長沙工學院

计算机硕士考研
知识点摘要

九三西域府前進四委員會 編
The Ahead Four Committee of Jiusan Western Region

许可协议

CC BY-NC 4.0

本作品采用知识共享署名-非商业性使用 4.0 国际许可协议进行许可

你可以自由地

共享 - 在任何媒介以任何形式复制、分享本作品

演绎 - 修改、转换或以本作品为基础进行创作

但需遵守以下条例

署名 - 你必须要给出原作者署名及其联系方式，并标明是否对原始作品进行修改

非商业性使用 - 不得将本作品用于商业目的

作者联系方式

原作者
九三西域府（tianwenzy@tianwenzy.com）

素材版权

HarmonyOS Sans：免费商用 华为&汉仪字库

狮尾 B2 加糖宋体：免费商用 Max（个人）

黃令東齊伋體：免費商用 黃令東（个人）

方正仿宋：免费商用 方正字库

方正书宋：免费商用 方正字库

正風毛笔：免費商用 Max（个人）

隸書：免費商用 台灣當局僞教育部

Asana：免费商用 Apostolos Syropoulos（个人）

封面 logo：九三西域府绘制

使用说明

本作以 B5 纸布局排版，可以直接不经编辑将本文件拿去打印店以 B5 打印，推荐用 B5 打孔纸打印，方便携带使用。

本作仅作为一个助记手册，不具备辅导功能，复习过程请以学校指定教材（以下简称绿书、黑书、白书）、中国大学 MOOC 课程（以下简称慕课）及其 PPT、爱课堂课程（以下简称爱课程）及其 PPT、王道/天勤辅导书（以下简称王道天勤）等资料进行主体复习。组成原理（以下简称组原），数据结构（以下简称数构）。黑书特指《计算机组成与设计-硬/软件接口》；需要用到《深入理解计算机系统》的地方简称“CSAPP”；需要用到袁春风、唐朔飞等编著的教材时，直接以名字称呼。

本作组原部分以慕课的知识点内容为基础，补上了爱课程的一些知识点，增添了王道的一些知识点，主要是作为知识点的汇总手册。

组原的复习，第一遍应当以看慕课/爱课程为主，第二遍再看的同时，辅以王道及其相关的习题，同时参考爱课程 PPT 中慕课缺失的内容（如校验码、DRAM 刷新、虚存、IO 通道中断查询等）。第三遍的时候结合真题进行练习，加深知识点印象。本手册可用可不用，仅供参考。

本作数构的部分，以白书知识结构为基础进行整理，增添了王道的一些知识点。

数构的复习，建议以白书为基础复习，同时辅加王道/天勤，KMP 章节的复习务必请使用天勤。复习的时候，也要有尺度的取舍部分内容（如红黑树、键树等）。

也可用作统考 408 复习使用。

能力有限，如有差错争议之处，请指出或群内@我。

个人精力所限，未来不会再更新。

捐赠

如果您认可我的这项工作和付出，觉得值得称赞，可以请我喝一瓶矿泉水（这不是强求）

版本更迭

2020 年 06 月 26 日 Beta 0.0.1 添加许可协议、添加笔记目录、添加组原第一章
2020 年 07 月 05 日 Beta 0.0.2 添加数构第一二章
2020 年 07 月 06 日 Beta 0.0.3 添加数构第三四章
2020 年 07 月 10 日 Beta 0.0.4 封面校徽更替为自主修改简化校徽、添加数构第五章
2020 年 07 月 11 日 Beta 0.0.5 补充数构第五六章、字体更替为商用免费字体
2020 年 07 月 26 日 Beta 0.0.6 增加数构组原大纲分析、增加数构第七八九十章
2020 年 07 月 27 日 Beta 0.0.7 补充数构第九章、补充增加组原第一二章、补充大纲分析
2020 年 08 月 01 日 Beta 0.0.8 补充数构笔记
2020 年 08 月 03 日 Beta 0.0.9 补充组原第二章
2020 年 08 月 04 日 Beta 0.1.0 添加组原第四章
2020 年 08 月 06 日 Beta 0.1.1 补充组原第四章
2020 年 08 月 07 日 Beta 0.1.2 补充组原第四章
2020 年 08 月 09 日 Beta 0.1.3 完成组原第七章
2020 年 08 月 10 日 Beta 0.1.4 完成组原第六章
2020 年 08 月 11 日 Beta 0.1.4 完成组原第五章
2020 年 08 月 31 日 Beta 1.0.0 补充组原第一章
2020 年 09 月 23 日 Beta 1.0.1 增加数构代码汇总
2020 年 09 月 27 日 Beta 2.0.0 排版重构、设计封面
2020 年 09 月 28 日 Beta 2.0.1 排版重构、引入多种商用免费字体、补充组原第五章
2020 年 10 月 03 日 Beta 2.0.2 补充数构第零一章
2020 年 11 月 09 日 Beta 2.0.3 修改补充
2020 年 11 月 10 日 Beta 2.1.0 修改补充
2020 年 11 月 12 日 Beta 2.1.1 修改补充、调整版面布局
2020 年 12 月 28 日 Beta 2.2.0 修改补充
2021 年 01 月 16 日 Beta 2.2.1 修改补充
2021 年 02 月 03 日 Beta 3.0.0 修改补充
2021 年 02 月 27 日 Beta 3.0.1 修改错字，补充几个点，应该是最后一版了
2021 年 06 月 21 日 Beta 3.0.2 引入新字体、修改一些错误、删除一些不必要内容

目录

數據結構

第零章 纲

- 数据结构与算法基本概念：顾名思义，了解基本的概念、性质，解答概念性质所衍生出的问题，一般不会太难。但也要牢牢掌握住，这些知识点也是下面应用部分的基础。
  - 线性表
  - 栈与队列
  - 串
    - 主要就是 KMP，求 nextval 数组（即白书上的 next 数组），以及匹配过程
  - 稀疏矩阵
    - 稀疏矩阵在白书上是没有的，反正我没找到，更别说三角阵对称阵三对角阵了，所以这一部分对应的王道内容看还是不看，还有很大的考量。但本身稀疏矩阵这部分内容是不难的。
    - 2020.12.27 补充，考了三元组的实现方法。
  - 树与二叉树重点的重点，细看就对了。
  - B-树和 B+-树内容不多，王道少量题存在理解冲突，建议白书为主。
  - 图重点的重点，细看就对了。
  - 算法分析
  - 贪心法、动态规划、递归与分治、回溯法、分支界限法
    - 老实说，这些知识点，我觉得考的可能不大，甚至没有什么可考察的，无论是选择还是填空，大题更不会考察了。至少对于今年（2020 年底）来说。但还是感觉有出填空的概率，尽管概率不大。
    - 2020.12.27 补充，依旧没考，保留上条观点。
- 基本数据结构的应用：应用则表现的较为复杂，但凡应用，除了要掌握其代码（如果给出了代码），还要能在无代码的情况下手动模拟，比如 KMP 求 nextval 数组的过程；甚至一些基本简单的应用可以脑中模拟，比如快排的整个流程、AVL 的翻转过程。此部分是主要难点，需要认真扎实地掌握。
  - 栈与队列的应用
  - 内排序算法
  - 线性表的查找
  - 二叉排序树的应用重点！！！
  - 堆排序 2020.12.27 更，今年考了
  - Huffman 算法考了两年了
  - 图的搜索

- 最小代价生成树
- 最短路径
- AOV 网的拓扑排序
- AOE 网的关键路径

- 算法设计与分析看不到未来的出题角度是什么样的
  - 能运用数据结构与算法的知识解决非数值问题的数据抽象
  - 算法设计与分析
  - C（或 C++）语言的实现
- 第二章的递归效率分析；第八章的键树、2-3 树、判定树、红黑树、等价类并查集。不在大纲之中，408 也非重点，私以为不会考察不会出题。数据结构也仅第八章这一章的知识点有争议，其他的照常完整复习即可。
- 填空其实与选择类似，还是简单应用的考察。但不一样的是，填空会考察一些概念性的，如性质、分类、定义等等。

第一章 绪

- 数据元素是数据的基本单位。由若干个数据项构成
  - 数据项分类：初等项，不可分割最小单位，诸如 UID、姓名、年龄等项；组合项，由若干个数据项构成，如成绩（包含了各科成绩）
- 数据结构，数据的组织形式，由数据对象和数据对象中数据元素的关系组成
  - 逻辑结构：数据元素之间的逻辑关系描述数据
    - 线性：每个结点仅一前驱一后继。一般线性表、受限线性表（栈、队列、串）、线性表推广（数组、广义表）
    - 非线性：每个结点有多个前驱后继。集合、树形结构（一般树、二叉树）、图状结构（有向图、无向图）
  - 存储结构：即物理结构，数据在计算机中的表示。顺序存储、链式存储、索引存储、散列存储
- 抽象数据类型部分可不看
- 算法五性：有穷、确定、可行、输入、输出
- 好算法：正确、可读、健壮、时空效能
- 渐进时间复杂度：一对 N 对幂指阶

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n) < O(n^k) < O(n!) < O(n^n)$$

- 算法原地工作指空间复杂度 O(1)
- 内外层无关相乘，内外层有关累乘（王道两层相关 for 的题）
- 掌握计算时间复杂度的能力。会分析空间复杂度。要做题。
  - 找出执行主体，设执行次数为 T，根据执行的规律可得到最后一次执行结果为 T 的一个式子，由题可知执行结果的受限值 n（或 n 相关数值），然后反解执行次数 T，可得到关于 n 的时间复杂度。
    - 统考 2011 选择和 2014 选择和 2017 选择。仔细品读。
- 一般此类题常与排序查找结合，属于记忆性考察。也会与算法题结合（参考 408），考察对于复杂度的分析能力。也会给选择题出一段代码来算复杂度，大多是与 log 相关的复杂度。
- 本章会在结构分类上考察，每年有个填空或者选择，别疏忽，以免考场上感觉很熟悉但想不起来。

第二章 表

- 稀疏矩阵：矩阵中零元素远远多于非零元素
  - 用三元组表示，非零元素、行、列构成
- 线性表：有限个数据元素构成的有限序列。有且仅有一个前驱和一个后继
  - （注意与王道上的定义的区别，以白书为主）
- 向量：数据元素类型相同的线性表。（后文中所说线性表皆指向量）
  - 王道上所讲的线性表，即是白书中所讲的向量
- 栈：后进先出线性表
  - 栈顶指针 top 取值为 0~Maxsize-1，为-1 时栈空，为 0 时仍有栈底元素，相当于数组中 a[0]，是非空的
- 前缀、中缀、后缀表达式的互相转化：建议手动模拟过程几遍体会思想。 关于括号（），参与进栈出栈扫描，但最后不会输出。理解思想后，再去看严谨步骤
- 后缀表达式求值：
  - 1. 扫描表达式，若为数，进栈，否则
  - 2. 连续出栈两个元素以当前操作符进行运算，然后结果进栈
  - 3. 重复 1. ，直到结束
- 中缀表达式求值：数栈和符栈
  - 1. 符栈空，扫描到操作符直接进栈
  - 2. 栈顶为*或/时，扫描到（进栈，否则出栈并执行 6. ，然后再进栈该符
  - 3. 栈顶为+或-时，扫描到（*/进栈，否则出栈并执行 6. ，然后再进栈该符
  - 4. 栈顶为（时，扫描到除）之外的符都可进栈
  - 5. 若扫描到），连续出栈，直到出栈符为（ “）并未进栈过”
  - 6. 除（外，每当一个符出栈，都要连续数栈出栈两次，以该符操作运算，然后结构进栈，（出栈时不做操作
  - 7. 表达式扫描完成后，若符栈不空，则一一出栈并执行 6. ，符栈为空时，数栈栈顶值即为表达式值
- 递归应用：
  - 递归不能用循环来表达，需满足两个条件：递归体、递归出口（终止条件）
- 递归算法求解一般比其非递归算法要效率低
- 递归效率分析：私以为不是考察重点，首先递归就不是考察重点，故不看

- 汉诺塔递归时间复杂度$O(2^n)$，n 圆盘汉诺塔移动次数为2^n-1
- n 个不同元素进栈，出栈序列的个数为卡特兰树，$C_{2n}^n/(n+1)$。求个数，代公式；求某开头序列，穷举
- 队列：先进先出线性表。尾进头出
- front 指向队头元素，rear 指向队尾元素的下一个位置（也即下一个元素插入的位置）
- 循环队列：为了克服假溢出时移动大量元素
  - 初始 front = rear = 0
  - 尾进 rear = (rear+1) % maxsize
  - 头出 front = (front+1) % maxsize
  - 队长 (rear+maxsize-front) % maxsize
  - 判满 (rear+1) % maxsize == front
  - 判空 front==rear
- 栈的先进后出和队列的先进先出，先后都是相对先后，不一定非要紧接着先后
- 顺序存储结构插入前必须先判满
- 队列应用：
  - 层次遍历二叉树：
    - 1. 根结点入队
    - 2. 若队空，结束，否则 3. 重复
    - 3. 队中第一结点出队，若有左子，入队；若有右子，入队。回到 2.
- 中缀转后缀、后缀表达式算值要掌握
- 循环队列判满空、判指针相关的考察
- 进出栈的考察，求数量、判序列

第三章 链

- 每个结点两个域，数据域（存放数据）、指针域（指向下一个结点的地址）
- 头指针：指向表头（第一个结点），用来标识一个单链表。可以这样理解，头指针本身就是第一个结点自己
- 头结点：表头之前附加一个结点，其数据域不设任何信息或记录表长，其指针域指向表头
  - 链表第一个位置的操作与其他结点无异，无需特殊处理
  - 无论链表空否，头指针都指向头结点，处理能够统一
- 结点的描述要清楚（以白书为标准）
- 查找插入删除结点的代码无需记忆，体会其思想
- 删除结点或删除表时要注意释放操作
- 单链表的建立
  - 头插法：创建新结点，新结点数据域拿到值，新结点指向头结点所指，头结点指向新结点，如此反复
  - 尾插法：需要一个表尾指针，创建新结点，数据域拿到值，尾结点指向新结点，表尾指针指向新结点，如此反复
- 链栈链队与单链表的构成无大区别，在其操作上会有各自衍生的特殊指针，特殊插入删除方法等，此部分建议看王道理解
- 循环链表：最后一个结点的指针指向头结点形成闭环，相关的考察以判满空为主，看王道体会思想
- 双链表：含两指针，指向前驱和后继
- 双链表的插入删除结点操作的思想可以总结为一句话，“索引权在移交不得断开”，具体代码无需记忆，要领会思想
  - 在删除操作时，不管删前一个还是后一个，都是同样的方法，最后看情况替代数据域
- 循环双链表判空：
  - 有头结点：L->next == L && L->prior == L
  - 无头结点：不会出现这种情况，
  - 带头双链表仅一结点：L->next->next == L && L->prior->prior == L && L->mext != L

- 带头单链表仅一结点：L->next->next == L && L->next != L
- 关于链类的问题，主要涉及在判空满、插入删除结点、指针指向的考察
- 本章不应当过分的看重代码，尤其是白书上几乎正篇幅都是代码，应当通过王道理解这些当中的思想

第四章 串

- 串是零个或多个字符的有限序列，线性表
- 存储、运算不算是考察重点，也无考察意义，主要还是模式匹配
- 模式匹配：重头戏，在待查串中查找子串出现位置并给出位置。P 为给定子串，称模式；T 为待查串，称目标。故为模式匹配
- 朴素模式匹配：子串 P 依次与主串 T 中的字符做比较
  - 设 $T = t_0t_1t_2t_3t_4 \cdots\cdots t_{n-1}$，$P = p_0p_1p_2p_3p_4 \cdots\cdots p_{m-1}$，$m \leq n$
  - 从 T 最左端开始比较，若对于所有 k=0、1、2 ・・・・ m-1 均有$t_k = p_k$，则成功
  - 否则 P 右移一个字符重新比较
  - 最坏情况，每趟比较都在最后出现不等，每趟最多比 m 次，最多比较n-m+1趟，总比较次数为$m \times (n - m + 1)$，其时间复杂度为$O(m \times n)$
  - 因为存在回溯所有影响匹配效率
- KMP：
  - **next 数组求解**，王道参考答案中（2019 版王道 P275 表格下方开始）的简便方法。如果遇到的选项要求以-1 开头，则整体-1 即可
  - **nextval 数组求解**，天勤方法，**也即白书方法**。在 next 数组基础上进行求解
    - 当 j=1 时，nextval[j]赋值为 0，作特殊标记
    - 当p_j不等于p_k时(k等于next[j])，nextval[j]赋值为 k
    - 当p_j等于p_k时(k等于next[j])，nextval[j]赋值为nextval[k]
  - 语言叙述反而不清，按照方法走一遍，只可意会不可言传。
  - 失配时，i 不变，j 回退到next[j]的位置并重新比较。这也是考点（21 年王道那道匹配过程的解答题值得学习）
- BM 匹配算法盲猜不考察
- 本章没多少可考察的地方，重点还是在 KMP 里的 nextval 数组求解，甚至 KMP 本身过程也没什么考察余地，手写代码就是默写，其他花样又考不出。掌握好 nextval 数组求解，主要掌握好王道的简便方法，KMP 本身原理甚至可以不追究，应试主义，但应该了解 KMP 运行时的回退过程，有可能填空选择

第五章 排

与王道 408 不同，白书将排序查找紧跟在了表结构之后，之所以如此是因为基本的排序查找操作都是建立在了表结构之上，故如此编排章节有利于数据结构+算法的连贯学习。而对于 408 王道来说，其理念是前期一律学表树图的数据结构，之后才统一的学习两类算法

- 本章默认排序为非递减排序
- 排序亦称分类，将结点称为记录，将一系列结点构成的表称为文件
- 排序码，结点中一个或多个字段，其值是作为排序运算中的依据。排序码的数据类型可以是整数也可以是实数、字符串
  - 关键码：按关键码排序得到唯一的排序结果
  - 非关键码：多结点排序码值可能相同，按其排序所得排序结果不唯一
- 按策略分类：插入、选择、交换、分配、归并
- 稳定性：待排表中，两值R_i和R_j相同，排序前 i 左 j 右，若排序后顺序不变，则稳定
- 时间复杂度由移动次数和比较次数决定
- 插入排序：每次选择待排表第一记录值，按大小插入已排表中合适位置，如此反复
  - 直插：边比较边移动，最终到合适位置
    - 稳定
    - 最好的情况，全部有序，仅一个在末端，比较$n-1$次；最坏按非递增排序，比较$n(n-1)/2$次。总比较次数和移动次数均为$n(n-1)/4$，时间复杂度$O(n^2)$。空间复杂度$O(1)$
  - 折插：已排表中折半比较出合适位置，再统一移动
    - 稳定
    - 仅减少了比较次数，比较次数与元素个数 n 有关，不依赖于表初始状态，约为$O(n\log_2 n)$。移动次数不变，取决于表初始状态。时间复杂度仍为$O(n^2)$。空间复杂度$O(1)$
  - 希尔：通过增量划分为若干子表，子表直插；减小增量，再分子表，直至有序
    - 增量的理解不能混淆：
      增量划分 $[i, i+d, i+2d, i+3d, \ldots\ldots, i+nd]$
      真实例子$[1,4,7][2,5,8][3,6,9]$
      ~~虚假例子[1,2,3][4,5,6][7,8,9]~~
    - $d_i = n/2$ ，$d_{i+1} = \lfloor d_i/2 \rfloor$

- 不稳定，平均比较次数和平均移动次数都为$O(n^{1.3})$。空间复杂度$O(1)$。一般希尔比直插快

- 选择排序：每次从待排表中选出最小的值进行插入，如此反复，直至有序
  - 直选排（简单选择）：在待排表中选出最小的值插入前方已排表的末尾
    - 移动次数最少 0，最多不超过$3(n-1)$。比较次数与序列初始状态无关，始终为$n(n-1)/2$。时间复杂度为$O(n^2)$。空间效率$O(1)$
  - 树型选择：胜者树、竞赛树排序。**不要理所应当的认为是王道后的堆排序！！！**
    - 把 n 个排序码两两比较，取出小的那一个，得到$\lceil n/2 \rceil$个较小的排序码，然后继续两两比较，如此反复，直到得出最小的值。将最小值以∞代替，继续进行新的筛选
    - 时间复杂度$O(n\log_2 n)$。空间开销较大
- 交换排序：每次将待排表中两个记录进行比较，如果不满足排序要求就交换顺序，直到文件中任意两个记录都满足排序要求
  - 起泡：从前往后（或从后往前），两两比较，若不满足排序则交换
    - 每一趟都把最小的放在最终位置
    - 产生的已排表（有序子列）是全局有序，且小于待排表（无序子列）中所有记录值
    - 稳定。
    - 比较次数$n(n-1)/2$，移动次数$3n(n-1)/2$，最好情况时间复杂度$O(n)$，最坏$O(n^2)$。空间复杂度$O(1)$
  - 快排：基于分治，任选一个记录作为中心值，将剩余记录值分为两部分，小于中心值和大于中心值的，并分别放置两侧。然后将两部分重复上述操作，直到每个部分为空或只有一个记录值
    - 每趟排序后，至少有一个值在最终位置
    - 不稳定。是所有内部排序中平均性能最优的算法
    - 从后往前找小，从前往后找大，交替进行
    - 一开始以第一个记录值作为中心值
    - 逆序时最慢；每次分为两个等长表时最快
    - 借助递归栈实现，空间复杂度为$O(\log_2 n)$。最好时比较次数约为$n\log_2 n$，最坏时比较次数$n(n-1)/2$，平均时间复杂度$O(n\log_2 n)$,平均需要$O(\log_2 n)$趟快排过程
- 分配排序：没什么可说的，看白书 p113 的文字，只可意会不可言传
  - 基数排序：没什么可说的，只可意会不可言传

    - 稳定
- 归并排序：待排序文件是部分有序的，将两个或两个以上的有序序列归并为一个新的有序序列。假定待排表有 n 个记录，则可看作 n 个有序子列，每个子列长为 1。然后进行两两归并，得到$\lceil n/2 \rceil$个长度为 2 或 1 的有序子列。如此反复，直到合并为一个长度为 n 的有序表
    - 以上称为“二路归并排序”，同理还有“三路归并排序”“多路归并排序”
    - 二路归并的性能：占用 n 个辅助空间，空间复杂度$O(n)$。每一趟归并时间复杂度为$O(n)$，需要进行$\lceil \log_2 n \rceil$趟，时间复杂度为$O(n \log_2 n)$
- 外部排序：待排序文件很大，无法一次性放入内存中，需要部分部分的将记录调入内存排序。主要依靠“内外存交换”和“内部归并”两者结合进行实现
    - 外部排序这一小章节，应当以最佳合并为主要复习点，其余两个理解过程即可。本身外部排序未出现在大纲中，个人感觉可考察的点不多
    - 二路合并：与内部排序的二路归并无本质区别。熟悉白书上的例子即可
    - 多路替代选择合并：败者树
    - 最佳合并：n 路最佳合并就是将初始顺串以 n 叉哈夫曼树的形式进行归并
        - 求 WPL 带权路径长，每个结点乘以到根结点的长度的总和
        - 读写外存块的次数$2 \times \text{WPL}$
- 内部排序归纳
    - 稳定的：直接插入、折半插入、冒泡、归并、基数
    - 空间复杂度
        - $O(1)$：直接插入、折半插入、冒泡、简单选择、希尔
        - $O(\log_2 n)$：快排
        - $O(n)$：二路归并
    - 时间复杂度
        - $O(n^2)$：直接插入、折半插入、冒泡、简单选择
        - $O(n \log_2 n)$：快排、树型、二路归并、堆排
        - 希尔一般是$O(n^{1.3})$

第六章 找

- 平均查找长度$E = \sum P_i C_i$，P_i查找到第 i 个元素的概率，C_i找到第 i 个元素所进行的比较次数
- 顺序查找：将每个结点值与待查值进行比较，直到找到或遍历结束整个表。顺序、链式存储皆可，适应性较好
  - 一般线性表 n 个元素，顺序查找的平均长度$E(n) = P_i(n - i + 1)$，当每个元素查找等概时，$E(n) = (n + 1)/2$。查找失败时$E(n) = n + 1$
  - 顺序查找前，先将待查值排序再查找，会降低平均查找长度
- 折半查找：即二分查找。仅适用于有序顺序表。查找时，先找到表的中间结点进行比较，若相等则查找成功。若小于则在前半部分折半查找，若大于则在后半部分折半查找，直到找到或没有找到
  - 最好的情况是第一次查找就找到，$O(1)$。最差的情况是表中没找到，大约进行了$\log_2 n$次比较，复杂度$O(\log_2 n)$。等概查找时，$E(n) \approx \log_2(n + 1) - 1$
  - 折半查找过程可用二叉判定树描述，故比较次数与结点所在层数有关，不会超过树的高度
  - 时间复杂度$O(\log_2 n)$
  - 折半查找需要方便定位，显然适合具有随机存取特性的数据结构，即顺序表
- 分块查找：索引顺序查找。将查找表分块为若干个子块，块内元素可有序可无序，但块间有序，即前一块中最大元素小于后一块中最小元素。建立索引表，索引表中每个元素含有各块的最大关键字和各块中第一个元素的地址。索引表按关键字有序排列
  - 查找步骤：1. 在索引表中确定待查记录所在的块。2.在块内查找
  - 若设长度为 n 的查找表均匀分为 b 块，每块有 s 个记录。则：
    - 平均查找长度$E = E_a + E_b$，E_a是索引查找长度，E_b是块内查找长度
    - 若块内索引都顺序查找$E = E_a + E_b = (b + 1)/2 + (s + 1)/2 = (s^2 + 2s + n)/2s$，若$s = \sqrt{n}$，则平均查找长度取最小值$\sqrt{n} + 1$。若索引表折半查找，则平均查找长度$E = E_a + E_b = \lceil \log_2(b + 1) \rceil + (s + 1)/2$
    - 空间复杂性上，增加了一个辅助数组
- 散列查找
  - 散列函数/哈希（Hash）函数：将分散的值映射到一个较小的区间，再利用映射后的函数值作为访问结点的下标。（王道人话：将关键字值映射为对应地址）

- 负载因子：定义哈希表的装满程度 $\alpha = \frac{\text{散列表中的结点数目}}{\text{散列表的长度}}$
- 散列表的查找效率取决于：散列函数、处理冲突的方法、负载因子
- 多个值对应一个地址，即为碰撞，也称冲突
- 散列函数的构造
  - 除留余数法：选择一个适当的正整数 p，用 p 除去关键码值，所得余数即是对应的散列值（地址）。一般选择不大于长度 n 的最大素数。Hash(key) = key%p
  - 数字分析法：关键码的位数很多时，通过对关键码的各位进行分析，丢掉分布不均匀的位，留下分布均匀的位作散列值。看书上例子。仅适合于静态值集合
  - 平方取中法：先计算关键码值的平方值，然后从平方值的中间位置选取连续若干位用作散列值
  - 随机乘数法：使用一个随机实数f，乘积$f \times key$的小数部分与长度 n 相乘，乘积整数部分就是散列值
  - 折叠法：关键码值的位数远多于散列表长度时采用。将关键码值分为若干段，至少一段的长度要等于散列表长度 n。将在这些段相加，舍弃进位，得散列值。具体方法看书上例子
  - 基数转换法：将关键码值看成是在另一个基数数值上的数，然后把它转换为原来基数上的数，再选其中若干位作为散列值。一般取大于原来基数的数作转换的基数，两基数应该互素。具体看书上例子
- 冲突处理方法：开放地址法、链表地址法
- 堆积：散列值对应表项被非同义的结点的探测所占用。散列函数选择不当，负载因子过大都会加剧堆积现象
- 开放地址法：散列时，先判断指定表项是否被占用，如果被占用，则依据一定的规则在表中寻找其他空闲表项
  - 线性探测法：当冲突发生，就顺序探测下一个表项是否为空闲。若散列值为 d，但第 d 项已经被占用，探测序列为 d、d+1、・・・、n-1、0、1、・・・、d-1
    - 一般散列表的长度大于实际的数据项，故沿此序列，总能找到空虚
  - 双散列法：使用两个散列函数。可有效减少堆积现象
    - 设Hash1(key) = d时冲突，则计算k = Hash2(key)
    - Hash1 以关键码值为自变量，产生 0~n-1 个散列值
    - Hash2 也以关键码值为自变量，产生 0~n-1 个散列值
      - 当 n 为素数，散列值为 0~n-1 任意一个数

- ◆ 当 n 是 2 的幂次数，散列值是 0~n-1 中任意一个奇数
- □ 探测序列为：$(d+k)\%n$ ，$(d+2k)\%n$ ，$(d+3k)\%n$ ，……

- ○ 开放地址法不能随便删除散列表中的表项，否则会使同义词序列断开，影响其他表项的查找
- ○ 链表地址法：为散列表的每个表项建立一个单链表，用于链接同义词子表，每个表项增加一个指针域
  - ■ 独立链表地址法：在散列表基本存储区域外开辟新区域存储同义词子表
  - ■ 公共链表地址法：将同义词子表存储在散列表的基本存储区域里。估计不考
- ○ 查找长度 ASL 的计算：
  - ■ ASL 成功：查找次数/元素个数。为了能将元素插入，操作了几次？
  - ■ ASL 失败：查找失败次数/Hash 后地址个数。按序看各地址多少次后查不到
  - ■ 线性探测法： 王道的例题很具有典型，认真品味
  - ■ 独立链表地址法： 王道的例题很具有典型，认真品味

- ● 根据装填因子可知表长，由表长间接知道除留余数的那个值，化散列表的时候，尽量把冲突的次数标明

第七章 树

- 术语
  - 度数：一个结点的子树的个数
    - 树的度：树中结点的最大度数
  - 层数：定义根结点为第一层，其他结点层数等于父节点层数+1
  - 高度/深度：树中结点的最大层数，空树高为 0
    - 深度自顶朝下累
    - 高度自底朝上累
  - 有序树：结点的子树从左到右是有序的，不能随意互换位置
  - 路径长度：两个结点之间路径上所经过的边数
- 树的性质
  - 结点 = 结点度 + 1
  - 度为 m、层为 i 的树，至多m^{i-1}个结点
  - 高为 h 的 m 叉树，至多$(m^h-1)/(m-1)$
  - N 结点 m 叉树最小高为$\lceil \log_m(n(m-1)+1) \rceil$
  - $1+N_1+2N_2+3N_3+\cdots\cdots = N_0+N_1+N_2+\cdots\cdots$，$N_x$表示度为 x 的结点个数
- 二叉树：有序，五种形态：空、单结点、左子树数空、右子树空、子树不空
- 二叉树性质
  - 任意 n 结点二叉树恰有 n-1 条边
  - 非空二叉树 k 层至多2^{k-1}个结点
  - 高为 h 二叉树至多2^h-1个结点
  - 完全二叉树中 i 号结点所在层次为$\lfloor \log_2 i \rfloor + 1$
  - 完全二叉树 n 结点高$\lceil \log_2 n+1 \rceil$
  - 完全二叉树中，父子结点编号的关系，自行了解不做表述
- 二叉树存储方式
  - 顺序存储：完全二叉树存储，从左至右按序存储在一维数组中，结点与结点关系由数组下标判断；非完全二叉树仍按上法。此法仅适合于二叉树结点与其对应的完全二叉树结点差不多的时候
  - 链接存储
    - LeftChild-RightChild 表示法：结点三域，指左子、数据、指右子
    - 三重链表示法：结点四域，指父、数据、指左子、指右子
- 无论是转换还是遍历，手动操作一遍，写写序列或树，才会有更深的理解领会。

- 树林与二叉树的互相转换：只可意会不可言传，自行领悟
- 森林 n 结点，叶结点 m，对应二叉树中无右子的结点为(n-m+1)。树转二叉树时，非终端结点的最右子右指针空；树林中各树根从左到右互为父子，最右树根结点右指针空（仅一棵树也为树林）
- 树林遍历
  - 按宽度遍历：依次访问层数为 1 的结点，然后层数为 2 的结点，以此类推，直至遍历完
  - 按深度遍历，领会 p147 的图例
    - 先根遍历：访问头一棵树的根，先根次序下遍历头一棵树树根的子树，再遍历其他树
    - 后根遍历：后根此序下遍历头一棵树的子树，访问头一棵树的根，遍历其他树
- 二叉树遍历，自行手动操作一遍写序列，以加深领会
  - 前序法：根左右
  - 后序法：左右根
  - 对称序：左根右
- 遍历算法（之后补代码）
  - 非递归
  - 线索化
- 如果是按照白书提纲来说，本章主要是基本概念，也主要是选择题，序列、结点数、结点相关的选择。
- 选择题善于使用代值法、取特殊情况

- 二叉排序树严格的左小右大。
- 二叉排序树的插入直接插。
- 二叉排序树的删除
  - 若删除的结点无左子，右子替之
  - 若删除的结点有左子，左子替之，原右子成为左子树中序最后结点右子
- 扩充二叉树：给二叉树结点空出子树的地方补上一个特殊的结点—空树叶，原二叉树结点为内部结点，空树叶为外部结点。
  - 根结点到外部结点的路径长度和 E，到内部结点的路径长度之和 I，$E = I + 2n$。2020.12.27 今年考了
  - 对于二叉排序树的查找，成功时，就是关键码所在的层数，失败时，就是所属外部结点的层数-1

- 在等概情况（内外部结点等概）下，查找的平均比较次数$E(n) = (2I + 3n)/(2n + I)$
- E(n)最小的二叉树即为最佳二叉排序树，这种树的特点：只有下面 2 层结点的度小于 2，其他结点度必然为 2
  - 构造过程：先将关键码集合进行排序，然后按照折半查找依次查找关键码结点，并插入二叉排序树中
- 平衡二叉树，不要求有序。高度都是$\log_2 n$，查找插入删除所需时间至多$O(\log_2 n)$
- AVL 旋转，实操理解，此口诀有待优化
  - 从最下面的那个不平衡点开始处理
  - LL 左子左孙，父夺祖位，右孙归祖
  - RR 右子右孙，父夺祖位，左孙归祖
  - LR 左子右孙，右孙夺祖，左曾归子，右曾归祖
  - RL 右子左孙，左孙夺祖，右曾归子，左曾归祖
- B 树，平衡多路检索树，m 阶 B 树或为空，或满足以下特性：（叶子仅指叶子）
  - 树中每个结点至多 m 棵子树
  - 若根结点非叶，至少 2 棵子树
  - 除根外所有非叶结点至少$\lceil m/2 \rceil$棵子树
  - 所有叶结点在同一层且不含信息
  - 非叶结点包含以下信息$(n, A_0, K_1, A_1, K_2, A_2, \cdots, K_n, A_n)$，$K_i$为关键字，$A_i$为指向子树根结点的指针，n 为结点中关键字个数
- B 树结点 n 高度 h，$\log_{\lceil m/2 \rceil}[(n+1)/2+1] \geq h \geq \log_m(n+1)$，最低为满 m 叉
- B 树插入：插入后关键字小于 m 则直接插，大于 m-1 则中上移，左右下随，若中上又大于，按规则再上
- B 树删除
  - 叶结点：删前关键字$> \lceil m/2 \rceil - 1$，直接删；删前关键字$= \lceil m/2 \rceil - 1$，左右旋，若兄弟也$= \lceil m/2 \rceil - 1$，双亲下合兄弟
  - 非叶结点：该元素左子树下最大或右子树下最小的元素，与之交换，重复多次，直至到叶
- B+树
  - n 棵子树含 n 个关键字
  - 所有叶结点包含全部关键字信息，以及指向这些关键字记录的指针，且按自小到大的顺序链接
  - 所有非终端结点看作索引部分，结点仅含子树的最大或最小关键字
  - B+有两个头指针，一个指向根结点，另一个指向关键码最小的叶结点

- 哈夫曼最优树：带权路径长度 WPL 最小的二叉树称为最优二叉树或哈夫曼树。哈夫曼树不局限于二叉，可以是 m 叉。哈夫曼树的构造自行理解，无需多言
- 树编码（哈夫曼编码）：将待编码字符对应哈夫曼树的一个外部结点，对分支进行标记，最左分支为 0，向右分支依次+1。从根到外部结点路径上的序列标记即为该字符对应编码
- 堆排序：父比子小，小根堆；父比子大，大根堆。**本书主要研究大根堆。**堆的实质是一棵完全二叉树的层次遍历
  - 根结点是堆中元素最大的值
  - 堆对应的完全二叉树的任何子树都具有对应性质（子堆同性质）
- 堆排序的过程：建堆、反复删除最大元素 2020.12.27 今年考了排序
  - 从$\lfloor n/2 \rfloor$结点的子树开始筛选，依次再向前对$\lfloor n/2 \rfloor$~1的结点的子树调整。若下一级堆被破坏，该根向下交换调整。插入操作则新结点放在末端，然后进行调整。所谓调整即使堆满足性质。删除一值，就用最末值补上，除非本身就是最末
  - 关于比较次数：如果未改变父子仨，则这部分不做调整，默认不比较。只有其中一个子结点发生变化时，才需要调整，且只和父结点做比较，因为未变子结点和父结点满足堆性质。若父结点改变，则和二子都比较（这种情况，原则上说只比较两次，父结点和任一子结点比较，取之间最大值再与另一子结点比较，最终最大值称为新的父结点）。双子验父，父验单子
  - 每趟都能确定一个元素的最终位置
  - 初始建堆时间$O(n)$，删堆时间不超过$O(\log_2 n)$，总时间开销最坏不过$O(n\log_2 n)$
- 判定树：判定树的结点不存储元素，仅表示一次比较（或比较对象），如果每次都产生二分支，所得到的是二叉判定树，若产生三分支，则是三元判定树（没啥可考的）
- 本章的键树、2-3 树、判定树、红黑树、等价类并查集，不在大纲之中，408 也非重点，私以为不会考察不会出题

第八章 图

- 有向图出尾入头，左尾右头
- 简单图，任意两点之间至多一条边，且不含自身回路（点 V 到 V 自身的边）
- 完全图，任意两点之间都有边。无向完全 n(n-1)/2 边，有向完全 n(n-1)边
- 连通（无向图）
  - 某点到某点有路径，连通
  - 任意两点之间有路径的图，连通图
  - 极大连通子图=连通分量=自己（若非连通图，每个连通部分自己）
  - 极小连通子图，连通但边最少。极大则要求该子图包含所有边
- 强连通（有向图）
  - 两点间双向有路径，强连通
  - 任意一对点间双向有路径，强连通图
  - 强连通分量=强极大连通子图=自己
- 生成树：含所有点的极小连通子图。图若 n 点，树 n-1 边。连通分量的生成树构成连通森林
- n 点 e 边无向图，度 2e；n 点 e 边有向图，入度=出度=e
- 带权图即网，本章只讨论权非负实数，权亦称耗费、路径长度
- 顶点 p 到 q 之间的一条路径是指序列 p・・・q，顶点不重复出现的路径为简单路径。边的数目为路径长
- 若要保证 n 个点任意情况下连通，则需使用 n-1 个点完全，再用一边添加一点，无论点怎样添加，都能连通，可得最少边数
- 子图的前提是先有图，再有点边集合包含。先包含可能会使某些点对应的边未能包进去
- n 个顶点 e 条边的无向图是一个森林，则森林中必有 n-e 棵树。将这些树连到一个共同的新结点上，就得到一棵生成树，则添加了 x 个边，即有 x 棵树；顶点则为 n+1，总边为 e+x，由生成树的性质：$(n+1)-1=e+x$，$x=n-e$
- 图的存储，书上两个邻接多重表的图例不直观，参考王道的图例
  - 相邻矩阵
    - 有权图中顶点值代表对应权值，0、∞表示无边
    - 无权图中 0 表示边不存在，1 表示存在
    - 一维数组存点，二维数组存边
    - 无向图对称且唯一，可压缩存储
    - 无向图第 i 行/列的非空元素个数为该点的度

        - 有向图行出列入
        - A^n的元素$A^n[i][j]$是顶点 i 到 j 的长度为 n 的路径数
        - 读入、存储空间初始化需要$O(n^2)$时间复杂度
        - 深广度遍历时时间复杂度$O(n^2)$
    - 邻接表
        - 由顶点表和边表构成，顶点表结构为顺序存储一维数组，数组中第 i 个元素为指向与顶点V_i相关联的第一条边的指针，边表结点结构为[no , next]，no 为与这条边相关联的一个顶点的序号，next 为指向下一条相关联边的指针。对于有向图只保存顶点表和出边表，或顶点表与入边表
        - n 点 m 边，无向图需要 n+2m 存储单元，有向图需要 n+m 存储单元
        - 表示权图，只需边表结点加一个权值字段，表示顶点到此点的权值
        - 深广度遍历时时间复杂度$O(n+e)$空间复杂度$O(n)$
        - 白书 p203 的图领会
    - 无向图邻接多重表
        - 顶点表和边表两部分组成。顶点表结点结构为[data , edge]，顺序存储，data 表示顶点相关信息，edge 指向与该顶点相关联的第一条边。边表中的结点由下面五个域构成[mark , i , ilink , j , jlink]，mark 代表是否被搜索过，ilink 指向依附 i 结点的下一条边，jlink 指向依附 j 结点的下一条边，该结点表示 i、j 两点夹的边
    - 有向图邻接多重表（十字链表）
        - 既保存有向图的入边表，又保存有向图的出边表
        - 顶点表和边表两部分组成。顶点表结构为[data , edge1 , edge2]，顺序存储，data 表示顶点相关信息，edge1 指向以该顶点为始点的边表中的第一条边，edge2 指向以该顶点为终点的边表中的第一条边（出头始入尾终）。边表中的结点由下面五个域构成[mark , i , ilink , j , jlink]，mark 代表是否被搜索过，i 指向头/始点，j 指向尾/终点，ilink 指向以 i 为始点的下一条边，jlink 指向以 j 为终点的下一条边
- 图的遍历，遍历思想只可意会不可言传，代码后期补上
    - 深度优先遍历
    - 广度优先遍历
- 最小代价生成树：生成树各边的权值总和称为权/总耗费/总代价，权最小的生成树
    - Prim 算法：选取一点，找其最小权边，得之一点，再寻其边，如此反复。时间复杂度为$O(n^2)$，与网的边数无关，适合于边稠密的网

- Kruskal 算法：全局选边，权最小，直至连通。时间开销与边数有关，若有 n 边时间复杂度为$O(n\log_2 n)$，适合边稀疏的网
- 单源最短路径：某点到其他点的最短路径
  - Dijkstra 算法：贪心策略，不适用于负权值边。选取一点，去往他点，留低权路，以此为基，再寻他点，若有最低，及时更新，循环往复
  - 这个算法，主要看王道，直观
- 多点最短路径：所有顶点之间最短路径
  - Dijkstra 算法也可以求多点最短路径，只需要每个点依次做单源即可
  - Floyed 算法：写邻接矩阵，各点依次做中转，未作中转则不可通过该点，欸此更新矩阵中最短路径的权和。最终矩阵即各点间最短路径
- DAG 图，有向无环图
- AOV 图，用 DAG 图表示一个工程，顶点表示活动，有向边表示活动先后进行关系
- 拓扑排序
  - 每个顶点出现且只一次；若 A 再序列中排在 B 前面，则不存在$B \to A$的路径
  - 算法：选一无前驱点输出，删除该点和与其相连边，在剩下的图里再找无前驱点，循环往复
  - 各点地位平等，若顶点有多个后继则序列不唯一
- AOE 网，带权有向图中，顶点表示事件，边表示活动且权值表示活动完成的开销
  - 顶点活动完成后，各边活动才能开始
  - 进入某点的所有活动都完成，该点事件才能开始发生
  - 网中仅一点入度为 0 表源点开始，仅一点出度为 0 表汇点结束
- 关键路径（一定要认真领会王道教授的方法、思路、例题，再来理解口诀）：源点到汇点所有路径中，具有最大路径长度的称为关键路径，其上活动为关键活动
  - 作选择的时候不要傻傻的套算法，手动穷举全局最大路径选择即可
  - 关键路径有多条的网，要加快公共关键活动才能加快整个网
  - 加快关键活动缩短工期，但不能任意缩短活动，可能会使其不再为关键活动
  - 算法(不要把符号含义和王道上的搞混，以白书为主)，列表
    - e(i)：事件最早发生时间　　去某点最长路径
    - l(i)：事件最迟发生时间　　从后往前，减去某点最长路径
    - ae(i)：活动最早发生时间　　该活动前面的最长路径，即 e(i)
    - al(i)：活动最迟发生时间　　从后往前，l(i)减去某活动最长路径
    - $d = ae - al$，为 0 的即为关键活动

第九章 法

- 未来有可能以简单题的形式考察，也可能填空题
- 递归与分治：对于一个规模为 n 的问题，分解为 k 个规模较小的子问题，这些子问题与原问题形式相同，递归地解决这些子问题。将各子问题的解合并得到原问题的解，这种算法设计策略称为分治法。分治是策略思想，递归是技术方法
- 回溯法：每次扩大当前部分解时，都面临一个可选的状态集合，新的部分解就通过再该集合中进行选择构造而成。这样的状态集合结构上是一棵多叉树，每个树结点代表一个可能的部分解，它的儿子是在它的基础上生成的其他部分解
  - 回溯与穷举有联系，都是基于试探。穷举要将一个解的各个部分全部生产后，才检查是否满足条件，若不满足就直接放弃该完整解，然后尝试另一个可能的完整解。对于穷举，一个解的各部分是逐步生成的，当发现当前生成的某部分不满足约束条件，就放弃该步所作的工作，退到上一步进行新的尝试，而不是放弃整个解重来
- 分支限界法：类似于回溯法，他们都是在空间树上搜索问题的解，也可以看作是回溯法的改进。回溯法是在整个状态空间树中搜索解，并用约束条件判断搜索进程，一旦发现不可能产生问题的解的部分解，就中止对相应子树的搜索，避免不必要的工作。分支限界法与回溯法的差异
  - 控制条件：回溯法一般用约束函数产生部分解，若满足约束条件，则继续扩大该解，否则丢弃重新搜索。而分支限界法除了使用约束函数外，还使用更有效的评判函数—目标函数控制搜索进程，使尽快能得到最优解
  - 搜索方式：回溯法中的搜索方式一般是深度优先搜索，分支限界法一般用广度优先搜索。 从活结点表中选择下一扩展结点的不同方式导致不同的分支限界法
    - 队列式（FIFO）：将活结点表组织成一个队列，并按队列先进先出的原则选取下一个结点为当前扩展结点
    - 优先队列式：将活结点表组织成一个优先队列，并按优先队列中规定的结点优先级选择优先级最高的下一结点成为当前扩展结点
- 贪心：通过一系列选择得到问题的解。它所做的每一个选择都是当前状态下局部最好选择。这种启发式的策略并不总能获得最优解，然鹅在许多情况下的确能得到最优解。可用贪心求解的问题一般要具备两个重要性质：
  - 贪心选择性质：所求问题的整体最优解可以通过一系列的局部最优解的选择来达到，它采用自顶向下的方式将所求问题化简为规模更小的子问题

- 最优子结构性质：一个问题的最优解包含其子问题的最优解

- 动态规划：与分治和回溯类似，基于问题的划分解决方案。动态规划是将问题实例分解为更小的、相似的子问题，并存储子问题的解而避免计算重复的子问题，以解决最优化问题的算法策略。效率更高，常用来求最优解，而不像回溯法那样可以直接求全解。使用动态规划的问题需满足以下条件
  - 最优化原理（最优子结构性质）：一个最优化策略具有这样的性质，不论过去状态和决策如何，对面前的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的
  - 子问题的重叠性：对于重复出现的子问题，只在第一次遇到时加以求解，并把答案保存起来，让以后再遇到时直接引用，不必重新求解

第十章 码

- 待补充
- 目前还没考到纯代码题，但目前的出题思路来看，出纯代码的题几率不是很大，可以暂时以 408 的方法复习

組成原理

第零章 大纲

- 计算机系统概述
  - 计算机系统层次结构
    - 计算机硬件的基本组成
    - 计算机软件的分类
  - 性能指标必考点
    - 主频（CPU 时钟周期）
    - 运算速度（CPI、CPU 执行时间、MIPS、MFLOPS）
    - 性能评测公式（Amdahl）
- 指令系统
  - 计算机中的数据表示
    - 定点数、浮点数的表示
    - 校验码
  - 计算机的指令格式
    - 指令的基本格式
    - 扩展操作码的指令格式
  - 指令的寻址方式
    - 数据寻址、指令寻址
    - 常用寻址方式
- 计算机中的运算
  - 计算机中的基本运算（多练，即便是一道例题，多练十几次，也就基本会了）
    - 逻辑、移位运算
    - 定点数的加减乘除运算
    - 浮点数的加减乘除运算
  - 运算器的设计（可战略性放弃，但考过的那两次要记住）
    - 算术逻辑单元 ALU
    - 串/并行加法器
- 中央处理器
  - 数据通路、控制器的原理及结构
  - 流水线的基本概念及原理

- 硬连线控制器的基本概念及原理（不是重点，记住硬连线和微程序差异即可）
- 微程序控制器的基本概念及原理

- 存储层次结构除了这三点外，其他存储层次如交叉编址这类也要看看
  - 主存储器的基本概念及设计使用
  - 高速缓冲器（Cache）的原理和性能评估
  - 虚拟存储器的基本概念的原理和性能评估
    - 2021 年的题是一道极具综合性的虚存+Cache 题，有 408 的味道，虚存和 Cache 要精看
- 输入输出（I/O）系统
  - 输入输出系统的基本概念
  - I/O 设备和 I/O 接口
  - 磁盘的原理及性能评估
  - 三种 I/O 控制方式（主要是涉及到传送时候的周期用时计算）
    - 程序查询
    - 中断
    - DMA
- 总线
  - 总线的基本概念和性能指标
    - 性能指标慕课、爱课程、绿书讲的不多，要考，看王道
  - 总线的设计
    - 总线仲裁
    - 总线操作
    - 总线定时

- 个人看法
  - 首先慕课的页面上说，本视频覆盖了考研内容，这句话不假，凡是慕课上的内容都是要看的。但实际上还是漏了些内容，比如虚存部分、三种 IO 控制方式，这些都是比较多也比较重要的内容，往年是考了好多的，所以不应该轻易放掉这块，要通过爱课程及其 PPT 来补上
  - 2020.12.27 补充，目前今年的形势来看，出题思路有向 408 靠拢的味道

第一章 概述

1.1 计算机系统

- 摩尔定律：每 18 个月，集成电路性能翻一番，集成晶体管规模翻一番，价格降一半
- 计算机五大功能：数据的传输、存储、处理；操作的控制、判断
- 现代计算机框架：以存储器为中心
- 经典冯诺伊曼架构（控制流驱动/存储程序原理）：输入设备、输出设备、控制器、运算器、存储器
  - 特点：以运算器 ALU 为中心。程序数据预先放在存储器，存储器按地址编址。操作时根据程序中指令的执行顺序，从存储器中取出指令或数据，由控制器解释，运算器运算（慕课中存储程序原理的解释）
  - 处理器：执行程序
    - Control Unit + Datapath = CPU
    - Control Unit：对指令进行译码产生控制信号
    - Datapath：完成指令的执行
  - 运算器：数据处理部件
    - 完成算术逻辑运算、指令的执行（Datapath）
    - ALU + Registers = Datapath
      - ALU 执行算术逻辑运算
      - Reg 用来存储临时数据或控制信息
    - 性能指标：MIPS、MFLOPS
  - 控制器：控制功能部件
    - 对指令译码，提供各部件工作所需的控制信号，协调各部件自动化工作
    - 三大部件：指令控制、定时控制、微操作控制
    - 定时部件
      - CP：同步时钟，其工作频率称为 CPU 的主频，协调计算机各部件进行操作的同步时钟
      - TSG：时序信号产生器
      - 节拍：相邻两个时钟脉冲的时间间隔，与主频同周期
    - 微操作控制部件

        - 微操作：计算机各部件在同一个节拍内能完成的基本操作。一条机器指令的执行可以解释为一系列的微操作的执行
        - 微程序：由若干条比机器指令低一层次的微指令所组成的有序集合，通常固化在 ROM 中
    - 指令控制部件
        - IR：当前正在执行的指令
        - ID：将指令转换成微操作控制信号
        - PC：指令地址寄存器和计数器，决定指令执行顺序
    - 地址形成部件
        - XR：变址寻址
        - MAR：需要访问的存储器单元地址
        - 地址计算部件
- 存储器：存储程序和数据
    - 内存：Cache + Main Memory
        - Cache 存放最近使用的数据指令
        - Main Memory 存放被启动程序中的部分数据指令
    - 外存：磁盘、光盘、磁带等
    - 存储单元：bit 比特/位、byte 字节、word 字
    - 存储单元按顺序编号：地址，每一个单元的唯一索引
    - 速度比较：Reg>Cache>M>Disk Cache>Disk
- IO 设备：各种信息的输入输出
    - IO Controller + IO Device
    - IO Controller 控制外设工作，完成主机和外设之间的通信
    - IO Device 输入输出信息
- 总线用于各功能部件的连接（应该并不是指冯诺依曼架构的一部分）

- 数据流驱动：数据全部准备好后才能执行相应的指令操作，执行结果流向下一条指令，以驱动下一指令的执行。数据驱动指令执行
- 控制流驱动/存储程序原理：指令控制数据读取
- 指令和数据实际上都在存储单元里，以指令周期不同来区分
- 软硬件均能实现的功能为“软硬件上逻辑等效”
- 主存按存储单元的地址存取，按地址存取方式；相联存储器按内容/地址访问
- (存疑内容，821 貌似没怎么提这个点)MAR 和 MDR 现在存在于 CPU 中，但在运控外。地址译码还在主存。
- PC 放指令地址，故与 MAR 同宽，也与存储空间（主存地址空间）大小有关。

- 运算器、控制器、存储体构成主机
- 计算机解题步骤：建立模型、设计算法、编写程序、调试运行
- 高级语言$\overset{编译}{\Longrightarrow}$汇编语言$\overset{汇编}{\Longrightarrow}$目标文件（机器语言模块或库文件）$\overset{链接}{\Longrightarrow}$可执行文件（机器语言程序）$\overset{加载}{\Longrightarrow}$存储器中（二进制机器指令流）$\overset{取指译码控制}{\Longrightarrow}$电路上的电信号⇒直接控制 CPU 各部件运作/通过接口电路控制外设运作
- 汇编是把汇编语言转为机器语言
- 编译是把高级语言转为目标语言，并生成目标代码和可执行文件
- 解释：解一运一，不生成目标代码
- 计算机硬件只执行机器语言
- 程序的执行过程就是周期性重复以上操作
- 管理软硬件资源的是操作系统，提供软硬件接口的也是操作系统

1.2 性能指标

- 机器字长：计算机进行一次整数运算所能处理的二进制数据位数，与寄存器同宽
- 指令字长：一条指令中所包含的二进制代码的位数
- 存储字长：一个存储单元能存的二进制代码长度。字节的整数倍
- 操作系统的位数，是可寻址范围的位数
- 系统吞吐量取决于主存存取周期
- CPU 时钟周期，节拍脉冲或 T 周期，即主频的倒数，CPU 最小时间单位。nHz 为每秒 n 次。决定计算机运行速度，主频高，运算速度快
  - 主频提高方法：工艺、外频和倍频
- 运算速度：计算机工作能力和效率的主要表征
- 运算精度：计算机能直接处理的二进制信息位数衡量，与寄存器位数相关，位数越多，精度越高
- 存储容量：主存越大，处理问题速度越快
- 存取周期：对主存储器连续两次访问所允许的最小时间间隔
  - 存取周期越小，系统性能越高
- 性能：慕课中指的性能 = 1/执行时间
- CPU 执行时间 = 程序的总 CPU 时钟周期数/主频 =（指令条数 xCPI）/主频
- CPI 是执行一条指令所需周期数

- 总执行时间 = 程序 1 时间 × 程序 1 占比频率 + 程序 2 时间 × 程序 2 占比频率 +……
  - 该指标用于基准程序评估计算机性能
- CPU 性能（执行时间）取决于：主频、CPI、指令条数
  - 单独改变某一指标可能会导致其他指标的变化，所以要全面的评估
  - 算法、编程语言、编译器影响指令条数和 CPI 从而影响性能；指令集三者都影响
- MIPS = 指令条数/(执行时间 $x10^6$) = 主频/CPI，每秒执行多少百万条指令
  - 这种形势下主频的单位通常是 MHz
- M/G/TFLOPS = 浮点操作次数/(执行时间 $x10^{6/9/12}$)，每秒执行多少百万/十亿/万亿次浮点运算
- Amdahl 定律：改进后的执行时间 = 受改进影响部分执行时间/改进提高倍数 + 不受影响部分的执行时间
  - 若没有直接给出改进提高倍数，而给的是时间关系，要慎重思考

第二章 指令

2.1 编码

- 用二进制表示信息的原因
  - 制造二稳态器件容易
  - 二进制编码、计数、运算规则简单
  - 与逻辑命题对应，便于逻辑运算，方便地用逻辑电路实现
- C 语言中数据类型大小以字节为单位
- 单精度浮点、双精度浮点即短浮点、长浮点
- 二进制小数实质是用若干个 2 的负幂相加来逼近，故存在有些小数无法被二进制表示的情况，直接体现在“乘基取余”这个进程无穷尽，比如1/3 、π
- C 语言支持的数据类型（单位：字节）
- 数据宽度
  - 位 bit：计算机处理、存储、传输信息的最小单位
  - 字节 byte：二进制信息计量单位，现代计算机大多按字节编址，是最小可寻址单位
  - 字 word：表示被处理信息的单位
- 机器字长=CPU 中数据通路的宽度=CPU 内部总线的宽度=运算器的位数=通用寄存器的宽度，通常是字节的整数倍
- MIPS 机器中，无符号的 char、short int、int、long int 分别是 8、16、32、32 位，所表示的最大值最小值及有符号时最大值最小值自行推算
- 小端机 C 语言中，用补码表示二进制数
- ASCII 码
- BCD 码：每位十进制数至少 4 位二进制来表示
  - 十进制有权码：表示每个十进制数位的 4 个二进制数位都有一个确定的权，8421
  - 十进制无权吗：表示每个十进制数位的 4 个基 2 码没有确定的权，余 3 码、格雷码
  - 其他编码：5 中取 2 码、独热码
- 西文字符的编码表示：拼音文字，只需对有限个少量字母和一些数学符号、标点、辅助字符进行编码。所有西文字符集总字符数不超过 256 个，使用 7 或 8 个二进制位表示。常用 7 位 ASCII 码。常用操作：传送、比较
- 汉字表意文字，一个字符为一个方块图形。编码形式如下

- 输入码：对每个汉字用相应的按键进行编码，用于输入
- 内码：用于在系统中进行存储、查找、传送
- 字模点阵码/轮廓描述：描述汉字的字模点阵或轮廓，用于显示、打印、

- 数据的度量单位（重点注意，如果再出磁盘综合，相关的计算问题不要出错）

度量单位	缩写	存储二进制换算关系	计算机通信带宽换算关系
千字节	KB	$1\text{KB} = 2^{10}$字节 $= 1024\text{B}$	$1\text{KB} = 10^{3}$字节 $= 1000\text{B}$
兆字节	MB	$1\text{MB} = 2^{20}$字节 $= 1024\text{KB}$	$1\text{MB} = 10^{6}$字节 $= 1000\text{KB}$
千兆字节	GB	$1\text{GB} = 2^{30}$字节 $= 1024\text{MB}$	$1\text{GB} = 10^{9}$字节 $= 1000\text{MB}$
兆兆字节	TB	$1\text{TB} = 2^{40}$字节 $= 1024\text{GB}$	$1\text{TB} = 10^{12}$字节 $= 1000\text{GB}$

- 大部分机器都采用字节编址
- 大小端存储方式：将一个数 0x12345678 从左往右铺平写出，如果地址起始在 78 处，地址朝左递增，小端存放；如果地址起始在 12 处，地址朝右递增，大端存放。数字 12 是这个数的最高位，如果地址从这里开始，那就叫大端了。反之 78 是数字的最低位，如果地址从这里开始就叫做小端。
- 2^0就是最低位。以后可能遇到写数字、画地址线、画主存单元的时候，一会从左开始，一会从右开始，可能会搞混，所以要以 0 在哪为准
- 数据对齐：
  - 按边界对齐（假定字的宽度 32 位，存储器按字节编址）
    - 字地址：4 的倍数，低两位为 0
    - 半字地址：2 的倍数，低位为 0
    - 字节地址：任意
  - 不按边界对齐：可能会增加访存次数
- 冗余校验，除原信息外，增加若干位编码，新增的称为校验位
- 奇偶校验：增加一位奇偶校验位并一起存储传送，由目的部件进行数据校验
  - 举例：1001101 奇校验 <u>1</u>1001101 偶校验 <u>0</u>1001101
    1010111 奇校验 <u>0</u>1010111 偶校验 <u>1</u>1010111
    原数视情况而定，不符合奇数要求则奇校验补 1，否则补 0；偶校验同理
  - 只能发现奇数位出错，不能发现偶数位出错，不能确定错误位置，不能纠错
  - 开销小，适用于校验 1 字节长的代码，常用于存储器读写检查、按字节传输校验
  - 多少年来，校验只考了一次奇偶校验的简单概念，可以判定，校验码不是考察重点

2.2 指令格式

- 能够直接控制计算机工作的语言：机器语言

- 指令系统连接软件抽象层和硬件抽象层，是二者的接口及界面
- 指令系统设计原则：完备性、有效性、规整性、兼容性
  - 兼容性
    - 上（下）兼容：按某档机器编制的程序，不加修改就能运行在比它高（低）档的电脑上
    - 前（后）兼容：按某时期某型号编制的程序，不加修改就能运行在它前（后）生产的设备上
  - 数据传送指令、输入输出指令、算术运算指令、逻辑运算指令、系统控制指令、程序控制指令
- CISC 和 RISC
  - CISC 复杂指令集
    - 指令系统复杂、指令周期长、各种指令都能访问存储器、有专用寄存器、微程序控制、难以编译优化生成高效目标代码
    - 研制周期长、难以保证设计正确性难以调试维护、机器时钟周期长降低系统性能、效率低下（二八原则）
  - RISC 精简指令集
    - 简化的指令系统、以寄存器-寄存器方式工作、指令周期短、采用大量通用寄存器减少访存次数、采用组合逻辑电路控制不用少用微程序控制、采用优化的编译系统有效支持高级语言程序
  - 现代处理器多为 RISC 结构，x86 保留 CISC 风格又借鉴 RISC 思想
- 指令是实现某个计算机基本操作的命令，指令含义决定指令格式
- 指令格式：操作码 + 地址码
  - 由于不同的操作数种类长度不同，所以不可能直接像操作码一样给出，于是给出操作数所在的地址，去访问得到操作数
    - 地址码包含了操作数所在的地址、寻址方式
  - 指令长度取决于操作码长度、操作数地址长度、地址个数。定长与不定长
  - 每条指令操作码只有唯一一个，不同的操作由不同的编码表示
    - 定长操作码：译码简单，信息可能会冗余
    - 变长/扩展操作码：操作码采用可变长编码，长度分为固定几种格式，操作码位数随地址数减少而增加。可以缩短指令长度，减少程序总位数，增加指令字所能表示的操作信息。采用前缀码
      - 重要原则：高频指令短码，低频指令长码
    - 关注程序规模：变长；关注性能：定长

        - 指令 16 位长：4-8-12-16 位操作码时，所能分别提供 15 三地址、15 二地址、15 一地址、16 零地址；前缀的缘故仅 16 个 1 作操作码，其他位数个 1 不作操作码
    - 地址码 0 到多个：零地址、一地址、二地址（常用）、三地址（RISC 风格）
        - 具体需要几个由操作码决定
        - 地址码个数与性能和实现难度密切相关
        - 地址个数越少，指令长度越短，指令功能越简单
        - 堆栈结构：零地址；累加器结构：一地址；通用寄存器结构：二三地址
- 一条指令包含一个操作码和多个地址码
    - 零地址：OP
        - 无需操作数（空操作/停机）
        - 所需操作数位默认的（堆栈）
    - 一地址：OP A1
        - 其地址即是源操作数地址，又是存放结果地址
        - 单目运算（取反/取负）
        - 双目运算，另一操作数默认（累加器）
    - 二地址：OP A1 A2
        - 分布存放双目运算中两个源操作数地址，其中一个地址作为存放结果地址
    - 三地址：OP A1 A2 A3
        - 分别为双目运算中两个源操作数地址和一个结果地址
    - 多地址：用于成批数据处理（向量指令）
- 地址码的编码由操作数的寻址方式决定

2.3 寻址方式

- 寻址方式：就是如何找到操作数存放位置的方法
    - 扩大访存范围
    - 提高访问数据的灵活性和有效性
- 指令寻址
    - 正常：PC 增值
    - 跳转：jump、branch、call、return，同操作数的寻址
- 注意区分操作码、操作数、地址码等词义
- 多种寻址方式的目的：缩短指令长度，扩大寻址空间，提高编程灵活性
- 操作数寻址
    - 立即数寻址：源操作数直接在指令中，[操作码 ，目的操作数 ，源操作数]

- 指令执行时间短、无需访存
- 操作数的大小受地址字段长度限制
- 只能作为双操作数指令的源操作数

- 存储器直接寻址：操作数在存储器中，指令地址字段直接给出操作数在存储器中的地址[操作码，目的操作数，地址]
  - 处理简单、直接
  - 寻址空间受到指令的地址字段长度限制
- 寄存器直接寻址：操作数在寄存器中，指令地址字段直接给出存放操作数的寄存器编号[操作码，目的操作数，寄存器编号]
  - 只需很短的地址字段
  - 无需访存快速执行
  - 地址范围有限，可使用的通用寄存器不多
- 存储器间接寻址：操作数和操作数地址都在存储器中，指令地址字段直接给出操作数地址在存储器中的地址[操作数，目的操作数，地址的地址]
  - 寻址空间大，灵活，便于编程
  - 至少需要访存两次才能取到操作数，速度慢
- 寄存器间接寻址：操作数在存储器中，操作数地址在寄存器中，指令地址字段给出的是寄存器编号[操作码，目的操作数，寄存器编号]
  - 比存储器间接寻址少访存一次
  - 寻址空间大
- 偏移寻址：[操作码，目的操作数，寄存器+偏移]
  - 相对寻址：(PC) + A 相对于当前指令处，位移量为 A 的单元
  - 基址寻址：(B) + A 相对于基址(B)处，位移量为 A 的单元。基址不变，所有地址都是相对于基址的变化，偏移量位数较短，面向系统
  - 变址寻址：(I) + A 相对于形式地址 A 处，位移量为(I)的单元。形式地址给出的是基准地址，变址寄存器存放的是偏移，偏移量位数足够表示全部存储空间，立足用户
- 堆栈寻址：
  - POP：从堆栈到寄存器。PUSH：从寄存器到堆栈
  - 每次执行 POP 或 PUSH 操作后，除了进行操作数的处理，还要修改栈顶指针 SP。遇到题目先改内容还是先改指针，要结合题目认真分析原理

- 寻址方式总结

方式	算法	优点	缺点

立即	操作数=A	执行速度快	操作数幅值有限
直接	EA=A	有效地址计算简单	地址范围有限
间接	EA=(A)	有效地址范围大	多次访存
寄存器	操作数=(R)	执行速度快指令短	地址范围有限
寄间接	EA=(R)	地址范围大	一次访存
偏移	EA=A+(R)	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

- 寻址方式的确定：操作码中给定寻址方式、专门的寻址方式位
- 复合寻址
  - 间接寻址+相对寻址
  - 间接寻址+变址寻址
  - 以上又分为先间接或后间接，参考 PPT 图例以及王道相关题
    - 先间接
      - 间接相对：EA=(PC)+(A)
      - 间接变址：EA=(X)+(A)
    - 后间接
      - 变址间接：EA=((X)+A)
      - 相对间接：EA=((PC)+A)

2.4 指令系统

- 完善的指令系统包括：
  - 数据传送指令：Move、Load、Store、Exchange 等
    - 寄存器之间、存储单元之间、寄存器与存储单元之间的数据传送
    - 参数：源操作数地址（指令给出）、目的操作数地址（指令给出）、传送数据长度（指令隐含给出）
  - 算术运算指令：定点数运算、浮点数运算、十进制运算
    - 操作过程：将源操作数送到算术逻辑运算单元 ALU 的输入端、运算、将 ALU 的结果传送到目的操作数地址单元
  - 逻辑运算指令：And、Or、Not、Xor、Compare 等
    - 按位运算，可针对任何一种可寻址单元中的位进行操作
    - 与、或、非、异或等逻辑操作，位测试、位清除、移位操作等非运算操作
  - 输入输出指令：IN、OUT

        - 专用的 IO 指令：通常包含两个地址（数据地址、外部设备地址）
        - 通用的数据传送指令：存储器映像的 IO 操作，要求外部设备的寄存器和主存统一编址
        - 通过 IO 处理机执行 IO 操作：对 IO 系统的管理和控制由通道和 IOP 完成
    - 系统控制指令：启动 IO 设备指令、存取特殊寄存器指令等
        - 只能由操作系统执行，不提供给用户的特权指令，只有在特权状态下才能执行这些指令
        - 主要实现对控制寄存器的操作、检测修改访问权限、改变系统工作方式、访问进程控制块等
        - 用户需要使用特权服务事，通过系统控制指令向操作系统发出请求，由操作系统完成
    - 程序控制指令：转移指令、循环控制指令（LOOP）、子程序调用与返回指令（CALL、RET）、程序中断指令及返回（INT、IRET）
        - CPU 每执行一条指令，程序计数器 PC 就自增 Δ，要打破这种顺序就需要程序控制指令。
            - 转移指令、子程序调用及返回指令
        - 改变 CPU 执行指令顺序，让 CPU 到新的地址去执行后续指令。
            - 无条件转移：直接将转移目标地址送入 PC
            - 条件转移：实现方法：条件码、条件寄存器、比较与转移指令
- 子程序的使用称为子程序调用，调用子程序的程序为调用程序或主程序
- 子程序和转移指令改变程序执行顺序的不同
    - 子程序要求返回，可嵌套和递归调用；转移指令不要求返回，通常只在同一程序段出现
    - 子程序用于实现程序与程序之间的转移；转移指令用于同一程序内转移
- 程序再入形式：嵌套、递归
    - 寄存器存放返回地址：单寄存器不支持嵌套递归；多寄存器支持嵌套不支持递归
    - 放在子程序起始位置：支持嵌套不支持递归
    - 堆栈保存返回地址：可以实现嵌套和递归
- 入口参数：调用程序提供给子程序以便于加工处理的信息；出口参数：子程序加工处理后回送给调用程序的信息
    - 参数传递方法：约定寄存器法、约定存储单元法、参数赋值法、堆栈法
- 现场原则：破坏什么现场就保护什么现场、保护什么现场就恢复什么现场

- 子程序结构：保存现场、从入口参数拿到信息、加工处理、向出口参数传送信息、恢复现场、返回调用程序

- MIPS 指令部分，还是要看看，不会单独考的，要和处理器那部分结合起来考，主要是掌握三类指令的特征构造用途
- ISA 设计原则：简单源自规整、越少越快、加速常用操作、均衡设计
- 所有指令都是 32 位宽，按字地址对齐。不同类型指令采取了不同格式
- R 型指令（用于寄存器操作）
  - 字段中包含了两个源操作数寄存器 rs、rt，如 swb rd , rs , rt
  - [op , rs , rt , rd , shamt , funct]
    - op：6bit，操作码（R 型指令 op 都一样）
    - rs：5bit，第一个源操作数寄存器
    - rt：5bit，第二个源操作数寄存器
    - rd：5bit，存放结果的目的寄存器
    - shamt：5bit，移位指令位移量
    - funct：6bit，功能码（区分各种操作）
  - 寻址方式：寄存器寻址
- I 型指令（立即数运算、数据传送、条件分支）
  - 一个寄存器一个立即数，例如：
    - 运算：ori rt , rs , imm16
    - Load/Store：lw rt , rs , imm16
    - 条件分支：bep rs , rt , imm16
  - [op , rs , rt , imm]
    - imm：16bit，立即数或 Load/Store、条件分支的偏移量
  - 寻址方式：立即数寻址、基址寻址、相对寻址
- J 型指令（无条件转移）
  - 例如：j target
  - [op , target]
    - target：26bit，无条件转移地址的低 26 位
  - 寻址方式：（伪）直接寻址，PC 高 4 位拼上 26 位地址后，在低位处补 00
- RISC 和 CISC 的一些区别

	CISC	RISC
指令系统	复杂庞大	简单精简
指令数目	一般大于 200 条	一般小于 200 条

指令字长	不固定	定长
可访存指令	不限制	仅 load/s′tore 指令
指令执行时间	相差较大	大多一个周期内
指令使用频度	二八原则	都常用
通用寄存器	较少	较多
控制方式	多为微程序	多为组合逻辑
流水线	可通过一定方式实现	必须实现
目标代码	难以优化编译高效代码	优化的编译程序、代码高效

第三章 运算

3.1 基本运算

- 原反补三码问题，并不是什么难的，自行意会
  - 真题中，给出的是带+-的十进制数，将其绝对值转为 n 位原码后，再加一位符号位，然后再开始接下来的转换补码反码；移码也是，再多一位，用2^n来算移码。如果是给出的带+-的二进制数同理
- 按位运算
  - 与：&、∧，全 1 才 1，提取操作
  - 或：|、∨，全 0 才 0，指定位赋值
  - 取反：~
  - 异或：⊕、^，相异为 1，判等价
- 表达式的值只有真假，逻辑操作后还是如此
- 逻辑操作
  - 与：&&
  - 或：‖
  - 非：！
- 无符号数逻辑移位：无论左右移，都添 0
- 有符号数算术移位：原符不参反补参，原码恒补 0，正反补 0 负反 1，补左添 0 右添符
- 扩展
  - 无符号扩展补 0
  - 有符号扩展补符
  - 对于补码形式不确定补什么的时候，转成原码进行扩展，扩展完转回补码
- 截断：直接截断，可能会溢出

3.2 定点

- 定点数格式：$[x_0\ x_1x_2\cdots\cdots x_{n-1}x_n]$
  - 定点整数：小数点固定在最低位的右边，表示范围$0 \le |x| \le 2^n - 1$
  - 定点小数：小数点固定在数值部分的最高位的左边，表示范围$0 \le |x| \le 1 - 2^n$
- 补码加减法

- $[A \pm B]_{补} = [A]_{补} + [\pm B]_{补}$
- 符号位参与运算
- 采用两位符号，相同正常，01 正溢，10 负溢

- 原码加减法
  - 符号数值分别处理。
  - 加法：先判符号位，若相同则绝对值相加，符号位不变；若不同，则做减法，绝对值大的数减去绝对值小的数，结果符号位和绝对值大的一致
  - 减法：首先将减数符号取反，然后将被减数与符号取反后的减数按 原码加法进行计算
- 原码乘（看王道例子）
  - 符号位异或，数值部分绝对值乘
- 补码乘（Booth 法）（看王道例子）
  - 乘数和被乘数的符号位参与运算
  - 乘数末尾设置附加位y_{n+1}，初值为 0
  - 类似原码乘，只是需要在加前判断乘数末两位差值
    - $y_{n+1} - y_n = 0$，下次+0；$y_{n+1} - y_n = -1$，下次$+[-x]_{补}$；$y_{n+1} - y_n = 1$，下次$+[x]_{补}$
  - 移位的时候记得符号
  - 最后拼接的时候取y_n之前的部分
  - $(-1) \times (-1)$是唯一会溢出的情况
  - **口诀**：负减正加 0 不动，y_n之前是剩余
- 原码除（看王道例子）
  - 符号位异或，数值部分绝对值除
  - **口诀**：负 0 下加，正 1 下减，初始相减
- 补码除，王道的和国防科方法不太一样，参考绿书
- 补码得到的积、余数都是补码，记得转为原码（如果需要转）
- **一定要看例题练习领悟，只看例题，反复看足够了，不需要再做其他新题**

3.3 浮点

- 浮点数的表示：尾数+阶码
  - 进一步改进，在总位数不变的情况下，为提高表示精度，使尾数的有效数字尽可能占满已有的位数
- $N = r^E \times M$

- r 是浮点数阶码的基/底，一般为 2，隐含
- E、M 是有符号定点数，E 是阶码，M 是尾数
  - 可能用原码表示，有可能用补码表示

- 规格化：小数点前只有一位非 0 数
  - 右规，尾数算术右移一位，阶码加 1，仅一次
    - 一般是结果尾数溢出的时候（符号位 01、10 时）
  - 左规，尾数算术左移一位，阶码减 1，可能需要多次
  - 基为 2 的时候，原码规格化数的尾数最高位 1，补码规格化数最高位与尾数符号相反
- IEEE754 标准：[s(符号)，e(阶码)，f(小数)]

类型	符号 s	阶码 e	尾数 f	总位数	偏置值	
					十六进制	十进制
单精度	1	8	23	32	7FH	127
双精度	1	11	52	64	3FFH	1023

  - s：1 表示负数；0 表示正数
  - f：尾数为原码，规格化尾数最高位总为 1，故隐含表示，精度 1+23 位（单精度），1+52 位（双精度）
    - 尾数精度=尾数位数+1， 即隐含的那一位
  - e：移码表示，即阶码数值加上一个偏置常数，当阶码位数为 n，偏置取2^{n-1}
    - 单精度偏置 127(7FH)，双精度偏置 1023(3FFH)
    - 阶码为 0，尾数为 0 表示 0，尾数非 0 表示非规格化数
    - 阶码 255，尾数为 0 表示无穷，尾数非 0 表示非数 NaN
    - 单精度浮点数$(-1)^s \times (1+f) \times 2^{e-127}$
    - 双精度浮点数$(-1)^s \times (1+f) \times 2^{e-1023}$
  - 浮点数表示的范围由基数 R 和阶码 e 的位数共同决定，越大，浮点数范围越大
- 浮点数精度由位数 f 的位数决定
- 设$A = M_a \times 2^{E_a}$，$B = M_b \times 2^{E_b}$，假设$E_a > E_b$
- 浮点乘除反而简单
  - $A \times B = (M_a \times M_b) \times 2^{(E_a+E_b)}$
  - $A \div B = (M_a \div M_b) \times 2^{(E_a-E_b)}$
  - E、M 之间都是最基本的定点加减乘除
- 浮点加减（要多回顾，多看，多练，不然忘得很快，定点计算也是）
  - 对阶：小对齐大，小阶尾数右移
  - 尾数相加减
  - 规格化尾数

        - 左规：尾数出现 00.0XXXX……和 11.1XXXX……，左规，直到 00.1……和 11.0……
        - 右规：尾数结果溢出（10.……和 01.……），右规
    - 判溢出
    - 舍入
    - 置 0
- 821 考察过的舍入法
    - 0 舍 1 入：丢的是 0 则不管；丢的是 1 则加 1
    - 恒置 1：无论丢 0 丢 1，尾数末尾恒为 1

3.4 运算器设计

- 暂时赌一把，今年大题考运算考加法器设计可能不太大
- 把考过的那两次背下，再不管了

第四章 处理

写在前面的话

- 流水线部分绿书未涉及到，所以要看另一本指定教材黑书，但仅有这本是不够的，还要看 CSAPP 这本，因为出题曾有参考这本书
- 谈一下王道和慕课在处理器部分的内容编排差异，尤其是章节分布安排方面
  - 关于单周期和多周期，实际上数据通路这部分主要就是在讲单周期的数据通路，慕课又拿出来细讲了一下在数据通路各部件的控制信号的含义和变化，这当然也是王道欠缺的部分，具体表现在，做王道习题时，各部件在不同的题中有着命名不同的信号，很乱
  - 多周期的章节，就是王道的“控制器设计”，但沈老师视频中提到，涉及数电的内容不是本课程重点，概不讲解，所以硬布线不是重点，主要是微程序部分，但慕课又把微程序拿出去单独做了一章，综上所述，慕课多周期的部分不是重点
  - 单、多周期的慕课课后题算性能的，实质上还是第一章的东西，也不重要，但要注意一下“加速比”这个概念。2020.12.27 果然被我猜中了，今年考了加速比
  - MOOC 视频 4.2.4 要认真看，不要倍速，要多看，看的时候手上跟着画

4.1 数据通路

- 每个基本操作的时间都是一个“机器周期”
- 每条指令都有一个机器周期用来取值，称为“取指周期”
- 如果有间址周期，介于取指和执行之间
- 多个机器周期完成指令功能，称为“执行周期”
  - 包括译码、执行、访存、写回等阶段
- 数据通路：指令执行过程中，数据所经过的路径（包括路径中的部件），是指令的执行部件
  - 要传递哪些数据
  - 数据传递方向
  - 数据传递顺序
- 控制器功能：对指令进行译码，生成对应的控制信号，控制数据通路的动作，能对指令的执行部件发出控制信号，是指令的控制部件
  - 控制指令流出（取指令）

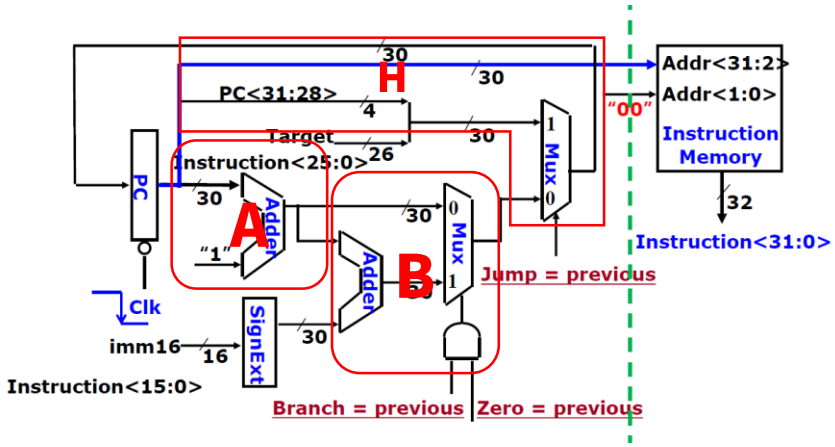
- 控制指令执分析（译码）
- 控制指令执行（执行指令，发出操作命令）
- 控制指令流向（确定下一条指令地址）
- 控制执行环境的维护（执行环境的建立保护，如修改寄存器 FLAG/PSW 的值）

- 以下例子中 PC 的自增 Δ 为 4（参考刘芳 PPT）
- **王道中()表示取该处内容，刘芳 PPT 则是[]**
- **下面的数据通路部分，详细看慕课视频，视频讲的很透彻**
- ADD R1 , R2 , R3 数据通路（功能：R1 <- R2 + R3）
  - 取值阶段：Inst <- Mem[PC]　PC <- PC + 4
  - 指令译码：A <- [R2]　B <- [R3]
  - 完成加法：ALUOutput <- A + B
  - 结果写回：[R1] <- ALUOutput
- Load 指令：LW R1 , R2 , #4 数据通路（功能：R1 <- Mem[R2 + 4]）
  - 取指令：Inst <- Mem[PC]　PC <- PC + 4
  - 指令译码：A <- [R2]　立即数符号扩展为 32 位
    - 将 16 位的立即数符号扩展到 32 位，便于
  - 计算访存地址：A + 4
  - 读取存储单元：LMD <- [A + 4]
  - 结果写回：[R1] <- LMD
- Store 指令：SW R1 , R2 , #4 数据通路（功能：Mem[R2 + 4] <- R1）
  - 取指令：Inst <- Mem[PC]　PC <- PC + 4
  - 指令译码：A <- [R2]　B <- [R1]　立即数符号扩展为 32 位
  - 计算访存地址：A + 4
  - 写入存储单元：Mem[A + 4] <- B
- 将 ADD、Load、Store 数据通路合并，该通路可以同时满足以上三条指令
- 条件分支指令与 ADD、Load、Store 指令合并的数据通路（红色为条件指令）
  - 在第三步的时候要判断转移条件，即 A 是否为 0；还有就是将原本的 Next PC 和偏置值相加，形成目标地址
  - 第四步，根据转移条件是否形成，来决定输出转移后的地址或原本的 Next PC
- 将以上的四条指令：条件转移指令、ADD 指令、Load 指令、Store 指令进行合并，即可以得到 MIPS 处理器基本的数据通路
  - 仅 Load 需要结果写回
  - 大部分指令都需要经过这五个步骤：取指、译码、执行、访存、写结果

- 取指令到指令结束的时间为一个“指令周期”
- MIPS 固定字段译码技术：无论执行哪条指令，译码的时候都准备好所有的源操作数，ALU 没有用到的源操作数都会丢弃（慕课 4.2.4 2:30 会）
- 每条指令功能开由四个基本操作实现，按照一定顺序进行组合即可设计处理器
  - 读取某一主存单元的内存，并将其装入寄存器中
    - 取指操作、Load 访存操作
  - 把一个数据从某个寄存器存入主存单元
    - Store 访存操作
  - 把一个数据从某个寄存器送入另一寄存器或 ALU
    - 译码操作、ADD 和 Load 指令的结果写回
  - 进行某种算术运算或逻辑运算，结果送入某个寄存器
    - 指令在 ALU 单元上运行的都是
- 最基本的操作叫做“微操作”，不可再分，可由 RTL（寄存器传输语言）描述
  - R[r]表示寄存器 r 的内容
  - M[addr]表示地址为 addr 的存储单元内容
  - <-表示数据传送，源右目左
  - PC 表示程序计数器的值（原则上应该是[PC]，可能一般省略不写，大概因为 PC 和[PC]不会产生歧义）
  - OP[data]表示对数据 data 进行 OP 操作
- 同一个功能单元可能会完成多个微操作
  - 寄存器：读、写。用 RegWr 表示 1 写 0 读操作（**王道则是用 in 和 out 来区分**）
  - 存储器：读、写。用 MemWr 表示 1 写 0 读操作（同上）
  - ALU：算术运算、逻辑运算。用 ALUctr
  - 多路选择 MUX：选择不同的输入作为输出。用 RegDst、ALUsrc 等
- 设计问题
  - 数据通路设计问题
    - 一条指令由哪些微操作完成
    - 一条指令按怎样的顺序完成这些微操作
  - 控制器设计问题：怎样为不同类型的指令生成每个机器周期需要的控制信号
    - 一个功能单元执行哪些微操作
    - 在某个机器周期，功能单元执行哪个微操作
- 单周期处理器设计步骤（数电内容，不做要求）
  - 确定数据通路上每个元件所需要的控制信号以及控制信号的取值
  - 汇总所有指令涉及到的控制信号，生成一张反映指令与控制信号的关系表

- 根据关系表得到每个控制信号的逻辑表达式，根据此设计控制器电路

- 取指令两个操作
  - 访问存储器，就是读出地址为 PC 的那个存储单元里的指令
  - 根据指令类型更新 PC，使得寄存器的内容为接下来要执行的那条指令的地址
    - 分支指令：if PC <- PC + 4 + Imm　else PC <- PC + 4
    - 跳转指令：PC <- PC + 4 + Target
    - 其他指令：PC <- PC + 4
- 因为会出现三种指令的不同状况，所有要引入控制信号



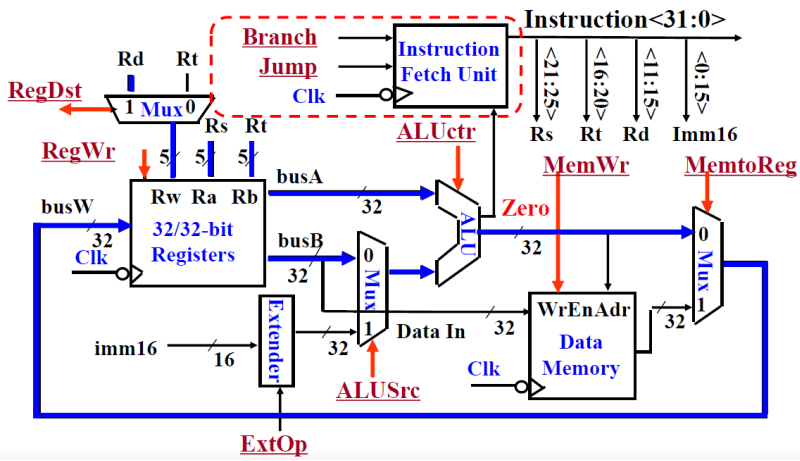
虚线左侧是当前指令修改 PC，右侧是从主存取指令

A 框：PC + 4 。H 框：Jump 指令 PC = PC + Target，否则再讨论其他情况

B 框：“如果是跳转指令且满足条件”PC = PC + 4 + Imm，否则就 PC + 4

- Jump：是否为跳转指令，1-是，0-否
- Branch：是否为分支指令，1-是，0-否
- Zero：分支转移条件是否为真，1-是，0-否

- MIPS 数据通路中的控制信号（数电内容，不做要求）



名称	控制对象
RegDst	选择目的寄存器
RegWr	读或写寄存器
ExtOp	立即数扩展方式
ALUSrc	ALU单元的输入
MemWr	读或写存储器
MemTo Reg	选择要写回寄存器的数据
ALUctr	ALU单元的操作
Branch	是否分支指令
Jump	是否跳转指令
Zero	分支转移条件是否为真

  - 指令译码阶段会读取寄存器文件，写回阶段会写寄存器文件，RegWr：1 写 0 读
  - ALUSrc：决定了 ALU 下面的输入是 B 还是 Imm，与指令类型有关
  - 读写存储器，MemWr：1 写 0 读/不操作存储
  - MemToReg：1 将 LMD 写回，0 将 ALU 的输出写回
  - ExtOp：1 符号扩展，0 零扩展

4.2 微程序

- 单周期处理器的缺陷
  - CPI 为 1，所有指令的执行时间以最长指令 Load 为准
  - 单周期处理器的时钟周期远大于其他指令实际所需的执行时间，效率极低，因为
    - R-型指令、立即数指令不需要读内存
    - Store 指令不需要读寄存器
    - 条件分支指令不需要放内存和写寄存器
    - Jump 不需要 ALU 运算、不需要读内存、不需要操作寄存器
  - 解决以上问题的方法就是“多周期处理器”
- 单周期处理器：时钟周期以最复杂的指令所需时间为准，太长，解决思路
  - 把指令分成多个阶段，每个阶段在一个周期里执行
    - 尽量分成大致相等的若干阶段
    - 每个阶段最多只完成一次访存、读写寄存器、ALU 运算
  - 每个阶段的结果保存在专用的内部寄存器中
- 多周期处理器的好处

- 时钟周期短
- 不同指令所用的时钟周期数可以不同
- 允许功能部件在一条指令执行过程中被重复使用

- 指令独占整个数据通路，直到执行结束
- 多周期处理器中很多控制信号会在执行过程中发生改变，而单周期处理器中控制信号一经决定就不再改变。一条指令的执行过程中，如何决定各个控制信号在不同周期的不同取值，是多周期处理器设计的关键
- 多周期处理器的功能描述方式：
  - 有限状态机（硬布线）：采用组合逻辑设计方法通过硬连线实现
  - 微程序：设计微程序，并用 ROM 存放微程序实现
- 组合逻辑控制单元和多周期数据通路的关系（这个暂时放一放，感觉考察可能不大）
  - 控制单元以当前指令的操作码和当前处理器的状态为输入
  - 两部分输出：当前指令所对应的控制信号（输出到多周期数据通路进行控制）、下一状态（保存至状态寄存器作为下一次的输入）
  - 不同的状态输出不同的控制信号，下一状态也被当做一种控制信号但不输出到数据通路去，而是单独输出到状态寄存器
- 状态表（这个暂时放一放，感觉考察可能不大）
  - 每条指令的执行都是从 0 状态开始，也就是取指令 0:IFetch 开始
  - R-type/Ori/Sw 指令：4 个周期
  - Beq/Jmp 指令：3 个周期
  - Lw 指令：5 个周期
- 每个状态由一组控制信号的取值来决定
- 硬布线和微程序差异
  - 硬布线
    - 优点：控制器速度取决于电路延迟，速度快
    - 缺点：控制部件用专门固定时序的逻辑电路制作，一经设计不能额外变动
  - 微程序
    - 优点：规整性、灵活性、可维护性
    - 缺点：采用程序存储原理，每条指令都需要取指一次，速度慢

	微程序	硬布线
工作原理	控制信号以微程序存在控存，执行时读出	控制信号由逻辑电路实时产生
执行速度	慢	快

规整性	比较规整	繁琐、不规整
应用场合	CISC	RISC
易扩充性	易扩充修改	不易扩充修改

- 加速比 Speedup= 多周期性能/单周期性能 = 单周期时间/多周期时间

- 微操作用微指令表示，固化在特定的存储器 CM 中，为每条机器指令编制一个微程序，执行该微程序，完成机器指令的执行
- 微程序控制器思想
  - 仿照程序设计的方法，编制每条指令所对应的微程序
  - 每个微程序由若干条微指令构成，一条微指令包含若干位微命令
  - 所有微程序放在一个只读存储器中，执行某条指令时，取出对应微程序中的各条微指令，对微指令译码产生对应的微命令，这些微命令就是控制信号
  - 该只读存储器为控制存储器 CM，即控存
- 微程序规整、可维护、灵活，但速度慢
- 微操作是微命令的执行过程，微命令是微操作的控制信号

- 微程序
  - 实现一条指令功能的需多微指令组成的序列
  - 微程序=微指令序列
  - 一条指令对应一段微程序
- 微指令
  - 将机器指令执行的微操作序列中能够一个节拍内同时完成的微操作，用控制位构成二进制代码串来表示
  - 指令的执行=微指令的序列
- 微命令
  - 控制部件通过控制线向执行部件发出各种控制命令

- 控制部件和执行部件通过控制线和反馈线进行联系

- 微操作
  - 执行部件接受微命令后所进行的最基本的操作
- 微指令周期：从控存读取一条微指令并执行完成相应的微操作所需要的时间。一般时间是固定的（指令周期是可变的），是微程序控制的重要指标
- Wilkes 模型阐述的微程序控制器的构成
  - 指令寄存器，用来保存正在执行的机器指令
  - 微程序顺序控制逻辑μC，由其来决定这条指令所对应的微程序，特别是微程序第一条的指令地址
  - 访问地址控制部件 FCMAR，根据当前正在执行的微指令地址字段来决定下一条微指令的地址，给控存发送地址
  - 控制存储器
  - 控制信号字段，吧控制信号发送给数据通路
  - 微地址码字段，发送给 FCMAR 生成下一条微指令
- 微程序控制器工作过程
  - 取 IR 操作码经μC 变换为该条指令微程序入口的微地址码
  - 访问地址部件 FCMAR 选择微地址码，作为当前 CM 的地址
  - 根据地址从 CM 读出一条微指令存入微指令寄存器中，其控制信号字段表示了当前运算器、存储器、控制器、FCMAR 所需要执行的微操作
  - 重复步骤二三，用微指令寄存器中的地址码作为当前微地址码，直到一条指令对应的微程序执行完成。接着执行下一段微程序取下一条指令放在 IR 中，然后返回第一步
- 微指令包含若干微命令、下条微指令地址（可选）、常数（可选）
  - 格式[μOP，μADD，常数] μOP 微操作码字段，产生微命令；μADD 微地址码字段，产生下一条微指令地址
  - 微指令的编码风格取决于微操作码的编码方式
- 微操作码编码方式
  - 水平型微指令风格：面向处理器内部控制逻辑的描述；相容微命令尽量多安排在一条微指令中；优点是指令译码简单，微指令最大限度表示微操作并行性，编制的微程序短，微程序执行速度块，适合高速场合；缺点是微指令字较长，编码空间利用率低，编制最佳水平微程序有难度
    - 不译法（直接控制编码）：[μOCF，下一地址]

- 微命令产生不必译码，从操作控制字段直接得到，每一位微命令用一位信息表示
- 优点：并行控制能力强，不需译码，控制电路简单，速度快，编制的微程序短
- 缺点：微指令字长，编码空间利用率低，控制存储器容量大
- 适用于简单告诉部件，如 YH-1 指令流水部件

■ 字段直接编码法：[μOCF1，μOCF2，μOCF3，・・・ ,下一地址]

- 将微操作控制字段划分为若干小字段，每个小字段单独编码，每个字段表示一种微命令，每段经译码后发出控制信号
- 优点：微指令分段数越多，并行控制能力越强；微指令字段短，能压缩到不译法的 1/2 或 1/3，节省控存容量
- 缺点：增加译码电路，增加一部分开销时间
- 已被大多数微程序控制的计算机采用
- 相容微操作，能同时进行的微操作，对应的微命令称为相容微命令，相容的微命令分布在不同的字段
- 互斥微操作，不能同时进行的微操作，对应的微命令称为互斥微命令，互斥的微命令分布在同一字段

■ 字段间接编码法

- 某些参与编码的微命令不能由一个控制字段直接定义，而需要两个或以上的控制字段参与。即，一个微命令字段可以表示多个微命令组，但具体代表哪一组微命令，则由另一个字段决定
- 优点：牺牲并行性、速度换取微指令字长的缩短，节省控存
- 缺点：如上，且译码电路复杂增加开销
- 只限于局部场合使用

○ 垂直型微指令风格：面向算法的描述；采用短格式，一条微指令只能控制一个微操作；优点是指令字长短，编码空间利用率高，格式与机器指令类似，编写微程序容易；缺点是微程序长，并行微操作能力有限，微指令译码复杂，速度慢

■ 最短（垂直）编码法

- 将所有微命令统一二进制编码，每条微指令只包含一个微命令，每次只产生一个微操作，通过译码器产生微操作控制信号
- 优点：微程序规整直观，易于编制，微指令字长短
- 缺点严重：微程序长；硬件设备复杂，需要大量译码电路和门电路；速度慢，每次只产生一个微命令，难以提高微指令执行速度

● 微指令地址产生的方法

- 顺序-转移（计数器）法：下条微指令地址隐含在微程序计数器μPC 中
  - 用类似程序计数器 PC 产生当前机器指令地址的方法
  - 设置一个微程序计数器μPC，指示当前微指令地址
  - 顺序执行微指令时，下条微指令地址由μPC 增加一个增量产生
  - 遇到转移时，由微指令给出转移微地址
  - 结构[mOCF， BAF , BCF]
    - 转移地址字段 BAF 和转移控制字 BCF 共同构成地址字段 SCF
    - BAF 的位数决定了转移范围大小和灵活性
    - BAF 指出转移地址的来源：由 BAF 确定的地址、机器指令所对应的微程序的入口地址、微子程序的返回地址
  - 优点：微指令中 SCF 字段较短，下条微地址产生机构简单
  - 缺点：不利于两路以上的并行微程序转移，从而不利于提高微程序的执行速度，微程序在 CM 中物理分配不方便
- 断定（下址字段）法：当前微指令中显式指定下条微指令地址
  - 下条微地址由微程序设计者直接指定，或由微程序设计者指定的测试判别字段控制产生
  - 结构[mOCF， HF , TCF]
    - 非测试字段 HF，测试控制字段 TCF
    - 测试地址的位数确定了转移的并行度：n 位为2^n路转移
  - 优点：能以较短的 SCF 配合实现多路并行转移，提高微程序执行效率和执行速度。微程序在 CM 中分配物理空间方便灵活
  - 缺点：下条微地址码的生成结构比较复杂

- 当前微指令执行结束后，下条要执行微指令的三种情况
  - 取微程序首址：每条指令都要先执行“取值微程序”
  - 某机器指令对应微程序的首条微指令：当执行完取值微程序的最后一条微指令，需根据当前指令的译码结果确定执行哪条指令所对应的微程序，然后转移到对应微程序的首条微指令执行
  - 某微程序执行过程中一条微指令的三种情况
    - 顺序执行：按顺序取出下条微指令执行
    - 无条件执行：无条件转到另一处微指令执行
    - 分支执行：根据条件码或者指令操作码转移到不同微指令执行

4.3 流水线

- 考点
  - 各种性能（吞吐量、加速比、延迟、时空图比等）会算
  - 两种图形会画且详细记忆
  - 三种冒险会分析找出、会解决（调整指令顺序、加气泡、插 NOP）
- 流水线是一种将多条指令的执行过程互相重叠的实现技巧，使 CPU 可以通过并行执行多条指令来提高吞吐率
- 常用时空图来描述流水线
- 流水线的特点
  - 单个任务执行时，所需要的总的时间并没有缩短
  - 多个任务执行时，所有工作并行处理，在单位时间内完成的工作量大大增加了
  - 在不改变单个任务执行时间的前提下，流水线提高了吞吐率
- 流水线适合大量重复的时序过程
- 在单周期、多周期处理器中，指令是会独占整个数据通路的，而在流水线中，在每一个时钟周期里，会有多条指令在一个数据通路上执行
- 流水线的两种图形表示方法
  - 多周期流水线图（流水线时空图）：描述了流水线的整体情况，可以清楚看出执行一段指令所花的通过时间、总时间
  - 单周期流水线图：描述了同一个时钟周期内整个数据通路的状态和操作，可以看到在数据通路上存在着不同流水段的不同操作阶段
    - 理解此点的前提是前面的数据通路烂熟于心
- 加速比Speedup = 单周期处理器执行时间/流水线处理器执行时间
- 流水线处理器执行时间$T_k = (k + n - 1)\,\Delta_t$　n 条指令 k 段流水，Δ_t为
- 吞吐量= n/T_k
- 适合流水线的指令集：规整、简单、一致
  - 指令长度尽量一致，有利于简化取指令和指令译码操作
  - 指令格式少，且源寄存器位置相同，有利于在指令未知时就可以取操作数
  - 只有 Load/Store 指令才可以访存，有利于减少操作步骤，规整流水线
  - 数据和指令要在内存中“对齐”存放，有利于减少访存次数和规整流水线
- 流水线冒险：流水线无法正确执行后续指令、执行不了该执行的指令
- 结构冒险/资源冲突：同一个功能部件同时被多条指令使用
  - 避免：每个功能部件每条指令只能用一次（如不能两次或两次以上对寄存器进行写）、每个功能部件必须在相同阶段被使用（如总是在第五阶段对寄存器进行写）
  - 方法：

        - 插入气泡（在流水段之间插入），抵达或通过某个流水段的指令正常进行，其余指令在当前各自流水段等待一个周期。但控制逻辑复杂、影响 CPI
        - 修改相关指令的操作，使其（流水段）延后一个周期执行，如加一个 NOP 以延迟写操作，会使流水线保持规整（比如读 Mem 操作）
- 控制/分支/转移冒险：转移或异常改变执行流程，顺序执行指令在目标地址产生前已被取出（结合视频里的那个例子，那个例子非常好）
    - 硬件阻塞 stall：同数据冒险
    - 软件插入 NOP 指令：同数据冒险
    - 分支预测
        - 简单（静态）预测：
            - 总是预测分支条件不满足，继续执行分支指令的后续指令：如果预测错误，需要丢弃三条本不该执行的指令，会给流水线带来损失
            - 用启发式规则，在特定情况下总是预测转移成功，其他情况则转移不成功（如循环顶部总是预测不成功，底部总是预测成功）
        - 动态预测：根据程序历史执行情况，动态预测调整
            - 转移发生的历史情况记录在特定的缓存中，分支历史记录表 BHT、分支预测缓冲 BPB、分支目标缓冲 BTB。每个表项由分支指令地址地位作索引，IF 阶段即可以检测到
    - 延迟分支：属于静态调度技术，由编译器重排指令顺序来实现
        - 把分支指令前面的与分支指令无关的指令调度到分支指令后面执行，以填充延迟时间片，找不到可调度的指令时用 NOP 填充
- 数据冒险/数据相关：后面指令用到前面指令结果，但结果还没产生
    - 硬件阻塞 stall：硬件通过阻塞方式阻止后续指令执行，延迟到新值被写入寄存器以后，也即插入气泡（在指令之间插入）。缺点是控制相当复杂，需要修改数据通路
    - 软件插入 NOP 指令：指令之间插入 NOP 以延迟执行。缺点是会浪费几条指令的时间和空间（整行气泡，非结构冒险的一段气泡）
    - 转发 Forwarding 或旁路 Bypassing 技术：把数据从流水线寄存器直接送到 ALU 的输入端（需要在哎译码段增加监测点，检测 ID 段源操作数和 EX 段目的操作数是否相同）、寄存器的写/读分别安排在前/后半周期（写入数据可在同一周期读出）。缺点，对 load-use 指令不起作用，要么阻塞要么插入空操作（因为 load 产生的数据来自第四阶段的访存，此时无法提前转发给下条流水线）
    - 编译优化：调整指令顺序
- 流水线寄存器：保存前一个流水段的结果、为后一个流水段提供输入

- CSAPP 的 p282 独有内容
- 以上的情况都是在指令规整的情况下，也就是每个流水段都一样长的情况，但各流水段时间不一致时，比如 50ns、150ns、100ns 这样的时候，就要考虑插入流水线寄存器，来进行划分，也即 2019 年真题的情况
- 如果要求固定段数让你划分，就进行平均划分即可（注意，不要把寄存器的延时划进任何一段去）；如果要求最大效率，那就找出最长的一个段，以次段的长度为基准，各拼接段的长度不要超过它
- 新的延迟为最长的那个段为基准的全部新段延迟+新加的若干个寄存器延迟
- 新吞吐量计算= $\frac{1}{(\text{新的基准流水段延时}+\text{寄存器延时})\text{ps}} \cdot \frac{1000ps}{1ns}$ 吞吐量单位 GIPS
  - 计算方法与前面的不一样，注意
  - 是按照一个流水段来算的

第五章 存储

5.1 存储器

- 存储器是计算机中的记忆设备，用来存放程序和数据。主要分为两类：主存和辅存。主存，Memory，常称为内存。辅存，Storage，常称为外存。存储器在现代计算机体系结构中是中心地位，向 CPU 提供数据和指令。
- 存储器的层次结构从上到下依次为：寄存器>片上高速缓存>片外高速缓存>主存储器>本地二级存储>远程二级存储。
  - 层次越高，更小更快更高；层次越低，更大更慢更便宜。
  - 每一层只存较低一层的部分数据
- 相关术语：
  - 记忆单元/存储位元/位元/Cell：具有两种稳态能够表示二进制数 01 的物理器件。
  - 存储单元/编址单位/Unit：存储器中拥有相同地址的位构成的一个存储单元。
  - 存储体/存储矩阵/存储阵列/Bank：所有存储单元构成的一个存储阵列。
  - 存储器地址寄存器/MAR：用于存放主存单元地址。
  - 存储器数据寄存器/MDR：用于存放主存单元中的数据。
  - 机器字长：运算器中参加运算的寄存器位数，等同于数据通路宽度。
  - 存储字长：存储芯片中一个读写单位，与存储器数据线等宽。
  - 编址方式：对存储体中各单元进行编号的方法。
  - 传输单位：对于主存一般是位，对于辅存一般是块。
- 存储器分类
  - 计算机中的作用
    - 寄存器型存储器：由寄存器构成，封装在 CPU 中，用于存放当前正在执行的指令和使用的数据，用触发器实现，速度快容量小
    - 高速缓冲器：位于 CPU 内部或 CPU 与主存之间，用来存放当前正在执行的局部程序段和数据，用 SRAM 实现
    - 主存储器：位于 CPU 之外，用来存放已被启动的程序及所用数据，用 DRAM 实现
    - 辅存储器：不能用 CPU 指令直接访问，用来存放暂时不运行的程序、数据、存储文档，容量大但速度慢

- 其他功能存储器：控存 CM（存微程序代码）、表格存储器（位加快 CPU 处理，函数表、倒数表等）、字库和缓冲存储器（显示和印刷输出设备）等

- 存储介质
  - 半导体存储器 SCM：速度快、用作内存；原理，双稳态触发器、电容
  - 磁表面存储器 MSM：非易失，容量大且价格低，用作外存陶瓷、非磁性材料作为磁载体
  - 光盘存储器 ODM：非易失，可靠性高，保存时间长，容量大；有机玻璃做磁载体，利用磁化、晶态非晶态表示信息
  - 铁电存储器 FeM、相变存储器 RCM、阻变存储器 ReRAM 等，掉电数据不丢失，速度快
- 存储方式
  - 随机访问存储器 RAM：存储器任意单元可随时访问且访问所需时间相同，访问时间与存储单元所在物理位置无关，速度快；主存和 Cache
  - 只读存储器 ROM：正常工作只读，不能随机读出，不能随机写入；MROM 只读；PROM 一次写；EPROM、E^2PROM 多次改写
  - 相联存储器 CAM：按内容检索到存储位置进行读写，速度快价格高；快表
  - 直接存取存储器 DAS：信息的组织同 SAS，介于随机和顺序存取之间；可以直接定位到要读写的数据块，存取时间的长度与数据所在的位置有关；速度慢；磁盘
  - 顺序存储器 SAS：存储时以数据块为单位存储，顺序地记录在存储介质上；数据按顺序从存储载体的始端读出或写入，存取时间的长短与数据所在的位置有关；速度慢、容量大、成本低、用作后援外存；磁带、VCD
- 存储器中信息可保存性
  - 断电后是否丢失数据
    - 挥发性存储器（易失性存储器）：断电后信息即丢失；RAM 类
    - 非挥发性存储器（非易失性/永久性存储器）：ROM、磁盘、闪存
  - 读出后是否保存数据
    - 破坏性读出：读出时原信息被破坏，需要重写；DRAM
    - 非破坏性读出：SRAM

- 内存主要技术指标
  - 存储容量=存储单元个数×每个存储单元的存储位数
  - 存取时间T_A和存储周期T_S（这部分看视频理解）
    - T_A启动一次存储器操作到完成该操作所需要的时间

        - 亦称访问时间，反映存储器速度的指标，决定了 CPU 发出读写命令后必须等待的时间
    - T_S连续两次启动同一存储器进行存储操作所需要的最小时间间隔，也称存取周期/访问周期。还包括存储介质和控制线路的恢复时间，若是破坏性读出还要写回
  - 存储带宽和存取宽度
    - 存取宽度：一次访存操作可存取的数据位数或字节数，存取宽度由编址方式决定
    - 存储带宽：每秒传输的最大数据量
  - 可靠性：两次故障之间的平均时间间隔
  - 功耗与集成度
    - 集成度：标识单个存储芯片的存储容量
  - 性价比
- 内外存的关系
  - 任务启动，执行任务的程序和数据从外存成批送入内存中
  - CPU 从内存中逐条读取该程序的指令及其数据
  - CPU 逐条执行指令，按指令要求完成运算和处理
  - 将指令的运算结果送回内存保存
  - 任务完成，将处理得到的全部结果成批传送到外存永久保存

外存储器	内存储器
存取速度慢	存取速度快
成本低、容量大	成本高，容量较小
CPU 不能直接访问	CPU 可直接访问
非易失	易失

- 半导体存储器
  - 随机存储器 RAM
    - 静态随机存取存储器 SRAM
      - 用作 Cache
      - 只要加电就能保持信息
      - 对电器干扰不敏感
      - 比 DRAM 快也更贵
    - 动态随机存取存储器 DRAM
      - 用作主存

        - 数据以电荷形式保存在电容中，电容漏电使得信息只能保存 2ms 左右，每隔一段时间必须刷新（读出后重新写回）
        - 对电器干扰比较敏感
        - 比 SRAM 便宜但慢
    - 只读存储器 ROM
        - 不可在线改写的 ROM：用作 BIOS
        - 闪存 Flash：用作主存扩展、U 盘等
- RAM 分类（考过）
    - 双极 RAM（BiRAM）
        - 集成低、功耗大、速度快
        - 不复用
    - 静态 RAM（SRAM）：双稳态触发器构成（六管静态 MOS 电路）；
        - 介于之中
        - 数复用
    - 动态 RAM（DRAM）：电容构成
        - 集成高、功耗小、速度慢
        - 数地复用
- DRAM 刷新：按一定时间间隔为记忆电容补充电荷
    - 按周期分配方式分类
        - 集中刷新：在一个刷新周期内，利用一段固定时间，依次对存储器的所有行进行刷新，此期间停止读写操作，称为死时间。读写操作不受刷新工作的影响，系统存取速度快。死时间期间不能访问存储器
        - 分散刷新：对每一行的刷新分散到各工作周期取。存储器系统工作周期分两部分：前半部分正常读写或保持，后半部分刷新某一行。增加了系统的存取时间，没有死时间，但增加了周期速度慢
        - 透明刷新：结合前两种方式，即可缩短死时间，又可充分利用最大间隔 2ms 的特点。刷新周期除以行数，得到两次刷新操作时间间隔 t，利用逻辑电路每隔 t 产生一次刷新请求，避免 CPU 连续等待过长的时间，减少刷新次数，提高效率
    - 按操作控制方式分类
        - 同步、异步、半同步
- 刷新对 CPU 透明
- 刷新 DRAM 的单位为行
- 刷新实质类似于读操作，但不需要信息输出

- 层次：存储位元->存储单元->存储矩阵->存储芯片（译码、驱动、读/写电路）->存储模块（内存条）->存储器
- 存储器芯片=存储体+外围电路（地址译码、读写控制）
- 存储容量=寻址总单元数×每单元位数（自行体会）
- PPT 中位片式与王道中译码片选法不一样，注意！
- ROM 按地址寻址，只能随机读而不能随机写，但也以随机存取方式工作！！
- 注意“支持随机存取的存储器”和“随机存取存储器”的区别
- ROM 分类
  - 固定掩膜 ROM：只能读不能修改
    - 结构简单、高集成、便宜
    - 一经生产无法修改，适合大批量生产
    - 厂家按客户要求制作存储内容
  - 一次可改写 PROM：只能修改一次，修改后只读
    - 熔断丝型 PROM 为主流，熔断后信息不能改变
    - 即按位寻址又按地址寻址，编程时按位寻址且脱机编程
  - 可多次改写 EPROM：改写之前要先擦除
    - 光擦除电可编程只读存储器 UV EPROM
    - 电擦除电可编程只读存储器E^2PROM
    - 块擦除电可编程只读存储器 FM

5.2 主存设计

- RAM 芯片的基本引脚 ：地址线 A、数据线 D、行选通 RAS、列选通 CAS、上电 Vcc、地线 GND、读写 WE、使能 OE
- 主存储器设计，利用存储芯片和其他逻辑芯片构成
- 存储容量的扩展
  - 位扩展
    - 字数（存储单元数）不变，位数扩展
    - 引脚的连接
      - 地址端、片选 CS、读写 WE、使能 OE（若有），分别并接
      - 数据输入、输出端，各位单独引出
  - 字扩展

        - 位数不变，容量扩展
        - 引脚的连接
            - 地址线 A、读写 WE、使能 OE（若有）、数据输入输出端，分别并接
            - 片选 CS 单独引出，与高位地址码的译码结果连接达到片选目的
    - 字位扩展
        - 存储单元数、字长同时增加
        - 引脚的连接
            - 地址线 A、读写 WE、使能 OE（若有），分别并接
            - 片选 CS 位向并接，字向独立引出
            - 输入输出数据线位向独立引出，字向并接
- 驱动负载
    - 存储器逻辑设计中，外围电路芯片是驱动，存储芯片各端点就是负载
    - 逻辑电路的负载能力有限
    - 双极型芯片各端点为电流负载；MOS 型芯片各端点为电容负载
    - 负载因数 $f_{端点名}$：存储芯片某种端点中的一个端点的负载量，如f_A表示地址端的负载因数
- 负载计算
    - 地址驱动线的负载$L_A = \lceil M/m \rceil \lceil N/n \rceil f_A$
    - 读写控制驱动线负载$L_{WE} = \lceil M/m \rceil \lceil N/n \rceil f_{WE}$
    - 行片选驱动负载$L_{CS} = \lceil N/n \rceil f_{CS}$
    - 数据输入线的负载$L_{DI} = \lceil M/m \rceil f_{DI}$
    - 数据输出线的负载$L_{DO} = \lceil M/m \rceil f_{DO}$
- 速度估算：外围电路传输要引起时延，事存储器的系统存储周期、存取时间比存储芯片的存储周期、存取时间要长。速度估算要留有适当的余地以保证能够正常工作，通常应比要求的小 10%（应该说不是考点）
    - 系统取数时间T_{SA}
        - T_{SA}由读出时涉及 A（低位）、WE、A′（高位）、CS 信号到达存储芯片的时间决定。WE 一定先于 AA′到达存储芯片
        - 若 A 最后到达芯片，系统地址取数时间$t_{SAA}=t_{AD}+t_{AA}$，分别为地址缓冲延迟时间和存储芯片寻址取数时间
        - 若 CS 最后到达芯片，系统片选取数时间$t_{SCSA}=t_{ADC}+t_{ACS}$，分别为片选译码延迟和系统片选取数时间
        - 以上两个由于不会共存，一般选择较大者为系统取数时间T_{SA}
        - 若输出有缓冲器，应考虑输出缓冲延时t_{BD}，$T_{SA}=\max\{t_{SAA},t_{SCSA}\}+t_{BD}$

- 系统存储周期$T_{SM} = T_M + t_D + t_R$
  - T_M为芯片存储周期：查手册
  - t_D为系统的传输时延：由外围电路逻辑级数确定
  - t_R为系统恢复时间：通过系统测试获得
- 关于主存设计这一块的解题步骤
  - 确定存储芯片数量
  - 确定芯片的连接
  - 负载计算与分配
  - 速度估算
- CPU 和存储器总线连接方式
  - 地址线的连接
    - CPU 地址线数决定了整个主存空间的寻址范围
    - 通常 CPU 地址线的地位与存储芯片地址连接，高位用作字扩展时的片选
  - 数据线的连接
    - CPU 数据线决定了一次读写的最大数据宽度
    - 通常将 CPU 数据线连接到多个位扩展芯片中
  - 控制线的连接
    - 若 CPU 的读写命令线和存储芯片的读写命令线是一根，且电平信号相等，可以直接连接，否则分开连接
- CPU 和主存的通信方式：异步、同步。可在后续总线章节继续了解同步异步
- 主存空间划分
  - ROM 区，存放系统程序、标准子程序等，由 ROM 芯片构造
  - RAM 区，存放用户程序，由 RAM 芯片构造
- 选择 ROM 和 RAM 芯片时要先确定两个区的范围
- 慕课上的重要例题，极大研究价值，务必认真对待

5.3 Cache

- 程序访问局部性：在较短时间内，程序产生的地址/数据往往集中在存储器的一个很小范围内
  - 产生原因：

        - 指令按序存放，地址连续；循环重复执行
        - 数据连续存放，如数组元素重复、按序访问
    - 时间局部性：刚被访问过的存储单元很可能不久又被访问
    - 空间局部性：刚被访问过的存储单元临近单元很可能不久被访问
- Cache 对程序员（编译器）透明，在编写低级语言程序时，无需了解 Cache 是否存在或如何设置
- Cache 是小容量高速缓冲存储器，由 SRAM 组成。内嵌在 CPU 中，与 CPU 同速。一般将 Cache 和主存的存储空间划分为若干块大小相同的块（主存称块、Cache 称行）
- 关于 Cache 有效位的操作
    - 开机或复位，所有 V=0
    - 命中 Cache 行，V=1
    - Flush（初刷新） Cache 行，V=0
    - 新装入 Cache 行，V=1
- Cache-主存层次的访问时间
    - 命中 Hit
        - 命中率 Hit Rate（p）：在 Cache 中的概率，命中次数/总访问次数
        - 命中时间 Hit Time（T_c）：访问 Cache 所需要的时间，包括判断时间+访问 Cache 时间
    - 失效 Miss
        - 失靶率 Miss Rate：1-p
        - 失效损失 Miss Penalty（失靶损失T_m）：从主存将一块信息替换到 Cache 所需的时间，包括访问主存块，向上逐层传输块直至将数据块放入缺失的那一层所需要的时间
    - 平均访问时间=$p \times T_c + (1-p) \times (T_M + T_c) = T_c + (1-p) \times T_M$
    - 访问速度和命中率关系非常大
- Cache 中存放一个主存块的对应单位称为行 Line/槽 Slot/项 Entry/块 Block
- 主存划分成大小相等的主存块 Block
- 映射方式
    - 直接映射[tag , Cache 槽号 , 块内地址]
        - 把主存的每一块映射到一个固定的 Cache 行中，即每个主存地址对应于高速缓存中的唯一地址，也称模映射
        - Cache 行号=主存块号 mod Cache 行数
        - 易实现，命中时间短；淘汰替换策略简单；不灵活，Cache 存储空间得不到充分利用，命中率低

- 全相联映射[tag，块内地址]
  - 主存块可以装到 Cache 任一行/槽中，称为全相联映射
  - Cache 的 tag 指出对应槽取自哪个主存块；主存 tag 指出对应地址位于哪个主存块。两个标记相等，说明要找地址在对应槽中，比较所有标记都不等，则失靶
  - 因为要同时比较所有 Cache 行的 tag 标记，故主存被简单划分为 tag 标记和块内地址，所以 Cache 行不再设置 index 索引
  - 块冲突概率低，只要有空闲块，都不会发生冲突；但实现复杂，比较逻辑电路代价大（相联存储器），速度慢，不适合大容量 Cache
- 组相联映射[tag，组号 ，块内地址]
  - 将所有 Cache 行分组，把主存块映射到固定的组内，组内采用相联映射
  - Cache 组号=主存块号 mod Cache 组数
  - 结合全相联和直接映射的优点，
  - 每组两个行（2 路组相联）比较常用。较大容量的 L2 和 L3 Cache 中使用 4 路以上组相联
  - Cache 的 tag 指出对应槽取自哪个主存组群；主存 tag 指出对应地址位于哪个主存组群

- 关联度：主存块映射到 Cache 时，可能存放的位置个数
  - 直接映射，关联度 1
  - 全相联映射，关联度为 Cache 行数
  - N-路组相联映射，关联度 N
  - 提高关联度可以降低缺失率，但会增大命中时间
- Cache 性能和 Miss Rate 有关，Miss Rate 由与 Cache 大小、Block 大小、映射方式、Cache 级数有关
  - Cache 越大，Miss Rate 越低，但成本越高
  - Block 越大，Miss Rate 越低。但增大到某一程度，Miss Rate 会反而增加，且访问时间越长，一旦 Miss，损失成本也会变高
  - 小容量 Cache 映射方式影响 Miss Rate；大容量 Cache 映射方式不影响
- Cache 失效类型
  - 强制失效
    - 首次访问某数据块时必然 Miss
    - 增大 Block 有利于降低此类失效
  - 容量失效
    - Cache 不能存放程序运行所需的所有块，替换后再次被使用所导致的 Miss

        - 增加 Cache 大小有利于降低此类失效
    - 冲突失效
        - 映射到同一组的数据块个数超过组内可容纳的块时，竞争所引起的 Miss
        - 全相联没有此类失效，但价格贵且访问速度慢
- Cache 抖动：某一 Cache 行频繁的进行替换，而其他的行空闲。通常由冲突失效和容量失效引起
- 设置的块如果太小，会使得缺失率增大
- Cache 一致性
    - 当 Cache 中的内容发生了更新但主存块中却没有相应的改变时，就会产生不一致
    - I/O 设备直接通过 DMA 读写主存，如果 Cache 内容被修改，则 I/O 设备读出的数据无效；若 I/O 设备修改了主存，则 Cache 内容无效
    - 当多个 CPU 都有私有 Cache 且共享主存时，则对应注册单元和其他 CPU 中对应的 Cache 行内容都要无效
- 写命中机制：要写的单元已经在 Cache 中（当 CPU 访存写一个字的时候，判断是否在 Cache 中，在则写命中）
    - Write Through（写直达、写通过、直写）
        - 当一个写操作产生时，新值同时写到 Cache 和主存块中
        - 写主存速度很慢，会增加开销
        - 增加一个写缓冲 Write Buff
            - 当数据等待被写入主存时，先写入缓冲
            - 把数据写入 Cache 和缓冲后，处理器继续执行
            - 写主存操作结束后，释放缓冲
    - Write Back（写回、一次性写、回写）
        - 当一个写操作产生，新值仅被写到 Cache 中，而不写入主存，只有此块被换出时才会写回主存
            - 降低主存带宽需求，提高系统性能，但控制可能很复杂
            - 设置一个脏位，若被修改则脏位置 1，若未被修改则脏位位 0
- 写不命中机制：要写的单元不在 Cache 中（当 CPU 访存写一个字的时候，判断是否在 Cache 中，若不在则写不命中）
    - Allocate-on-miss（写分配）
        - 将主存块装入 Cache，然后更新 Cache 行相应单元
        - 利用空间局部性，增加了从主存读入一个块的开销
    - No- allocate-on-write（写不分配）
        - 直接写主存，不把被写数据放入 Cache

        - 减少了从主存读一个块的开销，未利用空间局部性
- 写直达 Cache 可用写分配或写不分配；写回 Cache 常用写分配
- 替换算法（直接映射无条件替换，全相联映射和组相联映射都需要替换算法）
    - 先进先出 FIFO：总是替换最先进入的一块
    - 最近最少使用 LRU：总是把最近最少使用的那一块淘汰，利用时间局部性
        - 当分块局部化范围（某段时间集中访问的存储区）超过了 Cache 存储容量时，命中率会很低，极端情况下，使得命中率为 0。该现象称为抖动。如访问序列 123412341……，但 Cache 只有 3 行，无论 FIFO 还是 LRU，命中率 0
        - 给每个 Cache 行设定一个计数器，根据其值记录使用情况，称为 LRU 位
            - 每次被访问时清 0，否则自增，越小表示越被常用
            - 未命中且改组未满，新行置 0 其余加 1
            - 未命中但改组已满，最大值行淘汰且置 0 其余加 1
    - 随机替换 Random
        - 实验表明，性能上只逊于 LRU，代价非常低
- 多级 Cache：在 Cache-主存系统中，使用多层次结构，可以掩盖 CPU 访存延迟，提高处理器执行效率
    - 片内 Cache：将 Cache 和 CPU 做在一个芯片上
    - 片外 Cache：不做在 CPU 内而是独立设置一个
    - 单极 Cache：只用一个片内 Cache
    - 多级 Cache：同时使用 L1、L2Cache，，有些还有 L3
        - L1 更靠近 CPU，速度比 L2 快，容量比 L2 小
    - 联合：数据指令都放在一个 Cache 中
    - 分立：数据和指令分开存放在各自的 Cache 中
        - L1 一般是分立 Cache，L1 的命中时间要比命中率更重要，减少命中时间以获得更短的时钟周期
        - L2 一般都是联合 Cache，L2 的命中率要比命中时间更重要，降低缺失率以减少访问主存缺失损失
- 两级 Cache 缺失损失分析
    - 若 L2 包含所请求信息，则缺失损失为 L2 的访问时间
    - 否则要访问主存，并同时取到 L1 和 L2，损失更大
- 缓存技术，充分利用程序访问的局部性特点，将大容量、慢速存储器中当前刚用过的局部数据复制或暂存在小容量、快速存储器中，提高计算机访存系统效率

5.4 存储层次

- 解决主存访问慢的措施
  - 提高主存芯片本身速度
  - 在主存和 CPU 之间加入 Cache
  - 采用并行结构技术
- 双端口存储器：一个存储器提供两套独立工作的读写控制电路、地址缓存、地址译码、两个读写端口，根据地址线和数据线可以同时进行两个数据的读写
  - 通常作为双口 RAM 或指令预取部件
- 多模块存储器（多体交叉）
  - 包含多个小存储体，每个体有自己的 MAR、MDR、读写电路
  - 多个存储模块可以独立并行工作
  - 可提高数据访问速度
- 连续编址（高位交叉访问）
  - 主存地址码的高位区分存储体号，低位表示模块内地址
  - 存储地址连续的数据落在同一存储体内，容易发生访存冲突，并行存取可能性小
    - 放存冲突：同时有两个或两个以上访存地址指向同一存储体，不能同时访存
  - 用于非共享主存（每个处理机仅享用统一编址主存的部分连续地址空间）和专用 Cache 的多处理机系统
- 交叉编址（地位交叉访问）
  - 用主存地址码的低位区分存储体号，高位表示模块内地址
  - 连续字分布在多个体中，各体轮流编址、并行工作提高存储器访问速度
  - 存储地址在同一存储体中不连续，以存储体个数为模交叉编址
  - 连续的程序或数据将交叉存放在 m 个存储体中，可实现以 m 为模的交叉并行存取，访存冲突概率很小
  - m 个存储体分时启动：一般采用流水线的方式工作，每存储体的启动间隔为$t = \lfloor T_m/m \rfloor$，在不改变每个体的存取周期情况下，提高存储器的带宽
- 并行计算机的访存模型
  - UMA 模型：均匀存储访问模型
    - 物理存储器被所有处理器均匀共享
    - 访问任何存储字的时间相同
    - 每台设备可带有私有高速缓冲
    - 外围设备可以某种形式共享
  - NUMA 模型：非均匀存储访问模型

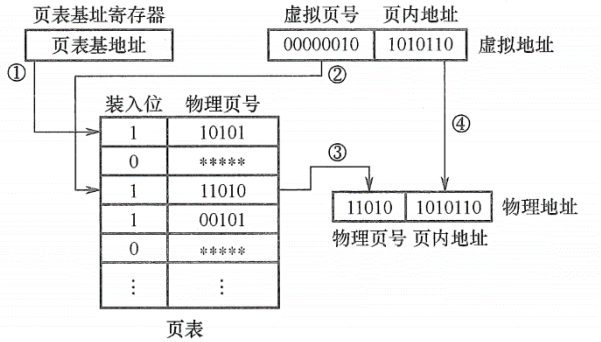
        - 被共享的存储器在物理上分布于各个处理器
        - 处理器访问不同存储器的时间不一样
        - 每台设备可带有私有高速缓冲
        - 外围设备可以某种形式共享
- 多核处理器，在单芯片上有多个处理器，可能会共享主存中的一个公共物理地址空间
- Cache 提供共享数据的迁移和复制
    - 迁移：数据项可以移入本地 Cache 并以透明的方式使用。迁移减少访问远程共享数据项的延迟和对共享存储器带宽的需求
    - 复制：当共享数据被同时读取时，Cache 在本地对数据项做了备份。复制减少了访问延迟和读取共享数据时的竞争现象
- Cache 共享数据带来的一致性问题
    - 两个不同处理器所保存的存储器视图是通过各自的 Cache 得到的（采用写直达）
    - 两个处理器可能分别得到两个不同的值
- 多处理机系统 Cache 一致性解决方法
    - 软件方法：借助编译程序进行分析，使共享数据只放在主存，不允许放入高速缓存中
    - 硬件方法：硬件协议维护，关键在于跟踪所有共享数据块的状态
        - 侦听协议：基于总线连接的多处理机系统中，每个处理器的高速缓存设置一个侦听部件，侦听总线上的事务活动。若侦听到主存中有一个单元被其他处理器修改，而自己在 Cache 中又有该单元副本时，将自己 Cache 中的副本置为无效，或更新该副本
- 目录协议：在主存中设置一个目录表，表中每一项记录共享数据的所有高速缓存行（数据块）的位置和状态，包括几个指示器（指示数据块的副本放在哪些处理器的高速缓存中）和指示位（指示是否已有高速缓存更新过此数据块的内容）。当一个处理器写入自己的高速缓存时，基于目录表，只需有选择的通知存有该数据块的处理器中的高速缓存，使相应副本被废弃或被更新

5.5 虚存

- 这一部分，可重点学习王道的相关章节，王道及 408 题目非常有价值，认真研读
- 将主存和辅存的地址空间统一编址，形成一个统一的虚拟空间
    - 虚地址：逻辑地址，对应虚拟空间

- 实地址：物理地址，对应实地址空间

- 使用虚地址的时候，由辅助硬件找出虚地址和实地址的对应关系，判断这个虚地址对应的存储单元是否在主存中。如果在，通过地址变换，CPU 直接访问该主存单元；如果不在，需要将包含该单元的一页或一段调入主存后再由 CPU 访问。若主存已满，还需要置换算法进行置换
- 页式虚拟存储器：
  - 主存空间和虚拟空间都划分为大小相同的页，主存的页称为实页，虚存的页称为虚页
  - 虚实地址的变换由页表完成。页表是存放虚页号和实页号的对照表，记录着程序的虚页调入主存时被安排在主存的位置。页表一般长久地保存在内存中
    - 页表基址寄存器存放当前运行程序的“页表的起始地址”
    - 页表基址和虚页号拼接成页表项的地址，每个页表项记录了某个虚页对应的虚页号、实页号、装入位等信息



- CPU 访存时，先查页表，为此需要访问一次主存，若不命中，还要进行页表替换和页表修改

第六章 出入

6.1 IO 概述

- I/O 系统是用于控制外设和主存、外设与 CPU 之间进行数据交换的软硬件系统
  - I/O 硬件：I/O 设备（外设）和 I/O 接口
  - I/O 软件：参与 I/O 任务的专用软件
- 主机之外，皆为外部设备（也称外围设备）
- I/O 设备分类
  - 交互方式
    - 人-机交互设备
    - 机器可读设备
  - 操作功能
    - 输入输出设备
    - 外部存储设备
- I/O 设备地位和作用
  - 计算机系统重要组成部分，如果没有 I/O 设备，计算机不能运转；如果 I/O 设备不全或性能不好，计算机效率低
  - 人机通信和对话的工具，了解机器运行状态，进行故障诊断、命令、控制计算机
  - 不同媒体间信息变换装置
  - 系统软件及信息资源的驻留地
  - 推广应用的桥梁
- I/O 设备的特点
  - 速度慢
  - 多样性：品种多、功能强、涉及多学科
  - 复杂性：传输方式、传输速率、工作原理差异大
- I/O 设备操作特点
  - 异步性：外设速度远远低于处理器，两者不使用统一时钟
    - I/O 设备通常按照自己的时钟工作，但又要在某些时刻接受主机的控制
    - 外设与处理器间的信息交换是随机的
    - 主机与 I/O 设备之间、I/O 设备与 I/O 设备之间能够并行工作

- 实时性：外设速度慢，一旦启动，则以固定速率工作，要求主机在规定的时间里完成信息交换
- 数据交换的复杂性和多样性
  - 数据交换的对象种类繁杂，所传输的数据类型不一
  - 数据传输的速率相差巨大，所传输的数据格式千差万别
- I/O 操作的实现与设备无关性
  - 主机尽可能少地考虑外设内部细节，让外设的特殊性隐藏在各自的设备控制器和接口的某些可变部分中
  - 处理器通过简单的命令控制外设并完成 I/O 操作

- I/O 系统常用性能指标
  - 响应时间：即 I/O 延迟，完成一次 I/O 事务所需要的时间
  - 吞吐率：单位时间内完成的输入输出操作次数
  - I/O 带宽：单位时间内从系统输入输出的数据量
- I/O 系统的功能，解决各种形式信息的输入输出
- I/O 接口，主机与 I/O 设备之间数据交换的界面。在主机和外设之间，屏蔽差异，提供一致的访问界面。主机看不到 I/O 设备的不同，I/O 设备看不到主机的不同
  - 数据格式转换和电平变换
    - 接口和主机之间通常采用并行传输，接口和外设之间有的采用并行有的采用串行
    - 外设及其控制电路所需要电源与主机可能不同，导致信号电平存在差异
  - 数据缓冲：用于解决速度不匹配导致的传输问题
  - 寻址功能：对外设接口进行编址，以方便操作访问
  - 提供外设和接口状态
    - I/O 接口中的状态寄存器：忙/闲、就绪、中断、屏蔽、数据传输错、故障等
  - 实现主机对外设的控制
    - 读写、端口选择、外设的启动、磁盘寻道、中断请求等
- I/O 接口的结构：由各种寄存器和控制电路构成
  - 缓冲寄存器：暂存外设与主机要交换的信息，与数据线相连
  - 状态寄存器：存放外设和接口的状态信息
  - 控制寄存器：寄存 I/O 指令中的命令码
  - I/O 控制逻辑模块：所有控制线路的集合，控制实现外设与主机的信息传输
  - 外设界面控制逻辑：接收送来的设备地址，和外设完成数据交换
- I/O 接口：主机和外设之间设置的硬件电路和相应的软件控制，包括各种寄存器和逻辑电路

I/O 端口：I/O 接口中包含的各种寄存器，是为了和主机中 CPU 内部的寄存器做区别而统一取了个名字。相应的寄存器被称为控制端口、状态端口、数据端口

- I/O 端口编址：为了使 CPU 能访问各寄存器
  - 统一编址（单总线结构才行）
    - 将 I/O 端口映射到主存空间的某区，与主存统一编址，将主存空间分出一部分地址给 I/O 端口进行编号，也被称为存储器映射方式
    - 将外设接口中可访问的寄存器和存储器的存储单元等同对待，可用访存指令去访问外设中的寄存器
  - 独立编址
    - 对 I/O 端口单独编号，使之称为独立的 I/O 地址空间，存储单元地址和 I/O 端口地址毫无关系
    - 需要专门的 I/O 指令，把 I/O 操作和存储器读写区分开来

	优点	缺点
统一编址	与访存指令一致的存/取指令 读写控制逻辑简单	主存空间减少； 外设寻址时间长
独立编址	不占用存储器地址空间 寻址速度快 使用专用 I/O 指令，程序清晰，便于理解检查	I/O 指令减少； 增加了控制逻辑的复杂性和处理器引脚数

6.2 磁盘综合

- 磁盘结构
  - 磁盘盘片：将硬磁材料涂敷、电镀、沉积在金属或者玻璃材质基板上，记录二进制信息的载体
  - 步进电机
  - 读写磁头：将电脉冲信号转换成介质上的磁化状态，又将介质的磁化状态转换成电脉冲信号，使电磁转换的桥梁
  - 传动手臂
- 磁盘组织

- 磁道：每个记录面划分为多个同心圆，每个同心圆的轨迹称为磁道。从外到内，从 0 编号
- 扇区：每个磁道划分为若干个大小相同的圆弧，每个扇区里的数据作为一个单元
- 柱面：多个盘面上位于同一半径的磁道构成一个柱面，柱面不是物理概念，更像是逻辑概念
- 磁盘地址：[台号,柱面号,盘面号,扇区号]（绿书）

- 磁盘性能指标：记录面 m、每面道数 n、转速 r、转一圈时间 t（$=1/r$）、每道扇区数 S_t、每扇区字节数B_s、位密度D_b、最内圈磁道直径D_{min}
  - 记录密度：单位长度或单位面积磁表面存储的二进制信息数
    - 道密度D_t：在垂直于磁道或光道方向上单位长度磁介质所容纳的磁道数，单位 tpm（道数每毫米）或 tpi（道数每英寸）
    - 位密度D_b：单位长度磁道上所记录的二进制信息位数，单位 bpm（位每毫秒）或 bpi（位每英寸），通常指最内圈磁道的位密度
    - 面密度：单位面积上记录的二进制信息位数，单位bpm^2或bpi^2
      - 面密度=道密度×位密度
  - 题目如果给出磁道内外直径（单位厘米）
    - 柱面数为$=\left(外直径-内直径\right)/2\times10\times m\times D_t+1$
    - 反正不知什么原因，要加 1
  - 数据传输率
    - 格式化数据传输率$f_n=B_sS_t/t$ (用扇区算)
    - 非格式化数据传输率$f_f=D_b\pi D_{min}/t$ (用密度算)
  - 磁盘容量：磁盘表面所能记录二进制信息的总量
    - 单位：B、MB、GB
    - 格式化容量：格式化后所能记录的信息量（用扇区算）
      - $C_f=B_sS_tmn$
    - 非格式化容量：完全从记录密度考虑的容量（用密度算）
      - $C_f=ftmn=D_b\pi D_{min}mn$
    - 提高面密度可以增大磁盘容量
      - 提高道密度：增加磁道数目
      - 提高平均位密度：不同磁道的扇区数不同，增加扇区总数
  - 存取时间：磁头从当前所在位置移动到目标扇区，并完成读写所需要的时间
    - 寻道时间t_S：磁头移动到目的磁道 （一般给出）
    - 旋转等待时间t_W：磁头定位到目标扇区
      - $t_W=1/2r$，r 为转速，单位 rps；若单位是 rpm，自行换算

        - 最大一圈，最小不动，取个平均值 1/2
    - 数据传输时间t_{WR}：如名字所言
        - $t_{WR} = b / f_n$，b 为要传输的数据量
    - 平均存取时间=寻道+旋转等待+数据传输
    - 连续读几个扇区和随机读几个扇区是有区别的
        - 连续读仅一次寻道和旋转等待，如果是从 0 扇区开始，甚至旋转等待也不需要
        - 随机读要多次寻道和旋转等待
- 误码率：读出数据出错位数占总位数的比例
    - 软错误：即随机错误（偶然性错误），要求误码率$< 10^{-9}$。比如蓝屏，因为内存信息位突然反转导致
    - 影错误：即突发性错误（永久性错误），要求误码率$< 10^{-12}$。如扇区损坏
- 平均存取时间主要开销在寻道和旋转延迟
- 磁盘格式化：依记录格式要求，对磁道划分扇区，留出各种间隙，并写入各种标志信息、地址信息和其他控制信息的过程
    - 留有间隙，是为了区分不同扇区和扇区内不同信息区；为同步、伺服定位、读地址等操作提供缓冲时间；留出控制余量，以便在环境、电压变化或更换盘组时确保可靠读出

6.3 其他存储

- RAID，廉价磁盘冗余阵列。将多个具有独立操作的廉价磁盘按某种方式组织成一个磁盘阵列，以增加容量，利用类似于主存中的多体交叉技术，将数据存储在多个盘体上，让这些盘并行工作来提高数据传输速度，并采用冗余磁盘技术来进行错误恢复以提高系统可靠性
    - 解决存储容量、读写速度、可靠性问题
    - RAID 是一组物理盘，而操作系统视其为单个逻辑盘
    - 由于技术发展磁盘价格降低，RAID 中的 I 不光视为廉价，也可视为独立
- RAID 具体分类（具体的还是看 ppt 图片加深理解记忆）
    - RAID0，条带化
    - RAID1，镜像盘实现 1+1 冗余
    - RAID2，条带化+使用海明码校验
    - RAID3，奇偶校验法生成单个冗余盘，条区增大为 KB，提高吞吐率

- RAID4，独立存取技术，使用更大的条区，各个盘独立互相访问，并发 I/O
- RAID5，奇偶校验块分布在各个磁盘中，各个盘相互独立的并发访问
  - 兼顾存储性能、可靠性、存储成本，一般采取 5 个或 9 个盘组合
  - 可以对单盘失效进行恢复
- RAID6，两个独立的奇偶校验块
  - 数据可靠性非常高，允许两个盘同时出错
  - 但开销大、写性能差、控制方式复杂
- RAID1-RAID5：广泛用于服务器
- RAID0-RAID1：同时拥有 RAID0 的速度和 RAID1 的可靠，用于银行、金融等
- RAID3：适用于大量顺序数据访问
- RAID4：校验盘成为瓶颈，应用少

- 闪存：一种电可擦写、可编程只读存储器（E^2PROM）
  - NOR Flash：随机的，可直接按字节访问；快速随机读；慢速写或擦除；主要用于存储程序代码
  - NAND Flash：块级 I/O 访问，更高集成度（大容量成本低），较好的擦除写性能，主要用于存储数据
- 闪存单元由一个带浮栅的晶体管构成，该晶体管的阈值电压可通过在其栅极上释加电场而被反复改变。若浮空栅上保存电荷，源漏极之间形成导电沟道，定义此时为“0”
- 损耗均衡技术：通过重映射，将写操作从擦写次数多的块转移到擦写次数少的块，以均衡对芯片内存储单元的擦写/磨损次数。虽然均衡损耗技术降低了闪存性能，但减少了块的损耗，提高了闪存可靠性。
- 闪存和 EEPROM 的比较
  - 相同点
    - 属于不挥发的存储器
    - 具有在线编程能力
  - 不同点
    - EEPROM 要求数据擦写的时候，必须整块芯片擦除和重新编程；而闪存允许以子块为存储单元写入或擦除，速度更快
    - EEPROM 寿命有限，仅几万-几十万次；闪存重新编程次数可达一百万次
    - EEPROM 的擦除、写、读取需要不同的电压；而闪存只需要单一电压，工作功率小
- 光存储技术：将激光聚集成极细光束在存储介质上存储信息，根据激光束和反射光的强弱不同，实现信息读写

- 光盘记录密度高，单片存储容量大，非接触式读写，光盘易于更换、便于保管，对环境条件没有苛刻要求
- 但寻道时间长，擦写性能不如磁盘快
- 常用于辅助存储、备份

- 光盘分类（按记录介质）
  - 形变型光盘
    - 记录层表面形状发生变化，如凹凸、发泡与否
    - 不可逆，用来当作只读光盘
  - 相变型光盘 PCD
    - 记录层发生相变，光学特性发生改变
    - 不可逆相变，只能写一次（追忆型光盘）
    - 可逆变，可写多次（可擦型光盘）
  - 磁光型光盘 MOD
    - 磁记录层的两种剩磁状态记录信息，可多次读写（可擦型光盘）
- 光盘分类（按存取方式）
  - 只读型（DVD-ROM、CD-ROM）
    - 只能顺序读出，不能擦写
    - 光道为由里向外螺旋线，扇区长度、位密度相等
  - 追记型（WORM、CD-R）
    - 只能写一次，一经写入不可更改
    - 光道为由里向外螺旋线（CD-R）或同心圆（WORM）
  - 可擦型（CD-R/W、DVD-R/W）
    - 可多次擦写可读
    - 光道为同心圆
- 光盘存储器的构成
  - 光盘盘片：记录信息的基体
  - 光盘驱动器 ODD
    - 电机驱动机构、定位机构、光头装置、逻辑电路
    - 驱动光盘转稳转准，寻找光道并借助光头和激光器完成读写操作
  - 光盘控制器 ODC
    - 数据输入输出缓冲器、编码器、译码器、串并转换电路
    - 通过 I/O 接口实现主机与 ODD 信息交换，控制 ODD 驱动盘片旋转、寻道、进行读写操作

- 提高光盘存储密度的途径就是要缩短激光的波长、增大光学系统的孔径
- 全息光存储技术：数据以全息图的形式记录在材料里

6.4 IO 控制方式

- 程序查询，和 CPU 串行，其他方式都是并行
- 中断
  - 中断源：引起处理机中断
  - 分类
    - 内中断：内部硬件和程序产生。硬件故障中断、陷阱；外中断：主机外部部件引起，IO 外设、控制台干预机器、时钟、外部信号
    - 可屏蔽、不可屏蔽
- 多级中断下的流程
  - 中断隐指令：CPU 响应中断后，暂停当前的程序
    - 保存断点、状态
    - 关中断
    - 转向中断服务程序
  - 中断服务流程
    - 保护现场
    - 屏蔽低级中断
    - 开中断
    - 执行中断源处理程序
    - 关中断
    - 恢复现场
    - 开中断
    - 返回被暂停程序
  - 根据是否支持多级中断，中断服务流程还会有些步骤差别
- CPU 响应中断的三个条件：中断源有中断请求、CPU 开中断、CPU 在一条指令执行完毕且没有更紧迫任务
- DMA：IO 和内存之间开辟直接的数据通路。仅开始和结束需要 CPU
  - 独立于处理器，在高速外设和主存之间直接传输数据
  - 由专门的硬件（DMA 控制器）控制总线进行传输
  - 请求-响应方式

        - 每当高速设备准备好数据，就进行一次 DMA 请求，DMA 控制器接收到 DMA 请求后，申请总线使用权
    - DMA 的请求优先级高于中断，由于数据成组传输，如果不及时处理将丢失信息
- DMA 数据传输方法
    - CPU 停止法（成组传送）
        - DMA 传输时，CPU 脱离总线，停止访存，直到 DMA 传送一块数据结束
        - 优点：控制简单，时候传输率很高的外设
        - 缺点：CPU 工作受影响，DMA 访存时 CPU 基本处于停止状元；主存周期没有被充分利用，两个数据间的准备间隔总大于一个存储周期，所以主存周期没有被充分利用
    - 周期挪用（窃取）法（单字传送）
        - DMA 传输时，CPU 让出一个总线事务周期，由 DMA 控制总线完成访存传送完一个数据后立即释放总线
        - 优点：及时响应 IO 请求，能较好的发挥 CPU 和主存的效率。这种方式下，在下一数据的准备阶段，主存周期被充分利用
        - 缺点：每次 DMA 访存都要申请总线控制权、占用总线进行传输释放，增加开销
    - 交替分时访问法
        - 每个存储周期分为两个时间片，一个给 CPU，一个给 DMA。每个存储周期内，CPU 和 DMA 都可以访问存储器
        - 优点：适合 CPU 工作周期比主存存取周期长的情况，不需要总线使用权申请释放
        - 这种情况下 CPU 既不停止主程序运行，也不进入等待状态，在 CPU 工作过程中不知不觉完成 DMA 数据传输，被称为“透明 DMA 方式”
- IO 设备要求 DMA 传送时的三种情况
    - CPU 不需要访问主存：不会冲突，两者并行
    - CPU 正在访存：必须等待存储周期结束，CPU 让出总线 DMA 才可以访存
    - CPU 同时也要访存：先让 DMA 占用总线，窃取一个主存周期，完成一个数据交换，这样 CPU 会延迟一段时间进行访存

	中断方式	DMA 方式
数据传输	程序控制	硬件控制
响应时间	指令执行结束	存取周期结束
处理异常情况	能	不能

中断请求	传送数据	结束处理
优先级	低	高
IO 设备	低、慢速	高速设备

- 通道：IO 通道是专门负责输入输出的处理机，是 DMA 的发展。硬件机制
  - 是一个特殊的处理器，有自己的指令（通道命令）和程序，专门负责数据输入输出的传输控制；把外围设备的管理从 CPU 分离出来
  - 特点：具有系统总线和通道总线；分为字节多路通道（低俗）、数组多路通道（高速）、选择通道（高速）
  - CPU 与通道并行；数据传送由通道完成，但 CPU 要传送相关的控制字、指令
- 外围：IO 处理机。采用具有丰富指令系统和相当容量主存的小型机作 IO 控制，除了完成通道的全部功能，还能完成数据库管理、运算功能、程序控制等
  - CPU 与外围并行；数据传送由外围完成，CPU 完全从 IO 事务中解脱

第七章 总线

7.1 总线概述

- 拓扑结构：把计算机各个部件或模块称为结点，结点的位置及其互连的几何布局称为拓扑结构。拓扑结构有这么几种：星型、树型、环型、总线型、交叉开关
- 总线：连接两个以上设备或部件的信息通路、是部件或设备间的共享传输介质
- 总线优点：成本低，一组线路可以被多种设备共享；可扩展性好，易于增加新设备
- 总线传输方式：串行、并行
- 总线速度受限于：总线长度（越长延迟越大）、总线设备数（越多冲突/竞争越多）
- 总线分类：
  - 片内总线：连接芯片内的各个元件单位（如连接 ALU、寄存器、指令部件等）
  - 系统总线：连接计算机系统主要功能部件（如连接 CPU、主存、I/O 控制器）
    - 单总线结构：将 CPU、主存、I/O 适配卡通过底板总线互连，底板总线称为标准总线
      - 多设备竞争总线、多种异速设备连接在一条总线上，性能受限、扩展能力差
    - 多总线结构：根据部件、设备速度的差异不同，设置多条总线，如局部总线、处理器-主存总线、高速 I/O 总线、扩展 I/O 总线
      - 存储总线：Processor-Memory Bus，连线短速度快，仅需与主存匹配，使 CPU-主存之间达到最大带宽
      - I/O 总线；I/O Bus，连线长且速度慢，需要适应多种 I/O 设备，一侧连接到 Processor-Memory Bus 或 Backplane Bus，另一侧连接 I/O 控制器
      - 改善了设备对总线的使用竞争、增加系统外接设备数量、允许速度差异大的设备连接在同一条系统总线上
    - 通信总线：主机和 I/O 设备之间或计算机系统之间提高连接
- 系统总线的构成：
  - 控制线：控制对地震学和数据线的访问使用，用来传输定时信号和命令信号
    - 时钟：用于总线同步
    - 复位：初始化所有设备
    - 总线请求：表明发出该请求信号的设备要使用总线

        - 总线允许：表明接收到该允许信号的设备可以使用总线
        - 中断请求：表明某个中断正在接受请求
        - 中断回答：表明某个中断请求已被接受
        - 存储器读：从指定主存单元中读取数据到数据总线上
        - 存储器写：数据总线上的数据写到指定主存单元中
        - I/O 读：从指定的 I/O 端口中读数据到数据总线上
        - I/O 写：将数据总线上的数据写到指定的 I/O 端口中
        - 传输确认：表明数据已被接收或已被送到总线
    - 地址线：承载源部件和目的部件之间的数据传输，数据线的宽度反映一次能传送的数据位数
    - 数据线：给出源数据或目的数据所在的主存单元或 I/O 端口的地址。地址线的宽度反映最大的寻址空间
- 有些总线没有提供单独的地址线，地址信息也是通过数据线来传送，称为数据/地址复用
- 总线上的三个主要部件
    - 总线设备
        - 按总线使用权：
            - 总线主设备：能够申请并获得总线控制权的设备，具有控制总线的能力，发起总线事务。如 CPU 是总线主设备
            - 总线从设备：不具有申请总线使用权的设备，被总线事务激活的模块或设备。如存储器模块
        - 按设备传送数据方向：总线源设备（发）、总线目标设备（收）
        - 按设备的访问方式：
            - 存储器设备：使用访问存储器的方法访问的设备，访存型总线指令。如主存储器
            - I/O 设备：使用访问 I/O 的方法访问的设备，I/O 型总线指令。如磁盘
    - 设备接口：连接设备与总线的桥梁，完成设备信号和总线信号之间的协调和转换
        - 总线接口在总线主设备角度的工作
            - 总线使用请求信号->总线使用应答信号->使用总线
            - 中断请求信号->中断响应信号和中断向量->等待 CPU 处理
            - DMA 请求信号->DMA 应答信号->数据交换->撤销 DMA 请求信号
        - 总线接口在总线从设备角度的工作
            - 译码器：选择从设备
            - 译码器命中后，则按照总线命令的指示进行总线操作，发送/接收数据
        - 总线设备接口对总线设备具有的处理能力

        - 总线使用请求
        - 总线使用应答
        - 存储器操作
        - I/O 操作
        - 读操作
        - 写操作
        - 中断信号
        - DMA 信号
    - 总线控制器：总线系统的核心，控制设备应答顺序，管理总线的使用，实现总线协议。
        - 总线系统资源的管理：对存储空间、设备端口空间、通道、中断等进行分配启动操作
        - 总线系统的定时：产生总线时序和总线命令
        - 总线的仲裁：确定哪个主设备获总线使用权
        - 总线的连接：不同总线协议之间的转换、完成总线之间的连接
- 总线特性
    - 机械特性：尺寸、形状
    - 电气特性：传输方向和有效的电平范围
    - 功能特性：每根传输线的功能
    - 时间特性：信号的时序功能

7.2 性能指标

- 总线性能指标（摘自王道）
    - 传输周期：一次总线操作所需要的时间（包括申请阶段、寻址阶段、传输阶段、结束阶段），简称总线周期。总线传输周期由若干个总线时钟周期构成
    - 工作频率：总线上各种操作的频率，为总线周期/传输周期的倒数，指一秒内传送几次数据。
        - 总线周期=N 个时钟周期
        - 总线工作频率=时钟频率/N
    - 时钟周期：即机器的时钟周期
    - 时钟频率：机器的时钟频率，时钟周期的倒数
    - 宽度：亦总线位宽，总线上能同时传输数据的位数，通常指数据总线的根数

- 带宽：总线的数据传输率，单位时间内总线上可传输数据的位数，通常用每秒钟传送信息的字节数衡量，单位字节/秒。
  - 总线带宽=总线工作频率×（总线宽度/8）
- 复用：一种信号线在不同时间传输不同的信息，节省空间、成本
- 信号线数：地址总线、数据总线、控制总线 3 种总线数的和

- 总线最主要的性能指标时宽度、工作频率、带宽
- 增加同步总线带宽的措施
  - 提高时钟频率
  - 增加数据线宽度
    - 能同时传送多位
    - 增大了成本
  - 允许大数据块传送
    - 背对背总线周期，也称突发传输方式
    - 只要开始送一次地址，后面连续送数据
    - 增加复杂性、延长响应时间
  - 拆分总线事务
    - 一次总线事务时间延长，但整个系统带宽增加
    - 增加复杂性、延长响应时间
  - 不采用分时复用方式
    - 地址和数据可以同时送出
- 增加成本，增加复杂性

7.3 总线设计

- 总线设计时要考虑的因素

选项	高性能	低成本
总线宽度	分离数据和地址线	地址/数据复用
数据宽度	越宽越快	越窄越便宜
传输大小	多字传输的总线开销小	单字传输的简单
总线主设备	多个（需要仲裁）	一个（无需仲裁）
分离事务处理	是	否
时序	同步	异步

- 总线设计基本要素
  - 总线带宽：单位时间内在总线上传输的最大数据量

- 信号线类型：复用、专用
- 仲裁方法：集中式、分布式
- 定时方式：同步通信、异步通信
- 事务类型：总线所支持的各种数据传输类型和其他总线操作类型，如存储器读写、IO 读写、读指令、中断响应

- 总线仲裁：多个设备需要使用总线来进行通信时，采用某种策略选择一个设备使用总线。总线被连接在其上的所有设备所共享。如果没有任何控制，那么多个设备需要进行通信时，每个设备都试图为各自的传输把信号送到总线上，这样就会产生混乱，所以需要仲裁
  - 避免总线混乱的方法
    - 在总线中引入一个或多个主设备，只有主设备才能控制总线
    - 利用总线仲裁决定哪个总线主设备下次会获得总线使用权
- 仲裁的分类
  - 按仲裁电路位置不同：集中、分布
  - 按设备请求进行仲裁的优先权是否变化：固定优先权、动态优先权
  - 按仲裁算法还是查询：设备请求、控制器查询
  - 按仲裁电路同时请求处理数量：串行、并行
- 部分仲裁方式详解
  - 集中式：将控制逻辑专门做在一个专用的总线控制器或仲裁器中，将所有的总线请求集中用一个特定的仲裁算法进行仲裁
    - 菊花链：串行查询方式工作。总线允许信号 Grant 从最高优先权的设备向最低优先权的设备传递，如果到达的设备有总线请求，则 Grant 信号不再往下传，该设备建立总线忙 busy 信号，表示它已获得总线使用权
      - 简单，只需一根线就能按一定次序实现总线裁决；易扩充设备
      - 不能保证公平性、对电路故障很敏感、菊花链仲裁限制了总线性能
    - 计数器定时查询：比菊花链查询多一组设备线，总线控制器接收到总线请求信号后，在总线未被使用的情况下，由计数器开始计数，并将计数值通过设备线向各设备发出。当某个有总线请求的设备号与计数值一致时，该设备即获得总线使用权，此时终止计数查询，同时该设备建立总线忙信号
      - 灵活，设备优先级可以通过设置不同的计数初值来改变，若每次初值皆为 0，则是固定的优先级方式。若每次初值总是刚获得总线使用权的设备，则是平等的循环优先级方式；对电路故障不如菊花链敏感
      - 需要增加一组设备线；总线设备控制逻辑复杂（需要译码、比较）

- 独立请求：每个设备都有一对总线请求线 Req 和总线允许线 Grant，当某个设备需要使用总线的时候，通过对应总线请求线将请求信号送到总线控制器。总线控制器有一个判优电路，可根据设备优先级来确定选择哪个设备。控制器可采用固定优先级，也可以编程设置优先级
  - 响应速度快；若可编程，优先级灵活
  - 控制逻辑复杂，控制线多
- 分布式：分布式没有专门的总线控制器，其控制逻辑分布在各个部件或设备中
  - 自举式
    - 优先级固定，各设备独立决定是否是请求者中优先级最高的那一个
    - 请求总线的设备在各自对应总线请求线上送出请求信号
    - 在总线仲裁期间，每个设备将比自己优先级高的请求线上的信号取回分析。若有总线请求信号，则本设备不能立即使用总线；若没有，则可立即使用总线，并通过总线忙信号阻止其他设备使用总线
  - 冲突检测
    - 当某个设备要使用总线时，首先检查一下有没有其他设备也在使用总线，如果没有，就置总线忙，使用总线
    - 若两个设备同时检测到总线空闲，则可能同时使用总线，会发生冲突。一个设备在传输过程中，它会侦听总线以检测是否发生了冲突；当冲突发生时，两个设备都停止传输，延迟一个随机时间后再重新使用总线
    - 一般用在网络通信上

- 总线通信的定时方式
  - 同步：用时钟来同步定时（处理器总线都会采用同步方式）
    - 以时钟为基础的协议：最简单的同步协议。控制线上有一个时钟信号进行定时，有确定的通信协议。
      - 控制逻辑少而速度快
      - 所有设备在同一时钟速率下运行，以最慢的设备为准；由于时钟偏移问题，同步总线不能很长
  - 异步：用握手信号定时（只有 I/O 总线才会使用）
    - 没有公共的时钟标准，因此能连接带宽范围很大的各种设备，总线能够加长而不用担心时钟偏移问题
    - 采用握手协议，即应答方式。只有双方都同意时，发送者或接收者才会进行下一步，协议通过附加的一对握手信号线实现
    - 三种方式：护士（主设备）递刀（发请求）给医生（从设备）

        - 全互锁：确认医生拿到（发回答）再松手，确认护手松手（撤请求）再拿走（撤回答）
        - 半互锁：确认医生拿到（发回答）再松手，不管护士松没松手（撤请求）都拿走（撤回答）
        - 非互锁：不管医生拿没拿到（发回答）都松手，医生不管护士松没松手（撤请求）都拿走（撤回答）
    - 灵活，可挂接不同工作速度的设备
    - 对噪声敏感（任何时候都能接收对方的应答信号），接口逻辑复杂
  - 半同步：同步和异步结合（I/O 总线大半采用）
    - 解决异步方式对噪声敏感的问题，引入时钟、就绪、应答和握手信号
    - 信号在时钟的上升沿有效，信号有效时间限制在时钟到达的时刻，不受其他时间的信号干扰
    - 从设备通过 wait 信号告知主设备何时数据有效
    - 结合了同步和异步的优点：所有信号都由时钟定+不同速度设备共存于总线
  - 拆分事务：从设备准备数据时，释放总线；可以提高总线的有效带宽
    - 一个事务拆分为两个阶段。改善系统效率；但单个事务响应时间变长、增加复杂性
    - 过程 1：主设备 A 获得总线使用权后，将请求的事务类型、地址及其其他信息发到总线，从设备 B 记录下这些信息。A 发完信息后便立即释放总线，其他设备便可使用总线
    - 过程 2：B 收到 A 发来的信息后，按照 A 的要求准备数据，准备好后，B 便请求使用总线，获得总线使用权后，B 将 A 的编号及所需数据传到总线，A 便可接收
- 请求应答方式和拆分总线事务方式的区别
  - 请求应答
    - CPU 启动一次读或写事务
      - 传送信息：address、data、command
    - 然后等待存储器应答
  - 拆分总线事务
    - CPU 启动一次读或写事务后，释放总线
      - 传送信息：address、data（write）、command
    - 存储器启动一次应答事务，请求使用总线
      - 传送信息：data（read）or acknowledge（write）

- 仲裁策略的两个因素
  - 等级性：具有高优先级的设备应该先被服务
  - 公平性：最低优先级的设备不能永远都得不到总线使用权

参考文献

[1] 袁春风. 计算机组成与系统结构[M]. 北京: 清华大学出版社, 2010.

[2] 熊岳山. 数据结构与算法（第 2 版）[M]. 北京: 清华大学出版社, 2013.

[3] 王道论坛. 2019 年计算机组成原理考研复习指导[M]. 北京: 电子工业出版社, 2018.

[4] 王道论坛. 2019 年计算机数据结构考研复习指导[M]. 北京: 电子工业出版社, 2018.

[5] 王保恒, 肖晓强, 张春元, 文梅. 计算机原理与设计[M]. 北京: 高等教育出版社, 2005.

[6] 唐玉华等. 计算机原理[CP/OL]. 湖南: 国防科技大学, 2007. https://www.icourses.cn/sCourse/course_3267.html.

[7] 刘芳, 沈力, 陈微等. 计算机原理[CP/OL]. 湖南: 国防科技大学, 2016. https://www.icourse163.org/course/NUDT-359002.

[8] [美]Randal E.Bryant, David R.O'Hallaron. 深入理解计算机系统[M]. 龚奕利, 贺莲等译. 北京：机械工业出版社, 2016.

[9] [美]David A.Patterson, John L.Hennessy. 计算机组成与设计软硬件接口[M]. 王党徽, 康继昌, 安建峰等译. 北京: 机械工业出版社, 2015.