4.4.3 全 Cache 技术

全 Cache 技术是最近出现的组织形式，尚不成熟，还未商品化。它没有主存，只用 Cache 与辅存中的一部分构成“Cache一辅存”存储系统。

全 Cache 存储系统的等效访问时间要接近于 Cache 的，容量是虚地址空间的容量。图 4-41 是在多处理机中实现的一种方案。

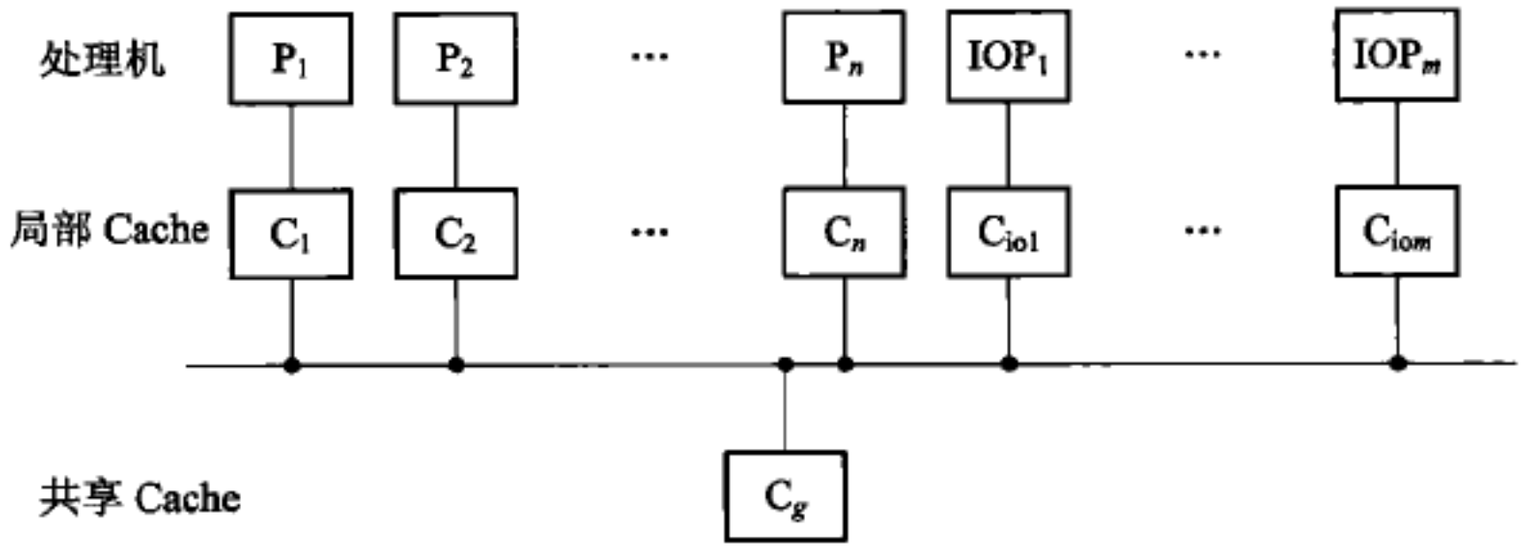


图 4-41 多处理机中的全 Cache 存储系统

由于磁盘辅存基本访问单位是物理块，每块有 512 B，因此，与磁盘存储器连接的局部 Cache 的块容量一般也应是 512 B，其他 Cache 块的容量可以是小于或大于 512 B，但应该是 512 B 的整数倍。

4.5 存储系统的保护

大多数计算机系统都设计成让其资源能被共行的多个用户所共享，就主存来说，就同时存有多个用户的程序和系统软件。为使系统能正常工作，应防止由于一个用户程序出错而破坏主存中其他用户的程序或系统软件，还要防止一个用户程序不合法地访问未分配给它的主存区域，哪怕这种访问不会引起破坏。因此，系统结构需要为主存的使用提供存储保护，它是多道程序和多处理机系统必不可少的。

首先介绍存储区域的保护，然后介绍访问方式的保护。

为实现区域保护，对于不是虚拟存储器的主存系统可采用第 2 章讲过的界限寄存器方式，由系统软件经特权指令置定上、下界寄存器，从而划定每个用户程序的区域，禁止它越界访问。由于用户程序不能改变上、下界的值，因此不论它如何出错，也只能破坏该用户自身的程序，侵犯不到别的用户程序及系统软件。

然而，界限寄存器方式只适用于每个用户程序占用主存一个或几个(当有多对上、下界寄存器时)连续的区域；而对于虚拟存储器系统，由于一个用户的各页能离散地分布于主存内，从而无法使用这种保护方式。对虚拟存储器的主存区域保护就需采用页表保护和键式保护等方式。

(1) 页表保护。每个程序有它自己的页表，其行数等于该程序的虚页数。例如它有 4 页，则只能有 0、1、2、3 这 4 个虚页号。设由操作系统建立的程序页表，这 4 个虚页号分别对应于实页号 4、8、10、14，则不论虚地址如何出错，只能影响主存中分配给该程序的第 4、8、10、14 号实页。假设虚页号错成“5”，肯定不可能在该程序的页表中找到，也就访问

不了主存，当然也就不会影响主存中其他程序的区域。这正是虚拟存储器系统本身固有的保护机能，也是它的一大优点。为了更便于实现这种保护，还可在段表中的每行内不仅设置页表起点，还设置段长(页数)项。若出现该段内的虚页号大于段长，则可发越界中断。

这种页表保护是在没形成主存实地址前进行的保护，使之无法形成侵犯别的程序区域的主存地址。然而，若地址形成环节由于软、硬件方面的故障而形成了不属于本程序区域的错误主存地址，则上述这种保护就无能为力了。因此，还应采取进一步的保护措施，键方式是其中成功的一种。

(2) 键式保护。键式保护由操作系统按当时主存的使用分配状况给主存的每页配一个键，称为存储键，它相当于一把“锁”。所有页的存储键存在主存相应的快速寄存器内，每个用户(任务)的各实页的存储键都相同。为了打开这把锁，需要有把“钥匙”，称为访问键。每个用户的访问键由操作系统给定，存在处理机的程序状态字(PSW)或控制寄存器中。程序每次访问主存前，要核对主存地址所在页的存储键是否与该程序的访问键相符，只有相符，才准访问。这样，就是错误地形成了侵犯别的程序的主存地址，也因为这种键保护而仍然不允许访问。例如保护键设成 4 位，就可以表示已调入主存的 16 个活跃的程序。其中“0000”访问键是操作系统的，对这个访问键不论是否和存储键相符都可访问，这是操作系统应能访问到主存整个区域所要求的。

上面讲的是保护别的程序区域不被侵犯，但不是保护正在执行的程序本身不被破坏。如何实现对正在执行的程序的重要关键部分的保护也是存储保护的一部分，可以有各种办法，环式保护是其中的一种。

这种保护把系统程序和用户程序按其重要性及对整个系统能否正常工作的影响程度分层，如图 4 - 42 所示。设 0、1、2 三层是系统程序的，之外的各层是同一用户程序的分层。环号大小表示保护的级别，环号愈大，等级愈低。在现行程序运行前，先由操作系统定好程序各页的环号，并置入页表。而后把该道程序的开始环号送入处理机内的现行环号寄存器，并且把操作系统规定给该程序的上限环号(规定该程序所能进入的最内层环号)也置入相应的寄存器。

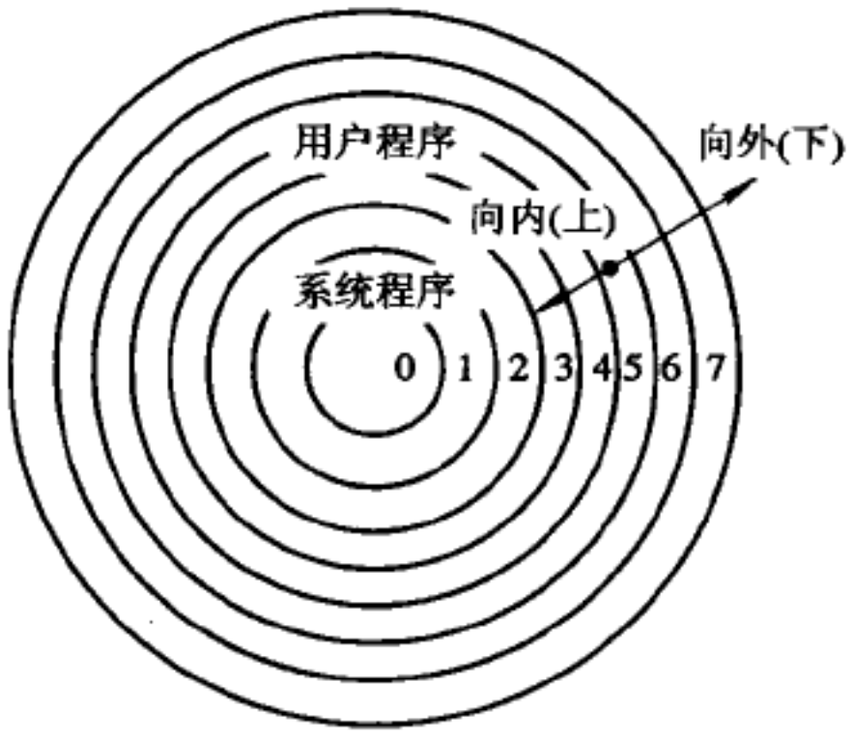


图 4 - 42　环式保护的分层

若 P_i 是在某一个时候属 i 层各页的集合，则当进程执行 $P \in P_i$ 页内的程序时，允许访问 $F \in P_j$ 页，这里对应的是 $j \geqslant i$。但是如果是 $j < i$，则需由操作系统环控制例行程序判定这个内向访问是否允许和是否正确之后才能访问，否则就是出错，进入保护处理。但 j 值

肯定不能小于给定的上限环号。只要 $j \neq i$，就进入中断，若允许访问，则需经特权指令把现行环号寄存器的值由 i 改为 j。

这种环式保护既能保证由于用户程序的出错不至侵犯系统程序，也能保证由于同一用户程序内的低级(环号大)部分的出错而不至破坏其高级(环号小)部分。这种环式保护对系统程序的研究和调试运行特别有利，因为可以做到能修改系统程序的某些部分而不必担心会影响到系统程序已设计好并调好的核心部分。

至于如何控制 $j \neq i$ 的跨层访问，有的机器规定只能由规定的转移指令执行，且对 $j > i$ 和 $j < i$ 分别只能用不同的转移指令。

上述种种区域保护，如判越界、判键相符、判环号相符、判不超出段长等，当然都是经硬件实现的，因此速度可以是很快的。

区域保护是指对允许访问的区域可以进行任何形式的访问，而在允许区域之外，则任何形式的访问都不允许。但在实际中，只有这种限制往往适应不了各种应用的要求，因此还得加上访问方式的限制(保护)。

对主存信息的使用可以有读(R)、写(W)和执行(E)三种方式。“执行”指的是作为指令来用。相应地就有 R、W 和 E 访问方式保护。这三者的逻辑组合可以反映出各种应用要求。例如：

$\overline{R \vee W \vee E}$——不允许进行任何访问(如专用的系统表格)；

$R \vee W \vee E$——可以进行任何访问；

$R \wedge \overline{W \vee E}$——只能进行读访问(如对各个用户都用到的表格常数)；

$(R \vee W) \wedge \overline{E}$——只能按数据进行读、写(例如阵列数据当然不能作为指令执行)；

$\overline{R \vee W} \wedge E$——只能执行，不能作为数据使用(如某个专门的程序)；

$\overline{R \vee E} \wedge W$——只能进行写访问(如用户对操作系统缓冲器的写入)；

$(R \vee E) \wedge \overline{W}$——不准写访问。

对前面讲过的各种区域保护，都可以加上相应的访问方式位以实现这种访问限制。

例如，对界限寄存器方式，可以在下界寄存器加一位访问方式位。它为“0”，表明该区域可读、可写；它为“1”时，表明只能读、不准写。又如，对键方式，也可以加上访问方式位。有时，它的作用还需和访问键与存储键是否相符一起来考虑。如有的机器就设计成当“读”保护位为“0”时，不论是否键相符，恒可进行读访问；当“读”保护位为“1”时，只有键相符时才能进行读访问；但对写访问均只有键相符时才能进行。这里，对于写访问因保护而被禁止时，被保护页的内容保持不变；对于读访问因保护而被禁止时，被保护页的内容既不能取到寄存器，也不传送到其他页或 I/O 设备。显然，这样一来，只需使读保护位为“0”，就可实现主存对只准读的部分的共享。

至于环式保护和页表保护，可以把 R、W、E 等访问方式位设在各个程序的段、页表的各行内，使得同一环内或同一段内的各页可以有上述种种不同的访问保护，以增强灵活性。而且还可以做到各个用户对同一共享页的访问方式不同。例如有的用户只能读访问，而有的用户则是读、写访问都允许。对于环式保护，还可做到使访问保护不仅取决于 R、W、E 的组合，还取决于 j 与 i 相比是大、是小、还是相等。

上述通过访问方式位的组合以及键相符、环号比较等，来决定允许进行何种访问也是由硬件实现的。

除了以上这些保护外，在某些应用中，我们既要求能实现多个用户可读、写访问共享的数据，又要保证只当一个用户访问完该数据后，别的用户才可访问，以防止在一个用户还未把某个共享文件写好之前，别的用户却能把它读了去。利用第 2 章讲过的“测试与置定”和“比较与交换”指令能实现这点。所以这也是一种保护的办法。

4.6 本章小结

4.6.1 知识点和能力层次要求

(1) 识记存储体系的两个分支、构成依据和性能参数。

(2) 领会段式、页式、段页式三种不同的虚拟存储管理方式的工作原理，其地址映像规则、映像表机构、虚实地址变换的过程要达到综合应用层次。识记三种虚存管理方式各自的优点和问题。

(3) 给出段页式虚拟存储器的虚实地址映像表的内容，能由程序虚地址判断出是否发生段失效、页失效或保护失效。如果没有段失效、页失效，则要能够计算出主存的实地址。熟悉页式虚拟存储器的虚、实地址字段对应关系和地址映像规则，会由虚地址查映像表判断出是否发生页失效；如果无页失效，则应会计算出实主存的实地址。以上内容均须达到综合应用层次。

(4) 熟练掌握在页式虚拟存储器中，通过给出分配给程序的实页数、程序页地址流，分别采用 FIFO、LRU、OPT 法，模拟页面替换时的程序页面装入和替换过程、LRU 替换算法的堆栈模拟过程，计算不同实页数时各自的命中率。以上内容要达到综合应用层次。领会替换算法属堆栈型的定义，以及 PFF 替换算法的思想。给出各道程序的页地址流，经 LRU 堆栈模拟后，如何分配给各道程序的实主存页数，使系统效率达到最高，对此要达到综合应用层次。

(5) 领会在虚拟存储器中对页面失效的处理及内部地址映像表中的快慢表机构的作用，页面大小 S_p、分配给程序的主存容量 S_1 与主存命中率 H 的变化关系，改进虚拟存储器等效访问速度的办法。

(6) 领会 Cache 存储器的组成、工作原理，并能与虚拟存储器进行对比。对于 Cache 存储器中的全相联、直接、组相联三种地址映像规则，相应的映像表机构和虚、实地址变换过程要达到综合应用层次。领会比较对法实现 Cache 块替换的机构和原理。对于用比较对法进行块替换时所用的比较对触发器个数要达到简单应用层次。

(7) 给出主存的块地址流，采用组相联或直接映像、LRU 或 FIFO 替换算法时，画出各主存块装入 Cache 和其被替换的过程示意图，并计算出 Cache 块的命中率，对此要达到综合应用层次。

(8) 领会为解决 Cache 存储器透明性问题所提出的各种算法以及为提高 Cache 块命中率的各种预取算法。能分析影响 Cache 性能的因素及其变化规律。正确理解和分析 Cache 存储器的等效访问速度与物理 Cache 速度及 Cache 容量、Cache 命中率等的关系。

(9) 一般了解存储系统中使用的各种保护方式。

4.6.2 重点和难点

1. 重点

段页式和页式虚拟存储器的原理；页式虚拟存储器的地址映像，用 LRU、FIFO、OPT 替换算法进行页面替换的过程模拟；用 LRU 替换算法对页地址流的堆栈处理模拟及性能分析；Cache 存储器的直接映像和组相联地址映像；用 LRU 替换算法进行块替换的硬件实现及替换过程模拟；Cache 存储器的性能分析等。

2. 难点

页式和段页式虚拟存储器中，虚、实地址的计算；各种页面替换算法的模拟和页命中率的计算；Cache 组相联映像和块替换算法的模拟。

习 题 4

4－1 设二级虚拟存储器的 $T_{A_1}=10^{-7}$ s、$T_{A_2}=10^{-2}$ s，为使存储层次的访问效率 e 达到最大值的 80%以上，命中率 H 至少要求达到多少？实际上这样高的命中率是很难达到的，那么从存储层次上如何改进？

4－2 有一个二级虚拟存储器，CPU 访问主存 M_1 和辅存 M_2 的平均时间分别为 1 μs 和 1 ms。经实测，此虚拟存储器平均访问时间为 100 μs。试定性提出使虚拟存储器平均访问时间能降到 10 μs 的几种方法，并分析这些方法在硬件和软件上的代价。

4－3 由 4.1.3 节所述二级存储层次的每位价格 c 和访问时间 T_A 的表达式，导出 n 级存储层次相应的表达式。

4－4 某虚拟存储器共 8 个页面，每页为 1024 个字，实际主存为 4096 个字，采用页表法进行地址映像。映像表的内容如表 4－3 所示。

表 4－3 映像表的内容

实 页 号	装 入 位
3	1
1	1
2	0
3	0
2	1
1	0
0	1
0	0

(1) 列出会发生页面失效的全部虚页号；

(2) 按以下虚地址计算主存实地址：0，3728，1023，1024，2055，7800，4096，6800。

4－5 有一个段页式虚拟存储器，虚地址有 2 位段号、2 位页号、11 位页内位移(按字编址)，主存容量为 32 K 字。每段可有访问方式保护，其页表和保护位如表 4－4 所示。

表 4－4　页表和保护位

段　号	段 0	段 1	段 2	段 3
访问方式	只读	可读/执行	可读/写/执行	可读/写
虚页 0 所在位置	实页 9	在辅存上	页表不在主存内	实页 14
虚页 1 所在位置	实页 3	实页 0		实页 1
虚页 2 所在位置	在辅存上	实页 15		实页 6
虚页 3 所在位置	实页 12	实页 8		在辅存上

（1）此地址空间中共有多少个虚页？

（2）当程序中遇到如表 4－5 所示的各种情况时，写出由虚地址计算出的实地址。说明哪个会发生段失效、页失效或保护失效。

表 4－5　程序的各种操作情况

方　式	段	页	页内位移
取数	0	1	1
取数	1	1	10
取数	3	3	2047
存数	0	1	4
存数	2	1	2
存数	1	0	14
转移至此	1	3	100
取数	0	2	50
取数	2	0	5
转移至此	3	0	60

4－6　设某程序包含 5 个虚页，其页地址流为 4，5，3，2，5，1，3，2，2，5，1，3。当使用 LRU 算法替换时，为获得最高的命中率，至少应分配给该程序几个实页？其可能的最高命中率为多少？

4－7　有一个虚拟存储器，主存有 0～3 四页位置，程序有 0～7 八个虚页，采用全相联映像和 FIFO 替换算法。给出如下程序页地址流：2，3，5，2，4，0，1，2，4，6。

（1）假设程序的 2，3，5 页已先后装入主存的第 3，2，0 页位置，请画出上述页地址流工作过程中，主存各页位置上所装程序各页页号的变化过程图，标出命中时刻。

（2）求出此期间虚存总的命中率 H。

4－8　采用 LRU 替换算法的页式虚拟存储器共有 9 页空间准备分配给 A、B 两道程序。已知 B 道程序若给其分配 4 页，则命中率为 8/15；而若分配 5 页，则命中率可达 10/15。现给出 A 道程序的页地址流为 2，3，2，1，5，2，4，5，3，2，5，2，1，4，5。

（1）画出用堆栈对 A 道程序页地址流的模拟处理过程图，统计给其分配 4 页和 5 页时的命中率。

（2）根据已知条件和上述统计结果，给 A、B 两道程序各分配多少实页，可使系统效率最高？

4－9　采用页式管理的虚拟存储器，分时运行两道程序。其中，程序 X 为

```
      DO 50 I=1,3
      B(I)=A(I)-C(I)
      IF(B(I) .LE. 0) GOTO 40
      D(I)=2 * C(I)-A(I)
      IF(D(I) .EQ. 0) GOTO 50
40    E(I)=0
50    CONTINUE
```

Data：A=(−4，+2，0)

C=(−3，0，+1)

每个数组分别放在不同的页面中；而程序 Y 在运行过程中，其数组将依次用到程序空间的第 3，5，4，2，5，3，1，3，2，5，1，3，1，5，2 页。如果采用 LRU 算法替换，实存却只有 8 页位置可供数组之用。试问为这两道程序的数组分别分配多少个实页最为合理？为什么？

4－10　设一个按位编址的虚拟存储器，它可对应 1 K 个任务，但在一段较长时间内，一般只有 4 个任务在使用，故用容量为 4 行的相联寄存器组硬件来缩短被变换的虚地址中的用户位位数；每个任务的程序空间可达 4096 页，每页为 512 个字节，实主存容量为 2^{20} 位；设快表用按地址访问存储器构成，行数为 32，快表的地址经散列形成；为减少散列冲突，配有两套独立的相等比较器电路。请设计该地址变换机构，内容包括：

(1) 画出其虚、实地址经快表变换之逻辑结构示意图。

(2) 相联寄存器组中每个寄存器的相联比较位数。

(3) 相联寄存器组中每个寄存器的总位数。

(4) 散列变换硬件的输入位数和输出位数。

(5) 每个相等比较器的位数。

(6) 快表的总容量(以位为单位)。

4－11　考虑一个 920 个字的程序，其访问虚存的地址流为 20，22，208，214，146，618，370，490，492，868，916，728。

(1) 若页面大小为 200 字，主存容量为 400 字，采用 FIFO 替换算法，请按访存的各个时刻，写出其虚页地址流，计算主存的命中率。

(2) 若页面大小改为 100 字，再做一遍。

(3) 若页面大小改为 400 字，再做一遍。

(4) 由(1)、(2)、(3)的结果可得出什么结论？

(5) 若把主存容量增加到 800 字，按第(1)小题再做一遍，又可以得到什么结论？

4－12　在一个页式二级虚拟存储器中，采用 FIFO 算法进行页面替换，发现命中率 H 太低，因此有下列建议：

(1) 增大辅存容量；

(2) 增大主存容量(页数)；

(3) 增大主、辅存的页面大小；

(4) FIFO 改为 LRU；

(5) FIFO 改为 LRU，并增大主存容量(页数)；

(6) FIFO 改为 LRU，且增大页面大小。

试分析上述各建议对命中率的影响情况。

4－13　有采用组相联映像的 Cache 存储器，Cache 大小为 1 KB，要求 Cache 的每一块在一个主存周期内能从主存取得。主存模 4 交叉，每个分体宽为 32 位，总容量为 256 KB。用按地址访问存储器构成相联目录表实现主存地址到 Cache 地址的变换，并约定用 4 个外相等比较电路。请设计此相联目录表，求出该表之行数、总位数及每个比较电路的位数。

4－14　有一个 Cache 存储器。主存共分 8 个块(0～7)，Cache 为 4 个块(0～3)，采用组相联映像，组内块数为 2 块，替换算法为近期最久未用过算法(LRU)。

(1) 画出主存、Cache 地址的各字段对应关系(标出位数)图。

(2) 画出主存、Cache 空间块的映像对应关系示意图。

(3) 对于如下主存块地址流：1，2，4，1，3，7，0，1，2，5，4，6，4，7，2，如主存中内容一开始未装入 Cache 中，请列出 Cache 中各块随时间的使用状况。

(4) 对于(3)，指出块失效又发生块争用的时刻。

(5) 对于(3)，求出此期间 Cache 之命中率。

4－15　有 Cache 存储器。主存有 0～7 共 8 块，Cache 有 4 块，采用组相联映像，分 2 组。假设 Cache 已先后访问并预取进了主存的第 5，1，3，7 块，现访存块地址流又为 1，2，4，1，3，7，0，1，2，5，4，6 时：

(1) 若使用 LRU 替换算法，画出 Cache 内各块的实际替换过程图，并标出命中时刻。

(2) 求出在此期间 Cache 的命中率。

4－16　有 Cache 存储器。主存有 0～7 共 8 块，Cache 为 4 块，采用组相联映像，设 Cache 已先后预取进了主存的第 1，5，3，7 块，现访存块地址流又为 1，2，4，1，3，7，0，1，2，5，4，6 时，在 Cache 分 2 组的条件下，

(1) 当使用 FIFO 替换算法时，画出 Cache 内各块的实际替换过程图，并标出命中的时刻。

(2) 求出在此期间 Cache 的命中率。

4－17　采用组相联映像、LRU 替换算法的 Cache 存储器，发现等效访问速度不高，为此提议：

(1) 增大主存容量。

(2) 增大 Cache 中的块数(块的大小不变)。

(3) 增大组相联组的大小(块的大小不变)。

(4) 增大块的大小(组的大小和 Cache 总容量不变)。

(5) 提高 Cache 本身器件的访问速度。

分别采用上述措施后，等效访问速度可能会有什么样的显著变化？其变化趋势如何？如果采取措施后并未能使等效访问速度有明显提高的话，又是什么原因？

4－18　用组相联映像的 Cache 存储器，块的大小为 2^8 个单元，主存容量是 Cache 容量的 4 倍。映像表用单体多字按地址访问存储器构成，已装入的内容如表 4－6 所示。用 4 套外比较电路实现组内相联查找块号。各字段用四进制编码表示。

表 4－6　题 4－18 的映像表

组号 q	n_d	s'	s	n_d	s'	s	n_d	s'	s	n_d	s'	s
0	0	0	1	2	0	0	2	2	2	3	2	3
1	0	1	3	0	3	2	3	3	1	0	0	0
2	3	3	0	3	2	1	3	1	2	1	1	3
3	0	2	2	3	1	3	2	1	1	2	0	0

(1) 给出以四进制码表示的主存地址 3122203，问主存该单元的内容能否在 Cache 中找到，若能找到，指出相应的 Cache 地址，并用四进制码表示。

(2) 给出四进制码表示的主存地址 1210000 及 2310333，回答(1)中的问题。

4－19　假设你对 Cache 存储器的速度不满，于是申请到一批有限的经费，为能发挥其最大经济效益，有人建议你再买一些同样速度的 Cache 片子以扩充其容量；而另有人建议你干脆去买更高速的 Cache 片子将现有低速 Cache 片子全部换掉。你认为哪种建议可取？你如何做决定？为什么？

第5章　标量处理机

加快标量处理机的机器语言的解释是计算机组成设计的基本任务，可从两方面实现。一是通过选用更高速的器件，采取更好的运算方法，提高指令内各微操作的并行程度，减少解释过程所需要的拍数等措施来加快每条机器指令的解释。二是通过控制机构同时解释两条、多条以至整段程序的方式来加快整个机器语言程序的解释。重叠和流水是其中常用的方式。本章主要讲述这两种方式的基本原理、实现中要解决的问题和办法以及性能分析，最后讲述指令级高度并行的超标量、超长指令字、超流水和超标量超流水线处理机。

5.1　重叠方式

重叠方式是最简单的流水方式。

5.1.1　重叠原理与一次重叠

指令的重叠解释使机器语言程序的执行速度会比采用顺序解释有较大的提高。顺序解释指的是各条指令之间顺序串行(执行完一条指令后才取下条指令)地进行，每条指令内部的各个微操作也顺序串行地进行。

解释一条机器指令的微操作可归并成取指令、分析和执行三部分，时间关系如图5－1所示。取指令是按指令计数器的内容访主存，取出该指令送到指令寄存器。指令的分析是对指令的操作码进行译码，按寻址方式和地址字段形成操作数真地址，并用此真地址去取操作数(可能访主存，也可能访寄存器)，为取下一条指令还要形成下一条指令的地址。指令的执行则是对操作数进行运算、处理，或存储运算结果(可能要访主存)。这样，图5－2(a)表示出了指令的顺序解释方式。

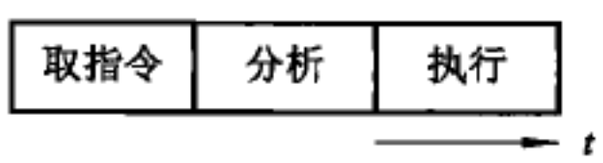


图5－1　对一条机器指令的解释

顺序解释的优点是控制简单，转入下条指令的时间易于控制。但缺点是上一步操作未完成，下一步操作便不能开始，速度上不去，机器各部件的利用率低。例如，取指令和取操作数时，主存忙着但运算器却闲着；在对操作数执行运算时，运算器忙着但主存却闲着。于是提出了重叠解释，即让不同指令在时间上重叠地解释。

指令的重叠解释是在解释第 k 条指令的操作完成之前，就可开始解释第 $k+1$ 条指令。

图 5 - 2(b)是可能的一种方式。显然，重叠解释虽不能加快一条指令的解释，却能加快相邻两条指令以至整段程序的解释。

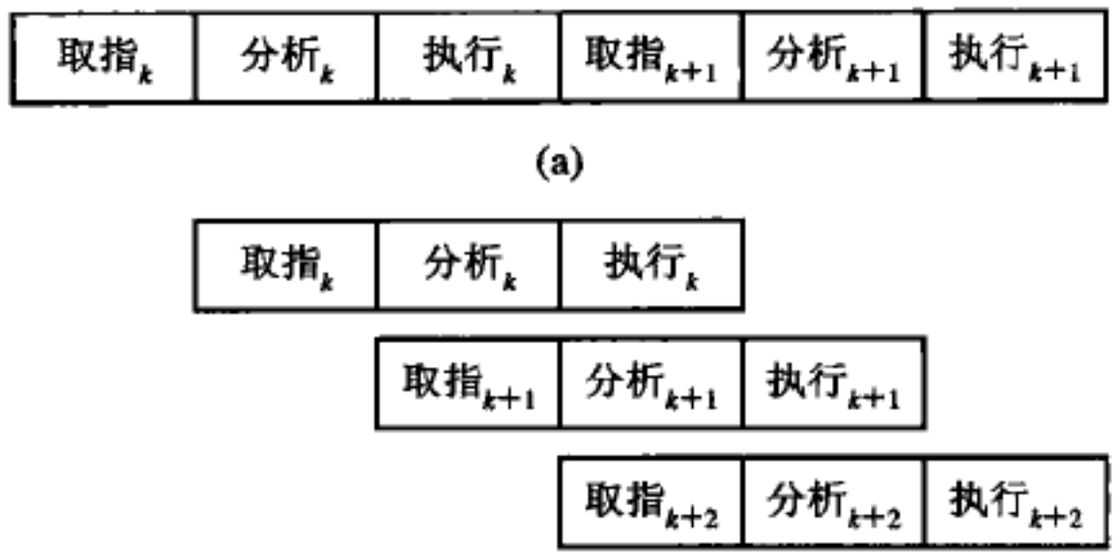


图 5 - 2　指令的顺序解释与重叠解释

(a) 顺序解释；(b) 重叠解释的一种方式

实现指令的重叠解释必须在计算机组成上满足以下几点要求：

(1) 要解决访主存的冲突。从图 5 - 2(b)上看，需要让“取指$_{k+1}$”与“分析$_k$”在时间上重叠，而取指要访存，分析中取操作数也可能要访存。在一般的机器上，操作数和指令混合存储于同一主存内，而且主存同时只能访问一个存储单元。如果不在硬件上花费一定的代价解决好访主存的冲突，就无法实现“取指$_{k+1}$”与“分析$_k$”的重叠。为此，一种办法是让操作数和指令分别存放于两个独立编址且可同时访问的存储器中，这有利于实现指令的保护，但是增加了主存总线控制的复杂性及软件设计的麻烦。另一种办法是仍维持指令和操作数混存，但采用多体交叉主存结构，只要第 k 条指令的操作数与第 $k+1$ 条指令不在同一个体内，仍可在一个主存周期取得，从而实现“分析$_k$”与“取指$_{k+1}$”重叠。然而，这两者若正好共存于一个体内时就无法重叠。因此，仅靠这种解决办法有一定的局限性。第三种办法是增设采用先进先出方式工作的指令缓冲寄存器(简称指缓)。由于大量中间结果只存于通用寄存器中，因此主存并不是满负荷工作的。设置指缓就可趁主存有空时，预取下一条或下几条指令存于指缓中。最多可预取多少条指令取决于指缓的容量。这样，“分析$_k$”与“取指$_{k+1}$”就能重叠了，因为只是前者需访主存取操作数，而后者是从指缓取第 $k+1$ 条指令。如果每次都可以从指缓中取得指令，则“取指$_{k+1}$”的时间很短，就可把这个微操作合并到“分析$_{k+1}$”内，从而由原先的“取指$_{k+2}$”、“分析$_{k+1}$”、“执行$_k$”重叠变成只是“分析$_{k+1}$”与“执行$_k$”的重叠，如图 5 - 3 所示。

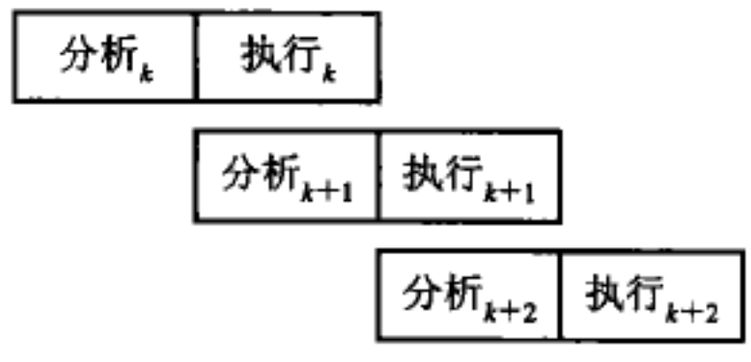


图 5 - 3　一次重叠工作方式

(2) 要解决“分析”与“执行”操作的并行。为了实现“执行$_k$”与“分析$_{k+1}$”重叠，硬件上还应有独立的指令分析部件和指令执行部件。以加法器为例，分析部件要有单独的地址加法器用于地址计算，执行部件也要有单独的加法器完成操作数的相加运算。这是以增加某些硬件为代价的。

(3) 要解决“分析”与“执行”操作控制上的同步。实际上“分析”和“执行”所需的时间常不相同，还需在硬件中解决控制上的同步，保证任何时候都只是“执行$_k$”与“分析$_{k+1}$”重叠。

就是说，即使“分析$_{k+1}$”比“执行$_k$”提前结束，“执行$_{k+1}$”也不紧接在“分析$_{k+1}$”之后与“执行$_k$”重叠进行；同样，即使“执行$_k$”比“分析$_{k+1}$”提前结束，“分析$_{k+2}$”也不紧接在“执行$_k$”之后与“分析$_{k+1}$”重叠进行。称这种指令分析部件和指令执行部件任何时候都只有相邻两条指令在重叠解释的方式为“一次重叠”。“一次重叠”解释的好处是节省硬件，机器内指令分析部件和指令执行部件均只需一套，也简化了控制。设计时应适当安排好微操作，使“分析”和“执行”时间尽量等长，重叠方式才能有较高的效率。如果采用多次重叠，不仅要设多套指令分析和执行部件，控制也相当复杂。所以，重叠方式的机器大多数都采用一次重叠，若仍达不到速度要求，宁可采用本章后面要介绍的流水方式。

（4）要解决指令间各种相关的处理。例如，第 k 条指令是按其执行结果进行转移的条件转移指令，并成功转移到 m 单元，则与“执行$_k$”重叠的“分析$_{k+1}$”需要撤销并从头分析第 m 条指令。若第 m 条指令还没有取到指缓，则在“执行$_k$”之后还需先进行“取指$_m$”才能“分析$_m$”。图 5－4 示意出条件转移时第 k 条指令和第 $k+1$ 条指令的时间关系。可见条件转移成功时，重叠实际变成了顺序。

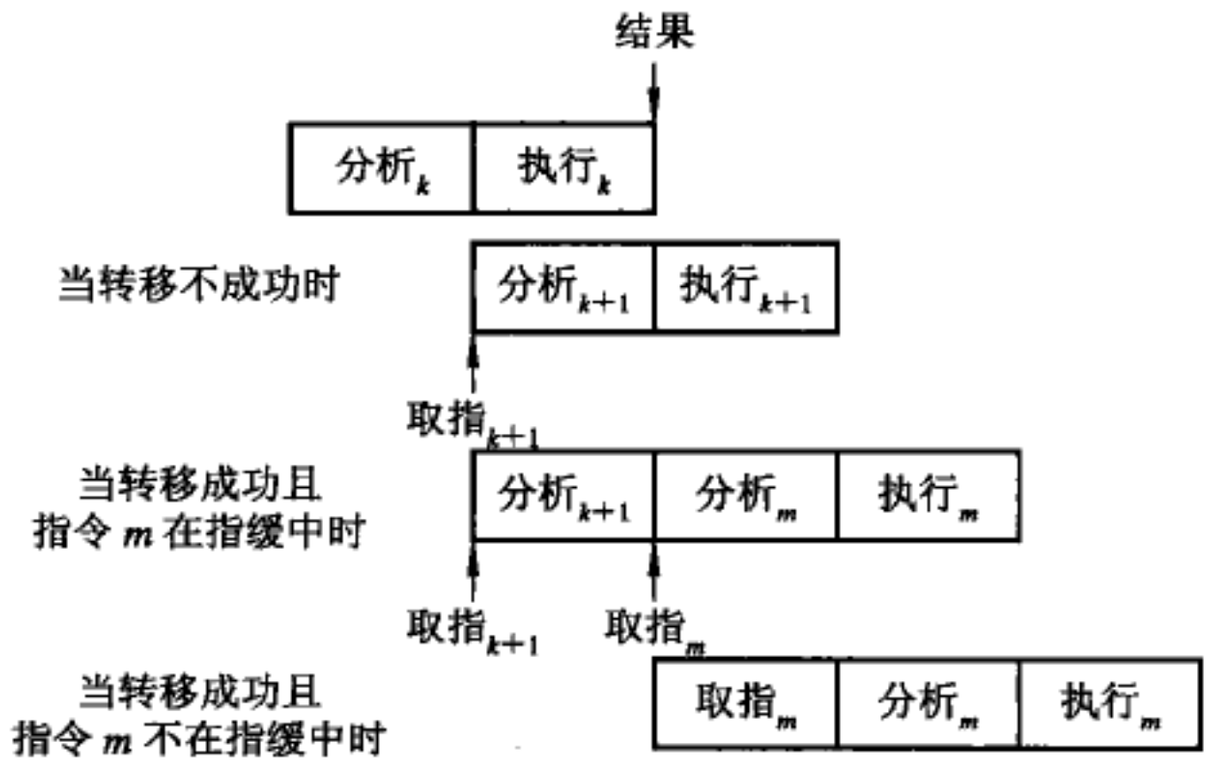


图 5－4　第 k 条指令和第 $k+1$ 条指令的时间关系

控制上还要解决好邻近指令之间有可能出现的某种关联。例如，第 $k+1$ 条指令的源操作数地址 i 正好是第 k 条指令存放运算结果的地址，在进行顺序解释时，由于先由第 k 条指令把运算结果存进主存 i 单元，而后再由第 $k+1$ 条指令从 i 单元取出，当然不会出错；但在“分析$_{k+1}$”与“执行$_k$”重叠解释时，“分析$_{k+1}$”从 i 单元取出的源操作数内容成了“执行$_k$”存进运算结果前的原存内容，并不是第 k 条指令应有的运算结果，这必然要出错。这种因机器语言程序中邻近指令之间出现了关联，为防出错让它们不能同时解释的现象就称为发生了“相关”。上面的例子是在第 k、$k+1$ 条指令的数据地址之间有了关联，称为发生了“数相关”。数相关不只是会发生在主存空间，还会发生在通用寄存器空间。下面将进一步叙述。

既然有“数相关”，就可能还会有“指令相关”。如果采用机器指令可修改的办法经第 k 条指令的执行来形成第 $k+1$ 条指令，如

k：存　通用寄存器，$k+1$；(通用寄存器)→($k+1$)

$k+1$：……

由于在“执行$_k$”的末尾才形成第 $k+1$ 条指令，按照一次重叠的时间关系，“分析$_{k+1}$”所分析

的是早已取进指缓的第 $k+1$ 条指令的旧内容，因此就会出错。为了避免出错，第 k、$k+1$ 条指令就不能同时解释，我们称此时这两条指令之间发生了“指令相关”。当指缓可缓冲存入 n 条指令时，第 k 条指令与已预取进指缓的第 $k+1$ 到 $k+n$ 条指令都有可能发生指令相关。指缓容量越大，或者说指令预处理能力越强的机器发生指令相关的概率就会越高。

无论发生何种相关，或者会使解释出错，或者会使重叠效率显著下降，所以必须加以正确处理。下面就来讲述相关处理的一些办法。

5.1.2 相关处理

1. 转移指令的处理

前面已提到，当程序中遇到条件转移时，一旦条件转移成功，重叠解释实际变成了顺序解释。在第 4 章并行主存系统中已提到，标量计算机中的条件转移概率可达 10%～30%。这样，重叠效率将显著下降。因此，重叠方式的机器在程序中应尽量减少使用条件转移指令。如果要用条件转移指令，可采用 3.3.3 节中介绍过的延迟转移技术，即由编译程序生成目标程序时，将转移指令与条件转移无关的第 $k-1$ 条指令交换一下位置，这样，即使条件转移成功也不会使重叠效率下降。当第 k 条指令是条件转移且转移成功时，传统做法与延迟转移做法的比较如图 5－5 所示。

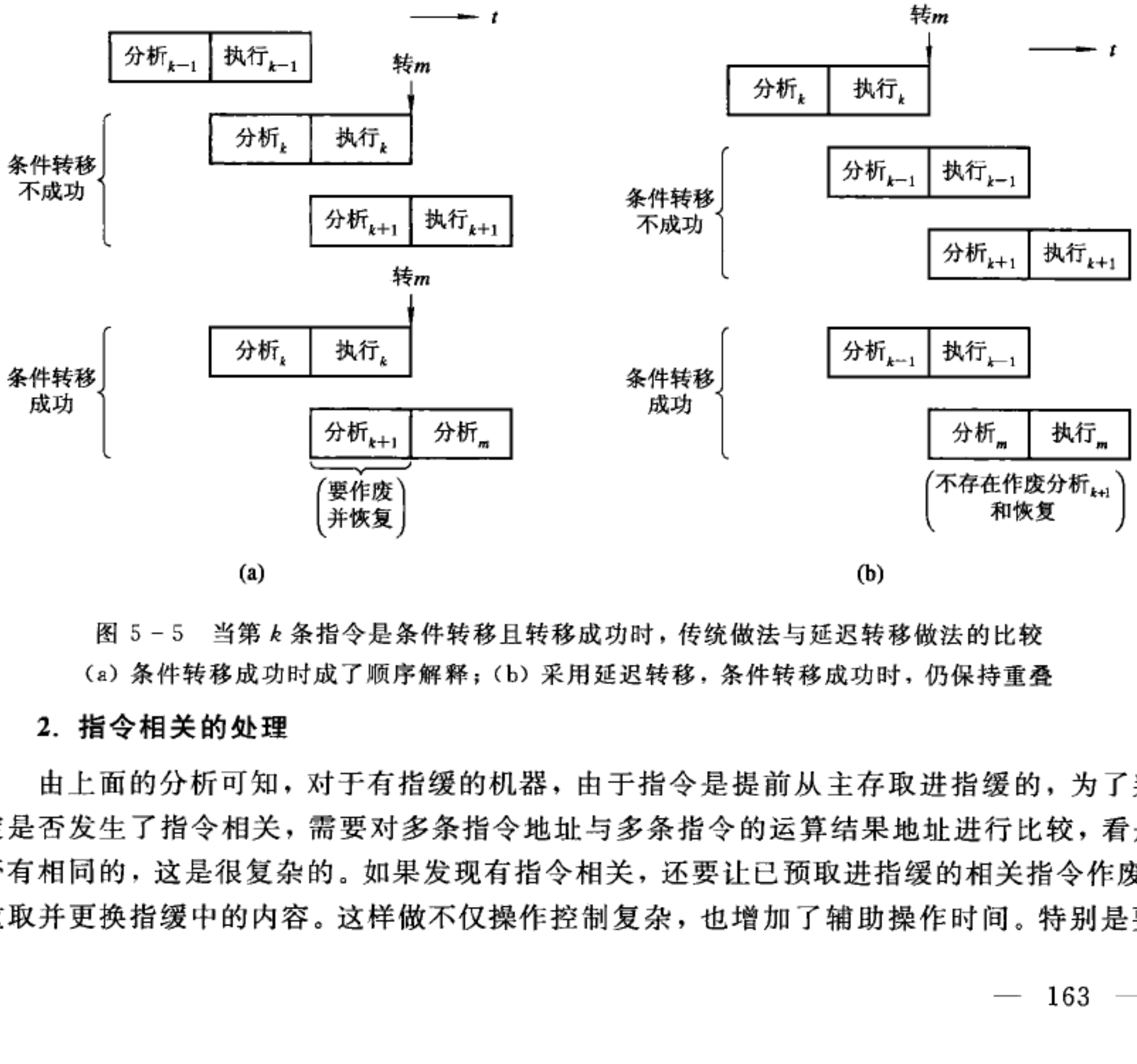


图 5－5　当第 k 条指令是条件转移且转移成功时，传统做法与延迟转移做法的比较

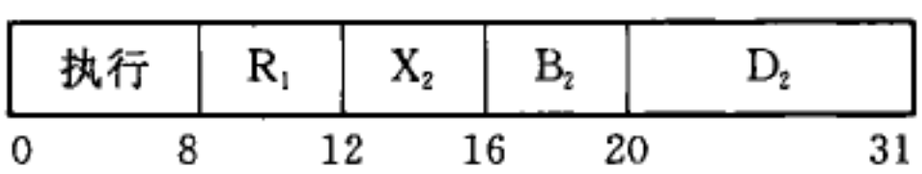
(a) 条件转移成功时成了顺序解释；(b) 采用延迟转移，条件转移成功时，仍保持重叠

2. 指令相关的处理

由上面的分析可知，对于有指缓的机器，由于指令是提前从主存取进指缓的，为了判定是否发生了指令相关，需要对多条指令地址与多条指令的运算结果地址进行比较，看是否有相同的，这是很复杂的。如果发现有指令相关，还要让已预取进指缓的相关指令作废，重取并更换指缓中的内容。这样做不仅操作控制复杂，也增加了辅助操作时间。特别是要

花一个主存周期去访存重新取指，带来的时间损失很大。

指令相关是因为机器指令允许修改而引出的。如果规定在程序运行过程中不准修改指令，指令相关就不可能发生。不准修改指令还可以实现程序的可再入和程序的递归调用。但是，为满足程序设计灵活性的需要，在程序运行过程中有时希望修改指令，这时可设置一条“执行”指令来解决。“执行”指令最初是在研制 IBM 370 机时专门设计的，其形式为



当执行到“执行”指令时，按第二操作数地址(X_2)+(B_2)+D_2 取出操作数区中单元的内容作为指令来执行，参见图 5－6。所执行的指令的第 8～15 位可与$(R_1)_{24\sim31}$位内容进行逻辑或，以进一步提高指令修改的灵活性。由于被修改的指令是以“执行”指令的操作数形式出现的，将指令相关转化成了数相关，因而只需统一按数相关处理即可。

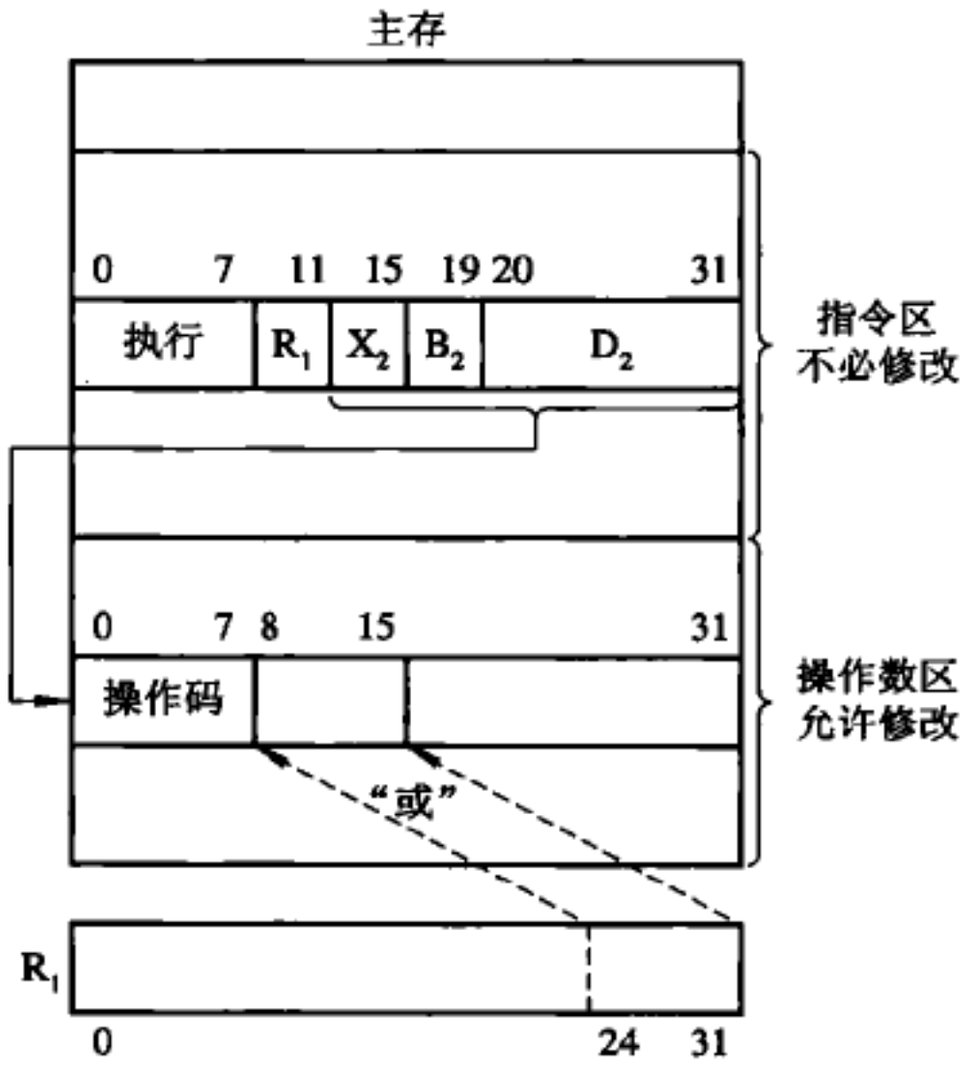


图 5－6　IBM 370“执行”指令的执行

3. 主存空间数相关的处理

主存空间数相关是相邻两条指令之间出现对主存同一单元要求先写而后读的关联，如图 5－7(a)所示。如果让“执行$_k$”与“分析$_{k+1}$”在时间上重叠，就会使“分析$_{k+1}$”读出的数不是第 k 条指令执行完应写入的结果，因而会出错。要想不出错，只有推后“分析$_{k+1}$”的读。

推后读常见的方法是由存控(存储器控制器)给读数、写数申请安排不同的访存优先级。因为中央处理机和通道都访存，中央处理机的访存可能是取指令、读数或写数，通道访存可能是读、写数据，取通道控制字，存通道状态字等。这些访存请求如果同时发出，就会出现访存冲突，因此需由存控在每个主存周期中对发生的各种访存申请排队，优先处理级别高的申请。只要在存控中将写数级别安排成高于读数级别，则当第 k 条和第 $k+1$ 条指令出现主存数相关时，存控就会先去处理“执行$_k$”的写数，而将“分析$_{k+1}$”的读申请推迟到下一个主存周期才处理，自动实现了推后“分析$_{k+1}$”。图 5－7(b)表示了此时的时间关系。

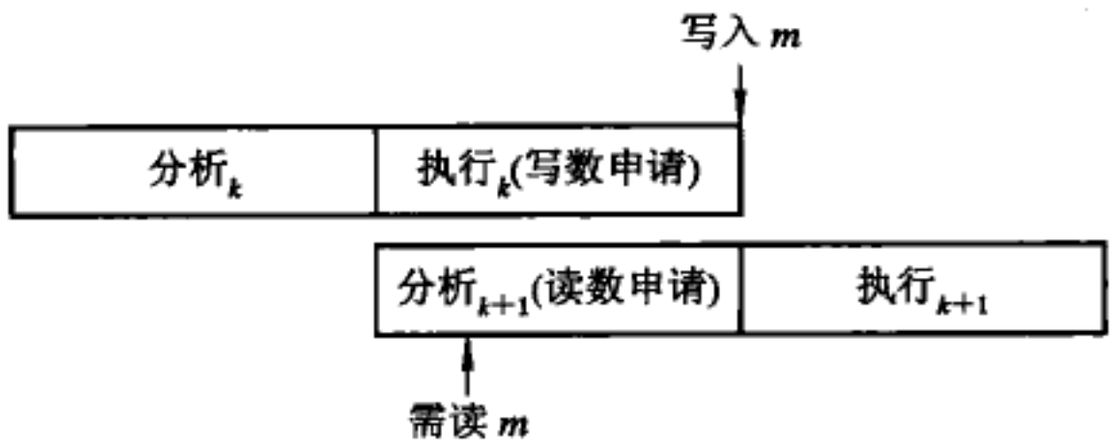


图 5－7　主存空间数相关的处理

（a）主存数相关的时间关系；（b）由存控推后“分析$_{k+1}$”的读

4. 通用寄存器组相关的处理

一般的机器中，通用寄存器除了存放源操作数、运算结果外，也可能存放形成访存操作数物理地址的变址值或基址值，因此，通用寄存器组的相关又有操作数的相关和变址值或基址值的相关两种。

假设机器的基本指令格式为

操作码	L_1	L_3	B_2	d_2

或

操作码	L_1	L_3		L_2

L_1、L_3 分别指明存放第一操作数和结果数的通用寄存器号，B_2 为形成第二操作数地址的基址值所在通用寄存器号，d_2 为相对位移量。在指令解释过程中，使用通用寄存器作不同用途所需微操作的时间是不同的。存放于通用寄存器中的基址值或变址值一般是在“分析”周期的前半段取用；操作数是在“分析”周期的后半段取出，到“执行”周期的前半段才用；运算结果是在“执行”周期末尾形成并存入通用寄存器中。图 5－8 示意出它们的时间关系。正因为时间关系不同，所以通用寄存器的数相关和基址值或变址值相关的处理方法不同。

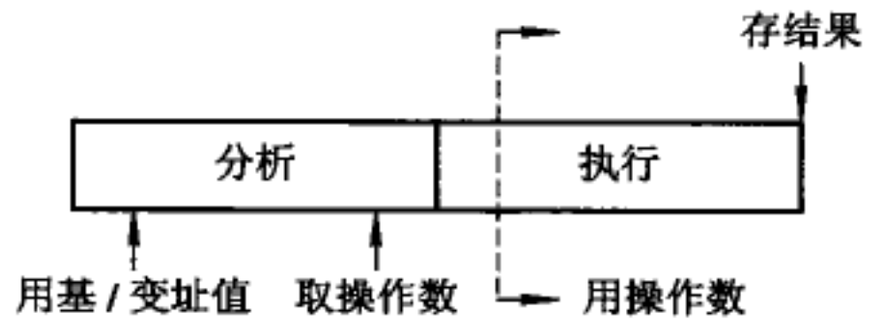


图 5－8　指令解释过程中与通用寄存器内容有关的微操作时间关系

先讨论通用寄存器组数相关的处理。

假设正常情况下，“分析”和“执行”的周期与主存周期一样都是 4 拍。有些指令需要从通用寄存器组中取两个操作数(L_1)和(L_2)，若通用寄存器组做在一个片子上，每次只能读出一个数，则在“分析$_{k+1}$”期间，操作数(L_1)和(L_2)就需要在不同拍时取得，分别送入运算器的 B 和 C 寄存器，以便在“执行$_{k+1}$”时供运算用。这样，“执行$_k$”与“分析$_{k+1}$”访问通用寄存器组的时间关系如图 5 - 9 所示。当程序执行过程中出现了 $L_{1(k+1)}=L_{3(k)}$ 时就发生了 L_1 相关，而当 $L_{2(k+1)}=L_{3(k)}$ 时就发生了 L_2 相关。一旦发生相关，如果仍维持让“执行$_k$”和“分析$_{k+1}$”一次重叠，从图 5 - 9 中的时间关系可以看出，“分析$_{k+1}$”取来的(L_1)或(L_2)并不是“执行$_k$”的真正结果，从而会出错。

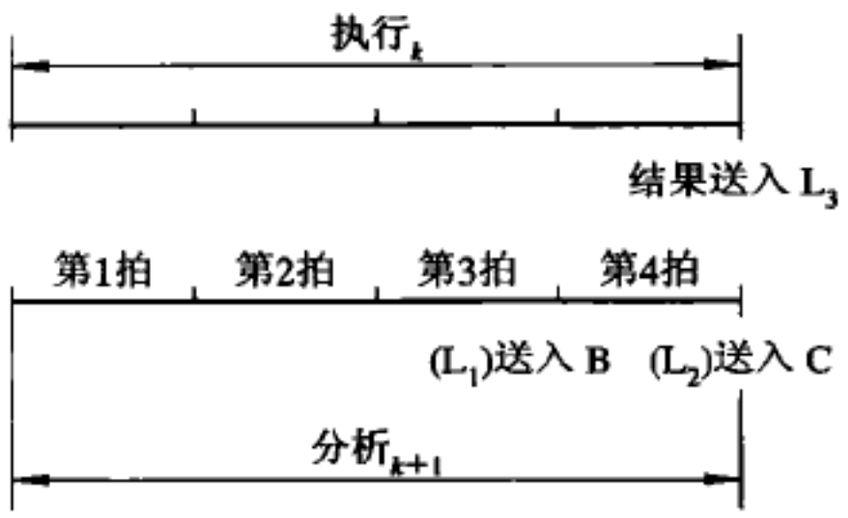


图 5 - 9 “执行$_k$”、“分析$_{k+1}$”重叠时，访问通用寄存器组的时间关系

显然，要想解决通用寄存器组数相关，一种办法是可以与前述处理主存空间数相关一样，推后“分析$_{k+1}$”的读到“执行$_k$”结束时开始，也可以推后到“执行$_k$”把结果送入 L_3，然后再由“分析$_{k+1}$”在取(L_1)或(L_2)时能取到即可。采用前者，只要发生数相关就使一次重叠变成了完全的顺序串行，速度明显下降；采用后者则发生数相关时，相邻两条指令的解释仍有部分重叠，可以减少速度损失，但控制要复杂一些。这两种办法都是靠推后读，牺牲速度来避免相关时出错。那么能否在不降低速度的情况下仍保证数相关时处理的正确呢?

考虑到多数机器都是把通用寄存器作为累加器使用的，或者是用它来保存中间结果的，因此，本条指令存进去的结果往往在下一条指令又作为操作数取出来使用，并且发生通用寄存器组数相关的概率是很高的。可以分析一下运算器与通用寄存器组之间的数据通路。一般情况下，运算器产生的运算结果可以直接送入通用寄存器组中，而通用寄存器组则需要通过数据总线将数据送入运算器的操作数寄存器 B 或 C 中。这样，一旦发生相关，为保证数据的正确，从第 k 条指令“执行$_k$”末尾将形成的运算结果送入通用寄存器，到第 $k+1$ 条指令再从通用寄存器组中读出该结果，并经数据总线送入运算器的操作数寄存器 B 或 C，直至稳定，就需要将“分析$_{k+1}$”推后很长的一段时间。这种推后“分析$_{k+1}$”读的办法会使重叠效率急剧下降。如果在运算器的输出到 B 或 C 输入之间增设“相关专用通路”，如图 5 - 10 所示，则在发生 L_1 或 L_2 相关时，接通相应的相关专用通路，“执行$_k$”时就可以在将运算结果送入通用寄存器完成其应有的功能的同时，直接将运算结果回送到 B 或 C 寄存器，从而大大缩短了其间的传送时间，并保证“执行$_{k+1}$”用此操作数时，它已在 B 或 C 寄存器中准备好了。就是说，尽管原先将通用寄存器的旧内容经数据总线分别在“分析$_{k+1}$”的第 3 拍或第 4 拍末送入了操作数寄存器 B 或 C 中，但之后经相关专用通路在“执行$_{k+1}$”真正用它之前，操作数寄存器 B 或 C 重新获得第 k 条指令送来的新结果。这样，既可以保证相关

时不用推后“分析$_{k+1}$”，使重叠效率不下降，又可以保证指令重叠解释时数据不出错。

推后“分析$_{k+1}$”和设置“相关专用通路”是解决重叠方式相关处理的两种基本方法。前者以降低速度为代价，使设备基本上不增加。后者以增加设备为代价，使重叠效率不下降。

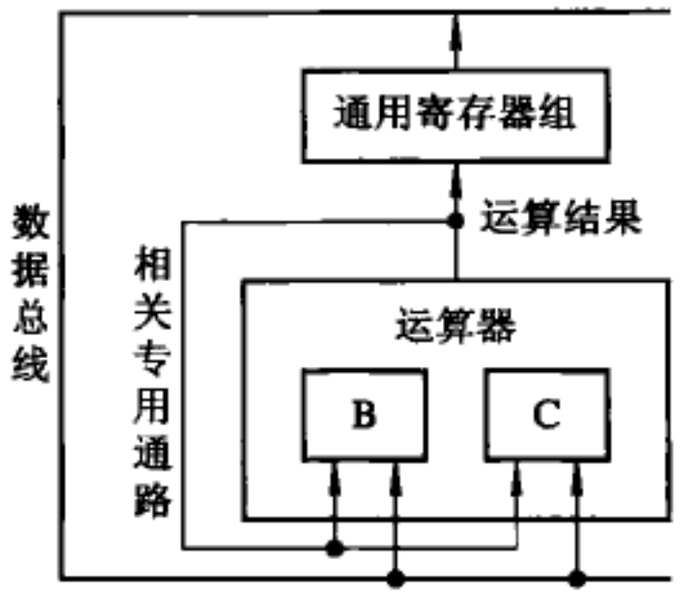


图 5 - 10　用相关专用通路解决通用寄存器组的数相关

可以看出，若相关的概率低，就不宜采用“相关专用通路”法，这样既节省了设备，也不会使重叠效率有明显下降。因此，尽管概念上只需把图 5 - 10 中的通用寄存器组改为主存，“相关专用通路”法就可用于解决前述的主存空间数相关，但由于出现主存数相关的概率要比出现通用寄存器组数相关的概率低得多，因此按哈夫曼(Huffman)思想解决主存数相关时，不用“相关专用通路”法，而采用推后读法。

现在来讨论通用寄存器组基址值或变址值相关的处理。下面只以基址值相关为例来讲述，变址值相关是类似的。

设操作数的有效地址$(X_d)+(B_2)+d_2$是由分析器中的地址加法器形成的。由于多数情况的“分析”周期等于主存周期，因此，从时间上要求，在“分析”周期的前半段就应由通用寄存器输出总线取得(B_2)，送入地址加法器。由于运算结果是在“执行”周期的末尾才送入通用寄存器组的，因此它当然不能立即出现在通用寄存器输出总线上。也就是说，在“执行$_k$”得到的、送入通用寄存器的运算结果是来不及为“分析$_{k+2}$”作基址值用，更不用说为“分析$_{k+1}$”作基址值用。因此，一次重叠时，基址值相关(B 相关)不仅会出现一次相关，还会出现二次相关。即$B_{(k+1)}=L_{3(k)}$时发生 B 一次相关，$B_{(k+2)}=L_{3(k)}$时发生 B 二次相关，如图 5 - 11 所示。这里所谓的一次和二次指的是相关指令相隔的指令条数。

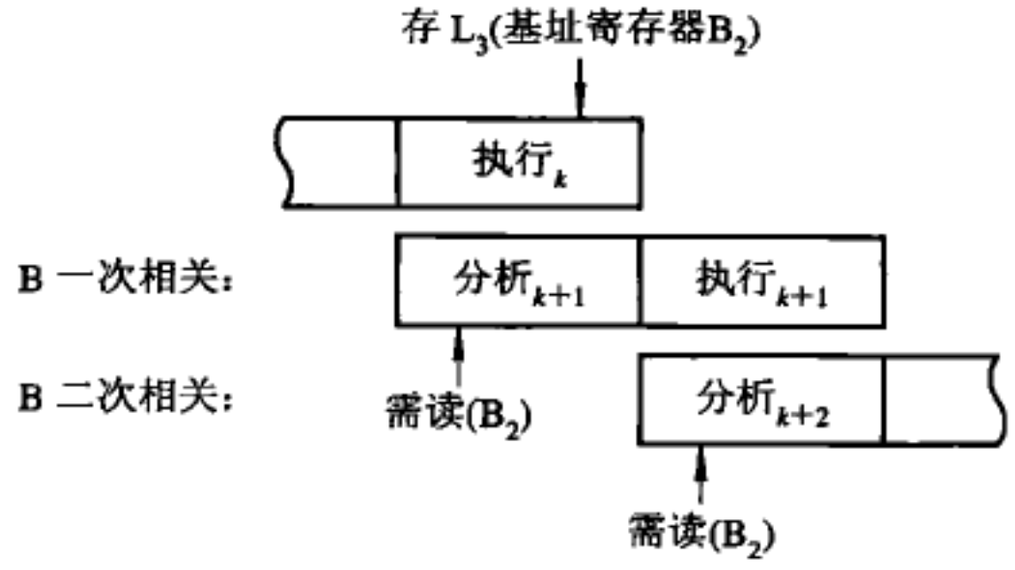


图 5 - 11　B 一次相关与二次相关

对于基址值相关，其解决办法同数相关一样，也可有推后分析和设置相关专用通路两种。先讲推后分析的方法。

由图 5 - 11 可见，B 二次相关时，只需推后“分析$_{k+2}$”的始点到“执行$_k$”送入通用寄存器的运算结果能在“分析$_{k+2}$”开始时出现于通用寄存器输出总线上即可，如图 5 - 12(a)所示。至于推后多少拍，这取决于通用寄存器组译码、读出机构的具体逻辑组成。而对 B 一次相关，则除此之外，还需再推后一个“执行”周期，如图 5 - 12(b)所示。

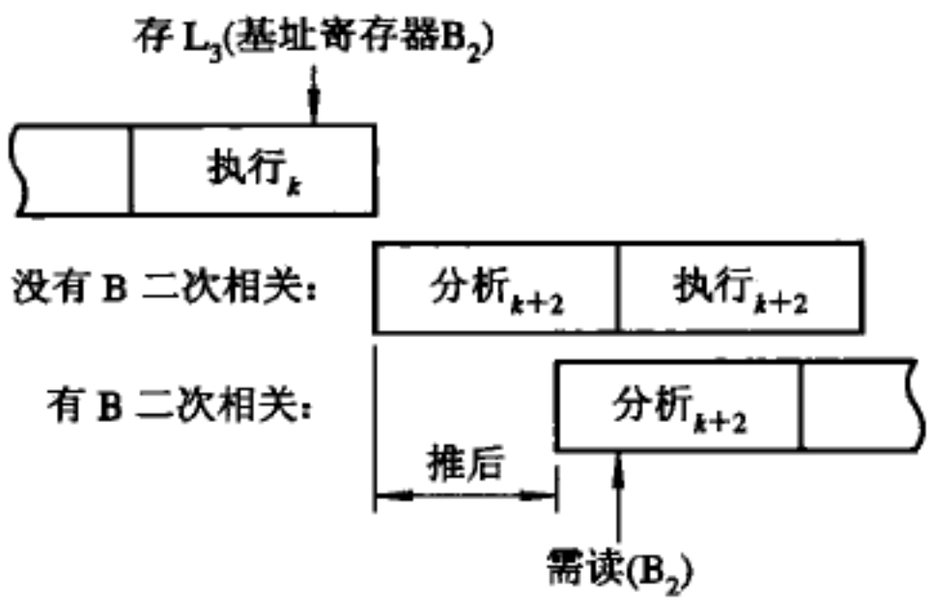


图 5 - 12　B 一次与二次相关的推后处理

(a) B 二次相关的推后处理；(b) B 一次相关的推后处理

由于 B 相关的概率并不是很低，增设 B 相关专用通路是值得的，办法如图 5 - 13 所示。在 B 二次相关时，“执行$_k$”得到的运算结果在送入通用寄存器组的同时，也经 B 相关专用通路直接送到访存操作数地址形成机构。因缩短了其间的传送延迟，使得不用推后“分析$_{k+2}$”就能使用正确的基址值。同理，在 B 一次相关时，还需推后“分析$_{k+1}$”到“执行$_k$”后开始。所以同样可以采用类似于延迟转移技术的思想，由编译程序调整指令生成的顺序，使之尽可能只发生 B 二次相关，不发生 B 一次相关，确保设置 B 相关专用通路后，在 B 相关时，重叠效率不会下降。

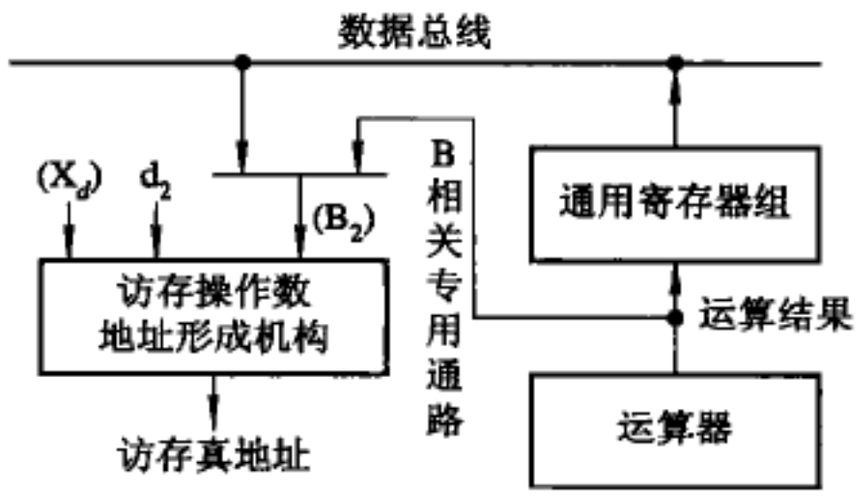


图 5 - 13　B 相关专用通路法

为了实现两条指令在时间上的重叠解释，首先需要付出空间代价，如增设数据总线、控制总线、指令缓冲器、地址加法器、相关专用通路，将指令分析部件和指令执行部件功能分开、单独设置，主存采用多体交叉存取，等等。其次，要处理好指令之间可能存在的关联，如转移的处理，指令相关、主存空间数相关、通用寄存器组的数相关或基址值或变址值相关等的处理。操作数相关或基址值或变址值相关处理的办法无非是推后读和设置相关专用通路两种，应根据哈夫曼思想在成本和效率上权衡选用。此外，还应合理调配好机器指令顺序及指令内部微操作的时间关系，并使“分析”和“执行”的时间尽可能相等，以提高重叠的效率。

5.2　流 水 方 式

如果采用一次重叠方式解释指令仍达不到速度要求，可采用同时解释多条指令的流水方式。

5.2.1　基本概念

1. 工作原理

“分析$_{k+1}$”与“执行$_k$”的一次重叠是把指令的解释过程分解成“分析”与“执行”两个子过

程，在独立的分析部件和执行部件上时间重叠地进行。若“分析”与“执行”子过程都需要 Δt_1 的时间，如图 5 - 14 所示，则一条指令的解释需要 $2\Delta t_1$ 完成，但机器每隔 Δt_1 就能解释完一条指令。就是说，由指令串行解释到一次重叠解释，机器的最大吞吐率(单位时间内机器所能处理的最多指令条数或机器能输出的最多结果数)提高了一倍。

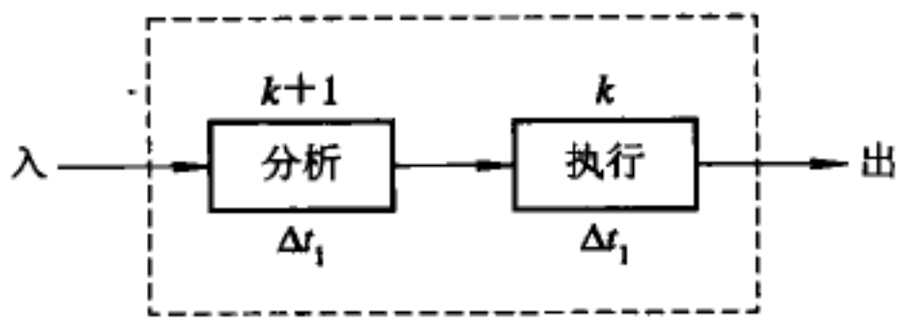


图 5 - 14　指令分解为“分析”与“执行”子过程

如果把“分析”子过程再细分成“取指令”、“指令译码”和“取操作数”3 个子过程，并改进运算器的结构以加快其“执行”子过程(如图 5 - 15(a)所示，这 4 个子过程分别由独立的子部件实现)，让经过的时间都等于 Δt_2，则指令解释的时(间)空(间)关系如图 5 - 15(b)所示。图中的 1、2、3、4、5 表示处理机所处理的第 1、2、3、4、5 条指令。

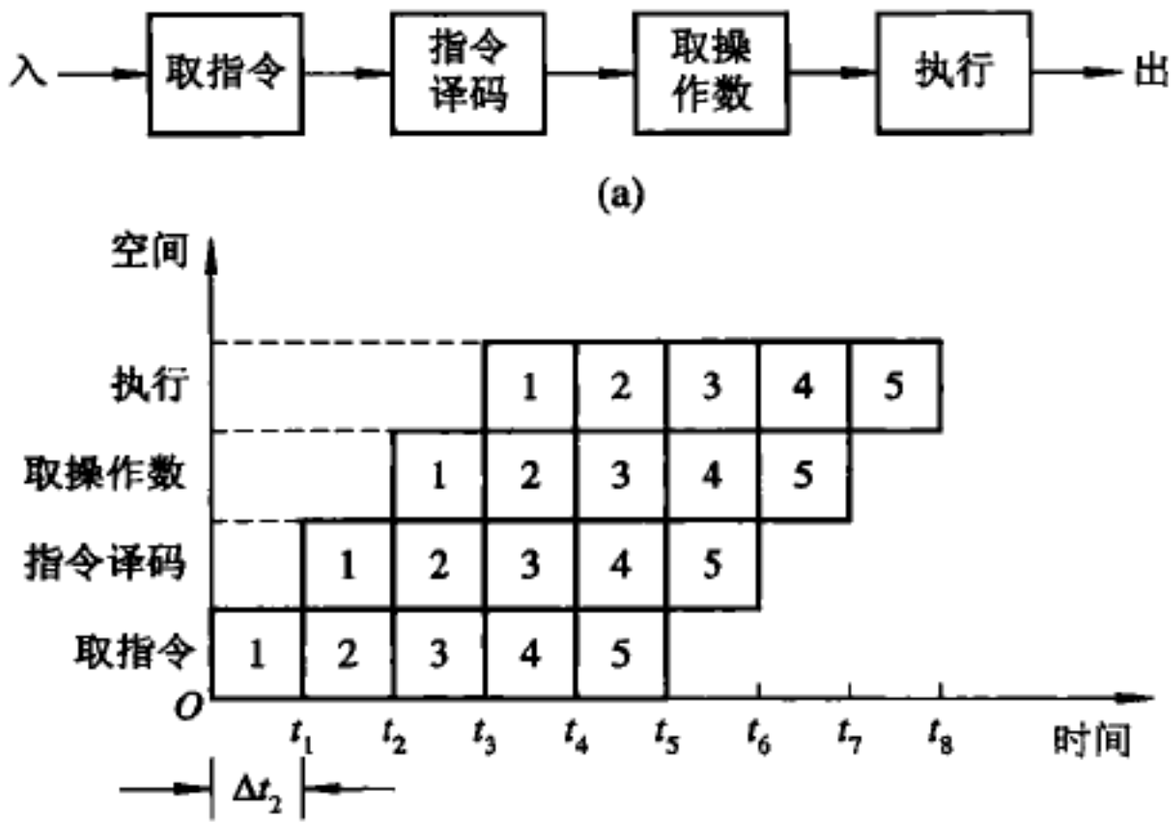


图 5 - 15　流水处理

(a) 指令解释的流水处理；(b) 流水处理的时(间)空(间)图

如果完成一条指令的时间为 T，对分解为“分析”和“执行”2 个子过程的，其 $T=2\Delta t_1$；而对分解为“取指令”、“指令译码”、“取操作数”和“执行”4 个子过程的，其 $T=4\Delta t_2$。这样，对图 5 - 14 是每隔 $\Delta t_1=T/2$ 就可由处理机流出一个结果，吞吐率提高了一倍；而对图 5 - 15 是每隔 $\Delta t_2=T/4$ 流出一个结果，吞吐率比顺序方式提高了 3 倍。

流水与重叠在概念上没有什么差别，可以看成是重叠的引申。差别只在于“一次重叠”是把一条指令的解释分为两个子过程，而流水是分为更多个子过程。前者同时解释两条指令，后者如图 5 - 15 所示可同时解释 4 条指令。显然，如能把一条指令的解释分解成时间相等的 m 个子过程，则每隔 $\Delta t=T/m$ 就可以处理一条指令。因此，流水的最大吞吐率取决于子过程的经过时间 Δt，Δt 越小，流水的最大吞吐率就越高。流水的最大吞吐率是指流水线满负荷每隔 Δt 流出一个结果时所达到的吞吐率。实际上，流水线从开始启动到流出第一个结果，需要经过一段流水线的建立时间 T_0，在这段时间里流水线并未流出任何结果。

所以，实际吞吐率总是低于其最大吞吐率的。

在计算机实际的流水线中，各子部件经过的时间会有所不同。为平滑这些子部件的速度差，一般在它们之间设有锁存器。锁存器都受同一时钟信号控制，用以实现各子部件信息流的同步推进。时钟信号周期不得低于速度最慢子部件的经过时间与锁存器的存取时间之和，还要考虑时钟信号到各锁存器可能存在时延。所以，子过程的细分会因锁存器数增多而增大任务或指令流过流水线的时间，这在一定程度上会抵消子过程细分使吞吐率提高的好处。

2. 流水的分类

从不同的角度对流水可进行不同的分类。

依据向下扩展和向上扩展的思路，可对在计算机系统不同等级上使用的流水线进行分类。所谓向下扩展，指的是把子过程进一步地细分，让每个子过程经过的时间都同等程度地减少，吞吐率就会进一步提高。例如，可以把图 5 - 14 的“分析”子过程和“执行”子过程再细分。

“执行”子过程的细分因指令功能和执行时间的不同而不同。如浮点加可进一步细分成“求阶差”、“对阶”、“尾数相加”、“规格化”4 个子过程。子过程的细分是以增加设备为代价的。例如，“求阶差”和“尾数相加”都要用加法器，为此需分别设置阶码加法器和尾数加法器；“对阶”和“规格化”都要用移位器，需设置两套移位器。设备的增加不仅使成本增加，也使控制变得复杂。但与完全靠重复设置多套分析部件和执行部件来提高指令的并行度相比，其设备量的增加要少得多。子过程细分并不是无止境的，因为级间缓冲器的增多会使成本提高、辅助延时增大、控制复杂、电路实现和组装困难，它们抵消了子过程细分的好处。所以，目前计算机功能级流水分割的子过程数很少有超过 10 的。

流水技术也可“向下”应用于对 Cache 存储器和多体并行主存的访问，使存储器频宽得以提高，这在第 4 章已提到过了。

流水的向上扩展可理解为在多个处理机之间流水，如图 5 - 16 所示。多个处理机串行地对数据集进行处理，每个处理机专门完成其中的一个任务。因为各处理机都在同时工作，所以能流水地对多个不同的数据集进行处理，可较大地提高计算机系统的处理能力。

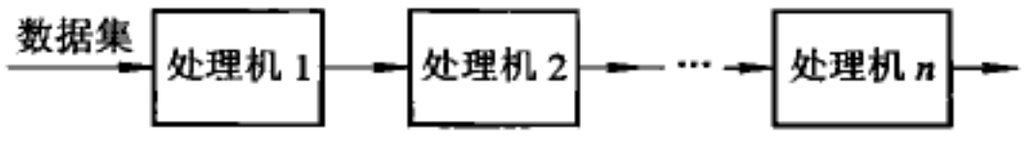


图 5 - 16　处理机间的流水处理

流水按处理的级别可分为部件级、处理机级和系统级。部件级流水是指构成部件内的各个子部件间的流水，如运算器内浮点加的流水、Cache 内和多体并行主存内的流水。处理机级流水是指构成处理机的各部件之间的流水，如“取指”、“分析”、“执行”间的流水。系统级流水是指构成计算机系统的多个处理机之间的流水，也称为宏流水。

从流水线具有功能的多少来看，可以将流水线分为单功能流水线和多功能流水线。

单功能流水线只能实现单一功能的流水，如只能实现浮点加减的流水线。要完成多种功能的流水可将多个单功能流水线组合。如 CRAY - 1 有 12 条单功能流水线，分别完成地址加、地址乘、标量加、标量移位、标量逻辑运算、标量数“数”、向量加、向量移位、向量逻辑运算、浮点加、浮点乘、浮点迭代求倒数。

多功能流水线指的是同一流水线的各个段之间可以有多种不同的连接方式，以实现多

种不同的运算或功能。例如 TI－ASC 计算机的运算器流水线就是多功能的，它有 8 个可并行工作的独立功能段，如图 5－17(a)所示。当要进行浮点加、减时，连接成如图 5－17(b)所示的形式。当要进行定点乘法运算时，连接成如图 5－17(c)所示的形式。除此之外，它还可以有数十种其他不同的连接，可实现多种其他功能。

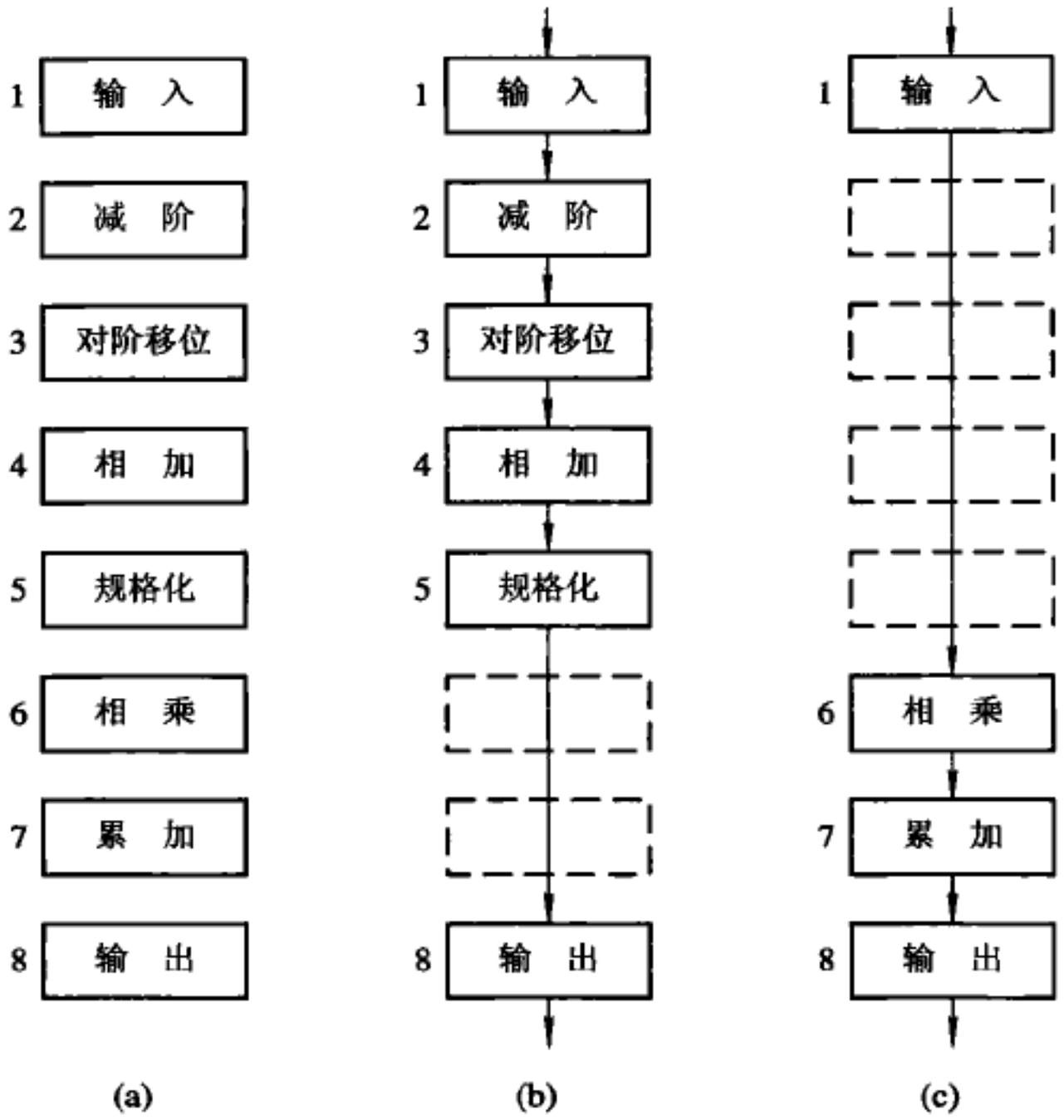


图 5－17　ASC 计算机运算器的流水线

（a）流水线的功能段；（b）浮点加、减法运算时的连接；（c）定点乘法运算时的连接

按多功能流水线的各段能否允许同时用于多种不同功能连接流水，可把流水线分为静态流水线和动态流水线。

静态流水线在某一时间内各段只能按一种功能连接流水，只有等流水线全部流空后，才能按另一种功能连接流水。这样，就指令级流水而言，仅当进入的是一串相同运算的指令时，才能发挥出静态流水线的效能。若进入的是浮加、定乘、浮加、定乘、……相间的一串指令时，静态流水线的效能会降低到比顺序方式的还要差。因此，在静态流水线机器中，要求程序员编制出(或是编译程序生成)的程序应尽可能调整成有更多相同运算的指令串，以提高其流水的效能。

动态流水线的各功能段在同一时间内可按不同运算或功能连接。如图 5－17 中所示各功能段在同一时间里，某些段按浮点加减连接流水，而另一些段却在按定乘连接流水。这样，就不要求流入流水线的指令串非得有相同的功能，也能提高流水的吞吐率和设备的利用率，但其控制复杂，成本高。图 5－18 画出了静态和动态多功能流水线的时空图，从中可以看出它们在工作方式和性能上的差异。目前大多数高性能流水处理机都采用多功能静态流水，因为其控制和实现比较简便。从软、硬件功能分配的观点上看，静态流水线将功能负担较多地加到软件上，以简化硬件控制；动态流水线则把功能负担较多地加到硬件控制上，以提高流水的效能。

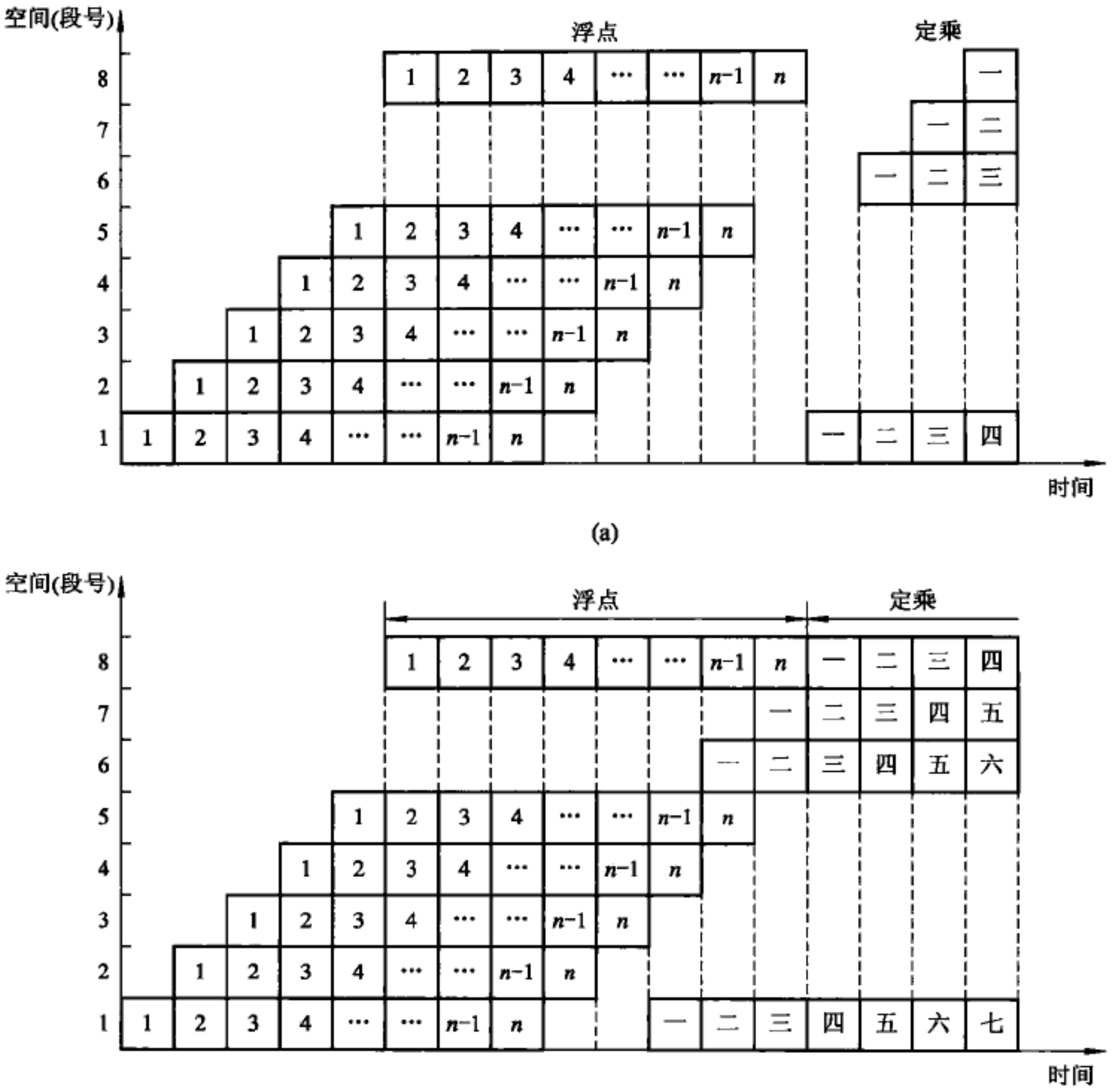


图 5 - 18　静、动态多功能流水线时空图

(a) 静态；(b) 动态

从机器所具有的数据表示可以把流水线处理机分为标量流水机和向量流水机。标量流水机没有向量数据表示，只能用标量循环方式来处理向量和数组，如 Amdahl 470 V/6 及后面要介绍的 IBM 360/91。向量流水机指的是机器有向量数据表示，设置有向量指令和向量运算硬件，能流水地处理向量和数组中的各个元素。向量流水机是向量数据表示和流水技术的结合，如后面要介绍的 CRAY - 1。

从流水线中各功能段之间是否有反馈回路，可把流水线分为线性流水和非线性流水。流水线各段串行连接，各段只经过一次，没有反馈回路的，称为线性流水线。流水线除有串行连接的通路，还有反馈回路，使任务流经流水线需多次经过某个段或越过某些段，则称为非线性流水线。图 5 - 19 就是一个非线性流水线，由 4 段组成，经反馈回路和多路开关使某些段如 2 段、3 段或 4 段可能要多次通过，而有些段如 2 段可能被跳过。非线性流水线的一个重要问题是确定新任务什么时候流入流水线，使之不会与先前的任务争用流水段。这将在流水线的调度一节中讨论。

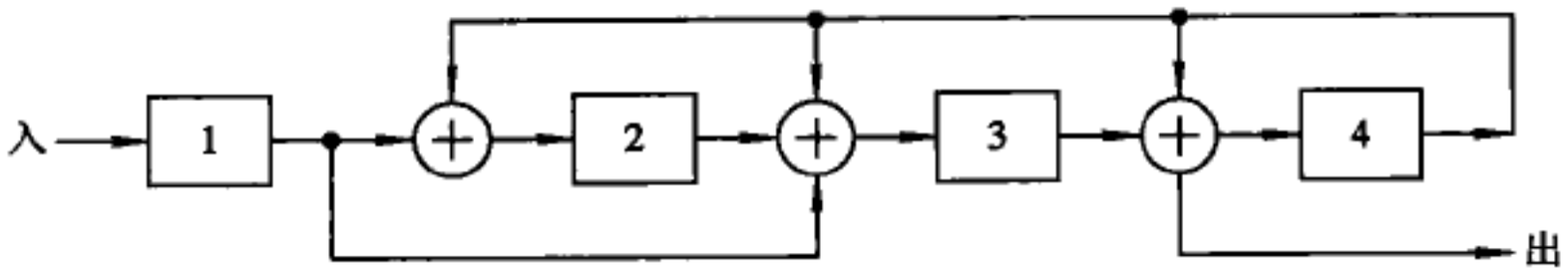


图 5 - 19　非线性流水线

随着 VLSI 技术的发展，除了简单一维流水外，又发展了复杂的多维流水线，即同时经多个方向流水，称之为脉动阵列流水。

5.2.2　标量流水线的主要性能

标量流水线的性能主要是吞吐率 T_p、加速比 S_p 和效率 η。

1. 吞吐率 T_p 和加速比 S_p

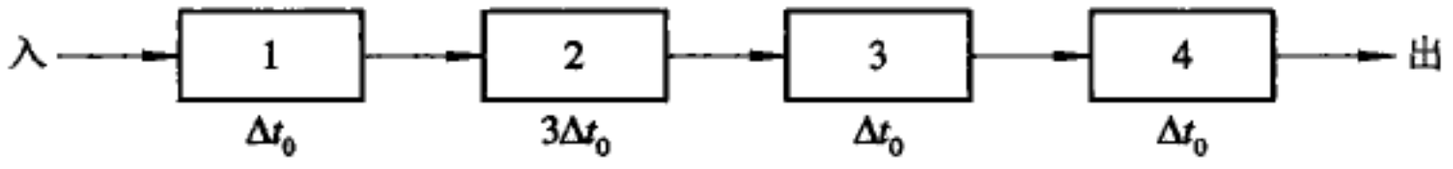
吞吐率是流水线单位时间里能流出的任务数或结果数。

在图 5 - 15(b)所示的流水线例中，各子过程经过的时间都是 Δt_2，满负荷后流水线每隔 Δt_2 解释完一条指令，其最大吞吐率 $T_{p_{max}}$ 为 $1/\Delta t_2$。实际上，各个子过程进行的工作不同，所经过的时间也就不一定相同，所以在子过程间设置了接口锁存器，让各锁存器都受同一时钟同步。时钟周期会直接影响流水线的最大吞吐率，总希望它越小越好。如果各个子过程所需的时间分别为 Δt_1、Δt_2、Δt_3、Δt_4，时钟周期应当为 $\max\{\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4\}$，即流水线的最大吞吐率

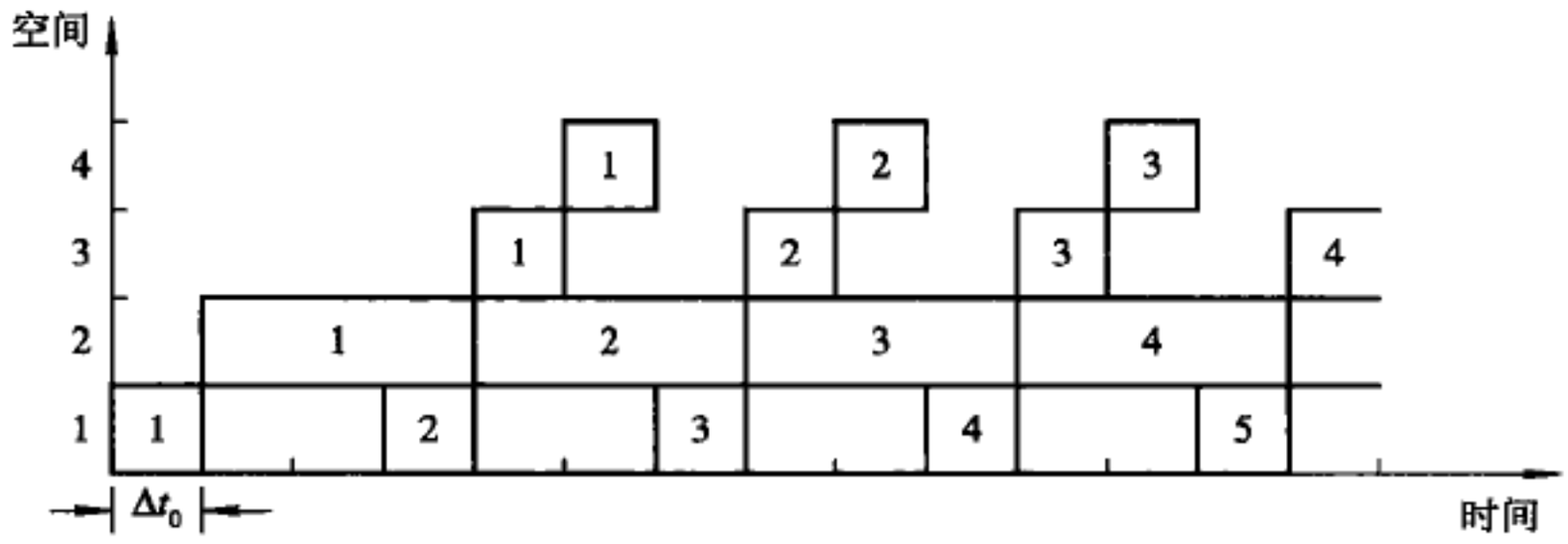
$$T_{p_{max}} = \frac{1}{\max\{\Delta t_1, \Delta t_2, \Delta t_3, \Delta t_4\}}$$

它受限于流水线中最慢子过程经过的时间。流水线中经过时间最长的子过程称为瓶颈子过程。

【例 5 - 1】 有一个 4 段的指令流水线如图 5 - 20(a)所示，其中，1、3、4 段的经过时间均为 Δt_0，只有 2 段的经过时间为 $3\Delta t_0$，因此瓶颈在 2 段，使整个流水线的最大吞吐率只有$1/(3\Delta t_0)$，其时空图如图 5 - 20(b)所示。即使流水线每隔 Δt_0 流入一条指令，也会因来不及处理被堆积于 2 段，致使流水线仍只能每隔 $3\Delta t_0$ 才流出一条指令。



(a)



(b)

图 5 - 20　最大吞吐率取决于瓶颈段的时间

为了提高流水线的最大吞吐率，首先要找出瓶颈，然后设法消除此瓶颈。消除瓶颈的一种办法是将瓶颈子过程再细分。例如将 2 段再细分成 21、22、23 三个子段，如图 5 - 21(a)所示。让各子段经过时间都减少到 Δt_0，这样，最大吞吐率就可提高到 $1/\Delta t_0$。图 5 - 21(b)是将瓶颈子过程再细分后的时空图。然而，并不是所有子过程都能再细分。例如 2 段已不能再细分了，则可以通过重复设置多套(如此例用 3 套)瓶颈段并联，让它们交叉并行，如图 5 - 22(a)所示。每隔 Δt_0 轮流给其中一个瓶颈段分配任务，使它们仍可每隔 Δt_0 解释完一条指令，其时空图见图 5 - 22(b)。这种办法需要解决好在各并行子过程之间的任务分配和同步控制，这比瓶颈子过程再细分控制要复杂、所需设备量要多。

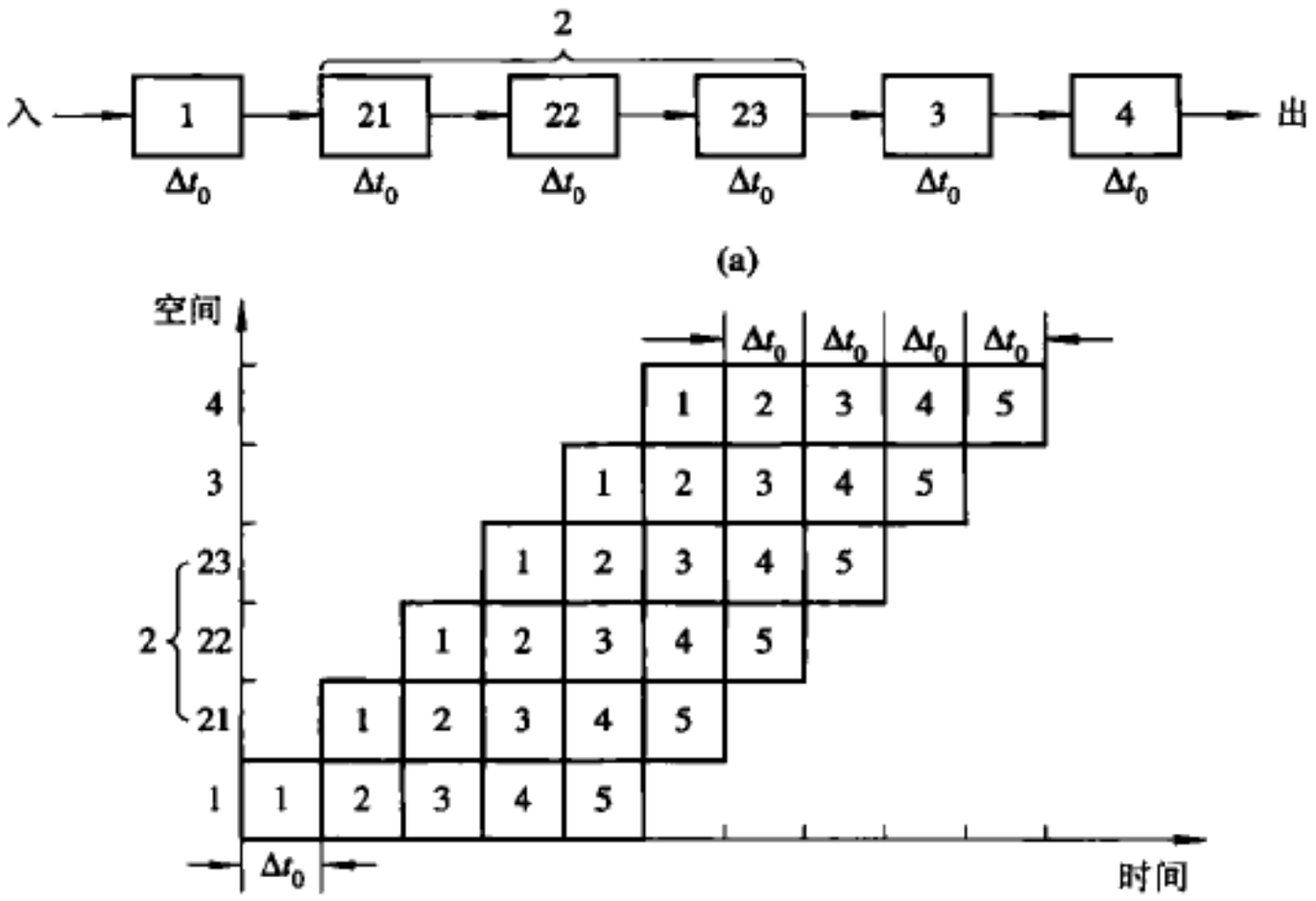


图 5 - 21　瓶颈子过程再细分

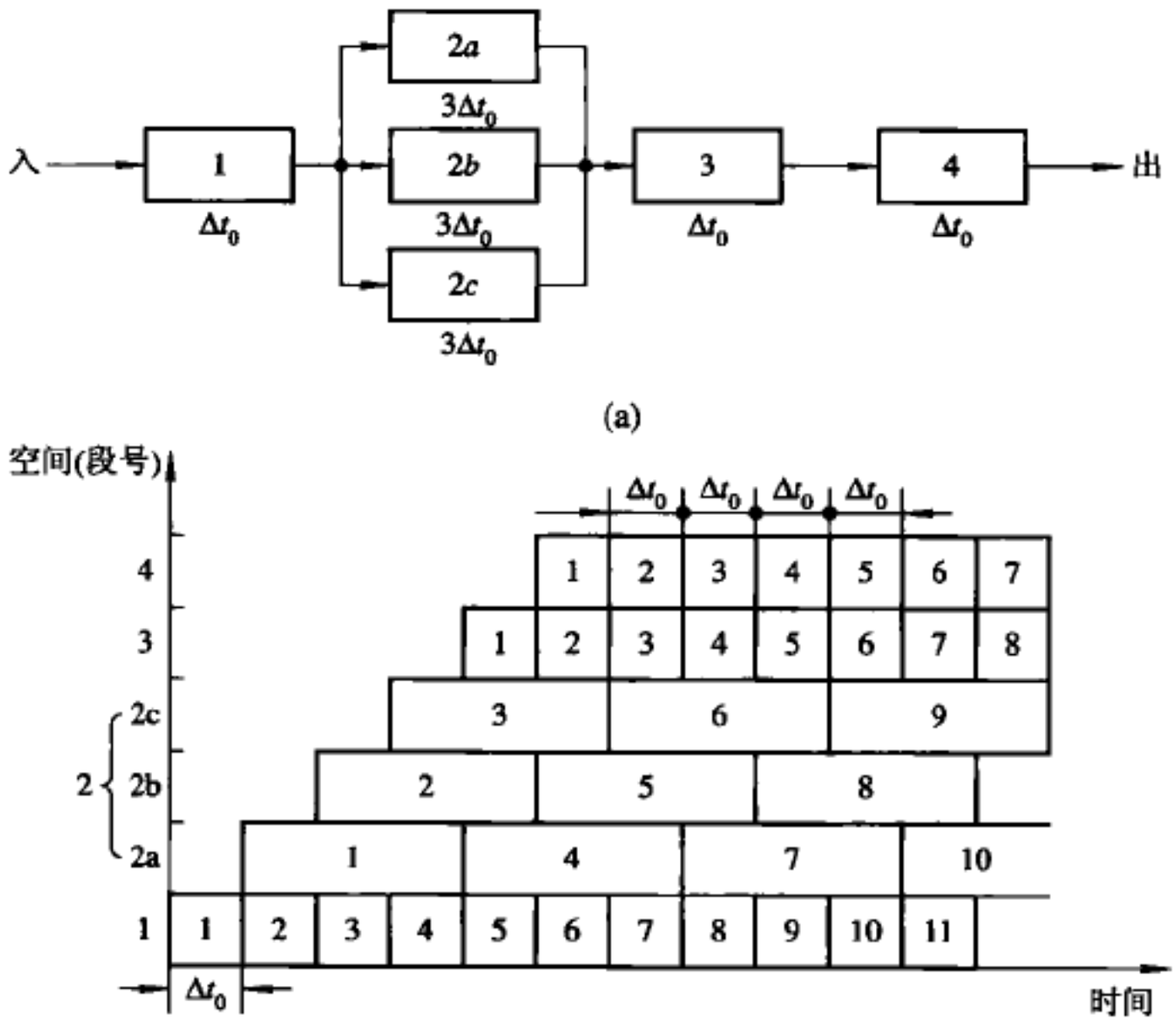


图 5 - 22　瓶颈子过程并联

以上讲的都是流水线连续流动时能达到的最大吞吐率。由于流水开始时总要有一段建立时间，加上各种原因使流水线不能连续流动，经常是流一段时间，停一段时间，因此流水线的实际吞吐率 T_p 总比最大吞吐率 $T_{p_{max}}$ 要小。

设一 m 段流水线的各段经过时间均为 Δt_0，则第 1 条指令从流入到流出需要 $T_0 = m\Delta t_0$ 的流水建立时间，之后每隔 Δt_0 就可流出一条指令，图 5 - 23 为其时空图(这里设 $m=4$)。这样，完成 n 个任务的解释共需时间 $T = m\Delta t_0 + (n-1)\Delta t_0$，流水线在这段时间里的实际吞吐率

$$T_p = \frac{n}{m\Delta t_0 + (n-1)\Delta t_0} = \frac{1}{\Delta t_0\left(1 + \frac{m-1}{n}\right)} = \frac{T_{p_{max}}}{1 + \frac{m-1}{n}}$$

可见不仅实际吞吐率总是小于最大吞吐率，而且只有当 $n \gg m$ 时，才能使实际吞吐率接近于最大吞吐率。如果用加速比 S_p 表示流水方式相对于非流水顺序方式速度提高的比值，那么非流水顺序方式连续完成 n 个任务需要 $nm\Delta t_0$ 的时间，因此，流水方式工作的加速比

$$S_p = \frac{nm\Delta t_0}{m\Delta t_0 + (n-1)\Delta t_0} = \frac{m}{1 + \frac{m-1}{n}}$$

所以线性流水线各段时间相等时，仅当 $n \gg m$，即连续流入的任务数 n 远多于流水线子过程数 m 时，其加速比才能趋于最大值，为流水线的段数 m。

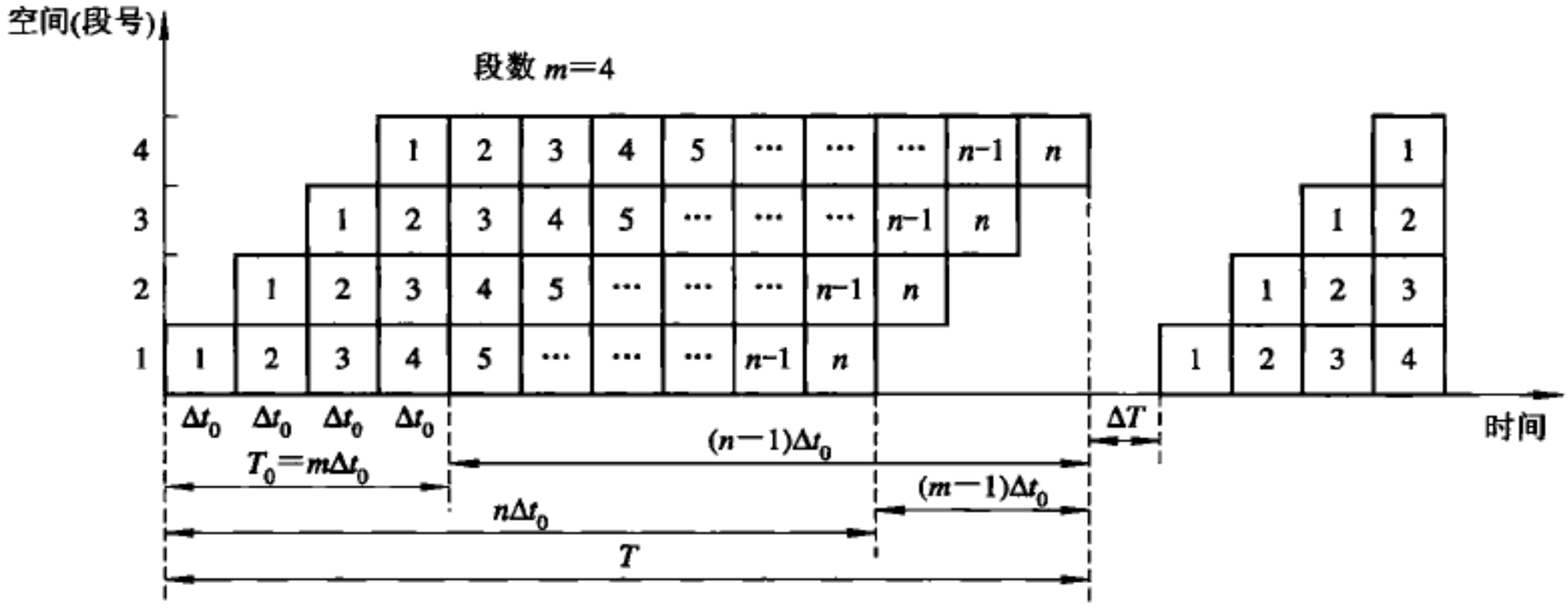


图 5 - 23　从时空图分析实际的吞吐率

如果只是通过细分子过程，增大 m 来缩短 Δt_0，而未能在软件、算法、语言编译、程序设计上保证连续流动的任务数 n 远远大于子过程数 m，则实际吞吐率将大大低于最大吞吐率。极限情况 $n=1$ 时，由于 m 的增大，锁存器增多，实际增大了任务在流水线上的通过时间，反而使其速度会比顺序串行的还要低。

如果线性流水线每段经过的时间 Δt_i 不等，其中瓶颈段的时间为 Δt_j，则完成 n 个任务所能达到的实际吞吐率

$$T_p = \frac{n}{\sum_{i=1}^{m} \Delta t_i + (n-1)\Delta t_j}$$

其加速比

$$S_p = \frac{n\sum_{i=1}^{m}\Delta t_i}{\sum_{i=1}^{m}\Delta t_i + (n-1)\Delta t_j}$$

2. 效率

流水线的效率是指流水线中设备的实际使用时间与整个运行时间之比，也称流水线设备的时间利用率。由于流水线存在建立时间和排空时间(最后一个任务流入到流出的时间)，因此在连续完成 n 个任务的时间里，各段并不总是满负荷工作的。

如果是线性流水线、任务间不相关且各段经过的时间相同，如图 5 - 23 所示那样，则在 T 时间里，流水线各段的效率都相同，均为 η_0，即

$$\eta_1 = \eta_2 = \cdots = \eta_m = \frac{n\Delta t_0}{T} = \frac{n}{m+(n-1)} = \eta_0$$

整个流水线的效率

$$\eta = \frac{\eta_1 + \eta_2 + \cdots + \eta_m}{m} = \frac{m\eta_0}{m} = \eta_0 = \frac{mn\Delta t_0}{mT}$$

式中，分母 mT 是时空图中 m 个段和流水总时间 T 所围成的面积，分子 $mn\Delta t_0$ 是时空图中 n 个任务实际使用的面积。因此，从时空图上看，效率实际上就是 n 个任务占用的时空区面积和 m 个段总的时空区面积之比。显然，只有当 $n \gg m$ 时，η 才趋于 1。同时还可以看出，当为线性流水且每段经过时间相等，任务间无相关时，流水线的效率才正比于吞吐率，即

$$\eta = \frac{n\Delta t_0}{T} = \frac{n}{n+(m-1)} = T_p\Delta t_0$$

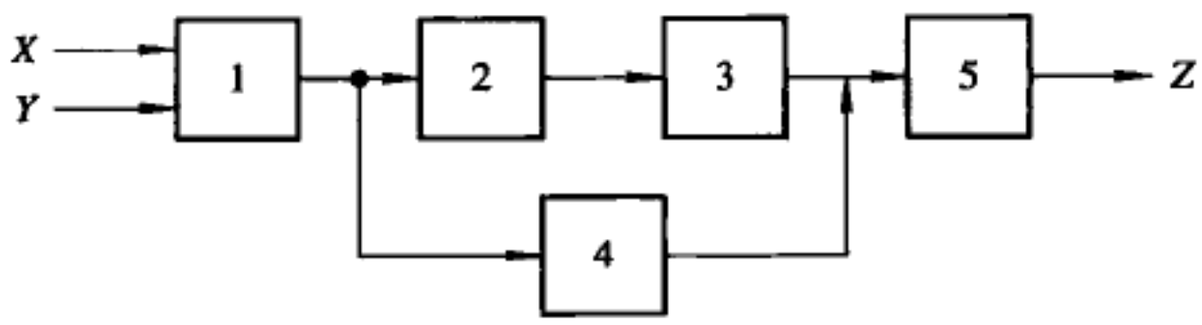
当然，当为非线性流水或为线性流水但各段经过的时间不等或任务间有相关时，这种正比的关系就不存在了，此时通过画实际工作的时空图才能求出吞吐率和效率。一般为提高效率、减少时空图中空白区所采取的措施也会为提高吞吐率带来好处。因此，在图5 - 18所示的多功能流水线中，动态流水比起静态流水减少了空白区，从而使流水线吞吐率和效率都得到了提高，但是参照图 5 - 23，不难得出整个流水线的效率

$$\eta = \frac{n\text{ 个任务实际占用的时空区}}{m\text{ 个段总的时空区}} = \frac{n\sum_{i=1}^{m}\Delta t_i}{m\left[\sum_{i=1}^{m}\Delta t_i + (n-1)\Delta t_j\right]}$$

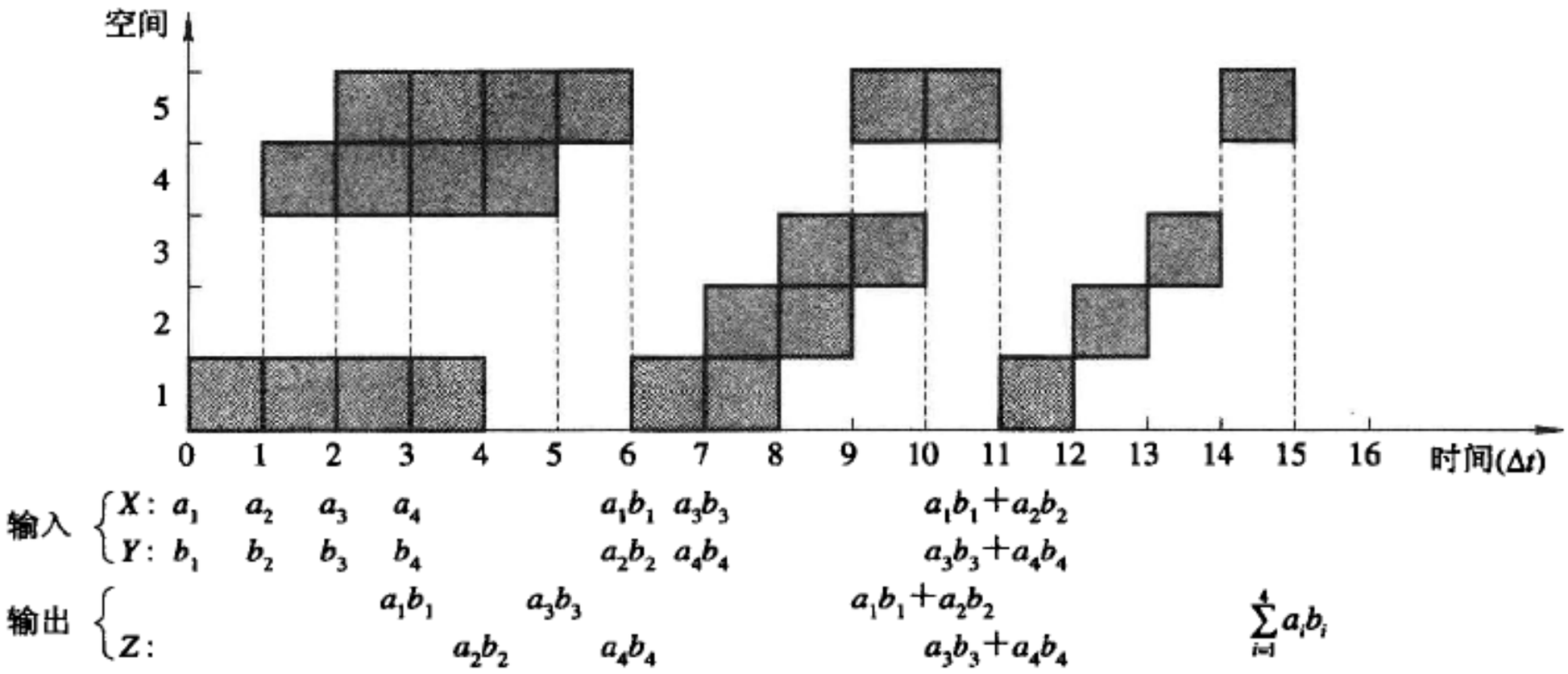
【例 5 - 2】 设向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 各有 4 个元素，要在图 5 - 24(a)所示的静态双功能流水线上计算向量点积 $\boldsymbol{A}\cdot\boldsymbol{B} = \sum_{i=1}^{4} a_i b_i$。其中，1→2→3→5 组成加法流水线，1→4→5 组成乘法流水线。又设每个流水线所经过的时间均为 Δt，流水线输出可直接返回输入或暂存于相应缓冲寄存器中，其延迟时间和功能切换所需的时间都可忽略。试求流水线从开始流入到结果流出这段时间的实际吞吐率 T_p 和效率 η。

先选择适合于静态流水线工作的算法使完成向量点积 $\boldsymbol{A}\cdot\boldsymbol{B}$ 所用的时间最短。本题可先连续计算 a_1b_1、a_2b_2、a_3b_3 和 a_4b_4 4 个乘法，然后切换功能，按$((a_1b_1+a_2b_2)+(a_3b_3+a_4b_4))$经 3 次加法来求得最后的结果。按此算法可画出流水线工作的时空图如图 5 - 24(b)所示。

在 15 个 Δt 时间内流出 7 个结果，其实际吞吐率 T_p 为$7/(15\Delta t)$，而顺序方式所需时间为 $4\times3\Delta t+3\times4\Delta t=24\Delta t$。因此，加速比 $S_p=24\Delta t/(15\Delta t)=1.6$。



(a)



(b)

图 5 - 24　流水线工作举例

该流水线的效率可用阴影区面积和全部 5 个段的总时空区面积之比求得，即

$$\eta=\frac{3\times4\Delta t+4\times3\Delta t}{5\times15\Delta t}=32\%$$

虽然效率连 1/3 都不到，但解题速度却提高为串行的 1.6 倍，而且如果向量 $\boldsymbol{A}$、$\boldsymbol{B}$ 的元素数增加时，还会使解题速度和效率进一步提高。影响吞吐率和效率提高的因素很多。一是静态多功能流水线按某种功能流水时，总有一些本功能用不到的段处于空闲；二是流水建立时，本功能要用到的某些段也有部分处于空闲；三是功能切换时，增加了前一功能流水的排空时间及后一功能流水的建立时间；四是经常需要等待把上一步计算的结果输出回授到输入，才能开始下一步的计算，这就是下一节要讨论的相关问题。所以，流水线最适合解具有同一操作类型，且输出与输入之间没有任何联系和相关的一串运算。只要能连续给流水线提供输入数据，就能使流水线不间断地流动。当 n 值很大时，流水线的效率就可接近于 1，实际吞吐率就可接近于最大吞吐率——$1/\Delta t$。

5.2.3　标量流水机的相关处理和控制机构

标量流水线只有连续不断地流动，不出现断流，才能获得高效率。造成流水线断流的原因除了编译形成的目标程序不能发挥流水结构的作用，或存储系统供不上为连续流动所需的指令和操作数以外，还可能是由于出现了相关和中断。

在 5.1 节已讲过重叠方式的转移指令引起的相关、指令相关、主存操作数相关、通用寄存器组的数相关及基址值或变址值相关。这些相关同样也会出现在流水机器中。由于流

水是同时解释更多条指令，因此相关状况比重叠机器的更复杂、更严重，如果处理不当，就会显著降低流水效率。

如果流水机器的转移条件码是由条件转移指令本身或是由它的前一条指令形成的，则只有该指令流出流水线后才能建立转移条件，并依此决定下条指令的地址。那么从指令进入流水线、译码出它是条件转移指令直至它流出的整个期间，流水线就不能继续流入新指令。若转移成功且转向的目标指令又不在指令缓冲器内，还得要重新从访存取指令开始流动。

转移指令和其后的指令之间存在关联，使之不能同时解释，其造成的对流水机器的吞吐率和效率下降的影响要比指令相关、主存操作数相关和通用寄存器组相关及基址值和变址值相关严重得多，所以被称为全局性相关。而后者只影响相关的两条或几条指令，最多会使流水线某些段工作推后，不会改动指缓中预取到的指令，影响是局部的，所以被称为局部性相关。

1. 局部性相关的处理

指令相关、访存操作数相关和通用寄存器组相关等局部性相关都是由于在机器同时解释的多条指令之间出现了对同一主存单元或寄存器要求“先写后读”。重叠机器处理这些局部性相关的方法有两种。一种是推后后续指令对相关单元的读，直至在先的指令写入完成；另一种是设置相关直接通路，将运算结果经相关直接通路直接送入所需部件，不必先把运算结果写入相关单元，再从此相关单元取出来用，从而省去“写入”和“读出”两个访问周期，减少流水线的停等。由于流水是重叠的引申，因此上述这两种方法同样也适用于流水机器。问题是流水线有多个子过程，多条指令同时处在不同子过程上解释。如何判定流入流水线的多条指令之间是否相关，如何控制推后对相关单元的读，如何设置相关直接通路并控制相关直接通路的连通和断开，这些问题若解决不好，就会使控制机构变得极其复杂。

另外，任务在流水线中流动顺序的安排和控制可以有两种方式。一种是让任务(指令)流出流水线的顺序保持与流入流水线的顺序一致，称为顺序流动方式或同步流动方式。另一种是让流出流水线的任务(指令)顺序可以和流入流水线的顺序不同，称为异步流动方式。例如，有一个 8 段的流水线，其中第 2 段为读段，第 7 段为写段，如图 5 - 25 所示。有一串指令 h，i，j，k，l，m，n，…依次流入，当指令 j 的源操作数地址与指令 h 的目的操作数地址相同时，h 和 j 就发生了先写后读的操作数相关。顺序流动时，就要求 j 流到读段时必须停下来等待，直到 h 到达写段并完成写入后，才能继续向前流动，否则 j 将会读出不

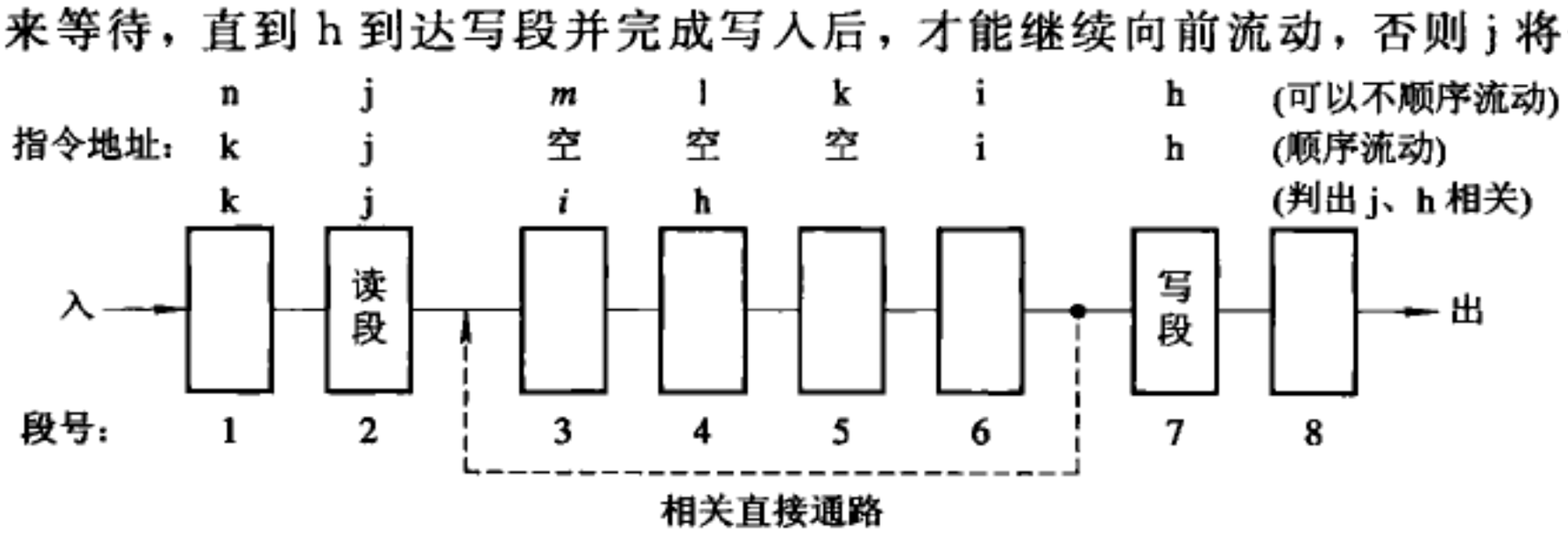


图 5 - 25　顺序流动和异步流动

是 h 写入后的内容而造成错误。这是一种推后对相关单元读的办法。由于 j 停下来，j 之后的指令也被迫停下来，从某个时期看，各功能段的工作情况如图 5 - 25 中“顺序流动”那行所示。采用顺序流动方式的好处是控制比较简单，ASC 机就采用这种方式，但相关后流水线的吞吐率和效率都要下降。

如果让 j 之后的那些指令，如 k，l，m，n 等，只要与 j 没有相关，就越过 j 继续向前流动，使得从某个时间看，指令在流水线内的流动顺序会如图 5 - 25 中“可以不顺序流动”那行所示，那么流水线的吞吐率和效率都未下降，这就是异步流动方式。实际上，产生异步流动还可能是因为某条指令经过的段数少或执行时间短，越过某些需段数多或执行时间长的指令向前流动等情形。

当流水线采用异步流动方式后，会出现顺序流动不会发生的其他相关。例如，若指令 i、k 都有写操作且是写入同一单元，该单元的最后内容本应是指令 k 的写入结果，但由于指令 i 执行时间长或有“先写后读”相关，出现指令 k 先于指令 i 到达“写段”，从而使得该单元的最后内容错为指令 i 的写入结果。我们称这种对同一单元要求在先的指令先写入，要求在后的指令后写入的关联为“写－写”相关。采用异步流动时，应在控制机构上保证，当发生“写－写”相关后，写入的先后顺序不变。另外，若指令 i 的读操作和指令 k 的写操作是对应于同一单元，指令 i 读出的本应是该单元的原存内容，但若指令 k 越过指令 i 向前流，且其写操作在指令 i 的读操作开始前完成，那指令 i 就会错误地读出指令 k 的写入结果。我们称这种对同一单元要求在先的指令先读出，在后的指令后写入的关联为“先读后写”相关。异步流动时，同样应能在控制机构上保证，当发生“先读后写”相关后，读、写的先后顺序不变。“写－写”相关和“先读后写”相关只有在异步流动时才有可能发生，同步流动时是不可能发生的。所以，采用异步流动方式工作，控制机构应能同时处理好这三种相关。

在流水线中同样可以通过设置相关直接通路来减少吞吐率和效率的损失。以“先写后读”相关为例，可以在写段和读段之间设置相关直接通路，如图 5 - 25 中虚线部分所示。一旦判出发生这种相关，就让通路接通，可以节省对相关单元的写入和重新读出这两个子过程时间，指令 j 就可以提前继续流动。但是，流水机器是同时解释多条指令的，并经常采用多个可并行工作的功能部件，如果在各功能部件之间为每种局部性相关都设置单独的相关直接通路，将会使硬件耗费大，控制复杂。因此，一般宜采用分布式控制和管理，并设置公共数据总线，以简化各种相关的判别和实现相关直接通路的连接。

下面介绍 IBM 360/91 浮点执行部件的相关处理控制机构。

IBM 360/91 的中央处理机是由指令处理部件、主存控制部件、定点执行部件和浮点执行部件组成的。指令处理部件以流水方式工作，每拍由主存取出一条指令送入指缓，同时按先进先出方式从指缓不断将指令取到指令寄存器，经预处理后分别送往定点或浮点执行部件的操作站缓冲。指缓最多可预取 8 条双倍字长指令。定点执行部件和浮点执行部件可以并行工作，分别将预处理过的指令从操作站以先进先出方式流水地取出译码和执行。

图 5 - 26 是 IBM 360/91 浮点执行部件的结构框图。浮点操作数缓冲器 FLB 接收和缓冲来自主存的操作数。要写入存储器的信息被送到存储数据缓冲器 SDB 中缓冲。浮点执行部件中的浮点加法器和浮点乘/除法器都是流水线，且能同时并行工作。

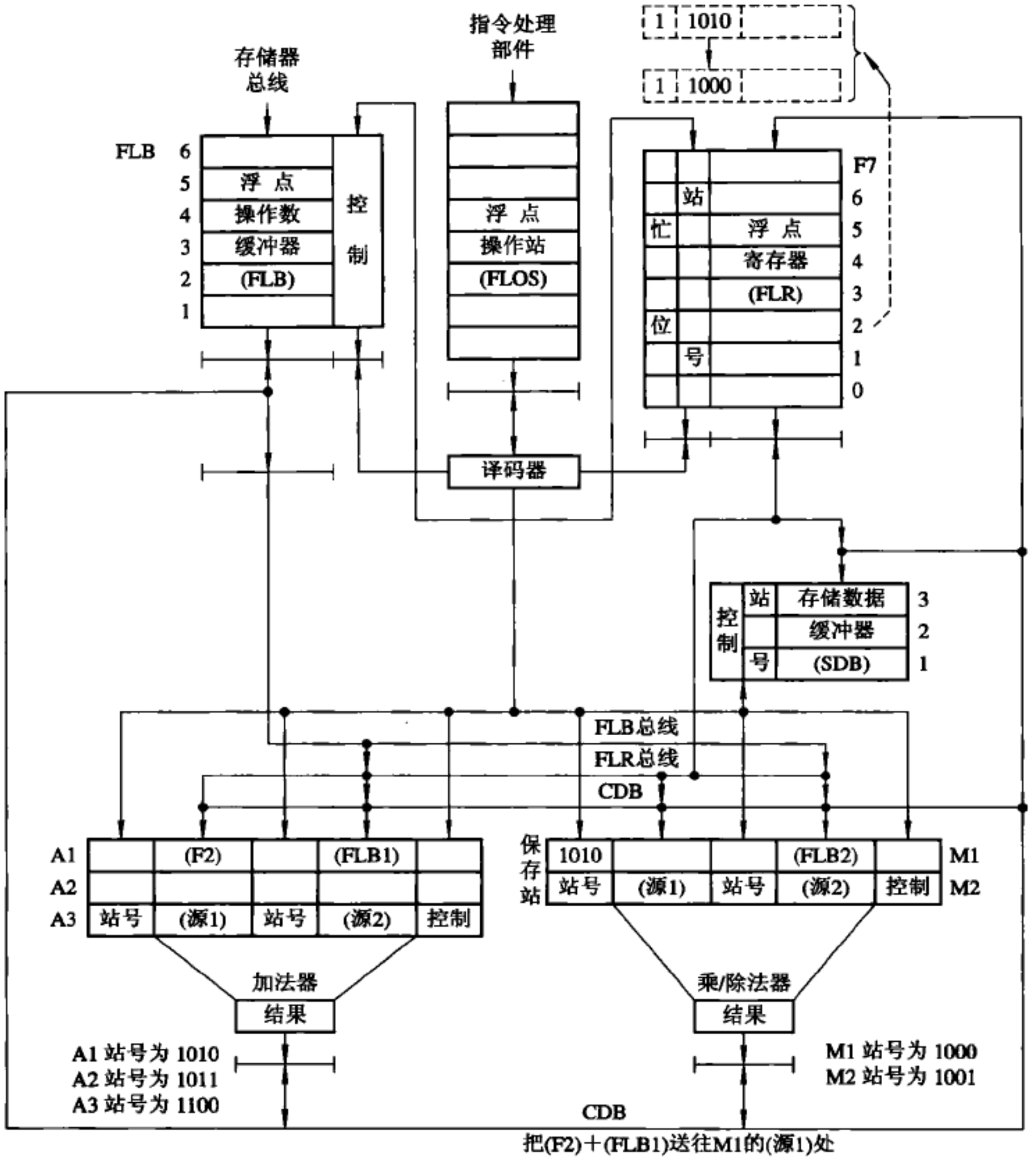


图 5 - 26　IBM 360/91 的浮点执行部件结构

浮点操作站 FLOS(Floating Point Operand Stack)缓冲的浮点操作命令格式为

操作　源 1(目的)，源 2

操作可以是浮点加、减、乘、除。源 1 指明存放源操作数的浮点寄存器 FLR 的号，并兼作存放中间结果的目的寄存器的号。源 2 指明存放经存储器总线送来的浮点操作数的缓冲器 FLB 的号。它们分别经 FLR 总线和 FLB 总线将数据送入浮点加法流水线或浮点乘/除法流水线输入端的保存站。浮点加法器流水线的输入端设有 3 个保存站 A1～A3，浮点乘/除法器流水线的输入端设有两个保存站 M1 和 M2，分别用规定的站号标记。保存站由控制部分控制，只要任意一个保存站的两个源操作数都到齐，且流水段空闲，就可以进入流水线向前流动。因此，它是采用异步流动方式工作的。

由于操作命令中源 1 兼作目的，因此，同时进入两条流水线的操作命令之间发生操作数相关的概率是较高的。设 $k+i$ 表示 k 之后同时在流水线流动的第 i 条指令，则只要 $k+i$

的源 1 与 k 的目的一样，就会发生“先写后读”相关；$k+i$ 的目的与 k 的源 1 一样，就会发生“先读后写”相关；$k+i$ 的目的与 k 的目的一样，就会发生“写一写”相关。也就是说，只要同时进入流水线的各个操作命令中使用了同一个浮点寄存器 FLR 的号，就会发生相关。

那么，怎样才能判别出相关呢？如果让进入流水线的各个操作命令将其源 1 和目的地址值逐个比较，分析是否有相关以及属于哪种相关，以便决定是否需要推后以及让谁推后处理，显然是很麻烦的。IBM 360/91 采用给每个浮点寄存器 FLR_i 设置一个“忙位”来判别相关。只要某个浮点寄存器 FLR_i 正在使用，就将其“忙位”置为“1”，一旦使用完，立即将其置成“0”。因此，若某个操作命令需要用 FLR_i 寄存器，就先看其“忙位”是否为“1”，若为“1”就表示发生了相关。通过设置保存站及“站号”字段和在相关后更改站号就可以推后处理及控制相关直接通路的连接。公共数据总线 CDB(Common Data Bus)就是一种总线方式的相关直接通路，可以为多种和多个不同相关所共用，通过给出不同站号来控制其不同连接。

现在，以 FLOS 依次送出

```
ADD F2, FLB1    ;(F2)+(FLB1)→F2
MD F2, FLB2     ;(F2) * (FLB2)→F2
```

这两条操作命令为例，说明怎样判别出发生相关以及怎样控制推后和相关直接通路的连接。很明显，当这两条命令异步流动时，“先写后读”、“写一写”、“先读后写”这三种相关都会发生。

当 FLOS 送出

```
ADD F2, FLB1
```

操作命令时，它控制由 FLR 取得(F2)，由 FLB 取得(FLB1)送往加法器保存站，例如，送往 A1，同时立即将 F2 的“忙位”置“1”，以指明该寄存器的内容已送往保存站等待运算，这样 F2 的内容再不能被其他操作命令作源操作数读出用。由于 F2 这时已成为“目的”寄存器，准备接收由加法器传来的运算结果，因此，将 F2 的“站号”字段置成是 A1 的站号“1010”，以便控制把站号为 1010 的保存站 A1 在加法流水线流出的结果经 CDB 总线送回 F2。一旦结果送回，即将 F2 的“忙位”和“站号”都置成“0”，以释放出 F2 为别的操作命令所用。

问题是当 F2 的“忙位”为“1”，而加法结果未从加法流水线流出时，FLOS 又送出操作命令

```
MD F2, FLB2
```

由译码控制去访问 F2 取源 1 操作数时，由于其“忙位”为“1”，表明出现了 F2 相关，此时就不能直接将 F2 送往乘法器保存站，而改为把原存在 F2 的“站号”字段中的站号 A1(即 1010，指明 F2 应有内容的来源)送往 M1 的“源 1 站号”，并把 F2 内的站号由 A1(1010)改为 M1(1000)，以指明应改为从 M1 接收运算结果。这一过程如图 5 - 26 右上方虚线所示。这样，当加法器对 A1 站的(源 1)、(源 2)进行相加，经 CDB 送出结果时，就不是送到 F2，而是直接送进 M1 保存站中站号为 A1(1010)的(源 1)部分，相当于接通相关直接通路。而乘法器是在 M1 的(源 1)、(源 2)都有了内容后，才进入乘法流水线的，相当于推后相乘的执行。乘积则经 CDB 送往“站号”为 M1(1000)的 F2 寄存器。

在加法器和乘/除法器输入站设置多个保存站，是为了使这些运算部件可在某个操作

命令或因相关需推后执行，或因执行时间过长而尚未完成时，仍能继续从 FLOS 接收操作命令，因此，它是以异步流动方式工作的。

结论：标量流水机对局部性相关的处理一般采用总线式分布方式控制管理，具体管理包括：一是相关的判断主要靠分布于各寄存器的“忙位”标志来管理；二是在分散于各流水线的入、出端处设置若干保存站来缓存信息；三是用站号控制公共数据总线的连接作相关专用通路，使之可为多个子过程的相关所共用；四是一旦发生相关，用更换站号来推后和控制相关专用通路的连接；五是采用多条流水线，每条流水线入端有多组保存站，以便发生相关后，可以采用异步的流动方式。

采用分布式控制方式大大简化了同时出现多种相关及多重相关的处理。它要比集中式(如 CDC－6600)灵活，且处理能力强。因此，大多数流水机器都采用类似于IBM 360/91 的分布式控制方式。

2. 全局性相关的处理

全局性相关指的是已进入流水线的转移指令(尤其是条件转移指令)和其后续指令之间的相关。下面介绍一些常用的处理方法。

(1) 使用猜测法。若指令 i 是条件转移指令，有两个分支，如图 5－27 所示。一个分支是 $i+1$，$i+2\cdots$，按原来的顺序执行下去，称为转移不成功分支。另一个分支是转向 p，$p+1$ …，称为转移成功分支。流水是同时解释多条指令，当指令 i 进入流水线，后面是进入 $i+1$ 还是 p，只有等条件码建立后，即条件转移指令快流出流水线时才能知道。如果此期间让 i 之后的指令停等，流水线就会断流，性能将急剧下降。在标量类机器指令程序中，条件转移指令约占 20%，其中 60%为转移成功。在指令流足够长时，这种条件转移会使流水性能下降 50%。为了在执行条件转移指令时不断流，可采用猜测法猜取 $i+1$ 和 p 两个分支之一继续向前流动。

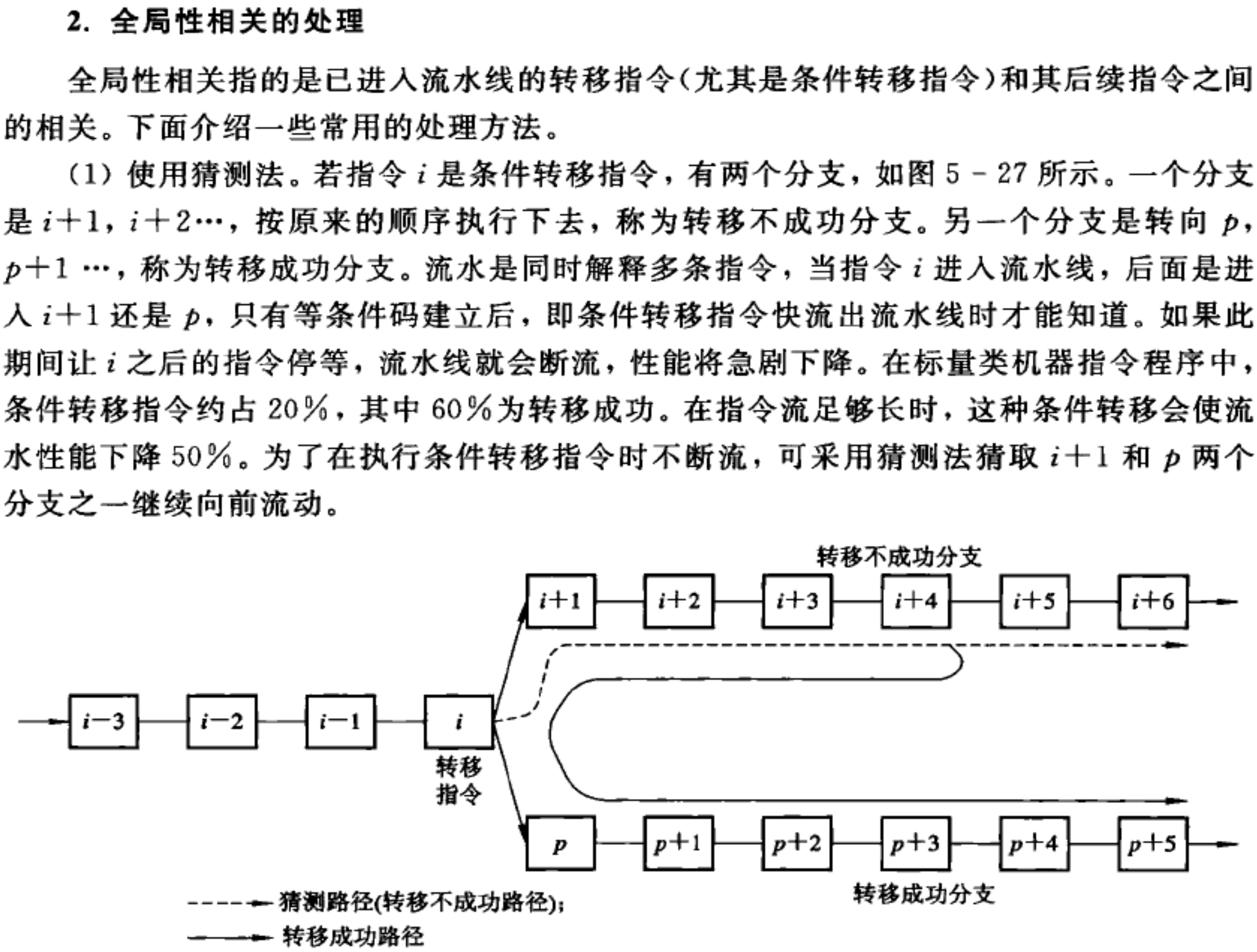


图 5－27　用猜测法处理条件转移

那么猜选哪个分支好呢？如果两个分支概率相近，宜选转移不成功分支，因为它已预取进指缓，可以很快从中取出进入流水线而不必等待。如果猜选转移成功分支，指令 p 很可能不在指缓中，需花较长时间访存去取，使流水线实际上断流。IBM 360/91 猜选的就是转移不成功分支。现假设指令 i 所用条件码是在 $i+4$ 流入流水线时才建立的，若条件码对应于转移不成功分支，就猜对了，可继续流下去；若条件码对应于转移成功分支，就猜错了，这时需对 $i+1$，$i+2$，$i+3$，$i+4$ 已有的解释作废，重新回到原分支点，沿转移成功分

支去解释 p，$p+1$ …，使流水线的吞吐率和效率都下降。但是，只要猜测法猜对的机会占大多数，流水线的吞吐率和效率就会比不用猜测法的要高得多。因此，当转移的两个分支概率不均等时，宜猜高概率分支。转移概率可以静态地根据转移指令类型或程序执行期间转移的历史状况来预测，但需要事先对大量程序的转移类型和转移概率进行统计，且统计出的概率也不一定能保证较高的猜测准确度。如果采用动态策略，由编译程序根据执行过程中转移的历史记录来动态预测未来的转移选择，可使预测准确度提高到 90%。

采用猜测法时应能保证猜错时可恢复分支点原先的现场，一般有三种办法。例如指令 $i+2$ 的功能是(R1)+(N)→N，如果全部解释完，N 的内容被修改，原有现场就需要恢复。因此在机器沿猜测分支解释时，应当与正常情况下的指令解释不同。例如 IBM 360/91 采取对指令只译码和准备好操作数，在转移条件码出现之前不进行运算。另一种是让它运算完但不送回运算结果，有的机器就是如此。然而早期所用的这两种办法不方便，因为若猜对后还要让这些指令继续完成余留的操作。随着器件价格、体积、技术的发展，已经可以让它们和正常情况一样，不加区别地全部解释完。只要把可能被破坏的原始状态都用后援寄存器保存起来，一旦猜错，就取出后援寄存器的内容来恢复分支点的现场即可。这些后援寄存器实际不是单独为流水线设置的，因为在出于提高系统可靠性，实现指令复执、程序卷回目的时，就已设置了这些后援寄存器。一般猜对的概率要高，猜对后既不用恢复，也不用再花时间去完成余留的操作。因此，采用后援寄存器法的实现效率会更高一些。

为了在猜错时能尽快回到原分支处转入另一分支，在沿猜测路径向前流动的同时，还可由存储器预取转移成功分支的头几条指令(IBM 360/91 是预取两条双字长指令字)，放在转移目标指令缓冲器中，以便在猜错时不必从访存取 p 开始，减少流水线的等待时间。IBM 3033 机的指令流水线除了设正常指令缓冲器外，还设了两套转移目标指令缓冲器，这样可为相邻近两条条件转移指令所分别使用。

(2) 加快和提前形成条件码。尽快、尽早获得条件码，以便提前知道流向哪个分支，会有利于流水机器简化对条件转移的处理。这可以从以下两方面采取措施。

一方面是加快单条指令内部条件码的形成，不等指令执行完就提前形成反映运算结果的条件码。例如，乘、除结果是正、是负还是零的条件码可在运算前形成。只要两个操作数符号位相同就是正，符号位相反就是负。相乘时有一个操作数为零或除法时被除数为零，则结果为零。由于相乘、相除操作时间较长，条件码提前形成对加速条件转移的处理大有好处。Amdahl 470 V/6 就按此思路，在流水运算器输入端设置 LUCK 部件，可以对大多数指令预判出它们的条件码，从而在具体运算前就能将运算结果的条件码送到指令分析部件。

另一方面是在一段程序内提前形成条件码，这特别适合于循环型程序在判断循环是否继续时的转移情况。例如 FORTRAN DO 循环，每当执行到循环终端语句时，总要对循环次数减 1，如果减下来的结果为 0 就跳出循环，否则转回去执行循环体，这通常用减 1(DEC)和不等于零条件转移(NE)两条指令来实现。为了使不等于零条件转移指令的条件码能提前形成，可以将减 1 指令提前到与其不相关的其他指令之前，甚至提前到循环体开始时进行。这样，执行到不等于零条件转移指令时，减 1 指令的条件码早已形成，马上就能知道是否需要转移，不致因等待条件码形成而使流水线的吞吐率和效率下降。不过，为此需要在硬件上增设为循环减 1 指令专用的条件码寄存器 CC_s，以便通用的条件码寄存器

CC 仍可为其他指令使用。

（3）采取延迟转移。这是用软件方法进行静态指令调度的技术，不必增加硬件，在编译生成目标指令程序时，将转移指令与其前面不相关的一条或多条指令交换位置，让成功转移总是延迟到在这一条或多条指令执行之后再进行。这样，可使转移造成的流水性能损失减少到 0。

（4）加快短循环程序的处理。采用这种处理，一是可以将长度小于指缓容量的短循环程序整个一次性地放入指缓内，并暂停预取指令，避免执行循环时由于指令预取导致指缓中需循环执行的指令被冲掉，减少主存重复取指的次数；二是由于循环分支概率高，因此，让循环出口端的条件转移指令恒猜循环分支，减少因条件分支造成流水线断流的机会。例如，IBM 360/91 为上述第一点设置了“向后 8 条”检查的硬件，当转向去址往回走且与条件转移指令之间相隔不超过 8 条指令时，将其间的指令全部移入指缓并停止预取新指令；为上述第二点设置了“循环方式”工作状态，使出口端的条件转移指令指向循环程序的始端。采取这些措施后可使循环时流水加快 1/3～3/4。

有的机器还采取顺序执行时，让预取的指令既放入正常使用的指令缓冲器，也放入转移目标指令缓冲器中的做法。一旦检测出是循环，可把转移目标指令缓冲器的内容作为短循环程序控制用，省去了第一次循环时重新从主存中取此短循环程序中指令的操作开销。还有的机器允许将这两种指令缓冲寄存器连接起来使用，使更大的循环程序也能得到加快处理。

3. 流水机器的中断处理

中断会引起流水线断流，但其出现概率比条件转移的概率要低得多，且又是随机发生的。所以，对于流水机器的中断，主要应考虑如何处理好断点现场的保存和恢复，而不是如何缩短流水线的断流时间。

在执行指令 i 时有中断，断点本应是在指令 i 执行结束，指令 $i+1$ 尚未开始执行的地方，但流水机器是同时解释多条指令的，指令 $i+1$，$i+2$，…可能已进入流水线被部分解释。对于异步流动流水线，这些指令中有些可能已流到指令 i 的前面去了。

早期的流水机器，如 IBM 360/91，为简化中断处理采用“不精确断点”法，即不论指令 i 在流水线的哪一段发生中断，未进入流水线的后续指令不再进入，已在流水线的指令仍继续流完，然后才转入中断处理程序。这样，断点就不一定是 i，可能是 $i+1$，$i+2$ 或 $i+3$ …，即断点是不精确的。仅当指令 i 在第 1 段响应中断时，断点才是精确的。“不精确断点”法不利于编程和程序的排错。

后来的流水机器多数采用“精确断点”法，如 Amdahl 470 V/6。不论指令 i 是在流水线中哪一段响应中断，给中断处理程序的现场全都是对应 i 的，i 之后流入流水线的指令的原有现场都能保存和恢复。“精确断点”法需设置很多后援寄存器，以保证流水线内各条指令的原有现场都能被保存和恢复。如前所述，这些寄存器也是“指令复执”所必需的。

4. 非线性流水线的调度

由于线性流水线在执行每个任务（指令、操作）的过程中，各段均只通过一次，因此，每拍都可以将一个新的任务送入流水线，这些任务不会争用同一个流水线。非线性流水线则不同，因为段间设置有反馈回路，一个任务在流水的全过程中，可能会多次通过同一段

或越过某些段。这样，如果每拍向流水线送入一个新的任务，将会发生多个任务争用同一功能段的使用冲突现象。要想不发生冲突就得间隔适当的拍数之后再向流水线送入下一个任务。究竟间隔几拍送入下一个任务，才既不发生功能段使用冲突，又能使流水线有较高的吞吐率和效率，是流水线调度要解决的问题。流水线调度对多功能动态流水线同样也是重要的，否则功能切换时也会发生功能部件使用冲突。

现介绍单功能非线性流水线的任务调度。

为了对流水线的任务进行优化调度和控制，1971 年 E. S. Davidson 等人提出使用一个二维的预约表(Reservation Table)。

如果有一个由 K 段组成的单功能非线性流水线，每个任务通过流水线需要 N 拍。利用类似画时空图的方法可以得到该任务使用流水线各段的时间关系表(即预约表)。其中拍号 n 为任务经过流水线的时钟节拍号。如果任务在第 n 拍要用到第 k 段，就在相应第 n 列第 k 行的交点处用 √ 表示。

现设流水线由 5 段组成，段号 k 分别为 1～5，任务经过流水线总共需 9 拍，其预约表如图 5 - 28(a)所示。

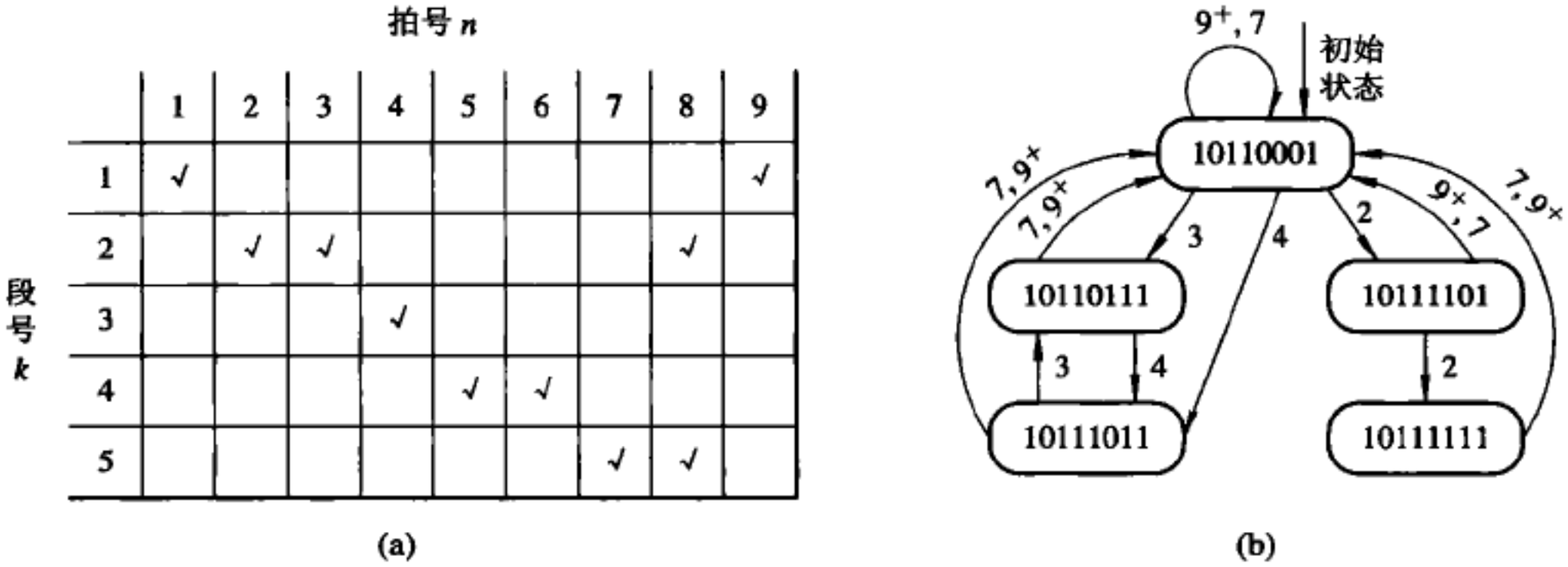


图 5 - 28　流水线预约表及状态图举例

(a) 单功能流水线预约表举例；(b) 单功能流水线的状态转移示意

根据预约表可以很容易地得出一个任务使用各段所需间隔的拍数(也称为延迟)。例如，1 段相隔 8 拍，2 段相隔有 1、5 和 6 拍三种。那么两个任务相隔 8 拍流入流水线必将会争用 1 段，而相隔 1、5 或 6 拍流入流水线必将会争用 2 段。将流水线中所有各段对一个任务流过时会争用同一段的节拍间隔数汇集在一起，便构成一个延迟禁止表 F(Forbidden List)。如本例为{1，5，6，8}。就是说，要想不争用流水线的功能段，相邻两个任务送入流水线的间隔拍数就不能为 1、5、6、8 拍，这些间隔拍数应当禁止使用。可以用一个有 $N-1$ 位的位向量来表示后续新任务间隔各种不同拍数送入流水线时，是否会发生功能段使用的冲突，称此位向量为冲突向量 C(Collison Vector)。冲突向量($c_{N-1}\cdots c_i\cdots c_2c_1$)中第 i 位的状态表示与当时相隔 i 拍给流水线送入后续任务是否会发生功能段的使用冲突。如果不会发生冲突，令该位为“0”，表示允许送入；否则让该位为“1”，表示禁止送入。冲突向量取 $N-1$ 位是因为经 N 拍后，该任务已流出流水线，不会与后续的任务争用流水功能段了。

不难看出，输入后续任务还需等待的拍数与前一个任务在流水线已进行了几拍有关。当第一个任务第 1 拍送入流水线时，根据禁止表 F={1，5，6，8}，可以形成此时的冲突向

量 $\boldsymbol{C}$(10110001)，称此为刚流入流水线时的初始冲突向量。由于初始冲突向量的 c_2、c_3、c_4、c_7 为 0，因此第二个任务可以距第一个任务 2、3、4 或 7 拍流入流水线。当第二个任务流入流水线后，应当产生新的冲突向量，以便决定第三个任务可以相隔多少拍流入流水线而不会与已进入流水线的前面第一、第二个任务争用功能段。依此顺序类推。显然，随着流水线中第一个任务每拍向前推进一段，原先禁止第二个任务流入流水线的各种间隔拍数均相应减去一拍。这意味着可将初始冲突向量放在一个移位器中，每拍逻辑右移 1 位，让左面移空的位补“0”，以表示如果间隔 8 拍以上流入后续任务时，前一个任务必定已流出流水线，不会发生功能部件使用冲突。因此，随着任务在流水线中的推进，会不断动态地形成当时的冲突向量。

如果选择第二个任务在间隔 2 拍时流入流水线，对第一个任务而言初始冲突向量右移 2 位成了(00101100)。这样，要想使第三个任务流入流水线后，既不与第一个任务发生功能段冲突，也不与第二个任务发生功能段冲突，新的冲突向量就应当是第一个任务当前的冲突向量(00101100)与第二个任务的初始冲突向量(10110001)的按位“或”，其结果为(10111101)。也就是说，第三个任务只能在第二个任务流入流水线后隔 2 拍或 7 拍流入才不会与先前流入流水线的那些任务争用功能段。按此思路选择各种可能的间隔拍数流入新任务，从而又产生新的冲突向量，一直进行到不再产生不同的冲突向量为止。由此可画出用冲突向量表示的流水线状态转移图。图中两个冲突向量之间用有向弧上的数字表示引入后续任务产生新的冲突向量所用的间隔拍数。本例的流水线状态转移图如图5－28(b)所示。图中，9^+ 表示大于或等于 9 拍进入后续任务也不会发生冲突。

因此，只要按流水线状态图中由初始状态出发，构成一种调度间隔延迟拍数呈周期性重复的方案来进行流水线的调度，都不会发生功能段使用冲突。要想找出一种最佳的调度方案使流水线的吞吐率最高，只要计算出每种调度方案的平均间隔拍数，从中找出其最小者即可。

表 5－1 给出了本例中各种调度方案的平均间隔拍数(平均延迟)。由表 5－1 可知，采用先隔 3 拍后隔 4 拍轮流给流水线送入任务的调度方案是最佳的，平均每隔 3.5 拍即可流入一个任务，吞吐率最高。尽管(4，3)调度方案平均间隔拍数也是 3.5 拍，但若实际流入任务数是循环所需任务数的整数倍，则其实际吞吐率相对会低些，所以不作为最佳调度方案。这是一种不等间隔的调度策略，比起相等间隔的调度策略在控制上要复杂一些。

表 5－1　各种调度方案的平均间隔拍数的例子

调度方案	平均间隔拍数	调度方案	平均间隔拍数
(2，2，7)	3.67	(3，7)	5.00
(2，7)	4.50	(4，3，7)	4.67
(3，4)	3.50	(4，7)	5.50
(4，3)	3.50	(7)	7.00
(3，4，7)	4.67		

为了简化控制也可以采用相等间隔调度，不过这常会使吞吐率和效率下降。例如本例只有一种相等间隔调度方案，当然应取其中间隔最小的，这样会使吞吐率的下降最少。

【例 5-3】 在一个 4 段的流水线处理机上需经 7 拍才能完成一个任务，其预约表如表 5-2 所示。

表 5-2　7 拍才能完成一个任务的预约表

段 \ 时间	1	2	3	4	5	6	7
S_1	✓				✓		✓
S_2		✓		✓			
S_3			✓				
S_4				✓		✓	

分别写出延迟禁止表 F、冲突向量 C；画出流水线状态转移图；求出最小平均延迟及流水线的最大吞吐率及其调度时的最佳方案。按此调度方案，输入 6 个任务，求实际的吞吐率。

此例可得延迟禁止表 $F=\{2, 4, 6\}$。

初始冲突向量 $C=(101010)$。

状态转移图如图 5-29 所示。

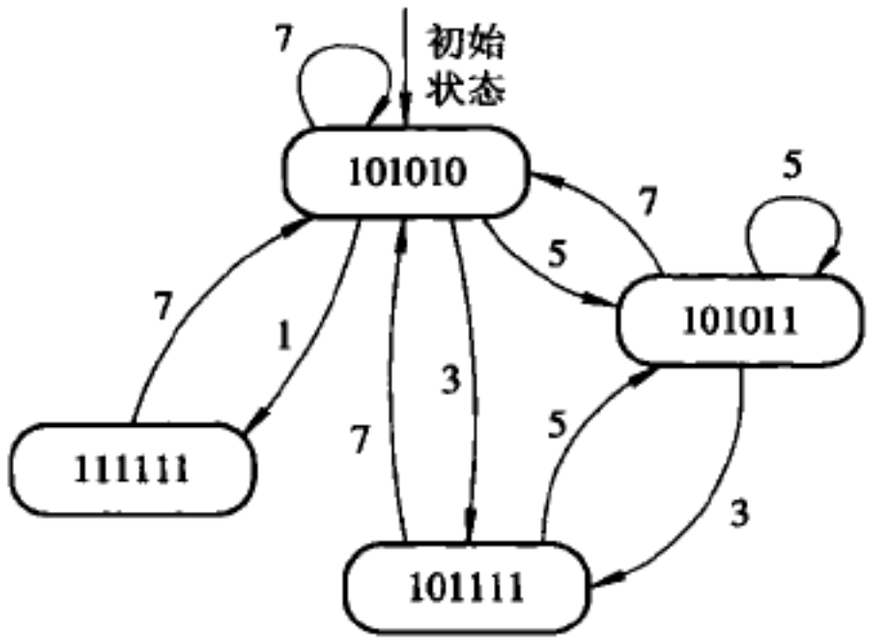


图 5-29　例 5-3 的状态转移示意图

各种调度方案及其相应的平均延迟如表 5-3 所示。

表 5-3　调度方案及其相应的平均延迟

调度方案	平均延迟/拍
(1, 7)	4
(3, 5)	4
(5, 3)	4
(5)	5

由表 5-3 可知，最小平均延迟为 4 拍。

此时流水线的最大吞吐率 $T_{p_{max}}=1/4$(任务/拍)。

最佳调度方案宜选其中按(1, 7)周期性调度的方案。

按(1, 7)调度方案输入 6 个任务，全部完成的时间为 1+7+1+7+1+7=24(拍)，实

际吞吐率 $T_p=6/24$(任务/拍)。

若按(3，5)调度方案输入 6 个任务，全部完成的时间为 3＋5＋3＋5＋3＋7＝26(拍)，实际吞吐率 $T_p=6/26$(任务/拍)。

若按(5，3)调度方案输入 6 个任务，全部完成的时间为 5＋3＋5＋3＋5＋7＝28(拍)，实际吞吐率 $T_p=6/28$(任务/拍)。

可见，最佳的方案应为(1，7)调度方案，输入 6 个任务的实际吞吐率较之其他方案要更高些。

很容易想到，可以通过找出最小的平均间隔拍数这种思路来优化流水线的性能。因为预约表中打√最多的行是流水线性能的瓶颈，所以其√的个数实际上限定了流水线可达到的最短的平均间隔拍数。如图 5－28 所示的预约表中，第二行有 3 个√，其最短的平均间隔拍数应为 3 拍，但按(3，4)方案调度，其平均间隔拍数实际为 3.5 拍，并不是最佳的，可以进行改进。流水线之所以未达到最佳性能是因为发生了功能段争用。如果在流水线中某些段增设延迟线，在满足流水功能前提下，将某些√向右移，移到新的列位置上，修改预约表，从而使初始冲突向量和状态转移图发生变化，就可以产生具有理想的最佳平均间隔拍数的周期性调度方案。虽然增加延迟线会延长每个任务通过流水线的总时间，但调度方案中平均间隔拍数的减少却可使流水线的吞吐率和效率得以改进。

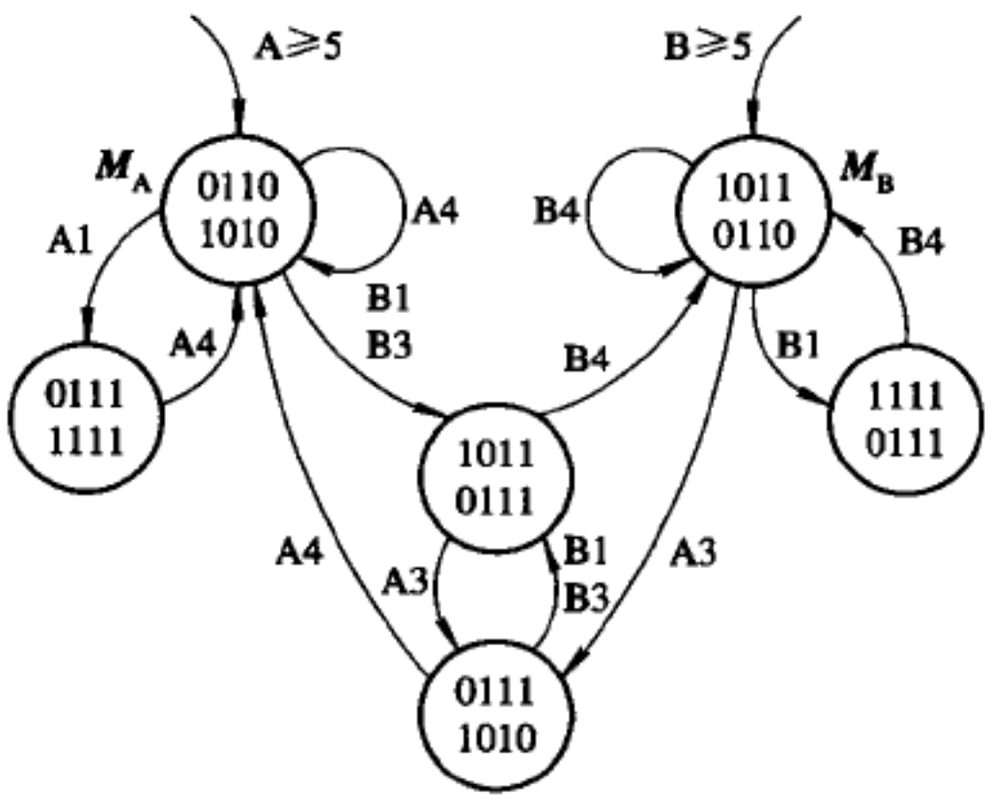
以上只是结合单功能流水线讨论了有关流水线调度的基本思想和方法。在此基础上，不难解决多功能流水线的调度。

对于一个多功能流水线，只需要将对应每种功能的预约表都重叠在一起即可。这里仅举一个二功能动态流水线的例子来简要说明其思路。图 5－30(a)为一个 A、B 两种功能重叠的预约表例子。

拍号 n

段号 k	1	2	3	4	5
1	A	B		A	B
2		A		B	
3	B		AB		A

(a)



(b)

图 5－30　多功能流水线预约表及状态图举例

(a) 一个二功能流水线的预约表；(b) 状态图

使用交叉冲突向量(Cross-collision Vector)来反映有 A、B 两种功能的动态流水线各个后继任务流入流水线所禁止使用的间隔拍数。这样，对于本例就应有 4 个交叉冲突向量，即 $\boldsymbol{V}_{AB}=(1011)$，$\boldsymbol{V}_{BA}=(1010)$，$\boldsymbol{V}_{AA}=(0110)$，$\boldsymbol{V}_{BB}=(0110)$。其中，$\boldsymbol{V}_{AA}$ 和 $\boldsymbol{V}_{BB}$ 分别表示都按 A 功能和 B 功能流水时，后继任务流入流水线的冲突向量；而 $\boldsymbol{V}_{AB}$ 表示先前按 B 功能流

水流入的任务与后继按 A 功能流水流入的任务之间的冲突向量，$\boldsymbol{V}_{BA}$则表示先前按 A 功能流水流入的任务与后继按 B 功能流水流入的任务之间的冲突向量。

就一般情况而言，一个有 P 个功能的流水线将有 P^2 个交叉冲突向量，它可以分别归类写成 P 个冲突矩阵$\boldsymbol{M}_p$，其中 p 分别为 1 至 P。冲突矩阵 $\boldsymbol{M}_p$ 表示按 p 功能流水线进入一个任务后与按各种功能流水线流入后继任务所产生的全部冲突向量的集合。对本例来说有两个初始冲突矩阵，分别为

$$\boldsymbol{M}_A = \begin{bmatrix} 0110(AA) \\ 1010(BA) \end{bmatrix}, \quad \boldsymbol{M}_B = \begin{bmatrix} 1011(AB) \\ 0110(BB) \end{bmatrix}$$

其中，$\boldsymbol{M}_A$ 表示按 A 功能流水刚流入一个任务后与按其他功能(本例是 A 功能和 B 功能)流水流入后继任务的所有禁止间隔拍数；$\boldsymbol{M}_B$ 则表示按 B 功能流水刚流入一个任务后与按其他功能(本例是 A 功能和 B 功能)流水流入后继任务的所有禁止间隔拍数。

P 个功能流水线的调度控制同单个功能流水线的基本相似，不同的是用 P 个(本例是两个)移位寄存器分别存放冲突矩阵的各行，让它们每拍同时右移 1 位，左面的移空位补以 0，并与后继任务相应功能的初始冲突矩阵对应行进行按位“或”，形成新的冲突矩阵。同样，可以画出用相应的冲突矩阵表示的流水线状态转移图。本例的流水线状态图如图 5 - 30(b)所示。

例如，按 A 功能刚流入一个任务后，根据 $\boldsymbol{V}_{AA}$为(0110)，知道可隔 1 拍或 4 拍流入一个 A 功能的新任务。将 $\boldsymbol{M}_A$ 初始冲突矩阵各行同时右移 1 位，再与 A 功能的初始冲突矩阵 $\boldsymbol{M}_B$ 对应行按位“或”，形成新的冲突矩阵$\begin{bmatrix} 0111 \\ 1111 \end{bmatrix}$。根据此时 $\boldsymbol{V}_{AA}$的(0111)，知道只有隔 4 拍流入一个 A 功能的新任务才能不发生冲突，从而形成在此基础上的新的冲突矩阵$\begin{bmatrix} 0110 \\ 1010 \end{bmatrix}$。

再如，根据初始冲突矩阵中的 $\boldsymbol{V}_{BA}$为(1010)，知道可在第一拍或第三拍进行 B 功能的新任务的送入而不发生冲突，于是将 $\boldsymbol{M}_A$ 初始冲突矩阵均右移 1 位或 3 位，再与 $\boldsymbol{M}_B$ 的初始冲突矩阵对应行按位“或”，形成新的冲突矩阵，它们形成的新冲突矩阵恰好都为$\begin{bmatrix} 1011 \\ 0111 \end{bmatrix}$。据此可知，或者是隔 3 拍流入 A 功能的新任务，或者是隔 4 拍流入 B 功能的新任务，又将分别产生不同的新的冲突矩阵。

在图 5 - 30(b)的状态图中，两个冲突矩阵间的有向弧上的功能标记和数字表示按该种功能(本例中用 A 或 B 标志)送入后续新任务所需间隔的拍数。根据状态图，同样可求得所有各种控制调度策略中平均拍数最小的最佳调度方案。由此可见，多功能流水线的调度实际上是单功能流水线调度的扩展。

5.3 指令级高度并行的超级处理机

自从 20 世纪 80 年代兴起 RISC 之后，出现了指令级高度并行的高性能超级处理机，它使单处理机在每个时钟周期里可解释多条指令。有代表性的例子是超标量(Superscalar)处理机、超长指令字(VLIW)处理机、超流水线(Superpipelining)处理机和超标量超流水线处理机。

5.3.1 超标量处理机

假设一条指令包含取指令、译码、执行、存结果四个子过程，每个子过程经过时间为 Δt。常规的标量流水线单处理机是在每个 Δt 期间解释完一条指令，如图 5 - 31 所示。执行完 12 条指令共需 $15\Delta t$。称这种流水机的度 $m=1$。

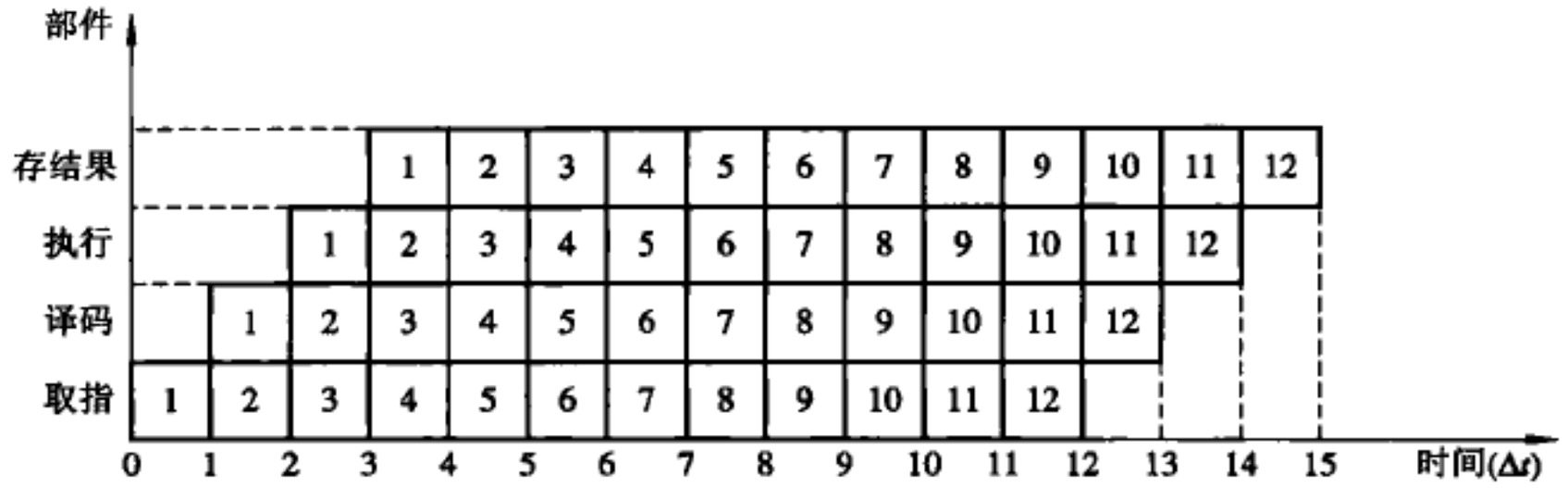


图 5 - 31　常规(度 $m=1$)的标量流水机时空图

超标量处理机采用多指令流水线，每个 Δt 同时流出 m 条指令(称为度 m)。假如度 m 为 3 的超标量处理机流水时空图如图 5 - 32 所示，每 3 条指令为一组，执行完 12 条指令只需 $7\Delta t$。

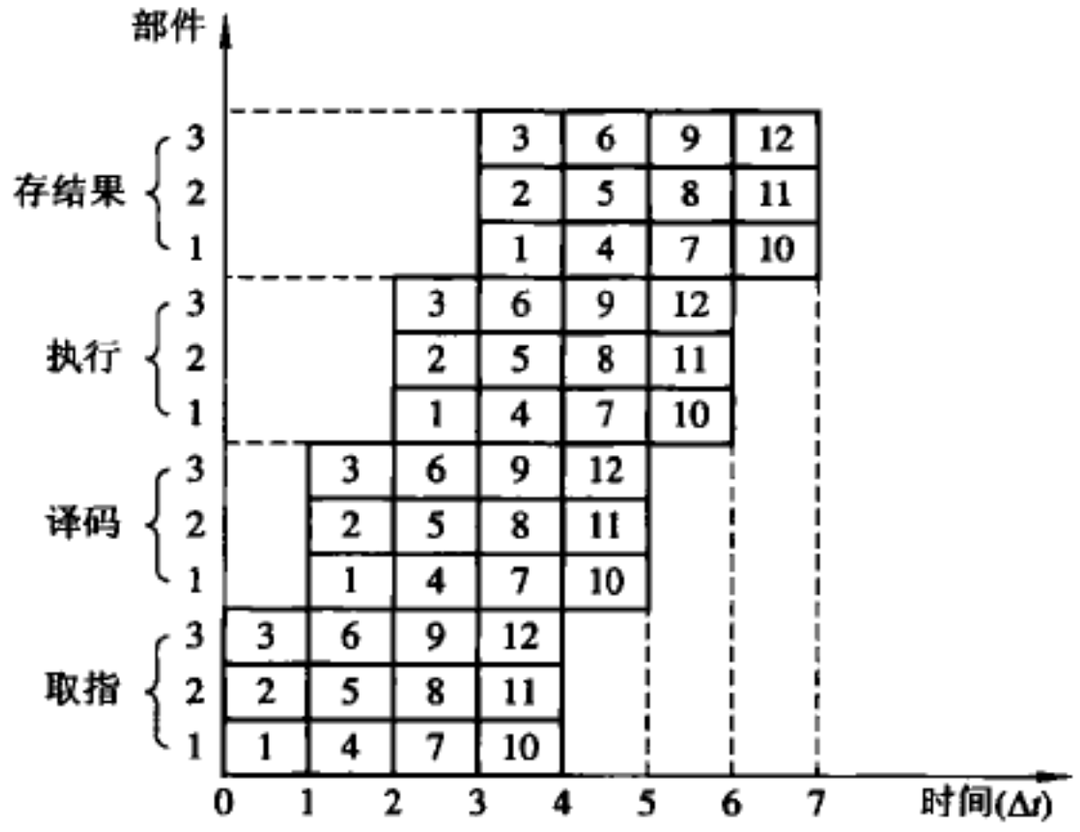


图 5 - 32　度 $m=3$ 的超标量处理机时空图

在超标量流水线处理机中配置有多套功能部件、指令译码电路和多组总线，寄存器也备有多个端口和多组总线。程序运行时由指令译码部件检测顺序取出的指令之间是否存在数据相关和功能部件争用，将可并行的相邻指令送往流水线。若并行度为 1，就逐条执行。超标量流水机主要靠编译程序来优化编排指令的执行顺序，将可并行的指令搭配成组，不让硬件调整指令顺序，这样实现起来比较容易些。

超标量流水处理机非常适合于求解像稀疏向量或稀疏矩阵这类标量计算问题，因为它们用向量流水线处理机求解很不方便。

【例 5 - 4】 典型的超标量流水机有 IBM RS/6000、Power PC 601、DEC 21064、Power PC 620、Intel i960CA、Pentium、Tandem Cyclone、SUN Ultra SPAC 和 Motorola MC 88110 等。1986 年的 Intel i960CA 时钟频率为 25 MHz，度 $m=3$，有 7 个功能部件可以并发使用。1990 年的 IBM RS/6000 时钟频率为 30 MHz，处理机中有转移处理、定点、浮点

三种功能部件可并行工作，转移处理部件每 Δt 可执行多达 5 条指令，度 $m=4$，性能可达 34 MIPS 和 11 MFLOPS，非常适合于在数值计算密集的科学工程上应用及在多用户商用环境下工作。许多基于 RS/6000 的工作站和服务器都是 IBM 生产的，如 POWER Station 530。1992 年的 DEC 21064 时钟频率为 150 MHz，度 m 为 2，10 段流水线，最高性能可达 300 MIPS 和 150 MFLOPS。Tandem 公司的 Cyclone(旋风)计算机由 4～16 台超标量流水处理机组成，每个处理机的寄存器组有 9 个端口(其中 5 个为读，4 个为写)，有 2 个算术逻辑部件，度 $m=2$。

由于程序中指令并行性的开发有限，因此超标量处理机的度 m 比较低。

5.3.2 超长指令字处理机

超长指令字(VLIW)结构是将水平型微码和超标量处理两者相结合，指令字长可达数百位，多个功能部件并发工作，共享大容量寄存器堆。与超标量处理机不同的是，VLIW 处理机在编译时，编译程序找出指令间潜在的并行性，将多个功能并行执行的不相关或无关的操作先行压缩组合在一起，形成一条有多个操作段的超长指令，运行时不再用软件或硬件来检测其并行性，而直接由这条超长字指令控制机器中多个相互独立的功能部件并行操作。每个操作段控制其中的一个功能部件，相当于同时在执行多条指令。因此其硬件结构和指令系统简单，是一种单指令多操作码多数据的系统结构(SIMOMD)，不同于 SIMD 计算机。图 5－33(a)示意出了典型的 VLIW 处理机的组成和指令格式。图 5－33(b)示意出了度 $m=3$ 时的流水时空图，经过 $7\Delta t$ 后得到 4×3 个结果。这种结构最先是 1983 年美国耶鲁大学 Fisher 教授提出的。

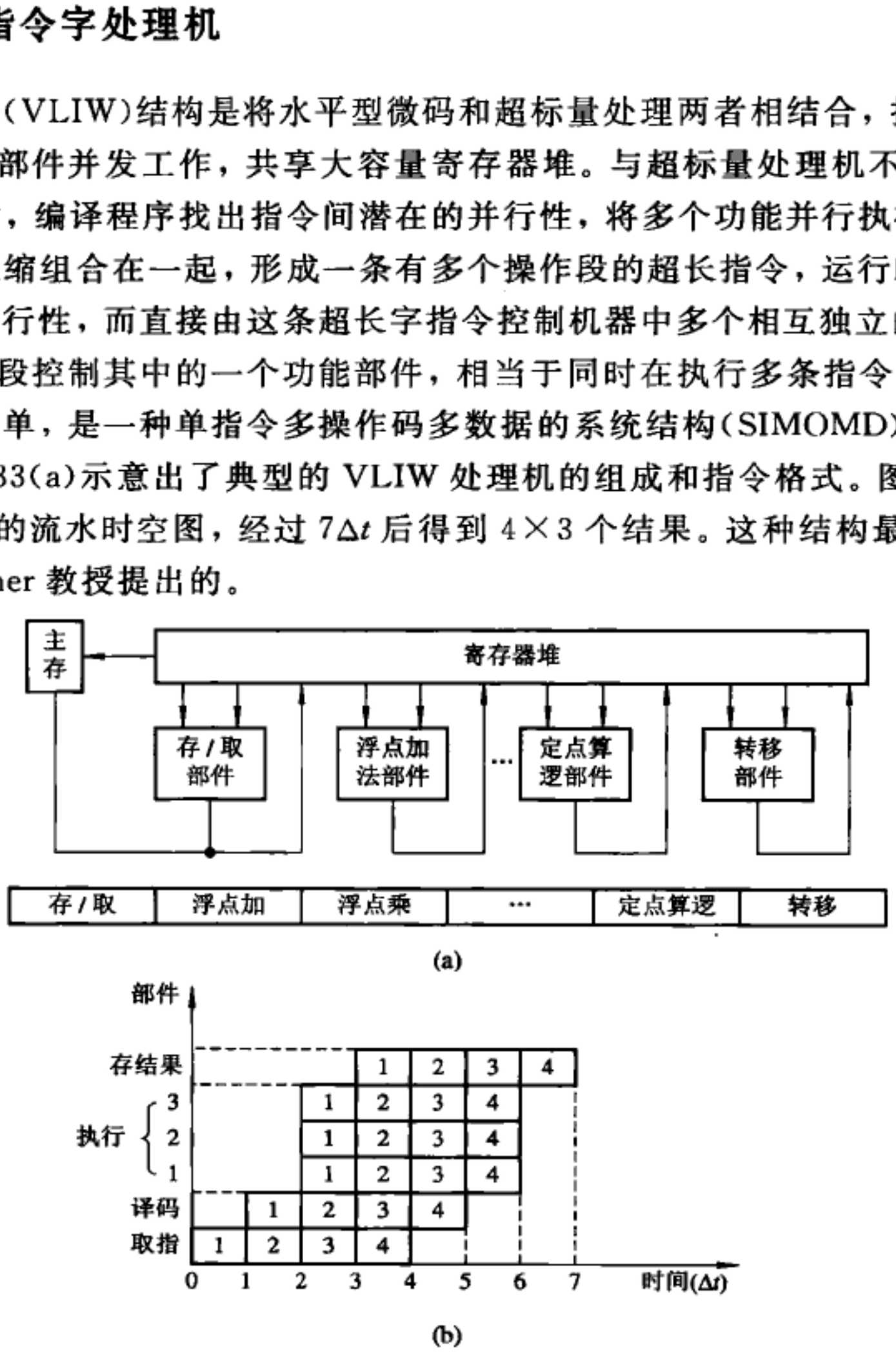


图 5－33 超长指令字(VLIW)处理机

(a) VLIW 处理机组成和指令格式；(b) 度 $m=3$ 时的流水时空图

超长指令字处理机的优点是每条指令所需拍数比超标量处理机的少，指令译码容易，开发标量操作间的随机并行性更方便，从而可使指令级并行性较高。问题是 VLIW 处理机

能否成功，很大程度取决于代码压缩的效率，其结构的目标码与一般的计算机不兼容，而且指令字很长而操作段格式固定，经常使指令字中的许多字段没有操作，白白浪费了存储空间，结构也不如超标量机的紧凑。通常设计计算机时，总是先确定系统结构，再设计编译程序，而对 VLIW 机来说，由于编译程序与系统结构关系非常密切，二者必须同时设计，缺乏对传统硬件和软件的兼容，因而不大适用于一般的应用领域。所以 VLIW 结构的思想是好的，却难以成为计算机的主流。20 世纪 80 年代前半期，VLIW 主要用于附挂式数组处理机上，现在也有用于小、巨型机上的。

典型的超长指令字处理机有 Multiflow 公司的 TARCE 计算机和 Cydrome 公司的 Cydra 5 计算机，每条指令字长为 256 位，可并发执行 7 种操作。

5.3.3 超流水线处理机

超流水线处理机不同于超标量处理机和 VLIW 处理机，每个 $\Delta t'$ 仍只流出一条指令，但它的 $\Delta t'$ 值小，一台度为 m 的超流水线处理机的 $\Delta t'$ 只是基本机器周期 Δt 的 $1/m$。因此，一条指令需花 $km\Delta t'$ 的时间，k 为一条指令所含的基本机器周期数。只要流水线性能得以充分发挥，其并行度就可达 m。图 5 - 34 给出了度 $m=3$ 的超流水线处理机工作的时空图。

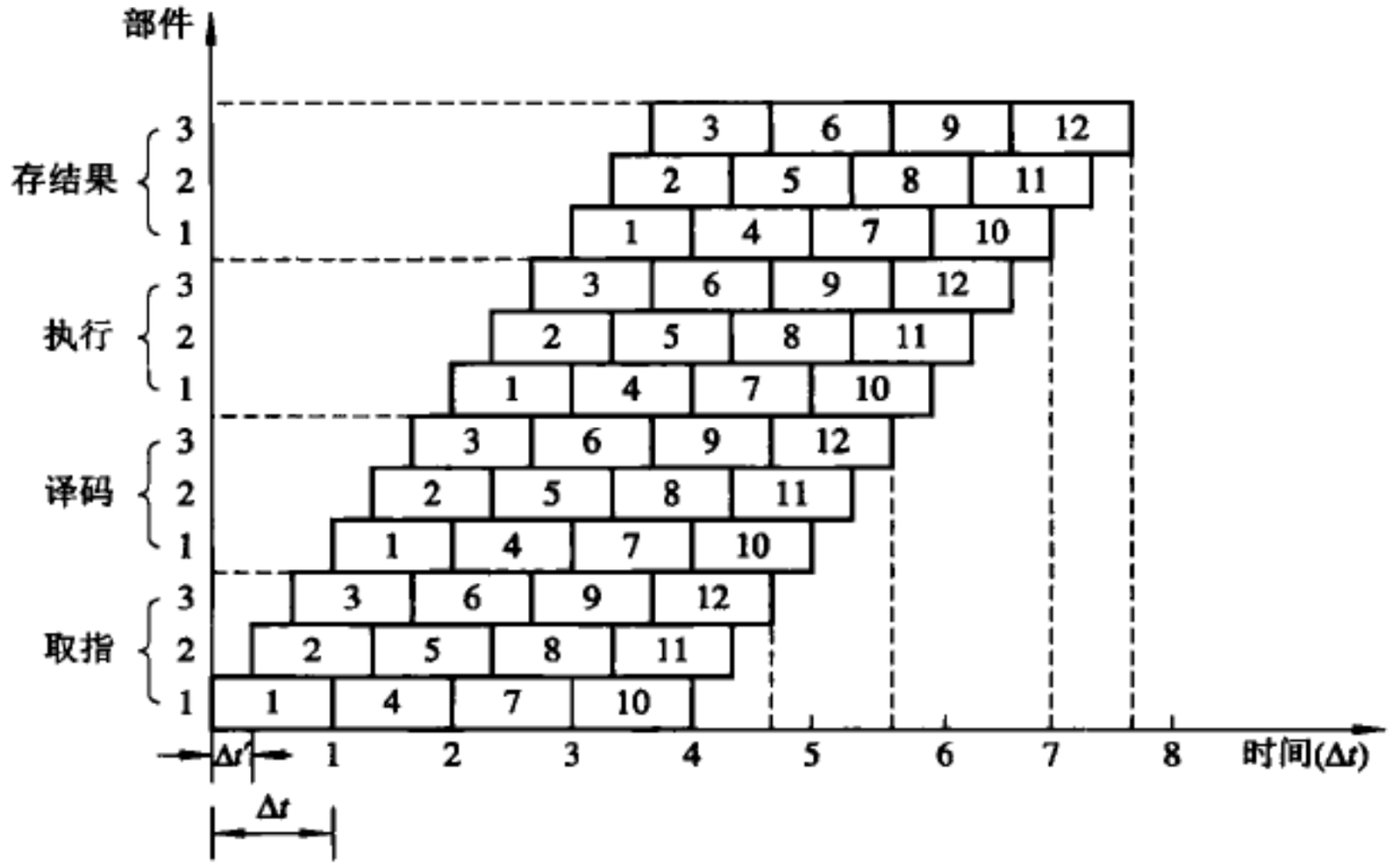


图 5 - 34 每 $\Delta t'$ 流出一条指令且度 $m=3$ 的超流水线处理机时空图

超标量处理机利用资源重复，设置多个执行部件寄存器堆端口来提高速度。超流水线处理机则着重开发时间并行性，在公共的硬件上采用较短的时钟周期，深度流水来提高速度，需使用多相时钟，时钟频率高达 100～500 MHz。没有高速时钟机制，超流水线处理机是无法实现的。例如一台有 k 段流水线的 m 度超流水线处理机，执行完 N 条指令的时间为 $\left(k+\frac{N-1}{m}\right)\Delta t$，如图 5 - 34 所示，所需时间为 $\left(4+\frac{12-1}{3}\right)\Delta t=7\ \frac{2}{3}\Delta t$，相对常规流水线处理机加速比为

$$S_p=\frac{(k+N-1)\Delta t}{(k+(N-1)/m)\Delta t}=\frac{m(k+N-1)}{(mk+N-1)}$$

当 N 趋于无穷大时，加速比 S_p 趋于 m。

超流水线处理机出现较早，如 CRAY－1 的定点加法为 $3\Delta t'$。1991 年 2 月 MIPS 公司的 64 位 RISC 计算机 R4000 的度 $m=3$。

5.3.4 超标量超流水线处理机

超标量超流水线处理机是超标量流水线与超流水线机的结合。它在一个 $\Delta t'$（等于 $\Delta t/n$）时间内发射 k 条指令（超标量），而每次发射时间错开 $\Delta t'$（超流水），相当于每拍 Δt 流出了 nk 条指令，即并行度 $m=kn$。例如 $k=3$，$n=3$，完成 12 个任务时，就只需 $5\Delta t$，其时间关系图如图 5－35 所示，其并行度 $m=9$。

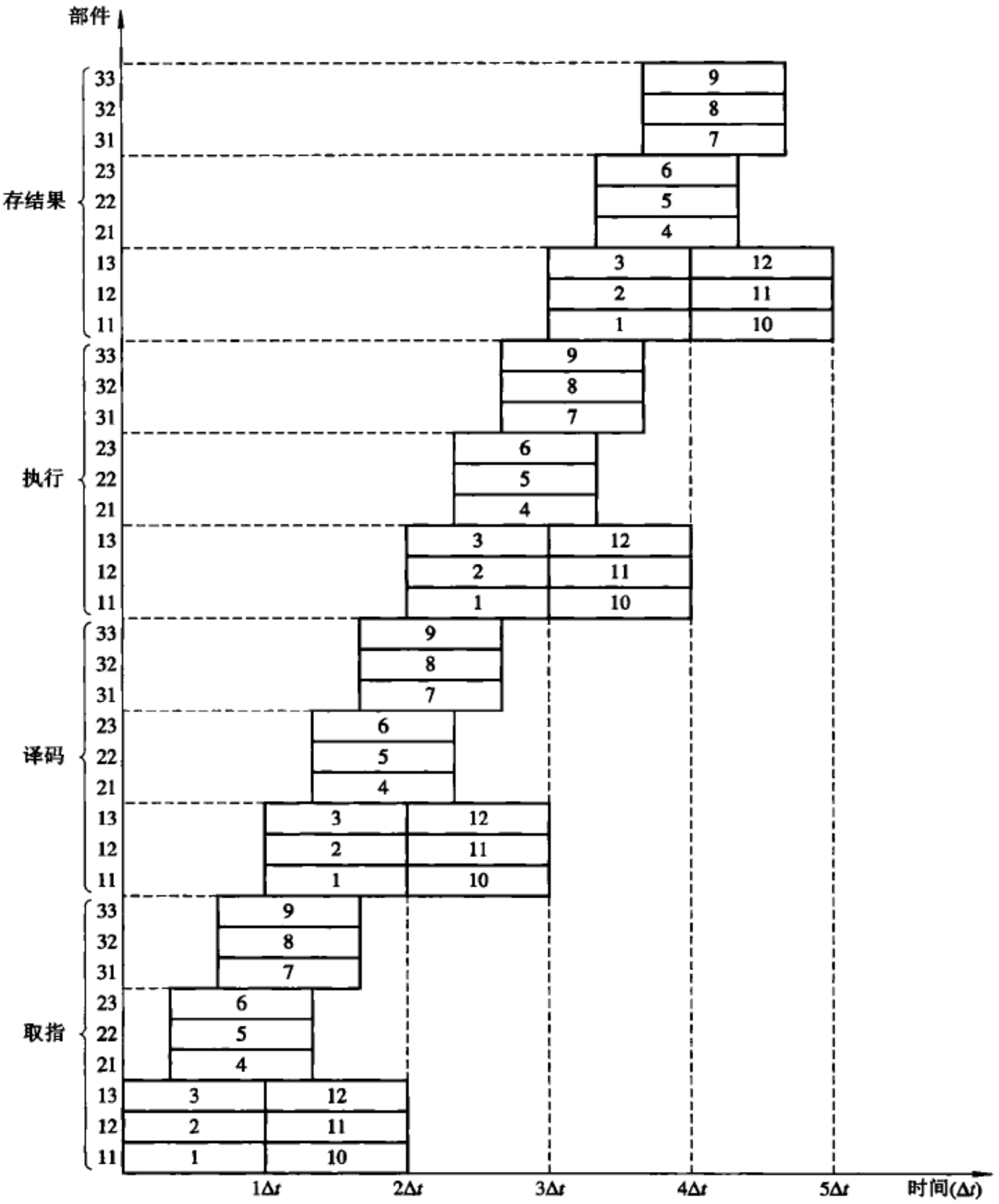


图 5－35 超标量超流水线时空图举例

5.4 本章小结

5.4.1 知识点和能力层次要求

(1) 领会重叠工作方式的原理、它对计算机组成设计的要求、“一次重叠”的定义和优点。领会“一次重叠”时条件转移与后续指令间的相关、指令相关、主存数相关、通用寄存器的数相关、通用寄存器组的变(基)址值相关等的定义与各自的处理办法。领会设置相关专用通路的目的及其适用的场合。在给出了指令之间各种微操作时间重叠关系的要求之后，计算连续执行完 N 条指令所需花费的全部时间，要达到简单应用层次。

(2) 领会流水方式的工作原理。识记流水或流水处理机的各种分类和定义。在给出的流水线上解具体题目时，要会画流水线时空图，计算流水的最大吞吐率、实际吞吐率、效率、加速比等，要达到综合应用层次。领会为消除流水线速度性能瓶颈可用的两种途径、时空图画法、吞吐率和效率的计算；在双功能静态流水线上调整操作的流入顺序，使具体计算有尽可能高的性能，画相应流水时空图，计算实际吞吐率、效率和加速比等。以上均要达到综合应用层次。

(3) 掌握流水机器对局部性相关的处理办法。领会流水线在异步流动时，指令之间发生先写后读、先读后写、写一写相关的定义。以 IBM 360/91 为例，综述在标量流水机上，处理局部性相关、全局性相关、中断等的方法，要达到综合应用层次。

(4) 在单功能非线性流水线上，会找任务最佳调度方案，会计算极限吞吐率；实际调度若干个任务时，会画流水线工作的时空图，对实际吞吐率和效率进行计算。以上均应达到综合应用层次。

(5) 领会超标量、超长指令字、超流水线、超标量超流水线四种处理机在指令级上并行的工作原理。给出指令数和并行的度数，画各自的时空图，计算所需的时间及相对于度为 1 的常规标量流水机的加速比等，要达到综合应用层次。

5.4.2 重点和难点

1. 重点

“一次重叠”方式中，各种相关的处理；流水线的时空图和性能分析；流水的局部性相关的处理；全局性相关的处理和对中断的处理；单功能非线性流水线的调度。

2. 难点

给出指令间微操作的时间重叠关系要求，算出全部指令完成所需要的时间，给出数学计算式；在二功能静态流水线上，为获得尽可能高的性能，如何调整其操作的流入顺序，画出此时的流水时空图，并计算出吞吐率、效率和加速比；为消除流水线速度性能瓶颈所采取的措施及相应的流水时空图画法，吞吐率和效率的计算；优化单功能非线性流水线的调度方案，相应的时空图画法，吞吐率和效率的计算；在超标量处理机、超长指令字处理机、超流水线处理机、超标量超流水线处理机上，给出指令数和并行度数，画出对应的时

空图，计算相对于度为 1 的常规标量流水线处理机的加速比。

习 题 5

5－1　假设指令的解释分取指、分析与执行 3 步，每步的时间相应为 $t_{取指}$、$t_{分析}$、$t_{执行}$，

(1) 分别计算下列几种情况下，执行完 100 条指令所需时间的一般关系式：

① 顺序方式；

② 仅“执行$_k$”与“取指$_{k+1}$”重叠；

③ 仅“执行$_k$”、“分析$_{k+1}$”与“取指$_{k+2}$”重叠。

(2) 分别在 $t_{取指}=t_{分析}=2$，$t_{执行}=1$ 和 $t_{取指}=t_{执行}=5$，$t_{分析}=2$ 两种情况下，计算出上述各结果。

5－2　流水线由 4 个功能部件组成，每个功能部件的延迟时间为 Δt，当输入 10 个数据后间歇 $5\Delta t$ 又输入 10 个数据，如此周期性地工作，求此时流水线的吞吐率，并画出其时空图。

5－3　有一个浮点乘流水线如图 5－36(a)所示，其乘积可直接返回输入端或暂存于相应缓冲寄存器中，画出实现 $A\times B\times C\times D$ 的时空图以及输入端的变化，并求出流水线的吞吐率和效率；当流水线改为图 5－36(b)所示的形式实现同一计算时，求该流水线的效率及吞吐率。

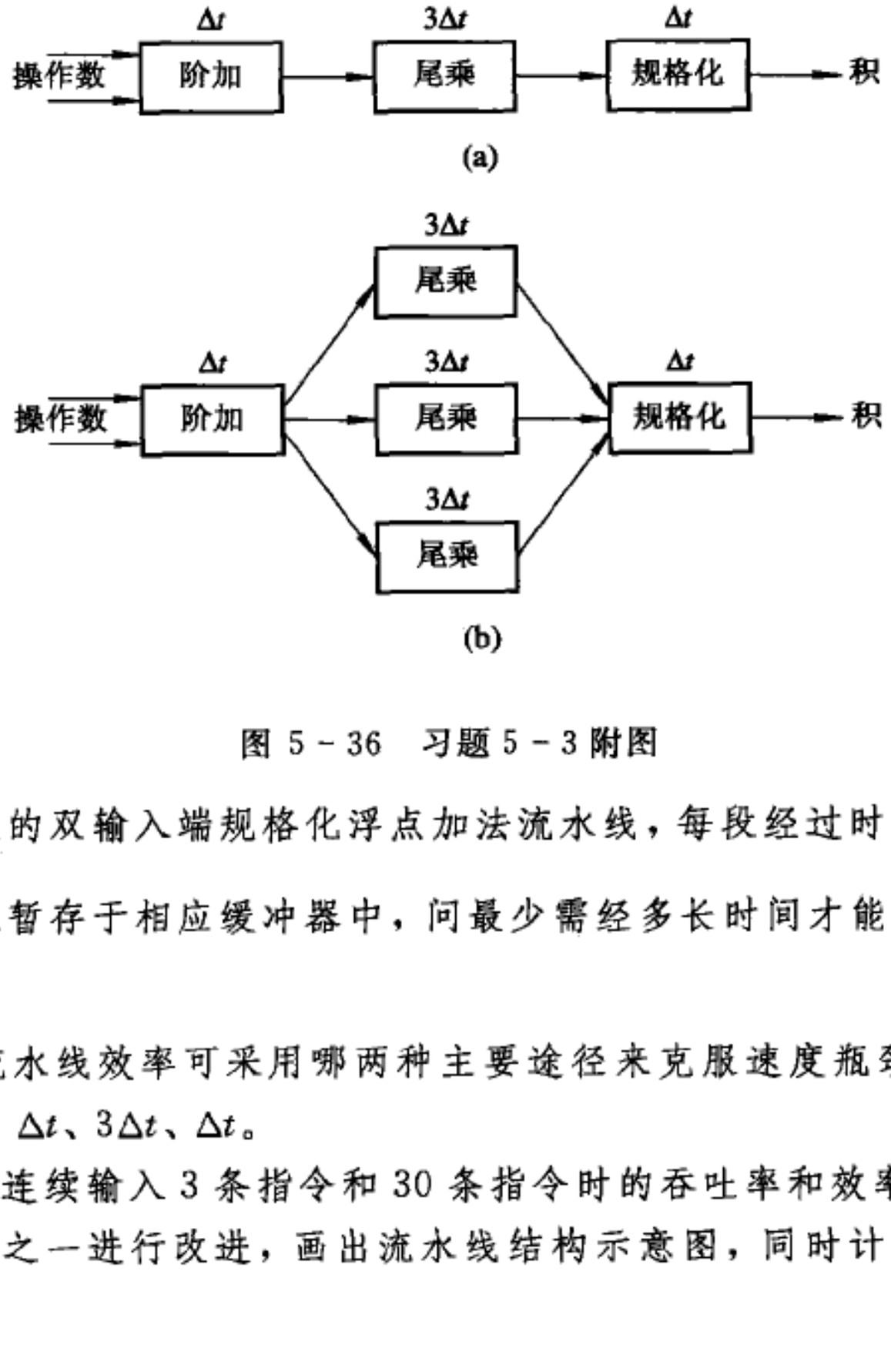


图 5－36　习题 5－3 附图

5－4　一个 4 段的双输入端规格化浮点加法流水线，每段经过时间为 10 ns，输出可直接返回输入或将结果暂存于相应缓冲器中，问最少需经多长时间才能求出 $\sum_{i=1}^{10} A_i$，并画出时空图。

5－5　为提高流水线效率可采用哪两种主要途径来克服速度瓶颈？现有 3 段流水线，各段经过时间依次为 Δt、$3\Delta t$、Δt。

(1) 分别计算在连续输入 3 条指令和 30 条指令时的吞吐率和效率。

(2) 按两种途径之一进行改进，画出流水线结构示意图，同时计算连续输入 3 条指令

和 30 条指令时的吞吐率和效率。

(3) 通过对(1)、(2)两小题的计算比较可得出什么结论?

5－6　有一个双输入端的加一乘双功能静态流水线，由经过时间为 Δt、$2\Delta t$、$2\Delta t$、Δt 的 1、2、3、4 四个子过程构成。“加”按 1→2→4 连接，“乘”按 1→3→4 连接，流水线输出设有数据缓冲器，也可将数据直接返回输入。现要执行 $A\times(B+C\times(D+E\times F))+G\times H$ 的运算，请调整计算顺序，画出能获得吞吐率尽量高的流水时空图，标出流水线入、出端数的变化情况，求出完成全部运算的时间及此期间流水线的效率。如对流水线瓶颈子过程再细分，最少需多长时间可完成全部运算?若子过程 3 不能再细分，只能用并联方法改进，则流水线的效率为多少?

5－7　有一个乘一加双功能静态流水线，“乘”由 1→2→3→4 完成，“加”由 1→5→4 完成，各段延时均为 Δt，输出可直接返回输入或存入缓冲器缓冲。现要求计算长度均为 8 的 $\boldsymbol{A}$、$\boldsymbol{B}$ 两个向量逐对元素求和的连乘积

$$S=\prod_{i=1}^{8}(\boldsymbol{A}_i+\boldsymbol{B}_i)$$

(1) 画出流水线完成此运算的时空图。

(2) 完成全部运算所需多少 Δt? 此期间流水线的效率是多少?

5－8　带双输入端的加一乘双功能静态流水线有 1、2、3、4 四个子部件，延时分别为 Δt、Δt、$2\Delta t$、Δt，“加”由 1→2→4 组成，“乘”由 1→3→4 组成，输出可直接返回输入或锁存，现欲执行 $\sum_{i=1}^{4}[(a_i+b_i)\times c_i]$。

(1) 画出此流水时空图，标出流水线入端数据变化情况。

(2) 计算运算全部完成所需时间及在此期间流水线的效率。

(3) 将瓶颈子部件再细分，画出解此题的时空图。

(4) 求出按(3)解此题所需时间及在此期间流水线的效率。

5－9　现有长度为 8 的向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，请分别画出下列 4 种结构的处理器上求点积 $\boldsymbol{A}\cdot\boldsymbol{B}$ 的时空图，并求完成全部结果的最少时钟拍数。设处理器中每个部件的输出均可直接送到任何部件的输入或存入缓冲器中，其间的传送延时不计，指令和源操作数均能连续提供。

(1) 处理器有一个乘法部件和一个加法部件，不能同时工作，部件内也只能以顺序方式工作，完成一次加法或乘法均需 5 拍。

(2) 与(1)基本相同，只是乘法部件和加法部件可并行。

(3) 处理器有一个乘一加双功能静态流水线，乘、加均由 5 个流水段构成，各段经过时间要 1 拍。

(4) 处理器有乘、加两条流水线，可同时工作，各由 5 段构成，每段经过时间为 1 拍。

5－10　试总结 IBM 360/91 解决流水控制的一般方法、途径和特点。

5－11　在一个 5 段的流水线处理机上需经 9 拍才能完成一个任务，其预约表如表 5－4 所示。

分别写出延迟禁止表 F、冲突向量 $\boldsymbol{C}$；画出流水线状态转移图；求出最小平均延迟及流水线的最大吞吐率及其调度方案。按此流水调度方案输入 6 个任务，求实际吞吐率。

表 5－4　9 拍才能完成一个任务的预约表

段 \ 时间	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
S_1	√								√
S_2		√	√						
S_3				√			√	√	
S_4				√	√				
S_5						√	√		

5－12　设指令由取指、分析、执行三个子部件组成。每个子部件经过时间为 Δt，连续执行 12 条指令。请分别画出在常规标量流水处理机及度 m 均为 4 的超标量处理机、超长指令字处理机、超流水线处理机上工作的时空图，分别计算它们相对常规标量流水处理机的加速比 S_p。

5－13　什么是超标量超流水线处理机?

第6章　向量处理机

向量处理机是具有向量数据表示的处理机，分为向量流水处理机和阵列处理机两类组织形式。向量流水处理机是以时间重叠途径开发出来的。阵列处理机是以资源重复途径开发出来的，它是将大量重复设置的处理单元(Processing Element，PE)按一定方式互连成阵列，在单一控制部件(Control Unit，CU)的控制下，对各自所分配的不同数据并行执行同一指令规定的操作，是操作级并行的 SIMD 计算机。本章先讲述向量的流水处理、向量流水处理机结构及其性能的优化问题；然后讲述阵列处理机的原理、并行算法、互连网络及并行存储器的无冲突访问等问题；最后简要讲述脉动阵列流水机的原理和通用脉动阵列结构。

6.1　向量的流水处理和向量流水处理机

由于向量内部各元素(分量)很少相关，且一般又是执行同一种操作，容易发挥流水线的效能，因此可将向量数据表示和流水线结合构成向量流水处理机，以提高主要是面向向量数组计算类应用的计算机的速度性能。

6.1.1　向量的处理和向量的流水处理

虽然向量运算比标量运算更易发挥出流水线的效能，但处理方式选择不当也不行。选择使向量运算最能充分发挥出流水效能的处理方式，就是向量的流水处理要研究的问题。它常和所采用计算机的结构有关。不同的向量处理方式会对流水处理机的结构、组成提出不同的要求，而结构和组成不同的向量处理机反过来也会要求采用不同的向量流水处理方式。

【例 6-1】 计算 $\boldsymbol{D}=\boldsymbol{A}\times(\boldsymbol{B}+\boldsymbol{C})$，其中 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$、$\boldsymbol{D}$ 都是有 N 个元素的向量，应该采用什么方式处理才能充分发挥流水线的效能？

如果采用逐个求 $\boldsymbol{D}$ 向量元素的方法，即访存取 a_i、b_i、c_i 元素求 d_i，再取 a_{i+1}、b_{i+1}、c_{i+1} 求 d_{i+1}，则这种处理方式称为横向(水平)处理方式。横向处理方式宜于在标量处理机上用循环程序实现，但却难以使流水线连续流动，因为每个 d_i 元素的计算需要用到加、乘两条指令，需分别进行 $b_i+c_i\rightarrow k$ 和 $k\times a_i\rightarrow d_i$。尽管整个运算只需用一个单元 k 来存放中间结果，但会频繁出现先写后读的操作数相关，每次只有等加法结果出来才能开始乘法，因而流水的吞吐率会下降。如果在 ASC 多功能静态流水机上实现，则加、乘相间的运算每次都得进行流水线功能切换，那流水线的吞吐率会比顺序执行的还要低。这时只有对整个向量按相同操作都执行完之后再转去执行别的操作，才能较好地发挥流水处理的效能。也就

是说，处理方式改为先执行 $b_i+c_i \rightarrow k_i$（$i=1$，…，N），然后执行 $k_i \times a_i \rightarrow d_i$（$i=1$，…，$N$），称这种处理方式为纵向（垂直）处理方式。在计算 $\boldsymbol{B}+\boldsymbol{C}$ 时，因为是对向量 $\boldsymbol{B}$、$\boldsymbol{C}$ 的所有元素都执行相同的加法操作，且无相关，流水线就能连续流动，从而使其效能得以发挥。计算 $k \times \boldsymbol{A}$ 时也是如此，功能切换总共只需进行一次。在 ASC 流水机上，将上述向量运算由横向处理改为纵向处理，就是向量的流水处理，流水线就能每拍流出一个结果元素。

然而，这只有在流水线输入端能每拍取得成对元素的情况下，才能每拍求得一个结果元素。由于 N 值不会过小，早期的向量机只好把向量运算指令的源向量和目的向量都存放在主存中，采用面向存储器—存储器型结构的流水线处理机。在这种机器上，必须显著提高主存与流水处理机之间的信息流量，才能支持流水线连续流动。因此，20 世纪 70 年代初问世的 STAR－100 就把主存设计成 32 个体交叉，且每个体的数据宽度都是 8 个字（字长 64 位），以使其最大信息流量达到 200 兆字每秒。

主存并非单单是为中央处理机所使用的，很多通道也要用。要保证源向量元素连续供应和结果向量元素及时存入主存很不容易。将主存直接连在流水线输入、输出端的做法并不是最好的方案。随着半导体存储器芯片价格的持续下降，20 世纪 70 年代中期问世的 CRAY－1 向量流水处理机改为把流水线输入、输出端连到容量足够大的向量寄存器组，采用面向寄存器—寄存器型结构的流水线处理机。向量寄存器组与主存之间采用成组传送，这样就较好地缓解了主存流量和流水线的处理速率不匹配的矛盾。

若向量的长度 N 太长，超出了向量寄存器组中寄存器的个数，则可以将该向量分割成若干个组，使每组都能装得进向量寄存器组中。这样，每一组内均按纵向方式处理，而组和组之间则采用软件方法编制循环程序的方式依次循环处理。我们称这种处理方式为分组纵横处理方式。采用这种分组纵横处理方式，就可以对向量长度 N 的大小不加限制。CRAY－1 就是采用这种方式来进行向量的流水处理的。

结论：向量横向处理是向量的处理方式，但不是向量的流水处理方式，而向量纵向处理和分组纵横处理是向量的处理方式，也是向量的流水处理方式。

6.1.2 向量流水处理机的结构举例

向量流水处理机的结构因具体机器的不同而不同。下面以 CRAY－1 机为例，介绍面向寄存器—寄存器型向量流水处理的一些结构特点。

CRAY－1 是由中央处理机、诊断维护控制处理机、大容量磁盘存储子系统、前端处理机组成的功能分布异构型多处理机。中央处理机的控制部分含有 256 个 16 位的指令缓冲器，分成 4 组，每组为 64 个。

中央处理机的运算部分有 12 条可并行工作的单功能流水线，可分别流水地进行地址、向量、标量的各种运算。另外，还有可为流水线功能部件直接访问的向量寄存器组 $V_0 \sim V_7$、标量寄存器 $S_0 \sim S_7$ 及地址寄存器 $A_0 \sim A_7$。

图 6－1 只画出了 CRAY－1 中央处理机中有关向量流水处理部分的简图。它为向量运算提供的 6 个流水线单功能部件是整数加、逻辑运算、移位、浮点加、浮点乘和浮点迭代求倒数，其流水经过的时间分别为 3、2、4、6、7 和 14 拍，一拍为 12.5 ns。任何一条流水线，只要满负荷流动，都可以每拍流出一个结果分量。

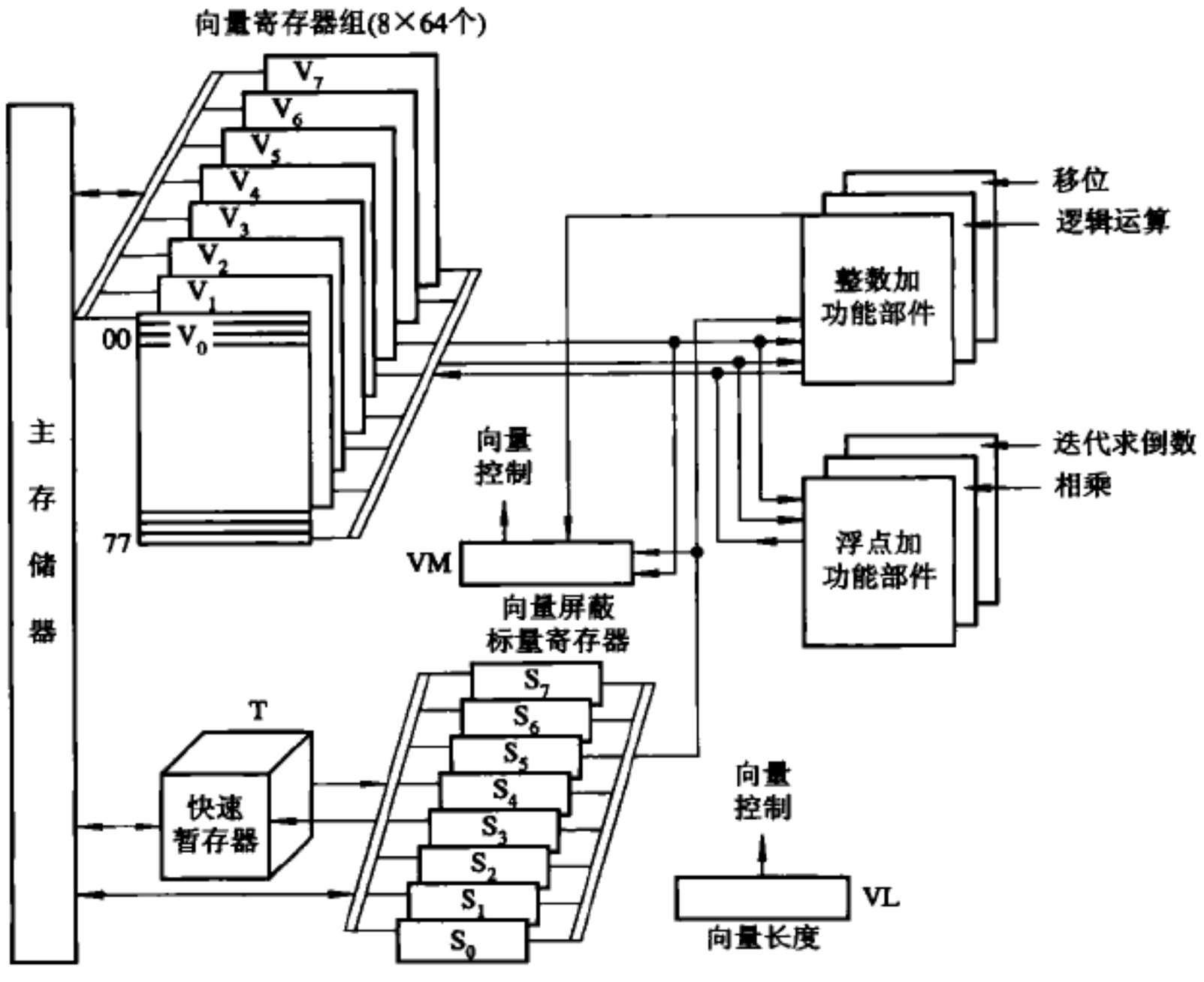


图 6-1　CRAY-1 的向量流水处理部分简图

向量寄存器部分由 512 个 64 位的寄存器组成，分成 8 组，编号为 $V_0 \sim V_7$。每个向量寄存器组 V_i 可存放最多含 64 个分量(元素)的一个向量。因此向量寄存器组部分同时可存放 8 个向量。对于长度超过 64 个分量的长向量，可以由软件加以分段处理，每段 64 个分量。为处理长向量而形成的程序结构称为向量循环。每经一次循环，处理一段。分段中，余下不足 64 个分量的段作为向量循环的首次循环，最先处理。

CRAY-1 有标量类和向量类指令共 128 条，其中有 4 种向量指令如图 6-2 所示。第Ⅰ种源向量分别取自两个向量寄存器组 V_j、V_k，结果送向量寄存器组 V_i。第Ⅱ种与第Ⅰ种的差别只在于它的一个操作数取自标量寄存器 S_j。其中的 n 是流水线功能部件经过的时钟数。第Ⅲ、Ⅳ种控制主存与 V_i 向量寄存器组之间成组数据的传送。访存流水线建立时间为 6 拍。

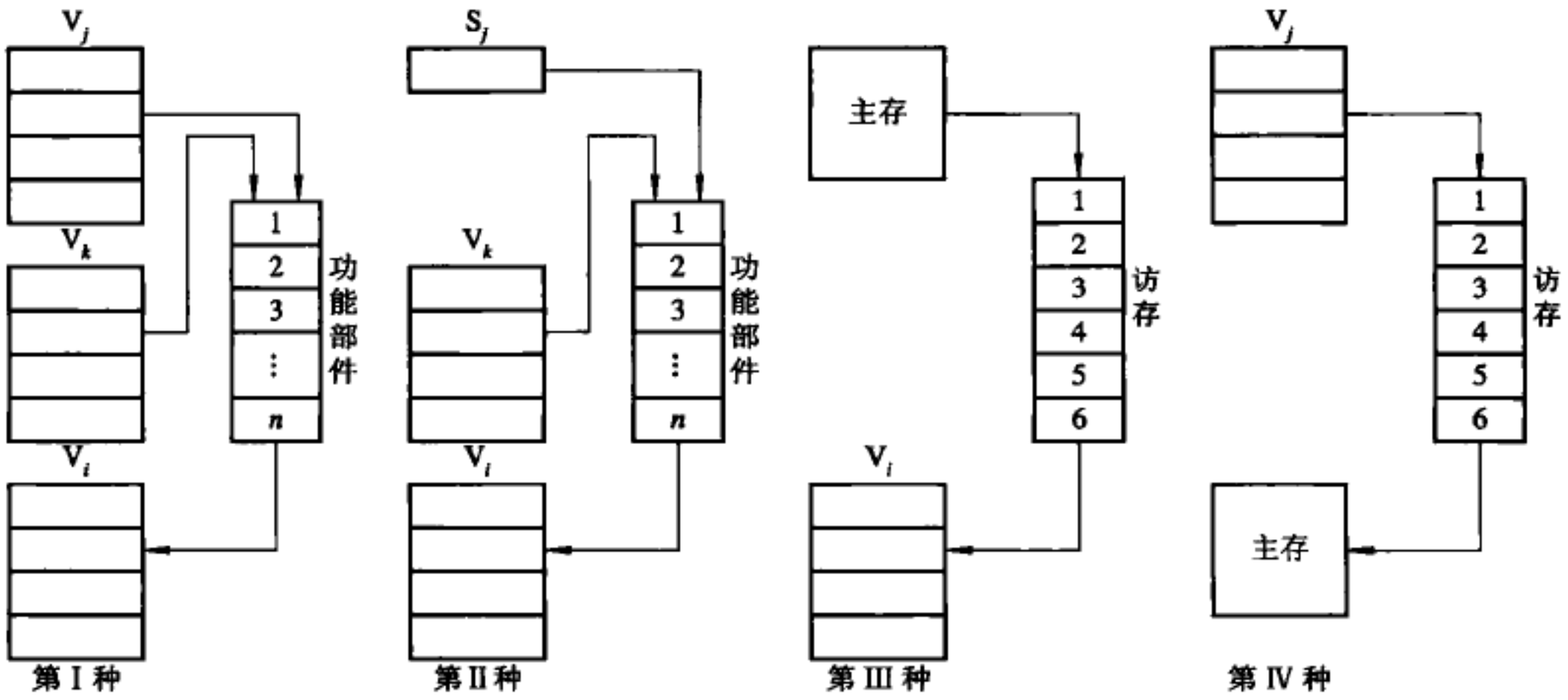


图 6-2　CRAY-1 的四种向量指令