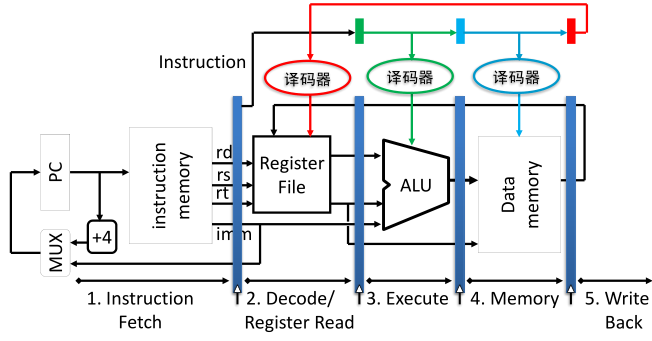
**p5实验报告**

田宇（14231006）

1. **总体结构：**

（采用分布式（译码）控制器）



1. **模块描述：**

1、Adder4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | IN | I | 31:0 | 通常为当前PC值 |
| 2 | OUT | O | 31:0 | 通常为下一PC值（PC+4） |

1. Adder8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | IN | I | 31:0 | 通常为当前PC值 |
| 2 | OUT | O | 31:0 | 跳转指令后下一PC值（PC+8） |

1. AlgLogUnit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | Operator | I | 31:0 | 运算符 |
| 2 | OperandSa | I | 4:0 | 移位指令的操作数（移动的位数） |
| 3 | Operand1 | I | 31:0 | 运算操作数1 |
| 4 | Operand2 | I | 31:0 | 运算操作数2 |
| 5 | Result | O | 31:0 | 运算结果 |

1. Comparator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | CmpMode | I | 2:0 | 分支比较操作符 |
| 2 | NUM1 | I | 31:0 | 分支比较操作数1 |
| 3 | NUM2 | I | 31:0 | 分支比较操作数1 |
| 4 | Branch | O | 1 | 分支比较结果（是否满足分支跳转条件） |

1. DataMemory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | MemWrite | I | 1 | 数据存储器的写信号 |
| 4 | DataAddr | I | 9:0 | 对数据存储器进行读写的寻址地址 |
| 5 | WriteData | I | 31:0 | 写入数据存储器的数据 |
| 6 | ReadData | O | 31:0 | 读出数据存储器的数据 |

1. DecoderD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | Branch | I | 1 | 分支比较结果（是否满足分支跳转条件） |
| 2 | InstructionD | I | 31:0 | D级流水线的指令 |
| 3 | ExtMode | O | 1:0 | 数据扩展操作符 |
| 4 | NPCSelect | O | 1:0 | 下一PC值的选择信号 |
| 5 | CmpMode | O | 2:0 | 分支比较操作符 |

1. DecoderE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | InstructionE | I | 31：0 | E级流水线的指令 |
| 2 | ALUSrc | O | 1 | 运算器操作数2的选择信号（寄存器值或立即数） |
| 3 | RegDst | O | 1:0 | 运算结果写入的寄存器的选择信号 |
| 4 | Operator | O | 3:0 | 运算操作符 |

1. DecoderM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | InstructionM | I | 31：0 | M级流水线的指令 |
| 2 | MemWrite | O | 1 | 数据存储器的写信号 |

1. DecoderW

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | InstructionW | I | 31：0 | M级流水线的指令 |
| 2 | RegWrite | O | 1 | 寄存器的写信号 |
| 3 | MemtoReg | O | 1:0 | 写入寄存器的数据来源选择（存储器值或运算器或PC+8） |

1. Extender

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | Mode | I | 1:0 | 数据扩展操作符（符号扩展或零扩展） |
| 2 | Immediate16 | I | 15:0 | 16位立即数（待扩展数据） |
| 3 | Immediate32 | O | 31:0 | 32位立即数（扩展结果） |

1. GenRegs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | RegWrite | I | 1 | 寄存器的写信号 |
| 4 | RS | I | 4:0 | 读出寄存器数据1的寻址地址 |
| 5 | RT | I | 4:0 | 读出寄存器数据2的寻址地址 |
| 6 | RD | I | 4:0 | 写入寄存器数据的寻址地址 |
| 7 | WriteData | I | 31:0 | 写入寄存器的数据 |
| 8 | ReadData1 | O | 31:0 | 读出寄存器数据1 |
| 9 | ReadData2 | O | 31:0 | 读出寄存器数据2 |

1. InstrMemory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | InstrAddr | I | 9:0 | 指令寻址地址 |
| 2 | Instruction | O | 31:0 | 指令 |

1. Multiplexer2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | select | I | 1 | 选择信号 |
| 2 | IN0 | I | 31:0 | 输入数据1 |
| 3 | IN1 | I | 31:0 | 输入数据2 |
| 4 | OUT | O | 31:0 | 选择结果（输出数据） |

1. Multiplexer3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | select | I | 1:0 | 选择信号 |
| 2 | IN1 | I | 31:0 | 输入数据1 |
| 3 | IN2 | I | 31:0 | 输入数据2 |
| 4 | IN3 | I | 31:0 | 输入数据3 |
| 5 | OUT | O | 31:0 | 选择结果（输出数据） |

1. Multiplexer5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | select | I | 1:0 | 选择信号 |
| 2 | IN1 | I | 31:0 | 输入数据1 |
| 3 | IN2 | I | 31:0 | 输入数据2 |
| 4 | IN3 | I | 31:0 | 输入数据3 |
| 5 | IN4 | I | 31:0 | 输入数据4 |
| 6 | IN5 | I | 31:0 | 输入数据5 |
| 7 | OUT | O | 31:0 | 选择结果（输出数据） |

1. MultiplexerR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | select | I | 1:0 | 选择信号 |
| 2 | RegAddr1 | I | 4:0 | RT（寄存器地址1） |
| 3 | RegAddr2 | I | 4:0 | RD（寄存器地址2） |
| 4 | RegAddr3 | I | 4:0 | 31 |
| 5 | RegAddr | O | 4:0 | 写入寄存器的寻址地址 |

1. NextPC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | select | I | 1:0 | 选择信号 |
| 2 | ReadData1 | I | 31:0 | RS寄存器的读出数据（跳转地址） |
| 3 | PCounter4 | I | 31:0 | PC+4 |
| 4 | Instuction | I | 31:0 | 指令（偏移地址信息） |
| 5 | nextPCounter | O | 31:0 | 下一条指令地址 |

1. NPCEnable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | Branch | I | 1 | 分支跳转信号 |
| 2 | NPCSelect | I | 1:0 | 下一条指令地址选择信号 |
| 3 | nextEnable | O | 1 | NPC使能信号 |

1. PC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | PC\_EN | I | 1 | PC使能信号（是否转到下一条指令地址） |
| 4 | nextPCounter | I | 31:0 | 下一条指令地址 |
| 5 | PCounter | O | 31:0 | 指令地址 |

1. PipeRegFD

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | IFID\_EN | I | 1 | IFID使能信号（F级和D级是否进行数据传递） |
| 4 | InstructionF | I | 31:0 | F级指令 |
| 5 | PCounter4F | I | 31:0 | F级下一条指令地址（PC+4） |
| 6 | PCounter8F | I | 31:0 | F级下一条指令地址（PC+8） |
| 7 | InstructionD | O | 31:0 | D级指令 |
| 8 | PCounter4D | O | 31:0 | D级下一条指令地址（PC+4） |
| 9 | PCounter8D | O | 31:0 | D级下一条指令地址（PC+8） |

1. PipeRegDE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | IDEX\_CLR | I | 1 | IDEX清除信号（E级流水线寄存器是否清空） |
| 4 | InstructionD | I | 31:0 | D级指令 |
| 5 | PCounter8D | I | 31:0 | D级下一条指令地址（PC+8） |
| 6 | ReadData1D | I | 31:0 | D级寄存器读出数据1 |
| 7 | ReadData2D | I | 31:0 | D级寄存器读出数据2 |
| 8 | ExtOutD | I | 31:0 | D级数据扩展结果 |
| 9 | InstructionE | O | 31:0 | E级指令 |
| 10 | PCounter8E | O | 31:0 | E级下一条指令地址（PC+8） |
| 11 | ReadData1E | O | 31:0 | E级寄存器读出数据1 |
| 12 | ReadData2E | O | 31:0 | E级寄存器读出数据2 |
| 13 | ExtOutE | O | 31:0 | E级数据扩展结果 |

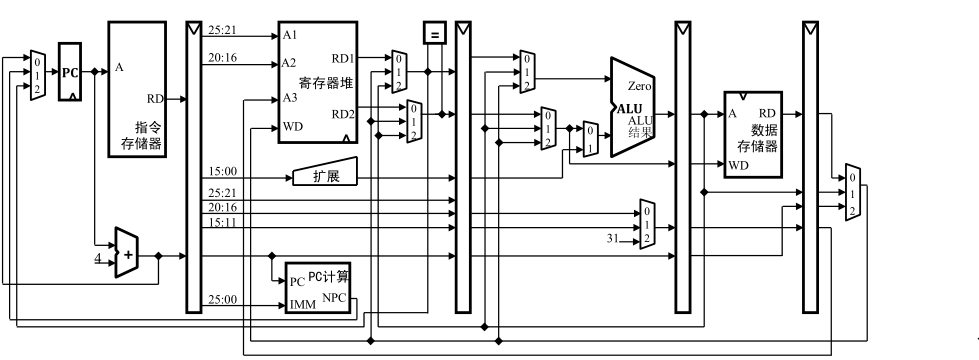
1. PipeRegEM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | WriteRegE | I | 4:0 | E级写入寄存器的寻址地址 |
| 4 | InstructionE | I | 31:0 | E级指令 |
| 5 | PCounter8E | I | 31:0 | E级下一条指令地址（PC+8） |
| 6 | ReadData2E | I | 31:0 | E级寄存器读出数据2 |
| 7 | ALUOutE | I | 31:0 | E级运算器输出结果 |
| 8 | WriteRegM | O | 4:0 | M级写入寄存器的寻址地址 |
| 9 | InstructionM | O | 31:0 | M级指令 |
| 10 | PCounter8M | O | 31:0 | M级下一条指令地址（PC+8） |
| 11 | ReadData2M | O | 31:0 | M级寄存器读出数据2 |
| 12 | ALUOutM | O | 31:0 | M级运算器输出结果 |

1. PipeRegMW

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 端口名称 | 方向 | 位宽 | 功能描述 |
| 1 | clk | I | 1 | 时钟信号 |
| 2 | reset | I | 1 | 复位信号 |
| 3 | WriteRegM | I | 4:0 | M级写入寄存器的寻址地址 |
| 4 | InstructionM | I | 31:0 | M级指令 |
| 5 | PCounter8M | I | 31:0 | M级下一条指令地址（PC+8） |
| 6 | ReadDataM | I | 31:0 | M级存储器读出数据 |
| 7 | ALUOutM | I | 31:0 | M级运算器输出结果 |
| 8 | WriteRegW | O | 4:0 | W级写入寄存器的寻址地址 |
| 9 | InstructionW | O | 31:0 | W级指令 |
| 10 | PCounter8W | O | 31:0 | W级下一条指令地址（PC+8） |
| 11 | ReadDataW | O | 31:0 | W级存储器读出数据 |
| 12 | ALUOutW | O | 31:0 | W级运算器输出结果 |

1. **冲突分析**



1. **转发**

rd<—cal\_r/cal\_s/jalr

rt<—cal\_i

31<—jal

* 1. mux\_rd1

D级分支跳转指令（B、Bz、jr、jalr指令）需用到Read1D值，该值可能来自D级通用寄存器数据Read1（GPR[rsD]）、M级运算器结果ALUOutM或下一条指令地址PCounter8M、W级写入寄存器的数据Write，使用Read1Select信号进行选择。

* + 1. Read1Select=2’b01

若未发生冲突（冒险），则直接使用当前从寄存器读出的数据Read1（GPR[rsD]）作为Read1D；

若有冲突发生（D级所需用到的Read1D值来自W级写入寄存器的数据Write），正常情况下还需一个时钟周期Write才会被写入寄存器GPR[rsD]，则可利用通用寄存器的内部转发实现，读出的数据Read1（GPR[rsD]）即为Write，故仍然直接使用当前从寄存器读出的数据Read1（GPR[rsD]）作为Read1D。

* + 1. Read1Select=2’b10

若有冲突发生（M级当前指令为cal\_r/cal\_s/cal\_i，D级所需用到的Read1D值来自M级运算器结果ALUOutM），正常情况下还需两个时钟周期ALUOutM才会被写入寄存器GPR[rsD]，则利用外部转发，使用当前M级运算器结果ALUOutM作为Read1D。

* + 1. Read1Select=2’b11

若有冲突发生（M级当前指令为jal/jalr，D级所需用到的Read1D值来自M级下一条指令地址PCounter8M），正常情况下还需两个时钟周期PCounter8M才会被写入寄存器GPR[rsD]，则利用外部转发，使用当前M级下一条指令地址PCounter8M作为Read1D。

* 1. mux\_rd2

D级分支跳转指令（B、jr、jalr指令）需用到Read2D值，该值可能来自D级通用寄存器数据Read2（GPR[rtD]）、M级运算器结果ALUOutM或下一条指令地址PCounter8M、W级写入寄存器的数据Write，使用Read2Select信号进行选择。

* + 1. Read2Select=2’b01

若未发生冲突（冒险），则直接使用当前从寄存器读出的数据Read2（GPR[rtD]）作为Read2D；

若有冲突发生（D级所需用到的Read2D值来自W级写入寄存器的数据Write），正常情况下还需一个时钟周期Write才会被写入寄存器GPR[rtD]，则可利用通用寄存器的内部转发实现，读出的数据Read2（GPR[rtD]）即为Write，故仍然直接使用当前从寄存器读出的数据Read2（GPR[rtD]）作为Read2D。

* + 1. Read1Select=2’b10

若有冲突发生（M级当前指令为cal\_r/cal\_s/cal\_i，D级所需用到的Read2D值来自M级运算器结果ALUOutM），正常情况下还需两个时钟周期ALUOutM才会被写入寄存器GPR[rtD]，则利用外部转发，使用当前M级运算器结果ALUOutM作为Read2D。

* + 1. Read1Select=2’b11

若有冲突发生（M级当前指令为jal/jalr，D级所需用到的Read2D值来自M级下一条指令地址PCounter8M），正常情况下还需两个时钟周期PCounter8M才会被写入寄存器GPR[rtD]，则利用外部转发，使用当前M级下一条指令地址PCounter8M作为Read2D。

* 1. mux\_opr1

E级运算指令（cal\_r/cal\_s/cal\_i指令）或存取指令（lw/sw）需使用Operand1值进行运算，该值可能来自当前从IDEX流水寄存器读出的数据Read1E、M级运算器结果ALUOutM或下一条指令地址PCounter8M、W级写入寄存器的数据Write，使用Opr1Select信号进行选择。

* + 1. Opr1Select=3’b001

若未发生冲突（冒险），则直接使用当前从IDEX流水寄存器读出的数据Read1E作为Operand1。

* + 1. Opr1Select=3’b010

若有冲突发生（M级当前指令为cal\_r/cal\_s/cal\_i，E级所需用到的Operand1值来自M级运算器结果ALUOutM），正常情况下还需三个时钟周期ALUOutM值才会被传递至E级，则利用外部转发，使用当前M级运算器结果ALUOutM作为Operand1。

* + 1. Opr1Select=3’b011

若有冲突发生（M级当前指令为jal/jalr，E级所需用到的Operand1值来自M级下一条指令地址PCounter8M），正常情况下还需三个时钟周期PCounter8M才会被传递至E级，则利用外部转发，使用当前M级下一条指令地址PCounter8M作为Operand1。

* + 1. Opr1Select=3’b100

若有冲突发生（W级当前指令为cal\_r/cal\_s/cal\_i/jal/jalr，E级所需用到的Operand1值来自W级写入通用寄存器的数据Write），正常情况下还需两个时钟周期Write才会被传递至E级，则利用外部转发，使用当前W级写入通用寄存器的数据Write。

* 1. mux\_opr2

E级运算指令（cal\_r/cal\_s指令）或存指令（sw）需使用Operand2值进行运算，该值可能来自当前从IDEX流水寄存器读出的数据Read2E、M级运算器结果ALUOutM或下一条指令地址PCounter8M、W级写入寄存器的数据Write，使用Opr2Select信号进行选择。

* + 1. Opr2Select=3’b001

若未发生冲突（冒险），则直接使用当前从IDEX流水寄存器读出的数据Read2E作为Operand2。

* + 1. Opr2Select=3’b010

若有冲突发生（M级当前指令为cal\_r/cal\_s/cal\_i，E级所需用到的Operand2值来自M级运算器结果ALUOutM），正常情况下还需三个时钟周期ALUOutM值才会被传递至E级，则利用外部转发，使用当前M级运算器结果ALUOutM作为Operand2。

* + 1. Opr2Select=3’b011

若有冲突发生（M级当前指令为jal/jalr，E级所需用到的Operand2值来自M级下一条指令地址PCounter8M），正常情况下还需三个时钟周期PCounter8M才会被传递至E级，则利用外部转发，使用当前M级下一条指令地址PCounter8M作为Operand2。

* + 1. Opr2Select=3’b100

若有冲突发生（W级当前指令为cal\_r/cal\_s/cal\_i/lw/jal/jalr，E级所需用到的Operand2值来自W级写入通用寄存器的数据Write），正常情况下还需两个时钟周期Write才会被传递至E级，则利用外部转发，使用当前W级写入通用寄存器的数据Write。

* 1. mux\_dwd

M级存指令（sw）需将WriteData值存入数据存储器，该值可能来自当前从EXMEM流水寄存器读出的数据WriteM、W级运算器结果ALUOutW或下一条指令地址PCounter8W，使用WriteSelect信号进行选择。

* + 1. WriteSelect=0

若未发生冲突（冒险），则直接使用当前从EXMEM流水寄存器读出的数据WriteM作为WriteData。

* + 1. WriteSelect=1

若有冲突发生（W级当前指令为cal\_r/cal\_s/cal\_i，M级所需用到的WriteData值来自W级运算器结果ALUOutW），正常情况下还需三个时钟周期ALUOutW值才会被传递至M级，则利用外部转发，使用当前W级运算器结果ALUOutW作为WriteData。

若有冲突发生（W级当前指令为jal/jalr，M级所需用到的WriteData值来自W级下一条指令地址PCounter8W），正常情况下还需三个时钟周期PCounter8W才会被传递至M级，则利用外部转发，使用当前W级下一条指令地址PCounter8W作为WriteData。

若有冲突发生（W级当前指令为lw，M级所需用到的WriteData值来自W级读出的存储器数据ReadDataW），正常情况下还需三个时钟周期ReadDataW才会被传递至M级，则利用外部转发，使用当前W级读出的存储器数据ReadDataW作为WriteData。

1. **暂停**

PC\_EN<—0

IFID\_EN<—0

IDEX\_CLR<—1

* 1. rsD
     1. D级当前指令为cal\_r/cal\_i/lw/sw

若有冲突发生（D级所需用到的Read1D值来自E级指令lw的执行结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。已经在流水线中设置相关转发机制，保持D级指令不变使当前E级指令lw运行至M级时，则冲突解除，故D级及其之前的流水线需要暂停一个周期。

注意：暂停一个周期以后D级所需用到的Read1D值来自M级指令lw的执行结果，正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。但是D级当前指令真正使用Read1D值是在E级运算器中，故当运算指令运行至E级时，lw指令运行至W级，可通过转发机制中的Opr1Select（=3’b100）解除冲突。

* + 1. D级当前指令为B/Bz/jr/jalr

若有冲突发生（D级所需用到的Read1D值来自E级运算结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前E级运算指令运行至M级时可利用Read1Select（=2’b10）信号进行转发，从而解除冲突，故D级及其之前的流水线需要暂停一个周期。

若有冲突发生（D级所需用到的Read1D值来自E级lw指令执行结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前E级运算指令运行至W级时可利用通用寄存器内部转发，从而解除冲突，故D级及其之前的流水线需要连续暂停两个周期（先暂停一个周期，再暂停一个周期）。

若有冲突发生（D级所需用到的Read1D值来自M级lw指令执行结果），正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前M级运算指令运行至W级时可利用通用寄存器内部转发，从而解除冲突，故D级及其之前的流水线需要暂停一个周期。（注意：若不暂停一个周期，D级当前指令真正使用Read1D值整整使用来自M级指令lw的执行结果，正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rsD]。但是D级当前指令真正使用Read1D值就在D级，不同于i中的情况，无法进行转发避免冲突，故必须暂停一个周期）

* 1. rtD
     1. D级当前指令为cal\_r/cal\_s

若有冲突发生（D级所需用到的Read2D值来自E级指令lw的执行结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。已经在流水线中设置相关转发机制，保持D级指令不变使当前E级指令lw运行至M级时，则冲突解除，故D级及其之前的流水线需要暂停一个周期。（注意：暂停一个周期以后D级所需用到的ReadtD值来自M级指令lw的执行结果，正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。但是D级当前指令真正使用Read2D值是在E级运算器中，故当运算指令运行至E级时，lw指令运行至W级，可通过转发机制中的Opr2Select（=3’b100）解除冲突）

* + 1. D级当前指令为B

若有冲突发生（D级所需用到的Read2D值来自E级运算结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前E级运算指令运行至M级时可利用Read2Select（=2’b10）信号进行转发，从而解除冲突，故D级及其之前的流水线需要暂停一个周期。

若有冲突发生（D级所需用到的Read2D值来自E级lw指令执行结果），正常情况下还需三个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前E级运算指令运行至W级时可利用通用寄存器内部转发，从而解除冲突，故D级及其之前的流水线需要连续暂停两个周期（先暂停一个周期，再暂停一个周期）。

若有冲突发生（D级所需用到的Read2D值来自M级lw指令执行结果），正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。已经在流水线中设置相关转发机制，则保持D级指令不变使当前M级运算指令运行至W级时可利用通用寄存器内部转发，从而解除冲突，故D级及其之前的流水线需要暂停一个周期。（注意：若不暂停一个周期，D级当前指令真正使用Read2D值整整使用来自M级指令lw的执行结果，正常情况下还需两个时钟周期D级所需数据才能写入通用寄存器GPR[rtD]。但是D级当前指令真正使用Read2D值就在D级，不同于i中的情况，无法进行转发避免冲突，故必须暂停一个周期）

1. **测试用例**
   1. 测试程序

addiu $t2,$zero,-345

ori $t0,$zero,123

xor $t0,$t0,$t2 #mux\_opr1：$t0转发自ALUoutM

xori $t1,$t0,234 #mux\_opr1：$t0转发自ALUoutM

lui $t3,456

slti $t4,$t2,567

slti $t6,$t2,567

beq $t4,$t6,lab1 #mux\_rd1：$t4转发自ALUoutM；$t6必须暂停（然后转发自ALUoutM）

addi $t5,$t2,345

lab1: sw $t2,($zero)

nor $t1,$t1,$t5

or $t3,$t1,$t2 #mux\_opr1：$t1转发自ALUoutM

sw $t3,8($zero) #mux\_dwd：$t3转发自ALUoutW

andi $t4,$t2,678

bne $t5,$zero,lab2

sll $t4,$t3,2

j lab2

lw $t6,8($zero)

nop

lab2: slt $t0,$t4,$t6 #$t4必须暂停（然后转发自ALUoutW）

srav $t1,$t3,$t0 #mux\_opr2：$t0转发自ALUoutM

lw $t5,8($zero)

subu $t7,$t5,$t4 #$t5必须暂停（然后转发自ALUoutW）

sllv $s0,$t7,$t3 #mux\_opr1：$t7转发自ALUoutM

srlv $s1,$s0,$t3 #mux\_opr1：$s0转发自ALUoutM

jal lab3

sltu $t0,$ra,$t5 #mux\_opr1：$ra转发自pc8M

lab3: sw $t6,16($zero)

or $s0,$s0,$s1

andi $s1,$s1,555

sub $t6,$t6,$t7

add $t7,$t0,$t1

sw $t7,20($zero) #mux\_dwd：$t7转发自ALUoutW

xor $t1,$t6,$t7 #mux\_opr2：$t7转发自ALUoutW

la $a0,lab4

nor $t5,$4,$a0 #mux\_opr2：$a0转发自ALUoutM

jr $a0 #mux\_rd1：$a0转发自ALUoutM

sub $t6,$t5,$zero #mux\_opr1：$t5转发自ALUoutW

lab4: sltiu $t6,$t1,-666

srl $t0,$t6,3 #mux\_opr2：$t6转发自ALUoutM

sub $t3,$t6,$zero #mux\_opr1：$t6转发自ALUoutW

lw $t5,($zero)

and $t2,$t2,$4

sra $t7,$t5,4 #mux\_opr2：$t5转发自Write

add $t4,$t1,$t5 #$t5转发自GPR内部

sw $t6,12($zero)

j lab5

sw $t4,20($zero) #$t4转发自GPR内部

lab5: sll $t4,$t7,8

la $a1,exit

jalr $t4,$a1 #$a1必须暂停（然后转发自ALUoutM）

slti $t7,$t4,484 #mux\_opr1：$t4转发自pc8M

exit: sw $t2,32($zero)

lw $t4,32($zero)

sw $t4,24($zero) #mux\_dwd：$t4转发自Write

sw $t4,40($zero) #mux\_opr2：$t4转发自Write

* 1. 测试结果

$10 <= fffffea7

$ 8 <= 0000007b

$ 8 <= fffffedc

$ 9 <= fffffe36

$11 <= 01c80000

$12 <= 00000001

$14 <= 00000001

$13 <= 00000000

\*00000000 <= fffffea7

$ 9 <= 000001c9

$11 <= ffffffef

\*00000008 <= ffffffef

$12 <= 000002a6

$12 <= ffffffbc

$14 <= ffffffef

$ 8 <= 00000001

$ 9 <= fffffff7

$13 <= ffffffef

$15 <= 00000033

$16 <= 00198000

$17 <= 00000033

$31 <= 0000306c

$ 8 <= 00000001

\*00000010 <= ffffffef

$16 <= 00198033

$17 <= 00000023

$14 <= ffffffbc

$15 <= fffffff8

\*00000014 <= fffffff8

$ 9 <= 00000044

$ 4 <= 00003098

$13 <= ffffcf67

$14 <= ffffcf67

$14 <= 00000001

$ 8 <= 00000000

$11 <= 00000001

$13 <= fffffea7

$10 <= 00003080

$15 <= ffffffea

$12 <= fffffeeb

\*0000000c <= 00000001

\*00000014 <= fffffeeb

$12 <= ffffea00

$ 5 <= 000030d0

$12 <= 000030d0

$15 <= 00000000

\*00000020 <= 00003080

$12 <= 00003080

\*00000018 <= 00003080

\*00000028 <= 00003080

1. 疑惑

对于连续的两条分支跳转语句（b、j指令）的处理：

暂停机制理论上可行，但仿真后发现PC值不受控，实际输出与MARS不能保持一致。

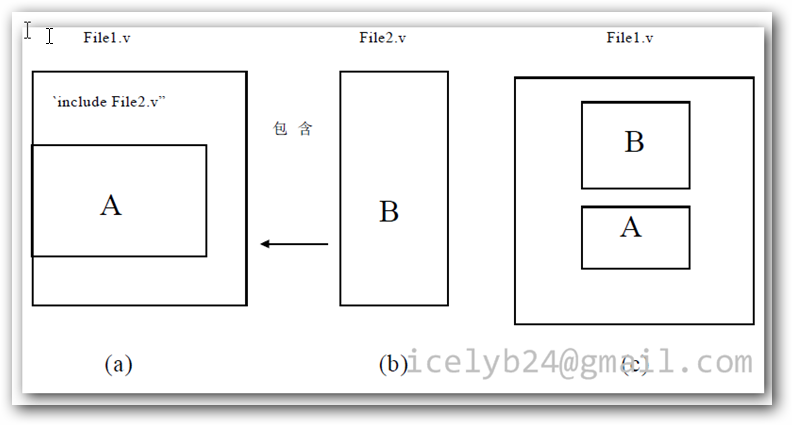
1. 实验收获

代码改进（降低冗余度）

因为我在p5采用了分布式译码，所以在每级流水线都需要一个Decoder，也就是说，我在每一级的都将所有指令译出，以便得到cal\_r、cal\_i、cal\_s等指令分类及其对应的控制信号。另外在冲突单元Hazard中则几乎要将D、E、M、W级的所有指令都在一个module中译出。

在需要实现的指令较少时，这样的代码风格可能并不会产生太明显的不良影响，但是随着我逐渐向代码中添加指令（尽早为p6做准备），我发现上述风格其实是一个巨大的隐患，牵一发而动全身，很有可能因为一点小的疏忽而导致指令不能正常执行。而且，可读性是比较差的，冗余度大也是一种空间上的浪费。

我找到了两种比较好的解决方案，一种是implement（可能比较常用），另一种是include。我主要使用了第二种，使用效果如下图所示。这种方法类似于C语言中的头文件，常常用来定义一些parameter，甚至比头文件的使用更加灵活一点。



在阅读相关博客时，看到留言说include使用不当可能会导致一些问题，但是并没有具体给出“使用不当”的定义以及“问题”的解释说明，我也暂时没有遇到任何异常，有待后续了解。希望在之后的试验中能够学习更多有关verilog编码的知识，继续改进自己的代码风格。