# P5-Verilog流水线CPU设计文档

## 一、设计说明

1. 处理器应支持MIPS-lite2指令集。

MIPS-lite2={ addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop }

2. 处理器为流水线设计。

## 二、数据通路设计

### 1. PC

#### (1) **模块接口定义**

表1 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| pc.v | module pc(  input clk,  input reset,  input [31:0] nextPC,  output reg [31:0] PC); |

#### (2) **接口说明**

表2 PC接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号  1：复位  0：无效 |
| 3 | pc\_en | I | PC更新使能信号 |
| 4 | nextPC | I | PC输入值 |
| 5 | PC | O | PC输出值 |

#### (3)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | reset有效时，PC被设置为0x00003000 |
| 2 | PC更新 | 时钟上升沿来临且pc\_en有效时，PC值更新为nextPC |

### 2. IM

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| im.v | module im(  input [31:0] PC,  output [31:0] instruction); |

#### (2) **接口说明**

表2 PC接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | PC | I | 32位PC值 |
| 2 | instruction | O | 取出32位指令值 |

#### (3)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 取指令 | 根据PC值取出指令 |

### 3. IF/ID

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IFID.v | module IFID(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] PC\_F,  input [31:0] Instr\_F,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] PC\_D,  output reg[31:0] PC4\_D,  output reg[31:0] PC8\_D,  output reg[31:0] Instr\_D  ); |

#### (2) **接口说明**

表2 PC接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 同步复位信号，将Instr\_D的值全部清零  1：复位  0：无效 |
| 3 | i\_en | I | 由暂停模块控制的使能信号 |
| 4 | PC\_F | I | PC在F级的值 |
| 5 | Instr\_F | I | Instr在F级的值 |
| 6 | PC\_D | O | PC在D级的值 |
| 7 | PC4\_D | O | PC+4在D级的值 |
| 8 | PC8\_D | O | PC+8在D级的值 |
| 9 | Instr\_D | O | Instr在D级的值 |

#### (3)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存IF/ID的流水线信号 |

### 4. NPC

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| npc.v | module npc(  input i\_Zero,  input [2:0] i\_branch,  input [25:0] i\_imm26,  input [31:0] i\_jr\_addr,  input [31:0] i\_PC,  input [31:0] i\_PC4\_D,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg [31:0] o\_nextPC  ); |

#### (2) **接口说明**

表2 PC接口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | zero | I | 判断PC是否满足beq指令跳转条件  1：满足  0：不满足 |
| 2 | branch | I | 判断当前指令是否需要跳转 |
| 3 | imm26 | I | j/offset |
| 4 | jr\_addr | I | jr指令跳转地址 |
| 5 | PC | I | PC输入值 |
| 6 | PC4\_D | I | PC+4输入值 |
| 7 | nextPC | O | nextPC计算结果 |

#### (3)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 计算PC的下一个值 | 计算PC的下一个值 |

### 5. GRF

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| grf.v | module grf(  input clk,  input reset,  input WE,  input [4:0] A1,  input [4:0] A2,  input [4:0] A3,  input [31:0] WD,  input [31:0] PC,  output [31:0] RD1,  output [31:0] RD2); |

#### (2) **接口说明**

表3 GRF端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| 3 | A1 | 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RD1 |
| 4 | A2 | 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RD2 |
| 5 | A3 | 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 |
| 6 | WD | I | 32位写入数据 |
| 7 | WE | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| 8 | PC | I | 当前PC值 |
| 9 | RD1 | O | 输出A1指定的寄存器的32位数据 |
| 10 | RD2 | O | 输出A2指定的寄存器的32位数据 |

#### (3)功能定义

表4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | Reset有效时，将32个寄存器中的值全部清零 |
| 2 | 读数据 | 读出A1, A2地址对应寄存器中所存储数据到RD1, RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |

### 6. EXT

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ext.v | module ext(  input [15:0] in,  input [1:0] EXTOp,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg [31:0] out  ); |

#### (2) **接口说明**

表9 EXT端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | in | I | 16位待扩展数据 |
| 2 | EXTop | I | 扩展操作信号  00：进行无符号扩展  01：进行符号扩展  10：加载至高位 |
| 3 | out | O | 32位输出数据 |

#### (3)功能定义

表10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 无符号扩展 | 将16位立即数Imm16无符号拓展至32位输出Imm32 |
| 2 | 符号扩展 | 将16位立即数Imm16符号拓展至32位输出Imm32 |
| 3 | 加载至高位 | 将16位立即数Imm16加载至高位后32位输出Imm32 |

### 7. EQUAL

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| equal.v | module equal(  input [31:0] RD1,  input [31:0] RD2,  output equal  ); |

#### (2) **接口说明**

表9 EXT端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | RD1 | I | 待比较数A |
| 2 | RD2 | I | 待比较数B |
| 3 | equal | O | 比较结果 |

#### (3)功能定义

表10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 判断是否相等 | A==B则equal=1 |

### 8. ID/EX

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IDEX.v | module IDEX(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_D,  input [31:0] PC\_D,  input [31:0] PC4\_D,  input [31:0] PC8\_D,  input [31:0] RD1\_D,  input [31:0] RD2\_D,  input [31:0] SignImm\_D,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_E,  output reg[31:0] PC\_E,  output reg[31:0] PC4\_E,  output reg[31:0] PC8\_E,  output reg[31:0] RD1\_E,  output reg[31:0] RD2\_E,  output reg[31:0] SignImm\_E  ); |

#### (2)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存ID/EX的流水线信号 |

### 9. ALU

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| alu.v | module alu(  input [31:0] A,  input [31:0] B,  input [2:0] ALUOp,  output [31:0] Result,  ); |

#### (2) **接口说明**

表5 ALU端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | A | I | 参与ALU计算的第一个32位数据 |
| 2 | B | I | 参与ALU计算的第二个32位数据 |
| 3 | ALUOp | I | ALU功能的选择信号：  000：ALU进行与运算  001：ALU进行或运算  010：ALU进行加法运算  011：ALU进行减法运算 |
| 4 | Result | O | ALU的计算结果 |

#### (3)功能定义

表6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 与运算 |  |
| 2 | 或运算 |  |
| 3 | 加运算 |  |
| 4 | 减运算 |  |

### 10. EX/MEM

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXMEM.v | module EXMEM(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_E,  input [31:0] PC\_E,  input [31:0] PC4\_E,  input [31:0] PC8\_E,  input [31:0] ALUOut\_E,  input [31:0] WD\_E,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_M,  output reg[31:0] PC\_M,  output reg[31:0] PC4\_M,  output reg[31:0] PC8\_M,  output reg[31:0] ALUOut\_M,  output reg[31:0] WD\_M  ); |

#### (2)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存EX/MEM的流水线信号 |

### 11. DM

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| dm.v | module dm(  input clk,  input reset,  input MemWrite,  input MemRead,  input [31:0] addr,  input [31:0] WriteData,  input [31:0] PC,  output [31:0] ReadData); |

#### (2) **接口说明**

表7 DM端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | clk | I | 时钟信号 |
| 2 | reset | I | 异步复位信号，将DM中的值全部清零  1：复位  0：无效 |
| 3 | MemRead | I | 写使能信号  1：可向DM中写入数据  0：无效 |
| 4 | MemWrite | I | 读使能信号  1：可读取DM中数据  0：无效 |
| 5 | addr | I | 32位地址输入信号，对DM指定地址进行读写操作 |
| 6 | WriteData | I | 32位写入数据 |
| 7 | PC | I | 当前PC值 |
| 8 | ReadData | O | 32位输出数据 |

#### (3)功能定义

表8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 复位 | Reset有效时，将DM中的值全部清零 |
| 2 | 读操作 | 读出addr地址对应存储数据到ReadData |
| 3 | 写操作 | 当时钟上升沿来临时，将WriteData写入addr地址对应位置 |

### 12. MEM/WB

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MEMWB.v | module MEMWB(  input clk,  input reset,  input i\_en,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  input [31:0] Instr\_M,  input [31:0] PC\_M,  input [31:0] PC4\_M,  input [31:0] PC8\_M,  input [31:0] ALUOut\_M,  input [31:0] RD\_M,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output reg[31:0] Instr\_W,  output reg[31:0] PC\_W,  output reg[31:0] PC4\_W,  output reg[31:0] PC8\_W,  output reg[31:0] ALUOut\_W,  output reg[31:0] RD\_W  ); |

#### (2)功能定义

表3 PC功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 流水线寄存器 | 保存MEM/RB的流水线信号 |

### 8. MUX

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| mux.v | module mux\_2\_32(  input select,  input [31:0] mi1,  input [31:0] mi2,  output [31:0] mo  );  module mux\_4\_32(  input [1:0] select,  input [31:0] mi1,  input [31:0] mi2,  input [31:0] mi3,  input [31:0] mi4,  output [31:0] mo  );  module mux\_3\_5(  input [1:0] select,  input [4:0] mi1,  input [4:0] mi2,  output [4:0] mo  );  module forward(  input [2:0] select,  input [31:0] mi1,  input [31:0] mi2,  input [31:0] mi3,  input [31:0] mi4,  input [31:0] mi5,  input [31:0] mi6,  input [31:0] mi7,  input [31:0] mi8,  output [31:0] mo  ); |

#### (2) **接口说明**

表9 EXT端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | select | I | 选择信号 |
| 2 | mi（多个）x | I | 输入信号 |
| 3 | mo | O | 输出数据 |

#### (3)功能定义

表10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 选择信号 | 根据select信号选择输入输出 |

## 三、控制器设计与冲突判断

## 

### 1. Control模块**端口与功能说明**

表11 Control模块端口说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 信号 | 方向 | 描述 |
| 1 | Op [5:0] | I | 6位控制信号 |
| 2 | Func [5:0] | I | 6位控制信号 |
| 3 | RegDst [1:0] | O | 00：将Rt作为GRF的A3写入地址  01：将Rd作为GRF的A3写入地址  10：将h1f作为GRF的A3写入地址 |
| 4 | ALUSrc | O | 0：将RD2作为参与ALU计算的第二个数据  1：将EXT输出作为参与ALU计算的第二个数据 |
| 5 | MemToReg [1:0] | O | 00：将ALU计算结果写入GRF  01：将DM输出值写入GRF  10：将填充到高位的 16位立即数写入GRF |
| 6 | RegWrite | O | 0：GRF写使能信号无效  1：GRF写使能信号有效 |
| 7 | MemRead | O | 0：DM读使能信号无效  1：DM读使能信号有效 |
| 8 | MemWrite | O | 0：DM写使能信号无效  1：DM写使能信号有效 |
| 9 | Branch [2:0] | O | 000：默认操作  001：当前指令为beq指令  010：当前指令为jar指令  011：当前指令为jr指令  101：当前指令为j指令 |
| 10 | ExtOp [1:0] | O | 00：进行无符号扩展  01：进行符号扩展  10：lui指令计算 |
| 11 | ALUOp [2:0] | O | ALU功能的选择信号：  000：ALU进行与运算  001：ALU进行或运算  010：ALU进行加法运算  011：ALU进行减法运算 |

### 2. Control模块**真值表**

表12 Control模块**真值表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | addu | subu | ori | lw | sw | beq | lui | jal | j | jr |
| Op字段 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000010 | 000000 |
| Func字段 | 100001 | 100011 | xxxxxx | | | | | | | 001000 |
| RegDst | 01 | 01 | 00 | 00 | 00 | 00 | 00 | 10 | 00 | 01 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| MemToReg | 00 | 00 | 00 | 01 | 00 | 00 | 10 | 11 | 00 | 00 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| MemRead | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| Branch | 000 | 000 | 000 | 000 | 000 | 001 | 000 | 010 | 101 | 011 |
| ExtOp | 00 | 00 | 01 | 00 | 00 | 00 | 10 | 00 | 00 | 00 |
| ALUOp | 010 | 011 | 001 | 010 | 010 | 011 | 111 | 111 | 111 | 010 |

### 3. 分布式Control模块

#### (1) **模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| mips.v | controller C\_ID (  .op(Instr\_D[`i\_op]),  .Fu(Instr\_D[`i\_func]),  .Branch(con\_o\_Branch\_3),  .EXTOp(con\_o\_EXTOp\_2)  );  controller C\_EX (  .op(Instr\_E[`i\_op]),  .Fu(Instr\_E[`i\_func]),  .ALUOp(con\_o\_ALUOp\_3),  .ALUSrcA(con\_o\_ALUSrcA),  .ALUSrcB(con\_o\_ALUSrcB)  );  controller C\_MEM (  .op(Instr\_M[`i\_op]),  .Fu(Instr\_M[`i\_func]),  .MemRead(con\_o\_MemRead),  .MemWrite(con\_o\_MemWrite)  );  controller C\_WB (  .op(Instr\_W[`i\_op]),  .Fu(Instr\_W[`i\_func]),  .RegDst(con\_o\_RegDst\_2),  .RegWrite(con\_o\_RegWrite),  .MemtoReg(con\_o\_MemtoReg\_2)  ); |

#### (2) **总模块接口定义**

表4 PC**模块接口定义**

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| controller.v | module controller(  input [5:0] op,  input [5:0] Fu,  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  output [2:0] ALUOp,  output [1:0] RegDst,  output ALUSrcA,  output ALUSrcB,  output RegWrite,  output MemRead,  output MemWrite,  output [1:0] MemtoReg,  output [2:0] Branch,  output [1:0] EXTOp  ); |

### 4. 暂停与转发机制

见附表

## 四、测试程序

### 1. test branch代码

module *tb*;

*// Inputs*

    reg clk;

    reg reset;

*// Instantiate the Unit Under Test (UUT)*

    mips uut (

        .clk(clk),

        .reset(reset)

    );

    initial begin

*// Initialize Inputs*

        clk = 0;

        reset = 1;

*// Wait 100 ns for global reset to finish*

        #100;

        reset = 0;

*// Add stimulus here*

    end

    always #10 clk = ~clk;

endmodule

### 2. mars测试程序与结果

#### (1) **暂停测试**

表9 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 冲突类型 | MIPS代码 | 预期结果 |
| Beq 与 cal\_r/ cal\_i/ load冲突 | .text  ori $3,$0,123  addu $4,$0,$3  beq $4,$3,next1  nop  ori $9,$0,99  next1:  addu $5,$4,$0  ori $6,$0,123  beq $6,$5,next2  nop  ori $9,$0,88  next2:  sw $6,0($0)  lw $7,0($0)  lw $8,0($0)  beq $7,$8,next3  nop  ori $9,$0,77  next3:  ori $9,$0,100  3403007b  00032021  10830002  00000000  34090063  00802821  3406007b  10c50002  00000000  34090058  ac060000  8c070000  8c080000  10e80002  00000000  3409004d  34090064 | @00003000: $ 3 <= 0000007b  @00003004: $ 4 <= 0000007b  @00003014: $ 5 <= 0000007b  @00003018: $ 6 <= 0000007b  @00003028: \*00000000 <= 0000007b  @0000302c: $ 7 <= 0000007b  @00003030: $ 8 <= 0000007b  @00003040: $ 9 <= 00000064 |
| Cal\_r/ cal\_i 与load冲突 | .text  ori $8,$0,123  sw $8,0($0)  lw $9,0($0)  addu $10,$9,$9  lw $10,0($0)  ori $11,$10,0  sw $11,1024($0)  3408007b  ac080000  8c090000  01295021  8c0a0000  354b0000  ac0b0400 | @00003000: $ 8 <= 0000007b  @00003004: \*00000000 <= 0000007b  @00003008: $ 9 <= 0000007b  @0000300c: $10 <= 000000f6  @00003010: $10 <= 0000007b  @00003014: $11 <= 0000007b  @00003018: \*00000400 <= 0000007b |
| load\_rs 与load\_rt冲突 | .text  ori $3,$0,12  sw $3,0($0)  lw $4,0($0)  sw $3,0($4)  lw $5,0($0)  lw $6,0($4)  3403000c  ac030000  8c040000  ac830000  8c050000  8c860000 | @00003000: $ 3 <= 0000000c  @00003004: \*00000000 <= 0000000c  @00003008: $ 4 <= 0000000c  @0000300c: \*0000000c <= 0000000c  @00003010: $ 5 <= 0000000c  @00003014: $ 6 <= 0000000c |
| store\_rs 与load rt冲突 | .text  ori $3,$0,16  sw $3,0($0)  lw $4,0($0)  sw $4,4($0)  lw $6,4($0)  sw $6,8($0)  lw $7,8($0)  sw $7,12($0)  lw $8,12($0)  sw $8,0($8)  lw $9,0($8)  sw $9,4($9)  lw $10,4($9)  addu $11,$10,$10  34030010  ac030000  8c040000  ac040004  8c060004  ac060008  8c070008  ac07000c  8c08000c  ad080000  8d090000  ad290004  8d2a0004  014a5821 | @00003000: $ 3 <= 00000010  @00003004: \*00000000 <= 00000010  @00003008: $ 4 <= 00000010  @0000300c: \*00000004 <= 00000010  @00003010: $ 6 <= 00000010  @00003014: \*00000008 <= 00000010  @00003018: $ 7 <= 00000010  @0000301c: \*0000000c <= 00000010  @00003020: $ 8 <= 00000010  @00003024: \*00000010 <= 00000010  @00003028: $ 9 <= 00000010  @0000302c: \*00000014 <= 00000010  @00003030: $10 <= 00000010  @00003034: $11 <= 00000020 |
| jr\_rs 与cal\_r/ cal\_i / load冲突 | .text  ori $3,$0,0x00003010 #16  jr $3 #4  nop#8  ori $4,$0,123 #12  ori $5,$0,0x00003010 #16  ori $6,$0,24#24  addu $7,$5,$6 #24  jr $7 #28  nop  ori $1,$0,123  addu $9,$7,$7  34033010  00600008  00000000  3404007b  34053010  34060018  00a63821  00e00008  00000000  3401007b  00e74821 | @00003000: $ 3 <= 00003010  @00003010: $ 5 <= 00003010  @00003014: $ 6 <= 00000018  @00003018: $ 7 <= 00003028  @00003028: $ 9 <= 00006050 |

#### (2) **转发测试**

表9 EXT端口说明

|  |  |  |
| --- | --- | --- |
| 转发类型 | MIPS代码 | 预期结果 |
| E->D | .text  lui $3, 203  lui $4, 203  beq $3,$4 next  ori $5,123  ori $6,123  next:  jal end  ori $8,$31,0  ori $9,$31,0  end:  ori $10,$31,0  3c0300cb  3c0400cb  10640002  34a5007b  34c6007b  0c000c08  37e80000  37e90000  37ea0000 | @00003000: $ 3 <= 00cb0000  @00003004: $ 4 <= 00cb0000  @0000300c: $ 5 <= 0000007b  @00003014: $31 <= 0000301c  @00003018: $ 8 <= 0000301c  @00003020: $10 <= 0000301c |
| M->D | .text  ori  $3, $0,203  addu $4,$0, 203 #要暂停  beq $3,$4 next  ori $5,123  ori $6,123  next:  jal end  ori $8,$31,0  ori $9,$31,0  end:  ori $10,$31,0  340300cb  3c010000  342100cb  00012021  10640002  34a5007b  34c6007b  0c000c0a  37e80000  37e90000  37ea0000 | @00003000: $ 3 <= 000000cb  @00003004: $ 1 <= 00000000  @00003008: $ 1 <= 000000cb  @0000300c: $ 4 <= 000000cb  @00003014: $ 5 <= 0000007b  @0000301c: $31 <= 00003024  @00003020: $ 8 <= 00003024  @00003028: $10 <= 00003024 |
| W->D | .text  ori $12, $0,321  ori $13, $0,321  ori $3, $0,4  sw $12, 0($3)  sw $13, 4($3)  lw $4, 4($3)  lw $5, 0($3)  beq $4,$5,end  ori $8,$0,123  ori $9,$0,123  end:  ori $31,$0,0  340c0141  340d0141  34030004  ac6c0000  ac6d0004  8c640004  8c650000  10850002  3408007b  3409007b  341f0000 | @00003000: $12 <= 00000141  @00003004: $13 <= 00000141  @00003008: $ 3 <= 00000004  @0000300c: \*00000004 <= 00000141  @00003010: \*00000008 <= 00000141  @00003014: $ 4 <= 00000141  @00003018: $ 5 <= 00000141  @00003020: $ 8 <= 0000007b  @00003028: $31 <= 00000000 |
| M->E/ W->E | .text  ori $3,$0,123  ori $4,$3,0  addu $5,$4,$3  addu $6,$5,$4  addu $7,$5,$6  subu $8,$7,$6  3403007b  34640000  00832821  00a43021  00a63821  00e64023 | @00003000: $ 3 <= 0000007b  @00003004: $ 4 <= 0000007b  @00003008: $ 5 <= 000000f6  @0000300c: $ 6 <= 00000171  @00003010: $ 7 <= 00000267  @00003014: $ 8 <= 000000f6 |
| W->M | .text  ori $3, $0,12  ori $4,$0,123  sw $4,0($3)  addu $5,$4,$0  sw $5,4($3)  3403000c  3404007b  ac640000  00802821  ac650004 | @00003000: $ 3 <= 0000000c  @00003004: $ 4 <= 0000007b  @00003008: \*0000000c <= 0000007b  @0000300c: $ 5 <= 0000007b  @00003010: \*00000010 <= 0000007b |

## 五、思考题

### 1. 在线测试相关说明

#### 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。

见“测试程序”部分