第一章①冯诺依曼结构的组成 计算机硬件由以下五个功能部件组成：存储器，运算器控制器，输入设备输出设备，此外还需要将这五个部件进行互连的总线和适配器。②有符号数的表示方法及补码的求解，原码和反码相同，而补码表示范围是比它们多一个负数的最小值。补码的溢出，当且仅当符号位向前进位和次高位向前进位异或等1，结果会发生溢出。BCD码是8421码，四位二进制数，压缩BCD码一个字节表示两位十进制数，非压缩BCD码一个字节表示一位BCD码，低4位为0000.

第二章 ①模型机指令执行流程：主要对 RISC的一条指令完成从取指，译码，执行三个主要过程的动作描述，对于第一条指令进行描述时基于如下假设:① 模型机是 32 位，数据总线 32 位，地址总线 32 位②)RISC 结构，定长指令设计，每条指令长度也是32位③ 待运行程序的首地址为 0x2000 0000，描述过程如下：① PC 内容 0x2000 0000 送至地址缓冲器/驱动器，地址总线的输出经地址译码器译码，寻址内存单元②PC 值自动加 4(假设 PC内容以字节为单位)，指向下一条指令的存放地址③ OC 发读信号，将“E3 A006FF”读出到数据总线:④ 由于是取指操作，数据总线上的数据被装入IR⑤D 对操作码译码，OC产生相应的控制信号⑥ 第一条指令源操作数是立即数(取指时能从指令编码中立即得到)，被装入 RO寄存器，指令执行完毕。在对指令进行动作分解描述时，取指、译码的步骤基本相同，也就是 1~5 步骤操作为模板可以基本套用，在 6 步骤可以根据该指令的具体命令完成相应的功能。

②流水线基本原理，典型的三级、五级流水线划分，三种相关冲突及解决：基本原理:

口将功能部件按指令操作步骤顺序进行排列部署，前后部件之间增加缓冲寄存器，构成指令处理流水线。

口前后两个部件经过缓冲寄存器隔离后，可以相对独立地并行工作。口部件之间的工作交换(数据传递)将通过缓存寄存器进行。口这种缓存寄存器被称为流水线寄存器

一个流水线需要n个步骤成为n级流水线，插入n-1个流水线寄存器。

三种相关冲突：一、资源相关，也称结构相关：。多条指令在同一个周期内用同一个公用部件。例如:冯诺依曼结构计算机的Fetch、Load 和 Store操作都使用公共总线访问同一个寄存器，前一条指令的数据存取操作可能会影响后续指令的取指操作

。解决办法:后面一条指令等待一个节拍再启动。称作向流水线插入气泡(Bubble)或插入阻塞，这将造成流水线性能下降；采用哈佛结构(指令和数据分别存储)。接触存取操作数与取指之间的资源相关。

1. 数据相关：后一条指令执行需要使用前一条指令的结果。常见：RAW，还有WAW，WAR。插入气泡可消除数据相关，但将造成流水线性能下降。解决之道:①定向推送，前一条指令执行结果通过专用通道直接推送给下一条，减少一个流水线周期，可减少数据相关②优化编译器，对前后指令进行检查，调整执行顺序
2. 控制相关：遇到转移指令时，后续已进入流水线的指令都应清空，以无条件转移(包括子程序调用)指令为例:假设指令L是无条件转移指令，其执行步骤为:取指、译码、计算转移地址(Computer)并更新程序计算器PC。在第4个周期读取转移目标指令。在此之前流水线上的指令1+和1+2应清除，造成流水线断流。产生两个流水线周期延迟被称为转移代价。减少转移代价的方法：对于无条件转移指令，增加电路，在译码阶段提前计算转移目标地址，在第3个周期读取转移目标指令!，将转移代价减少到一个流水线周期，类似方法同样适用于少数条件转移指令。但是，大多数条件转移指令是否转移取决于状态标志位，而标志位在ALU运算后才更新，转移代价较大。流水线级数却多，代价越大。该如何解决呢?转移延迟槽，:转移指令,后面的一个时间片。无论是否转移，位于转移延迟槽的指令总是会被执行。一个启示:〇如果能对转移与否做出正确预测，则可根据预测结果选择合适的指令“装入”转移延迟槽。例如，假设预测结果是转移，位于转移延迟槽的恰好是转移目的指令，转移没有任何代价，然后是对转移指令的行为进行预测：根据转移指令过去的行为进行预测；对发生转移的可能性进行加权量化。；使用BTB(branch target bufer，转移目标缓冲器)，收集和存储了近期所有转移指令的有关信息，并按照查找表的形式进行组织。每条指令在取指时，处理器根据其地址在BTB中进行快速搜索，如果有记录则表明转移指令，再根据其“档案”记录进行相应处理，最后再根据实际行为修正权值。

③微程序设计思想：三层模型：指令系统层，微体系结构层（微程序），硬件子层：CPU、存储器等。其特点为：一条指令可以分解为多个微操作、微操作可以用微指令实现、多条微指令组成微程序实现指令功能、微程序存储在 ROM中，执行时逐条读出完成微操作。

④存储器分级设计思想：要想速度快、容量大、成本低，所用：分级存储体系结构，：O 使用外存满足大容量、低成本和非易失的要求O 使用DRAM型内存，兼顾容量、速度和成本0使用高速缓存，减少CPU访问内存的开销



⑤小端与大端：大端：低位高地址，高位低地址，小端：低位低地址，高位高地址。Intelx86用小端，Motorola用大端

⑥字长与字的对齐：字长:机器字长是指 CPU一次能处理数据的位数，通常与 CPU 的寄存器位数有关。字长越长，数的表示范围越大，精度也越高。对齐原因：实际中在访问特定类型变量时经常在特定的内存地址访问，为了使得 CPU能快速对变量进行访问，变量存的起始地址必须具备某些特性，即“对齐”，对齐理由：① 根本原因在于 CPU 访问数据的效率问题②)不同硬件平台对存储空间的处理上存在不同③ 节约空间。

⑦RISC与CISC特性与区别

CISC：① 指令长度不一(没有采用定长指令，最短1字节(CLC)，最长6字节，增加控制的复杂度和规模，不利于采用流水线和超标量等新技术)② 指令执行时间不同(指令长度不一导致的。)③ 非 Load/Store 体系(指令更加精简，相比于 RISK，操作量更低)④ Move操作(Move destination,source的传送指令，可实现寄存器与寄存器之间，以及寄存器与存储器之间的数据(复制)传送)⑤ 两操作数:OPR(操作码) DST(目的操作数)，SRC(源操作数)⑥ 指令功能强大，寻址方式多样，程序简介//微程序控制

RISC：① 寻址方式简单，种类较少;② 指令集中的指令数量较少;③ Load/Store 体系结构;④每条指令长度一致，执行时间相同:⑤ 面向寄存器的编程思想;⑥算数和逻辑运算指令普遍支持三操作数;⑦只能对寄存器操作数进行算数和逻辑运算:⑧ 程序代码量较大，因为执行复杂操作需要使用较多的简单指令//硬连线方式控制器

⑧计算机性能评测：评测指标:① 机器字长(影响精度和运算速度)②) 存储容量(高速缓存和内存容量越大，存取速度越快、处理能力越强)③ 总线带宽和数据吞吐速度(对处理速度有一定影响)④ 能耗与环保⑤ RASIS 特性(Reliability 可靠性;Availability 可用性;Serviceability 可维护性;Integrity 完整性;Security 安全性;RASIS)⑥ 运算速度(最重要的一项指标。CPU 或内核数量、高速缓存和主存容量、总线传输速度共同影响运算速度)

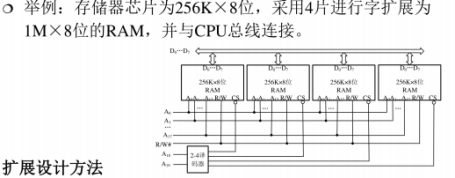
第三章存储器

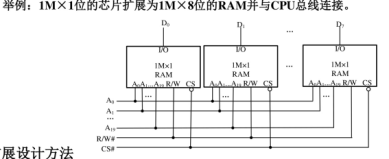
①地址译码、字线、位线

掩膜ROM：存储矩阵、地址译码器和输出缓冲器三个组成部分。存储矩阵由许多存储单元排列而成。存储单元可以用二极管构成，也可以用双极型三极管或MOS管构成。每个单元能存放一位二值代码(0或1)。每一个或一组存储单元有一个对应的地址代码，/地址译码器的作用是将输入的地址代码译成相应的控制信号，利用这个控制信号从存储矩阵中将指定的单元选出，并把其中的数据送到输出缓冲器。/三态输出缓冲器的作用有两个，一是能提高存储器的带负载能力，二是实现对输出状态的三态控制，以使与系统的总线连接与隔离，

②地址空间与存储器连接，存储器的位扩展、字扩展 按字编址、按字节编址，还有地址线的条数决定了地址空间。

微机中CPU与存储器的连接：设计微机系统时，存储器应与地址、数据、控制总线正确连接，并应考虑如下问题:

一、CPU总线的负载能力/CPU通过总线直接驱动负载的能力有限，应根据需要连接的存储器芯片参数，考虑在总线上增加缓冲器或驱动器，增大CPU的负载能力。二、CPU时序与存储器存取速度间的配合/CPU要对存储器频繁读/写，选芯片时要考虑其存取速度能否与CPU读/写时序匹配。三、存储器的地址分配和片选/需要为存储器分配地址范围。1由于每块芯片存储容量有限，一个存储器系统可能是由多块芯片组成，要重点考虑容量的扩充方案和片选信号的形成。四、控制信号的连接/CPU提供的存储器控制信号，如CS#、OE#、WE#等，应与存储器的相关引脚正确连接 ，才能实现读/写等控制功能。关于位扩展与字扩展和混合扩展

③cache基本工作原理及作用（描述概念）

在CPU的所有操作中，存储器的存取访问是最频繁的操作CPU速度比DRAM存储器的存取速度高得多

存储器的访问速度低是制约计算机系统性能的关键因素程序在一定时间段内通常只访问较小的地址空间

o时间局部性:最近访问的信息很可能再次被访问

o空间局部性:最近访问信息的邻近信息可能被访问

根据程序访问的时空局部性，把经常访问的代码和数据保存到高速缓冲存储器(Cache)中，把不常访问的代码和数据保存到大容量的相对低速DRAM中，尽量减少CPU访问DRAM的概率，在保证系统性能的前提下，降低存储器系统的实现代价

Cache设置在CPU与主存储器之间，通常采用存取速度快并且无需刷新的SRAM来实现

第四章总线与接口

①总线操作和时序

通过总线进行信息交换的过程称为总线操作。如读存储器、写存储器、读 I/O 端口和写 I/O 端口等。

总线设备完成一次完整信息交换的时间称为总线周期(或总线传输周期)，阶段如下:

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/0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。

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。

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，全互锁方式：主从设备相互等待，传输可靠性最高。

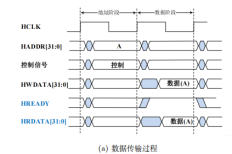
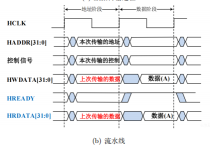
周期分裂式总线时序：分离式总线的思想:将一个传输周期(或总线周期)分解为两个子周期(或者称为子阶段)。

在第一个子周期(寻址子周期，或称为地址阶段)中，主块A获得总米线使用权后，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 模块根据所行收到的命令，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。

②AHB数据传输过程，AHB“流水线”分离操作（理解）

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

-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。“SPLTT”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2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。

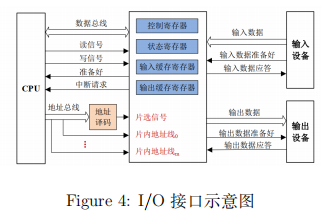
当主机获得总线使用权后，重新发送地址、控制等信息，继续刚才挂起的传输操作。

③USB、PCIe（了解）

USB: USB(universal serial bus)，通用串行总线，是一种外部总线标准。用于电脑与外部设备的连接和通讯，是应用在 PC 领域的接口技术。USB 接口支持设备的即插即用和热插拔功能。在 USB1.0/USB2.0 中,“D+”和“D-”组成一对差分信号线用于数据传输，VBUS和 GND 对应 5V 电源和地。USB3.0 后，又增加了两对差分信号线以提供更高速的数据传输。

PCIe:两个设备间的一条 PCLE 链路(link)可包含1至 32 个通道(lane)。习惯上用 X1、X4、X8、X16、X32 等方式表示链路所包含的通道数目，也称为 PCIE 的“宽度”或“位宽”。单个通道(lane)包含两对差分传输信号线，一对用于接收数据，另一对用于发送数据。故每个通道共4根信号线(wire)。

④I/O接口电路的典型结构（了解）

第五章ARM处理器体系结构和编程模型

①微架构的概念：ISA的硬件实现方式为微架构，即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。

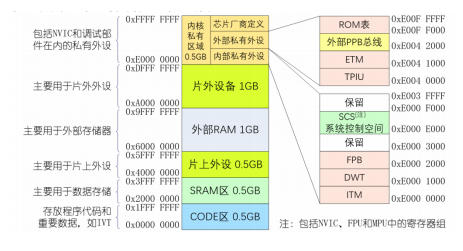
②哈佛结构的特点以及冯诺依曼结构的区别

哈佛：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：必须有一个存储器;必须有一个控制器:必须有一个运算器，用于完成算术运算和逻辑运算;，必须有输人和输出设备，用于进行人机通信。

③cortex-m3m4处理器的存储器映射

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，·ICode 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 类型取決于 RAM 控制器内核私有区域:0xE000 00000xFFFFFFFF，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 E0000xE000EFFF，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

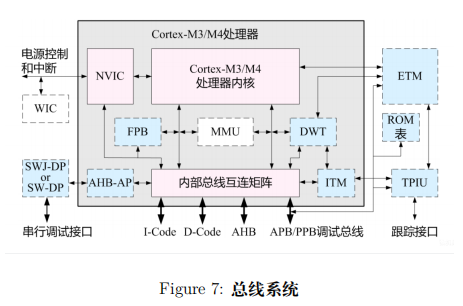
有助于提高设备之间的软件可移植性和代码可重用性

④cortex-m3m4处理器总线系统

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

ICode 总线，基于 AHB-Lite 总线协议的 32 位总线，负责在 0x0000 0000 0x1FFFFFFF 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，但只负贵数据读写

·ICode 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 0xEO0F 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 8所示

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 ICode 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

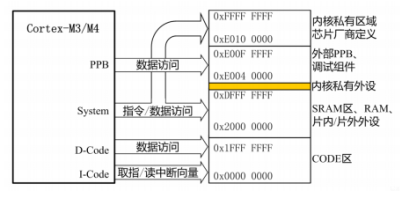
如果选用总线复用器，I-Code 和 D-Code 对 CODE 区的访问只能分时进行数据传送不再有具有并行性

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 ICode 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

⑤cortex-m3m4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）

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CON-TROL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，因此不能采用类似主回到特权线程模式

⑥cortex-m3m4处理器16个常规寄存器及程序状态寄存器PSR

通用寄存器:RO~R12

RO~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

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)

·BO~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：N负，Z零，C进位，V溢出，Q饱和，GE[3:0]大于或等于标志，ICI/IT指令状态标志位，T为Thumb指令标志，异常编号表示正在处理的异常/中断编号

⑦堆栈

堆栈是一种只能在一端进行插人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):向堆栈写人数据时，堆栈区由低地址向高地址生长；递减栈(Descending Stack):向堆栈写人数据时，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

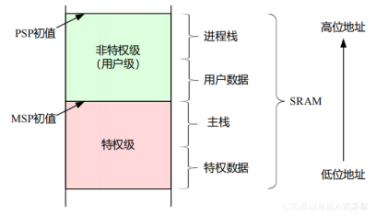
空堆栈(Empty Stack):SP 始终指向下一个将要放人元素的位置，也就是指向堆栈的第一个没有使用的地址或者空位置

。满递增(FA):SP 指向最后压人的数据，且由低地址向高地址生长。满递减(FD):SP 指向最后压人的数据，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、ProcessStack)，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一，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

⑧位带

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 0x200fH(SRAM区中的最低 1MB)

0x4000 00000x400fI(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)

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

⑨异常处理与异常优先级分组

异常状态:mactive、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人口地址(该人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(grouppriority)，也称为抢占优先级(preemptpriority)，另外一类为子优先级(subpriority)

·异常流程:进人异常处理;执行异常处理程序;异常返回

。向量表重定位机制:VTOR(VectorTable Ofset Register)

。中断请求和挂起:- 挂起状态、挂起状态清除，进人/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ARM处理器指令系统

①T32指令的汇编语法

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]lg][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。

I6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 RO,R1;寄存器 R1 中的数值装载到寄存器 RO·MOVS RO, R1;寄存器 R1 中的数值装载到寄存器 RO，更新 APSR。MOVEQ RO,R1 ;Z==1 时把寄存器 R1 中的数值装载到寄存器 RO。ITT EQ;随后的两条指令是条件执行的。MOVEQ RO,R1;“Z=-1”时执行，否则不执行·MOVEQ R2,R3;“Z=-1”时执行，否则不执行

②T32指令集的10种寻址方式

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 RO #66·2)寄存器寻址<Rn>，示例:ADD RO,R1,R2·3)寄存器移位寻址[<Rn>,LSL #imm]，示例:MOV RO, R2,LSL #3·4)寄存器间接寻址[<Rn>]，示例:DR RO,[R1]7·5)寄存器偏移寻址[<Rn>,<ofset>]，示例:LDR RO,[R1, #4]·6)前变址寻址[<Rn>,<ofset>]!，示例:LDR RO,[R1,#4]!(执行后 R1=R1+4)·7)后变址寻址[<Rn>],<ofset>，示例:LDR RO,R1]，#4·8)多寄存器寻址<Rn>!,<registers>，示例:LDMIA RO!,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 RO, [PC,#0xC

③基本指令功能和用法：



第八章基于ARM的硬件与软件系统设计开发

①嵌入式系统的交叉开发环境（了解）

宿主机和目标机是不同的机器。嵌人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。

②嵌入式系统开发过程各阶段（理解）

·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

③微处理器最小硬件系统概念（了解）

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 引脚输人，-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)、NANDFlash(存放程序)

④STM32时钟树的基本概念、功能、作用、意义、特点等

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。常用的信号源有品体振荡器和 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。嵌人式中除了内核，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: 时钟是嵌人式系统的脉搏，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

⑤SPI、I2C接口原理（大致传输过程）

IC 部分:

·I20 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

。I2C 数据传输过程--主设备向从设备发送信息:主设备开始发送信号（S），主设备发送7比特的从设备地址，然后发送写命令（R/W低电平）接着从设备应答（发送ACK）主设备发送8位字节数据，然后从设备应答（发送ACK），若应答不成功（N）则主设备发送停止信号（P）

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)，主设备数据输人，从设备数据输出;- MOSI(Master Output Slave Input)，主设备数据输出，从设备数据输人;- 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。

IC与SPI区别：

- I2℃ 数据输人/输出用同一根线;SPI 输人输出分开

- I2℃ 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 PC;外部通信最好用SPI带隔离(提高抗干扰能力)

第八章△

①GPIO四种模式：IN复位状态输入；OUT通用输出模式；AF复用功能模式；AN模拟输入模式

给定库函数时 GPIO的基本输入输出编程;引脚复用功能(掌握)

、基本配置过程(以 STM32F407为例)

对一个 GPIO 端口进行初始化编程需要三步:

①使能该 GPIO 端口所在时钟，调用函数 void RCC AHB1PeriphClockCmd(uint32 t RCC AHBlPeriph,FunctionalState NewState)，进行时钟使能，参数 RCC AHBlPeriph 可代表挂接在 AHB1 上的外设变量值，如 RCC AHB1Periph GPIOA(端口 GPIOA)，参数NewState 为状态参数，ENABLE例如 GPIOA进行时钟使能，可以使用函数RCC AHBlPeriphClockCmd(RCC AHBIPeriph GPIOA, ENABLE)cPE这里实验中 GPIO 端口都挂接在了 AHB1 总线上，所以使用含 AHB1 的时钟使能函数

②对需要使用的 GPIO 端口进行初始化配置

③调用 void GPI0\_Init(GPIO\_TypeDef\* GPIOx,GPI0\_InitTypeDef\* GPI0 \_InitStruct)函数进行初始化，GPIOx=GPIOA等，GPI0\_InitStruct为GPI0\_InitTypeDef结构体

GPIO 端口初始化:

在实验对 GPIO 端口初始化需要对 GPIO InitTypeDef中五个参数进行赋值，完成初始化，GPIO InitTypeDef如下:

typedef struct

{uint32 tGPlO Pin;

GPIOMode TypeDefGPlO Mode;GPIOSpeed TypeDefGPlO Speed;GPI0OType TypeDefGPI0 OType;GPIOPuPd TypeDefGPlO PuPd;}GPIO InitlypeDef;其中GPIO\_Speed有四种模式：低速2MHz，中速25MHz，快速50MHz，高速100MHz，GPIO\_OType有两种模式：推挽PP，开漏OD，GPIO\_PuPd有三种模式：不使用上下拉NOPULL，UP上拉，DOWN下拉，例子：RCC AHB1PeriphClockCmd(RCC AHBIPeriph GPIOF,ENABLE);//时钟使能GPlO InitTypeDef GPlO InitStructure.GPIO InitStructure.GPlO Pin=GPIO Pin 11://定义一个GPI0 InitypeDef结构体，用于初始化GPI0 InitStructure.GPI0 Mode=GPO Mode OUT:GPI0 InitStructure.GPI0 OType=GPIO OType PP.GPI0 InitStructure.GPl0 Speed=GPl0 Speed 100MHz,GPIO InitStructure.GPIO PuPd=GPIO PuPd UP:GPI0 Init(GPI0G,&GPIO InitStructure);

引脚复用通常有以下几种情况:

GPIO引脚除通用功能外，还可设置为一些片上外设的复用功能;〇 一个IO引脚除可作为某个默认外设的复用引脚外，还可作为其他(多个)不同外设的复用引脚;O 一个片上外设，除了默认的复用引脚，还可有多个备用的复用引脚。0 IO引脚的复用功能重映射O可把某些外设的复用功能从默认引脚转移到备用引脚上。引脚复用的优点〇节约芯片引脚，可分时复用外设，虚拟增加端口数量，优化引脚配置和布线设计PCB，同时减少信号交叉干扰

对GPIO引脚复用进行配置时，需要两步：mode设成复用模式AF，使用 void GPI0\_PinAFConfig(GPI0\_TypeDefGPIOx,uint16 t GPI0\_ PinSource,uint8 t GPIO\_AF)函数进行端口复用

1. 定时器

定时器(基本和通用)的3种计数模式，普通输入捕获、PWM输入捕获、比较输出、PWM输出的基本原理(掌握)给定库函数时，定时器的基本功能编程，包括硬件连线、相关 GPIO口及定时器的初始化配置、精确延时的实现、结合中断的综合应用(掌握)

void TlM3 Int Init(ul6 arr,ul6 psc){

TlM TimeBaselnitTypeDefTlM TimeBaselnitStructure,RCC APBlPeriphClockCmd(RCC APB1Periph TIM3.ENABLE):

TIM TimeBaselnitStructure.TlM Period = arr,

TIM TimeBaselnitStructure.TlM Prescaler-psc,

TIM TimeBaselnitStructure.TlM CounterMode-TlM CounterMode Up,TIM TimeBaselnitStructure.TlM ClockDivision=TIM CKD DIV1;

TlM TimeBaselnit(TlM3,&TIM TimeBaselnitStructure);

TIM ITConfg(TIM3.TIM IT Update.ENABLE):TIM Cmd(TIM3.ENABLE):}

PWM示例

void TIM2 PWM Init(u32 arr, u32 psc){

GPIO InitTypeDefGPlO InitStructure.

TIM TimeBaselnitTypeDefTIM TimeBaseStructure,

TIM OCIitTypeDefTlM OCInitStructure,

RCC APBlPeriphClockCmd(RCC APBIPeriph TIM2,ENABLE);RCC AHBlPeriphClockCmd(RCC AHBlPeriph GPIOB, ENABLE);

GPIO PinAFConfig(GPIOB,GPIO PinSource10,GPIO AF TIM2):GPIO InitStructure.GPIO Pin=GPIO Pin 10:

GPIO InitStructure.GPIO Mode =GPIO Mode AF:

GPI0 InitStructure.GPl0 Speed=GPl0 Speed 100MHz;

GPl0 InitStructure.GPl0 OType = GPIO OType PP

GPIO InitStructure.GPIO PuPd = GPIO PuPd UP:

GPI0 Init(GPIOB, &GPIO InitStructure);

TIM TimeBaseStructure.TlM Prescaler=psc,

TIM TimeBaseStructure.TIM CounterMode TlM CounterMode Up,

TIM TimeBaseStructure.TlM Period=arr,

TIM TimeBaseStructure.TIM ClockDivision=TIM CKD DIV1:

TIM TimeBaselnit(TIM2,&TIM TimeBaseStructure),

TIM OCInitStructure.TlM OCMode-TIM OCMode PWM1:TIM OCInitStructure.TlM OutputState TlM OutputState Enable.TM OCInitStructure.TIM OCPolarity=TlM OCPolarity High :TIM OC3Init(TIM2, &TIM OCInitStructure);

TIM Cmd(TIM2, ENABLE);}

中断控制器

USART