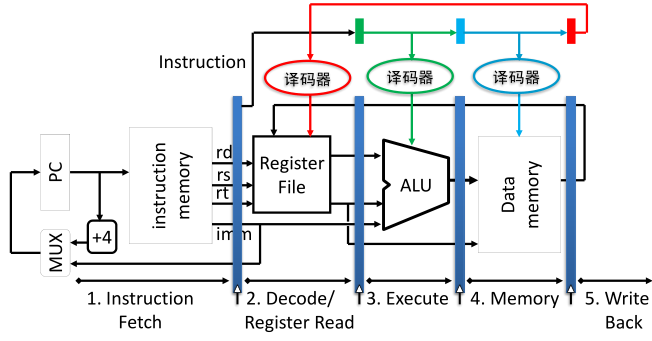
**p6实验报告**

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. MultDivUnit

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | Start | I | 1 | 乘除运算开始信号 |
| 4 | HLSelect | I | 1 | 写入寄存器的选择信号 |
| 5 | HLRegWrite | I | 1 | 乘除寄存器的写信号 |
| 6 | MDOperator | I | 1:0 | 乘除运算符 |
| 7 | MDOperand1 | I | 31:0 | 乘除运算的操作数1 |
| 8 | MDOperand2 | I | 31:0 | 乘除运算的操作数2 |
| 9 | Busy | O | 1 | 乘除寄存器的阻塞信号 |
| 10 | HiData | O | 31:0 | Hi寄存器中读出的数据 |
| 11 | LoData | O | 31:0 | Lo寄存器中读出的数据 |

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 | 2:0 | 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 |
| 7 | 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 | 2:0 | 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**

新增的lb/lbu/lh/lhu和指令sb/sh分别与lw和sw指令相同处理。

下面仅说明乘除单元部分。

1. **暂停**

假定执行乘法的时间是5个周期，执行除法的时间是10个周期，则在乘除单元执行运算时，所有与该部件相关的指令都应该被阻塞，此时考虑暂停信号。

当Busy为1时，mfhi、mflo、mthi、mtlo、mult、multu、div、divu均被阻塞在IFID流水线寄存器，即需要增加一个暂停分量Stall\_1=(mfhiD|mfloD|mthiD|mtloD|multD|multuD|divD|divuD)&Busy；

当Busy为0时，若乘除指令（mult、multu、div、divu）到达EX级，且下一条指令与乘除单元冲突（即指令为mfhi、mflo、mthi、mtlo、mult、multu、div、divu），则此时也需要将ID级指令阻塞在IFID流水线寄存器，即增加另一个暂停分量Stall\_2=(mfhiD|mfloD|mthiD|mtloD|multD|multuD|divD|divuD)&(multE|multuE|divE|divuE)。

1. **转发**
   1. Read1Select

对于bD/bzD/jrD/jalrD等需要在D级使用RData1D（正常情况下来自GPR[rsD]）的指令，相比较p5，又增加了两个寄存器作为转发来源，即HiDataM和LoDataM。

* 1. Read2Select

对于bD等需要在D级使用RData2D（正常情况下来自GPR[rtD]）的指令，相比较p5，同样增加HiDataM和LoDataM作为转发来源。

* 1. Opr1Select

在p5中CalRE/CalIE\_Log/CalIE\_Alg/LdE/StoE等指令需要在E级使用流水线寄存器中的RData1E（正常情况下来自GPR[rsE]），相比较p5，增加了MtHLE（mthi、mtlo）指令需要使用RData1E作为写入Hi或Lo寄存器的数据，和CalHLE（mult、multu、div、divu）指令需要使用RData1E作为乘除法运算的第一个操作数。

另外，与Read1Select相似，由于新增Hi和Lo两个寄存器，RData1E又增加了四个转发来源，即HiDataM、LoDataM、HiDataW和LoDataW。

* 1. Opr2Select

在p5中CalRE/CalSE/StoE等指令需要在E级使用流水线寄存器中的RData2E（正常情况下来自GPR[rtE]），相比较p5，增加了CalHLE（mult、multu、div、divu）指令需要使用RData2E作为乘除法运算的第二个操作数。

另外，与Read2Select相似，由于新增Hi和Lo两个寄存器，RData2E又增加了四个转发来源，即HiDataM、LoDataM、HiDataW和LoDataW。

* 1. WriteSelect

在p5中StoM等指令需要在M级使用流水线寄存器中的RData2M（正常情况下来自GPR[rtM]），相比较p5，由于新增Hi和Lo两个寄存器，RData2M又增加了两个转发来源，即HiDataW和LoDataW。

1. **测试用例**
   1. 测试程序

li $t0,-123

sb $t0,5($zero) #mux\_opr2：$t0转发自ALUOutW

lb $t1,5($zero)

addi $t2,$t1,234 #$t1必须暂停（然后转发自Write）

lbu $t3,5($zero)

and $t4,$t2,$t3 #暂停1个周期后，$t2转发自GPR内部，$t3转发自Write

li $s0,-123

sw $s0,36($zero) #mux\_opr2：$s0转发自ALUOutW

lw $t5,36($zero)

sub $t0,$t5,$zero #$t5必须暂停（然后转发自Write）

bltz $t0,label1 #$t0必须暂停（然后转发自ALUOutM）

lui $t0,946

label1: or $t1,$t4,$t0 #mux\_opr2：$t0转发自ALUOutM

mult $t1,$t0 #mux\_opr1：$t1转发自ALUOutM；mux\_opr2：$t2转发自ALUOutW

#mult指令阻塞5个时钟周期

mfhi $t6

mflo $t7

div $t5,$t2 #div指令阻塞10个时钟周期

mfhi $t6

mflo $t7

andi $t6,$t6,345 #mux\_opr1：$t6转发自HiDataW

mtlo $t6 #mux\_opr1：$t6转发自ALUOutM

mflo $t7

ori $ra,$t7,456 #mux\_opr1：$t7转发自LoDataM

jal label2

addiu $ra,$ra,567 #mux\_opr1：$ra转发自pc8M

label2: subu $t2,$ra,$t6 #mux\_opr1：$ra转发自ALUOutM

li $t3,678

divu $t2,$t3 #mux\_opr1：$t2转发自ALUOutW；mux\_opr2：$t3转发自ALUOutM

#div指令阻塞10个时钟周期

mult $t3,$zero #mux\_opr1：$t3转发自ALUOutW

#mult指令阻塞5个时钟周期

mfhi $ra

mtlo $t2

mflo $s1

xori $s0,$s1,345 #mux\_opr1：$s1转发自LoDataM

la $s0,label3

jr $s0 #$s0必须暂停（然后转发自ALUOutM）

add $s0,$s0,4 #$s0转发自ALUOutW

label3: sll $s2,$s1,6

sll $s2,$s1,7

sllv $s3,$s2,$s0 #mux\_opr1：$s2转发自ALUOutM；$s0转发自GPR内部

nor $s0,$s3,$s2 #mux\_opr1：$s3转发自ALUOutM；mux\_opr2：$s2转发自ALUOutW

addu $s4,$s0,$s2 #mux\_opr1：$s3转发自ALUOutM

li $s5,-1

mult $s4,$s5 #mux\_opr2：$s5转发自ALUOutM

mflo $s6

sh $s6,10($zero) #mux\_dwd：$s6转发自Write

slti $t5,$s2,-678

lhu $s7,10($zero)

bgtz $s7,label4 #$s7必须暂停2个周期（然后转发自Write）

srl $t5,$t5,8 #$t5转发自GPR内部

label4: sltiu $t8,$t5,-789 #mux\_opr1：$t5转发自ALUOutM

lw $t9,8($zero)

sb $t9,13($zero) #mux\_dwd：$t9转发自Write

sw $t9,16($zero) #mux\_opr2：$t9转发自Write

sh $t9,22($zero) #$t9转发自GPR内部

sltu $at,$t8,$t9

addiu $t8,$t8,1472583690

subu $t9,$t8,$t9 #mux\_opr1：$t8转发自ALUOutM

mult $t8,$t9 #mux\_opr1：$t8转发自ALUOutW；mux\_opr2：$t9转发自ALUOutM

#mult指令阻塞5个时钟周期

mfhi $t8

mflo $t9

divu $t8,$t9 #mux\_opr1：$t8转发自HiDataW；mux\_opr2：$t9转发自LoDataM

#mult指令阻塞5个时钟周期

j label5

mthi $zero

label5: mfhi $t4

sra $t9,$t4,8 #mux\_opr1：$t4转发自HiDataM

srlv $at,$at,$t9 #mux\_opr2：$t9转发自ALUOutM

xor $v0,$at,$t9 #mux\_opr1：$at转发自ALUOutM；mux\_opr2：$t9转发自ALUOutW

lw $v1,16($zero)

slt $k0,$v0,$v1 #暂停1个周期后，$v0转发自GPR内部，$v1转发自Write

sub $k0,$zero,655357

bgez $k0,label6 #$k0必须暂停（然后转发自ALUOutM）

la $fp,label6

jalr $k1,$fp #$fp必须暂停（然后转发自ALUOutM）

or $k0,$k0,$fp #mux\_opr2：$fp转发自ALUOutW

label6: srav $ra,$k0,$k1 #mux\_opr1：$k0转发自ALUOutM；mux\_opr2：$k1转发自pc8W

beq $ra,$ra,label7 #$ra必须暂停（然后转发自ALUOutM）

addi $a0,$ra,7426 #mux\_opr1：$ra转发自ALUOutW

label7: sh $a0,170($zero) #mux\_dwd：$a0转发自Write

ori $a1,$a0,467982 #mux\_opr1：$a0转发自ALUOutW

bne $a1,$zero,lab8 #$a1必须暂停（然后转发自ALUOutM）

lh $a2,170($zero)

lab8: xor $a3,$a2,$a0 #$a2必须暂停（然后转发自Write）

mtlo $a3 #mux\_opr1：$a3转发自ALUOutM

sub $t7,$a3,$a1 #mux\_opr1：$a3转发自ALUOutW

mfhi $t8

mflo $s0

sw $t7,148($zero) #$t7转发自GPR内部

bgtz $zero,label9

sra $t0,$t7,3

label9: li $v0,-1406

blez $v0,exit #$v0必须暂停（然后转发自ALUOutM）

addi $v0,$v0,1423 #mux\_opr1：$v0转发自ALUOutW

li $k0,985

sub $s5,$v0,$k0 #mux\_opr2：$k0转发自ALUOutM

blez $s5,exit #$s5必须暂停（然后转发自ALUOutM）

exit: sw $k0,68($zero) #k0转发自GPR内部

sw $s5,($zero)

* 1. 测试结果

$ 8 <= ffffff85

\*000000005 <= 85

$ 9 <= ffffff85

$10 <= 0000006f

$11 <= 00000085

$12 <= 00000005

$16 <= ffffff85

\*000000024 <= ffffff85

$13 <= ffffff85

$ 8 <= ffffff85

$ 8 <= 03b20000

$ 9 <= 03b20005

$14 <= 000da7c4

$15 <= 127a0000

$14 <= fffffff4

$15 <= ffffffff

$14 <= 00000150

$15 <= 00000150

$31 <= 000001d8

$31 <= 00003064

$31 <= 0000329b

$10 <= 0000314b

$11 <= 000002a6

$31 <= 00000000

$17 <= 0000314b

$16 <= 00003012

$16 <= 00003090

$16 <= 00003094

$18 <= 000c52c0

$18 <= 0018a580

$19 <= 58000000

$16 <= a7e75a7f

$20 <= a7ffffff

$21 <= ffffffff

$22 <= 58000001

\*00000000a <= 0001

$13 <= 00000000

$23 <= 00000001

$13 <= 00000000

$24 <= 00000001

$25 <= 00010000

\*00000000d <= 00

\*000000010 <= 00010000

\*000000016 <= 0000

$ 1 <= 00000001

$ 1 <= 57c50000

$ 1 <= 57c5d80a

$24 <= 57c5d80b

$25 <= 57c4d80b

$24 <= 1e17b9f7

$25 <= d9359079

$12 <= 00000000

$25 <= 00000000

$ 1 <= 57c5d80a

$ 2 <= 57c5d80a

$ 3 <= 00010000

$26 <= 00000000

$ 1 <= 00090000

$ 1 <= 0009fffd

$26 <= fff60003

$30 <= 00003138

$27 <= 00003138

$26 <= fff6313b

$31 <= ffffffff

$ 4 <= 00001d01

\*0000000aa <= 1d01

$ 1 <= 00070000

$ 1 <= 0007240e

$ 5 <= 00073d0f

$ 6 <= 00001d01

$ 7 <= 00000000

$15 <= fff8c2f1

$24 <= 00000000

$16 <= 00000000

\*000000094 <= fff8c2f1

$ 8 <= ffff185e

$ 2 <= fffffa82

$ 2 <= 00000011

\*000000044 <= fff6313b

\*000000000 <= ffffffff

1. **实验思考**
   1. lb等指令的数据扩展

Load系列的指令是要从存储器中读取数据，而与存储器相关的操作往往是整个CPU中延迟最长的，所以为了使流水线达到最大效率，我们就必须尽可能地均衡各级流水线的延迟（“木桶原理”）。

这样一来，如果把lb等指令的数据扩展模块放置在M级，则M级的延迟将明显大于其他级，从而大大降低了整个CPU的效率，基于这一考虑，我们应该把lb等指令的数据扩展模块放置在W级。

* 1. “延迟槽”vs.“乘除槽”

引入“延迟槽”主要是为了提高流水线的效率。我们规定跳转指令后面的一个时间片为“延迟槽”，位于“延迟槽”中的指令总是被执行，从而有效利用了一个时间片。

引入“乘除槽”同样可以提高流水线的效率，首先，乘法指令有5个周期的延迟，除法指令有10个周期的延迟，如果直接将整个流水线阻塞，则浪费太多时间片。所以，仅阻塞与乘除部件相关的指令，而保证后续其他指令总是正常执行，以提高效率。

* 1. 设计风格

上一次的实验我就已经在实验收获中提到了这个问题，我主要使用了include这种方法，另外在Forward和Stall单元中也用到了implement，都使得代码更加简洁。

具体到设计风格，诚如助教所提到的两种极具代表性的风格各有利弊，我比较倾向于第二种，因为它更符合我的整个设计思路，在增加指令或测试修改时能更快地对代码进行定位。