[结构 2](#_Toc507316727)

[数据通路 2](#_Toc507316728)

[关键操作 4](#_Toc507316729)

[指令系统 5](#_Toc507316730)

[Load 6](#_Toc507316731)

[Store 7](#_Toc507316732)

[ALU 8](#_Toc507316733)

[Branch-冻结/排空 9](#_Toc507316734)

[Branch-延迟分支 10](#_Toc507316735)

[Branch-失败预测法 11](#_Toc507316736)

[多流出技术 12](#_Toc507316737)

[超标量 12](#_Toc507316738)

[VLIW 15](#_Toc507316739)

# 结构

## 数据通路



|  |  |  |
| --- | --- | --- |
| 通用  寄存器  GPR | ID段读通用寄存器组  WB写通用寄存器组  因此将写操作安排在时钟周期的前半拍，读操作在后半拍 | |
| 加法器 | 用于PC值的顺序递增 | |
| 存储器 | 使用分离的指令存储器和数据存储器  避免 **IF段读指令** 与 **MEM段访存** 发生结构冲突 | |
| 流水  寄存器 | 设置在段与段之间：IF/ID、ID/EX、EX/MEM、MEM/WB  [由若干个寄存器构成](file:///F:\CS\Architecture\3-2-21%20流水线寄存器的构成.exe)，保存上一段处理结果，传递给后一段  ID/EX.IR[op]：流水寄存器的ID/EX中的子寄存器IR中的op字段 | |
| NPC：下条指令的地址  IR：当前指令的指令字  A：从GPR读出的操作数  B：从GPR读出的另一操作数 | Imm：符号扩展后的立即数  cond：条件判定结果：cond=1分支成功，cond=0分支失败  ALUo：ALU运算结果  LMD：load从存储器读出的数据 |

|  |  |  |  |
| --- | --- | --- | --- |
| 多路  选择器 | MUX1 | (EX/MEM.IR=branch)&(EX/MEM.cond=1) | EX/MEM.ALUo |
| else | PC+4 |
| MUX2 | ID/MEM.IR=分支指令 | ID/EX.NPC |
| else | ID/EX.A |
| MUX3 | ID/EX.IR=RR-ALU | ID/EX.B |
| else | ID/EX.Imm |
| MUX4 | MEM/WB.IR=load | MEM/WB.LMD |
| else | MEM/WB.ALUo |
| MUX5 |  | MEM/WB.IR[rd] |
|  | MEM/WB.IR[rt] |

## 关键操作

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 冲突 | RAW | ID/EX.IR[op]  本条指令 | | IF/ID.IR[op]  下一指令 | | 检测 | | 处理 | |
| load | | RR-ALU | | ID/EX.IR[rt]=IF/ID.IR[rs] | | 停止当前处于IF段和ID段的指令：  ID/EX.IR中的操作码改为全0(表示空操作)  IF/ID寄存器的内容回送到自己的入口 | |
| ID/EX.IR[rt]=IF/ID.IR[rt] | |
| load/store/ALU/branch | | ID/EX.IR[rt]=IF/ID.IR[rs] | |
|  | | | | | |  | |
| 定向  逻辑 | 发生数据冲突时执行冲突处理，停顿后继指令的运行，直到前面指令的结果写入寄存器，定向逻辑使结果从其产生的地方直接送到其他指令需要得地方，不必经过寄存器，以此减少停顿时间  检测定向：在ID段比较流水寄存器中的寄存器地址  实现定向：增加通路，ALU运算结果、访存得到的数据→ALU输入端  优点：降低流水线的硬件复杂度，不必挂起已经改变了机器状态的指令 | | | | | | | |  |
| 本条指令  ID/EX.IR[op] | | 前驱指令 | | 检测：  前驱指令写，后继指令读 | | 定向措施 | |  |
| RR-ALU | | EX/MEM.IR[op]  =RR-ALU | | EX/MEM.IR[rd]=ID/EX.IR[rt]  EX/MEM.IR[rd]=ID/EX.IR[rs] | | EX/MEM.ALUo  →ALU | |  |
| MEM/WB.IR[op]  =load | | MEM/WB.IR[rt]=ID/EX.IR[rt]  MEM/WB.IR[rt]=ID/EX.IR[rs] | | MEM/WB.LMD  →ALU | |  |
| store | | EX/MEM.IR[op]  =RR-ALU | | EX/MEM.IR[rd]=ID/EX.IR[rs] | | EX/MEM.ALUo  →ALU | |  |
| MEM/WB.IR[op]  =load | | MEM/WB.IR[rt]=ID/EX.IR[rs] | | MEM/WB.LMD  →ALU | |  |

# 指令系统

## Load

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=Load】  PC+4→IF/ID.NPC  PC=PC+4 | | | |
| ID | 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR | | |
| RO | Regs[IF/ID.IR[rs]]→ID/EX.A | | |
| 数据  冲突 | 相关：执行避免冲突措施，  不相关：直接进入下一段 | | |
| 定向  逻辑 |  | | |
| EX | 传递 | ID/EX.IR→EX/MEM.IR | | |
|  | ID/EX.A + ID/EX.Imm→EX/MEM.ALUo；  ID/EX.B→EX/MEM.B | | |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR | | |
|  | Mem[EX/MEM.ALUo]→MEM/WB.LMD | | |
| 定向  逻辑 | Load写寄存器，有后继指令想读取本Load所写数据 | | |
| ID/EX.IR[op] | 检测寄存器 | 定向通路 |
| ALU-RR | MEM/WB.IR[rt]=ID/EX.IR[rt] | MEM/WB.LMD→ALU |
| MEM/WB.IR[rt]=ID/EX.IR[rs] | MEM/WB.LMD→ALU |
| Store | MEM/WB.IR[rt]=ID/EX.IR[rs] | MEM/WB.LMD→ALU |
|  |  |  |
| WB | 传递 |  | | |
|  | MEM/WB.LMD→Regs[MEM/WB.IR[rt]] | | |

## Store

|  |  |  |
| --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=Store】  PC+4→IF/ID.NPC  PC=PC+4 | |
| ID | 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR |
| RO | Regs[IF/ID.IR[rs]]→ID/EX.A |
| 数据  冲突 | 相关：执行避免冲突措施，  不相关：直接进入下一段 |
| 定向  逻辑 |  |
| EX | 传递 | ID/EX.IR→EX/MEM.IR |
|  | ID/EX.A + ID/EX.Imm→EX/MEM.ALUo；  ID/EX.B→EX/MEM.B |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR |
|  | EX/MEM.B→Mem[EX/MEM.ALUo] |
| WB | 传递 |  |
|  | MEM/WB.LMD→Regs[MEM/WB.IR[rt]] |

## ALU

|  |  |  |  |
| --- | --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=ALU】  PC+4→IF/ID.NPC  PC=PC+4 | | |
| ID | 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR | |
| RO | ALU-RR | Regs[IF/ID.IR[rs]]→ID/EX.A  Regs[IF/ID.IR[rt]]→ID/EX.B |
| ALU-RImm | Regs[IF/ID.IR[rs]]→ID/EX.A  Regs[IF/ID.IR[imm]]→16..32→ID/EX.Imm |
| 数据  冲突 | 相关：执行避免冲突措施，  不相关：直接进入下一段 | |
| 定向  逻辑 |  | |
| EX | 传递 | ID/EX.IR→EX/MEM.IR | |
|  | ALU-RR | ID/EX.A func ID/EX.B→EX/MEM.ALUo |
| ALU-RImm | ID/EX.A op ID/EX.Imm→EX/MEM.ALUo |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR | |
|  | ALU | EX/MEM.ALUo→MEM/WB.ALUo |
| WB | 传递 |  | |
|  | RR-ALU | MEM/WB.ALUo→Regs[MEM/WB.IR[rd]] |
| RImm-ALU | MEM/WB.ALUo→Regs[MEM/WB.IR[rt]] |

## Branch-冻结/排空

|  |  |  |
| --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=branch】  PC+4→IF/ID.NPC  PC=PC+4 | |
| ID | 查看  操作码 | IF/ID.IR[op]→CU  IF/ID.IR[op]=branch⇒终止所有后继指令 |
| 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR |
| RO | Regs[IF/ID.IR[rs]]→ID/EX.A  Regs[IF/ID.IR[imm]]→16..32→ID/EX.Imm |
| EX | 传递 | ID/EX.IR→EX/MEM.IR |
|  | ID/EX.NPC+ID/EX.Imm<<2→EX/MEM.ALUo  判断ID/EX.A：成功cond=1；失败cond=0 |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR |
|  | cond=1：EX/MEM.ALUo→PC  cond=0：PC+4→PC |
| WB |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分支  成功 | branch | IF | ID | EX | MEM | WB |  |  |  |  |  |
| branch+1 |  | IF |  |  |  |  |  |  |  |  |
| branch\_dst |  |  |  |  | IF | ID | EX | MEM | WB |  |
| branch\_dst+1 |  |  |  |  |  | IF | ID | EX | MEM | WB |
| 分支  失败 | branch | IF | ID | EX | MEM | WB |  |  |  |  |  |
| branch+1 |  | IF |  |  |  |  |  |  |  |  |
| branch+1 |  |  |  |  | IF | ID | EX | MEM | WB |  |
| branch+2 |  |  |  |  |  | IF | ID | EX | MEM | WB |

## Branch-延迟分支

|  |  |  |
| --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=branch】  PC+4→IF/ID.NPC  PC=PC+4 | |
| ID | 查看  操作码 | IF/ID.IR[op]→CU  IF/ID.IR[op]=branch |
| 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR |
| RO | Regs[IF/ID.IR[rs]]→ID/EX.A  Regs[IF/ID.IR[imm]]→16..32→ID/EX.Imm |
| EX | 传递 | ID/EX.IR→EX/MEM.IR |
|  | ID/EX.NPC+ID/EX.Imm<<2→EX/MEM.ALUo  判断ID/EX.A：成功cond=1；失败cond=0 |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR |
|  | cond=1：EX/MEM.ALUo→PC  cond=0：PC+4→PC |
| WB |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分支  成功 | branch | IF | ID | EX | MEM | WB |  |  |  |  |
| DelaySlot |  | IF | ID | EX | MEM | WB |  |  |  |
| DelaySlot+1 |  |  | IF | ID | EX | MEM | WB |  |  |
| DelaySlot+2 |  |  |  | IF | ID | EX | MEM | WB |  |
| branch\_dst |  |  |  |  | IF | ID | EX | MEM | WB |
| 分支  失败 | branch | IF | ID | EX | MEM | WB |  |  |  |  |
| DelaySlot |  | IF | ID | EX | MEM | WB |  |  |  |
| DelaySlot+1 |  |  | IF | ID | EX | MEM | WB |  |  |
| DelaySlot+2 |  |  |  | IF | ID | EX | MEM | WB |  |
| branch+1 |  |  |  |  | IF | ID | EX | MEM | WB |

|  |
| --- |
| 编译器在指令调度时，选择合适的指令(称为延迟槽指令)，插入到分支指令和分支目标指令之间，填补延迟空白；  DelaySlot：延迟槽指令，分支指令实际的后继指令  branch\_dst：分支成功时的后继指令，实际执行时在经过3个DelaySlot后才执行  branch+1：分支失败时的后继指令，实际执行时在经过3个DelaySlot后才执行 |
| 从前调度：该分支指令的某条独立前驱指令  从目标处调度：预测分支成功，将目标处指令**复制**到延迟槽  从失败处调度：预测分支成功，将顺序下一指令**复制**到延迟槽 |

## Branch-失败预测法

|  |  |  |
| --- | --- | --- |
| IF | Mem[PC]→IF/ID.IR【IF/ID.IR[op]=branch】  PC+4→IF/ID.NPC  PC=PC+4 | |
| ID | 查看  操作码 | IF/ID.IR[op]→CU  IF/ID.IR[op]=branch⇒终止所有后继指令 |
| 传递 | IF/ID.NPC→ID/EX.NPC  IF/ID.IR→ID/EX.IR |
| RO | Regs[IF/ID.IR[rs]]→ID/EX.A  Regs[IF/ID.IR[imm]]→16..32→ID/EX.Imm |
| EX | 传递 | ID/EX.IR→EX/MEM.IR |
|  | ID/EX.NPC+ID/EX.Imm<<2→EX/MEM.ALUo  判断ID/EX.A：成功cond=1；失败cond=0 |
| MEM | 传递 | EX/MEM.IR→MEM/WB.IR |
|  | cond=1：EX/MEM.ALUo→PC  cond=0：PC+4→PC |
| WB |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分支  成功 | branch | IF | ID | EX | MEM | WB |  |  |  |  |  |
| branch+1 |  | IF | ID |  |  |  |  |  |  |  |
| branch+2 |  |  | IF |  |  |  |  |  |  |  |
| branch\_dst |  |  |  |  | IF | ID | EX | MEM | WB |  |
| branch\_dst+1 |  |  |  |  |  | IF | ID | EX | MEM | WB |
| 分支  失败 | branch | IF | ID | EX | MEM | WB |  |  |  |  |  |
| branch+1 |  | IF | ID | EX | MEM | WB |  |  |  |  |
| branch+2 |  |  | IF | ID | EX | MEM | WB |  |  |  |

# 多流出技术

## 超标量

单流出和多流出处理机执行指令的时空图对比

多流出处理机有两种基本风格

|  |  |
| --- | --- |
| 超标量  Superscalar | 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。  设这个上限为n，就称该处理机为n流出。  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。 |
| 超长指令字  VLIW | 每个时钟周期流出的指令条数固定，这些指令构成一条长指令或者一个指令包。  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。  指令调度是由编译器静态完成的。 |

超标量处理机与VLIW处理机相比有两个优点：

1.超标量结构对程序员是透明的，因为处理机能自己检测下一条指令能否流出，从而不需要重新排列指令来满足指令的流出。

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： 使用动态超标量调度技术。

在典型的超标量处理器中，每个时钟周期可流出1到8条指令。

指令按序流出，在流出时进行冲突检测。在当前流出的指令序列中，不存在数据冲突或者相关冲突。

举例：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（称为流出包）。

在一个时钟周期内，对收到的指令，有可能是全部都能流出，也可能是只有一部分能流出。

流出部件检测结构冲突或者数据冲突。

一般分两阶段实现：

第一阶段：进行流出包内的冲突检测，选出初步判定可以流出的指令。

第二阶段：检测所选出的指令与正在执行的指令是否有冲突。

MIPS处理机是怎样实现超标量的呢？

1.假设：每个时钟周期流出两条指令：1条整数型指令＋1条浮点操作指令

其中，把load指令、store指令、分支指令归类为整数型指令。

2.要求：同时取两条指令（64位），译码两条指令（64位）。

3.对指令的处理包括以下步骤：

从Cache中取两条指令。

确定哪几条指令可以流出（0～2条指令）。

把它们发送到相应的功能部件。

4.双流出超标量流水线中指令的执行过程

假设：所有的浮点指令都是加法指令，其执行时间为两个时钟周期。

为简单起见，下图中总是把整数指令放在浮点指令的前面。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令类型 | 流水线工作情况 | | | | | | | |
| 整数指令 | IF | ID | EX | MEM | WB |  |  |  |
| 浮点指令 | IF | ID | EX | EX | MEM | WB |  |  |
| 整数指令 |  | IF | ID | EX | MEM | WB |  |  |
| 浮点指令 |  | IF | ID | EX | EX | MEM | WB |  |
| 整数指令 |  |  | IF | ID | EX | MEM | WB |  |
| 浮点指令 |  |  | IF | ID | EX | EX | MEM | WB |
| 整数指令 |  |  |  | IF | ID | EX | MEM | WB |
| 浮点指令 |  |  |  | IF | ID | EX | EX | MEM |

4.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。

5.浮点load或浮点store指令将使用整数部件，会增加对浮点寄存器的访问冲突。增设一个浮点寄存器的读/写端口。

6.由于流水线中的指令多了一倍，定向路径也要增加。

7.限制超标量流水线的性能发挥的障碍。

load指令：load后续3条指令都不能使用其结果，否则就会引起停顿。

分支延迟 ：如果分支指令是流出包中的第一条指令，则其延迟是3条指令；否则就是流出包中的第二条指令，其延迟就是两条指令。

一个时钟周期内流出多条指令， CPI＜1。

## VLIW

1.把能并行执行的多条指令组装成一条很长的指令。（100多位到几百位）

2.设置多个功能部件。

3.指令字被分割成一些字段，每个字段称为一个操作槽，直接独立地控制一个功能部件。

4.在VLIW处理机中，所有的处理和指令安排都是由编译器完成的。

VLIW存在的一些问题

1.程序代码长度增加了

提高并行性而进行的大量的循环展开。

指令字中的操作槽并非总能填满。

解决：采用指令共享立即数字段的方法，或者采用指

令压缩存储、调入Cache或译码时展开的方法。

2.采用了锁步机制：任何一个操作部件出现停顿时，整个处理机都要停顿。

3.机器代码的不兼容性

多流出处理器受到的限制

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