[数据通路 2](#_Toc505861068)

[指令系统 5](#_Toc505861069)

[Load 6](#_Toc505861070)

[Store 7](#_Toc505861071)

[ALU 8](#_Toc505861072)

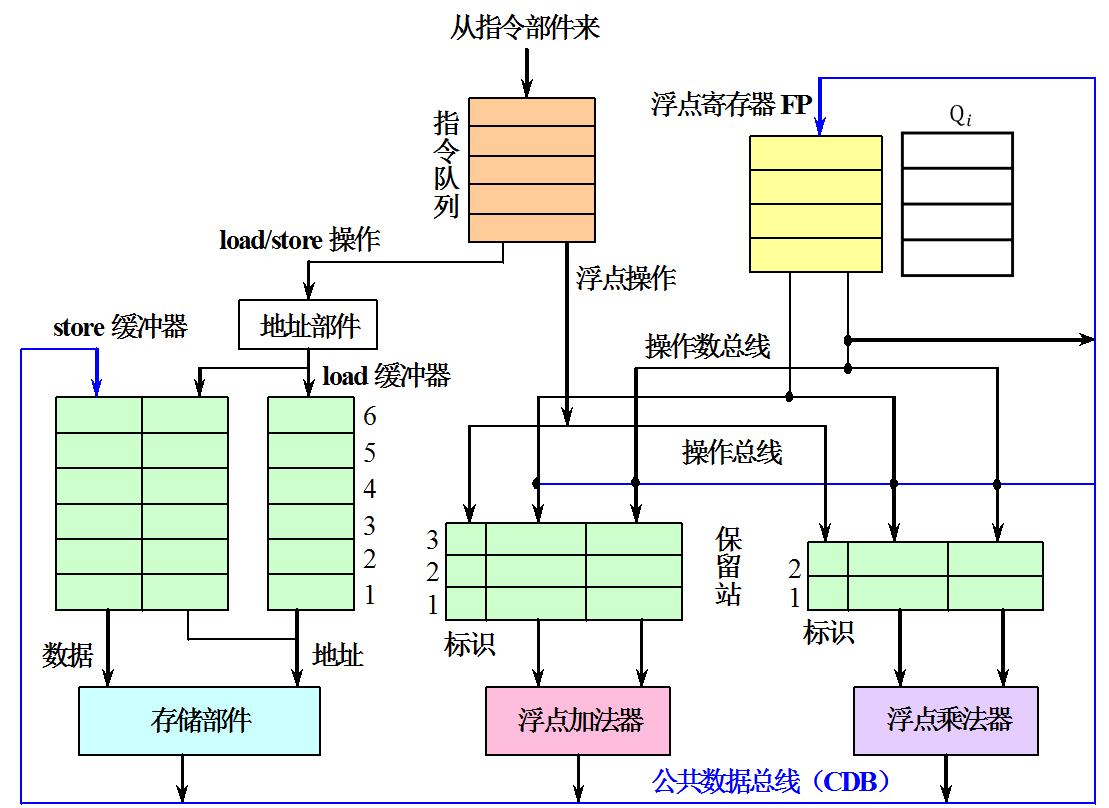
[Branch-BHT 9](#_Toc505861073)

[Branch-BTB 10](#_Toc505861074)

[JMP 11](#_Toc505861075)

[多流出(超标量) 12](#_Toc505861076)

## 数据通路



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 保留站 | 标识 | busy | op | | | vs | | vt | | qs | | qt | | A |
| RSlb1 | 0 |  | | |  | |  | |  | |  | |  |
| RSlb2 | 0 |  | | |  | |  | |  | |  | |  |
| RSlb3 | 1 |  | | |  | |  | |  | |  | |  |
| RSlb4 | 0 |  | | |  | |  | |  | |  | |  |
| RSlb5 | 0 |  | | |  | |  | |  | |  | |  |
| RSsb1 | 1 |  | | |  | |  | |  | |  | |  |
| RSsb2 | 1 |  | | |  | |  | |  | |  | |  |
| RSsb3 | 0 |  | | |  | |  | |  | |  | |  |
| RSsb4 | 0 |  | | |  | |  | |  | |  | |  |
| RSsb5 | 0 |  | | |  | |  | |  | |  | |  |
| RSadd1 | 1 |  | | |  | |  | |  | |  | |  |
| RSadd2 | 0 |  | | |  | |  | |  | |  | |  |
| RSadd3 | 0 |  | | |  | |  | |  | |  | |  |
| RSmult1 | 1 |  | | |  | |  | |  | |  | |  |
| RSmult2 | 0 |  | | |  | |  | |  | |  | |  |
| 本例 | 共15个保留站：其中有，5\*load缓冲器；5\*store缓冲器；  3\*加法运算器保留站；5\*乘法运算器保留站； | | | | | | | | | | | | |
| 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 | | | | | | | | | | | | |
| 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=RSadd1：寄存器Fi中的数据未就绪，由RSadd1提供 | | | | | | | | | | | | | |
| 公共数  据总线  CDB | 运算器、LB→CDB→除LB以外的所有部件的入口 | | | | | | | | | | | | | |

|  |  |  |
| --- | --- | --- |
| 结构冲突 |  | |
| 数据冲突 | RAW | 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小 |
| WAR | 寄存器换名：操作数→要么是数据本身，要么是保留站的标识，都不再与寄存器有关系。  计算结果通过CDB直接从产生它的保留站传送到所有需要它的功能部件，也不用经过寄存器，由此解决消除了WAW冲突和WAR冲突导致的停顿 |
| WAW |  |
| 控制冲突 |  | |

|  |  |
| --- | --- |
| 异常 | 指令乱序完成带来的最大问题: 异常处理比较复杂  （精确异常处理、不精确异常处理）  动态调度要保持正确的异常行为：  只有那些在程序严格按程序顺序执行时会发生的异常，才能真正发生。  保持正确的异常行为：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后，才允许它产生异常。  3.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 |
| 精确异常 | 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。 |
| 不精确异常 | 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。  发生不精确异常的原因：因为当发生异常（设为指令i）时：  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；  流水线可能还没完成按程序顺序是指令i之前的指令。  不精确异常使得在异常处理后难以接着继续执行程序 |

## 指令系统

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Load | 0~5 | 6~10 | 11~15 | 16~31 | | |
| 操作码 | rs | rt | immediate | | |
| Store | 0~5 | 6~10 | 11~15 | 16~31 | | |
| 操作码 | rs | rt | immediate | | |
| ALU | 0~5 | 6~10 | 11~15 | 16~20 | 21~25 | 26~31 |
| 操作码 | rs | rt | rd | shamt | funct |
| Branch | 0~5 | 6~10 | 11~15 | 16~31 | | |
| 操作码 | rs | rt | immediate | | |
| JMP | 0~5 | 6~31 | | | | |
| 操作码 | 与PC相加的偏移量 | | | | |

## Load

|  |  |  |  |
| --- | --- | --- | --- |
| IF | 取指  PC=PC+4(设一条指令4B) | | |
| ID | IS | 查找空闲RSlb，RSlb2.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 | |
| RSlb2.busy=1 | |
| Imm→RSlb2.A | |
| RSlb2.op=IR[op] | |
| RSlb2→RSlb2.vt | |
| 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，RSsb3.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); |
| 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→Mem[RSlb2.A]  RSsb2.busy=0； //释放该保留站 |
|  | |  |

## ALU

|  |  |  |
| --- | --- | --- |
| IF | 取指  PC=PC+4(设一条指令4B) | |
| ID | IS | 查找空闲RSadd，RSadd2.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]→RSadd2.vs；0→RSadd2.qs  Q[rs]=RSadd1：RSadd1→RSadd2.vs |
| Q[rt]=0：F[rt]→RSadd2.vt；0→RSadd2.qt  Q[rt]=RSlb1：RSlb1→RSadd2.vt |
| RSadd2→Q[rd] |
| RSadd2.busy=1;  RSadd2.op=IR[op] |
| EX | while[ (RSadd2.qs=0) & (RSadd2.qt=0) ];  用相应的功能部件开始执行指令规定的操作 | |
| WR | while (CDB就绪) ;  查看所有等待本指令结果的缓存器：  Q[x]=RSadd2：F[rd]→F[x]  RSlb1.qs=RSadd2：F[rd]→RSlb1.vs  RSmult1.qt=RSadd2：F[rd]→RSlb1.vt  RSx.busy=0；释放该保留站 | |

## Branch-BHT

|  |  |  |
| --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR  PC+4→IF/ID.NPC | |
| ID | 查看  操作码 | IF/ID.IR[op]→CU  IF/ID.IR[op]=branch⇒终止所有后继指令 |
| 查看  BHT | IF/ID.IR[op]→BHT，找到本指令对应的表项，查看分支记录 |
|  |  |
|  |  |
| EX |  |  |
|  |  |
| MEM |  |  |
|  |  |
| WB |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 分支  历史表  BHT | BHT表项：分支指令+分支记录(2bit)  存储：跟分支指令一起放在指令Cache中，或使用专门的硬件 | | |
| 动态  分支  预测 | 访问BHT，  00或01：执行失败预测，PC值保持不变，继续处理后续的指令，流水线不断流  10或11：执行成功预测，从分支目标地址取后续指令，作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令  原理：给两次机会，连续两次预测错误才改变预测  BHT方法使用条件：判定分支是否成功所需的时间大于确定分支目标地址所需的时间。  5段经典流水线由于判定分支是否成功和计算分支目标地址都是在ID段完成，所以BHT方法不会给该流水线带来好处。 | | |
| 状态  修改 |  | 实际成功 | 实际失败 |
| 预测成功 | 11→11；10→11 | 11→10；10→00 |
| 预测失败 | 01→11；00→01 | 00→00；01→00； |

## Branch-BTB

|  |  |
| --- | --- |
| 目标缓冲器  BTB | BTB表项：1.执行过的成功分支指令的地址；  2.预测的分支目标地址 |

|  |  |
| --- | --- |
| IF | 取指  PC→BTB，查看BTB中是否有匹配的项  匹配：执行成功预测； 无匹配：执行失败预测 |
| ID | 分支失败：正常执行  分支成功： |
| EX |  |



采用BTB后，各种可能情况下的延迟：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 预测 | 实际 | 延迟周期 |
| 指令在BTB中 | 成功 | 成功 | 0 |
| 失败 | 2 |
| 指令不在BTB中 | 失败 | 成功 | 2 |
| 失败 | 0 |

BTB的另一种形式：

BTB表项：1.执行过的成功分支指令的地址；2.预测的分支目标地址3.预测的分支目标地址 4.预测的分支目标地址 ……

有三个潜在的好处：

1.更快地获得分支目标处的指令；

2.可以一次提供分支目标处的多条指令，这对于多流出处理器是很有必要的；

3.使我们可以进行称为分支折叠（branch folding）的优化。实现零延迟无条件分支，甚至有时还可以做到零延迟条件分支。

## JMP

## 多流出(超标量)

扩展Tomasulo算法：支持两路超标量，每个时钟周期流出两条指令；一条是整数指令，另一条是浮点指令。

采用一种比较简单的方法

指令按顺序流向保留站，否则会破坏程序语义。

将整数所用的表结构与浮点用的表结构分离开，分别进行处理，这样就可以同时地流出一条浮点指令和一条整数指令到各自的保留站。

有两种不同的方法可以实现多流出。

关键在于：对保留站的分配和对流水线控制表格的修改。

(1)在半个时钟周期里完成流出步骤，这样一个时钟周期就能处理两条指令。

(2)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两种方法都采用。

举例

上述双流出动态调度流水线的性能受限于以下3个因素：

1.整数部件和浮点部件的工作负载不平衡，没有充分发挥出浮点部件的作用。应该设法减少循环中整数型指令的数量。

2.每个循环迭代中的控制开销太大：5条指令中有两条指令是辅助指令。应该设法减少或消除这些指令。

3.控制相关使得处理机必须等到分支指令的结果出来后才能开始下一条L.D指令的执行。