. 优先级天花板协议 ：

优先级天花板协议用来解决优先级继承协议中存在的死锁和阻塞链问题。

内容：对于控制临界区的信号量，设置信号量的优先级天花板为可能申请该信号量的所有任务中具有最高优先级任务的优先级；如果任务成功获得信号量，任务的优先级将被抬升为信号量的优先级天花板；任务执行完临界区，释放信号量后，其优先级恢复到其最初的优先级；如果任务不能获得所申请的信号量，任务将被阻塞。



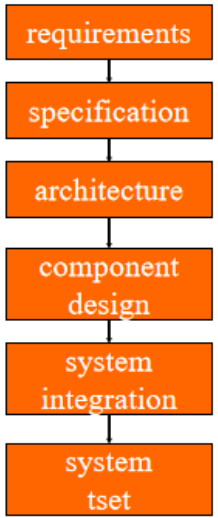
- 假设T_1、T_2、T_3的优先级分别为p1、p2、p3，并且T_1和T_3通过信号量S共享一个临界资源。根据优先级天花板协议，信号量S的优先级天花板为p1。
- 假定在时刻t_1，T_3获得信号量S，按照优先级天花板协议，T_3的优先级将被抬升为信号量S的优先级天花板p1，直到T_3退出临界区。这样，T_3在执行临界区的过程中，T_1和T_2都不能抢占T_3，确保T_3能尽快完成临界区的执行，并释放信号量S。
- 当T_3退出临界区后，T_3的优先级又回落为p3。如果在T_3执行临界区的过程中，任务T_1或T_2已经就绪，则此时T_1或T_2将抢占T_3的执行。

两个协议的比较：

-关于执行效率的比较 ：优先级继承协议可能多次改变占有某临界资源任务的优先级，而优先级天花板协议只需改变一次。从这个角度看，优先级天花板协议的效率高，因为若干次改变占有资源的任务的优先级会引入更多的额外开销，导致任务执行临界区的时间增加。

-对程序运行过程影响程度的比较：优先级天花板协议的特点是一旦任务获得某临界资源，其优先级就被抬升到可能的最高程度，不管此后在它使用该资源的时间内是否真的有高优先级任务申请该资源，这样就有可能影响某些中间优先级任务的完成时间。但在优先级继承协议中，只有当高优先级任务申请已被低优先级任务占有的临界资源这一事实发生时，才抬升低优先级任务的优先级，因此优先级继承协议对任务执行流程的影响相对要较小。

- 嵌入式系统开发流程
  - 需求
  - 规格说明
  - 体系结构
  - 组件（软硬件详细设计）
  - 系统集成
  - 系统测试



确定所实现的系统需要采用哪些主要部件，确定硬件平台和软件平台是在体系结构设计阶段

硬件功耗来源

- CMOS电路功耗$P_{总}$有以下近似计算公式：
  - $P_{总}= P_{动态}+ P_{直流开关功耗} + P_{静态}$
- $P_{动态}= aC_L fV_{dd}^2$
  - 其中：a为开关系数，即每个时钟周期中发生状态变化器件的个数，C_L为负载电容，f为电路的工作频率，V_{dd}为电路的电源电压值。
- $P_{直流开关功耗} = V_{dd}I_{st}$
  - 其中：I_{st}为短路电流值。
- $P_{静态}= V_{dd}I_{leakage}$
  - 其中：$I_{leakage}$为漏电流值。

电压是影响功耗的主要因素