纲要

2018年1月9日 15:41

1. 记得写公式和步骤
2. 应用题考点：
    a. 主存扩展，存储体地址分配（画图）
    b. 指令设计
        i. 分析指令格式的设计：
            1) 指令长度（几个字）
            2) 操作码个数
            3) 指令的操作数：个数、类型：是寄存器-寄存器类型或者其他
            4) 寄存器个数
            5) 寻址方式、个数
    c. 流水线性能评价（时空图）
    d. cache的性能评价
    e. 浮点数运算（流程图）
    f. ALU的设计（画图）
    g. 中断（画图）
    h. 相关性分析

3. 计算还需掌握：
    a. 浮点运算（IEEE 754)
    b. 补码运算
    c. 定点小数

4. 重点复习所有和计算有关的公式，待总结。
    a. Amdahl定律：
    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
    b. 加速比：
    加速比=系统性能（改进后）/系统性能（改进前）
    =总执行时间（改进前）/总执行时间（改进后）
    c. 流水线通过时间=流水级数*每一级通过时间+（级数-1）*每一级通过时间
    d. CPU性能公式：
    CPU 时间=指令数*CPI*时钟周期时间
    或
    CPU 时间=指令数*CPI/时钟频率

e. 流水线处理机的实际CPI：

理想流水线的CPI 加上各类停顿的时钟周期数：

CPI 流水线= CPI 理想+ 停顿结构冲突+ 停顿数据冲突+ 停顿控制冲突

f. 吞吐率

$TP=n/T_k$

n：任务数；Tk：处理完成n 个任务所用的时间

流水线的实际吞吐率和最大吞吐率TP=[n/(k+n-1)]Tpmax

k是流水级数

g. 效率：E=n/(k+n-1)

即流水线中的设备实际使用时间与整个运行时间的比值，即流水线设备的利用率。

h. CPU 时间 = (CPU 执行时钟周期数 + 存储器阻塞的时钟周期数) x 时钟周期

存储器阻塞时钟周期数 = 读操作引起阻塞的时钟周期数 + 写操作引起阻塞的时钟周期数

读操作阻塞的时钟周期数 = （读的次数 / 程序数）×读缺失率x 读缺失代价

对于写直达：写操作阻塞的时钟周期数 = [（写的次数 / 程序数）×写缺失率×写缺失代价] + 写缓冲区阻塞

存储器阻塞时钟周期数 = （存储器访问次数 / 程序数）×缺失率×缺失代价

也可以表示如下：

存储器阻塞时钟周期数 = （指令数 / 程序数）×（缺失数 / 指令） x 缺失代价

平均存储器访问时间（AMAT）作为检测cache 设计的方法：

AMAT = 命中时间 + 缺失率×缺失代价

i. 年失效率：

AFR = 某时间单位的一年的时间/相对应的时间单位的MTTF

MTBF = MTTF + MTTR

可用性 = MTTF/ (MTTF + MTTR)

一个系统的几个部件（缺一不可）的年失效率分别为A，B，C，则该系统的MTTF为1/（A+B+C）

j. 存储体带宽：

B = W/T

W为数据位数，T为存储周期

k. cache的命中率$h=N_c/(N_c+N_m)$ N_c为命中次数，N_m为失效次数

若 t_c 表示命中时的 cache 访问时间，t_m 表示未命中时的主存访问时间，$1-h$ 表示未命中率，则 cache/主存系统的平均访问时间 t_a 为：

$$t_a=ht_c+(1-h)t_m \tag{3.5}$$

我们追求的目标是，以较小的硬件代价使 cache/主存系统的平均访问时间 t_a 越接近 t_c 越好。设 $r=t_m/t_c$ 表示主存慢于 cache 的倍率，e 表示访问效率，则有

$$e=\frac{t_c}{t_a}=\frac{t_c}{ht_c+(1-h)t_m}=\frac{1}{h+(1-h)r}=\frac{1}{r+(1-r)h} \tag{3.6}$$

第一章

2018年1月8日 23:13

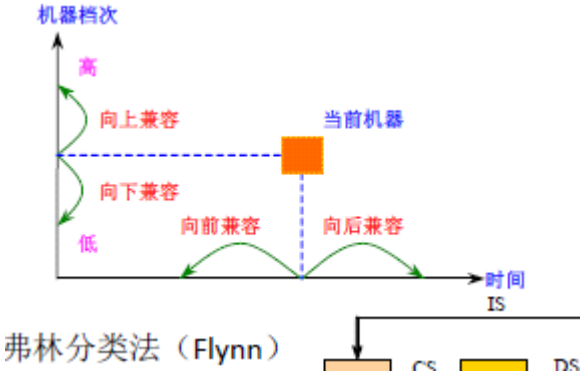
1. 计算机的分类、划代和应用特性
   早期：按功能（规模）、器件（电子管晶体管集成电路）、行业应用来划分
   现代（后PC 时代）：着重从应用特性来划分：桌面计算、服务器、嵌入式计算、个人移 动设备、集群/仓库级计算机（Clusters/Warehouse Scale Computer，WSC，通过云计算实现，软件即服务，Software as a Service，SaaS）
2. 计算机中的数据传输分为数据流、指令流（控制流）、地址流
3. 什么是程序块：
   单一入口和单一出口的程序
4. 八大思想
   1. 面向摩尔定律的设计
   2. 使用抽象简化设计
   3. 加速大概率事件
   4. 通过并行提高性能
   5. 通过流水线提高性能
   6. 通过预测提高性能
   7. 存储器层次：解决存储器容量、速度和价格三者矛盾，理论依据：程序局部性（空间和时间）
   8. 通过冗余提高可靠性：冗余编码、磁盘冗余阵
5. 计算机的设计要在CPU速度、主存容量、I/O带宽三方面平衡。
   每1M的CPU性能提升，需要1MB主存的提升和1Mb/s的带宽提升。
6. SPEC是基准测试程序，预测实际负载的性能，结果记为SPECratio，越大越好。
7. amdahl定律：
   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
8. 5大经典部件：运算器、控制器、存储器、输入和输出设备
9. 加速比的公式（两个）

$$加速比=\frac{系统性能_{改进后}}{系统性能_{改进前}}=\frac{总执行时间_{改进前}}{总执行时间_{改进后}}$$

   硬件设备：计算机系统中一切物理装置的总称。
10. 冯·诺依曼体系结构的特点：按地址访问和顺序执行
    变化：以运算器为中心→以存储器为中心，5 大部件→3 大部件，数据通路：分离式-总线式，单核CPU→多核微处理器，依托Cache 实现哈佛结构的存储体系。
    3大部件：CPU、内部存储器和输入/输出设备
11. 功能：（Function），事物或方法所发挥的有利作用、效能。性能（performance）器物所具有的性质与效用。

12. 对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方式是向下的，是向着内存地址减小的方向增长。

13. 兼容性：



弗林分类法（Flynn）

14. 数据的存放方式有大端和小端：数据低位放在高/低地址。

第二章

2018年1月8日 23:15

1. 硬件设计三条基本原则
    1. 简单源于规整
    2. 越小越快
    3. 优秀的设计需要适宜的折中方案
2. 指令格式设计(完备性、规整性、有效性、兼容性)，具有数据处理、数据存储、数据传送、程序控制四大类指令。
3. MIPS寻址模式
    1. 立即数寻址
    2. 寄存器寻址
    3. 基址寻址或偏移寻址：操作数在内存中，其地址是指令中基址寄存器和常数的和。
    4. PC相对寻址
    5. 伪直接寻址：J指令的寻址方式
4. 并行与指令：同步锁
    1. 加锁解锁
    2. 链接取数和条件存数
5. 汇编语言的伪指令：
   汇编器处理的一些机器语言指令的常见变种，硬件不需要实现这些指令，存在的目的是在汇编语言中简化程序转换和编程。
6. 在嵌入式设备领域中最流行的指令集体系结构是ARM。
   ARM和MIPS的主要区别是MIPS 有更多的寄存器，而ARM 有更多的寻址模式，都是RISC结构。

	ARM	MIPS	X86
整数寄存器	15通用寄存器x32位	31通用寄存器x32位	16个寄存器，其中8个32位通用寄存器
寻址方式	9种	5种	8种

寻址模式	ARM	MIPS
寄存器操作数	×	×
立即数操作数	×	×
寄存器+偏移（转移或基地址）	×	×
寄存器+寄存器（下标）	×	—
寄存器+寄存器倍乘（倍乘）	×	—
寄存器+偏移和更新寄存器	×	—
寄存器+寄存器和更新寄存器	×	—
自增，自减	×	—
相对 PC 的数据	×	—

ARMv8（64位）与ARMv7的异同：

舍弃：

- v8 中没有条件执行字段，而在 v7 中几乎每条指令都有该字段。
- 立即数字段仅仅是一个 12 位的常数，而在 v7 中是产生一个常数的函数的输入。
- ARM 舍弃了 Load Multiple 和 Store Multiple 指令。
- PC 不再是一个寄存器，因此如果对其进行写操作将会导致非预期的分支转移。

添加：

- v8 有 32 个通用寄存器，编译器设计者非常喜欢该特点。与 MIPS 相同，一个寄存器永远存放 0，虽然在 load 和 store 指令中该寄存器将由栈指针替代。
- ARMv8 的寻址方式是用于所有的字长，而在 ARMv7 中并非如此。
- 它包含了 ARMv7 中省掉的除法指令。
- 它增加了 MIPS 中的相等或不等的条件分支指令。

8. MIPS的指令：

I型

Op	Rs	Rt	Immediate

R型

Op	Rs	Rt	Rd	Shamt	func

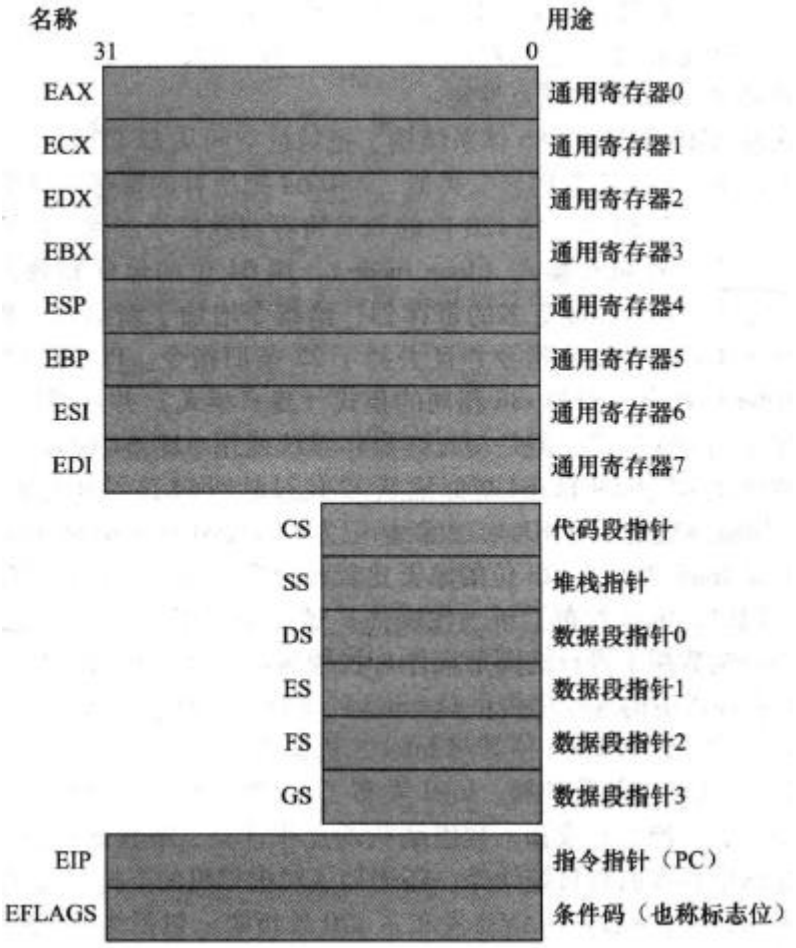
J型

Op	address

9. X86

寄存器组（只有8个通用寄存器）

从80386开始最上面的8个寄存器扩展到32位



和MIPS与ARM的不同：

1. X86的算术和逻辑指令中的一个操作数必须既是源操作数又是目的操作数
2. 一个操作数可以存放在存储器中，唯一的限制是不能有存储器-存储器模式（当然立即数不能放在源/目的操作数位置）

源/目标操作数类型	第二个源操作数
寄存器	寄存器
寄存器	立即数
寄存器	存储器
存储器	寄存器
存储器	立即数

3. 寻址方式：
    a. 立即数寻址：读取立即数作为操作数
    b. 寄存器寻址：读取寄存器作为操作数
    c. 直接寻址：直接给出地址（可以使用段寄存器）
    d. 寄存器间接寻址：偏移地址使用寄存器的直接寻址
    e. 存储器相对寻址：偏移地址使用寄存器加上立即数的寄存器间接寻址
    f. 基址变址寻址：偏移地址使用基址寄存器（BX或BP）和变址寄存器（SI或DI）的寄存器间接寻址
    g. 基址变址相对寻址：在基址变址寻址的基础上偏移地址可以使用立即数
    h. 寄存器比例寻址：偏移地址中变址寄存器的内容乘上比例因子

总结：相对指的是加立即数，间接指的是用寄存器的内容作为地址，变址指的是变址寄

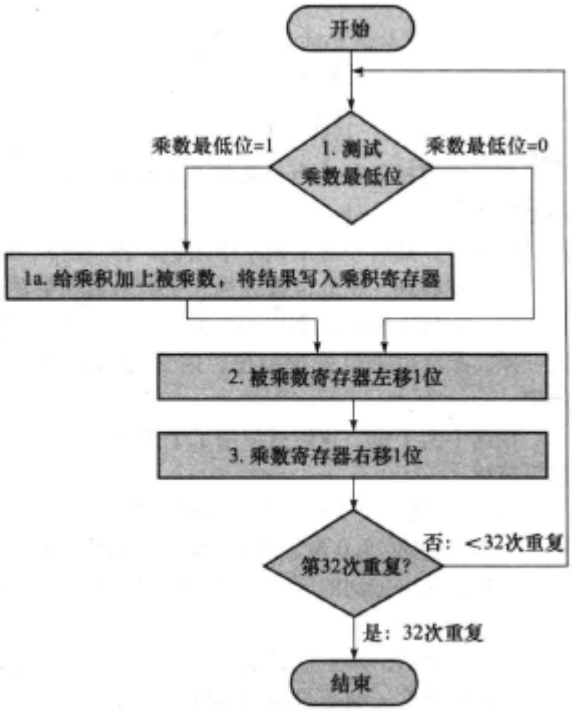
存器，基址指的是基址寄存器，比例指的是变址寄存器乘上比例因子
所谓基址寻址和变址寻址，前者的基址寄存器范围广，能访问整个程序空间，后者的变址寄存器的范围相对较下，只能访问数组（这是应用上的区别）

4. X86的指令长度可以从1 字节到15 字节变化。
5. ASCII编码：
   0-48
   A-65
   a-97
   汉字编码GB2312-80（区位码-内码）：
   把换算成十六进制的区位码加上2020H，就得到国标码。国标码加上8080H，就得到常用的内码（编码）。

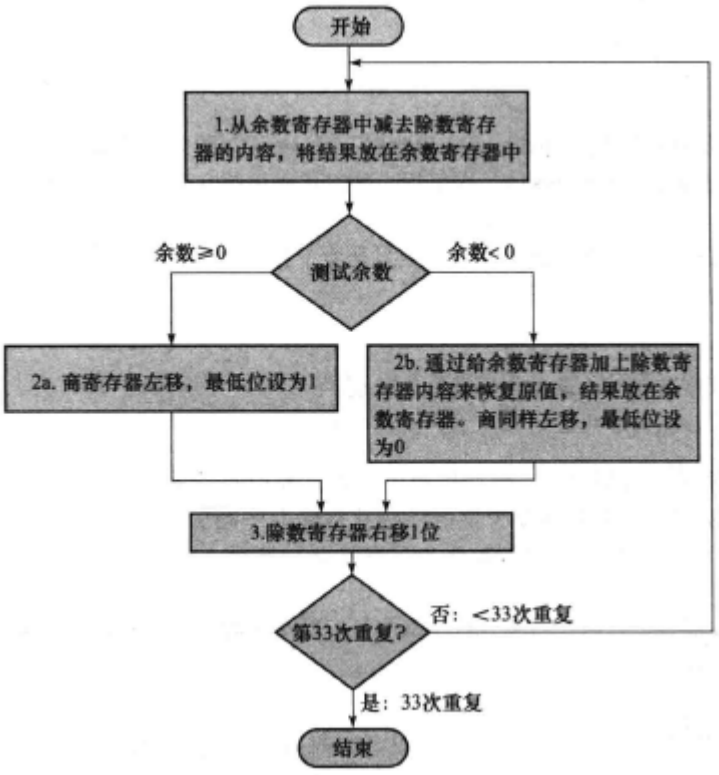
第三章

2018年1月8日　23:16

1. 进制系统的要点：**基数**和**位权**
2. 定点数就是将小数点前移的整数
    1. 乘除流程图
        a. 乘

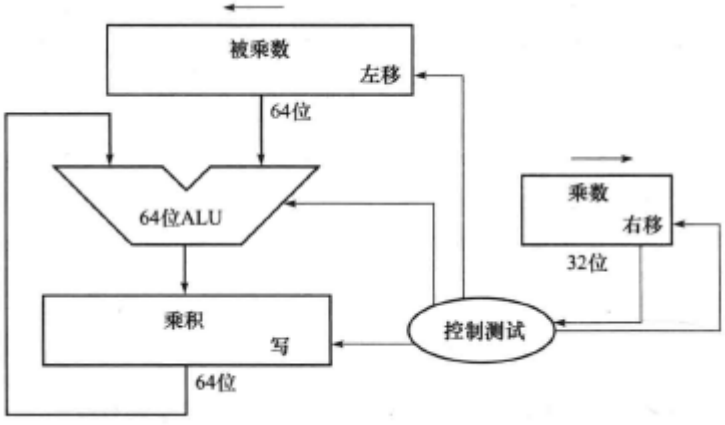


        b. 除（恢复余数法）

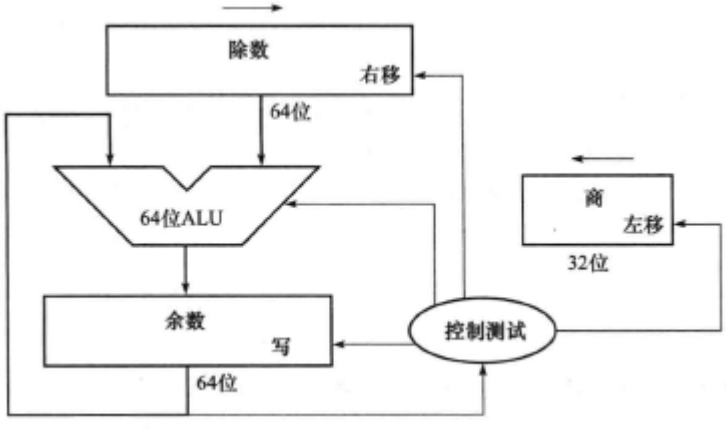


2. 结构框图

a. 乘



b. 除



除法要注意的是为了避免

-(x/y) != (-x)/y，规定被除数和余数的符号一致。

3. IEEE 754

单精度：

1	8	23

双精度：

1	11	52

带偏阶的记数法：最小的负指数表示为全0，最大的指数表示为全1（此时变成**无符号数**），所以阶码=指数+127的补码=指数的移码减一（移码：补码的符号位取反，而8位无符号数中符号位取反相当于加上128，所以变成移码减一即加上127）

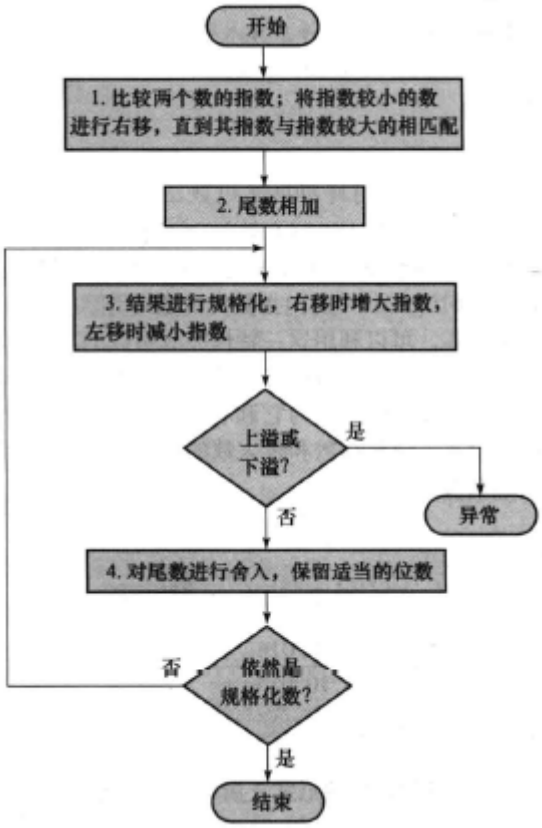
指数全0 保留下来用来表示0（其实是最小负数阶码）；指数全1保留下来标记指定的值和超出正常浮点数范围的情况，所以对于单精度，最大的指数为127，最小的指数为-126，阶码为1~254。

浮点运算：

1. 加法：

   对阶->尾数相加->规格化->舍入（有效位下一位为1则进一，这样可能要回到规格化这一步）

   判溢出（每次规格化之后）

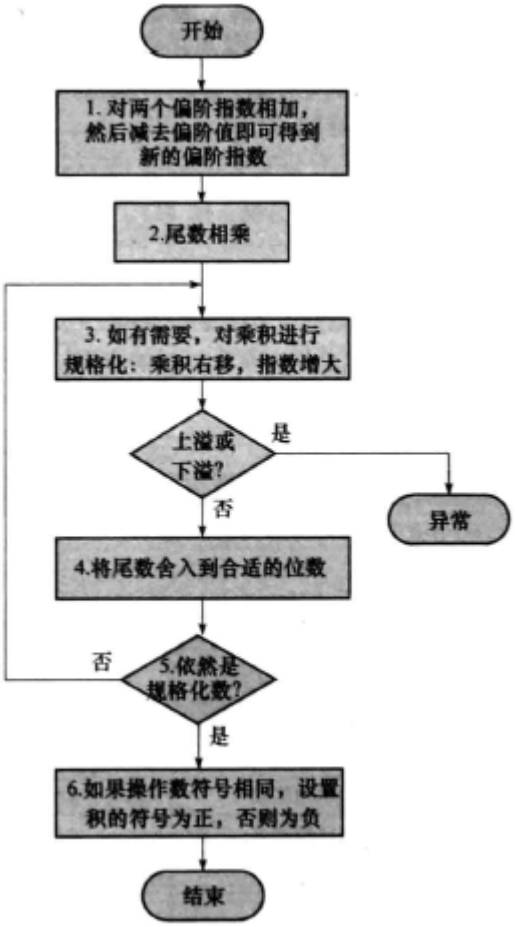


2. 乘法：

   阶码相加-127->尾数相乘->规格化->舍入（同样，可能需要重新规格化）->定符号

   判溢出（每次规格化之后）：

和加法相同



单符号位指数（原码）：最高位（除符号位）进位和符号位不相同时溢出。
双符号位指数（原码）：相加判溢出：相同无溢出，10下溢，01上溢。

浮点四则运算通式：

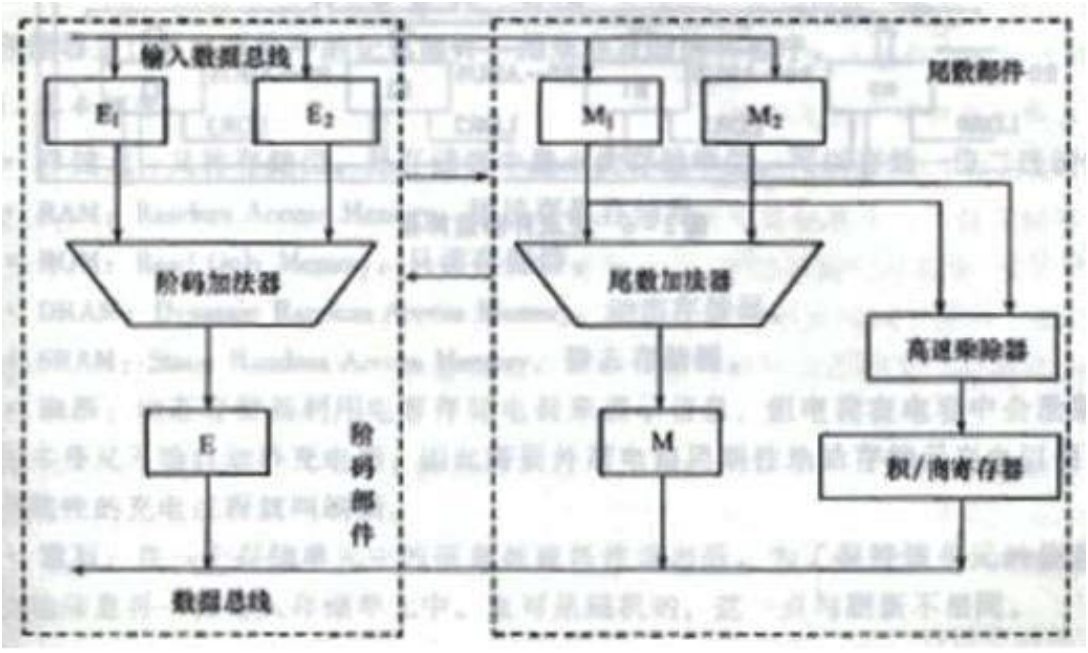
1）加法：$X+Y=(X_m 2^{Xe-Ye}+Y_m)\times 2^{Ye}$ （设 $Xe \leqslant Ye$）

2）减法：$X-Y=(X_m 2^{Xe-Ye}-Y_m)\times 2^{Ye}$ （设 $Xe \leqslant Ye$）

3）乘法：$X\times Y=(X_m \times Y_m)\times 2^{Xe+Ye}$

4）除法：$X\div Y=(X_m \div Y_m)\times 2^{Xe-Ye}$

运算器结构框图：



子字并行：

对一个宽字（书上的例子是128位）进位链进行分割，使得处理器可以同时对多个数据进行并行操作，这被称为子字并行，或者数据级并行，这样的宽字被称为向量或SIMD。

4. 陷阱与谬误：
    1. 右移指令不能表示与2的幂次相除（包括算术右移），原因是负数奇数（补码）右移一位并不代表除了2，但是原码可以（截断）。
    2. 浮点加法不能使用结合律，原因是浮点数只是近似表示，当两个不同符号的大数和一个小数相加时，就会出错，书上的例子：

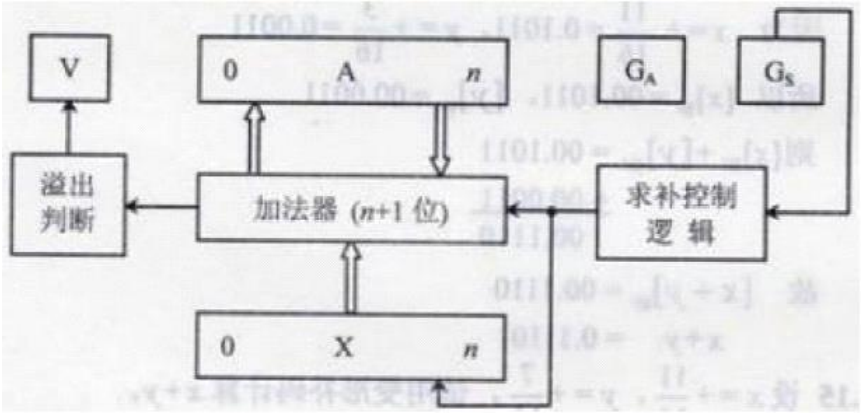
$$
\begin{aligned}
c + (a + b) &= -1.5_{10} \times 10^{38} + (1.5_{10} \times 10^{38} + 1.0) \\
&= -1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38} \\
&= 0.0 \\
(c + a) + b &= (-1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38}) + 1.0 \\
&= (0.0)_{10} + 1.0 \\
&= 1.0
\end{aligned}
$$

    3. 因为浮点数运算不可结合，所以也不可并行。
5. 多功能算逻运算单元设计（ALU）

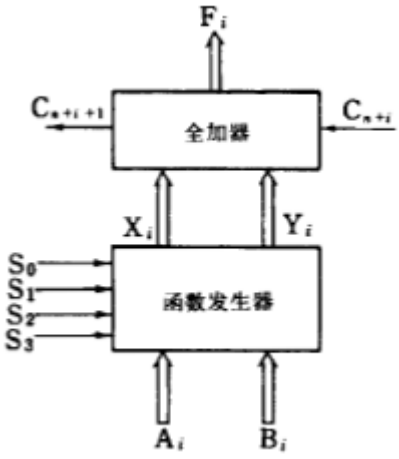
    一位全加器-四位全加器-溢出判断（异或电路）-加减控制（求补电路）-先行进位（进位链设计）-函数发生器设计。

    样例：

    加法器



ALU



X86中具有MMX（MultiMedia eXtension，多媒体扩展）指令和SSE（Streaming SIMD Extension，流处理SIMD 扩展）指令和高级向量扩展（advanced vector extension, AVX）。

以x86 为例，多媒体扩展可以被视作一种采用短向量的仅支持顺序向量数据传输的向量体系结构，但是缺少提升性能的特性。

第四章

2018年1月8日 23:16

1. cpu的功能：指令控制、操作控制、时间控制、数据加工
   cpu的控制方式：同步控制、异步控制、联合控制
2. X86：
   数据寄存器（DR）、指令寄存器（IR）、程序计数器（PC）、地址寄存器（AR）、累加器（AC）、状态寄存器（PSW）。
3. 为什么不使用单周期CPU设计：
   单周期CPU设计中时钟周期取得是最长的那条指令的时间长度，效率极低。
4. 流水线：
   流水线带来的性能提高是通过增加指令的吞吐率（单位时间内处理的指令条数）。
   相关：两条指令之间存在某种依赖关系。
   1. 数据相关（真数据相关）
      （先写后读）
      注意数据相关具有传递性。
   2. 名相关（名相关在我们的5段流水线中不会发生）
      如果两条指令使用相同的名，但是它们之间并没有数据流动，则称这两条指令存在名相关。
      a. 反相关（先读后写）
      b. 输出相关（写了又写）
      寄存器重命名技术可以解决。
   3. 控制相关
5. 几个周期
   时钟周期（节拍周期）
   CPU周期：从内存读取一条指令字的最短时间来定义
   指令周期（取值时间+执行时间）
   相互关系：1 个指令周期= 若干个CPU 周期；1 个CPU 周期= 若干时钟周期
   流水线冒险（冲突）：
   1. 结构冒险
      因缺乏硬件支持而导致指令不能在预定的时钟周期内执行的情况。即硬件不支持多条指令同时进行。
      解决办法：
      a. 插入暂停周期
      b. 设置独立地指令和数据存储器（或cache），即使用哈佛结构。
   2. 数据冒险
      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。
      解决方法：旁路（前推）或阻塞
      特殊情况：取数 - 使用型数据冒险是一类特殊的数据冒险，指当装载指令要取的数还没

取回来时其他指令就需要使用的情况。

其解决方法是让流水线阻塞一个步骤（流水线阻塞，也叫气泡）

数据冒险的4个条件：

1a. EX/MEM. RegisterRd = ID/EX. RegisterRs
1b. EX/MEM. RegisterRd = ID/EX. RegisterRt
2a. MEM/WB. RegisterRd = ID/EX. RegisterRs
2b. MEM/WB. RegisterRd = ID/EX. RegisterRt

同时保证目标寄存器不为零号寄存器。

总结起来就是前面的指令写入了后面的指令要读取的寄存器，无论是计算指令还是读内存指令。

对于1a和1b，不能使用旁路，只能阻塞。

3. 控制冒险：

决策依赖于一条指令的结果，但是其他指令正在执行中。

解决方法：预测（预测分支发生、不发生、动态预测）或阻塞（排空或冻结流水线，带来3个时钟周期延时），还有就是延迟决定：**延迟分支（**这是一个名词**）**执行下一条指令，在一条指令延迟之后再开始执行分支。（三种调度方法：从前调度、从目标处调度、从失败处调度）

缩短分支的延迟：

a. 计算分支目标地址
b. 判断分支条件

都移到ID阶段进行。

动态分支预测：根据运行信息在运行中进行分支预测。

分支预测缓存： 也称为分支历史记录表。一小块按照分支指令的低位地址索引的存储器区，其中包括一位或多位数据用以说明最近是否发生过分支。

评价指标：

1. 吞吐率
2. 加速比
3. 效率

CPU性能公式：

CPU 时间=指令数*CPI*时钟周期时间

或

CPU 时间=指令数*CPI/时钟频率

流水线处理机的实际CPI：

理想流水线的CPI 加上各类停顿的时钟周期数：

CPI 流水线= CPI 理想+ 停顿结构冲突+ 停顿数据冲突+ 停顿控制冲突

理想CPI 是衡量流水线最高性能的一个指标。

流水线通过时间=流水级数*每一级通过时间+（级数-1）*每一级通过时间

解决流水线瓶颈问题的常用方法：细分瓶颈段和重复设置瓶颈段

技术	降低CPI的哪一组成部分
转发和旁路	潜在的数据冒险停顿
延迟分支和简单分支调度	控制冒险停顿
基本编译器流水线调度	数据冒险停顿
基本动态调度（记分牌）	由真相关引起的数据冒险停顿
循环展开	控制冒险停顿
分支预测	控制停顿
采用重命名的动态调度	由数据冒险、输出相关和反相关引起的停顿
硬件预测	数据冒险和控制冒险停顿
动态存储器消除二义性	涉及存储器的数据冒险停顿
每个周期发出多条指令	理想CPI
编译器 相关性分析、软件流水线、踪迹调试	理想CPI、数据冒险停顿
硬件支持编译器推测	理想CPI、数据冒险停顿、分支冒险停顿

6. 解决流水线瓶颈问题的常用方法：细分瓶颈段和重复设置瓶颈段。
7. X86的指令执行：

   取指->执行->间址->中断

   MIPS的指令执行：

   取指->译码->执行->访存->写回

8. 异常：

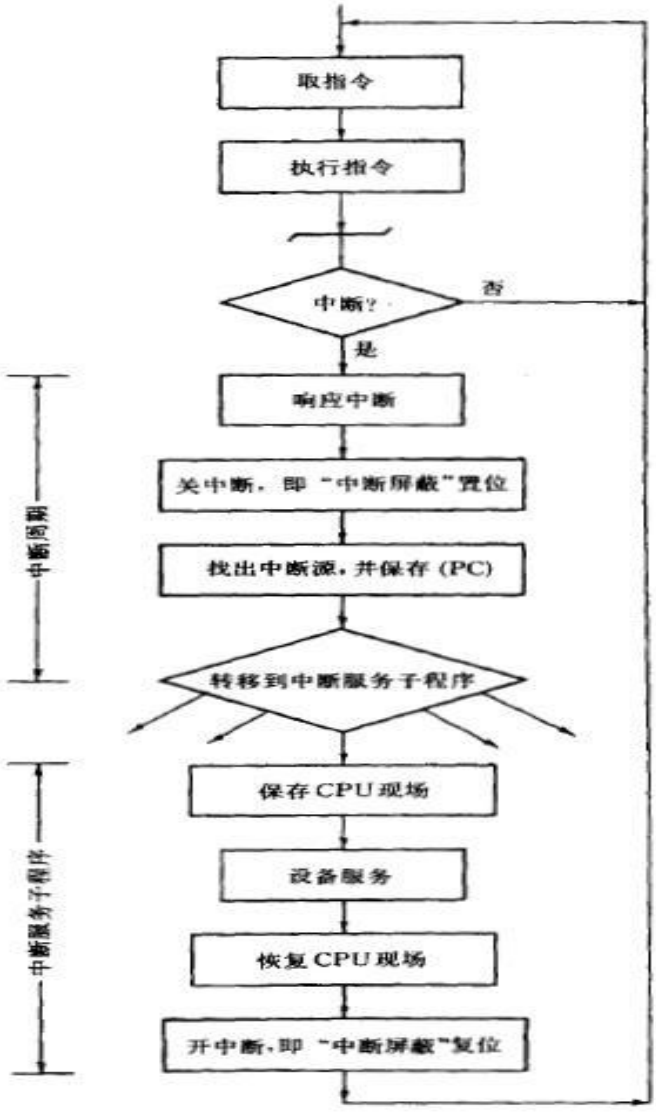


图 8.3 中断处理过程流程图

（画中断图的时候画像直方图那个）

异常和中断的区别：前者的发生不可控制，后者的发生可控制；

中断服务程序和子程序的区别在于中断未知；前者保护现场的执行者是中断隐指令，后者保护现场的执行者是子程序调用者；前者分为可屏蔽和不可屏蔽，后者没有这个概念；中断不一定可以嵌套，子程序一定可以。

中断隐指令：CPU响应中断之后，经过中断隐指令，转去执行中断服务程序，由硬件直接完成。

习惯上：

术语异常指控制流中任何意外的改变，而无论其产生原因是来自处理器内部还是外部，术语中断指由外部引起的事件。

异常处理的流程：

1. 由异常程序计数器保存出错指令的地址（地址加了4，使用异常处理例程要将地址减4），转让控制权。
2. 找到异常的源，有两种方法：

   MIPS的方法是设置一个状态寄存器（Cause寄存器），其中有一个字段用来记录异常产生的原因。所有异常使用同一入口地址，操作系统根据状态寄存器确定异常原因。

   向量中断：操作系统维护一个中断向量，每一个异常的原因对应于向量中的一个地址，操作系统通过异常向量地址得知异常原因。
3. 保护现场
4. 处理异常：异常具有优先级。
5. 返回现场：许多异常需要我们能够最终正常执行引起异常的指令。做到这点最简单的方法是先清除这条指令，然后在异常处理完后再重新执行这条指令。恢复现场。

响应中断的必要条件：

   a. 中断标志位置1
   b. 当前指令周期结束

是否支持多重中断。

9. 指令级并行：

   并行性的概念：计算机系统在同一时刻（同时性）或者同一时间间隔内（并发性）进行多种运算或操作。

   实现并行性的三种技术途径：时间重叠、资源重复、资源共享。**（流水线三者都用上了）。**

   从执行程序的角度来看，并行性等级从低到高可分为：指令内部并行、指令级并行、线程级并行、任务级或过程级并行、作业或程序级并行。从计算机体系结构的角度来看：现代计算机的并行性等级可分为：（指令级并行Instruction-Level Parallelism ）、数据级并行（Data-Level Parallelism）、多处理机和线程级并行（Multiprocessors and Thread-Level Parallelism）、需求级并行（Request-Level）。

   1. 有两种方法可以提高指令并行程度：
      a. 增加流水线的深度
      b. 多发射
   2. 推测： 一种编译器或处理器推测指令结果以消除执行其他指令对该结果依赖的方法。

推测技术应该包含一种机制来取消推测错误产生的影响。

还要延迟异常直至异常被肯定会发生。

多发射关注于减少每条指令的时钟周期数（CPI）。（因为多条指令同时分担这些时钟周期）

3. 静态多发射：（超长指令字）

   一个周期内发射多条指令，称为发射包，发射包可被视为是一条完成多个操作的长指令。

   使用延迟：在装载指令与可以使用其结果的指令间相隔的时钟周期数。

   循环级并行：

   重命名技术

   循环展开技术： 一种从存取数组的循环中获取更多性能的技术，其中循环体会被复制多份并且不同循环体中的指令可能会调度到一起。

4. 动态多发射：（超标量处理器）

   在最简单的超标量处理器中，指令顺序发射，每个周期处理器决定是发射0 条、1 条，还是多条指令。

   动态和静态的区别：

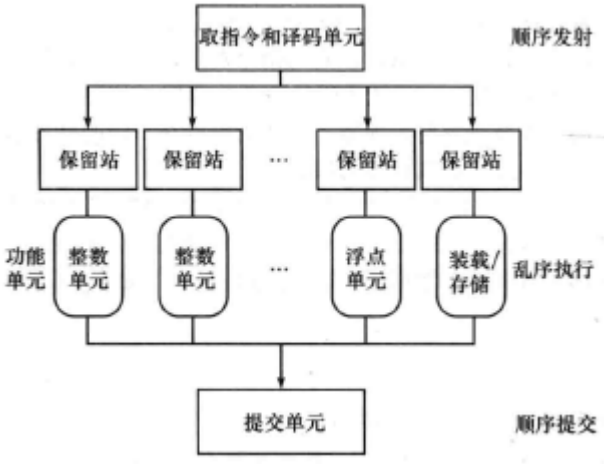
   在超标量处理器中，不管代码是否经过调度，都是由硬件来保证执行的正确性。并且，编译得到的代码应当始终正确执行，而与指令发射速率和处理器的流水线结构无关。在某些VLIW（超长指令字） 的设计中情况并非如此，当把代码从一个处理器移到另一个处理器上运行时，可能需要重新编译。在其他一些静态发射处理器上，代码可以在不同的处理器上实现正确运行，但效果可能很差以至于不得不进行更加有效的编译。

   就是说，动态多发射由硬件保证正确性，静态多发射由编译器保证。

   超标量利用了动态流水调度。

   动态流水调度：（对指令进行重排序以避免阻塞的硬件支持。）

   动态流水线调度选择下一条要执行的指令，可能的话会重排指令以避免阻塞。这种处理器的结构如下：



   提交单元：位于动态流水线和乱序流水线中的一个单元，用以决定何时可以安全地将操作结果送至程序员可见的寄存器和存储器。

   保留站： 功能单元的缓冲区，用来保存操作数和操作。

   重排序缓冲区：动态调度处理器中用于暂时保存执行结果的缓冲区， 等到安全时才将其中的结果写回寄存器或存储器。

将操作数缓存在保留站中并将结果放在重排序缓冲区中，实际上提供了一种寄**存器重命名机制。**

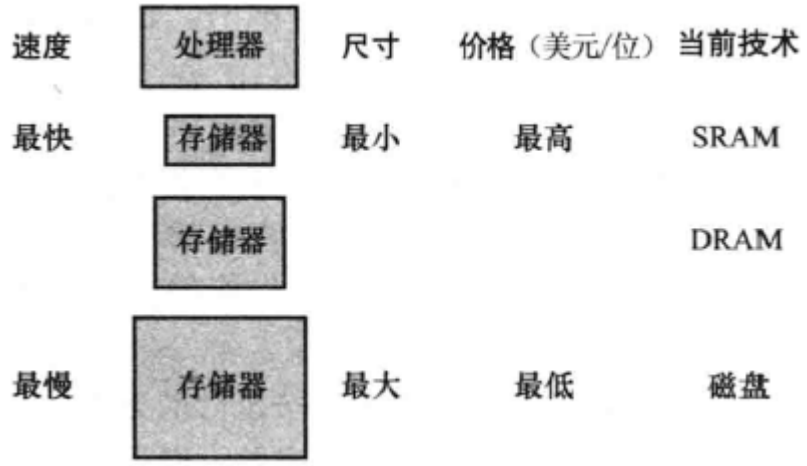
10. 小结：

    指令延迟：执行一条指令所真正花费的时间。

第五章

2018年1月8日　23:16

PC中的储存层次：
CPU中的寄存器-cache-主存-辅存
时间局部性：某个数据项在被访问之后可能很快被再次访问的特性。
空间局部性：某个数据项在被访问之后，与其地址相近的数据项可能很快被访问的特性。
存储器层次结构：一种由多存储器层次组成的结构，存储器的容量和访问时间随着离处理器距离的增加而增加。
存储器层次的基本结构：



SRAM：使用晶体管，靠电位（触发器）记录信息，不需要刷新
DRAM：使用电容，靠电容记录信息，需要刷新

每个DRAM内部组织成多个bank加大带宽（地址交叉）。
或者最普通的增大位宽也可以加大带宽。
刷新方式：（注意都是逐行刷新）
集中式：在一个时间段内完成全部刷新
分散式：在每个时钟周期刷新一部分
异步式：不依靠时钟，在单独的时间间隔内刷新一部分

闪存：电可擦除的可编程只读存储器，对闪存的写操作可以使存储位损耗，其中采用的损耗均衡技术，使得写操作的目的存储器尽量分散。
磁盘：

磁道：位于磁盘表面的数万个同心圆坏中的任意一个圆环称为一个磁道。
扇区： 构成磁盘上磁适的基本单位，是磁盘上数据读写的最小单位。
旋转延时：在磁头定位后，指定扇区通过读写头的所需时间。通常是磁盘转动一周时间的一半。

写储存有关的物理过程有时是不稳定的，因此所存放的信息可能丢失，有三种破坏信息的重要存储特性：破坏性读出、动态存储、断电后信息丢失。

块或行：可存在于或不存在于cache 中的信息的最小单元（也就是说低层次存储器的信息转移单位是块或行）

“Cache-主存”层次的目标：速度
“主存-辅存”层次的目标：容量
处理“写”的问题：

- 写直达
- 写回
- 前后台分离（即使用缓存区）

1. Cache
   如何分配cache（即cache的结构）？
   a. 直接映射：一种cache 结构，其中每个存储器地址仅仅对应到cache 中的一个位置。
      常用映射方法：块在cache中的位置=（块地址）mod（cache中的块数）
      特别的，如果块数是2的幂次，取模操作就变成了取块地址的低log2 （cache中的块数）位。
      块地址 = 数据地址/块大小
   b. 全相联：块可以放直到cache 中的任何位置。
      要与cache中的每一个位置进行比较，为减少开销，使用一个与cache 中每个项都相关的比较器并行完成，这些比较器加大了硬件开销，因而，全相联只适合块数较少的cache 。
   c. 组相联：块可以放直到cache 中的部分位置（至少两个）
      每个块有n 个位置可放的cac he 被称作n 路组相联cache。一个n 路组相联cache由很多个组构成，每个组中有n 块。根据索引域，存储器中的每个块对应到cache中唯一的组，并且可以放在这个组中的任何一个位置上。
      组号 =（块号）mod（cache 中的组数）
      提高相联度的好处在于它通常能够降低缺失率，而主要的缺点则是增加了命中时间（来自多路选择器）。
      替换策略：
      i. LRU（最近最少使用 Least Recently Used）法
      ii. 随机法
      iii. FIFO法

   如何判断数据是否在cache中?
   添加标记
   标记：表中的一个字段，包含了地址信息（可能只需要存放高位，因为低位地址可能被用来索引块在cache中的位置），这些地址信息可以用来判断cache 中的字是否就是所请求的字。

而标记中的信息可能是无效的，比如在刚启动机器时，所以需要一个有效位来标识一个块是否含有一个有效地址。

cache的访问：

查找数据的方式：

首先明确查找目标：cache中某个块中某个字的某个字节。

所以地址（该字节的地址）包含三部分：

（注意块地址可以由**字节地址/每块字节数**算出）

标记域（与cache中一个块中的标记对比）

cache的索引（用来索引cache中的块和块中的字（如果一个块中有多个字））

字节偏移（由于cache以字为单位，一个字有4字节，需要两位来索引字节）

标记域(32-n-m-2)	索引域(n+m)	字节偏移(2)

（2^n为cache块数，2^m为块中的字数）

而一个块的组成：

有效位	标记域	数据

由此可以计算cache中一个块的大小为：

1+32-n-m-2+2^m*32

则cache大小为：

2^n*块大小

但是cache的命名只考虑数据的大小

计算块在cache中对应的位置->

如果该位置没有要访问的块->从低层次存储器中取出要访问的块，替换cache中的内容（时间局部性）。（cache缺失）

如果该位置有要访问的块->命中

cache缺失

现在我们可以定义发生指令cache 缺失的处理步骤：

1）把程序计数器（ PC ）的原始值（当前PC- 4）送到存储器中。

2）通知主存执行一次读操作，并等待主存访问完成。

3）写cache项，将从主存取回的数据写入cache 中存放数据的部分，并将地址的高位（从ALU 中得到）写入标记域，设置有效位。

4）重启指令执行第一步，重新取指，这次该指令在cache中。

数据访问时对cache 的控制基本相同：发生缺失时，处理器发生阻塞，直到从存储器中取回数据后才响应。

缺失代价分为两部分：

a. 第一个字的延迟时间
b. 剩余部分块的传输时间

故此增加块的大小会导致缺失代价的第二部分增大，当存储器带宽增大时可以考虑更大的块，当存储器延时减小时可以考虑更小的块（虽然缺失率相对增加，但是缺失代价很小）。

Cache写操作处理

不一致：cache中的值与主存不一样。

写直达：也译为写通过或写穿。写操作总是同时更新cache 和下一存储器层次，以保持二者一致性。

写缺失：要写入的地址在cache中没有缓存，解决办法是首先从主存中取出块中的字。数据块被取回并存入cache 中后，我们就可以将引起缺失的字重新写入cache 中。同时，我们使用全地址将该字写入主存。

写缓冲：一个保存等待写入主存数据的缓冲队列。

加速写直达：当一个数据在等待被写入主存时，先将它放入写缓冲中。当把数据写入cache 和写缓冲后，处理器可以继续执行。存储器来完成写操作。

写回：在写回机制中，当发生写操作时，新值仅仅被写入cache 块中。只有当修改过的块被替换时才需要写到较低层存储结构中。

Cache性能的评估与改进

两个方面：

a. 减少存储器中不同数据块争用cache中同一位置的概率

b. 增加额外的一层，即多级高速缓存技术

Cache 性能优化技术：根据公式：平均访存时间 = 命中时间 + 失效率×失效开销，可以从三个方面改进Cache 的性能：降低失效率；减少失效开销；减少Cache 命中时间

CPU 时间 = (CPU 执行时钟周期数 + 存储器阻塞的时钟周期数) x 时钟周期

存储器阻塞时钟周期数 = 读操作引起阻塞的时钟周期数 + 写操作引起阻塞的时钟周期数

读操作阻塞的时钟周期数 = （读的次数 / 程序数）×读缺失率x 读缺失代价

对于写直达：写操作阻塞的时钟周期数 = [（写的次数 / 程序数）×写缺失率×写缺失代价] + 写缓冲区阻塞

在大部分写直达cache 结构中，读和写的缺失代价是一样的（都是从主存中取回数据块的时间）。如果假设写缓冲区阻塞可以被忽略，那么我们可以合并读写操作并共用一个缺失率和缺失代价：

存储器阻塞时钟周期数 = （存储器访问次数 / 程序数）×缺失率×缺失代价

也可以表示如下：

存储器阻塞时钟周期数 = （指令数 / 程序数）×（缺失数 / 指令） x 缺失代价

平均存储器访问时间（AMAT）作为检测cache 设计的方法：

AMAT = 命中时间 + 缺失率×缺失代价

使用多级Cache在发生缺失时的代价会更大，并且对于距离处理器较远的cache来说，访问时间已经不是关键，所以容量大、块大、相联度高。

可信存储器层次

- 可靠性
- 可用性

- 可信性

可靠性和可用性可以量化，可信性不可量化。

可靠性是一个系统或模块能够持续提供用户需求的服务的度量， 即从开始使用到失效的时间间隔，与之相应的是衡量**服务实现**的平均无故障时间（mean time to failure, MITF）。

提高MTTF:

a. 故障避免技术（ fault avoidance ） ：通过合理构建系统来避免故障的出现。
b. 故障容忍技术（ fault tolerance ） ： 采用冗余措施，当发生故障时，通过冗余措施保证系统仍然正常工作。
c. 故障预报技术（ fault forecasting ） ： 对故障进行预测，从而允许在器件失效前进行替换。

而年失效率（annual failure rate，AFR）是在给定MTTF情况下，在一年内预期的器件失效比例）。

相互转化方式为：AFR = 某时间单位的一年的时间/相对应的时间单位的MTTF

由系统各部分的年失效率算总的MTTF:

相加取倒数

而**服务中断**用维修平均时间（mean time to repair, MTTR）来度量。

失效间隔平均时间(mean time between failure, MTBF)=MTTF+MTTR

可用性是指系统正常工作时间在连续两次服务中断间隔时间中所占的比例:

可用性 = MTTF/ (MTTF + MTTR)

汉明编码:

检验2位错，纠正1位错:

1 ）对数据部分从左到右由1开始依次编号，这跟通常采用的从最右边开始由0 开始编号的做法相反。

2 ）将所有编号为2 的整数次幂的位标记为奇偶校验位（ 1， 2, 4, 8, 16， ...） 。

3 ）其他剩余位置用作数据位（位置3, 5, 6 , 7 , 9, 10, 11, 12, 13, 14, 15， ...） 。

4 ）奇偶校验位的位置决定了其对应的数据位，即对应位置为1，如下图所示:

位置		1	2	3	4	5	6	7	8	9	10	11	12
编码之后的数据位		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
奇偶校验位覆盖范围	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

注意到每个数据位都被至少两个奇偶校验位覆盖。

5 ）设置奇偶校验位，对各组进行偶校验。我们将校验位从大到小排列，得到的二进制数指示出错位置（0表示没有出错）。（即再算一遍校验位）

检验2位错，纠正1位错

在最后加上一位奇偶校验码P，对整个字进行计算校验，校验算法为计算出纠错码组的奇偶性H，再计算P，结构分四种情况：

1) H 为偶并且P为偶，这表示没有错误发生。

2) H 为奇并且P 为奇，这表明出现了一位可纠正错误。（当出现一位错时，P应当为奇。）

3) H 为偶并且P为奇，这说明出现的仅仅是P，因此将P取反即可。

4) H 为奇并且P为偶，这表示出现了两位错。（当出现两位错时，P应当为偶。）

为了计算出SEC 需要的位数，假定p 表示校验位的位数， d 表示数据位的位数，则整个码字为p+d 位。如果采用p 个纠错位指示错误（码字长度为p+d 的情况下），再加上没有出现错误的情况，不难得到下面的不等式：

2^p >= p+d+1 位， 因此p >= log(p + d + 1)

（即能检验出的错误位置（包括没有错误）大于等于错误位置的个数（包括不出错））

虚拟机

在虚拟存储器系统中，由于写到存储器层次结构的较低层（磁盘） 的延迟很大，因此只有写回策略是可行的。

虚拟机监视器（virtual machine monitor，VMM）或管理程序（hypervisor）：支持虚拟机的软件

主机（host）：底层的硬件平台

优点：

a. 提供保护功能
b. 软件管理
c. 硬件管理

VMM的需求：

- 除了性能相关的行为或因多虚拟机共享而造成的固定资源限制以外，客户软件在虚拟机上的运行应该和它在本地硬件上的运行完全相同。
- 客户软件不能直接改变实际系统中的资源分配。

必备条件为：

- 至少两个处理器模式，系统级和用户级
- 提供一部分处理器的状态， 这部分内容是用户进程可读而不可写的。特权级指令集合只能在系统模式下使用，如果在用户模式下执行将会产生trap中断；所有系统资源只能由这些指令控制。

硬件设备除了上面的两点之外，还需要有另外一个特点：

- 提供能让处理器在用户态和管理态下相互切换的机制。

虚拟存储器：

虚拟存储器：一种将主存用作辅助存储器高速缓存的技术。

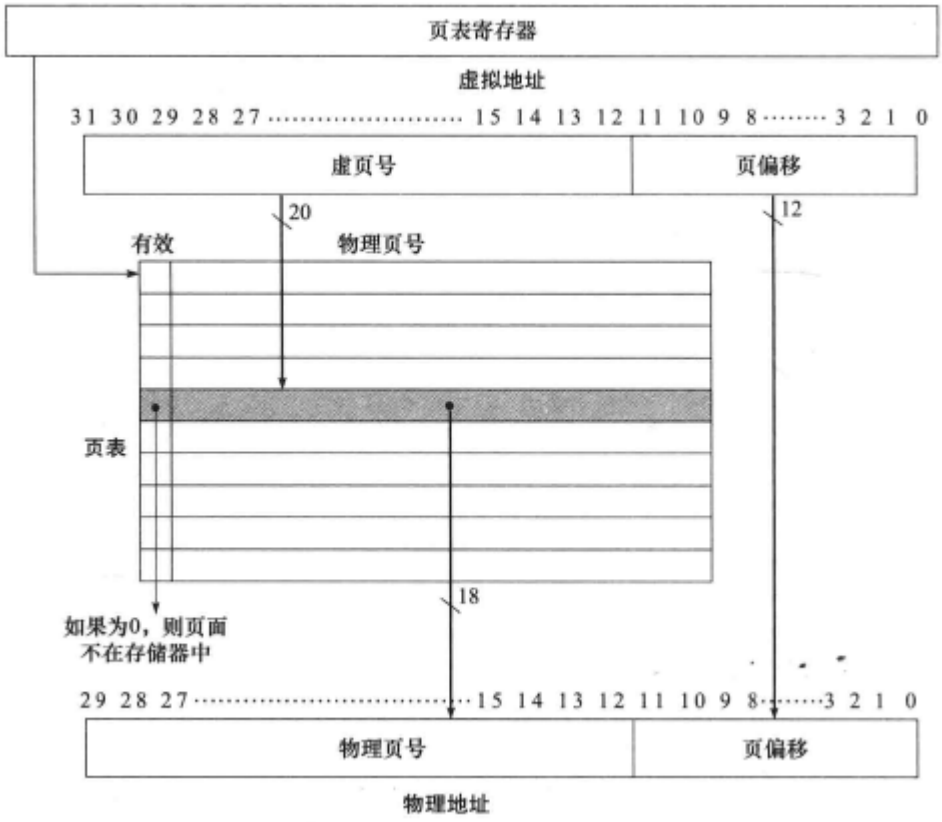
虚拟存储器的空间大小取决于计算机地址位数。

目的：允许云计算在多个虚拟机之间有效而安全地共享存储器；消除一个小而受限的主存容量对程序设计造成的影响。

物理地址：主存储器的地址。
保护：一组确保共享处理器、主存、I/O设备的多个进程之间没有故意地、无意地读写其他进程的数据机制，这些保护机制可以将操作系统和用户的进程隔离开来。
在使用虚拟机的时候，为了避免程序之间只用划分给它的对应部分主存，将每个程序都编译到它自己的地址空间，同时虚拟机要实现程序地址空间到物理地址的转换。
由于历史原因，在虚拟存储器中，块被称为**页，**访问缺失被称为**缺页。**
在虚拟存储器中，处理器产生一个虚拟地址（ virtual address ），再结合软硬件（TLB和页表）转换成一个物理地址(physical address ）， 然后就可以被用来访问主存了。
重定位：将虚拟地址映射到不同的物理地址，允许我们将程序加载到主存的任何位置。
虚拟地址被划分为**虚页号**和**页偏移**，虚页号会转换为物理页号，物理页号构成物理地址的高位部分，而页偏移是不变的，构成物理地址的低位部分，页偏移域的位数决定了页的大小。
缺页的损失高达数百万个时钟周期，而主要有取得标准大小的页的第一个字所需的时间来确定，所以：

- 为了弥补较长的访问时间，页应该足够大。
- 使用能够降低却页率的组织结构（主要是全相联技术）
- 软件来减小缺页率
- 写直达机制的时间太长，使用写回机制。

访问
使用页表来定位页。
页表：保存着虚拟地址和物理地址之间转换关系的表。页表保存在主存中，通常使用虚页号来索引，如果这个虚页当前在主存中，页表中的对应项将包含虚页对应的物理页号，同时硬件包含一个指向页表首地址的寄存器（页表寄存器），还应注意到因为不同的进程使用相同的虚拟地址，因此各个进程有各自的页表。
一个例子：

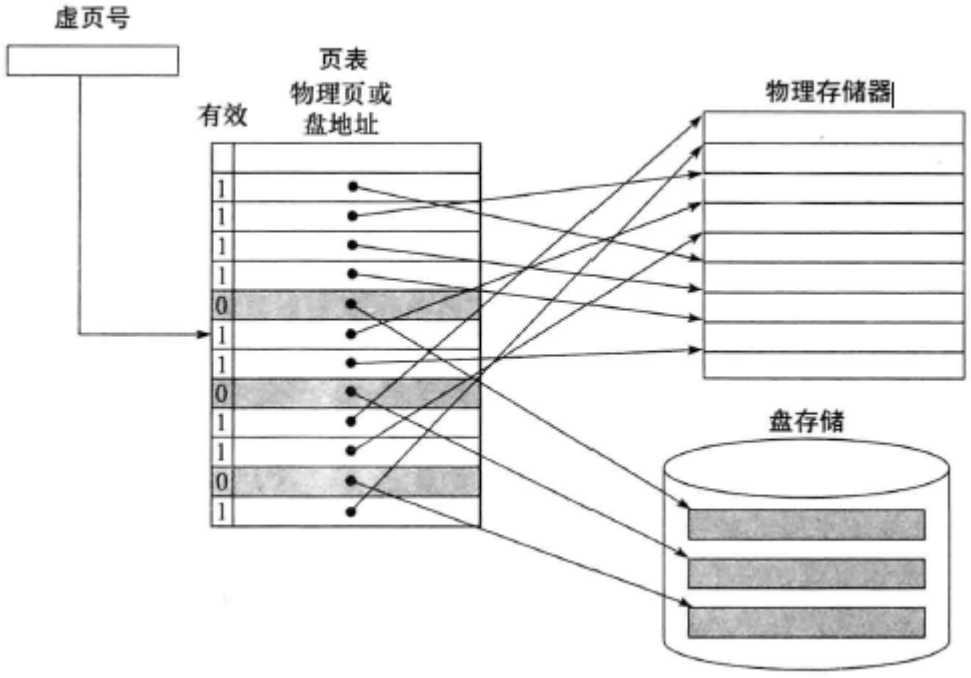


每次访问都要两次访存：第一次是获得物理地址，第二次是获得数据。

缺页故障：

保存虚拟地址空间的每一页在磁盘上的位置以准备缺页时取磁盘内容。操作系统为为进程的全部虚拟地址空间所预留了磁盘空间（交换区）。同时建立一个表来记录每个虚拟页在磁盘上的存放位置，和页表类似。

例子：



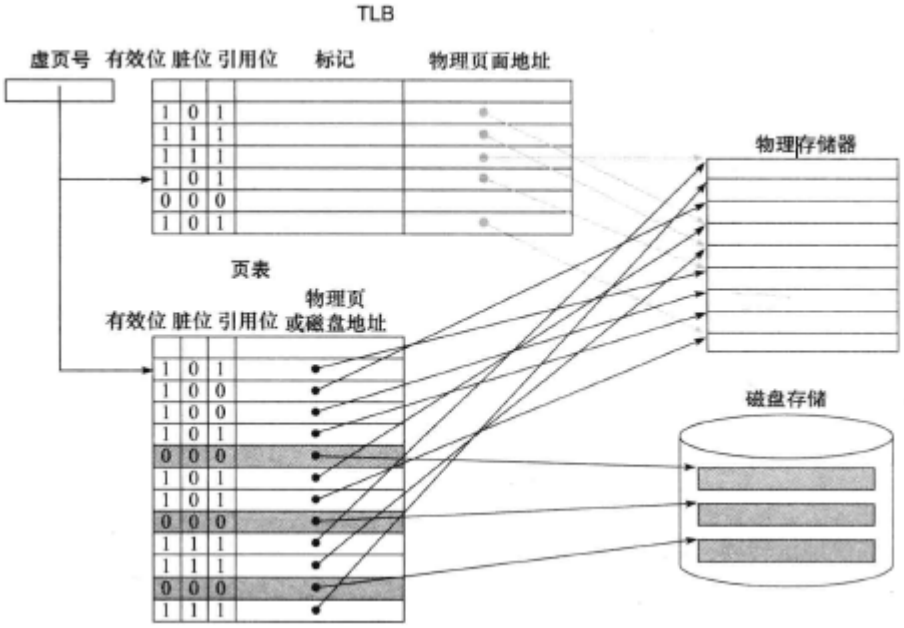
（画成一个表的原因是物理页地址和磁盘页地址的表在逻辑上是一个表，但是保存在两个独立的数据结构中。）

当一次缺页发生时，如果主存中所有的页都在使用，操作系统仍必须选择一页进行替换（因为有多个进程同时使用主存，页会不够），替换原则同样是LRU，被替换的页写入

磁盘交换区（交换区的作用是暂时存放被替换的页，在该页被再次访问时再移入主存，因为进程可能未结束）。
页表中还有一个脏位（对应的有脏页），表示页是否被修改过。

TLB

利用时间局部性和空间局部性：一个转换的虚页号被使用时，可能很快被再次使用。
所以TLB（Translation-Lookaside Buffer）（地址变换高速缓存），常称为**块表**。
块表：用于记录最近使用地址的映射信息的高速缓存（cache），从而可以避免每次都要访问页表。
例子：

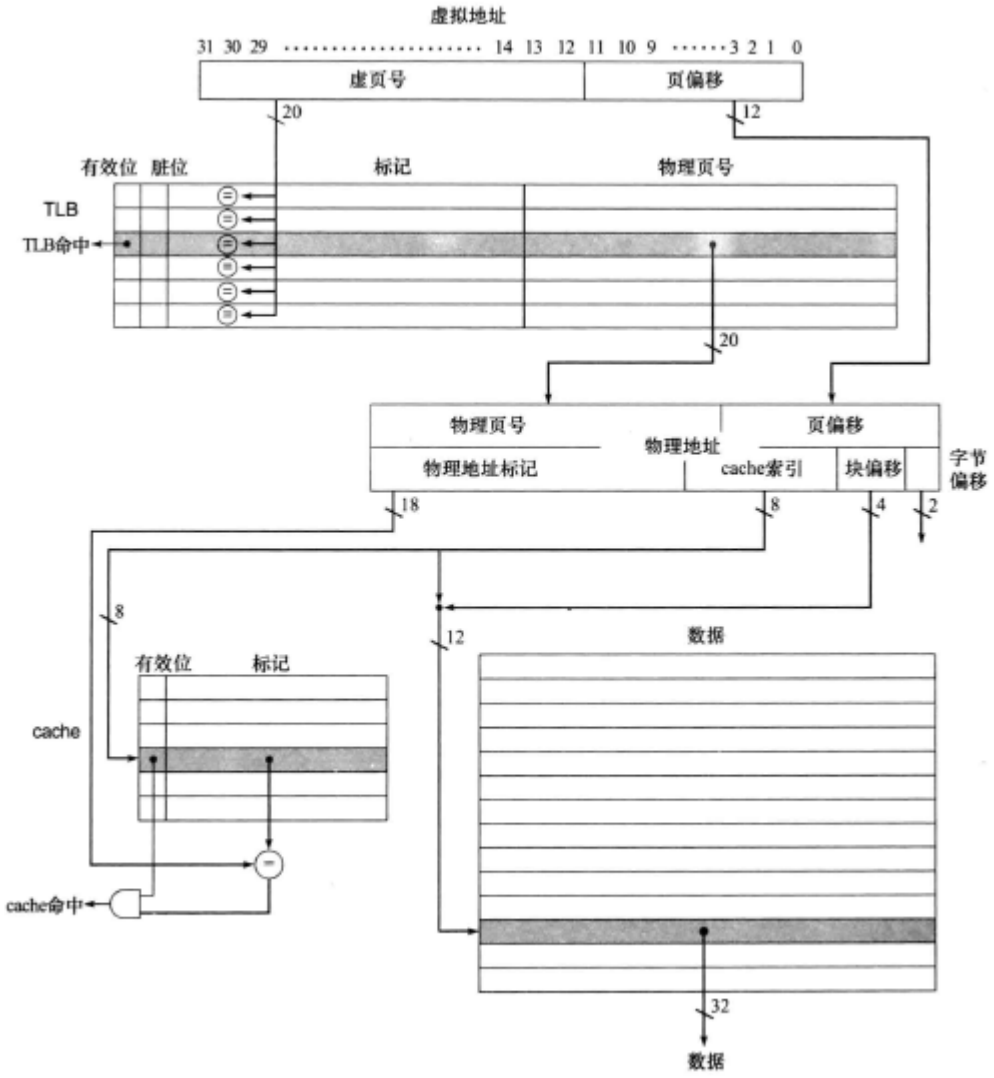


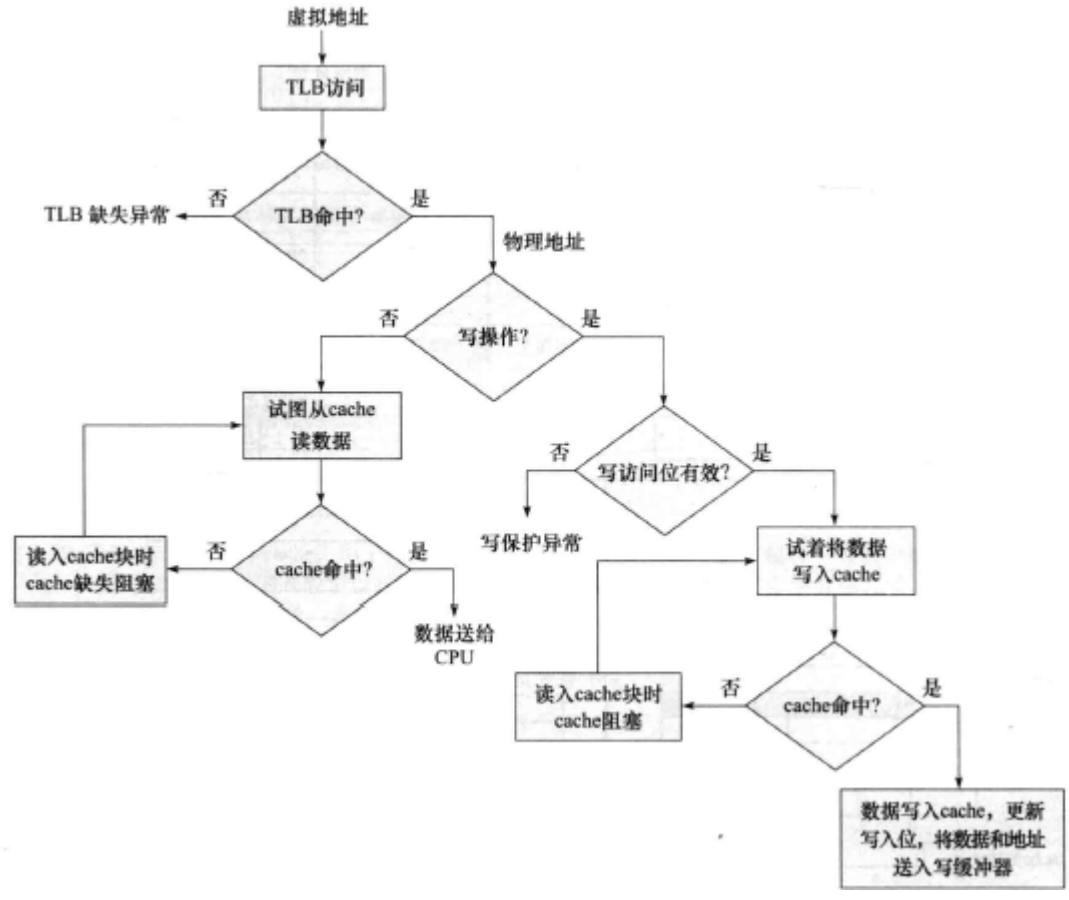
因为TLB是一个Cache，其必须要有标记域，而页表不是。

使用TLB后页缺失分两种：

- TLB缺失
  替换TLB的一项
- 真的缺页

在处理器中的使用
首先注意到处理器使用的是虚拟地址，要转换为物理地址才能使用（主存访问的第一步），并且处理器首先从cache读取数据，只有发生了cache缺失才会访问主存（第二步）。
e.g.


虚拟地址
31 30 29 ····· 14 13 12 11 10 9 ·····3 2 1 0
虚页号
页偏移
20
12
有效位 脏位
标记
物理页号
TLB
TLB命中
20
物理页号
物理地址
页偏移
物理地址标记
cache索引
块偏移
字节偏移
18
8
4
2
8
12
数据
有效位
标记
cache
cache命中
32
数据


虚拟地址
TLB访问
否
TLB命中?
是
TLB 缺失异常
物理地址
否
写操作?
是
试图从cache
读数据
否
写访问位有效?
是
写保护异常
读入cache块时
cache缺失阻塞
否
cache命中?
是
数据送给
CPU
试着将数据
写入cache
读入cache块时
cache阻塞
否
cache命中?
是
数据写入cache，更新
写入位，将数据和地址
送入写缓冲器

TLB缺失：

有两种可能性：

1）页在主存中，只需要创建缺失的TLB 表项。

2）页不在主存中，需要将控制权交给操作系统来解决缺页。

调用异常机制。

另外， TLB 缺失或者缺页异常必须在的存发生的同一个时钟周期的末尾被判定，因此下一个时钟周期就开始进行异常处理而不是继续正常的指令执行。

缺页的三个处理步骤：

1）使用虚拟地址查找页表项，并在磁盘上找到被访问的页的位置。

2）选择替换一个物理页；如果被选中的页被修改过， 需要在把新的虚拟页装入之前将这个物理页写回到磁盘上。

3）启动读操作，将被访问的页从磁盘上取回到所选择的物理页的位置上。

引发缺失的虚拟地址取决于当前缺失是指令缺失还是数据缺失。

数据访问引起的缺页异常较指令缺失难处理，这是由于以下三个特性：

1）它们发生于指令中间，不同于指令缺页。

2）在异常处理前指令没有结束。

3）异常处理之后，指令必须重新执行，就好像什么都没发生过一样。

集成虚拟存储器、TLB和cache

三者同时工作

对于**访问**：

考虑全部缺失情况

TLB	页表	cache	这种情况可能发生么？如果可能，在什么情况下发生？
命中	命中	缺失	可能，但若 TLB 命中就不可能检查页表
缺失	命中	命中	TLB 缺失，但在页表中找到表项；重试后在 cache 中找到数据
缺失	命中	缺失	TLB 缺失，但在页表中找到表项；重试后在 cache 中未找到数据
缺失	缺失	缺失	TLB 缺失，随后发生缺页；重试后在 cache 中必找不到数据
命中	缺失	缺失	不可能：如果页不在主存中，TLB 中没有此转换
命中	缺失	命中	不可能：如果页不在主存中，TLB 中没有此转换
缺失	缺失	命中	不可能：如果页不在主存中，数据不允许在 cache 中存在

注意三者的工作顺序

对于**替换**：

当操作系统决定将某一页移到磁盘上去时，就在cache 中将该页中的内容刷新。同时，操作系统修改页表和TLB ， 而后尝试访问该页上的数据都将发生缺页。

存储器层次结构的一般框架

问题1：一个块可以被放在何处

注意失效率公式：

大小为N的直接映射的失效率和大小为N/m的m路组相联相等。

问题2：如何找到一个块

相联度	定位方法	需要比较的次数
直接映射	索引	1
组相联	索引组，查找组中元素	相联的度
全相联	查找所有 cache 项	cache 的容量
	独立的查找表	0

（注意独立的查找表）

问题3：当cache 缺失时替换哪一块

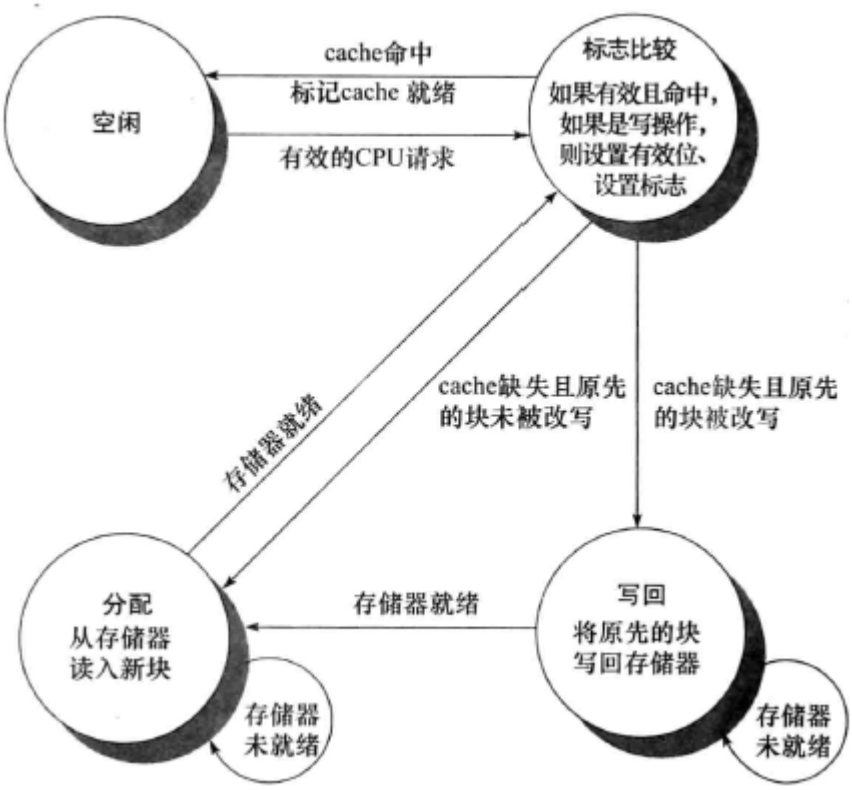
问题4：写操作如何处理

3C模型：

- **强制缺失**（ compulsory miss ）：对从没有在cache中出现的块第一次进行访问引起的缺失。也称为**冷启动缺失**（ cold- start miss ）。
    - 减少方法：增加块的大小，这样同样的程序所使用的块数目变少了，可以减少对不同块的访问次数（负面效应是增加缺失代价）
- **容量缺失**（ capacity miss ）：由于cache容纳不了一个程序执行所需要的所有块而引起的cache缺失，当某些块被替换出去，随后再被调入时，将发生容量缺失。
    - 减少方法：增大cache容量（负面效应是成本增加和访问时间）
- **冲突缺失**（ conflict miss ）：在组相联或者直接映射的cache中，多个块竞争同一个组时而引起的cache 缺失。冲突缺失在直接映射或组相联cache中存在，而在同样大小的全相联cache 中不存在（在全相联中就变成了容量缺失）。这种cache 缺失也称为**碰撞缺失**（collision miss）。
    - 减少方法：提高相联度（负面效应是增加了访问时间）

设计变化	对缺失率的影响	可能对性能产生的负面影响
增加 cache 容量	减少了容量缺失	可能增加访问时间
提高相联度	由于减少了冲突缺失，因此降低了缺失率	可能增加访问时间
增加块的容量	由于空间局部性，因此对很宽范围内变化的块大小，都能降低缺失率	增加缺失代价，块太大还会增加缺失率

有限状态机实现cache：



Cache的一致性

在多核多处理器中，多个处理器共享物理地址空间，如果不采取防范措施，不同的处理器可能得到不同的值。（注意各处理器有自己的cache）

一般情况下，如果在一个存储器系统中读取任何一个数据项的返回结果总是最近写入的值，那么可以认为该存储器具有一致性。

包括两个方面：

- 一致性（coherence），它定义了读操作可以返回什么样的数值。
- 连贯性（consistency），它定义了写人的数据什么时候才能被读操作返回。

关于一致性：

1）处理器P 对位置X 的写操作后面紧跟着处理器P 对X 的读操作，并且在这次读操作和写操作之间没有其他处理器对X 进行写操作，这时读操作总是返回P 写入的数值。

2）在其他处理器对X 的写操作后，处理器P 对X 执行读操作，这两个操作之间有足够的间隔并且没有其他处理器对X 进行写操作， 这时，读操作返回的是写人的数值。

3）对同一个地址的写操作是串行执行的（serializea）；也就是说，任何两个处理器对同一个地址的两个写操作在所有处理器看来都有相同的顺序。

如何实现一致性：

使用cache一致性协议，最常见的是监听协议（snooping）。

监听协议：

写无效协议：在处理器写数据之前，保证该处理器能独占访问该数据项，即cache中该数据项的其它副本都是无效的。

目录协议：将监听协议中各个处理器对于cache的状态集合起来，集合的地方称为目

录。
使用写无效协议时，在写操作执行时，其它副本无效，则其它处理器在执行读操作时cache会发生缺失，需要取回新的数据副本。同样的，多个处理器同时要求执行写操作时，会发生竞争，落败的处理器要执行写操作必须获得新的副本，于是处理器写操作完成了串行化。
关于连贯性：
可以做两个假设：
第一，直到所有处理器看见写操作的结果，这个写操作才能完成（没有完成时可以允许下一个写操作发生）。
第二，处理器不能改变与存储器访问相关的写操作的次序。

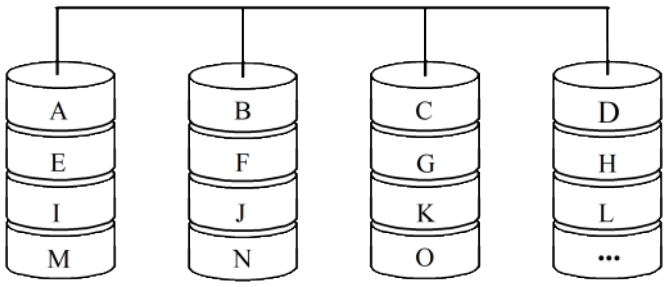
对于共享cache，组相联度至少等于核的数量或共享该cache的线程数。
具有一致性的cache提供两种服务：
迁移：数据项可以移入本地cache 并以透明的方式使用。迁移不但减少了访问远程共享数据项的延迟，而且减少了对共享存储器带宽的需求。
复制： 当共享数据被同时读取时， cache 在本地对数据项做了备份。复制减少了访问延迟和读取共享数据时的竞争现象。

冗余廉价磁盘阵列（Redundant Arrays of Inexpensive Disks, RAID）
（如何采用多块磁盘并行工作来提高吞吐率，同时采用适当数量的冗余磁盘来提高可靠性）
RAID分9种RAID0~RAID6，RAID01和RAID10
三大特性：

- RAID由一组物理磁盘驱动器组成，操作系统视之为一个逻辑驱动器
- 数据分布在一组物理磁盘上
- 冗余信息被存储在冗余磁盘空间中，保证磁盘在万一损坏时可以恢复数据

RAID0不支持第三特性
关键点：

a. 条带（RAID0）：
把数据切分成条带，以条带为单位交叉地分布存放到多个磁盘中（并行提高吞吐量）。
e.g.

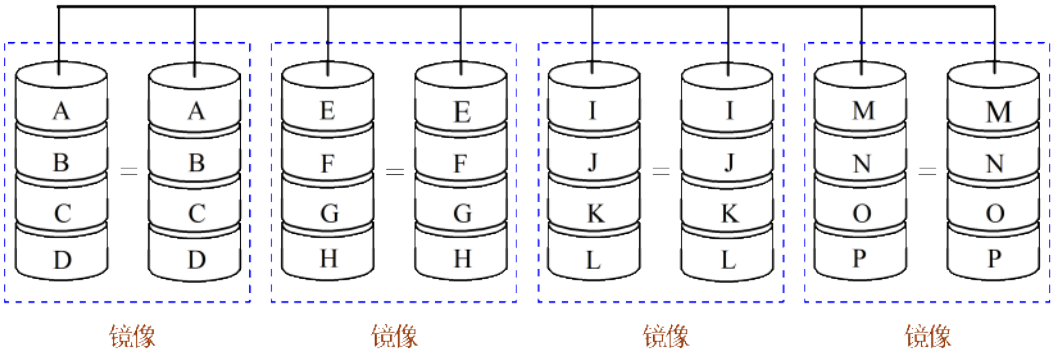


（AB…为连续数据）

b. 镜像（RAID1）：

这里要提到冗余信息（主要书奇偶检验码和汉明码），存放方式有两种：集中存放在几个盘里面和均匀的存放在所有盘中。

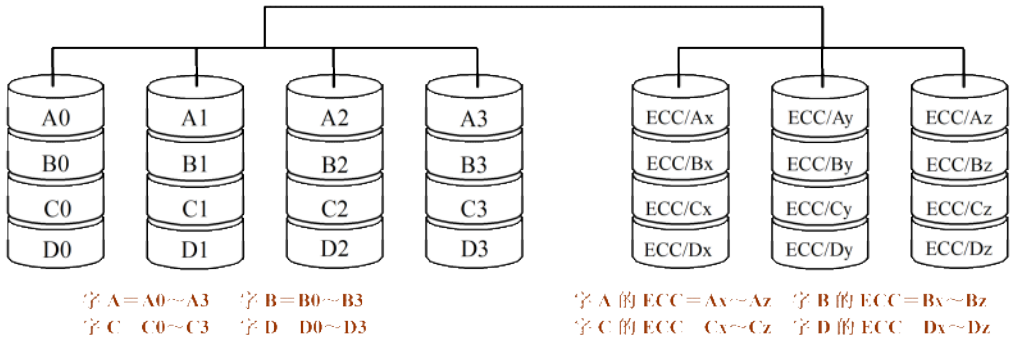
每当把数据写入磁盘时，将该数据也写入其镜像盘，形成信息的两个副本。



(说明：

RAID0	无冗余信息
RAID1	镜像
RAID2	存储器式的磁盘阵列（按汉明纠错码的思路构建），冗余信息单独存放。
RAID3	位交叉奇偶校验盘阵列，冗余信息单独存放。
RAID4	块交叉奇偶校验磁盘阵列。 采用比较大的条带，以块为单位进行交叉存放和计算奇偶校验，冗余信息单独存放。
RAID5	块交叉分布奇偶校验磁盘阵列 （和RAID4的区别在于冗余信息均匀放在所有磁盘）。
RAID6	P + Q双校验磁盘阵列，和RAID5的区别是能容忍两个磁盘出错，当然冗余信息是两倍。

RAID2



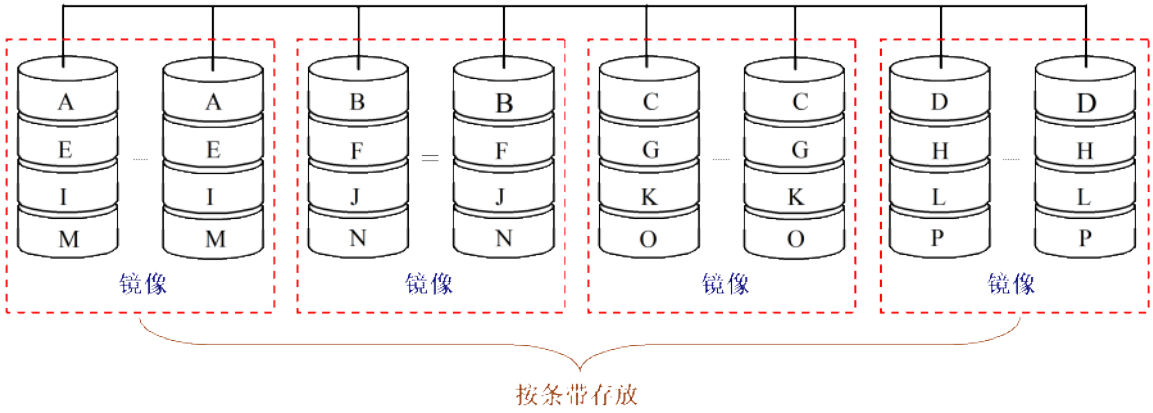
RAID6 思想最常见的实现方式是采用两个独立的校验算法，假设称为 P 和 Q，校验数据可以分别存储在两个不同的校验盘上，或者分散存储在所有成员磁盘中。当两个磁盘同时失效时，即可通过求解两元方程来重建两个磁盘上的数据。

）

主要使用：

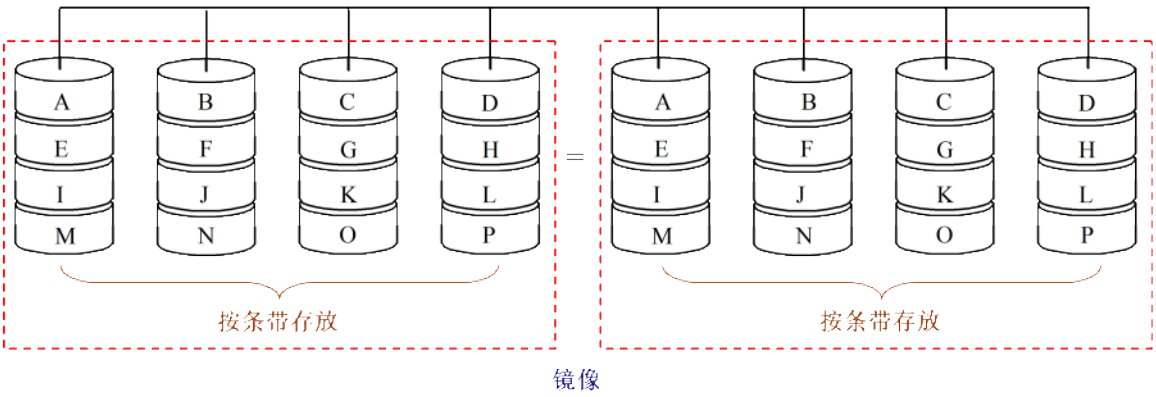
RAID10

先镜像，再条带存放



RAID01

先条带存放，再镜像



存储体扩展（重要）：

位扩展：扩展每个字的位宽

字扩展：增加存储体容量

第六章

2018年1月8日　23:16

并行的分级：

指令级并行

数据级并行：对不同数据执行相同操作所获得的并行。

多处理机级或线程级或进程级或任务级并行：通过同时运行独立程序的方法来利用多处理器。

需求级并行

并行：把两个或多个事件在同一时刻发生的并行性叫做同时性(simultaneity)；而把两个或多个事件在同一时间间隔内发生的并行性叫做并发性(concurrency)。

多处理器是任务级并行，其可以在每单位焦耳上获得更高的性能。
注意多处理器还支持可变数量的处理器。
并行处理程序：同时运行在多个处理器上的单一程序。
集群：通过局域网连接的一组计算机，其作用等同于一个大型的多处理器。
多核微处理器：在单一集成电路上包含多个处理器（“核”）的微处理器。
共享内存处理器：共享一个物理地址空间的并行处理器。

并发软件和顺序软件都可以运行在并行硬件和串行硬件上，并且顺序程序也能在多处理器上得到好处。

并行的难点不在于硬件，而是在于应用程序很难经过重新编写后能在多处理器上获得更快的执行时间，因为并行编程需要处理**调度、将任务分割成可并行的部分，负载均衡、同步时间和通信开销。**除此之外，**Amdahl定律**也指出，为了充分利用多核，程序中任何一个小部分都需要并行化。（串行的部分属于定律中的“未受改进影响的时间”）

强比例缩放（ strong scaling ）指保持问题规模固定所测得的加速比。弱比例缩放（ weakscaling ）指问题规模随处理器数量按比例增加所获得的加速比。假定问题规模M 是主存中的工作集，处理器数量为P ，那么每个处理器所占用的内存对于强比例缩放大约是M/P ，对于弱比例缩放大约是M 。（传统上认为弱比例缩放比强比例缩放简单，弱缩放可以补偿程序的串行部分，强缩放的缩放性会被串行部分所限制。）

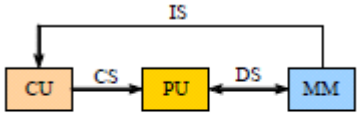
负载均衡：如果某个处理器的负载比其它处理器更重，那么要等到该处理器完成任务才算整个任务完成，而这样其他处理器的利用率就降低了。

基本并行硬件的两种类型：

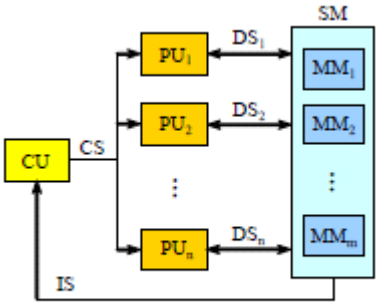
- 共享存储多处理器
- 集群

并行硬件的四种分类：

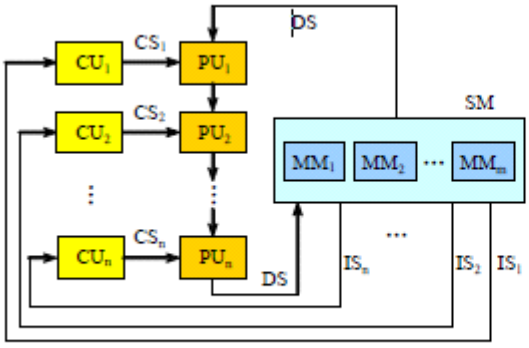
		数据流	
		单	多
指令流	单	SISD：Intel Pentium 4	SIMD：x86 的 SSE 指令
	多	MISD：至今没有实例	MIMD：Intel Core i7



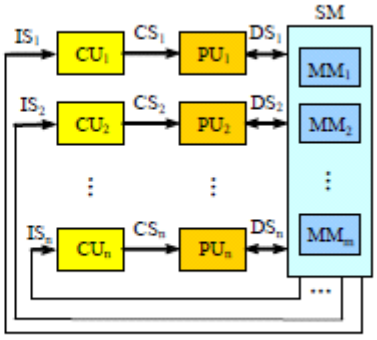
（a）SISD 计算机



（b）SIMD 计算机



（c）MISD 计算机



（d）MIMD 计算机

单处理器是SISD，常规多处理器是MIMD。

SPMD是一种在MIMD处理器上**编程的模型。**

SPMD ： 单程序多数据流。传统的MIMD 编程模型，其中一个程序运行在所有处理器之上。

向量机：

为了提高计算机的通用性，向量处理机需要同时具有处理标量和向量的功能。

向量体系结构的基本理念是从存储器中收集数据元，并将它们按顺序放到一大组寄存器中，然后在寄存器中使用流水化的执行单元对它们依次操作，最后将结果写回存储器。

向量体系结构的关键特征是拥有一组向量寄存器。

一个向量寄存器含有多个数据单元，方便使用流水执行单元。

在向量处理器中，每条向量指令只会在每个向量的起始数据元阻塞，在随后的数据元都会顺畅地通过流水线。因此，流水线阻塞在每次向量操作时只会发生一次，而不是每次

对向量数据元进行操作时都会发生一次。

与标量体系结构（常规指令集体系结构）相比：
主要在这几点：（6.3.3）

- 一条指令代表一个循环，编译器和硬件进行检查的次数减少，也不存在循环转移引起的控制相关。
- 如果取数据是连续的，延时只有一次。

与多媒体扩展相比：

- 向量体系结构的操作较多。
- 数据传输不必连续，支持按步长存取（ strided access）和变址存取（ indexed access ），前者是硬件每隔n 个存储器中的数据元读取一次，后者是按照数据项地址读取到向量寄存器中。变址存取也称作聚集分散（ gathers scatter ），变址的读取操作将内存中的数据元素聚集成连续的向量元素，变址的存储操作将向量元素分散到内存中。
- 支持不同数据位宽。

向量通道：一个或多个向量功能羊元与一部分向量寄存器。由为提高交通流量的高速公路的道路数启发而来，多个通道同时执行向量操作。

注意向量寄存器的巨大状态增加了上下文切换时间；向量存取产生的缺页故障难以处理； SIMD 指令也可以获得向量指令的部分优势。另外，只要指令级并行可以提供摩尔定律要求的性能提升，就没有理由要去改变体系结构的类型。

硬件多线程

MIMD使用多个线程和进程来使多个处理器处于忙碌状态，而硬件多线程允许多个线程以重叠的方式共享一个处理器的功能单元，因此为了共享资源，处理器必须为每个线程复制独立的状态。硬件必须具有以相对较快的速度切换钱程的能力，其中线程切换相对进程切换应该更加有效，线程切换可以是实时的，而进程切换一般需要数百个到数千个处理器周期。

硬件多线程： 在线程阻塞时处理器可切换到另一线程的实现。
进程： 一个进程包含一个或多个线程、地址空间和操作系统。因此一次进程切换通常需要操作系统的介入，但是线程切换不需要。
线程：一个线程包含程序计数器、寄存器状态和内存裁。它是一个轻量级的进程； 多个线程通常共享一个地址空间，而进程则不是。

实现方式：

- 细粒度多线程（ fine - grained multi threading ）
  在每条指令执行之后都进行线程切换，结果就是在多个线程之间交叉执行，通常以

循环的方式进行，每个时钟周期跳过处于阻塞状态的线程。

- 优点：同时隐藏由短阻塞和长阻塞引起的吞吐量损失，因为当一个线程阻塞时可以执行其他线程的指令。
- 缺点：降低了单个线程的执行速度， 因为就绪状态的线程会因为其他线程而延迟执行。

- 粗粒度多钱程（ coarse-grained multithreading）
  仅在一些高开销阻塞时（如最后一级缓存缺失）之后进行线程切换，单一线程发射指令。
  - 优点：不会降低单个线程的执行速度
  - 缺点：它在隐藏吞吐量损失的能力方面受限，特别是短阻塞。
- 同时多线程（simultaneous multithreading, SMT）
  利用多发射、动态调度微体系结构中的资源实现多线程，从而降低多线程的开销。

内存共享多处理器（SMP）

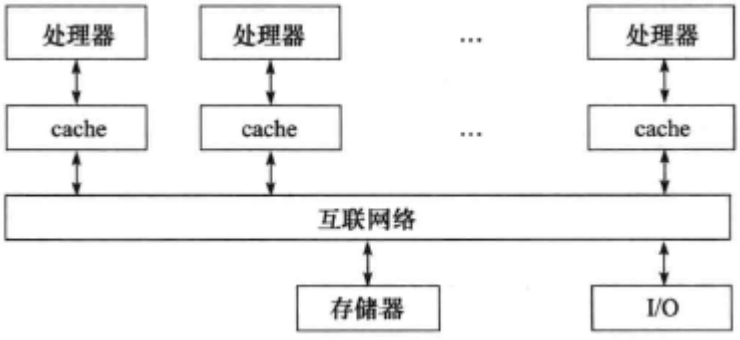
统一存储访问：无论访存的是哪个处理器，也无论访存的是哪个字，访存时间都大致相同的多处理器。

非统一存储访问：使用单一地址空间多处理器的一种类型，某些存储访存速度高于其他访存，访存速度与访问哪个处理器及访问哪个字相关。

两种基本结构是**集中式共享存储器多处理器**和**分布式共享存储器多处理器**

注意即使这些处理器共享同一个物理地址空间， 它们仍然可以在自己的虚拟地址空间中单独地运行程序。

e.g.



（互联网络在存储器和cache之间。）

约简：处理一个数据结构并返回单一值的函数。

图形处理单元

GPU和CPU的主要差别：

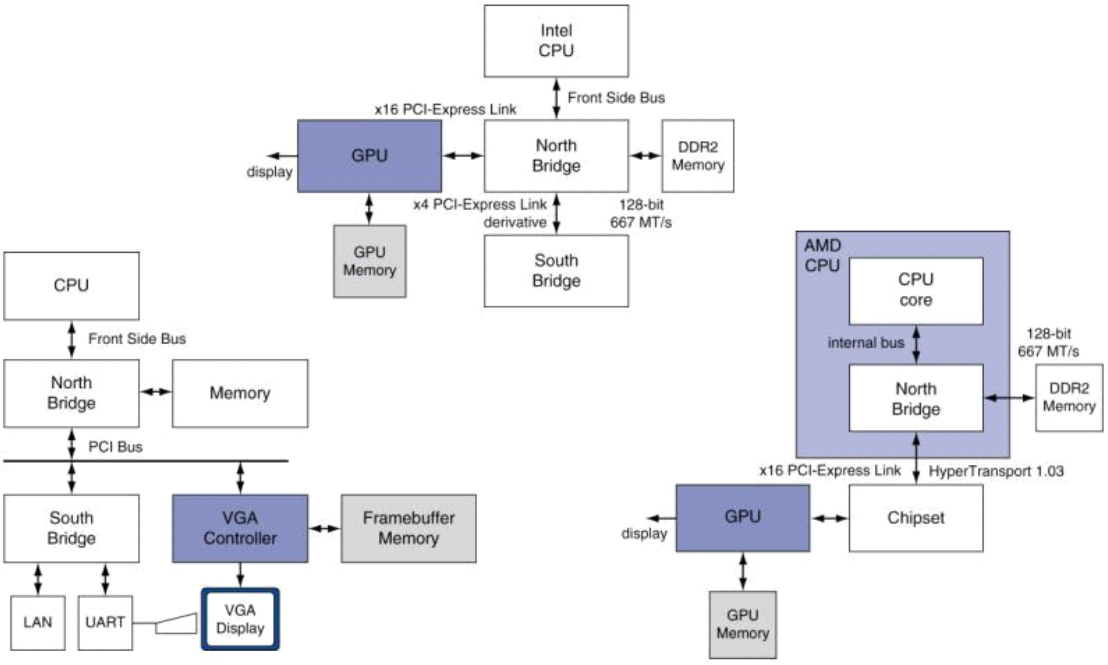
任务上的不同：

- GPU是补充CPU的加速器，因此不必执行CPU的全部任务。
- GPU解决的问题规模通常为几百MB到GB（较小）。（题外话，这过时了吧。）

设计上的不同：

- GPU不依赖多级缓存来隐藏访问存储器的长延迟，而是以来硬件多线程隐藏访存的延迟（即在数据到达之前执行其他请求）。
- GPU的主存是面向带宽的，而不是面向延迟的。
- GPU还有许多并行处理器（MIMD），对比CPU，GPU含有更多的线程和处理器。
- GPU的cache较小。

GPU位置



南桥：连接低速设备

北桥：连接高速设备

消息传递多处理器

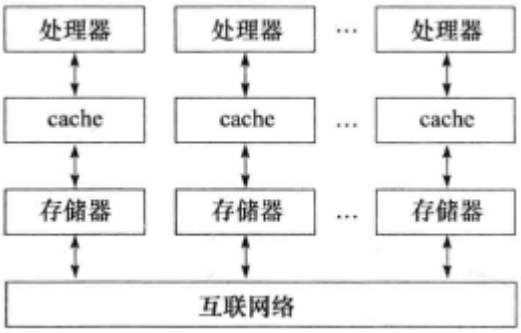
一个共享地址空间的方法是每个处理器具有自己私有的物理地址空间。这种处理器必须通过显式的消息传递进行通信，称为消息传递计算机。

消息传递： 通过显式发送和接收信息的方式在多传递多处理器。

发送消息例程： 具有私有存储器的机器中一个处理器将消息友送给另一个处理器的例程。

接收消息例程： 具有私有存储器的机器中一个处理器接收来自其他处理器消息的例程。

e.g.



（互联网络在存储器-处理器节点之间。）

集群

集群：通过标准的网络开关将VO 链接起来的计算机集合，以此来构建消息传递机制的多处理器。

成本低、快速可扩展能力、高可用性。

仓储级计算机(Warehou se-Scale Computer, WSC)

本质是上大型的集群，但是需要重新构建一些设施。

处理器网络拓补

性能度量：

总网络带宽，是每条链路带宽与链路数量的乘积，该度量表示带宽的峰值。

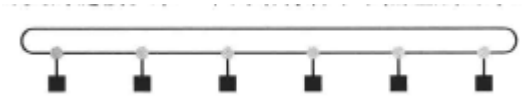
而一条总线的总网络带宽仅仅是该总线的带宽。

切分带宽(bisection bandwidth ）（接近最差情况的度量）。它的计算是通过将机器分割为两半，然后将跨越假想分割线的链路带宽加起来。

环：

环可以同时进行多个传输。

e.g.



总线：

允许一组连线向所有相连的节点发送广播。

全连接网络（fully connected network），每个处理器都与其它处理器有一个双向链路。

对全连接网络，总网络带宽是Px(P-1)/2 ，而切分带宽是 （ P/2)^2。