



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第1章 计算机系统结构的基础知识

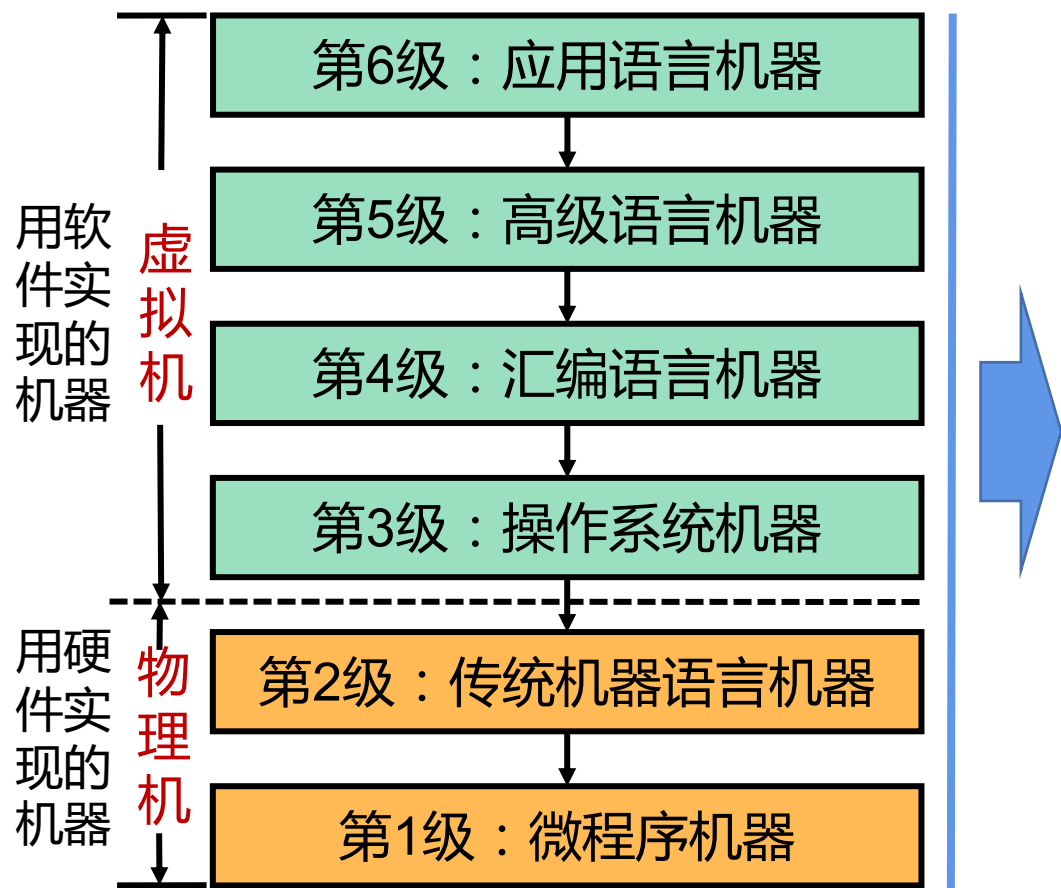
(复习)

2023年春季学期

计算机系统的层次结构

第 2 页

- 每一层以一种计算机语言为特征，计算机语言从低级向高级发展，高一级语言的语句相对于低一级语言功能更强，更便于应用，但又都以低级语言为基础



- 高级机器上的程序在低级机器上执行
 - 翻译与解释，或两者的结合
 - 上面3级：经常用翻译；下面3级：经常用解释
- 传统机器语言与微程序的区别
 - 微程序由硬件来解释执行
 - 无微程序技术，第2级指令系统由硬件直接执行
- 操作系统机器的执行机制
 - 传统机器级指令+操作系统级指令
- 虚拟机不一定由软件实现，些操作可以由硬件或固件实现

计算机系统结构的定义

第 3 页

1. **经典定义**：传统机器程序员看到的计算机属性，按照计算机系统的多级层次结构，不同级程序员所看到的计算机具有不同的属性（**透明性**）。
2. **广义定义**：指令集结构（Instruction Set Architecture）+ 计算机组织（Organization）+ 计算机实现（Implementation）
 - 计算机组成：计算机系统结构的逻辑实现
 - 计算机实现：计算机组成的物理实现
3. **指令系统设计与实现**：
 - 计算机系统结构：确定指令系统中是否有乘法指令
 - 计算机组成：乘法指令是用专用的乘法器还是用加法器经过多步操作实现
 - 计算机实现：乘法器、加法器的物理实现，如器件的选定及所用的微组装技术

系列机：由同一厂家生产的具有相同系统结构、但具有不同组成和实现的一系列不同型号的机器



计算机系统结构的分类

第 4 页

常见的计算机系统结构分类法有**3**种：

1. Flynn分类法：SISD、SIMD、MISD、MIMD

2. 冯氏分类法：

◆**字串位串**： $n = 1, m = 1$ ，第一代计算机发展初期的纯串行计算机

◆**字串位并**： $n > 1, m = 1$ ，传统的单处理机，同时处理单个字的多个位

◆**字并位串**： $n = 1, m > 1$ ，同时处理多个字的同一位（位片）。

◆**字并位并**： $n > 1, m > 1$ ，同时处理多个字的多个位。

3. Handler分类法 $t(\text{系统型号}) = (k \times k', d \times d', w \times w')$

1. 以经常性事件为重点

■以经常性事件为重点是计算机设计中最重要、最广泛采用的设计原则

2. Amdahl定律——评估方案优化效果

加快某部件执行速度所能获得的系统性能加速比，受限于该部件的执行时间占系统中总执行时间的百分比。

系统性能加速比：

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

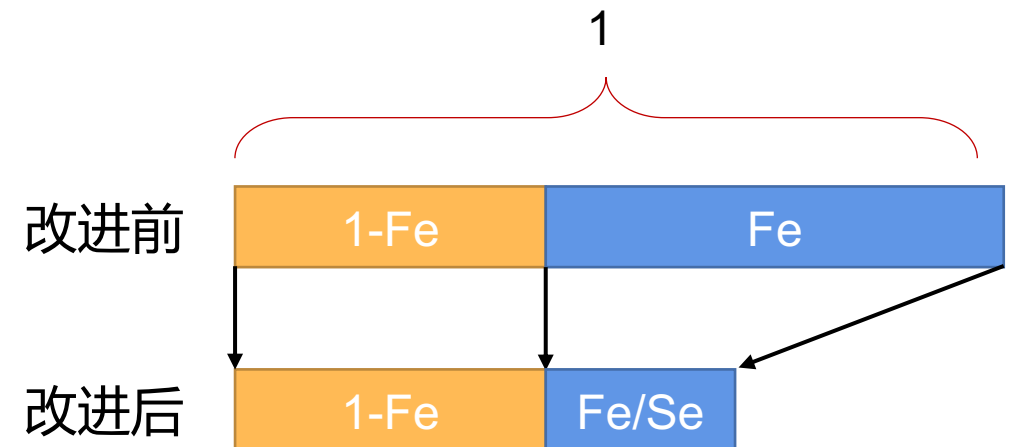
改进后程序的总执行时间 T_n

$$T_n = T_0 \left(1 - Fe + \frac{Fe}{Se} \right)$$

- T_0 : 改进前整个程序的执行时间
- $1 - Fe$: 不可改进比例

系统加速比 S_n 为改进前与改进后总执行时间之比:

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - Fe) + \frac{Fe}{Se}}$$



3. CPU性能公式

- 执行一个程序所需的CPU时间

$$\begin{aligned}\text{CPU时间} &= \text{执行程序所需的时钟周期数} \times \text{时钟周期时间} \\ &= \text{执行程序所需的时钟周期数} / \text{时钟频率}\end{aligned}$$

其中，时钟周期时间T是系统时钟频率的倒数 $1/f$ 。

- 每条指令执行的平均时钟周期数CPI (Cycles Per Instruction)

$$\text{CPI} = \text{执行程序所需的时钟周期数} / \text{IC}$$

其中，IC表示所执行的指令条数

- 程序执行的CPU时间可以写成

$$\text{CPU时间} = \text{IC} \times \text{CPI} \times \text{时钟周期时间}$$

CPU时间 = 执行程序所需的时钟周期数 × 时钟周期时间

$$= \sum_{i=1}^n (CPI_i \times IC_i) \times \text{时钟周期时间}$$

$$CPI = \frac{\text{时钟周期数}}{IC} = \frac{\sum_{i=1}^n (CPI_i \times IC_i)}{IC} = \sum_{i=1}^n (CPI_i \times \frac{IC_i}{IC})$$

其中： $\frac{IC_i}{IC}$ 反映了第*i*种指令在程序中所占的比例。



保持不变？

衡量计算机性能的唯一且可靠的标准是真实程序的执行时间

1. 执行时间和吞吐率：如何评测一台计算机的性能，与测试者看问题的角度有关。

- 用户关心的是：**单个程序的执行时间**（执行单个程序所花的时间很少）
- 数据处理中心的管理员关心的是：**吞吐率**（在单位时间里能够完成任务很多）

假设两台计算机为X和Y，X比Y快的意思是：对于给定任务，X的执行时间比Y的执行时间少。

X的性能是Y的n倍：
$$\frac{\text{执行时间Y}}{\text{执行时间X}} = n$$

执行时间与性能成反比：
$$n = \frac{\text{执行时间Y}}{\text{执行时间X}} = \frac{\frac{1}{\text{性能Y}}}{\frac{1}{\text{性能X}}} = \frac{\text{性能X}}{\text{性能Y}}$$

如何计算执行时间？

执行时间的计算：

1. 计算机完成某一任务所花费的**全部时间T**，

执行时间 $T = \text{CPU时间} + \text{磁盘访问时间} + \text{存储器访问} + \text{输入/输出开销} + \text{操作系统开销}$

2. **CPU时间**：CPU执行所给定的程序所花费的时间，不包含I/O等待时间以及运行其它程序的时间。

- **用户CPU时间**：用户程序所耗费的CPU时间。

- **系统CPU时间**：用户程序运行期间操作系统所耗费的CPU时间。

计算机系统的性能评测

第 11 页

	机器A	机器B	机器C
程序P1	1.00	10.00	20.00
程序P2	1000.00	100.00	20.00

从该表可以得出：

■ 执行程序1:

- A机的速度是B机的10倍
- A机的速度是C机的20倍
- B机的速度是C机的2倍

■ 执行程序2:

- B机的速度是A机的10倍
- C机的速度是A机的50倍
- C机的速度是B机的5倍

- 主要问题：计算机的配置基本相同，但不同计算机针对不同测试程序的测试结果是否一致？
- 为了公平地综合比较处理器性能，建立综合度量标准：

◆ 各程序的比重相同时

- 总执行时间
- 算术平均时间
- 调和平均时间

◆ 各程序的比重不同时

- 加权执行时间
- 标准化执行时间

计算机系统的性能评测

总执行时间

$$S_m = \sum_{i=1}^n T_i$$

算术平均执行时间

$$A_m = \frac{1}{n} \sum_{i=1}^n T_i$$

调和平均执行时间

$$H_m = \frac{n}{\sum_{i=1}^n \frac{1}{T_i}}$$

其中： T_i 为第*i*个程序的执行时间， n 为程序数， $R_i=1/T_i$ 。

	机器A	机器B	机器C
程序P1	1秒	10秒	20秒
程序P2	1000秒	100秒	20秒
总执行时间S	1001秒	110秒	40秒
算术平均执行时间A	500.5秒	极大值 影响55秒	20秒
调和平均执行时间H	2秒	极小值 影响18秒	20秒

加权算术平均执行时间

$$A_m = \sum_{i=1}^n W_i \times T_i$$

其中： T_i 为第*i*个程序的执行时间， n 为程序数， W_i 为第*i*个程序的权重， $\sum_{i=1}^n W_i = 1$

	机器A	机器B	机器C		W_1	W_2
程序P1	1秒	10秒	20秒	组合1	0.5	0.5
程序P2	1000秒	100秒	20秒	组合2	0.909	0.091
算术平均 A_1	500.5秒	55秒	20秒	组合3	0.999	0.001
算术平均 A_2	91.91秒	18.19秒	20秒			
算术平均 A_3	2秒	10.09秒	20秒			

加权调和平均执行时间

$$H_m = \frac{\sum_{i=1}^n W_i}{\sum_{i=1}^n \frac{W_i}{T_i}}$$

其中： T_i 为第*i*个程序的执行时间， n 为程序数， W_i 为第*i*个程序的权重

	机器A	机器B	处理器C
程序P1	1秒	10秒	20秒
程序P2	1000秒	100秒	20秒
调和平均A ₁	2秒	18.18秒	20秒
调和平均A ₂	1.1秒	10.89秒	20秒
调和平均A ₃	1秒	10.01秒	20秒

	W ₁	W ₂
组合1	0.5	0.5
组合2	0.909	0.091
组合3	0.999	0.001

标准化执行时间的计算方法：先将各程序的执行时间对一台参考机器进行标准化，然后取标准化执行时间的平均值（可以是算术平均值，也可以是几何平均值）

$$A_m = \frac{1}{n} \sum_{i=1}^n ETR_i$$

算术平均

$$G_m = \sqrt[n]{\prod_{i=1}^n ETR_i}$$

几何平均

ETR_i ：为第*i*个程序对参考机器标准后的执行时间， n ：为程序数

1. **并行性**：计算机系统在同一时刻或者同一时间间隔内进行多种运算或操作。
只要在时间上相互重叠，就存在并行性。
 - **同时性**：两个或两个以上的事件在同一时刻发生。
 - **并发性**：两个或两个以上的事件在同一时间间隔内发生。
2. 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：单条指令中各微操作之间的并行
 - 指令级并行：并行执行两条或两条以上的指令
 - 线程级并行：并行执行两个或两个以上的线程
 - 任务级或过程级并行：并行执行两个或两个以上的过程或任务（程序段）
 - 作业或程序级并行：并行执行两个或两个以上的作业或程序

三种途径：

1. 时间重叠

- 引入时间因素，让多个处理过程在时间上相互错开，轮流重叠地使用同一套硬件设备的各个部分，以加快硬件周转而赢得速度。

2. 资源重复

- 引入空间因素，以数量取胜。通过重复设置硬件资源，大幅度地提高计算机系统的性能。

3. 资源共享

- 这是一种软件方法，它使多个任务按一定时间顺序轮流使用同一套硬件设备。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

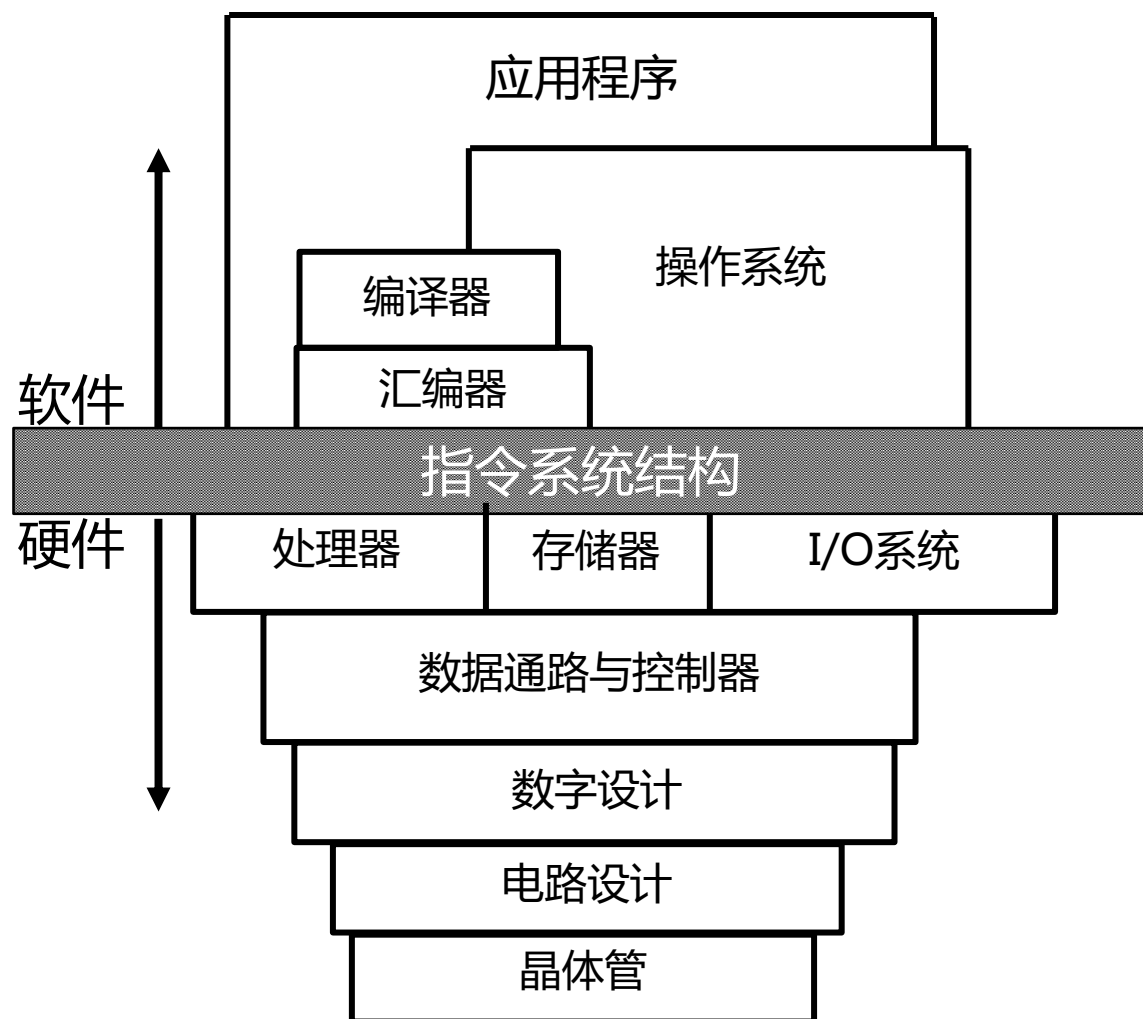
第2章 指令系统的设计

(复习)

2023年春季学期

指令系统结构的分类

第 19 页



指令分类：

- 1、堆栈
- 2、累加器
- 3、寄存器-存储器结构
- 4、寄存器-寄存器结构（现代处理器主流）

物理地址空间的信息存放方式：

- 1、允许跨边界存储，存储效率高，访问速度低
- 2、不允许跨边界存储，存储效率低，访问速度高

七种基本寻址方式：

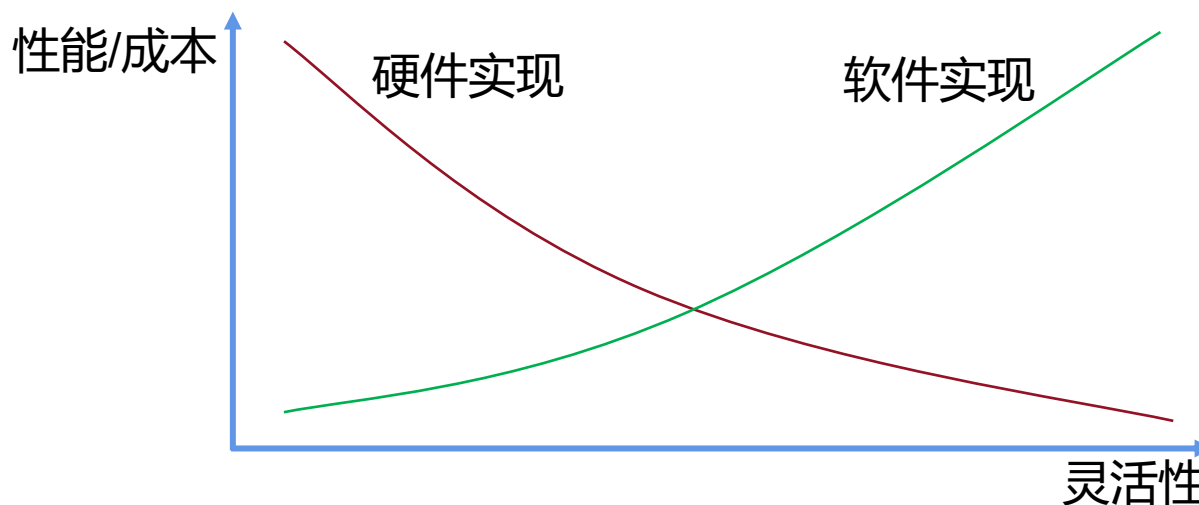
立即数寻址：	立即数 → 寄存器
直接寻址：	[偏移地址] → 寄存器
寄存器寻址：	寄存器 → 寄存器
寄存器间接寻址：	[基址寄存器/变址寄存器] → 寄存器
寄存器相对寻址：	[基址寄存器/变址寄存器+偏移量值] → 寄存器
基址变址寻址：	[基址寄存器+变址寄存器] → 寄存器
相对基址变址寻址：	[基址寄存器+变址寄存器+偏移量值] → 寄存器

1. 指令系统的设计

- 首先考虑所应实现的基本功能，确定哪些基本功能应该由硬件实现，哪些功能由软件实现比较合适，具体包括：
 - ✓ 指令的功能设计
 - ✓ 指令格式的设计

2. 在确定哪些基本功能用硬件来实现时，主要考虑3个因素：速度、成本、灵活性。

- 硬件实现的特点：
 - ✓ 速度快、成本高、灵活性差
- 软件实现的特点：
 - ✓ 速度慢、价格便宜、灵活性好

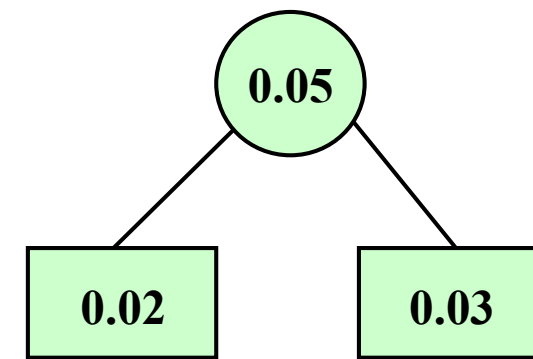


3. 对指令系统的基本要求：**完整性、规整性、正交性、高效率、兼容性**

- 完整性：在一个有限可用的存储空间内，对于任何可解的问题，编制计算程序时，指令系统所提供的指令足够使用，要求指令系统功能齐全、使用方便。
- 规整性：主要包括对称性和均匀性
 - 对称性：所有与指令系统有关的存储单元的使用、操作码的设置等都是对称的
 - 均匀性：指对于各种不同的操作数类型、字长、操作种类和数据存储单元，指令的设置都要同等对待
- 正交性：在指令中各个不同含义的字段，如操作类型、数据类型、寻址方式字段等，在编码时应互不相关、相互独立。
- 高效率：指指令的执行速度快、使用频度高。
- 兼容性：主要是要实现向后兼容，指令系统可以增加新指令，但不能删除指令或更改指令的功能——**指令数越来越多的原因**

■ 哈夫曼编码的实现方法——构造哈夫曼树

- 将各事件按其使用频度从小到大依次排列；
- 每次从中选择两个频度值最小的结点，将其合并成一个新的结点，并把新结点画在所选结点的上面，然后用两条边把新结点分别与那两个结点相连。新结点的频度值是所选两个结点的频度值的和。
- 把新结点与其他剩余未结合的结点一起，再以上面的步骤进行处理，反复进行，直到全部结点都结合完毕、形成根结点为止。
- 操作码优化的程度可以用信息熵来衡量



画哈夫曼树的一个基本步骤

$$H = - \sum_{i=1}^n p_i \log_2 p_i$$

表示用二进制编码表示n个码点时理论上的最短平均编码长度

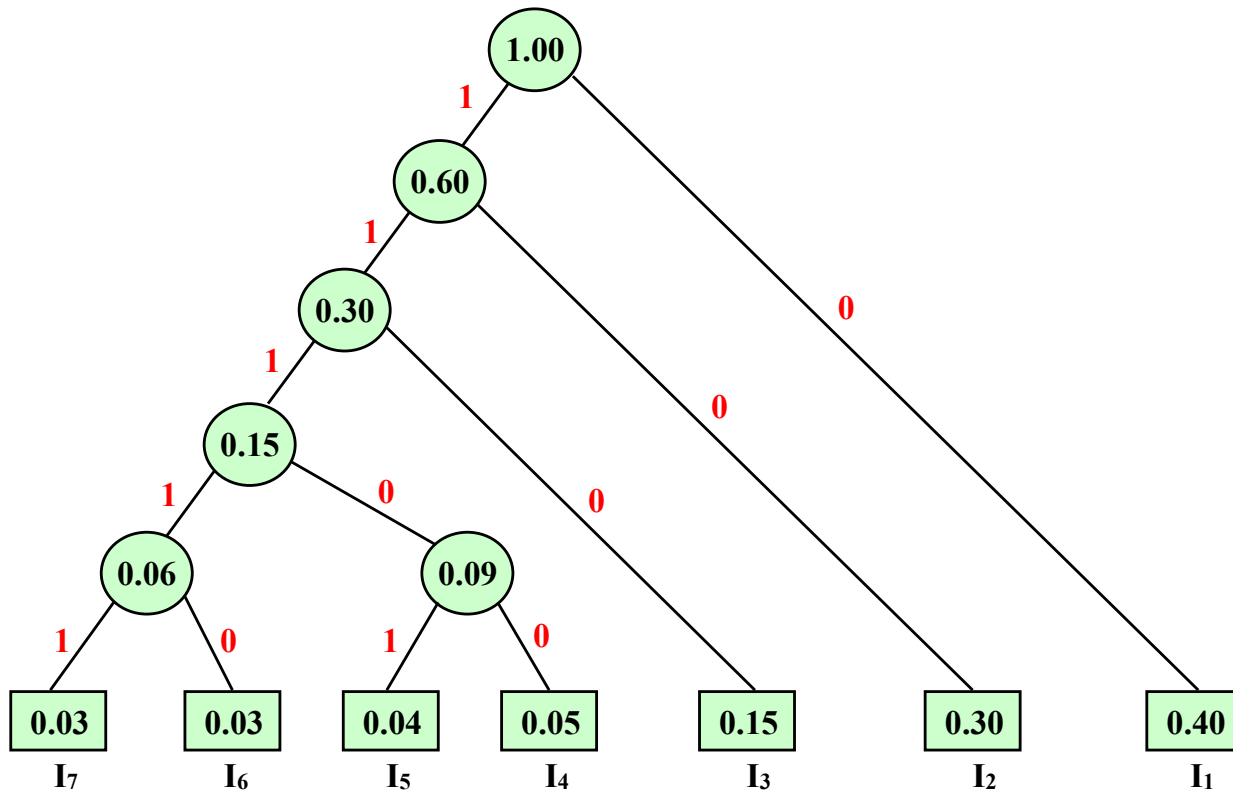
该哈夫曼编码的平均码长是：

$$L = \sum_{i=1}^7 p_i l_i = 2.20$$

其信息冗余量为：

$$\frac{2.20 - 2.17}{2.20} \approx 1.36\%$$

优缺点：可以减少操作码的平均位数，但所获得的编码是变长度的，不规整，不利于硬件处理。



哈夫曼树实例

■ m/n扩展编码

- 为了便于分级译码，一般都采用等长扩展码。在早期的计算机上，例如：15/15/15法和8/64/512法

✓ 选用哪种编码法取决于指令使用频度 p_i 的分布。若在前15种指令中 p_i 的值都比较大，但在后30种指令后急剧减少，则应选择15/15/15法；若 p_i 的值在前8种指令中较大，之后的64种指令的 p_i 值也不太低，则应选择8/64/512法。

✓ 衡量标准：看哪种编码法能使平均码长最短。

15 {
0000
0001
⋮
1110
1111 0000
1111 0001
⋮
1111 1110
1111 1111 0000
1111 1111 0001
⋮
1111 1111 1110

15/15/15 编码法

8 {
0 000
0 001
⋮
0 111
64 {
1 000 0 000
1 000 0 001
⋮
1 111 0 111
512 {
1 000 1 000 0 000
1 000 1 000 0 001
⋮
1 111 1 111 0 111

8/64/512 编码法

■ 定长操作码

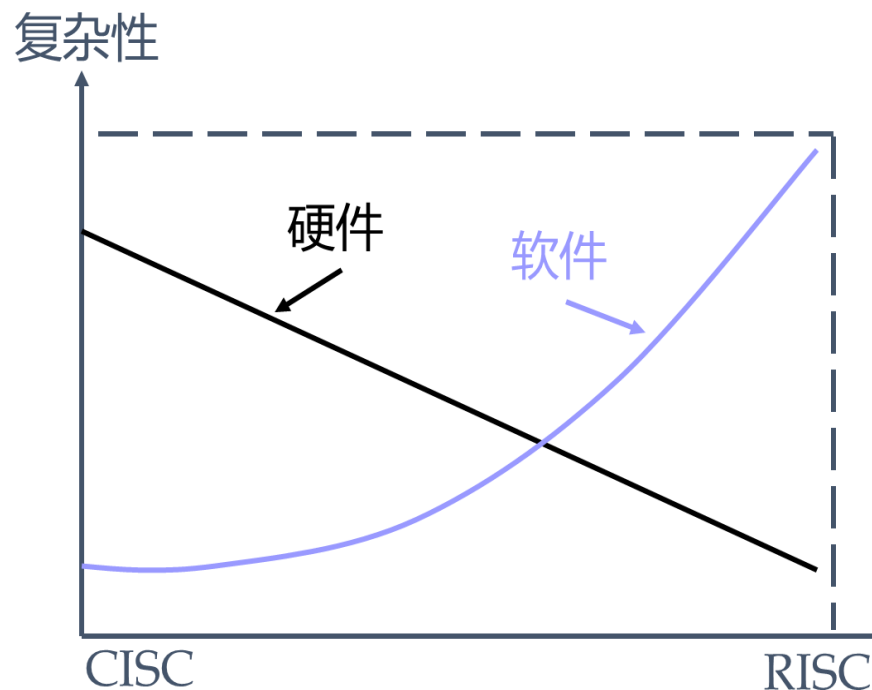
- 固定长度的操作码：所有指令的操作码都是同一的长度（如8位）。许多计算机都采用（特别是RISC结构的计算机）
- 保证操作码的译码速度、减少译码的复杂度。
- 以程序的存储空间为代价来换取硬件实现上的好处。

0001100	rs2	rs1	rm	rd	101001	R fdiv.s
0001100	00000	rs1	rm	rd	101001	R fsqrt.s
0010000	rs2	rs1	000	rd	101001	R fsgnj.s
0010000	rs2	rs1	001	rd	101001	R fsgnjn.s
0010000	rs2	rs1	010	rd	101001	R fsgnjx.s
0010100	rs2	rs1	000	rd	101001	R fmin.s

沿CISC方向发展和改进指令系统

第 28 页

复杂而高效



简约而不简单

思路

特点

应用

CISC体系

为了便于软件编程和提高程序运行速度，增加可实现复杂功能的指令和多种灵活的编址方式

- 1) 编译优化、程序代码量低
- 2) 指令执行效率高
- 3) 处理器性能高

通用计算机系统
(PC、笔记本、服务器、集群、巨型机等)

RISC体系

为了便于硬件实现和降低处理器实现成本，只实现使用频率很高的少数指令与规则的编址方式

- 1) 易于硬件实现
- 2) 易于流水实现与并行处理
- 3) 指令与数据存储效率高

嵌入式计算机系统
(手机、工业控制、医疗电子)

1. CISC指令系统的一大特点： 指令数量多、功能多样
2. 增强指令功能主要是从以下3个方面着手：
 - 面向目标程序增强指令功能
 - 面向高级语言的优化实现来改进指令系统
 - 面向操作系统的优化实现改进指令系统

2. 设计RISC机器遵循的原则，指令条数少、指令功能简单，方法如下：

- 指令系统：选取使用频度高的指令，在此基础上补充一些最有用的指令；
- 寻址机制：采用简单又统一的指令格式，并减少寻址方式；一般来说，指令字长都为32位或64位；
- 执行机制：采用流水线机制，指令的执行在单个机器周期内完成；
- 访存机制：采用load-store结构，只有load和store指令才能访问存储器，其它指令的操作都是在寄存器之间进行；
- 实现机制：大多数指令都采用硬连逻辑来实现；
- 编译机制：强调优化编译器的作用，为高级语言程序生成优化的代码。

■ 系统结构设计者首先要解决的问题：数据类型与表示

- 数据类型和表示是软硬件主要界面之一。
- 数据类型：面向应用、面向软件系统所处理的各种数据结构。例如定点、浮点、十进制、字符、字符串、逻辑、堆栈和向量等。
- 数据表示：硬件结构能够识别、指令系统可以直接调用的那些结构。

■ 确定数据类型和数据表示的原则：

- 一是缩短程序的运行时间；
- 二是减少CPU与主存储器之间的通信量；
- 三是这种数据表示的通用性和利用率。

数据类型和数据表示在不断发展，例如矩阵、树、图、表及自定义数据类型等。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第3章 流水线技术

(复习)

2023年春季学期

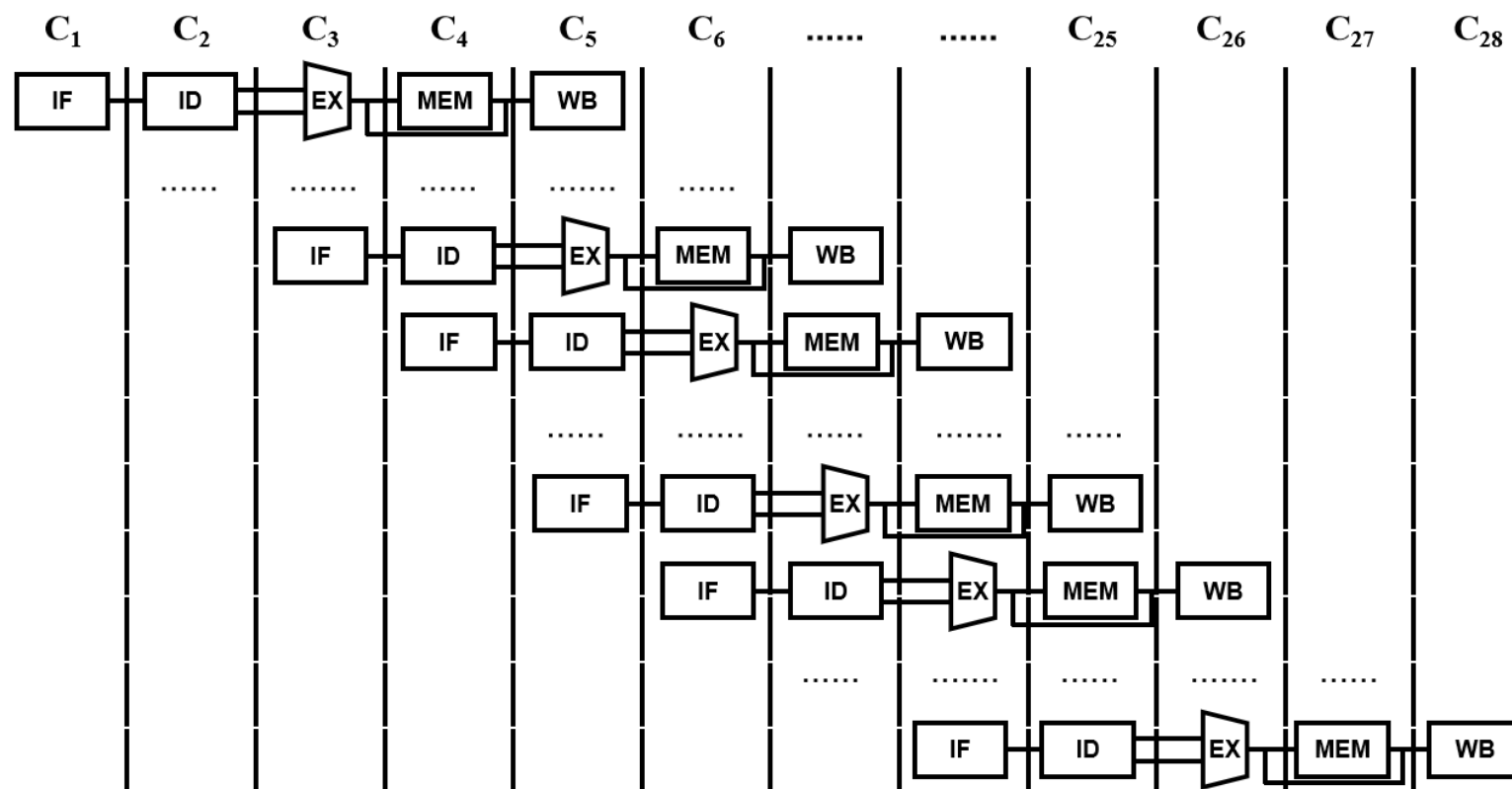
流水线的实现前提：

■ 空间并行性

- ◆ 指令操作可以由多个独立的操作部件共同完成

■ 时间并行性

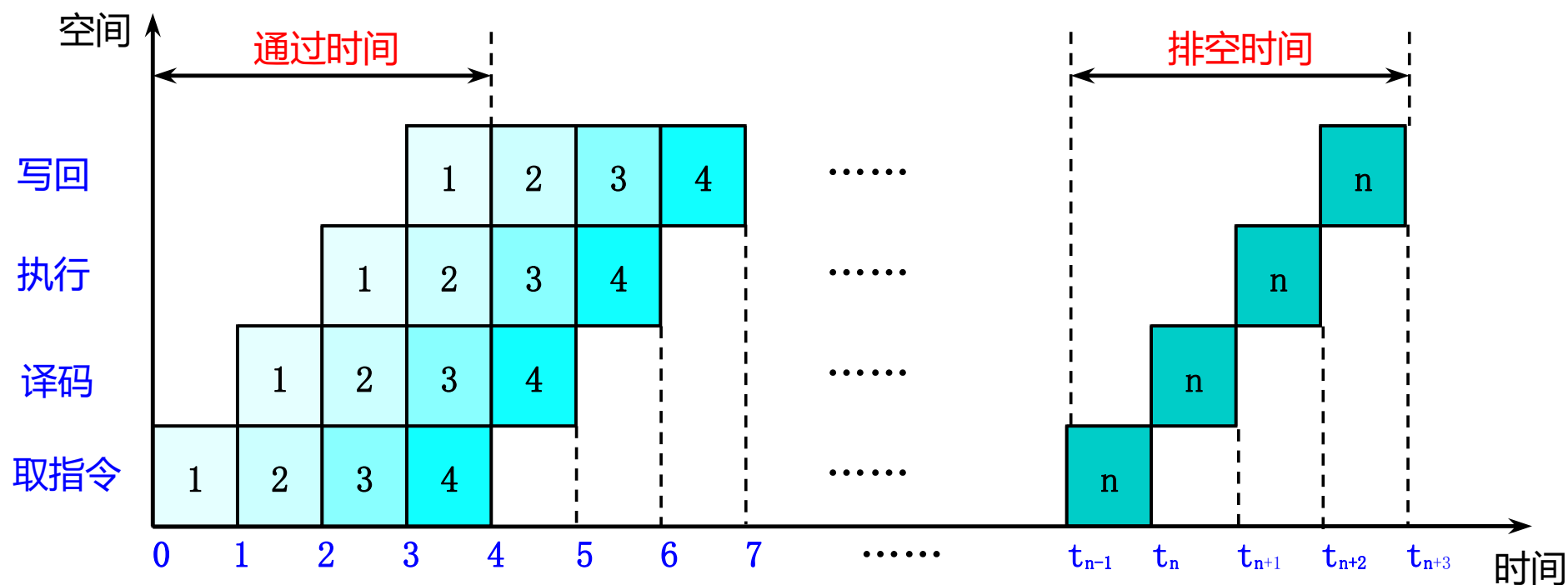
- ◆ 不同指令可以分时使用同一个部件的不同部分



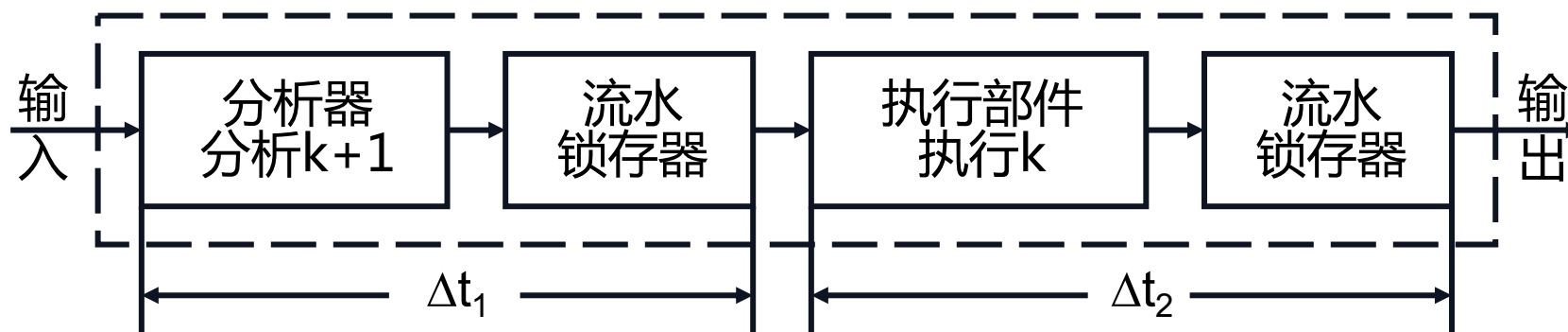
5段指令流水线 → **理想情况下**速度提高5倍

如何分析不理想情况下的流水线执行的实际效果？

- **时一空图**从时间和空间两个方面描述了流水线的工作过程
- 时一空图中横坐标代表时间，纵坐标代表流水线的各个阶段指令执行流水线的时空图



■ 连接图



- **功能段**：流水线的每一个阶段称为流水步、流水步骤、流水段、流水线阶段、流水功能段、流水级、流水节拍等。
- **流水寄存器**：在每一个流水段的末尾或开头必须设置一个寄存器，称为流水寄存器、流水锁存器等。
 - 作用：在相邻的两段之间传送数据，以保证提供后面要用到的信息，并把各段的处理工作相互隔离。

■ 按照流水技术的应用等级不同

- 部件级流水线：运算操作分段执行
- 处理器级流水线：指令分段执行
- 系统级流水线：程序（任务）分段执行

■ 按照流水线所完成的功能来分类

- 单功能流水线
- 多功能流水线
 - ✓ 静态流水线
 - ✓ 动态流水线（线性流水线与非线性流水线）

■ 按照任务流入和流出的顺序是否相同进行分类

- 顺序流水线
- 乱序流水线

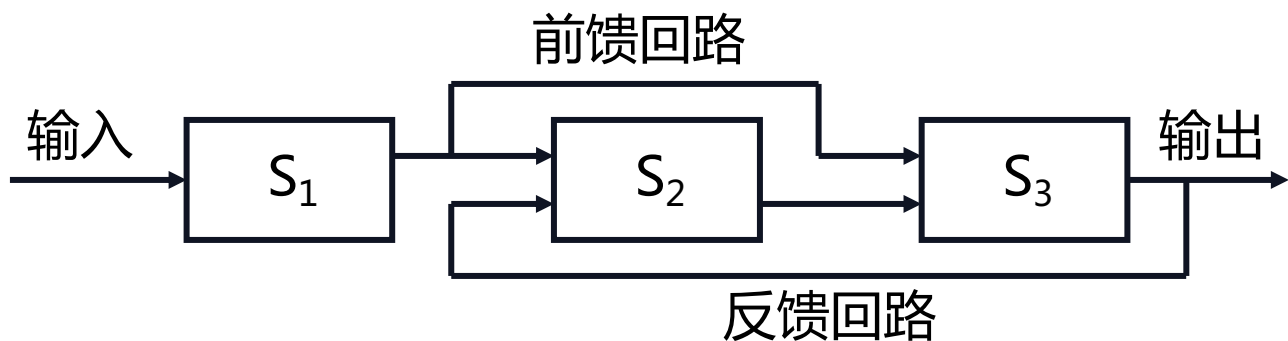
■ 按照处理数据对象来进行分类

- 标量流水线
- 向量流水线

非线性流水线

第 37 页

- 非线性流水线必须用**连接图**和**预约表**共同表示，一条非线性流水线可以对应有很多张预约表，同样一张预约表实际上仅表示了一条非线性流水线的一种工作方式



一种简单的非线性流水线

	1	2	3	4
S1	×			
S2			×	
S3		×		×

	1	2	3	4	5
S1	×				
S2		×		×	
S3			×		×

对应的两种预约表

■ 流水线的实际吞吐率

$$TP = \frac{n}{(m + n - 1)\Delta t}$$

■ 最大吞吐率

$$TP = \lim_{n \rightarrow \infty} \frac{n}{(m + n - 1)\Delta t} = \frac{1}{\Delta t_{\max}}$$

■ 最大吞吐率与实际吞吐率的关系

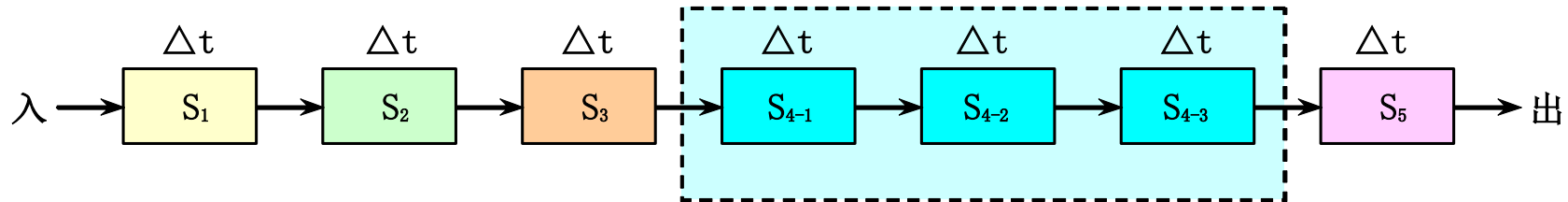
$$TP = \frac{n}{m + n - 1} TP_{\max}$$

只有当 $n \gg m$ 时，
才有 $TP \approx TP_{\max}$

■解决流水线瓶颈问题的常用方法

◆ 方法一：细分瓶颈段

例如：对前面的5段流水线把瓶颈段 S_4 细分为3个子流水线段： S_{4-1} ， S_{4-2} ， S_{4-3}

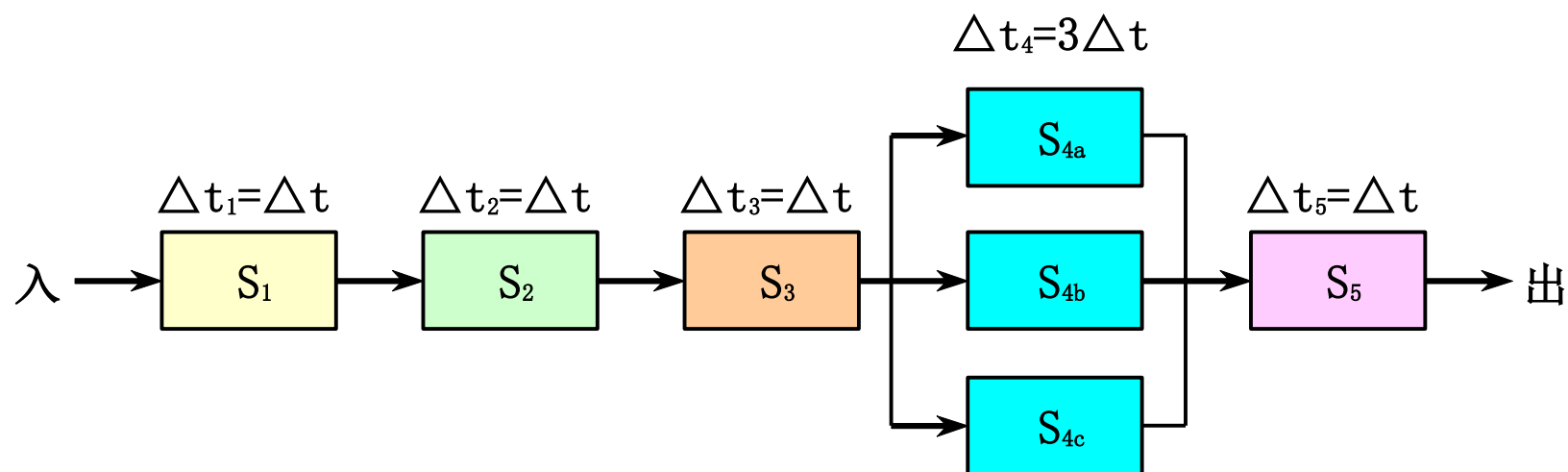


改进后的流水线的吞吐率：
$$TP_{\max} = \frac{1}{\Delta t}$$

■解决流水线瓶颈问题的常用方法

◆ 方法二：重复设置瓶颈段

- 缺点：控制逻辑比较复杂，所需的硬件增加了。
- 例如：对前面的5段流水线重复设置瓶颈段 S_4 ： S_{4a} ， S_{4b} ， S_{4c}



■ 流水线各段时间相等（都是 Δt ）

- ◆ 一条 m 段流水线完成 n 个连续任务所需要的时间为：

$$T_m = (m + n - 1)\Delta t$$

- ◆ 顺序执行 n 个任务所需要的时间：

$$T_s = nm\Delta t$$

- ◆ 流水线的实际加速比为：

$$S = \frac{nm}{m + n - 1} \quad \longrightarrow \quad S = \lim_{n \rightarrow \infty} \frac{nm}{m + n - 1} = S_{\max}$$

当 $n \gg m$ 时, $S \approx m$

思考：流水线的段数愈多愈好？

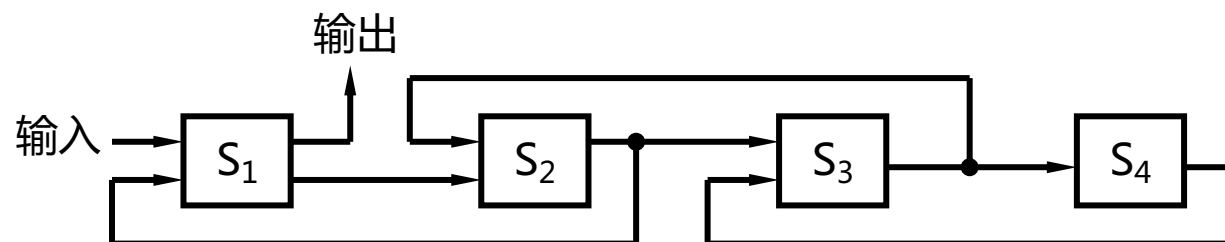
假设在流水线各段的执行时间不相等，输入到流水线中的任务是连续的理想情况下。

■ 实际效率（功能段权值相等）

$$E = \frac{n \cdot \sum_{i=1}^m \Delta t_i}{m \cdot \left[\sum_{i=1}^m \Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \dots, \Delta t_m) \right]}$$

单功能非线性流水线的调度

第 43 页



时间 流水段	1	2	3	4	5	6	7
S1	×						×
S2		×				×	
S3			×		×		
S4				×			

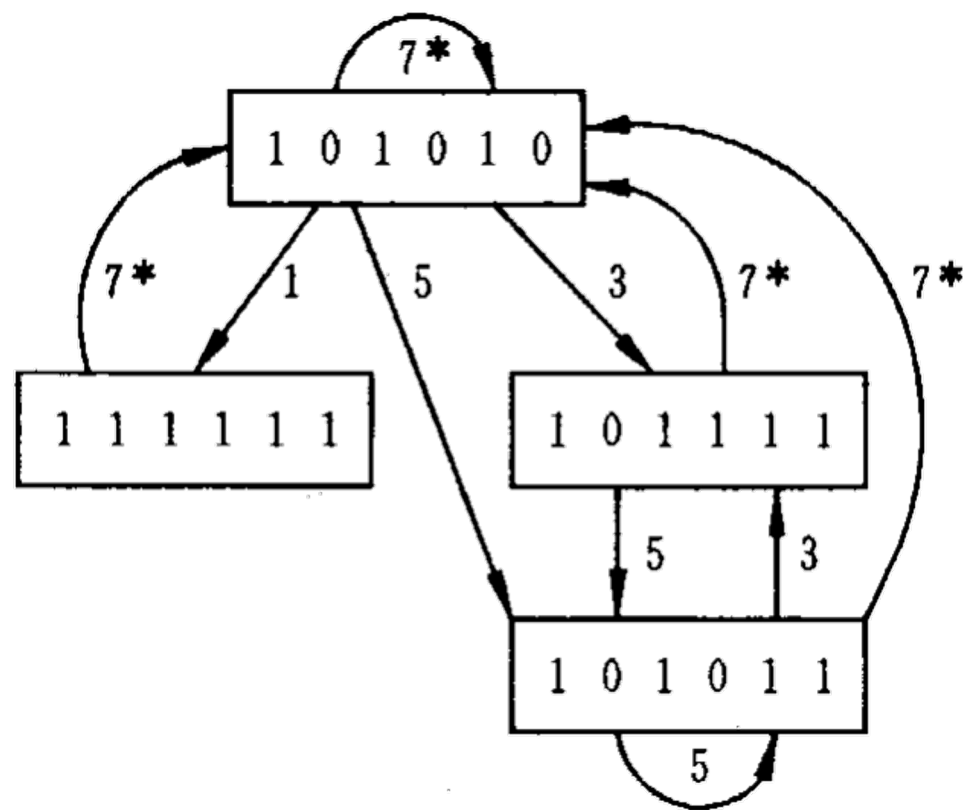
求最小启动循环。

- 步骤一：由预约表得到禁止表F
- 步骤二：由禁止表F得到冲突向量C
- 步骤三：由冲突向量构造调度流水线的状态转换图
- 步骤四：在状态图中找出可用启动距离，并计算平均启动距离
- 步骤五：找出平均启动距离最小的启动循环或恒定循环

单功能非线性流水线的调度

第 44 页

由冲突向量构造调度流水线的状态转换图



$\text{SHR}^{(j)} (C_0) \vee C_0$
 其中： $\text{SHR}^{(j)}$ 表示逻辑右移j位

$$\begin{aligned}
 \text{SHR}^{(1)} (101010) &= 010101 \\
 (010101) \vee (101010) &= 111111 \\
 \text{SHR}^{(7)} (111111) &= 000000 \\
 (000000) \vee (101010) &= 101010
 \end{aligned}$$

第一组

$$\begin{aligned}
 \text{SHR}^{(3)} (101010) &= 000101 \\
 (000101) \vee (101010) &= 101111 \\
 \text{SHR}^{(5)} (101111) &= 000001 \\
 (000001) \vee (101010) &= 101011 \\
 \text{SHR}^{(3)} (101011) &= 000101 \\
 (000101) \vee (101010) &= 101111
 \end{aligned}$$

第二组

- **相关**：两条指令之间存在某种依赖关系。

如果两条指令相关，则它们就有可能不能在流水线中重叠执行或者只能部分重叠执行。

- 相关有3种类型

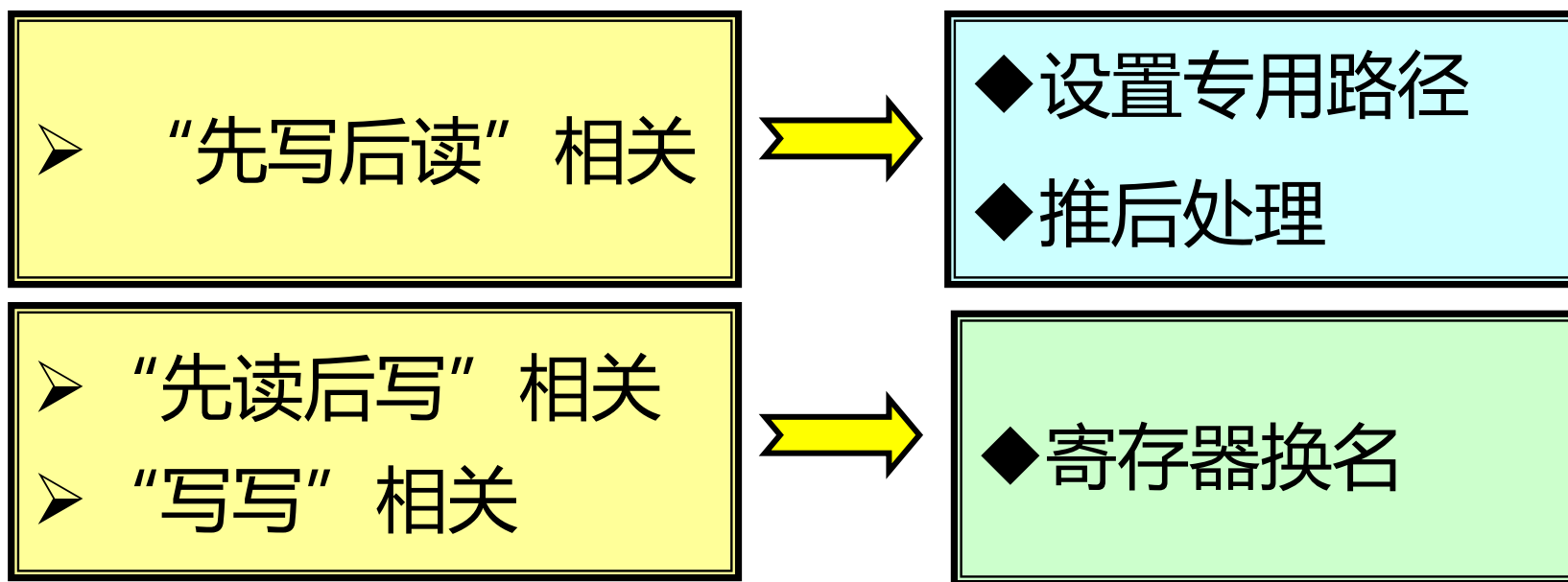
- **数据相关（也称真数据相关）**
- **名相关**
- **控制相关**

- 流水线冲突是指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行。
- 流水线冲突有3种类型：
 - 结构冲突：因硬件资源满足不了指令重叠执行的要求而发生的冲突。
 - 数据冲突：当指令在流水线中重叠执行时因需要用到前面指令的执行结果而发生的冲突。
 - 控制冲突：流水线遇到分支指令和其它会改变PC值的指令所引起的冲突。
- 带来的几个问题：
 - 导致错误的执行结果。
 - 流水线可能会出现停顿，从而降低流水线的效率和实际的加速比。

流水线冲突

第 47 页

- “先写后读” 相关在流水线顺序执行和乱序执行时都可能发生，
“先读后写” 相关和 “写写” 相关只有在流水线乱序执行时才可能发生，而 “读读” 相关无需处理。



■ 控制冲突

■ 执行分支指令的结果有两种：

- 分支成功：PC值改变为分支转移的目标地址。在条件判定和转移地址计算都完成后，才改变PC值。
- 不成功或者失败：PC的值保持正常递增，指向顺序的下一条指令。

■ 处理分支指令最简单的方法：“冻结”或者“排空”流水线

- 优点：简单

- 前述5段流水线中，改变PC值是在MEM段进行的。给流水线带来了3个时钟周期的延迟

■ 3种通过软件（编译器）来减少分支延迟的方法

- 预测分支失败
- 预测分支成功
- 延迟分支



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第四章 向量处理器

(复习)

2023年春季学期

■ 发挥向量处理机的计算优势：

关键点一：从标量表示到向量数据的表示方式：

1. 等间距向量表示法
2. 带位移量的向量表示法
3. 稀疏向量表示法

关键点二：在向量处理机上实现有3种处理方式：

1. **横向处理方式**，又称水平处理方式，横向加工方式等，向量计算是按行的方式从左至右横向地进行
2. **纵向处理方式**，又称垂直处理方式，纵向加工方式等，向量计算是按列的方式自上而下纵向地进行
3. **纵横处理方式**，又称分组处理方式，纵横加工方式等，横向处理和纵向处理相结合的方式

向量处理机的结构

第 51 页

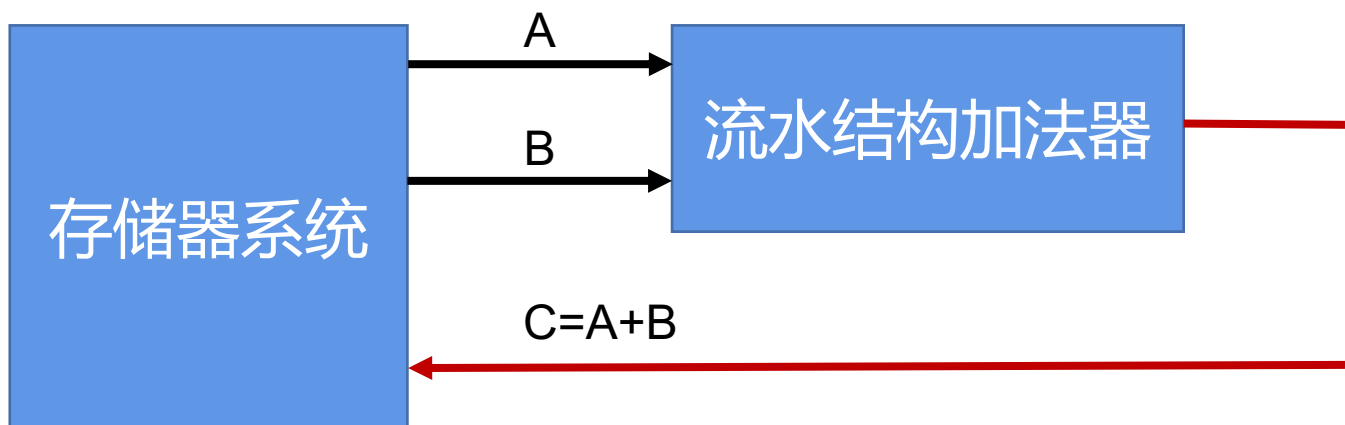
向量处理机的基本思想是把两个向量的对应分量进行运算，产生一个结果向量。

例：如果A、B、C都是向量，各有N个元素，则一台向量处理机能够完成如下运算：

$$C_i = A_i + B_i$$

$0 \leq i \leq N-1$ ，其中A、B、C用分量形式表示

采用流水线运算部件实现上述运算：



如果要求取操作数、运算、写结果回存储器在一个时钟周期内完成，则要求存储系统能在一个时钟周期内读出两个操作数和写回一个运行结果。



存储器系统的带宽至少应3倍于一般的存储器系统

- 向量处理机的结构由所采用的向量处理方式决定，有两种典型结构
 - **存储器-存储器型结构**：纵向处理方式采用，利用几个独立的存储器模块来支持向对独立的数据并发访问
 - ✓ 多个独立得到存储模块并行工作
 - ✓ 处理机结构简单，对存储系统的访问速度要求很高
 - **寄存器-寄存器型结构**：分组处理方式采用，构造一个具有所要求带块的高速中间寄存器，实现高速中间寄存器与主寄存器之间的快速数据交换
 - ✓ 运算通过向量寄存器实现，需要大量高速寄存器
 - ✓ 对存储系统访问速度要求低

“存储器-存储器”结构

第 53 页

解决访问冲突的方法之一：把存储体的个数 n 选为质数，且 $n \geq$ 向量长度，变址位移量就必然与 n 互质，一维数组的访问冲突自然也就不存在了。

- 例如美国研制的BSP巨型计算机的存储体个数为17，我国研制的银河巨型计算机的存储体个数为37。

$m \times L$ 不能整除 n



地址 $A_i + m \times L$ ($L=1 \sim 16$) 访问的存储器与地址 A_i 访问的存储器是一致的概率是0

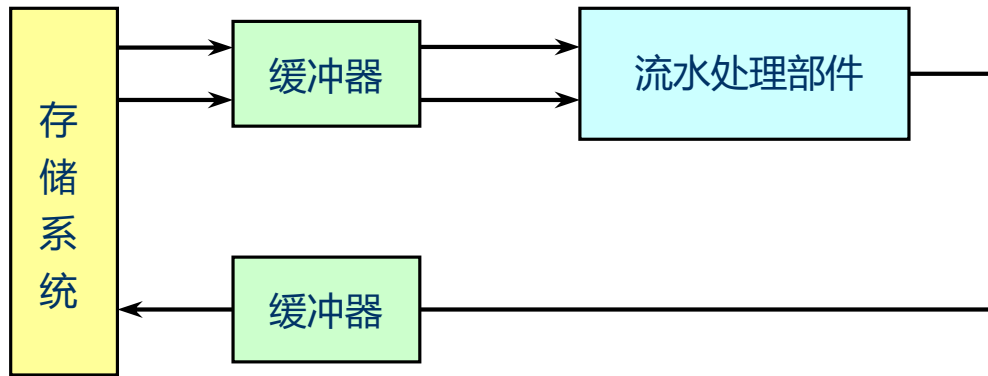
体内地址	0号体	1号体	2号体	3号体	4号体
0	A0	A1	A2	A3	A4
1	A5	A6	A7	A8	A9
2	A10	A11	A12	A13	A14
3	A15	A16	A17	A18	A19

存在问题：虽然采用质数个存储体能够缓解访问存储器的冲突，但是数据经过多次运算之后，在存储体中的分布必然发生改变，无法克服访问冲突问题。

“存储器-存储器”结构

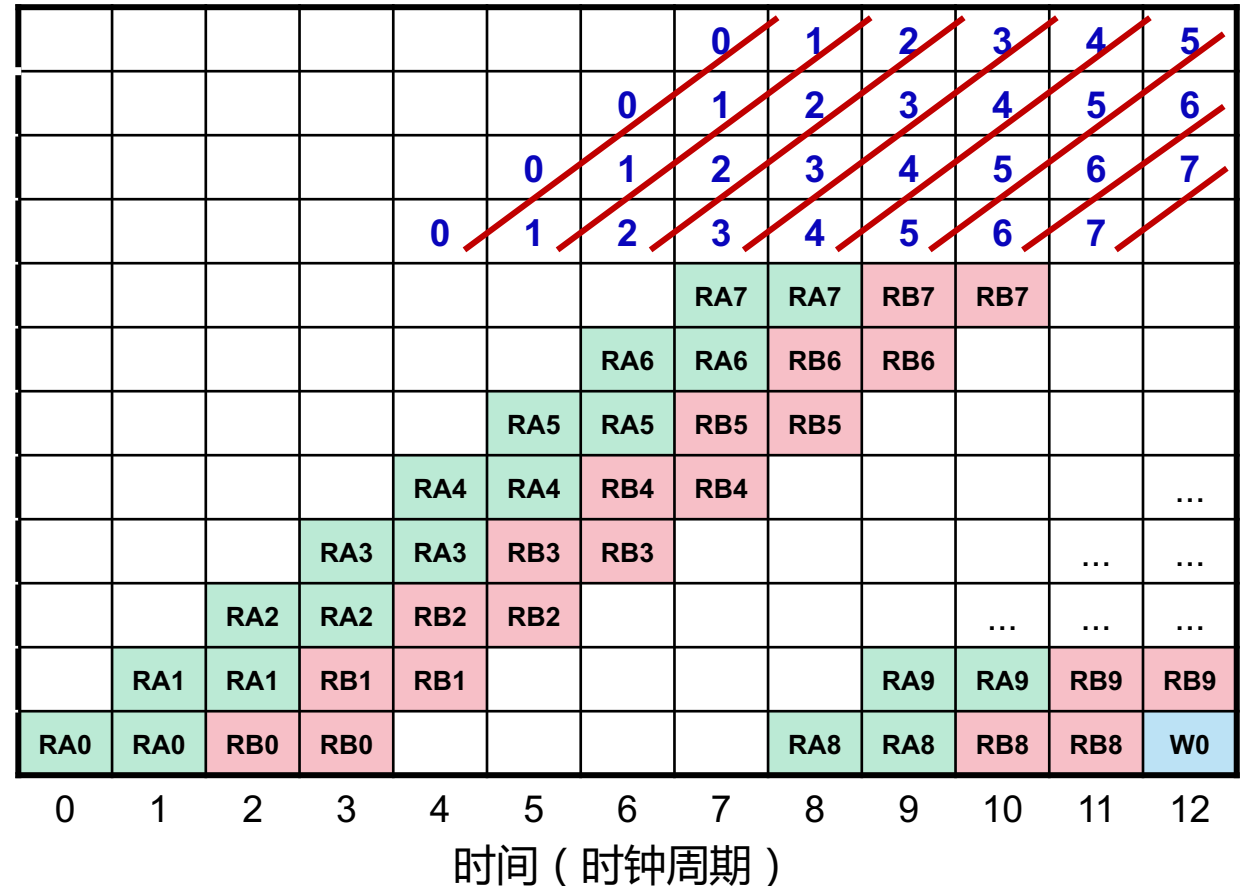
第 54 页

解决访问冲突的方法之二：在运算流水线的输入端和输出端增加操作数**可变缓冲器**和写结果**可变缓冲器**，以便消除征用存储器的现象



方法：向量A的输入缓冲器延迟**2**个时间周期；输出缓冲器延迟**4**个时间周期
结果：向量A和B的对应元素同时到达流水结构加法器；输出缓冲器将每个输出值延迟**4**个时间周期再送入存储器系统，第1个结果在时钟周期**12**时写入主存储器。

流水线3
流水线3
流水线2
流水线1
存储器7
存储器6
存储器5
存储器4
存储器3
存储器2
存储器1
存储器0



“寄存器-寄存器”结构

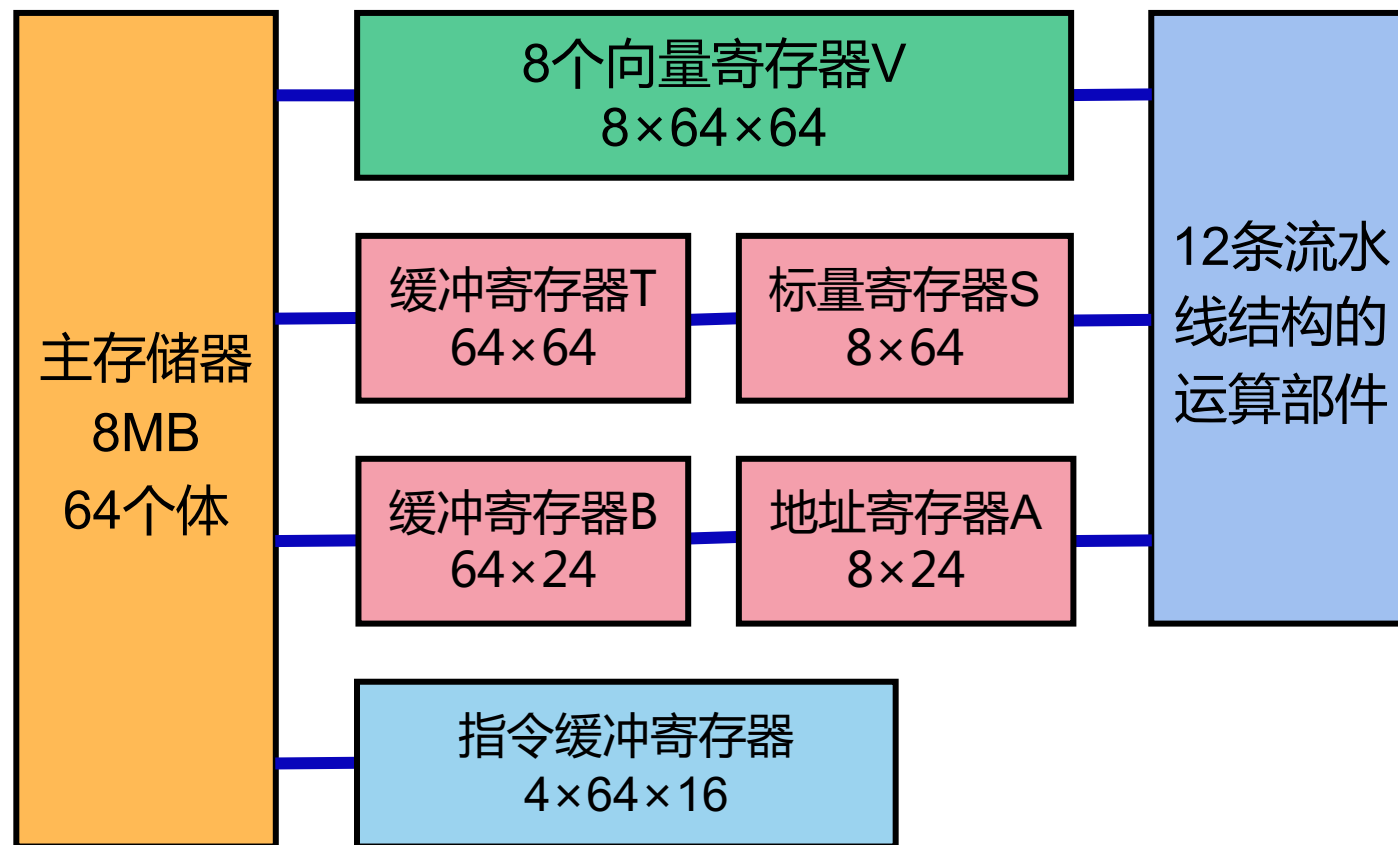
第 55 页

■典型的寄存器 - 寄存器结构的向量处理机：美国的CRAY-1、我国的YH-1巨型机等

■以CRAY-1机为例

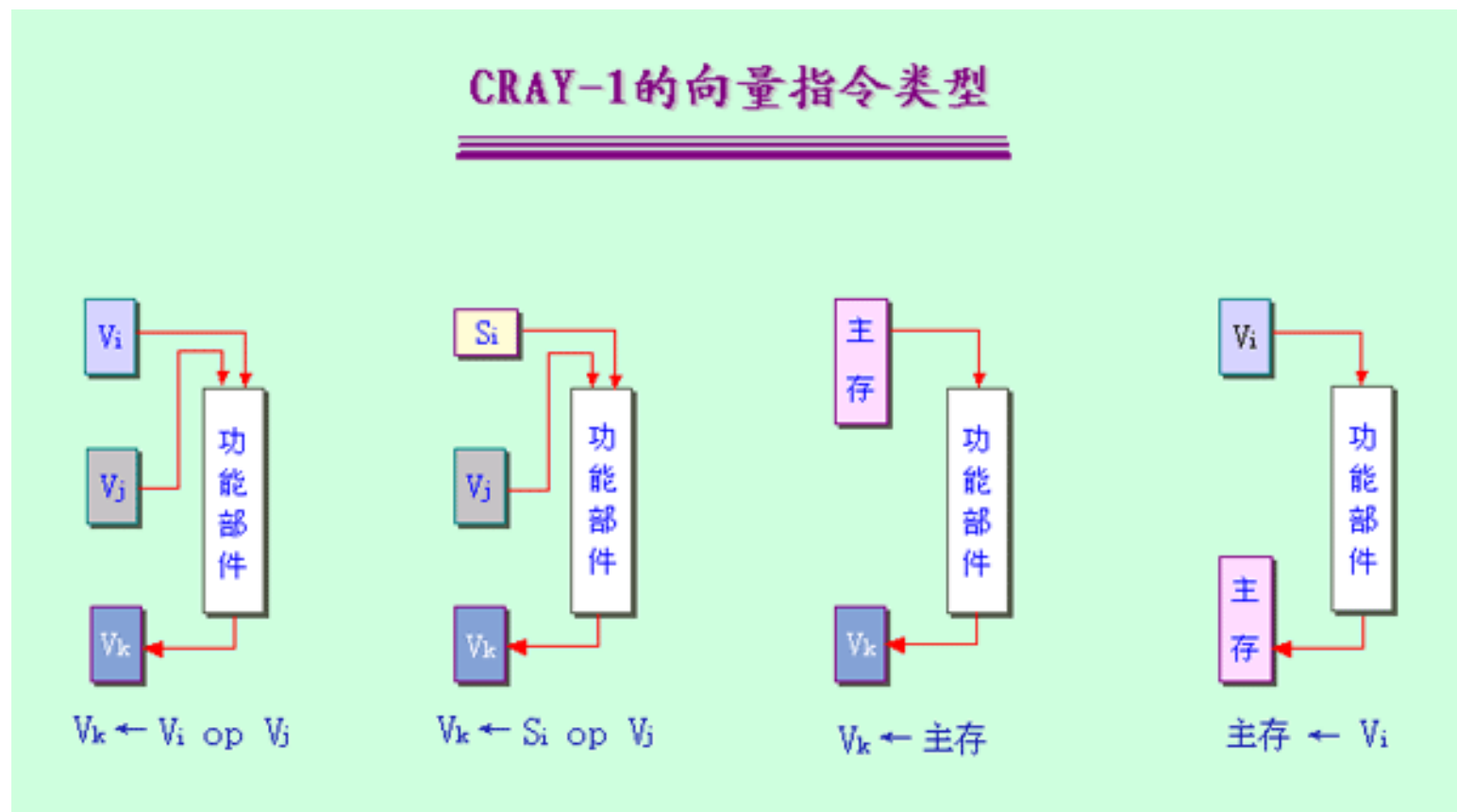
- 美国CRAY公司
- 1976年
- 每秒1亿次浮点运算
- 时钟周期：12.5ns

与存储器-存储器结构的STAR-100比较，
运算速度3倍多（时钟周期为40:12.5），
主存储器流量低2.5倍。



CRAY-1向量指令类型

- ① $V_k \leftarrow V_i \text{ op } V_j$
- ② $V_k \leftarrow S_i \text{ op } V_j$
- ③ $V_k \leftarrow \text{主存}$
- ④ $\text{主存} \leftarrow V_i$



CRAY-1向量处理冲突

■ 主要类型： V_i 冲突和功能部件冲突

- **V_i 冲突**：并行工作的各向量指令的源向量或结果向量使用了相同的 V_i ，如源向量相同

$$V_3 \leftarrow \mathbf{V_1} + V_2$$

$$V_5 \leftarrow V_4 \wedge \mathbf{V1}$$

- **功能部件冲突**：并行工作的各向量指令要使用同一个功能部件，如都需使用乘法功能部件

$$V_3 \leftarrow V_1 \times V_2$$

$$V_5 \leftarrow V_4 \times V_6$$

提高向量处理机性能的方法

- 设置多个功能部件，使它们并行工作；
- 采用链接技术，加快一串向量指令的执行； ←
- 采用分段开采技术，加快循环的处理；
- 采用多处理机系统，进一步提高性能。

2. 向量流水线冲突分析

■设计目标：保证处理器一起处于忙状态，导致无法满负荷工作的原因是向量流水线冲突。

■向量流水线冲突分析：两条向量指令占用功能流水线和向量寄存器的4种情况：

① 指令不相关

例如： $V0 \leftarrow V1 + V2$

$V6 \leftarrow V4 * V5$

这两条指令分别使用各自所需的流水线和向量寄存器，可以并行执行。

② 功能部件冲突

例如： $V3 \leftarrow V1 + V2$

$V6 \leftarrow V4 + V5$

这两条指令都要使用加法流水线，发生了功能部件冲突（但向量寄存器不冲突）。当第一条指令流出时，占用加法流水线。第二条指令要等加法流水线变成空闲后，才能流出。

③ 源寄存器冲突

例如： $V3 \leftarrow V1 + V2$

$V6 \leftarrow V1 * V4$

这两条向量指令的源向量之一都取自V1。由于两者的首元素下标可能不同，向量长度也可能不同，所以难以由V1同时提供两条指令所需要的源向量。这两条向量指令不能同时执行。只有等第一条向量指令执行完、释放V1之后，第二条向量指令才能开始执行。

④ 结果寄存器冲突——两条向量指令使用了相同的结果向量寄存器。

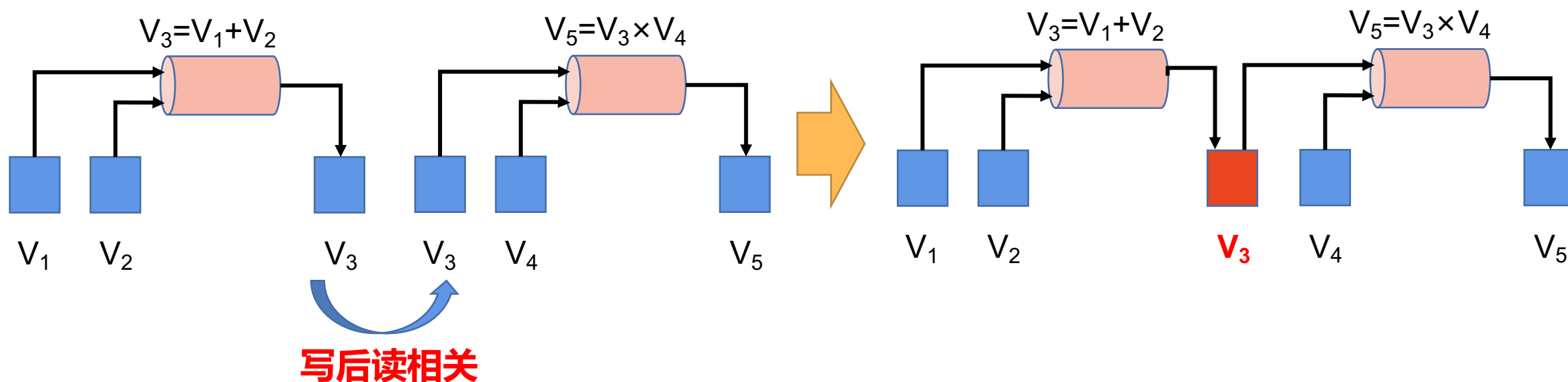
例如： $V4 \leftarrow V1 + V2$

$V4 \leftarrow V3 * V5$

这两条指令都要访问目的寄存器V4。由于第一条指令在先，所以它先占用V4直到运算完成，然后再流出后一条指令。

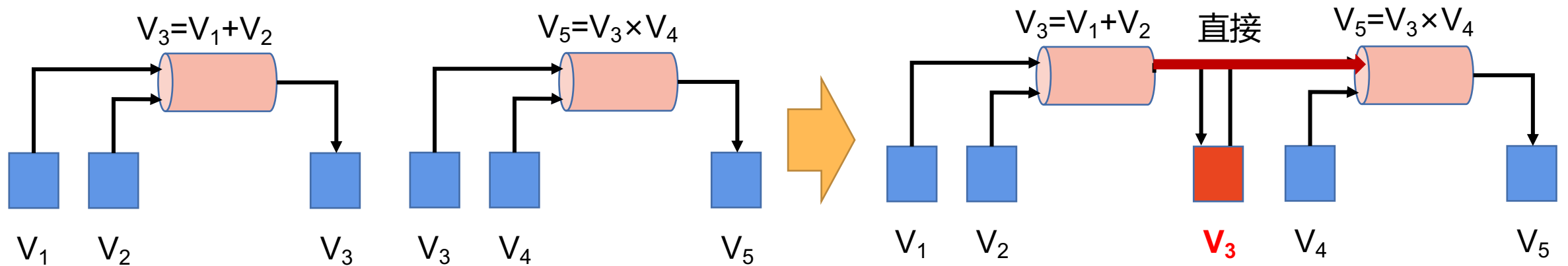
向量流水线链接技术：具有**先写后读**相关的两条指令，在不出现功能部件冲突和源向量冲突的情况下，可以把功能部件链接起来进行流水处理，以达到加快执行的目的。

■前一条指令的结果不必送回存储器，①直接作为后一条指令的操作数



向量流水线链接技术：

- 前一条指令的结果不必送回存储器，②甚至可在前一条指令完成之前就使用其结果。



结果寄存器可能成为后继指令的操作数寄存器，在CRAY-1中这种技术被称为两条流水线的链接（chaining）

衡量向量处理机性能的主要参数：

- 向量指令的处理时间
- 向量长度为无穷大时的向量处理机的最大性能
- 半性能向量长度
- 向量长度临界值

1. Cray-1向量处理的一个显著特点是：只要不出现向量寄存器冲突和功能部件冲突，各V之间和各功能部件之间都能并行工作；
2. 把能在同一个时钟周期内一起开始执行的几条向量指令称为一个编队；
3. 在向量流水处理机上，向量指令序列中的一个编队内的指令可以同时执行，编队执行时间为编队内所有的向量指令执行时间的最大值。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第5章 指令级并行及其开发—硬件方法

(复习)

2023年春季学期

1. 流水线处理机的实际CPI分析

- 理想流水线的CPI加上各类停顿的时钟周期数：

$$CPI_{\text{流水线}} = CPI_{\text{理想}} + \text{停顿}_{\text{结构冲突}} + \text{停顿}_{\text{数据冲突}} + \text{停顿}_{\text{控制冲突}}$$

- 理想CPI是衡量流水线最高性能的一个指标。

设计
目标

$$\left\{ \begin{array}{ll} CPI_{\text{流水线}} \approx CPI_{\text{理想}} & \leftarrow \text{思路：减少结构、数据与控制冲突} \\ \text{进一步减少} CPI_{\text{流水线}} & \leftarrow \text{思路：并行执行两条或两条以上的指令} \end{array} \right\}$$

指令级
并行

- 指令级并行指机器语言一级，如存储器访问指令、整型指令、浮点指令之间的并行性等。
- 指令级并行由处理器硬件和编译程序自动识别和利用，**不需要程序员对顺序程序作任何修改。**

2. 指令级并行设计对象

- **基本程序块**：一串连续的代码（同时执行相邻或相近的多条指令），除了入口和出口以外没有其他的分支指令和转入点
- **分支指令和转入点**：程序平均每4 ~ 7条指令就会有一个分支

3. 循环级并行是指令级并行研究的重点之一

- 开发循环的不同叠代之间存在的并行性
- 循环展开（loop unrolling）技术
- 采用向量指令和向量数据表示

例如，考虑下述语句：

```
for ( i=1 ; i<=500 ; i=i+1 )  
    a[i]=a[i] + s ;
```

1. 指令相关与流水线冲突

一个简单基本程序块

DIV.D	F2, F8, F4
ADD.D	F8, F0, F12
SUB.D	F2, F8, F14

相关是程序固有的一种属性，反映了程序中的指令之间的相互依赖关系

■ **指令相关**有三种类型：

- 数据相关
- 名相关（输出相关、反相关）
- 控制相关

■ 流水线冲突有三种类型：

- 结构冲突
- 数据冲突
- 控制冲突

具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿则是流水线的属性

相关与指令级并行

第 69 页

DIV.D F2, F8, F4
ADD.D F8, F0, F12
SUB.D F2, F8, F14

指令相关
流水线冲突

$$CPI_{\text{流水线}} \geq CPI_{\text{理想}}$$

消除相关：代码变换
避免冲突：指令调度

是否影响
程序正确 ?

代码变换与指令调度必须保持的两个关键属性：

1. 数据流不变

- **定义**：数据值从其产生者指令到其消费者指令的实际流动。
- **原则**：分支指令使得数据流具有动态性，不能影响数据流，**原因**：一条指令有可能数据相关于多条先前的指令，分支指令的执行结果决定了哪条指令真正是所需数据的产生者。

2. 异常行为不变

- **定义**：保持异常行为指原来程序中是怎么发生的，改变执行顺序后还是怎么发生。
- **原则**：无论怎么改变指令的执行顺序都不能改变程序中异常的发生情况，也就是说改变指令执行顺序不能导致程序中发生新的异常属性。

■ 静态调度

- 依靠编译器对代码进行静态调度，以减少相关和冲突。
- 它不是在程序执行的过程中、而是在编译期间进行代码调度和优化。
- 通过把相关的指令拉开距离来减少可能产生的停顿。

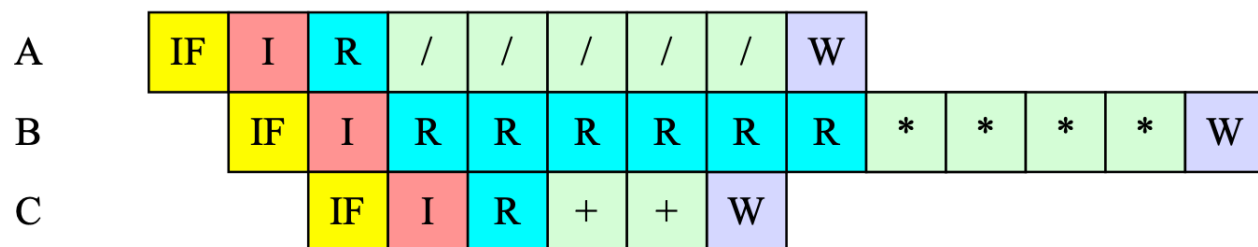
■ 动态调度

- 在程序的执行过程中，依靠专门硬件对代码进行调度，以减少数据相关导致的停顿。
- 优点：
 - ✓ 能够处理一些在编译时情况不明的相关（例如涉及到存储器访问的相关），并简化了编译器；
 - ✓ 能够使本来是面向某一流水线优化编译的代码在其它的流水线（动态调度）上也能高效地执行。
- 缺点：以硬件复杂性的显著增加为代价

动态调度的基本思想

第 71 页

结论：适当且正确地改变程序顺序不会影响程序的正确性。



A: DIV.D F0, F2, F4
B: MUL.D F6, F0, F8
C: ADD.D F10, F12, F14

乱序执行：一种将各条指令不按顺序拆散后执行的运行方式，即指令执行顺序是由输入数据可用性所决定的顺序，而不是由程序的原始数据所决定。

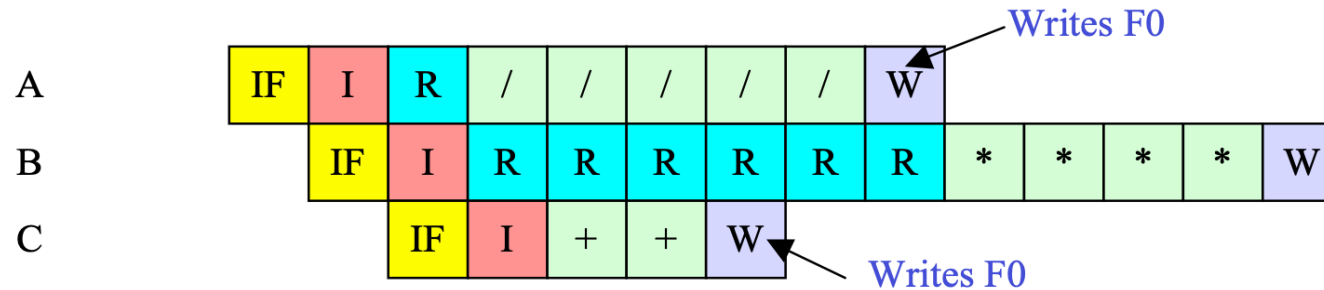
优点：可以避免因为获取下一条程序指令所引起的处理器等待，取而代之的处理下一条可以立即执行的指令。

乱序执行必须不会影响或引入新的WAR冲突和WAW冲突的

动态调度的基本思想

第 72 页

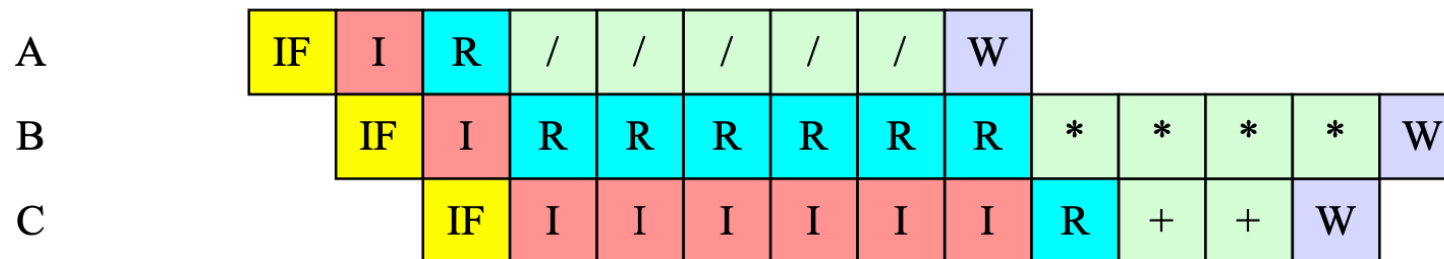
乱序执行流水线：



A: DIV.D F0, F2, F4
B: MUL.D F6, F0, F8
C: ADD.D F0, F10, F12

WAW如何处理？← ROB技术

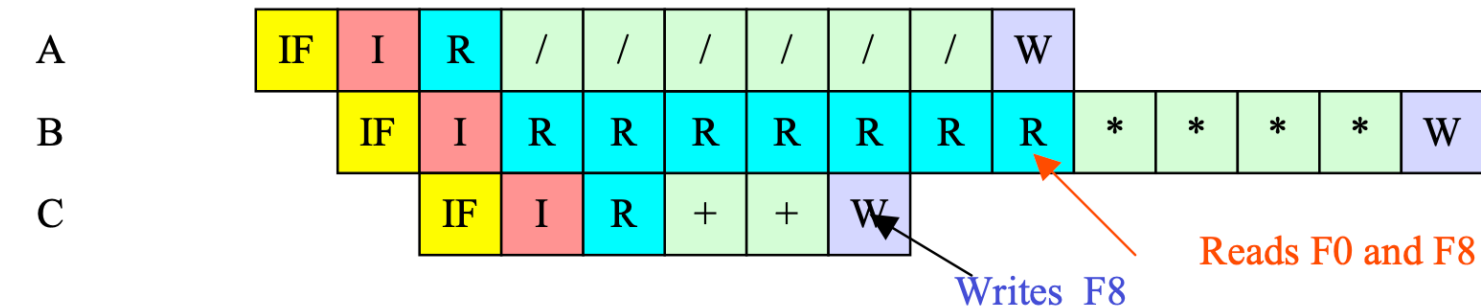
若有WAW，则不发射



动态调度的基本思想

第 73 页

乱序执行流水线：



A: DIV.D
B: MUL.D
C: ADD.D

F0, F2, F4

F6, F0, F8

F8, F10, F12

F9

WAR如何处理？


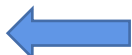

可以通过使用寄存器重命名来消除

精确异常处理与不精确异常处理

- 精确异常：如果发生异常时处理机的现场跟严格按程序顺序执行时指令 i 的现场相同。
- 不精确异常：当执行指令 i 导致发生异常时，处理机的现场（状态）与严格按程序顺序执行时指令 i 的现场不同。
- 发生不精确异常的原因：因为当发生异常（设为指令 i ）时，流水线可能已经执行完按程序顺序是位于指令 i 之后的指令；流水线可能还没完成按程序顺序是指令 i 之前的指令。

记分牌算法和Tomasulo算法是两种比较典型的动态调度算法

基本概念

- CDC 6600 (1963) 计算机最早采用一个称为记分牌 (Scoreboard , 记分板) 的硬件实现了对指令的动态调度。
- **记分牌目标** : 在没有结构冲突时 , 尽可能早地执行没有数据冲突的指令 , 实现每个时钟周期执行一条指令
- **记仇牌实现** : 硬件维护着用于记录指令的三张表
 - ✓ 执行状态表  当前流出的所有指令
 - ✓ 功能部件状态表  当前流出的所有指令是否存在结构冲突
 - ✓ 寄存器状态表  当前流出的所有指令是否存在数据冲突

■ 显式动态寄存器重命名

➤ 硬件维护一个映射表 (Rename Table)

✓ 写寄存器：为该结构寄存器 R_i (Name) 分配一个新的物理寄存器 P_j (Location) 建立映射关系；

✓ 读寄存器：查找映射表，查找最近一次写寄存器 R_w

Initial Mapping

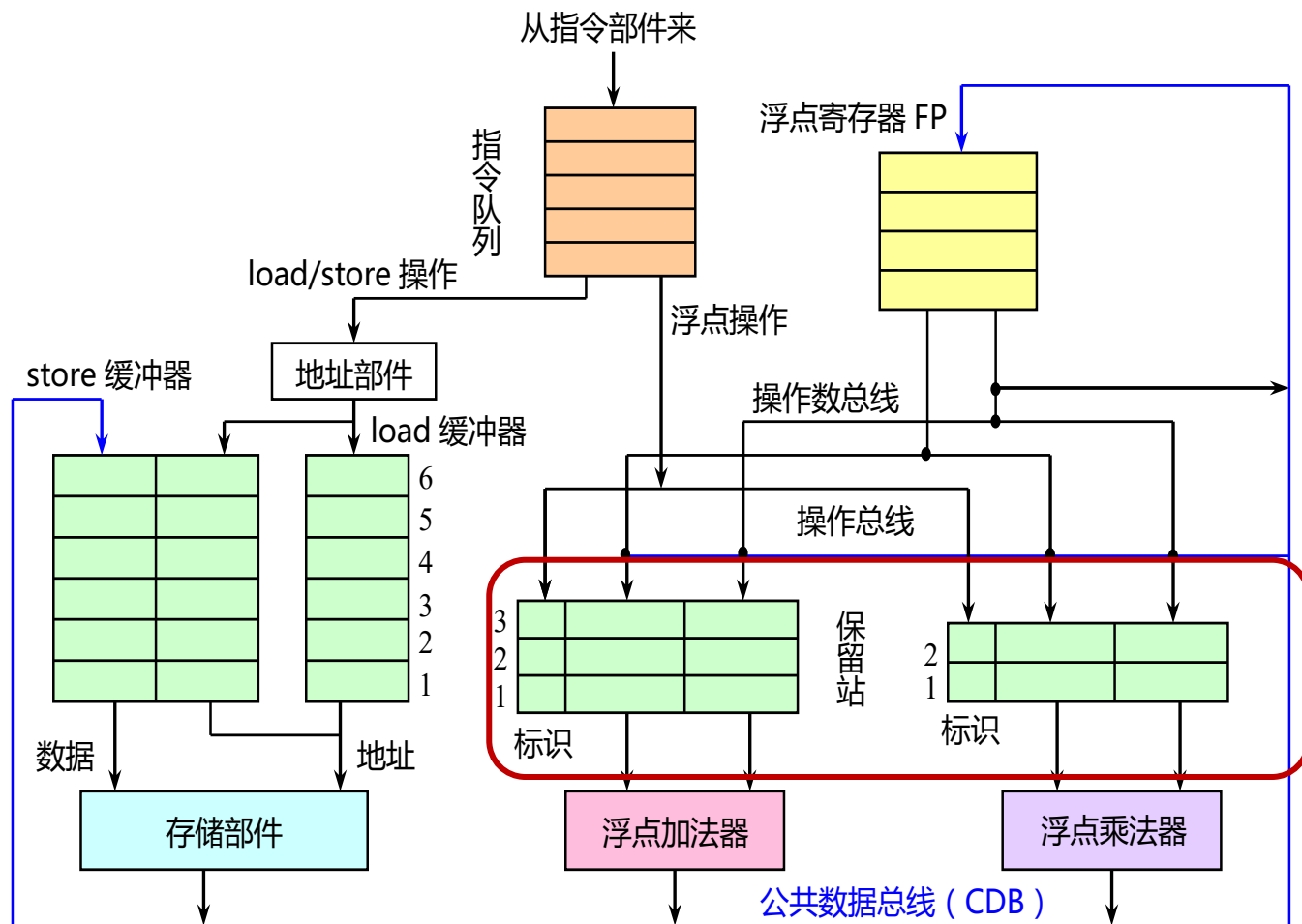
ADD R1, R2, R4
SUB R4, R1, R2
ADD R3, R1, R3
ADD R1, R3, R2

Map Table			
R1	R2	R3	R4
P1	P2	P3	P4
P5	P2	P3	P4
P5	P2	P3	P6
P5	P2	P7	P6
P8	P2	P7	P6

4个 AR
8个 PR

ADD P5, P2, P4
SUB P6, P5, P2
ADD P7, P5, P3
ADD P8, P7, P2

基于Tomasulo算法的MIPS处理器浮点部件的基本结构



保留站 (Reservation Station)

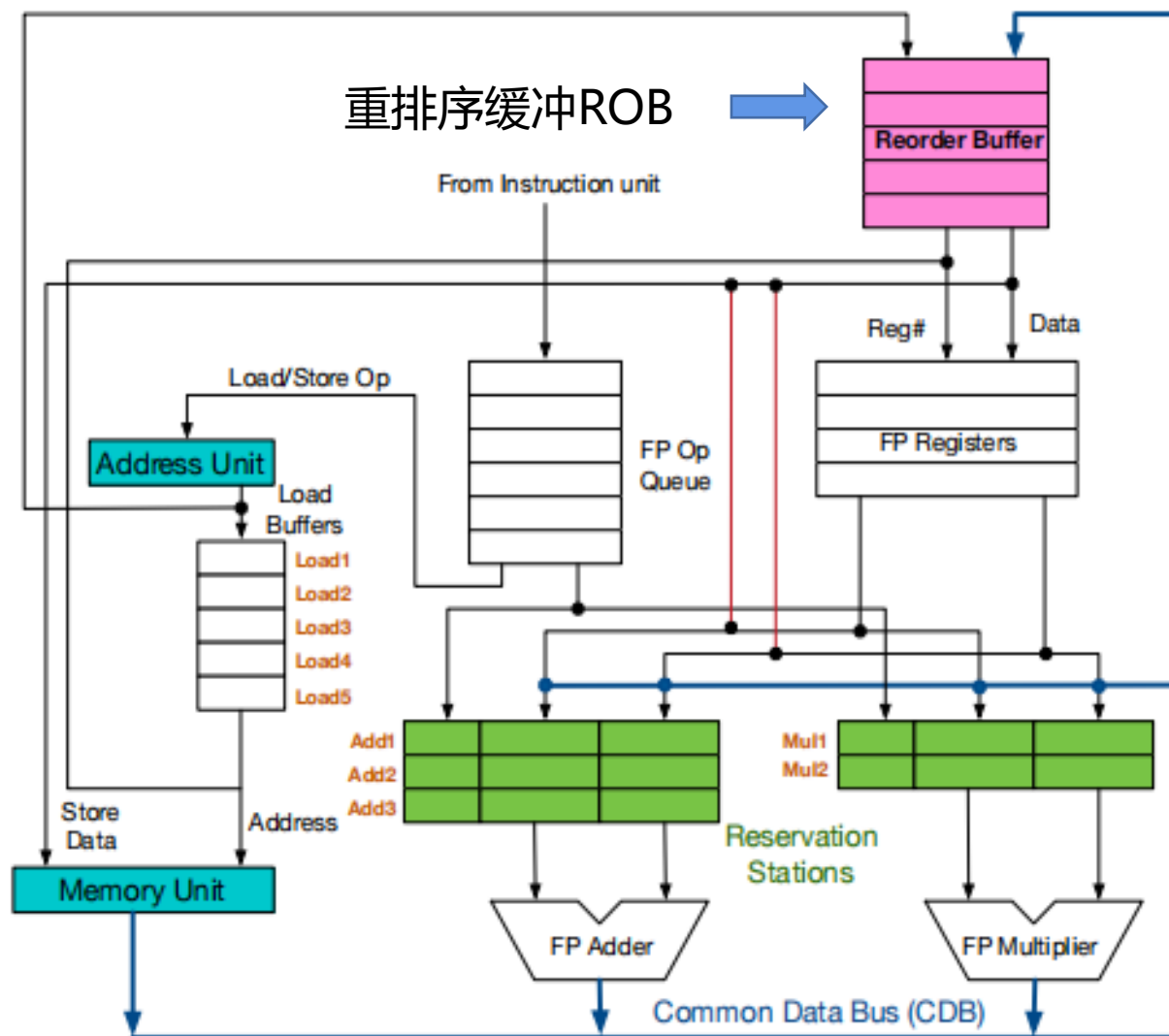
- 浮点加法器有3个保留站：ADD1-3
- 浮点乘法器有两个保留站：MULT1-2
- 每个保留站都有一个标识字段，唯一地标识了该保留站。

公共数据总线CDB

- 所有功能部件的计算结果都是送到CDB上，由它把这些结果直接送到（播送到）各个需要该结果的地方。
- 在具有多个执行部件且采用多流出（即每个时钟周期流出多条指令）的流水线中，需要采用多条CDB。

重排序缓存ROB

第 78 页



改进Tomasulo算法结构图：

1. 增加了Reorder Buffer (即ROB)；
2. CDB总线不再直通逻辑寄存器堆，而是直通ROB；
3. 指令需要从ROB读取数据。ROB在这里就像是一个FIFO队列，指令在发射时入队，在提交时出队

■软件解决方案

- 减少分支：循环展开（Loop Unrolling）
- 提前形成条件码（Reduce resolution time）
- 延迟转移（Delay Slot）——编译技术

■硬件解决方案

- 延迟转移（Delay Slot）——FLUSH技术
- 推测（前瞻）执行或分支预测（Speculation - branch prediction）

■ 动态预测

➤ 分支预测缓冲器



➤ 相关转移预测器

➤ 自适应预测器

➤ 转移目标缓冲器

■ 基本原理

■ 实现

➤ 1-bit转移预测缓冲器

➤ 2-bit转移预测缓冲器

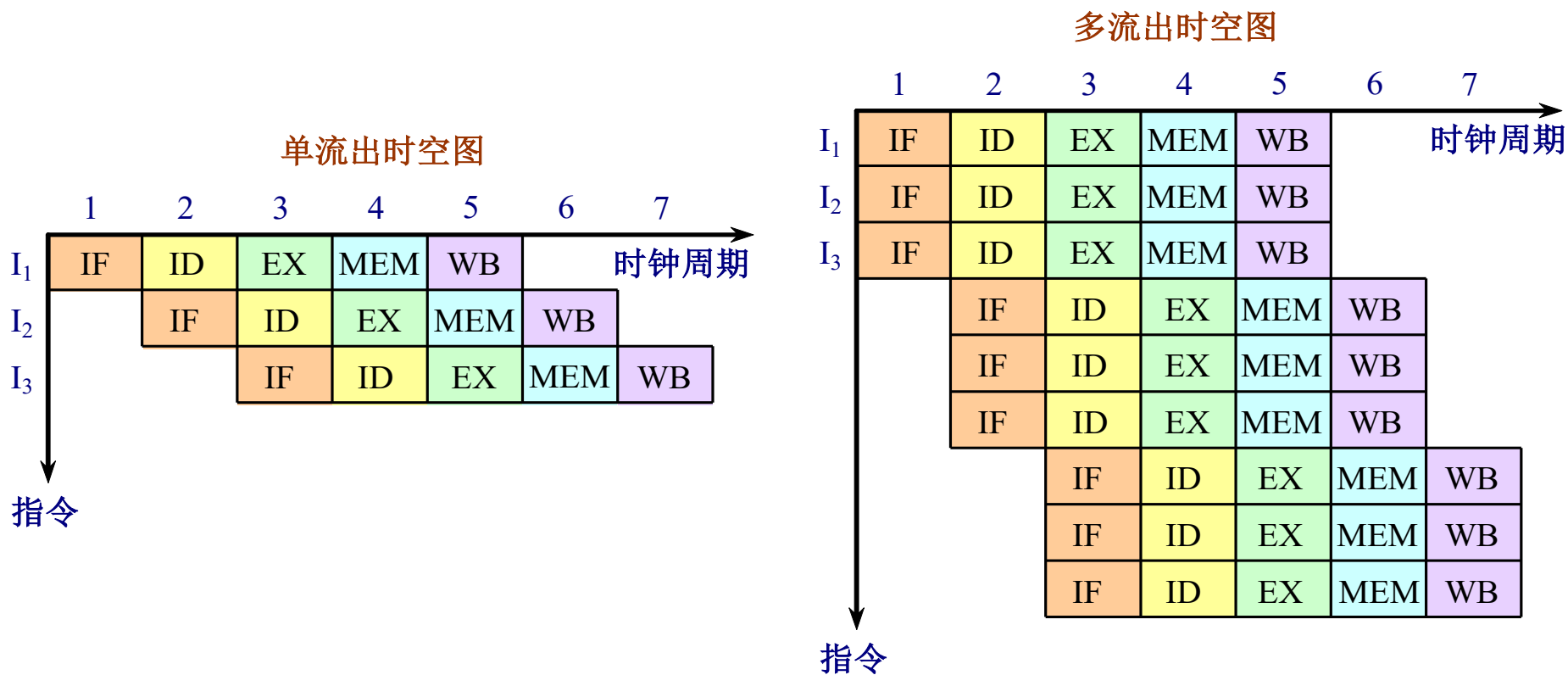
➤ n-bit转移预测缓冲器

■ 性能

多指令发射技术

第 81 页

- 在每个时钟周期内流出多条指令， $CPI < 1$ 。
- 单发射和多发射处理机执行指令的时空图对比



单流出和多流出处理机执行指令的时空图

多发射处理机有两种基本风格：

■ 超标量 (Superscalar)

- 在每个时钟周期流出的指令条数不固定，依代码的具体情况而定。（有个上限）
- 设这个上限为 n ，就称该处理机为 n -流出。
- 可以通过编译器进行静态调度，也可以基于Tomasulo算法进行动态调度。

■ 超长指令字VLIW (Very Long Instruction Word)

- 在每个时钟周期流出的指令条数是固定的，这些指令构成一条长指令或者一个指令包。
- 指令包中，指令之间的并行性是通过指令显式地表示出来的。
- 指令调度是由编译器静态完成的。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第6章 指令级并行及其开发—软件方法

(复习)

2023年春季学期

1. 指令调度：找出不相关的指令序列，让它们在流水线上重叠并行执行。
2. 制约编译器指令调度的因素
 - 程序固有的指令级并行
 - 流水线功能部件的延迟
3. 主要手段：通过循环展开、寄存器重分配和指令调度减少空转
4. 循环展开和指令调度的注意事项
 - 保证正确性；注意有效性；使用不同的寄存器；
 - 删除多余的测试指令和分支指令，并对循环结束代码和新的循环体代码进行相应的修正；注意对存储器数据的相关性分析，例如不同循环体中的存储器地址不同，则可以相互对调；注意新的相关性

■ 循环展开存在问题

```
Loop :   L.D    F0, 0(R1)
         ADD.D  F4, F0, F2
         S.D    F4, 0(R1)
         DADDIU R1, R1, #-8
         BNE   R1, R2, Loop
```

循环体是顺序结构

如果循环体是分支结构，则优化含有分支结构的循环体需要在多个基本块间移动指令，这种调度技术被称为“全局指令调度”，主要包括：

- 踪迹调度

- 超块调度

1. 基本概念：

把同时流出的或者满足特定约束的一组操作打包在一起，得到一条更长的（64位、128位或更长）的指令。



2. 与超标量的对比与分析

- 在动态调度的超标量处理器中，相关检测和指令调度基本都由硬件完成。
- 在静态调度的超标量处理器中，部分相关检测和指令调度工作交由编译器完成。
- 在VLIW处理器中，相关检测和指令调度工作全部由编译器完成，它需要更“智能”的编译器。

■ 什么是EPIC？

- 指令级并行主要由编译器负责开发，处理器为保证代码正确执行提供必要的硬件支持，只有在这些硬件机制的辅助下这些优化技术才能高效完成。
- 系统结构必须提供某种通信机制，使得流水线硬件能够了解编译器“安排”好的指令执行顺序。

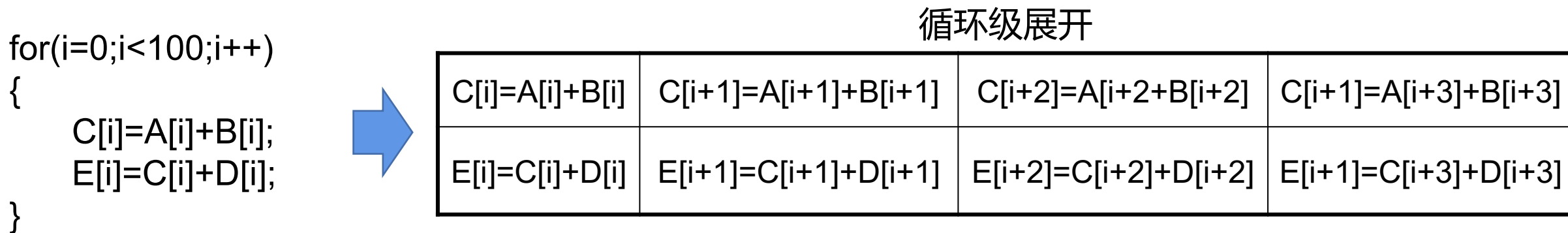
■ EPIC编译器的高级优化技术

- 非绑定分支
 - 谓词执行
 - 前瞻执行
- } 系统结构负责设计流水线硬件执行机制，能够了解编译器“安排”好的指令，动态优化执行顺序

开发更多的指令并行

第 88 页

循环级并行 (Loop-Level Parallelism) : 循环中不同迭代之间的并行。由于每个循环中含有多条指令，所以它的粒度比指令级并行大。



分析：从一个循环中开发出多少并行受到的制约因素

- 迭代内部相关：同一循环迭代内部的指令相关，这属于循环体内部的基本指令调度问题；
- 结论：迭代内各语句间的数据相关对于各次迭代是否能够并行没有影响
- 条件：只要保证同一迭代内存在数据相关的各语句之间的相对顺序不变，多个循环迭代就可以并行执行。

1. 循环携带相关

- 循环携带相关是指一个循环的某个迭代中的指令与其他迭代中的指令之间的数据相关，会大大限制循环展开的效果。

例6.7 在下面的循环中，

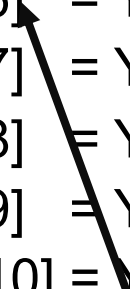
```
for ( i=1 ; i<=100 ; i=i+1 ) {  
    A[i+1] = A[i] + C[i] ;           /* S1 */  
    B[i+1] = B[i] + A[i+1] ;        /* S2 */  
}
```

假设数组A、B和C中所有元素的存储地址都互不相同，请问语句S1与S2之间存在哪些数据相关？

■ 复杂循环携带数据相关的处理

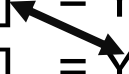
```
for ( i=6 ; i<=100 ; i=i+1 )  
    Y[i] = Y[i-5] + Y[i] ; // 相关距离为5
```

$Y[6] = Y[1] + Y[6]$
 $Y[7] = Y[2] + Y[7]$
 $Y[8] = Y[3] + Y[8]$
 $Y[9] = Y[4] + Y[9]$
 $Y[10] = Y[5] + Y[10]$
 $Y[11] = Y[6] + Y[11]$



```
for ( i=2 ; i<=100 ; i=i+1 )  
    Y[i] = Y[i-1] + Y[i] ; // 相关距离为1
```

$Y[2] = Y[1] + Y[2]$
 $Y[3] = Y[2] + Y[3]$



编译器必须检测出这种递归关系

- (1) 某些系统结构（特别是向量计算机）为递归提供了专门的硬件支持
- (2) 这样的递归结构中通常隐藏着大量的循环级并行

2. 存储别名分析

■ 数组是仿射的

- 如果一个一维数组 $A[m:n]$ 的下标可以被表示为形如 $a \times i + b$ 的形式，那么就称该数组是仿射的（affine）。 i 是循环索引变量，而 m 和 n 分别表示 i 取值的下界和上界。
- 一个多维数组，如果它每一维的下标都是仿射的，那么它就是仿射的。
- 下标为 $x[y[i]]$ 是一个典型的非仿射数组。

问题：判断访问数组 $A[m:n]$ 的两条指令是否相关的问题就可以转换为判断这两条指令所访问的数组元素是否是具有相同的下标

■ GCD测试法

- 假设用下标 $a \times j + b$ 将数组A的一个元素写入存储器，而用下标 $c \times k + d$ 将数组A的一个元素从存储器中读出，这里j和k都是循环A的同一元素， $m \leq j$ ， $k \leq n$ 。显然，当且仅当 $a \times j + b = c \times k + d$ 时，这两条指令将会访问数组A的同一元素。
- 算法描述
 - ✓ 假设a、b、c、d的值均已知
 - ✓ 如果 $\text{GCD}(c, a)$ 可以整除 $(d - b)$ ，那么有可能存在存储别名。
 - ✓ 如果GCD测试的结果为假（不能整除），那么一定没有存储别名存在。

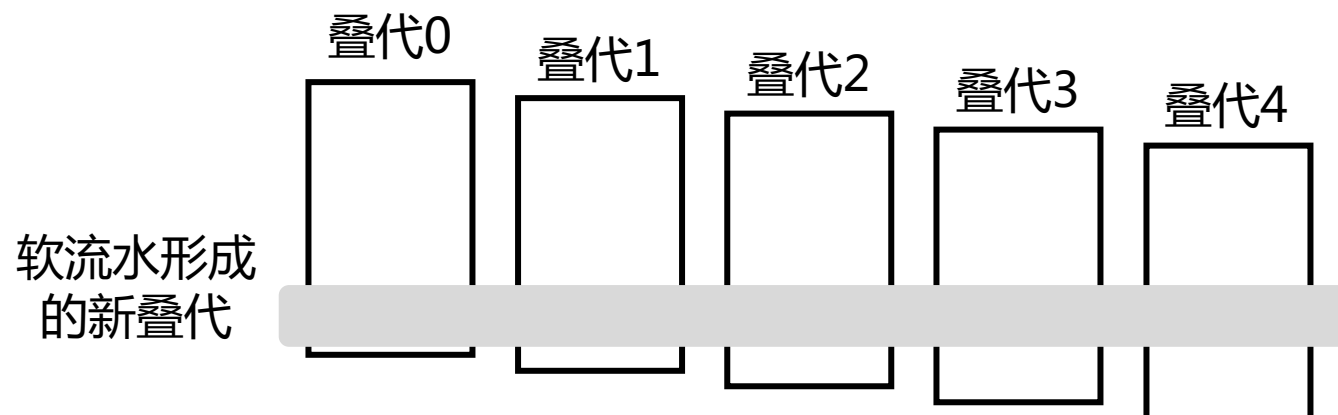
知识点：GCD(a,b)是求a和b的最大公约数，得到的最大公约数

3. 数据相关分析

- 除了检测指令之间是否存在数据相关外，编译器还会将识别出的数据相关进一步细分为**真数据相关、输出相关和反相关**等不同类型，以便利用不同的优化技术消除这些相关。
- 常用的优化有**重命名、值传播、高度消减**等。

■ 简介

- 软流水技术的核心思想是从循环不同的叠代中抽取一部分指令（循环控制指令除外）拼成一个新的循环叠代。
- 目的
 - ✓ 将同一叠代中的相关指令分布到不同的叠代中
 - ✓ 将不同叠代中的相关指令封装到同一叠代中



软流水=虚拟的硬件流水线

注意：与硬件流水线不同的是，充满或排空的指令无法被封装到任何一个新的迭代中，只能放在新循环之前或之后。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第7章 存储系统

(复习)

2023年春季学期

■尽可能快的存取速度

- 应能基本满足CPU对数据的访问要求

■尽可能大的存储空间

- 可以满足程序对存储空间的要求

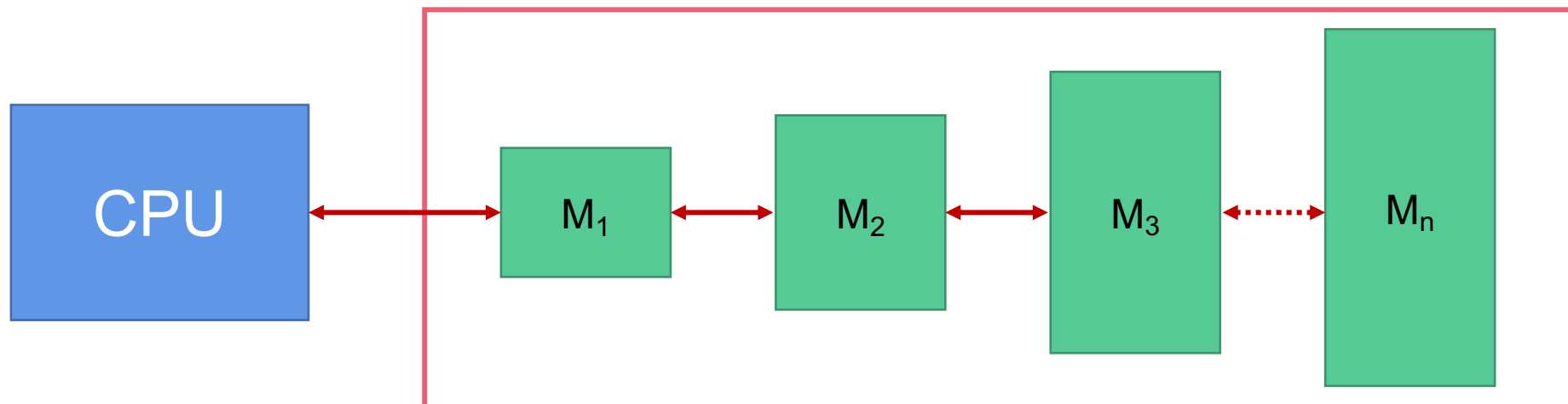
■尽可能低的单位成本（价格/位）

- 用户能够承受的范围內

存储系统的层次结构

第 97 页

解决方法：采用多种存储器技术，构成多级存储层次结构。



- **设计原则**：离CPU越近，存储器容量越小，但访问速度越快；离CPU越远，存储器容量越大，但访问速度越慢。
- **访问原则**：CPU只访问最近的存储器。
- **存在问题**：如果CPU在M₁中没有访问到所需的数据时如何操作？

程序访问的局部性

- 程序访问的局部性原理：对于绝大多数程序来说，程序所访问的指令和数据在地址上不是均匀分布的，而是相对簇聚的。

局部性原理：CPU访问存储器时，无论是存取指令还是存取数据，所访问的存储单元都趋于聚集在一个较小的连续区域中。

- **时间局部性 (Temporal Locality)**：一个数据或指令被访问，近期内它被再次访问的概率很大，例如程序循环等。
- **顺序局部性 (Order Locality)**：在典型程序中，除转移类指令外，大部分指令是顺序进行的。
- **空间局部性 (Spatial Locality)**：一个数据被访问，近期内它附近的数据被再次访问的概率很大，例如数组的连续存放。

3. 命中率H

- 命中率：CPU访问存储系统时，在M₁中找到所需信息的概率。

$$H = \frac{N_1}{N_1 + N_2}$$

➤ N₁ —— 访问M₁的次数

➤ N₂ —— 访问M₂的次数

- 不命中率（缺失率）：F = 1 - H

■ 存储系统的访问时间T

$$T = H \cdot T_1 + (1 - H) \cdot T_2 \quad \text{当 } H \rightarrow 1 \text{ 时, } T \rightarrow T_1$$

假设： M_1 是 M_2 的真子集；所有的数据都能在 M_2 中找到。

访问方式： 先访问 M_1 ；如果在 M_1 中没有找到，则访问 M_2 ，再把数据提供给处理器

■ 存储系统的访问效率

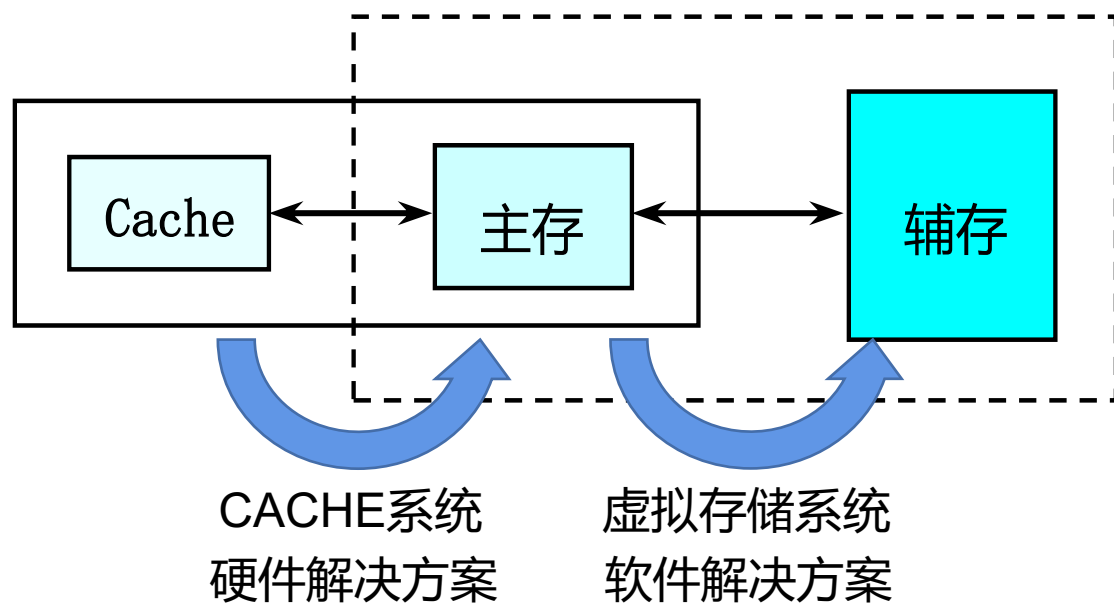
$$e = \frac{T_1}{T} = \frac{T_1}{H \cdot T_1 + (1 - H) \cdot T_2}$$

典型的三级存储系统

第 101 页

由“Cache—主存”层次和“主存—辅存”层次构成的系统

- Cache是一种小容量高速缓冲存储器，由快速SRAM组成，直接集成在处理器芯片内，速度快，与处理器处于同一个级，通常包括指令Cache和数据Cache
- 主存储器由DRAM组成，例如DDR内存
- 磁盘存储器（辅存），例如机械硬盘、光盘和SSD硬盘等。



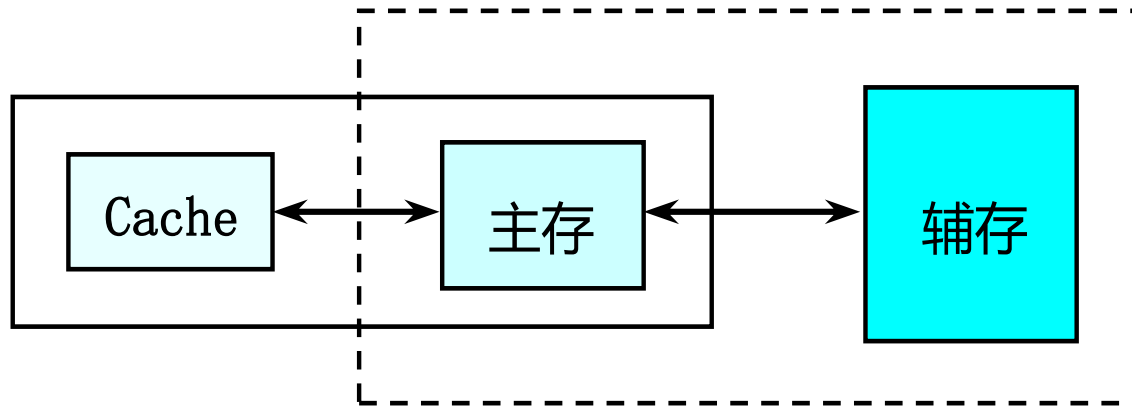
在硬件与软件开销容许的前提下：

$$e = f(H, \frac{T_2}{T_1})$$

如何提升H？

Cache基本知识

第 102 页



数据大小为32位=4B (字节)

数据Cache：一个存储体大小为4KB

地址为：

11 : 3	1 : 0
--------	-------

主存DRAM：存储容量大小为4MB

地址为：

21 : 3	1 : 0
--------	-------

映象规则问题：当把一个块调入低一层(靠近CPU)存储器时，可以放在哪些位置上？

查找算法问题：当CPU访问Cache时，如何确定Cache中是否有所要访问的块？

替换算法问题：当发生不命中时，应替换哪一块？

写策略问题：当进行写访问时，是否写回主存，执行哪些操作？

1. 平均访存时间 = 命中时间 + 缺失率 × 不命中开销

2. 可以从三个方面改进Cache的性能：

- 降低缺失率
- 减少不命中开销
- 减少Cache命中时间

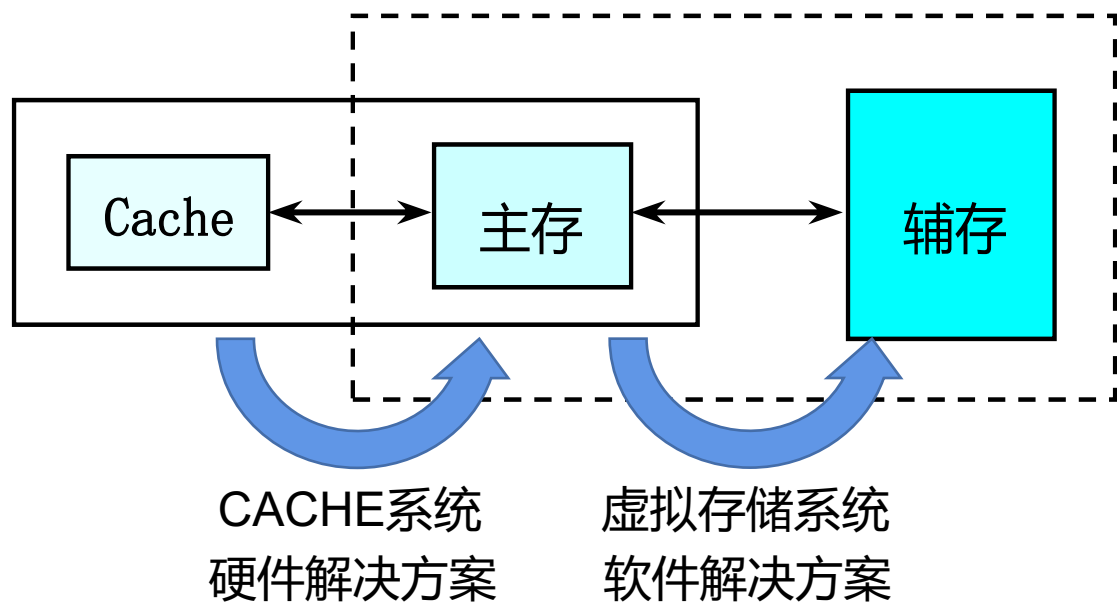
3. Cache优化技术

- 8种用于降低不命中率
- 5种用于减少不命中开销
- 4种用于减少命中时间

1. 多体交叉存储器：由多个单字存储体构成，每个体都有自己的地址寄存器以及地址译码和读/写驱动等电路。
2. 问题：对多体存储器如何进行编址？
 - 存储器是按顺序线性编址的。如何在二维矩阵和线性地址之间建立对应关系？
 - 两种编址方法
 - 高位交叉编址
 - 低位交叉编址（有效地解决访问冲突问题）

虚拟存储器的基本原理

第 105 页



虚拟存储器实现了内存扩充功能。但该功能并非是从物理上实际地扩大内存的容量，而是从逻辑上实现对内存容量的扩充。

■ 存在现象：

- 有的作业很大，其所要求的内存空间超过了内存总容量；
- 有大量作业要求运行，但由于内存容量不足以容纳所有这些作业。

■ 原因：由于内存容量不够大导致的

■ 解决方法：从逻辑上扩充内存容量



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第9章 互连网络

(复习)

2023年春季学期

- 变量 x ：输入（设 $x=0, 1, \dots, N-1$ ） \rightarrow 函数 $f(x)$ ：输出
- 互连函数通过数学表达式建立输入端号与输出端号的连接关系，即在互连函数 f 的作用下，输入端 x 连接到输出端 $f(x)$ 。
 - 互连函数反映了网络输入数组和输出数组之间对应的置换关系或排列关系，有时也称为置换函数或排列函数
 - 互连函数 $f(x)$ 有时可以采用循环表示，即： $(x_0 \ x_1 \ x_2 \ \dots \ x_{j-1})$ ， j 称为该循环的长度，则表示： $f(x_0)=x_1, f(x_1)=x_2, \dots, f(x_{j-2})=x_{j-1}, f(x_{j-1})=x_0$
 - 设 $n=\log_2 N$ ，则可以用 n 位二进制来表示 N 个输入端和输出端的二进制地址，互连函数表示为

$$f(x_{n-1}x_{n-2}\dots x_1x_0)$$

掌握几种常用的基本互连函数及其主要特征。

1. 恒等函数
2. 交换函数
3. 均匀洗牌函数 (子函数、超函数、逆均匀洗牌函数) <混洗函数、置换函数>
4. 碟式函数 (子函数、超函数)
5. 反位序函数 (子函数、超函数)
6. 移数函数
能力：现有16个处理器，编号分别为0，1，...，15，分给与第13号处理器连接的处理器号
7. PM2I函数
(1) $Cube_3$ (2) $PM2_{+3}$ (3) $PM2_{-0}$ (4) σ (5) $\sigma(\sigma)$

1. 网络通常是用有向边或无向边连接有限个结点的图来表示。
2. 互连网络的主要特性参数有：
 - **网络规模 N** ：网络中结点的个数，表示该网络所能连接的部件的数量。
 - **结点度 d** ：与结点相连接的边数（通道数），包括入度和出度。
 - ✓ 进入结点的边数叫入度。
 - ✓ 从结点出来的边数叫出度。
 - **结点距离**：对于网络中的任意两个结点，从一个结点出发到另一个结点终止所需要跨越的边数的最小值。
 - **网络直径 D** ：网络中任意两个结点之间距离的最大值，网络直径应当尽可能地小。

- 等分宽度 b ：把由 N 个结点构成的网络切成结点数相同（ $N/2$ ）的两半，在各种切法中，沿切口边数的最小值。

线等分宽度： $B = b \times w$

其中： w 为通道宽度（用位表示），该参数主要反映了网络最大流量。

- 对称性：从任何结点看到的拓扑结构都是相同的网络称为对称网络，对称网络比较容易实现，编程也比较容易。

评估互连网络性能的两个基本指标：时延和带宽

1. 通信时延

指从源结点到目的结点传送一条消息所需的总时间，它由以下4部分构成：

- 软件开销：在源结点和目的结点用于收发消息的软件所需的执行时间。
 - ✓ 主要取决于两端端结点处理消息的软件内核。
- 通道时延：通过通道传送消息所花的时间。
 - ✓ 通路时延 = 消息长度/通道带宽
 - ✓ 通常由瓶颈链路的通道带宽决定。
- 选路时延：消息在传送路径上所需的一系列选路决策所需的时间开销
(与传送路径上的结点数成正比)
- 竞争时延：多个消息同时在网络中传送时，会发生争用网络资源的冲突。为避免或解决争用冲突所需的时间就是竞争时延
(很难预测，它取决于网络的传输状态)

2. 网络时延

指通道时延与选路时延的和，它是由网络硬件特征决定的，与程序行为和网络传输状态无关。

3. 端口带宽

- 对于互连网络中的任意一个端口来说，其端口带宽是指单位时间内从该端口传送到其他端口的最大信息量。
 - 在对称网络中，端口带宽与端口位置无关。网络的端口带宽与各端口的端口带宽相同。
 - 非对称网络的端口带宽则是指所有端口带宽的最小值。

4. 聚集带宽

- 网络从一半结点到另一半结点，单位时间内能够传送的最大信息量。

例如，HPS是一种对称网络

网络规模N的上限：512

端口带宽：40MB/s

HPS的聚集带宽： $(40\text{MB/s} \times 512) / 2 = 10.24\text{GB/s}$

5. 等分带宽

与等分宽度对应的切平面中，所有边合起来单位时间所能传送的最大信息量

互连网络通常可以分为两大类：

- **静态互连网络**：各结点之间有固定的连接通路、且在运行中不能改变的网络。
- **动态互连网络**：由交换开关构成、可按运行程序的要求动态地改变连接状态的网络。

了解几种静态互连网络（其中： N 表示结点的个数）

1. 线性阵列
2. 环和带弦环
3. 全连接网络
4. 循环移数网络
5. 胖树形
6. 网格形和环网形
7. 超立方体
8. 带环立方体
9. k 元 n -立方体网络等

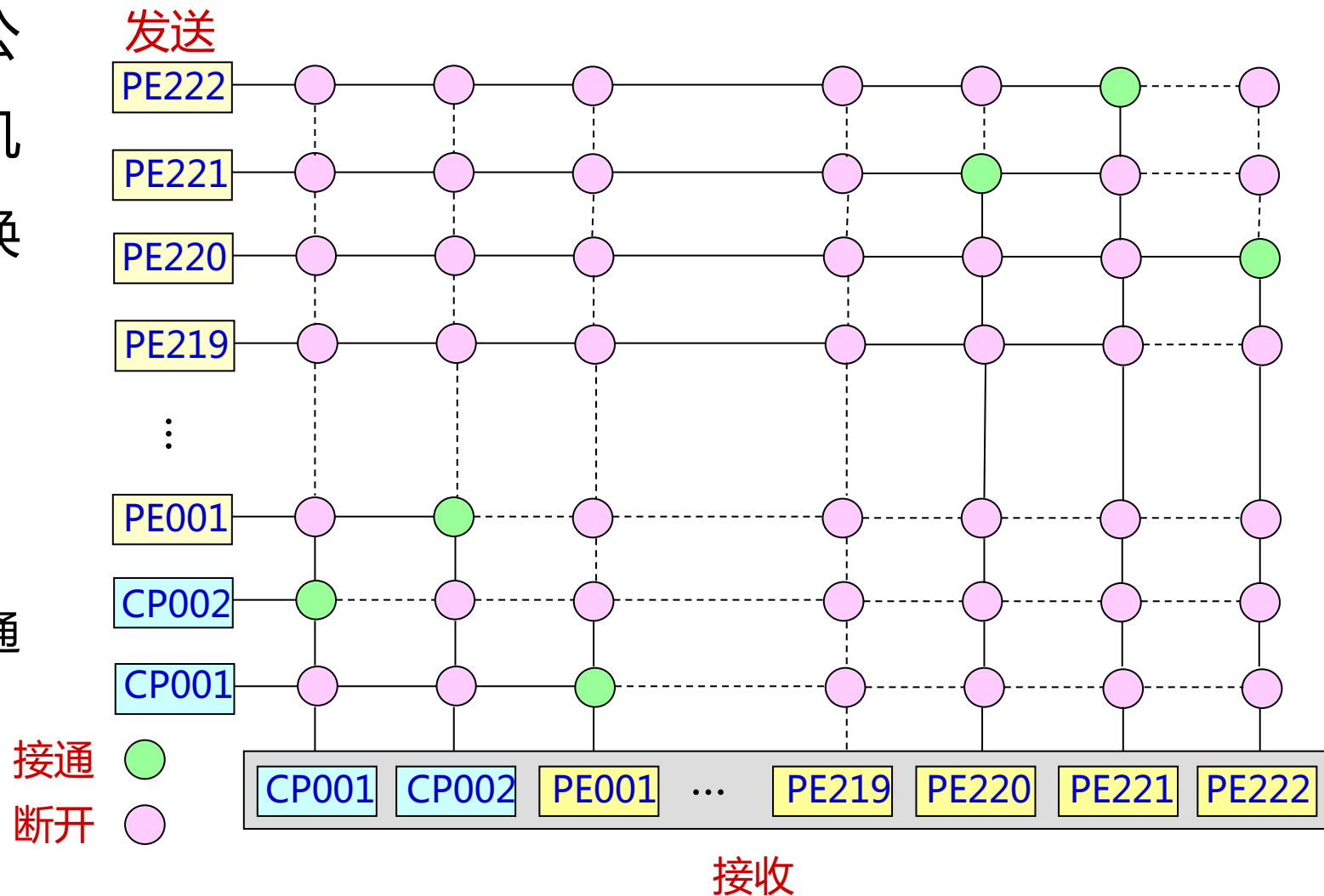
了解结点度、网络直径、链路数、等分宽度、对称性等特征

交叉开关网络

第 115 页

1. 单级开关网络：Fujitsu公司制造的向量并行处理机VPP500所采用的大型交换开关网络（ 224×224 ）

- PE：带存储器的处理机
- CP：控制处理机
- 每一行和每一列只能接通一个交叉点开关



1. 多级互连网络的构成

- MIMD和SIMD计算机都采用多级互连网络MIN (Multistage Interconnection Network)

- 一种通用的多级互连网络

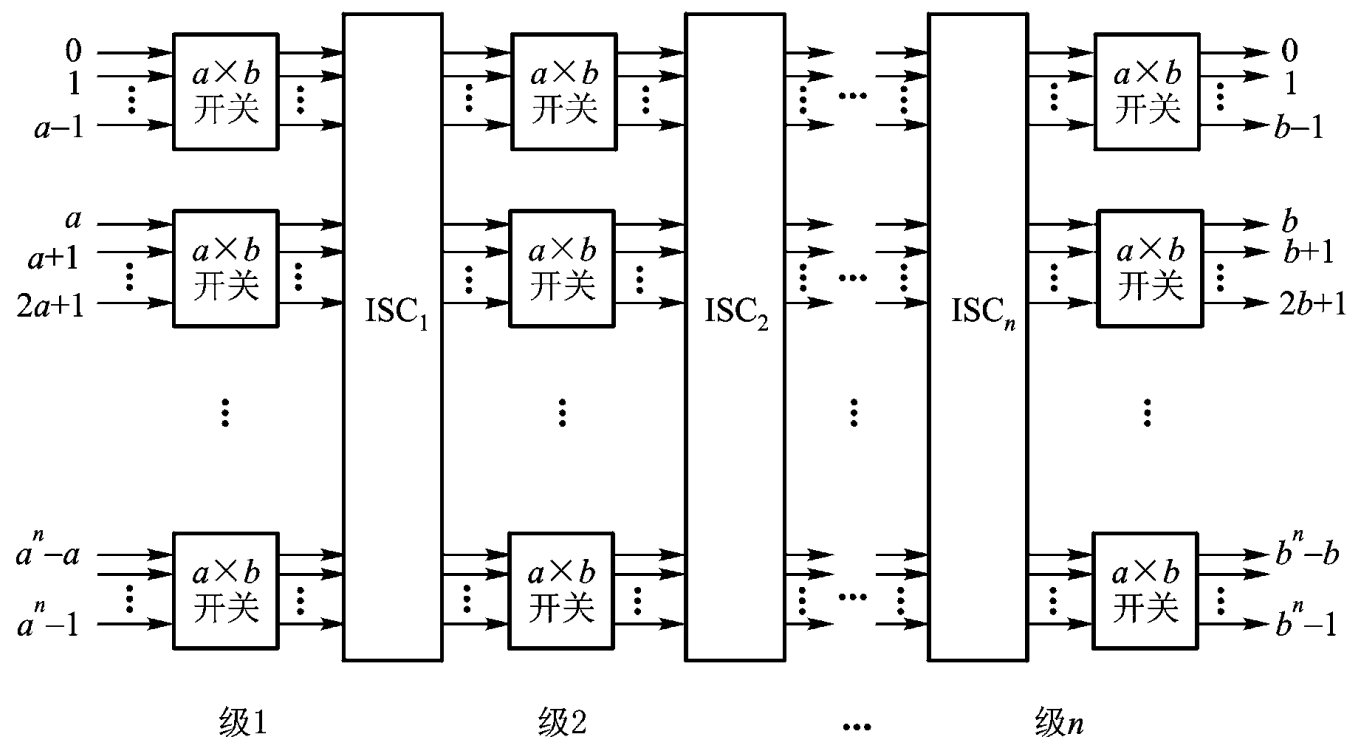
➤由 $a \times b$ 开关模块和级间连接构成的通用多级互连网络结构

➤每一级都用了多个 $a \times b$ 开关

✓ a 个输入和 b 个输出

✓在理论上， a 和 b 不一定相等，然而实际上 a 和 b 经常选为2的整数幂，即 $a = b = 2^k, k \geq 1$ 。

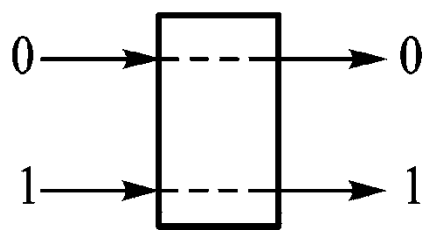
➤相邻各级开关之间都有固定的级间连接



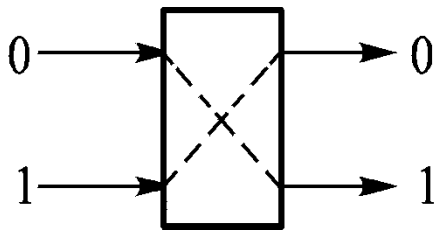
■ 几种常用的开关模块

模块大小	合法状态	置换连接
2×2	4	2
4×4	256	24
8×8	16 777 216	40 320
$n \times n$	n^n	$n!$

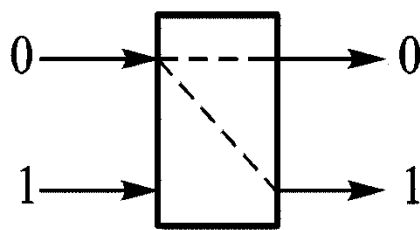
■ 最简单的开关模块：2×2开关



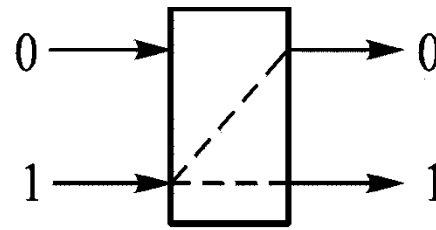
(a) 直送



(b) 交叉



(c) 上播



(d) 下播

- 各种多级互连网络的区别在于所用开关模块、控制方式和级间互连模式的不同。

- 控制方式：对各个开关模块进行控制的方式。

- ✓ 级控制：每一级的所有开关只用一个控制信号控制，只能同时处于同一种状态。

- ✓ 单元控制：每一个开关都有一个独立的控制信号，可各自处于不同的状态。

- ✓ 部分级控制：第 i 级的所有开关分别用 $i + 1$ 个信号控制， $0 \leq i \leq n - 1$ ， n 为级数。

- 常用的级间互连模式：

- 均匀洗牌、蝶式、多路洗牌、纵横交叉、立方体连接等

■ 确定性寻径和自适应寻径

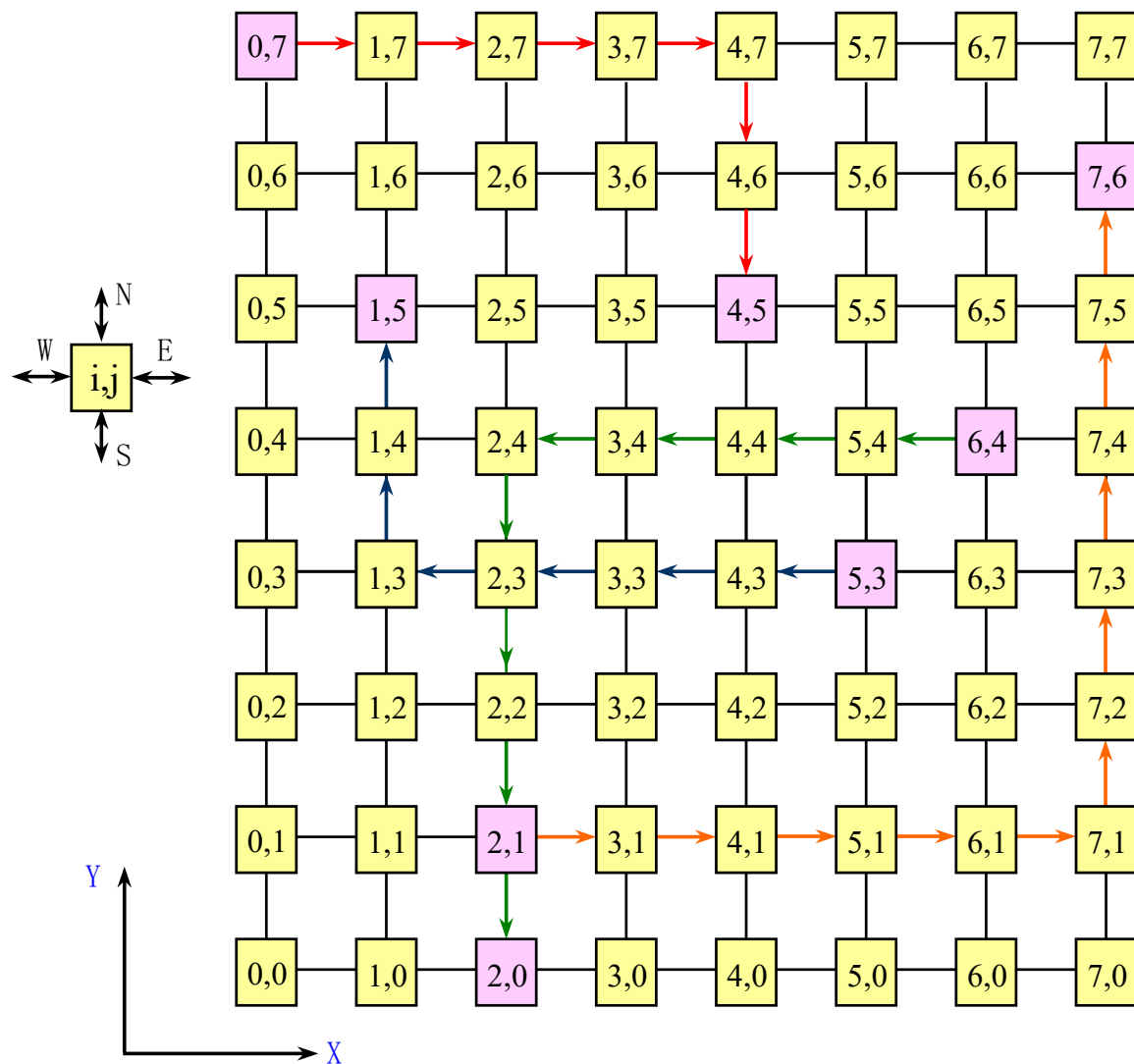
- 确定性寻径：通信路径完全由源结点地址和目的地址来决定，也就是说，寻径路径是预先唯一地确定好了的，而与网络的状况无关。
- 自适应寻径：通信的通路每一次都要根据资源或者网络的情况来选择。
 - 可以避开拥挤的或者有故障的结点，使网络的利用率得到改进。
- 两种确定性寻径算法
 - 都是建立在维序概念之上的
 - 对于一个多维网来说，维序寻径要求对后继通道的选择是按照各维的顺序来进行的。
 - 对于二维的网格网络来说，这种寻径方法被称为X-Y寻径。
 - ✓ 先沿X维方向进行寻径，然后再沿Y维方向寻找路径。
 - 对于超立方体来说，这种寻径方法被称为E-cube寻径。

■ 二维网格网络的X-Y寻径

- 任意一个源结点： $s = (x_1, y_1)$
- 任意一个目的结点： $d = (x_2, y_2)$
- 从 s 出发，先沿X轴方向前进，直到找到 d 所在的列 x_2 ；
- 然后再沿Y轴方向前进，直到找到目标结点 (x_2, y_2) 。

例 对于图所示的二维网格，确定以下4组“源结点-目的结点”所需要的路径。

- (2, 1) 到 (7, 6) (一条东-北路径)
- (0, 7) 到 (4, 5) (一条东-南路径)
- (6, 4) 到 (2, 0) (一条西-南路径)
- (5, 3) 到 (1, 5) (一条西-北路径)



考虑一个由 $N = 2^n$ 个结点构成的 n 方体，每个结点的编号是形为 $b = b_{n-1}b_{n-2}\dots b_1b_0$ 的二进制编码。设：源结点 $s = s_{n-1}s_{n-2}\dots s_1s_0$ ，目的结点 $d = d_{n-1}d_{n-2}\dots d_1d_0$ ，现在要确定一条从 s 到 d 的步数最少的路径，将这个 n 方体的各维表示成 $i = 1, 2, \dots, n$ ，其中第 i 维对应于结点地址中的第 $i - 1$ 位。

设 $v = v_{n-1}v_{n-2}\dots v_1v_0$ 是路径中的任一结点，路径可以根据以下算法唯一地确定：

① 计算方向位 $r_i = s_{i-1} \oplus d_{i-1}$ ，其中 $i = 1, 2, \dots, n$ 。

令 $v = s$ ， $i = 1$ ，反复执行以下步骤：

② 如果 $r_i = 1$ ，则从当前结点 v 寻径到下一结点；否则，就跳过这一步。

③ $i \leftarrow i + 1$ 。如果 $i \leq n$ ，则转第②步，否则退出。

■多计算机网络中会出现以下4种通信模式：

- **单播**：对应于一对一的通信情况，即一个源结点发送消息到一个目的结点。
- **选播**：对应于一到多的通信情况，即一个源结点发送同一消息到多个目的结点。
- **广播**：对应于一到全体的通信情况，即一个源结点发送同一消息到全部结点。
- **会议**：对应于多到多的通信情况。

■ 通道流量和通信时延是常用的两个参数。

- **通道流量**可用传输有关消息所使用的通道数来表示。
- **通信时延**则用包的最大传输时间来表示。

■ 优化的寻径网络应能以最小流量和最小时延实现相关的通信模式。



清华大学计算机科学与技术系本科专业基础课

计算机系统结构

第10章 多处理机

(复习)

2023年春季学期

■根据存储器的组织结构，把现有的MIMD机器分为两类：

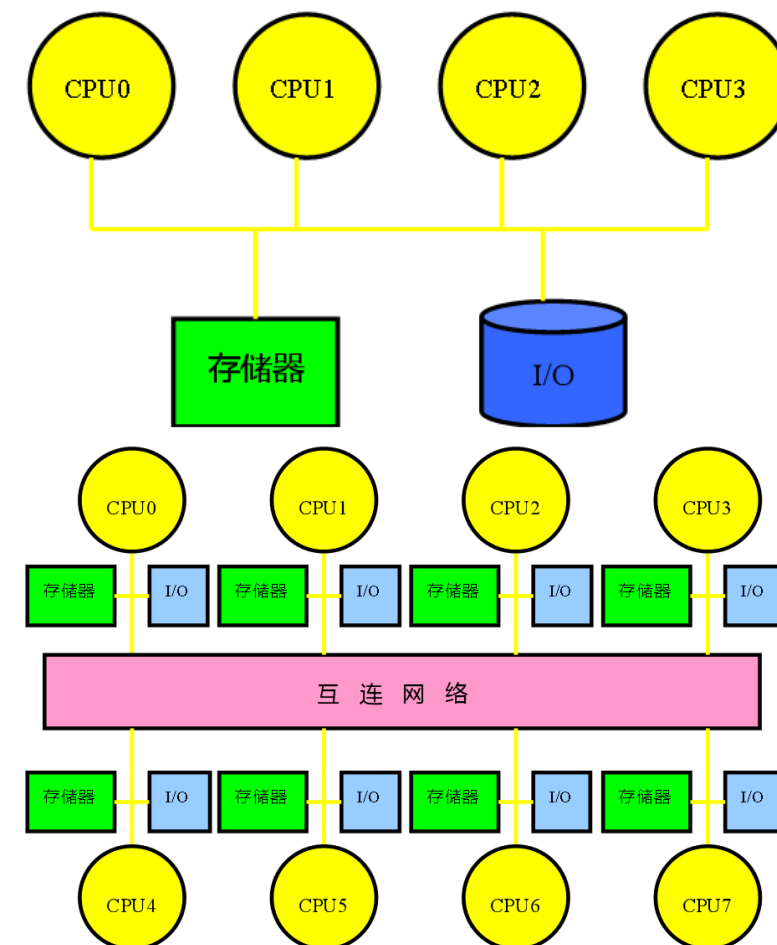
- 集中式共享存储器结构（SMP、UMA）
- 分布式存储器多处理机

■两种存储器系统结构

- 共享地址空间
- 独立地址空间

■两种通信机制

- 共享存储器通信机制
- 消息传递通信机制



并行处理(parallel Processing)：在同一时间段内，在多个处理器中执行同一任务的不同部分。

- 并行算法的**加速比** (speedup) 或**加速系数**(speedup factor)：完成计算的最佳串行算法所需的时间与完成相同任务的并行算法所需的时间之比。
- **并行效率** (parallel efficiency)：加速比与所用处理器个数之比。并行效率表示在多核处理器执行并行算法时，平均每个处理器的执行效率。

并行处理加速比

第 126 页

$$\text{加速比 } S(p) = \frac{\text{使用单处理器最佳串行算法执行时间}}{\text{在具有 } p \text{ 台处理机的系统上算法的执行时间}}$$

- 对于 p 个处理器来说，当计算任务可被分成相等执行时间的进程时，一个进程映射到一个处理器上时，若此时没有其它开销就可获得最大加速比 p 。

- p ：并行系统中处理器的个数 (the number of processors)

- W ：问题规模或工作负载 (workload)

- W_s ：应用问题中必须串行执行的部分

$$W = W_s + W_p$$

- W_p ：应用问题中可并行执行的部分

- f ：串行执行部分 W_s 占全部工作负载 W 的比例

$$f = W_s / W$$

并行处理加速比

第 127 页

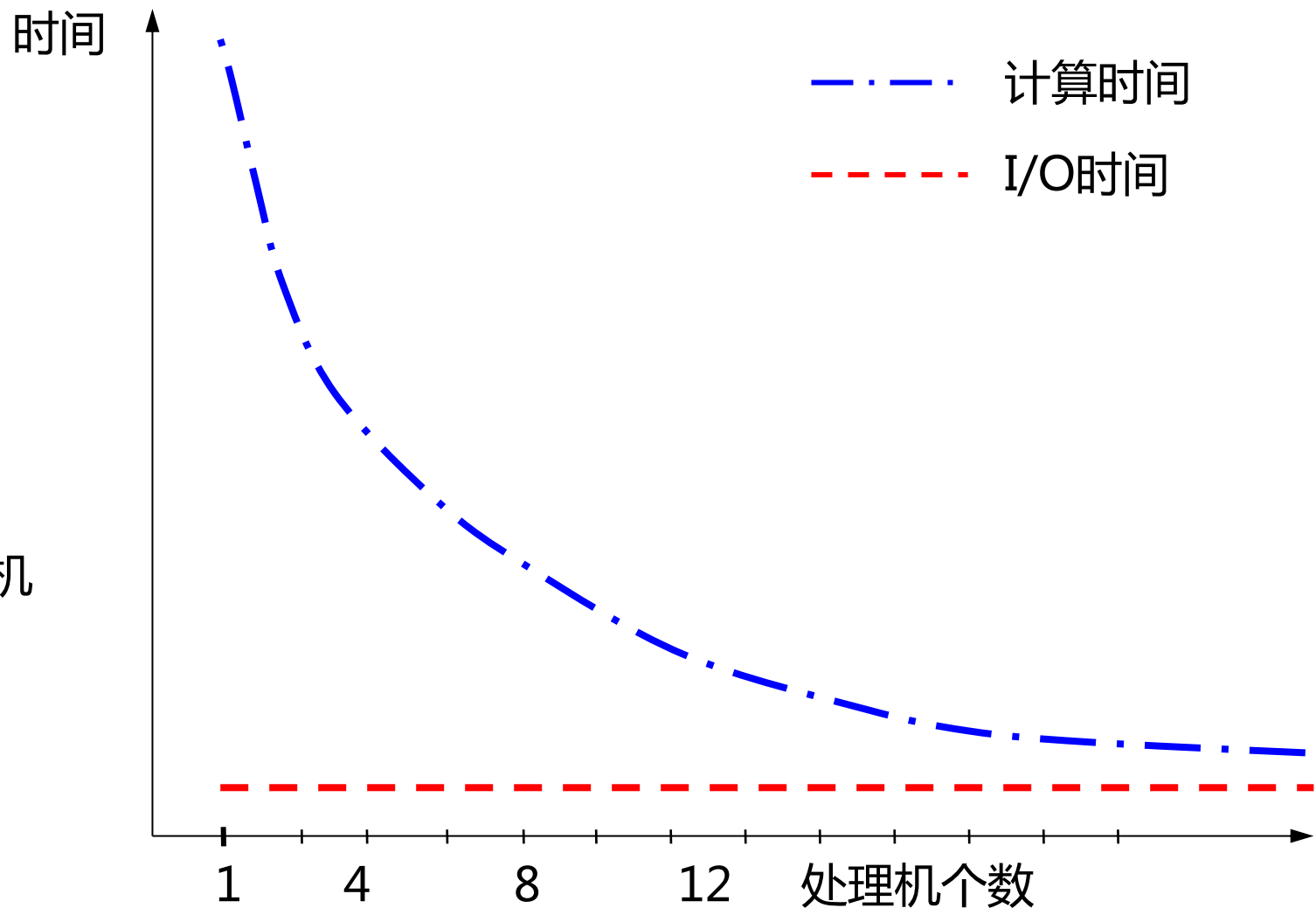
$$\text{加速比 } S(p) = \frac{\text{使用单处理器最佳串行算法执行时间}}{\text{在具有 } p \text{ 台处理机的系统上算法的执行时间}}$$

- t_s : 整个任务在串行机上执行所需的时间
- t_p : 整个任务在并行机上执行所需的时间
- T_s : 完成 W_s 所需要的时间
- T_p : 在并行机上完成 W_p 所需要的时间
- S : 加速比 : $S(p) = t_s / t_p$
- E : 并行效率 : $E = S(p) / p * 100 \%$

■ Amdahl定律

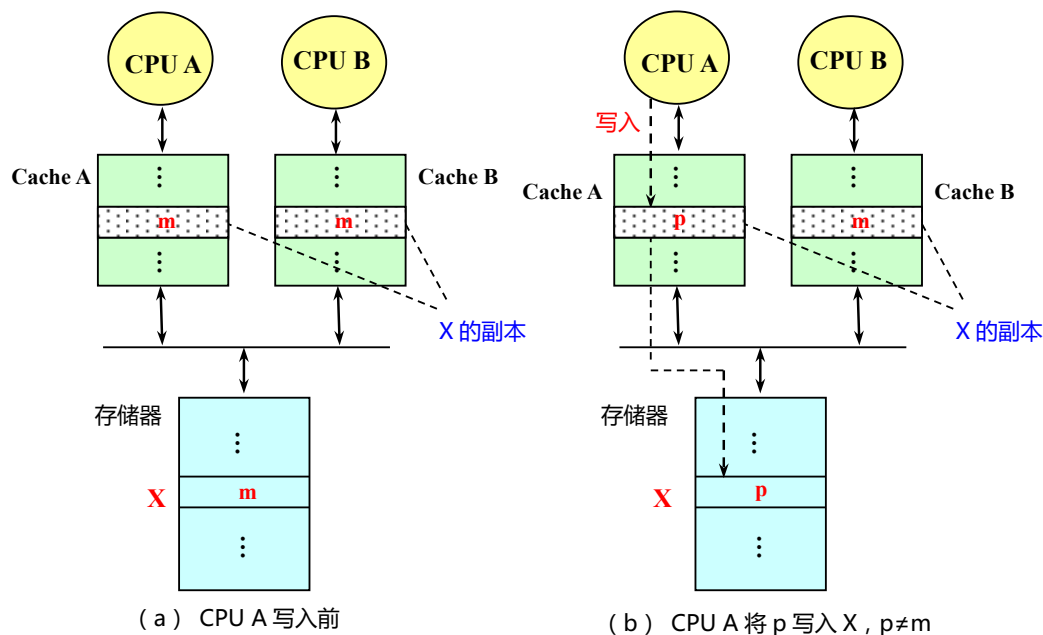
$$S(p) = \frac{1}{f + (1-f)/p}$$

- 当 p 无限增加时，加速比趋于 $1/f$
- 主要缺点：固定负载妨碍了并行机性能可扩展性的开发



■多处理机的Cache一致性问题

- 允许共享数据进入Cache，就可能出现多个处理器的Cache中都有同一存储块的副本，
- 当其中某个处理器对其Cache中的数据进行修改后，就会使得其Cache中的数据与其他Cache中的数据不一致。



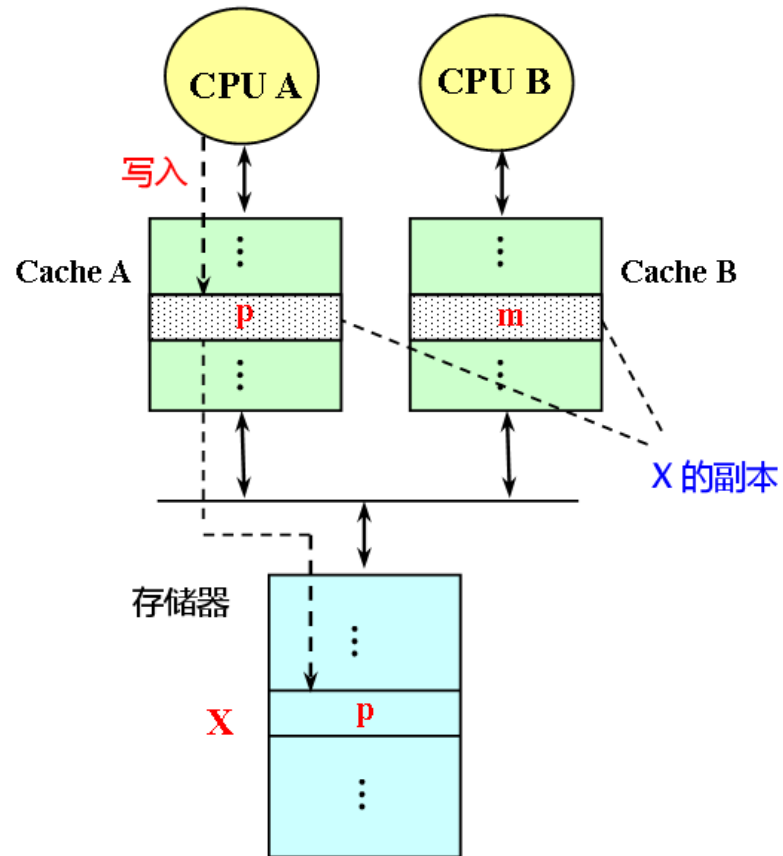
由两个处理器（A和B）读写引起的Cache一致性问题

实现一致性的基本方案

第 130 页

1. Cache一致性协议：在多个处理器中用来维护一致性的协议。

■ 关键：跟踪记录共享数据块的状态



思路1：如果能够监听数据的某一副本发生变化？

➤ **协议一：监听式协议 (snooping)**

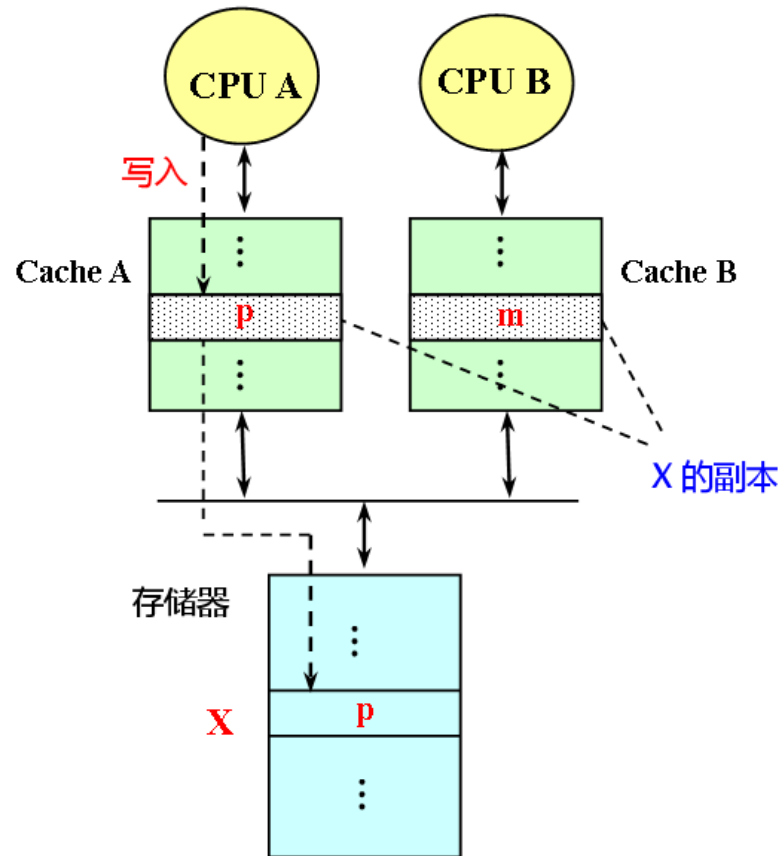
- ✓ 每个Cache除了包含物理存储器中块的数据拷贝之外，也保存着各个块的共享状态信息。
- ✓ Cache通常连在共享存储器的总线上，当某个Cache需要访问存储器时，它会把请求放到总线上广播出去，其他各个Cache控制器通过监听总线（它们一直在监听）来判断它们是否有总线上请求的数据块。如果有，就进行相应的操作。

实现一致性的基本方案

第 131 页

1. Cache一致性协议：在多个处理器中用来维护一致性的协议。

■ 关键：跟踪记录共享数据块的状态

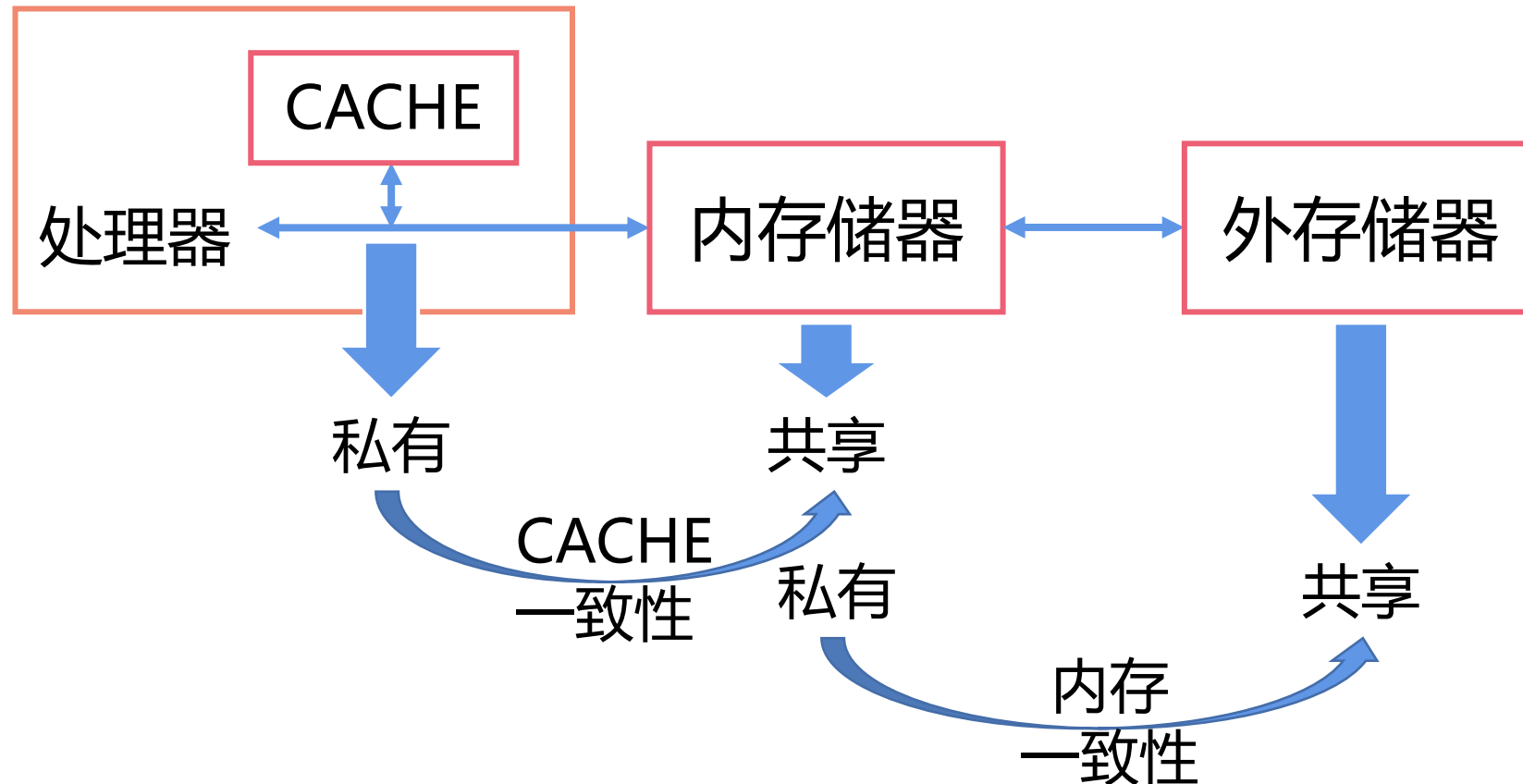


思路2：如果数据的所有副本存储与变化均被记录下来？

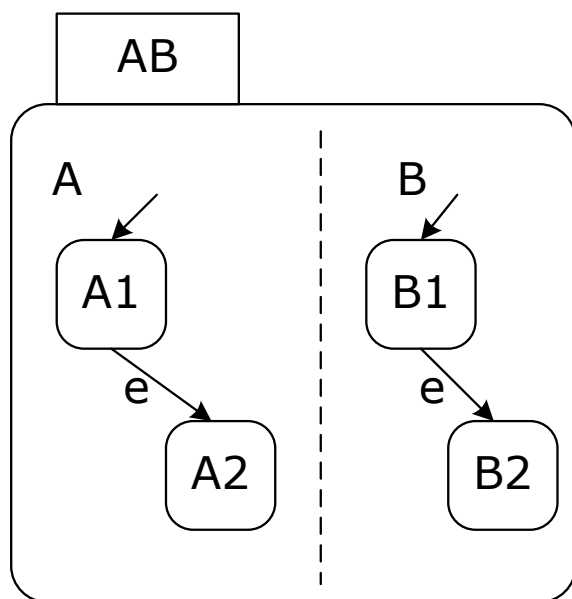
➤ 协议二：目录式协议 (directory)

- ✓ 物理存储器中数据块的共享状态被保存在一个目录；
- ✓ 关键是建立一个目录，维护每个缓存块的一致性状态，跟踪哪些缓存拥有缓存块以及处于什么状态；
- ✓ 需要发出一致性请求的缓存控制器直接发送给目录，再根据目录确定接下来要采取的行动。
- ✓ 目录协议分为3类：全映象目录、有限映象目录、链式目录

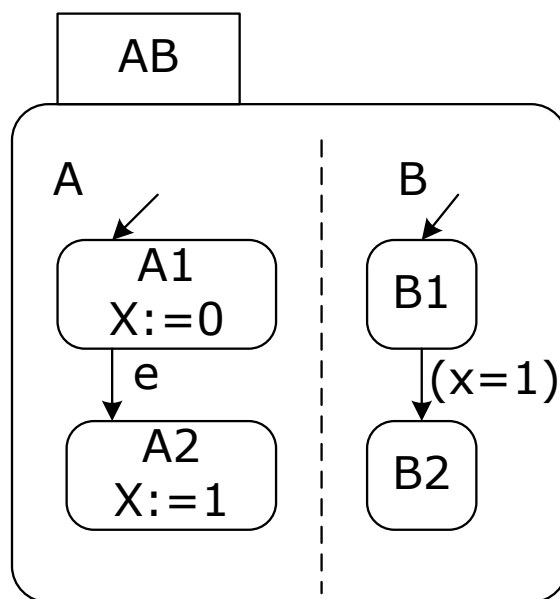
存储系统 通过软、硬件结合，形成了CACHE—内存—外存储器组合的层次存储



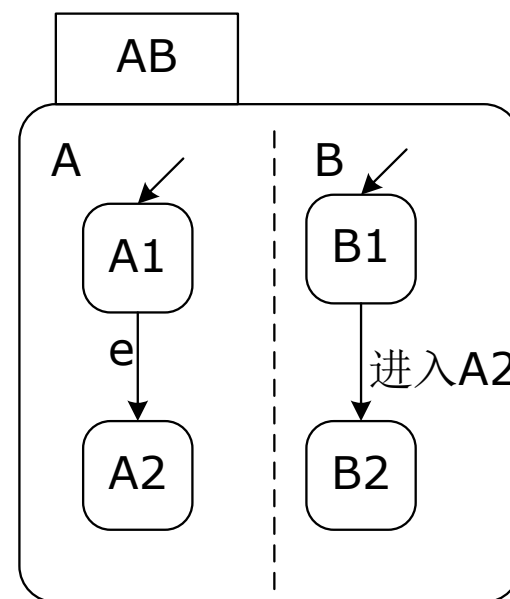
同步机制通常是在硬件提供的同步指令的基础上，通过用户级软件例程来建立的。



(a) 基于公共事件的同步



(b) 基于公共数据的同步

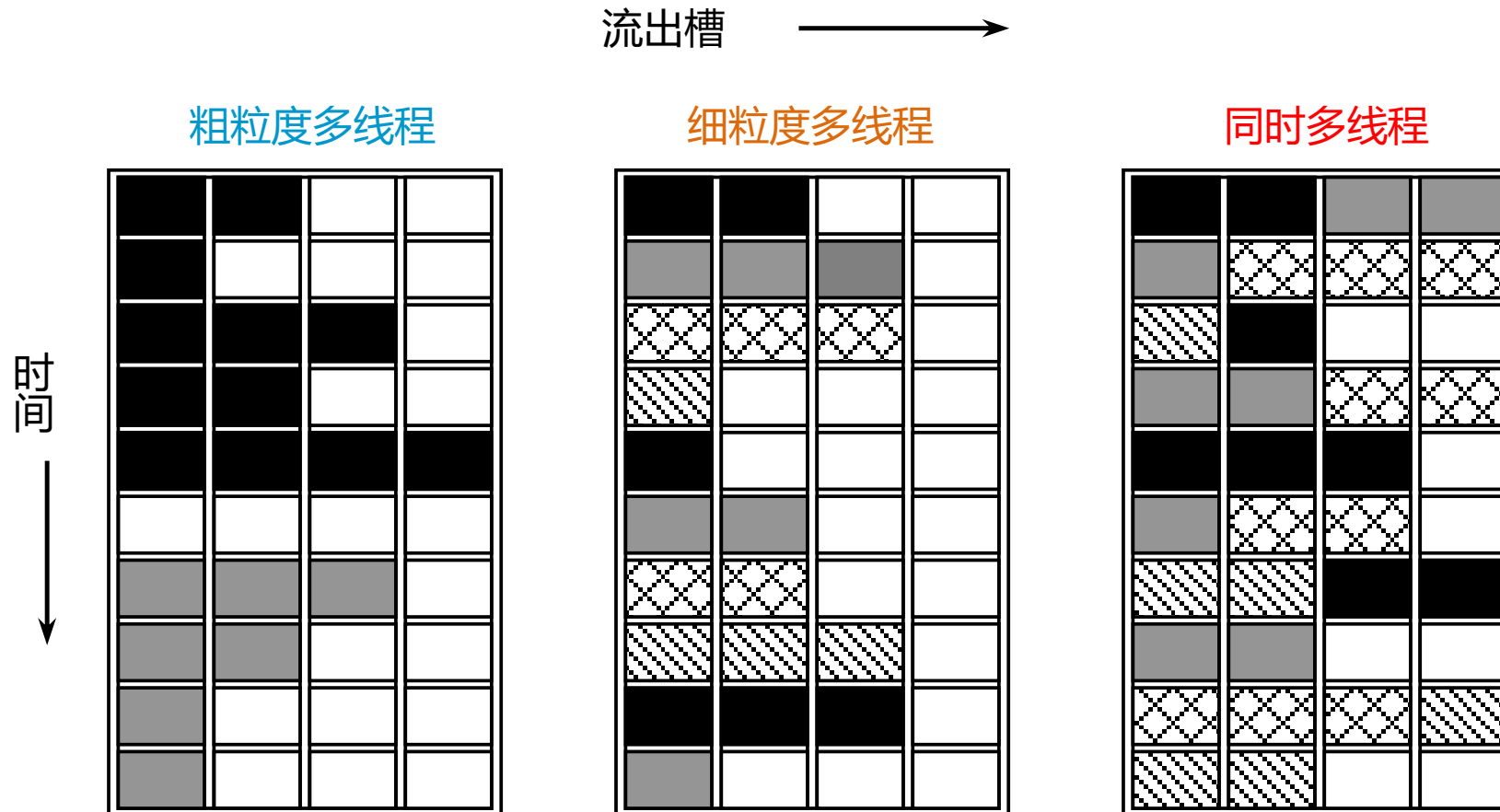


(c) 基于状态监测的同步

将线程级并行转换为指令级并行

第 134 页

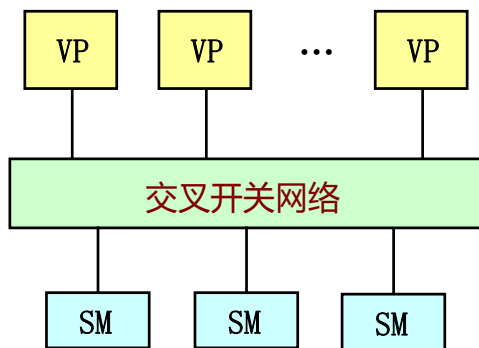
- 同时多线程技术 (Simultaneous MultiThreading , 简称SMT)
 - 一种在多流出、动态调度的处理器上同时开发线程级并行和指令级并行的技术。



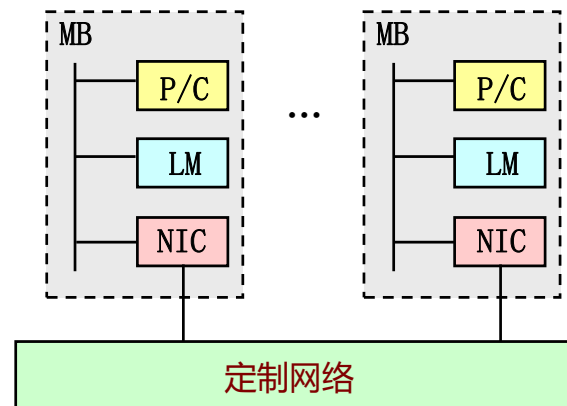
目前流行的高性能并行计算机系统

结构通常可以分成以下5类：

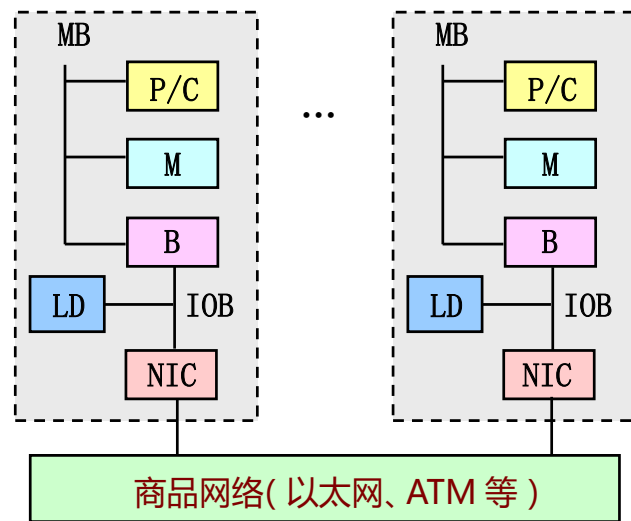
- 并行向量处理机 (PVP)
- 对称式共享存储器多处理机 (SMP)
- 分布式共享存储器多处理机 (DSM)
- 大规模并行处理机 (MPP)
- 机群计算机 (Cluster)



(a) PVP



(b) MPP



(c) 机群

VP : 向量处理器
SM : 共享存储器模块
P/C : 商品微处理器/Cache
LM、M : 本地存储器
NIC : 网络接口电路
MB : 存储器总线
LD : 本地磁盘
IOB : I/O 总线
B : 存储总线与 I/O 总线之间