《微机原理与嵌入式系统》第七章与第八章前四节内容总结

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。

第七章

题目 1：数据定义伪指令的用法（掌握）

答：

数据定义伪指令一般用于为特定的数据分配存储单元，同时可完成已分配存储单元的初始化。

- DCB 用于分配一段连续的字节存储单元并用指定的数据初始化
- DCW（DCWU）用于分配一段连续的半字存储单元并用指定的数据初始化
- DCD（DCDU）用于分配一段连续的字存储单元并用指定的数据初始化
- DCFD（DCFDU）用于为双精度的浮点数分配一段连续的字存储单元并用指定数据初始化
- DCFS（DCFSU）用于为单精度的浮点数分配一段连续的字存储单元并用指定数据初始化
- DCQ（DCQU）用于分配一段以 8 字节为单位的连续的存储单元并用指定的数据初始化
- SPACE 用于分配一段连续的存储单元
- MAP 用于定义一个结构化的内存表首地址
- FIELD 用于定义一个结构化的内存表的数据域

题目 2：能读写完整的汇编程序（掌握）

答：

- ARM 汇编语言的语句格式:{symbol} {instruction | directive | pseudo-instruction} {;comment}
  - symbol: 通常就是标号，在一些 directive 中有时为常数或者变量 (代码中要顶格写)
  - Instruction | directive | pseudo-instruction:
    - 如果前面没有 symbol，必须有空格或者 TAB
    - Instruction，即 ARM 或 Thumb 指令
    - pseudo-instructions，ARM 汇编器支持少数几条伪指令，伪指令在编译的时候被翻译为一组 ARM 或 Thumb 指令
    - directive，指示符，指示 ARM 汇编器行为
  - comment，注释以英文符号 “;” 起为注释
- ARM 汇编语言程序结构
  - 在 ARM（Thumb）汇编语言程序中，通常以段为单位来组织代码段是具有特定名称且功能相对独立的指令或数据序列根据段的内容，分为代码段和数据段。
  - 一个汇编程序至少应该有一个代码段，当程序较长时，可以分割为多个代码段和数据段。
  - 示例可参考 PPT 第 54 页

题目 3：C 程序调用汇编函数及汇编程序调用 C 函数的编程方法（掌握）

答：

- 这部分实践内容居多，大家最好参考汇编实验讲义，自己尝试一下编程。我会发一下之前我写的实验代码作为参考。
- C 语言与汇编语言之间的函数调用
  - ATPCS 规定了一些不同语言撰写的函数之间相互调用（mix calls）的基本规则，这些基本规则包括子程序调用过程中**寄存器的使用规则**、**数据栈的使用规则**、以及**参数的传递规则**。
    - 寄存器的使用规则：

· 1）子程序间通过寄存器 R0 至 R3 来传递参数。这时，寄存器 R0 至 R3 可记作 a0 至 a3. 被调用的子程序在返回前无需恢复寄存器 R0 至 R3 的内容。

· 2）在子程序中，使用寄存器 R4 至 R11 来保存局部变量。这时，寄存器 R4 至 R11 可以记作 vl 至 v8。如果在子程序中使用了寄存器 v1 至 v8 中的某些寄存器，则子程序进入时必须保存这些寄存器的值，在返回前必须恢复这些寄存器的值。在 Thumb 程序中，通常只能使用寄存器 R4 至 R7 来保存局部变量。

· 3）寄存器 R12 用作过程调用中间临时寄存器，记作 IP。在子程序之间的连接代码段中常常有这种使用规则。

· 4）寄存器 R13 用作堆栈指针，记作 SP。在子程序中寄存器 R13 不能用作其他用途。寄存器 SP 在进入子程序时的值和退出子程序时的值必须相等。

· 5）寄存器 R14 称为连接寄存器，记作 LR。它用于保存子程序的返回地址。如果在子程序中保存了返回地址，寄存器 R14 则可以用作其他用途。

· 6）寄存器 R15 是程序计数器，记作 PC。它不能用作其它用途。

* 数据栈使用规则:

· 数据栈指针根据指向位置和增长方向的不同可分为 4 种:FD(Full Descending)，ED(Empty Descending)，FA (Full Ascending) 和 EA (Empty Ascending)。当栈指针指向栈顶元素时，称为 Full 栈。

· ARM 的 ATPCS 规定默认的数据栈为 Full Descending (FD）类型，SP 指向最后一个压入的值，数据栈由高地址向低地址生长类型，即满递减堆栈，经常使用的指令有 STMFD 和 LDMFD。并且对数据栈的操作是 8 字节对齐的。对于汇编程序来说，如果目标文件中包含了外部调用，在汇编程序中需要使用 PRESERVE8 伪指令告诉链接器，本汇编程序堆栈数据是 8 字节对齐的。

* 参数传递规则：当参数个数不超过 4 个时，可以使用寄存器 R0 ~ R3 来传递参数，如果参数多于 4 个，则将剩余的字数据通过堆栈传递，入栈的顺序与参数传递顺序相反，即**最后一个字数据先入栈，第一个字数据最后入栈**。

- C/C++ 语言与汇编语言的混合编程

  – 在 C 程序中嵌入汇编常用两种方法: 内联汇编 (inline assembler)、内嵌汇编 (Embedded assembler)

    * 内联汇编可以直接嵌入 C 代码使用，而内嵌汇编更像是一个函数。由于内联式汇编只能在 ARM 状态中进行，而 Cortex-M3/M4 只支持 Thumb-2，

所以 Cortex-M3/M4 只能使用内嵌汇编的方式，也就是第二种方式。

* 在 C 语言中内嵌汇编指令与汇编程序中的指令有些不同，存在一些限制，主要有下面几个方面:
  - （1）不能直接向 PC 寄存器赋值，程序跳转要使用 B 或者 BL 指令。
  - （2）在使用物理寄存器时，不要使用过于复杂的 C 表达式，避免物理寄存器冲突。
  - （3）R12 和 R13 可能被编译器用来存放中间编译结果，计算表达式值时可能将 R0 到 R3、R12 及 R14 用于子程序调用，因此要避免直接使用这些物理寄存器。
  - （4）一般不要直接指定物理寄存器，而让编译器进行分配。

– 使用内联或者内嵌汇编不用单独编辑汇编语言文件使用起来比较方便，但是有诸多限制。当汇编文件较多的时候就需要使用专门的汇编文件编写汇编程序，在 C 语言和汇编语言进行数据传递的最简单的形式是使用**全局变量**。

第八章前四节

题目 1：嵌入式系统的交叉开发环境（了解）

答：

宿主机和目标机是不同的机器。嵌入式软件在宿主机上使用嵌入式开发工具进行编写、编译、链接和定位，生成可以在目标机上执行的二进制代码，然后通过 JTAG 接口、串口或以太网接口将代码下载到目标机上进行调试。调试完后，将二进制代码烧录进目标机微处理器 ROM 运行。

题目 2：嵌入式系统开发过程各阶段（理解）

答：

- 需求分析
  - 系统规格说明书, 含功能性/非功能性需求
- 系统设计 (也称总体/概要设计)
  - 体系架构设计
  - 软硬件划分
  - 硬件设计 (处理器、外设、器件及开发工具的选择)
  - 软件设计 (软件架构设计和软件模块划分 (有/无 OS）)
- 系统实现

  - 又称详细设计，包括硬件和软件实现

- 系统测试
  - 包括：测试方法、工具及步骤

- 系统发布

题目 3：微处理器最小硬件系统概念（了解）

答：

仅含正常工作所需最少元件: 电源、时钟、复位、调试和下载.

- 电源模块
  - 内核和 IO 接口能量来源
  - 直接影响系统工作稳定性：电压、内阻、波纹、驱动能力、防干扰等

- 时钟模块
  - 提供同步工作信号，供各模块使用
  - 系统主时钟（高频率）
  - 实时时钟（低频率）
  - 倍频和同步处理后不同频率时钟信号

- 复位模块
  - 加电和手动复位：信号来自外部复位电路，从 nRESET 引脚输入
  - 内部复位：信号来自系统内部事务处理，如看门狗复位

- JTAG 调试接口模块
  - 完成基本调试工作

- 外部存储器模块
  - 保存和运行系统程序：SRAM（引导）、SDRAM（程序运行）、NAND Flash（存放程序）

题目 4：STM32 时钟树的基本概念、功能、作用、意义、特点等（理解）

答：

太长不看版 ↓

- 基本概念：时钟系统是由振荡器（信号源）、定时唤醒器、分频器等组成的电路。常用的信号源有晶体振荡器和 RC 振荡器
- 功能：给不同模块提供合适的时钟信号
- 作用: 有了时钟树，就有了时钟域。嵌入式中除了内核，还有各个单元，每个单元工作在不同的时钟频率下，时钟树就是给每个单元提供不同的时钟的。不同的时钟源，通过分频或者倍频处理后送到相应的外设单元。实际应用中根据需要配置外设的时钟控制开关，选择需要的时钟频率，并可关闭不用外设时钟。
- 意义: 时钟是嵌入式系统的脉搏，处理器内核在时钟驱动下完成指令执行，状态变换等动作。外设部件在时钟的驱动下完成各种工作，比如串口的数据发送，A/D 转化，定时器计数等。因此时钟对于计算机系统是至关重要的，通常时钟系统出现问题也是致命的，比如振荡器不起振，振荡不稳，停振等。
- 特点: 时钟树从左至右，相关时钟依次可分为 3 种：输入时钟、系统时钟和由系统时钟分频所得其他时钟

详细版 ↓

在 STM32 中每个外设都有其单独的时钟，在使用某个外设之前必须打开该外设的时钟，为什么要这么麻烦来设置每一个外设的时钟而不是将所有外设的时钟统一打开? 因为 STM32 的外设繁多，外设的运作所需要的最佳时钟各不相同，如果所有时钟同时运行会给微处理器（MCU）带来极大的负载，所以 STM32 采取自助式的时钟管理方式——随用随开。

认识 STM32 的时钟管理方式后就要对其时钟树系统有一个大体的掌握，时钟树系统的结构如下，从图 1中可以知晓有多少时钟、时钟速度为多少、时钟如何分配等。图看起来很复杂，但我们只需要大体了解其运作流程即可。

- ①——输入，外部晶振 HSE，可选为 2 16MHZ
- ②——第一个分频器 PLLXTPRE，可选 1 分频/2 分频
- ③——时钟源选择，开关 PLLSRC，可选其输出为：外部高速时钟 HSE 或内部高速时钟 HSI
- ④——锁相环 PLL，具有倍频功能 (2 至 16 倍)，经过 PLL 的时钟称为 PLLCLK，若设 9 倍频，即从 8MHz 的 HSE 变为 72MHz
- ⑤——开关 SW，经过 SW 后即系统时钟 SYSCLK。SW 可选 SYSCLK 时钟源为以下之一：HSI、PLLCLK、HSE
- ⑥——AHB 预分频器，分频系数为 1/2/4/8/16/64/128/256/512

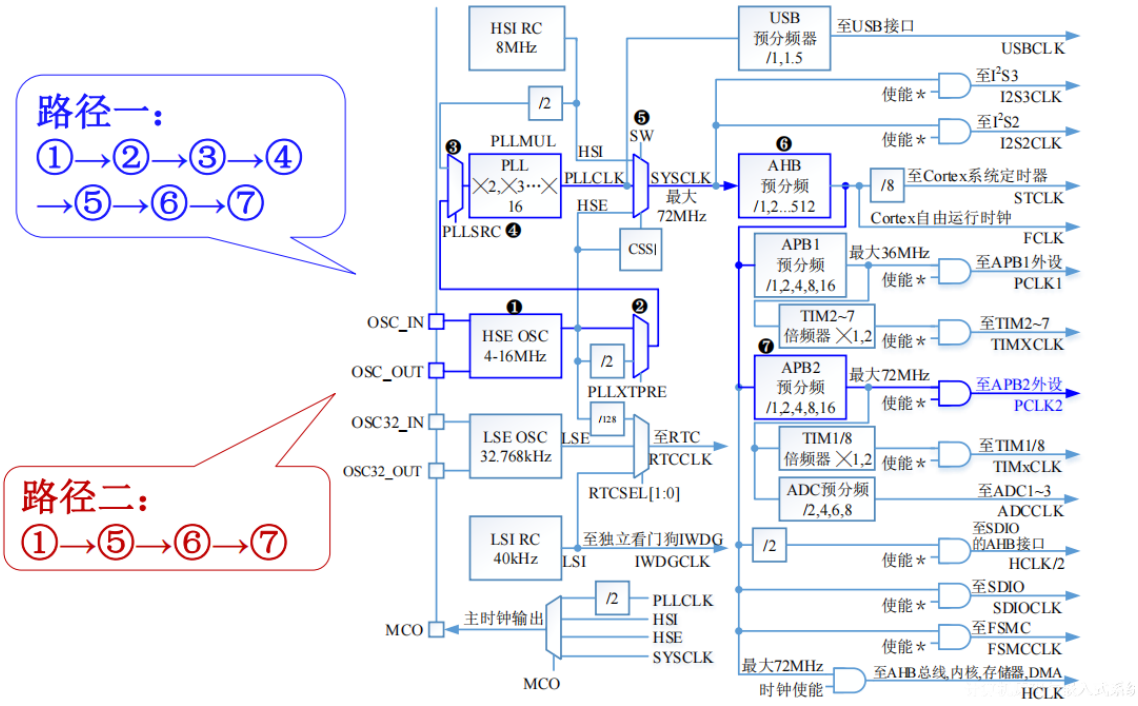

Figure 1: **时钟树示意图**

- ⑦——APB2 预分频器，分频系数为 1/2/4/8/16。若为 1，则高速外设 APB2（PCLK2）为 72MHz（AHB 输出为 72MHz 时）

时钟树从左至右，相关时钟依次可分为 3 种：**输入时钟**、**系统时钟**和**由系统时钟分频所得其他时钟**。

输入时钟：

- 从时钟频率分：
  - 高速时钟和低速时钟
- 从芯片角度分：
  - 内部时钟 (片内时钟) 和外部时钟源 (片外时钟)
- 结合频率及内外，有以下 5 种：
  - 高速外部时钟 HSE
    - ①→②→③→④→⑤得 SYSCLK。来源为外部无源晶振，通常速度 8M。由 RCC_CR 时钟控制寄存器中的 16:HSEON 控制。
  - 高速内部时钟 HSI
    - 由片内 RC 振荡器产生，不稳定，上电开始作为初始系统时钟，来源为芯片内部，大小为 8M，当 HSE 故障时，系统时钟会自动切换到 HSI，直到 HSE 启动成功，相当于 HSE 的替补。由 RCC_CR 时钟控制寄存器的位 0:HSION 控制。

  - 低速外部时钟 LSE
    * 外部晶振，提供给实时时钟，输出 32kHz
  - 低速内部时钟 LSI
    * 片内 RC 振荡器产生，提供给实时时钟和看门狗，输出 40kHz
  - 锁相环倍频输出 PLL
    * 输入源可选 HSI/2、HSE 或 HSE/2，倍频 2 16 倍，输出最大 72MHz，由 CFGR(时钟配置寄存器) 中 PLLXTPRE、PLLMUL 控制。

系统时钟 SYSCLK：

- 由 SW 根据用户设置，选择以下 3 个中的一路输出而得：PLLCLK、HSE 或 HSI
- 最高可达 72MHz(通常即 72MHz)
- 大部分部件时钟来源，用 AHB 预分频器分配至各部件
- 通常，上电开始，选用 HSI 作为初始系统时钟. 完成初始化后，选用更加稳定可靠的 HSE 作为系统时钟来源
- 专门提供引脚 MCO(主时钟输出)，以实时检测时钟系统是否运行正常。通过软件编程，选择以下 4 个中的一路在 MCO 上输出：SYSCLK、PLLCLK、HSE、HSI

由系统时钟分频所得其他时钟：

- 即 SYSCLK 经过 AHB 预分频器输出
- HCLK
  - AHB 时钟 (通常，预分频系数为 1，常为 72MHz)
  - 最高 72MHz
  - 为内核、存储器和 DMA 时钟信号
- FCLK
  - 内核“自由运行”时钟
  - 最高 72MHz
  - 与 HCLK 互相同步
  - HCLK 停止时仍能继续运行，保证内核睡眠时也能采样到中断和跟踪休眠事件
- PCLK1

  - 外设时钟
  - 再经 APB1 预分频器 (系数常为 2) 后得到
  - 最高 36MHz(常为 36MHz)
  - 为 APB1 总线上 (低速) 外设时钟 (如需使用外设，须先开启其时钟)
- PCLK2
  - 类似 PCLK1
  - 外设时钟
  - 最高 72MHz(常为 72MHz)
  - 为 APB2 总线上 (高速) 外设时钟
- SDIOCLK
  - SDIO 外设时钟
- FSMCCLK
  - 可变静态存储控制器时钟
- STCLK
  - 系统时间定时器 SYSTICK 的外部时钟源
  - AHB 输出再经过 8 分频后得到
  - 等于 HCLK/8
- TIMXCLK
  - 定时器 2 到定时器 7 内部时钟源
  - PCLK1 经过倍频 (*1 或 *2, 由 APB1 分频系数是否为 1 判断得出) 所得
- TIMxCLK
  - 定时器 1 和定时器 8 内部时钟源
  - PCLK2 经过倍频 (*1 或 *2, 由 APB2 分频系数是否为 1 判断得出) 所得
- ADCCLK
  - ADC1 到 ADC3 时钟
  - PCLK2 经过 ADC 预分频器 (/2,4,6,8) 所得

时钟输出中大多带使能控制，如 AHB 总线、内核、各种 APB1 外设、APB2 外设等时钟。当需使用它们时，须先使能对应时钟

时钟输出：

连接在 APB1 上的设备（低速外设）有：

- 电源接口、备份接口、CAN、USB、I^2C1、I^2C2、UART2 至 5
- SPI2/I^2S、SPI3/I^2S、BKP、IWDG、WWDG、RTC、CAN
- DAC、PWR、TIM2 至 7

连接在 APB2 上的设备（高速外设）有：

- GPIOA 至 G、TIM1、TIM8、USART1、ADC1 至 3、SPI1
- EXTI、AFIO

使用 HSE 时钟，程序设置时钟参数流程：

1、将 RCC 寄存器重新设置为默认值 RCC_DeInit;

2、打开外部高速时钟晶振 HSE RCC_HSEConfig(RCC_HSE_ON);

3、等待外部高速时钟晶振工作 HSEStartUpStatus = RCC_WaitForHSEStartUp();

4、设置 AHB 时钟 RCC_HCLKConfig;

5、设置高速 AHB 时钟 RCC_PCLK2Config;

6、设置低速速 AHB 时钟 RCC_PCLK1Config;

7、设置 PLL RCC_PLLConfig;

8、打开 PLL RCC_PLLCmd(ENABLE);

9、等待 PLL 工作 while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)

10、设置系统时钟 RCC_SYSCLKConfig;

11、判断是否 PLL 是系统时钟 while(RCC_GetSYSCLKSource() != 0x08)

12、打开要使用的外设时钟 RCC_APB2PeriphClockCmd() 或 RCC_APB1PeriphClockCmd()