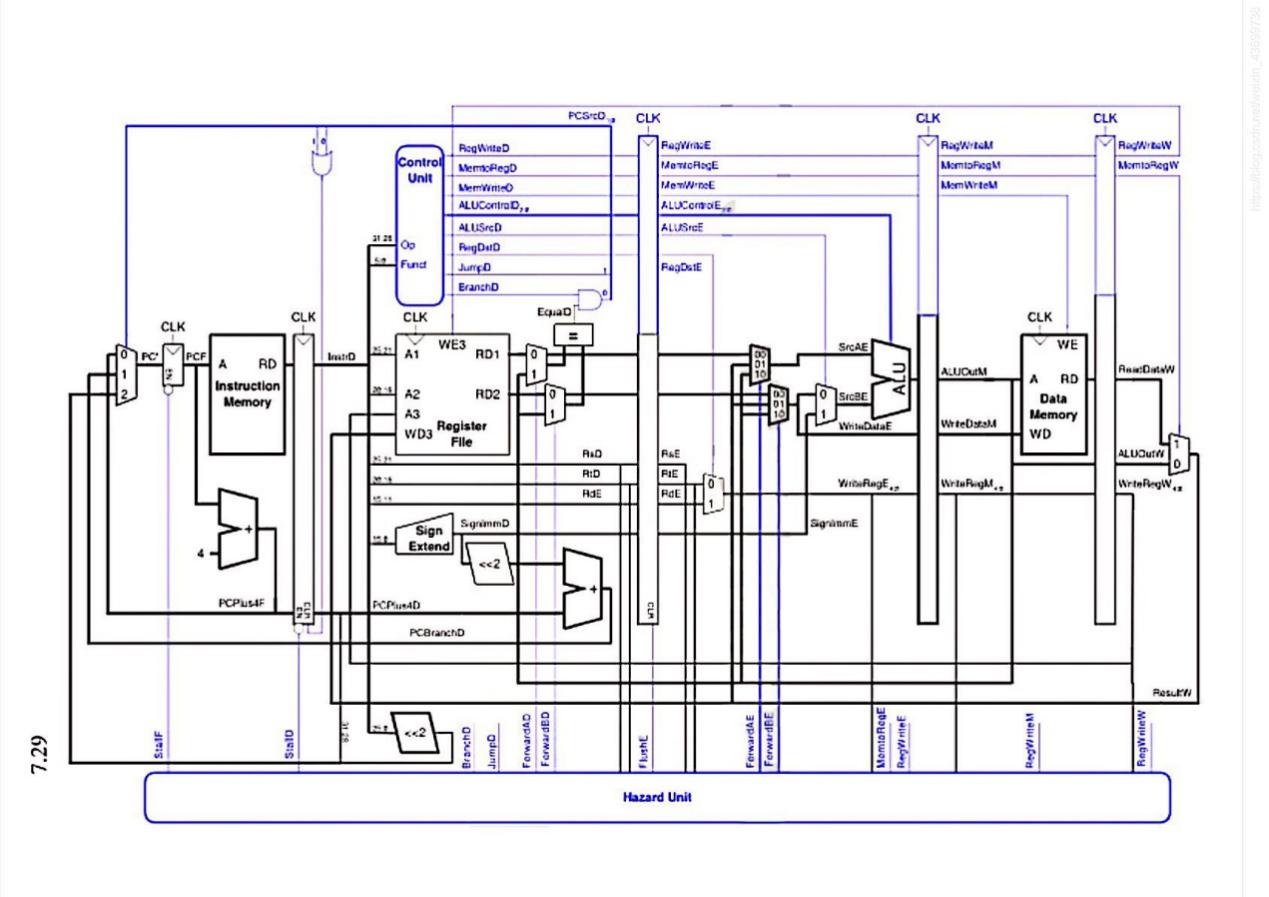
P5流水线设计方案

## **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、……}。为了实现这些功能，CPU主要包含了IM、GRF、……，这些模块按照….的顶层设计逐级展开。

本次设计的参考图：



实际上在设计过程中修改了很多部分，因此此图只作为参考。

**（二）设计单元描述**

一般来说我们认为有五个单元，IF、ID、EXT、MEM、WB。

1. IF：

（1）MUX\_PC

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 输入/输出 | 位数 | 功能 |
| PCplus | I | 32 | PC+4 |
| PCextern | I | 32 | 位扩展（beq）得到PC |
| PCjump | I | 32 | j/jal指令得到PC |
| PCtemp | I | 32 | Jr指令得到PC |
| Sel | I | 2 | PC选择端：  sel ==0：PCplus  sel == 1：PCextern  sel == 2：PCjump:  sel == 3：PCtemp; |
| Out | O | 32 | 输出的PC值 |

B：功能定义

根据Sel值确定输出的PC值，色狼选取规则见上方。

（2）PC

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| PC\_in | I | 32 | 输入的PC值 |
| Clk | I | 1 | 时钟 |
| Clr | I | 1 | 复位 |
| En | I | 1 | 使能信号 |
| PC\_out | O | 32 | 输出PC值 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 说明 |
| 1 | 复位PC | 时钟上升沿时，如果clr =1，更新PC值为0x0000\_3000 |
| 2 | 冻结PC | 时钟上升沿时，如果en=1，冻结PC值不变化。 |
| 3 | 更新PC | 时钟上升沿到来时，如果两个条件均不满足，则更新输入的PC值并输出。 |

（3）IM

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| PC | I | 32 | 输入当前PC值 |
| Instr | O | 32 | 输出指令 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 说明 |
| 1 | 读取指令 | 初始化时利用$readmemh |
| 2 | 输出指令 | 选取PC[11:2]位进行定位指令和输出 |

## 2、ID

（1）GRF

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Data | I | 32 | 将要存入的数据 |
| Rs | I | 5 | Rs寄存器 |
| Rt | I | 5 | Rt寄存器 |
| Rd | I | 5 | Rd寄存器 |
| PC | I | 32 | PC值 |
| Clr | I | 1 | 复位端 |
| Clk | I | 1 | 时钟端 |
| RegW | I | 1 | 写入使能 |
| Data\_in\_rs | O | 32 | Rs寄存器输出值 |
| Data\_in\_rt | O | 32 | Rt寄存器输出值 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 说明 |
| 1 | 复位寄存器 | 时钟上升沿到来时，如果clr=1，则清空所有寄存器 |
| 2 | 存入寄存器 | 时钟上升沿到来时，如果clr=0且regw=1，rd！=0，则写入寄存器 |
| 3 | 输出寄存器 | 时钟下降沿到来时，输出当前寄存器内部值 |

（2）EXT位扩展

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Im | I | 16 | 输入的立即数 |
| Zero\_extern | I | 1 | 零扩展 |
| Sign\_extern | I | 1 | 符号扩展 |
| extern | O | 32 | 扩展结果 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 说明 |
| 1 | 零扩展 | 当zero\_extern = 1,extern为im零扩展结果 |
| 2 | 符号扩展 | 当sign\_extern = 2,extern为im符号扩展结果 |

（3）CMP

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| RD1 | I | 32 | 读入值1 |
| RD2 | I | 32 | 读入值2 |
| Equal | O | 1 | 输出结果  RD1=RD2：equal=1  RD1!=RD2:equal=0 |

B：功能定义

用作beq的比较，将其从E级迁移到D级。

（4）CONTROL

由于CONTROL 中很多信号需要流水，因此在流水阶段再进行解释。

## **3、EX级**

（1）ALU

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| ALUop | I | 3 | 计算机制 |
| Data\_in\_rs | I | 32 | 输入值1 |
| Data\_in\_rt | I | 32 | 输入值2 |
| result | O | 32 | 输出结果 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | ALUop取值 | 计算方式 |
| 1 | 0 | 与 |
| 2 | 1 | 或 |
| 3 | 2 | 加法 |
| 4 | 3 | 减法 |
| 5 | 6 | 加载到高位 |

4、MEM级

（1）DM

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Memw | I | 1 | 写入使能 |
| Clk | I | 1 | 时钟端 |
| Memr | I | 1 | 读取使能 |
| Clr | I | 1 | 复位端 |
| Add | I | 32 | 地址 |
| Wdata | I | 32 | 写入数据 |
| PC | I | 32 | PC值 |
| Rdata | O | 32 | 读出数据 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 时钟上升沿到来时，如果clr=1，则清空mem |
| 2 | 存入存储器值 | 当时钟上升沿到来时，如果memw=1，则通过add和wdata写入。 |
| 3 | 取出存储器值 | 当memr = 1，通过add读出数据至rdata |

5、WB级

6、IF\_ID级

（1）R\_IF\_ID级

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Instr\_in | I | 32 | 输入的指令 |
| PCplus\_in | I | 32 | 输入的PC值 |
| Clk | I | 1 | 时钟端 |
| Clr | I | 1 | 复位端 |
| Instr\_out | O | 32 | 输出的指令 |
| PCplus\_out | O | 32 | 输出的PC |
| FlushD | I | 1 | 是否冻结 |

B：功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 当时钟上升沿到来时清空 |
| 2 | 冻结 | 当时钟上升沿到来时，如果flushD=1，冻结寄存器的值不变 |
| 3 | 流水 | 当时钟上升沿到来而且不满足上面情况时，将其流水 |

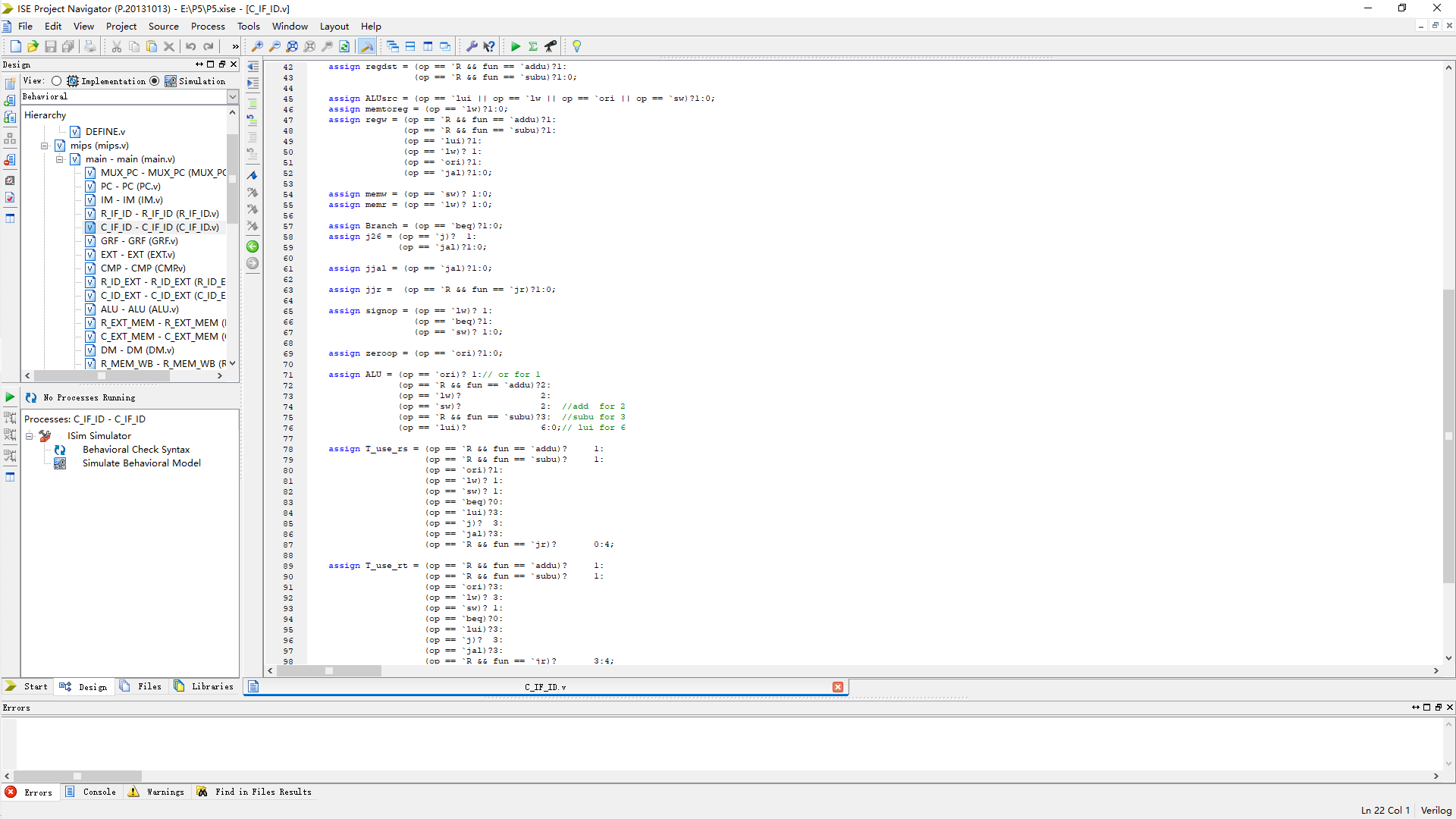
（2）C\_IF\_ID级

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Op | I | 6 | Operation |
| Fun | I | 6 | Function |
| Regdst | O | 1 | 选取写入的寄存器 |
| ALUsrc | O | 1 | 选取输入的ALU端口2 |
| Memtoreg | O | 1 | 从mem写入GRF使能 |
| Regw | O | 1 | GRF写入使能 |
| Memw | O | 1 | MEM写入使能 |
| Memr | O | 1 | Mem读出使能 |
| Branch | O | 1 | Beq |
| Jjal | O | 1 | Jal |
| Jjr | O | 1 | Jr |
| J26 | O | 1 | 无条件跳转信号 |
| Signop | O | 1 | 符号扩展 |
| Zeroop | O | 1 | 零扩展 |
| ALUop | O | 3 | 计算方式 |
| T\_use\_rs | O | 3 | 使用rs寄存器最短时间 |
| T\_use\_rt | O | 3 | 使用rt寄存器最短时间 |

B：功能定义

判断指令并按照需求输出



7、ID\_EXT

（1）R\_ID\_EXT

A：端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| RD1D | I | 32 | 流水 |
| RD2D | I | 32 |
| RsD | I | 5 |
| RtD | I | 5 |
| RdD | I | 5 |
| Clk | I | 1 |
| Clr | I | 1 |
| FlushE | I | 1 |
| PCD | I | 32 |
| RD1E | O | 32 |
| RD2E | O | 32 |
| RsE | O | 5 |
| RtE | O | 5 |
| RdE | O | 5 |
| ExternE | O | 32 |
| PCE | O | 32 |
|  |  |  |  |

B、功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | 当时钟上升沿到来时清空 |
| 2 | 冻结清零 | 当时钟上升沿到来时，如果flushE=1，清零寄存器的值 |
| 3 | 流水 | 当时钟上升沿到来而且不满足上面情况时，将其流水 |

（2）C\_ID\_EXT

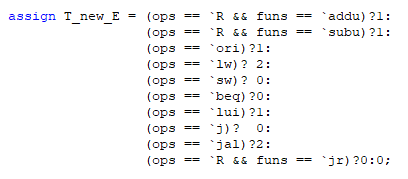
A：新增输出

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 位数 | 功能 |
| Op | I | 6 | Operation |
| Fun | I | 6 | Function |
| T\_new\_E | O | 3 | 作为跳转信号的判断 |

B：流水单元

|  |  |
| --- | --- |
| 名称 | 流水方向 |
| Regw | 由D到E |
| Memtoreg |
| Memw |
| Memr |
| ALUop |
| ALUsrc |
| Jjal |
| Regdst |

C：功能定义



8、EX\_MEM

（1）R\_EX\_MEM

A：流水单位

|  |  |
| --- | --- |
| 名称 | 流水对应 |
| AddE | AddM |
| WdataE | WdataM |
| Rd\_rsE | Rd\_rsM |
| PCE | PCM |

（2）C\_EX\_MEM

A：流水单位

|  |  |
| --- | --- |
| 名称 | 流水方向 |
| Regw | 由D到E |
| Memtoreg |
| Memw |
| Memr |
| Jjal |
| T\_new |

9、MEM\_WB

（1）R\_MEM\_WB

A:流水单位

|  |  |
| --- | --- |
| 名称 | 流水对应 |
| AddM | AddW |
| rdataM | rdataW |
| Rd\_rsM | Rd\_rsW |
| PCM | PCW |

（2）C\_MEM\_WB

A：流水单位

|  |  |
| --- | --- |
| 名称 | 流水方向 |
| Regw | 由D到E |
| Memtoreg |
| Jjal |
| T\_new |

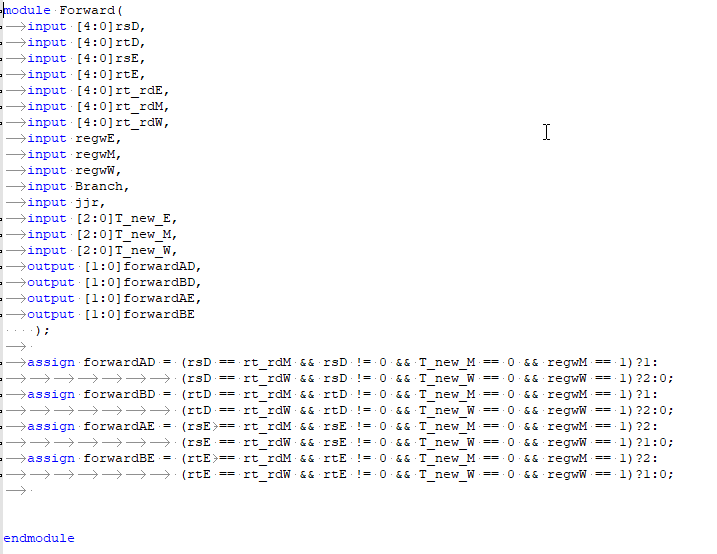
**（三）关键机制说明**

**1、转发机制**

（1）forward单元

根据T\_new\_E、T\_new\_M、T\_new\_W和T\_use进行比较即可，现在这个地方附上代码。

另外在处理过程中由于具有某些指令（如sw）不写入寄存器但是会用到寄存器的值，因此判断时需要加上regw。



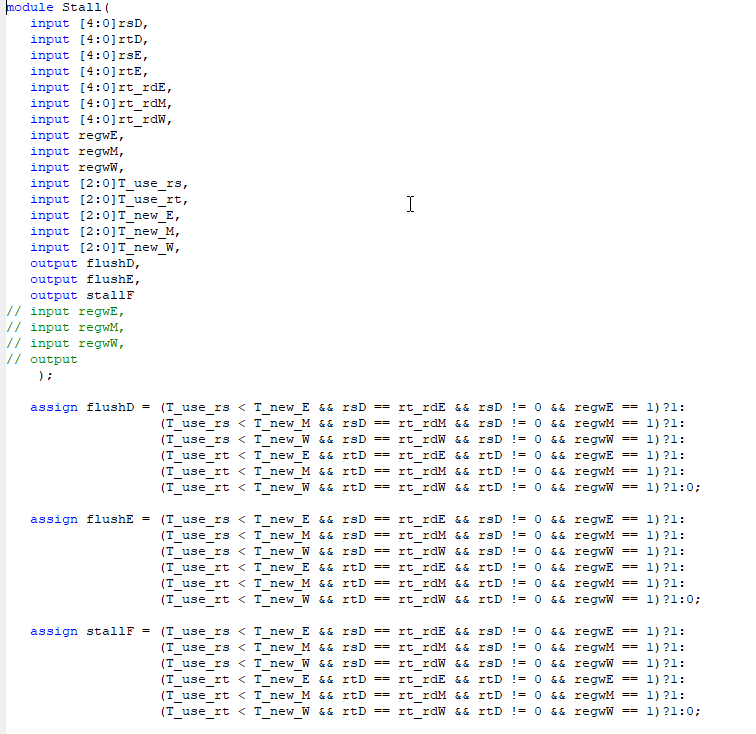
转发的地方有两个：

1、D级转发，主要运用于跳转指令的转发。

2、E级转发，大部分转发都是在E级ALU单元前进行的，通过M级和W级得到需要的结果。

**2、暂停机制**

（1）stall单元



判断条件和forward判断条件类似，注意优先级。

暂停后需要注意有三个地方处理：

（1）冻结PC，PC值不变

（2）冻结ID，ID值不变

（3）清空IE，插入nop。

## **二、测试方案**

### **（一）典型测试样例**

#### **1、基础功能测试**

该部分过程只是简单测试多周期的正确性，并没有考虑转发。

ori $1,12

ori $2,12

lui $3,4

ori $5,40

addu $4,$1,$2

addu $2,$1,$1

lw $1,0($1)

jal label

addu $2,$1,$1

label: sw $1,0($4)

addu $3,$3,$3

addu $31,$31,$3

j label2

subu $3,$3,$3

label2: addu $31,$31,$5

subu $5,$5,$5

beq $5,$0,label3

addu $2,$2,$2

label3:ori $3,10

nop

jr $31

期待输出：（未考虑延迟槽）

@00003000: $ 1 <= 0000000c

@00003004: $ 2 <= 0000000c

@00003008: $ 3 <= 00040000

@0000300c: $ 5 <= 00000028

@00003010: $ 4 <= 00000018

@00003014: $ 2 <= 00000018

@00003018: $ 1 <= 00000000

@0000301c: $31 <= 00003020

@00003024: \*00000018 <= 00000000

@00003028: $ 3 <= 00080000

@0000302c: $31 <= 00083020

@00003038: $31 <= 00083048

@0000303c: $ 5 <= 00000000

@00003048: $ 3 <= 0008000a

实际输出：（考虑延迟槽）

18@00003000: $ 1 <= 0000000c

22@00003004: $ 2 <= 0000000c

26@00003008: $ 3 <= 00040000

30@0000300c: $ 5 <= 00000028

34@00003010: $ 4 <= 00000018

38@00003014: $ 2 <= 00000018

42@00003018: $ 1 <= 00000000

46@0000301c: $31 <= 00003024

50@00003020: $ 2 <= 00000000

50@00003024: \*00000018 <= 00000000

58@00003028: $ 3 <= 00080000

62@0000302c: $31 <= 00083024

70@00003034: $ 3 <= 00000000

74@00003038: $31 <= 0008304c

78@0000303c: $ 5 <= 00000000

90@00003044: $ 2 <= 00000000

94@00003048: $ 3 <= 0000000a

#### **2、转发测试**

ori $28, $0, 0

ori $29, $0, 0

ori $23, $0, 0

sw $30, 8($23)

beq $23, $23, label1

ori $23, $0, 0

lw $23, 8($23)

ori $23, $0, 0

sw $23, 8($23)

label1: ori $23, $30, 60592

beq $23, $23, label2

lui $23, 18146

addu $30, $29, $23

label2: ori $30, $0, 0

sw $23, 0($30)

addu $23, $30, $30

lui $23, 52368

beq $23, $23, label3

lui $23, 1409

ori $23, $29, 38829

label3: subu $30, $23, $23

j label4

ori $23, $0, 0

sw $23, 8($23)

lui $29, 56288

label4: ori $29, $0, 0

sw $23, 4($29)

ori $29, $0, 0

lw $23, 8($29)

beq $23, $29, label5

ori $23, $0, 0

lw $29, 4($23)

subu $30, $23, $29

label5: lui $30, 45103

ori $30, $0, 0

sw $29, 0($30)

ori $30, $0, 0

lw $29, 4($30)

lui $30, 1374

ori $30, $23, 23500

addu $29, $30, $29

ori $29, $0, 0

sw $29, 0($29)

ori $29, $30, 43572

ori $23, $0, 0

lw $23, 4($23)

subu $23, $29, $29

ori $30, $0, 0

sw $23, 12($30)

subu $23, $23, $23

j label6

ori $23, $0, 0

sw $23, 8($23)

addu $29, $29, $29

label6: subu $29, $30, $29

ori $29, $0, 0

sw $23, 8($29)

jal label7

ori $29, $0, 16

ori $23, $29, 51714

label7: addu $29, $29, $31

jr $29

nop

期待输出：

18@00003000: $28 <= 00000000

22@00003004: $29 <= 00000000

26@00003008: $23 <= 00000000

26@0000300c: \*00000008 <= 00000000

38@00003014: $23 <= 00000000

42@00003024: $23 <= 0000ecb0

54@0000302c: $23 <= 46e20000

58@00003034: $30 <= 00000000

58@00003038: \*00000000 <= 46e20000

66@0000303c: $23 <= 00000000

70@00003040: $23 <= cc900000

82@00003048: $23 <= 05810000

86@00003050: $30 <= 00000000

94@00003058: $23 <= 00000000

98@00003064: $29 <= 00000000

98@00003068: \*00000004 <= 00000000

106@0000306c: $29 <= 00000000

110@00003070: $23 <= 00000000

126@00003078: $23 <= 00000000

130@00003084: $30 <= b02f0000

134@00003088: $30 <= 00000000

134@0000308c: \*00000000 <= 00000000

142@00003090: $30 <= 00000000

146@00003094: $29 <= 00000000

150@00003098: $30 <= 055e0000

154@0000309c: $30 <= 00005bcc

158@000030a0: $29 <= 00005bcc

162@000030a4: $29 <= 00000000

162@000030a8: \*00000000 <= 00000000

170@000030ac: $29 <= 0000fbfc

174@000030b0: $23 <= 00000000

178@000030b4: $23 <= 00000000

182@000030b8: $23 <= 00000000

186@000030bc: $30 <= 00000000

186@000030c0: \*0000000c <= 00000000

194@000030c4: $23 <= 00000000

202@000030cc: $23 <= 00000000

206@000030d8: $29 <= ffff0404

210@000030dc: $29 <= 00000000

210@000030e0: \*00000008 <= 00000000

218@000030e4: $31 <= 000030ec

222@000030e8: $29 <= 00000010

226@000030f0: $29 <= 000030fc

#### **3、暂停测试**

ori $1,$1,10

addu $1,$1,$1

lw $1,0($1)

sw $1,0($1)

jal label1

addu $31,$31,$1

label1:addu $31,$31,$31

jr $31

期待输出：

18@00003000: $ 1 <= 0000000a

22@00003004: $ 1 <= 00000014

26@00003008: $ 1 <= 00000000

30@0000300c: \*00000000 <= 00000000

38@00003010: $31 <= 00003018

46@00003014: $31 <= 00003018

50@00003018: $31 <= 00006030

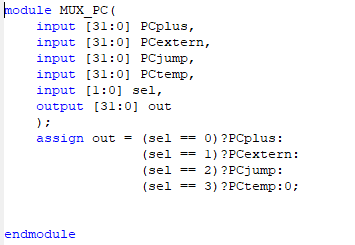
## **三、思考题**

### 流水线冒险

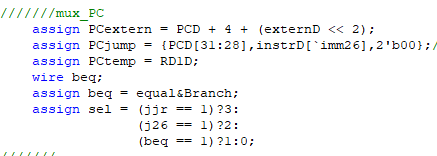
1、在采用本节所述的控制冒险处理方式下，PC的值应当如何被更新？请从数据通路和控制信号两方面进行说明。

解答：

首先示例代码如下：



对于每一个单元的值显示如下：



简单而言，当产生跳转指令时：

（1）BEQ:比较单元位于D级，并通过CMP单元得到equal值，以及C\_IF\_ID单元得到Branch判断是否将sel选择信号置1，如果置1，则跳转到PCextern，否则不跳转。

（2）j指令：C\_IF\_ID输出信号之后sel根据jr还是jal+j指令直接对PC进行更新，且更新一定发生。

（3）延迟槽：由于此过程于D级发生，F级新出指令在跳转后进入D级从而执行。因此延迟槽指令一定执行。

2、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？

解答：

由于延迟槽。因为延迟槽（PC+4）的指令一定执行，因此在返回时应返回到PC+8的指令，而不是PC+4，以免延迟槽指令重复执行。

### 数据冒险的分析

3、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？

解答：

因为存在数据冒险，从而需要进行转发，ALU和DM自身得到的数据可能并不是正确的数据。

### AT法处理流水线数据冒险

4、“转发（旁路）机制的构造”中的Thinking 1-4；

解答：

（1）可能会有多级转发，因此得到的必须是已经转发后正确的数据，否则就容易发生冲突而得不到正确的数据。举例如下

Ori $1,100

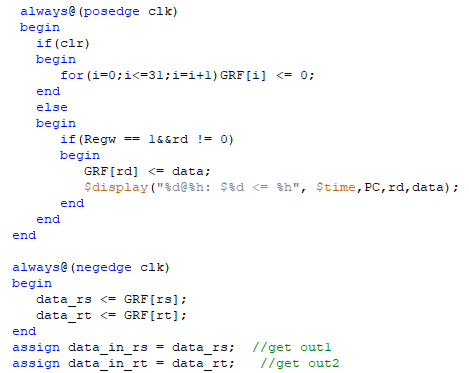
addu $1,$1,$1

addu $1,$1,$1

addu $1,$1,$1

该过程从只有到第8个周期才能得到1号寄存器正确的数据，如果不是转发的已经转发过的数据，这势必会使得结果出错。

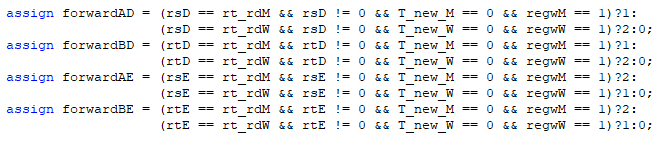
（2）内部转发实际上是剔除了WB阶段和ID阶段同时存值和取值的冲突。一般来说可以通过控制时钟上升沿和下降沿来将其分配到一个周期内。可以在时钟上升沿存入数据，然后在时钟下降沿输出寄存器中数据。



（3）零号寄存器是不能被更改的，因此有关0号寄存器的写入和输出都不用进行转发。

（4）最新产生的数据优先级最高如下：比如某一寄存器同时在WB阶段、MEM阶段、E级均有新的输出，从而应该优先选择最新的输出进行判断。因为最新的输出得到的是最新的值而不是“旧值”，从而避免出错。

代码示例如下：



先判断M级再判断W级。

5、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？

解答：实际上这个方法我并没有实现，我还是加入了regw进行附加判断。一个想法是认为可能we和A同时判断，如果we为0则将A置0从而避免转发？

### 在线测试相关说明

6、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？

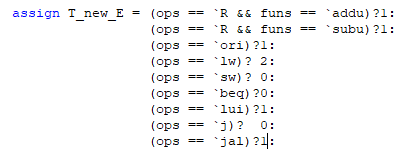
解答：这个地方我觉得比较重要的就是两个冲突没有注意：31号寄存器产生的冲突和sw指令产生的冲突（跳转和存取）。

我就拿测评机前后输出进行说明吧。

（1）31号寄存器产生冲突

实际上是跳转指令和存取指令发生的冲突。问题在于设置jal的T\_new\_E时没有和实际PC+8产生的单位对应。

**问题如下：**在E级中设置jal的T\_new\_E如下：



本身来说这么设置也可以，说明PC+8位于M级产生也行，只需要对应就行。但是我实际处理将PC+8产生放在了W级：



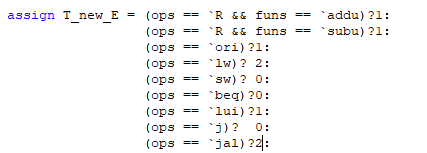
两者的不对应会使得jal紧接着的对于31号寄存器操作发生错误。

比如：jal label

Addu $31,$31,$1

会发生不可预置的错误。

**修改方式很简单：**



将1改为2即可，从而能够发生对应（在修改幅度最小下）。

**实际上我发现很多指令发生冲突后在CPU大体无错下，只要修改一下T\_NEW和T\_use，从而其会在适当的时候加上nop，很多问题就解决了。下面一个错误也是修改了T\_use之后得到正确结果。**

（2）sw指令冲突（存取和跳转指令）

最开始设置的T\_USE如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 名称 | 取值 |
| Sw | T\_USE\_RS | 1 |
| T\_USE\_RT | 2 |

实际上我从同学那里得知其设置就是1和2没有发生问题。但是这种方法并不适用我的CPU，因此我修改了一下其值，不过不得说可能我的CPU性能会因此下降。

|  |  |  |
| --- | --- | --- |
| 名称 | 名称 | 取值 |
| Sw | T\_USE\_RS | 1 |
| T\_USE\_RT | 1 |

我将T\_USE\_RT提前到E级的ALU前的MUX处。从而符合我CPU转发时只有四个转发途径：M到D、M到E、W到D、W到E。