总结

一、考试形式：闭卷，笔试
基本概念、基本原理、设计应用技术
范围：以课件涵盖内容为主。

二、题型：
1. 简答题
2. 问答题
3. 设计题

三、复习
1. 嵌入式系统的定义，特点。分类。典型应用。**
嵌入式系统（嵌入式计算机系统）定义：

- IEEE（从应用上考虑）：嵌入式系统是“用于控制、监视或者辅助操作机器和设备的装置”。

嵌入式系统是软件和硬件的综合体，还可以涵盖机电等附属装置。

- 国内普遍被认同的定义：

嵌入式系统是“以应用为中心，以计算机技术为基础，软硬件可裁减，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统”。

嵌入式系统就是一个具有特定功能或用途的隐藏在某种设备中的计算机软硬件集合体，没有固定的特征形状。

- 其他：

看不见的计算机，一般不能被用户编程，它有一些专用的I/O设备，对用户的接口是应用专用的。

嵌入式系统的特点
•形式多样、面向特定应用的
- 一般用于特定的任务，其硬件和软件都必须高效率地设计，量体裁衣、去除冗余，而通用计算机则是一个通用的计算平台。
- 它通常都具有低功耗、体积小、集成度高等特点，能够把通用微处理器中许多由板卡完成的任务集成在芯片内部。
- 嵌入式软件是应用程序和操作系统两种软件的一体化程序。

•得到多种类型的处理器和处理器体系结构的支持
- 通用计算机采用少数的处理器类型和体系结构，而且主要掌握在少数大公司手里。
- 嵌入式系统可采用多种类型的处理器和处理器体系结构，有上千种的嵌入式微处理器和几十种嵌入式微处理器体系结构可以选择。
- 在嵌入式微处理器产业链上，IP设计、面向应用的特定嵌入式微处理器的设计、芯片的制造已形成巨大的产业。分工协作，形成多赢模式。

•极其关注成本
- 嵌入式系统通常需要注意的成本是系统成本，特别是量大的消费类数字化产品，其成本是产品竞争的关键因素之一。

- 嵌入式的系统成本包括:
  - 一次性的开发成本NRE(Non-Recurring Engineering)成本
  - 产品成本:硬件BOM、外壳包装和软件版税等
  - 批量产品的总体成本=NRE成本+每个产品成本*产品总量
  - 每个产品的最后成本=总体成本/产品总量=NRE成本/产品总量+每个产品成本

•有实时性和可靠性的要求

- 一方面大多数实时系统都是嵌入式系统。
- 另一方面嵌入式系统多数有实时性的要求，软件一般是固化运行或直接加载到内存中运行，具有快速启动的功能。
- 嵌入式系统一般要求具有出错处理和自动复位功能，特别是对于一些在极端环境下运行的嵌入式系统而言，其可靠性设计尤其重要。
- 在大多数嵌入式系统的软件中一般都包括一些机制，比如硬件的看门狗定时器，软件的内存保护和重启动机制。

•使用的操作系统一般是适应多种处理器、可剪裁、轻量型、实时可靠、可固化的嵌入式操作系统

- 由于嵌入式系统应用的特点，像嵌入式微处理器一样，嵌入式操作系统也是多姿多彩的。
- 大多数商业嵌入式操作系统可同时支持不同种类的嵌入式微处理器。可根据应用的情况进行剪裁、配置。
- 嵌入式操作系统规模小，所需的资源有限如内核规模在几十KB，能与应用软件一样固化运行。
- 一般包括一个实时内核，其调度算法一般采用基于优先级的可抢占的调度算法。
- 高可靠嵌入式操作系统：时、空、数据隔离。

•开发需要专门工具和特殊方法

- 由于嵌入式系统资源有限，一般不具备自主开发能力，产品发布后用户通常也不能对其中的软件进行修改，必须有一套专门的开发环境。
- 该开发环境包括专门的开发工具（包括设计、编译、调试、测试等工具），采用交叉开发的方式进行

嵌入式系统的分类

- 按嵌入式处理器的位数来分类
  - 4位嵌入式系统——目前已大量应用
  - 8位嵌入式系统——目前已大量应用
  - 16位嵌入式系统——目前已大量应用
  - 32位嵌入式系统——正成为主流发展趋势
  - 64位嵌入式系统——高度复杂的、高速的嵌入式系统已开始采用
- 按应用来分类：信息家电类、移动终端类、汽车电子类、工业控制类、通信类等
- 按速度分类
  - 强实时系统，其系统响应时间在毫秒或微秒级。
  - 一般实时系统，其系统响应时间在几秒的数量级上，其实时性的要求比强实时系统要差一些。
  - 弱实时系统，其系统响应时间约为数十秒或更长。这种系统的响应时间可能随系统负载的轻重而变化。
- 按确定性来分类：根据确定性的强弱，分为硬实时、软实时系统：

    - 硬实时：系统对系统响应时间有严格的要求，如果系统响应时间不能满足，就要引起系统崩溃或致命的错误。
    - 软实时：系统对系统响应时间有要求，但是如果系统响应时间不能满足，不会导致系统出现致命的错误或崩溃。
- 按嵌入式系统软件复杂程度来分类
    - 循环轮询系统
    - 有限状态机系统
    - 前后台系统
    - 单处理器多任务系统
    - 多处理器多任务系统

典型应用：

环境工程与自然：水文资料实时监测，防洪体系及水土质量监测、堤坝安全，地震监测网，实时气象信息网，水源和空气污染监测。
信息家电：空调，电视、音响、电子表、冰箱，防火防盗器
智能玩具：遥控汽车
汽车电子：汽车（内嵌 GPS 模块）、飞机
智能仪表
军用电子
通信设备：手机
工控设备：工业过程控制、数字机床、电力系统、电网安全、电网设备监测、石油化工系统
机器人：智能宠物
航天领域：火星探测器（火星车）
电子商务：公共交通无接触智能卡、自动售货机，各种智能 ATM 终端、公共电话卡发行系统

2. 嵌入式系统的基本组成。

- 嵌入式系统一般由嵌入式硬件和软件组成
- 硬件：以微处理器为核心集成存储器和系统专用的输入/输出设备
- 软件：初始化代码及驱动、嵌入式操作系统和应用程序等，这些软件有机地结合在一起，形成系统特定的一体化软件。

3. 嵌入式系统设计

嵌入式系统面临挑战

嵌入式系统要解决的主要问题

- 需要用什么样的系统结构来实现？
- 如何满足系统接口要求，嵌入式应用直接和系统接口输入输出信息？
- 如何满足时限要求，如何处理多项功能在时间上的协调一致关系？
- 如何保证系统可靠地工作？
- 如何降低系统的功耗？
- 如何使系统可升级？

传统开发过程与软硬件协同设计

嵌入式系统的设计过程的基本流程：需求分析→规格说明→体系结构→构件设计→系统集成（分为自顶向下、自底向上设计）

需求分析（功能和非功能需求）

- Plain language description of what the user wants and expects to get.
- May be developed in several ways:
    - talking directly to customers;
    - talking to marketing representatives;
    - providing prototypes to users for comment.

Specification
- A more precise description of the system:
    - should not imply a particular architecture;
    - provides input to the architecture design process.
- May include functional and non-functional elements.
- May be executable可执行的 or may be in mathematical form for proofs.

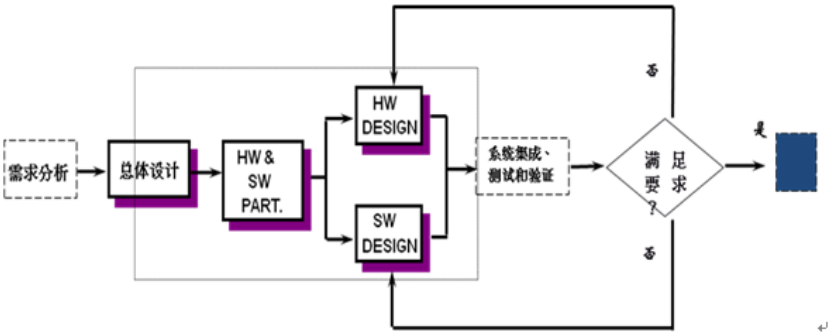
体系结构设计
- What major components go satisfying the specification?
- Hardware components:
    - CPUs, peripherals外围设备, etc.
- Software components:
    - major programs and their operations.
- Must take into account functional and non-functional specifications.

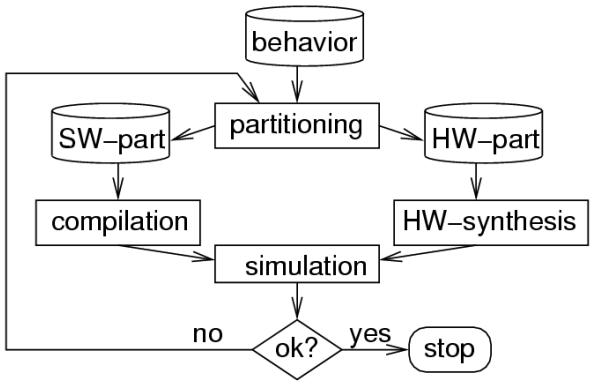
Designing hardware and software components
- Must spend time architecting the system before you start coding.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.

系统结构设计
- 系统如何实现设计说明书描述的功能
- 基于组件的系统结构
- 软件/硬件划分
    - 嵌入式系统中软件和硬件协同完成系统的功能
    - 软件硬件划分通常由速度、灵活性以及开销来决策

系统集成与测试
- Put together the components.
    - Many bugs appear only at this stage.
- Have a plan for integrating components to uncover bugs quickly, test as much functionality as early as possible.

传统开发过程





- 传统软硬件设计过程的基本特征:
  - 系统在一开始就被划分为软件和硬件两大部分
  - 软件和硬件独立进行开发设计
  - “Hardware first” approach often adopted
- 隐含的一些问题:
  - 软硬件之间的交互受到很大限制
    - 凭经验划分软硬件
    - 软硬件之间的相互性能影响很难评估
  - 系统集成相对滞后，NRE较大
- 因此:
  - Poor quality designs（设计质量差）
  - Costly modifications（设计修改难）
  - Schedule slippages（研制周期不能有效保障）

软硬件协同设计

定义

- The meeting of system-level objectives by exploiting 利用 the trade-offs 交互、协定 between（the synergism of）hardware and software in a system through their concurrent design 协同设计

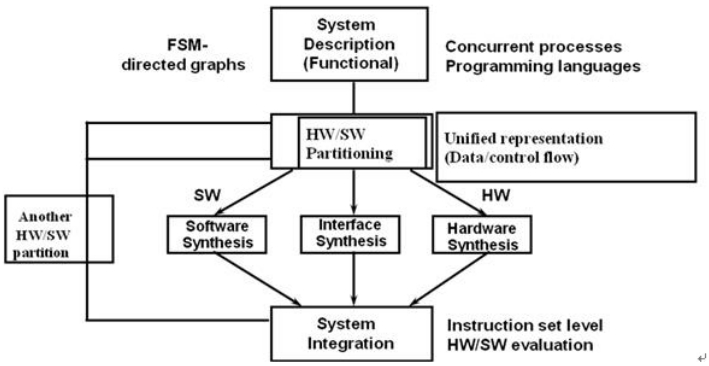
主要概念

- Concurrent（并发）: hardware and software developed at the same time on parallel paths
- Integrated（一体化）: interaction between hardware and software developments to produce designs that meet performance criteria 标准 and functional specifications

软硬件协同设计的基本需求

- 统一的软硬件描述方式
  - 软硬件支持统一的设计和分析工具（技术）
  - 允许在一个集成环境中仿真（评估）系统软硬件设计
  - 支持系统任务在软件和硬件设计之间的相互移植
- 交互式软硬件划分技术
  - 允许多个不同的软硬件划分设计进行仿真和比较
  - 辅助最优系统实现方式决策
  - 将软硬件划分应用到模块设计，以便最佳地实现系统的设计指标(功能和性能目标)
- 完整的软硬件模型基础
  - 支持在设计过程中的几个阶段的综合评价
  - 支持软硬件逐步的开发和集成
- 正确的验证方法
  - 确保系统设计达到的目标要求

典型的软硬件协同



软硬件划分**

• 嵌入式系统的设计涉及硬件与软件部件，设计中必须决定什么功能由硬件实现，什么功能由软件实现。
• 硬件和软件具有双重性
• 软硬件变动对系统的决策造成影响
• 划分和选择需要考虑多种因素
• 硬件和软件的双重性是划分决策的前提

通常由软件实现的部分

• 操作系统功能
  • 任务调度
  • 资源管理
  • 设备驱动
• 协议栈
  • TCP / IP
• 应用软件框架
• 除基本系统、物理接口、基本逻辑电路，许多由硬件实现的功能都可以由软件实现。

双重性部分

- 算法
  - 加密 / 解密
  - 编码 / 解码
  - 压缩 / 解压……
- 数学运算
  - 浮点运算， FFT， ……

软硬件技术对系统结构的影响

- 硬软件设计的趋势——融合、渗透
  - 硬件设计的软件化
    - VHDL, Verilog
    - HANDL-C
  - 软件实现的硬件化
    - 各种算法的ASIC
- 对系统设计的影响——协同设计
  - 增加灵活性
  - 增加了风险

软硬件划分

结构规划 - 处理器类型, 软硬件之间的接口类型, 等.

划分目的 - 满足系统速度,延迟, 体积,成本等方面的要求.

划分策略 - high level partitioning by hand, automated partitioning using various techniques...

调度：

Operation scheduling in hardware

Instruction scheduling in compilers

Process scheduling in operating systems

4. 嵌入式硬件系统基础。

hardware in a loop（回环）

嵌入式硬件系统基础

嵌入式系统的硬件是以嵌入式微处理器为核心，主要由嵌入式微处理器、总线、存储器、输入/输出接口和设备：networking, sensors, actuators 螺线管等组成。

4.1 嵌入式微处理器基础

4.1.1 嵌入式微处理器体系结构

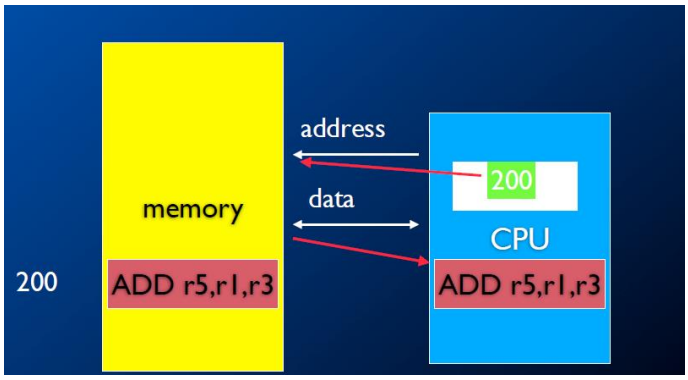
冯诺伊曼结构与哈佛结构

传统的微处理器采用的**冯 诺依曼**结构将指令和数据存放在**同一存储空间**中，**统一编址**，指令和数据通过**同一总线访问**。

- Memory holds data, instructions.
- Central processing unit (CPU) fetches instructions from memory.

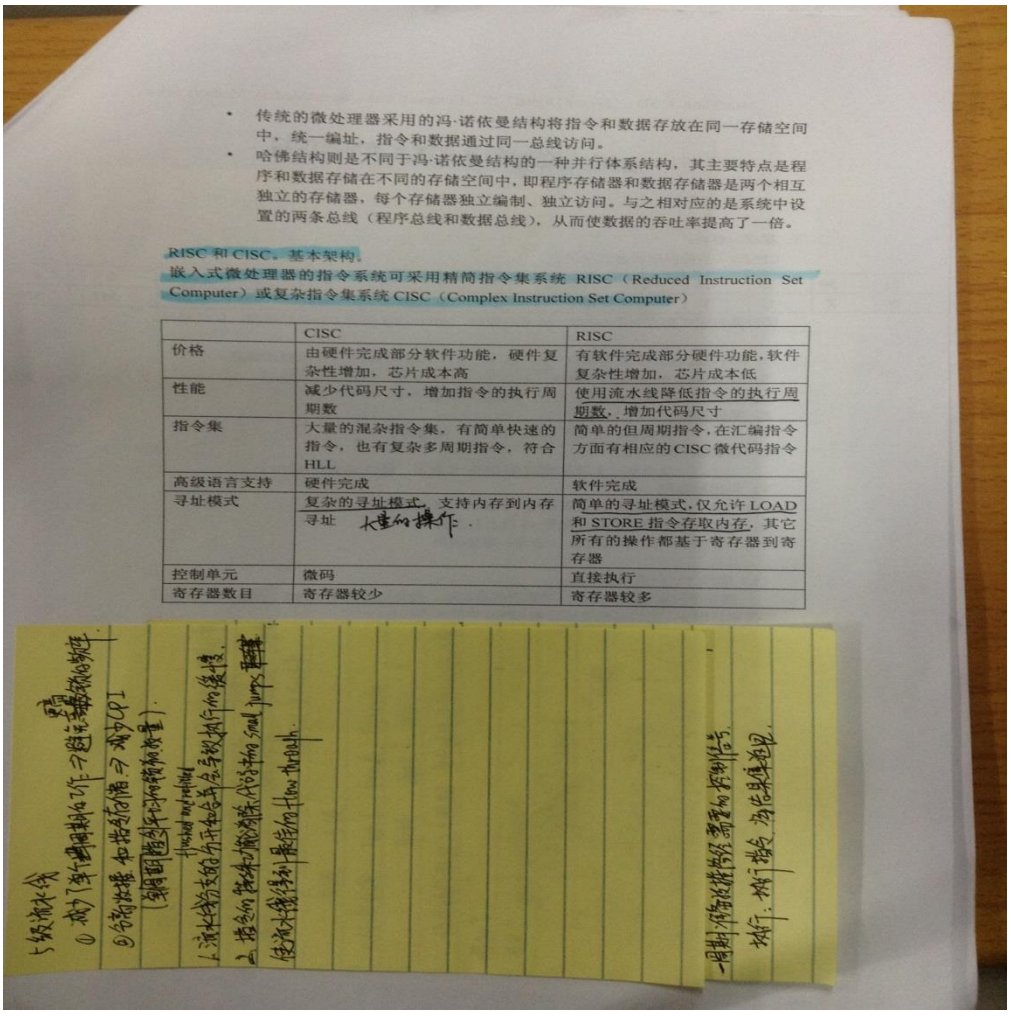
• Separate CPU and memory distinguishes programmable computer.

• CPU registers寄存器 help out: program counter (PC), instruction register (IR), general-purpose registers, etc.



哈佛结构则是不同于冯·诺依曼结构的一种**并行体系结构**，其主要特点是程序和数据存储在**不同的存储空间**中，即程序存储器和数据存储器是两个相互独立的存储器，每个存储器**独立编制、独立访问**。与之相对应的是系统中设置的**两条总线**（程序总线和数据总线），从而使数据的吞吐率提高了一倍。

CISC 与 RISC

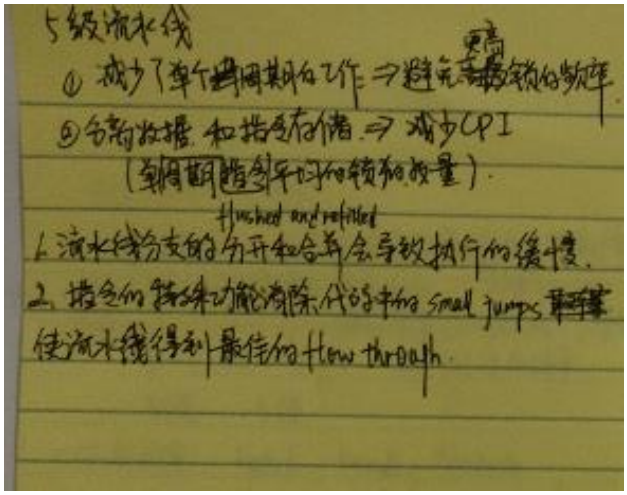


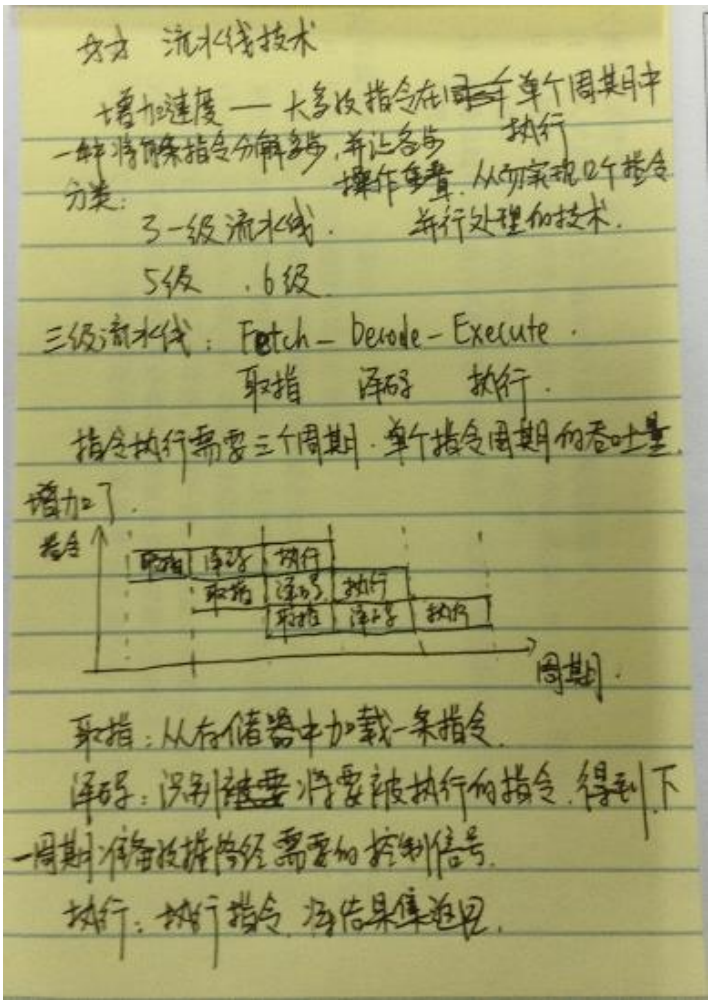
- 传统的微处理器采用的冯·诺依曼结构将指令和数据存放在同一存储空间中，统一编址，指令和数据通过同一总线访问。
- 哈佛结构则是不同于冯·诺依曼结构的一种并行体系结构，其主要特点是程序和数据存储在不同的存储空间中，即程序存储器和数据存储器是两个相互独立的存储器，每个存储器独立编制、独立访问。与之相对应的是系统中设置的两条总线（程序总线和数据总线），从而使数据的吞吐率提高了一倍。

RISC 和 CISC。基本架构：
嵌入式微处理器的指令系统可采用精简指令集系统 RISC（Reduced Instruction Set Computer）或复杂指令集系统 CISC（Complex Instruction Set Computer）

	CISC	RISC
价格	由硬件完成部分软件功能，硬件复杂性增加，芯片成本高	有软件完成部分硬件功能，软件复杂性增加，芯片成本低
性能	减少代码尺寸，增加指令的执行周期数	使用流水线降低指令的执行周期数，增加代码尺寸
指令集	大量的混杂指令集，有简单快速的指令，也有复杂多周期指令，符合 HLL	简单的但周期指令，在汇编指令方面有相应的 CISC 微代码指令
高级语言支持	硬件完成	软件完成
寻址模式	复杂的寻址模式，支持内存到内存寻址 大量的操作	简单的寻址模式，仅允许 LOAD 和 STORE 指令存取内存，其它所有的操作都基于寄存器到寄存器
控制单元	微码	直接执行
寄存器数目	寄存器较少	寄存器较多

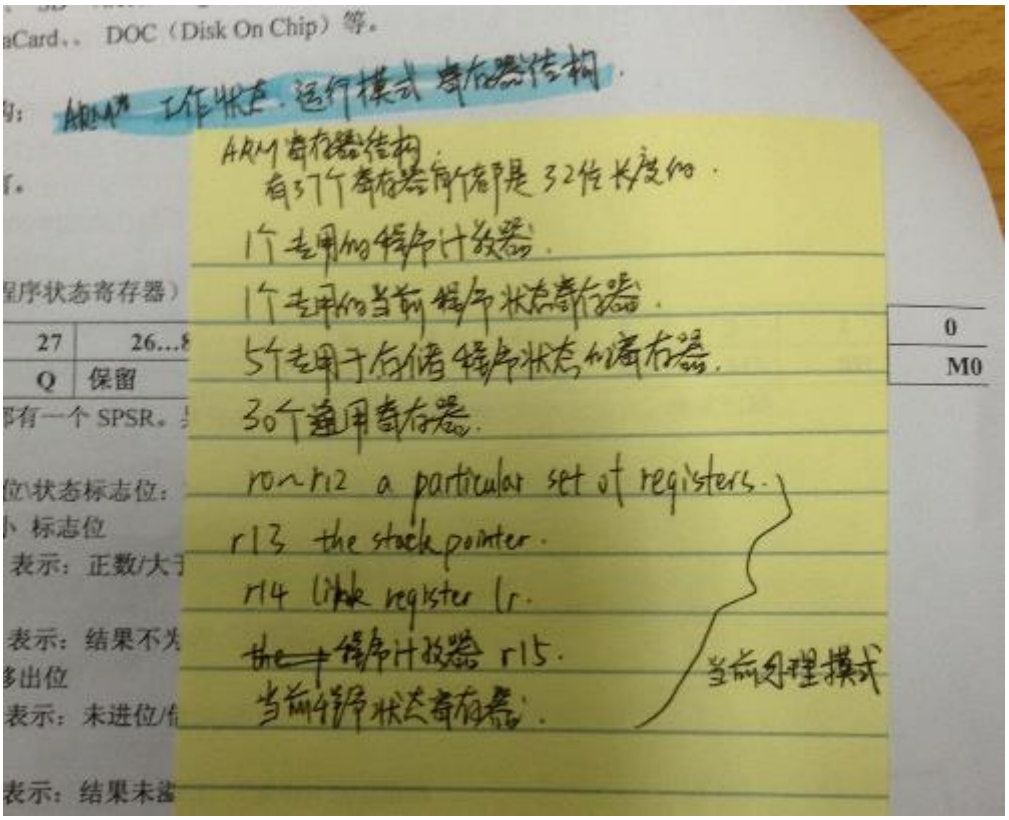
流水线技术*





信息存储的字节顺序（大端和小端）

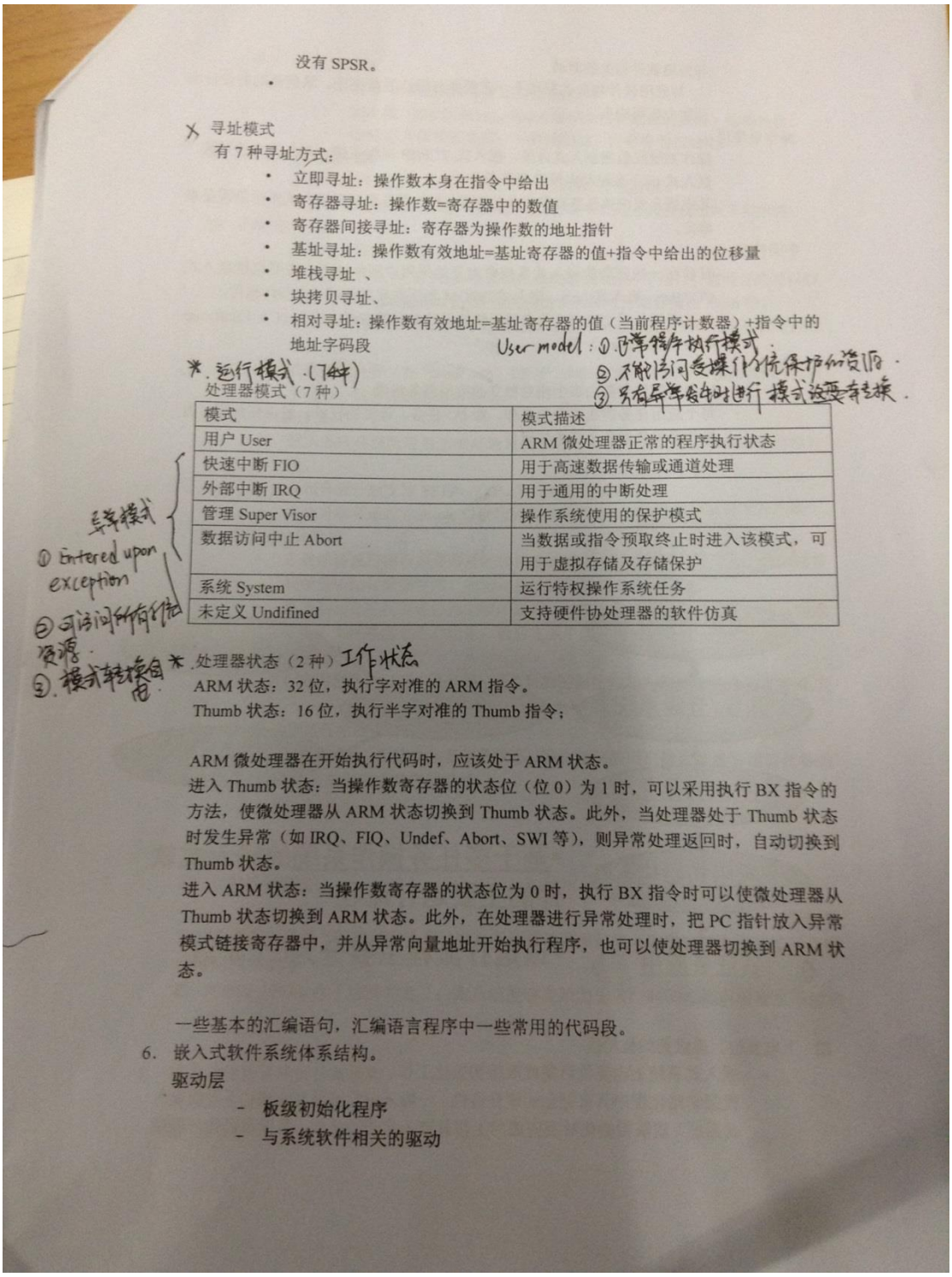
4.1.2 ARM*



工作状态

运行模式

寄存器结构



没有 SPSR。

寻址模式

有 7 种寻址方式：

- 立即寻址：操作数本身在指令中给出
- 寄存器寻址：操作数=寄存器中的数值
- 寄存器间接寻址：寄存器为操作数的地址指针
- 基址寻址：操作数有效地址=基址寄存器的值+指令中给出的位移量
- 堆栈寻址 、
- 块拷贝寻址、
- 相对寻址：操作数有效地址=基址寄存器的值（当前程序计数器）+指令中的地址字码段

处理器模式（7 种）

模式	模式描述
用户 User	ARM 微处理器正常的程序执行状态
快速中断 FIO	用于高速数据传输或通道处理
外部中断 IRQ	用于通用的中断处理
管理 Super Visor	操作系统使用的保护模式
数据访问中止 Abort	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
系统 System	运行特权操作系统任务
未定义 Undifined	支持硬件协处理器的软件仿真

处理器状态（2 种）

ARM 状态：32 位，执行字对准的 ARM 指令。

Thumb 状态：16 位，执行半字对准的 Thumb 指令；

ARM 微处理器在开始执行代码时，应该处于 ARM 状态。

进入 Thumb 状态：当操作数寄存器的状态位（位 0）为 1 时，可以采用执行 BX 指令的方法，使微处理器从 ARM 状态切换到 Thumb 状态。此外，当处理器处于 Thumb 状态时发生异常（如 IRQ、FIQ、Undef、Abort、SWI 等），则异常处理返回时，自动切换到 Thumb 状态。

进入 ARM 状态：当操作数寄存器的状态位为 0 时，执行 BX 指令时可以使微处理器从 Thumb 状态切换到 ARM 状态。此外，在处理器进行异常处理时，把 PC 指针放入异常模式链接寄存器中，并从异常向量地址开始执行程序，也可以使处理器切换到 ARM 状态。

一些基本的汇编语句，汇编语言程序中一些常用的代码段。

6. 嵌入式软件系统体系结构。

驱动层

- 板级初始化程序
- 与系统软件相关的驱动

4.2 嵌入式系统的存储体系

4.2.1 存储器系统

存储器系统的层次结构

4.2.2 ROM 的种类与选型

4.2.3 Flash 的种类与选型*

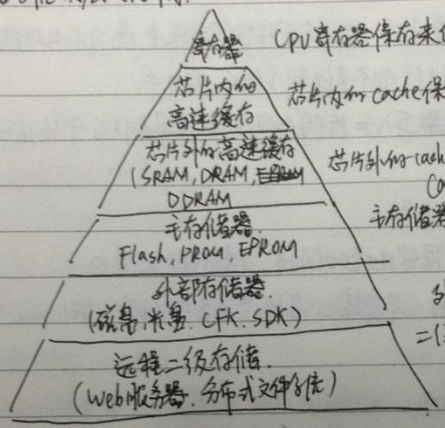
4.2.4 RAM 的种类与选型

Date No.

4.2. 嵌入式系统的存储体系.

4.2.1. 存储器系统.

存储器系统的层次结构:



4.2.2. ROM的种类与选型. (找不到)

4.2.3 Flash的种类与选型 *

NOR Flash: Word-accessible read.

(64~128KB). 块级擦除. 可直接运行装载在NOR Flash里的代码.

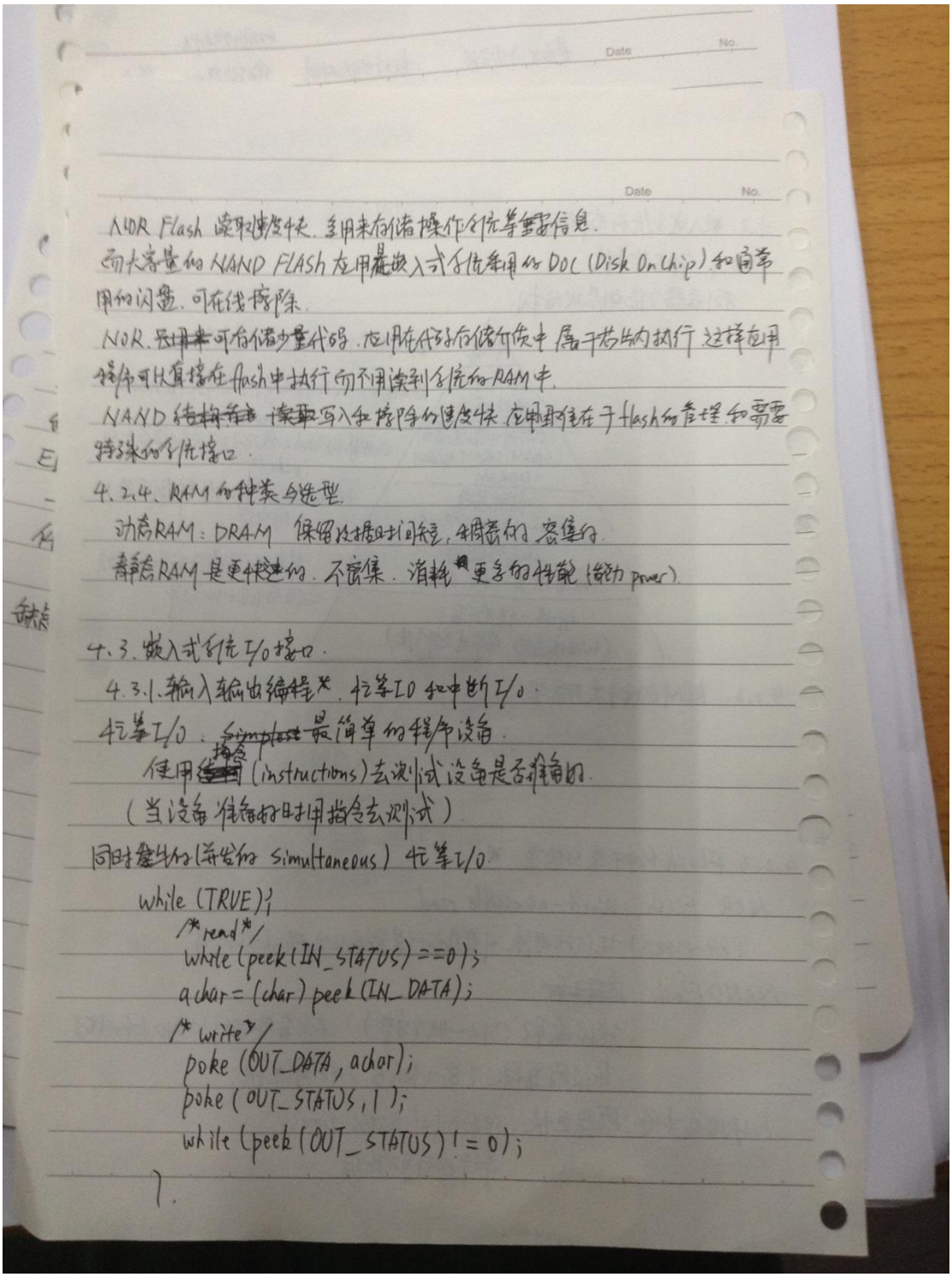
NAND Flash: ~~这句话开头~~

块级读取. (512~4K个字节). 不能直接运行NAND上的代码.

块级擦除. (8~32KB) 时间很快.

NAND价格低. 擦除更快. sequential access times.

串行地存取数据.



NOR Flash 读取速度快，多用来存储操作系统等重要信息。
而大容量的 NAND FLASH 应用最嵌入式系统采用的 DOC (Disk On Chip) 和闪存
用的闪盘，可在线擦除。
NOR 最好来可存储少量代码，应用在代码存储系统中，属于芯片内执行，这样应用
程序可以直接在 flash 中执行而不用读到系统的 RAM 中。
NAND 结构非常 读取写入和擦除的速度快，应用时往往于 flash 的管理和需要
特殊的系统接口。

4.2.4、RAM 的种类与选型

动态 RAM：DRAM 保留数据时间较短，速度较快，容量好。

静态 RAM 是更快速的，不容集，消耗更多的性能 (省功 power)

4.3. 嵌入式系统 I/O 接口。

4.3.1. 输入输出编程*，忙等 IO 和中断 I/O。

忙等 I/O，simplest 最简单的程序设备。

使用指令 (instructions) 去测试设备是否准备好。

(当设备准备好时用指令去测试)

同时发生的 (并发的 simultaneous) 忙等 I/O

```
while (TRUE);
    /*read*/
    while(peek(IN_STATUS)==0);
    achar=(char)peek(IN_DATA);
    /*write*/
    poke(OUT_DATA, achar);
    poke(OUT_STATUS,1);
    while(peek(OUT_STATUS)!=0);
}.
```

4.3 嵌入式系统 I/O 接口

4.3.1 输入输出编程* ：忙等 IO 和中断 IO

Date No.

Date No.

~~Scan~~ 中断 I/O

中断允许设备去改变在CPU中的控制流程.

因为有了程序去处理设备.

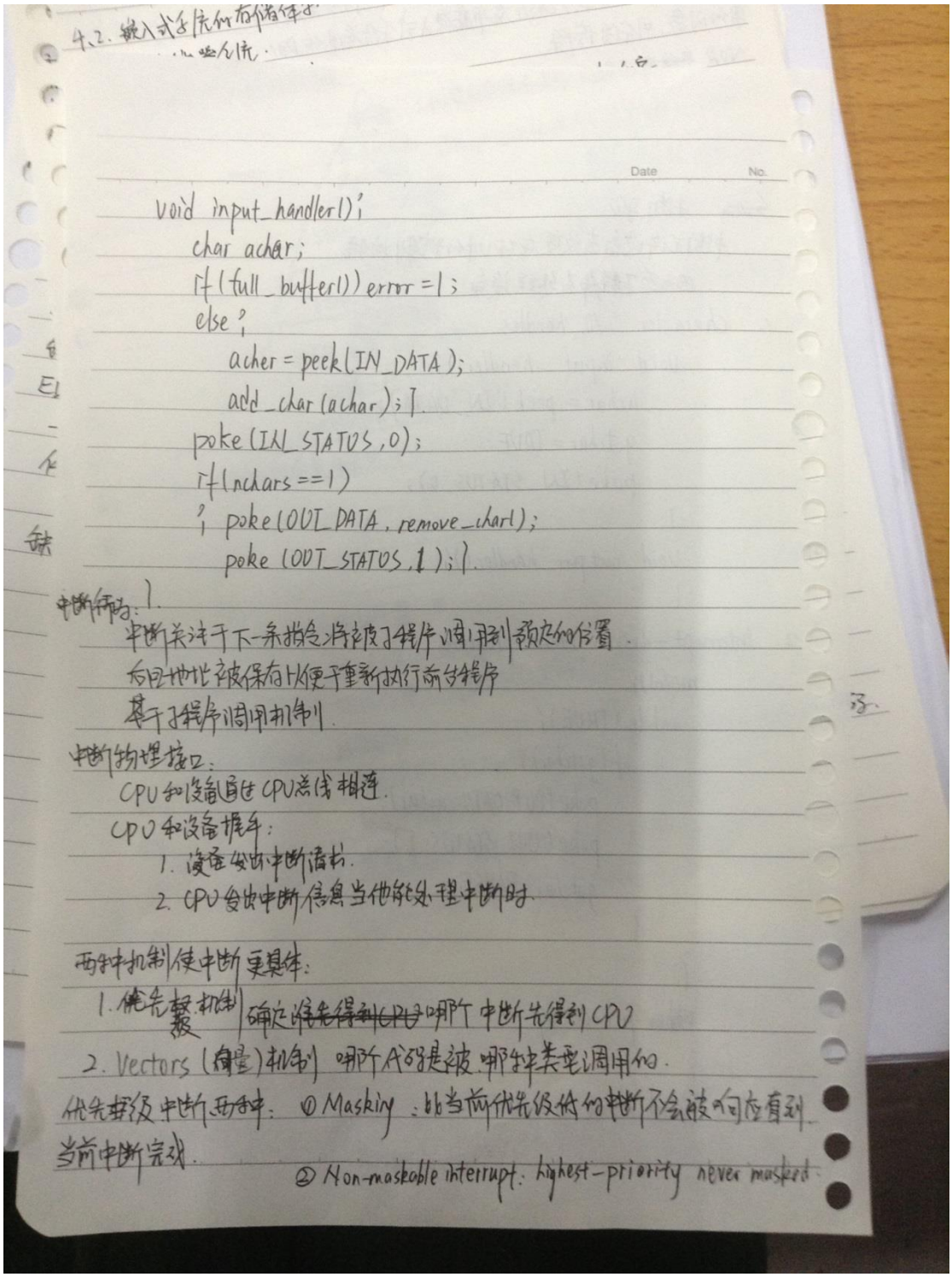
1. character I/O handles:

```
void input_handler();
    achar = peek(IN_DATA);
    gotchar = TRUE;
    poke(IN_STATUS, 0);
}
void output_handler();
}.
```

2. interrupt-driven main program

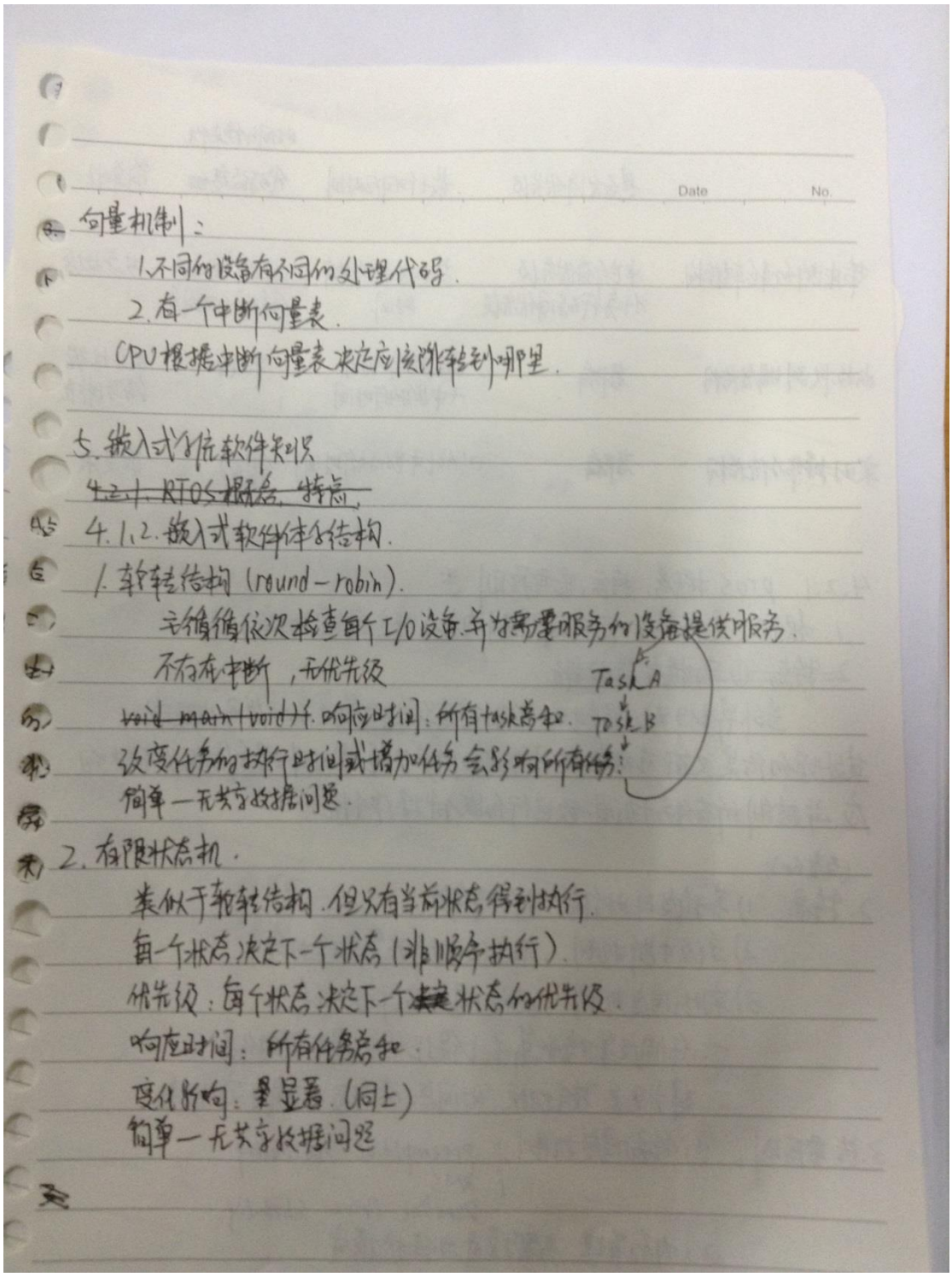
```
main();
 while (TRUE);
    if(gotchar){
      poke(OUT_DATA, achar);
      poke(OUT_STATUS, I);
      gotchar=FALSE;
    }
 }.
 other processing...
}.
```

3. Buffer-based input handler



5. 嵌入式系统软件知识

5.1 嵌入式软件基础知识

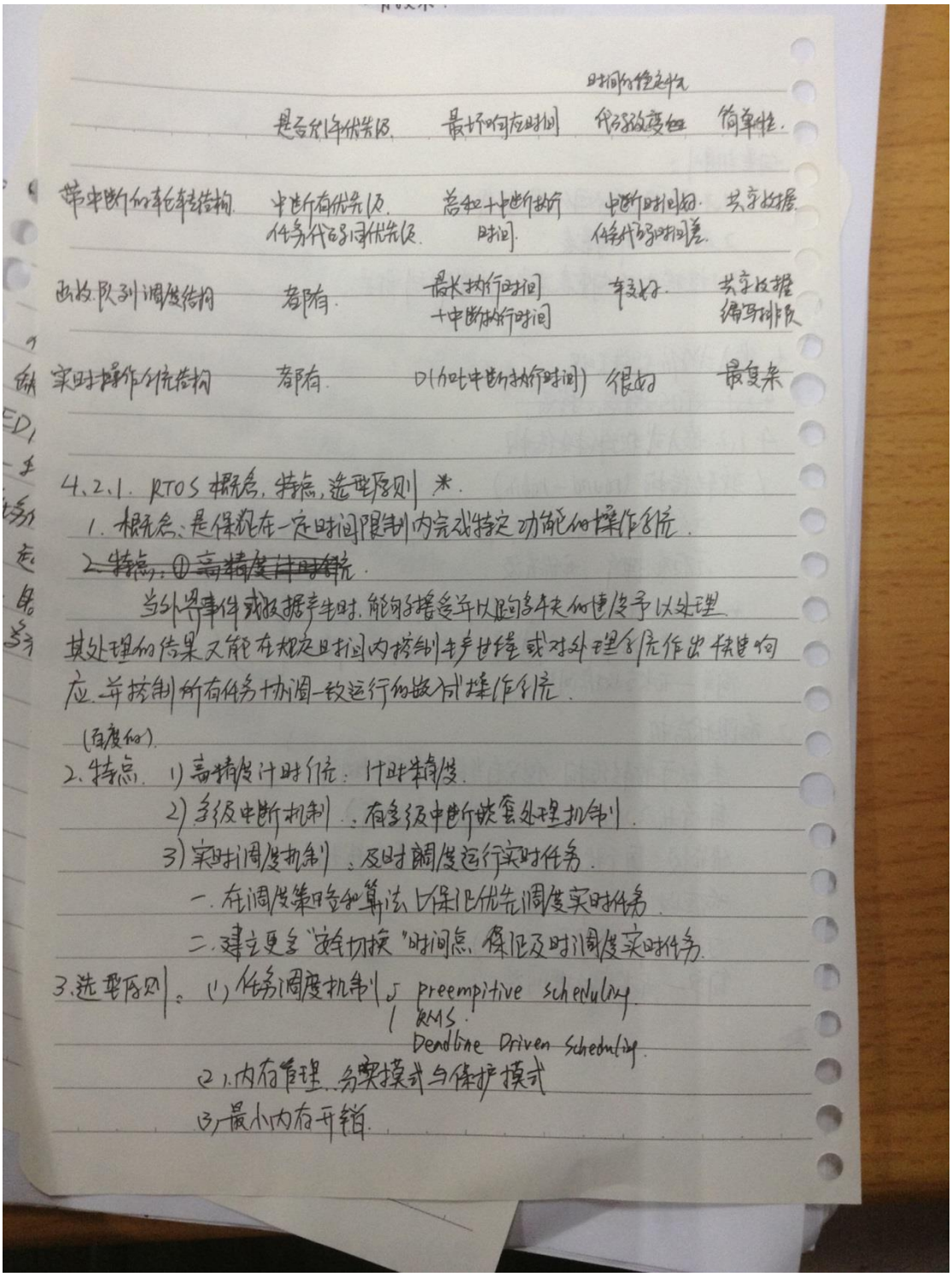


5.1.1 嵌入式软件的分类（系统软件、支撑软件、应用软件）

5.1.2 嵌入式软件体系结构*

5.2 嵌入式操作系统基础知识

5.2.1RTOS 概念、特点、选型原则*



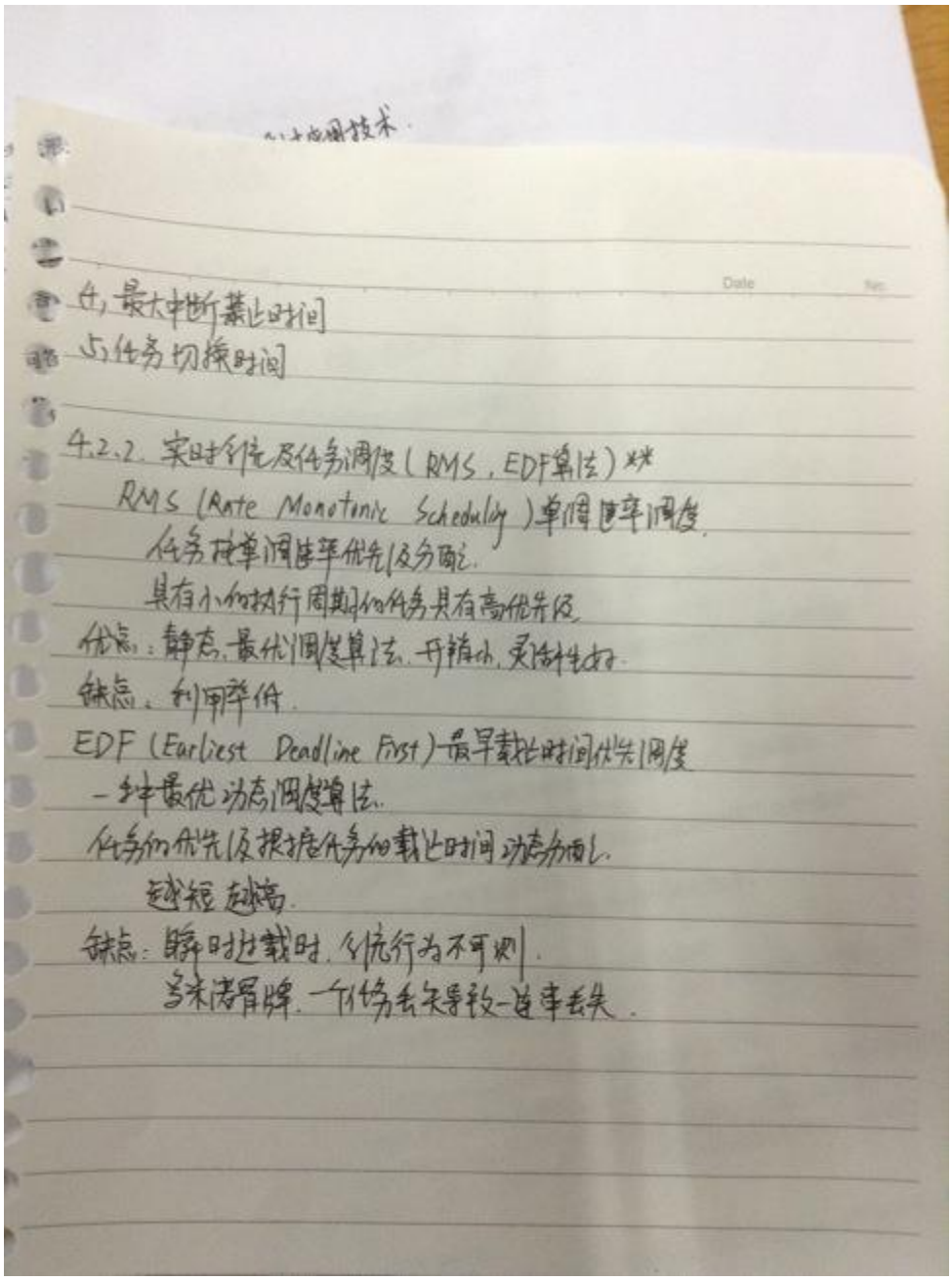
5.2.2 任务管理

进程、线程、任务的概念

任务的实现（任务的层次结构、任务控制块、任务的状态及状态转换、任务队列）

任务调度（可抢占调度、不可抢占调度、先来先服务、时间片轮转算法、优先级算法）

实时系统及任务调度（RMS、EDF 算法）**



RMS:

任务按单调速率优先级分配（RMPA）的调度算法，称为单调速率调度（RMS）。RMPA 是指任务的优先级按任务周期 T 来分配。它根据任务的执行周期的长短来决定调度优先级，那些具有小的执行周期的任务具有较高的优先级，周期长的任务优先级低。可抢占。

- **Theorem: If any priority assignment yields a feasible schedule, then priorities ordered by period (smallest period has the highest priority) also yields a feasible schedule.**

It is sufficient to show that if a non-RMS schedule is feasible可行的, then the RMS schedule is feasible.

• Completion time of the lower priority task is worst when its starting phase matches that of higher priority tasks.

• Thus, when checking schedule feasibility, it is sufficient to consider only the

worst case: All tasks start their cycles at the same time.

EDF:

EDF全称Earliest Deadline First。最早截止时间优先算法（EDF）也称为截止时间驱动调度算法（DDS），是一种动态调度算法。EDF在调度时，任务的优先级根据任务的截止时间动态分配。截止时间越短，优先级越高。如果一个任务集负载U<=1，则是可调度的。EDF 调度算法已被证明是动态最优调度

任务间通信（共享内存、消息、管道、信号）

同步与互斥（竞争条件、临界区、互斥、信号量、死锁）

5.2.3 存储管理（结合 ucOSII）

5.3 嵌入式系统程序设计

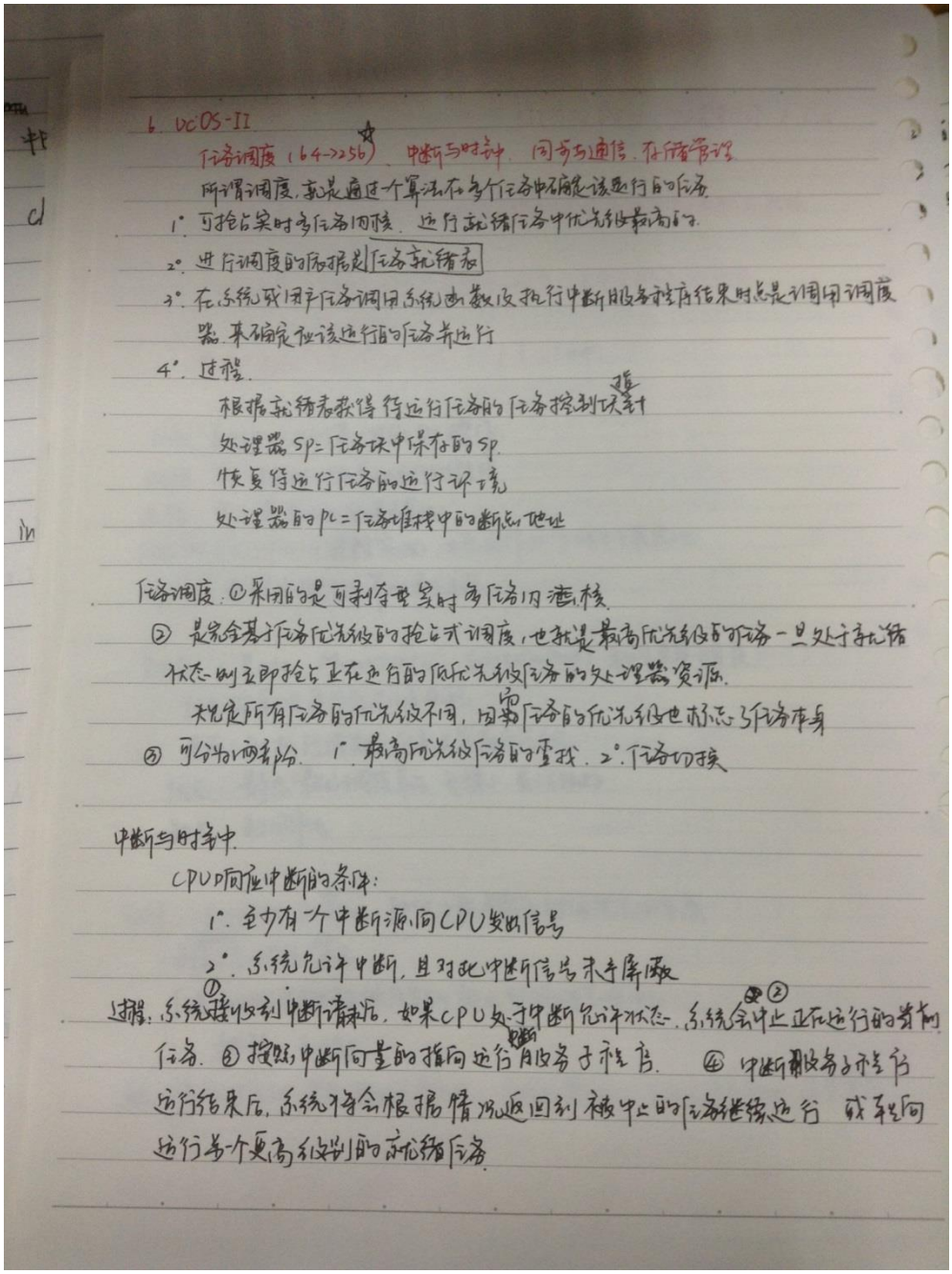
5.3.1 嵌入式软件开发基础知识

5.3.2 嵌入式程序设计语言

汇编、编译、解释系统的基础知识和基本工作原理

6. ucOS-II

任务调度*（64->256）、中断与时钟、同步与通信、存储管理



任务调度*

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

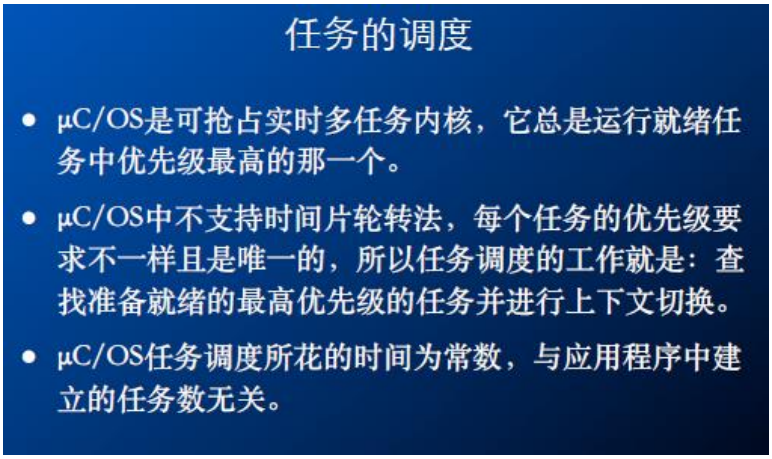
根据就绪表获得待运行任务的任务控制块指针

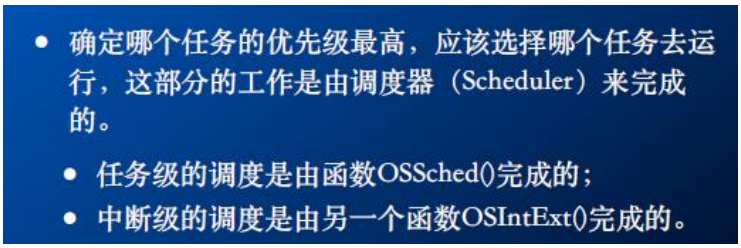
处理器的 SP=任务块中保存的 SP

恢复待运行任务的运行环境

处理器的 PC=任务堆栈中的断点地址

其实，调度器在进行调度时，在这个位置还要进行一下判断：究竟是待运行任务是否为当前任务，如果是，则不切换；如果不是才切换，而且还要保存被中止任务的运行环境

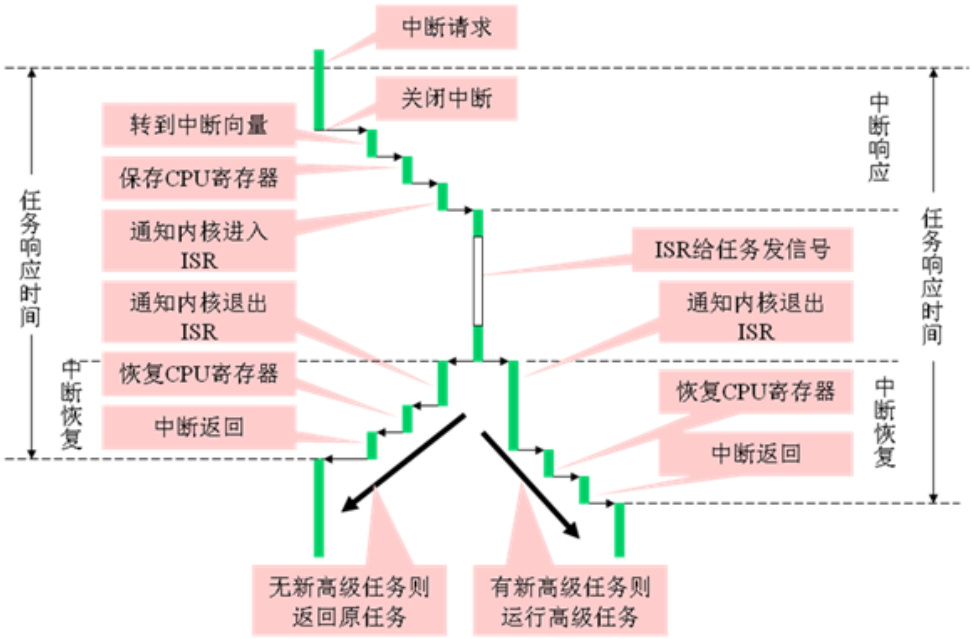




中断与时钟

μC/OS-II 系统响应中断的过程为：

系统接收到中断请求后，这时如果 CPU 处于中断允许状态（即中断是开放的），系统就会中止正在运行的当前任务，而按照中断向量的指向转而去运行中断服务子程序；当中断服务子程序的运行结束后，系统将会根据情况返回到被中止的任务继续运行或者转向运行另一个具有更高优先级别的就绪任务。



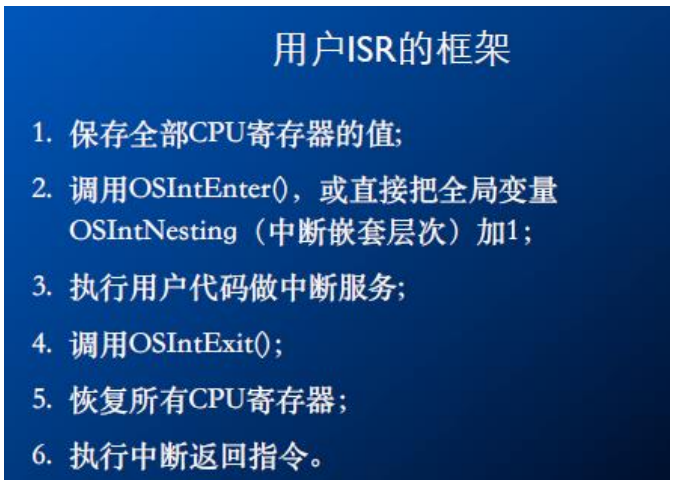
中断的响应过程

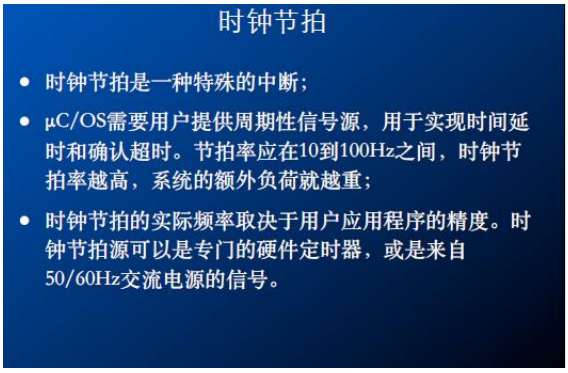
中断处理

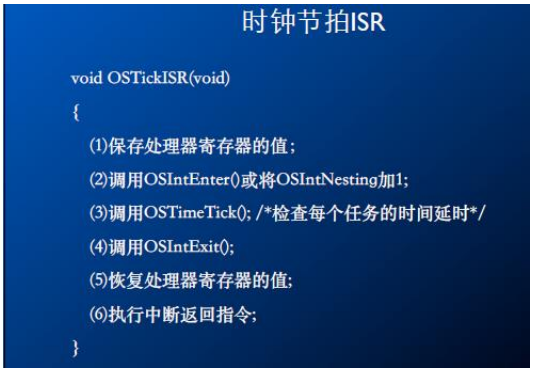
- 中断：由于某种事件的发生而导致程序流程的改变。产生中断的事件称为中断源。
- CPU响应中断的条件：
  - 至少有一个中断源向CPU发出中断信号；
  - 系统允许中断，且对此中断信号未予屏蔽。

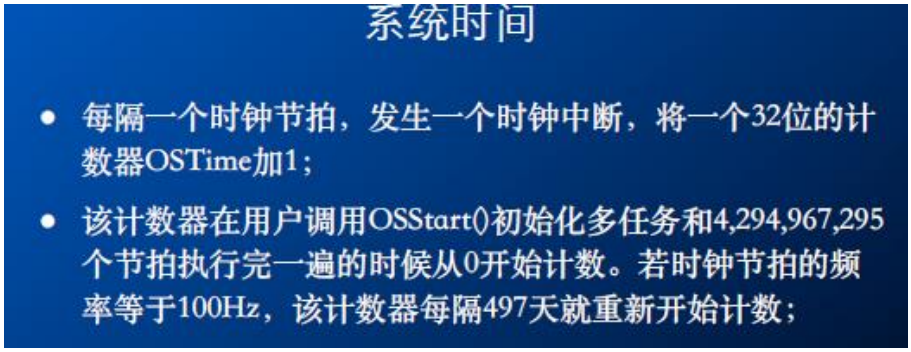
中断服务程序ISR

- 中断一旦被识别，CPU会保存部分（或全部）运行上下文（context，即寄存器的值），然后跳转到专门的子程序去处理此次事件，称为中断服务子程序(ISR)。
- μC/OS-Ⅱ中，中断服务子程序要用汇编语言来编写，然而，如果用户使用的C语言编译器支持在线汇编语言的话，用户可以直接将中断服务子程序代码放在C语言的程序文件中。









同步与通信

系统中的多个任务在运行时，经常需要互相无冲突地访问同一个共享资源，或者需要互相支持和依赖，甚至有时还要互相加以必要的限制和制约，才保证任务的顺利运行。因此，操作系统必须具有对任务的运行进行协调的能力，从而使任务之间可以无冲突、流畅地同步运行，而不致导致灾难性的后果。

与人们依靠通信来互相沟通，从而使人际关系和谐、工作顺利的做法一样，计算机系统是依靠任务之间的良好通信来保证任务与任务的同步的。

任务间通信与同步

- 任务间通信的管理：事件控制块ECB
- 同步与互斥
  - 临界区（Critical Sections）
  - 信号量（Semaphores）
- 任务间通信
  - 邮箱（Message Mailboxes）
  - 消息队列（Message Queues）

事件控制块ECB

- 所有的通信信号都被看成是事件(event)，μC/OS-II通过事件控制块(ECB)来管理每一个具体事件。

同步与互斥

- 为了实现资源共享，一个操作系统必须提供临界区操作的功能；
- μC/OS采用关闭/打开中断的方式来处理临界区代码，从而避免竞争条件，实现任务间的互斥；
- μC/OS定义两个宏(macros)来开关中断，即：OS_ENTER_CRITICAL()和OS_EXIT_CRITICAL()；
- 这两个宏的定义取决于所用的微处理器，每种微处理器都有自己的OS_CPU.H文件。

任务间通信

- 低级通信
  - 只能传递状态和整数值等控制信息，传送的信息量小；
  - 例如：信号量
- 高级通信
  - 能够传送任意数量的数据；
  - 例如：共享内存、邮箱、消息队列

存储管理

概述

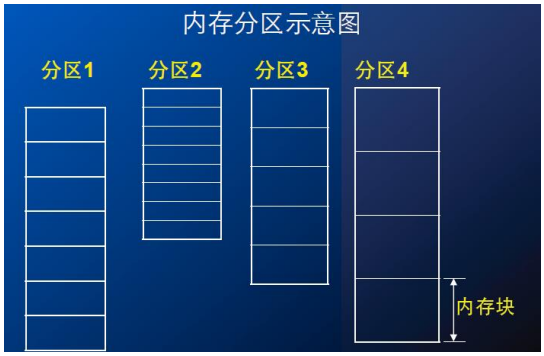
- μC/OS中是实模式存储管理
  - 不划分内核空间和用户空间，整个系统只有一个地址空间，即物理内存空间，应用程序和内核程序都能直接对所有的内存单元进行访问；
  - 系统中的“任务”，实际上都是线程－－－只有运行上下文和栈是独享的，其他资源都是共享的。
- 内存布局
  - 代码段(text)、数据段(data)、bss段、堆空间、栈空间；
  - 内存管理，管的是谁？

malloc/free?

- 在ANSI C中可以用malloc()和free()两个函数动态地分配内存和释放内存。在嵌入式实时操作系统中，容易产生碎片。
- 由于内存管理算法的原因，malloc()和free()函数执行时间是不确定的。μC/OS-II对malloc()和free()函数进行了改进，使得它们可以分配和释放固定大小的内存块。这样一来，malloc()和free()函数的执行时间也是固定的了

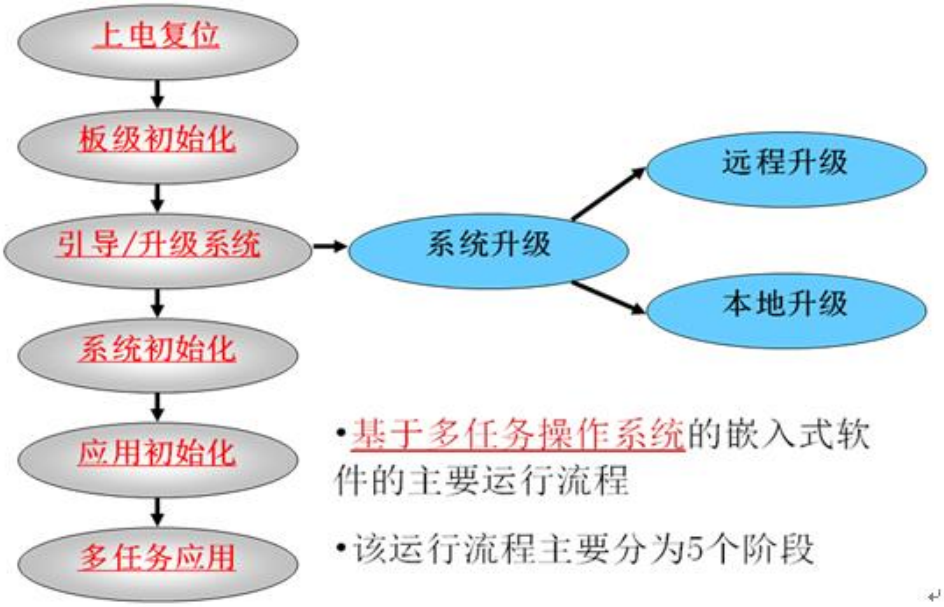
μC/OS中的存储管理

- μC/OS采用的是固定分区的存储管理方法
  - μC/OS把连续的大块内存按分区来管理，每个分区包含有整数个大小相同的块；
  - 在一个系统中可以有多个内存分区，这样，用户的应用程序就可以从不同的内存分区中得到不同大小的内存块。但是，特定的内存块在释放时必须重新放回它以前所属于的内存分区；
  - 采用这样的内存管理算法，上面的内存碎片问题就得到了解决。



7. bsp, bootloader

嵌入式软件运行流程*



嵌入式软件运行流程-1

- **上电复位、板级初始化阶段**
  - 嵌入式系统上电复位后完成板级初始化工作。
  - **板级初始化程序**具有完全的硬件特性，一般采用汇编语言实现。不同的嵌入式系统，板级初始化时要完成的工作具有一定的特殊性，但以下工作一般是必须完成的：
    - **CPU 中堆栈指针寄存器**的初始化。
    - **BSS 段**（Block Storage Space 表示未被初始化的数据）的初始化。
    - **CPU 芯片级的初始化：**中断控制器、内存等的初始化。

嵌入式软件运行流程-2

- **系统引导/升级阶段**
  - 根据需要分别进入系统软件引导阶段或系统升级阶段。
  - 软件可通过测试通信端口数据或判断特定开关的方式分别进入不同阶段。

嵌入式软件运行流程-3

- **系统引导阶段，**系统引导有几种情况：
  - **将系统软件从 NOR Flash 中读取出来加载到 RAM 中运行：**这种方式可以解决成本及 Flash 速度比 RAM 慢的问题。软件可压缩存储在 Flash 中。
  - 不需将软件引导到 RAM 中而是让其**直接在 NorFlash** 上运行，进入系统初始化阶段。
  - **将软件从外存（如 NandFlash、CF 卡、MMC 等）中读取出来加载到 RAM 中运行：**这种方式的成本更低。

嵌入式软件运行流程-4

- **系统升级阶段**
  - 进入系统升级阶段后系统可通过网络进行**远程升级**或通过串口进行**本地升级**。
  - **远程升级**一般支持 **TFTP、FTP、HTTP** 等方式。
  - **本地升级**可通过 **Console 口**使用超级终端或特定的升级软件进行。

嵌入式软件运行流程-5

- **系统初始化阶段**
  - 在该阶段进行操作系统等系统**软件各功能部分必需的初始化工作**，如根据系统配置**初始化数据空间、初始化系统所需的接口和外设**等。
  - 系统初始化阶段需要**按特定顺序进行**，如首先完成内核的初始化，然后完成网络、文件系统等的初始化，最后完成中间件等的初始化工作。

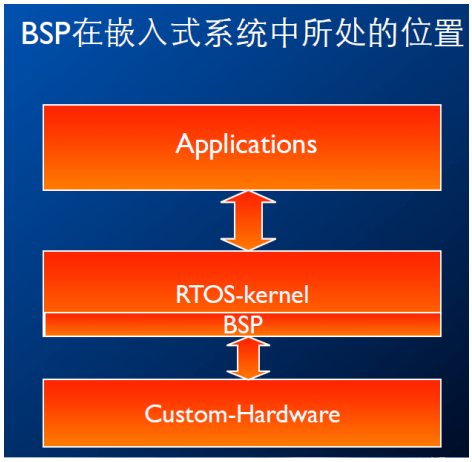
嵌入式软件运行流程-6

- **应用初始化阶段**
  - 在该阶段进行应用任务的创建，信号量、消息队列的创建和与应用相关的其它初始化工作。

- **多任务应用运行阶段**
  - 各种初始化工作完成后，系统进入多任务状态，操作系统按照已确定的算法进行任务的调度，各应用任务分别完成特定的功能

bsp，特点，与 bios 区别

BSP 的概念：BSP 全称“板级支持包”（Board Support Packages），说的简单一点，就是一段启动代码，和计算机主板的 BIOS 差不多，但提供的功能区别就相差很大。



BSP 是嵌入式系统的基础部分，也是实现系统论可移植的关键。它负责上电复位时的**硬件初始化、启动 RTOS 或应用程序模块、提供底层硬件驱动，为上层软件提供访问底层硬件**的手段。

BSP 针对具体的目标板设计，其结构和功能随目标版的不同呈现较大的差异。

不同系统中的 BSP

- 一个嵌入式操作系统针对不同的CPU，会有不同的BSP。
- 即使同一种CPU，由于外设的一点差别BSP相应的部分也不一样

BSP 特点

- **硬件相关性**
  - 因为嵌入式实时系统的硬件环境具有应用相关性，所以，作为高层软件与硬件之间的接口，BSP 必须为操作系统提供操作和控制具体硬件的方法。
- **操作系统相关性**

- 不同的操作系统具有各自的软件层次结构， 因此，不同的操作系统具有特定的硬件接口形式。

与 bios 区别

- BIOS主要是负责在电脑开启时检测、初始化系统设备（设置栈指针，中断分配，内存初始化..）、装入操作系统并调度操作系统向硬件发出的指令。
- BSP是和操作系统绑在一起运行，尽管BSP的开始部分和BIOS所做的工作类似，但是 BSP 还包含和**系统**有关的基本驱动 。
- BIOS程序是用户不能更改，编译编程的，只能对参数进行修改设置，但是程序员还可以编程修改BSP，在BSP中任意添加一些和系统无关的驱动或程序，甚至可以把上层开发的统统放到BSP中

rtos 的引导模式

- **操作系统引导概念：**将操作系统装入内存并开始执行的过程。
- 按时间效率和空间效率不同的要求，分为两种模式：
  - **需要 BootLoader 的引导模式：**节省空间，牺牲时间，适用于硬件成本低，运行速度快，但启动速度相对慢
  - **不需要 BootLoader 的引导模式：**时间效率高，系统快速启动，直接在 NOR flash 或 ROM 系列非易失性存储介质中运行，但不满足运行速度的要求

bootloader及其启动过程

bootloader 的概念：bootloader 是在操作系统内核运行之前执行的一段小程序，它**将操作系统内核从外部存储设备拷贝到内存中，并跳转到内核的首条指令**

bootloader

- 嵌入式系统中的 OS 启动加载程序
- 引导加载程序
  - 包括固化在固件(firmware)中的 boot 代码(可选)，和Boot Loader两大部分
  - 是系统加电后运行的第一段软件代码
- 相对于操作系统内核来说，它是一个硬件抽象层

Boot Loader 的启动过程是单阶段（Single Stage）还是多阶段（Multi-Stage）

- 多阶段的 Boot Loader
  - 提供更为复杂的功能，以及更好的可移植性
  - 从固态存储设备上启动的 Boot Loader 大多都是 2 阶段的启动过程
- BOOTLOADER一般分为2部分
  - 汇编部分执行简单的硬件初始化
  - C语言部分负责复制数据,设置启动参数,串口通信等功能
- BOOTLOADER的生命周期
  - 1. 初始化硬件,如设置UART(至少设置一个),检测存储器等
  - 2. 设置启动参数,告诉内核硬件的信息,如用哪个启动界面,波特率.
  - 3. 跳转到操作系统的首地址.
  - 4. 消亡

Boot Loader 的主要任务

- stage1 通常包括以下步骤
  - 硬件设备初始化
  - 为加载 Boot Loader 的 stage2 准备 RAM 空间
  - 拷贝 Boot Loader 的 stage2 到 RAM 空间中
  - 设置好堆栈
  - 跳转到 stage2 的 C 入口点
- Boot Loader 的 stage2 通常包括以下步骤
  - 初始化本阶段要使用到的硬件设备
  - 检测系统内存映射(memory map)
  - 将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中
  - 为内核设置启动参数
  - 调用内核

8. 建模

有限状态机及其应用。

Ref：Chapter3, Practical UML Statecharts in C/C++,Second Edition.

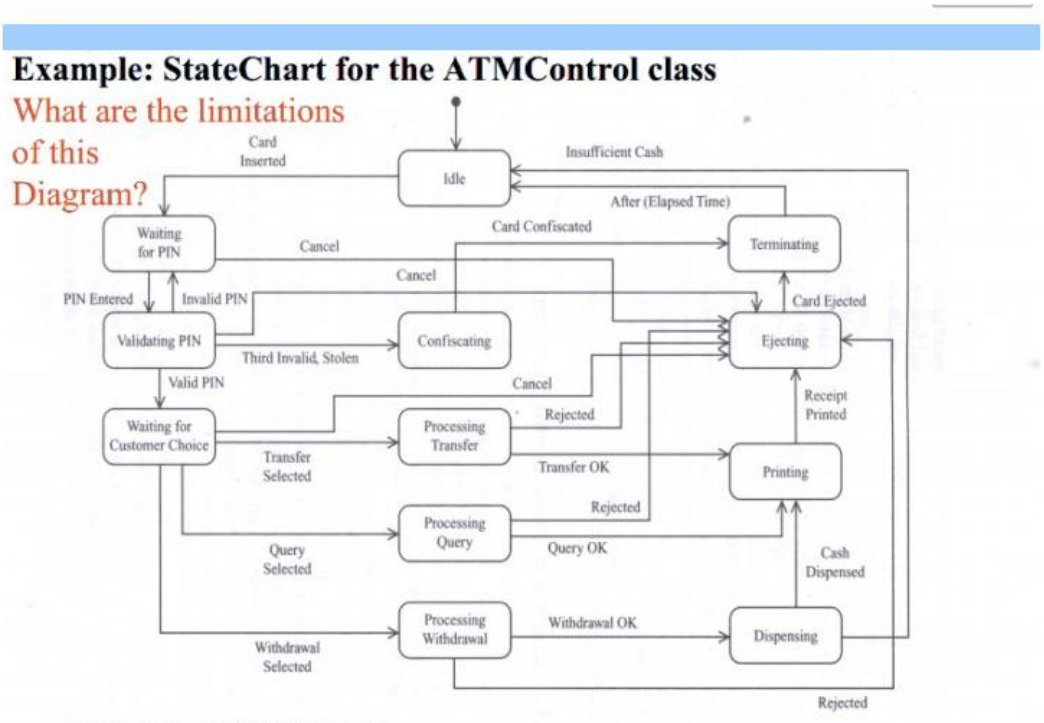


Figure 10.2 *Example of flat ATM statechart*