编号： 09-021830122-贾志军

**计 算 机 组 成 原**

**理 课 程 设 计**

|  |  |
| --- | --- |
| 题 目 | 五级流水线处理器设计() |

**（实现45条指令）**

|  |  |
| --- | --- |
| 学生姓名 | 贾志军 |
| 学 号 | 021830122 |
| 学 院 | 计算机科学与技术学院 |
| 专 业 | 计算机科学与技术 |
| 班 级 | 1618101 |
| 指导教师 | 施慧彬 |

二〇二零年六月

目录

[1．课程设计说明 3](#_Toc45027684)

[1.1 指令实现 3](#_Toc45027685)

[1.2 完成情况 3](#_Toc45027686)

[1.3 流水线设计说明 3](#_Toc45027687)

[2．模块化和层次化设计说明 4](#_Toc45027688)

[2.1 总电路设计图 4](#_Toc45027689)

[3.具体模块化定义 7](#_Toc45027691)

[3.1流水段寄存器的定义 7](#_Toc45027692)

[3.1..2 IF/ID寄存器 7](#_Toc45027693)

[3.1..4 EX/MEM寄存器 9](#_Toc45027694)

[3.2其他模块定义 10](#_Toc45027695)

[3.2.1 PC 模块定义 11](#_Toc45027696)

[3.2.2 NPC 模块定义 11](#_Toc45027697)

[3.2.3 ALU 模块定义 13](#_Toc45027698)

[3.2.4 dm 模块定义 14](#_Toc45027699)

[3.2.5 im 模块定义 14](#_Toc45027700)

[3.2.6 regist 模块定义 15](#_Toc45027701)

[3.2.7 ext 模块定义 16](#_Toc45027702)

[3.3Loaduse冒险 17](#_Toc45027703)

[3.3.1基本描述 17](#_Toc45027704)

[3.3.2Loaduse冒险检测 17](#_Toc45027705)

[3.3.3Loaduse解决方法 18](#_Toc45027706)

[3.4其他冒险 18](#_Toc45027707)

[3.4.1 ByPass模块(1）基本描述 18](#_Toc45027708)

[3.4.2 ByPass\_M(1）基本描述 19](#_Toc45027709)

[3.4.2 ByPass\_T(1）基本描述 20](#_Toc45027710)

[3.4.2 ByPass\_X(1）基本描述 20](#_Toc45027711)

[选择ALU的运算结果和Lo寄存器的值，进行转发 20](#_Toc45027712)

[4．Modelsim运算结果 22](#_Toc45027713)

[4.1波形图 22](#_Toc45027714)

[4.2GPR寄存器值 23](#_Toc45027715)

[4.4CPR寄存器值 23](#_Toc45027716)

[4.5Hi寄存器 23](#_Toc45027717)

[4.6Lo寄存器 24](#_Toc45027718)

[5.ISE运算结果 25](#_Toc45027719)

[5.1程序图 25](#_Toc45027720)

[5.2ISE速度说明 25](#_Toc45027721)

[附录一 26](#_Toc45027722)

[附录二 verilog代码 32](#_Toc45027723)

[附录 3 48](#_Toc45027724)

# 1．课程设计说明

## 1.1 指令实现

实现指令如下：

addu, addi, subu, slt, and, nor, or, xor, sll, srl, addiu, beq, bne, lw, sw, lui, j, sltu, jalr, jr, rs, sllv, sra, srav, srlv, slti, sltiu, bgez, bgtz, blez, bltz, lb, lbu, sb, andi, ori, xori, jal, mflo, mfhi, mtlo, mthi, mult共 45 条指令

## 1.2 完成情况

1 处理了绝大多数可能的数据冒险，解决了分支指令和跳转指令所引起的控 制冒险，这些采用了阻塞，转发技术实现，完美运行老师所给的测试指令。

2 采用五级流水线设计

3 可以在龙芯的开发板上进行测试

## 1.3 流水线设计说明

处理了一条指令的执行过程被分成五个阶段，每个阶段由相应的功能部件完 成。如果将各阶段看成相应的流水段，则指令的执行过程就构成了一条指令流水 线。

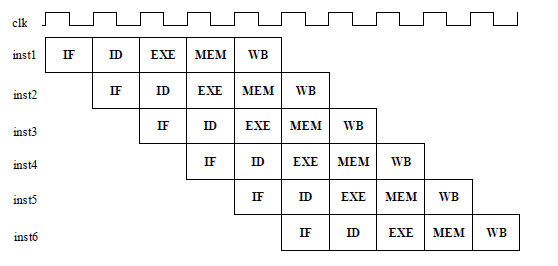
（1）Ifetch (取指) : 取指令并计算 PC+4 。

（2）Reg/Dec (取数和译码) : 取数同时译码 。

（3）Exec (执行) : 计算内存单元地址 。

（4）Mem (读存储器) : 从数据存储器中读 。

（5）Wr(写寄存器): 将数据写到寄存器中。

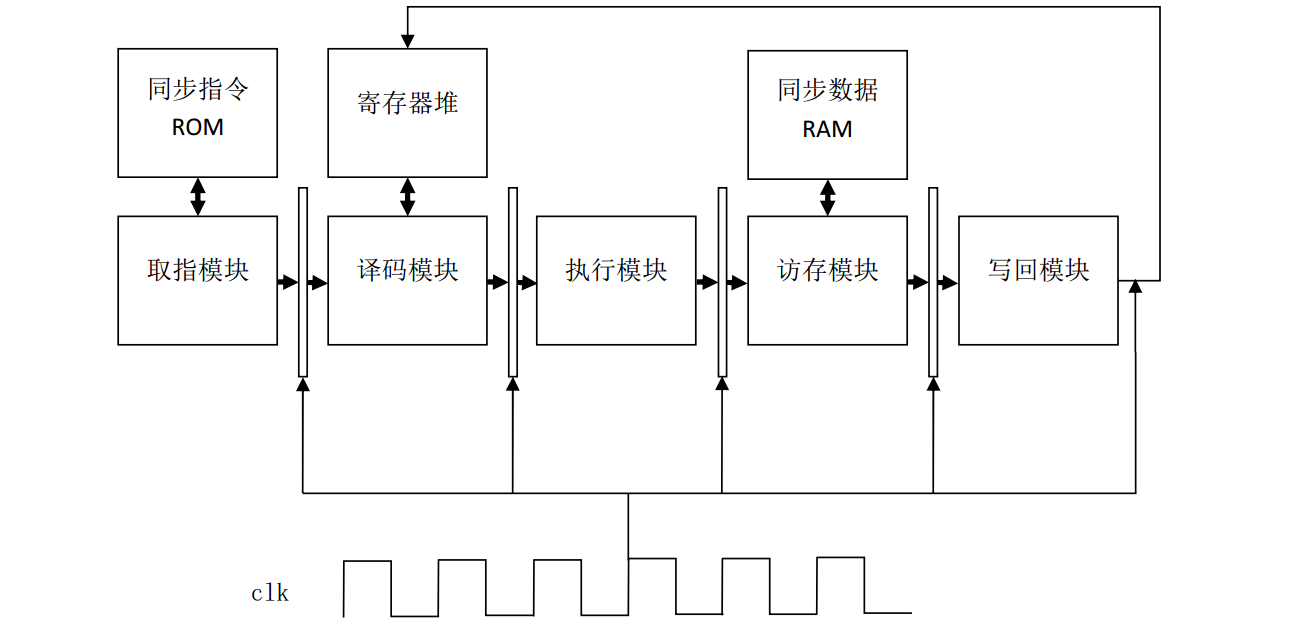


# 2．模块化和层次化设计说明

## 2.1 总电路设计图

## 

2.2 模块化设计图





2.4设计说明

电路设计使用模块化和层次化设计，根据五级流水线处理器的特点，首先是流水段寄存器，分别是IF/ID,ID/EX,EX/MEM,MEM/WR，同时为了处理转发，数据冒险，控制冒险，将各个部件整合到 了三个模块，分别是BYPASS,BYPASS\_T,BYPASS\_M,BYPASS\_X用来处理转发，Loaduse模块用来处理Loaduse冒险，还有一些模块，ALU模块ALU\_P,寄存器模块RFile\_P,主存模块Datamem，跨展模块EXT\_P，NPC\_P,PC\_P,数据选择器MUX\_A,MUX\_B.以上就是cpu 主要的大模块，把他们连接起来就是 pipeline\_cpu。同时还有一些小部件，他们都整合 到主要模块中。

# 3.具体模块化定义

## 3.1流水段寄存器的定义

3.1.1模块说明

流水段寄存器，时钟信号上升沿时把信号传给下一流水段寄存器

### 3.1..2 IF/ID寄存器

（1）模块介绍

时钟信号上升沿时把信号从if阶段传给id阶段

（2）接口定义

|  |  |  |
| --- | --- | --- |
| 信号 | I/O | 简介 |
| clk | I | 时钟 |
| [31:0] pc | I | 当前指令 |
| [31:0]instructions | I | 指令 |
| loaduse | I | loaduse冒险 |
| xiaoc | O | 是否阻塞该指令 |
| [5:0] op\_id | O | id阶段操作码 |
| [4:0] Rs\_id | O | id阶段指令Rs寄存器值 |
| [4:0]Rt\_id | O | id阶段指令Rt寄存器值 |
| [4:0]Rd\_id | O | id阶段指令Rd寄存器 |
| [5:0]func\_id | O | id阶段指令func字段 |
| [4:0]shamt\_id | O | id阶段指令shamt字段 |
| [15:0]imm16\_id | O | id阶段指令立即数字段 |
| [25:0]Target\_id | O | id阶段指令Target字段 |
| xiaoc\_id | O | id阶段指令阻塞信号 |
| [31:0]pc\_id | O | id阶段指令 |

3.1..3 ID/EX寄存器

（1）模块介绍

时钟信号上升沿时把信号从id阶段传给ex阶段

（2）接口定义

|  |  |  |
| --- | --- | --- |
| 信号 | I/O | 简介 |
| clk | I | 时钟信号 |
| [31:0]pc\_id | I | id阶段指令 |
| xiaoc\_id | I | id阶段阻塞指令 |
| loaduse | I | loaduse冒险 |
| [31:0]busA\_id | I | id阶段Rs寄存器值 |
| [31:0]busB\_id | I | id阶段Rt寄存器值 |
| [31:0]Lo\_out\_id | I | id阶段Lo寄存器值 |
| [31:0]Hi\_out\_id | I | id阶段Hi寄存器值 |
| [31:0]CPR\_out\_id | I | id阶段CPR寄存器值 |
| [5:0] op\_id | I | id阶段指令操作码 |
| [5:0]func\_id | I | id阶段指令func字段 |
| [4:0]Rs\_id | I | id阶段指令Rs寄存器字段 |
| [4:0]Rt\_id | I | id阶段指令Rt寄存器字段 |
| [4:0]Rd\_id | I | id阶段指令Rd寄存器字段 |
| [4:0]shamt\_id | I | id阶段指令shamt字段 |
| [15:0]imm16\_id | I | id阶段指令imm16字段 |
| [5:0]op | O | ex阶段操作码 |
| [5:0]func | O | ex阶段func字段 |
| [4:0] ALUctr\_ex | O | ex阶段ALU控制信号 |
| [31:0]busA\_ex | O | ex阶段Rs寄存器值 |
| [31:0]busB\_ex | O | ex阶段Rt寄存器值 |
| loaduse\_ex | O | ex阶段loaduse冒险 |
| [4:0]shamt\_ex | O | ex阶段shamt字段 |
| [4:0]Rs\_ex | O | ex阶段Rs寄存器值 |
| [4:0]Rt\_ex | O | ex阶段Rt寄存器值 |
| [4:0]Rd\_ex | O | ex阶段Rd寄存器 |
| ExtOp\_ex | O | ex阶段符合扩展信号 |
| RegDst\_ex | O | ex阶段写寄存器选择信号 |
| MemRead\_ex | O | ex阶段读主存信号 |
| imm16\_ex | O | ex阶段立即数 |
| ALUSrc\_ex | O | ex阶段ALUSrc信号 |
| xiaoc\_ex | O | ex阶段阻塞信号 |
| pc\_ex | O | ex阶段指令 |
| Lo\_out\_ex | O | ex阶段Lo寄存器的值 |
| Hi\_out\_ex | O | ex阶段Hi寄存器的值 |
| CPR\_out\_ex | O | ex阶段CPR寄存器的值 |

### 3.1..4 EX/MEM寄存器

（1）模块介绍

时钟信号上升沿时把信号从ex阶段传给mem阶段

（2）接口定义

|  |  |  |
| --- | --- | --- |
| 信号 | I/O | 简介 |
| RegWr\_me\_j | I | me阶段写寄存器信号 |
| [4:0]Rw\_me\_j | I | me阶段写寄存器 |
| [4:0]Rt\_ex | I | ex阶段指令Rt寄存器 |
| [31:0]Result\_me\_j | I | me阶段ALU结果 |
| [31:0]pc\_ex | I | ex阶段指令 |
| xiaoc\_ex | I | ex阶段阻塞信号 |
| zero\_ex | I | ex阶段ALU结果零信号 |
| [31:0]Result\_ex | I | ex阶段ALU结果 |
| [31:0]Result\_next\_ex | I | ex阶段Lo寄存器值 |
| [31:0]busB\_ex | I | ex阶段Rs寄存器值 |
| loaduse\_ex | I | ex阶段loaduse冒险信号 |
| [4:0]Rw\_ex | I | ex阶段写寄存器 |
| [5:0]op\_ex | I | ex阶段操作码 |
| [5:0]func\_ex | I | ex阶段func字段 |
| [4:0]Rs\_ex | I | ex阶段Rs寄存器 |
| [4:0]Rd\_ex | I | ex阶段Rd寄存器 |
| loaduse\_me | O | me阶段loaduse冒险信号 |
| [5:0]op\_me | O | me阶段操作码 |
| [5:0]func\_me | O | me阶段func字段 |
| [4:0]Rs\_me | O | me阶段Rs寄存器 |
| [4:0]Rt\_me | O | me阶段Rt寄存器 |
| [4:0]Rd\_me | O | me阶段Rd寄存器 |
| zero\_me | O | me阶段ALU零信号 |
| [31:0]Result\_me | O | me阶段ALU结果 |
| [31:0]busB\_me | O | me阶段Rt寄存器值 |
| [4:0]Rw\_me | O | me阶段写寄存器 |
| RegWr\_me | O | me阶段写寄存器信号 |
| sb\_me | O | me阶段sb指令信号 |
| MemWr\_me | O | me阶段写主存 |
| xiaoc\_me | O | me阶段阻塞 |
| [31:0]pc\_me | O | me阶段信号 |
| [31:0]Result\_next\_me | O | me阶段Lo寄存器值 |
| CPR\_wr\_me | O | me阶段写CPR信号 |
| Hi\_wr\_me | O | me阶段写Hi信号 |
| Lo\_wr\_me | O | me阶段写Lo信号 |
| Hi\_Lo\_wr\_me | O | me阶段同时写HiLo信号 |

3.1..5 MEM/WR寄存器

（1）模块介绍

时钟信号上升沿时把信号从MEM阶段传给WR阶段

（2）接口定义

|  |  |  |
| --- | --- | --- |
| 信号 | I/O | 简介 |
| clk | I | 时钟周期 |
| [31:0]pc\_me | I | me阶段指令 |
| xiaoc\_me | I | me阶段阻塞指令 |
| loaduse\_me | I | me阶段loaduse冒险信号 |
| [5:0]op\_me | I | me阶段操作码 |
| [5:0]func\_me | I | me阶段func字段 |
| [4:0]rd\_me | I | me阶段Rd寄存器 |
| [31:0]Do\_me | I | me阶段主存读的数据 |
| [31:0]Result\_me | I | me阶段ALU阶段 |
| [31:0]Result\_next\_me | I | me阶段Lo寄存器值 |
| [4:0]Rw\_me | I | me阶段写寄存器 |
| CPR\_wr\_me | I | me阶段写CPR信号 |
| Hi\_wr\_me | I | me阶段Hi写信号 |
| Lo\_wr\_me | I | me阶段Lo写信号 |
| Hi\_Lo\_wr\_me | I | me阶段HiLo写信号 |
| RegWr\_me | I | me阶段写寄存器信号 |
| loaduse\_wr | O | wr阶段loaduse冒险 |
| [5:0]op\_wr | O | wr阶段操作码 |
| [5:0]func\_wr | O | wr阶段func字段 |
| [4:0]Rd\_wr | O | wr阶段Rd寄存器 |
| [4:0]Do\_wr | O | wr阶段主存值 |
| [4:0]Result\_wr | O | wr阶段ALU结果 |
| [4:0]Rw\_wr | O | wr阶段写寄存器值 |
| MemtoReg\_wr | O | wr阶段主存到寄存器 |
| Regwr\_wr | O | wr阶段写寄存器值 |
| xiaoc\_wr | O | wr阶段阻塞信号 |
| pc\_wr | O | wr阶段指令 |
| CPR\_wr | O | wr阶段cpr写信号 |
| Hi\_wr | O | wr阶段Hi写信号 |
| Lo\_wr | O | wr阶段Lo写信号 |
| Hi\_Lo\_wr | O | wr阶段Hi\_Lo写信号 |
| [31:0]Result\_next\_wr | O | wr阶段Lo寄存器值 |

## 3.2其他模块定义

### 3.2.1 PC 模块定义

（1）基本描述

PC 主要功能是完成输出当前指令地址。复位后，PC 指向 0x0000\_0034，此处为第一条指令的地址。

（2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.1 PC接口的模块 |  |
| 信号名 | 方向 | 描述 |
| [31:0] NPC | I | 输入的指令地址 |
| clk | I | 时钟信号 |
| rst | I | 复原信号 |
| loaduse | I | Loaduse冒险 |
| [31:0] PC | O | 输出当前指令地址 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | 表1.2 PC接口的功能 | |  | |
| 序号 | | | | 功能名称 | | 功能描述 | |
| 1 | | | | 复位 | | rst=1时地址复位为0x0000 0034 | |
| 2 | | | | 输出指令地址 | | 时钟到来是，将NPC赋给PC | |
| 3 | | | | loaduse冒险 | | 如果loaduse=1，pc值不更新 | |
|  |  | 3.2.2 NPC 模块定义 | |  | |  | |  |
|  |  | （1）基本描述 | |  | |  | |  |

NPC 主要功能是根据当前指令是否为跳转分支指令，输出下一条指令的地址。该模块

调用了 signext 模块。

（2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.3 NPC接口的模块 |  |
| 信号名 | 方向 | 描述 |
| [31:0] ID\_CPR\_out\_14 | I | ID阶段14号CPR寄存器值 |
| [4:0]Rd\_ex | I | Ex阶段Rd寄存器值 |
| [4:0] Rs\_id | I | id阶段Rs寄存器 |
| [31:0]Result\_ex | I | Ex阶段ALU运行结果 |
| [31:0] ID\_PC | I | Id阶段当前指令 |
| [31:0]IF\_PC | I | IF阶段当前指令 |
| [31:0]ID\_busA | I | Id阶段Rs寄存器值 |
| [31:0]ID\_busB | I | Id阶段Rt寄存器值 |
| J\_Q\_1 | I | 是否执行延迟槽指令 |
| [31:0]NPC\_J\_1 | I | 延迟槽指令的上一条指令 |
| [31:0]B\_ALU | I | ALU运算的值 |
| [5:0]EX\_OP | I | Ex阶段操作码 |
| [4:0]Rs\_ex | I | Ex阶段Rs寄存器值 |
| [5:0]IF\_FUNC | I | If阶段FUNC字段 |
| [15:0]ID\_imm16 | I | Id阶段立即数 |
| [25:0]ID\_Target | I | Id阶段Target段 |
| [25:0]IF\_Target | I | If阶段Target段 |
| [5:0]ID\_OP | I | Id阶段操作码 |
| [5:0]IF\_OP | I | If阶段操作码 |
| [4:0]rt | I | Id阶段Rt寄存器 |
| [31:0]NPC | O | 下条指令 |
| [31:0]NPC\_J | O | 延迟槽指令 |
| xiaoc | O | 阻塞指令 |
| J\_Q | O | 是否执行延迟槽指令 |
|  | 表1.4 NPC接口的功能 |  |
| 序号 | 功能名称 |  |
| 1 | 计算下条指令地址 |  |
| 2 | 如果J\_Q\_1=1,先执行延迟槽指令，后执行跳转指令 |  |

### 3.2.3 ALU 模块定义

1. 基本描述

实现addu，subu，slt，and，nor，or，xor，sll，srl，sltu ，sllv，sra，srav，srlv，lui，slti，sltiu 等基本操作

（2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.5 ALU的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [4:0]ALUctr | I | ALU控制信号 |
| [31:0]busA | I | ALU运算值 |
| [31:0] tempBus | I | ALU运算值 |
| [4:0] shamt | I | shamt段 |
| [31:0] Result | O | ALU运算结果 |
| Zero | O | ALU运算结果是否为0 |
| [5:0]op\_ex | I | Ex阶段操作码 |
| func\_ex | I | Ex阶段func字段 |
| [4:0]Rs\_ex | I | Ex阶段Rs寄存器 |
| [31:0]busA\_ex | I | ex阶段Rs寄存器值 |
| [31:0]busB\_ex | I | Ex阶段Rt寄存器值 |
| [31:0]Lo\_out\_ex | I | Ex阶段Lo寄存器值 |
| [31:0]Hi\_out\_ex | I | Ex阶段Hi寄存器值 |
| [31:0]CPR\_out\_ex | I | Ex阶段CPR寄存器值 |
| [31:0]pc\_ex | I | Ex阶段指令 |
| [31:0]Result\_next | O | 要存入Lo寄存器值 |

|  |  |  |
| --- | --- | --- |
|  | 表1.4 ALU接口的功能 |  |
| 序号 | 功能名称 | 功能解释 |
| 1 | ALU运算 | 如果MFLO指令，ALU结果为Lo寄存器的值 |
| 2 | ALU运算 | 如果MFHI指令，ALU结果为Hi寄存器的值 |
| 3 | ALU运算 | 如果MULT指令，ALU结果为Hi,Lo寄存器的值 |
| 4 | ALU运算 | 如果为SYSCALL，ALU结果为pc值 |
| 5 | ALU运算 | ALU结果为Rs寄存器值 |
| 6 | ALU运算 | ALU结果为Rt寄存器的值 |
| 7 | ALU运算 | 根据ALUCtrl，执行ALU运算 |

### 

### 3.2.4 dm 模块定义

数据内存大小为 4K，根据输入的地址读出数据内存中的数据，并根据数据写信号，将输入的数据选择写入数据内存中。

1. 模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.7dm的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [31:0] addr | I | 地址信号 |
| we | I | 写使能 |
| clk | I | 时钟 |
| [31:0] din | I | 写入地址的值 |
| [5:0] op | I | 操作码 |
| [31:0] dout | O | 读取内存地址的值 |

1. 功能模块

|  |  |  |
| --- | --- | --- |
|  | 表1.8dm的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据主存数据 | 根据地址，读取内存值 |
| 2 | 写入主存 | 根据写使能和写地址主存写值 |

### 3.2.5 im 模块定义

(1）基本描述

指令内存大小为 4K，初始化从 code.txt 载入指令。根据输入的指令地址，输出当前位置存储的指令。

（2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.9im的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [31:0]addr | I | 指令地址 |
| [31:0]dout | O | 指令地址 |

（3）功能模块

|  |  |  |
| --- | --- | --- |
|  | 表1.10im的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 载入指令 | 初始化载入 code.txt 中的指令 |
| 2 | 输出指令 | 根据输入指令地址，输出当前指令 |

### 3.2.6 regist 模块定义

（1）基本描述

根据输入的两个寄存器地址，输出相应寄存器的值，根据寄存器写信号和寄存器地址，将输入的数据选择写入寄存器。

(2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.11regist的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [31:0]Result\_wr | I | 指令地址 |
| [4:0] Rw | O | 指令地址 |
| [4:0] Ra | I | Rs寄存器 |
| [4:0] Rb | I | Rt寄存器 |
| [4:0] rt | I | rt段 |
| [5:0]op | I | 操作码 |
| [5:0]func | I | func段 |
| [31:0] PC | I | 当前指令地址 |
| [31:0] busW | I | 写入寄存器的值 |
| [31:0] busA | O | Rs寄存器的值 |
| [31:0] busB | O | Rt寄存器的值 |
| [31:0]Result\_next\_wr | I | 要存入Lo寄存器 |
| [2:0]sel\_id | I | Id阶段选择位 |
| [2:0]sel | I | Wr阶段选择位 |
| [31:0]pc\_wr | I | Wr阶段指令值 |
| [31:0]Di | I | 要写的值 |
| WE | I | GPR寄存器写使能 |
| WE\_HI | I | Hi寄存器写使能 |
| WE\_LO | I | Lo寄存器写使能 |
| WE\_HI\_LO | I | Hi，Lo寄存器写使能 |
| WE\_CPR | I | CPR寄存器写使能 |
| [31:0]Lo\_out | O | Lo寄存器值 |
| [31:0]Hi\_out | O | Hi寄存器值 |
| [31:0]CPR\_out | O | CPR寄存器值 |
| [31:0]CPR\_out\_14 | O | 14号CPR寄存器值 |

|  |  |  |
| --- | --- | --- |
|  | 表1.12register的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器数据 | 读 rs、rt，rd 寄存器的数据 |
| 2 | 向寄存器写入数据 | 根据写信号向寄存器选择写数据 |

### 3.2.7 ext 模块定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 表1.15ext的接口模块 | |  | |
| 信号名 | 方向 | | 描述 | |
| [15:0] imm16 | I | | 操作码 | |
| [31:0] imm32 | O | | func段 | |
| ExtOp | I | | 是否满足分支 | |
| （3）功能模块 | 表1.16ext的功能模块 | |  | |
| 序号 | 功能名称 | | 功能描述 | |
| 1 | 立即数扩展 | | 根据Extop对立即数扩展 | |
| 2 | 向寄存器写入数据 | | 根据写信号向寄存器选择写数据 | |
| 3.2.8 instruction 模块定义  （1） 更具指令输出译码信息  （2）模块接口  表1.17instruction的接口模块 | |  | |
| 信号名 | 方向 | | 描述 | |
| [31:0] Instruction | I | | 指令 | |
| [31:26] op | O | | 操作码 | |
| [25:21] rs | O | | rs寄存器 | |
| [20:16] rt | O | | rt寄存器 | |
| [15:11] rd | O | | rd寄存器 | |
| [10:6] shamt | O | | shamt段 | |
| [5:0] func | O | | func段 | |
| [15:0] imm16 | O | | 立即数 | |
| [25:0] Target | O | | Target段 | |

（1） 根据ExtOp是否需要符号扩展对立即数扩展

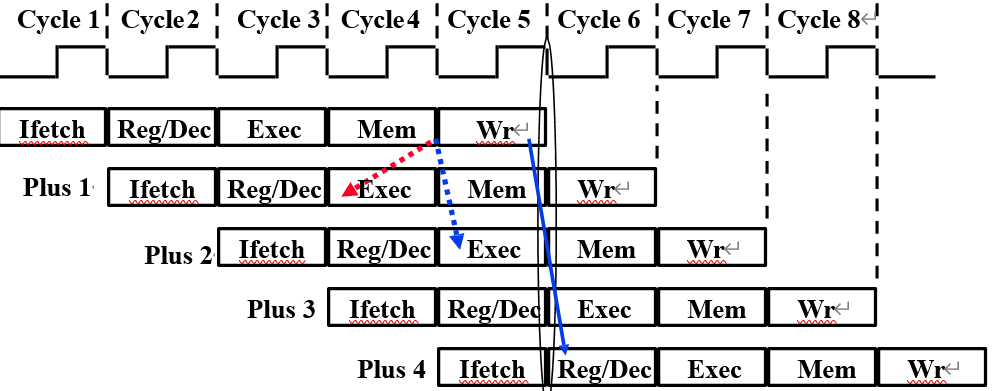
（2）模块接口

（3）功能模块

|  |  |  |
| --- | --- | --- |
|  | 表1.18instruction的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 指令译码 | 根据指令译码 |

## 3.3Loaduse冒险

### 3.3.1基本描述



Load指令最早在在第四周期结束时，数据在流水段寄存器中已经有值。

采用数据转发技术可以使load指令后面第二条指令得到所需的值 但不能解决load指令和随后第一条指令间的数据冒险，要延迟执行一条指令！ 回答了若不采用转发，在何时才能使用Load指令的结果这个问题

这种load指令和随后指令间的数据冒险，称为“装入- 使用数据冒险

### 3.3.2Loaduse冒险检测

1）基本描述

根据输入的两个寄存器地址，输出相应寄存器的值，根据寄存器写信号和寄存器地址，将输入的数据选择写入寄存器。

(2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.19Loaduse的接口模块 |  |
| 信号名 | 方向 | 描述 |
| MemRead\_ex | I | Ex阶段读主存信号 |
| [4:0] Rs\_id | I | Id阶段Rs寄存器 |
| [4:0] Rt\_id | I | Id阶段Rt寄存器 |
| [4:0] Rt\_ex | I | Ex阶段Rt寄存器 |
| loaduse | O | Loaduse冒险 |

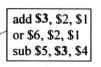
|  |  |  |
| --- | --- | --- |
|  | 表1.20Loaduse的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | Loaduse冒险检测 | 根据输入信号，检测是否发送loaduse冒险 |

### 3.3.3Loaduse解决方法

3.3.1（1）①将 ID / Ex 流水段寄存器中的所有控制信号清零（相当于插人了一个气泡），而不是将当时译码出来的控制信号送 ID / Ex 流水段寄存器； ② 保持 IF / ID 流水段寄存器的值不变，而不是送人当时取出的指令，这样，使 Load 后面的一条指令继续保存在 IF / lD 流水段寄存器中，在下一个时钟周期，该指令重新译码／取数； ③ 保持 PC 的值不变，使 Load 后面的第二条指令在下一个时钟周期重新执行取指令操作。图 6 . 18 给出了带转发和 Load - use 冒险处理的部分流水线数据通路。从图中可以看出，当检测到存在 Loaduse数据冒险时，Loaduse 检测单元送出 3 个控制信号，分别控制上述 3 个操作的实现。

## 3.4其他冒险

### 3.4.1 ByPass模块(1）基本描述

 利用转发技术解决第二条，第三条指令源操作数需要第一条指令目的操作数的数据冒险

|  |  |  |
| --- | --- | --- |
|  | 表1.21Bypass的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [5:0]op\_me | I | Me阶段指令操作码 |
| RegWr\_me | I | Me阶段寄存器写信号 |
| ALUSrc | I | ALUSrc信号 |
| RegWr\_wr | I | Wr阶段寄存器写信号 |
| [4:0]RegRd\_me | I | Me阶段RegRd段 |
| [4:0]RegRd\_wr | I | Wr阶段RegRd段 |
| [4:0]Rs | I | Rs寄存器 |
| [4:0]Rt | I | Rt寄存器 |
| [1:0]ALUSrcA | O | 数据选择器控制信号 |
| [1:0]ALUSrcB | O | 数据选择器控制信号 |

(2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.22Bypass的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据选择器控制信号 | 根据输入信号，产生数据选择器控制信号 |

### 

### 3.4.2 ByPass\_M(1）基本描述

利用转发技术，解决下调sw指令寄存器用到上条指令的目的寄存器的冒险

|  |  |  |
| --- | --- | --- |
|  | 表1.23Bypass\_M的接口模块 |  |
| 信号名 | 方向 | 描述 |
| RegWr\_wr | I | Wr阶段寄存器写信号 |
| [31:0]busB\_ex | I | Ex阶段Rs寄存器值 |
| [4:0]Rt\_ex | I | Ex阶段Rt寄存器 |
| [4:0] Rd\_ex | I | Ex阶Rd寄存器 |
| [5:0]op\_ex | I | Ex阶段操作码 |
| [31:0]Result\_wr | I | Wr阶段ALU运算结果 |
| [31:0]busB\_ex\_end | O | 主存数据存储器输入值 |

(2）模块接口

（3）功能模块

|  |  |  |
| --- | --- | --- |
|  | 表1.24Bypass\_M的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 准发 | 根据输入信号，产生busB\_ex\_end |

### 3.4.2 ByPass\_T(1）基本描述

|  |  |  |
| --- | --- | --- |
|  | 表1.25Bypass\_T的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [5:0]op | I | Ex阶段操作码 |
| [5:0]op\_id | I | Id阶段操作码 |
| [5:0] op\_me | I | Me阶段操作码 |
| [5:0] func\_id | I | Id阶段func字段 |
| [4:0]Rw\_me | I | Me阶段写寄存器 |
| [4:0]Rw\_ex | I | Ex阶段写寄存器 |
| [4:0]Rs\_id | I | Id阶段Rs寄存器 |
| [4:0]Rt\_id | I | Id阶段Rt寄存器 |
| RegWr\_me | I | Me阶段寄存器写信号 |
| [1:0]Ce\_A | O | 数据选择器选择信号 |
| [1:0]Ce\_B | O | 数据选择器选择信号 |

(2）模块接口

|  |  |  |
| --- | --- | --- |
|  | 表1.26Bypass的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 冒险转发技术 | 根据输入信号，产生数据选择器选择信号 |

### 3.4.2 ByPass\_X(1）基本描述

### 选择ALU的运算结果和Lo寄存器的值，进行转发

|  |  |  |
| --- | --- | --- |
|  | 表1.27Bypass\_X的接口模块 |  |
| 信号名 | 方向 | 描述 |
| [5:0]fun\_ex | I | Ex阶段func段 |
| [31:0]Result\_ex | I | Ex阶段ALU运算结果 |
| [31:0]Result\_me | I | me阶段ALU运算结果 |
| [31:0]Resultnextme | I | Lo寄存器值 |
| [31:0]Resultwr | I | Wr阶段ALU运算结果 |
| [31:0]Resultnextwr | I | Wr阶段Lo寄存器值 |
| Hi\_wr\_me | I | Me阶段Hi写信号 |
| Hi\_wr\_wr | I | Wr阶段Hi写信号 |
| Lo\_wr\_me | I | Me阶段Lo写信号 |
| Lo\_wr\_wr | I | Wr阶段Lo写信号 |
| Hi\_Lo\_wr\_me | I | Me阶段Hi，Lo写信号 |
| Hi\_Lo\_wr\_wr | I | Wr阶段Hi，Lo写信号 |
| [31:0]Result\_ex\_new | O | EX/MEM流水段寄存器Result输入 |

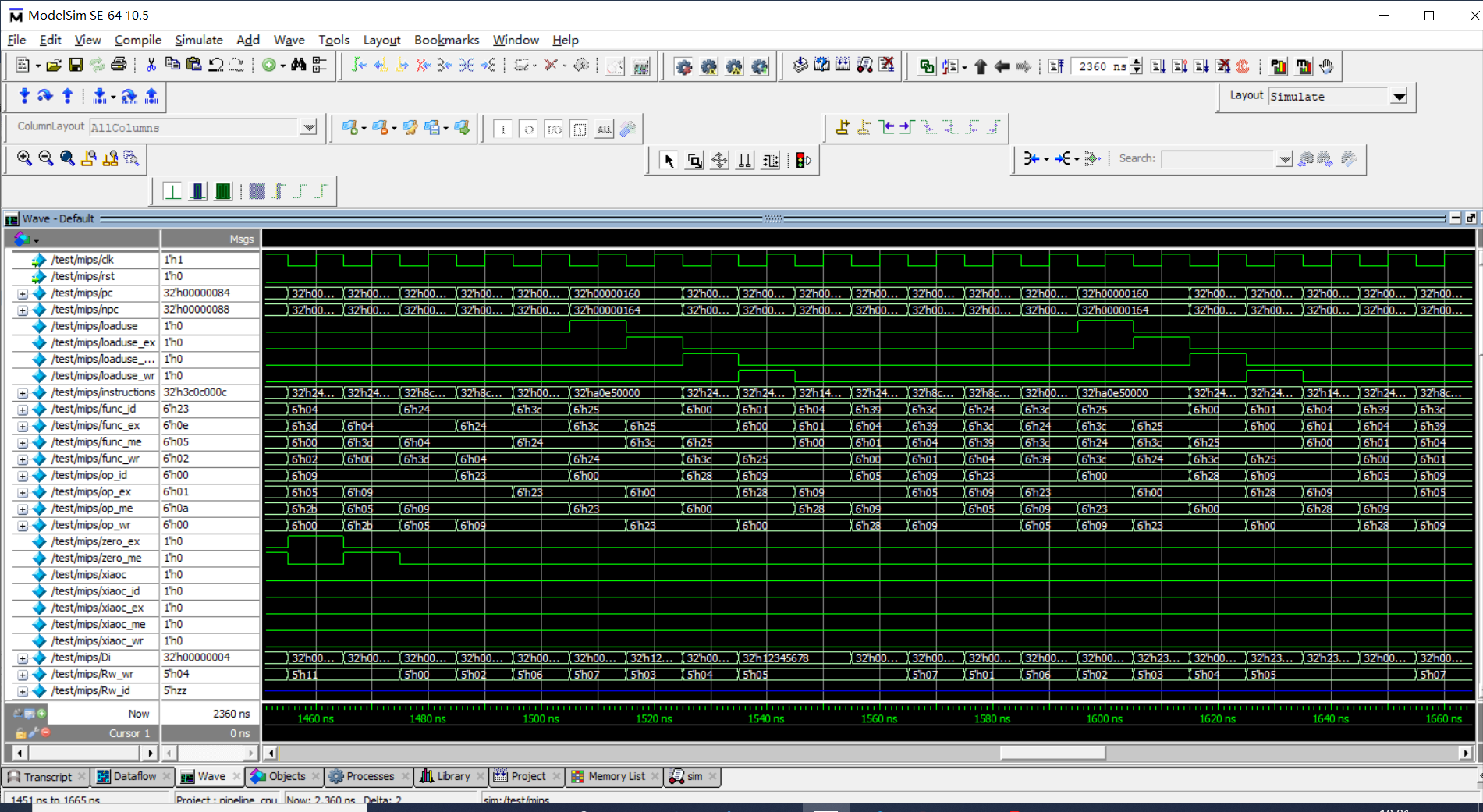
(2）模块接口

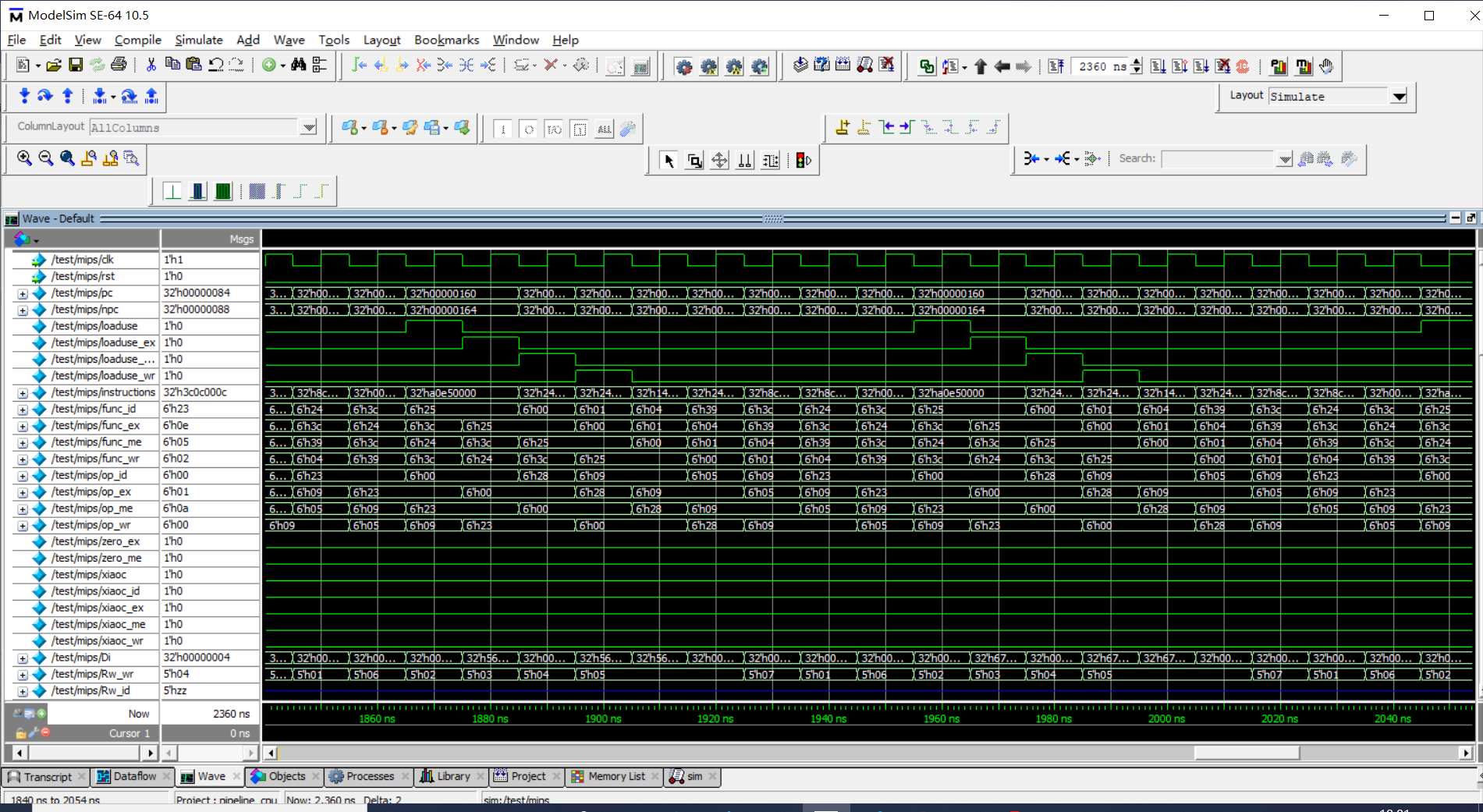
|  |  |  |
| --- | --- | --- |
|  | 表1.28Bypass\_X的功能模块 |  |
| 序号 | 功能名称 | 功能描述 |
| 1 | 冒险转发技术 | 产生EX/MEM流水段寄存器Result输入 |

# 

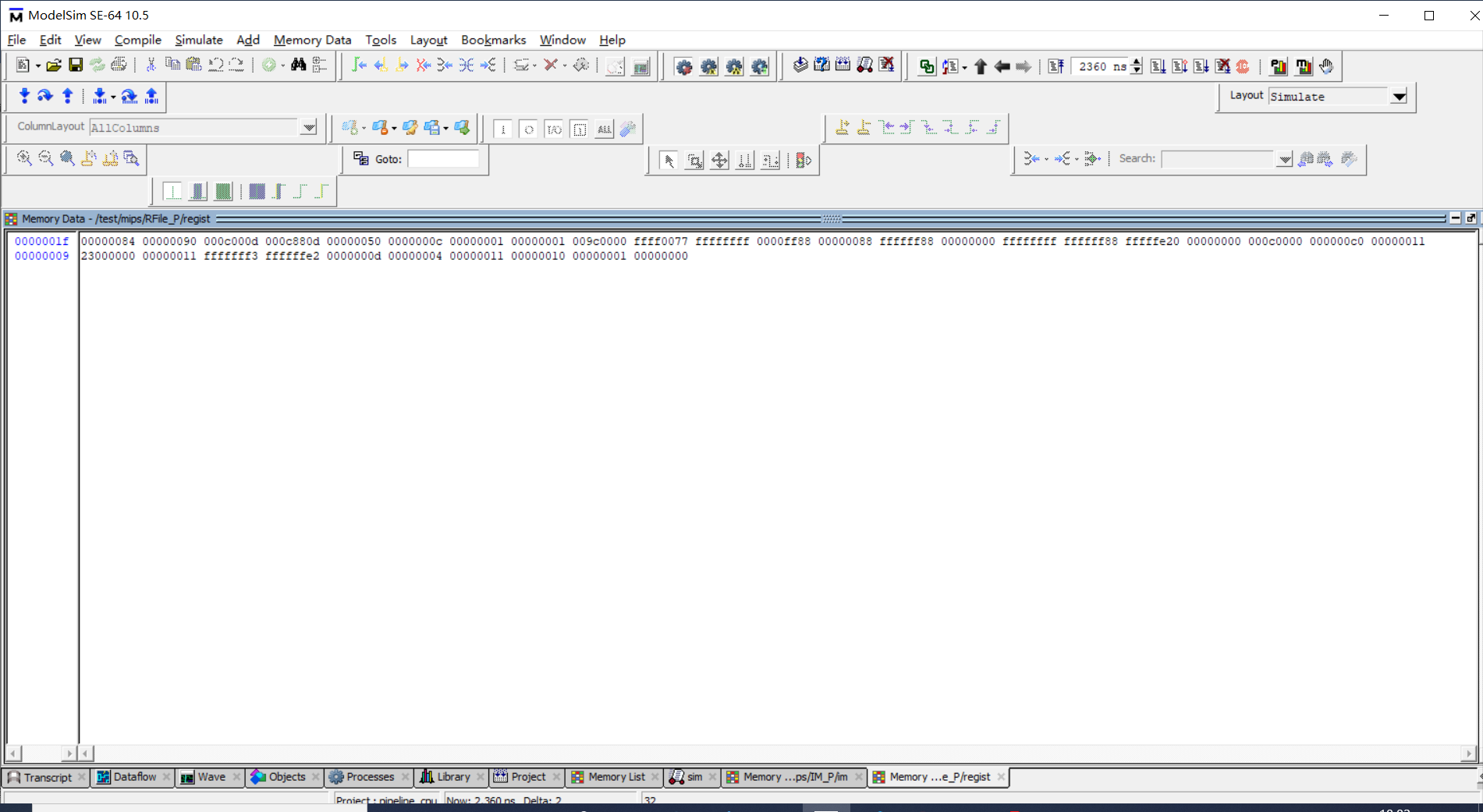
# 4．Modelsim运算结果

### 4.1波形图

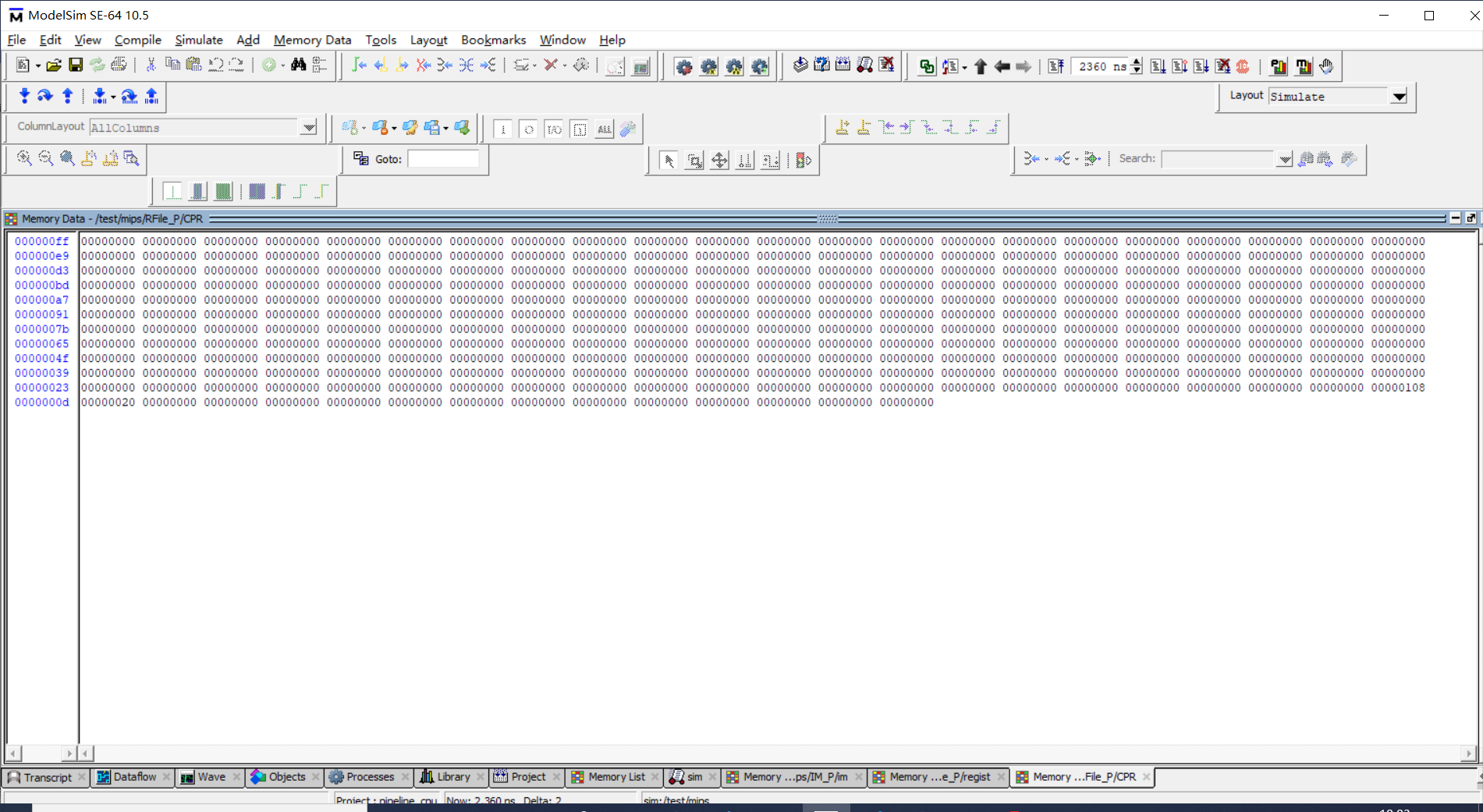




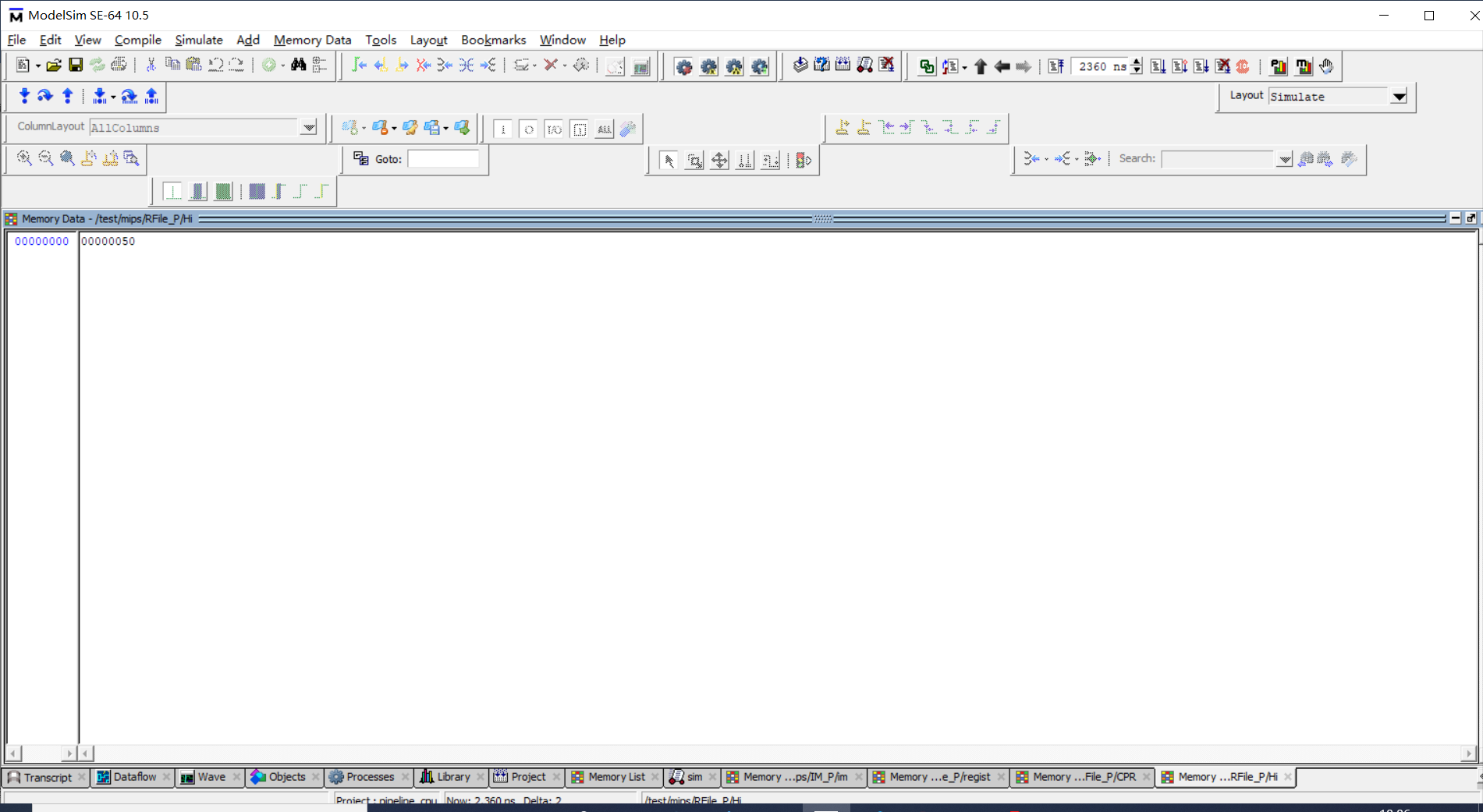
### 4.2GPR寄存器值



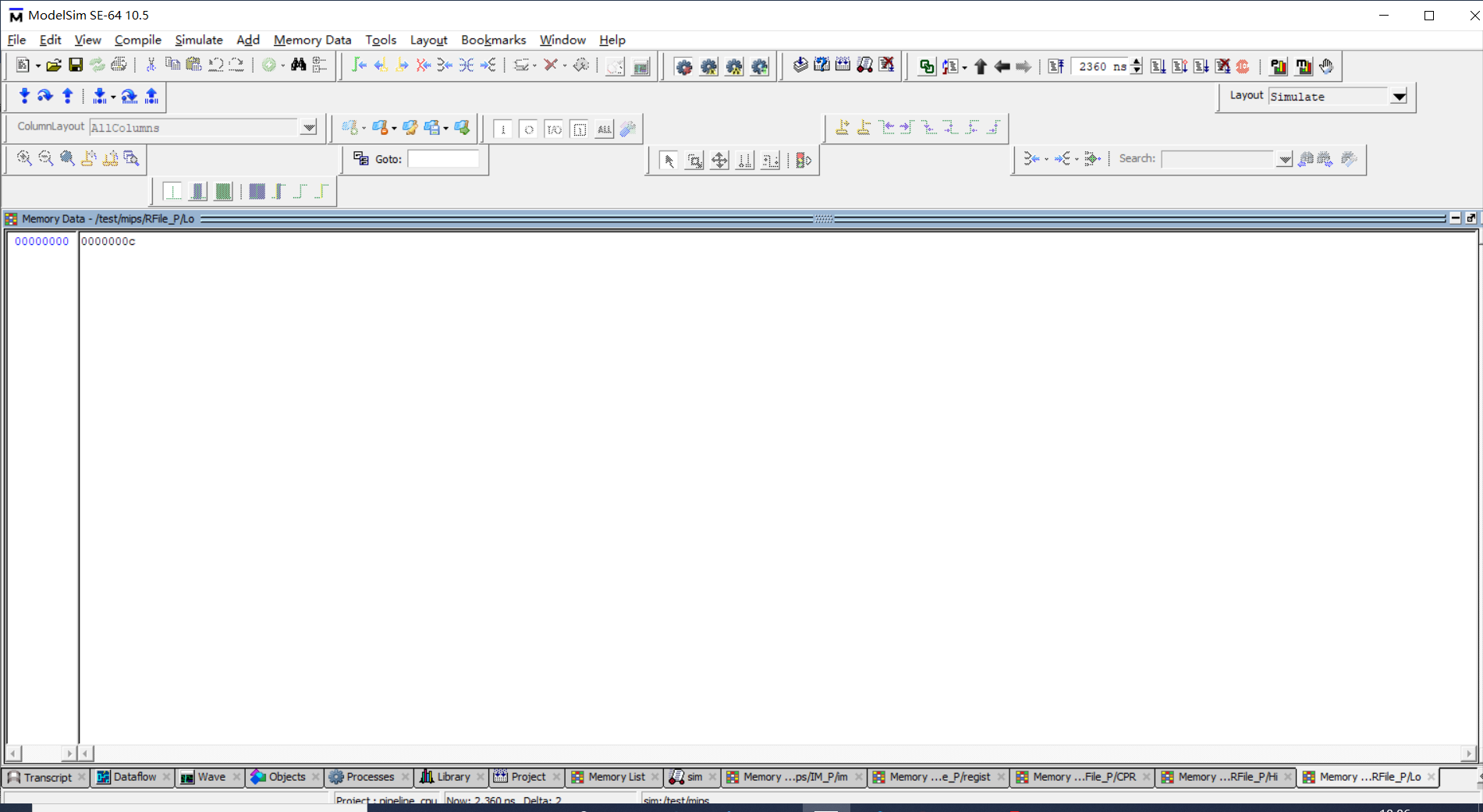
4.4CPR寄存器值



4.5Hi寄存器

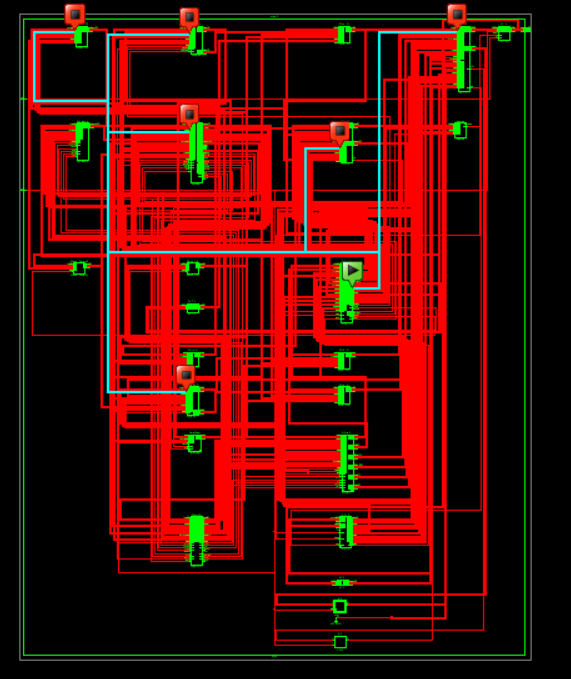


4.6Lo寄存器



5.ISE运算结果

5.1程序图



5.2ISE速度说明

Speed Grade: -3

Minimum period: 18.298ns (Maximum Frequency: 54.650MHz)

Minimum input arrival time before clock: 2.943ns

Maximum output required time after clock: 4.950ns

Maximum combinational path delay: No path found

附录一

CPU 测试 45 条指令所用汇编程序详述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
| Exception 入口地址，在 SYSCALL 指令执行后进入此处执行 | | | | |
| 00H | sw $1,#0($0) | Mem[0000\_0000H]  = 0000\_0008H | AC010000 | 1010\_1100\_0000\_0001\_0000\_0000\_0000\_0000 |
| 04H | sw $2,#4($0) | Mem[0000\_0004H]  = 0000\_0010H | AC020004 | 1010\_1100\_0000\_0010\_0000\_0000\_0000\_0100 |
| 08H | sw $3,#8($0) | Mem[0000\_0008H]  = 0000\_0011H | AC030008 | 1010\_1100\_0000\_0011\_0000\_0000\_0000\_1000 |
| 0CH | sw $4,#12($0) | Mem[0000\_000CH]  = 0000\_0004H | AC04000C | 1010\_1100\_0000\_0100\_0000\_0000\_0000\_1100 |
| 10H | sw $5,#16($0) | Mem[0000\_0010H]  = 0000\_000DH | AC050010 | 1010\_1100\_0000\_0101\_0000\_0000\_0001\_0000 |
| 14H | sw $6,#24($0) | Mem[0000\_0018H]  = FFFF\_FFE2H | AC060018 | 1010\_1100\_0000\_0110\_0000\_0000\_0001\_1000 |
| 18H | sw $7,#112($0) | Mem[0000\_0070H]  = FFFF\_FFF3H | AC070070 | 1010\_1100\_0000\_0111\_0000\_0000\_0111\_0000 |
| 1CH | sw $25,#116($0) | Mem[0000\_0074H]  = 0000\_0001H | AC190074 | 1010\_1100\_0001\_1001\_0000\_0000\_0111\_0100 |
| 20H | sw $13,#24($0) | Mem[0000\_0078H]  = 0000\_0000H | AC0D0078 | 1010\_1100\_0000\_1101\_0000\_0000\_0111\_1000 |
| 24H | mfc0 $1, cp0(14.0) | [$1] = 0000\_0104H | 40017000 | 0100\_0000\_0000\_0001\_0111\_0000\_0000\_0000 |
| 28H | addiu $1,$1,#4 | [$1] = 0000\_0108H | 24210004 | 0010\_0100\_0010\_0001\_0000\_0000\_0000\_0100 |
| 2CH | mtc0 $1, cp0(14.0) | cp0(14.0)  = 0000\_0108H | 40817000 | 0100\_0000\_1000\_0001\_0111\_0000\_0000\_0000 |
| 30H | eret | 返回 108H | 42000018 | 0100\_0010\_0000\_0000\_0000\_0000\_0001\_1000 |
| CPU 复位地址 0000\_0034H | | | | |
| 34H | addiu $1, $0,#1 | [$1] = 0000\_0001H | 24010001 | 0010\_0100\_0000\_0001\_0000\_0000\_0000\_0001 |
| 38H | sll $2, $1,#4 | [$2] = 0000\_0010H | 00011100 | 0000\_0000\_0000\_0001\_0001\_0001\_0000\_0000 |
| 3CH | addu $3, $2,$1 | [$3] = 0000\_0011H | 00411821 | 0000\_0000\_0100\_0001\_0001\_1000\_0010\_0001 |
| 40H | srl $4, $2,#2 | [$4] = 0000\_0004H | 00022082 | 0000\_0000\_0000\_0010\_0010\_0000\_1000\_0010 |
| 44H | slti $25,$4,#5 | [$25] = 0000\_0001H | 28990005 | 0010\_1000\_1001\_1001\_0000\_0000\_0000\_0101 |
| 48H | bgez $25,#14 | 跳转到 84H | 0721000E | 0000\_0111\_0010\_0001\_0000\_0000\_0000\_1110 |
| 4CH | subu $5, $3,$4 | [$5] = 0000\_000DH | 00642823 | 0000\_0000\_0110\_0100\_0010\_1000\_0010\_0011 |
| 50H | sw $5, #20($0) | Mem[0000\_0014H]  = 0000\_000DH | AC050014 | 1010\_1100\_0000\_0101\_0000\_0000\_0001\_0100 |
| 54H | nor $6, $5,$2 | [$6] = FFFF\_FFE2H | 00A23027 | 0000\_0000\_1010\_0010\_0011\_0000\_0010\_0111 |
| 58H | or $7, $6,$3 | [$7] = FFFF\_FFF3H | 00C33825 | 0000\_0000\_1100\_0011\_0011\_1000\_0010\_0101 |
| 5CH | xor $8, $7,$6 | [$8] = 0000\_0011H | 00E64026 | 0000\_0000\_1110\_0110\_0100\_0000\_0010\_0110 |
| 60H | beq $8, $3,#2 | 跳转到 6CH | 11030002 | 0001\_0001\_0000\_0011\_0000\_0000\_0000\_0010 |
| 64H | sw $8, #28($0) | Mem[0000\_001CH]  = 0000\_0011H | AC08001C | 1010\_1100\_0000\_1000\_0000\_0000\_0001\_1100 |
| 68H | slt $9, $1,$2 | 不执行 | 0022482A | 0000\_0000\_0010\_0010\_0100\_1000\_0010\_1010 |
| 6CH | lw $10,#28($0) | [$10] = 0000\_0011H | 8C0A001C | 1000\_1100\_0000\_1010\_0000\_0000\_0001\_1100 |
| 70H | bne $10,$5,#2 | 跳转到 7CH | 15450002 | 0001\_0101\_0100\_0101\_0000\_0000\_0000\_0010 |

表 2 CPU 测试 45 条指令所用汇编程序详述（续）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
| 74H | and $11,$2,$1 | [$11] = 0000\_0000H | 00415824 | 0000\_0000\_0100\_0001\_0101\_1000\_0010\_0100 |
| 78H | sw $11,#28($0) | 不执行 | AC0B001C | 1010\_1100\_0000\_1011\_0000\_0000\_0001\_1100 |
| 7CH | jal #38 | 跳转到 98H,  [$31] =0000\_0084H | 0C000026 | 0000\_1100\_0000\_0000\_0000\_0000\_0010\_0110 |
| 80H | sw $4, #16($0) | Mem[0000\_0010H] | AC040010 | 1010\_1100\_0000\_0100\_0000\_0000\_0001\_0000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | = 0000\_0004H |  |  |
| 84H | lui $12,#12 | [$12] = 000C\_0000H | 3C0C000C | 0011\_1100\_0000\_1100\_0000\_0000\_0000\_1100 |
| 88H | srav $26,$12,$2 | [$26] = 0000\_000CH | 004CD007 | 0000\_0000\_0100\_1100\_1101\_0000\_0000\_0111 |
| 8CH | addiu $27,$26,#68 | [$27] = 0000\_0050H | 275B0044 | 0010\_0111\_0101\_1011\_0000\_0000\_0100\_0100 |
| 90H | jalr $27 | 跳转到 50H,  [$31] = 0000\_0098H | 0360F809 | 0000\_0011\_0110\_0000\_1111\_1000\_0000\_1001 |
| 94H | addiu $1, $0,#8 | [$1] = 0000\_0008H | 24010008 | 0010\_0100\_0000\_0001\_0000\_0000\_0000\_1000 |
| 98H | sb $26,#5($3) | MEM[0000\_0014H]  = 000C\_000DH | A07A0005 | 1010\_0000\_0111\_1010\_0000\_0000\_0000\_0101 |
| 9CH | sltu $13,$10,$3 | [$13] = 0000\_0000H | 0143682B | 0000\_0001\_0100\_0011\_0110\_1000\_0010\_1011 |
| A0H | bgtz $13,#2 | 不跳转 | 1DA00002 | 0001\_1101\_1010\_0000\_0000\_0000\_0000\_0010 |
| A4H | sllv $14,$6,$4 | [$14] = FFFF\_FE20H | 00867004 | 0000\_0000\_1000\_0110\_0111\_0000\_0000\_0100 |
| A8H | sra $15,$14,#2 | [$15] = FFFF\_FF88H | 000E7883 | 0000\_0000\_0000\_1110\_0111\_1000\_1000\_0011 |
| ACH | srlv $16,$15,$1 | [$16] = 00FF\_FFFFH | 002F8006 | 0000\_0000\_0010\_1111\_1000\_0000\_0000\_0110 |
| B0H | blez $16,#7 | 不跳转 | 1A000007 | 0001\_1010\_0000\_0000\_0000\_0000\_0000\_0111 |
| B4H | srav $16,$15,$1 | [$16] = FFFF\_FFFFH | 002F8007 | 0000\_0000\_0010\_1111\_1000\_0000\_0000\_0111 |
| B8H | bltz $16,#6 | 跳转到 D4H | 06000006 | 0000\_0110\_0000\_0000\_0000\_0000\_0000\_0110 |
| BCH | sll $11,$26,#4 | [$11] = 0000\_00C0H | 001A5900 | 0000\_0000\_0001\_1010\_0101\_1001\_0000\_0000 |
| C0H | lw $28,#3($10) | [$28]=000C\_000DH / 000C\_880DH | 8D5C0003 | 1000\_1101\_0101\_1100\_0000\_0000\_0000\_0011 |
| C4H | bne $28,$29,#7 | 不跳转/跳转 E4H | 179D0007 | 0001\_0111\_1001\_1101\_0000\_0000\_0000\_0111 |
| C8H | sb $15,#8($5) | Mem[0000\_0014H]  = 000C\_8800H | A0AF0008 | 1010\_0000\_1010\_1111\_0000\_0000\_0000\_1000 |
| CCH | lb $18,#8($5) | [$18] = FFFF\_FF88H | 80B20008 | 1000\_0000\_1011\_0010\_0000\_0000\_0000\_1000 |
| D0H | lbu $19,#8($5) | [$19] = 0000\_0088H | 90B30008 | 1001\_0000\_1011\_0011\_0000\_0000\_0000\_1000 |
| D4H | sltiu $24,$15,#0xFFFF | [$24] = 0000\_0001H | 2DF8FFFF | 0010\_1101\_1111\_1000\_1111\_1111\_1111\_1111 |
| D8H | or $29,$12,$5 | [$29] = 000C\_000DH | 0185E825 | 0000\_0001\_1000\_0101\_1110\_1000\_0010\_0101 |
| DCH | jr $11 | 跳转到 C0H | 01600008 | 0000\_0001\_0110\_0000\_0000\_0000\_0000\_1000 |
| E0H | andi $20,$15,#0xFFFF | [$20] = 0000\_FF88H | 31F4FFFF | 0011\_0001\_1111\_0100\_1111\_1111\_1111\_1111 |
| E4H | ori $21,$15,#0xFFFF | [$21] = FFFF\_FFFFH | 35F5FFFF | 0011\_0101\_1111\_0101\_1111\_1111\_1111\_1111 |
| E8H | xori $22,$15,#0xFFFF | [$22] = FFFF\_0077H | 39F6FFFF | 0011\_1001\_1111\_0110\_1111\_1111\_1111\_1111 |
| ECH | mult $12,$29 | [HI] = 0000\_0090H, [LO] = 009C\_0000H | 019D0018 | 0000\_0001\_1001\_1101\_0000\_0000\_0001\_1000 |
| F0H | mflo $23 | [$23] = 009C\_0000H | 0000B812 | 0000\_0000\_0000\_0000\_1011\_1000\_0001\_0010 |
| F4H | mfhi $30 | [$30] = 0000\_0090H | 0000F010 | 0000\_0000\_0000\_0000\_1111\_0000\_0001\_0000 |
| F8H | mtlo $26 | [LO] = 0000\_000CH | 03400013 | 0000\_0011\_0100\_0000\_0000\_0000\_0001\_0011 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
| FCH | mthi $27 | [HI] = 0000\_0050H | 03600011 | 0000\_0011\_0110\_0000\_0000\_0000\_0001\_0001 |
| 100H | mtc0 $0,c14 | cp0[14.0]  = 0000\_0000H | 40807000 | 0100\_0000\_1000\_0000\_0111\_0000\_0000\_0000 |
|  |  | cp0[14.0] |  |  |
|  |  | =0000\_0104H, |  |  |
|  |  | cp0[13.0][6..2] |  |  |
| 104H | syscall | =01000B, | 0000000C | 0000\_0000\_0000\_0000\_0000\_0000\_0000\_1100 |
|  |  | cp0[12.0][1]=1, |  |  |
|  |  | 跳 转 Exception 入 口 |  |  |
|  |  | 地址，00H |  |  |
| 108H | mfc0 $2, cp0(14.0) | [$2] = 0000\_0108H | 40027000 | 0100\_0000\_0000\_0010\_0111\_0000\_0000\_0000 |
| 10CH | mfc0 $3, cp0(13.0) | [$3] = 0000\_0020H | 40036800 | 0100\_0000\_0000\_0011\_0110\_1000\_0000\_0000 |
| 110H | mfc0 $4, cp0(12.0) | [$4] = 0000\_0000H | 40046000 | 0100\_0000\_0000\_0100\_0110\_0000\_0000\_0000 |
| 114H | addiu $1, $0,#32 | [$1] = 0000\_0020H | 24010020 | 0010\_0100\_0000\_0001\_0000\_0000\_0010\_0000 |
| 118H | slt $17,$15,$14 | [$17] = 0000\_0000H | 01EE882A | 0000\_0001\_1110\_1110\_1000\_1000\_0010\_1010 |
| 11CH | lui $17,#1234H | [$17] = 1234\_0000H | 3C111234 | 0011\_1100\_0001\_0001\_0001\_0010\_0011\_0100 |
| 120H | addiu $17,$17,#5678 | [$17] = 1234\_5678H | 26315678 | 0010\_0110\_0011\_0001\_0101\_0110\_0111\_1000 |
|  |  | Mem[0000\_0020H] |  |  |
|  |  | =1234\_5678H / |  |  |
|  |  | Mem[0000\_0024H] |  |  |
|  |  | =2345\_6780H / |  |  |
|  |  | Mem[0000\_0028H] |  |  |
|  |  | =3456\_7800H / |  |  |
| 124H | sw $17,#0($1) | Mem[0000\_002CH]  =4567\_8000H / | AC310000 | 1010\_1100\_0011\_0001\_0000\_0000\_0000\_0000 |
|  |  | Mem[0000\_0030H] |  |  |
|  |  | =5678\_0000H / |  |  |
|  |  | Mem[0000\_0034H] |  |  |
|  |  | =6780\_0000H / |  |  |
|  |  | Mem[0000\_0038H] |  |  |
|  |  | =7800\_0000H |  |  |
|  |  | [$17] = 2345\_6780H / |  |  |
|  |  | 3456\_7800H / |  |  |
|  |  | 4567\_8000H / |  |  |
| 128H | sll $17,$17,#4 | 5678\_0000H / | 00118900 | 0000\_0000\_0001\_0001\_1000\_1001\_0000\_0000 |
|  |  | 6780\_0000H / |  |  |
|  |  | 7800\_0000H / |  |  |
|  |  | 8000\_0000H |  |  |
|  |  | 跳转到 124H / 跳转/ |  |  |
| 12CH | bgtz $17,#-3 | 跳转/跳转/跳转/跳转 | 1E20FFFD | 0001\_1110\_0010\_0000\_1111\_1111\_1111\_1101 |
|  |  | 124H/不跳转 |  |  |
|  |  | [$1] = 0000\_0024H / |  |  |
|  |  | 0000\_0028H / |  |  |
|  |  | 0000\_002CH / |  |  |
| 130H | addiu $1,$1,#4 | 0000\_0030H / | 24210004 | 0010\_0100\_0010\_0001\_0000\_0000\_0000\_0100 |
|  |  | 0000\_0034H / |  |  |
|  |  | 0000\_0038H / |  |  |
|  |  | 0000\_003CH |  |  |
| 134H | addiu $2,$0,#60 | [$2] = 0000\_003CH | 2402003C | 0010\_0100\_0000\_0010\_0000\_0000\_0011\_1100 |
| 138H | lw $17,#-28($1) | [$17] = 1234\_5678H | 8C31FFE4 | 1000\_1100\_0011\_0001\_1111\_1111\_1110\_0100 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
|  |  | [$17] = 0123\_4567H / |  |  |
|  |  | 0012\_3456H / |  |  |
|  |  | 0001\_2345H / |  |  |
| 13CH | srl $17,$17,#4 | 0000\_1234H /  0000\_0123H / | 00118902 | 0000\_0000\_0001\_0001\_1000\_1001\_0000\_0010 |
|  |  | 0000\_0012H / |  |  |
|  |  | 0000\_0001H / |  |  |
|  |  | 0000\_0000H |  |  |
|  |  | Mem[0000\_003CH] |  |  |
|  |  | =0123\_4567H / |  |  |
|  |  | Mem [0000\_0040H] |  |  |
|  |  | =0012\_3456H / |  |  |
|  |  | Mem [0000\_0044H] |  |  |
|  |  | =0001\_2345H / |  |  |
|  |  | Mem [0000\_0048H] |  |  |
| 140H | sw $17,#0($2) | =0000\_1234H / Mem [0000\_004CH] | AC510000 | 1010\_1100\_0101\_0001\_0000\_0000\_0000\_0000 |
|  |  | =0000\_0123H / |  |  |
|  |  | Mem [0000\_0050H] |  |  |
|  |  | =0000\_0012H / |  |  |
|  |  | Mem [0000\_0054H] |  |  |
|  |  | =0000\_0001H / |  |  |
|  |  | Mem [0000\_0058H] |  |  |
|  |  | =0000\_0000H |  |  |
|  |  | 跳转到 13CH /跳转/ |  |  |
| 144H | bne $17,$0,#-3 | 跳转/跳转/跳转/跳转 | 1620FFFD | 0001\_0110\_0010\_0000\_1111\_1111\_1111\_1101 |
|  |  | /跳转 13CH/不跳转 |  |  |
|  |  | [$2] = 0000\_0040H / |  |  |
|  |  | 0000\_0044H / |  |  |
|  |  | 0000\_0048H / |  |  |
| 148H | addiu $2,$2,#4 | 0000\_004CH /  0000\_0050H / | 24420004 | 0010\_0100\_0100\_0010\_0000\_0000\_0000\_0100 |
|  |  | 0000\_0054H / |  |  |
|  |  | 0000\_0058H / |  |  |
|  |  | 0000\_005CH |  |  |
| 14CH | addiu $6,$0,#68 | [$6] = 0000\_0044H | 24060044 | 0010\_0100\_0000\_0110\_0000\_0000\_0100\_0100 |
| 150H | addiu $7,$0,#100 | [$7] =0000\_0064H | 24070064 | 0010\_0100\_0000\_0111\_0000\_0000\_0110\_0100 |
|  |  | [$3] = 1234\_5678H / |  |  |
|  |  | 2345\_6780H / |  |  |
|  |  | 3456\_7800H / |  |  |
| 154H | lw $3,#-28($1) | 4567\_8000H / | 8C23FFE4 | 1000\_1100\_0010\_0011\_1111\_1111\_1110\_0100 |
|  |  | 5678\_0000H / |  |  |
|  |  | 6780\_0000H / |  |  |
|  |  | 7800\_0000H |  |  |
|  |  | [$4] = 0000\_0000H / |  |  |
|  |  | 0000\_0001H / |  |  |
|  |  | 0000\_0012H / |  |  |
| 158H | lw $4,#-4($2) | 0000\_0123H / | 8C44FFFC | 1000\_1100\_0100\_0100\_1111\_1111\_1111\_1100 |
|  |  | 0000\_1234H / |  |  |
|  |  | 0001\_2345H / |  |  |
|  |  | 0012\_3456H |  |  |
|  |  | [$5] = 1234\_5678H / |  |  |
|  |  | 2345\_6781H / |  |  |
|  |  | 3456\_7812H / |  |  |
| 15CH | or $5,$3,$4 | 4567\_8123H / | 00642825 | 0000\_0000\_0110\_0100\_0010\_1000\_0010\_0101 |
|  |  | 5678\_1234H / |  |  |
|  |  | 6781\_2345H / |  |  |
|  |  | 7812\_3456H |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
|  |  | Mem[0000\_0064H] |  |  |
|  |  | =0000\_0078H / |  |  |
|  |  | Mem [0000\_0064H] |  |  |
|  |  | =0000\_8178H / |  |  |
|  |  | Mem[0000\_0064H] |  |  |
|  |  | =0012\_8178H / |  |  |
| 160H | sb $5,#0($7) | Mem[0000\_0064H]  =2312\_8178H / | A0E50000 | 1010\_0000\_1110\_0101\_0000\_0000\_0000\_0000 |
|  |  | Mem[0000\_0068H] |  |  |
|  |  | =0000\_0034H / |  |  |
|  |  | Mem[0000\_0068H] |  |  |
|  |  | =0000\_4534H / |  |  |
|  |  | Mem[0000\_0068H] |  |  |
|  |  | =0056\_4534H |  |  |
|  |  | [$7] = 0000\_0065H / |  |  |
|  |  | 0000\_0066H / |  |  |
|  |  | 0000\_0067H / |  |  |
| 164H | addiu $7,$7,#1 | 0000\_0068H / | 24E70001 | 0010\_0100\_1110\_0111\_0000\_0000\_0000\_0001 |
|  |  | 0000\_0069H / |  |  |
|  |  | 0000\_006AH / |  |  |
|  |  | 0000\_006BH |  |  |
|  |  | [$1] = 0000\_0040H / |  |  |
|  |  | 0000\_0044H / |  |  |
|  |  | 0000\_0048H / |  |  |
| 168H | addiu $1,$1,#4 | 0000\_004CH / | 24210004 | 0010\_0100\_0010\_0001\_0000\_0000\_0000\_0100 |
|  |  | 0000\_0050H / |  |  |
|  |  | 0000\_0054H / |  |  |
|  |  | 0000\_0058H |  |  |
|  |  | 跳转到 154H/跳转/跳 |  |  |
| 16CH | bne $2,$6,#-7 | 转 / 跳 转 / 跳 转 / 跳 转 | 1446FFF9 | 0001\_0100\_0100\_0110\_1111\_1111\_1111\_1001 |
|  |  | 154H/不跳转 |  |  |
|  |  | [$2] = 0000\_0058H / |  |  |
|  |  | 0000\_0054H / |  |  |
|  |  | 0000\_0050H / |  |  |
| 170H | ADDIU $2,$2,#-4 | 0000\_004CH / | 2442FFFC | 0010\_0100\_0100\_0010\_1111\_1111\_1111\_1100 |
|  |  | 0000\_0048H / |  |  |
|  |  | 0000\_0044H / |  |  |
|  |  | 0000\_0040H |  |  |
| 174H | addiu $9,$0,#100 | [$9] = 0000\_0064H | 24090064 | 0010\_0100\_0000\_1001\_0000\_0000\_0110\_0100 |
| 178H | lbu $9,#3($9) | [$9] = 0000\_0023H | 91290003 | 1001\_0001\_0010\_1001\_0000\_0000\_0000\_0011 |
| 17CH | addiu $13,$0,#104 | [$13] = 0000\_0068H | 240D0068 | 0010\_0100\_0000\_1101\_0000\_0000\_0110\_1000 |
| 180H | lw $13,#0($13) | [$13] = 0056\_4534H | 8DAD0000 | 1000\_1101\_1010\_1101\_0000\_0000\_0000\_0000 |
| 184H | ll $9,$9,#24 | [$9] = 2300\_0000H | 00094E00 | 0000\_0000\_0000\_1001\_0100\_1110\_0000\_0000 |
| 188H | xori $13,$13,#9 | [$13] = 0056\_453DH | 39AD0009 | 0011\_1001\_1010\_1101\_0000\_0000\_0000\_1001 |
| 18CH | sw $13,#1($7) | MEM[0000\_006CH]= 0056\_453DH | ACED0001 | 1010\_1100\_1110\_1101\_0000\_0000\_0000\_0001 |
| 190H | lw $1,#0($0) | [$1] = 0000\_0008H | 8C010000 | 1000\_1100\_0000\_0001\_0000\_0000\_0000\_0000 |
| 194H | lw $2,#4($0) | [$2] = 0000\_0010H | 8C020004 | 1000\_1100\_0000\_0010\_0000\_0000\_0000\_0100 |
| 198H | lw $3,#8($0) | [$3] = 0000\_0011H | 8C030008 | 1000\_1100\_0000\_0011\_0000\_0000\_0000\_1000 |
| 19CH | lw $4,#12($0) | [$4] = 0000\_0004H | 8C04000C | 1000\_1100\_0000\_0100\_0000\_0000\_0000\_1100 |
| 1A0H | lw $5,#16($0) | [$5] = 0000\_000DH | 8C050010 | 1000\_1100\_0000\_0101\_0000\_0000\_0001\_0000 |
| 1A4H | lw $6,#24($0) | [$6] = FFFF\_FFE2H | 8C060018 | 1000\_1100\_0000\_0110\_0000\_0000\_0001\_1000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指令 地址** | **汇编指令** | **结果描述** | **机器指令的机器码** | |
| **16 进制** | **二进制** |
| 1A8H | lw $7,#112($0) | [$7] = FFFF\_FFF3H | 8C070070 | 1000\_1100\_0000\_0111\_0000\_0000\_0111\_0000 |
| 1ACH | lw $25,#116($0) | [$25] = 0000\_0001H | 8C190074 | 1000\_1100\_0001\_1001\_0000\_0000\_0111\_0100 |
| 1B0H | lw $13,#120($0) | [$13] = 0000\_0000H | 8C0D0078 | 1000\_1100\_0000\_1101\_0000\_0000\_0111\_1000 |
| 1B4H | j #34H | 跳转到 34H | 0800000D | 0000\_1000\_0000\_0000\_0000\_0000\_0000\_1101 |

# 附录二 verilog代码

module ByPass\_M( //add 7,1,3,+ nop +sw 7

input RegWr\_wr,

input[31:0] busB\_ex,

input [4:0] rt\_ex,

input [4:0] rd\_wr,

input [5:0] op\_ex,

input [31:0] Result\_wr,

output [31:0] busB\_ex\_end

);

assign busB\_ex\_end=(op\_ex==6'b101011 && RegWr\_wr==1 && rt\_ex==rd\_wr)? Result\_wr:busB\_ex ;

Endmodule

module ByPass\_T( //busA\_id,Result\_ex,Result\_me,Do\_me,busA\_end //busB\_id,Result\_ex,Result\_me,Do\_me,busB\_end

input [5:0] op,//op\_ex

input [5:0] op\_id,

input [5:0] op\_me,

input [5:0] func\_id,

input [4:0] Rw\_me,

input [4:0] Rw\_ex,

input [4:0] Rs\_id,

input [4:0] Rt\_id,

// input RegWr\_ex,

input RegWr\_me,

output reg [1:0] Ce\_A, //MUX ctrl

output reg [1:0] Ce\_B

);

wire RegWr\_ex;

assign RegWr\_ex=(!op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&!op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0])|(op[5]&!op[4]&!op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0])|(!op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0]);

always @(\*)

begin

if(RegWr\_ex&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|(op\_id==6'b000000 && func\_id==6'b001001)|op\_id==6'b000110|op\_id==6'b000001)&&(Rs\_id==Rw\_ex)) //jalr BYPASS

Ce\_A<=2'b01;//Result\_me

else if((op\_me==6'b100011)&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|op\_id==6'b000110|op\_id==6'b000001)&&(Rs\_id==Rw\_me))

Ce\_A<=2'b11; //Do\_me

else if(RegWr\_me&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|op\_id==6'b000110|op\_id==6'b000001)&&(Rs\_id==Rw\_me))

Ce\_A<=2'b10; //Result\_ex

else Ce\_A<=2'b00;//busA\_id

if(RegWr\_ex&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|op\_id==6'b000110|op\_id==6'b000001)&&(Rt\_id==Rw\_ex))

Ce\_B<=2'b01;//Result\_ex

else if(RegWr\_me&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|op\_id==6'b000110|op\_id==6'b000001)&&(Rt\_id==Rw\_me))

Ce\_B<=2'b10;//Result\_me

else if((op\_me==6'b100011)&&(op\_id==6'b000100|op\_id==6'b000101|op\_id==6'b000001|op\_id==6'b000111|op\_id==6'b000110|op\_id==6'b000001)&&(Rt\_id==Rw\_me))

Ce\_B<=2'b11;//Do\_me

else Ce\_B<=2'b00;//busB\_id

end

endmodule

module ByPass\_X( //mflo 23 +mfhi30+ mtlo26 func\_ex

input [5:0] func\_ex,

input [31:0] Result\_ex,

input [31:0] Result\_me,

input [31:0] Resultnextme,

input [31:0] Result\_wr,

input [31:0] Rusultnextwr,

input Hi\_wr\_me,

input Hi\_wr\_wr,

input Lo\_wr\_me,

input Lo\_wr\_wr,

input Hi\_Lo\_wr\_\_me,

input Hi\_Lo\_wr\_wr,

output reg [31:0] Result\_ex\_new

);

always @(\*)

begin

if((Lo\_wr\_me|Hi\_Lo\_wr\_\_me) && func\_ex==6'b010010) //MFLO, LO->register

Result\_ex\_new<=Resultnextme; //alu resultnextme[31:0]

else if((Hi\_wr\_wr|Hi\_Lo\_wr\_wr) && func\_ex==6'b010000 ) //MFHI Hi->register

Result\_ex\_new<=Result\_wr;

else Result\_ex\_new<=Result\_ex; // alu result

end

endmodule

module DataMem(input clk,

input sb\_me, //sb inst?

input MemWr\_me,

input [31:0] Result\_me,// Adr

input[31:0] busB\_me, //data is stored

output [31:0] Do);//DataMem-outdata

reg [31:0] dm[1023:0];

integer i;

initial

begin

for(i=0;i<1024;i=i+1)

dm[i]=32'b0;

end

assign Do=dm[Result\_me[11:2]];

always @(posedge clk)

if(MemWr\_me)

begin

if(sb\_me) //sb store byte

begin

if(Result\_me[1:0]==2'b00)

dm[Result\_me[11:2]][7:0]<=busB\_me[7:0];

if(Result\_me[1:0]==2'b01)

dm[Result\_me[11:2]][15:8]<=busB\_me[7:0];

if(Result\_me[1:0]==2'b10)

dm[Result\_me[11:2]][23:16]<=busB\_me[7:0];

if(Result\_me[1:0]==2'b11)

dm[Result\_me[11:2]][31:24]<=busB\_me[7:0];

end

else

dm[Result\_me[11:2]]<=busB\_me; //store word

end

endmodule

module ExMem(input clk, // Rs\_me,func\_me,

//转发 sb，sw

input RegWr\_me\_j,

input [4:0] Rw\_me\_j,

input [4:0] Rt\_ex,

input [31:0] Result\_me\_j,

input [31:0] pc\_ex,

input xiaoc\_ex,

input zero\_ex,

input [31:0] Result\_ex,

input [31:0] Result\_next\_ex,

input [31:0] busB\_ex,

input loaduse\_ex, //

input [4:0] Rw\_ex,

input [5:0] op\_ex, //////////////////

input [5:0] func\_ex, ///////////////////

input [4:0] Rs\_ex, ///////////////////

input [4:0] Rd\_ex, ////////////

// input [15:0] imm16\_ex, ////////////

// input [25:0] Target\_ex,//////////////

output reg loaduse\_me,/////////////

output reg [5:0] op, ///////////////op\_me

output reg [5:0] func\_me, /////////////

output reg [4:0] Rs\_me, ////////////

output reg [4:0] Rt\_me, ///////////

output reg [4:0] Rd\_me, ///////////

// output reg [15:0] imm16\_me, ///////////

// output reg [25:0] Target\_me, ///////////

output reg zero\_me,

output reg [31:0] Result\_me, //Adr

output reg [31:0] busB\_me, //Datain

output reg [4:0] Rw\_me,

output wire RegWr\_me,

output wire sb\_me,//控制信号 out

output wire MemWr\_me ,//注意消除

output reg xiaoc\_me,

output reg [31:0] pc\_me,

output reg [31:0] Result\_next\_me, ///////

output CPR\_wr\_me,

output Hi\_wr\_me,

output Lo\_wr\_me,

output Hi\_Lo\_wr\_me

);

////

assign CPR\_wr\_1=(op==6'b010000&&Rs\_me==5'b00100)==1?1:0;

assign Hi\_wr\_1=(op==6'b000000&&func\_me==6'b010001)==1?1:0;

assign Lo\_wr\_1=(op==6'b000000&&func\_me==6'b010011)==1?1:0;

assign Hi\_Lo\_wr\_1=(op==6'b000000&&func\_me==6'b011000)==1?1:0;

assign CPR\_wr\_me=((loaduse\_me==1)|(xiaoc\_me==1))==1?0:CPR\_wr\_1;

assign Hi\_wr\_me=((loaduse\_me==1)|(xiaoc\_me==1))==1?0:Hi\_wr\_1;

assign Lo\_wr\_me=((loaduse\_me==1)|(xiaoc\_me==1))==1?0:Lo\_wr\_1;

assign Hi\_Lo\_wr\_me=((loaduse\_me==1)|(xiaoc\_me==1))==1?0:Hi\_Lo\_wr\_1;

wire RegWr\_me\_1,MemWr\_me\_1;

//并不是R型指令都要写寄存器了

assign RegWr\_me\_1=((op==6'b010000)&&(func\_me[5:0]!=6'b011000))|((op==6'b000000)&&(func\_me[5:0]!=6'b011000)&&(func\_me[5:0]!=6'b010011)&&(func\_me[5:0]!=6'b010001)&&(func\_me[5:0]!=6'b001000))|(!op[5]&!op[4]&op[3]&!op[2]&!op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0])|(op[5]&!op[4]&!op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0])|(!op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0]);

assign RegWr\_me=((loaduse\_me==1)|(xiaoc\_me==1))?0:RegWr\_me\_1;

// assign RegWr\_me=RegWr\_me\_1;

assign sb\_me=(op==6'b101000);

assign MemWr\_me\_1=(op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&op[3]&!op[2]&!op[1]& !op[0]);

assign MemWr\_me=((loaduse\_me==1)|(xiaoc\_me==1))?0:MemWr\_me\_1;

always @(negedge clk)

begin

Result\_next\_me<=Result\_next\_ex;

// instructions\_me<=instructions\_ex;

op<=op\_ex;

zero\_me<=zero\_ex;

Result\_me<=Result\_ex;

func\_me<=func\_ex;

Rs\_me<=Rs\_ex;

Rt\_me<=Rt\_ex;

Rw\_me<=Rw\_ex;

Rd\_me<=Rd\_ex;

// imm16\_me<=imm16\_ex;

// Target\_me<=Target\_ex;

loaduse\_me<=loaduse\_ex;

xiaoc\_me<=xiaoc\_ex;

pc\_me<=pc\_ex;

//转发 sw，sb的Di，上次指令的result

if((op\_ex==6'b101000|op\_ex==6'b101011)&&RegWr\_me\_j&&(Rw\_me\_j==Rt\_ex))

busB\_me<=Result\_me\_j; //错误

else

busB\_me<=busB\_ex;

end

endmodule

// module ExMem(input clk, //

// //转发 sb，sw

// input RegWr\_me\_j,

// input [4:0] Rw\_me\_j,

// input [4:0] Rt\_ex,

// input [31:0] Result\_me\_j,

// input [31:0] pc\_ex,

// input xiaoc\_ex,

// input zero\_ex,

// input [31:0] Result\_ex,

// input [31:0] Result\_next\_ex,

// input [31:0] busB\_ex,

// input loaduse\_ex, //

// input [4:0] Rw\_ex,

// input [31:0] instructions\_ex,

// output reg loaduse\_me,//

// output reg [31:0] instructions\_me,

// output reg zero\_me,

// output reg [31:0] Result\_me, //Adr

// output reg [31:0] busB\_me, //Datain

// output reg [4:0] Rw\_me,

// output wire RegWr\_me,

// output wire sb\_me,//控制信号 out

// output wire MemWr\_me ,//注意消除

// output reg xiaoc\_me,

// output reg [31:0] pc\_me,

// output reg [31:0] Result\_next\_me, ///////

// output CPR\_wr\_me,

// output Hi\_wr\_me,

// output Lo\_wr\_me,

// output Hi\_Lo\_wr\_me

// );

// ////

// assign CPR\_wr\_1=(instructions\_me[31:26]==6'b010000&&instructions\_me[25:21]==5'b00100&&instructions\_me[10:3]==8'b00000000)==1?1:0;

// assign Hi\_wr\_1=(instructions\_me[31:26]==6'b000000&&instructions\_me[5:0]==6'b010001&&instructions\_me[20:6]==15'b000000000000000)==1?1:0;

// assign Lo\_wr\_1=(instructions\_me[31:26]==6'b000000&&instructions\_me[5:0]==6'b010011&&instructions\_me[20:6]==15'b000000000000000)==1?1:0;

// assign Hi\_Lo\_wr\_1=(instructions\_me[31:26]==6'b000000&&instructions\_me[15:6]==10'b0000000000&&instructions\_me[5:0]==6'b011000)==1?1:0;

// assign CPR\_wr\_me=((instructions\_me==32'h11111111)|(loaduse\_me==1)|(xiaoc\_me==1))==1?0:CPR\_wr\_1;

// assign Hi\_wr\_me=((instructions\_me==32'h11111111)|(loaduse\_me==1)|(xiaoc\_me==1))==1?0:Hi\_wr\_1;

// assign Lo\_wr\_me=((instructions\_me==32'h11111111)|(loaduse\_me==1)|(xiaoc\_me==1))==1?0:Lo\_wr\_1;

// assign Hi\_Lo\_wr\_me=((instructions\_me==32'h11111111)|(loaduse\_me==1)|(xiaoc\_me==1))==1?0:Hi\_Lo\_wr\_1;

// wire [5:0] op\_ex;

// assign op\_ex=instructions\_ex[31:26];

// wire [5:0] op;

// wire RegWr\_me\_1,MemWr\_me\_1;

// assign op=instructions\_me[31:26];

// //并不是R型指令都要写寄存器了

// assign RegWr\_me\_1=((op==6'b010000)&&(instructions\_me[5:0]!=6'b011000))|((op==6'b000000)&&(instructions\_me[5:0]!=6'b011000)&&(instructions\_me[5:0]!=6'b010011)&&(instructions\_me[5:0]!=6'b010001)&&(instructions\_me[5:0]!=6'b001000))|(!op[5]&!op[4]&op[3]&!op[2]&!op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0])|(op[5]&!op[4]&!op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0])|(!op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0]);

// assign RegWr\_me=((loaduse\_me==1)|(xiaoc\_me==1))?0:RegWr\_me\_1;

// // assign RegWr\_me=RegWr\_me\_1;

// assign sb\_me=(op==6'b101000);

// assign MemWr\_me\_1=(op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&op[3]&!op[2]&!op[1]& !op[0]);

// assign MemWr\_me=((loaduse\_me==1)|(instructions\_me==32'h11111111)|(xiaoc\_me==1))?0:MemWr\_me\_1;

// always @(negedge clk)

// begin

// Result\_next\_me<=Result\_next\_ex;

// instructions\_me<=instructions\_ex;

// zero\_me<=zero\_ex;

// Result\_me<=Result\_ex;

// Rw\_me<=Rw\_ex;

// loaduse\_me<=loaduse\_ex;

// xiaoc\_me<=xiaoc\_ex;

// pc\_me<=pc\_ex;

// //转发 sw，sb的Di，上次指令的result

// if((op\_ex==6'b101000|op\_ex==6'b101011)&&RegWr\_me\_j&&(Rw\_me\_j==Rt\_ex))

// busB\_me<=Result\_me\_j; //错误

// else

// busB\_me<=busB\_ex;

// end

// endmodule

module EXT\_P(imm16,imm32,ExtOp);

input [15:0] imm16;

output [31:0] imm32;

input ExtOp;

assign imm32=(ExtOp==1)?{{16{imm16[15]}},imm16}:{16'b0,imm16};

endmodule

module IDEX\_P(input clk,

input [31:0]pc\_id,

input xiaoc\_id,

input loaduse,

input [31:0]busA\_id,

input [31:0] busB\_id,

//

input [31:0] Lo\_out\_id,

input [31:0] Hi\_out\_id,

input [31:0] CPR\_out\_id,

input [5:0] op\_id,

input [5:0] func\_id,

input [4:0] Rs\_id,

input [4:0] Rt\_id,

input [4:0] Rd\_id,

input [4:0] shamt\_id,

input [15:0] imm16\_id,

output reg [5:0] op,

output reg [5:0] func,

output [4:0] ALUctr\_ex,

output reg [31:0] busA\_ex,

output reg [31:0] busB\_ex,

output reg loaduse\_ex,

output reg [4:0] shamt\_ex,

output reg [4:0] Rs\_ex,

output reg [4:0] Rt\_ex,

output reg [4:0] Rd\_ex,

output ExtOp\_ex,

output RegDst\_ex,

output reg MemRead\_ex ,

output reg [15:0]imm16\_ex,

output ALUSrc\_ex,

output reg xiaoc\_ex,

output reg [31:0]pc\_ex, //

output reg [31:0] Lo\_out\_ex,

output reg [31:0] Hi\_out\_ex,

output reg [31:0] CPR\_out\_ex

);

wire [4:0] funcop,ALUop;

wire Rtype;

assign funcop[0]=(!func[5]&func[4]&func[3]&!func[2]&!func[1]&!func[0])|(func[5]&!func[4]&!func[3]&!func[2]&func[1]&func[0])|(func[5]&!func[4]&!func[3]&func[2]&!func[1]&!func[0])|(func[5]&!func[4]&!func[3]&func[2]&!func[1]&func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&!func[1]&!func[0])|(func[5]&!func[4]&func[3]&!func[2]&func[1]&func[0])|(!func[5]&!func[4]&func[3]&!func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&func[1]&func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&!func[0]);

assign funcop[1]=(func[5]&!func[4]&func[3]&!func[2]&func[1]&!func[0])|(func[5]&!func[4] &!func[3]&func[2]&!func[1]&!func[0])|(func[5]&!func[4]&!func[3]&func[2]&func[1]& !func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[ 1]&!func[0])|(!func[5]&!func[4]&func[3]&!func[2]&!func[1]&func[0])|(!func[5]&!func[4]&func[3]&!func[2]&!func[1]&!func[0]);

assign funcop[2]=(func[5]&!func[4]&!func[3]&func[2]&func[1]&func[0])|(func[5]&!func[4]&!func[3]&func[2]&!func[1]&func[0])|(func[5]&!func[4]&!func[3]&func[2]&func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&func[1]&func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&!func[0]);

assign funcop[3]=(!func[5]&!func[4]&!func[3]&!func[2]&func[1]&!func[0])|(func[5]&!func[4]&func[3]&!func[2]&func[1]&func[0])|(!func[5]&!func[4]&func[3]&!func[2]&!func[1]&func[0])|(!func[5]&!func[4]&func[3]&!func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&func[2]&!func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&!func[2]&func[1]&func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&!func[0])|(!func[5]&!func[4]&!func[3]&func[2]&func[1]&func[0]);

assign funcop[4]=!func[5]&func[4]&func[3]&!func[2]&!func[1]&!func[0];//

assign Rtype=!op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0];

assign ALUop[0]=(!op[5]&!op[4]&!op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&!op[3]&op[2]& !op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2 ]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0]);

assign ALUop[1]=(!op[5]&!op[4]&op[3]&!op[2]&op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0])|(!op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0]);

assign ALUop[2]=(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0]);

assign ALUop[3]=(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0]);

assign ALUop[4]=(!op[5]&!op[4]&op[3]&op[2]&op[1]&op[0]);

assign ExtOp\_ex=(op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&op[3]&!op[2]&op[1]& op[0])|(!op[5]&!op[4]&op[3]&!op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[ 1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0]);

assign RegDst\_ex=(!op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0]==1)?0:1;

assign ALUctr\_ex=Rtype?funcop:ALUop;

assign ALUSrc\_ex=(!op[5]&!op[4]&op[3]&!op[2]&!op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&!op[0])|(!op[5]&!op[4]&op[3]&!op[2]&op[1]&op[0])|(op[5]&!op[4]&!op[3]&!op[2]&!op[1]&!op[0])|(op[5]&!op[4]&!op[3]&op[2]&!op[1]&!op[0])|(op[5]&!op[4]&op[3]&!op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&!op[0])|(!op[5]&!op[4]&op[3]&op[2]&!op[1]&op[0])|(!op[5]&!op[4]&op[3]&op[2]&op[1]&!op[0]);

always @(negedge clk)

begin

loaduse\_ex<= loaduse;

// instructions\_ex<=instructions\_id;

busA\_ex<=busA\_id;

busB\_ex<=busB\_id;

xiaoc\_ex<=xiaoc\_id;

pc\_ex<=pc\_id;

//

Hi\_out\_ex<=Hi\_out\_id;

Lo\_out\_ex<=Lo\_out\_id;

CPR\_out\_ex<=CPR\_out\_id;

Rd\_ex<=Rd\_id;

Rs\_ex<=Rs\_id;

Rt\_ex<=Rt\_id;

shamt\_ex<=shamt\_id;

imm16\_ex<=imm16\_id;

op<=op\_id;

func<=func\_id;

end

initial

loaduse\_ex=0;

always @(\*)

MemRead\_ex=(loaduse\_ex==1)?0:(op==6'b100011);

initial

begin

MemRead\_ex<=0;

end

endmodule

module IFID\_P(input clk,

input [31:0] pc,

input [31:0] instructions,

input loaduse ,//load use

input xiaoc, //id block

output reg [5:0] op\_id,

output reg [4:0] Rs\_id,

output reg [4:0] Rt\_id,

output reg [4:0] Rd\_id,

output reg [5:0] func\_id,

output reg [4:0] shamt\_id,

output reg [15:0] imm16\_id,

output reg [25:0] Target\_id,

output reg xiaoc\_id,

output reg [31:0] pc\_id

);

always @(negedge clk)

if(loaduse);

else

begin

pc\_id<=pc;

xiaoc\_id<=xiaoc;

op\_id<=instructions[31:26]; //////////

Rs\_id<=instructions[25:21]; /////////

Rt\_id<=instructions[20:16]; //////////

Rd\_id<=instructions[15:11]; ////////

func\_id<=instructions[5:0]; ///////////

shamt\_id<=instructions[10:6]; /////////

imm16\_id<=instructions[15:0]; //////////

Target\_id<=instructions[25:0]; ////////

end

endmodule

module IM\_P(input [31:0] Adr,output [31:0] instructions);

reg [31:0] im[1023:0]; //IM\_P

initial begin

$readmemb("codep.txt",im); //inst loaded

end

assign instructions=im[Adr[11:2]];

endmodule

module Loaduse(input MemRead\_ex,input [4:0] Rs\_id,input [4:0] Rt\_id,input [4:0] Rt\_ex,output reg loaduse); //Rs,Rt

always @(\*)

begin

if(MemRead\_ex==1'bx)

loaduse<=0;

else if((MemRead\_ex==1 )&&((Rt\_ex==Rs\_id)|(Rt\_ex==Rt\_id)))

loaduse<=1;

else loaduse<=0;

end

initial

begin

loaduse<=0;

end

endmodule

module MemWr(

input clk,

input [31:0] pc\_me,

input xiaoc\_me,

input loaduse\_me,

// input [31:0] instructions\_me,

input [5:0] op\_me, /////////op\_wr\_wr

input [5:0] func\_me,

input [4:0] rd\_me,

// input MemtoReg\_me,

input [31:0] Do\_me,//DataMem -data stored

input [31:0] Result\_me,//Adr/register stored

input [31:0] Result\_next\_me,

input [4:0] Rw\_me,

/////

input CPR\_wr\_me,

input Hi\_wr\_me,

input Lo\_wr\_me,

input Hi\_Lo\_wr\_me,

input RegWr\_me,

output reg loaduse\_wr,

output reg [5:0] op\_wr, /////////op\_wr\_wr

output reg [5:0] func\_wr,

output reg [4:0] rd\_wr,

output reg [31:0] Do\_wr,

output reg [31:0] Result\_wr,

output reg [4:0] Rw\_wr,

output MemtoReg\_wr,

output reg RegWr\_wr,

output reg xiaoc\_wr,

output reg [31:0] pc\_wr,//

output reg CPR\_wr,

output reg Hi\_wr,

output reg Lo\_wr,

output reg Hi\_Lo\_wr,

output reg [31:0] Result\_next\_wr

);

// wire [5:0] op\_wr;

// wire RegWr\_wr\_1;

// assign op\_wr=instructions\_wr[31:26];

assign MemtoReg\_wr=(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&op\_wr[1]&op\_wr[0])|(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&!op\_wr[1]&!op\_wr[0])|(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&op\_wr[2]&!op\_wr[1]&!op\_wr[0]);

// assign RegWr\_wr\_1=(!op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&!op\_wr[1]&!op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&!op\_wr[2]&!op\_wr[1]&op\_wr[0])|(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&op\_wr[1]&op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&op\_wr[2]&op\_wr[1]&op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&!op\_wr[2]&op\_wr[1]&!op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&!op\_wr[2]&op\_wr[1]&op\_wr[0])|(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&!op\_wr[1]&!op\_wr[0])|(op\_wr[5]&!op\_wr[4]&!op\_wr[3]&op\_wr[2]&!op\_wr[1]&!op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&op\_wr[2]&!op\_wr[1]&!op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&op\_wr[2]&!op\_wr[1]&op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&op\_wr[3]&op\_wr[2]&op\_wr[1]&!op\_wr[0])|(!op\_wr[5]&!op\_wr[4]&!op\_wr[3]&!op\_wr[2]&op\_wr[1]&op\_wr[0]);

// assign RegWr\_wr=((instructions\_wr==32'h11111111)|(loaduse\_wr==1)|(xiaoc\_wr==1))?0:RegWr\_wr\_1;

//

// wire CPR\_wr\_1,Hi\_wr\_1,Lo\_wr\_1,Hi\_Lo\_wr\_1;

initial

begin

loaduse\_wr<=0;

end

// assign CPR\_wr\_1=(op\_wr==6'b010000&&instructions\_wr[25:21]==5'b00100&&instructions\_wr[10:3]==8'b00000000)==1?1:0;

// assign Hi\_wr\_1=(op\_wr==6'b000000&&instructions\_wr[5:0]==6'b010001&&instructions\_wr[20:6]==15'b000000000000000)==1?1:0;

// assign Lo\_wr\_1=(op\_wr==6'b000000&&instructions\_wr[5:0]==6'b010011&&instructions\_wr[20:6]==15'b000000000000000)==1?1:0;

// assign Hi\_Lo\_wr\_1=(op\_wr==6'b000000&&instructions\_wr[15:6]==10'b0000000000&&instructions\_wr[5:0]==6'b011000)==1?1:0;

// assign CPR\_wr=((instructions\_wr==32'h11111111)|(loaduse\_wr==1)|(xiaoc\_wr==1))==1?0:CPR\_wr\_1;

// assign Hi\_wr=((instructions\_wr==32'h11111111)|(loaduse\_wr==1)|(xiaoc\_wr==1))==1?0:Hi\_wr\_1;

// assign Lo\_wr=((instructions\_wr==32'h11111111)|(loaduse\_wr==1)|(xiaoc\_wr==1))==1?0:Lo\_wr\_1;

// assign Hi\_Lo\_wr=((instructions\_wr==32'h11111111)|(loaduse\_wr==1)|(xiaoc\_wr==1))==1?0:Hi\_Lo\_wr\_1;

always @(negedge clk)

begin

Result\_next\_wr<=Result\_next\_me;

Do\_wr<=Do\_me;

Result\_wr<=Result\_me;

Rw\_wr<=Rw\_me;

loaduse\_wr<=loaduse\_me;

// instructions\_wr<=instructions\_me;

xiaoc\_wr<=xiaoc\_me;

pc\_wr<=pc\_me;

// MemtoReg\_wr<=MemtoReg\_me;

RegWr\_wr<=RegWr\_me;

CPR\_wr<=CPR\_wr\_me;

Hi\_wr<=Hi\_wr\_me;

Lo\_wr<=Lo\_wr\_me;

Hi\_Lo\_wr<=Hi\_Lo\_wr\_me;

op\_wr<=op\_me ; /////////op\_wr\_wr

func\_wr<=func\_me;

rd\_wr<=rd\_me;

end

endmodule

module mips(input clk,input rst);

//IF

wire [31:0] pc,npc;

wire loaduse,loaduse\_ex,loaduse\_me,loaduse\_wr;

wire [31:0] instructions;

PC\_P PC\_P( clk, rst, loaduse, npc, pc);

IM\_P IM\_P(pc,instructions);

wire [5:0] func\_id,func\_ex,func\_me,func\_wr;

//ID

wire [5:0] op\_id,op\_ex,op\_me,op\_wr;

wire zero\_ex,zero\_me;

wire xiaoc,xiaoc\_id,xiaoc\_ex,xiaoc\_me,xiaoc\_wr;

wire [31:0] Di;

wire [4:0] Rw\_wr,Rw\_id,Rw\_ex,Rw\_me;

wire RegWr\_wr,RegWr\_id,RegWr\_ex,RegWr\_me;

wire [31:0] busA\_id,busA\_ex,busA\_me,busA\_wr;

wire [31:0] busB\_id,busB\_ex,busB\_me,busB\_wr;

wire [31:0] busA\_end,busB\_end;

wire [31:0] busB\_ex\_end;

wire [31:0] pc\_id,pc\_ex,pc\_me,pc\_wr;

wire [4:0] Rs\_ex,Rt\_ex,Rd\_ex,Rs\_id,Rt\_id,Rd\_id,Rs\_me,Rt\_me,Rd\_me,Rs\_wr,Rt\_wr,Rd\_wr;

wire MemRead\_ex,MemRead\_id,MemRead\_me,MemRead\_wr;

wire ExtOp\_id,ExtOp\_ex,ExtOp\_me,ExtOp\_wr;

wire sb\_id,sb\_ex,sb\_me,sb\_wr;

wire MemWr\_id,MemWr\_ex,MemWr\_me,MemWr\_wr;

wire [31:0] Do\_me,Do\_wr;

wire [4:0] shamt\_ex,shamt\_id,shamt\_me,shamt\_wr;

wire [4:0] ALUctr\_ex;

wire [15:0]imm16\_ex,imm16\_id,imm16\_me,imm16\_wr;

wire RegDst\_ex,RegDst\_ID;

wire ALUSrc\_ex;

wire [31:0] imm32\_ex;

wire [1:0] ALUSrcA\_ex,ALUSrcB\_ex;

wire MemtoReg\_wr,MemtoReg\_me;

wire [25:0] Target\_id;

wire [31:0] Result\_wr,Result\_ex,Result\_me,Result\_next\_ex,Result\_next\_me,Result\_next\_wr,Result\_ex\_new;

IFID\_P IFID\_P( clk, pc, instructions, loaduse , xiaoc,op\_id,Rs\_id,Rt\_id,Rd\_id,func\_id,shamt\_id,imm16\_id,Target\_id,xiaoc\_id, pc\_id);

Loaduse Loaduse(MemRead\_ex,Rs\_id,Rt\_id, Rt\_ex, loaduse);

wire WE\_HI\_wr,WE\_LO\_wr,WE\_HI\_LO\_wr,WE\_CPR\_wr;

wire [31:0] A\_ALU,B\_ALU; //

wire [31:0] Lo\_out,CPR\_out,CPR\_out\_14,Hi\_out;

RFile\_P RFile\_P(clk,Result\_next\_wr,Result\_wr,func\_id[2:0],Rd\_id,op\_wr,func\_wr,Rd\_wr,func\_wr[2:0],pc\_wr, Rs\_id, Rt\_id, Rw\_wr, Di, RegWr\_wr,WE\_HI\_wr,WE\_LO\_wr,WE\_HI\_LO\_wr,WE\_CPR\_wr, busA\_id,busB\_id,Lo\_out,Hi\_out,CPR\_out,CPR\_out\_14);

wire[1:0] Ce\_A,Ce\_B;

ByPass\_T ByPass\_T( op\_ex, op\_id,op\_me,func\_id, Rw\_me, Rw\_ex, Rs\_id, Rt\_id,RegWr\_me, Ce\_A,Ce\_B);

MUX\_B MUX\_B\_2(Ce\_A,busA\_id,Result\_ex,Result\_me,Do\_me,busA\_end);

MUX\_B MUX\_B\_1(Ce\_B,busB\_id,Result\_ex,Result\_me,Do\_me,busB\_end);

reg J\_Q\_1;

reg [31:0] NPC\_J\_1;

wire [31:0] NPC\_J;

initial

begin

NPC\_J\_1<=0;

J\_Q\_1<=0;

end

wire J\_Q;

always @(negedge clk) NPC\_J\_1=NPC\_J;

always @(negedge clk) J\_Q\_1=J\_Q;

NPC\_P NPC\_P(CPR\_out\_14,Rd\_ex,Rs\_id,Result\_ex,pc\_id,pc, busA\_end,busB\_end,J\_Q\_1,NPC\_J\_1,B\_ALU,op\_ex,Rs\_ex,instructions[5:0],func\_id,imm16\_id,instructions[15:0],Target\_id,instructions[25:0],op\_id,instructions[31:26],Rt\_id,npc,NPC\_J,xiaoc,J\_Q); //下条指令

//EX

wire [31:0] Lo\_out\_ex, Hi\_out\_ex, CPR\_out\_ex;

IDEX\_P IDEX\_P(clk, pc\_id,xiaoc\_id,loaduse,busA\_id,busB\_id,Lo\_out,Hi\_out,CPR\_out,op\_id,func\_id,Rs\_id,Rt\_id,Rd\_id,shamt\_id,imm16\_id,op\_ex,func\_ex,ALUctr\_ex,busA\_ex,busB\_ex,loaduse\_ex,shamt\_ex,Rs\_ex,

Rt\_ex,Rd\_ex, ExtOp\_ex, RegDst\_ex,MemRead\_ex,imm16\_ex,ALUSrc\_ex,xiaoc\_ex,pc\_ex,Lo\_out\_ex, Hi\_out\_ex, CPR\_out\_ex);

EXT\_P EXT\_P(imm16\_ex,imm32\_ex,ExtOp\_ex);

MUX\_A MUX\_A(ALUSrcA\_ex,busA\_ex,Result\_me, Di,A\_ALU);

MUX\_B MUX\_B(ALUSrcB\_ex,busB\_ex,Result\_me,Di,imm32\_ex,B\_ALU);

ALU\_P ALU\_P(op\_ex,func\_ex,Rs\_ex,pc\_ex,busA\_ex,busB\_ex,Lo\_out\_ex,Hi\_out\_ex,CPR\_out\_ex,ALUctr\_ex,A\_ALU,B\_ALU,shamt\_ex,Result\_ex,zero\_ex,Result\_next\_ex);

assign Rw\_ex=(RegDst\_ex==1)?Rt\_ex:Rd\_ex;

ByPass\_M ByPass\_M( RegWr\_wr,busB\_ex,Rt\_ex,Rd\_wr,op\_ex,Di,busB\_ex\_end);

//MEM

wire CPR\_wr\_me,Hi\_wr\_me,Lo\_wr\_me,Hi\_Lo\_wr\_me;

ByPass\_X ByPass\_X(func\_ex,Result\_ex,Result\_me,Result\_next\_me,Result\_wr,Result\_next\_wr,Hi\_wr\_me,WE\_HI\_wr,Lo\_wr\_me,WE\_LO\_wr,Hi\_Lo\_wr\_me,WE\_HI\_LO\_wr,Result\_ex\_new);

ExMem ExMem(clk,RegWr\_me,Rw\_me,Rt\_ex,Result\_me,pc\_ex,xiaoc\_ex,zero\_ex,Result\_ex\_new,Result\_next\_ex,busB\_ex\_end,loaduse\_ex, Rw\_ex, op\_ex,func\_ex,Rs\_ex,Rd\_ex, loaduse\_me,op\_me,

func\_me,Rs\_me,Rt\_me,Rd\_me,zero\_me,Result\_me, //Adr

busB\_me, //Datain

Rw\_me,RegWr\_me, sb\_me,//控制信号 out

MemWr\_me ,//注意消除

xiaoc\_me,

pc\_me,

Result\_next\_me,

CPR\_wr\_me,

Hi\_wr\_me,

Lo\_wr\_me,

Hi\_Lo\_wr\_me

);

DataMem DataMem( clk,sb\_me, MemWr\_me, Result\_me, busB\_me, Do\_me);

ByPass ByPass(op\_ex,op\_me,RegWr\_me,ALUSrc\_ex,RegWr\_wr, Rw\_me, Rw\_wr,Rs\_ex,Rt\_ex,ALUSrcA\_ex,ALUSrcB\_ex);

//REGWR

MemWr MemWr\_1( clk,pc\_me,xiaoc\_me,loaduse\_me,op\_me,func\_me,Rd\_me,Do\_me, Result\_me, Result\_next\_me,Rw\_me, CPR\_wr\_me,

Hi\_wr\_me,

Lo\_wr\_me,

Hi\_Lo\_wr\_me,RegWr\_me,loaduse\_wr,op\_wr,func\_wr,Rd\_wr, Do\_wr,Result\_wr,Rw\_wr, MemtoReg\_wr, RegWr\_wr,xiaoc\_wr,pc\_wr,WE\_CPR\_wr,WE\_HI\_wr,WE\_LO\_wr,WE\_HI\_LO\_wr,Result\_next\_wr);

assign Di=(MemtoReg\_wr==1)?Do\_wr:Result\_wr;

endmodule

# 附录 3

