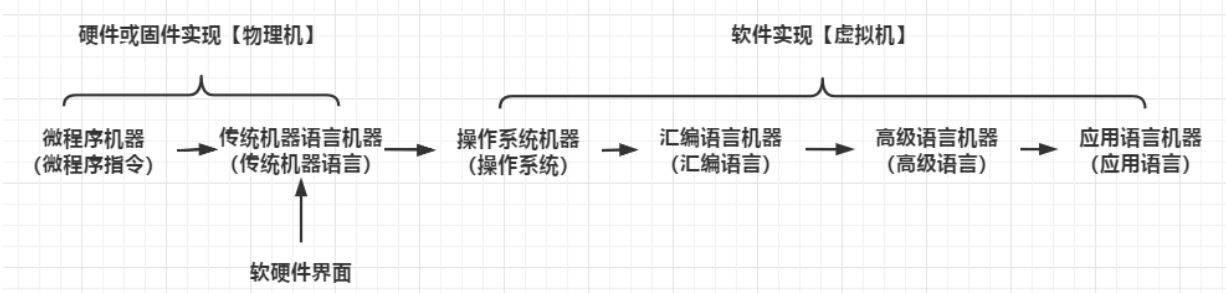
其实可以看做计组的补充

从计算机语言的角度，把计算机系统按功能划分为**多级层次结构——每层以一种语言为特征 操作系统在汇编下面！**

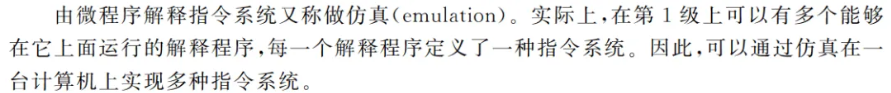


**翻译、解释和仿真：**

翻译：转换程序把高一级机器上的程序转换为低一级机器上等效的程序 （速度快，占用存储空间大）

解释：把高一级机器上程序的**每一条语句**转换为低一级机器上的一段等效程序并执行（**速度慢，占用存储空间小**） 一解 解释完还要执行

**仿真：用微程序来解释指令系统！**



翻译：全部翻译一遍再说，所以占的空间大，速度快

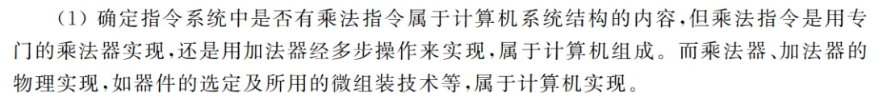
（一般，**左边那三个是解释实现，右边三个是翻译** 右边四个是软件实现（笼统））

微程序语言！——最底层的微程序机器层，是硬件逻辑的直接实现方式。它控制着CPU内部的微操作，**传统机器语言在其之上，是CPU可直接执行的二进制指令集**

**计算机系统结构的经典定义**！——**传统机器程序员所看到的计算机属性，即概念性结果与功能特性**

计算机系统的实质：——确定软硬件的界面

计算机实现——计算机组成的物理实现



例子：**系列机：由统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**

系统结构分类：

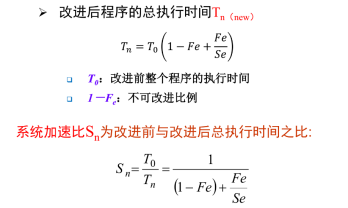
Flynn分类法【**依据指令流和数据流的多倍性**】：SISD，SIMD；MISD；MIMD

冯氏分类法【**依据最大并行度**】：**字串位串；字串位并；字并位串；字并位并**

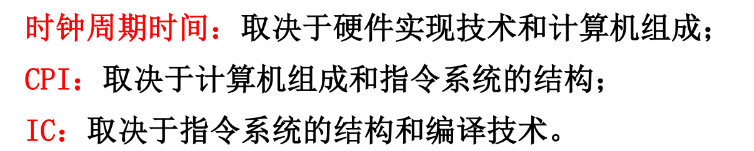
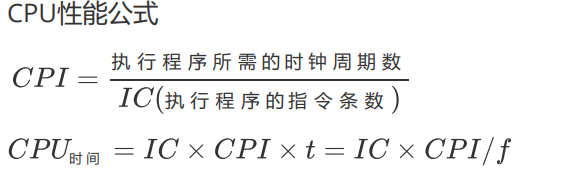
**最大并行度：计算机系统在单位时间内能够处理的最大的二进制位数。**

handler分类法【**依据计算机并行度和流水线**】：三个层次（程序控制器PCU；算数逻辑部件ALU或运算部件PE；运算部件所包含的位级电路BLC的套数）

Adaml定律：



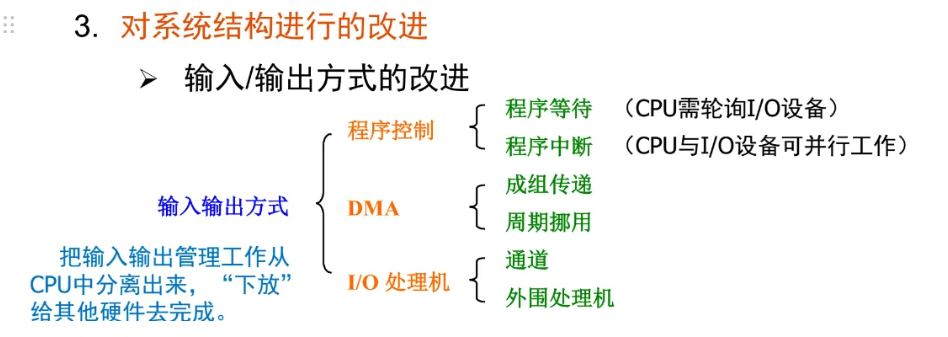
CPU性能计算： **IC是程序的指令条数！ 优化CPI**



计算机系统的设计：

**由上往下【专用机设计**，不适用于通用机】，由下往上【软件处于被动，会造成软硬件脱节】，从中间开始【传统机器语言机器 级与操作系统机器级之间】

冯诺依曼结构：**运算器为中心**；指令和数据同等对待；存储器顺序线性编址；顺序执行，程序分支由转移指令实现；指令数据都是二进制码 **存储程序原理（程序预先放在存储器中，指令驱动）**



成组传递：一次DMA请求批量传输多数据块。  
在DMA控制器介入下，外设与主存间直接完成整组数据连续传输（如一个磁盘扇区），仅开始和结束时通知CPU，大幅减少中断次数。

周期挪用​​  
——DMA抢占空闲总线周期窃取传输时机。  
当CPU未使用系统总线时，DMA控制器立即占用总线周期进行数据传输；若CPU正使用总线，则暂停其1~2个周期完成I/O操作（又称“周期窃取”）

通道​​  
——专用I/O处理器独立执行通道程序。通道是弱化版协处理器，通过执行内存中的通道指令控制外设

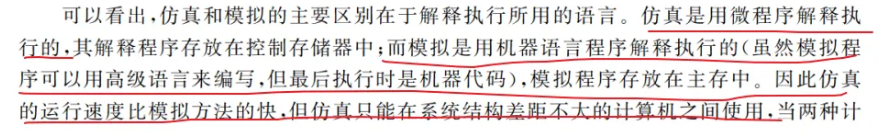
外围处理机​​  
——全功能协处理器接管复杂I/O管理。  
本质是独立于CPU的微型计算机，拥有完整指令集和寄存器，可执行数据格式转换、错误校验等任务，彻底解放CPU。

CISC和RISC：注意RISC有更多的寄存器

实现软件在不同机器上**移植？——仿真与模拟**：

模拟：用软件的方法在一台现有机器（宿主机）上实现另一台机器（虚拟机） 直接在机器语言上

仿真：用一台现有机器（宿主机）上的**微程序解释实现**另一台机器（目标机）的指令集



并行性（parallelism）：同一时刻或同一时间间隔内进行操作

分：同时性（simultaneity） **并发性（concurrency）**

提高并行性的技术途径：**时间重叠 资源重复**（引入空间因素，以数量取胜） **资源共享**（软件方法，多个任务按一定时间顺序轮流使用一套设备 **例子是分时系统**）起主导作用的是时间重叠原理

流水线：

**每个子过程及其功能部件称为流水线的级或段（stage），段与段相互连接形成流水线，流水线的段数称为深度**

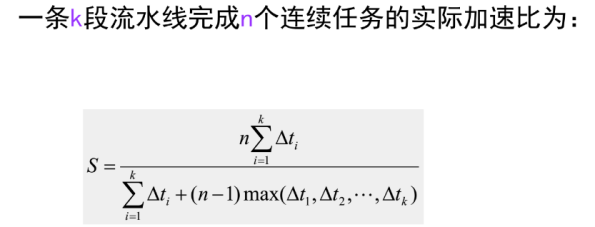
流水线每一个段后面通常都有一个缓冲寄存器,称为**流水寄存器**。**在相邻两段间传送数据**，有缓冲、隔离、同步作用

分类：

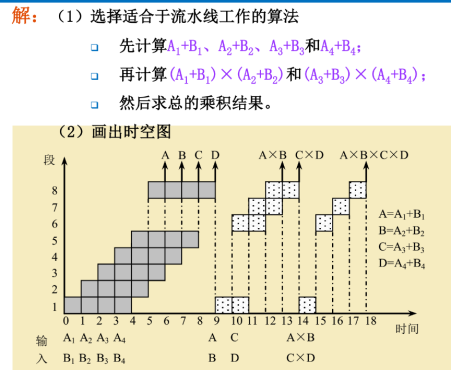
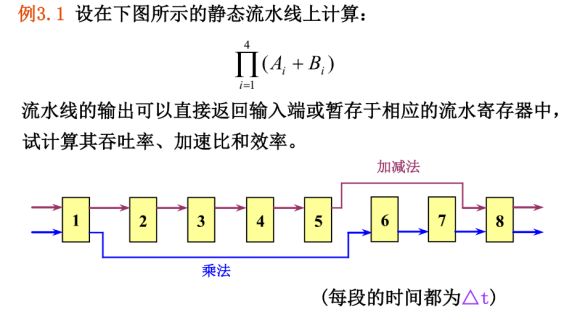
注意按功能：单功能 多功能 **多功能又分为静态流水**（要切换功能，必须等前面的任务都流出流水）和**动态流水（ps：任何一个功能段只能参加到一种连接中**

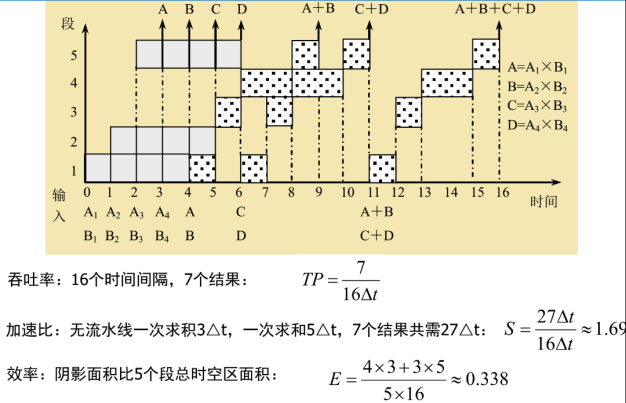
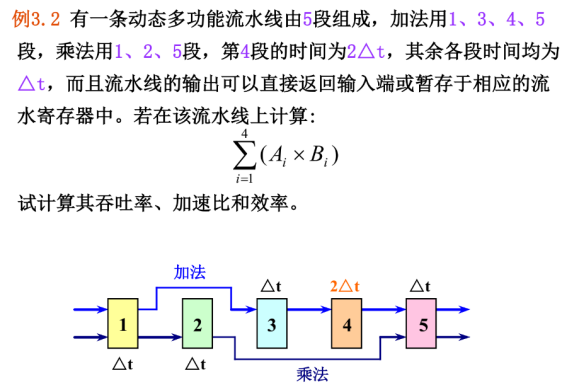
性能指标：

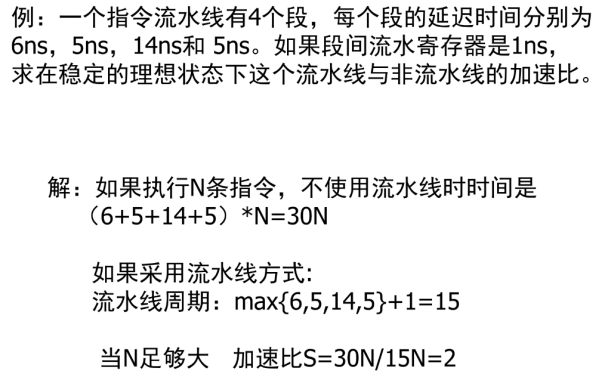
注意瓶颈问题： **效率就是n个任务占用的时空面积和k个段总的时空面积之比**



计算举例：经典例子！ **先给算术分流程！**





（**不均匀时，流水线周期取决于瓶颈段**）

五段流水线：

**取指令【IF】** 以PC值为地址，从存储器中取值放入指令寄存器IR

**指令译码【ID】**：对指令译码，访问通用寄存器，读出所需操作数 拿操作数！**注意冲突可能拿不到！——时空图**

**执行 / 有效地址计算【EX】**

ALU：执行计算

load / store ：形成访存有效地址

分支指令：形成转移目标地址

**存储器访问 / 分支完成 【MEM】** 【该周期只有load store和分支指令】

store：把指定的数据写入存储器单元

load：从存储器中读取相应的数据

分支：若分支成功，**将转移目标地址送入PC**

ALU在此为空操作

**写回【WB】：把结果数据写入通用寄存器组**

ALU：结果数据来自ALU 【与ID段读寄存器可能会造成冲突，解决方法：WB写操作在前半周期，ID 读操作在后半周期】

load：结果数据来自存储器

分支 / store：空操作 分支指令和ALU指令、store指令需要4个周期；load指令需要5个周期。

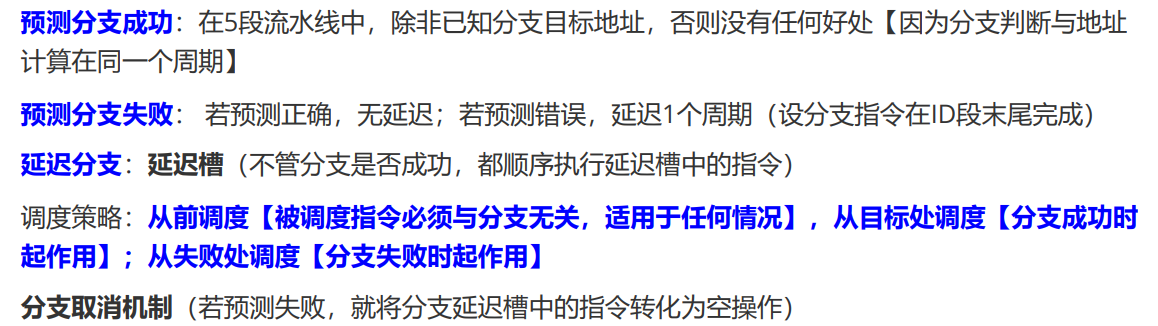
相关与冲突：

数据相关（也称真数据相关） 名相关 控制相关

**名相关**（name dependence）：如果两条指令使用相同的名（访问的寄存器or存储器单元名称），但他们之间并没有数据流动（所以可以用换名技术解决）

包括**反相关：指令j写的名和i读的名相同WAR**；**输出相关**：j和i写相同的名 WAW

**结构冲突**（某种指令组合因为资源冲突而不能正常执行）、**数据冲突、控制冲突**



向量处理机：

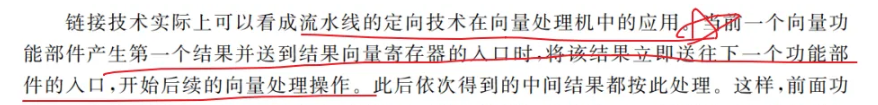
**向量由一组有序，具有相同类型和位数的元素组成**，在计算机系统中经常使用，特别适合流水处理，所以有的流水线对应设置了向量数据表示和向量指令——这种处理机称为向量处理机

横向处理 纵向处理（适合存储器-存储器型） 纵横（分组）处理（组内纵向，组间横向，适合寄存器-寄存器型）

**提高处理机性能——4个方法**：

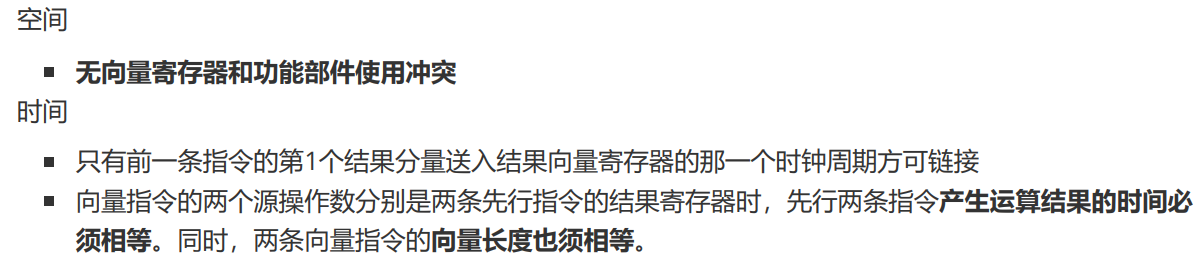
**方法一——设置多个功能部件（部件按流水的方式并行）**

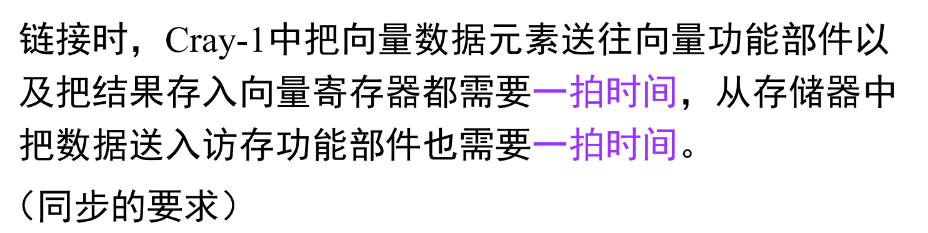
**方法二——链接技术 流水线定向在向量处理机的应用**



前面功能部件的结果元素一产生就被用，**而不用等结果向量全部产出**后再用（对于某个功能部件而言，他要执行向量长度次数的操作）

（链接的要求要注意：不能有功能部件冲突比如两条指令都是add，还有源寄存器冲突等）





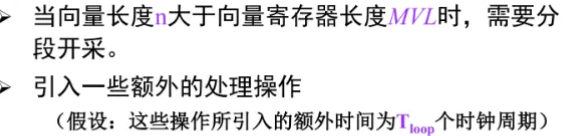
**方法三——分段开采技术：**

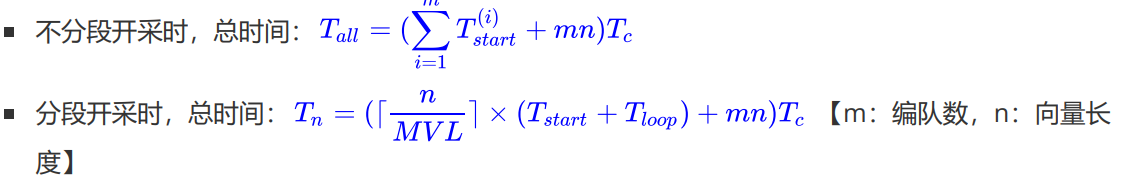
当向量长度大于向量寄存器的长度时，需要把长向量分成长度固定的段，然后循环分段处理，每次循环处理一个段（由系统自动完成，对程序员透明）

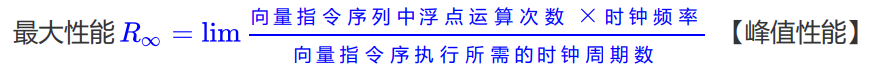
**方法四——采用多处理机系统**

编队：能在同一周期内开始的向量指令（**不存在流水功能部件的冲突，资源冲突或者数据相关性**）

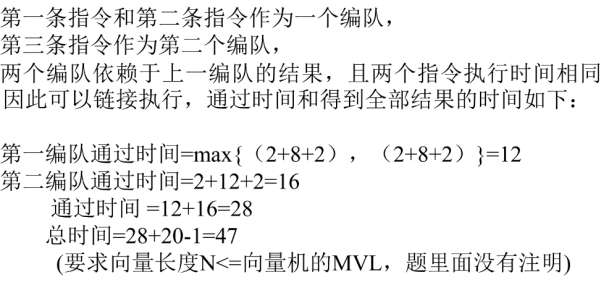
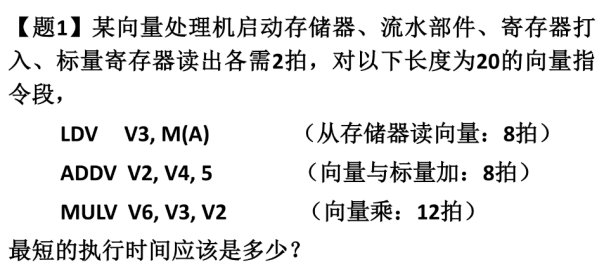
**一个编队的执行时间由各指令时间的最大值决定**

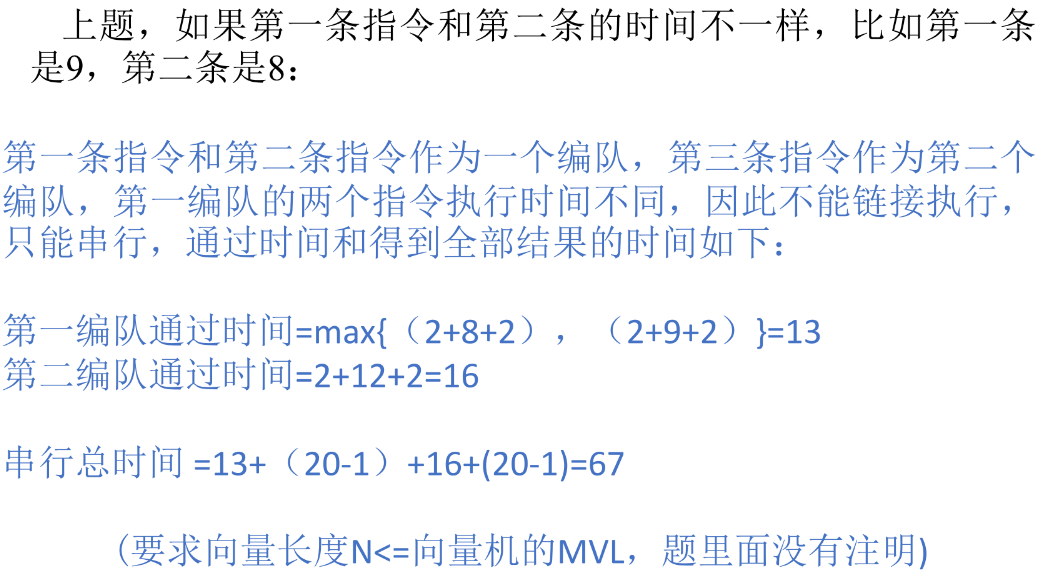




单位MFLOPS

半向量长度：到达最大性能一半所需要的向量长度（越小越好）





**相关是程序的一种固有属性！它反映程序中指令的依赖关系**

正确执行程序，**必须保持最关键的两个属性：数据流和异常行为**

指令的调度：静态和动态 静态靠编译器

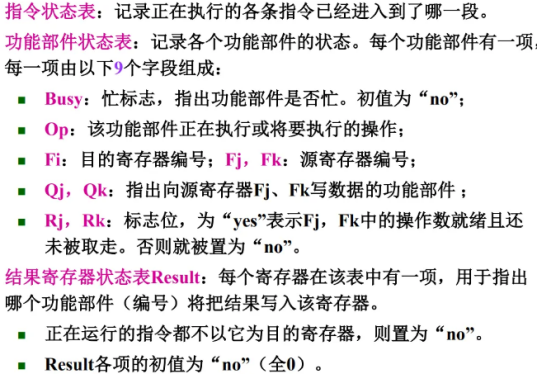
两种典型的动态调度算法！——记分牌算法和tomasulo算法

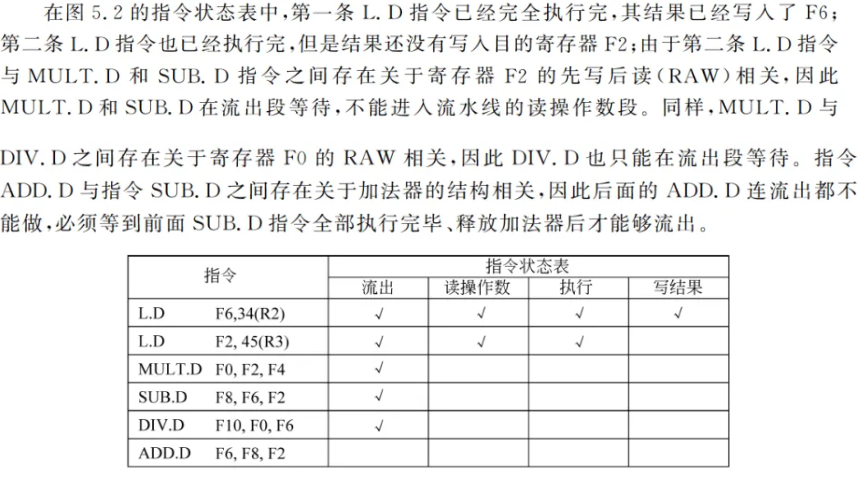
动态——硬件——记分牌

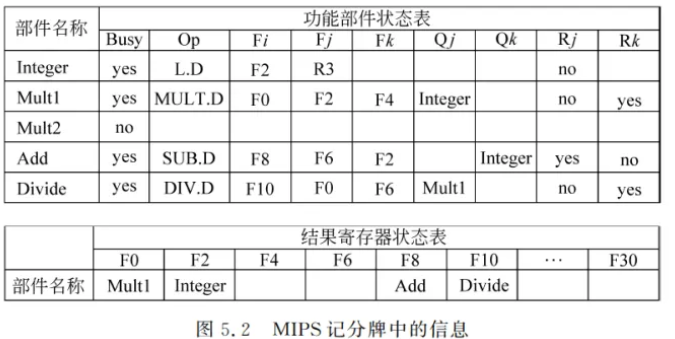
硬件中维护了**三张表**：记录**指令的执行状态，功能部件状态，寄存器状态**以及数据相关关系

记分牌的目标**：没有结构冲突时，尽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

三个表：**指令状态表 功能部件状态表 结果寄存器状态表**



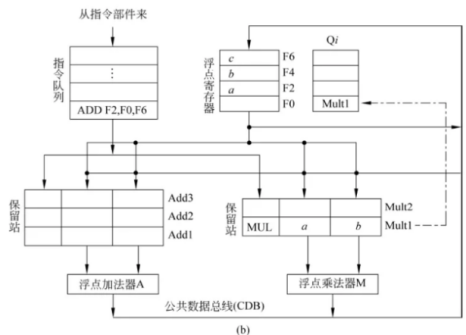
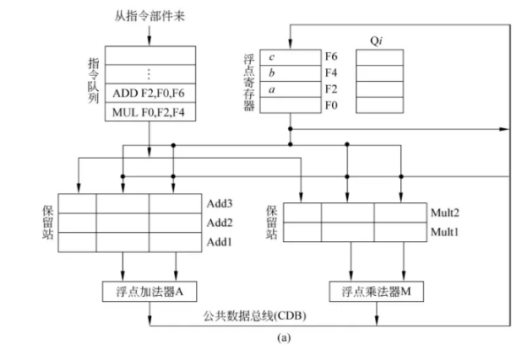




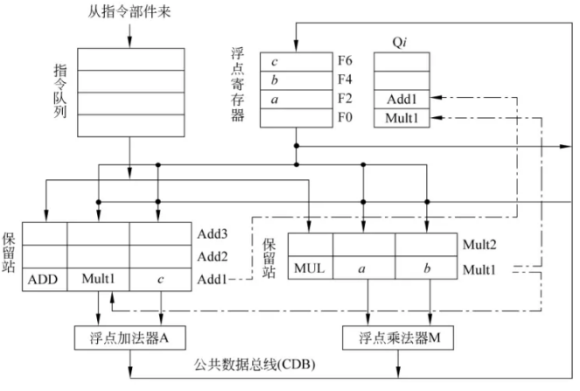
**Tomasulo（公共数据总线法）**：**通过寄存器换名**来消除RW冲突和WW冲突，核心思想是**记录和检测指令相关**，操作数一旦就绪就立即执行，把RAW的可能性降到最小

**保留站设置在运算部件的入口**（看图，要用该功能部件的指令在这里排队，先入先出）

公共数据总线CDB——所有功能部件的计算结果都送到CDB上面！由它把这些结果送到需要的地方 Tomasulo的经典特征是换名操作



首先mul指令流出到浮点乘法器前设置的保留站中，在f2和f4中的操作数a和b也一起流到保留站（**这样以后指令就跟这俩寄存器没关系了，执行时直接从保留站中拿数据**），目的寄存器F0对应的Qi设置为mult1（状态表，即浮点乘法器保留站1的标识，表示f0的内容由这个提供）



后面的add命令流到加法器保留站中，标识符为add1，**它缺的一个数据为f0（浮点寄存器中没找到）**，f0的标识为mult1，所以这里填mult1（表明它在等待保留站这个的结果，后续产生后将通过CDB传到），它的目的寄存器F2对应的Qi设置为add1

**动态分支预测技术：**

从转移指令近期转移是否成功的记录，动态地预测下一次转移的方向

分支历史表BHT（1位：完全取决于上一次 2位：只有连续两次预测错误才改变）

**分支目标缓冲寄存器BTB：**

将分支成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存

格式：**[成功转移的分支指令地址 ，转移目标指令地址]**

缓存的意思：仅仅知道分支是否预测成功还不够，还需要尽早知道分支目标的地址

在每次取指时，与BTB所有项目的第一字段比较，有匹配的，那么上一次也执行成功，从而预测这一次也执行成功，拿地址。

基于硬件的前瞻执行：

对分支指令结果进行猜测(假设猜测总是对的),并按猜测结果继续取指、流出和执行后续的指令，但结果不再写回，而是**写入再定序缓冲器ROB，等相应指令得到确定后再写回**

多流出处理机两种基本风格！——超标量和超长指令字

超标量：每个周期流出的指令条数不定（上限n，称n-流出） 由编译器or硬件完成调度 最通用（MIPS：一般一条整数一条浮点指令）

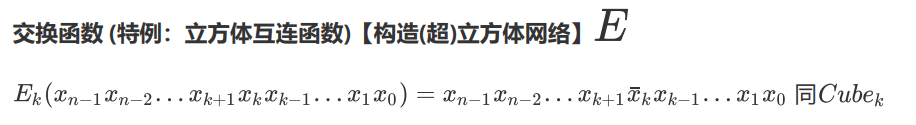
超长指令字VLIW：每个周期流出一条超长指令（多个单操作数指令构成一个超长指令，或称一个指令包） 编译器调度（静态）

互连网络：

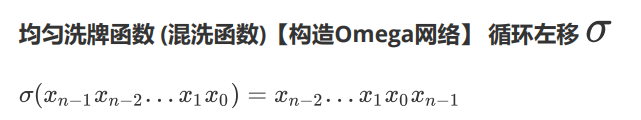
**第k个子函数：s作用于输入端的二进制编号的低k位**

**第k个超函数：s作用于输入端的二进制编号的高k位**

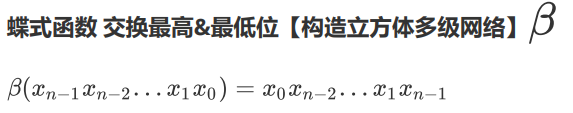
静态互连网络：

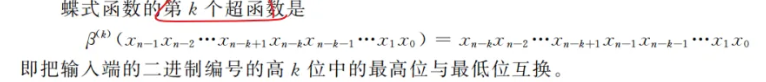


E：exchange **第k位互反** 构造立方体网络

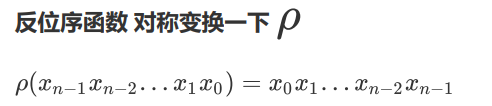


均匀洗牌：**输入循环左移一位！（圆圈就像循环）** 逆均匀洗牌：σ-1 右移一位

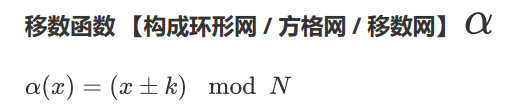
**β像蝴蝶翅膀**



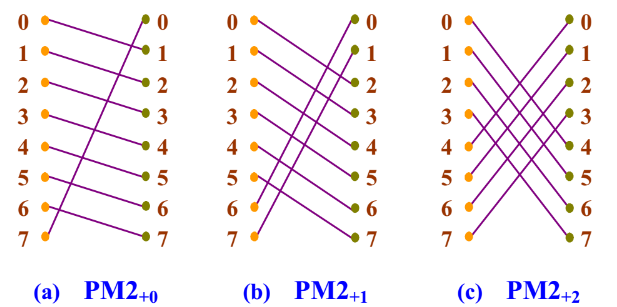
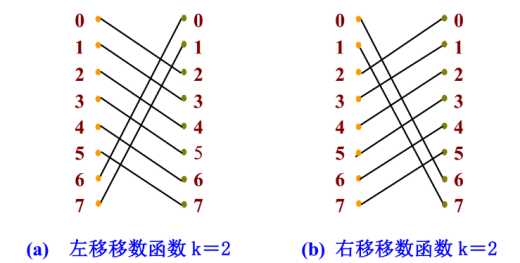
高低位互换



反位序：二进制编号倒过来

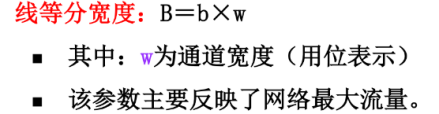
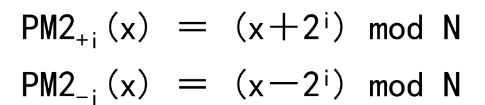


移数α：就是把输入和输出错开一定位置，比如：



PM2I（是i的大写）：——加减2^i函数（特殊的移数函数）

PM2I网络有2n个互连函数（n+n）



**处理器与哪个处理器连接？——不仅考虑连入，还有连出！**

互连网络的结构参数：

网络规模N：网络中结点的数目（表示该网络能连接的部件的数量）

节点度d：与节点相连的边数（通道数），包括入度和出度

**结点距离：对网络中任意两个节点，从一个点到另一个点跨越边数的最小值**

**网络直径：任意两个节点距离的最大值**

等分宽度：把N个节点的网络切成N/2的两半（数目近似相同），**各种切法中，沿切口边数的最小值** 线等分宽度——见上

**对称性：从任何节点看到的拓扑结构都相同**

线性阵列（不对称哦）、环和带弦环、全连接、**循环移数网络（环上每个节点到所有与其距离为2的整数幂的节点都有边）**、树形（不对称）、**星形（二层树**，非二叉树，不对称）

还有网格形、胖树（目的是为了缓解一般树根节点以及根节点链路负载过重的问题）、Illiac、环网形

**（Illaic：列首尾连，行尾和下一个头连） （环网：行列均首尾自连）**

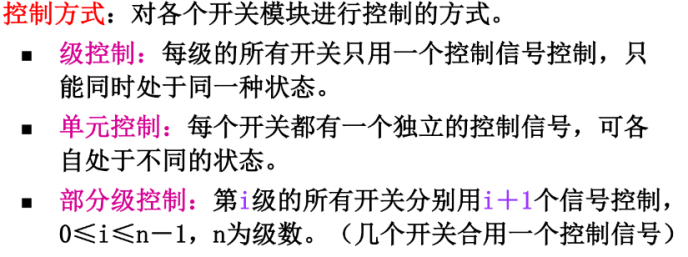
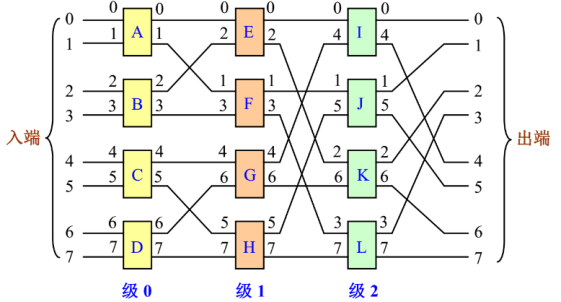
超立方体、带环超立方体（n-CCC）

动态互连网络：

总线，交叉开关网络，多级互连网络

Staran：

**第i级开关实现的是cubei交换函数 cubei：第i低位取反——交换连接**



Omega网络：多级混洗网络 **级间互连采用混洗（每一级都是循环左移，入口和出口也选一个）**

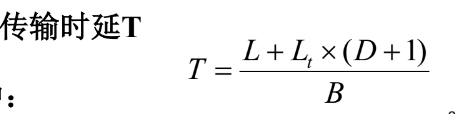
消息传递机制：

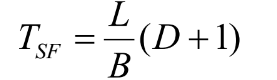
**消息是节点之间进行通信的逻辑单位**，有若干个包组成（packet），**包的长度固定，消息不定长**（可含不同数量的包），**包可以进一步分为更小的固定长度单位，称为片**

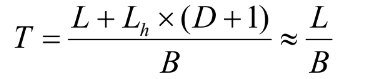
**线路交换/电路交换——以消息粒度运行**

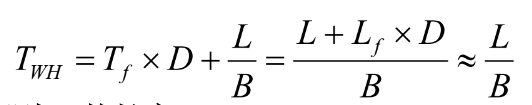
**包/分组交换——存储转发（包粒度）、虚拟直通（包粒度）、虫蚀（片粒度）**

传输时延计算：

电路：（L信息包长度，Lt建立路径信息包长度，D中间节点数，B带宽）

存储转发：（中间节点设包缓冲器，先存包，然后在出口链路和下一个节点包缓冲器可用的情况下传）

虚拟直通：（**改进版的存储转发，没必要等到信息包全部放入缓冲器再路由选择，收到寻径用的包头即可发** Lh：信息包寻径头部的长度 和电路交换公式一样）

虫蚀：（和上面一样，收到头片就可以转

多处理机：（即有多个CPU）

Flynn分类：SISD SIMD MISD MIMD

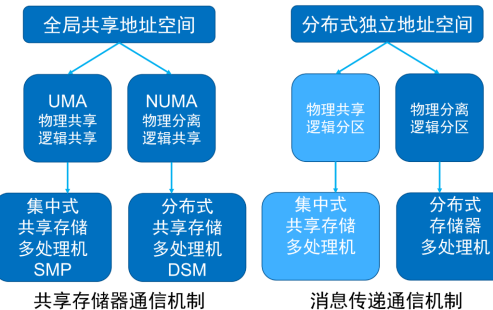
这里把现有的**MIMD分为两类：集中式共享存储器，分布式存储器**

集中式共享的：多处理器共享一个物理存储器，称为**对称式共享存储器多处理机SMP，结构称为UMA（uniform memory access）**，各处理器访存的时间相同

分布式存储器多处理机：存储在物理上是分布的，每个节点包含处理器，存储器，I/O，互连网络接口

**分布式存储器也有两种系统结构：共享全局地址空间（NUMA架构（DSM））、独立地址空间**

共享逻辑空间——所有机器都可以访问任何单元；非共享——只能有本地处理器访问



对称式共享存储器系统结构（SMP/UMA）： 每个CPU是自己有私有cache的

Cache的一致性问题：

**Cache一致性协议：**

**目录式协议：每个cache的共享状态被保存在一个称为目录的地方**

（设置目录表：记录每个可以调入Cache的数据块的状态位和若干指针位 ；目录承担了一致性协议操作的主要功能）

**监听式协议：每个cache保留共享状态的副本，通过广播和监听实现一致性**

两种**写更新操作方法**来解决Cache一致性问题：

**写作废协议：**在处理器对某个数据项进行写入之前，保证它拥有对该数据项的唯一的访问权。(先作废其它的副本，再写入)应用广泛，对于监听或者目录方式皆可用。

**写更新协议：**当一个处理器对某数据项进行写入时，**通过广播使其它Cache中所有对应于该数据项的副本进行更新**

当cache写完后，**更新存储器**时，两种策略：

写直达法：写入时不仅写入cache，也写入主存，主存的数据永远最新

写回cache：值写入cache，仅当cache中相应的块被替换时，写入主存

对于目录协议：

位向量是关键字段，**记录哪些cache中有副本**，**每一位对应一个处理器**，1/0表示该处理器有/无副本

由位向量指定的处理机的集合称为**共享集**

三类目录协议：全映像目录 有限映像目录 链式目录

阵列处理机：

array processor 一个由多个处理单元构成的阵列（**采用资源重复的方法**，设置较多处理单元来提高并行性） 多个处理单元**对各自的数据**进行**相同的运算**和操作（这是核心） 又称SIMD计算机！（**单指令流多数据**）

再次**强调单指令流多数据流 没有流水线！ 利用同时性**

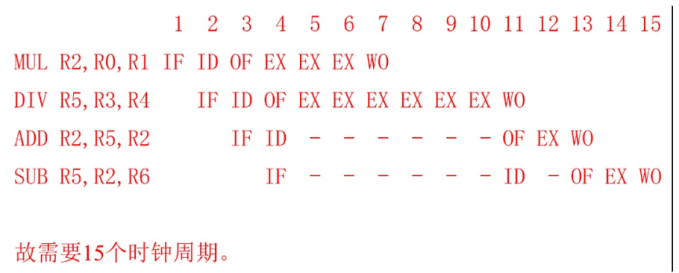
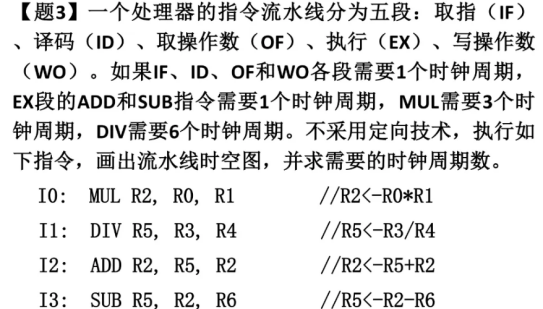
共享存储器的阵列机：

**共享多体并行存储器SM通过对准网络与各处理单元PE相连**

作业与习题：

算术平均、几何平均、调和平均

**记忆：调倒**

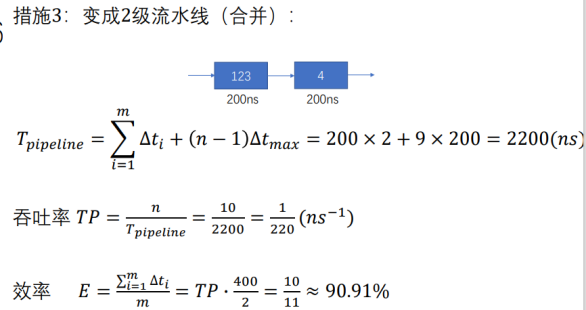
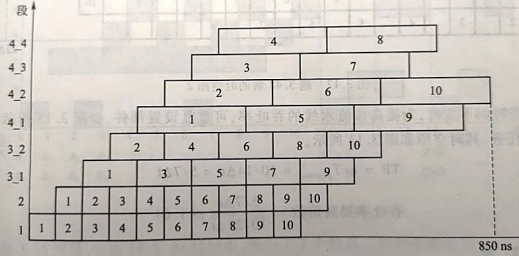


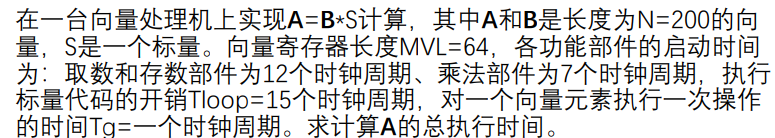
后两条指令，由于倒数第二条ID进不了OF拿数据，所以导致倒数第一条的ID跟着延迟

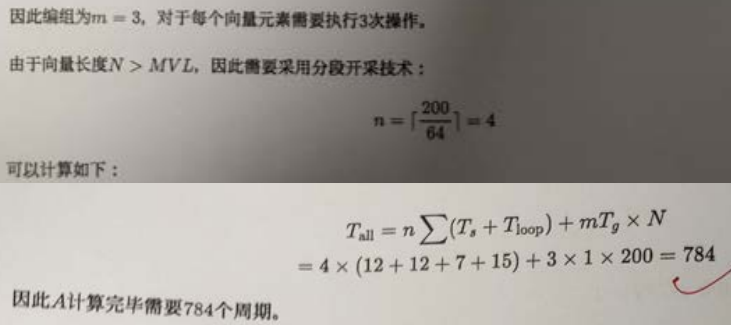
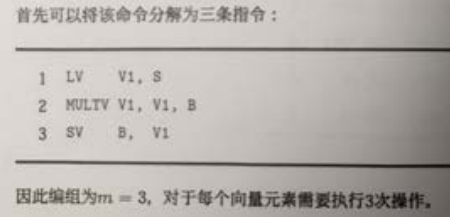
这其实也是个经典易错，只要它没进入下一阶段，就不会放这一阶段的部件，所以一定小心可能的stall！

**这里MUL和DIV不冲突**

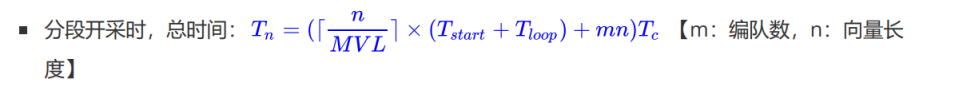
**重复设置部件画图：**







要看明白这题首先需要明晰分段开采的公式：



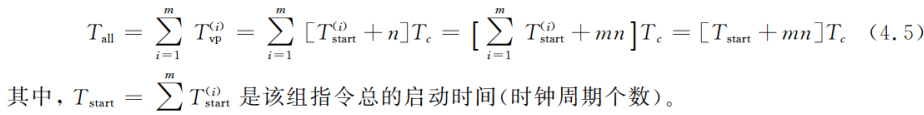
公式推导如下：

首先是一条向量长度为n的向量指令的执行时间：

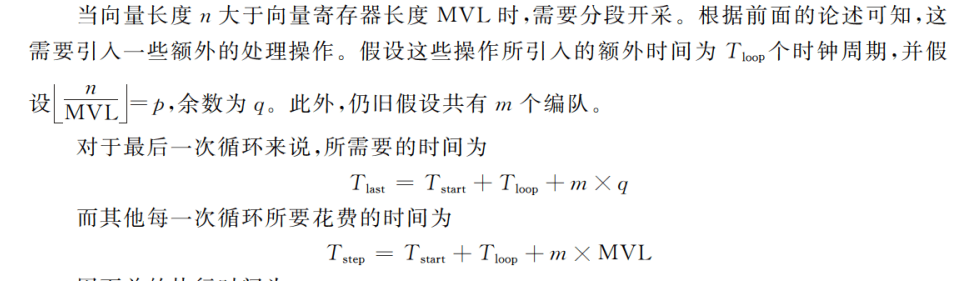
（类比流水线公式，只不过把-1挪到start了）

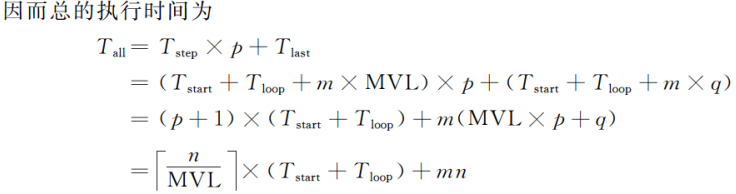
如果指令编队，那么指令序列的总执行时间应该为各编队时间和，即：

然后套用第一个公式：

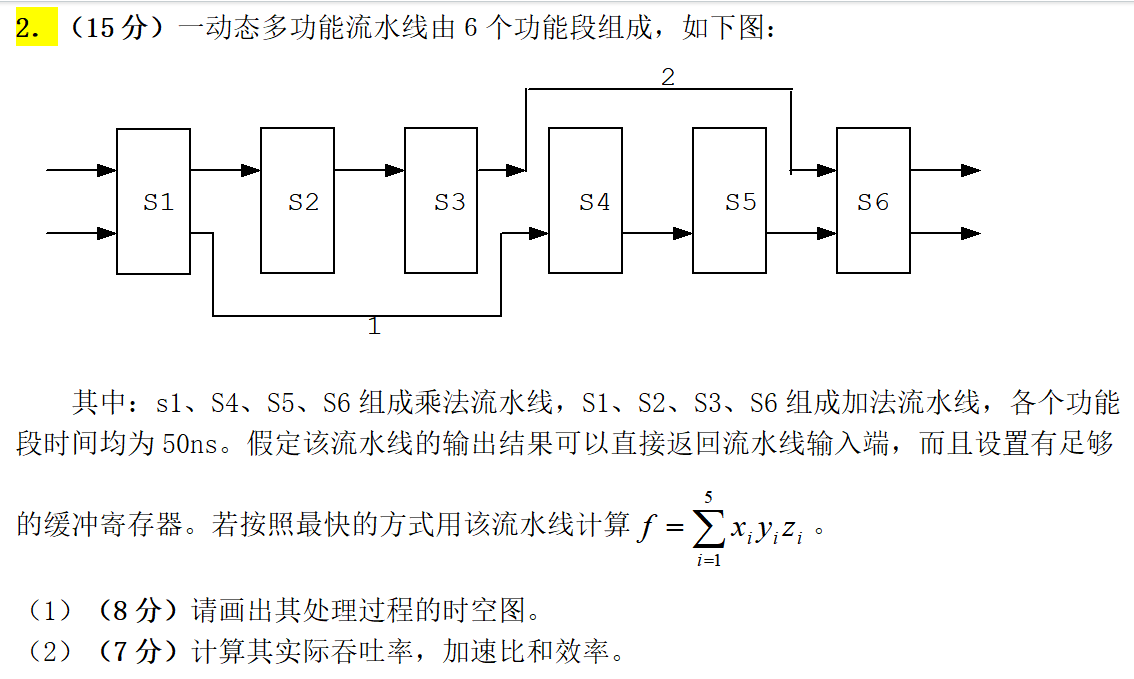


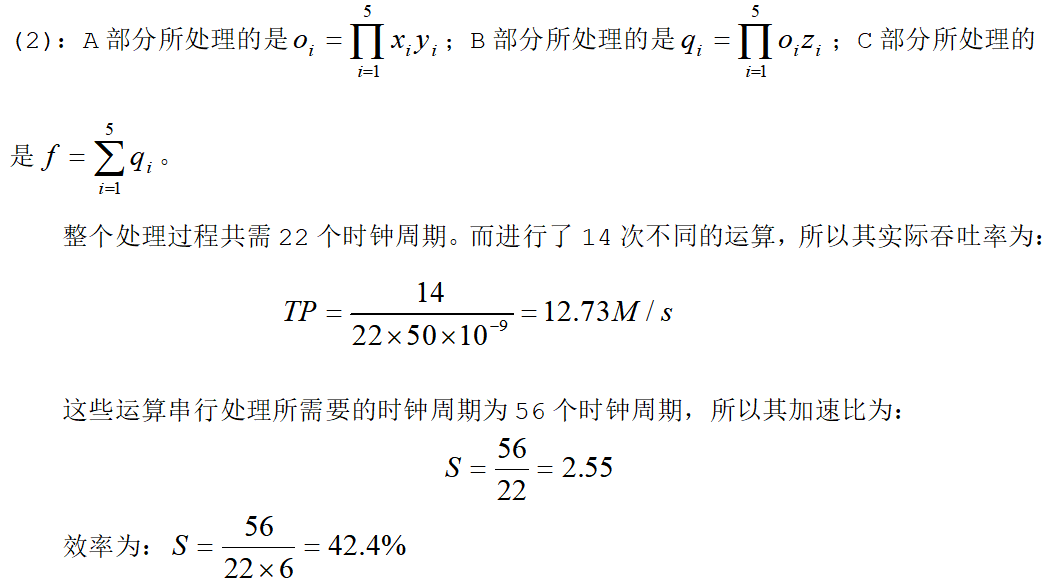
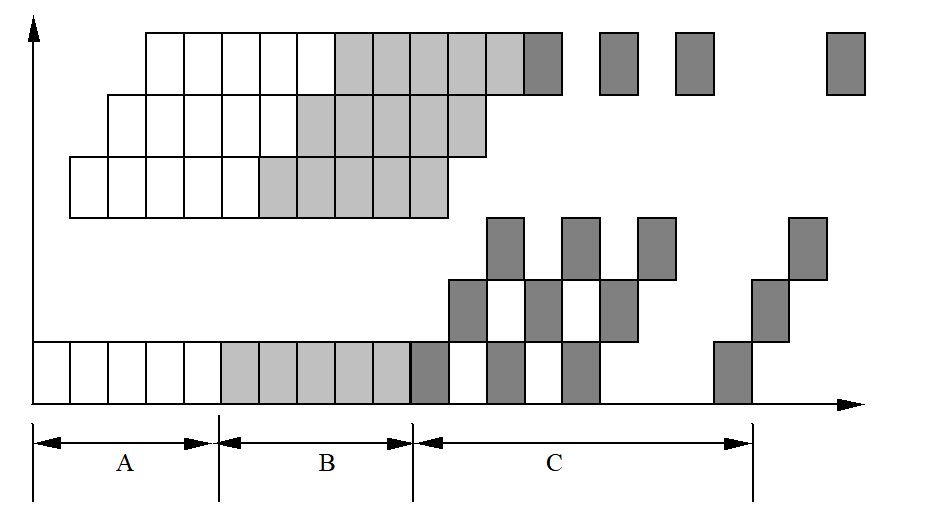
对于分段开采：

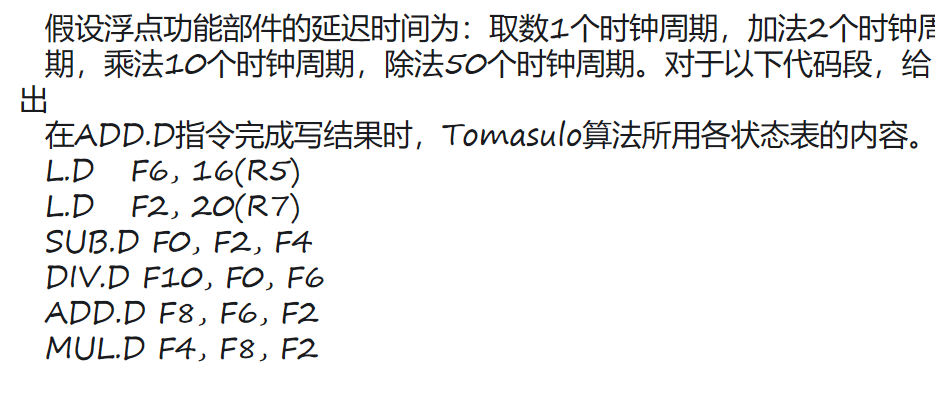




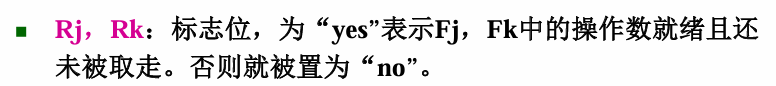
时空图好题：







此时在add前面的div还没执行完！



所谓取走：**该指令有没有进到读操作数这一步！ YES：已经生产好了，没到取数**