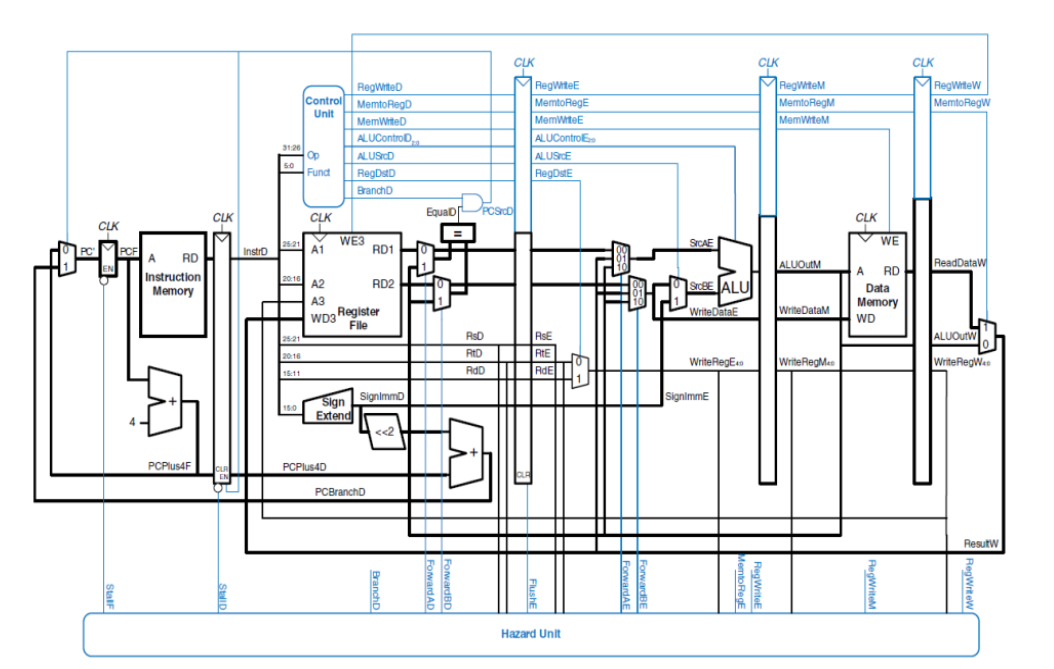
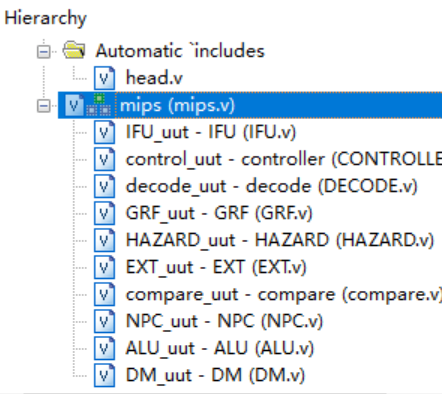
**P5 流水线CPU 实验报告**

**第一部分 CPU模型架构**

1. **流水线CPU顶层架构视图和模块架构视图**





1. **流水线CPU数据通路和控制器**
2. **各指令对应的数据通路表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | | addu | subu | ori |
| PC | | pc->IM | pc->IM | pc->IM |
| IM | | IM->INS | IM->INS | IM->INS |
| ADD8(4) | | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| D级 | INS | new | new | new |
| PC8 | new | new | new |
| decode | | rd,rs,rt | rd,rs,rt | rt,rs,imm16 |
| EXT | |  |  | ZE(imm16) |
| NPC | |  |  |  |
| GRF | | RF[rs]->RS\_OUT RF[rt]->RT\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT |
| compare | |  |  |  |
| E级 | WB | new(rd) | new(rd) | new(rt) |
| EXT\_OUT |  |  | new(EXT) |
| RS\_OUT | new(GRF) | new(GRF) | new(GRF) |
| RT\_OUT | new(GRF) | new(GRF) | new(GRF) |
| PC8 | old | old | old |
| ALU | | RS\_OUT+RT\_OUT unsigned | RS\_OUT-RT\_OUT unsigned | RS\_OUT|EXT\_OUT |
| M级 | WB | old | old | old |
| ALU\_result | new(ALU) | new(ALU) | new(ALU) |
| DIN |  |  |  |
| DM | |  |  |  |
| W级 | WrData | old(ALU\_result) | old(ALU\_result) | old(ALU\_result) |
| WB | old | old | old |
| GRF | | WrData->RF[WB] | WrData->RF[WB] | WrData->RF[WB] |

|  |  |  |  |
| --- | --- | --- | --- |
| lui | lw | sw | beq |
| pc->IM | pc->IM | pc->IM | pc->IM |
| IM->INS | IM->INS | IM->INS | IM->INS |
| PC+8->PC8 | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| new | new | new | new |
| new | new | new | new |
| rt,imm16 | rt,rs,imm16 | rt,rs,imm16 | rt,rs,imm16 |
| LUI(imm16) | SE(imm16) | SE(imm16) | SE(imm16) |
|  |  |  | B\_type PC8+SE |
| RF[rs]->RS\_OUT (32'b0,rs=00000) | RF[rs]->RS\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT |  |
|  |  |  | RF[rs]==RF[rt] |
| new(rt) | new(rt) |  |  |
| new(EXT) | new(EXT) | new(EXT) |  |
| new(GRF) | new(GRF) | new(GRF) |  |
|  |  | new(GRF) |  |
| old | old | old | new(npc) |
| RS\_OUT|EXT\_OUT | RS\_OUT+RT\_OUT unsigned | RS\_OUT+RT\_OUT unsigned |  |
| old | old |  |  |
| new(ALU) | new(ALU) | new(ALU) |  |
|  |  | old(RT\_OUT) |  |
|  | read:DM[ALU\_re] | DIN->DM[ALU\_re] |  |
| old(ALU\_result) | new(DM) |  |  |
| old | old |  |  |
| WrData->RF[WB] | WrData->RF[WB] |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| j | jal | jr | jalr |
| pc->IM | pc->IM | pc->IM | pc->IM |
| IM->INS | IM->INS | IM->INS | IM->INS |
| PC+8->PC8 | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| new | new | new | new |
| new | new | new | new |
| imm26 | imm26 | rs | rd,rs |
|  |  |  |  |
| J\_type PC[31:28]|imm26|00 | J\_type PC[31:28]|imm26|00 | JR\_type RF[rs] | JR\_type RF[rs] |
|  | PC8->RF[31] | RF[rs]->RS\_OUT | RF[rs]->RS\_OUT PC8->RF[rd] |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| new(npc) | new(npc) | new(npc) | new(npc) |

1. **CPU各模块规格**
2. **IFU**
3. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| clk | IN | 1 | 时钟信号 |
| Reset | IN | 1 | 同步复位信号 |
| if\_jump | IN | 1 | pc跳转执行信号 |
| stall\_pc | IN | 1 | pc暂停信号 |
| next\_pc | IN | 32 | pc跳转地址 |
| instr | OUT | 32 | 从IM中读取到的指令 |
| PC | OUT | 32 | PC当前值 |
| PC4 | OUT | 32 | PC+4值 |
| PC8 | OUT | 32 | PC+8值 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | 同步复位 | 复位信号为1时，在时钟上升沿复位pc |
| 2 | pc顺序执行 | pc执行pc+4 |
| 3 | pc跳转执行 | pc跳转至next\_pc地址 |
| 4 | pc暂停执行 | pc保持当前值不变，即暂停 |
| 5 | 读取指令 | 根据pc所指地址从IM读取响应指令并输出 |

1. **GRF**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| clk | IN | 1 | 时钟信号 |
| WrData | IN | 32 | 回写32位数据 |
| RWAddr | IN | 5 | 回写寄存器地址 |
| rs | IN | 5 | 读寄存器地址1 |
| rt | IN | 5 | 读寄存器地址2 |
| RegWr | IN | 1 | 寄存器堆写使能信号 |
| Reset | IN | 1 | 复位信号 |
| RS\_OUT | OUT | 32 | RS对应输出32位数据 |
| RT\_OUT | OUT | 32 | RT对应输出32位数据 |

（2）模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 同步复位 | 当Reset信号有效时同步复位所有寄存器 |
| 2 | 读寄存器 | 读出rs, rt对应寄存器内存储的数据 |
| 3 | 写寄存器 | 将WrData输入的数据写入RW对应的寄存器 |

1. **ALU**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| ALUOp | IN | 3 | ALU控制信号，控制进行何种运算 |
| inputA | IN | 32 | ALU的第一个操作数 |
| inputB | IN | 32 | ALU的第二个操作数 |
| shift | IN | 5 | 移位操作移位位数 |
| zero | OUT | 1 | beq指令操作数相等信号 |
| ALU\_result | OUT | 32 | ALU计算结果 |

（2）模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 加 | ALUOp=000，将两个操作数相加 |
| 2 | 减 | ALUOp=001，将两个操作数相减 |
| 3 | 或 | ALUOp=010，将两个操作数按位或 |
| 4 | 逻辑左移 | ALUOp=011，将操作数A左移shift位 |

1. **DM**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| clk | IN | 1 | 时钟信号 |
| DIN | IN | 32 | 写入32位数据 |
| DMAddr | IN | 32 | 写入32位地址 |
| DMWr | IN | 1 | DM写使能信号 |
| Reset | IN | 1 | 同步复位信号 |
| L\_S\_SL | IN | 3 | load/store类型选择信号 |
| DOUT | OUT | 32 | 读出32位数据 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 读内存数据 | DMWr信号无效时，读出DMAddr地址存储的数据 |
| 2 | 写内存数据 | DMWr信号有效时，向DMAddr地址写入DIN数据 |
| 3 | 同步复位 | 当Reset信号有效时，同步复位DM |

1. **EXT**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| EXT\_Op | IN | 2 | 扩展类型选择信号 |
| imm16 | IN | 16 | 要执行扩展操作的16位立即数 |
| EXT\_OUT | OUT | 32 | 扩展结果 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 符号扩展 | EXT\_Op信号为01时，对16位立即数进行符号扩展 |
| 2 | 无符号（零）扩展 | EXT\_Op信号为00时，对16位立即数进行零扩展 |
| 3 | 加载立即数到高位 | EXT\_Op信号为11时，将16位立即数加载到高位 |

1. **CMP**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| RS\_OUT | IN | 32 | RS寄存器值 |
| RT\_OUT | IN | 32 | RT寄存器值 |
| opcode | IN | 6 | 比较指令解码1 |
| rt | IN | 5 | 比较指令解码2 |
| judge | OUT | 1 | 判断结果信号 |

1. 功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | 相等时跳转 | Rs和RT相等时判断信号为1 |
| 2 | 不等时跳转 | Rs和RT不等时判断信号为1 |

1. **NPC**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| judge | IN | 1 | 比较判断信号 |
| NPC\_SL | IN | 3 | NPC跳转模式选择信号 |
| IMM16 | IN | 16 | 16位立即数 |
| IMM26 | IN | 16 | 26位立即数 |
| RS\_OUT | IN | 32 | RS寄存器值 |
| PC\_4 | IN | 32 | PC+4的值 |
| next\_pc | OUT | 32 | 跳转pc地址 |
| if\_jump | OUT | 1 | 跳转判断信号 |

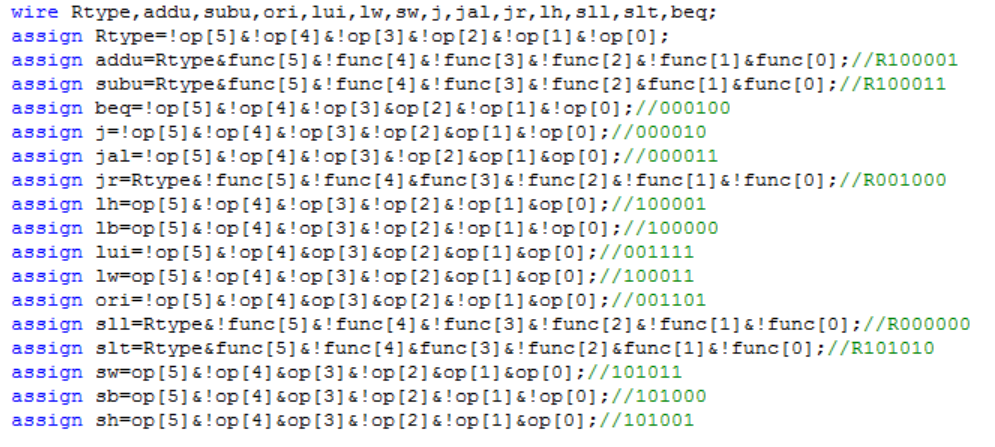
1. 功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | B型跳转 | 当比较判断信号为1，NPC\_SL为B型跳转时，相对跳转 |
| 2 | J型跳转 | NPC\_SL为J型时跳转到26位立即数所指地址 |
| 3 | JR跳转 | NPC\_SL为JR型时跳转到RS寄存器所指地址 |

1. **CPU控制器设计**
2. **端口定义和功能描述**

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| opcode | IN | 6 | 指令高六位[31:26]，用于判断指令类型 |
| func | IN | 6 | 指令低六位[5:0]，用于判断R指令 |
| RegDst | OUT | 2 | 写入寄存器选择信号,00选择rd，01选择rt，10选择ra |
| NPC\_SL | OUT | 2 | PC跳转选择信号，00顺序执行，01为B型指令，10为J型指令，11为JR指令 |
| RegData | OUT | 2 | 寄存器写入数据选择信号，00为ALU，01为DM，10为PC\_4 |
| RegWr | OUT | 1 | 寄存器写使能信号 |
| ExtOp | OUT | 2 | 扩展类型选择信号，01是符号扩展，00是零扩展，10是加载立即数到高位 |
| ALUsrc | OUT | 1 | ALU输入端2选择信号，为1选择EXT数据，为0选择rt寄存器数据 |
| DMWr | OUT | 1 | DM模式选择信号，为1向DM写入数据，为0从DM读出数据 |
| ALUOp | OUT | 3 | ALU功能选择信号，ADD:000，SUB:001，OR:010，SLL:011 |
| L\_S\_SL | OUT | 3 | Load/Store类型选择信号 |
| newdatatype | OUT | 2 | 新产生数据来源信号，目前有三种来源ALU,DM,PC |
| rt\_tuse | OUT | 2 | D级指令还有几个周期使用rt寄存器 |
| rs\_tuse | OUT | 2 | D级指令还有几个周期使用rs寄存器 |

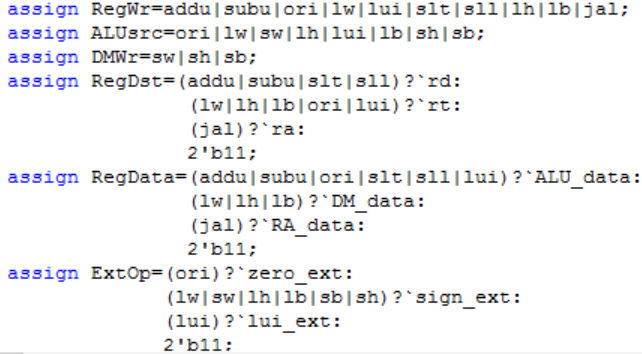
1. **控制器的指令解码（与逻辑AND）**

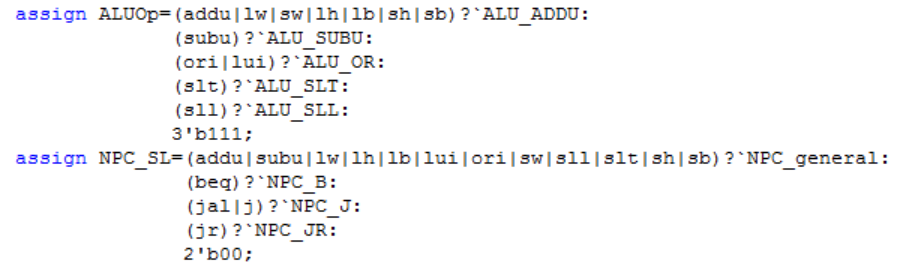


1. **控制器指令与控制信号真值表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | ori | lw | sw | beq | lui | jal | jr | j |
| RegDst | 00 | 00 | 01 | 01 | X | X | 01 | 10 | X | X |
| NPC\_SL | 00 | 00 | 00 | 00 | 00 | 01 | 00 | 10 | 11 | 10 |
| RegData | 00 | 00 | 00 | 10 | X | X | 00 | 10 | X | X |
| RegWr | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| ExtOp | X | X | 00 | 01 | 01 | X | 00 | X | X | X |
| ALUsrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | X | X | X |
| DMWr | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALUOp | 000 | 001 | 010 | 000 | 000 | 001 | 010 | X | X | X |

**(4) 控制器的控制信号生成（或逻辑OR）**





**(5) 控制器信号的宏定义**

1. **冒险单元（HAZARD）**
2. **Tuse-Tnew时间模型**
3. **Tuse**

我们定义某条指令位于流水线D级的时候，再经过Tuse个时钟周期就必须要使用相应的数据。而在D级只有rs和rt两个使用数据的寄存器地址，由此得到指令集的Tuse表如下：

|  |  |  |
| --- | --- | --- |
| 指令 | rs | rt |
| addu | 1 | 1 |
| subu | 1 | 1 |
| ori | 1 | \ |
| lui | \ | \ |
| lw | 1 | \ |
| sw | 1 | 2 |
| jal | \ | \ |
| jr | 0 | \ |
| j | \ | \ |
| jalr | \ | \ |
| beq | 0 | 0 |
| sll | \ | 1 |
| slti | 1 | \ |
| slt | 1 | 1 |
| srav | 1 | 1 |
| blez | 0 | \ |
| 合计 | 0,1 | 0,1,2 |

1. **Tnew**

对于那些能产生新数据的指令，经过分析发现他们产生数据的来源目前有三种：ALU,DM,PC。我们定义对于这些这令在某一流水级还需经过Tnew个时钟周期便可将新数据写入下级流水寄存器，由此得到Tnew表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | 功能部件 | E | M | W |
| addu | ALU | 1 | 0 | 0 |
| subu | ALU | 1 | 0 | 0 |
| ori | ALU | 1 | 0 | 0 |
| lui | ALU | 1 | 0 | 0 |
| sll | ALU | 1 | 0 | 0 |
| slav | ALU | 1 | 0 | 0 |
| lw | DM | 2 | 1 | 0 |
| jal | PC8 | 0 | 0 | 0 |
| jalr | PC8 | 0 | 0 | 0 |

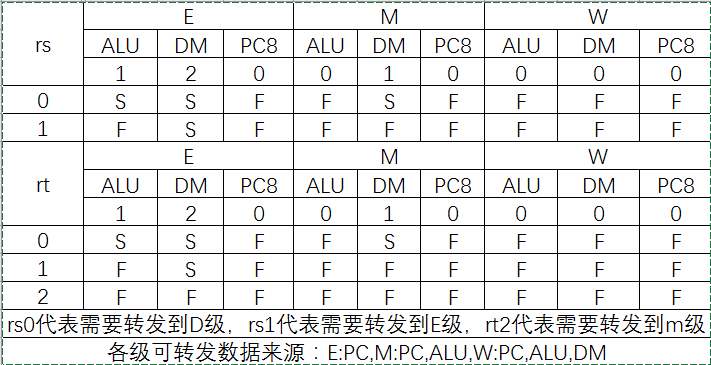
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | | | M | | | W | | |
| ALU | DM | PC8 | ALU | DM | PC8 | ALU | DM | PC8 |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

1. **暂停和转发策略**

得到Tuse和Tnew表之后，便可以比较两个时间确定暂停和转发策略。

1. 当Tnew<=Tuse时：后续指令执行需要的数据已经在数据通路中产生，可以全部采用显式转发策略进行解决。
2. 当Tnew>Tuse时：后续指令执行需要的数据未在数据通路中产生，需要暂停流水线cpu运行，等待满足Tnew<=Tuse时再通过相应的转发进行解决。

根据上述原则得到策略矩阵如下：



使用数据：

在D级使用rs寄存器的指令:jr,beq

在D级使用rt寄存器的指令:beq

在E级使用rs寄存器的指令:addu,subu,ori,lw,sw

在E级使用rt寄存器的指令:addu,subu

在M级使用rt寄存器的指令:sw

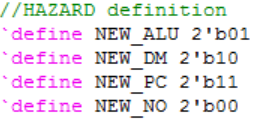
产生数据：

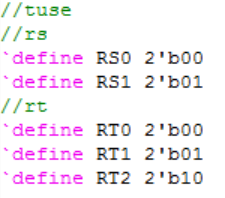
在E级流水线寄存器可转发的数据:PC

在M级流水线寄存器可转发的数据:PC,ALU

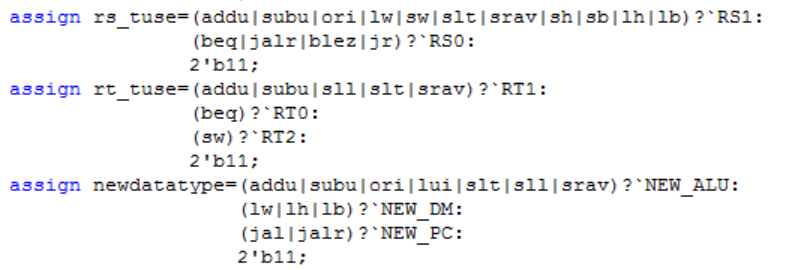
在E级流水线寄存器可转发的数据:PC,ALU,DM

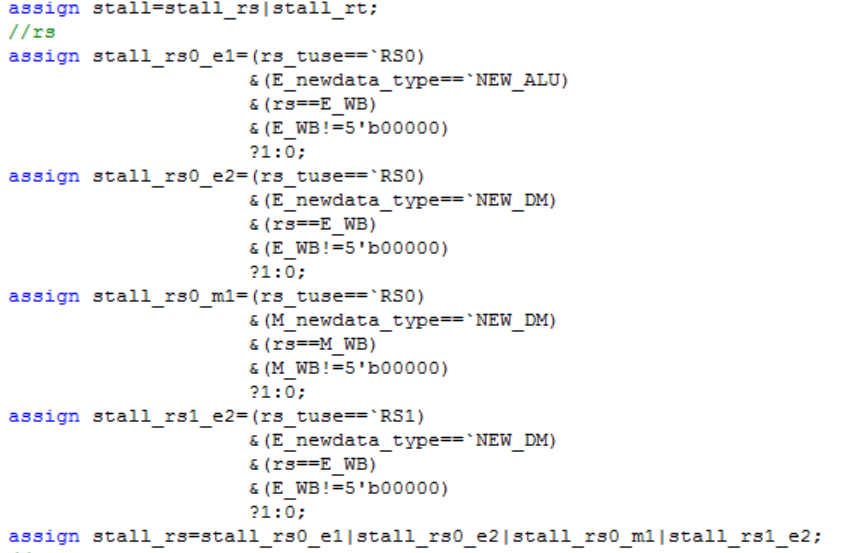
对应的Verilog宏定义如下：





对应的控制器实现如下：



1. **暂停功能具体实现**
2. 暂停策略
3. 使pc暂停，不让其值改变；
4. 使D级流水线暂停，不让其值改变；
5. 清零E级流水线，相当于插入nop指令
6. 暂停情况实现

rs和rt都是相同的四种暂停情况，故此处以rs为例。

根据策略矩阵中S情况对应的Tuse和Tnew，构造如图所示的暂停条件：

1. 满足此时对应的Tuse和Tnew条件
2. 当前指令使用数据的寄存器和前级产生新数据要写入的寄存器相同
3. 为了避免0号寄存器带来的误转发，还要保证产生数据的寄存器不是0号寄存器。

当暂停条件满足时，HAZARD单元使stall\_pc,stall\_D,reset\_E三个信号为1，执行相应操作完成暂停。

1. **转发功能具体实现**
2. 转发条件：

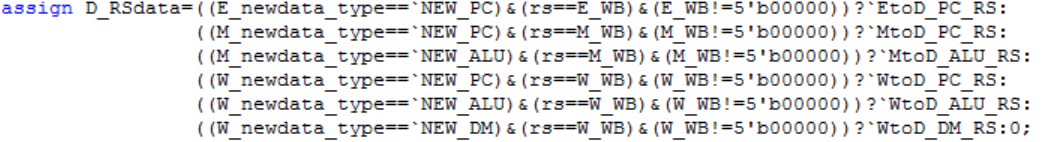
i.下一级寄存器确实产生了新数据；

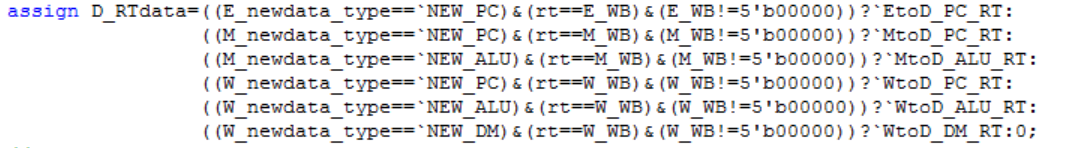
ii.当前指令使用数据的寄存器和前级产生新数据要写入的寄存器相同.

iii.为了避免0号寄存器带来的误转发，还要保证产生数据的寄存器不是0号寄存器。

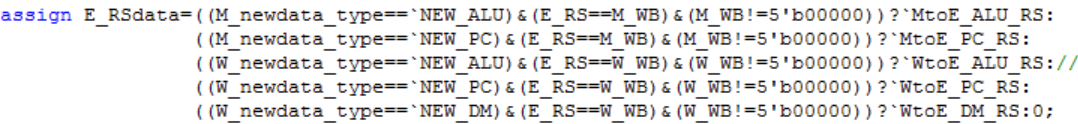
(2) 各级的转发情况

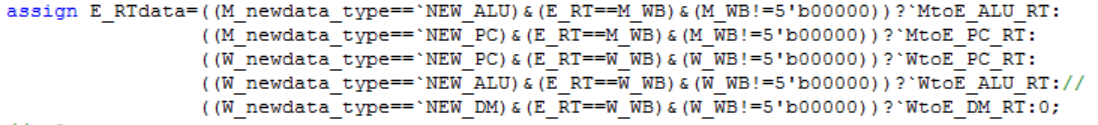
D级：rs和rt位置都需要转发，D级可以接受来自E,M,W三个级的转发数据，E级的转发来源PC,M级的转发来源PC,ALU，W级的转发来源PC,ALU,DM，故总共有可能的六个转发条件，Verilog实现如下：



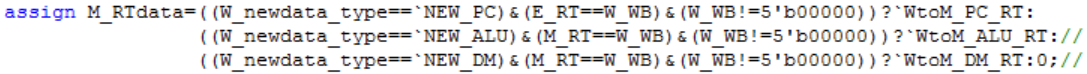


E级：rs和rt位置都需要转发，E级可以接受来自M,W两个级的转发数据，M级的转发来源PC,ALU，W级的转发来源PC,ALU,DM，故总共有可能的五个转发条件，Verilog实现如下：





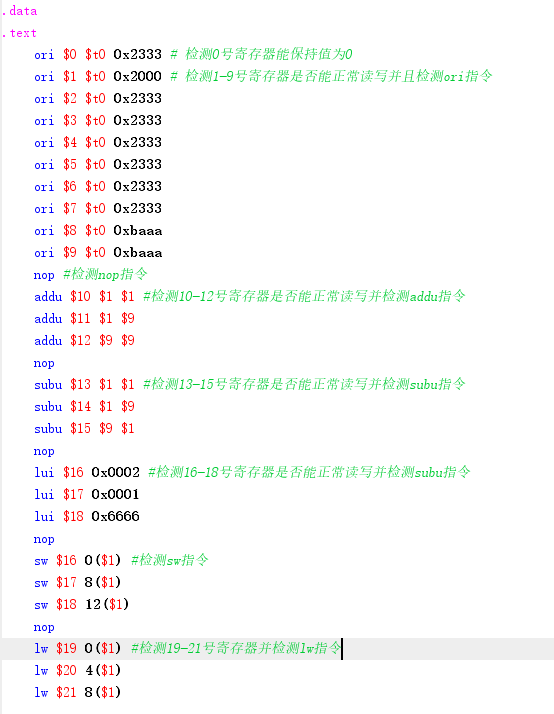
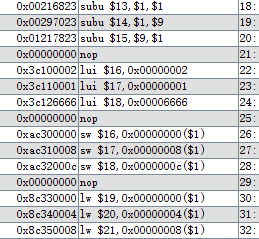
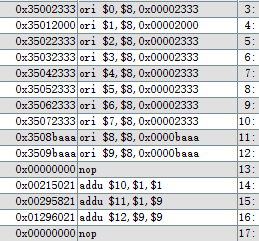
M级：目前指令集只有rt位置需要转发，M级可以接受来自W级的转发数据， W级的转发来源PC,ALU,DM，故总共有可能的三个转发条件，Verilog实现如下：



**第二部分 测试验证**

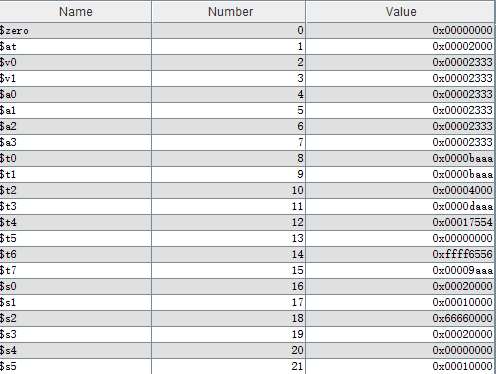
1. **基础指令和基础模块测试**
2. **测试目标**
   1. addu, subu, ori, lw, sw, beq, lui, nop, jal, jr十条指令都能做到完全正确执行。
   2. PC，IFU，GRF, ALU, DM五大数据通路模块的运行和读出写入操作都完全正确。
3. **测试程序及期望**

测试程序1：程序源代码，有关注释，机器代码：



Mars运行结果（期望结果）：





Verilog运行结果

监视器（$display）:

@00003004: $ 1 <= 00000000

@00003008: $ 2 <= 00002333

@0000300c: $ 3 <= 00002333

@00003010: $ 4 <= 00002333

@00003014: $ 5 <= 00002333

@00003018: $ 6 <= 00002333

@0000301c: $ 7 <= 00002333

@00003020: $ 8 <= 0000baaa

@00003024: $ 9 <= 0000baaa

@0000302c: $10 <= 00000000

@00003030: $11 <= 0000baaa

@00003034: $12 <= 00017554

@0000303c: $13 <= 00000000

@00003040: $14 <= ffff4556

@00003044: $15 <= 0000baaa

@0000304c: $16 <= 00020000

@00003050: $17 <= 00010000

@00003054: $18 <= 66660000

@0000305c: \*00000000 <= 00020000

@00003060: \*00000008 <= 00010000

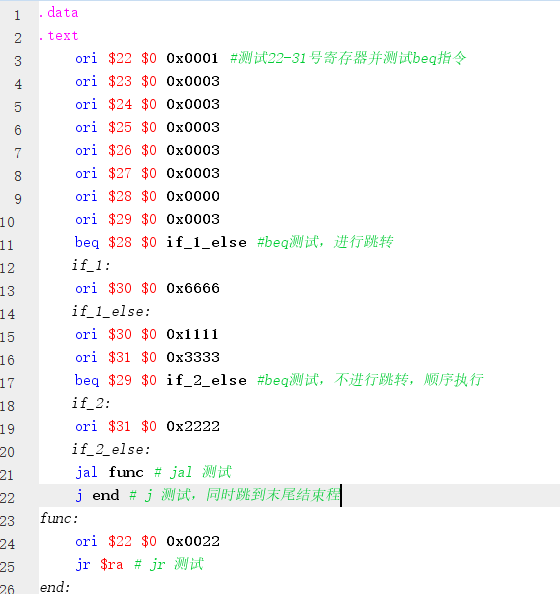
@00003064: \*0000000c <= 66660000

@0000306c: $19 <= 00020000

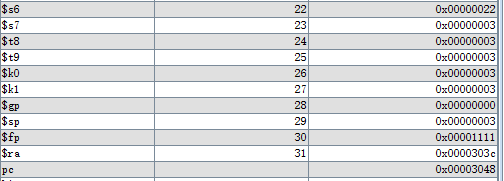
@00003070: $20 <= 00000000

@00003074: $21 <= 00010000

测试程序2：程序源代码，有关注释，机器代码：



Mars运行结果（期望结果）：



Verilog运行结果

监视器（$display）:

@00003000: $22 <= 00000001

@00003004: $23 <= 00000003

@00003008: $24 <= 00000003

@0000300c: $25 <= 00000003

@00003010: $26 <= 00000003

@00003014: $27 <= 00000003

@00003018: $28 <= 00000000

@0000301c: $29 <= 00000003

@00003028: $30 <= 00001111

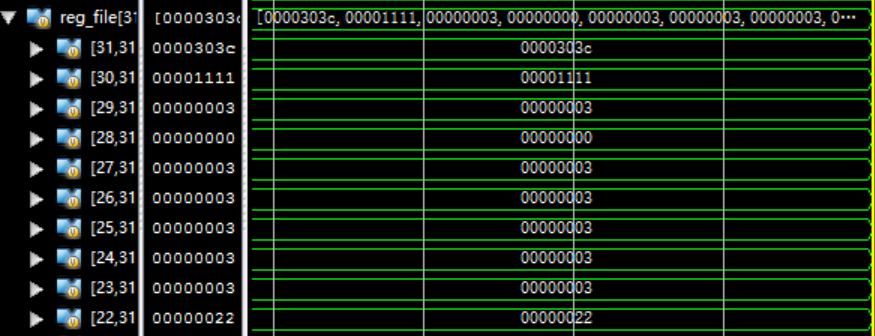
@0000302c: $31 <= 00003333

@00003034: $31 <= 00002222

@00003038: $31 <= 0000303c

@00003040: $22 <= 00000022

寄存器堆波形图结果：



1. **数据冒险测试**
2. **测试目标**

测试四种暂停情况和五大类转发情况，流水线CPU能够正确运行，并且尽可能多的采用转发策略解决数据冒险。

1. **测试程序及期望**

符号说明，Tuse=0,1,2,分别代表当前D级指令要在D,E,M级使用数据。Tnew，e1，与当前D级指令冲突指令在E级，它将在1个周期后把新数据写入E的下一级，即M级寄存器。

暂停情况：

1. Tuse=0,Tnew=e1

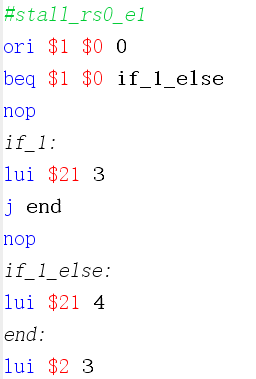
对于rs,此时使用rs的指令有beq,jr；

对于rt，此时使用rt的指令有beq。

满足e1的指令addu,subu,ori,lui

测试样例如下：

（其他指令本身行为没有问题则暂停也没有问题）



1. Tuse=0,Tnew=e2

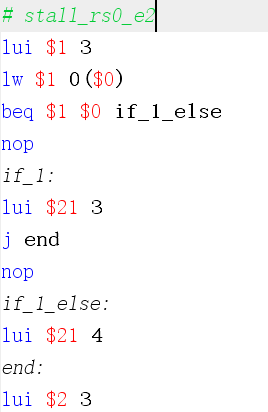
对于rs,此时使用rs的指令有beq,jr；

对于rt，此时使用rt的指令有beq。

满足e2的指令lw

测试样例如下：

（其他指令本身行为没有问题则暂停也没有问题）



1. Tuse=0,Tnew=m1

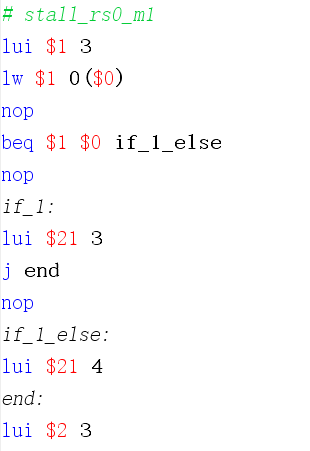
对于rs,此时使用rs的指令有beq,jr；

对于rt，此时使用rt的指令有beq。

满足m1的指令lw

测试样例如下：

（其他指令本身行为没有问题则暂停也没有问题）



1. Tuse=1,Tnew=e2

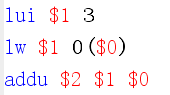
对于rs,此时使用rs的指令有addu,subu,ori,sw,lw；

对于rt，此时使用rt的指令有addu,subu。

满足e2的指令lw

测试样例如下：

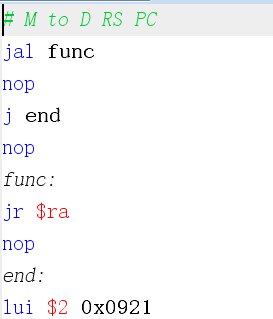
（其他指令本身行为没有问题则暂停也没有问题）



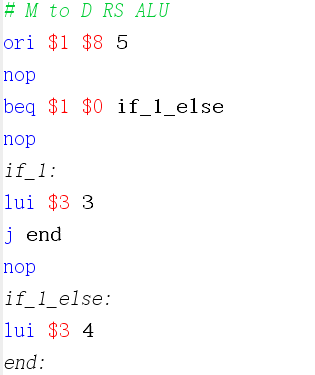
转发情况：

在指令本身行为正确的前提下，仅测试一类情况中某一样例便可验证该转发情况的正确性。

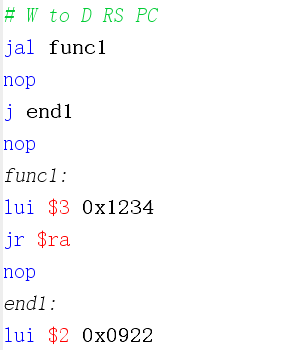
1. 向D级rs转发：
2. E级流水线寄存器转发。由于延迟槽的缘故，在E级产生新数据的jal指令后跟不会发生冲突的延迟槽指令，故这种情况在正常指令中不存在。
3. M级流水线寄存器转发，新数据来源为PC。



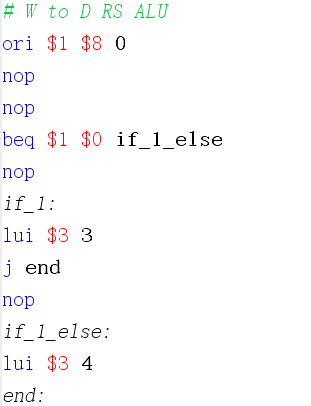
1. M级流水线寄存器转发，新数据来源为ALU



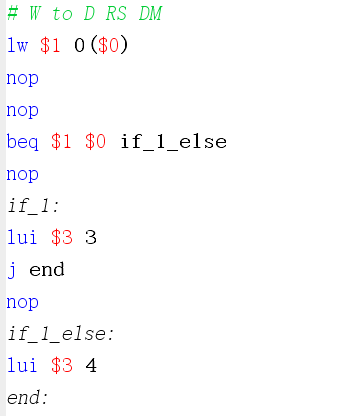
1. W级流水线寄存器转发，新数据来源为PC



1. W级流水线寄存器转发，新数据来源为ALU



1. W级流水线寄存器转发，新数据来源为DM



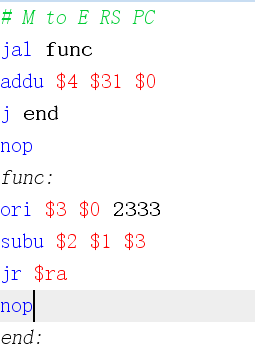
2) 向D级RT转发，类似RS情况。

3） 向E级RS转发

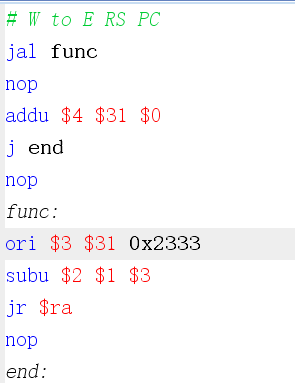
1. M级流水线寄存器转发，新数据来源为ALU



1. M级流水线寄存器转发，新数据来源为PC



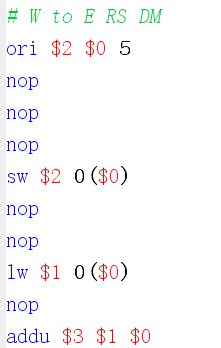
1. W级流水线寄存器转发，新数据来源为PC



1. W级流水线寄存器转发，新数据来源为ALU



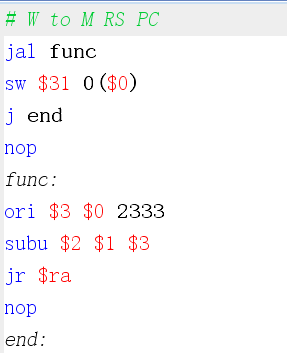
1. W级流水线寄存器转发，新数据来源为DM



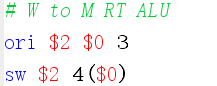
4） 向E级RT转发，类似RS情况

5） 向M级RT转发

1. W级流水线寄存器转发，新数据来源为PC



1. W级流水线寄存器转发，新数据来源为ALU



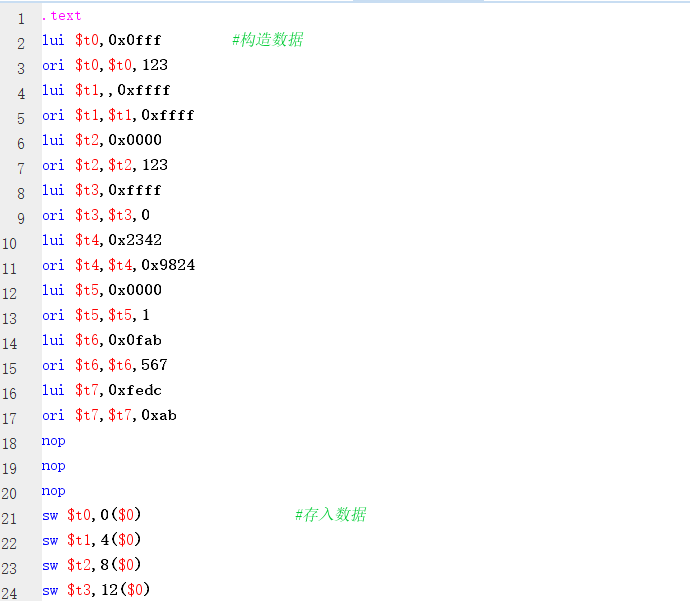
1. W级流水线寄存器转发，新数据来源为PC

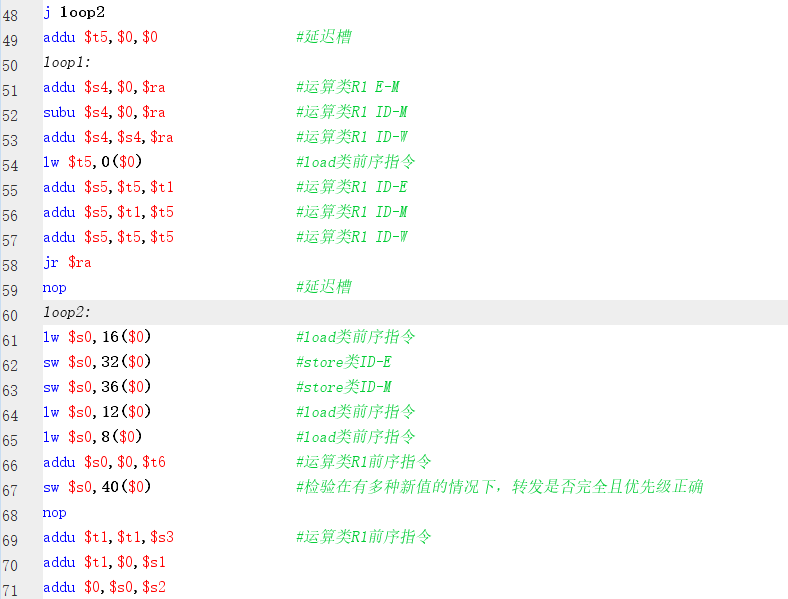
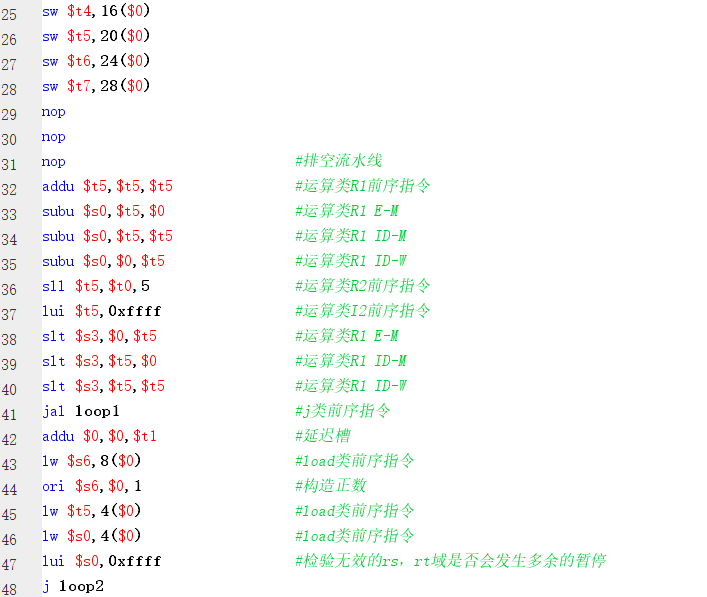


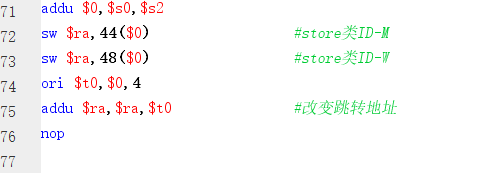
1. **综合程序测试**
2. **测试目的：**

综合测试指令集和数据冒险控制单元

1. **测试程序：**







**第三部分 思考题**

1. 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(**非常重要**)

答：见上文数据冒险测试部分。