第 3 章 流水线技术

3.1 基本要求与难点

3.1.1 基本要求

(1) 掌握有关流水线的基本概念。

(2) 掌握流水线的工作原理，了解如何从不同的角度对流水线进行分类。

(3) 理解流水线的各性能指标，能熟练地用时空图或公式计算吞吐率、加速比和效率。能熟练地画出时空图。

(4) 掌握消除流水线瓶颈的方法。

(5) 熟练掌握单功能非线性流水线的最优调度方法。能熟练地画出状态转换图，并根据状态转换图写出最优调度方案。了解双功能非线性流水线的最优调度方法。

(6) 掌握经典 5 段流水线的结构。

(7) 理解 3 种相关和 3 种冲突的概念，掌握解决结构冲突、数据冲突和控制冲突的方法，特别是：如何利用定向技术来解决数据冲突？如何用猜测法和延迟分支来解决控制冲突？

(8) 掌握基本的 MIPS 流水线的组成以及在各流水段完成的操作。掌握对该流水线进行改进后(把分支延迟减少为 1 个时钟周期)IF 段和 ID 段的操作的变化。

3.1.2 难点

(1) 流水线时空图的画法，如何计算流水线的吞吐率、加速比和效率？

(2) 单功能非线性流水线的最优调度方法。

(3) 如何解决结构冲突、数据冲突和控制冲突？

(4) 基本的 MIPS 流水线的各段完成的操作。

3.2 知 识 要 点

3.2.1 流水线的基本概念

1. 什么是流水线

在计算机中，把一个重复的过程分解为若干个子过程，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**。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**，流水线的段数称为**流水线的深度**。

一般采用时空图来描述流水线的工作过程。时空图的横坐标表示时间，纵坐标表示空间，即流水线的段。

流水线技术有以下特点：

(1) 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**。

(3) 流水线每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**。

(4) 流水技术适合于大量重复的时序过程。

(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。在这两个时间段中，流水线都不是满负荷工作。

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、处理机级及系统级流水线

这是按照流水技术用于计算机系统的等级不同来分的。

- **部件级流水线**是把处理机中的部件进行分段，再把这些部件分段相互连接而成。它使得运算操作能够按流水方式进行。这种流水线也称为**运算操作流水线**。
- 处理机级流水线又称**指令流水线**。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。
- **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。前一个处理机的输出结果存入存储器中，作为后一个处理机的输入。这种流水线又称为**宏流水线**。

2) 单功能流水线与多功能流水线

- **单功能流水线**是指流水线的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。
- **多功能流水线**是指各段可以进行不同的连接，以实现不同的功能的流水线。

3) 静态流水线与动态流水线

- **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。
- **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。

一般来说，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。

4) 线性流水线与非线性流水线

- **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。
- **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。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流水线中的任务发生争用流水段的冲突。这就是**非线性流水线的调度问题**。

5) 顺序流水线与乱序流水线

这是根据流水线中任务流入和流出的顺序是否相同来区分的。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。而在**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。

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，则称之为**向量流水处理机**，简称向量机；否则，就称之为**标量流水处理机**。

3.2.2 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。

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。

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

其中，n 为任务数，T_k 是处理完成 n 个任务所用的时间。

1) 各段时间均相等的流水线

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

其中，Δt 为各段的时间，k 为段数。

2) 各段时间不完全相等的流水线

各段时间不等的流水线的实际吞吐率为：

$$\mathrm{TP}=\frac{n}{\sum_{i=1}^{k}\Delta t_i+(n-1)\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{3.3}$$

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：

$$\mathrm{TP}_{\max}=\frac{1}{\max(\Delta t_1,\Delta t_2,\cdots,\Delta t_k)} \tag{3.4}$$

从式(3.3)和式(3.4)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。

可以用以下两种方法来消除瓶颈段。

(1) **细分瓶颈段**。这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。

(2) **重复设置瓶颈段**。如果无法把瓶颈段再细分,则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。

2. 流水线的加速比

流水线的加速比是指使用顺序处理方式处理一批任务所用的时间(设为 T_s)与按流水处理方式处理同一批任务所用的时间(设为 T_k)之比:

$$S = \frac{T_s}{T_k} \tag{3.5}$$

假设流水线各段时间相等,都是 Δt,则该流水线的实际加速比为:

$$S = \frac{nk}{k+n-1} \tag{3.6}$$

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:

$$S = \frac{n\sum_{i=1}^{k}\Delta t_i}{\sum_{i=1}^{k}\Delta t_i + (n-1)\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{3.7}$$

3. 流水线的效率

流水线的效率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。如果各段时间相等,则各段的效率是相同的,且都等于整条流水线的效率。

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

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:

$$E = \frac{n \cdot \sum_{i=1}^{k}\Delta t_i}{k\left[\sum_{i=1}^{k}\Delta t_i + (n-1) \cdot \max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)\right]} \tag{3.8}$$

计算流水线效率的一般公式可以表示为:

$$E = \frac{n\text{个任务实际占用的时空区的面积}}{k\text{个段总的时空区的面积}} \tag{3.9}$$

画出流水线的时空图,然后根据式(3.9)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。

4. 流水线设计中的若干问题

1) 瓶颈问题

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。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。

3) 冲突问题

如果流水线中的指令或数据之间存在关联,则它们可能要相互等待,引起访问冲突,造

成流水线的停顿。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。

3.2.3 非线性流水线的调度

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**。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。

对流水线的任务进行优化调度和控制的步骤是：

(1) 根据**预约表**写出**禁止表** F。

(2) 根据禁止表 F 写出**初始冲突向量** $C_0=(c_Nc_{N-1}\cdots c_i\cdots c_2c_1)$。

(3) 根据初始冲突向量 C_0 画出状态转换图。

令 $C_k=C_0$，按下式计算新的**冲突向量**：

$$\mathrm{SHR}^{(j)}(C_k)\ \vee\ C_0 \tag{3.10}$$

其中，$\mathrm{SHR}^{(j)}$ 表示逻辑右移 j 位。对于所有允许的时间间隔都按上述步骤求出其新的冲突向量，并且把新的冲突向量作为当前冲突向量，反复使用上述步骤，直到不再产生不同的冲突向量为止。由此可以画出用冲突向量表示的**流水线状态转移图**。

(4) 根据状态转换图写出最优调度方案。

由初始状态出发，任何一个闭合回路即为一种调度方案。为了找到最佳的调度方案，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。

2. 多功能非线性流水线的调度

双功能(设为功能 A 和 B)非线性流水线的最优调度方法类似于单功能非线性流水线的调度方法。只是其状态转移图中结点状态的表示不同，是由两个冲突向量构成的**冲突矩阵**。其初始结点有两个，分别对应于第一个任务是 A 类和 B 类的情况。

3.2.4 流水线的相关与冲突

1. 一个经典的 5 段流水线

先考虑在非流水情况下是如何实现的，可以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 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)

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

(2) ALU 指令：ALU 对从通用寄存器组中读出的数据进行运算。

(3) 分支指令：ALU 把偏移量与 PC 值相加，形成转移目标的地址。同时，判断分支是否成功。

4）存储器访问/分支完成周期(MEM)

(1) load 和 store 指令：根据有效地址从存储器中读出相应的数据(load 指令)，或者是把指定的数据写入有效地址所指出的存储器单元(store 指令)。

(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。

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，这个结果是来自存储器。

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，就构成了一个经典的 5 段流水线(见图 3.1)。这些锁存器称为流水线寄存器。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。

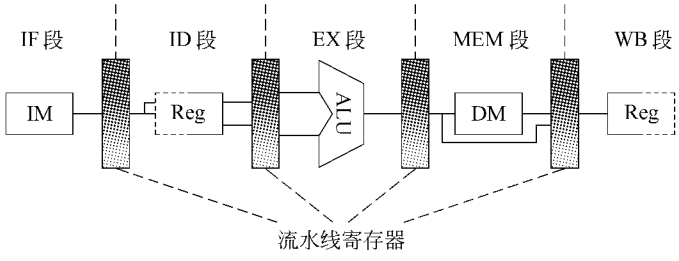

图 3.1　一个经典的 5 段流水线

首先，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。

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，可以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。

第四，没有考虑 PC 的问题。在每个时钟周期中，都要在 IF 段把 PC 值加 4，为此需要设置一个专门的加法器。另外，分支指令在 MEM 段也要修改 PC 的值。

2. 相关与流水线冲突

1）相关

相关是指两条指令之间存在某种依赖关系。如果两条指令相关，那它们可能就不能在流水线中重叠执行或者只能部分重叠。

相关有数据相关(也称真数据相关)、名相关、控制相关 3 种类型。

(1) 数据相关：

考虑两条指令 i 和 j，i 在 j 的前面(下同)，如果下述条件之一成立，则称指令 j 与指令 i **数据相关**：

① 指令 j 使用指令 i 产生的结果；

② 指令 j 与指令 k 数据相关，而指令 k 又与指令 i 数据相关。

其中，第二个条件表明，数据相关具有传递性。两条指令之间如果存在第一个条件所指出的相关的链，则它们是数据相关的。数据相关反映了数据的流动关系，即如何从其产生者流动到其消费者。

(2) 名相关：

这里的名是指指令所访问的寄存器或存储器单元的名称。如果两条指令使用了相同的名，但是它们之间并没有数据流动，则称这两条指令存在**名相关**。指令 j 与指令 i 之间的名相关有以下两种：

① **反相关**。如果指令 j 所写的名与指令 i 所读的名相同，则称指令 i 和 j 发生了反相关。反相关指令之间的执行顺序是必须严格遵守的，以保证 i 读的值是正确的。

② **输出相关**。如果指令 j 和指令 i 所写的名相同，则称指令 i 和 j 发生了输出相关。输出相关指令的执行顺序是不能颠倒的，以保证最后的结果是指令 j 写进去的。

与真数据相关不同，名相关的两条指令之间并没有数据的传送，只是使用了相同的名而已。可以通过改变指令中操作数的名来消除名相关，这就是**换名**技术。对于寄存器操作数进行换名称为**寄存器换名**。寄存器换名既可以用编译器静态实现，也可以用硬件动态完成。

(3) 控制相关：

控制相关是指由分支指令引起的相关。它需要根据分支指令的执行结果来确定后面该执行哪个分支上的指令。一般来说，为了保证程序应有的执行顺序，必须严格按照控制相关确定的顺序执行。

2) 流水线冲突

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开始执行。

流水线冲突有结构冲突、数据冲突、控制冲突 3 种类型。

在后面的讨论中，我们约定：当一条指令被暂停时，在该暂停指令之后流出的所有指令都要被暂停，而在该暂停指令之前流出的指令则继续进行。显然，在整个暂停期间，流水线不会启动新的指令。

(1) 结构冲突：

在流水线处理机中，如果某种指令组合因为资源冲突而不能正常执行，则称该处理机有**结构冲突**。这种情况发生在功能部件不是完全流水或者资源份数不够时。

(2) 数据冲突：

① 数据冲突。

当相关的指令彼此靠得足够近时，它们在流水线中的重叠执行或者重新排序会改变指令读写操作数的顺序，使之不同于它们串行执行时的顺序。这就是发生了**数据冲突**。

按照指令读访问和写访问的先后顺序，可以将数据冲突分为 3 种类型。习惯上，这些冲突是按照流水线必须保持的访问顺序来命名的。考虑两条指令 i 和 j，且 i 在 j 之前进入流水线，可能发生的数据冲突有：

A. **写后读冲突**(Read After Write，RAW)。指令 j 用到指令 i 的计算结果，而且在 i 将结果写入寄存器之前就去读该寄存器，因而得到的是旧值。这是最常见的一种数据冲突，它对应于真数据相关。

B. **写后写冲突**(Write After Write，WAW)。指令 j 和指令 i 的结果寄存器相同，而且

j 在 i 写入之前就先对该寄存器进行了写入操作，从而导致写入顺序错误。最后在结果寄存器中留下的是 i 写入的值。这种冲突对应于输出相关。

写后写冲突仅发生在这样的流水线中：流水线中不只一个段可以进行写操作；或者指令被重新排序(第 5 章介绍)。前面介绍的 5 段流水线不会发生写后写冲突。

C. **读后写冲突**(Write After Read，WAR)。指令 j 的目的寄存器和指令 i 的源操作数寄存器相同，而且 j 在 i 读取该寄存器之前就先对它进行了写操作，导致 i 读到的值是错误的。这种冲突是由反相关引起的。

读后写冲突在前述 5 段流水线中不会发生。读后写冲突仅发生在这样的情况下：(a)有些指令的写结果操作提前了，而且有些指令的读操作滞后了；或者(b)指令被重新排序了。

② 使用定向技术减少数据冲突引起的停顿。

为了减少停顿时间，可以采用定向技术来解决写后读冲突。**定向技术**(也称为旁路)的关键思想是：在发生写后读相关的情况下，在计算结果尚未出来之前，后面等待使用该结果的指令并不见得是马上就要用该结果。如果能够将该计算结果从其产生的地方(ALU 的出口)直接送到其他指令需要它的地方(ALU 的入口)，那么就可以避免停顿。

③ 需要停顿的数据冲突。

并不是所有的数据冲突都可以用定向技术来解决。对于这种情况，为保证代码能在流水线中正确执行，需要设置一个称为“流水线互锁机制”的功能部件。一般来说，**流水线互锁机制**的作用是检测和发现数据冲突，并使流水线停顿，直至冲突消失。停顿是从等待相关数据的指令开始，到相应的指令产生该数据为止。停顿导致在流水线中插入气泡，使得被停顿指令的 CPI 增加了相应的时钟周期数。

④ 依靠编译器解决数据冲突。

为了减少停顿，对于无法用定向技术解决的数据冲突，可以通过在编译时让编译器重新组织指令顺序来消除冲突，这种技术称为**“指令调度”**或**“流水线调度”**。实际上，对于各种冲突，都有可能用指令调度来解决。

(3) 控制冲突：

在流水线中，控制冲突可能会比数据冲突造成更多的性能损失，所以同样需要得到很好的处理。执行分支指令的结果有两种，一种是分支“成功”，PC 值改变为分支转移的目标地址。另一种则是“失败”，这时 PC 的值保持正常递增，指向顺序的下一条指令。对分支指令“成功”的情况来说，是在条件判定和转移地址计算都完成后，才改变 PC 值。对于 3.4.1 节中的 5 段流水线来说，改变 PC 值是在 MEM 段进行的。

处理分支指令最简单的方法是“冻结”或者“排空”流水线。即一旦在流水线的译码段 ID 检测到分支指令，就暂停其后的所有指令的执行，直到分支指令到达 MEM 段、确定出是否成功并计算出新的 PC 值为止。然后，按照新的 PC 值取指令。在这种情况下，分支指令给流水线带来了 3 个时钟周期的延迟。显然，这种让流水线空等的方法不是一种好的选择。

由分支指令引起的延迟称为**分支延迟**。为减少分支延迟，可采取以下措施：

- 在流水线中尽早判断出(或者猜测)分支转移是否成功；
- 尽早计算出分支目标地址。

这两种措施要同时采用，缺一不可。因为只有判断出转移是否成功而且得到分支目标

地址后才能进行转移。

在下面的讨论中，假设这两步工作被提前到ID段完成，即分支指令是在ID段的末尾执行完，所带来的分支延迟为一个时钟周期。

减少分支延迟的方法有许多种。下面只介绍3种通过软件(编译器)来处理的方法，这3种方法有一个共同的特点：它们对分支的处理方法在程序的执行过程中始终是不变的。它们要么总是预测分支成功，要么总是预测分支失败。

① 预测分支失败。

当ID段检测到分支指令时，我们可以让流水线通过预测选择两条分支路径中的一条，继续处理后续指令。预测有两种选择：猜测分支成功，或者猜测分支失败。不管哪一种，都可以通过编译器来优化性能，即让代码中最常执行的路径与所选的预测方向一致。

预测分支失败的方法是沿失败的分支继续处理指令，就好像什么都没发生似的。当确定分支是失败时，说明预测正确，流水线正常流动；当确定分支是成功时，流水线就把在分支指令之后取出的指令转化为空操作，并按分支目标地址重新取指令执行。

采用这种方法处理分支指令的后续指令时，要保证分支结果出来之前不能改变处理机的状态，以便一旦猜错时，处理机能够回退到原先的状态。

② 预测分支成功。

这种方法按分支成功的假设进行处理。当流水线ID段检测到分支指令后，一旦计算出了分支目标地址，就开始从该目标地址取指令执行。

③ 延迟分支。

这种方法的主要思想是从逻辑上“延长”分支指令的执行时间。把**延迟分支**看成是由原来的分支指令和若干个延迟槽构成。不管分支是否成功，都要按顺序执行延迟槽中的指令。在采用延迟分支的实际机器中，绝大多数的延迟槽都是一个，即：

分支指令
延迟槽

后面只讨论这种情况。

可以看出，只要分支延迟槽中的指令是有用的，流水线中就没有出现停顿，这时延迟分支的方法能很好地减少分支延迟。

放入延迟槽中的指令是由编译器来选择的。实际上延迟分支能否带来好处完全取决于编译器能否把有用的指令调度到延迟槽中。这也是一种指令调度技术。常用的调度方法有3种：从前调度、从目标处调度、从失败处调度。

上述方法受到两个方面的限制，一个是可以被放入延迟槽中的指令要满足一定的条件，另一个是编译器预测分支转移方向的能力。为了提高编译器在延迟槽中放入有用指令的能力，许多处理机采用了**分支取消机制**。在这种机制中，分支指令隐含了预测的分支执行方向。当分支的实际执行方向和事先所预测的一样时，执行延迟槽中的指令，否则就将该指令转化成空操作。

3.2.5 流水线的实现

1. MIPS的一种简单实现

考虑实现MIPS指令子集的一种简单数据通路。该数据通路的操作分成5个时钟周

期：取指令、指令译码/读寄存器、执行/有效地址计算、存储器访问/分支完成、写回。下面只讨论整数指令的实现，包括数据字的 load 和 store、等于 0 转移、整数 ALU 指令等。

在这个数据通路上，最多花 5 个时钟周期就能实现一条 MIPS 指令。这 5 个时钟周期及相应的操作如下所示。

1）取指令周期(IF)

IR←Mem[PC]
NPC←PC+4

2）指令译码/读寄存器周期(ID)

A←Regs[rs]
B←Regs[rt]
Imm←$((IR_{16})^{16}$ ## $IR_{16..31})$

这里准备放在 A、B 和 Imm 中的数据在后面周期中也许用不上，但也没关系，并不影响程序执行的正确性。而且统一这样处理，可以减少硬件的复杂度，是有益无害的。

3）执行/有效地址计算周期(EX)

在这个周期，ALU 对在前一个周期准备好的操作数进行运算。不同指令所进行的操作不同：

(1) load 指令和 store 指令：

ALUo←A+Imm

(2) 寄存器-寄存器 ALU 指令：

ALUo←A funct B

funct 字段给出了运算操作码。

(3) 寄存器-立即值 ALU 指令：

ALUo←A op Imm

(4) 分支指令：

ALUo←NPC+(Imm<<2);
cond←(A==0)

这里只考虑一种分支，即 BEQZ(Branch if EQual Zero)，它的判断操作是：判断是否为 0。判断的结果存入寄存器 cond，供以后使用。

这里将有效地址计算周期和执行周期合并为一个时钟周期，这是因为 MIPS 指令集采用 load/store 结构，任何指令都不会同时进行数据有效地址的计算、转移目标地址的计算和对数据进行运算。

4）存储器访问/分支完成周期(MEM)

所有指令都要在该周期对 PC 进行更新。除了分支指令，其他指令都是做：

PC←NPC

在该周期处理的指令只有 load、store 和分支 3 种指令。