# 流水线CPU的设计

## 一、CPU设计方案综述

### （一）总体设计概述

本CPU为Verilog实现的流水线MIPS - CPU，支持的指令集包含{addu、subu、ori、lw、sw、beq、lui、j、jal、jr、nop}。为了实现这些功能，CPU在顶层模块mips下并列包含了ALU、Compare、Controller、Conflict、DM、Ext、GRF、IM、PC、Pipelineregs等模块。

流水线一共分为五个级别F、D、E、M、W。F级包含了PC和IM模块以及在顶层mips文件中实现的选择器和加法器；D级包含了Controller、Compare、Ext、GRF等模块；E级包含了ALU部件以及选择器；M级包含了DM部件；W级连接到D级寄存器，并且包含了在顶层模块中实现了用于存入寄存器数据选择的功能。相邻两流水级之间还各设置了一系列流水线寄存器，用于存储流水的信息。此外，为满足数据冒险的转发需求，还顶层模块中设置了一系列控制信号用于控制转发。并且还设置了一个Conflict模块用于控制暂停信号，可以控制D级和E级的暂停；PipeLineRegs模块用于将上级信号向下一级进行传递。

### （二）总体设计概述

1. PC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号（同步复位至0x00003000） |
| en | I | 使能信号，为1时PC正常工作，为0时PC冻结 |
| next | I | 32位输入次指令地址 |
| IAddr | O | 32位输出当前指令地址 |

2. ALU

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Op1 | I | 32位运算数1 |
| Op2 | I | 32位运算数2 |
| Sel | I | 5位的功能选择信号 |
| Result | O | 32位的运算结果 |

3. compare

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| C1 | I | 32位运算数1 |
| C2 | I | 32位运算数2 |
| Sel | I | 3位的比较功能选择信号 |
| true | O | 32位的比较结果 |

4. Conflict

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| WriteRegE | I | 5位E阶段寄存器写地址 |
| WriteRegM | I | 5位M阶段寄存器写地址 |
| RegWriteE | I | E阶段寄存器写使能 |
| RegWriteM | I | M阶段寄存器写使能 |
| RegSrcE | I | 3位E阶段寄存器写数据选择 |
| RegSrcM | I | 3位M阶段寄存器写数据选择 |
| BranchD | I | D阶段分支信号 |
| JumpD | I | D阶段跳转信号 |
| JumpE | I | E阶段跳转信号 |
| RegDstD | I | D阶段寄存器写地址选择 |
| rsD | I | D阶段rs寄存器地址 |
| rtD | I | D阶段rt寄存器地址 |
| rsE | I | E阶段rs寄存器地址 |
| rtE | I | E阶段rt寄存器地址 |
| WriteRegW | I | W阶段寄存器写使能 |
| rtM | I | M阶段rt寄存器地址 |
| stall | O | 暂停信号 |
| ForwardrsD | O | D阶段rs寄存器转发选择 |
| ForwardrtD | O | D阶段rt寄存器转发选择 |
| ForwardrsE | O | E阶段rs寄存器转发选择 |
| ForwardrtE | O | E阶段rt寄存器转发选择 |
| ForwardrtM | O | M阶段rt寄存器转发选择 |

5. Controller

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| cmd | I | 32位指令 |
| Jump | O | 跳转信号 |
| RegSrc | O | 寄存器写数据选择信号 |
| MemWrite | O | 内存写使能 |
| Branch | O | 分支跳转 |
| ALUSrc | O | ALU操作数选择信号 |
| RegDst | O | 寄存器写选择信号 |
| RegWrite | O | 寄存器写使能 |
| ExtOp | O | 符号扩展选择 |
| ALUCtrl | O | ALU操作选择 |
| loen | O | HI寄存器使能 |
| hien | O | LO寄存器使能 |

6. DM

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| we | I | 写使能信号 |
| reset | I | 重置信号 |
| isu | I | 是否是无符号操作 |
| MemDst | I | 内存操作类型 |
| Addr | I | 读取/写入地址 |
| WData | I | 写入数据 |
| IAddr | I | 当前指令地址 |
| RData | O | 读取的数据 |

7. ext

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm | I | 待扩展的16位立即数 |
| EOp | I | 扩展类型 |
| ext | O | 扩展结果 |

8. GRF

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| WEnable | I | 写使能 |
| reset | I | 重置信号 |
| RAddr1 | I | 寄存器读地址1 |
| RAddr2 | I | 寄存器读地址2 |
| WAddr | I | 寄存器写地址 |
| WData | I | 寄存器写数据 |
| IAddr | I | 当前指令地址 |
| RData1 | O | 读取的数据1 |
| RData2 | O | 读取的数据2 |

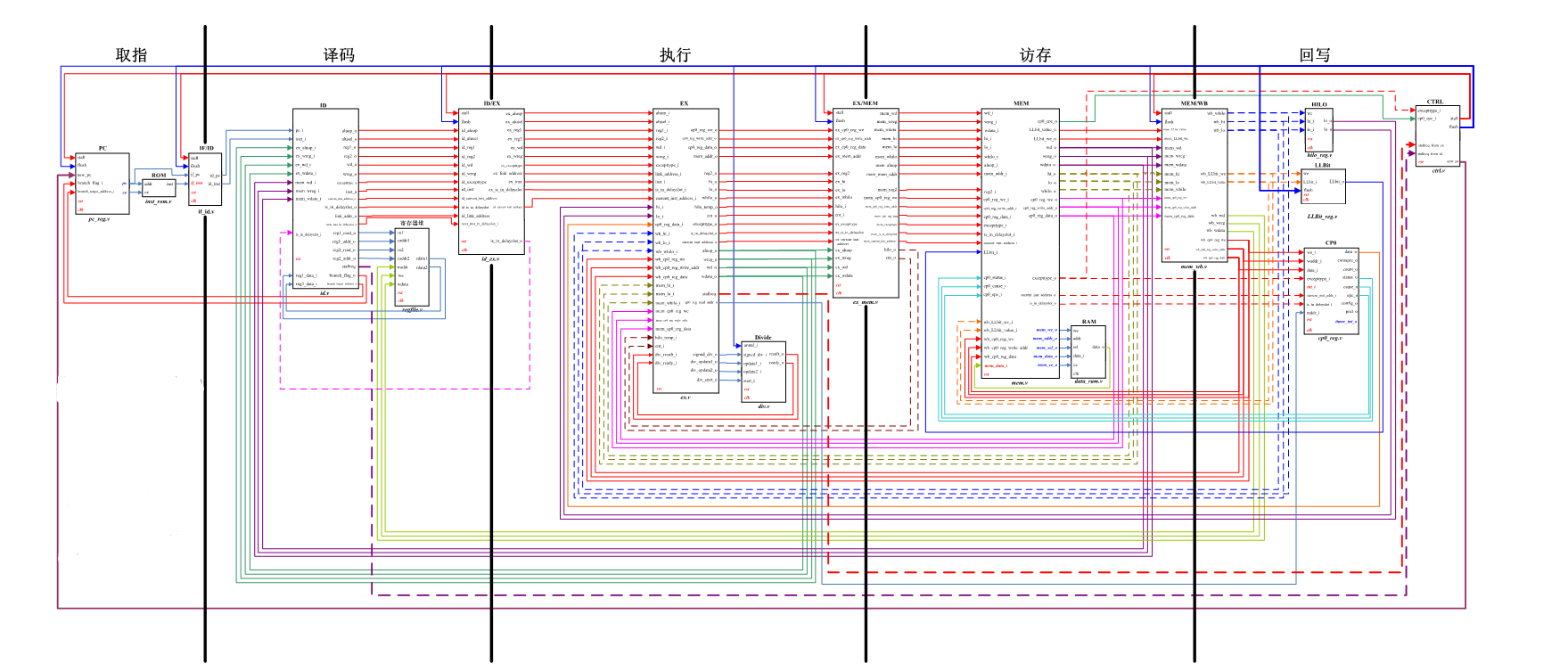
9. IM

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RAddr | I | 读地址 |
| RData | O | 根据地址读到的指令 |

10. PipeLineRegs

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| en | I | 使能信号 |
| reset | I | 重置信号 |
| in | I | 输入数据 |
| out | O | 输出数据 |

### 数据通路的综合

数据通路图：

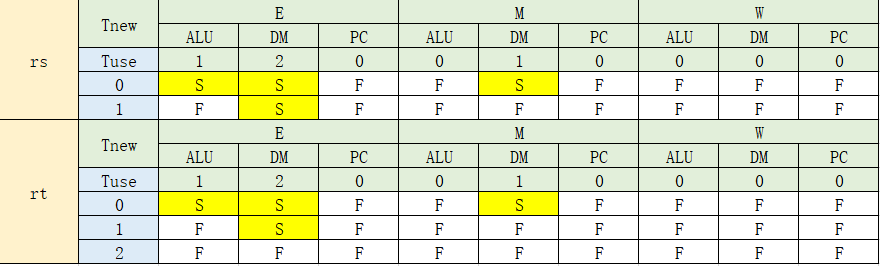
### 重要机制实现方法

1. 跳转

在顶层模块mips中译码出next的计算控制信号，计算出next后输出到F级MUX\_PcSel，MUX\_PcSel对来自D级寄存器的指令译码判断是否为跳转指令，若是则输出next至PC；反之输出当前F级pc+4（来自add4部件的输出）至PC。

1. 暂停





构建策略矩阵，当Tnew>Tuse时必须使用暂停，产生暂停信号，此时需要冻结PC的值，冻结D级流水线的值并清零E级流水线。

暂停信号由rs、rt寄存器的暂停信号取并集，当Tnew与Tuse满足矩阵中暂停条件，且写入寄存器与读取寄存器相同时相应暂停信号置1。

对D、E、M级指令分别译码即可得到Tuse与Tnew值。

1. 转发

沿用Tnew信号，本设计在宏定义时不同Tnew信号反映该级目前指令产生写入寄存器值的部件，因此在转发控制中只需要根据该级处指令写入寄存器编号与D级指令读取寄存器编号相同、该级指令产生写入寄存器值的部件两个信号判断转发哪一数据。

对M、W级指令分别译码即可得到该两级产生的写入寄存器数据来源，从而明确转发的数据来源。

此处需注意，如果写入寄存器为0号寄存器，不进行转发。

## 测试方案

### 典型测试样例

|  |  |
| --- | --- |
| Tuse\_rs=0，Tnew\_E=ALU，暂停 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x302c ##  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003014: $10 <= 00000005  @0000301c: $12 <= 0000302c  @00003024: $ 1 <= 00000009  @0000302c: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_E=DM，暂停 | |
| ori $1, $0, 256  ori $6, $0, 256  ori $5, $0, 4  sw $1, 4($5)  lw $10, 4($5) ##  beq $10, $6, label ##  nop  ori $20, $0, 2  label:  lui $4, 0x1234 | @00003000: $ 1 <= 00000100  @00003004: $ 6 <= 00000100  @00003008: $ 5 <= 00000004  @0000300c: \*00000008 <= 00000100  @00003010: $10 <= 00000100  @00003020: $ 4 <= 12340000 |
| Tuse\_rs=0，Tnew\_E=PC | |
| Tuse\_rs=0，Tnew\_M=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $30, $0, 56  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x3034 ##  ori $29, $0, 58  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $30 <= 00000038  @00003018: $10 <= 00000005  @00003020: $12 <= 00003034  @00003024: $29 <= 0000003a  @0000302c: $ 1 <= 00000009  @00003034: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_M=DM，暂停 | |
| ori $1, $0, 0x3038  ori $5, $0, 0x3038  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $3, $0, 123  beq $10, $5, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) ##  lui $26, 0x1234  jr $6 ##  nop  ori $23, $0, 125  ori $24, $0, 156 | @00003000: $ 1 <= 00003038  @00003004: $ 5 <= 00003038  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003038  @00003010: $10 <= 00003038  @00003014: $ 3 <= 0000007b  @00003024: $ 6 <= 00003038  @00003028: $26 <= 12340000  @00003038: $24 <= 0000009c |
| Tuse\_rs=0，Tnew\_M=PC，转发 | |
| ori $5, $0, 0x3010  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  beq $31, $5, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003010  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @00003020: $ 9 <= 12340000 |
| Tuse\_rs=0，Tnew\_W=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $27, $0, 256  ori $22, $0, 355  beq $5, $2, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x303c ##  addu $23, $2, $4  subu $24, $23, $4  jr $12 ##  ori $1, $0, 9  lui $9, 0x2222  ori $12, $0, 326 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $27 <= 00000100  @00003014: $22 <= 00000163  @0000301c: $10 <= 00000005  @00003024: $12 <= 0000303c  @00003028: $23 <= 00003001  @0000302c: $24 <= 00003000  @00003034: $ 1 <= 00000009  @0000303c: $12 <= 00000146 |
| Tuse\_rs=0，Tnew\_W=DM，转发 | |
| ori $1, $0, 0x3040  ori $5, $0, 0x3040  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $16, $0, 258  ori $3, $0, 123  beq $10, $5, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) ##  lui $26, 0x1234  addu $17, $16, $3  jr $6 ##  nop  ori $23, $0, 125  ori $24, $0, 156 | @00003000: $ 1 <= 00003040  @00003004: $ 5 <= 00003040  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003040  @00003010: $10 <= 00003040  @00003014: $16 <= 00000102  @00003018: $ 3 <= 0000007b  @00003028: $ 6 <= 00003040  @0000302c: $26 <= 12340000  @00003030: $17 <= 0000017d  @00003040: $24 <= 0000009c |
| Tuse\_rs=0，Tnew\_W=PC，转发 | |
| ori $5, $0, 0x3010  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  ori $10, $0, 0x1234  beq $31, $5, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003010  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @00003014: $10 <= 00001234  @00003024: $ 9 <= 12340000 |
| Tuse\_rs=1，Tnew\_E=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_E=DM，暂停 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_E=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $3, $31, $1 ##  addu $4, $1, $2  label:  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 3 <= 000030aa  @00003014: $ 5 <= 00000093 |
| Tuse\_rs=1，Tnew\_M=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  ori $3, $0, 289  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 3 <= 00000121  @0000300c: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_M=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_M=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  addu $3, $31, $1 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 3 <= 000030aa  @00003018: $ 5 <= 00000093 |
| Tuse\_rs=1，Tnew\_W=ALU，转发 | |
| ori $1, $0, 256  ori $2, $0, 255 ##  lui $5, 0x1256  ori $3, $0, 289  addu $4, $2, $1 ## | @00003000: $ 1 <= 00000100  @00003004: $ 2 <= 000000ff  @00003008: $ 5 <= 12560000  @0000300c: $ 3 <= 00000121  @00003010: $ 4 <= 000001ff |
| Tuse\_rs=1，Tnew\_W=DM，转发 | |
| ori $1, $0, 4  ori $2, $0, 0x1234  sw $2, 4($1)  lw $3, 4($1) ##  ori $25, $0, 156  lui $26, 0x1475  addu $4, $3, $1 ## | @00003000: $ 1 <= 00000004  @00003004: $ 2 <= 00001234  @00003008: \*00000008 <= 00001234  @0000300c: $ 3 <= 00001234  @00003010: $25 <= 0000009c  @00003014: $26 <= 14750000  @00003018: $ 4 <= 00001238 |
| Tuse\_rs=1，Tnew\_W=PC，转发 | |
| ori $1, $0, 154  ori $2, $0, 111  jal label ##  addu $4, $1, $2  lui $8, 0x1235  label:  lui $9, 0x58  addu $3, $31, $1 ##  ori $5, $0, 147 | @00003000: $ 1 <= 0000009a  @00003004: $ 2 <= 0000006f  @00003008: $31 <= 00003010  @0000300c: $ 4 <= 00000109  @00003014: $ 9 <= 00580000  @00003018: $ 3 <= 000030aa  @0000301c: $ 5 <= 00000093 |

|  |  |
| --- | --- |
| Tuse\_rt=0，Tnew\_E=ALU，暂停 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x302c | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003014: $10 <= 00000005  @0000301c: $12 <= 0000302c |
| Tuse\_rt=0，Tnew\_E=ALU，暂停 | |
| ori $1, $0, 256  ori $6, $0, 256  ori $5, $0, 4  sw $1, 4($5)  lw $10, 4($5) ##  beq $6, $10, label ##  nop  ori $20, $0, 2  label:  lui $4, 0x1234 | @00003000: $ 1 <= 00000100  @00003004: $ 6 <= 00000100  @00003008: $ 5 <= 00000004  @0000300c: \*00000008 <= 00000100  @00003010: $10 <= 00000100  @00003020: $ 4 <= 12340000 |
| Tuse\_rt=0，Tnew\_E=PC | |
| Tuse\_rt=0，Tnew\_M=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $30, $0, 56  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  ori $12, $0, 0x3034 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $30 <= 00000038  @00003018: $10 <= 00000005  @00003020: $12 <= 00003034 |
| Tuse\_rt=0，Tnew\_M=DM，暂停 | |
| ori $1, $0, 0x3038  ori $5, $0, 0x3038  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $3, $0, 123  beq $5, $10, label ##  nop  ori $25, $0, 256  label:  ori $27, $0, 145 | @00003000: $ 1 <= 00003038  @00003004: $ 5 <= 00003038  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003038  @00003010: $10 <= 00003038  @00003014: $ 3 <= 0000007b  @00003024: $27 <= 00000091 |
| Tuse\_rt=0，Tnew\_M=PC，转发 | |
| ori $5, $0, 0x3011  ori $1, $0, 256  jal label ##  nop  ori $2, $0, 255  label:  beq $5, $31, label2 ##  nop  ori $6, $0, 247  label2:  lui $9, 0x1234 | @00003000: $ 5 <= 00003011  @00003004: $ 1 <= 00000100  @00003008: $31 <= 00003010  @0000301c: $ 6 <= 000000f7  @00003020: $ 9 <= 12340000 |
| Tuse\_rt=0，Tnew\_W=ALU，转发 | |
| ori $2, $0, 0x3000  ori $3, $0, 0x2fff  ori $4, $0, 1  addu $5, $3, $4 ##  ori $27, $0, 256  ori $22, $0, 355  beq $2, $5, label ##  ori $10, $0, 5  ori $11, $0, 6  label:  addu $23, $2, $4 | @00003000: $ 2 <= 00003000  @00003004: $ 3 <= 00002fff  @00003008: $ 4 <= 00000001  @0000300c: $ 5 <= 00003000  @00003010: $27 <= 00000100  @00003014: $22 <= 00000163  @0000301c: $10 <= 00000005  @00003024: $23 <= 00003001 |
| Tuse\_rt=0，Tnew\_W=DM，转发 | |
| ori $1, $0, 0x3040  ori $5, $0, 0x3040  ori $2, $0, 4  sw $1, -4($2)  lw $10, -4($2) ##  ori $16, $0, 258  ori $3, $0, 123  beq $5, $10, label ##  nop  ori $25, $0, 256  label:  lw $6, -4($2) | @00003000: $ 1 <= 00003040  @00003004: $ 5 <= 00003040  @00003008: $ 2 <= 00000004  @0000300c: \*00000000 <= 00003040  @00003010: $10 <= 00003040  @00003014: $16 <= 00000102  @00003018: $ 3 <= 0000007b  @00003028: $ 6 <= 00003040 |

## 思考题

1. **在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面说明**

答：在数据通路方面：寄存器堆被放在了D级部分，因此寄存器值比较也放在了D级，所以将确定next\_pc的任务也放在D级，这样可以保证比较和跳转操作的连续性，此处注意beq指令判断为0的时候，next\_pc应该为pc + 4，jal等写如寄存器的指令的地址也为pc + 4；由于延迟槽的使用，在未产生跳转操作时，应保持每个周期pc + 4的操作，因此需要在F级每次给pc + 4，然后再输入回pc中。因此在pc处设置一个选择器，在本cpu中采用三目运算实现，用于确定是pc + 4，还是跳转的地址。并且由于暂停的需要，还需要从暂停控制器接到pc，使pc可以暂停。

在控制信号方面，在D级译码之后可以得到next\_pc操作控制信号，一般有三种b类型，j类型或者r类型。Pc中的使能信号为暂停信号(stall)取反得到，当使能信号为1时，pc正常工作；为0时，pc冻结。

1. **对于jal等需要将指令地址写入寄存器的指令，为什么需要回写PC+8？**

答：

由于使用了延迟槽，CPU会自动执行跳转指令后一个指令，该指令是在编译过程中编译器优化加入的，因此返回至跳转位置时应执行的是跳转指令后第二条指令，即pc+8，所以回写入寄存器的指令应是pc+8。

1. **为什么所有的供给者都是存储了上一级传来的各种数据的流水级寄存器，而不是由ALU或者DM等部件来提供数据？**

答：

因为可能产生死循环：若供给者为ALU，需求者也为ALU，此时ALU的输出端口与输入端口相接产生错误。而所有供给者都设置为流水级寄存器在我看来是为了功能上的协调统一，否则相同的效率下暂停与转发的控制将更加复杂难以操作。

1. **如果不采用已转发过的数据，而采用上一级中的原始数据，会出现什么样的问题？试列举指令序列说明这个问题。**

答：

若上一级的数据来自W级转发而之后并未使用转发过的数据，则可能会导致数据的错误。比如序列：

ori $1, $0, 0x1234

addu $1, $2, $3 （$2+$3=0x5678）

xxx

xxx

addu $5, $1, $2

当第二条指令位于W级时，第五条指令位于D级，此时原始数据RD1为0x1234，转发后数据V1为0x5678，下一个时钟周期，若使用原始数据，第五条指令到达E级，此时参与计算的$1为0x1234，但实际上第二条指令刚刚写入了寄存器，$1值实际为0x5678，因此产生错误。

1. **我们为什么要对GRF采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？**

若某一时钟周期，GRF需要写入寄存器，同时有新的指令需要在D级读取该寄存器的值，时钟上升沿到来时，两操作并行导致读取的寄存器值仍为原来的值，可能产生错误。

1. **为什么0号寄存器需要特殊处理？**

答：

因为0号寄存器无法写入，且任何时候读出均为0，不涉及数据冲突与冒险，若未进行特殊处理，当某一指令需要写入0寄存器，且后面指令需要读取0号寄存器时，根据普通转发机制，产生转发，将要写入的值转发，导致参与运算或比较的数据不是应读出的0，可能产生错误。

1. **什么是“最新产生的数据”？**

答：对某个寄存器而言，最新指令产生要写入该寄存器的数据就是最新产生的数据。

1. **在AT方法讨论转发条件的时候，只提到了“供给者需求者的A相同，且不为0”，但在CPU写入GRF的时候，是有一个we信号来控制是否要写入的。为何在AT方法中不需要特判we呢？为了用且仅用A和T完成转发，在翻译出A的时候，要结合we做什么操作呢？**

答：供给者既然已经称为了供给者，说明指令在它的位置上产生的值在之后一定会有写入寄存器的操作（若不被转发覆盖），因此不需要写使能的特判。为了仅用AT完成转发，我认为在翻译A时，若WE为1，则照常翻译A，若WE为0，则直接将A翻译成0，即写入0寄存器，在判断转发时也将直接被判定为不转发。

（四） 在线测试相关说明

1. **在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。此思考题请同学们结合自己测试CPU使用的具体手段，按照自己的实际情况进行回答。**

答：在测试中，我采用了大量的测试样例，基本上覆盖了所有的组合情况，未使用自动化测试工具。