南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

计算机组成原理期末 20 问题整理

期末问题迭代组织

冯思程

年级：2021 级

专业：计算机科学与技术

指导教师：张金

2023 年 6 月 21 日

目录

一、 主体

(一) 系统设计（总体设计）

1. 目标

目标需要重点关注性能评价，其中包括两个方面，分别是时间和空间。

时间 这里个人电脑主要关注的是响应时间。

性能 =1/执行时间性能比例是 X 比 Y 快 n 倍：性能 X/性能 Y= 执行时间 Y/执行时间 X=n

CPUtime= 时钟周期数 * 时钟周期长 =cycles*T

不同的机器指令需要的周期数不同

所以在计算 CPU 执行时间的时钟周期数是一个平均值：平均每条指令所需要的时钟周期数 = 总时钟周期数/总指令数

总时钟周期数 = 总指令数 * 平均每条指令所需要的时钟周期数 =IC*CPI

时钟频率 =1/时钟周期长 =1/T=f

CPUtime=IC*CPI*T=IC*CPI/f（f：时钟周期频率（主频）一般单位是 GHz（109））

给出一个例题：某程序在一台时钟频率为 2GHz 的计算机上运行需要 10 秒，现在将设计一台计算机 B, 希望将运行时间缩短为 6 秒，计算机的设计者采用的方法是提高时钟频率，但这会影响 CPU 余部分的设计，使计算机运行该程序时需要相当千计算机 1.2 倍的时钟周期数，那么计算机设计者应该将时钟频率提高到多少?

答：利用 CPUtime=IC*CPI/f 公式，由于本题中 CPI 的值是不变的，所以可以列出下列方程:

cycles/2×6 = 10 × 1.2$cycles$/T'

解得 T'=4GHz

空间 这里更加关注吞吐率的并行化,要掌握吞吐率的计算方法,注意吞吐率的单位是条指令/秒。

2. 困境

在设计过程中，遇到了困境：功耗墙。

功耗：CMOS 是占统治地位的集成电路技术

晶体管的能耗主要来自动态消耗

公式：功耗 =1/2* 负载电容 * 电压 2* 开关频率（与主频相关）

功率墙：由于功耗问题不能再继续提升处理器的速度。

功率墙面前摩尔定律没有失效。

功率墙的成因：由于功耗达到极限，已经无法再使处理器冷却下来。

影响因素：电压、开关频率（与主频相关）、负载电容

解决办法：降低电压，连接大设备以冷却、将芯片中在一些时钟周期不用的部分关闭、单处理器到多核处理器（并行编程）

3. 原则

这里是整本教材的重中之重，也是整体系统设计的核心思想，八大原则：

- 1）两个设计原则：面向摩尔定律的设计、使用抽象简化设计

- 2）四个提高性能的方法：加速大概率事件、并行（多核处理器）、流水线、预测
- 3）存储器层次（速度最快、容量最小并且价格最昂贵的存储器位于顶层，速度最慢、容量最大、价格最便宜的存储器处于最底层。）
- 4）冗余提高可靠性，类似车上的备胎

给出一道例题：设计航空航天中使用的计算机，针对这一特殊应用场景，结合八大原则谈谈如何设计。

答：设计航空航天中使用的计算机，我们需要特别考虑其工作环境的严酷，如极端温度、辐射、高压力等，同时也要考虑到空间环境中可能出现的异常和故障。同时还需要考虑到其对计算准确度的超高要求。

面向摩尔定律的设计：我们将持续优化处理器的性能，提高集成度，以便获取更多的计算能力。

使用抽象简化设计：为了使设计更为高效和可控，我们将使用抽象概念来简化硬件和软件设计。

加速大概率事件：对常见的指令和计算路径进行优化，确保这些大概率事件能得到快速处理。

利用并行、流水线、预测提升性能：利用已知的可靠技术进行性能上的优化，但是需要注意这里设计不追求极致的性能，而是更加关注可靠性、稳定性。

设计好的存储器层次：我们将设计一个优化的存储层次结构，包括寄存器、缓存、主存和硬盘等多个层次。每个层次的速度和容量都会有所不同，我们需要在性能和成本之间找到最佳的平衡。

利用冗余提高可靠性：考虑到航空航天应用对可靠性的高要求，我们将设计冗余系统，以便在主系统出现故障时仍能保持正常运行。同时由于可能面临的极端环境，需要增加冗余来确保在极端环境下也可以正常运行。

4. 切入点

- 1）构建式：流程（完整闭环）
- 2）改良式：流水线

(二) 指令设计（类比设计）

1. 路线选择

这里指令设计需要考虑指令集类型，分为 RISC 和 CISC。

RISC 指令集（简单性）：所有指令都是 32 位长（体现简单源于规整）

主要特征是：是精简指令集架构，用少数的指令组合实现新增的功能。例如：ARMv7、MIPS

CISC 指令集（复杂性）：不断加入新的指令来实现功能。例如 x86 指令集等。

这里学习的 MIPS 指令集是一种精简指令集。

2. 目标

- 1）功能上实现运算。
- 2）改进方面，利用流水线技术。

3. 原则

这里需要掌握指令设计的四大原则:

- 1）简单源于规整（所有指令都是 32 位）
- 2）越小越快（寄存器只有 32 个）
- 3）优秀的设计需要折中方案（虽然指令长度都是 32 位，但是指令格式有区别）
- 4）加速大概率事件（PC 相对寻址和立即数寻址）。

4. 指令集设计

从具体到一般，然后不断的进行迭代验证（功能是否完整），然后不断泛化验证，进行迭代，直到形成闭环。

三类汇编指令：运算指令、数据传送、决策指令:

1 运算指令：算术运算、逻辑运算（类似的指令都是三个操作数体现了简单源于规整）

逻辑按位运算：and、or、nor（或非，任何数据和 0 进行 nor 运算，都会 0/1 反转）指令

逻辑移位运算：sll 左（移位后右边空位补 0）、srl 右（可用来实现乘法）

2 数据传送指令（lw、sw 格式相同）:

lw 取数指令：从存储器复制数据到寄存器，例子：lw s0,5(s1) 把数组 a[5] 的数据放到 s0 寄存器。5*4 称为偏移量，存放基址的寄存器叫做基址寄存器。

寄存器之间的数据传送（move 伪指令：move s1,t0）

装载立即数到寄存器（利用 addi 将立即数和零寄存器相加）

取立即数伪指令：li，装载 32 位立即数到寄存器：先用 lui 指令把立即数的高 16 位放到寄存器，然后用 ori 将立即数低 16 位与寄存器进行或运算。（注意不能用 addi 代替 ori 运算）

3 决策指令：条件分支 beq 和 bne

Beq（相同则分支）、Bne（不同则分支）不发生分支则继续指令内存中相邻的下一条指令。

小于则置位 slt/slti/sltu，sltu 用于无符号数。置位：设置成 1；复位：设置成 0

例子：slt t0,s0,s1，若 s0 寄存器小于 s1 寄存器则将 t0 寄存器置为 1

通过 slt、beq、bne 指令的组合，可以实现全部比较条件，例如：通过 slt 然后将置的结果和 0 进行比较就可以实现分支判断。

5. 指令间设计

指令间设计是一个单条指令到多条指令的过渡形成问题。

寻址方式

- 1）R 型：寄存器寻址：运算指令（加减、逻辑运算、移位运算、slt、jr）
- 2）I 型：立即数寻址和基址偏移寻址（第三个操作数是常数的指令）（addi、ori、lui）
- 3）I 型：lw 和 sw 是基址偏移寻址。
- 4）I 型：PC 相对寻址（beq 和 bne）以 PC+ 为基准相加的字地址数目。（分支前后 128KB）分支 32 位地址 =PC+4+ 字节地址偏移量（beq 和 bne 的操作数要乘 4）（字节地址）。

- 5）J 型：伪直接寻址，只有操作码和目标地址两个字段，寻址时候，先将 26 位地址左移 2 位变成 28 位字节地址，再和 PC 的高 4 位拼接成 32 位地址。（和 PC 高四位相同的一切地址，256MB 的地址块）
- 6）分支到更远距离，可以将 beq/bne 取反，下接一条可能绕过的指令，书上例题（P78）跳转到更远距离，可以先将 32 位地址装载到某临时寄存器（用 lui 和 ori 指令组合装载），再用 jr 指令。

程序执行

- 1）编译器将高级语言翻译成汇编语言（编译）
- 2）汇编器先将伪指令替换成等价的真正指令，再将汇编语言翻译成机器语言（汇编）
- 3）链接器将目标文件和静态链接库和动态链接库拼接成可执行文件（链接）
- 4）加载器将可执行文件放入内存，装载执行。（加载）

整体流程如下图：

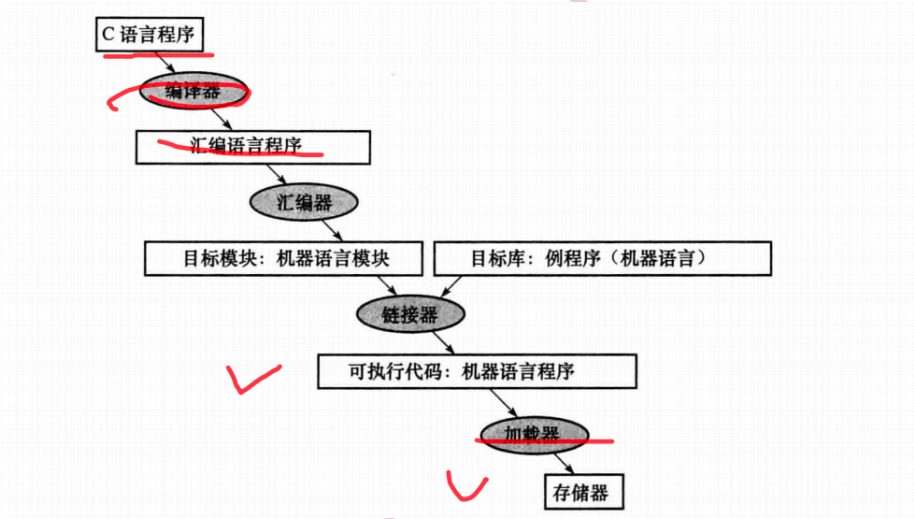

图 1: 程序执行图

(三) 部件设计（类比设计）

1. 优化

这里优化有两个方面：降本（降低成本）和增效（增加效率）。将 64 位计算器换成 32 位计算器

2. 浮点数

单精度浮点数表示格式：一位符号位、8 位指数域，23 位尾数。单精度浮点表示范围：小数点左边一直都是一个 1，小数点右边是 23 位尾数，范围是从全 0 到全 1，指数有 8 位，但是只有 1-254 是有效的，所有 1-254 分别对应-126 到 127，在计算的时候需要注意。表示范围是：

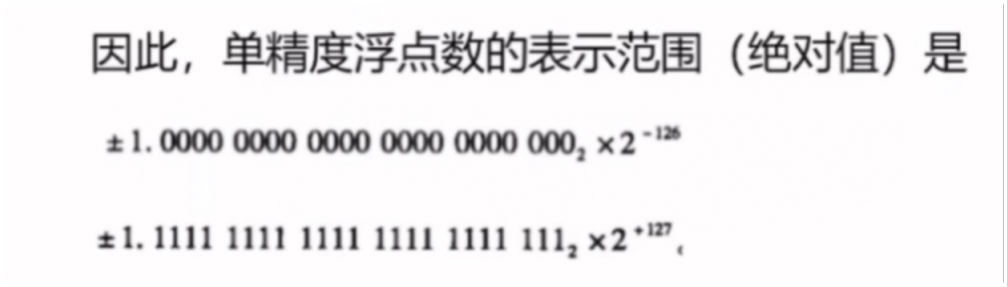

图 2: 单精度浮点数表示范围

双精度浮点数表示格式：一位符号位、11 位指数域，52 位尾数。单精度浮点表示范围：小数点左边一直都是一个 1，小数点右边是 52 位尾数，范围是从全 0 到全 1，指数有 11 位，但是只有 1-2046 是有效的，所有 1-2046 分别对应-1022 到 1023，在计算的时候需要注意。表示范围是：

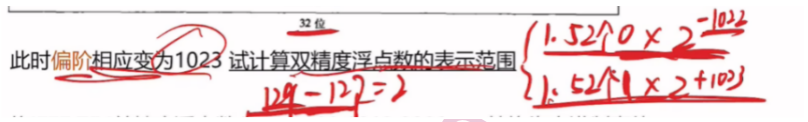

图 3: 双精度浮点数表示范围

浮点数加法：首先需要规格化，统一指数；有效数相加；规格化；按要求舍入。
浮点数乘法：首先指数相加；有效数相乘；规格化；舍入；判断符号

3. 浮点数的工程价值

- 1）为浮点数运算提供了标准
- 2）浮点数表示和计算一致性
- 3）精确的数值计算
- 4）可移植性和互操作性
- 5）数值稳定性和错误处理。

（四） 数字通路设计（常规改进设计和类比设计）

1. 常规方法

建立流程-优化流程-迭代解决问题

2. 单周期计算机的建立

一条 MIPS 指令的执行分为五个阶段-一个指令周期：

- 1）IF 取指令：根据 PC 给的地址，从存储器中取出指令。
- 2）ID 译码和读寄存器：分析指令字段，读取一个或者两个寄存器。
- 3）EX 运算：ALU 运算 R 型指令的结果/访存指令的地址/beq 两个源操作数是否相等。
- 4）MEM 访存与分支：访存指令向存储器进行读写，分支指令完成分支。

- 5）WB 写回：将结果送回寄存器。

下图是一个数据通路部件设计图：

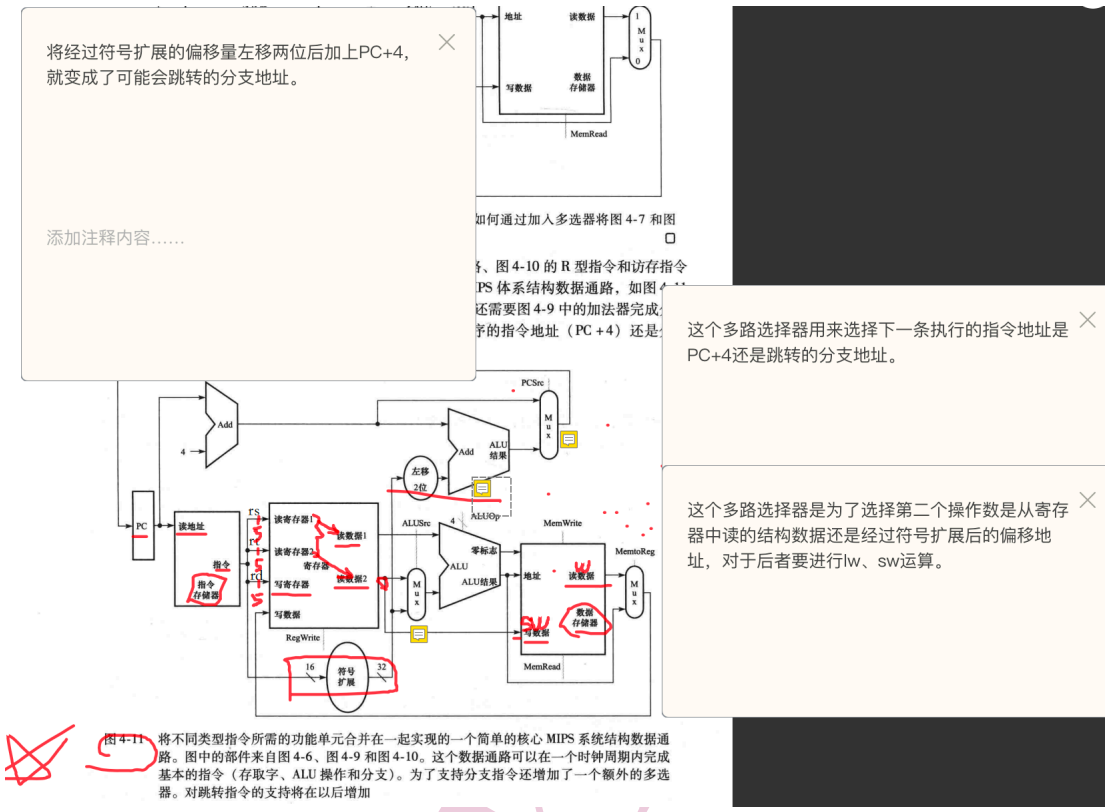

图 4: 数据通路部件图

在单周期计算机中，每条指令都在一个时钟周期内完成，CPI 是 1，在 MIPS 核心指令集中（9 条指令）只有 lw 指令需要完整的五个阶段，其他指令只需要 4 个阶段，但仍需要花费五个阶段的时间。

控制信号：当分支控制信号 Branch 和 ALU 零标志 Zero 同时为真，才将 PC 源控制信号 PCSrc 置为 1，选择将分支目标地址写回 PC，完成分支。

op 字段决定是 ALUOp 控制信号。ALUOp 和 6 位功能码通过多级译码的方式来控制 ALU（最后映射到 4 位控制码上）

下图是不同指令执行时的控制信号情况：

输入或输出	信号名	R 型	lw	sw	beq
输出	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

图 4-22 （续）

图 5: 控制信号总结图

3. 多周期计算机（类比设计）

评价：流水线的计算与设计 与指令周期的五个阶段相对应，把数据通路分为 5 个流水级，形成流水线。

公式：

时钟周期数 = 指令数 + 流水线级数-1

理想加速比 = 流水线级数

流水线的时钟周期要以五个阶段中最慢的阶段的时间为标准，确保任何一个指令的任何阶段都可以在一个流水线周期内完成。

流水线增加指令吞吐率来提高性能。（同一时间处理多条指令的不同阶段，实现指令级并行）理想情况下，流水线的 CPI=1，IC 不变。

可以进一步划分流水级，缩短时钟周期。减少 CPUtime，提高吞吐率。注意：过度划分会导致调度开销增大，分支性能下降，抵消性能提升。

困境：三种阻塞与解决办法

- 1）结构冒险：多个指令抢占同一流水级的硬件。

  解决办法：

  指令和数据存储器分开，否则会让时钟周期的 IF 和 MEM 冲突。

  数据缓存，将一些常用的数据存入缓存中，如果其中一个指令需要的数据已经在缓存中，那么这个指令就可以直接从缓存读取数据，而不必等待内存总线，这样就可以避免或减少结构冒险。

  指令拆分：这种策略是指在处理器设计中，将可能引起结构冒险的复杂指令拆分成几个更小的简单指令，这些简单指令可以单独地在不同的处理单元或在不同的时间进行处理。

- 2）数据冒险：即因无法提供指令执行所需数据而导致指令不能在预定的时钟周期内执行的情况。

  解决办法：旁路技术：当一个指令结果被需要时，在其可以被用的时候直接传到需要的位置，而不是等待当前指令执行完再去使用该结构。

  气泡技术：当 lw-use 冒险发生时候，旁路技术也无法及时为下一条指令提供需要的数据，这时候不得不将中间插入一个空指令，即一个气泡，阻塞执行一个周期，从下一个周期重新取指令执行下一条指令。

- 3）控制冒险：遇到分支指令后会发生。

  解决办法：

  第一种预测是总预测分支未发生，如果预测正确可以正常流水线，如果错误，则也要阻塞。

  第二种预测是预测一个分支发生，预测另一些分支不发生。

  一个比较新的技术是动态预测，观察上次该指令分支是否发生。

  还有一种是根据对以往的分支记录，根据历史记录进行预测。

  还有一种方法是延迟分支，还可以缩短分支的执行时间。

困境：异常与中断 异常：系统内部，打断程序正常执行，如溢出检测。

中断：来自处理器外部的溢出，如 I/O 设备请求、硬件故障。

处理机制：EPC 保存出错指令地址，然后去特定地址执行异常处理程序。

优化：流水线的优化方向

- 1）进一步深化分解指令执行过程，提高吞吐率。（超流水）
- 2）更高效的预测机制。
- 3）动态调度技术。
- 4）超长指令字
- 5）超标量

（五） 存储设计（组合设计）

用下面这幅图可以概括出存储器的层次：从 CPU 到磁盘：速度变慢、内存变大、价格变低。从 CPU 到磁盘：分别是寄存器、多级 cache、内存、磁盘或者闪存

1. 存储层次

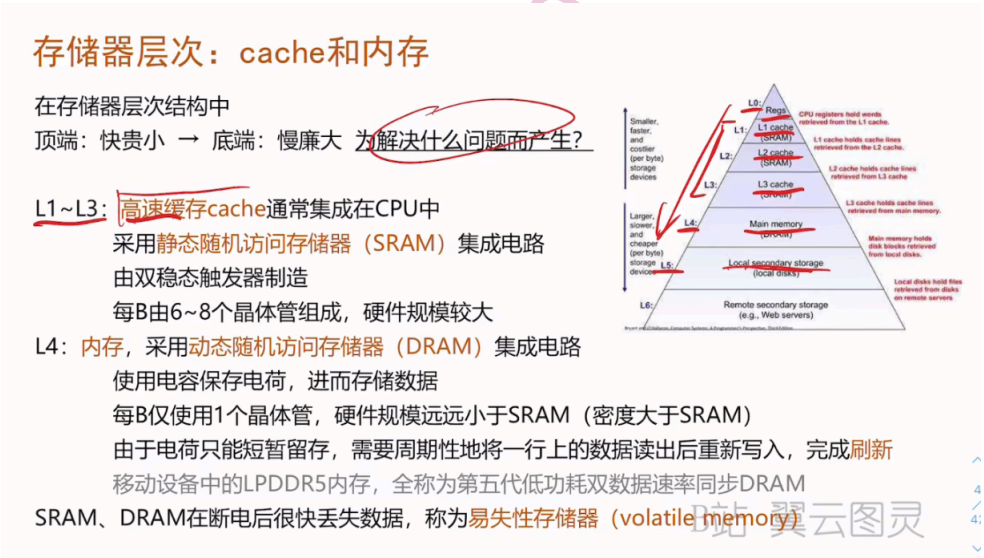

图 6: 存储层次图

2. 局部性原理：cache 建立、优化

cache 建立：Cache 的设计需要考虑以下几个主要参数：

- 1）大小：Cache 的大小直接影响其性能和成本。更大的 Cache 可以容纳更多的数据，从而提高命中率，但也会增加成本和复杂性。
- 2）块大小（Block/Line Size）：Cache 被分成称为块或行的一系列存储单元。当请求的数据不在 Cache 中时（即发生缺失），一个完整的块会从主内存中取出并存入 Cache 中。块的大小会影响 Cache 的空间利用率和命中率。
- 3）关联度（Associativity）：这是描述当新的数据块被加载到 Cache 中时，可以放置在哪些位置的参数。在直接映射（Direct-mapped）的 Cache 中，每个数据块只能放在一个特定的位置。在完全相联（Fully Associative）的 Cache 中，一个数据块可以放在 Cache 的任

何位置。还有一种折中方案，称为组相联（Set Associative）的 Cache，其中 Cache 被划分为若干组，新的数据块可以放在其对应组中的任何位置。

Cache 优化：

Cache 优化主要目标是提高命中率（hit rate，指请求的数据已在 Cache 中的概率）和降低缺失惩罚（miss penalty，指当请求的数据不在 Cache 中需要从主内存中取出的时间）。以下是一些常见的 Cache 优化技术：

- 1）高速缓存预取（Cache Prefetching）：当处理器从 Cache 中取数据时，会预先取出接下来可能要使用的数据。这可以预防由于处理器等待数据而产生的延迟。
- 2）优化替换算法（Replacement Policy）：这是决定当 Cache 满了以后，哪些数据应该被替换出去。常见的替换策略有最近最少使用（Least Recently Used, LRU）策略，最不经常使用（Least Frequently Used, LFU）策略等。
- 3）多级 Cache（Multi-Level Caches）：现代计算机系统通常有多级 Cache，比如 L1、L2、L3 Cache。这些 Cache 的大小和速度各不相同，通常离处理器越近的 Cache 越小且越快。多级 Cache 可以有效地平衡存储容量、成本和访问延迟。

3. 局部性原理：页表是磁盘的 cache、虚存

在计算机系统中，虚拟存储器、页表以及 TLB 都是与内存管理和地址翻译相关的重要概念。下面是对这三者的基本介绍：

虚拟存储器 (Virtual Memory)：虚拟存储器是一种内存管理技术。它使得应用程序认为它拥有连续的可用的内存，而实际上，它经常在硬盘和物理内存 RAM 之间交换数据。虚拟存储器的主要优势在于使得大型程序的运行不再受制于物理内存的大小，以及隔离各个应用程序，防止其操作对其他程序产生影响。

页表 (Page Table)：页表是虚拟内存系统中用于存储虚拟地址到物理地址映射的数据结构。每个进程都有一个独立的页表。在虚拟内存系统中，内存被划分为固定大小的页，每个虚拟页都可以映射到一个物理页。页表就是用来记录这种映射关系的。

TLB（快表）：TLB 是一种特殊的硬件缓存，用于改善虚拟地址到物理地址的转换过程。在虚拟存储器系统中，每次内存访问都需要进行地址转换，这将导致显著的性能下降。TLB 通过存储最近使用的虚拟地址到物理地址的映射来缩短地址转换时间。如果需要的映射在 TLB 中可以找到（称为 TLB 命中），那么可以快速完成地址转换；否则，就需要从页表中查找地址映射（称为 TLB 缺失），这会花费更多的时间。

4. 局部性原理：系统外部辅存

磁带库：这是一种数据存储设备，它使用磁带来存储大量的信息。它包含一个或多个驱动器，用于读取和写入磁带，以及一些机器人装置，用于在存储区域中移动磁带。磁带库通常用于长期存储和备份，因为它们提供了大量、廉价的存储空间，而且比硬盘更能耐受时间的考验。然而，磁带库的读写速度较慢，且不支持随机访问，通常只用于备份或归档数据，而不适合频繁访问的数据。

光盘塔：这是一种可以包含多个光盘驱动器（如 CD、DVD 或 Blu-ray 驱动器）的存储设备。它可以一次写入多个光盘，以备份数据或制作多个光盘副本。光盘塔也常常配备有硬盘，以便先将数据存储到硬盘，再从硬盘写入光盘。这种设备通常用于需要大量光盘副本的环境，如软件发布、音乐和电影制作等。虽然光盘的存储量相比磁带或硬盘较小，但光盘便于携带和分发，且对于长期存储来说，光盘的耐久性也很好。

RAID，全称为冗余独立磁盘阵列 (Redundant Array of Independent Disks)，是一种用于提高数据可靠性和性能的存储技术。RAID 将多个磁盘（通常是硬盘或固态硬盘）组合成一个逻辑单元，以实现数据的冗余存储、读写性能的提高或这两者的结合。

RAID 的主要级别包括：

- RAID 0：通过条带化技术提高磁盘的读写性能，但并不提供数据冗余。如果任何一个磁盘失败，所有数据都会丢失。

- RAID 1：通过镜像技术提供数据冗余，一种磁盘的完全备份方式，但需要两倍的磁盘空间。在单磁盘失效的情况下，系统可以无缝地从剩余的磁盘上恢复数据。

- RAID 5：使用奇偶校验技术来实现数据冗余，能够在单个磁盘故障时恢复数据。RAID 5 优化了存储空间的使用，但其写入性能相对较差。

- RAID 6：与 RAID 5 类似，但它使用两个奇偶校验块，可以容忍两个磁盘同时故障。

- RAID 10：结合了 RAID 1 的镜像和 RAID 0 的条带化，提供了数据冗余和性能提升。

每种 RAID 级别都有其适用场景，取决于对性能、数据安全性和存储成本的权衡。

5. 汉明码与奇偶校验

这部分我决定利用例题来进行说明，如下：

例题：假定存在某个单字节数据 10010010，首先写出对应的汉明纠错码，然后把第 10 位取反，说明纠错码如何找到并纠正该错误。（默认为偶校验）

答：首先将校验位空出来：___1__001__0010.

位置 1 检查 1、3、5、7、9、11 位，偶校验，该校验位是 1

位置 2 检查 2、3、6、7、10、11 位，偶校验，该校验位是 1

位置 3 检查 4、5、6、7、12 位，偶校验，该校验位是 1

位置 4 检查 8、9、10、11、12 位，偶校验，该校验位是 1

所以汉明码是 111100110010。把第 10 位取反得到 111100110110。重新计算 4 位校验位，同理上文分别是 1010，然后将两次的校验位从右向左读取，分别是 1111 和 0101，进行异或运算，是 1010，代表 10，说明是第 10 位发生了错误，改正就可。

(六) 谬误与陷阱

1. 第一章

陷阱：期望改进大小与总性能提高成正比，解释：改进后的执行时间＝受改进影响的执行时间/改进量＋不受影响的执行时间。

谬误：利用率低的计算机功率低，解释：利用率低不一定会功耗低。

谬误：面向性能的设计和面向能量效率的设计具有不相关的目标。解释：运行时间减少也可以减少整个系统的能耗。

陷阱：用一个性能公式的子集去度量性能，这是容易导致错误的，例如上文中的 MIPS 指标正比 IC/CPUtime。

2. 第二章

谬误：更强大的指令代表更强大的性能、使用汇编语言编程来获得最强的性能、商用计算机二进制兼容的重要性意味着成功的指令集不需要改变。

陷阱：地址要 +4、在自动变量的定义过程外，用指针指向该变量

3. 第三章

谬误：右移可以实现除法、并行执行策略不但适用于整型数据类型，同样也适用于浮点数据类型、只有理论数学家才会关心浮点精度

陷阱：浮点加法不能用结合律、MIPS 指令 addiu（无符号立即数加）会对 16 位立即数域进行符号扩展

4. 第四章

谬误：流水线是一种简单的结构、流水线概念的实现与工艺无关陷阱：没有考虑指令集的设计反过来会影响流水线。

5. 第五章

谬误：实际的磁盘故障率和规格书中声明的一致、操作系统是调度磁盘访问的最好地方、在不为虚拟化设计的指令集体系结构实现虚拟机监视器。

陷阱：在写程序或编译器生成代码时忽略存储系统的行为、在模拟 cache 的时候，忘记说明字节编址或者 cache 块大小、对于共享 cache, 组相联度少于核的数量或者共享该 cache 的线程数、用存储器平均访问时间来评估乱序处理器的存储器层次结构、通过在未分段地址空间的顶部增加段来扩展地址空间。

二、 参考资料

1. 计算机组成与设计：硬件软件接口，David A.Patterson、John L.Hennessy，2015

2. 张金老师的 20 个问题期末总结视频