第 5 章　指令级并行性及其开发——硬件方法

5.1　基本要求与难点

5.1.1　基本要求

（1）掌握有关指令级并行的基本概念。

（2）理解指令的静态调度和动态调度的概念，掌握动态调度的基本思想。理解精确异常和不精确异常的不同。

（3）掌握记分牌动态调度方法的基本思想。

（4）掌握 Tomasulo 算法的基本思想和算法。能画出在给定的情况下状态表的内容。

（5）理解和掌握动态分支预测技术，特别是基于硬件的前瞻执行。掌握采用前瞻执行机制后指令的执行步骤发生了哪些变化。

（6）理解超标量、超长指令字和超流水这 3 种机制的原理及特点。

（7）了解基于静态调度的多流出技术和基于动态调度的多流出技术。

（8）了解多流出处理器受到的限制。

5.1.2　难点

（1）记分牌方法。

（2）Tomasulo 算法。

（3）基于硬件的前瞻执行。

（4）超标量、超长指令字和超流水。

5.2　知识要点

指令级并行(ILP)是指指令之间存在的一种并行性，利用它，计算机可以并行执行两条或两条以上的指令。开发 ILP 的途径有两种，一种是资源重复，重复设置多个处理部件，让它们同时执行多条指令；另一种是采用流水线技术，使指令重叠并行执行。

5.2.1　指令级并行的概念

开发 ILP 的方法可以分为两大类：主要基于硬件的动态开发方法以及基于软件的静态开发方法。两者若紧密结合，则效果更好。

流水线处理机的实际 CPI 为：

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

其中，理想 CPI 是衡量流水线最高性能的一个指标。通过减少该式右边的各项，我们就能减少总的 CPI，从而提高 IPC(Instructions Per Cycle)。

如果一串连续的代码除了入口和出口以外，没有其他的分支指令和转入点，则称之为**一个基本程序块**。在基本程序块中能开发出的并行性是很有限的，为了明显地提高性能，必须跨越多个基本块开发 ILP。

最简单和最常用的开发 ILP 的方法，是开发循环的不同迭代之间存在的并行性。这种并行性称为**循环级并行性**。在第 6 章，将讨论如何把这种循环级并行性转换为 ILP。

5.2.2　相关与指令级并行

如果两条指令相关，则它们就不能并行执行，或只能部分重叠执行。

流水线冲突是指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行。流水线冲突有 3 种类型：结构冲突，数据冲突，控制冲突。结构冲突是因硬件资源冲突造成的，数据冲突是由数据相关和名相关造成的，控制冲突是由控制相关造成的。

相关是程序固有的一种属性，它反映了程序中指令之间的相互依赖关系。而具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿，则是流水线的属性。

可以从以下两个方面来解决相关问题。

(1) 保持相关，但避免发生冲突。

(2) 进行代码变换，消除相关。

指令调度是一种用来避免冲突的主要方法，它并不改变相关。

程序顺序是指：由原来程序确定的在完全串行方式下指令的执行顺序。并不需要在所有存在相关的地方都保持程序顺序。后面要介绍的各种软硬件技术的目标是尽可能地开发并行性，只有在可能会导致错误的情况下，才保持程序顺序。

对于提高性能来说，控制相关本身并不是一个主要的限制。它并不是一个必须严格保持的关键属性。为了保证程序执行的正确性，必须保持的最关键的两个属性是：**数据流**和**异常行为**。

保持异常行为是指：无论怎么改变指令的执行顺序，都不能改变程序中异常的发生情况。即原来程序中是怎么发生的，改变执行顺序后还是怎么发生。这个条件经常被弱化为：指令执行顺序的改变不能导致程序中发生新的异常。

数据流是指数据值从其产生者指令到其消费者指令的实际流动。分支指令使得数据流具有动态性，因为一条指令有可能数据相关于多条先前的指令。分支指令的执行结果决定

了哪条指令真正是所需数据的产生者。

后面将讨论的**前瞻执行**不仅能解决异常问题,而且能够使我们在保持数据流的情况下,减少控制相关对开发ILP的影响。

5.2.3 指令的动态调度

静态调度的流水线依靠编译器对代码进行静态调度,以减少相关和冲突。之所以称之为**静态调度**,是因为它不是在程序执行的过程中而是在编译期间进行代码调度和优化的。静态调度通过把相关的指令拉开"距离"来减少可能产生的停顿。

动态调度能在保持数据流和异常行为的情况下,通过硬件对指令执行顺序进行重新安排,减少数据相关导致的停顿。动态调度有许多优点:①能够处理一些编译时情况不明的相关(比如涉及存储器访问的相关),并简化了编译器;②能够使本来是面向某一流水线优化编译的代码在其他动态调度的流水线上也能高效地执行。当然,动态调度的这些优点是以硬件复杂性的显著增加为代价的。

1. 动态调度的基本思想

第3章中讨论的5段流水线有一个很大的局限性,即其指令是按序流出和按序执行的。如果相近的指令存在相关,就很可能会导致冲突,引起停顿。这样其后面所有的指令也都停止了前进。

为了解决这个问题,我们将前述5段流水线的译码(ID)段细分为以下两个段。

(1) 流出:指令译码,并检查是否存在结构冲突。如果不存在结构冲突,就将指令流出。

(2) 读操作数:等待数据冲突消失(如果有),然后读操作数,并立即开始执行。

可以看出,这样修改后指令的流出仍然是按序流出。但是,它们在读操作数段可能停顿或互相跨越,因而进入执行段时就可能已经是乱序的了,即**乱序执行**。指令的完成也是**乱序完成的**。

指令乱序完成大大增加了异常处理的复杂度。动态调度的处理机是这样来保持正确的异常行为的:对于一条会产生异常的指令来说,只有当处理机确切地知道该指令将被执行时,才允许它产生异常。

即使保持了正确的异常行为,动态调度处理机仍可能发生不精确异常。所谓**不精确异常**是指:当执行指令 i 导致发生异常时,处理机的现场(状态)与严格按程序顺序执行时指令 i 的现场不同。反之,如果发生异常时,处理机的现场跟严格按程序顺序执行时指令 i 的现场相同,就称为**精确异常**。不精确异常使得在异常处理后难以接着继续执行程序。

之所以会发生不精确的异常,是因为当发生异常(设为指令 i)时:①流水线可能已经执行完按程序顺序是位于指令 i 之后的指令;②流水线可能还没完成按程序顺序是指令 i 之前的指令。

记分牌方法和Tomasulo算法是两种比较典型的动态调度算法。下面先简单介绍记分牌方法,然后详细论述Tomasulo算法。许多现代处理机都采用了Tomasulo算法或其变形。

2. 记分牌动态调度方法

1) 基本思想

记分牌方法这一名称起源于最早采用此功能的 CDC 6600 计算机中的记分牌。该机器用一个称为记分牌的硬件实现了对指令的动态调度。它把前述简单流水线中的译码段 ID 分解成了两个段：流出和读操作数，以避免当某条指令在 ID 段被停顿时挡住其后面无关指令的流动。

记分牌的目标是在没有结构冲突时，尽可能早地执行没有数据冲突的指令，实现每个时钟周期执行一条指令。如果某条指令被暂停，而后面的指令与流水线中正在执行或被暂停的指令都不相关，那么这些指令可以跨越它，继续流出和执行下去。记分牌全面负责和管理这些指令的流出和执行，当然也包括检测所有的冲突。

每条指令都要经过记分牌。指令流出时，记分牌在表中记录相关的信息，并决定什么时候该指令可以读出操作数和开始执行。如果确定该指令不能马上执行，记分牌在后面就会监视硬件中信息的每一个变化，一旦所需的操作数就绪，就立即启动该指令的执行。

在采用了记分牌的流水线中。每条指令的执行过程分为 4 段：流出、读操作数、执行和写结果。由于我们主要考虑浮点操作，运算在浮点寄存器之间进行，因此不涉及存储器访问段。

(1) 流出

如果当前流出指令所需的功能部件空闲，并且所有其他正在执行的指令的目的寄存器与该指令的不同，记分牌就向功能部件流出该指令，并修改记分牌内部的记录表。如果存在结构相关或 WAW 冲突，该指令就不流出。在这里就解决了 WAW 冲突。

(2) 读操作数

记分牌监测源操作数的可用性，一旦数据可用，它就通知功能部件从寄存器中读出源操作数并开始执行。这一步动态地解决了 RAW 冲突，并可能导致指令乱序执行。

(3) 执行

取到操作数后，功能部件开始执行。当结果产生后，就通知记分牌它已经完成执行。这一步相当于前述标准流水线中的执行段(EX)。在浮点流水线中，这一段可能要占用多个时钟周期。

(4) 写结果

记分牌一旦知道执行部件完成了执行，就检测是否存在 WAR 冲突。如果不存在，或者已有的 WAR 冲突已消失，记分牌就通知功能部件把结果写入目的寄存器，并释放该指令使用的所有资源。这一步对应于前述标准流水线中的写回段(WB)。

记分牌中记录的信息由以下三部分构成。

① 指令状态表：记录正在执行的各条指令已经进入到了哪一段。

② 功能部件状态表：记录各个功能部件的状态。每个功能部件有一项，每一项由 9 个字段组成。

③ 结果寄存器状态表 Result：每个寄存器在该表中有一项，用于指出哪个功能部件(编号)将把结果写入该寄存器。

结果寄存器状态表中的字段与每个寄存器一一对应，它记录了当前机器状态下将把结

果写入该寄存器的功能部件的名称。

2) 具体算法(略)

3. Tomasulo 算法

1) 基本思想

Tomasulo 算法是由 Robert Tomasulo 发明的,因而以他的名字命名。首先采用这种方法的是 IBM 360/91 机器中的浮点部件。尽管许多现代处理机采用了这种方法的各种变形,但其核心思想都是:①记录和检测指令相关,操作数一旦就绪就立即执行,把发生 RAW 冲突的可能性减小到最少;②通过寄存器换名来消除 WAR 冲突和 WAW 冲突。

寄存器换名可以消除 WAR 冲突和 WAW 冲突。

下面在 MIPS 指令集的情况下来介绍 Tomasulo 算法,我们重点关心的是浮点部件以及 load/store 部件。

在 Tomasulo 算法中,寄存器换名是通过保留站和流出逻辑来共同完成的。当指令流出时,如果其操作数还没有计算出来,则将该指令中相应的寄存器号换名为将产生这个操作数的保留站的标识。所以,指令流出到保留站后,其操作数寄存器号或者换成了数据本身(若已就绪),或者换成了保留站的标识,不再与寄存器有关系。这样后面指令对该寄存器的写入操作就不可能产生 WAR 冲突了。

在详细介绍 Tomasulo 算法之前,先来看一下指令执行的步骤。这里只需要以下三步。

(1) 流出

从指令队列的头部取一条指令。如果该指令的操作所要求的保留站有空闲的,就把该指令送到该保留站(设为 r)。并且,如果其操作数在寄存器中已经就绪,就将这些操作数送入保留站 r。如果其操作数还没有就绪,就把将产生该操作数的保留站的标识送入保留站 r。这样,一旦被记录的保留站完成计算,它将直接把数据送给保留站 r。这一步实际上是进行了寄存器换名(换成保留站的标识)和对操作数进行缓冲,消除了 WAR 冲突。另外,还要完成对目的寄存器的预约工作,将其设置为接收保留站 r 的结果。这实际上相当于提前完成了写操作(预约)。由于指令是按程序顺序流出的,当出现多条指令写同一个结果寄存器时,最后留下的预约结果肯定是最后一条指令的,就是说消除了 WAW 冲突。

当然,如果没有空闲的保留站,指令就不能流出。这是发生了结构冲突。

(2) 执行

如果某个操作数还没有被计算出来,本保留站将监视 CDB,等待所需的计算结果。一旦那个结果产生,它将被放到 CDB 上,本保留站将立即获得该数据。当两个操作数都就绪后,本保留站就用相应的功能部件开始执行指令规定的操作。这里是等到所有操作数都备齐后才开始执行指令。也就是说是靠推迟执行的方法解决 RAW 冲突。由于结果数据是从其产生的部件(保留站)直接送到需要它的地方,所以这已经是最大限度地减少了 RAW 冲突的影响。

load 和 store 指令的执行需要两个步骤:计算有效地址(要等到基地址寄存器就绪)和把有效地址放入 load 或 store 缓冲器。load 缓冲器中的 load 指令的执行条件是存储器部件就绪。而 store 缓冲器中的 store 指令在执行前必须等到要存入存储器的数据到达。通过按顺序进行有效地址计算来保证程序顺序,这有助于避免访问存储器的冲突。

(3) 写结果

功能部件计算完毕后,就将计算结果放到 CDB 上,所有等待该计算结果的寄存器和保留站(包括 store 缓冲器)都同时从 CDB 上获得所需要的数据。store 指令在这一步完成对存储器的写入:当写入地址和数据都备齐时,将它们送给存储器部件,store 指令完成。

保留站、寄存器组和 load/store 缓冲器都包含附加标志信息,用于检测和消除冲突。不同部件的附加信息略有不同。标识字段实际上就是用于换名的一组虚拟寄存器的名称(编号)。

与在 Tomasulo 算法之前提出的其他更简单的动态调度方法相比,Tomasulo 算法具有以下两个主要的优点。

① 冲突检测逻辑和指令执行控制是分布的(通过保留站和 CDB 实现)。

每个功能部件的保留站中的信息决定了什么时候指令可以在该功能部件开始执行。如果有多条指令已经获得了一个操作数,并同时在等待同一运算结果(作为另一个操作数),那么这个结果一产生,就可以通过 CDB 同时播送给所有这些指令,使它们可以同时执行。

② 消除了 WAW 冲突和 WAR 冲突导致的停顿。

这是通过使用保留站进行寄存器换名,并且在操作数一旦就绪就将之放入保留站来实现的。

2) 具体算法

下面给出 Tomasulo 算法中指令进入各阶段的条件以及在各阶段进行的操作和状态表内容修改。其中,各符号的意义如下。

r:分配给当前指令的保留站或者缓冲器单元(编号);

rd:目的寄存器编号;

rs、rt:操作数寄存器编号;

imm:按符号位扩展后的立即数;

RS:保留站;

result:浮点部件或 load 缓冲器返回的结果;

Qi:寄存器状态表;

Regs[]:寄存器组;

Op:当前指令的操作码。

对于 load 指令来说,rt 是保存所取数据的寄存器号;对于 store 指令来说,rt 是保存所要存储的数据的寄存器号。与 rs 对应的保留站字段是 Vj,Qj;与 rt 对应的保留站字段是 Vk,Qk。

(1) 指令流出

① 浮点运算指令

进入条件:有空闲保留站(设为 r)
操作和状态表内容修改:

```
if  (Qi[rs] ≠ 0)              //检测第一操作数是否就绪
   {RS[r].Qj ← Qi[rs]         //第一操作数没有就绪,进行寄存器换名,即把将产生该操作数的保
                              //留站的编号放入当前保留站的 Qj。该编号是一个大于 0 的整数
else {RS[r].Vj ← Regs[rs]; //第一操作数就绪。把寄存器 rs 中的操作数取到当前保留站的 Vj
   RS[r].Qj ← 0};             //置 Qj 为 0,表示当前保留站的 Vj 中的操作数就绪
```

```
if  (Qi[rt] ≠ 0)              //检测第二操作数是否就绪
   {RS[r].Qk ← Qi[rt]}        //第二操作数没有就绪,进行寄存器换名,即把将产生该操作数的保
                              //留站的编号放入当前保留站的 Qk。该编号是一个大于 0 的整数
else {RS[r].Vk← Regs[rt];     //第二操作数就绪。把寄存器 rt 中的操作数取到当前保留站的 Vk
   RS[r].Qk← 0};              //置 Qk 为 0,表示当前保留站的 Vk 中的操作数就绪
RS[r].Busy ← yes;             //置当前保留站为"忙"
RS[r].Op ← Op;                //设置操作码
Qi[rd] ← r;                   //把当前保留站的编号 r 放入 rd 所对应的寄存器状态表项,
                              //以便 rd 将来接收结果
```

② load 和 store 指令

进入条件:缓冲器有空闲单元(设为 r)

操作和状态表内容修改:

```
if  (Qi[rs] ≠ 0)              //检测第一操作数是否就绪。
   {RS[r].Qj ← Qi[rs]}        //第一操作数没有就绪,进行寄存器换名,即把将产生该操作数的保
                              //留站的编号存入当前缓冲器单元的 Qj
else
  {RS[r].Vj ← Regs[rs];       //第一操作数就绪,把寄存器 rs 中的操作数取到当前缓冲器单元
                              //的 Vj
   RS[r].Qj ← 0};             //置 Qj 为 0,表示当前缓冲器单元的 Vj 中的操作数就绪
RS[r].Busy ← yes;             //置当前缓冲器单元为"忙"
RS[r].A ← Imm;                //把按符号位扩展后的偏移量放入当前缓冲器单元的 A
```

对于 load 指令:

```
Qi[rt] ← r;                   //把当前缓冲器单元的编号 r 放入 load 指令的目的寄存器 rt 所对应
                              //的寄存器状态表项,以便 rt 将来接收所取的数据
```

对于 store 指令:

```
if  (Qi[rt] ≠0)               //检测要存储的数据是否就绪
   {RS[r].Qk ← Qi[rt]}        //该数据尚未就绪,进行寄存器换名,即把将产生该数据的保留站的
                              //编号放入当前缓冲器单元的 Qk
else
  {RS[r].Vk ← Regs[rt];       //该数据就绪,把它从寄存器 rt 取到 store 缓冲器单元的 Vk
    RS[r].Qk ← 0};            //置 Qk 为 0,表示当前缓冲器单元的 Vk 中的数据就绪
```

(2) 执行

① 浮点操作指令

进入条件:(RS[r].Qj = 0)且(RS[r].Qk = 0); //两个源操作数就绪

操作和状态表内容修改:进行计算,产生结果

② load/store 指令

进入条件:(RS[r].Qj = 0)且 r 成为 load/store 缓冲队列的头部

操作和状态表内容修改:

```
RS[r].A ← RS[r].Vj + RS[r].A;          //计算有效地址
```

对于 load 指令,在完成有效地址计算后,还要进行:

```
从 Mem[RS[r].A]读取数据;               //从存储器中读取数据
```

(3) 写结果

① 浮点运算指令和 load 指令

进入条件:保留站 r 执行结束,且 CDB 就绪

操作和状态表内容修改：

```
∀x  (if(Qi[x] = r)                  //对于任何一个正在等该结果的浮点寄存器 x,
  { Regs[x] ← result;               //向该寄存器写入结果
  Qi[x] ← 0};                       //把该寄存器的状态置为数据就绪
∀x  (if(RS[x].Qj = r)               //对于任何一个正在等该结果作为第一操作数的保留站 x,
  {RS[x].Vj ← result;               //向该保留站的 Vj 写入结果
  RS[x].Qj ← 0});                   //置 Qj 为 0,表示该保留站的 Vj 中的操作数就绪
∀x  (if(RS[x].Qk = r)               //对于任何一个正在等该结果作为第二操作数的保留站 x,
  {RS[x].Vk ← result;               //向该保留站的 Vk 写入结果
  RS[x].Qk ← 0});                   //置 Qk 为 0,表示该保留站的 Vk 中的操作数就绪
RS[r].Busy ← no;                    //释放当前保留站,将之置为空闲状态
```

② store 指令

进入条件：保留站 r 执行结束，且 RS[r].Qk = 0　//要存储的数据已经就绪

操作和状态表内容修改：

```
Mem[RS[r].A]← RS[r].Vk              //数据写入存储器,地址由 store 缓冲器单元的 A 字段给出
RS[r].Busy← no;                     //释放当前缓冲器单元,将之置为空闲状态
```

如果能够准确地预测分支，采用 Tomasulo 算法将获得很高的性能。这种方法的主要缺点是其复杂性，实现它需要大量的硬件。所以在单流出的流水线中，采用 Tomasulo 算法所带来的好处与所花的代价相比不一定值得。但是，对于多流出的处理机来说，随着流出能力的提高，寄存器换名以及动态调度技术就变得越来越重要了。特别是，Tomasulo 算法还是硬件前瞻执行的基础，因此该算法得到了广泛的应用。

5.2.4　动态分支预测技术

开发的 ILP 越多，控制相关的制约就越大，就要求分支预测有更高的准确度。本节中介绍的方法对于每个时钟周期流出多条指令的处理机来说是非常重要的，这是因为：①在 n-流出（每个时钟周期流出 n 条指令）的处理机中，遇到分支指令的可能性增加了 n 倍。要给处理器连续提供指令，就需要预测分支的结果。②Amdahl 定律告诉我们，机器的 CPI 越小，控制停顿的相对影响就越大。

采用这些动态分支预测技术的目的有两个：预测分支是否成功和尽快找到分支目标地址（或指令），从而避免控制相关造成流水线停顿。在这些方法中，需要解决以下关键问题：①如何记录分支的历史信息；②如何根据这些信息来预测分支的去向，甚至提前取出分支目标处的指令。

1. 采用分支历史表 BHT

分支历史表（Branch History Table，BHT）法是最简单的动态分支预测方法。它用 BHT 来记录相关分支指令的“历史”，并据此进行预测。这个“历史”是指最近一次或几次的执行情况是成功还是失败。常采用两位二进制位来记录历史。有研究结果表明，两位分支预测的性能与多位（两位以上）分支预测的性能差不多。因而大多数处理机是采用两位分支预测。

研究结果表明，对于 SPEC89 测试程序来说，大小为 4K 的 BHT 的预测准确率为

82%～99%,并且与大小为无穷大的BHT的准确率相近。所以一般来说,采用4K的BHT就可以了。

2. 采用分支目标缓冲器BTB

在多流出的处理机中,只准确地预测分支还不够,还要能够快速地提供足够的指令流。许多现代的处理器都要求每个时钟周期能提供4～8条指令。这需要尽早知道分支是否成功,尽早知道分支目标地址,尽早获得分支目标指令。

对于前述5段流水线来说,BHT方法是在ID段对BHT进行访问,所以在ID段的末尾,能够获得分支目标地址(在ID段计算出)、顺序下一条指令地址以及预测的结果。如果能再提前一拍,即在IF段就知道这些信息,那么分支开销就可以减少为0。BTB能够实现这一点。BTB是Branch Target Buffer的缩写,其中文名称是**分支目标缓冲器**。BTB有时也称为分支目标Cache。

BTB可以看成是用专门的硬件实现的一张表格。表格中的每一项至少有两个字段:①执行过的成功分支指令的地址;②预测的分支目标地址。以第一个字段作为该表的匹配标识。在每次取指令的同时,用该指令的地址与BTB中所有项目的第一个字段进行比较。如果有匹配的,我们就知道该指令是分支指令且上一次执行是分支成功,据此我们预测这次执行也将分支成功,其分支目标地址由匹配项的第二个字段给出。如果没有匹配的,就把当前指令当作普通的指令(即不是分支指令)来执行。

当采用BTB后,如果当前指令的地址与BTB中的第一字段匹配,则从分支目标处开始取指令。如果预测正确,则不会有任何分支延迟。如果预测错误或者在BTB中没有匹配的项,则至少会有两个时钟周期的开销。

BTB的另外两种形式是在分支目标缓冲器中增加一个字段来存放BHT表,或者增加一个字段来存放若干条分支目标处的指令。后者可以一次性提供分支目标处的多条指令,这对于多流出处理器来说是很有必要的。

3. 基于硬件的前瞻执行

对于多流出的处理机来说,控制相关已经成了开发更多ILP的一个主要障碍。**前瞻执行**能很好地解决控制相关的问题,它对分支指令的结果进行猜测,然后按这个猜测结果继续取指、流出和执行后续的指令。只是指令执行的结果不是写回到寄存器或存储器,而是放到一个称为**ROB**(ReOrder Buffer)的缓冲器中。等到相应的指令得到"确认"(即确实是应该执行的)后,才将结果写入寄存器或存储器。之所以要这样,是为了在猜测错误的情况下能够恢复原来的现场

基于硬件的前瞻执行是把3种思想结合在了一起。

(1) 动态分支预测。用来选择后续执行的指令。

(2) 在控制相关的结果尚未出来之前,前瞻地执行后续指令。

(3) 用动态调度对基本块的各种组合进行跨基本块的调度。

对Tomasulo算法加以扩充,就可以支持前瞻执行。当然,硬件也需要做相应的扩展。在Tomasulo算法中,写结果和指令完成是一起在"写结果"段完成的。现在我们把该段分为"写结果"和"指令确认"两个段。"**写结果**"段是把前瞻执行的结果写到ROB中,并通过

CDB 在指令之间传送结果，供需要用到这些结果的指令使用。**“指令确认”**段是在分支指令的结果出来后，对相应指令的前瞻执行给予确认，把在 ROB 中的结果写到寄存器或存储器（如果前面所做的猜测是对的）。如果发现前面对分支结果的猜测是错误的，那就不予以确认，并从那条分支指令的另一条路径开始重新执行。

前瞻执行允许指令乱序执行，但要求按程序顺序确认。并且在指令被确认之前，不允许它进行不可恢复的操作，如更新机器状态或发生异常。

支持前瞻执行的浮点部件的结构与基于 Tomasulo 算法的 MIPS 处理器浮点部件的基本结构相比，主要是增加了一个 ROB 缓冲器。ROB 是为实现前瞻执行而设置的，它在指令操作完成后到指令被确认这段时间，为指令保存数据。正在前瞻执行的指令之间也是通过 ROB 传送结果的，因为前瞻执行的指令在被确认前是不能写寄存器的。

ROB 中的每一项由以下 4 个字段组成。

（1）指令类型：指出该指令是分支指令、store 指令或寄存器操作指令。

（2）目的地址：给出指令执行结果应写入的目的寄存器号（如果是 load 和 ALU 指令）或存储器单元的地址（如果是 store 指令）。

（3）数据值字段：用来保存指令前瞻执行的结果，直到指令得到确认。

（4）就绪字段：指出指令是否已经完成执行并且数据已就绪。

在前瞻执行机制中，Tomasulo 算法中保留站的换名功能是由 ROB 来完成的。但在指令流出到开始执行期间，仍然需要有地方来存放运算操作码和操作数。这个功能仍由保留站来完成。由于每条指令在被确认前，在 ROB 中都有相应的一项，所以对于执行结果我们是用 ROB 项的编号作为标识，而不像 Tomasulo 算法那样是用保留站的编号。这就要求在保留站中记录分配给该指令的 ROB 项编号。

采用前瞻执行机制后，指令的执行步骤如下。

1）流出

从浮点指令队列的头部取一条指令，如果有空闲的保留站（设为 r）且有空闲的 ROB 项（设为 b），就流出该指令，并把相应的信息放入保留站 r 和 ROB 项 b。即：如果该指令需要的操作数已经在寄存器或 ROB 中就绪，就把它（们）送入保留站 r 中。修改 r 和 b 的控制字段，表示它们已经被占用。ROB 项 b 的编号也要放入保留站 r，以便当该保留站的执行结果被放到 CDB 上时可以用它作为标识。如果保留站或 ROB 全满，便停止流出指令，直到它们都有空闲的项。

2）执行

如果有操作数尚未就绪，就等待，并不断地监测 CDB。这一步检测 RAW 冲突。当两个操作数都已在保留站中就绪后，就可以执行该指令的操作。load 指令的操作还是分两步完成（有效地址计算和读取数据），store 指令在这一步只进行有效地址计算。

3）写结果

当结果产生后，将该结果连同本指令在流出段所分配到的 ROB 项的编号放到 CDB 上，经 CDB 写到 ROB 以及所有等待该结果的保留站。然后释放产生该结果的保留站。store 指令的操作有些特殊（与 Tomasulo 算法不同）：如果要写入存储器的数据已经就绪，就把该数据写入分配给该 store 指令的 ROB 项。否则，就监测 CDB，直到那个数据在 CDB 上播送出来，才将其写入分配给该 store 指令的 ROB 项。

4) 确认

这一阶段对分支指令、store指令以及其他指令的处理不同。

(1) 对于除分支指令和store指令以外的指令来说,当该指令到达ROB队列的头部而且其结果已经就绪时,就把该结果写入该指令的目的寄存器,并从ROB中删除该指令。

(2) 对store指令的处理与(1)类似,只是它是把结果写入存储器。

(3) 当预测错误的分支指令到达ROB队列的头部时,就表示是错误的前瞻执行。这时要清空ROB,并从分支指令的另一个分支重新开始执行。

(4) 当预测正确的分支指令到达ROB队列的头部时,该指令执行完毕。

一旦指令得到确认,就释放它所占用的ROB项。当ROB满时,就停止指令的流出,直到有空闲项被释放出来。

由于前瞻执行通过ROB实现了指令的顺序完成,所以它不仅能够进行前瞻执行,而且能够实现精确异常。

前瞻执行的主要缺点是:所需的硬件太复杂。与Tomasulo算法相比,在控制方面复杂多了,因而在控制逻辑硬件方面增加了许多。

5.2.5 多指令流出技术

在单流出的情况下,CPI不可能小于1。如果想进一步提高性能,使CPI小于1,就必须采用**多流出技术**,在每个时钟周期流出多条指令。

多流出处理机有两种基本风格:超标量和超长指令字(Very Long Instruction Word,VLIW)。**超标量**在每个时钟周期流出的指令条数不固定,依代码的具体情况而定,不过有个上限。如果这个上限为n,就称该处理机为**n-流出**。对于超标量处理机,既可以通过编译器进行静态调度,也可以基于Tomasulo算法进行动态调度。静态调度的超标量处理机一般采用按序执行,而动态调度的处理机一般采用乱序执行。

与超标量处理机不同,超长指令字VLIW处理机在每个时钟周期流出的指令条数是固定的,这些指令构成一条长指令或者一个指令包。在这个指令包中,指令之间的并行性是通过指令显式地表示出来的。这种处理机的指令调度由编译器静态完成。

与VLIW处理机相比,超标量处理机有两个优点。

(1) 超标量结构对程序员是透明的,处理机能自己检测下一条指令是否能流出,不需要由编译器或专门的变换程序对程序中的指令进行重新排列。

(2) 即使是没有经过编译器针对超标量结构进行调度优化的代码或是旧的编译器生成的代码也可以运行,当然运行的效果不会很好。要想达到很好的效果,方法之一就是使用动态超标量调度技术。

1. 基于静态调度的多流出技术

在静态调度的超标量处理机中,指令按序流出。所有的冲突检测都在流出时进行,由硬件检测当前流出的指令之间是否存在冲突以及它们与正在执行的指令之间是否有冲突。如果在当前流出的指令序列中,某条指令存在冲突,那么就只流出该指令之前的指令。

考虑一个4-流出的静态调度超标量处理机。在取指令阶段,流水线将从取指令部件收

到 1～4 条指令，流出部件通过检测冲突来确定这些指令是全部流出还是部分流出。由于这些检测比较复杂，难以在一个时钟周期内完成，所以许多静态调度的超标量处理机都是将其分成多个流水段，按流水方式工作。动态调度的超标量处理机中也是如此。

如果 MIPS 处理机按超标量方式工作，结果将会怎样？我们假设每个时钟周期可以流出两条指令："1 条整数型指令＋1 条浮点操作指令"，其中把 load 指令、store 指令、分支指令也归类为整数型指令。与任意的双流出相比，把整数指令和浮点指令结合流出是简单了不少，对硬件的要求也没那么高。

为了实现每个时钟周期流出两条指令，显然是要能够同时取两条指令(64 位)，也要能同时译码两条指令(64 位)。对指令的处理包括以下步骤：①从 Cache 中取两条指令；②确定哪几条指令可以流出(0～2 条指令)；③把它们发送到相应的功能部件。取两条指令还比较容易实现，若要取更多的指令，所要进行的处理就复杂多了。

对于上述简单的超标量处理机来说，冲突检测还比较简单，因为"1 条整数型指令＋1 条浮点指令"的流出方式消除了大多数流出包内的冲突。主要的难点出现在当整数型指令是一条浮点 load、store 或 move 指令的情况。这时有可能会出现争用浮点寄存器端口或者产生新的 RAW 冲突。采用"1 条整数型指令＋1 条浮点指令"并行流出的方式，需要增加的硬件很少。这是因为整数指令和浮点指令使用不同的寄存器组和不同的功能部件。

为了能有效地利用超标量处理机所具有的并行性，需要采用更有效的编译技术或者硬件调度技术。如果不采用这些技术，超标量技术所能带来的性能上的提高可能很有限。

2. 基于动态调度的多流出技术

在多流出处理机中，动态调度技术是提高性能的一种方法。动态调度不仅拥有能解决数据冲突和提高性能的典型优点，而且有可能克服指令流出所受的限制。尽管从硬件的角度来看，在每个时钟周期最多只能启动一个整数操作和一个浮点操作的执行，但动态调度可以使得在指令流出时不受这个限制，至少在保留站被全部占用之前是如此。

假设我们要对 Tomasulo 算法进行扩展，使之能支持双流出超标量流水线，但又不想乱序地向保留站流出指令，因为这会破坏程序语义。为了充分利用动态调度的好处，我们也许应该去掉每个时钟周期只能流出"1 条整数型指令＋1 条浮点指令"的限制，但这会大大增加指令流出的硬件复杂度。

在采用动态调度的处理机中，有两种不同的方法可以用来实现多流出。它们都是建立在这样一个观点之上的：动态调度关键在于对保留站的分配和对流水线控制表格的修改。一种方法是在半个时钟周期内完成流出步骤，这样一个时钟周期就能处理两条指令。另一种方法是设置一次能同时处理两条指令的逻辑电路。现代的流出 4 条或 4 条以上指令的超标量处理机经常是综合采用这两种方法，即：不仅采用流水，而且还把流出电路加宽。

3. 超长指令字技术

下面只做简单的介绍。更详细的论述见第 6 章。

超长指令字(Very Long Instruction Word，VLIW)技术是另一种多指令流出技术。与超标量不同，它在指令流出时不需要进行复杂的冲突检测，而是依靠编译器全部安排好了。在编译时，编译器找出指令之间潜在的并行性，并通过指令调度把可能出现的数据冲突减少

到最少,最后把能并行执行的多条指令组装成一条很长的指令。这种指令字经常是一百多位到几百位,超长指令字因此得名。

在VLIW处理机中一般设置有多个功能部件。相应地,指令字也被分割成一些字段,每个字段称为一个操作槽,直接独立地控制一个功能部件。为了使功能部件充分忙碌,程序指令序列中应有足够的并行性,从而尽量填满每个操作槽。这种并行性是完全依靠编译器来挖掘的。它不需要超标量处理机中用于指令流出控制的硬件,因而控制硬件比较简单。特别是当流出宽度增加时,VLIW技术的优点更加明显。

当然,VLIW也存在一些问题,包括:

(1) 程序代码长度增加了;

(2) 采用了锁步机制;

(3) 机器代码的不兼容性。

4. 多流出处理器受到的限制

指令多流出处理器的流出能力主要受以下3个方面的影响。

(1) 程序所固有的指令级并行性;

(2) 硬件实现上的困难;

(3) 超标量和超长指令字处理器固有的技术限制。

其中,第一个限制是最简单的也是最根本的因素。对于流水线处理器,需要有大量可并行执行的操作才能避免流水线出现停顿。如果浮点流水线的延迟为5个时钟周期,要使该浮点流水线不停顿,就必须有5条无相关的浮点指令。通常情况下,所需要的无相关指令数等于流水线的深度乘以可以同时工作的功能部件数。

第二个限制,是多流出的处理器需要大量的硬件资源。因为每个时钟周期不仅要流出多条指令,而且要执行它们。随着每个时钟周期流出指令数的增加,所需的硬件成正比例地增长,所需的存储器带宽和寄存器带宽也大大增加了,这样的带宽要求必然导致大量增加硅片面积,加大面积就导致时钟频率下降、功耗增加、可靠性降低等一系列问题。

如果要使流出指令的数目增加,就需要进一步增加更多的存储器端口。多端口、层次化的存储系统带来的系统复杂性和访问延迟,可能是指令多流出技术所面临的最严重的硬件的限制。

多指令流出所需的硬件量随实现方法的不同有很大的差别。一个极端是动态调度的超标量处理器,无论采用计分牌技术还是Tomasulo算法,都需要大量的硬件,而且动态调度也大大增加了设计的复杂性,使提高时钟频率更加困难。另一个极端是VLIW处理器,指令的流出和调度仅需要很少甚至不需要额外的硬件,因为这些工作全都由编译器进行。这两种极端之间是现存的多数超标量处理器,它们将编译器的静态调度和硬件的动态调度机制结合起来,共同决定可同时并行流出的指令数。

5. 超流水线处理机

在第3章介绍的流水处理机中,是把一条指令的执行过程分解为取指令、译码、执行、访存、写结果5个流水段。如果把其中的每个流水段进一步细分,例如,分解为两个延迟时间更短的流水段,则一条指令的执行过程就要经过10个流水段。这样,在一个时钟周期内,取

指令、译码、执行、访存、写结果等各段都在处理各自的两条指令。这种在一个时钟周期内能够分时流出多条指令的处理机称为**超流水线处理机**。

与超标量处理机不同，超流水线处理机只需增加少量硬件，是通过各部分硬件的充分重叠工作来提高性能的。超标量处理机采用的是空间并行性，而超流水线处理机采用的是时间并行性。

对于一台每个时钟周期能流出 n 条指令的超流水线计算机来说，这 n 条指令不是同时流出的，而是每隔 $1/n$ 个时钟周期流出一条指令。因此，实际上该流水线的工作周期是系统时钟周期的 $1/n$。

在有的资料中，把指令流水线级数为 8 或 8 以上的流水线处理机称为超流水线处理机。

习　　题

1. 概念题

【题 5.1】 解释下列名词

指令级并行	IPC	循环级并行性	指令的动态调度
指令的静态调度	不精确异常	精确异常	CDB
BHT	分支目标缓冲	ROB	超标量
超流水	超长指令字		

2. 填空题

【题 5.2】 开发指令级并行的方法主要有两类：基于硬件的_____方法以及基于软件的_____方法。

【题 5.3】 如果一串连续的代码除了入口和出口以外，没有其他的分支指令和转入点，则称之为一个_____。

【题 5.4】 说出两种比较典型的动态调度算法：_____和_____。

【题 5.5】 要扩充 Tomasulo 算法支持前瞻执行，需将 Tomasulo 算法中的“写结果”段分为_____和_____两个段。

【题 5.6】 前瞻执行允许指令_____执行，但要求按_____确认。

【题 5.7】 Tomasulo 算法中换名功能是由_____来完成；而在前瞻执行机制中，换名功能是由_____来完成的。

【题 5.8】 静态指令调度技术是优化的_____来完成，其基本思想是重排指令序列，拉开具有_____的有关指令间的距离。

【题 5.9】 动态分支预测的依据是从_____指令过去的行为来预测它将来的行为，即根据近期转移是否成功的_____记录，来预测下一次转移的_____。

【题 5.10】 多流出处理机有_____和_____两种基本风格。

3. 问答题

【题 5.11】 简述开发指令级并行 ILP 的两种途径。

【题5.12】 为了保证程序执行的正确性,必须保持哪两个最关键的属性?并简述其含义。

【题5.13】 指令的动态调度有何优点?

【题5.14】 记分牌算法中,记分牌中记录的信息由哪三部分构成?

【题5.15】 简述Tomasulo算法的基本思想。

【题5.16】 简述Tomasulo算法中,指令流出段所做的主要工作。

【题5.17】 与在Tomasulo算法之前提出的其他更简单的动态调度方法相比,Tomasulo算法具有哪两个主要的优点?

【题5.18】 采用动态分支预测技术的目的是什么?在所采用的方法中,需要解决哪些关键问题?

【题5.19】 给出采用分支目标缓冲器BTB后,在流水线3个阶段(IF段、ID段、EX段)所进行的相关操作。

【题5.20】 基于硬件的前瞻执行是把哪3种思想结合在了一起?

【题5.21】 ROB中的每一项由哪4个字段组成?并简述其含义。

【题5.22】 简述采用前瞻执行机制后,指令确认段所做的主要工作。

【题5.23】 与VLIW处理机相比,超标量处理机有什么优点?

【题5.24】 指令多流出处理器的流出能力主要受哪3个方面的影响?

4. 应用题

【题5.25】 有A、B、C、D 4个存储器操作数,要求完成$(A\times B)+(C+D)$的运算,原来使用的程序如下:

```
I1  LOAD  R1,M(A)          ;R1←M(A)
I2  LOAD  R2,M(B)          ;R2←M(B)
I3  MUL   R5,R1,R2         ;R5←(R1)*(R2)
I4  LOAD  R3,M(C)          ;R3←M(C)
I5  LOAD  R4,M(D)          ;R4←M(D)
I6  ADD   R2,R3,R4         ;R2←(R3)+(R4)
I7  ADD   R2,R2,R5         ;R2←(R2)+(R5)
```

现采用静态指令调度方法,请写出该程序调度后的指令序列。

【题5.26】 假定有多个加法器,不存在加法器的资源冲突。有3条连续指令组成的程序代码如下:

```
I1  ADD  R1, R2, R4        ;R1←(R2)+(R4)
I2  ADD  R2, R1, 1         ;R2←(R1)+ 1
I3  SUB  R1, R4, R5        ;R1←(R4)-(R5)
```

(1) 分析程序代码段中的数据相关。

(2) 采用何种硬件技术可解决这些数据相关?请加以说明。

【题5.27】 假设有一条长流水线,仅对条件转移指令使用分支目标缓冲。假设分支预测错误的开销为4个时钟周期,缓冲不命中的开销为3个时钟周期。假设:命中率为90%,预测精度为90%,分支频率为15%,没有分支的基本CPI为1。

(1) 求程序执行的 CPI。

(2) 相对于采用固定的两个时钟周期延迟的分支处理,哪种方法程序执行速度更快?

【题 5.28】 假设分支目标缓冲的命中率为 90%,程序中无条件转移指令的比例为 5%,没有无条件转移指令的程序 CPI 值为 1。假设分支目标缓冲中包含分支目标指令,允许无条件转移指令进入分支目标缓冲,则程序的 CPI 值为多少? 假设原来的 CPI=1.1。

【题 5.29】 对于下述指令序列:

```
L.D       F6, 34(R2)
L.D       F2, 45(R3)
MUL.D     F0, F2, F4
SUB.D     F8, F2, F6
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

(1) 给出当第一条指令完成并写入结果时,Tomasulo 算法所用的各信息表中的内容。

(2) 假设各种操作的延迟为:

```
load: 1 个时钟周期
加法: 2 个时钟周期
乘法: 10 个时钟周期
除法: 40 个时钟周期
```

给出 MUL.D 指令准备写结果时各状态表的内容。

【题 5.30】 假设浮点功能部件的延迟时间为:加法两个时钟周期,乘法 10 个时钟周期,除法 40 个时钟周期。对于下面的代码段,在基于 Tomasulo 算法的支持前瞻执行的浮点部件的结构上,给出当指令 MUL.D 即将确认时的状态表内容。

```
L.D       F6, 34(R2)
L.D       F2, 45(R3)
MUL.D     F0, F2, F4
SUB.D     F8, F6, F2
DIV.D     F10, F0, F6
ADD.D     F6, F8, F2
```

【题 5.31】 下面的一段 MIPS 汇编程序是计算高斯消去法中的关键一步,用于完成下面公式的计算:

$$Y = a \times X + Y$$

指令的延迟如表 5.1 所示。

表 5.1　指令的延迟

产生结果的指令	使用结果的指令	延迟(时钟周期数)
浮点计算	另一个浮点计算	3
浮点计算	浮点 store(S.D)	2
浮点 load(L.D)	浮点计算	1
浮点 load(L.D)	浮点 store(S.D)	0

整数指令均为一个时钟周期完成,浮点和整数部件均采用流水。整数操作之间以及与其他所有浮点操作之间的延迟为0,转移指令的延迟为0。X 中的最后一个元素存放在存储器中的地址为DONE。

```
FOO:    L.D     F2, 0(R1)
        MUT.D   F4, F2, F0
        L.D     F6, 0(R2)
        ADD.D   F6, F4, F6
        S.D     F6, 0[R2]
        DADDIU  R1, R1, #8
        DADDIU  R2, R2, #8
        DSUBIU  R3, R1, #DONE
        BNEZ    R3, FOO
```

(1) 对于标准的MIPS单流水线,上述循环计算一个 Y 值需要多少时间?其中有多少空转周期?

(2) 对于标准的MIPS单流水线,将上述循环顺序展开4次,不进行任何指令调度,计算一个 Y 值平均需要多少时间?加速比是多少?其加速是如何获得的?

(3) 对于标准的MIPS单流水线,将上述循环顺序展开4次,优化和调度指令,使循环处理时间达到最优,计算一个 Y 值平均需要多少时间?加速比是多少?

(4) 对于采用前瞻执行机制的MIPS处理器(只有一个整数部件),当循环第二次执行到BNEZ R3,FOO时,写出前面所有指令的状态,包括指令使用的保留站、指令起始节拍、执行节拍和写结果节拍,并写出处理器当前的状态。

(5) 对于两路超标量的MIPS流水线,设有两个指令流出部件,可以流出任意组合的指令,系统中的功能部件数量不受限制。将上述循环展开4次,优化和调度指令,使循环处理时间达到最优。计算一个 Y 值平均需要多少时间?加速比是多少?

(6) 对于超长指令字MIPS处理器,将上述循环展开4次,优化和调度指令,使循环处理时间达到最优。计算一个 Y 值平均需要多少时间?加速比是多少?

【题5.32】 对于两路超标量处理器,从存储器取数据有两拍附加延迟,其他操作均有1拍附加延迟,对于下列代码,请按要求进行指令调度。

```
LW      R4, (R5)
LW      R7, (R8)
DADD    R9, R4, R7
LD      R10, (R11)
DMUL    R12, R13, R14
DSUB    R2, R3, R1
SW      R15, (R2)
DMUL    R21, R4, R7
SW      R23, (R22)
SW      R21, (R24)
```

(1) 假设两路功能部件中同时最多只有一路可以是访问存储器的操作,同时也最多只有一路可以是运算操作,指令顺序不变。

(2) 假设两路功能部件均可以执行任何操作,指令顺序不变。

(3) 假设指令窗口足够大,指令可以乱序(out-of-order)流出,两路功能部件均可以执行任何操作。

【题 5.33】 设指令流水线由取指令、分析指令和执行指令 3 个部件构成,每个部件经过的时间为 Δt,连续流入 12 条指令。分别画出标量流水处理机以及 ILP 均为 4 的超标量处理机、超长指令字处理机、超流水处理机的时空图,并分别计算它们相对于标量流水处理机的加速比。

【题 5.34】 用一台每个时钟周期发射两条指令的超标量处理机运行下面一段程序。所有指令都要进行取指(IF)、译码(ID)、执行、写结果(WB)4 个阶段。其中,IF、ID、WB 三个阶段各为一个流水段,其延迟时间都为 10ns。在执行阶段,LOAD 操作、AND 操作各延迟 10ns,ADD 操作延迟 20ns,MUL 操作延迟 30ns。这 4 种功能部件各设置一个,它们可以并行工作。ADD 部件和 MUL 部件都采用流水结构,每一级流水线的延迟时间都是 10ns。

```
I1   LOAD   R0, M(A)          ;R0←M(A)
I2   ADD    R1, R0            ;R1←(R1) + (R0)
I3   LOAD   R2, M(B)          ;R2←M(B)
I4   MUL    R3, R4            ;R3←(R3) × (R4)
I5   AND    R4, R5            ;R4←(R4) ∧ (R5)
I6   ADD    R2, R5            ;R2←(R2) + (R5)
```

(1) 请列出程序代码中所有的数据相关及其相关类型。

(2) 假设所有运算型指令都在译码(ID)流水段读寄存器,在写结果(WB)流水段写寄存器,采用顺序发射顺序完成的调度方法。画出流水线的时空图;计算执行这个程序所用的时间。

【题 5.35】 对于采用了 Tomasulo 算法和多流出技术的 MIPS 流水线,考虑以下简单循环的执行。该程序把 F2 中的标量加到一个向量的每个元素上。

```
Loop:   L.D      F0, 0(R1)        //取一个数组元素放入 F0
        ADD.D    F4, F0, F2       //加上在 F2 中的标量
        S. D     F4, 0(R1)        //存结果
        DADDIU   R1, R1, # -8     //指针减 8(每个数据占 8 个字节)
        BNE      R1, R2, Loop     //若 R1 不等于 R2,表示尚未结束,转移
                                  //到 Loop 继续
```

现做以下假设:

(1) 每个时钟周期能流出一条整数指令和一条浮点指令,即使它们相关也是如此。

(2) 整数 ALU 运算和地址计算共用一个整数部件;并且对于每一种浮点操作类型都有一个独立的流水化了的浮点功能部件。

(3) 指令流出和写结果各占用一个时钟周期。

(4) 具有动态分支预测部件和一个独立的计算分支条件的功能部件。

(5) 跟大多数动态调度处理器一样,写回段的存在意味着实际的指令延迟会比按序流动的简单流水线多一个时钟周期。所以,从产生结果数据的源指令到使用该结果数据的指令之间的延迟为:整数运算一个周期,load 两个周期,浮点加法运算 3 个周期。

(1) 请列出该程序前面三遍循环中各条指令的流出、开始执行和将结果写到 CDB 上的时间。

(2) 如果分支指令单流出,没有采用延迟分支,但分支预测是完美的。请列出整数部件、浮点部件、数据 Cache 以及 CDB 的资源使用情况。

【题 5.36】 在基于记分牌的 MIPS 处理器的基本结构上,运行下列代码,指令状态表的内容如表 5.2 所示。给出功能部件状态表和结果寄存器状态表中保存的信息。其中,有一个整数部件 Integer,两个乘法部件 Mult1 和 Mult2,一个加法部件 Add 和一个除法部件 Divide。

```
L. D       F6, 34(R2)
L. D       F2, 45(R3)
MULT. D    F0, F2, F4
SUB. D     F8, F6, F2
DIV. D     F10, F0, F6
ADD. D     F6, F8, F2
```

表 5.2 指令状态表

指令		指令状态表			
		流出	读操作数	执行	写结果
L. D	F6,34(R2)	√	√	√	√
L. D	F2, 45(R3)	√	√	√	
MULT. D	F0, F2, F4	√			
SUB. D	F8, F6, F2	√			
DIV. D	F10, F0, F6	√			
ADD. D	F6, F8, F2				

【题 5.37】 假设浮点流水线中各部件的延迟如下:加法需两个时钟周期、乘法需 10 个时钟周期、除法需 40 个时钟周期。运行下列代码段:

```
L. D       F6, 34(R2)
L. D       F2, 45(R3)
MULT. D    F0, F2, F4
SUB. D     F8, F6, F2
DIV. D     F10, F0, F6
ADD. D     F6, F8, F2
```

在记分牌算法中,给出 MULT. D 准备写结果之前的记分牌状态。其中,有一个整数部件 Integer,两个乘法部件 Mult1 和 Mult2,一个加法部件 Add 和一个除法部件 Divide。

题 解

1. 概念题

【题 5.1】 解释下列名词

指令级并行——简称 ILP。是指指令之间存在的一种并行性,利用它,计算机可以并行

执行两条或两条以上的指令。

IPC——Instructions Per Cycle 的缩写。每个时钟周期完成的指令条数。

循环级并行性——循环的不同迭代之间存在的并行性。

指令的动态调度——是指在保持数据流和异常行为的情况下，通过硬件对指令执行顺序进行重新安排，减少数据相关导致的停顿。

指令的静态调度——指依靠编译器对代码进行静态调度，以减少相关和冲突。它不是在程序执行的过程中，而是在编译期间进行代码调度和优化的。

不精确异常——指当执行指令 i 导致发生异常时，处理机的现场（状态）与严格按程序顺序执行时指令 i 的现场不同。不精确异常使得在异常处理后难以接着继续执行程序。

精确异常——指当执行指令 i 导致发生异常时，处理机的现场跟严格按程序顺序执行时指令 i 的现场相同。

CDB——公共数据总线。

BHT——分支历史表。用来记录相关分支指令最近一次或几次的执行情况是成功还是失败，并据此进行预测。

分支目标缓冲——是一种动态分支预测技术。将执行过的成功分支指令的地址以及预测的分支目标地址记录在一张硬件表中。在每次取指令的同时，用该指令的地址与表中所有项目的相应字段进行比较，以便尽早知道分支是否成功，尽早知道分支目标地址，达到减少分支开销的目的。

ROB——ReOrder Buffer。前瞻执行缓冲器。

超标量——一种多指令流出技术。它在每个时钟周期流出多条的指令，但指令的条数不固定，依代码的具体情况而定，但有个上限。

超流水——在一个时钟周期内分时流出多条指令。

超长指令字——一种多指令流出技术。VLIW 处理机在每个时钟周期流出的指令条数是固定的，这些指令构成一条长指令或者一个指令包，在这个指令包中，指令之间的并行性是通过指令显式地表示出来的。

2. 填空题

【题 5.2】 答：动态开发、静态开发

【题 5.3】 答：基本程序块

【题 5.4】 答：记分牌方法、Tomasulo 算法

【题 5.5】 答：写结果、指令确认

【题 5.6】 答：乱序、程序顺序

【题 5.7】 答：保留站的编号、ROB

【题 5.8】 答：编译器、数据相关

【题 5.9】 答：转移、历史、方向

【题 5.10】 答：超标量、超长指令字 VLIW

3. 问答题

【题 5.11】 答：开发 ILP 的途径有两种，一种是资源重复，重复设置多个处理部件，让

它们同时执行相邻或相近的多条指令;另一种是采用流水线技术,使指令重叠并行执行。

【题5.12】 答:最关键的两个属性是:数据流和异常行为。

保持异常行为是指:无论怎么改变指令的执行顺序,都不能改变程序中异常的发生情况。即原来程序中是怎么发生的,改变执行顺序后是怎么发生的。这个条件经常被弱化为:指令执行顺序的改变不能导致程序中发生新的异常。

数据流是指数据值从其产生者指令到其消费者指令的实际流动。

【题5.13】 答:优点:①能够处理一些编译时情况不明的相关,并能简化编译器;②能够使本来是面向某一流水线优化编译的代码在其他动态调度的流水线上也能高效地执行。

但动态调度的这些优点是以硬件复杂性的显著增加为代价的。

【题5.14】 答:

(1) 指令状态表:记录正在执行的各条指令已经进入到了哪一段。

(2) 功能部件状态表:记录各个功能部件的状态。每个功能部件有一项,每一项由9个字段组成。

(3) 结果寄存器状态表Result:每个寄存器在该表中有一项,用于指出哪个功能部件(编号)将把结果写入该寄存器。

【题5.15】 答:其核心思想是:①记录和检测指令相关,操作数一旦就绪就立即执行,把发生RAW冲突的可能性减小到最少;②通过寄存器换名来消除WAR冲突和WAW冲突。寄存器换名是通过保留站来实现,它保存等待流出和正在流出指令所需要的操作数。

基本思想:只要操作数有效,就将其取到保留站,避免指令流出时才到寄存器中取数据,这就使得即将执行的指令从相应的保留站中取得操作数,而不是从寄存器中。指令的执行结果也是直接送到等待数据的其他保留站中去。因而,对于连续的寄存器写,只有最后一个才真正更新寄存器中的内容。一条指令流出时,存放操作数的寄存器名被换成为对应于该寄存器保留站的名称(编号)。

【题5.16】 答:从指令队列的头部取一条指令。如果该指令的操作所要求的保留站有空闲的,就把该指令送到该保留站(设为r)。并且,如果其操作数在寄存器中已经就绪,就将这些操作数送入保留站r。如果其操作数还没有就绪,就把将产生该操作数的保留站的标识送入保留站r。另外,还要完成对目的寄存器的预约工作,将其设置为接受保留站r的结果。

【题5.17】 答:

(1) 冲突检测逻辑和指令执行控制是分布的(通过保留站和CDB实现)。

每个功能部件的保留站中的信息决定了什么时候指令可以在该功能部件开始执行。如果有多条指令已经获得了一个操作数,并同时在等待同一运算结果,那么这个结果一产生,就可以通过CDB同时播送给所有这些指令,使它们可以同时执行。

(2) 消除了WAW冲突和WAR冲突导致的停顿。

这是通过使用保留站进行寄存器换名,并且在操作数一旦就绪就将其放入保留站来实现的。

【题5.18】 答:目的有两个:预测分支是否成功和尽快找到分支目标地址(或指令),从而避免控制相关造成流水线停顿。

需要解决两个关键问题：①如何记录分支的历史信息；②如何根据这些信息来预测分支的去向，甚至提前取出分支目标处的指令。

【题 5.19】 答：流水线 3 个阶段所进行的相关操作如图 5.1 所示。

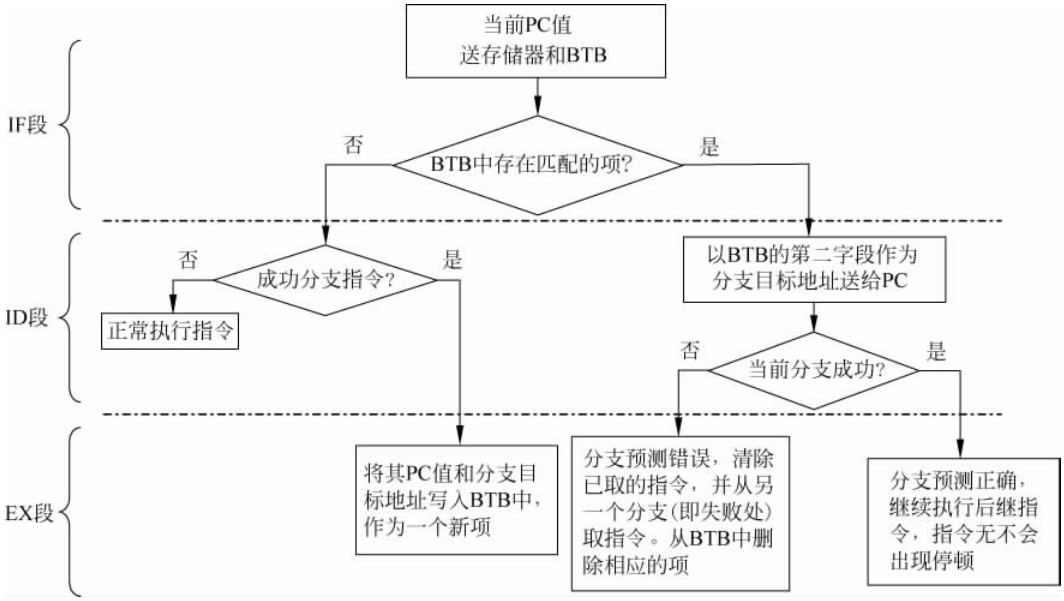

图 5.1　流水线 3 个阶段所进行的相关操作

【题 5.20】 答：

(1) 动态分支预测。用来选择后续执行的指令。

(2) 在控制相关的结果尚未出来之前，前瞻地执行后续指令。

(3) 用动态调度对基本块的各种组合进行跨基本块的调度。

【题 5.21】 答：

(1) 指令类型：指出该指令是分支指令、store 指令或寄存器操作指令。

(2) 目的地址：给出指令执行结果应写入的目的寄存器号(如果是 load 和 ALU 指令)或存储器单元的地址(如果是 store 指令)。

(3) 数据值字段：用来保存指令前瞻执行的结果，直到指令得到确认。

(4) 就绪字段：指出指令是否已经完成执行并且数据已就绪。

【题 5.22】 答：这一阶段对分支指令、store 指令以及其他指令的处理不同。

(1) 对于除分支指令和 store 指令以外的指令来说，当该指令到达 ROB 队列的头部而且其结果已经就绪时，就把该结果写入该指令的目的寄存器，并从 ROB 中删除该指令。

(2) 对 store 指令的处理与(1)类似，只是它是把结果写入存储器。

(3) 当预测错误的分支指令到达 ROB 队列的头部时，就表示是错误的前瞻执行。这时要清空 ROB，并从分支指令的另一个分支重新开始执行。

(4) 当预测正确的分支指令到达 ROB 队列的头部时，该指令执行完毕。

【题 5.23】 答：

(1) 超标量结构对程序员是透明的，处理机能自己检测下一条指令是否能流出，不需要

由编译器或专门的变换程序对程序中的指令进行重新排列。

(2) 即使是没有经过编译器针对超标量结构进行调度优化的代码或是旧的编译器生成的代码也可以运行,当然运行的效果不会很好。

【题 5.24】 答:

(1) 程序所固有的指令级并行性。

(2) 硬件实现上的困难。多流出的处理器需要大量的硬件资源,随着每个时钟周期流出指令数的增加,所需的硬件成正比例地增长,所需的存储器带宽和寄存器带宽也大大增加了,这样的带宽要求必然导致大量增加硅片面积,加大面积就导致时钟频率下降、功耗增加、可靠性降低等一系列问题。

(3) 超标量和超长指令字处理器固有的技术限制。

4. 应用题

【题 5.25】

解:

```
I1   LOAD  R1,M(A)
I2   LOAD  R2,M(B)
I3   LOAD  R3,M(C)
I4   LOAD  R4,M(D)
I5   MUL   R5,R1,R2
I6   ADD   R2,R3,R4
I7   ADD   R2,R2,R5
```

【题 5.26】

解: (1) 指令 I_1 和 I_2 之间有 RAW 相关,I_2 和 I_3 之间有 RAW 相关,I_1 和 I_3 之间有 WAW 相关,I_1 和 I_2 之间还有 WAR 相关。

(2) 对 I_1 和 I_2 之间的 WAR 相关,可用定向传送解决。根据寄存器重命名技术,对引起 RAW 相关的 I_2 中的 R_2,对引起 WAW 相关的 I_3 中的 R_1,可分别换成备用寄存器 R_2'、R_1'。经寄存器重命名后,程序代码段实际执行时变为:

```
I1      ADD  R1,R2,R4
I2      ADD  R2',R1,1
I3      SUB  R1',R4,R5
```

【题 5.27】

解: (1)程序执行的 CPI =没有分支的基本 CPI(1) +分支带来的额外开销

分支带来的额外开销是指在分支指令中,缓冲命中但预测错误带来的开销与缓冲没有命中带来的开销之和。

分支带来的额外开销=15%×(90%命中×10%预测错误×4+10%没命中×3)=0.099

所以,程序执行的 CPI=1+0.099=1.099。

(2) 采用固定的两个时钟周期延迟的分支处理 CPI=1+15%×2=1.3

由(1)(2)可知分支目标缓冲方法执行速度快。

【题 5.28】

解: 设每条无条件转移指令的延迟为 x,则有:

$$1+5\%\times x=1.1$$

$$x=2$$

当分支目标缓冲命中时，无条件转移指令的延迟为 0。

所以，程序的 CPI=1+2×5%×(1−90%)=1.01

【题 5.29】

解：(1) 表 5.3～表 5.5 给出了当采用 Tomasulo 算法时，在上述给定的时刻，保留站、load 缓冲器以及寄存器状态表中的内容。标志 Add1 表示是第一个加法功能部件，Mult1 表示是第一个乘法功能部件，其余以此类推。

表 5.3　指令执行状态

指　令		指令执行状态		
		流出	执行	写结果
L.D	F6，34(R2)	√	√	√
L.D	F2，45(R3)	√	√	
MUL.D	F0，F2，F4	√		
SUB.D	F8，F2，F6	√		
DIV.D	F10，F0，F6	√		
ADD.D	F6，F8，F2	√		

表 5.4　保留站的内容

名称	保留站内容						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	L.D					45+Regs[R3]
Add1	yes	SUB.D		Mem[34+Regs[R2]]	Load2		
Add2	yes	ADD.D			Add1	Load2	
Add3	no						
Mult1	yes	MUL.D		Regs[F4]	Load2		
Mult2	yes	DIV.D		Mem[34+Regs[R2]]	Mult1		

表 5.5　寄存器状态表中的内容

域	寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1	Load2		Add2	Add1	Mult2		

(2) MUL.D 指令准备写结果时各状态表的内容如表 5.6～表 5.8 所示。

这里，由于 ADD.D 指令与 DIV.D 指令的 WAR 冲突已经消除，ADD.D 可以先于 DIV.D 完成并将结果写入 F6，不会出现错误。

表 5.6 指令状态表的内容

指令	指令状态表		
	流出	执行	写结果
L.D F6, 34(R2)	√	√	√
L.D F2, 45(R3)	√	√	√
MUL.D F0, F2, F4	√	√	
SUB.D F8, F6, F2	√	√	√
DIV.D F10, F0, F6	√		
ADD.D F6, F8, F2	√	√	√

表 5.7 保留站的内容

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL.D	Mem[45+Regs[R3]]	Regs[F4]			
Mult2	yes	DIV.D		MEM[34+Regs[R2]]	Mult1		

表 5.8 寄存器状态表

域	寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1					Mult2		

【题 5.30】

解：状态表的内容如表 5.9～表 5.11 所示。这时指令 SUB.D 尽管已经执行完毕，但需要等到 MUL.D 得到确认后才能确认。

表 5.9 保留站的内容

名称	保留站							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45+Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34+Regs[R2]]	#3		#5	

表 5.10　ROB

项号	ROB				
	Busy	指令	状态	目的	Value
1	no	L.D　F6，34(R2)	确认	F6	Mem[34＋Regs[R2]]
2	no	L.D　F2，45(R3)	确认	F2	Mem[45＋Regs[R3]]
3	yes	MUL.D　F0，F2，F4	写结果	F0	＃2×Regs[F4]
4	yes	SUB.D　F8，F6，F2	写结果	F8	＃1－＃2
5	yes	DIV.D　F10，F0，F6	执行	F10	
6	yes	ADD.D　F6，F8，F2	写结果	F6	＃4＋＃2

表 5.11　浮点寄存器状态

字　　段	浮点寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
ROB 项编号	3			6	4	5		
Busy	yes	no	no	yes	yes	yes	…	no

【题 5.31】

解：(1)

```
L. D      F2, 0(R1)        1
Stall
MUT.D     F4, F2, F0       2
L. D      F6, 0(R2)        3
Stall
Stall
ADD.D     F6, F4, F6       4
Stall
Stall
S. D      F6, 0[R2]        5
DADDIU    R1, R1, ＃8       6
DADDIU    R2, R2, ＃8       7
DSUBIU    R3, R1, ＃DONE    8
BNEZ      R3, FOO          9
```

所以，共有 14 个时钟周期，其中有 5 个空转周期。

(2) 循环顺序展开 4 次，不进行任何指令调度，则指令 1～5 及其间的 stall 都是必要的，只是指令 6～9 只需执行一次，因此，共有 10×4＋4＝44 个时钟周期，计算出 4 个 Y 值，所以计算一个 Y 值需要 11 个时钟周期，加速比为：14/11＝1.27。加速主要是来自减少控制开销，即减少对 R1、R2 的整数操作以及比较、分支指令而来的。

(3) 循环顺序展开 4 次，优化和调度指令，如下。

```
L. D      F2, 0(R1)
L. D      F8, 8(R1)
L.D       F14, 16(R1)
L. D      F20, 24(R1)
MUT.D     F4, F2, F0
```

```
MUT.D     F10, F8, F0
MUT.D     F16, F14, F0
MUT.D     F22, F20, F0
L. D      F6, 0(R2)
L. D      F12, 8(R2)
L. D      F18, 16(R2)
L. D      F24, 24(R2)
ADD.D     F6, F4, F6
ADD.D     F12, F10, F12
ADD.D     F18, F16, F18
ADD.D     F24, F22, F24
S. D      F6, 0[R2]
S. D      F12, 8[R2]
S. D      F18, 16[R2]
S. D      F24, 24[R2]
DADDIU    R1, R1, #32
DADDIU    R2, R2, #32
DSUBIU    R3, R1, #DONE
BNEZ      R3, FOO
```

共用了 24 个时钟周期,则计算一个 Y 值平均需要 24/4=6 个时钟周期。

加速比:14/6=2.33

(4) 状态表的内容如表 5.12～表 5.15 所示。

表 5.12 指令执行时钟

指令	指令执行时钟			
	流出	执行	写结果	确认
L. D F2, 0(R1)	1	2	3	4
MUL.D F4, F2, F0	2	4	5	6
L. D F6, 0(R2)	3	4	6	7
ADD.D F6, F4, F6	4	8	9	10
S. D F6, 0(R2)	5	11	12	13
DADDIU R1, R1, #8	6	7	8	
DADDIU R2, R2, #8	7	8	9	
DSUBIU R3,R1,#DONE	8	9	10	
BNEZ R3, FOO	9	10		
L. D F2, 0(R1)	10	11	13	14
MUL.D F4, F2, F0	11	13	14	15
L. D F6, 0(R2)	12	13	15	16
ADD.D F6, F4, F6	13	17	18	19
S. D F6, 0(R2)	14	20	21	22
DADDIU R1, R1, #8	15	16	17	
DADDIU R2, R2, #8	16	17	18	
DSUBIU R3,R1,#DONE	17	18	19	
BNEZ R3, FOO	18			

表 5.13　保留站

名称	保留站							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	yes	ADD.D	Regs[F4]	Regs[F6]				
Add2	no							
Add3	no							
Mult1	yes							
Mult2	no							

表 5.14　ROB

项号	ROB				
	Busy	指令	状态	目的	Value
1	yes	ADD.D　F6，F4，F6	执行	F6	Regs[F4]＋Regs[F6]
2	yes	S.D　F6，0(R2)	流出	Mem[0＋Regs[R2]]	＃2

表 5.15　浮点寄存器状态

字　段	浮点寄存器状态							
	F0	F2	F4	F6	F8	F10	…	F30
ROB 项编号				1				
Busy				yes			…	

(5) 时钟周期数如表 5.16 所示。

表 5.16　时钟周期数

整 数 指 令	浮 点 指 令	时钟周期数
L.D　F2，0(R1)		1
L.D　F8，8(R1)		2
L.D　F14，16(R1)	MUT.D　F4，F2，F0	3
L.D　F20，24(R1)	MUT.D　F10，F8，F0	4
L.D　F6，0(R2)	MUT.D　F16，F14，F0	5
L.D　F12，8(R2)	MUT.D　F22，F20，F0	6
L.D　F18，16(R2)	ADD.D　F6，F4，F6	7
L.D　F24，24(R2)	ADD.D　F12，F10，F12	8
DADDIU R1，R1，＃32	ADD.D　F18，F16，F18	9
S.D　F6，0(R2)	ADD.D　F24，F22，F24	10
S.D　F12，8(R2)		11
S.D　F18,16(R2)		12
S.D　F24，24(R2)		13
DADDIU R2，R2，＃32		14
DSUBIU R3，R1，＃DONE		15
BNEZ　R3，FOO		16

计算一个Y值需要16/4=4个时钟周期,加速比=14/4=3.5。

(6) 循环展开和指令调度如表5.17所示。

表5.17 循环展开和指令调度

访存1	访存2	浮点指令1	浮点指令2	整数指令	时钟周期
L. DF2, 0(R1)	L. DF8, 8(R1)				1
L. DF14, 16(R1)	L. DF20, 24(R1)				2
L. DF6, 0(R2)	L. DF12, 8(R2)	MUT. DF4, F2, F0	MUT. DF10, F8, F0		3
L. DF18, 16(R2)	L. DF24, 24(R2)	MUT. DF16, F14, F0	MUT. DF22, F20, F0		4
		ADD. DF6, F4, F6	ADD. DF12, F10, F12		5
		ADD. DF18, F16, F18	ADD. DF24, F22, F24	DADDIU R1, R1, #32	6
				DADDIU R2, R2, #32	7
				DSUBIUR3, R1, #DONE	8
				BNEZR3, FOO	9
S. DF6, -32(R2)	S. DF12, -24(R2)				10
S. DF18,-16(R2)	S. DF24, -8(R2)				11

计算一个Y值需要11/4个时钟周期,加速比=14/(11/4)=56/11。

【题5.32】

解:(1) 指令调度情况如表5.18所示。

表5.18 指令调度(1)

第 一 路	第 二 路
LW R4, (R5)	
LW R7, (R8)	
DADD R9, R4, R7	LD R10, (R11)
DMUL R12, R13, R14	
DSUB R2, R3, R1	SW R15, (R2)
DMUL R21, R4, R7	SW R23, (R22)
SW R21, (R24)	

(2) 指令调度情况如表 5.19 所示。

表 5.19　指令调度(2)

第　一　路	第　二　路
LW　R4,(R5)	LW　R7,(R8)
DADD　R9,R4,R7	LD　R10,(R11)
DMUL　R12,R13,R14	DSUB R2,R3,R1
SW　R15,(R2)	DMUL　R21,R4,R7
SW　R23,(R22)	
SW　R21,(R24)	

(3) 指令调度情况如表 5.20 所示。

表 5.20　指令调度(3)

第　一　路	第　二　路
LW　R4,(R5)	LW　R7,(R8)
DSUB　R2,R3,R1	LD　R10,(R11)
SW　R23,(R22)	DMUL　R12,R13,R14
DADD　R9,R4,R7	DMUL　R21,R4,R7
SW　R15,(R2)	
SW　R21,(R24)	

【题 5.33】

解：标量流水处理机的时空图如图 5.2 所示。

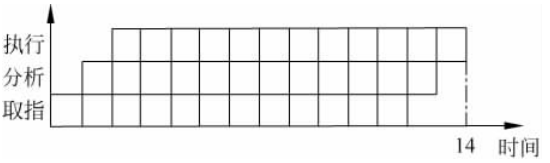

图 5.2　标量流水处理机

执行完 12 条指令需 $T_1=14\Delta t$。

超标量流水处理机与超长指令字处理机的时空图如图 5.3 所示。

超标量流水处理机中，每一个时钟周期同时启动 4 条指令。执行完 12 条指令需 $T_2=5\Delta t$，相对于标量流水处理机的加速比为：

$$S_2=\frac{T_1}{T_2}=\frac{14\Delta t}{5\Delta t}=2.8$$

超长指令字处理机中，每 4 条指令组成一条长指令，共形成 3 条长指令。执行完 12 条指令需 $T_3=5\Delta t$，相对于标量流水处理机的加速比为：

$$S_3=\frac{T_1}{T_3}=\frac{14\Delta t}{5\Delta t}=2.8$$

超流水处理机的时空图如图 5.4 所示。

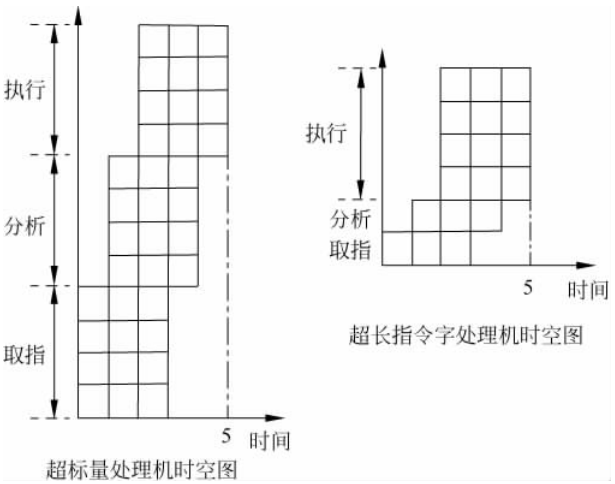

图 5.3　超标量流水处理机与超长指令字处理机的时空图

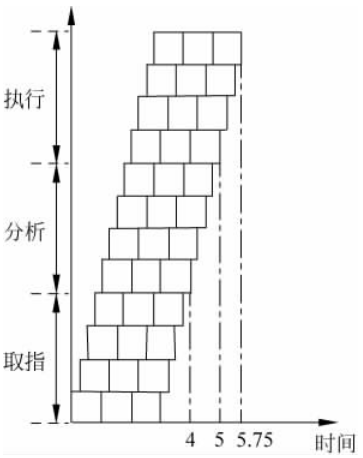

图 5.4　超流水处理机的时空图

超流水处理机中,每 1/4 个时钟周期启动一条指令。执行完 12 条指令需 $T_4=5.75\Delta t$,相对于标量流水处理机的加速比为:

$$S_4=\frac{T_1}{T_4}=\frac{14\Delta t}{5.75\Delta t}=2.435$$

【题 5.34】

解:(1) 指令 I_1、I_2间有寄存器 R_0的 WAR 相关;

指令 I_3、I_6间有寄存器 R_2的 WAR 相关;

指令 I_4、I_5间有寄存器 R_4的 RAW 相关;

指令 I_3、I_6间有寄存器 R_2的 WAW 相关。

(2) 采用顺序发射顺序完成调度方法的流水线时空图如图 5.5 所示。

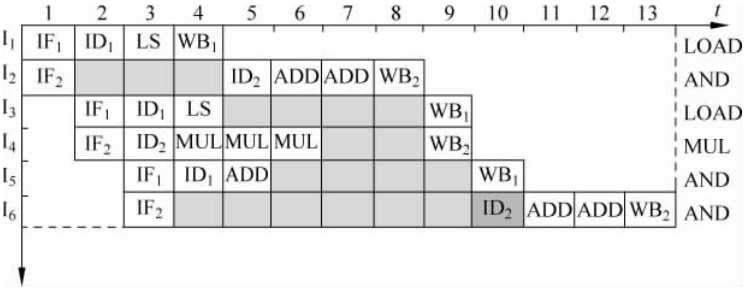

图 5.5　流水线时空图

从时空图看到,执行这个程序共用 130ns。

【题 5.35】

解:(1) 执行时,该循环将动态展开,并且只要可能就流出两条指令。从表 5.21 中可以看出,每 3 个时钟周期就执行一个新循环,每个循环 5 条指令,因此其 IPC 为:5/3=1.67。虽然指令的流出率比较高,但是执行效率并不是很高,16 拍共执行 15 条指令,平均指令执行速度为 15/16=0.94 条/拍。

表 5.21　基于 Tomasulo 算法的双流出超标量流水线中指令的流出、执行和写 CDB 的时间

遍数	指　令		流出	执行	访存	写 CDB	说　明
1	L. D	F0, 0(R1)	1	2	3	4	流出第一条指令
1	ADD. D	F4, F0, F2	1	5		8	等待 L. D 的结果
1	S. D	F4,0(R1)	2	3	9		等待 ADD. D 的结果
1	DADDIU	R1, R1, #-8	2	4		5	等待 ALU(计算指令 S. D 的有效地址也是用该 ALU)
1	BNE	R1,R2, Loop	3	6			等待 DADDIU 的结果
2	L. D	F0, 0(R1)	4	7	8	9	等待 BNE 完成
2	ADD. D	F4, F0, F2	4	10		13	等待 L. D 的结果
2	S. D	F4,0(R1)	5	8	14		等待 ADD. D 的结果
2	DADDIU	R1, R1, #-8	5	9		10	等待 ALU
2	BNE	R1,R2, Loop	6	11			等待 DADDIU 的结果
3	L. D	F0, 0(R1)	7	12	13	14	等待 BNE 完成
3	ADD. D	F4, F0, F2	7	15		18	等待 L. D 的结果
3	S. D	F4,0(R1)	8	13	19		等待 ADD. D 的结果
3	DADDIU	R1, R1, #-8	8	14		15	等待 ALU
3	BNE	R1,R2, Loop	9	16			等待 DADDIU 的结果

(2) 资源使用情况如表 5.22 所示。

表 5.22　资源使用情况

时钟周期	整型 ALU	浮点 ALU	数据 Cache	CDB
2	1/L. D			
3	1/S. D		1/L. D	
4	1/DADDIU			1/L. D
5		1/ADD. D		1/DADDIU
6				
7	2/L. D			
8	2/S. D		2/L. D	1/ADD. D
9	2/DADDIU		1/S. D	2/L. D
10		2/ADD. D		2/DADDIU
11				
12	3/L. D			
13	3/S. D		3/L. D	2/ADD. D
14	3/DADDIU		2/S. D	3/L. D
15		3/ADD. D		3/DADDIU
16				
17				
18				3/ADD. D
19			3/S. D	
20				

【题 5.36】

解：功能部件状态表和结果寄存器状态表如表 5.23 和表 5.24 所示。

表 5.23 功能部件状态表

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	L. D	F2	R3				no	
Mult1	yes	MULT. D	F0	F2	F4	Integer		no	yes
Mult2	no								
Add	yes	SUB. D	F8	F6	F2		Integer	yes	no
Divide	yes	DIV. D	F10	F0	F6	Mult1		no	yes

表 5.24 结果寄存器状态表

	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
部件名称	Mult1	Integer			Add	Divide		

【题 5.37】

解：指令状态表、功能部件状态表和结果寄存器状态表如表 5.25～表 5.27 所示。

表 5.25 指令状态表

指　令		指令状态表			
		流出	读操作数	执行	写结果
L. D	F6, 34(R2)	√	√	√	√
L. D	F2, 45(R3)	√	√	√	√
MULT. D	F0, F2, F4	√	√	√	
SUB. D	F8, F6, F2	√	√	√	√
DIV. D	F10, F0, F6	√			
ADD. D	F6, F8, F2	√	√	√	

表 5.26 功能部件状态表

部件名称	功能部件状态表								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	yes	MULT. D	F0	F2	F4			no	no
Mult2	no								
Add	yes	ADD. D	F6	F8	F2			no	no
Divide	yes	DIV. D	F10	F0	F6	Mult1		no	yes

表 5.27 结果寄存器状态表

	结果寄存器状态表							
	F0	F2	F4	F6	F8	F10	…	F30
部件名称	Mult1			Add		Divide		