[数据通路 2](#_Toc505800398)

[Load 3](#_Toc505800399)

[Store 4](#_Toc505800400)

[ALU 7](#_Toc505800401)

[Branch 8](#_Toc505800402)

[JMP 11](#_Toc505800403)

## 数据通路



Tomasulo算法中保留站的换名功能是由ROB来完成的

基于硬件的前瞻执行结合了三种思想：

动态分支预测。用来选择后续执行的指令。

在控制相关的结果尚未出来之前，前瞻地执行后续指令。

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。

把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：

实现前瞻的关键思想：允许指令乱序执行，但必须顺序确认。

前瞻执行

通过ROB实现了指令的顺序完成。

能够实现精确异常。

很容易地推广到整数寄存器和整数功能单元上。

主要缺点：所需的硬件太复杂。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留站  RS |  | tag | | busy | | op | | | vs | | vt | | qs | | qt | | dest | | A |
| load  缓冲器  (LB) | RSlb1 | | 0 | |  | | |  | |  | |  | |  | |  | |  |
| RSlb2 | | 0 | |  | | |  | |  | |  | |  | |  | |  |
| …… | | …… | |  | | |  | |  | |  | |  | |  | |  |
| store  缓冲器  (SB) | RSsb1 | | 1 | |  | | |  | |  | |  | |  | |  | |  |
| RSsb2 | | 1 | |  | | |  | |  | |  | |  | |  | |  |
| …… | | …… | |  | | |  | |  | |  | |  | |  | |  |
| 运算器  保留站 | RSadd1 | | 1 | |  | | |  | |  | |  | |  | |  | |  |
| RSadd2 | | 0 | |  | | |  | |  | |  | |  | |  | |  |
| …… | | …… | |  | | |  | |  | |  | |  | |  | |  |
| RSmult1 | | 1 | |  | | |  | |  | |  | |  | |  | |  |
| RSmult2 | | 0 | |  | | |  | |  | |  | |  | |  | |  |
| …… | | …… | |  | | |  | |  | |  | |  | |  | |  |
| tag | 每个保留站有一个唯一的标识 | | | | | | | | | | | | | | | | | |
| busy | busy=1：占用； busy=0：空闲 | | | | | | | | | | | | | | | | | |
| vs | 保存指令字中rs对应的数值 | | | | | | | | | | | | | | | | | |
| vt | 保存指令字中rt对应的数值 | | | | | | | | | | | | | | | | | |
| A | 保存符号位扩展后的立即数Imm | | | | | | | | | | | | | | | | | |
| qs | qs=0：vs存储的数据就绪；  qs=RSadd1：vs存储的数据未就绪，由RSadd1提供 | | | | | | | | | | | | | | | | | |
| qt | qt=0：vt存储的数据就绪；  qt=RSmult1：vt存储的数据未就绪，由RSmult1提供 | | | | | | | | | | | | | | | | | |
| dest | dest=ROB2，ROB2将接受该保留站的结果 | | | | | | | | | | | | | | | | | |
| ROB  缓冲器 | num | busy | op | | | | | state | | | dst | | | | | value | | | |
| 1 | 0 |  | | | | |  | | | F4 | | | | |  | | | |
| 2 | 0 |  | | | | |  | | | F2 | | | | |  | | | |
| 3 | 1 |  | | | | |  | | | F1 | | | | |  | | | |
| …… |  |  | | | | |  | | |  | | | | |  | | | |
| 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  目标地址：该指令执行结果应写入的目标寄存器或存储单元地址  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。  就绪字段：指出指令是否已经完成执行并且数据已就绪 | | | | | | | | | | | | | | | | | | |
| 寄存器 | F0 | F1 | F2 | | F3 | | F4 | | | F5 | | F6 | | F7 | | | | F[rs] | |
| 寄存器  状态表 | Q0 | Q1 | Q2 | | Q3 | | Q4 | | | Q5 | | Q6 | | Q7 | | | | Q[rs] | |
| Qi=0：寄存器Fi中的数据就绪  Qi=3：寄存器Fi中的数据未就绪，由ROB3提供 | | | | | | | | | | | | | | | | | | |

## Load

|  |  |  |  |
| --- | --- | --- | --- |
| IF | 取指  PC=PC+4(设一条指令4B) | | |
| ID | IS | 查找空闲RSlb和ROB：  while [(RSlb2.busy=0) & (ROB1.busy=0)];  有空闲→不冲突：指令流出，进入下一段  无空闲→冲突：不流出，停止等待 | |
| RO | Q[rs]=0：F[rs]→RSlb2.vs；0→RSlb2.qs  Q[rs]=RSadd1：RSadd1→RSlb2.vs | |
| Imm→RSlb2.A | |
| RSlb2→RSlb2.vt | |
| RSlb2.busy=1； ROB1.busy=1； | |
| RSlb2.op=IR[op]； ROB1.op=IR[op]； | |
|  | |
| EX | 等待就绪 | | while (RSlb2.qs=0); |
| 计算有效地址 | | RSlb2.A+RSlb2.vs→RSlb2.A |
| 访存读取数据 | | Mem[RSlb2.A]→RSlb2.vt |
| WR | while (CDB就绪) ;  RSlb2.vt→Reg[x]；Q[x]=0；等待该结果的寄存器x  RSlb2.vt→RS[y]；Q[y]=0；等待该结果的保留站y  RSlb2.busy=0； //释放该保留站 | | |

查看操作码，检测结构冲突：查看操作码对应的保留站(设为X)

有空闲→不冲突：指令流出，进入下一段

无空闲→冲突：不流出，停止等待

查看操作数在寄存器中

就绪→不冲突：将操作数送入保留站X，进入下一段

未就绪→冲突：将产生该操作数的保留站e的标识送入保留站X，

## Store

|  |  |  |  |
| --- | --- | --- | --- |
| IF | 取指  PC=PC+4(设一条指令4B) | | |
| ID | IS | 查找空闲RSlb和ROB：  while [(RSlb2.busy=0) & (ROB1.busy=0)];  有空闲→不冲突：指令流出，进入下一段  无空闲→冲突：不流出，停止等待 | |
| RO | Q[rs]=0：F[rs]→RSsb3.vs；0→RSsb3.qs  Q[rs]=RSadd1：RSadd1→RSsb3.vs | |
| RSsb3.busy=1 | |
| Imm→RSsb3.A | |
| RSsb3.op=IR[op] | |
| Q[rt]=0：F[rt]→RSsb3.vt；0→RSsb3.qt  Q[rt]=RSlb1：RSlb1→RSsb3.vt | |
| EX | 等待就绪 | | while (RSsb2.qs=0);  不断地监测CDB |
| 计算有效地址 | | RSsb2.A+RSsb2.vs→RSsb2.A |
| WR | 等待就绪 | | while (CDB就绪 & RSsb3.qt=0)； |
|  | | RSsb3.vt→ROB1.value  RSsb3.vt→RSadd2；  RSsb3.vt→RSadd1； |
| 释放保留站 | | RSsb2.busy=0； |
| Commit | 等待就绪 | | while (前面的指令都确认) ; |
|  | | 猜测正确：ROB1.value→Mem[RSlb2.A]  猜测错误：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。 |
|  | |  |
|  | |  |

基本思想：对分支指令的结果进行猜测，并假设这个猜测总是对的，然后按这个猜测结果继续取、流出和执行后续的指令。

执行指令的结果不是写回到寄存器或存储器，而是放到ROB缓冲器中

等到相应的指令得到“确认”（commit）（即确实是应该执行的）之后，才将结果写入寄存器或存储器

## ALU

|  |  |
| --- | --- |
| Commit | 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。  猜测正确，把在ROB中的结果写到寄存器或存储器  ROB1.value→Mem[RSlb2.A]  猜测错误，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。  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标寄存器，并从ROB中删除该指令。 |

## Branch

基于硬件的前瞻执行(speculation)

基本思想：对分支指令的结果进行猜测，并假设这个猜测总是对的，然后按这个猜测结果继续取、流出和执行后续的指令。

执行指令的结果不是写回到寄存器或存储器，而是放到ROB缓冲器中

等到相应的指令得到“确认”（commit）（即确实是应该执行的）之后，才将结果写入寄存器或存储器

基于硬件的前瞻执行结合了三种思想：

动态分支预测。用来选择后续执行的指令。

在控制相关的结果尚未出来之前，前瞻地执行后续指令。

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。

把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：

实现前瞻的关键思想：允许指令乱序执行，但必须顺序确认。

|  |  |  |
| --- | --- | --- |
| IF |  | |
| ID | IS | 从浮点指令队列的头部取一条指令。  如果有空闲的保留站r且有空闲的ROB项b，就流出该指令，并把相应的信息放入保留站r和ROB项b。  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。 |
| RO | 如果有操作数尚未就绪，就等待，并不断地监测CDB  (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 |
| EX |  | |
| WR | 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。  释放产生该结果的保留站。  store指令在本阶段完成，其操作为:  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，这时才将之写入分配给该store指令的ROB项 | |
| Commit | 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。  如果前面所做的猜测是对的，把在ROB中的结果写到寄存器或存储器。  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。  当预测错误的分支指令到达ROB队列的头部时，清空ROB，并从分支指令的另一个分支重新开始执行。(错误的前瞻执行)  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。 | |

前瞻执行

通过ROB实现了指令的顺序完成。

能够实现精确异常。

很容易地推广到整数寄存器和整数功能单元上。

主要缺点：所需的硬件太复杂。

## JMP