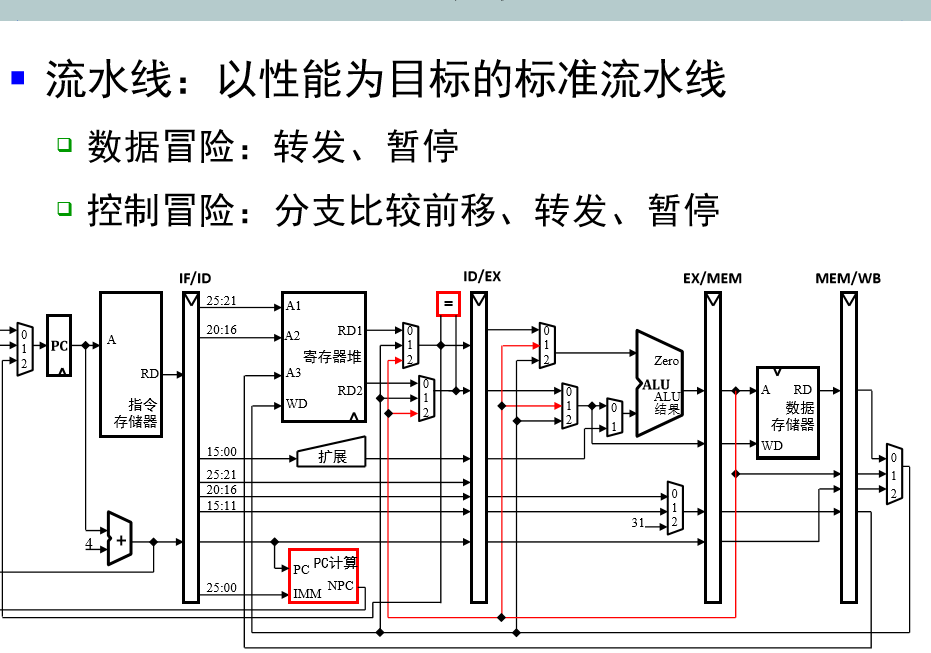
MIPS流水线处理器设计文档

1. 整体结构：

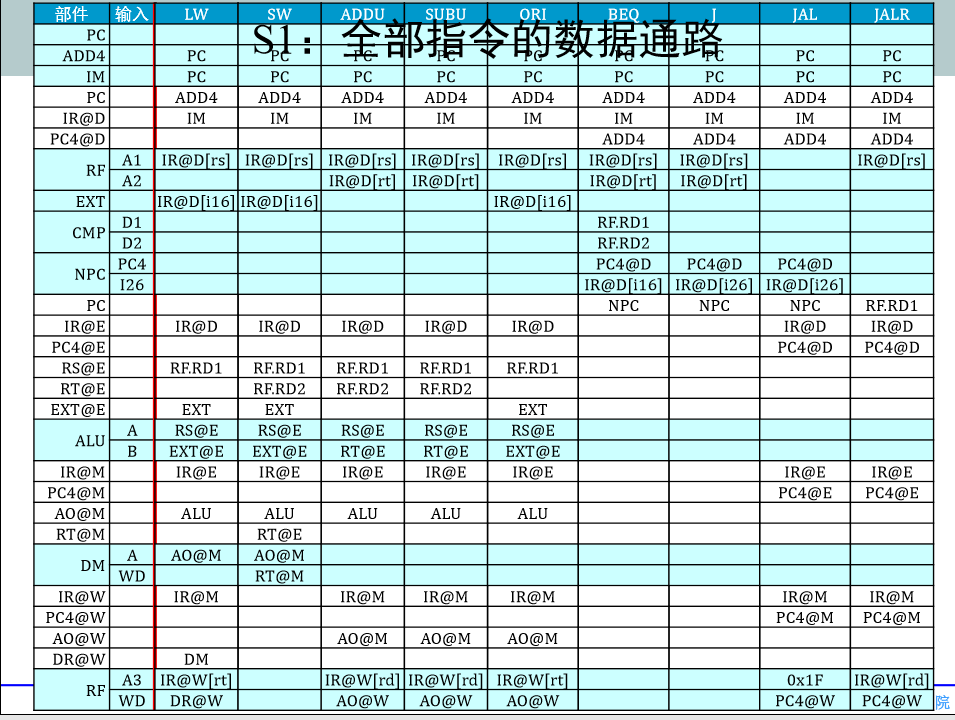
流水线处理器包括流水寄存器、各级组合逻辑以及各级控制器三大部分

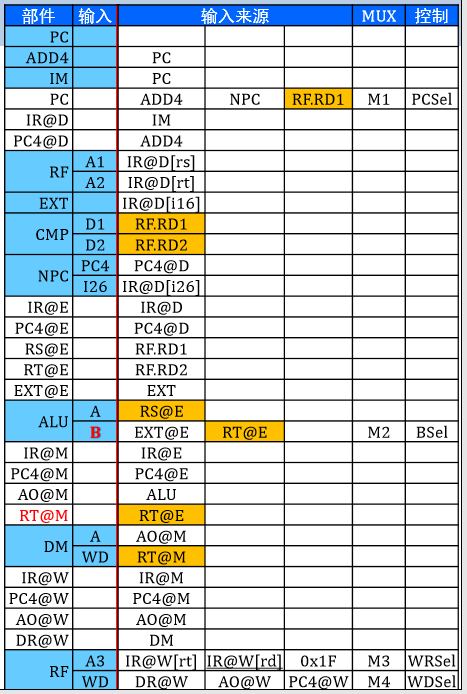
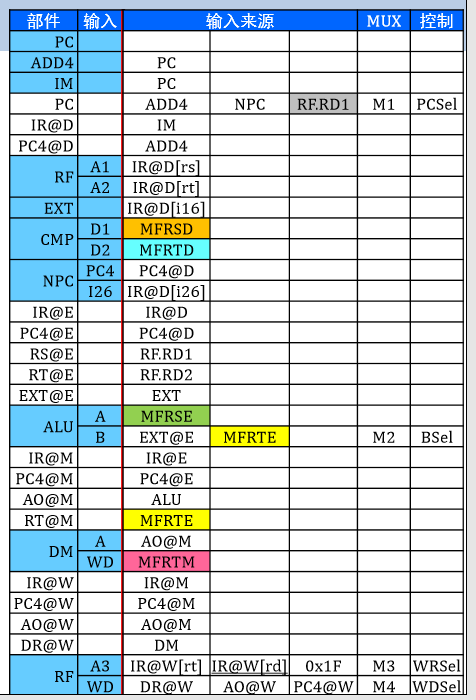
它们均放在mips.v层次下，其中code.txt中存储相应指令码

处理器为32位处理器，支持的指令集为：addu,subu, ori, lw, sw, beq, lui, j,jal, jr,nop



1. **数据通路**



命名规范：由流水级中流出的数据命名为\*\*\_\*,流水寄存器流出的值命名为\*\*\_m\*.

**（1）PC.V**

**模块定义：**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| **clk** | **I** | **时钟信号** |
| **reset** | **I** | **复位信号**  **0：无效**  **1：有效** |
| **stall** | **I** | **阻塞/暂停信号：**  **0：pc=npc**  **1：pc保持不变** |
| **npc[31:0]** | **I** | **输入的PC地址** |
| **pc** | **O** | **输出当前PC地址** |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能定义 |
| 1 | 复位 | 当时钟上升沿来临时，若复位信号有效，PC=0x00003000 |
| 2 | 取地址 | 时钟上升沿来临输出读取地址 |

（2）IM.V

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 当前PC地址 |
| Instr[31:0] | O | 当前读取的指令 |

（3）ADD4.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 当前pc地址 |
| pcplus4[31:0] | O | 输出数据为地址加4 |

（4）ADD8.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 当前pc地址 |
| Pcplus8[31:0] | O | 输出数据为地址加8 |

2.IF/ID级流水寄存器：

IF\_ID\_register.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  0:无效  1：有效 |
| en | I | 写使能信号  0：不可写流水寄存器  1：可写流水寄存器 |
| IR\_D\_in[31:0] | I | 传入该寄存器的指令 |
| PC4\_D\_in[31:0] | I | 传入该寄存器的PC+4 |
| PC8\_D\_in[31:0] | I | 传入该寄存器的PC+8 |
| IR\_D\_out[31:0] | O | 传出该寄存器的指令 |
| PC4\_D\_out[31:0] | O | 传出该寄存器的PC+4 |
| PC8\_D\_out[31:0] | O | 传出该寄存器的PC+8 |

功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能定义 |
| 1 | 复位 | 当时钟上升沿来临时，若复位信号有效，寄存器内容全为零 |
| 2 | 取地址 | 时钟上升沿来临时输出读取地址 |
| 3 | 取指令 | 时钟上升沿来临时取出当前指令 |

3.ID级组合逻辑：

（1）GRF.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，将32个寄存器中的值全部清零  1：有效  0：无效 |
| pc[31:0] | I | W级PC地址（PC4\_W-4） |
| RegWrite\_W | I | W级写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| Read\_register1[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到D1 |
| Read\_register2[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到D2 |
| Write\_register\_W | I | 5位地址输入信号，指定32个寄存器中的一个作为写入的目标寄存器 |
| Write\_data\_W[31:0] | I | 向写入寄存器写入的数据 |

功能定义：

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| 1 | 复位 | reset信号有效时，所有寄存器存储的数值清零 |
| 2 | 读数据 | 读出Read\_register1,Read\_register2地址对应寄存器中所存储的数据到RF.RD1,RF.RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将Write\_data\_W写入Write\_register\_W所对应的寄存器中 |

（2）EXT.v:

功能：选择立即数扩展方式

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm[15:0] | I | 输入数据 |
| Extop[1:0] | I | 选择信号：  00：无符号扩展  01：有符号扩展  10：加载至高位，低位补零 |
| after\_ext[31:0] | O | 符号扩展后输出数据 |

1. CMP.v

功能：比较器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| D1[31:0] | I | 第一个比较的数 |
| D2[31:0] | I | 第二个比较的数 |
| judge | O | 判断信号  1：D1=D2  0:D1!=D2 |

1. PC\_beq.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| after\_ext[31:0] | I | EXT扩展后的数 |
| PC4\_D[31:0] | I | PC+4的值 |
| equal | I | 相等信号 |
| pc\_beq | O | beq指令跳转地址 |

1. PC\_jal.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr[31:0] | I | 指令 |
| PC4\_D[31:0] | I | PC+4的值 |
| pc\_jal | O | jal指令跳转地址 |

1. MFRSD.v

功能：D级rs转发多选器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RF\_RD1[31:0] | I | rs寄存器里面内容 |
| ALUout\_M[31:0] | I | M级ALUout数据 |
| Write\_data\_W[31:0] | I | W级多选器的输出内容 |
| ForwardRSD[1:0] | I | 选择信号  00：RF\_RD1\_trans=RF\_RD1  01：RF\_RD1\_trans=ALUout\_M  10：RF\_RD1\_trans=Write\_data\_W |
| RF\_RD1\_trans[31:0] | O | 选择出来的数据 |

1. MFRTD.v

功能：D级rt转发多选器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RF\_RD2[31:0] | I | rs寄存器里面内容 |
| ALUout\_M[31:0] | I | M级ALUout数据 |
| Write\_data\_W[31:0] | I | W级多选器的输出内容 |
| ForwardRTD[1:0] | I | 选择信号  00：RF\_RD2\_trans=RF\_RD2  01：RF\_RD2\_trans=ALUout\_M  10：RF\_RD2\_trans=Write\_data\_W |
| RF\_RD2\_trans[31:0] | O | 选择出来的数据 |

1. nextpc\_2.v

功能：跳转pc的选择

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc\_jal[31:0] | I | jal跳转的地址 |
| pc\_beq[31:0] | I | beq跳转的地址 |
| RF\_RD1\_trans[31:0] | I | jr跳转的地址 |
| pc\_sel2[1:0] | I | 选择信号  00：nextpc=pc\_jal  01：nextpc=pc\_beq  10：nextpc=RF\_RD1\_trans |
| nextpc[31:0] | O | 选择出来的nextpc |

1. Wreg\_D.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr[20:16] | I | rt寄存器 |
| Instr[15:11] | I | rd寄存器 |
| 5'b11111 | I | 31号（$ra）寄存器 |
| RegDst[1:0] | I | 写寄存器选择信号  00：write\_register\_D=rt  01：write\_register\_D=rd  10：write\_register\_D=$ra |
| write\_register\_D[4:0] | O | 选择出来的写寄存器 |

4.ID/EX级流水寄存器：

ID\_EX\_register.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  0:无效  1：有效 |
| stall | I | 阻塞/暂停信号 |
| IR\_E\_in[31:0] | I | 传入该寄存器的指令 |
| PC4\_E\_in[31:0] | I | 传入该寄存器的PC+4 |
| PC8\_E\_in[31:0] | I | 传入该寄存器的PC+8 |
| RS\_E\_in[31:0] | I | 由rs寄存器传出，传入该寄存器的值 |
| RT\_E\_in[31:0] | I | 由rt寄存器传出，传入该寄存器的值 |
| EXT\_E\_in[31:0] | I | 传入该寄存器的立即数扩展之后的值 |
| write\_register\_E\_in[4:0] | I | 传入该寄存器的写寄存器 |
| IR\_E\_out[31:0] | O | 传出该寄存器的指令 |
| PC4\_E\_out[31:0] | O | 传出该寄存器的PC+4 |
| PC8\_E\_out[31:0] | O | 传出该寄存器的PC+8 |
| RS\_E\_out[31:0] | O | 由rs寄存器传出，传出该寄存器的值 |
| RT\_E\_out[31:0] | O | 由rt寄存器传出，传出该寄存器的值 |
| EXT\_E\_out[31:0] | O | 传出该寄存器的立即数扩展之后的值 |
| write\_register\_E\_out[4:0] | O | 传出该寄存器的写寄存器 |

5.EX级组合逻辑：

（1）ALU\_data\_B.v

功能：选择进入ALU的第二个数据值

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RF\_RD2\_trans[31:0] | I | 经过转发选择后的1寄存器的值 |
| after\_ext[31:0] | I | 立即数扩展之后的值 |
| ALUSrc | I | 选择信号 |
| ALUB | O | 输入ALU的第二个数据值 |

（2）ALU.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 输入A数据 |
| B[31:0] | I | 输入B数据 |
| ALUop[1:0] | I | 选择信号：  00：A1+A2  01：A1-A2  10：A1|A2 |
| Result[31:0] | O | 计算后输出数据 |

1. MFRSE.v

功能：E级转发多选器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RF\_RD1[31:0] | I | 第一个寄存器传出来的值 |
| ALUout\_M[31:0] | I | M级ALUout数据 |
| Write\_data\_W[31:0] | I | W级多选器的输出内容 |
| ForwardRSE[1:0] | I | 选择信号  00：RF\_RD1\_trans=RF\_RD1  01：RF\_RD1\_trans=ALUout\_M  10：RF\_RD1\_trans=Write\_data\_W |
| RF\_RD1\_trans[31:0] | O | rs转发多选器选出来的值 |

1. MFRTE.v

功能：E级转发多选器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RF\_RD2[31:0] | I | 第二个寄存器传出来的值 |
| ALUout\_M[31:0] | I | M级ALUout数据 |
| Write\_data\_W[31:0] | I | W级多选器的输出内容 |
| ForwardRTE[1:0] | I | 选择信号  00：RF\_RD2\_trans=RF\_RD2  01：RF\_RD2\_trans=ALUout\_M  10：RF\_RD2\_trans=Write\_data\_W |
| RF\_RD2\_trans[31:0] | O | rt转发多选器选出来的值 |

6.EX/MEM级流水寄存器：

EX\_MEM\_register.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  0:无效  1：有效 |
| IR\_M\_in[31:0] | I | 传入该寄存器的指令 |
| PC4\_M\_in[31:0] | I | 传入该寄存器的PC+4 |
| PC8\_M\_in[31:0] | I | 传入该寄存器的PC+8 |
| ALUout\_M\_in[31:0] | I | 由ALU传出，传入该寄存器的值 |
| RT\_M\_in[31:0] | I | 由rt寄存器传出，传入该寄存器的值 |
| RegWrite\_M\_in | I | 传入该寄存器的写信号 |
| write\_register\_M\_in[4:0] | I | 传入该寄存器的写寄存器 |
| IR\_M\_out[31:0] | O | 传出该寄存器的指令 |
| PC4\_M\_out[31:0] | O | 传出该寄存器的PC+4 |
| PC8\_M\_out[31:0] | O | 传出该寄存器的PC+8 |
| ALUout\_M\_out[31:0] | O | 由ALU传出，传出该寄存器的值 |
| RT\_M\_out[31:0] | O | 由rt寄存器传出，传出该寄存器的值 |
| RegWrite\_M\_out | O | 传出该寄存器的写信号 |
| write\_register\_M\_out[4:0] | O | 传出该寄存器的写寄存器 |

7.MEM级组合逻辑

（1）DM.v

功能：对内存进行读写操作

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  0：无效  1：有效 |
| pc[31:0] | I | pc现在地址 |
| addr[31:0] | I | 存数据的地址 |
| MemWrite | I | 写内存信号  0：不可写内存  1：可写内存 |
| MemData[31:0] | I | 存入的数据 |
| DMout[31:0] | O | 读出对应内存位置的数据 |

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| 1 | 读数据 | 读出pc地址对应内存中所存储的数据到DMout |
| 2 | 写数据 | 当MemWrite有效且时钟上升沿来临时，将MemData写入addr所对应的内存位置 |

功能定义：

1. MFRTM.v

功能：M级rt转发选择器

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| WD[31:0] | I | M级ALUout\_M的数据 |
| Write\_data\_W[31:0] | I | W级多选器的输出内容 |
| ForwardRTM | I | 选择信号  0：Write\_data\_trans=WD  1：Write\_data\_trans=Write\_data\_W |
| Write\_data\_trans[31:0] | O | 输出传至DMWD端口数据 |

8.MEM/WB级流水寄存器：

MEM\_WB\_register.v

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  0:无效  1：有效 |
| IR\_W\_in[31:0] | I | 传入该寄存器的指令 |
| PC4\_W\_in[31:0] | I | 传入该寄存器的PC+4 |
| PC8\_W\_in[31:0] | I | 传入该寄存器的PC+8 |
| ALUout\_W\_in[31:0] | I | 由ALU传出，传入该寄存器的值 |
| DM\_W\_in[31:0] | I | 由DM传出，传入该寄存器的值 |
| RegWrite\_W\_in | I | 传入该寄存器的写信号 |
| write\_register\_W\_in[4:0] | I | 传入该寄存器的写寄存器 |
| IR\_W\_out[31:0] | O | 传出该寄存器的指令 |
| PC4\_W\_out[31:0] | O | 传出该寄存器的PC+4 |
| PC8\_W\_out[31:0] | O | 传出该寄存器的PC+8 |
| ALUout\_W\_out[31:0] | O | 由ALU传出，传出该寄存器的值 |
| DM\_W\_out[31:0] | O | 由DM传出，传出该寄存器的值 |
| RegWrite\_W\_out | O | 传出该寄存器的写信号 |
| write\_register\_W\_out[4:0] | O | 传出该寄存器的写寄存器 |

9.WB级组合逻辑：

（1）DATAtoREG.v

功能：选择回写寄存器堆的数据来源

模块定义：

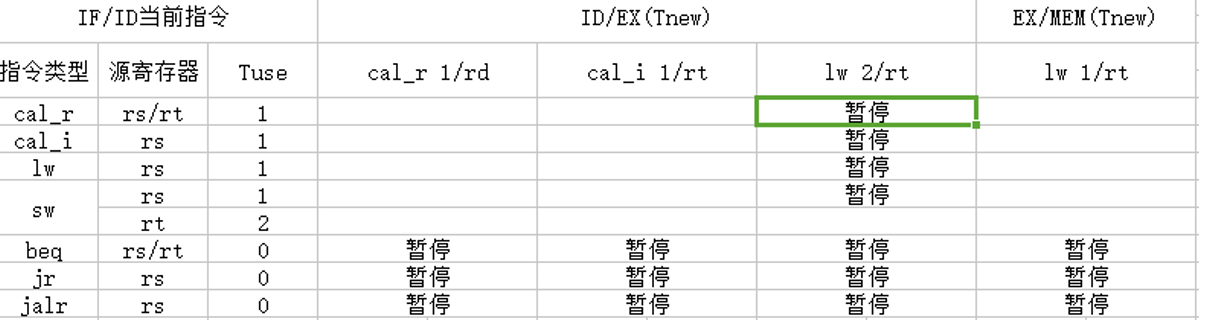
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALUout\_W[31:0] | I | 从ALU出来的数据 |
| DMout[31:0] | I | 从DM出来的数据 |
| MemtoReg | I | 选择信号：  0：WD=ALUout\_W  1：WD=DMout |
| WD[31:0] | O | 输出作为回写内容 |

控制器：

模块定义：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| func[5:0] | I | 6位func |
| op[5:0] | I | 6位op |
| RegDst[1:0] | O | 写寄存器选择信号 |
| pc\_sel1 | O | 是否跳转信号 |
| pc\_sel2[1:0] | O | 哪种跳转方式选择信号 |
| Cal\_r | O | Cal\_r类信号 |
| Cal\_i | O | Cal\_i类信号 |
| B | O | Beq类信号 |
| Load | O | Load类信号 |
| Save | O | Save类信号 |
| ALUSrc | O | 进入ALU的第二个值选择信号 |
| MemtoReg | O | 写回写寄存器的数据选择信号 |
| RegWrite | O | 写入寄存器信号 |
| MemWrite | O | 写入DM信号 |
| Extop[1:0] | O | 位扩展信号 |
| ALUop[1:0] | O | ALU功能选择信号 |
| J | O | J类信号 |

1. 转发和暂停：



转发采用标记转发法：

转发，本质上是一种有关时间的问题，即流水线机制所导致的在状态改变之前就要获取这个改变。比如比较经典的lw+R型这种冒险，其实就是lw改变GRF的状态，R型要获取这个改变，但时间上靠后执行的那条R型因为流水线的机制，想要先于改变来获取。对于控制冒险，同样可以理解为针对PC寄存器的获取（下一条指令的取指）早于对其的改变（NPC算出下一个PC并写回）。从工程化方法中对于转发位置与流水级的分类讨论跳出来，我们可以从另一个角度看待冒险冲突这件事情。

控制冒险在实验中通过延迟槽来解决，在规定了延迟槽指令不能非正常跳PC之后，这种冒险可以说是被大大简化了。因此流水线CPU有关冒险的全部难点，就全部集中于GRF状态改变之上。从GRF状态改变入手分析问题，问题将会变得更简单。

对于GRF的改变，考虑计组课程所要求的CPU架构，每一周期只能最多改变1个寄存器1次，对于任何一次改变，都能通过GRF的A3（地址）和WD（数据）完全确定。从转发来看，获取了A3和WD，即为获取了这个改变的全部信息，即为Tnew。换句话说，后续指令需要这个改变的时候，如果前方指令的A3与WD已经确定，就可以转发，否则必须阻塞。

所谓标记，就是在每条指令的每一级，如果可以确定A3和WD，就将其标记，并沿着流水线寄存器随着这条指令一起流下去。举个简单的例子：基础做法分布式控制将会在W级才解析R型指令的A3，并将前面传过来的Aluresult作为WD，标记法则为在E级末尾判断出Aluresult，同时解析A3.对于集中式译码，这一点已经做到，对于分布式译码，做到这一点也并不困难。考虑到集中式译码相对难写，笔者个人推荐分布式译码，32位指令本身随着流水线传递，随用随译。

标记之后，即可把每个需求位点（需求某个从GRF中读的值，比如M级的addr，比如E级的srcA，srcB等等）找出来，连入一个模块。将每个位点最新的转发值，同时传进去。也就是说，这个转发模块接受了所有需求位点本来读入的值，本来需求的地址，所有可能转发过来的值，所有可能转发过来的地址（这里的地址指的是寄存器的那5位地址）。当需求地址等于转发过来的地址时，将输出更新为转发过来的值，否则不更新直接连原来的需求位点本来读入的值。同学们应该可以分析出，这是一个本质mux，纯组合逻辑。在不需要阻塞的情况下，这一个简单的mux已经完成了所有的转发。需要判断的只是标记的地址和需求的地址是否相同。如何获取需求的地址呢，比如如何在E级获取srcasrcb当年对应的地址呢？这就有赖于笔者之前提到过的指令本身随流水传递了，随用随译。

1. 思考题

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。

1. 无数据冒险测试：

nop

nop

lui $a1,$0,0xffff

nop

nop

ori $a2,$a1,0xffff

nop

nop

addu $t1,$a1,$a1

nop

nop

subu $t2,$a2,$a1

nop

nop

sw $a1,8($0)

nop

nop

lw $t3,8($0)

nop

nop

beq $a1,$a1,branch

addu $s1,$0,$0

addu $s2,$0,$0

branch:

nop

nop

j jump1

addu $s1,$0,$0

addu $s2,$0,$0

jump1:

nop

nop

jal jump2

addu $s1,$0,$0

addu $s2,$0,$0

jump2:

nop

nop

jr $ra

addu $s1,$0,$0

addu $s2,$0,$0

1. 由于标记转发法中rs、rt转发方法完全相同且语法检查方便，故理论上不需要两寄存器的数据冒险测试重复测试。

R-M测试：

ori $t0,$0,1

ori $t1,$0,2

ori $t2,$0,7

addu $t3,$t1,$t0

subu $t4,$t2,$t3

R-W测试：

ori $t0,$0,1

ori $t1,$0,2

ori $t2,$0,7

addu $t3,$t1,$t0

nop

subu $t4,$t2,$t3

I-M测试：

ori $t1,$0,2

nop

ori $t2,$0,7

addu $t3,$t2,$t0

I-W测试

ori $t1,$0,2

nop

ori $t2,$0,7

nop

addu $t3,$t2,$t0

Load-M测试：

ori $t2,$0,7

sw $t2,8($0)

nop

nop

lw $t5,8($0)

addu $t6,$t5,$t5

Load-W测试

ori $t2,$0,7

sw $t2,8($0)

nop

nop

lw $t5,8($0)

nop

addu $t6,$t5,$t5

考虑跳转、暂停需要的综合测试：

ori $t0,$0,2

ori $t1,$0,4

beq $t0,$t1,jump1

addu $a0,$t0,$t1

subu $a0,$t1,$t0

jump1:

ori $t3,$0,3

ori $a1,$0,2

beq $a0,$a1,jump2

ori $t7,$0,8

ori $t7,$0,9

jump2:

ori $ra,$0,0x303c

jr $ra

ori $a0,$0,1

ori $a0,$0,2

ori $a0,$0,3

ori $a0,$0,4

ori $a0,$0,2026

sw $a0,0($0)

lw $a1,0($0)

lw $a2,0($0)

beq $a1,$a2,branch

ori $t0,$0,1

ori $t1,$0,2

branch:

ori $t1,$0,3

jal jump

lui $s1,1

lui $s2,2

jump:

lui $s3,3

j finish

finish:

jr $ra