第 3 章 流水线技术

内容提要

(1) 流水线的基本概念；
(2) 流水线的性能指标；
(3) 非线性流水线的调度；
(4) 流水线的相关与冲突；
(5) 流水线的实现(以 MIPS 为例)。

3.1 流水线的基本概念

3.1.1 什么是流水线

工业上的流水线大家一定都很熟悉了，例如汽车装配生产流水线等。在这样的流水线中，整个装配过程被分为多道工序，每道工序由一个人(或多人)完成，各道工序所花的时间也差不多。整条流水线流动起来后，每隔一定的时间间隔(差不多就是一道工序的时间)就有一辆汽车下线。如果我们跟踪一辆汽车的装配全过程，就会发现其总的装配时间并没有缩短，但由于多辆车的装配在时间上错开后，重叠进行，因此最终能达到总体装配速度(吞吐率)的提高。

在计算机中也可以采用类似的方法，把一个重复的过程分解为若干个子过程(相当于上面的工序)，每个子过程由专门的功能部件来实现。把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其他的子过程并行进行。这就是流水线技术(pipelining)。流水线中的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。流水线的段数称为流水线的深度(Pipeline Depth)。

把流水线技术应用于指令的解释执行过程，就形成了指令流水线。把流水线技术应用于运算的执行过程，就形成了运算操作流水线，也称为部件级流水线。图 3.1 是一条浮点加法流水线，它把执行过程分解为求阶差、对阶、尾数相加、规格化 4 个子过程，每一个子过程在各自独立的部件上完成。如果各段的时间相等，都是 Δt，那么，虽然完成一次浮点加法所需要的总时间(从“入”到“出”)还是 $4\Delta t$，但若在输入端连续送入加法任务，则从加法器的输出端来看，却是每隔一个 Δt 就能出一个浮点加法结果。因此，该流水线能把浮点加法运算的速度提高 3 倍。

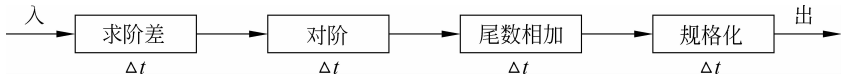


图 3.1　浮点加法流水线

一般采用时空图来描述流水线的工作过程。图 3.2 是上述 4 段流水线的时空图。图中横坐标表示时间,纵坐标表示空间,即流水线中的流水段。格子中的数字 1 代表第 1 个运算,2 代表第 2 个运算……。第 1 个运算在时刻 0 进入流水线;第 2 个运算在时刻 1 进入流水线,同时第 1 个运算离开"求阶差"段而进入"对阶"段;第 3 个运算在时刻 2 进入流水线,同时第 1 个运算离开"对阶"段而进入"尾数相加"段,第 2 个运算离开"求阶差"段而进入"对阶"段;第 4 个运算在时刻 3 进入流水线,同时第 1 个运算离开"尾数相加"段而进入"规格化"段,第 2 个运算离开"对阶"段而进入"尾数相加"段,第 3 个运算离开"求阶差"段而进入"对阶"段;以此类推。

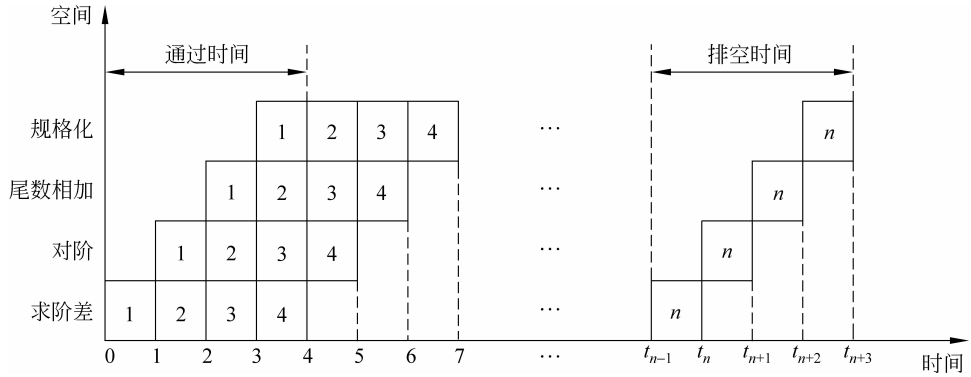


图 3.2　浮点加法流水线的时空图

从上面的分析可以看出,流水技术有以下特点:

(1) 流水线把一个处理过程分解为若干个子过程,每个子过程由一个专门的功能部件来实现。因此,流水线实际上是把一个大的处理功能部件分解为多个独立的功能部件,并依靠它们的并行工作来提高处理速度(吞吐率)。

(2) 流水线中各段的时间应尽可能相等,否则将引起流水线堵塞和断流,因为时间最长的段将成为流水线的瓶颈(Bottleneck of a Pipeline),此时流水线中的其他功能部件就不能充分发挥作用了。因此瓶颈问题是流水线设计中必须解决的。

(3) 流水线每一个段的后面都要有一个缓冲寄存器(锁存器),称为流水寄存器。其作用是在相邻的两段之间传送数据,以提供后面流水段要用到的信息。其另一个作用是隔离各段的处理工作,避免相邻流水段电路的相互打扰。

(4) 流水技术适合于大量重复的时序过程,只有在输入端不断地提供任务,才能充分发挥流水线的效率。

(5) 流水线需要有通过时间和排空时间。它们分别是指第一个任务和最后一个任务从进入流水线到流出结果的那个时间段,如图 3.2 所示。在这两个时间段中,流水线都不是满负荷的。经过"通过时间"后,流水线进入满载工作状态,整条流水线的效率才能得到充分发挥。

3.1.2 流水线的分类

流水线可以从不同的角度和观点来分类，下面是几种常见的分类。

1. 部件级流水线、处理机级流水线及系统级流水线

按照流水技术用于计算机系统的等级不同，可以把流水线分为3种：部件级流水线、处理机级流水线和系统级流水线。

部件级流水线是把处理机中的部件进行分段，再把这些分段相互连接而成的。它使得运算操作能够按流水方式进行。图3.1中的浮点加法流水线就是一个典型的例子。这种流水线也称为运算操作流水线(Arithmetic Pipeline)。

处理机级流水线又称指令流水线(Instruction Pipeline)。它是把指令的执行过程按照流水方式进行处理，即把一条指令的执行过程分解为若干个子过程，每个子过程在独立的功能部件中执行。3.4.1节中论述的5段指令流水线就是一个例子，它能同时重叠执行5条指令。

系统级流水线是把多个处理机串行连接起来，对同一数据流进行处理，每个处理机完成整个任务中的一部分。前一台处理机的输出结果存入存储器中，作为后一台处理机的输入。这种流水线又称宏流水线(Macro Pipeline)。

2. 单功能流水线与多功能流水线

这是按照流水线所完成的功能来分类的。

1) 单功能流水线(Unifunction Pipeline)

单功能流水线是指流水线各段之间的连接固定不变、只能完成一种固定功能的流水线。如前面介绍的浮点加法流水线就是单功能流水线。若要完成多种功能，可采用多条单功能流水线。例如Cray-1巨型机有12条单功能流水线。

2) 多功能流水线(Multifunction Pipeline)

多功能流水线是指各段可以进行不同的连接，以实现不同功能的流水线。美国TI公司ASC处理机中采用的运算流水线就是多功能流水线，它有8个功能段，按不同的连接可以实现浮点加减法运算和定点乘法运算，如图3.3所示。

3. 静态流水线与动态流水线

多功能流水线可以进一步分为静态流水线和动态流水线两种。

1) 静态流水线(Static Pipeline)

静态流水线是指在同一时间内，多功能流水线中的各段只能按同一种功能的连接方式工作的流水线。当流水线要切换到另一种功能时，必须等前面的任务都流出流水线之后，才能改变连接。例如，上述ASC的8段只能或者按浮点加减运算连接方式工作，或者按定点乘运算连接方式工作。在图3.4中，当要在n个浮点加法后面进行定点乘法时，必须等最后一个浮点加法做完、流水线排空后，才能改变连接，开始新的运算。

2) 动态流水线(Dynamic Pipeline)

动态流水线是指在同一时间内，多功能流水线中的各段可以按照不同的方式连接，同时执行多种功能的流水线。它允许在某些段正在实现某种运算时，另一些段却在实现另一种运算。当然，多功能流水线中的任何一个功能段只能参加到一种连接中。动态流水线的优点是：更加灵活，能提高各段的使用率，能提高处理速度，但其控制复杂度增加了。

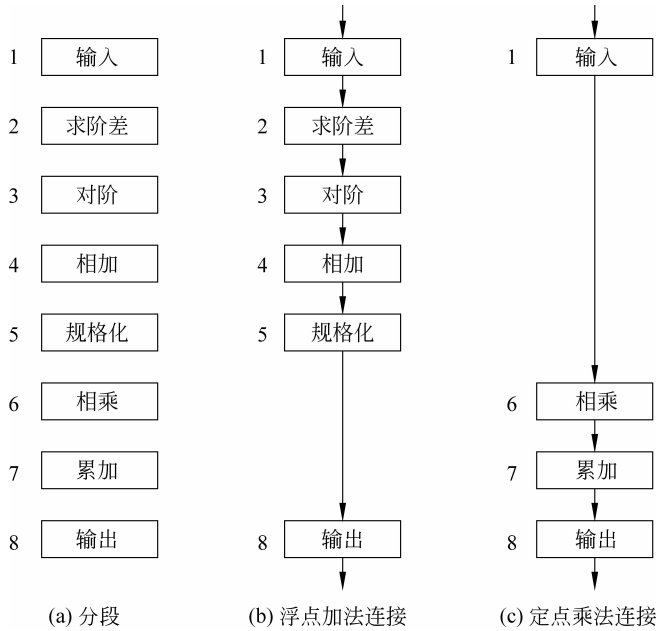


图 3.3　ASC 处理机的多功能流水线

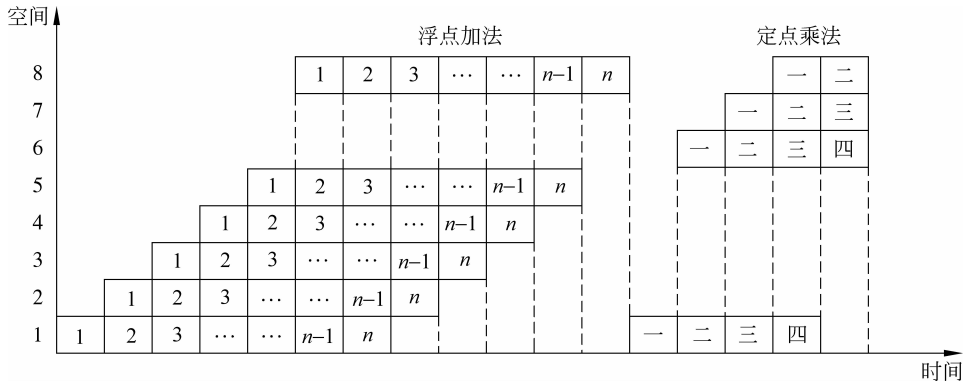


图 3.4　静态流水线的时空图

对于图 3.3 的情况,动态流水线的工作过程如图 3.5 所示。这里,定点乘法提前开始了(相对于静态流水线而言)。可以提前多少取决于任务的流动情况,要保证不能在公用段发生冲突。

对于静态流水线来说,只有当输入的是一串相同的运算任务时,流水的效率才能得到充分的发挥。如果交替输入不同的运算任务,则流水线的效率会降低到和顺序处理方式的一样。而动态流水线则不同,它允许多种运算在同一条流水线中同时进行。因此,在一般情况下,动态流水线的效率比静态流水线的高。但是,动态流水线的控制要复杂得多。所以目前大多数的流水线是静态流水线。

4. 线性流水线与非线性流水线

按照流水线中是否存在反馈回路,可以把流水线分为以下两类。

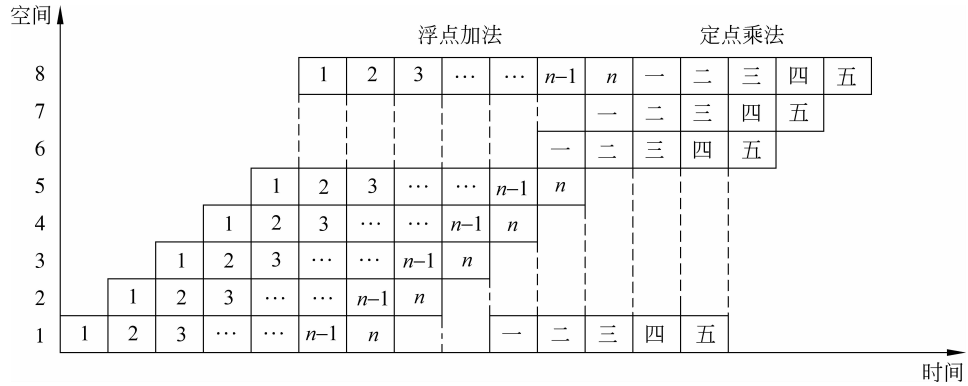


图 3.5　动态流水线的时空图

1）线性流水线(Linear Pipeline)

线性流水线是指各段串行连接、没有反馈回路的流水线。数据通过流水线中的各段时，每一个段最多只流过一次。

2）非线性流水线(Nonlinear Pipeline)

非线性流水线是指各段除了有串行的连接外，还有反馈回路的流水线。图 3.6 是一个非线性流水线的示意图。它由 4 段组成，经反馈回路和多路开关使某些段要多次通过。S_3 的输出可以反馈到 S_2，而 S_4 的输出可以反馈到 S_1。

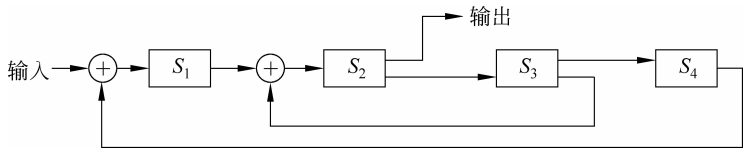


图 3.6　非线性流水线示意图

非线性流水线常用于递归或组成多功能流水线。在非线性流水线中，一个重要的问题是确定什么时候向流水线引进新的任务，才能使该任务不会与先前进入流水线的任务发生争用流水段的冲突。这就是所谓的非线性流水线的调度问题，3.3 节将详细讨论这个问题。

5. 顺序流水线与乱序流水线

根据流水线中任务流入和流出的顺序是否相同，可以把流水线分为以下两种。

1）顺序流水线(In-order Pipeline)

在顺序流水线中，流水线输出端任务流出的顺序与输入端任务流入的顺序完全相同。每一个任务在流水线的各段中是一个跟着一个顺序流动的。

2）乱序流水线(Out-of-order Pipeline)

在乱序流水线中，流水线输出端任务流出的顺序与输入端任务流入的顺序可以不同，允许后进入流水线的任务先完成。这种流水线又称为无序流水线、错序流水线、异步流水线。

通常把指令执行部件中采用了流水线的处理机称为流水线处理机。如果处理机具有向量数据表示和向量指令，则称为向量流水处理机，简称向量机；否则就称为标量流水处理机。

3.2 流水线的性能指标

衡量流水线性能的主要指标有吞吐率、加速比和效率。

3.2.1 流水线的吞吐率

流水线的吞吐率 TP(ThroughPut)是指在单位时间内流水线所完成的任务数量或输出结果的数量。

$$TP = \frac{n}{T_k} \tag{3.1}$$

其中 n 为任务数,T_k 是处理完 n 个任务所用的时间。该式是计算流水线吞吐率最基本的公式。

1. 各段时间均相等的流水线

图 3.7 是各段时间均相等(都是 Δt)的线性流水线的时空图。这里假设段数为 k,连续输入 n 个任务。第一个任务输入后,经过 $k\Delta t$ 的时间从输出端流出(完成)。此后的 $n-1$ 个 Δt 中,每个 Δt 时间完成一个任务。在这种情况下,流水线完成 n 个连续任务所需要的总时间为

$$T_k = k\Delta t + (n-1)\Delta t = (k+n-1)\Delta t \tag{3.2}$$

将式(3.2)代入式(3.1),得流水线的实际吞吐率为

$$TP = \frac{n}{(k+n-1)\Delta t} \tag{3.3}$$

这种情况下的最大吞吐率为

$$TP_{max} = \lim_{n\to\infty} \frac{n}{(k+n-1)\Delta t} = \frac{1}{\Delta t} \tag{3.4}$$

最大吞吐率与实际吞吐率的关系是

$$TP = \frac{n}{k+n-1} TP_{max} \tag{3.5}$$

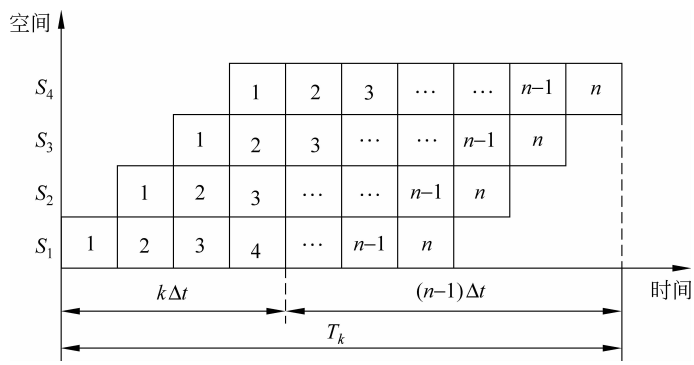


图 3.7 各段时间相等的流水线时空图

从式(3.5)可以看出,流水线的实际吞吐率总是小于最大吞吐率,它除了与每个段的时间有关外,还与流水线的段数 k 和输入流水线中的任务数 n 有关。只有当 $n>>k$ 时,才有

$TP \approx TP_{max}$。

2. 各段时间不完全相等的流水线

在如图 3.8(a)所示的流水线中，各段时间不完全相等。其中 S_1，S_2，S_3，S_5 各段的时间都是 Δt，S_4 的时间是 $3\Delta t$，是其他各段时间的 3 倍。S_4 是该流水线的瓶颈段。除了第一个任务外，其余$(n-1)$个任务必须按瓶颈段的时间间隔 $\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)$连续流入流水线。图 3.8(b)是该流水线的时空图，图中的灰色方格表示相应流水段在这一段时间内是空闲的。

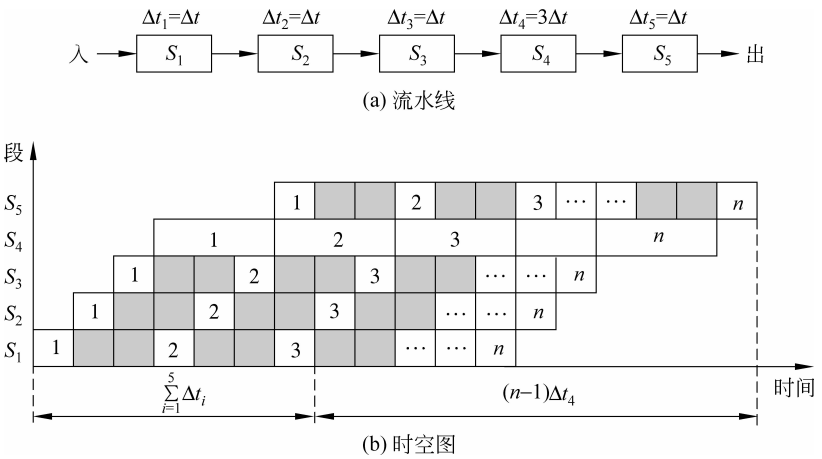


图 3.8　各段时间不等的流水线及其时空图

一般地，各段时间不等的流水线的实际吞吐率为

$$TP = \frac{n}{\sum_{i=1}^{k} \Delta t_i + (n-1)\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{3.6}$$

其中 Δt_i 为第 i 段的时间，共有 k 个段。分母中的第一部分是流水线完成第一个任务所用的时间，第二部分是完成其余 $n-1$ 个任务所用的时间。

流水线的最大吞吐率为

$$TP_{max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{3.7}$$

对于图 3.8 的例子，最大吞吐率为

$$TP_{max} = \frac{1}{3\Delta t} \tag{3.8}$$

从式(3.6)和式(3.7)可以看出，当流水线各段的时间不完全相等时，流水线的最大吞吐率和实际吞吐率由时间最长的那个段决定，这个段就成了整条流水线的瓶颈。这时，瓶颈段一直处于忙碌状态，而其余各段则在许多时间内都是空闲的，硬件使用效率低。

可以用下面的两种方法来消除瓶颈段。

1) 细分瓶颈段

该方法是把流水线中的瓶颈段切分为几个独立的功能段，从而使流水线各段的处理时间都相等。在图 3.9 中，把瓶颈段 S_4 细分为三个子流水段：$S_{4\text{-}1}$，$S_{4\text{-}2}$，$S_{4\text{-}3}$。这样，所产生的流水线的各段时间均为 Δt，每隔 Δt 流出一个结果。

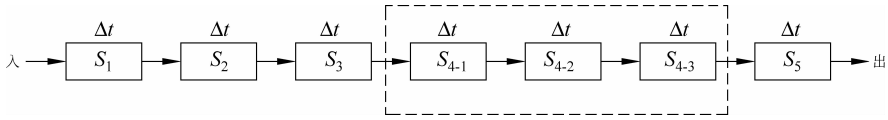
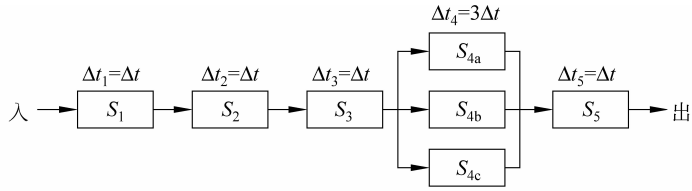


图 3.9　细分瓶颈段

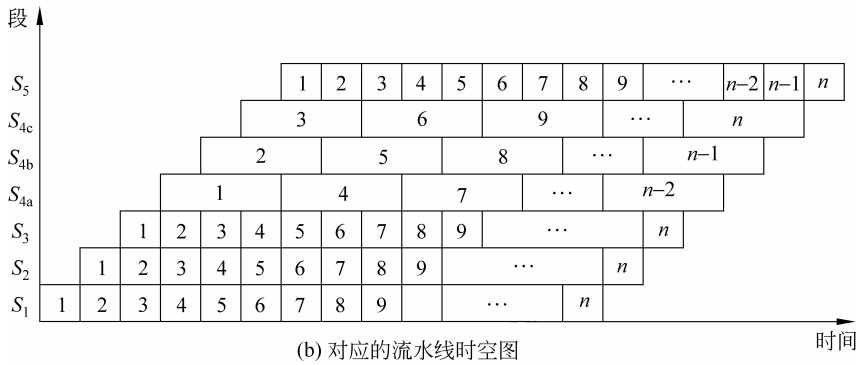
2) 重复设置瓶颈段

如果无法把瓶颈段再细分,就可以采用重复设置瓶颈段的方法来解决问题。重复设置的段并行工作,在时间上依次错开处理任务。这种方法的缺点是控制逻辑比较复杂,所需要的硬件也增加了。

图 3.10 给出了把 S_4 重复设置后的流水线及时空图。这里,从 S_3 到并列的 S_{4a},S_{4b},S_{4c} 之间需要设置一个数据分配器,它把从 S_3 输出的第一个任务分配给 S_{4a},第二个任务分配给 S_{4b},第三个任务分配给 S_{4c},之后按此重复。而在 S_{4a},S_{4b},S_{4c} 到 S_5 之间需要设置一个数据收集器,依次分时将数据收集到 S_5 中。改进后的流水线能做到每隔 Δt 流出一个结果。



(a) 重复设置瓶颈段



(b) 对应的流水线时空图

图 3.10　重复设置瓶颈段及对应的流水线时空图

对于图 3.8 的例子,这两种方法都能使改进后的流水线的吞吐率达到

$$\mathrm{TP}_{\max} = \frac{1}{\Delta t} \tag{3.9}$$

3.2.2　流水线的加速比

流水线的加速比(speedup)是指使用顺序处理方式处理一批任务所用的时间与流水线使用流水处理方式处理同一批任务所用的时间之比。设顺序执行所用的时间为 T_s,按流水线方式处理所用的时间为 T_k,则流水线的加速比为

$$S = \frac{T_s}{T_k} \tag{3.10}$$

假设流水线各段时间都是 Δt，则一条 k 段流水线完成 n 个连续任务所需要的时间为 $T_k=(k+n-1)\Delta t$。这 n 个任务若是顺序执行，则所需要的时间为 $T_s=nk\Delta t$。代入式(3.10)，得流水线的实际加速比为

$$S=\frac{nk}{k+n-1} \tag{3.11}$$

这种情况下的最大加速比为

$$S_{\max}=\lim_{n\to\infty}\frac{nk}{k+n-1}=k \tag{3.12}$$

即当 $n>>k$ 时，流水线的加速比等于流水线的段数。从这个意义上看，流水线的段数越多越好，但这会给流水线的设计带来许多问题，对此后面将作进一步讨论。

当流水线的各段时间不完全相等时，一条 k 段流水线完成 n 个连续任务的实际加速比为

$$S=\frac{n\sum_{i=1}^{k}\Delta t_i}{\sum_{i=1}^{k}\Delta t_i+(n-1)\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{3.13}$$

3.2.3 流水线的效率

流水线的效率(efficiency)即流水线设备的利用率，它是指流水线中的设备实际使用时间与整个运行时间的比值。由于流水线有通过时间和排空时间，所以在连续完成 n 个任务的时间内，各段并不是满负荷工作的。

如果各段时间相等，如图 3.7 所示，则各段的效率 e_i 是相同的。

$$e_1=e_2=\cdots=e_k=\frac{n\Delta t}{T_k}=\frac{n}{k+n-1} \tag{3.14}$$

整条流水线的效率为

$$E=\frac{e_1+e_2+\cdots+e_k}{k}=\frac{ke_1}{k}=\frac{kn\Delta t}{kT_k} \tag{3.15}$$

还可以写成

$$E=\frac{n}{k+n-1} \tag{3.16}$$

最高效率为

$$E_{\max}=\lim_{n\to\infty}\frac{n}{k+n-1}=1 \tag{3.17}$$

显然，当 $n>>k$ 时，流水线的效率接近最大值 1。这时流水线的各段均处于忙碌状态。

根据式(3.3)和式(3.16)，可得

$$E=\mathrm{TP}\cdot\Delta t \tag{3.18}$$

即当流水线各段时间相等时，流水线的效率与吞吐率成正比。

根据式(3.11)和式(3.16)，可得

$$E=\frac{S}{k} \tag{3.19}$$

即流水线的效率是流水线的实际加速比 S 与它的最大加速比 k 的比值。只有当 $E=1$ 时，

S=k,实际加速比达到最大。

式(3.15)中的分母 kT_k 实际上就是时空图中 k 个段和流水总时间 T_k 所围成的总面积,而分子 $kn\Delta t$ 则是时空图中 n 个任务实际占用的总面积。所以,从时空图上看,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。

虽然当流水线的各段时间不等时,各段的效率会不同,但同样也有上述的结论。因此,计算流水线效率的一般公式可以表示为

$$E=\frac{n\text{ 个任务实际占用的时空区的面积}}{k\text{ 个段总的时空区的面积}} \tag{3.20}$$

画出流水线的时空图,然后根据式(3.20)来计算效率,是一种比较直观通用的方法。对于线性流水线、非线性流水线、多功能流水线、任务不连续的情况等都适用。

在各段时间不等的情况下,k 段流水线连续处理 n 个任务的流水线效率为

$$E=\frac{n\cdot\sum_{i=1}^{k}\Delta t_i}{k\left[\sum_{i=1}^{k}\Delta t_i+(n-1)\cdot\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)\right]} \tag{3.21}$$

3.2.4 流水线的性能分析举例

例 3.1 要在图 3.3 所示的静态流水线上计算 $\prod_{i=1}^{4}(A_i+B_i)$,流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中,试计算其吞吐率、加速比和效率。

解 首先,应选择适合流水线工作的算法。对于本题,应先计算 A_1+B_1、A_2+B_2、A_3+B_3 和 A_4+B_4;再计算$(A_1+B_1)\times(A_2+B_2)$和$(A_3+B_3)\times(A_4+B_4)$;然后求总的乘积结果。

其次,画出完成该计算的时空图,如图 3.11 所示,图中阴影部分表示相应的段在工作。

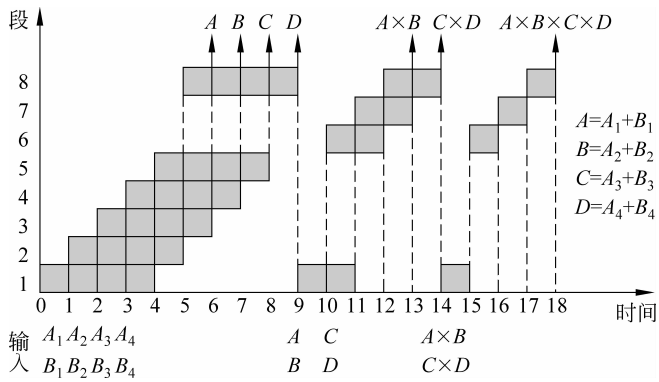


图 3.11 完成乘加运算的多功能静态流水线时空图

由图 3.11 可见,它在 18 个 Δt 的时间中,给出了 7 个结果,所以吞吐率为

$$\mathrm{TP}=\frac{7}{18\Delta t}$$

如果不用流水线,由于一次求和需 $6\Delta t$,一次求积需 $4\Delta t$,则产生上述 7 个结果共需

$(4\times6+3\times4)\Delta t=36\Delta t$。所以加速比为

$$S=\frac{36\Delta t}{18\Delta t}=2$$

该流水线的效率可由阴影区的面积和 8 个段总时空区的面积的比值求得

$$E=\frac{4\times6+3\times4}{8\times18}=0.25$$

该流水线的效率很低，其主要原因是：

① 多功能流水线在做某一种运算时，总有一些段是空闲的；

② 静态流水线在进行功能切换时，要等前一种运算全部流出流水线后才能进行后面的运算；

③ 运算之间存在关联，后面有些运算要用到前面运算的结果。这就是后面要讨论的相关问题；

④ 流水线的工作过程有建立与排空部分。

例 3.2 有一条动态多功能流水线由 5 段组成(如图 3.12 所示)，加法用 1、3、4、5 段，乘法用 1、2、5 段，第 4 段的时间为 $2\Delta t$，其余各段时间均为 Δt，而且流水线的输出可以直接返回输入端或暂存于相应的流水寄存器中。若在该流水线上计算 $\sum_{i=1}^{4}(A_i\times B_i)$，试计算其吞吐率、加速比和效率。

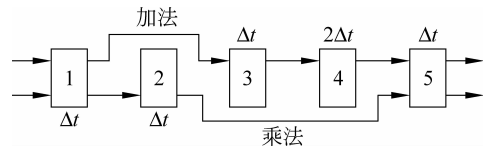


图 3.12 完成加、乘运算的多功能动态流水线

解 首先，应选择适合流水线工作的算法。对于本题，应先计算 $A_1\times B_1$、$A_2\times B_2$、$A_3\times B_3$ 和 $A_4\times B_4$；再计算 $(A_1\times B_1)+(A_2\times B_2)$ 和 $(A_3\times B_3)+(A_4\times B_4)$；然后求总的累加结果。

其次，画出完成该计算的时空图，如图 3.13 所示，图中阴影部分表示相应段在工作。

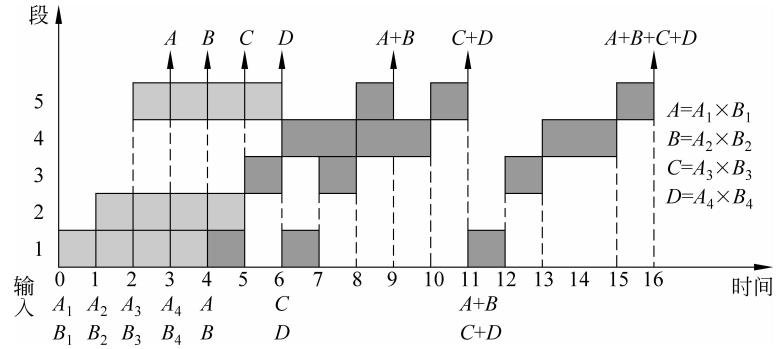


图 3.13 完成加乘运算的多功能动态流水线的时空图

由图 3.13 可见，它在 16 个 Δt 的时间中，给出了 7 个结果，所以吞吐率为

$$TP=\frac{7}{16\Delta t}$$

如果不用流水线，由于一次求积需 $3\Delta t$，一次求和需 $5\Delta t$，则产生上述 7 个结果共需 $(4\times3+3\times5)\Delta t=27\Delta t$，所以加速比为

$$S = \frac{27\Delta t}{16\Delta t} \approx 1.69$$

该流水线的效率可由阴影区的面积和5个段总时空区的面积的比值求得

$$E = \frac{4 \times 3 + 3 \times 5}{5 \times 16} \approx 0.338$$

3.2.5 流水线设计中的若干问题

1. 瓶颈问题

当流水线的各段时间相等时,其中的任务是同步地每一个节拍往前流动一段。这个节拍往往就是机器的时钟周期。当流水线各段时间不相等时,时间最大的那个段就成了瓶颈。机器的时钟周期取决于这个瓶颈段的延迟时间。因此,在设计流水线时,要尽可能使各段时间相等。

2. 流水线的额外开销

流水线的额外开销由两部分构成:流水寄存器延迟和时钟偏移开销。在非流水方式中,是没有这个开销的,它是采用流水线技术后额外产生的。前面讲过,流水线的段与段之间都要设置流水寄存器,而该寄存器需要有建立时间和传输延迟。时钟偏移开销则是指流水线中的时钟到达各流水寄存器的最大差值时间。时钟到达各流水寄存器的时间不是完全相同的。

采用流水技术后,虽然可以提高指令执行的吞吐率,从而提高程序的执行速度,但流水线并不能真正减少单条指令的执行时间。实际上,由于这些开销的存在,反而会使每条指令的执行时间有所增加。这限制了流水线深度的增加。前面曾指出,增加流水线的段数可以提高流水线的性能,但是流水线段数的增加是受到这些额外开销限制的,一旦时钟周期时间减少到和额外开销时间相接近的时候,流水线就没有任何意义了,因为这时在每个时钟周期内几乎没有时间来做有用的工作了。

由于这些额外开销对流水线的性能有较大的影响,特别是当流水深度比较大、时钟周期比较小时更是如此,所以设计者必须选择高性能的锁存器作为流水寄存器。

3. 冲突问题

在运算操作流水线中,如果后面的计算要用到前面的计算结果,流水线的性能和效率就会受到比较大的影响。对于指令流水线来说,也是如此。如果流水线中的指令之间存在关联,则它们可能要相互等待,引起流水线的停顿。如何处理好冲突问题,是流水线设计中要解决的重要问题之一。

3.3 非线性流水线的调度

在线性流水线中,由于每一个任务在流水线的各段只经过一次,所以可以每个时钟周期向流水线输入一个新任务(假设该流水线没有瓶颈)。这些任务不会争用同一个流水段,所以线性流水线的调度非常简单。然而,非线性流水线则不同,由于反馈回路的存在,当一个任务在流水线中流过时,可能要多次经过某些段。在这种情况下,就不能每个时钟周期向流水线送入一个新任务;否则就会发生多个任务争用同一段的冲突现象。那么,究竟应按什么

样的时间间隔向流水线输入新任务，才能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呢？这就是流水线调度所要解决的问题。

下面先讨论单功能非线性流水线的最优调度方法，然后在此基础上，论述多功能非线性流水线的最优调度方法。

3.3.1 单功能非线性流水线的最优调度

向一条非线性流水线的输入端连续输入两个任务之间的时间间隔称为非线性流水线的启动距离(Initiation Interval)。而会引起非线性流水线功能段使用冲突的启动距离则称为禁用启动距离。启动距离和禁用启动距离一般都用时钟周期数来表示，是一个正整数。

为了对流水线的任务进行优化调度和控制，E. S. Davidson 等人提出使用预约表(Reservation Table)。这是一个二维表，其横向(向右)表示时间(一般用时钟周期表示)，纵向(向下)表示流水线的段。它表示当一个任务在该流水线中流过时，对各段的使用情况。如果在第 n 个时钟周期使用第 k 段，则在第 k 行和第 n 列的交叉处的格子里有一个√。反过来，如果在第 k 行和第 n 列的交叉处的格子里有一个√，则表示在第 n 个时钟周期要使用第 k 段。这个表可以采用类似画时空图的方法得到。图 3.14 是预约表的一个例子。

时间 / 功能段	1	2	3	4	5	6	7	8	9
S_1	√								√
S_2		√	√					√	
S_3				√					
S_4					√	√			
S_5							√	√	

图 3.14 一个 5 功能段非线性流水线预约表

1. 根据预约表写出禁止表 F

禁止表(Forbidden List)F 是一个由禁用启动距离构成的集合。可以很容易地由预约表写出禁止表 F。具体方法是：对于预约表每一行的任何一对√，用它们所在的列号相减(大的减小的)，列出各种可能的差值，然后删除相同的，剩下的就是禁止表的元素。这是因为对于同一行中的任何一对√，其差值 d 表示在前一次使用该段后，间隔 d 个时钟周期，又会再次使用该段。这样，如果后一个任务正好跟前一个任务相隔 d 个时钟周期，那么就会在该段产生冲突。也就是说，d 是禁用启动距离。

在图 3.14 的例子中，第一行的差值只有一个：8；第二行的差值有三个：1,5,6；第三行只有一个√，没有差值；第四和第五行的差值都只有一个：1。所以其禁止表是：$F=\{1,5,6,8\}$。

2. 根据禁止表 F 写出初始冲突向量 C_0

冲突向量(Collison Vector)$\boldsymbol{C}$ 是一个 N 位的二进制位串。这一步实际上是进行从一个集合到一个二进制位串的变换，目的是便于进行后面的运算。

设 $\boldsymbol{C}_0=(c_N c_{N-1}\cdots c_i\cdots c_2 c_1)$，则

$$c_i=\begin{cases}1 & i\in F\\ 0 & i\notin F\end{cases}$$

显然,$c_i=0$ 表示允许间隔 i 个时钟周期后送入后续任务,$c_i=1$ 表示不允许间隔 i 个时钟周期后送入后续任务。

对于上面的例子,$F=\{1,5,6,8\}$,$\boldsymbol{C}_0=(10110001)$。

3. 根据初始冲突向量 $\boldsymbol{C}_0$ 画出状态转换图

当第一个任务流入流水线后,初始冲突向量 $\boldsymbol{C}_0$ 决定了下一个任务需间隔多少个时钟周期才可以流入。在第二个任务流入后,哪些时间间隔是允许的,哪些时间间隔又是禁止的呢?也就是说,新的冲突向量是怎样的呢?假设第二个任务是在与第一个任务间隔 j 个时钟周期流入的,这时,由于第一个任务已经在流水线中前进了 j 个时钟周期,其相应的禁止表中各元素的值都应该减去 j,并丢弃小于等于 0 的值。对冲突向量来说,就是逻辑右移 j 位(左边补零)。这样,对后续任务流入的限制就是两个任务叠加起来的共同结果。反映在冲突向量上,就是对它们的冲突向量进行"或"运算,即

$$\mathrm{SHR}^{(j)}(\boldsymbol{C}_0)\ \vee\ \boldsymbol{C}_0$$

其中 $\mathrm{SHR}^{(j)}$ 表示逻辑右移 j 位。

推广到更一般的情况,假设当前的冲突向量是 $\boldsymbol{C}_k$,j 是允许的时间间隔,则新的冲突向量为

$$\mathrm{SHR}^{(j)}(\boldsymbol{C}_k)\ \vee\ \boldsymbol{C}_0 \tag{3.22}$$

对于所有允许的时间间隔都按上述步骤求出新的冲突向量,并把新的冲突向量作为当前冲突向量,反复使用上述步骤,直到不再产生新的冲突向量为止。

从初始冲突向量 $\boldsymbol{C}_0$ 出发,反复应用上述步骤,可以求得所有的冲突向量以及产生这些向量所对应的时间间隔。由此可以画出用冲突向量表示的流水线状态转移图。图中冲突向量之间用有向弧表示状态转移的方向。弧上的数字表示引入后续任务(从而产生新的冲突向量)所用的时间间隔(时钟周期数)。

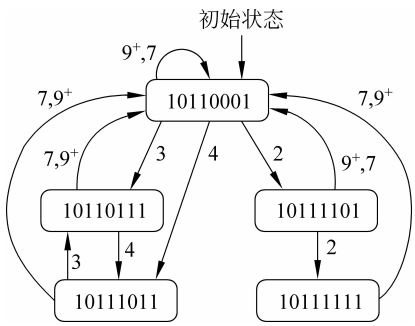


图 3.15 单功能流水线的状态转移示意图

对于图 3.14 的例子,$\boldsymbol{C}_0=(10110001)$,引入后续任务可用的时间间隔为 2、3、4、7 个时钟周期。如果采用 2,则新的冲突向量为 $(00101100)\vee(10110001)=(10111101)$;如果采用 3,则新的冲突向量为 $(00010110)\vee(10110001)=(10110111)$;如果采用 4,则新的冲突向量为 $(00001011)\vee(10110001)=(10111011)$;如果采用 7,则新的冲突向量为 $(00000001)\vee(10110001)=(10110001)$,如图 3.15 所示。

对于新向量(10111101),其可用的时间间隔为 2 个和 7 个时钟周期。用类似上面的方法,可以求出其后续的冲突向量分别为(10111101)和(10110001)。对于其他新向量,也照此处理。

在此基础上,可以画出如图 3.15 所示的状态转移示意图。

4. 根据状态转换图写出最优调度方案

根据流水线状态图,由初始状态出发,任何一个闭合回路即为一种调度方案。例如(2,7)表示反复循环使用启动距离 2 和 7。按这个回路所规定的时间间隔,给流水线送入任务,就不会发生功能段使用冲突。想要找到一种最佳的调度方案,使流水线的吞吐率最高,

只要列出所有可能的调度方案，计算出每种方案的平均时间间隔，从中找出其最小者即可。

表 3.1 给出了图 3.15 的例子中，各种调度方案及其平均间隔时间。从表中可以看出，方案(3,4)的平均间隔时间为 3.5 个时钟周期，是最佳的，吞吐率最高。虽然方案(4,3)的平均间隔时间也是 3.5，但由于它是先采用 4 个时钟周期的间隔，后采用 3 个时钟周期的间隔，当任务数是奇数时，方案(4,3)中最后一个任务是间隔 4 个时钟周期，而方案(3,4)中最后一个任务是间隔 3 个时钟周期。所以，只有方案(3,4)是最优的。

表 3.1　各种调度策略及平均延迟拍数

调度策略	平均延迟拍数	调度策略	平均延迟拍数
(2,7)	4.5	(3,4,7)	4.67
(2,2,7)	3.67	(4,3,7)	4.67
(3,7)	5	(4,7)	5.5
(3,4)	3.5	(7)	7
(3,4,3,7)	4.25		

方案(3,4)是一种不等时间间隔的调度方案，与等间隔的调度方案相比，在控制上要复杂得多。为了简化控制，也可以采用等时间间隔的调度方案，但吞吐率和效率往往会下降不少。在上述例子中，等时间间隔的方案只有一个：(7)，其吞吐率下降了一半。

3.3.2　多功能非线性流水线的调度

对于多功能流水线来说，由于不同功能的任务可以相互穿插在一起流入流水线，其调度复杂得多。下面仅以双功能(设为功能 A 和 B)非线性流水线为例，说明多功能非线性流水线的最优调度方法。

双功能非线性流水线的最优调度方法类似单功能非线性流水线的调度方法。只是其状态转移图中节点状态的表示不同，是由两个冲突向量构成的冲突矩阵，这两个冲突向量分别对应于下一个任务的功能是 A 类和 B 类的情况。而且，其初始节点有两个，分别对应于第一个任务是 A 类和 B 类的情况。假设当第一个任务是 A 类时，其冲突矩阵为 $\boldsymbol{M}_{\mathrm{A}}^{(0)}$；当第一个任务是 B 类时，其冲突矩阵为 $\boldsymbol{M}_{\mathrm{B}}^{(0)}$，则有

$$\boldsymbol{M}_{\mathrm{A}}^{(0)}=\begin{bmatrix}\boldsymbol{C}_{\mathrm{AA}}\\ \boldsymbol{C}_{\mathrm{AB}}\end{bmatrix},\quad \boldsymbol{M}_{\mathrm{B}}^{(0)}=\begin{bmatrix}\boldsymbol{C}_{\mathrm{BA}}\\ \boldsymbol{C}_{\mathrm{BB}}\end{bmatrix}$$

其中，$\boldsymbol{C}_{pq}$($p,q\in\{\mathrm{A},\mathrm{B}\}$)表示的是：在一个 p 类任务流入流水线后，对后续 q 类任务的冲突向量。它们可以由预约表求得。显然，$\boldsymbol{C}_{pq}$ 共有 $2^2=4$ 个。对于 N 功能流水线，这种冲突向量有 N^2 个。

后续状态的冲突矩阵由下式求得。

$$\mathrm{SHR}^{(i)}(\boldsymbol{M}_k)\ \vee\ \boldsymbol{M}_r^{(0)} \tag{3.23}$$

其中，$\boldsymbol{M}_k$ 为当前状态，r 表示下一个流入任务的类型(A 或 B)，i 是当前状态允许流入的 r 型任务的时间间隔。$\mathrm{SHR}^{(i)}(\boldsymbol{M}_k)$ 表示把当前状态中的各冲突向量逻辑右移 i 位。例如 $\mathrm{SHR}^{(3)}(\boldsymbol{M}_k)\vee\boldsymbol{M}_{\mathrm{A}}^{(0)}$ 表示的是：把当前状态 $\boldsymbol{M}_k$ 中的各冲突向量逻辑右移 3 位，再与初始矩阵 $\boldsymbol{M}_{\mathrm{A}}^{(0)}$ 进行“或”运算。

下面举一例来进一步说明双功能非线性流水线的最优调度。

例 3.3 有一条三段双功能非线性流水线,实现的功能是 A 和 B,其预约表分别如表 3.2 和表 3.3 所示。各段的通过时间都是一个时钟周期。请找出该流水线单独处理 A 类任务和单独处理 B 类任务以及混合处理两类任务的最优调度方案。

表 3.2 A 类对象预约表

段 \ 时间	1	2	3	4	5
S_1	√			√	
S_2		√			
S_3			√		√

表 3.3 B 类对象预约表

段 \ 时间	1	2	3	4	5
S_1		√			√
S_2				√	
S_3	√		√		

解 (1) 把两个预约表重叠起来,得到如表 3.4 所示的预约表。

表 3.4 A 和 B 两类对象预约表

段 \ 时间	1	2	3	4	5
S_1	A	B		A	B
S_2		A		B	
S_3	B		AB		A

(2) 由预约表求初始冲突向量和初始冲突矩阵。

由于有两种任务,所以有两个初始矩阵。

$$\boldsymbol{M}_{\mathrm{A}}^{(0)}=\begin{bmatrix}\boldsymbol{C}_{\mathrm{AA}}\\ \boldsymbol{C}_{\mathrm{AB}}\end{bmatrix},\quad \boldsymbol{M}_{\mathrm{B}}^{(0)}=\begin{bmatrix}\boldsymbol{C}_{\mathrm{BA}}\\ \boldsymbol{C}_{\mathrm{BB}}\end{bmatrix}$$

其中 $\boldsymbol{C}_{\mathrm{AA}}$ 表示一个 A 类任务流入流水线后,对下一个 A 类任务进入流水线的时间间隔的限制。根据表 3.2 可知,禁用时间间隔是 2 和 3,故禁止表为{2,3},所以 $\boldsymbol{C}_{\mathrm{AA}}=(0110)$。

$\boldsymbol{C}_{\mathrm{BB}}$ 表示一个 B 类任务流入流水线后,对下一个 B 类任务进入流水线的时间间隔的限制。根据表 3.3 可知,禁用时间间隔是 2 和 3,所以 $\boldsymbol{C}_{\mathrm{BB}}=(0110)$。

$\boldsymbol{C}_{\mathrm{AB}}$ 表示一个 A 类任务流入流水线后,对下一个 B 类任务进入流水线的时间间隔的限制。根据表 3.4 可知:

① 为了避免在 S_1 发生冲突,禁用时间间隔是 4−2=2;

② 在 S_2,不会发生冲突,这是因为根据表 3.4,A 类任务先于 B 类任务通过 S_2,而现在

的实际情况又是A类任务先于B类任务流入流水线；

③ 为了避免在S_3发生冲突，禁用时间间隔是3－1＝2，5－1＝4，5－3＝2。综合起来，有$\boldsymbol{C}_{AB}=(1010)$。

$\boldsymbol{C}_{BA}$表示一个B类任务流入流水线后，对下一个A类任务进入流水线的时间间隔的限制。根据表3.4可知：

① 为了避免在S_1发生冲突，禁用时间间隔有2－1＝1，5－1＝4，5－4＝1；

② 为了避免在S_2发生冲突，禁用时间间隔是4－2＝2；

③ 在S_3，不会发生冲突，这是因为根据表3.4，B类任务先于A类任务通过S_3，或者同时通过S_3（第三个时钟周期），而现在的实际情况又是B类任务先于A类任务流入流水线。综合起来，有$\boldsymbol{C}_{AB}=(1011)$。

因此，初始矩阵为

$$\boldsymbol{M}_A^{(0)}=\begin{bmatrix}\boldsymbol{C}_{AA}\\ \boldsymbol{C}_{AB}\end{bmatrix}=\begin{bmatrix}0110\\ 1010\end{bmatrix},\quad \boldsymbol{M}_B^{(0)}=\begin{bmatrix}\boldsymbol{C}_{BA}\\ \boldsymbol{C}_{BB}\end{bmatrix}=\begin{bmatrix}1011\\ 0110\end{bmatrix}$$

(3) 由初始冲突矩阵画出状态图。

如果第一个流入的任务是A类，初始状态就是$\boldsymbol{M}_A^{(0)}$；如果第一个流入的任务是B类，则初始状态是$\boldsymbol{M}_B^{(0)}$。

所有后续状态的冲突矩阵可用式(3.23)求得。例如，在流入一个A类任务后，从$\boldsymbol{C}_{AA}=(0110)$可知，可以隔一个或4个时钟周期再流入一个A类任务。假设是前者，则把初始状态$\boldsymbol{M}_A^{(0)}$中的各冲突向量同时右移一位（即进行$SHR^{(1)}$操作），再与$\boldsymbol{M}_A^{(0)}$进行或运算，可以得到新的冲突矩阵$\begin{bmatrix}0111\\ 1111\end{bmatrix}$。根据该矩阵的第一个冲突向量(0111)可知，只有隔4个时钟周期流入一个A类任务，才能不发生冲突。把冲突矩阵$\begin{bmatrix}0111\\ 1111\end{bmatrix}$中的各冲突向量同时右移4位（即进行$SHR^{(4)}$操作），再与$\boldsymbol{M}_A^{(0)}$进行按位或运算，可以得到新的冲突矩阵$\begin{bmatrix}0110\\ 1010\end{bmatrix}$。

再如，在流入一个A类任务后，从$\boldsymbol{C}_{AB}=(1010)$可知，可以隔一个或三个时钟周期再流入一个B类任务。假设是前者，则把初始状态$\boldsymbol{M}_A^{(0)}$中的各冲突向量同时右移一位（即进行$SHR^{(1)}$操作），再与$\boldsymbol{M}_B^{(0)}$进行或运算，可以得到新的冲突矩阵$\begin{bmatrix}1011\\ 0111\end{bmatrix}$。据此可知，允许的流入为：或者是隔3个时钟周期流入一个A类任务，或者是隔4个时钟周期流入一个B类任务。按与上述类似的方法，又可以得到新的冲突矩阵。

求出所有可能的状态后，就可以画出状态图，如图3.16所示。图中弧线上的标记$r.i$表示隔i个时钟周期流入r类的任务。

(4) 由状态图得出最优调度方案。

从状态图可以找出各种情况下的最优调度方案。只流入A类任务的最优调度方案是(A.1，A.4)，只流入B类任务的最优调度方案是(B.1，B.4)，混合流入A、B两类任务的最优调度方案是(B.1，A.3，A.4)。

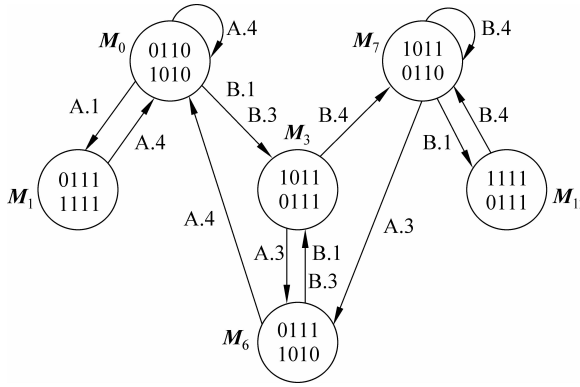


图 3.16　双功能非线性流水线的状态图

3.4　流水线的相关与冲突

3.4.1　一条经典的 5 段流水线

在论述流水线的相关与冲突之前,先来介绍一条经典的 5 段 RISC 流水线。

先考虑在非流水情况下是如何实现的。把一条指令的执行过程分为以下 5 个时钟周期。

1. 取指令周期(IF)

以程序计数器(PC)中的内容作为地址,从存储器中取出指令并放入指令寄存器(IR);同时 PC 值加 4(假设每条指令占 4 个字节),指向顺序的下一条指令。

2. 指令译码/读寄存器周期(ID)

对指令进行译码,并用 IR 中的寄存器地址去访问通用寄存器组,读出所需的操作数。

3. 执行/有效地址计算周期(EX)

在这个周期,ALU 对在上一个周期准备好的操作数进行运算或处理。不同指令所进行的操作不同。

(1) load 和 store 指令:ALU 把指令中所指定的寄存器的内容与偏移量相加,形成访存有效地址。

(2) 寄存器-寄存器 ALU 指令:ALU 按照操作码指定的操作对从通用寄存器组中读出的数据进行运算。

(3) 寄存器-立即数 ALU 指令:ALU 按照操作码指定的操作对从通用寄存器组中读出的操作数和指令中给出的立即数进行运算。

(4) 分支指令:ALU 把指令中给出的偏移量与 PC 值相加,形成转移目标的地址。同时,对在前一个周期读出的操作数进行判断,确定分支是否成功。

4. 存储器访问/分支完成周期(MEM)

1) load 指令和 store 指令

如果是 load 指令,就用上一个周期计算出的有效地址从存储器中读出相应的数据;如果是 store 指令,就把指定的数据写入这个有效地址所指出的存储器单元。

2）分支指令

如果分支“成功”，就把在前一个周期中计算好的转移目标地址送入 PC。分支指令执行完成；否则，就不进行任何操作。

其他类型的指令在此周期不做任何操作。

5. 写回周期（WB）

把结果写入通用寄存器组。对于 ALU 运算指令来说，这个结果来自 ALU，而对于 load 指令来说，这个结果来自存储器。

在上述 5 个周期的实现方案中，分支指令和 store 指令需要 4 个周期，其他指令需要 5 个周期才能完成。在追求更高性能的方案中，可以把分支指令的执行提前到 ID 周期完成，这样分支指令只需要两个周期。不过，为了实现这一点，需要增设一个专门用于计算转移目标地址的加法器。

把上述实现方案改造为流水线实现是比较简单的，只要把上面的每一个周期作为一个流水段，并在各段之间加上锁存器，就构成了如图 3.17 所示的 5 段流水线。这些锁存器称为流水寄存器。如果在每个时钟周期启动一条指令，则采用流水方式后的性能将是非流水方式的 5 倍。当然，事情也没这么简单，还要解决好流水处理带来的一些问题。

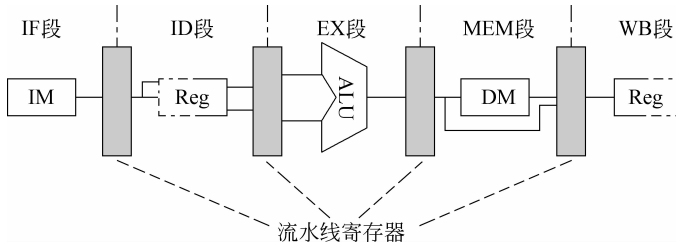


图 3.17　一条经典的 5 段流水线

首先，在流水线方式下，要保证不会在同一时钟周期要求同一个功能段做两件不同的工作。例如，不能要求 ALU 既做有效地址计算，又同时做算术运算。RISC 指令集比较简洁，所以该要求不难实现。

其次，为了避免 IF 段的访存（取指令）与 MEM 段的访存（读/写数据）发生冲突，必须采用分离的指令存储器和数据存储器，或者是仍采用一个公用的存储器，但要采用分离的指令 Cache 和数据 Cache。一般是采用后者。

第三，ID 段要对通用寄存器组进行读操作，而 WB 段要对通用寄存器组进行写操作，为了解决对同一通用寄存器的访问冲突，我们把写操作安排在时钟周期的前半拍完成，把读操作安排在后半拍完成。在图 3.17 以及后面的图中，用部件 Reg 的边框为实线来表示进行读或写操作，而虚线则表示不进行操作。

第四，图 3.17 中没有考虑 PC 的问题。为了做到每一个时钟周期启动一条指令，必须在每个时钟周期都进行 PC 值加 4 的操作。这要在 IF 段完成，为此需要设置一个专门的加法器。另外，分支指令也要修改 PC 的值，它是到 MEM 段才会进行修改的。3.4.2 节将详细讨论分支的处理问题。

需要说明的是，这里给出的方案并不是性能最好或者成本最低的，它只是用来帮助我们更好地理解指令流水线的原理和实现。

3.1.1节中介绍了流水线的时空图。为便于后面的讨论,下面介绍另外一种时空图,如图3.18所示。它是如图3.17所示的流水线的时空图。其横向与图3.2中的横坐标类似,表示的是时间。这里用的是时钟周期。但纵向和时空区中的内容与图3.2的不同。图3.2的纵坐标(向上)是各流水段,时空区中填的是所处理的任务,而图3.18中的纵向(向下)却是所执行的指令(相当于图3.2中的任务)按顺序列出,时空分区中填的是各流水段的名称。这种时空图更直观地展现了指令的重叠执行情况。图3.19是这种时空图的又一种画法,它以数据通路的快照形式更直观地展现了部件重叠工作的情况。在后面的论述中,将经常采用这两种时空图。

时钟周期 / 指令	1	2	3	4	5	6	7	8
指令k	IF	ID	EX	MEM	WB			
指令k+1		IF	ID	EX	MEM	WB		
指令k+2			IF	ID	EX	MEM	WB	
指令k+3				IF	ID	EX	MEM	WB

图3.18　5段流水线的另一种时空图

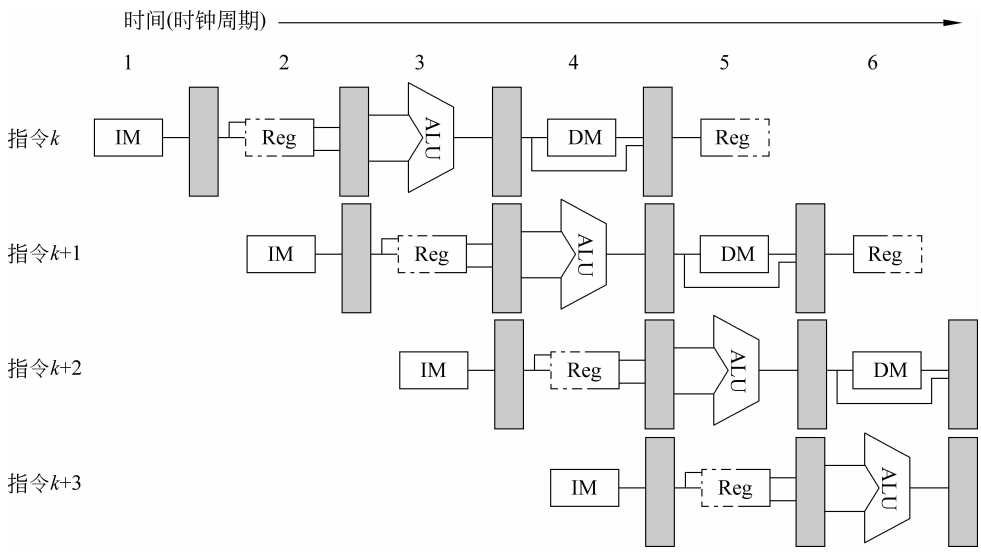


图3.19　时空图的另一种画法

3.4.2　相关与流水线冲突

1. 相关

相关(dependence)是指两条指令之间存在某种依赖关系。如果指令之间没有任何关系,那么当流水线有足够的硬件资源时,它们就能在流水线中顺利地重叠执行,不会引起任何停顿。但如果两条指令相关,它们也许就不能在流水线中重叠执行或者只能部分重叠。研究程序中指令之间存在什么样的相关,对于充分发挥流水线的效率有重要的意义。