《微机原理与嵌入式系统》第四五六章内容总结

姓名：赵泊尧　　学号：PB18061426

红色为掌握部分：需要（准确地）记忆、定量计算或编程实现，出现在任意题型中；
青色为理解部分：能够（具体地）说明基本概念和原理，主要出现在填空和简答题中；
蓝色为了解部分：可以（大致地）运用知识分析、判断给定材料，主要出现在选择和判断题中。

第四章

题目 1：总线的五种分类方式，主要是 DB、AB、CB（理解）

答：

- 按照物理接口分类：电缆式、主板式、背板式
- **根据总线信号传输类型**；数据总线、地址总线、控制总线和电源总线
- **按照控制特性分类**：同步总线、异步总线和半同步总线按照通信方式分类：串行总线和并行总线
- **按位置分类**：片内总线、芯间总线、内总线、外总线按数据传输方式分类：串行传输、并行传输
- **按时序控制方式分类**：异步传输、同步传输
- **按时分复用方式分类**：非复用、复用
- 总线按功能分类
  - 地址总线（AB，Address Bus）：地址总线宽度决定了最大存储器空间寻址范围
  - 数据总线（DB，Data Bus：用于传送数据信息。通常为双向三态形式，可传指令
  - 控制总线（CB，Control Bus）：用于传输（完成各项操作所需要的）控制信号，协调计算机不同部件

- 有序化地使用数据总线和地址总线；除电源线、地线、数据总线和地址总线外的所有信号线都归纳为控制总线

题目 2：总线周期的四个阶段（掌握）

答：

- 请求及仲裁（Request and Arbitration）阶段。主模块请求，仲裁机构决定把下一个总线传输周期分给哪一个请求源。
- 寻址（Addressing）阶段。取得总线使用权的主模块通过总线发出本次要访问的从模块（存储器地址或 I/O 端口）地址及有关命令，通知参与传输的从模块开始启动。
- 数据传输（Data Transferring）阶段。主模块和从模块进行数据传输，数据由源模块发出，经数据总线到达目的模块。
- 结束阶段。主模块、从模块的有关信息均从总线上撤销，让出总线，以便下一个总线传输周期其他模块能够使用总线。

题目 3：常见集中式仲裁、分布式仲裁方法的原理及不同方法的优缺点（了解）

答：

（太长不看版 ↓）

- **集中式仲裁**：将控制逻辑（即总线仲裁器 arbitrator）集中在一处，分为**串行仲裁、并行仲裁、串并行混合式仲裁**
- **串行仲裁优点**：只用很少几根线就能按一定优先次序实现总线仲裁，很容易扩充设备。结构简单，造价较低。
- **串行仲裁缺点**：优先级存在传播延迟，这种延迟与模块数量成正比，所以判优速度较慢，一般只能接少量的（几个）模块；链形结构可靠性低，一个模块有故障就会造成整条“链”失效.
- **并行仲裁优点**：响应时间快，确定优先响应的设备所花费的时间少，用不着一个设备接一个设备地查询。其次，对优先次序的控制相当灵活，可以预先固定也可以通过程序来改变优先次序；还可以用屏蔽（禁止）某个请求的办法，不响应来自无效设备的请求。判优速度快，且与模块数无关；
- **并行仲裁缺点**：缺点：所需“请求线”和“允许线”较多，N 个模块需要 2N 条。
- **串并混和**：兼具有串行仲裁和并行仲裁的优点，既有较好的灵活性、可扩充性又可容纳较多的设备而结构也不会过于复杂，且有较快的响应速度。

- **分布式仲裁方法**：将控制逻辑分散在（与总线连接的）各个部件或设备上，由各个节点竞争使用权。分为自举分布式仲裁；使用多个请求线，不需要中心裁决器，每个设备独立地决定自己是否是最高优先级请求者
- 特点：使用多个请求线，**不需要中心裁决器**，每个设备独立地决定自己是否是最高优先级请求者。

（详细版 ↓）

- 集中式仲裁：将控制逻辑（即总线仲裁器 arbitrator）集中在一处，分为**串行仲裁、并行仲裁、串并行混合式仲裁**
  - **串行仲裁**：串行仲裁又称为“菊花链”（Daisy Chain）仲裁。信号线包括：总线请求 BR（Bus Request）信号、总线允许 BG（Bus Grant）信号和总线忙 BB（Bus Busy）信号。
    * BB 无效（总线空闲）时，主设备才能提出总线请求。各主设备的 BR 信号用“线与”方式接到仲裁器的请求输入端。仲裁器在接到 BR 信号后输出 BG 信号，BG 信号通过主设备链向后传递，直到提出总线请求的那个设备为止。请求总线的主设备收到 BG 号后，获得总线的控制权，将 BB 信号置为有效。

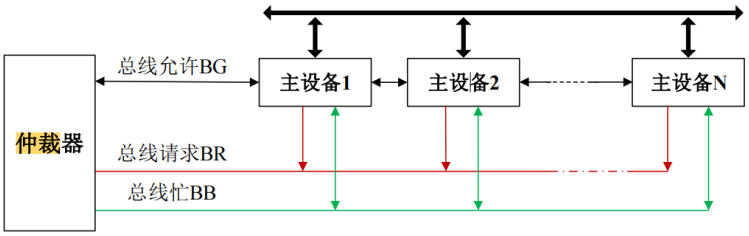


Figure 1: **串行仲裁示意图**

    * 越靠近控制器的模块，优先级越高；
    * 优点：只用很少几根线就能按一定优先次序实现总线仲裁，很容易扩充设备。结构简单，造价较低。
    * 缺点：对询问链的电路故障很敏感，如果第 i 个设备的接口中有关链的电路有故障，那么第 i 个以后的设备都不能进行工作。查询链的优先级是固定的，如果优先级高的设备出现频繁的请求时，优先级较低的设备可能长期不能使用总线。链形优先级存在传播延迟，这种延迟与模块数量成正比，所以判优速度较慢，一般只能接少量的（几个）模块；链形结构可靠性低，一个模块有故障就会造成整条“链”失效.
  - **并行仲裁**又称为“独立请求式仲裁”。每个主设备都有独立的 BR 和 BG 信号线，并分别接到仲裁器上。

* 任一主设备使用总线都要**通过 BR 信号向仲裁器发出请求**，仲裁器按规定的优先级算法选中一个主设备，并**把 BG 信号送给该设备**。被选中的设备撤销 BR 信号，并**输出有效的 BB 信号**，通知其他设备“总线已经被占用”。

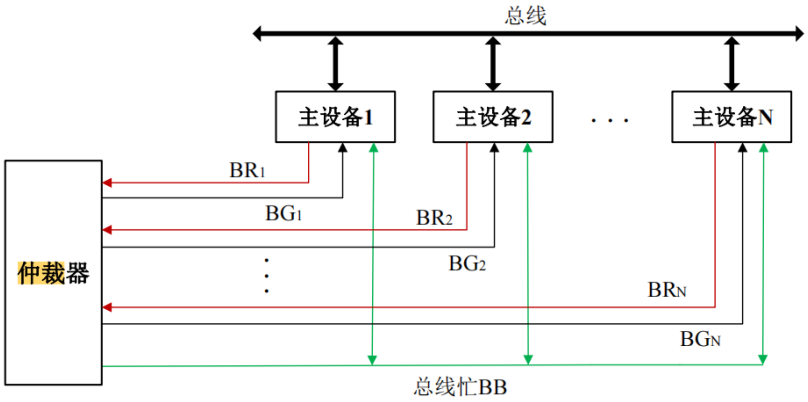


Figure 2: **并行仲裁示意图**

* 每个模块有一条总线请求（BR）、一条总线允许（BG）和一条所有模块共用的总线忙（BUSY）信号；
* 控制器内置一个优先级编码器和优先级译码器，用以选择优先级最高的请求，并产生出相应的“总线允许”信号；
* 当 BUSY 信号有效时，表示有模块正使用总线，因此请求使用总线的模块必须等待，直至 BUSY 信号变为无效。所有需要使用总线的模块都可发“总线请求”信号，总线仲裁器仅向优先级最高的模块发出“总线允许”信号。
* 优点：响应时间快，确定优先响应的设备所花费的时间少，用不着一个设备接一个设备地查询。其次，对优先次序的控制相当灵活，可以预先固定也可以通过程序来改变优先次序；还可以用屏蔽（禁止）某个请求的办法，不响应来自无效设备的请求。判优速度快，且与模块数无关；
* 缺点：所需“请求线”和“允许线”较多，N 个模块需要 2N 条。

– **串并行混合式仲裁**：混合仲裁又称为多级仲裁，结合了并行仲裁和串行仲裁两种思路，更灵活。

* **先并行**，所有主设备的总线请求首先经过总线判决器（BR1 或 BR2），总线判决器决定是 BR1 所连主设备还是 BR2 所连主设备获得总线控制权。
* 然后**再按串行**方式来决定 BR1 上的设备是主设备 2 还是主设备 4、BR2 上的设备是主设备 1 还是主设备 3 应该获得总线控制权。
* **并行请求线 BR1 和 BR2 的优先级由总线判决器内部逻辑确定**，同一链路上各设备的优先级则由该设备与总线判决器的远近程度确定。

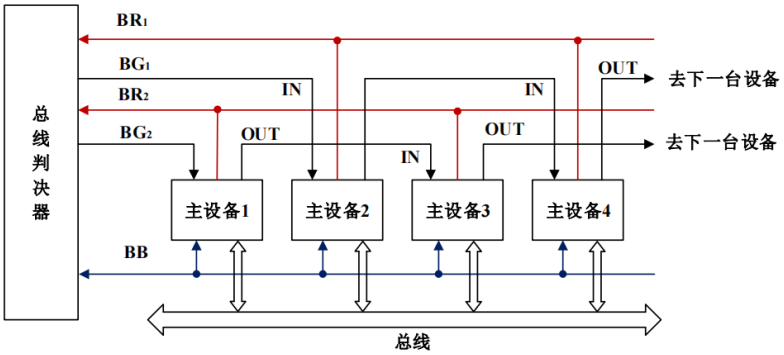


Figure 3: **混合仲裁示意图**

  * 兼具有串行仲裁和并行仲裁的优点，既有较好的灵活性、可扩充性又可容纳较多的设备而结构也不会过于复杂，且有较快的响应速度。

- 分布式仲裁方法：将控制逻辑分散在（与总线连接的）各个部件或设备上，由各个节点竞争使用权。分为自举分布式仲裁；使用多个请求线，不需要中心裁决器，每个设备独立地决定自己是否是最高优先级请求者
  - 特点：使用多个请求线，**不需要中心裁决器**，每个设备独立地决定自己是否是最高优先级请求者。
  - 原理：分为申请期和裁决期。在申请期需要请求总线控制权的设备在各自对应的总线**请求线上送出请求信号**。在裁决期，每个设备将有关请求线上的合成信号取回分析，以确定自己能否拥有总线控制权。
  - 每个设备通过取回的合成信息能够检测出其它设备是否发出了总线请求。如果检测到其它优先级更高的设备也请求使用总线，则本设备暂时不能使用总线；否则，本设备就可立即使用总线。
  - 分布式仲裁的仲裁器为各个参与的主设备，实现所谓“比大小”的仲裁方式，最终获胜的主设备获得 BUS 的使用权。

题目 4： 总线时序（理解）

答：

总线时序是指总线操作过程中总线上各信号在时间顺序上的配合关系。不同总线操作需要不同的总线时序予以配合。

（太长不看版 ↓）

- 同步总线时序（总线上的数据传输由统一时标控制）、异步总线时序（主从模块之间通信时，采用**应答方式**进行联络和协调工作）、半同步总线时序（高速模块同步，低速模块异步）、周期分列式总线时序（两个子周期）。

- 异步总线时序又可分为**不互锁方式**、**半互锁方式**和**全互锁方式**。

（详细版 ↓）

- 同步总线时序
  - 总线上的数据传输由**统一时标**控制。
  - 时标通常由 CPU 的总线控制部件发出，送到总线上的所有部件；也可以由每个部件各自的时序发生器发出，但是必须由总线控制部件发出的时钟信号对它们进行同步。
  - 优点：模块间的配合简单一致；
  - 缺点：主从模块之间的时间配合属强制性同步，必须按速度最慢的部件来设计公共时钟。
- 异步总线时序
  - 异步总线允许各模块速度的不一致性，提高了模块的适应性，给设计者带来更多的选择余地。
  - 异步总线中系统没有公用的时钟，主从模块之间通信时，采用**应答方式**进行联络和协调工作。根据问答信号之间的关系，异步总线时序可分成**不互锁方式**、**半互锁方式**和**全互锁方式**。
  - 主设备在发数据的同时发选通信号 STB (STroBe)，从设备收到数据后应答 ACK (ACKnowledgement)；主从模块之间增加两条握手应答线。
    - 不互锁方式
      - 主设备发 STB，告诉从设备当前数据有效。间隔固定时间后，认为从设备已经收到，撤销 STB；
      - 从设备收到数据后发 ACK 应答。间隔固定时间后，撤销 ACK。
    - 半互锁方式
      - 主设备发 STB 后，等待从设备的 ACK 应答。只有收到 ACK 后才撤销 STB；
      - 从设备发 ACK 后，不等待主设备的应答。在间隔固定时间后，撤销 ACK。
    - 全互锁方式（四边沿协议）
      - 主设备发 STB 后，等待从设备的 ACK 应答。只有收到 ACK 后才撤销 STB；
      - 从设备发 ACK 后，等待主设备的应答（撤销 STB），然后才撤销 ACK。
      - 主从设备相互等待，传输可靠性最高。
- **关于同步和异步通信方式的几点讨论**

  - 传输速度
    - 同步：如果从设备太慢，就无法满足时序要求。
    - 异步：应答过程的交互次数越多，速度越慢。
  - 可靠性
    - 最可靠的方式：异步全互锁，每步操作“环环相扣”。

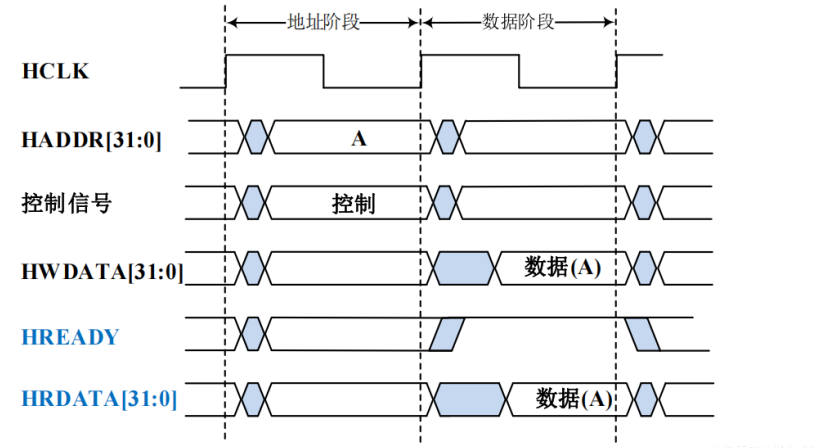
- 半同步总线时序
  - 半同步总线是对同步总线的一种优化，对于大多数速度较快的传送对象，均按照同步方式定时。
  - 对于系统所连接的少数速度较慢的设备，增加一条 Ready/wait 状态信号线，当慢速设备被访问时，可以利用这条信号线请求主模块延长传送周期。
  - 半同步总线提高了系统的适应性，但是速度仍然偏低。

- 周期分裂式总线时序
  - 分离式总线的思想：将一个传输周期（或总线周期）分解为两个子周期（或者称为子阶段）。
    - 在第一个子周期（寻址子周期，或称为地址阶段）中，主模块 A 获得总线使用权后，将命令、地址、A 模块编号等信息发到系统总线上，由相关的从模块 B 接收下来。然后 A 模块放弃总线，供其它模块使用。
    - 在第二个子周期（数据传输子周期，或称为数据阶段）中，B 模块根据所收到的命令，经过一系列的内部操作，将 A 模块所需的数据准备好，然后由 B 模块申请总线使用权，一旦获准，B 模块将 A 模块的编号和所需数据、B 模块的地址等信息送到总线上，供 A 模块接收。
  - 将一个总线读周期或写周期分解为两个分离的子周期：**寻址子周期**、**数据传送子周期**。

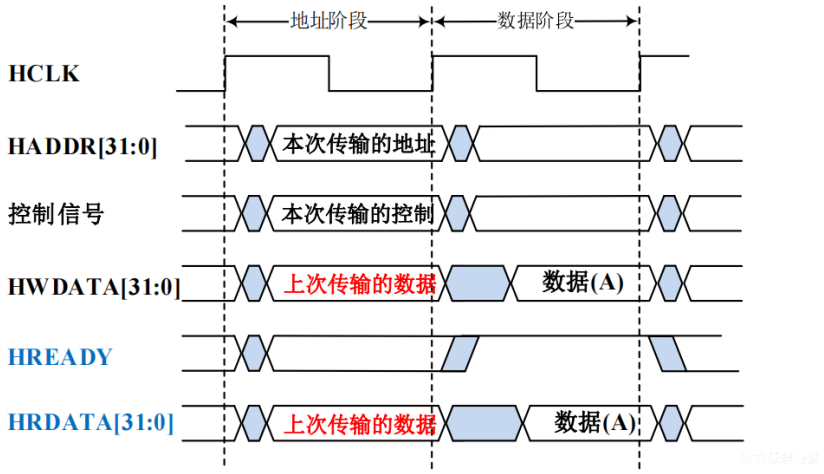
题目 5：AHB 数据传输过程，AHB“流水线”分离操作（理解）

答：

- AHB 系统的构成：AHB 主机、AHB 从机、AHB 仲裁器、AHB 译码器
- AHB 特点概述：支持突发传输、支持分裂式操作、单周期总线主机移交、单一时钟沿操作、非三态的实现、可扩展至更宽的数据总线架构
- AHB 系统数据传输过程：2 个阶段：地址阶段、数据阶段
- “流水线”的分离



(a) 数据传输过程



(b) 流水线

Figure 4: **AHB 的“流水线”机制**

– 2 级流水线

  * 第 n 次传输的地址在第 n-1 次传输时被驱动到了地址总线上。“驱动地址”和“驱动数据”两个操作构成 2 级流水线操作。
  * 从机因某种原因不能及时响应时，这个流水线就会被打断。

– “SPLIT”思想（即**周期分裂式时序**）

  * 周期分裂式时序：**地址阶段和数据阶段可以被分离**。
  * 从机不能及时响应时，发送控制信号 **HSPLITx** 通知仲裁器。
  * 仲裁器**检测到 HSPLITx** 后，知道从机当前不进行传输，则可以把总线的使用权出让给其他主机。
  * 当从机做好接收数据准备后，通过控制信号 HSPLITx 发出重新启动传输的信号，仲裁器根据挂起操作主机的优先级决定何时再次分配总线使用权。
  * 当主机获得总线使用权后，重新发送地址、控制等信息，继续刚才挂起

的传输操作。

题目 6：I/O 接口的功能；两种 I/O 端口的编址方式及特点（了解）

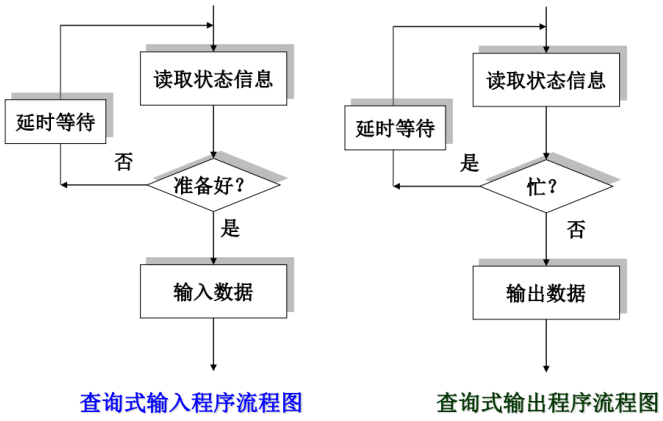
答：

- I/O 接口的功能:
  - 设置**数据缓冲**解决速度不匹配问题
  - 设置**电平转换**电路解决电平不一致问题
  - 设置**信息转换逻辑**满足各自格式要求
  - 设置**时序控制**电路同步 CPU 和外设的工作
  - 提供**地址译码**电路
  - 提供 **I/O 控制、读/写控制**及**中断控制**等逻辑
- I/O 端口和存储器统一编址，也称**存储器映像**的 I/O（Memory Mapped I/O）方式
  - 系统中每个 I/O 端口都看作 1 个存储单元，并**与存储单元统一编址**，所有访存指令均可用来访问 I/O 端口，不用设置专门的 I/O 指令。
  - 优点：(1) 对 I/O 口的操作与对存储器的操作完全相同，无须专用的 I/O 指令；(2) 外设数目或 I/O 寄存器数目几乎不受指令限制；(3)CPU 读/写控制逻辑较简单
  - 缺点：(1) 占用了存储器的一部分地址空间；(2) 增加了地址译码电路的复杂性
- I/O 端口和存储器分开单独编址，也称 **I/O 映像**的 I/O（I/O Mapped I/O）方式
  - **对系统中的 I/O 端口地址单独进行编址，不占用存储空间**；使用专门的 IN/OUT 指令来访问 I/O 端口。
  - 优点：(1)I/O 端口地址不占用存储器地址空间；(2)I/O 端口地址译码较简单，寻址速度较快；(3) 使用专用 I/O 指令和存储器访问指令有明显区别，可使程序编制得清晰
  - 缺点：(1) 专用 I/O 指令类型少，远不如存储器访问指令丰富；(2)CPU 提供存储器读/写、I/O 端口读/写两组控制信号

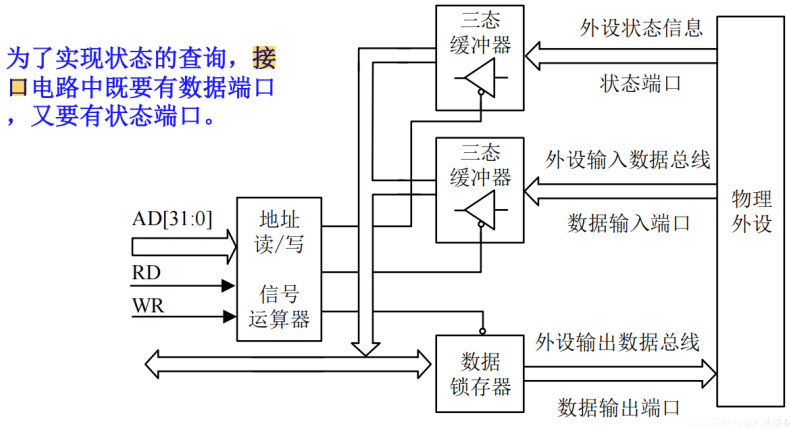
题目 7：状态查询方式 I/O 接口电路原理（理解）

答：

对外设要求：提供状态口和数据口



(a) CPU 查询流程



(b) 状态查询方式 I/O 接口电路

Figure 5: **状态查询方式 I/O 接口电路原理**

在有多个外设的系统中，CPU 的查询顺序由外设的优先级确定

题目 8：SPI、I^{2}C 接口原理（大致传输过程）（了解）

答：

I^{2}C **部分：**

- I^{2}C 两线式串行总线：一条串行数据线 SDA，一条串行时钟线 SCL，用于连接微控制器及其外围设备
- I^{2}C **数据传输过程——主设备向从设备发送信息：**
  - 主设备（master）发送开始信号 (S)，对应图 6中 (S) 状态；
  - 主设备发送 7 比特的从设备（slave）地址，对应图 6中发送地址阶段；
  - 主设备发送写命令 (低电平，W#)，对应图 6中 R/W#；
  - 接着从设备应答 (A)，ACK 为应答成功 (A)，表示有这个设备；
  - 主设备发送 8 位字节数据；

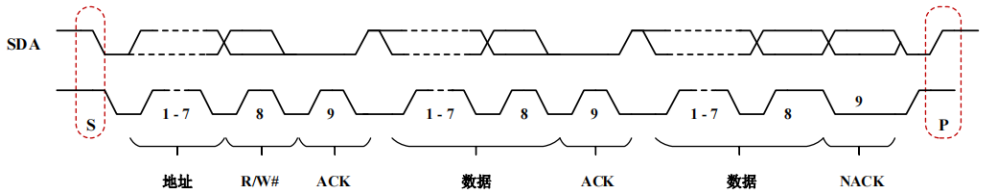


Figure 6: I^2C **总线上的数据传输过程的时序**

- 接着从设备应答，ACK 为应答成功 (A)，或，NACK 为不成功为 (N)；
- 如果从设备应答成功 (A) 继续主设备发送数据；若应答不成功 (N) 则主设备发送停止信号 (P)，对应图 6中 (P) 状态。

- I^2C **数据传输过程——主设备读取从设备信息:**
  - 主设备发送开始信号 (S)，对应图 6中 (S) 状态；
  - 主设备发送 7 比特的从设备地址，对应图 6中发送地址阶段；
  - 主设备发送读命令 (高电平，R)，对应图 6中 R/W#；
  - 接着从设备应答 (A)，ACK 为应答成功 (A)，表示有这个设备；
  - 从设备发送 8 位字节数据，主设备从 SDA 上读取数据；
  - 接着主设备应答，ACK 为应答成功 (A)，或，NACK 为不成功为 (N)；
  - 如果主设备应答成功 (A) 则从设备继续往 SDA 上送数据；
  - 若应答不成功 (N) 则主设备在发送 NACK 后发送停止信号 (P)，对应图 6中 (P) 状态。

SPI 部分：

- SPI 接口主要应用在 EEPROM，FLASH，实时时钟，AD 转换器，还有数字信号处理器和数字信号解码器之间。
- 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 数据传输过程:**
  - 通信过程由主设备（master）发起，从设备（slave）参与。当一个主设备需要向从设备发送数据，或者希望读取从设备数据的时候，主设备通过拉低对应

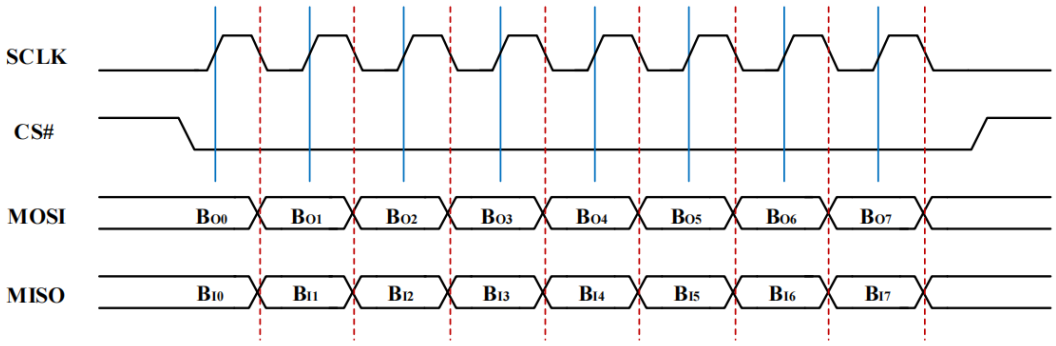


Figure 7: **SPI 数据传输过程**

从设备的 CS# 来告知从设备。对于主设备来说，发送数据就是把比特逐个放到 MOSI 信号线上，而读取数据就是在 MISO 信号线上进行采样。

- I^2C 与 SPI 区别:
  - I^2C 数据输入/输出用同一根线；SPI 输入输出分开
  - I^2C 占用端口更少，但因数据线双向，隔离与协议较复杂；SPI 较易
  - 一般系统内部通信用 I^2C；外部通信最好用 SPI 带隔离（提高抗干扰能力）

第五章

题目 1： 微架构的概念（了解）

答： 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，属于计算机的组织和实现技术

题目 2： 哈佛结构的特点以及与冯 · 诺依曼结构的区别（了解）

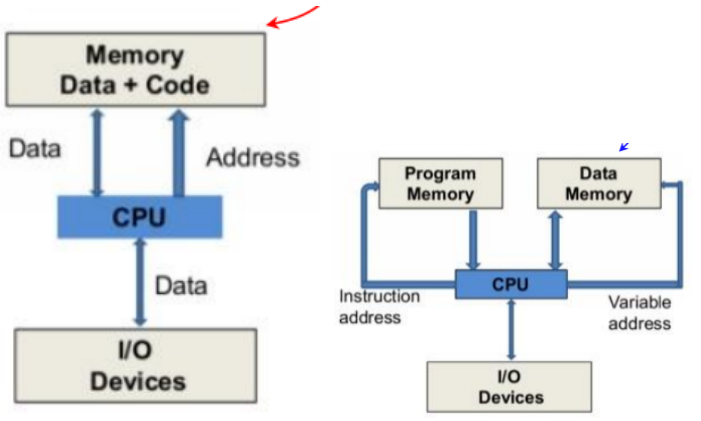
答： 两者架构示意如图 8所示。

哈佛结构特点:

- 使用两个独立的存储器模块，分别存储指令和数据，每个存储模块都不允许指令和数据并存。
- 使用独立的两条总线，分别作为 CPU 与每个存储器之间的专用通信路径，而这两条总线之间毫无关联。

冯 · 诺依曼结构特点:

- 必须有一个存储器；
- 必须有一个控制器；



(a) 冯 · 诺依曼结构　　(b) 哈佛结构

Figure 8: **哈佛结构的特点以及与冯 · 诺依曼结构的区别**

- 必须有一个运算器，用于完成算术运算和逻辑运算；
- 必须有输入和输出设备，用于进行人机通信。

题目 3：Cortex-M3/M4 处理器的存储器映射（掌握）

答：存储器映射如图 9所示。

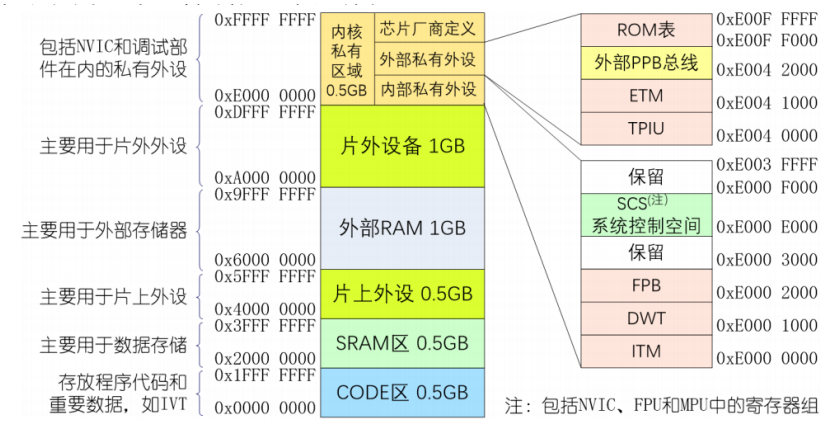


Figure 9: **存储器映射**

CODE 区

- 只能被 I-Code 和 D-Code 总线访问，为使两条总线能够同时访问，CODE 区可使用两个独立的存储器
- 最低端部分使用 flash 器件，用于存储程序代码
- 另外一部分使用 SRAM，存储重要数据
- I-Code 和 D-Code 对 CODE 区的所有空间都能访问

- 只要不是同时访问同一个存储器，两条总线上可以同时传输数据但是，为了降低成本，许多 SOC 芯片并没有在 CODE 区部署 SRAM

SRAM 区

- 又称为片上主存储区，存放数据
- 由 AHB-Lite 系统总线管理
- 许多芯片制造商只使用 SRAM 区存放数据

外设区

- 片上外设连接到系统总线上
- 片外外设与 APB 总线相连，APB 与 AHB 之间有一个总线桥

外部 RAM 区

- 通过 RAM 控制器连接片外扩展 RAM，可使用的 RAM 类型取决于 RAM 控制器

内核私有区域：0xE000 0000 0xFFFF FFFF，0.5GB

- 用于处理器的内部控制和调试部件
- 分为三部分：内部私有外设、外部私有外设、芯片厂商定义的存储区
- 内部私有外设：
  - ITM
  - DWT
  - FPB
  - SCS：系统控制区
- 外部私有外设：
  - ETM、TPIU 和 ROM 表，也都属于调试组件
  - 外部 PPB（private peripheral bus）总线

系统控制空间 SCS（System Control Space）

- 位于内核私有区域，0xE000 E000 0xE000 EFFF，64KB
- 该区域集中了 NVIC、Systick、FPU 和 MPU 等在内的各种系统部件的寄存器组，只有特权访问等级才能访问

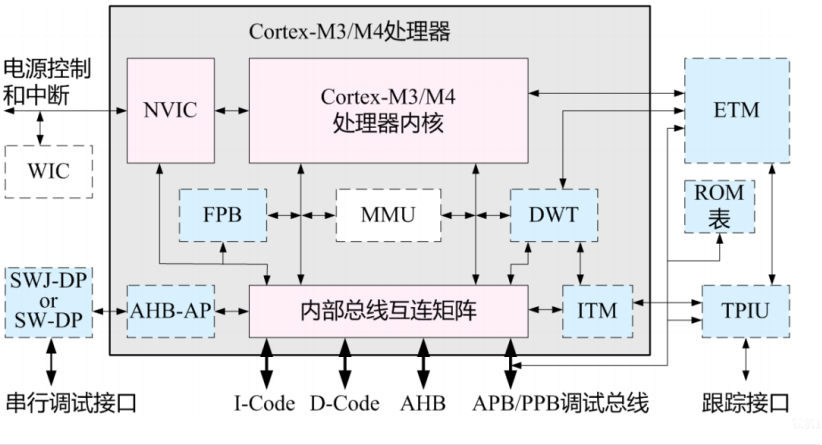


Figure 10: **总线系统**

有助于提高设备之间的软件可移植性和代码可重用性

题目 4：Cortex-M3/M4 处理器的总线系统（掌握）

答：总线系统如图 10所示

核心：基于 AHB 总线协议的内部总线互连矩阵。其所连接的各类总线的作用以及主要特性简介如下

- I-Code 总线，基于 AHB-Lite 总线协议的 32 位总线，负责在 0x0000 0000 0x1FFF FFFF 之间的**取指操作**。取指是以字为单位，对于 16 位的 Thumb 指令，一次取指操作可以取出两条指令
- D-Code 总线，与 I-Code 基本相同，但**只负责数据读写**
- I-Code 总线与 D-Code 在物理上彼此独立，但两者之间有一个仲裁器，当 I-Code 和 D-Code 同时访问同一区域时，**D-CODE 优先**

AHB，基于 AHB-Lite 的 32 位系统总线，访问区域包括：

- SRAM 区
- 片上外设
- 外部 RAM
- 片外设备
- 芯片厂商定义的区域

APB/PPB，32 位 APB 总线，访问区域包括：

- 外部私有外设子区域 0xE004 0000 0xE00F FFFF

- 但在外部私有外设子区域中，有一部分空间已被 ETM、TPIU 和 ROM 表等调试组件所占用，只有 0xE004 2000 0xE00F EFFF 可用于连接外部私有设备
- 由于内核私有区域需要特权访问权限，该总线一般是专门用于连接调试组件，不用于普通的外设，否则将会出现因特权管理导致的各种错误

四条总线各自管理的区域如图 11所示

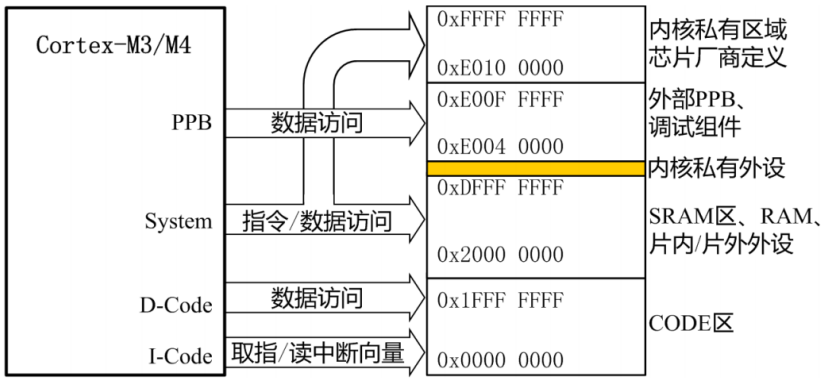


Figure 11: **四条总线各自管理的区域**

总线部分需要注意的内容：

- AHB 总线互联矩阵属于处理器内核设备
- CODE 区的总线矩阵或者总线复用器是 ARM 公司提供的两种不同的选件，其作用是让 I-Code 和 D-Code 都能够访问 CODE 区的 flash 和 SRAM
  - 如果选用总线矩阵，I-Code 对 flash 的取指操作与 D-Code 对 SRAM 的数据存取操作可以同时进行
  - 如果选用总线复用器，I-Code 和 D-Code 对 CODE 区的访问只能分时进行，数据传送不再有具有并行性
  - 有些芯片 CODE 区没有配置 SRAM，利用 SRAM 区存储数据，通过系统总线与 I-Code 总线的并行性传送数据，虽然浪费了一点存储空间，但可以降低成本
- 片上 SRAM 也称为主 SRAM，应该连接到系统总线上，使其位于 SRAM 区，以便可以使用位带操作
- 如果需要使用片外存储，必须使用片外 RAM 控制器作为接口，Cortex- M3/M4 系统总线不能直接连接片外存储器
- 总线矩阵或者总线复用器、片外 RAM 控制器、AHB 到 APB 的总线桥、AHB-AP 调试访问端口以及 UART 等各种外设接口都可以根据需要选配。

调试访问端口（DAP）

- 基于“增强型 APB”总线规范的 32 位总线
- 主要用于连接处理器内部的调试访问接口 AP 与外部调试端口 DP，如 SWJ-DP 和 SW-DP
- 框图如图 12所示

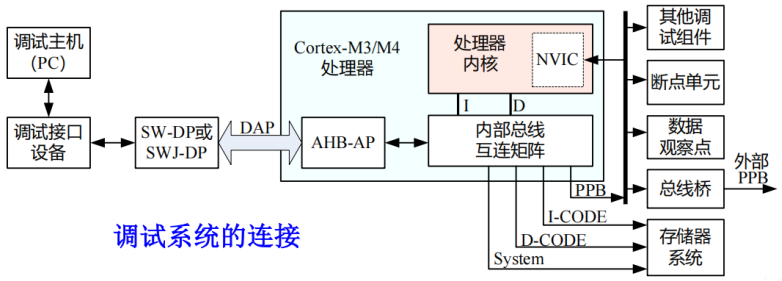


Figure 12: **调试访问端口**

题目 5：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）（了解）

答：

- 两种操作状态：Thumb 状态、调试状态
- 两种操作模式：处理模式、线程模式
- 两种访问等级：特权访问等级、非特权访问等级
- 切换原理：
  - 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
  - 但是非特权线程模式不能访问 CONTROL 寄存器，因此不能采用类似方式回到特权线程模式

题目 6：Cortex-M3/M4 处理器 16 个常规寄存器及程序状态寄存器 PSR（掌握）

答：

通用寄存器：R0~R12

- R0~R7，8 个低位寄存器，因受指令编码空间限制，许多 16 位 Thumb 指令只能访问低位寄存器

位	描述
N	N=1，结果为负
Z	Z=1，结果为零
C	C=1，出现进位
V	V=1，出现溢出
Q	Q=1，出现饱和 基于ARMv6-M版架构的Cortex-M0/M0+/M1不存在此位
GE[3:0]	大于或等于标志，分别对应每个字节的数据通路 ARMv6-M版架构以及Cortex-M3中没有
ICI/IT	Interrupt-Continuable Instruction/ IF-THEN指令状态标志位，用于指令条件执行， ARMv6-M无
T	Thumb指令标志。由于总是处于Thumb状态，所以该位总是1，清除此位会引起错误异常
异常编号	表示正在处理的异常/中断编号，是新的异常/中断能否实现抢占（嵌套）的主要依据

Figure 13: **程序状态寄存器 PSR**

- R8~R12，5 个高位寄存器，可用于 32 位指令和少数几个 16 位指令（如 MOV 指令）
- R0~R3：用于子程序之间的参数传递
- R4~R11：用于保存子程序的局部变量
- R12：作为子程序调用中间寄存器
- 栈指针：R13 ： MSP 为默认栈指针，在系统复位后或处理器处于处理模式时，处理器使用 MSP；PSP 只能用于线程模式
- 链接寄存器：R14（又称 LR）：用于保存函数或子程序调用时的返回地址，在函数或子程序结束时，LR 中的数值用于调用返回
- 程序计数器：R15（又称 PC）
- xPSR：程序状态寄存器，PSR 中各个标志位的含义如图 13

题目 7： 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（了解）

答：

- 堆栈是一种只能在一端进行插入和删除操作的线型表
- 压栈指令 PUSH，向堆栈中增加数据；出栈指令 POP，从堆栈中提取数据
- 按照堆栈区在存储器中的地址增长方向，可分为：
  - 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，又分为：
  - 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
- 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
- 双堆栈结构中，Cortex-M3/M4 有 MSP 和 PSP 两个堆栈指针（Main Stack、Process Stack），分别服务于不同的操作模式和特权访问等级
  - 处理模式总是使用 MSP、线程模式可以使用 MSP 或 PSP
  - 如果使用双堆栈，需要在 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
  - Cortex-M3/M4 的堆栈是满递减（FD，Full Descending Stack ）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，如图 14

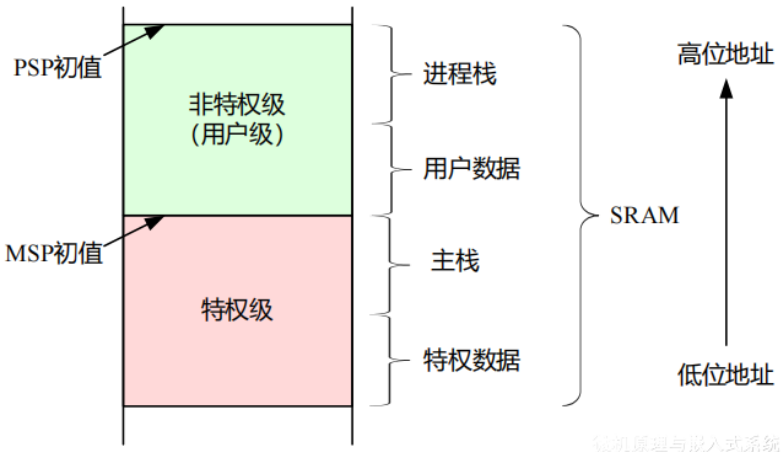


Figure 14: **双堆栈结构**

题目 8： 位段（位带）操作（了解）

答：

位操作就是可以读/写单独的一个比特位，在 STM32 中没有像 51 单片机的 sbit 来实行位定义，但是它可以通过位带别名区来实现。

在 STM32 中有两个地方实现了位带操作，一个是 SRAM 区的最低 1MB 空间，另一个是外设区最低 1MB 空间。

- 0x2000 0000　0x200f ffff (SRAM 区中的最低 1MB)
- 0x4000 0000　0x400f ffff (片上外设区中的最低 1MB，已覆盖了全部的片上外设的寄存器)

这两个 1MB 的空间可以像普通 RAM 一样操作外 (修改内容时用读-改-写)，它们还有自己的位带别名区，位带别名区把这 1MB 的空间的每一位膨胀为一个 32 位的字。确切的说，这个字就是一个地址，当操作这个地址时，就可以达到操作这个位带区某个位的目的。

在位带区中，每个比特位都映射到别名地址区的一个地址，注意，这只是只有 LSB 有效的字 (最低一位有效的字)。当一个别名地址被访问时，会把该地址转换为为位带操作。

对片上外设位带区的某个比特位，记它的所在字节的地址为 A，位序号为 n(0≤ n≤ 7)，则该比特位在别名区的地址为：

- AliasAddr = 0x42000000 + ((A - 0x40000000) * 8 + n) * 4 = 0x42000000 + (A - 0x40000000) * 32 + n * 4
- 4 表示一个字 4 个字节，8 表示一个字节 8 个比特。
- 在位带区中，不是以一个寄存器一个寄存器为分隔单元，而是以一个字节一个字节来分隔单元的。
  - A - 0x40000000 = 当前字节偏离外设基地址的偏移字节数
  - 偏移字节数 * 8 = 偏移了多少位
  - 因为位带区每一位对应位带别名区的一个地址 (4 字节)，而地址是以字节计算的，所以位带别名区对应偏移量最后一个的地址 = 偏移了多少位 * 4
  - n * 4 = 偏移量后面的 n 位对应位带别名区的地址

同理,对于 SRAM 位带区的某个比特位,记它所在字节地址为 A,位序号为 n(0≤n≤7),则该比特位在别名区的地址为：

- AliasAddr = 0x22000000 + ((A - 0x20000000) * 8 + n) * 4 = 0x22000000 + (A - 0x20000000) * 32 + n * 4

例如：

- 1) 往地址为 0x40000001 的位带区写入 0x4466aadd (0b100 0100 0110 0110 1010 1010 1101 1101)
- 2) 读取地址为 42000020 的位带别名区得 1 ，读取地址为 42000024 的位带别名区得 0 ，读取地址为 42000028 的位带别名区得 1
- 3) 往地址为 42000020 的位带别名区写 0，往地址为 42000024 的位带别名区写 1 后，地址为 0x40000001 的位带区的数据为 0b100 0100 0110 0110 1010 1010 1101 1110，即 0x4466AADE

为方便操作，STM32 有：

```
//把位带区地址 + 位序号转换成位带别名去的宏
#define BITBAND(addr, bit_num) ((addr & 0xf0000000) + 0x02000000 + ((addr & 0x00ffffff) << 5) + (bit_num << 2))
```

题目 9：异常处理的基本过程，及异常优先级及优先级分组（概念）（了解）

答：

- 异常状态：Inactive、Pending、Active、Active and pending
- 异常处理程序：中断服务子程序、错误程序、系统异常程序
- 异常向量表：向量表的起始是主栈指针（MSP）的初始值，包含了各个异常对应的处理程序入口地址（该入口地址也称作异常向量）。异常向量的地址为异常编号乘 4。并且，异常向量的 LSB 必须是 “1”，表示该异常处理程序是 Thumb 代码。
- 异常的优先级:Cortex-M 处理器在设计上具有 3 个固定的最高优先级（复位为-3、NMI 为 -2、硬件错误为-1）以及 256 个可编程优先级
- 中断优先级分组: 分组优先级（group priority），也称为抢占优先级（preempt priority），另外一类为子优先级（subpriority）
- 异常流程：进入异常处理；执行异常处理程序；异常返回
- 向量表重定位机制：VTOR（Vector Table Offset Register）
- 中断请求和挂起：
  - 挂起状态、挂起状态清除，进入/清除激活状态
  - 线程模式 → 处理模式 → 线程模式
- NVIC 寄存器：管理异常类型 16 255 的外部异常/中断
- SCB 寄存器：管理系统异常/中断

第六章

题目 1： T32 指令的汇编语法（了解）

答：

机器指令的要素：指令的功能、源操作数、目的操作数、操作数地址。

ARM 处理器汇编指令的通用格式如下：

<opcode>[cond][q][S] <Rd>, <Rn>[, Oprand2]

- 其中，<> 内的参数是必选参数，而 [] 内参数是可选参数。
- opcode，操作码，也称为助记符。
- cond：condition，条件码（可选后缀），描述指令的执行条件。
- q，可选后缀，指令宽度选择。
- S，可选后缀，含 S 代表指令执行会更新 APSR。
- Rd，目标操作数，总是一个寄存器。
- Rn，存放第一源操作数寄存器，该操作数必须是寄存器。
- Oprand2，第二源操作数，不仅可以是寄存器，还能是立即数，而且能用经过偏移量计算的寄存器和立即数。

T32 中多数 Thumb 指令可以根据应用程序状态寄存器 APSR 中的标志位决定当前指令是否被执行。

典型例子：

- MOV R0, R1 ; 寄存器 R1 中的数值装载到寄存器 R0
- MOVS R0, R1 ; 寄存器 R1 中的数值装载到寄存器 R0，更新 APSR
- MOVEQ R0, R1 ;Z==1 时把寄存器 R1 中的数值装载到寄存器 R0
- ITT EQ ; 随后的两条指令是条件执行的
- MOVEQ R0, R1 ； “Z == 1” 时执行，否则不执行
- MOVEQ R2, R3 ； “Z == 1” 时执行，否则不执行

题目 2： T32 指令集 10 种寻址方式（掌握）

答：

寻址——根据指令内容确定操作数地址的过程。

寻址方式——如何寻找操作数的方法。不同寻址方式实质上是构成操作数地址的方法不同。

寻址包括两种情形：

- 确定当前指令中的操作数地址，称作操作数寻址或简称数据寻址；
- 确定下一条待执行指令的地址，常称作指令寻址

寻址方式分类

- 1) 立即数寻址 #imm，示例：MOV R0 #66
- 2) 寄存器寻址 <Rn>，示例：ADD R0, R1, R2
- 3) 寄存器移位寻址 [<Rn> , LSL #imm]，示例：MOV R0, R2, LSL #3
- 4) 寄存器间接寻址 [<Rn>]，示例：LDR R0, [R1]
- 5) 寄存器偏移寻址 [<Rn>,<offset>]，示例：LDR R0, [R1, #4]
- 6) 前变址寻址 [<Rn>,<offset>]!，示例：LDR R0, [R1, #4]!（执行后 R1=R1+4）
- 7) 后变址寻址 [<Rn>],<offset>，示例：LDR R0, [R1], #4
- 8) 多寄存器寻址 <Rn>!, <registers>，示例：LDMIA R0!, R1, R2, R3, R4
- 9) 堆栈寻址 <SP>!, <registers>，示例：STMFD SP!, R1-R7 或 LDMFD SP!, R1-R7
- 10) PC 相对寻址 [PC, #imm] 或 Label，示例：ADR R2, start 或 LDR R0, [PC, #0xC]

操作数位置	可能的寻址方式	操作数地址
内含在指令中	1) 立即数寻址	直接从指令得到
存放在寄存器中	2) 寄存器寻址	即寄存器的地址
	3) 寄存器移位寻址	寄存器中存放的数值经过移位
在存储器数据区	4) 寄存器间接寻址	寄存器中存放的数值
	5) 寄存器偏移寻址	寄存器中存放的数值与偏移量相加
	6) 前变址寻址	基址寄存器中存放的数值与偏移量相加
	7) 后变址寻址	基址寄存器中存放的数值
	8) 多寄存器寻址	基址寄存器中存放的数值
	9) 堆栈寻址	堆栈指针寄存器 SP
在存储器代码区	10) PC 相对寻址	PC 寄存器中存放的数值与偏移量相加

(1)

题目 3：基本指令功能和用法：MOV、LDR、STR、PUSH、POP、ADD、SUB、B、BL，条件码（掌握）

答：

MOV	处理器内的数据传送指令	MOV R0, R2
LDR	存储器读取指令	LDR R1, [R0]
STR	存储器写入指令	STR R1, [R0]
PUSH	压栈指令	PUSH {R4, LR}
POP	出栈指令	POP {R4, PC}
ADD	加法	ADD Rd, Rs1, Rs2
SUB	减法	SUB Rd, Rs1, Rs2
B	无条件跳转	B loop1
BL	调用函数	BL func1
条件码	操作码助记符的条件码后缀	MOVEQ R0, R1

(2)

条件码后缀：

条件码	助记符	含义	标志位取值
0000	EQ	等于 (Equal)	$Z == 1$
0001	NE	不等于 (Not equal)	$Z == 0$
0010	CS	产生了进位 (Carry set)	$C == 1$
0011	CC	进位为零 (Carry clear)	$C == 0$
0100	MI	负数 (Minus, negative)	$N == 1$
0101	PL	整数或零 (Plus, positive or zero)	$N == 0$
0110	VS	溢出 (Overflow)	$V == 1$
0111	VC	没有溢出 (No overflow)	$V == 0$
.....		.....	

(3)