## 概念题

解释下列名词：

**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。

**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。

## 填空题

2.1 开发指令级并行的方法主要有两类：基于硬件的动态开发方法以及基于软件的静态开发方法。

2.2 说出两种比较典型的动态调度算法：记分牌方法、Tomasulo算法。

2.3 要扩充Tomasulo算法支持前瞻执行，需将Tomasulo算法中的“写结果”段分为写结果和指令确认两个段。

2.4 前瞻执行允许指令乱序执行，但要求程序顺序确认。

2.5 动态分支预测的依据是从转移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历史记录，来预测下一次转移的方向。

## 问答题

3.1 指令的动态调度有何优点？

（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。

3.2 简述Tomasulo算法的基本思想。

其**核心思想**是:

(1)记录和检测指令相关,操作数一日就绪就立即执行，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)。

## 应用题

4.1 对于下述指令序列：（注：于书中题目略有改动，第四条指令F2，F6位置互换）

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

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

(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。

（1）下面给出了当采用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。

|  |  |  |  |
| --- | --- | --- | --- |
| 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, **F6, F2** | √ |  |  |
| DIV.D F10, F0, F6 | √ |  |  |
| ADD.D F6, F8, F2 | √ |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | 保留站 | | | | | | |
| 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 |  |  |

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

（2）MUL.D指令准备写入结果时各状态表的内容如下所示。

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完成并写入F6，不会出现错误。

|  |  |  |  |
| --- | --- | --- | --- |
| 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, **F6, F2** | √ | √ | √ |
| DIV.D F10, F0, F6 | √ |  |  |
| ADD.D F6, F8, F2 | √ | √ | √ |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |

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