编号 091_161810225_王烨文 实验报告

# 南京航空航天大学

# 课 设 报 告

题 目　　五级流水线（45 条+转发冒险）

| | |
|---|---|
| 学生姓名 | 王烨文 |
| 学　　号 | 161810225 |
| 学　　院 | 计算机科学与技术学院 |
| 专　　业 | 计算机科学与技术 |
| 班　　级 | 1618104 |
| 指导教师 | 施慧彬 |

二〇二〇年七月

# 南京航空航天大学
# 课设报告诚信承诺书

本人郑重声明：所呈交的课程设计（论文）（题目：五级流水线（45 条+转发冒险））是本人在导师的指导下独立进行研究所取得的成果。尽本人所知，除了课程设计（论文）中特别加以标注引用的内容外，本课程设计（论文）不包含任何其他个人或集体已经发表或撰写的成果作品。

作者签名：　王烨文　　2020 年 6 月 9 日

（学号）：161810225

# 五级流水线（45 条+转发冒险+一位预测）的设计


## 摘　要

研究的目的：为了更深入了解计算机组成原理课程并提高自己动手编程和思考能力，完成一个五级流水线的设计，从而更加清晰地认识到 CPU 的运行机制与数据通路的搭建，并掌握 Verilog 语言。

研究的结果与主要结论：能写出一个能执行 add，sub，and，or，slt，lw，sw，beq 和 jump 等 45 条指令并完成转发冒险和一位动态预测的五级流水线。


**关键词：**流水线，处理器，Verilog，mips

# The Design of Five Stage Pipeline in MIPS Instruction Set


## Abstract

The purpose of the research:By accomplishing the work of making a Five Stage Pipeline MIPS CPU,understand deeply the Principle of Computer Organization lesson and improve the ability of programming and logic thinking,which can show us a more explicit operating mechanism of CPU and the components of the datapath.

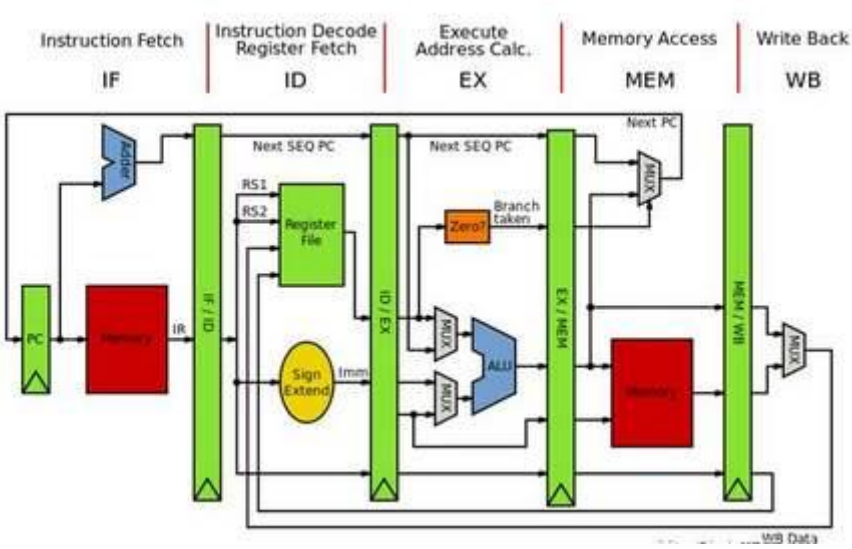
The results of the study and the main conclusions: Can write a 36 instructions that can execute add, sub, and, or, slt, lw, sw, beq and jump Five Stage Pipeline MIPS CPU,which can also make forward and Hazard and one-bit dynamic predict.


Key Words：Five Stage Pipeline MIPS；processor；Verilog；mips

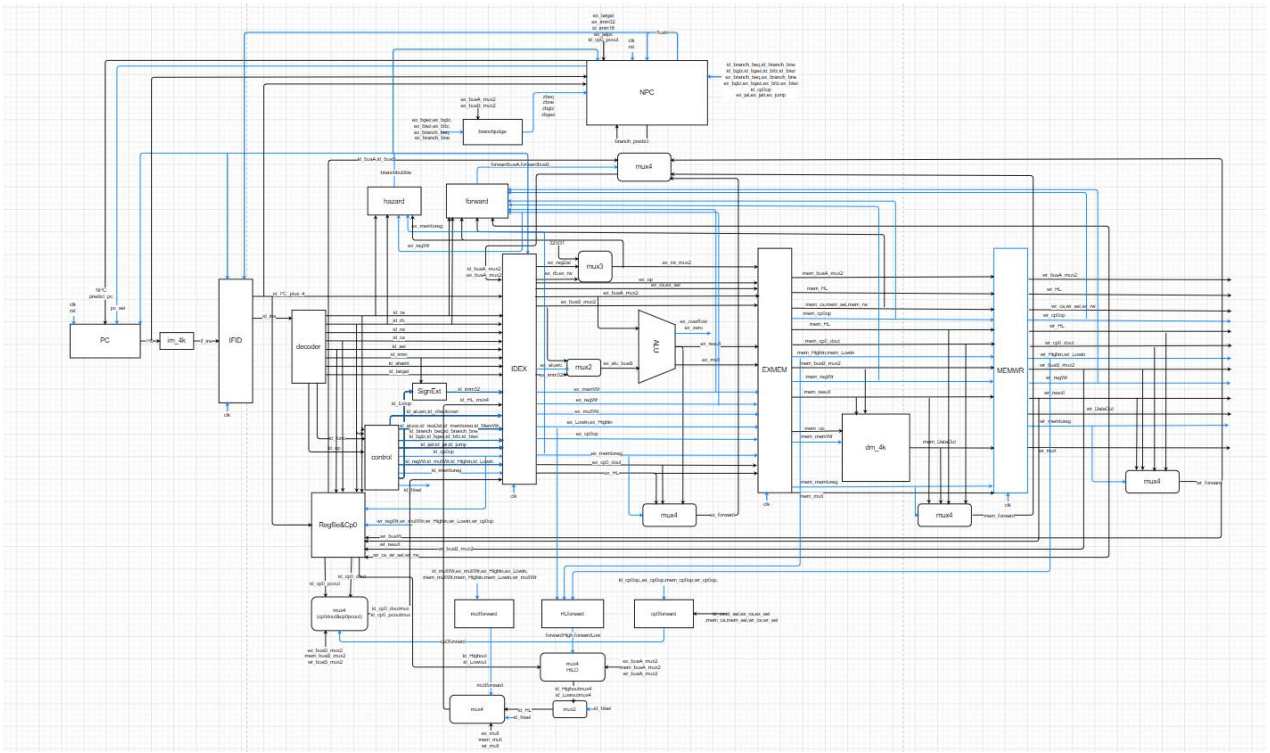
# 目 录

# 第一章　引　言

## 1.1　五级流水线的理论结构




五级流水线（45条+转发冒险+一位动态预测）的个人架构



### 1.1.1 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_3000，此处为第一条指令的地址。阻塞时，PC 值不变。

（2）模块接口

表 1.1 PC 接口的模块定义

| 信号名 | 方向 | 描述 |
|---|---|---|
| clk | Input | 时钟信号 |
| rst | Input | 复位信号（高电平有效） |
| Branchbubble | Input | 数据冒险信号 |
| [29:0]Predict_pc | Input | 预测地址 |
| Pc_sel | Input | 选择此次选用预测的地址还是正常运算地址 |
| [29:0]NPC | Input | 输入的指令地址 （下一条） |
| [29:0]PC | Output | 输出的指令地址（当前） |

（3）功能定义

表 1.2 PC 接口的功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 复位 | rst=1 时，将 pc 置为 0X0000_3000 |
| 2 | 输出指令地址 | 时钟信号到来时，将 adin 赋给 pc |
| 3 | 停止 | BrunchBubble 等于 1 时，需要从 lw 的 mem 阶段得到数据，所以需要阻塞一个时钟周期。 |

### 1.1.2 NPC 模块定义

（1）基本描述

NPC 主要功能是在分支指令 id 阶段预测是否跳转，并根据分支指令在 ex 阶段的实际跳转情况更新一位分支预测表。其余同单周期相同，即在分支跳转指令 ex 段，syscall，eret 等指令 id 阶段进行指令跳转，并不清除延迟槽。如果与预测相同，则继续 CPU 运行，若预测失败则清空在

if 阶段的指令并重新读取。

（2）模块接口

表 1.3 NPC 接口的模块定义

| 信号名 | 方向 | 描述 |
|---|---|---|
| clk | Input | 时钟周期 |
| rst | Input | 清零信号 |
| BrunchBubble | Input | 阻塞周期信号 |
| [2:0]cp0op | Input | Cp0 寄存器操作 |
| [29:0]cp0_pcout | Input | Syscall eret 等指令跳转地址 |
| id_branch_beq,id_branch_bne | Input | Id 阶段分支指令信号 |
| id_bgtz,id_bgez,id_bltz,id_bltz | Input | Id 阶段分支指令信号 |
| id_branch_beq,id_branch_bne | Input | Ex 阶段分支指令信号 |
| id_bgtz,id_bgez,id_bltz,id_bltz | Input | Ex 阶段分支指令信号 |
| jump,jal,jalr | Input | Ex 阶段跳转信号 |
| zbgtz,zbgez,zbne,zbeq | Input | Ex 段数据比较信号 |
| [15:0]ex_immediate,id_immediate | Input | Branch 指令所需的偏移量 |
| [25:0]target | Input | Jump 指令目标地址的[27:2]位 |
| [29:0]cur_pc | Input | 当前指令地址 |
| [29:0]pre_pc | Input | 分支指令后第二条指令 PC 值（由于传的是 id_PC_plus4） |
| [29:0]jal_pc | Input | 为 jalr 或 jr 指令时需要前往的 pc |
| [29:0]branch_predict2 | Input | 循环指令上一次循环的预测 |
| flushbefore | Input | 清除 if 段信号(lw+branch)的前一周期清除信号 |
| [29:0]next_pc | Output | 输出下一条指令地址 |
| Flush,pc_sel | Output | If 段数据冲洗信号,NPC 选择 |
| branch_predict | Output | 本次分支指令在 id 阶段预测的 next_pc 地址，ex 段回收比较 |

（3）功能定义

表 1.4 NPC 接口的功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1<br>2 | 输出正常指令地址<br>输出 Branch 实际跳转 | 1.将 PC 值加 4 后输出<br>2.分支指令 Ex 段完成，若预测正确则直接输出，若预测失败则输出冲洗信号为 1 并在 BHT 表中更新新的预测地址 |
| 3 | 输出 Jump 或 Jal 指令地址 | 3.当 Jump 信号为高电平时，将 target 与 PC 进行拼接,输出到 NPC 的地址 |
| 4 | 预测 Branch 指令跳转 | 4.如果该 PC 存在于 BHT 则直接读出 BHT 中的预测地址，若不存在则预测跳转并写入 BHT 中 |
| 5 | 输出 JALR/JR 指令地址 | 5.当指令为 JALR/JR 时，取寄存器的数据输出到 NPC |

### 1.1.3 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，mult 等基本操作。

(2)模块接口 （包括 ext 模块）

表 1.5 ALU 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [29:0]cur_pc | Input | 当前 pc 值，jalr 指令时需要进行+1 存储到 31 号寄存器 |
| [31:0]DataA | Input | 源操作数输入端 |
| [31:0]DataB | Input | 源操作数输入端 |

| | | |
|---|---|---|
| [4:0]shamt | Input | 移位操作数输入端（偏移量） |
| [4:0]aluop | Input | ALU 控制信号 |
| Checkover | Input | 是否判断溢出信号 |
| Zero | Output | ALU 结果为 0 输出高电平 |
| Overflow | Output | ALU 结果溢出则输出高电平 |
| [31:0]result | Output | ALU 计算输出的结果 |
| [63:0]mult | Output | ALU 计算的乘法结果 |

(3)功能定义

表 1.6 ALU 接口的功能定义

| 序号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 输出 result 计算结果 | 根据 alu 的控制信号，输出 DataA 和 DataB 的计算结果或者是 cur_pc+1(jalr) |
| 2 | 输出 Overflow | 输出溢出信号。 |
| 3 | 输出乘法结果 | 输出乘法结果。 |

### 1.1.4 MUX模块定义

(1)基本描述

实现多选一数据选择器。

(2)模块接口

表1.7 MUX2接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [31:0]C1 | Input | 0 号接口数 |
| [31:0]C2 | Input | 1 号接口数 |
| s | Input | 选择信号 |
| [31:0]y | Output | 结果 |

表1.8 MUX3接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [31:0]C1 | Input | 0 号口 |
| [31:0]C2 | Input | 1 号口 |
| [31:0]C3 | Input | 2 号口 |
| s | Input | 选择信号 |
| [31:0]y | Output | 输出数 |

表1.8 MUX4 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [31:0]C1 | Input | 0 号口 |
| [31:0]C2 | Input | 1 号口 |
| [31:0]C3 | Input | 2 号口 |
| [31:0]C4 | Input | 3 号口 |
| s | Input | 选择信号 |
| [31:0]y | Output | 输出数 |

(3)功能定义

表1.8 MUX接口的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 输出选定值 | 输出选定值<br>0 对应 0，1 对应 1 等等 |

### 1.1.5 RGEFILE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1.9 REGFILE 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| clk | Input | 时钟信号 |
| regWr,multWr,Lowin,Highin | Input | 写入信号(不同寄存器有区别，High，Low 寄存器) |
| [4:0]Rs | Input | 预读取寄存器编号 |
| [4:0]Rt | Input | 预读取寄存器编号 |
| [4:0]Rw | Input | 预写入寄存器编号 |
| [2:0]cp0op | Input | 当前指令 cp0op 操作码 |
| [31:0]cp0_dout | Input | Cp0 寄存器得到的数据 |
| [5:0]op | Input | 当前指令 op 值（可省去） |
| [31:0]busW,wr_busA_mux2 | Input | 写入寄存器的值，HILO 为 busA_mux2 |
| [63:0]busmult | Input | 写入寄存器的乘法结果 |
| [31:0]busA | Output | Rs 寄存器值 |
| [31:0]busB | Output | Rt 寄存器值 |
| [31:0]Highout | Output | Hi 寄存器值 |
| [31:0]Lowout | Output | Lo 寄存器值 |
| [29:0]jalpc | Output | Jalr，jr 指令从寄存器中读取的将要跳转的指令 |

(3)功能定义

表1.10　REGFILE 接口的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 寄存器读取操作 | busA 和 busB 分别输出地址 Ra 和 Rb 的数据，jalpc 输出 jalr 与 jr 指令将要跳转的数据，Highout，Lowout 为 Hi，Lo 寄存器的值 |
| 2 | 寄存器写入操作 | 时钟周期下半周期将 busW 写入指定寄存器 |

### 1.1.6 im_4k 模块的定义

(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1.11 im_4k接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [11:2]addr | Input | 输入指令地址 |
| [31:0]dout | Output | 输出指令 |

(3)功能定义

表1.12 im_4k接口的功能定义

| 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 | 输出操作 | 根据输入指令地址，输出当前指令 |

### 1.1.7 dm_4k 模块定义

(1)基本描述

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。

(2)模块接口

表1.13 dm_4k 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [11:2]addr | Input | 输入数据地址 |
| [31:0]DataIn | Input | 输入的数据 |
| [31:0]DataOut | Output | 读出的数据 |
| WrEn | Input | 写入使能信号，高电平有效 |
| clk | Input | 时钟信号 |
| [5:0]op | Input | 当前指令操作码，用于判断是否要字节读取 |

(3)功能定义

表1.14 dm_4k模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 读数据内存数据 | 根据输入的数据地址，读出数据内存中的数据。 |
| 2 | 写数据内存数据 | 时钟信号上升沿触发，且 WrEn=1 时向数据内存写入数据。另外需判断是否对内存中的字节进行操作。 |

### 1.1.8 decoder 模块定义

(1)基本描述

根据输入的指令进行拆分，拆分成各个单元。

(2)模块接口

表1.15 decoder接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [31:0]ir | Input | 指令 |
| [5:0]op | Output | 指令高 6 位 |
| [4:0]rs | Output | 预读取寄存器 |
| [4:0]rt | Output | 预读取或预存储寄存器 |
| [4:0]rd | Output | 预存储寄存器 |
| [4:0]shamt | Output | 偏移量 |
| [4:0]func | Output | 指令低 6 位 |
| [15:0]immediate | Output | 立即数 |
| [25:0]target | Output | 跳转目标地址 |
| checkover | Output | 输出判断溢出指令信号 |

(3)功能定义

表1.16 decoder 接口的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 分解指令 | 根据输入的指令输出各部分 |

### 1.1.9 control 模块定义

(1)基本描述

根据输入的指令高6位(op和func字段)，利用真值表化简，输出Branch, Jump, RegDst,

ALUSrc, MemtoReg, RegWr, MemWr, ExtOp, aluop 等控制信号。

(2)模块接口

表1.15 control 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [5:0]op | Input | 指令高 6 位 |
| [5:0]func | Input | 指令低 6 位 |
| Branch_beq,branch_bne,bgtz,bgez,bltz,blez | Output | 当前指令是否时分支指令 |
| Jump,jal,jalr | Output | 当前指令是否时跳转指令 |
| [4:0]ALUop | Output | 输出 ALU 的控制信号 |
| [1:0]memtoReg | Output | 输出主存写入寄存器信号 |
| [1:0]regDst | Output | 控制选择 Rd/Rt 为目的寄存器 |
| memWr | Output | 输出主存写入信号 |
| ExtOp | Output | 符号扩展/零扩展 |
| regWr,Highin,Lowin,r31Wr | Output | 寄存器写使能信号 |
| alusrc | Output | 控制选择 ALU 的 B 口操作数 |
| Checkover | Output | 输出判断溢出指令信号 |
| Hlsel | Output | Hi，Lo 寄存器选择信号 |
| [2:0]Cp0op | Output | Cp0 寄存器操作数 |

(3)功能定义

表1.16 control 接口的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 输出控制信号 | 根据输入的 op 输出各种控制信号 |

### 1.1.10 SignExt 模块定义

(1)基本描述

将输入的 16 位数据根据扩展信号扩展为 32 位。

(2)模块接口

表1.19　SignExt模块的接口定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [15:0]immedia | Input | 输入的16位原始数据 |
| [31:0]Extimmediate | Output | 输出的32位扩展数据 |
| ExtSel | Input | 扩展的信号 |

(2)功能定义

表1.16　SignExt模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 输出扩展的32位地址 | 根据ExtSel将imm16扩展为imm32 |

### 1.1.11 Forward&multforward&Hlforward&cp0forward 模块定义

(1)基本描述

各种条件下的转发

(2)模块接口

表1.19　multforwad模块的接口定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| Ex_multWr,mem_multWr,wr_multWr | Input | Ex,mem,wr段的乘法结果写入信号 |
| Multforward | Output | 要转发到id段的mult结果 |

表1.19　HLforwad模块的接口定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| Ex_Highin,ex_Lowin,mem_Highin,mem_Lowin,wr_Highin,wr_Lowin | Input | Ex,mem,wr段的Hi,Lo寄存器写入信号 |
| Multforward | Output | 要转发到id段的选择信号 |

表1.19　branchforwad模块的接口定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|

| | | |
|---|---|---|
| [2:0]ex_cp0op,mem_cp0op,wr_cp0op | Input | Ex,mem,wr 段的 cp0 信号 |
| [4:0] id_rs,id_rt,mem_rw,ex_rw,wr_rw | Input | Id,mem,ex,wr 的寄存器信号 |
| mem_regWr,ex_regWr,wr_regWr | Input | 写入信号 |
| [1:0]mem_memtoreg,wr_memtoreg | Input | 写入选择信号 |
| [1:0]branchforwardA,branchforwardB | Output | 要转发到 id 段的选择信号 |

表1.19 cp0forward接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [2:0]id_cp0op,ex_cp0op,mem_cp0op,wr_cp0op,id_sel,ex_sel,mem_sel,wr_sel | Input | 各种条件信号 |
| [4:0]id_cs,ex_cs,mem_cs,wr_cs | Input | 各种条件信号 |
| [1:0] cp0forward | Output | 选择 cp0 转发信号 |

(3)功能定义

表1.16 SignExt模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 输出转发信息 | 根据 id，ex，mem，wr 段信号选择转发信号所在段 |

### 1.1.12 cp0 模块定义

(1)基本描述

Cp0 寄存器的读写操作。

(2)模块接口

表1.20 cp0 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| [4:0]id_cs,wr_cs | Input | 协处理器的地址 |
| [2:0]id_sel,wr_sel | Input | 协处理器的选择（8 个） |
| [2:0]wr_cp0op,id_cp0op | Input | 协处理器操作 |
| [31:0]busB | Input | 写入 cp0 寄存器的数据 |
| [29:0]PC | Input | 当前指令 PC 值 |
| [31:0]Cp0_dout,cp0_pcout | Output | Cp0 输出数据 |

(3)功能定义

表1.16　cp0 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 根据 cp0op 对 cp0 寄存器进行相应操作 | Id_cp0op=4,Eret<br>Id_cp0op=3,Syscall 并阻塞后面的 cp0op<br>Wr_cp0op=2，写入<br>Wr_cp0op=1,读出数据 |

### 1.1.13 BrunchBubble 模块定义

(1)基本描述

用于 lw+R-Type

(2)模块接口

表1.20　cp0HLtoalu 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [4:0]id_rs,id_rt,ex_rw | Input | Id,ex 的寄存器信号 |
| ex_regWr | Input | ex 写入信号 |
| Hazard | Output | Load-use 阻塞信号 |

(3)功能定义

表1.16　cp0HLtoalu 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 进行 load-use 的阻塞 | 根据条件进行 load-use 阻塞 |

### 1.1.14 IFIDReg 模块定义

(1)基本描述

IFIDREG 段寄存器，用于传输数据。

(2)模块接口

表1.21 IFID 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| clk,BranchBubble,flush | Input | 用于阻断的信号和时钟信号 |
| [29:0]pc_plus_4 | Input | 下一指令地址 |
| [31:0] if_ins | Input | 当前指令 |
| [29:0] id_pc_plus_4 | Output | 传到 id 段得下一指令地址 |
| [31:0] id_ins | Output | 传到 id 段得指令 |

(3)功能定义

表1.16 IFID 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 向 id 段传输信息 | 向 id 段传输 PC 和指令信息，并进行 load-use 阻塞和 flush 预测失败的指令冲刷 |

### 1.1.15 IDEXReg 模块定义

(1)基本描述

IDEXREG 段寄存器，用于传输数据。

(2)模块接口

表1.22 IDEX 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| clk,BranchBubble | Input | 用于阻断的信号和时钟信号 |
| id_regWr,[1:0]id_reg | Input | Id 段需要传输到 ex 段的信号 |

| | | |
|---|---|---|
| Dst,id_alusrc,id_memwr,id_checkover,id_multWr,id_Lowin,id_Highin,id_branch_beq,id_branch_bne,id_bltz,id_blez,id_bgez,id_bgtz,id_jalr,id_jal,id_jump,[1:0]id_memtoreg,[2:0]id_cp0op | | |
| [31:0]id_imm32,id_HL,id_busA_mux2,id_cp0_dout,id_busB_mux2,[29:0]id_cp0_pcout, | Input | Id 段需要传输到 ex 段的数据 |
| [4:0]id_ra,id_rb,id_rw,id_cs,id_aluop,id_shamt,[5:0]id_op,[2:0]id_sel,[25:0]id_Target | Input | Id 段 decoder 得到的信息 |
| Ex_regWr 等一众信号 | Output | 由 id 传来的上述数据 |

(3)功能定义

表1.16　IDEX 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 向 id 段传输信息 | 向 ex 段传输 PC 和指令信息，并进行 load-use 阻塞 |

### 1.1.16 EXMEMReg 模块定义

(1)基本描述

EXMEMREG 段寄存器，用于传输数据。

(2)模块接口

表1.22　EXMEM 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| clk | Input | 用于时钟信号 |
| BranchBubble | Input | 用于阻断的信号 |

| | | |
|---|---|---|
| ex_zero,ex_HL,ex_result,ex_mult,ex_busA_mux2,ex_busB_mux2,,ex_rw,ex_regWr,ex_multWr,ex_memwr,ex_memtoreg,ex_op,ex_Lowin,ex_Highin,ex_cp0op,ex_cs,ex_sel,ex_cp0_dout | Input | EX 传入的所有信号 |
| mem_zero,mem_HL,mem_result,mem_mult,mem_busA_mux2,mem_busB_mux2,mem_rw,mem_regWr,mem_multWr,mem_memwr,mem_memtoreg,mem_op,mem_Lowin,mem_Highin,mem_cp0op,mem_cs,mem_sel,mem_cp0_dout | Output | 由 EX 传来的上述数据 |

(3)功能定义

表1.16　EXMEM 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 向 mem 段传输信息 | 向 mem 段传输 ex 段经过处理的数据 |

### 1.1.17 MEMWRReg 模块定义

(1)基本描述

MEMWR 段寄存器，用于传输数据。

(2)模块接口

表1.22　MEMWR 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| clk | Input | 用于时钟信号 |

| | | |
|---|---|---|
| BranchBubble | Input | 用于阻断的信号 |
| mem_dout,mem_HL,mem_result,mem_mult,mem_busA_mux2,mem_busB_mux2,mem_rw,mem_regWr,mem_multWr,mem_memtoreg,mem_op,mem_Lowin,mem_Highin,mem_cp0op,mem_cs,mem_sel,mem_cp0_dout | Input | MEM 传入的所有信号 |
| wr_dout,wr_HL,wr_result,wr_mult,wr_busA_mux2,wr_busB_mux2,wr_rw,wr_regWr,wr_multWr,wr_memtoreg,wr_op,wr_Lowin,wr_Highin,wr_cp0op,wr_cs,wr_sel,wr_cp0_dout | Output | 由 MEM 传来的上述数据 |

(3)功能定义

表1.16　MEMWR 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 向 wr 段传输信息 | 向 wr 段传输 mem 段经过处理的数据 |

### 1.1.18　zerobranchjudge 模块定义

(1)基本描述

用于在 ex 阶段判断是否实现跳转或顺序执行。

(2)模块接口

表1.23　zerobranchjudge 接口的模块定义

| 信号名 | 方 向 | 描 述 |
|---|---|---|
| [31:0]busA,busB | Input | rs，rt 寄存器所得值 |

| | | |
|---|---|---|
| bgez,bgtz,blez,bltz,branch_beq,branch_bne | Input | 跳转类别信号 |
| zbgez,zbgtz,zbeq,zbne | Output | 判断信号 |

(3)功能定义

表1.16 zerobranchjudge 模块的功能定义

| 序 号 | 功能名称 | 功能描述 |
|---|---|---|
| 1 | 计算跳转信息 | 根据传入的busA与busB对是否跳转进行判断 |

## 1.2 各种冒险解决的办法

### 1.2.1 Load-use 冒险

对于 Load-use 冒险，大部分时候出现于 lw 或者 lb、lbu 指令之后出现，由于 ALU 的 result 计算的为 DM 模块中的地址，所以无法在下一条指令读取对应寄存器（即处于 id 阶段时）就转发，需要等待一个时钟周期阻塞，由于大部分数据都一再后面通过转发实现，所以我们只需要查看 EX 段与 ID 段之间信号相关是否存在问题。由 Load-use 冒险定义可知道，当处于第二阶段的指令中读取 id_busA,id_busB 时，需要等待一个时钟周期才能够转发到 id 阶段，所以条件就是(ex_rw==id_rs||ex_rw==id_rt)&&(ex_regWr==1)&&(ex_memtoreg==1)，其意义为 Ex 段表示该指令需要有数据写入寄存器，并且该数据为从 DM 中获取的数据，并且存入的寄存器与后一条指令需要读取的寄存器相同。

Examples：lw+beq，lw+add 等指令。

### 1.2.2 分支跳转冒险

对于分支跳转冒险，在分支指令 id 阶段进行分支预测，即建立 BHT 与 BTB 表，类 Cache 存储，在遇到分支跳转指令时总是先进行查表，如果存在直接输出表中的预测地址，如果不存在总是先预测不跳转（这样可以检查预测有效性与正确性），并且存入 BHT 之中。然后在 EX 阶段下半周期进行是否跳转的校验，如果预测地址和实际跳转地址不同，则需要在传入 ID 段时洗刷已经在 IF 段存在的指令，其余情况则继续进行指令的接下操作。（其实真觉得预测在有延迟槽的情况下不如静态预测）

### 1.2.3 数据冒险

**Condition1.** 存在于 Regfile 中，即当前三条指令中有指令需要写入目的寄存器与当前指令的寄存器读取地址冲突，则需要考虑转发，即(id_ra==ex_rw&&ex_regWr==1)||(id_ra==mem_rw&&mem_regWr==1)||(id_

ra==wr_rw&&wr_regWr==1),同时还需要考虑 cp0 寄存器的 mfc0 指令，条件见下图。需要注意的是，此处如果出现 Load-use 冒险，即前一条指令为 lw 或者 lb 等指令，则会直接阻塞这一周期并且到下一周期进行转发，然后根据 memtoreg 来选定每个阶段需要转发到 id 段的数据如 HILO 寄存器读取的数，CP0 寄存器读取的数，以及 ALU 计算结果和 lw 等指令所得到的来自于 DM 的输出数据，然后在 id 阶段通过 forwardA，forwardB 得到的选择信号进行选择。

**Condition2.** 存在于 HILO 寄存器中，当前面的指令需要向 HILO 寄存器写入数据时，而本条指令需要从 HILO 寄存器中取出，此时会出现数据冒险，需要通过数据转发实现正常的调用。个人使用的方法是通过读取指令时选择控制信号 hlsel 来确定是读取 HI 还是 LO 寄存器，并根据后面阶段中 Lowin、Highin、multin 信号来判断转发到 Highout 与 Lowout 的数据。个人方法是，先通过 EX，MEM，WR 段得 HILO 寄存器写信号来转发到 id 段完成第一次有关于 mtlo 等指令的处理，其次再处理乘法结果的转发，如果乘法出现在 mtlo 等指令后端则选择已经选择的第一次处理得到的 Highout，Lowout 数据，如果乘法结果在 mtlo 等指令前面，则选择转发乘法结果，并根据 hlsel 来选择转发高位还是低位。

**Condition3.** 存在于 cp0 寄存器中的数据冒险，由于时间有限，只处理了 mtc0+mfc0 的数据冒险，由于 Syscall 和 Eret 的指令特性，转发考虑对于五级流水线过于麻烦，目前可以采用的方法是在 cp0 指令 id 阶段写入 cp0 寄存器然后下一指令直接读取，但这样就会存在两个数据同时存入寄存器的可能情况，故不推荐。其转发条件很简单，即当 mfc0 指令检测到前面存在 mtc0 正在执行，并比较两个指令目的的 cs 和 sel 地址段，就可以完成转发。

# 第二章　45 条流水线 CPU 的设计与调试

## 2.1　Verilog 代码

### 2.1.1 mips 模块代码

```
modu
le
mips
(clk
,rst
,PC)
;
            input clk;
            input rst;
```

```
    output wire[29:0] PC;
    wire[29:0] NPC,PC_plus_4,jalpc1,ex_jalpc;
    wire[31:0] if_ins;
    wire[29:0] branch_predict,predict_pc;

    wire zbgez,zbgtz,zbeq,zbne;
    wire branchbubble;
    wire flush,pc_sel;
    wire[1:0] branchforwardA,branchforwardB,jalforward;
    wire[31:0] id_ins;
    wire[29:0] id_PC_plus_4,id_cp0_pcout,id_cp0_pcoutmux;
    wire[31:0]
id_busA,id_busA_mux2,id_busB,id_busB_mux2,mem_forward,wr_forward,id_Highout,id_Lowout,id_
HL,id_Highoutmux4,id_Lowoutmux4,id_HL_mux4,id_cp0_dout,id_cp0_doutmux;
    wire[4:0] id_ra,id_rb,id_rw,id_cs;
    wire[31:0] id_imm32;
    wire[15:0] id_imm16;
    wire[4:0] id_shamt;
    wire[5:0] id_func;
    wire[25:0] id_target,ex_target;
    wire[2:0] id_cp0op,id_sel;

    wire
id_regWr,id_multWr,id_hlsel,id_Extop,id_alusrc,id_branch_beq,id_branch_bne,id_bltz,id_ble
z,id_bgez,id_bgtz,id_jalr,id_jal,id_jump,id_memWr,id_checkover,id_Highin,id_Lowin;
    wire[1:0] id_memtoreg,id_regDst;
    wire[4:0] id_aluop;
    wire[5:0] id_op;
    //ex
    wire ex_zero,ex_overflow;
    wire[31:0] ex_result,ex_busA,ex_busB,ex_HL,ex_busA_mux2,ex_busB_mux2,ex_cp0_dout;
    wire[4:0] ex_ra,ex_rb,ex_rw,ex_rw_mux2,ex_shamt,ex_cs;
    wire[63:0] ex_mult;

    wire
ex_regWr,ex_multWr,ex_alusrc,ex_memwr,ex_checkover,ex_Highin,ex_Lowin,ex_branch_beq,ex_br
anch_bne,ex_bltz,ex_blez,ex_bgez,ex_bgtz,ex_jalr,ex_jal,ex_jump;
    wire[1:0] ex_memtoreg,ex_regDst;
    wire[4:0] ex_aluop;
    wire[5:0] ex_op;
    wire[2:0] ex_cp0op,ex_sel;
```

```
    wire[31:0] ex_imm32,ex_alu_busB;
    //mem
    wire[31:0] mem_Dataout,mem_HL,mem_busA_mux2,mem_cp0_dout,mem_busB_mux2;
    wire[31:0] mem_result;
    wire[63:0] mem_mult;
    wire[4:0] mem_rw,mem_cs;
    wire[2:0] mem_sel,mem_cp0op;

    wire mem_regWr,mem_multWr,mem_memWr,mem_zero,mem_Highin,mem_Lowin;
    wire[1:0] mem_memtoreg;
    wire[5:0] mem_op;
    //wr
    wire[31:0] wr_Dataout;
    wire[31:0] wr_result,wr_HL,wr_busA_mux2,wr_busB_mux2,wr_cp0_dout;
    wire[63:0] wr_mult;
    wire[4:0] wr_rw,wr_cs;
    wire[5:0] wr_op;
    wire[2:0] wr_cp0op,wr_sel;

    wire wr_regWr,wr_multWr,wr_Highin,wr_Lowin;
    wire[1:0] wr_memtoreg;
    //foward
    wire[1:0] multforward,cp0forward;


    pc get_pc(clk,pc_sel,NPC,predict_pc,rst,PC,branchbubble);

    npc
get_npc(clk,rst,flush,branchbubble,PC,id_PC_plus_4,branch_predict,ex_target,id_imm16,ex_i
mm32[15:0],ex_jalpc,id_cp0op,id_cp0_pcoutmux,id_branch_beq,id_branch_bne,id_bgez,id_bgtz,
id_blez,id_bltz,ex_branch_beq,ex_branch_bne,ex_bgez,ex_bgtz,ex_blez,ex_bltz,zbgez,zbgtz,z
beq,zbne,ex_jalr,ex_jal,ex_jump,NPC,predict_pc,branch_predict,pc_sel,flush);
    assign PC_plus_4 = PC+1;
    im_4k get_im(PC[9:0],if_ins);

    IFIDReg ifidreg(clk,flush,PC_plus_4,if_ins,branchbubble,id_PC_plus_4,id_ins);

    decoder
decoder(id_ins,id_op,id_ra,id_rb,id_rw,id_shamt,id_func,id_cs,id_sel,id_imm16,id_target);

    Control
ctr(id_op,id_rb,id_ra,id_func,id_regWr,id_multWr,id_Lowin,id_Highin,id_hlsel,id_regDst,id
_Extop,id_alusrc,id_aluop,id_memWr,id_memtoreg,id_checkover,id_jump,id_branch_beq,id_bran
```

```
ch_bne,id_bgez,id_bgtz,id_blez,id_bltz,id_jalr,id_jal,id_cp0op);

    regfile
rf(clk,id_ra,id_rb,wr_rw,wr_op,wr_cp0op,wr_cp0_dout,wr_result[11:0],wr_regWr,wr_multWr,wr
_Lowin,wr_Highin,wr_busA_mux2,wr_forward,wr_mult,id_busA,id_busB,id_Highout,id_Lowout,jal
pc1);

    SignExt SignExt(id_imm16,id_Extop,id_imm32);

    mux3 mux_rw(ex_rb,ex_rw,32'd31,ex_regDst,ex_rw_mux2);

    branchforward
branchforward(id_ra,id_rb,ex_cp0op,ex_rw_mux2,ex_regWr,mem_cp0op,mem_rw,mem_regWr,wr_cp0o
p,wr_rw,wr_regWr,branchforwardA,branchforwardB);

    wire[31:0] ex_forward;
    mux4 mux_exforward(ex_result,32'd0,ex_HL,ex_cp0_dout,ex_memtoreg,ex_forward);

    mux4
mux_memforward(mem_result,mem_Dataout,mem_HL,mem_cp0_dout,mem_memtoreg,mem_forward);

    mux4 mux_wrforward(wr_result,wr_Dataout,wr_HL,wr_cp0_dout,wr_memtoreg,wr_forward);

    mux4
mux_judgeA(id_busA,ex_forward,mem_forward,wr_forward,branchforwardA,id_busA_mux2);

    mux4
mux_judgeB(id_busB,ex_forward,mem_forward,wr_forward,branchforwardB,id_busB_mux2);

    branchjudge
branchjudge(ex_busA_mux2,ex_busB_mux2,ex_bgez,ex_bgtz,ex_blez,ex_bltz,ex_branch_beq,ex_br
anch_bne,zbgez,zbgtz,zbeq,zbne);

    branchbubble branchbubble1(id_ra,id_rb,ex_regWr,ex_rw_mux2,ex_memtoreg,branchbubble);

    id_ex
idexreg(clk,branchbubble,id_PC_plus_4,id_busA,id_busA_mux2,id_busB,id_busB_mux2,id_HL_mux
4,id_ra,id_rb,id_rw,id_imm32,id_regWr,id_multWr,id_regDst,id_alusrc,id_memWr,id_memtoreg,
id_checkover,id_aluop,id_shamt,id_op,id_Lowin,id_Highin,id_cp0op,id_cs,id_sel,id_cp0_dout
mux,id_cp0_pcoutmux,id_branch_beq,id_branch_bne,id_bltz,id_blez,id_bgez,id_bgtz,id_jalr,i
d_jal,id_jump,id_target,

    ex_busA,ex_busA_mux2,ex_busB,ex_busB_mux2,ex_HL,ex_ra,ex_rb,ex_rw,ex_imm32,ex_regWr,e
```

```
x_multWr,ex_regDst,ex_alusrc,ex_memwr,ex_memtoreg,ex_checkover,ex_aluop,ex_shamt,ex_op,ex
_Lowin,ex_Highin,ex_cp0op,ex_cs,ex_sel,ex_cp0_dout,ex_cp0_pcout,ex_branch_beq,ex_branch_b
ne,ex_bltz,ex_blez,ex_bgez,ex_bgtz,ex_jalr,ex_jal,ex_jump,ex_jalpc,ex_target);

    mux_memtoreg mux_alusrc_to_busB(ex_busB_mux2,ex_imm32,ex_alusrc,ex_alu_busB);

    alu
alu(ex_checkover,id_PC_plus_4,ex_aluop,ex_shamt,ex_busA_mux2,ex_alu_busB,ex_zero,ex_overf
low,ex_result,ex_mult);

    ex_mem
exmemreg(clk,ex_zero,ex_HL,ex_result,ex_mult,ex_busA_mux2,ex_busB_mux2,ex_rw_mux2,ex_regW
r,ex_multWr,ex_memwr,ex_memtoreg,ex_op,ex_Lowin,ex_Highin,ex_cp0op,ex_cs,ex_sel,ex_cp0_do
ut,mem_zero,mem_HL,mem_result,mem_mult,mem_busA_mux2,mem_busB_mux2,mem_rw,mem_regWr,mem_m
ultWr,mem_memWr,mem_memtoreg,mem_op,mem_Lowin,mem_Highin,mem_cp0op,mem_cs,mem_sel,mem_cp0
_dout);

    dm_4k dm_4k(clk,mem_op,mem_memWr,mem_result[11:0],mem_busB_mux2,mem_Dataout);

    mem_wr
mem_wr(clk,mem_Dataout,mem_HL,mem_result,mem_mult,mem_busA_mux2,mem_busB_mux2,mem_rw,mem_
regWr,mem_multWr,mem_memtoreg,mem_op,mem_Lowin,mem_Highin,mem_cp0op,mem_cs,mem_sel,mem_cp
0_dout,wr_Dataout,wr_HL,wr_result,wr_mult,wr_busA_mux2,wr_busB_mux2,wr_rw,wr_regWr,wr_mul
tWr,wr_memtoreg,wr_op,wr_Lowin,wr_Highin,wr_cp0op,wr_cs,wr_sel,wr_cp0_dout);

    mux_memtoreg muxHL(id_Lowoutmux4,id_Highoutmux4,id_hlsel,id_HL);

    multforward
multforward1(id_multWr,ex_multWr,ex_Highin,ex_Lowin,mem_multWr,mem_Highin,mem_Lowin,wr_mu
ltWr,multforward);

    muxHL muxHL1(id_HL,ex_mult,mem_mult,wr_mult,multforward,id_hlsel,id_HL_mux4);

    cp0forwardUnit
cp0forwardd(id_cp0op,id_cs,id_sel,ex_cs,ex_sel,ex_cp0op,mem_cs,mem_sel,mem_cp0op,wr_cs,wr
_sel,wr_cp0op,cp0forward);

    CP0
cp0(clk,id_cs,id_sel,wr_cs,wr_sel,wr_busB_mux2,id_PC_plus_4-1,id_cp0_dout,id_cp0_pcout,wr
_cp0op,id_cp0op);

    mux4
mux_cp0dout(id_cp0_dout,ex_busB_mux2,mem_busB_mux2,wr_busB_mux2,cp0forward,id_cp0_doutmux
```

```
);

    mux4pc
mux_cp0(id_cp0_pcout,ex_busB_mux2[31:2],mem_busB_mux2[31:2],wr_busB_mux2[31:2],cp0forward
,id_cp0_pcoutmux);

    wire[1:0]forwardHigh,forwardLow;
    HLforward
HLforward1(ex_Highin,ex_Lowin,mem_Highin,mem_Lowin,wr_Highin,wr_Lowin,forwardHigh,forward
Low);

    mux4
muxHighout(id_Highout,ex_busA_mux2,mem_busA_mux2,wr_busA_mux2,forwardHigh,id_Highoutmux4)
;

    mux4
muxLowout(id_Lowout,ex_busA_mux2,mem_busA_mux2,wr_busA_mux2,forwardLow,id_Lowoutmux4);
endmodule
```

### 2.1.2 PC 模块代码

```
module pc(
                clk,
                pc_sel,
                adin,
                predict_pc,
                rst,
                adout,
                BranchBubble
            );
            input wire clk,rst,BranchBubble,pc_sel;
            input wire[29:0] adin,predict_pc;
            output reg[29:0] adout;
            wire[31:0] temp = 32'h00003034;
            initial begin
                adout=temp[31:2];
            end

            always@(posedge clk)
            begin
                if(!BranchBubble)
                begin
                    if(rst == 1)
```

```
                    adout = temp[31:2];
                else
                    adout = (pc_sel==1)?adin:predict_pc;
            end
        end
        endmodule
```

### 2.1.3 NPC 模块代码

```
modu
le
npc(
        clk,
        rst,
        flushbefore,
        Brunchbubble,
        cur_pc,
        pre_pc,
        branch_predict2,
        target,
        id_immediate,
        immediate,
        jal_pc,
        cp0op,
        cp0_pcout,
        id_branch_beq,id_branch_bne,id_bgez,id_bgtz,id_blez,id_bltz,
        branch_beq,
        branch_bne,
        bgez,
        bgtz,
        blez,
        bltz,
        zbgez,
        zbgtz,
        zbeq,
        zbne,
        jalr,
        jal,
        jump,
        next_pc,
        predict_pc,
        branch_predict,
        pc_sel,
```

```
    flush
);
input wire clk,rst,Brunchbubble,flushbefore;
input wire[15:0] immediate,id_immediate;
input wire[2:0] cp0op;
input wire jump;  //zero-alu??????
input wire
branch_beq,branch_bne,bgez,bgtz,blez,bltz,jalr,jal,zbgez,zbgtz,zbeq,zbne,id_branch_beq,id_
branch_bne,id_bgez,id_bgtz,id_blez,id_bltz;
input wire[25:0] target;
input wire[29:0] cur_pc,pre_pc;
input wire[29:0] jal_pc,cp0_pcout;
output reg[29:0] next_pc,predict_pc;
output reg flush,pc_sel;
wire[31:0] temp = 32'h00003034;
input wire[29:0] branch_predict2;
output reg [29:0] branch_predict;
wire [29:0] new_immediate;
reg [29:0] jump_next;
reg [61:0] BHT[16:0];
reg flag;

integer i,t;
initial begin
    next_pc <= temp[31:2];
    flush <= 0;
    for(i=0;i<=15;i=i+1)
        BHT[i] = 0;
end

always@(id_branch_bne or id_branch_beq or id_bgez or id_bgtz or id_blez or id_bltz) begin
    for(i=0;i<=15;i=i+1) begin
        if(BHT[i][61]==1&&BHT[i][60:31]==cur_pc-1&&BHT[i][30]==1) begin
            flag = 1;
            predict_pc = BHT[i][29:0];
            branch_predict = predict_pc;
            pc_sel = 0;
        end
        else if(BHT[i][61]==1&&BHT[i][60:31]==cur_pc-1&&BHT[i][30]==0) begin
            flag = 1;
            predict_pc = BHT[i][29:0];
            branch_predict = predict_pc;
            pc_sel = 0;
```

```
            end
            else if(BHT[i][61]==0) begin
                flag = 0;
                t = i;
                pc_sel = 0;
            end
        end
        if(flag == 0) begin
            if(id_branch_bne||id_branch_beq||id_bgez||id_bgtz||id_blez||id_bltz) begin
                predict_pc = cur_pc+1;
                branch_predict = predict_pc;
                BHT[t][61] = 1;
                BHT[t][60:31] = cur_pc-1;
                pc_sel = 0;
            end
            else begin
                pc_sel = 1;
            end
        end
    end

    always@(posedge clk or negedge clk) begin
        if(clk) begin
            if(Brunchbubble) begin
            if(flushbefore==1)
                flush = 1;
            end
            flush = 0;
        end
        if(!clk) begin
        if(!Brunchbubble) begin
        end
        if(rst == 1 && !clk)
            next_pc = temp[31:2];
        else if(jalr == 1 && !clk) begin//jalr
            next_pc = jal_pc;
            flush = 1;
        end
        else if(jal == 1 && !clk)  begin//jal
            next_pc = {pre_pc[29:26],target[25:0]};
            flush = 1;
        end
        else if(branch_bne == 1 && !clk)  begin//bne
```

```
            BHT[t][29:0] = predict_pc;
            next_pc = (zbne)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
            if(next_pc != branch_predict2) begin
                for(i=0;i<=15;i=i+1) begin
                    if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                        BHT[i][30] = !BHT[i][30];
                        BHT[i][29:0] = next_pc;
                        flush = 1;
                    end
                end
            end
            else begin
                next_pc = next_pc+1;
                flush = 0;
            end
        end
        else if(branch_beq == 1 && !clk)  begin//beq
            BHT[t][29:0] = predict_pc;
            next_pc = (zbeq)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
            if(next_pc != branch_predict2) begin
                for(i=0;i<=15;i=i+1) begin
                    if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                        BHT[i][30] = !BHT[i][30];
                        BHT[i][29:0] = next_pc;
                        flush = 1;
                    end
                end
            end
            else begin
                next_pc = next_pc+1;
                flush = 0;
            end
        end
        else if(bgez == 1 && !clk)  begin//bgez
            BHT[t][29:0] = predict_pc;
            next_pc = (zbgez==1)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
            if(next_pc != branch_predict2) begin
                for(i=0;i<=15;i=i+1) begin
                    if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                        BHT[i][30] = !BHT[i][30];
                        BHT[i][29:0] = next_pc;
                        flush = 1;
                    end
```

```
            end
        end
        else begin
            next_pc = next_pc+1;
            flush = 0;
        end
    end
    else if(bltz == 1 && !clk)  begin//bltz
        BHT[t][29:0] = predict_pc;
        next_pc = (zbgez==0)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
        if(next_pc != branch_predict2) begin
            for(i=0;i<=15;i=i+1) begin
                if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                    BHT[i][30] = !BHT[i][30];
                    BHT[i][29:0] = next_pc;
                    flush = 1;
                end
            end
        end
        else begin
            next_pc = next_pc+1;
            flush = 0;
        end
    end
    else if(bgtz == 1 && !clk)  begin//bgtz
        BHT[t][29:0] = predict_pc;
        next_pc = (zbgtz==1)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
        if(next_pc != branch_predict2) begin
            for(i=0;i<=15;i=i+1) begin
                if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                    BHT[i][30] = !BHT[i][30];
                    BHT[i][29:0] = next_pc;
                    flush = 1;
                end
            end
        end
        else begin
            next_pc = next_pc+1;
            flush = 0;
        end
    end
    else if(blez == 1 && !clk)  begin//blez
        BHT[t][29:0] = predict_pc;
```

```
                next_pc = (zbgtz==0)?pre_pc-1+{{14{immediate[15]}},immediate}:pre_pc;
                if(next_pc != branch_predict2) begin
                    for(i=0;i<=15;i=i+1) begin
                        if(BHT[i][61]==1&&BHT[i][60:31]==pre_pc-2) begin
                            BHT[i][30] = !BHT[i][30];
                            BHT[i][29:0] = next_pc;
                            flush = 1;
                        end
                    end
                end
                else begin
                    next_pc = next_pc+1;
                    flush = 0;
                end
            end
            else if(jump == 1 && !clk)  begin//jump
                next_pc = {pre_pc[29:26],target[25:0]};
                flush = 1;
            end
            else if(cp0op == 3'b011 && !clk)  begin//syscall
                next_pc = 30'd0;
                flush = 1;
            end
            else if(cp0op == 3'b100 && !clk)  begin//eret
                next_pc = cp0_pcout;
                flush = 1;
            end
            else if(!id_bgtz&&!id_bgez&&!id_blez&&!id_bltz&&!id_branch_beq&&!id_branch_bne) begin
                next_pc = cur_pc+1;
            end
            end
        end
endmodule
```

### 2.1.4 im_4k 模块代码

```
module im_4k(
                    addr,
                    dout
                );

                input wire[11:2] addr;

                output wire[31:0] dout;
```

```
reg [31:0] im [1023:0];

initial begin
    $readmemh("code.txt",im);
end

assign dout = im[addr];

endmodule
```

### 2.1.5 decoder 模块代码

```
module decoder(
                ir,
                op,
                rs,
                rt,
                rd,
                shamt,
                func,
                cs,
                sel,
                immediate,
                target
              );

              input wire[31:0] ir;
              output reg[5:0] op;
              output reg[4:0] rs;
              output reg[4:0] rt;
              output reg[4:0] rd;
              output reg[4:0] cs;
              output reg[2:0] sel;
              output reg[4:0] shamt;
              output reg[5:0] func;
              output reg[15:0] immediate;
              output reg[25:0] target;

              initial begin
                  op = 6'b000000;
                  rs = 5'b00000;
                  rt = 5'b00000;
```

```
            rd = 5'b00000;
            cs = 5'b00000;
            sel = 3'b000;
        end

        always@(*)
        begin
            op = ir[31:26];
            rs = ir[25:21];
            rt = ir[20:16];
            rd = ir[15:11];
            shamt = ir[10:6];
            func = ir[5:0];
            immediate = ir[15:0];
            target = ir[25:0];
            cs = ir[15:11];
            sel = ir[2:0];
        end

        endmodule
```

### 2.1.6 alu 模块代码

```
module alu(
            checkover,
            pc,
            aluop,
            shamt,
            DataA,
            DataB,
            zero,
            overflow,
            result,
            mult,
        );

        input wire[4:0] aluop;
        input wire[4:0] shamt;
        input wire[31:0] DataA;
        input wire[31:0] DataB;
        input wire[29:0] pc;
        input wire checkover;
        output reg zero,overflow;
```

```
output reg[31:0] result;
output reg[63:0] mult;
reg Cout;

always@(aluop or DataA or DataB) begin
    case(aluop)
        5'b00000: begin  //add
        result = DataA + DataB;
        Cout = DataA[31]&DataB[31];
        zero = (result == 0)?1:0;
        overflow = checkover&(Cout^result[31]);
        end
        5'b00001: begin  //sub
        result = DataA - DataB;
        Cout = DataA[31]&DataB[31];
        zero = (result == 0)?1:0;
        overflow = checkover&(Cout^result[31]);
        end
        5'b00010: begin  //slt
        result = (DataA < DataB)?1:0;
        zero = (result == 0)?1:0;
        end
        5'b00011: begin  //and
        result = DataA & DataB;
        zero = (result == 0)?1:0;
        end
        5'b00100: begin  //nor
        result = ~(DataA | DataB);
        zero = (result == 0)?1:0;
        end
        5'b00101: begin  //or
        result = DataA | DataB;
        zero = (result == 0)?1:0;
        end
        5'b00110: begin  //xor
        result = DataA ^ DataB;
        zero = (result == 0)?1:0;
        end
        5'b00111: begin  //sll
        result = DataB << shamt;
        zero = (result == 0)?1:0;
        end
        5'b01000: begin  //srl
```

```
                result = DataB >> shamt;
                zero = (result == 0)?1:0;
                end
                5'b01001: begin  //sltu
                result = (DataA < DataB)?1:0;
                zero = (result == 0)?1:0;
                end
                5'b01010: begin  //jalr
                result = {pc,{2'b00}};
                zero = (result == 0)?1:0;
                end
                5'b01011: begin  //jr
                end
                5'b01100: begin  //sllv
                result = DataB << DataA;
                zero = (result == 0)?1:0;
                end
                5'b01101: begin  //sra
                result = ($signed(DataB)) >>> shamt;
                zero = (result == 0)?1:0;
                end
                5'b01110: begin  //srav
                result = ($signed(DataB)) >>> DataA;
                zero = (result == 0)?1:0;
                end
                5'b01111: begin  //srlv
                result = DataB >> DataA;
                zero = (result == 0)?1:0;
                end
                5'b10000: begin  //lui
                result = DataB * 65536;
                zero = (result == 0)?1:0;
                end
                5'b10001: begin  //mult
                mult = ($signed(DataA)) * ($signed(DataB));
                zero = (mult == 0)?1:0;
                end
            endcase
        end
        endmodule
```

### 2.1.7 dm_4k 模块代码

```
module dm_4k(
                    clk,
                    op,
                    WrEn,
                    Addr,
                    DataIn,
                    DataOut
                );

                input wire clk,WrEn;
                input wire[11:0] Addr;
                input wire[5:0] op;
                input wire[31:0] DataIn;
                output reg[31:0] DataOut;

                reg[31:0] dm[1023:0];
                integer i;
                initial begin
                    for(i=0;i<1024;i=i+1)
                        dm[i]=0;
                end
                always@(*) begin
                    if(op == 6'b100000) begin
                        if(Addr[1:0] == 2'b00) begin
                            DataOut = {{24{dm[Addr[11:2]][7]}},dm[Addr[11:2]][7:0]};
                        end
                        else if(Addr[1:0] == 2'b01) begin
                            DataOut = {{24{dm[Addr[11:2]][15]}},dm[Addr[11:2]][15:8]};
                        end
                        else if(Addr[1:0] == 2'b10) begin
                            DataOut = {{24{dm[Addr[11:2]][23]}},dm[Addr[11:2]][23:16]};
                        end
                        else if(Addr[1:0] == 2'b11) begin
                            DataOut = {{24{dm[Addr[11:2]][31]}},dm[Addr[11:2]][31:24]};
                        end
                    end
                    else if(op == 6'b100100) begin
                        if(Addr[1:0] == 2'b00) begin
                            DataOut = {{24'b0},dm[Addr[11:2]][7:0]};
                        end
                        else if(Addr[1:0] == 2'b01) begin
                            DataOut = {{24'b0},dm[Addr[11:2]][15:8]};
                        end
```

```
            else if(Addr[1:0] == 2'b10) begin
                DataOut = {{24'b0},dm[Addr[11:2]][23:16]};
            end
            else if(Addr[1:0] == 2'b11) begin
                DataOut = {{24'b0},dm[Addr[11:2]][31:24]};
            end
        end
        else begin
            DataOut = dm[Addr[11:2]];
        end
    end

    always@(negedge clk) begin
        if(WrEn)
            begin
            if(op == 6'b101000)
            begin
            if(Addr[1:0] == 2'b00) begin
                dm[Addr[11:2]][7:0] = DataIn[7:0];
                $display("%h->dm[%h][7:0]",DataIn[7:0],Addr[11:0]);
            end
            else if(Addr[1:0] == 2'b01) begin
                dm[Addr[11:2]][15:8] = DataIn[7:0];
                $display("%h->dm[%h][15:8]",DataIn[7:0],Addr[11:0]);
            end
            else if(Addr[1:0] == 2'b10) begin
                dm[Addr[11:2]][23:16] = DataIn[7:0];
                $display("%h->dm[%h][23:16]",DataIn[7:0],Addr[11:0]);
            end
            else if(Addr[1:0] == 2'b11)
                dm[Addr[11:2]][31:24] = DataIn[7:0];
                $display("%h->dm[%h][31:24]",DataIn[7:0],Addr[11:0]);
            end
            else
            begin
                dm[Addr[11:2]] = DataIn;
                $display("%h->dm[%h]",DataIn,Addr[11:0]);
            end
            end
    end
    endmodule
```

### 2.1.8 SignExt 模块代码

```
`timescale 1ns / 1ps
                    module SignExt(
                        immediate,
                        ExtSel,
                        Extimmediate
                    );

                    input wire[15:0] immediate;
                    input ExtSel;

                    output [31:0] Extimmediate;

                    always@(Extimmediate)
                    begin

                    end

                    assign Extimmediate[15:0] = immediate;
                    assign Extimmediate[31:16] = ExtSel ? (immediate[15] ? 16'hffff : 16'h0000) :
                    16'h0000;

                    endmodule
```

### 2.1.9 mux 模块代码

```
module
mux_memtoreg(result,Dataout,memtoreg,busW);
                                                    input wire memtoreg;
                                                    input wire[31:0] result,Dataout;
                                                    output wire[31:0] busW;

                                                    assign busW=(memtoreg==0)?result:Dataout;
                                                endmodule

                                                module mux_rw(rt,rd,regDst,rw);
                                                    input wire[4:0] rt,rd;
                                                    input wire regDst;
                                                    output wire[4:0] rw;

                                                    assign rw=(regDst==0)?rt:rd;
                                                endmodule
```

```
module mux3(result,C1,C2,s,y);
    input wire[31:0] result,C1,C2;
    input wire[1:0] s;
    output wire[31:0] y;

    assign
y=(s==2'b00)?result:(s==2'b01)?C1:C2;
endmodule

module mux4(C1,C2,C3,C4,s,y);
    input wire[31:0] C1,C2,C3,C4;
    input wire[1:0] s;
    output wire[31:0] y;

    assign
y=(s==2'b00)?C1:(s==2'b01)?C2:(s==2'b10)?C3:C4;
endmodule

module mux4pc(C1,C2,C3,C4,s,y);
    input wire[29:0] C1,C2,C3,C4;
    input wire[1:0] s;
    output wire[29:0] y;

    assign
y=(s==2'b00)?C1:(s==2'b01)?C2:(s==2'b10)?C3:C4;
endmodule

module muxHL(C1,C2,C3,C4,s,hlsel,y);
    input wire[31:0] C1;
    input wire[63:0] C2,C3,C4;
    input wire[1:0] s;
    input wire hlsel;
    output reg[31:0] y;

    always@(*) begin
    if(s==0)
        y=C1;
    else if(s==2'b01&&hlsel==0)
        y=C2[31:0];
    else if(s==2'b01&&hlsel==1)
        y=C2[63:32];
    else if(s==2'b10&&hlsel==0)
```

```
            y=C3[31:0];
        else if(s==2'b10&&hlsel==1)
            y=C3[63:32];
        else if(s==2'b11&&hlsel==0)
            y=C4[31:0];
        else if(s==2'b11&&hlsel==1)
            y=C4[63:32];
        end
    endmodule
```

### 2.1.10 regfile 模块代码

```
module regfile(
                    clk,
                    rs,
                    rt,
                    rw,
                    op,
                    cp0op,
                    cp0_dout,
                    addr,
                    regWe,
                    multWe,
                    Lowin,
                    Highin,
                    Wr_busA_mux2
                    busW,
                    busmult,
                    busA,
                    busB,
                    Highout,
                    Lowout,
                    jalpc
                );

                input wire clk,regWe,multWe,Highin,Lowin;
                input wire[2:0] cp0op;
                input wire[4:0] rs,rt,rw;
                input wire[5:0] op;
                input wire[11:0] addr;
                input wire[31:0] busW,cp0_dout,wr_busA_mux2;
                input wire[63:0] busmult;
                output reg[31:0] busA,busB,Highout,Lowout;
```

```
output reg[29:0] jalpc;

reg [31:0]registers[31:0];
reg [31:0]High;
reg [31:0]Low;
integer i;
initial begin
    for(i=0;i<32;i=i+1)
        registers[i]=0;
    High=0;
    Low=0;
end

always@(*) begin
    busA = registers[rs];
    busB = registers[rt];
    jalpc = registers[rs][31:2];
    Highout = High[31:0];
    Lowout = Low[31:0];
end

always@(negedge clk) begin
    if(regWe)
    begin
        registers[rw] = busW;
        $display("%h->reg[%d]",busW,rw);
    end
    if(multWe)
    begin
        High = busmult[63:32];
        Low = busmult[31:0];
        $display("%h->Hign",busmult[63:32]);
        $display("%h->Low",{{24'b0},busmult[31:0]});
    end
    if(Highin)
    begin
        High = wr_busA_mux2;
        $display("%h->Hign",wr_busA_mux2);
    end
    if(Lowin)
    begin
        Low = wr_busA_mux2;
        $display("%h->Low",wr_busA_mux2);
```

```
                    end
                    if(cp0op == 3'b001)
                    begin
                        registers[rw] = cp0_dout;
                        $display("%h->reg[%d]",cp0_dout,rw);
                    end
                end

                endmodule
```

### 2.1.11 control 模块代码

```
`timesc
ale 1ns
/ 1ps
          module Control(
              op,
              rt,
              rs,
              func,
              regWr,
              multWr,
              Lowin,
              Highin,
              hlsel,
              regDst,
              Extop,
              alusrc,
              aluop,
              memWr,
              memtoreg,
              checkover,
              jump,
              branch_beq,
              branch_bne,
              bgez,
              bgtz,
              blez,
              bltz,
              jalr,
              jal,
              cp0op,
          );
```

```
input wire[5:0] op;
input wire[4:0] rs,rt;
input wire[5:0] func;
output reg
regWr,Extop,alusrc,memWr,checkover,jump,branch_beq,branch_bne,bgez,bgtz,blez,bltz,jalr
,jal,multWr,hlsel,Lowin,Highin;
output reg[1:0] memtoreg,regDst;
output reg[2:0] cp0op;
output reg[4:0] aluop;

initial begin
    regWr = 0;
    regDst = 0;
    Extop = 0;
    alusrc = 0;
    memWr = 0;
    memtoreg = 0;
    jump = 0;
    branch_beq = 0;
    branch_bne = 0;
    bgez = 0;
    bgtz = 0;
    blez = 0;
    bltz = 0;
    jalr = 0;
    jal = 0;
    hlsel = 0;
    multWr = 0;
    Lowin = 0;
    Highin = 0;
    cp0op = 3'd0;
end

always@(*)
begin
    case(op)
        6'b001001: begin  //addi
            Extop = 1;
            regDst = 0;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
```

```
            alusrc = 1;
            memtoreg = 0;
            checkover = 1;
            memWr = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;// addu
            end
        6'b100011: begin  //load word
            Extop = 1;
            regDst = 0;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
            alusrc = 1;
            memtoreg = 1;
            checkover = 0;
            memWr = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //addu
            end
        6'b101011: begin  //store word
```

```
            Extop = 1;
            regDst = 0;  //have nothing to do with
            regWr = 0;
            multWr = 0;
            hlsel = 0;
            alusrc = 1;
            memtoreg = 1;  //have nothing to do with
            memWr = 1;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //addu
            end
        6'b000100: begin  //beq
            Extop = 1;  //have nothing to do with
            regDst = 0;  //have nothing to do with
            regWr = 0;
            multWr = 0;
            hlsel = 0;
            alusrc = 0;
            memtoreg = 0;  //have nothing to do with
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 1;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
```

```
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b00001;  //subu
        end
    6'b000101: begin  //bne
        Extop = 1;  //have nothing to do with
        regDst = 0;  //have nothing to do with
        regWr = 0;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;
        memtoreg = 0;  //have nothing to do with
        memWr = 0;
        checkover = 0;
        jump = 0;
        branch_beq = 0;
        branch_bne = 1;
        bgez = 0;
        bgtz = 0;
        blez = 0;
        bltz = 0;
        jalr = 0;
        jal = 0;
        Lowin = 0;
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b00001;  //subu
        end
    6'b000010: begin  //jump
        Extop = 1;  //have nothing to do with
        regDst = 0;  //have nothing to do with
        regWr = 0;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;  //have nothing to do with
        memtoreg = 0;  //have nothing to do with
        memWr = 0;
        checkover = 0;
        jump = 1;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
        bgtz = 0;
```

```
        blez = 0;
        bltz = 0;
        jalr = 0;
        jal = 0;
        Lowin = 0;
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b00001;  //have nothing to do with
        end
    6'b000011: begin  //jal
        Extop = 1;  //have nothing to do with
        regDst = 3;  //have nothing to do with
        regWr = 1;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;  //have nothing to do with
        memtoreg = 0;
        memWr = 0;
        checkover = 0;
        jump = 1;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
        bgtz = 0;
        blez = 0;
        bltz = 0;
        jalr = 0;
        jal = 1;
        Lowin = 0;
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b01010;  //jalr
        end
    6'b001111: begin  //lui
        Extop = 0;
        regDst = 0;
        regWr = 1;
        multWr = 0;
        hlsel = 0;
        alusrc = 1;
        memtoreg = 0;
        memWr = 0;
        checkover = 0;
```

```
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b10000;  //lui
            end
        6'b001010: begin  //slti
            Extop = 1;
            regDst = 0;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
            alusrc = 1;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00010;  //slti
            end
        6'b001011: begin  //sltiu
            Extop = 1;
            regDst = 0;
            regWr = 1;
            multWr = 0;
```

```
            hlsel = 0;
            alusrc = 1;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b01001;  //sltiu
            end
        6'b000001: begin  //bgez bltz
            if(rt == 1) begin
            Extop = 0;  //have nothing to do with
            regDst = 0;  //have nothing to do with
            regWr = 0;
            multWr = 0;
            hlsel = 0;
            alusrc = 0;
            memtoreg = 0;  //have nothing to do with
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 1;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //have nothing to do with
```

```
        end
        else begin
        Extop = 0;  //have nothing to do with
        regDst = 0;  //have nothing to do with
        regWr = 0;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;
        memtoreg = 0;  //have nothing to do with
        memWr = 0;
        checkover = 0;
        jump = 0;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
        bgtz = 0;
        blez = 0;
        bltz = 1;
        jalr = 0;
        jal = 0;
        Highin = 0;
        cp0op = 3'b000;
        end
        end
    6'b000111: begin  //bgtz
        Extop = 0;  //have nothing to do with
        regDst = 0;  //have nothing to do with
        regWr = 0;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;
        memtoreg = 0;  //have nothing to do with
        memWr = 0;
        checkover = 0;
        jump = 0;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
        bgtz = 1;
        blez = 0;
        bltz = 0;
        jalr = 0;
        jal = 0;
```

```
        Lowin = 0;
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b00000;  //have nothing to do with
        end
    6'b000110: begin  //blez
        Extop = 0;  //have nothing to do with
        regDst = 0;  //have nothing to do with
        regWr = 0;
        multWr = 0;
        hlsel = 0;
        alusrc = 0;
        memtoreg = 0;  //have nothing to do with
        memWr = 0;
        checkover = 0;
        jump = 0;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
        bgtz = 0;
        blez = 1;
        bltz = 0;
        jalr = 0;
        jal = 0;
        Lowin = 0;
        Highin = 0;
        cp0op = 3'b000;
        aluop = 5'b00000;  //have nothing to do with
        end
    6'b100000: begin  //lb
        Extop = 1;
        regDst = 0;
        alusrc = 1;
        memtoreg = 1;
        regWr = 1;
        multWr = 0;
        hlsel = 0;
        memWr = 0;
        checkover = 0;
        jump = 0;
        branch_beq = 0;
        branch_bne = 0;
        bgez = 0;
```

```
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //have nothing to do with
            end
        6'b100100: begin  //lbu
            Extop = 1;
            regDst = 0;
            alusrc = 1;
            memtoreg = 1;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //have nothing to do with
            end
        6'b101000: begin  //sb
            Extop = 1;
            regWr = 0;
            multWr = 0;
            hlsel = 0;
            memWr = 1;
            alusrc = 1;
            memtoreg = 0;
            checkover = 0;
```

```
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00000;  //have nothing to do with
            end
        6'b001100: begin  //andi
            Extop = 0;
            regDst = 0;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
            alusrc = 1;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00011;  //and
            end
        6'b001101: begin  //ori
            Extop = 0;
            regDst = 0;
            regWr = 1;
            multWr = 0;
```

```
            hlsel = 0;
            alusrc = 1;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00101;  //or
            end
        6'b001110: begin  //xori
            Extop = 0;
            regDst = 0;
            regWr = 1;
            multWr = 0;
            hlsel = 0;
            alusrc = 1;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b000;
            aluop = 5'b00110;  //xor
            end
```

```
6'b010000: begin
    if(rs==5'b00000) begin  //MFC0
    Extop = 0;
    regDst = 0;
    regWr = 0;
    multWr = 0;
    hlsel = 0;
    alusrc = 0;
    memtoreg = 3;
    memWr = 0;
    checkover = 0;
    jump = 0;
    branch_beq = 0;
    branch_bne = 0;
    bgez = 0;
    bgtz = 0;
    blez = 0;
    bltz = 0;
    jalr = 0;
    jal = 0;
    Lowin = 0;
    Highin = 0;
    cp0op = 3'b001;  //MFC0
    aluop = 5'b00000;  //
    end
    else if(rs == 5'b00100) begin  //MTC0
    Extop = 0;
    regDst = 0;
    regWr = 0;
    multWr = 0;
    hlsel = 0;
    alusrc = 0;
    memtoreg = 0;
    memWr = 0;
    checkover = 0;
    jump = 0;
    branch_beq = 0;
    branch_bne = 0;
    bgez = 0;
    bgtz = 0;
    blez = 0;
    bltz = 0;
    jalr = 0;
```

```
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b010;  //MTC0
            aluop = 5'b00000;  //
            end
            else begin  //ERET
            Extop = 0;
            regDst = 0;
            regWr = 0;
            multWr = 0;
            hlsel = 0;
            alusrc = 0;
            memtoreg = 0;
            memWr = 0;
            checkover = 0;
            jump = 0;
            branch_beq = 0;
            branch_bne = 0;
            bgez = 0;
            bgtz = 0;
            blez = 0;
            bltz = 0;
            jalr = 0;
            jal = 0;
            Lowin = 0;
            Highin = 0;
            cp0op = 3'b100;  //ERET
            aluop = 5'b00000;  //
            end
            end
        6'b000000: begin  //R-type
            case(func)
                6'b100000: begin  //add
                Extop = 1;  //have nothing to do with
                regDst = 1;
                regWr = 1;
                multWr = 0;
                hlsel = 0;
                alusrc = 0;
                memtoreg = 0;
                memWr = 0;
                checkover = 1;
```

```
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00000;  // add
end
6'b100001: begin  //addu
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00000;  // add
end
6'b100010: begin  //sub
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
```

```
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 1;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00001;  // sub
end
6'b100011: begin  //subu
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00001;  // sub
end
```

```
6'b101010: begin  //slt
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00010;  // slt
end
6'b100100: begin  //and
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
```

```
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00011;  // and
end
6'b100111: begin  //nor
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00100;  // nor
end
6'b100101: begin  //or
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
```

```
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00101;  //  or
end
6'b100110: begin  //xor
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00110;  //  xor
end
6'b000000: begin  //sll
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
```

```
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00111;  //sll
end
6'b000010: begin  //srl
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01000;  //srl
end
6'b101011: begin  //sltu
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
```

```
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01001;  //sltu
end
6'b001001: begin  //jalr
Extop = 1;  //have nothing to do with
regDst = 3;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 1;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01010;  //jalr
```

```
end
6'b001000: begin  //jr
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 0;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 1;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01011;  //jr
end
6'b000100: begin  //sllv
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
```

```
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01100;  //sllv
end
6'b000011: begin  //sra
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01101;  //sra
end
6'b000111: begin  //srav
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
```

```
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01110;  //srav
end
6'b000110: begin  //srlv
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b01111;  //srlv
end
6'b011000: begin  //mult
Extop = 1;  //have nothing to do with
regDst = 0;
regWr = 0;
multWr = 1;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
```

```
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b10001;  //mult
end
6'b010010: begin  //MFLO
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 1;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 2;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00000;
end
6'b010000: begin  //MFHI
Extop = 1;  //have nothing to do with
regDst = 1;
```

```
regWr = 1;
multWr = 0;
hlsel = 1;
alusrc = 0;
memtoreg = 2;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 0;
cp0op = 3'b000;
aluop = 5'b00000;
end
6'b010011: begin  //MTLO
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 0;
multWr = 0;
hlsel = 1;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 1;
Highin = 0;
cp0op = 3'b000;
```

```
aluop = 5'b00000;
end
6'b010001: begin  //MTHI
Extop = 1;  //have nothing to do with
regDst = 1;
regWr = 0;
multWr = 0;
hlsel = 1;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
jalr = 0;
jal = 0;
Lowin = 0;
Highin = 1;
cp0op = 3'b000;
aluop = 5'b00000;
end
6'b001100: begin  //SYSCALL
Extop = 1;  //have nothing to do with
regDst = 0;
regWr = 0;
multWr = 0;
hlsel = 0;
alusrc = 0;
memtoreg = 0;
memWr = 0;
checkover = 0;
jump = 0;
branch_beq = 0;
branch_bne = 0;
bgez = 0;
bgtz = 0;
blez = 0;
bltz = 0;
```

```
                    jalr = 0;
                    jal = 0;
                    Lowin = 0;
                    Highin = 0;
                    cp0op = 3'b011;
                    aluop = 5'b00000;
                    end
                endcase
            end
            default: begin
                Extop = 0;  //have nothing to do with
                regDst = 0;
                regWr = 0;
                multWr = 0;
                hlsel = 0;
                alusrc = 0;
                memtoreg = 0;
                memWr = 0;
                checkover = 0;
                jump = 0;
                branch_beq = 0;
                branch_bne = 0;
                bgez = 0;
                bgtz = 0;
                blez = 0;
                bltz = 0;
                jalr = 0;
                jal = 0;
                Lowin = 0;
                Highin = 0;
                cp0op = 3'b000;
                aluop = 5'b00000;
            end
        endcase
    end
    endmodule
```

### 2.1.12 IFIDREG 模块代码

```
module IFIDReg(
                clk,
                flush,
                pc_plus_4,
```

```
    if_ins,
    BranchBubble,
    id_pc_plus_4,
    id_ins
);

input clk,BranchBubble,flush;
input wire[29:0] pc_plus_4;
input wire[31:0] if_ins;
output reg[29:0] id_pc_plus_4;
output reg[31:0] id_ins;

initial begin

end

always@(posedge clk)
begin
    if(BranchBubble) begin

    end
    else if(flush) begin
        id_ins = 0;
    end
    else begin
        id_ins = if_ins;
        id_pc_plus_4 = pc_plus_4;
    end
end

endmodule
```

### 2.1.13 IDEXREG 模块代码

```
module
id_ex(clk,BranchBubble,id_PC_plus_4,id_busA,id_
busA_mux2,id_busB,id_busB_mux2,id_HL,id_ra,id_r
b,id_rw,id_imm32,id_regWr,id_multWr,id_regDst,i
d_alusrc,id_memwr,id_memtoreg,id_checkover,id_a
luop,id_shamt,id_op,id_Lowin,id_Highin,id_cp0op
,id_cs,id_sel,id_cp0_dout,id_cp0_pcout,id_branc
h_beq,id_branch_bne,id_bltz,id_blez,id_bgez,id_
bgtz,id_jalr,id_jal,id_jump,id_target,
```

```
    ex_busA,ex_busA_mux2,ex_busB,ex_busB_mux2,
ex_HL,ex_ra,ex_rb,ex_rw,ex_imm32,ex_regWr,ex_mu
ltWr,ex_regDst,ex_alusrc,ex_memwr,ex_memtoreg,e
x_checkover,ex_aluop,ex_shamt,ex_op,ex_Lowin,ex
_Highin,ex_cp0op,ex_cs,ex_sel,ex_cp0_dout,ex_cp
0_pcout,ex_branch_beq,ex_branch_bne,ex_bltz,ex_
blez,ex_bgez,ex_bgtz,ex_jalr,ex_jal,ex_jump,ex_
jalpc,ex_target);

input wire
clk,BranchBubble,id_regWr,id_alusrc,id_memwr,id
_checkover,id_multWr,id_Lowin,id_Highin,id_bran
ch_beq,id_branch_bne,id_bltz,id_blez,id_bgez,id
_bgtz,id_jalr,id_jal,id_jump;
input wire[31:0]
id_busA,id_busB,id_imm32,id_HL,id_busA_mux2,id_
cp0_dout,id_busB_mux2;
input wire[4:0] id_ra,id_rb,id_rw,id_cs;
input wire[4:0] id_aluop,id_shamt;
input wire[5:0] id_op;
input wire[1:0] id_memtoreg,id_regDst;
input wire[2:0] id_cp0op,id_sel;
input wire[29:0] id_cp0_pcout,id_PC_plus_4;
input wire[25:0] id_target;

output reg
ex_regWr,ex_alusrc,ex_memwr,ex_checkover,ex_mul
tWr,ex_Lowin,ex_Highin,ex_branch_beq,ex_branch_
bne,ex_bltz,ex_blez,ex_bgez,ex_bgtz,ex_jalr,ex_
jal,ex_jump;
output reg[31:0]
ex_busA,ex_busB,ex_imm32,ex_HL,ex_busA_mux2,ex_
cp0_dout,ex_busB_mux2;
output reg[4:0] ex_ra,ex_rb,ex_rw,ex_cs;
output reg[4:0] ex_aluop,ex_shamt;
output reg[5:0] ex_op;
output reg[1:0] ex_memtoreg,ex_regDst;
output reg[2:0] ex_cp0op,ex_sel;
output reg[29:0] ex_cp0_pcout,ex_jalpc;
output reg[25:0] ex_target;

initial begin
    ex_ra = 5'd0;
```

```
        ex_rb = 5'd0;
        ex_rw = 5'd0;
        ex_busA = 32'd0;
        ex_busB = 32'd0;
        ex_imm32 = 32'd0;
        ex_HL = 32'b0;
        ex_aluop = 5'd0;
        ex_shamt = 5'd0;
        ex_regWr = 0;
        ex_multWr = 0;
        ex_regDst = 0;
        ex_alusrc = 0;
        ex_memwr = 0;
        ex_memtoreg = 2'd0;
        ex_op = 6'd0;
        ex_checkover = 0;
        ex_busA_mux2 = 32'd0;
        ex_Lowin = 0;
        ex_Highin = 0;
        ex_cp0op = 3'd0;
        ex_sel = 3'd0;
        ex_cs = 5'd0;
        ex_cp0_dout = 32'd0;
        ex_branch_beq = 0;
        ex_branch_bne = 0;
        ex_bltz = 0;
        ex_blez = 0;
        ex_bgez = 0;
        ex_bgtz = 0;
        ex_jalr = 0;
        ex_jal = 0;
        ex_jump = 0;
        ex_jalpc = 30'd0;
        ex_cp0_pcout = 30'd0;
        ex_target = 26'd0;
        ex_busB_mux2 = 32'd0;
    end

    always@(posedge clk)
    begin
        if(clk) begin
        if(BranchBubble) begin
            ex_regWr = 0;
```

```
            ex_multWr = 0;
            ex_regDst = 0;
            ex_alusrc = 0;
            ex_memwr = 0;
            ex_memtoreg = 2'd0;
            ex_checkover = 0;
            ex_aluop = 5'd0;
            ex_Lowin = 0;
            ex_Highin = 0;
            ex_branch_beq = 0;
            ex_branch_bne = 0;
            ex_bltz = 0;
            ex_blez = 0;
            ex_bgez = 0;
            ex_bgtz = 0;
            ex_jalr = 0;
            ex_jal = 0;
            ex_jump = 0;
            ex_cp0op = 3'b000;
        end
        else begin
            ex_ra = id_ra;
            ex_rb = id_rb;
            ex_rw = id_rw;
            ex_busA = id_busA;
            ex_busB = id_busB;
            ex_imm32 = id_imm32;
            ex_HL = id_HL;
            ex_aluop = id_aluop;
            ex_shamt = id_shamt;
            ex_regWr = id_regWr;
            ex_multWr = id_multWr;
            ex_regDst = id_regDst;
            ex_alusrc = id_alusrc;
            ex_memwr = id_memwr;
            ex_memtoreg = id_memtoreg;
            ex_op = id_op;
            ex_checkover = id_checkover;
            ex_busA_mux2 = id_busA_mux2;
            ex_Lowin = id_Lowin;
            ex_Highin = id_Highin;
            ex_cp0op = id_cp0op;
            if(ex_cp0op == 3'b011) begin
```

```
                ex_busB_mux2 = {id_PC_plus_4 -
1,{2'b0}};
            end
            else begin
                ex_busB_mux2 = id_busB_mux2;
            end
            ex_cs = id_cs;
            ex_sel = id_sel;
            ex_cp0_dout = id_cp0_dout;
            ex_branch_beq = id_branch_beq;
            ex_branch_bne = id_branch_bne;
            ex_bltz = id_bltz;
            ex_blez = id_blez;
            ex_bgez = id_bgez;
            ex_bgtz = id_bgtz;
            ex_jalr = id_jalr;
            ex_jal = id_jal;
            ex_jump = id_jump;
            ex_cp0_pcout = id_cp0_pcout;
            ex_jalpc = id_busA_mux2[31:2];
            ex_target = id_target;
        end
        end
    end
    endmodule
```

### 2.1.14 EXMEMREG 模块代码

```
module
ex_mem(clk,ex_zero,ex_HL,ex_result,ex_mult,ex
_busA_mux2,ex_busB_mux2,ex_rw,ex_regWr,ex_mul
tWr,ex_memwr,ex_memtoreg,ex_op,ex_Lowin,ex_Hi
ghin,ex_cp0op,ex_cs,ex_sel,ex_cp0_dout,
    mem_zero,mem_HL,mem_result,mem_mult,mem_busA
_mux2,mem_busB_mux2,mem_rw,mem_regWr,mem_multWr,
mem_memwr,mem_memtoreg,mem_op,mem_Lowin,mem_High
in,mem_cp0op,mem_cs,mem_sel,mem_cp0_dout);

input wire clk;
input wire ex_zero;
input wire[31:0]
ex_result,ex_HL,ex_busA_mux2,ex_cp0_dout,ex_busB
_mux2;
input wire[63:0] ex_mult;
```

```
input wire[4:0] ex_rw,ex_cs;
input wire
ex_regWr,ex_memwr,ex_multWr,ex_Lowin,ex_Highin;
input wire[1:0] ex_memtoreg;
input wire[5:0] ex_op;
input wire[2:0] ex_cp0op,ex_sel;

output reg mem_zero;
output reg[31:0]
mem_result,mem_HL,mem_busA_mux2,mem_cp0_dout,mem
_busB_mux2;
output reg[63:0] mem_mult;
output reg[4:0] mem_rw,mem_cs;
output reg
mem_regWr,mem_memwr,mem_multWr,mem_Lowin,mem_Hig
hin;
output reg[1:0] mem_memtoreg;
output reg[5:0] mem_op;
output reg[2:0] mem_cp0op,mem_sel;


initial begin
    mem_zero = 0;
    mem_result = 32'd0;
    mem_HL = 32'd0;
    mem_rw = 5'd0;
    mem_regWr = 0;
    mem_multWr = 0;
    mem_memwr = 0;
    mem_memtoreg = 2'd0;
    mem_op = 6'd0;
    mem_busA_mux2 = 32'd0;
    mem_Lowin = 0;
    mem_Highin = 0;
    mem_cp0op = 3'd0;
    mem_sel = 3'd0;
    mem_cs = 5'd0;
    mem_mult = 32'd0;
    mem_cp0_dout = 32'd0;
    mem_busB_mux2 = 32'd0;
end

always@(posedge clk)
```

```
begin
    mem_HL = ex_HL;
    mem_result = ex_result;
    mem_mult = ex_mult;
    mem_zero = ex_zero;
    mem_rw = ex_rw;
    mem_regWr = ex_regWr;
    mem_multWr = ex_multWr;
    mem_memwr = ex_memwr;
    mem_memtoreg = ex_memtoreg;
    mem_op = ex_op;
    mem_busA_mux2 = ex_busA_mux2;
    mem_Lowin = ex_Lowin;
    mem_Highin = ex_Highin;
    mem_cp0op = ex_cp0op;
    mem_sel = ex_sel;
    mem_cs = ex_cs;
    mem_cp0_dout = ex_cp0_dout;
    mem_busB_mux2 = ex_busB_mux2;
end

endmodule
```

### 2.1.15 MEMWRREG 模块代码

```
module
mem_wr(clk,mem_dout,mem_HL,mem_result,mem_mult,mem
_busA_mux2,mem_busB_mux2,mem_rw,mem_regWr,mem_mult
Wr,mem_memtoreg,mem_op,mem_Lowin,mem_Highin,mem_cp
0op,mem_cs,mem_sel,mem_cp0_dout,
    wr_dout,wr_HL,wr_result,wr_mult,wr_busA
_mux2,wr_busB_mux2,wr_rw,wr_regWr,wr_multWr,
wr_memtoreg,wr_op,wr_Lowin,wr_Highin,wr_cp0o
p,wr_cs,wr_sel,wr_cp0_dout);
input wire clk;
input wire[31:0]
mem_dout,mem_result,mem_HL,mem_busA_mux2,mem
_cp0_dout,mem_busB_mux2;
input wire[63:0] mem_mult;
input wire[4:0] mem_rw,mem_cs;
input wire
mem_regWr,mem_multWr,mem_Lowin,mem_Highin;
input wire[1:0] mem_memtoreg;
input wire[5:0] mem_op;
```

```
input wire[2:0] mem_cp0op,mem_sel;

output reg[31:0]
wr_dout,wr_result,wr_HL,wr_busA_mux2,wr_cp0_
dout,wr_busB_mux2;
output reg[63:0] wr_mult;
output reg[4:0] wr_rw,wr_cs;
output reg
wr_regWr,wr_multWr,wr_Lowin,wr_Highin;
output reg[1:0] wr_memtoreg;
output reg[5:0] wr_op;
output reg[2:0] wr_cp0op,wr_sel;

initial begin
    wr_dout = 32'd0;
    wr_result = 32'd0;
    wr_HL = 32'd0;
    wr_rw = 5'd0;
    wr_regWr = 0;
    wr_multWr = 0;
    wr_memtoreg = 2'd0;
    wr_op = 6'd0;
    wr_busA_mux2 = 32'd0;
    wr_Lowin = 0;
    wr_Highin = 0;
    wr_cp0op = 3'd0;
    wr_sel = 3'd0;
    wr_cs = 5'd0;
    wr_cp0_dout = 32'd0;
    wr_busB_mux2 = 32'd0;
    wr_mult = 64'd0;
end


always@(posedge clk)
begin
    wr_dout = mem_dout;
    wr_HL = mem_HL;
    wr_result = mem_result;
    wr_mult = mem_mult;
    wr_rw = mem_rw;
    wr_regWr = mem_regWr;
    wr_multWr = mem_multWr;
```

```
        wr_memtoreg = mem_memtoreg;
        wr_op = mem_op;
        wr_busA_mux2 = mem_busA_mux2;
        wr_Lowin = mem_Lowin;
        wr_Highin = mem_Highin;
        wr_cp0op = mem_cp0op;
        wr_sel = mem_sel;
        wr_cs = mem_cs;
        wr_cp0_dout = mem_cp0_dout;
        wr_busB_mux2 = mem_busB_mux2;
    end
    endmodule
```

### 2.1.16 Branchforward 模块代码（用于 id 段读取数据处理转发）

```
module
branchforward(id_rs,id_rt,ex_cp0op,ex_rw,ex_regWr,mem_cp0op,mem_rw,mem_reg
Wr,wr_cp0op,wr_rw,wr_regWr,branchforwardA,branchforwardB);

    input wire[4:0]
    id_rs,id_rt,mem_rw,
    ex_rw,wr_rw;
    input
    mem_regWr,ex_regWr,
    wr_regWr;
    input wire[2:0]
    ex_cp0op,mem_cp0op,
    wr_cp0op;
    output reg
    [1:0]branchforwardA
    ,branchforwardB;

    initial begin
        branchforwardA =
    2'd0;
        branchforwardB =
    2'd0;
    end

    always@(*) begin
        if((ex_regWr ==
    1||ex_cp0op ==
    3'b001) && ex_rw != 0
    && ex_rw == id_rs)
```

```
    branchforwardA =
2'b01;
    else
if((mem_regWr ==
1||mem_cp0op ==
3'b001) && mem_rw !=
0 && mem_rw == id_rs)

    branchforwardA =
2'b10;
    else
if((wr_regWr ==
1||wr_cp0op ==
3'b001) && wr_rw != 0
&& wr_rw == id_rs)

    branchforwardA =
2'b11;
    else

    branchforwardA =
2'b00;

    if((ex_regWr ==
1||ex_cp0op ==
3'b001) && ex_rw != 0
&& ex_rw == id_rt)

    branchforwardB =
2'b01;
    else
if((mem_regWr ==
1||mem_cp0op ==
3'b001) && mem_rw !=
0 && mem_rw == id_rt)

    branchforwardB =
2'b10;
    else
if((wr_regWr ==
1||wr_cp0op ==
3'b001) && wr_rw != 0
```

```
&& wr_rw == id_rt)

    branchforwardB =
2'b11;
    else

    branchforwardB =
2'b00;
end

endmodule
```

### 2.1.17 Load-use 模块代码（用于解决 load-use 冒险）

```
module
branchbubble(id_rs,id_rt,ex_regWr,ex_rw,ex
_memtoreg,branchbubble);

input wire[4:0] id_rs,id_rt,ex_rw;
input wire ex_regWr,ex_memtoreg;
output reg branchbubble;

initial begin
    branchbubble = 0;
end

always@(*) begin
    if((ex_regWr==1&&ex_memtoreg==1)&&(ex_rw!=0)&&(
ex_rw==id_rs||ex_rw==id_rt)) begin
        branchbubble = 1;
    end
    else begin
        branchbubble = 0;
    end

end

endmodule
```

### 2.1.18 cp0forward 模块代码（用于解决 cp0 寄存器的数据冒险）

```
module
cp0forwardUnit(id_cp0op,id_cs,id_sel,ex_c
s,ex_sel,ex_cp0op,mem_cs,mem_sel,mem_cp0o
```

```
p,wr_cs,wr_sel,wr_cp0op,cp0forward);

    input wire[2:0]
    id_cp0op,ex_cp0op,mem_cp0op,wr_cp0op,id_sel,ex_sel,me
    m_sel,wr_sel;
    input wire[4:0] id_cs,ex_cs,mem_cs,wr_cs;
    output reg[1:0] cp0forward;

    initial begin
        cp0forward = 2'b00;
    end

    always@(*) begin
        if((id_cp0op==3'b001&&ex_cp0op==3'b010&&id_cs==ex
    _cs&&id_sel==ex_sel)||(id_cp0op==3'b100&&ex_cp0op==3'
    b010&&ex_cs==14&&ex_sel==0))
            cp0forward = 2'b01;
        else
    if((id_cp0op==3'b001&&mem_cp0op==3'b010&&ex_cp0op!=3'
    b11&&id_cs==mem_cs&&id_sel==mem_sel)||(id_cp0op==3'b1
    00&&mem_cp0op==3'b010&&mem_cs==14&&mem_sel==0))
            cp0forward = 2'b10;
        else
    if((id_cp0op==3'b001&&wr_cp0op==3'b010&&ex_cp0op!=3'b
    11&&id_cs==wr_cs&&id_sel==wr_sel)||(id_cp0op==3'b100&
    &wr_cp0op==3'b010&&mem_cs==14&&mem_sel==0))
            cp0forward = 2'b11;
        else
            cp0forward = 2'b00;
    end

    endmodule
```

### 2.1.19 HLforward 模块代码（用于解决 HiLo 寄存器的数据冒险）

```
module
HLforward(ex_Highin,ex_Lowin,mem_Highin,mem_Lowin,wr_High
in,wr_Lowin,forwardHigh,forwardLow);

    input wire
    ex_Highin,mem_Highin,wr_Highin,ex_Lo
    win,mem_Lowin,wr_Lowin;
    output reg[1:0]
    forwardHigh,forwardLow;
```

```
initial begin
    forwardHigh = 2'b00;
    forwardLow = 2'b00;
end

always@(*) begin
    if(ex_Highin == 1)
        forwardHigh = 2'b01;
    else if(mem_Highin == 1)
        forwardHigh = 2'b10;
    else if(wr_Highin == 1)
        forwardHigh = 2'b11;
    else
        forwardHigh = 2'b00;

    if(ex_Lowin == 1)
        forwardLow = 2'b01;
    else if(mem_Lowin == 1)
        forwardLow = 2'b10;
    else if(wr_Lowin == 1)
        forwardLow = 2'b11;
    else
        forwardLow = 2'b00;
end

endmodule
```

### 2.1.20 multforward 模块代码（用于解决 HiLo 寄存器的数据冒险）

```
module
multforward(id_multWr,ex_multWr,ex_Highin,ex_Lowin,me
m_multWr,mem_Highin,mem_Lowin,wr_multWr,multforward);

input wire
id_multWr,ex_multWr,mem_multWr,wr_multWr
,ex_Highin,ex_Lowin,mem_Highin,mem_Lowin
;
output reg[1:0] multforward;

initial begin
    multforward = 2'd0;
end

always@(*) begin
```

```
        if(ex_multWr == 1)
            multforward = 2'b01;
        else if(mem_multWr ==
1&&(ex_Lowin!=1||ex_Highin!=1))
            multforward = 2'b10;
        else if(wr_multWr ==
1&&((mem_Lowin!=1||mem_Highin!=1)||(ex_L
owin!=1||ex_Highin!=1)))
            multforward = 2'b11;
        else
            multforward = 2'b00;
    end
    endmodule
```

### 2.1.21 zerobranchjudge 模块（用于校验跳转条件是否成立）

```
module
branchjudge(busA,busB,bgez,bgtz,blez,bltz,branch_beq,branch_b
ne,zbgez,zbgtz,zbeq,zbne);
        input wire[31:0]
    busA,busB;
        input wire
    bgtz,bgez,blez,bltz,branch_be
    q,branch_bne;
        output reg
    zbgez,zbgtz,zbeq,zbne;

    initial begin
        zbgez = 0;
        zbgtz = 0;
        zbeq = 0;
        zbne = 0;
    end

    always@(*) begin
        if(bgez == 1) begin
            zbgez = (busA[31]==0);
        end
        else if(bgtz == 1) begin
            zbgtz =
    (busA!=32'b0&&busA[31]==0);
        end
        else if(blez == 1) begin
            zbgtz
```

```
= !(busA[31]==1||busA==32'b0);
    end
    else if(bltz == 1) begin
        zbgez
= !(busA[31]==1);
    end
    else if(branch_beq == 1)
begin
        zbeq = (busA==busB);
    end
    else if(branch_bne == 1)
begin
        zbne = (busA!=busB);
    end
end

endmodule
```

### 2.1.22 CP0 寄存器模块

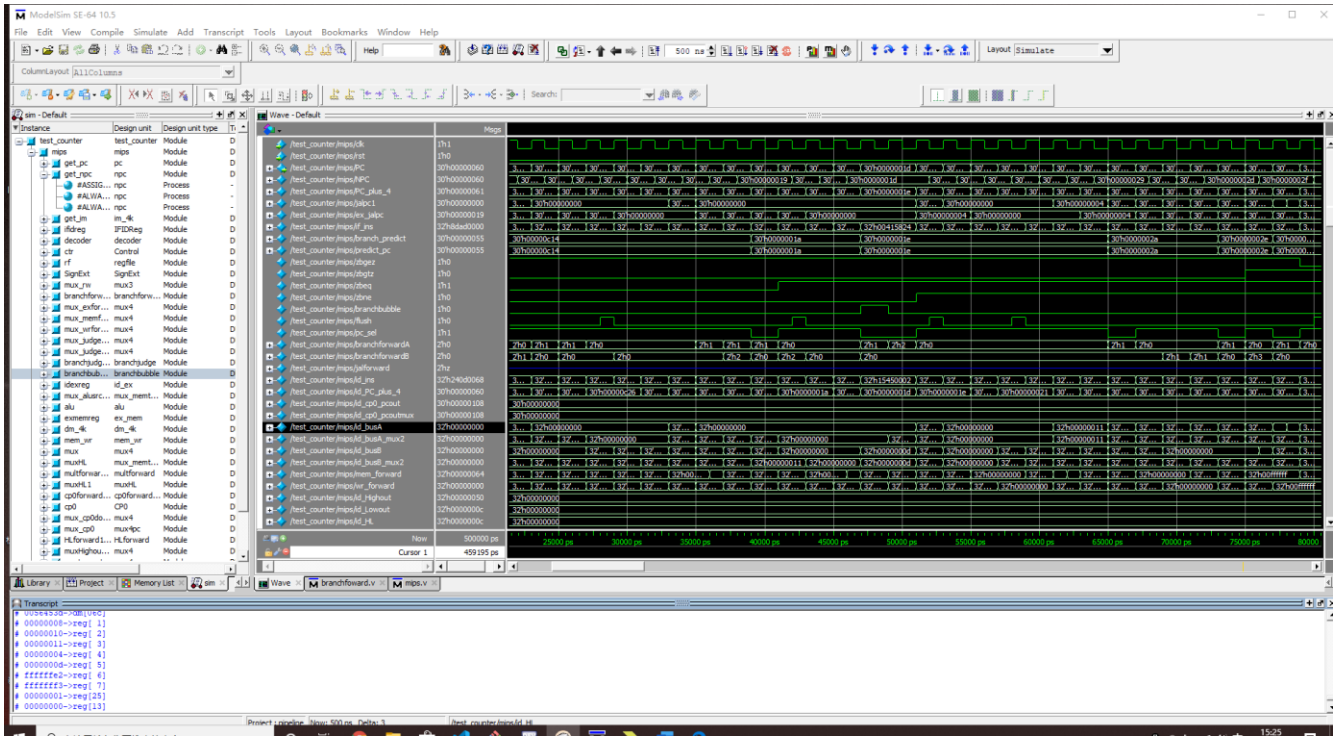
```
module
CP0(clk,id_cs,id_sel,wr_cs,wr_sel,busB,PC,cp0_dout,cp0_pco
ut,wr_cp0op,id_cp0op);

input wire clk;
input wire[4:0] id_cs,wr_cs;
input wire[2:0]
id_sel,wr_sel,wr_cp0op,id_cp0op;
input wire[31:0] busB;
input wire[29:0] PC;
output reg[31:0] cp0_dout,cp0_pcout;

reg[31:0] cp0[0:255];

integer i;
initial begin
    for(i=0;i<=255;i=i+1)
        cp0[i]=0;
    cp0_dout = 32'd0;
    cp0_pcout = 30'd0;
end

always@(negedge clk) begin
    if(wr_cp0op == 3'b010) begin
        cp0[{wr_cs,wr_sel}] = busB;
```

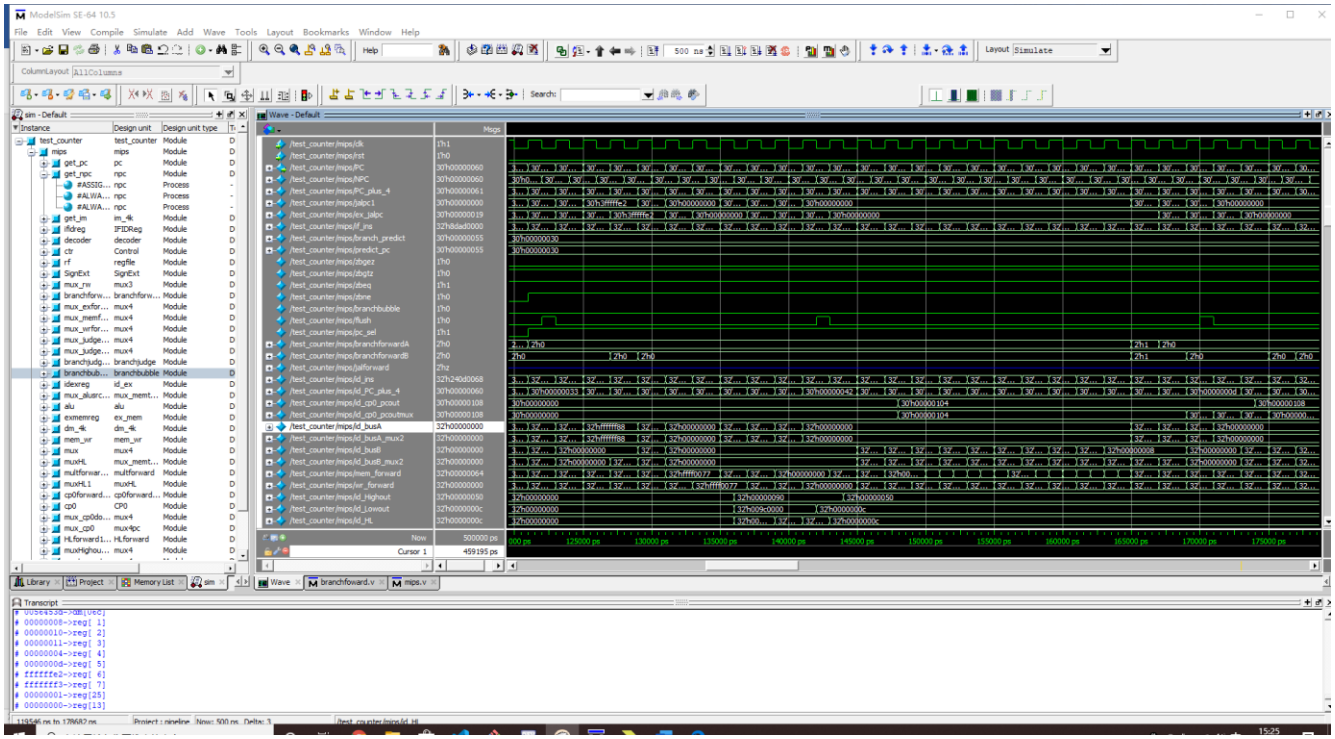
```
        $display("%h->cp0[%d,%d]",busB,
wr_cs,wr_sel);
        end

        if(wr_cp0op == 3'b011) begin
            cp0[112] = busB;
            cp0[104][6:2] = 5'b01000;
            cp0[96][1] = 1;

        $display("%h->cp0[14,0]",busB);
        end

        if(wr_cp0op == 3'b100)  begin
            cp0[96][1] = 0;
        end
    end

    always@(*) begin
        if(id_cp0op == 3'b001)
            cp0_dout =
    cp0[{id_cs,id_sel}];


        cp0_pcout = cp0[112];
    end

    endmodule
```

## 2.2 Modelsim 模拟及测试

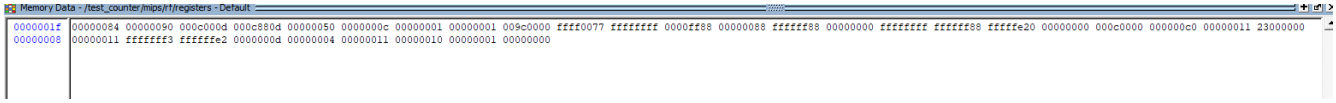
**测试指令如群里文件发的 45 条流水线指令**

**测试结果如下：**

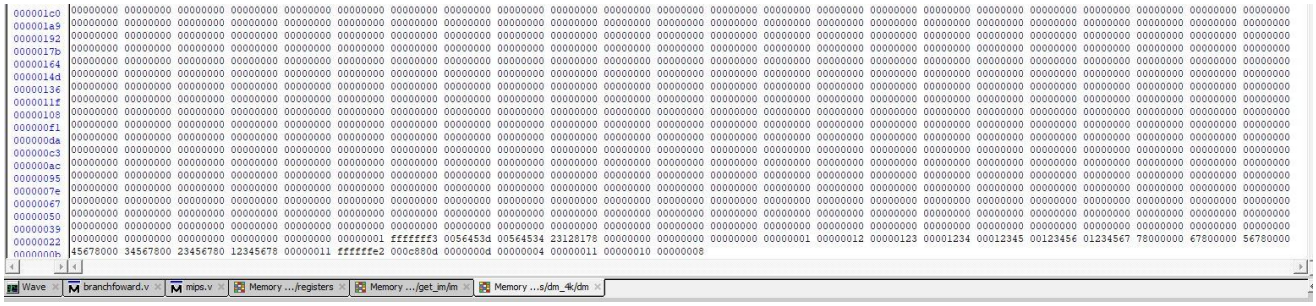




经校验，符合样例所给的结果。



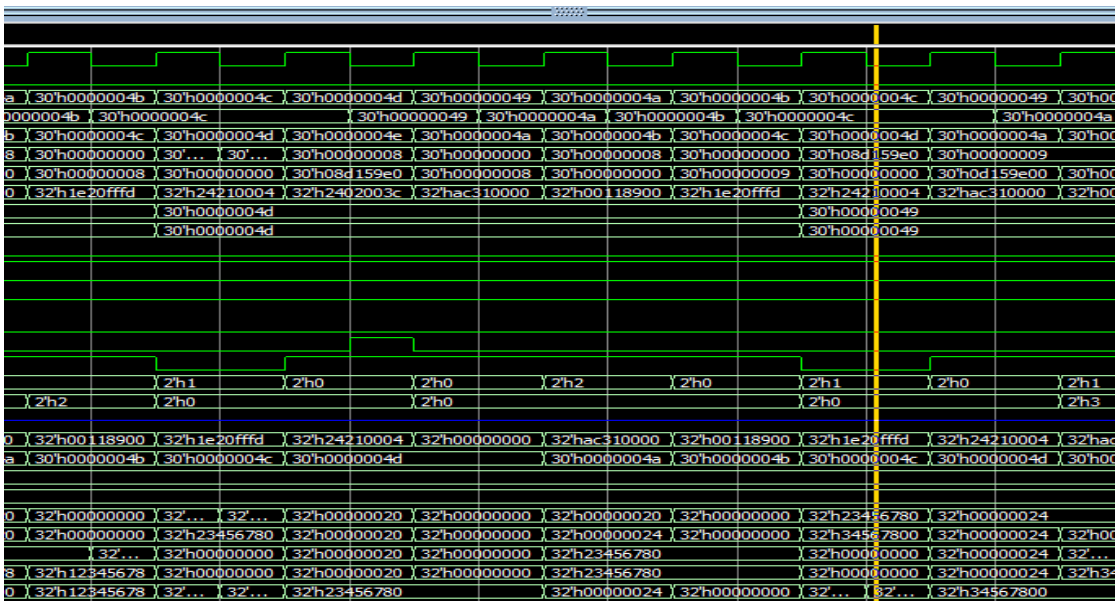

寄存器数据情况也符合




DataMemory 最终情况。同样符合样例测试。

1. 关于延迟槽和动态预测

延迟槽是一种通过改变指令位置来加快流水线 CPU 运行效率的手段，即在遇到分支跳转指令时通过运行分支指令后一条指令之后再跳转，相较于无延迟槽减少了阻塞时钟周期导致的效率变低。如下图所示，1e20fffd 为跳转指令，在进行预测后（不跳转预测），跳转到了预测地址并执行下一指令，ex 段发现错误取消预测并修正预测位信息阻塞多余指令，在下一次直接预测正确位置并进行跳转在 ex 判断预测正确后不需要阻塞，所以实现了动态预测和分支延迟槽技术，但 BHT 表（预测表）结构复杂，容易导致 CPU 速度降低，且和分支延迟槽共用会降低 CPU 处理效率。

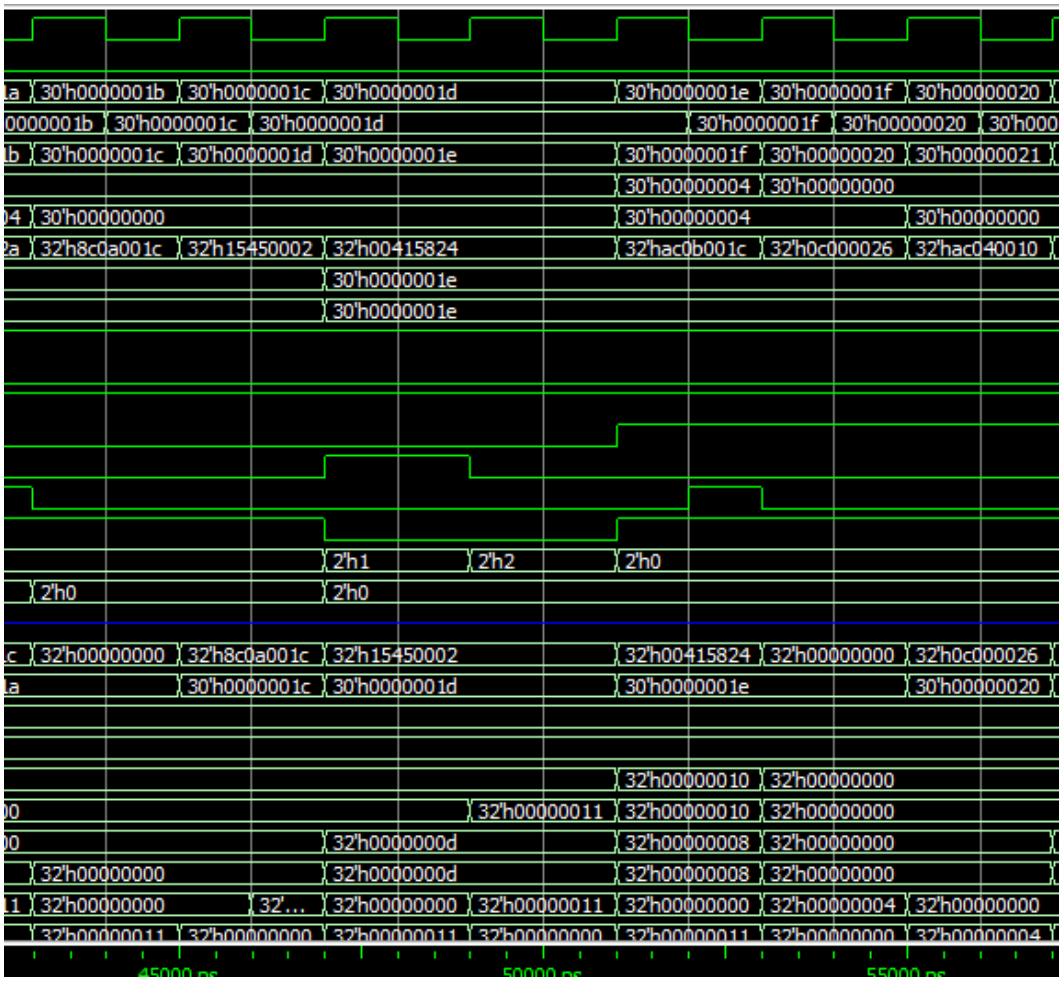


2. load-use 冒险

可以看到在执行了 lw 指令时，beq 指令在 ex 阶段执行了阻塞，等到下一时钟周期成功完成跳转到预测地址，并且根据错误的预测地址进行修正后清空多余指令。

| | | | | |
|---|---|---|---|---|
| 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 |

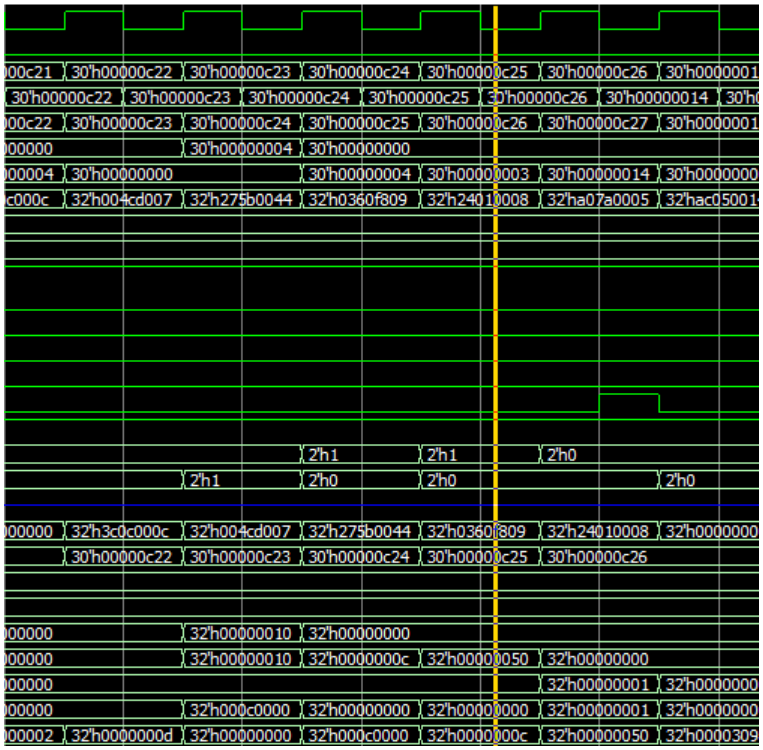
表 2 CPU 测试 45 条指令所用汇编程序详述（续）




3. 普通数据转发

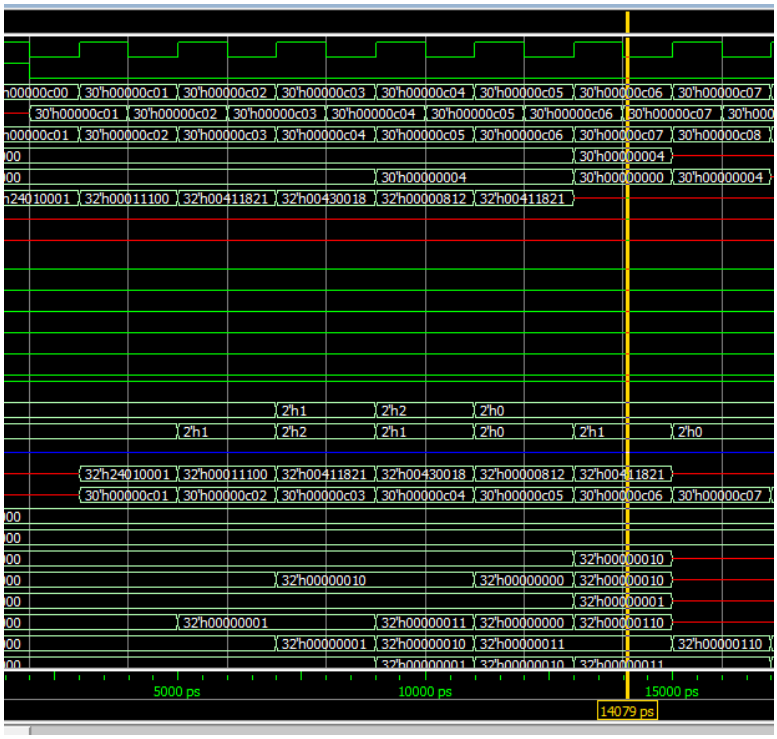
由图和代码样例可见，先是将 00000050H 存入 27 号寄存器，并在 jalr 指令 ex 阶段调用，调用成功，在 jalr 指令 id 阶段，所得到的 jalpc 就已经是 00000050H，说明该数据已经经过数据转转发到当前。

| | | | | |
|---|---|---|---|---|
| 8CH | addiu $27,$26,#68 | [$27] = 0000_0050H | 275B0044 | 0010_0111_0101_1011_0000_0000_0100_0100 |
| 90H | jalr $27 | 跳转到 50H,<br>[$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] | A07A0005 | 1010_0000_0111_1010_0000_0000_0000_0101 |



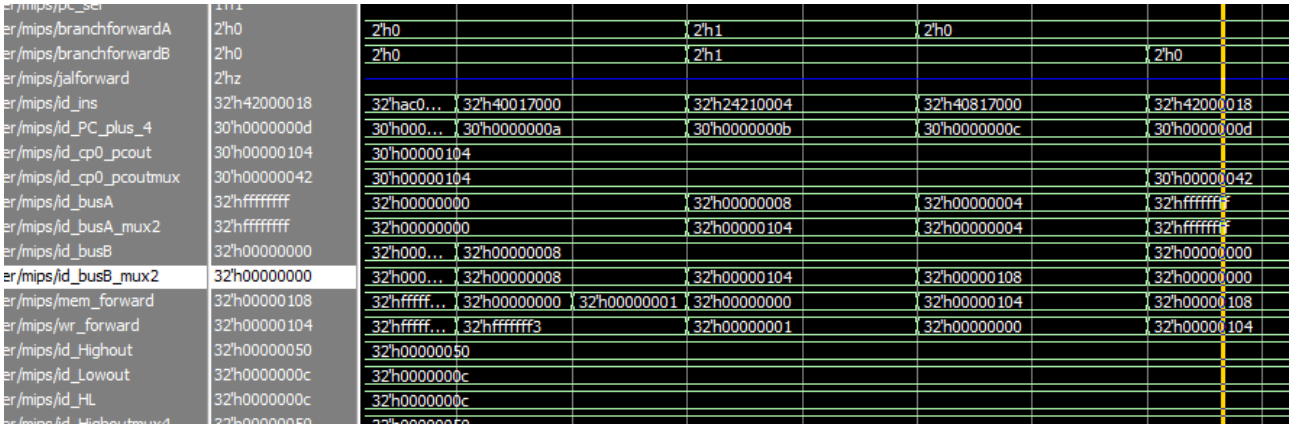
4. HILO 寄存器数据转发

该指令完成的是 1->reg[1], reg[1]<<1->reg[2], reg[2]+reg[1]->reg[3], reg[2]*reg[3]->HILO, LO->reg[1], reg[1]+reg[2]->reg[3]，可以看到，数据转发没有出现问题，并且顺利完成指令。

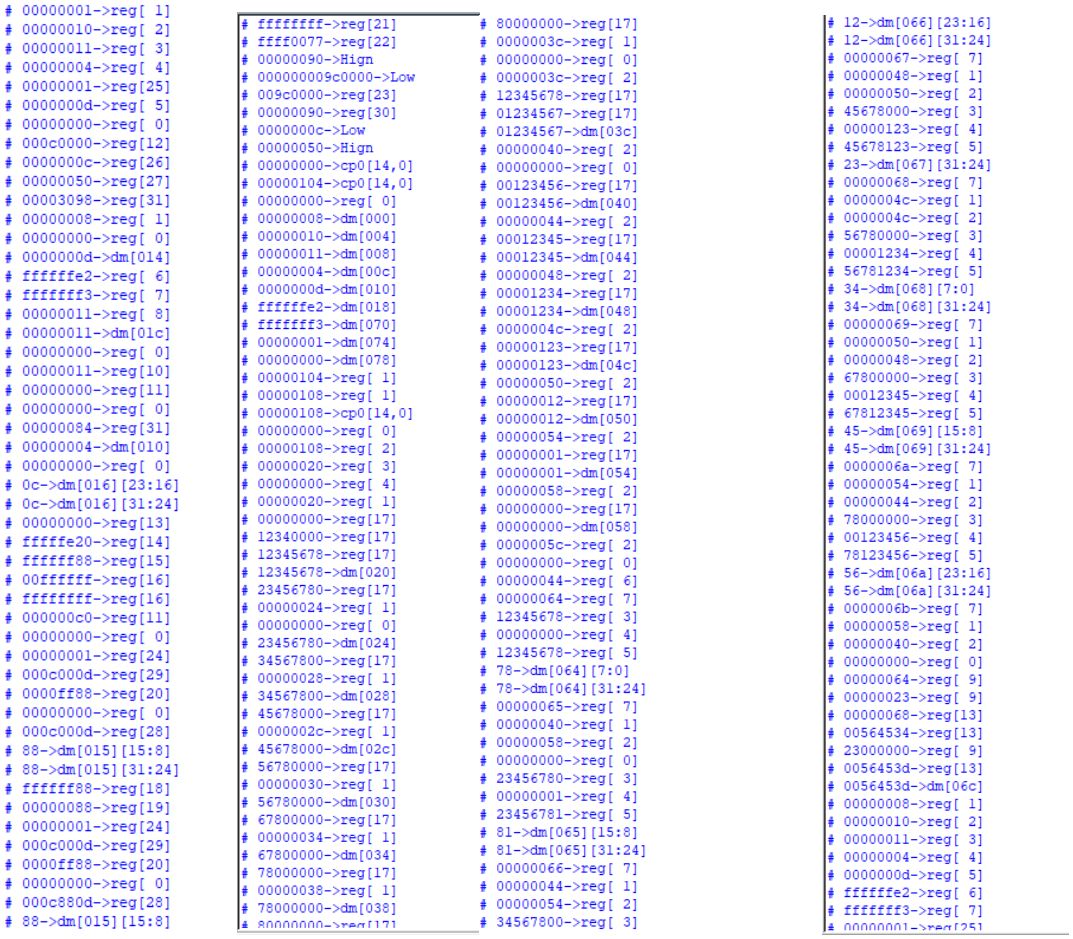


5. Cp0 寄存器数据转发

| | | | | |
|---|---|---|---|---|
| 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 | | | | |




可见在 mtc0 之后，根据指令的信号比较，在 eret 读取 cp0 寄存器是将还没有写入 cp0 寄存器的 108H 送到了 id_pcoutmux，在之后完成了成功跳转。



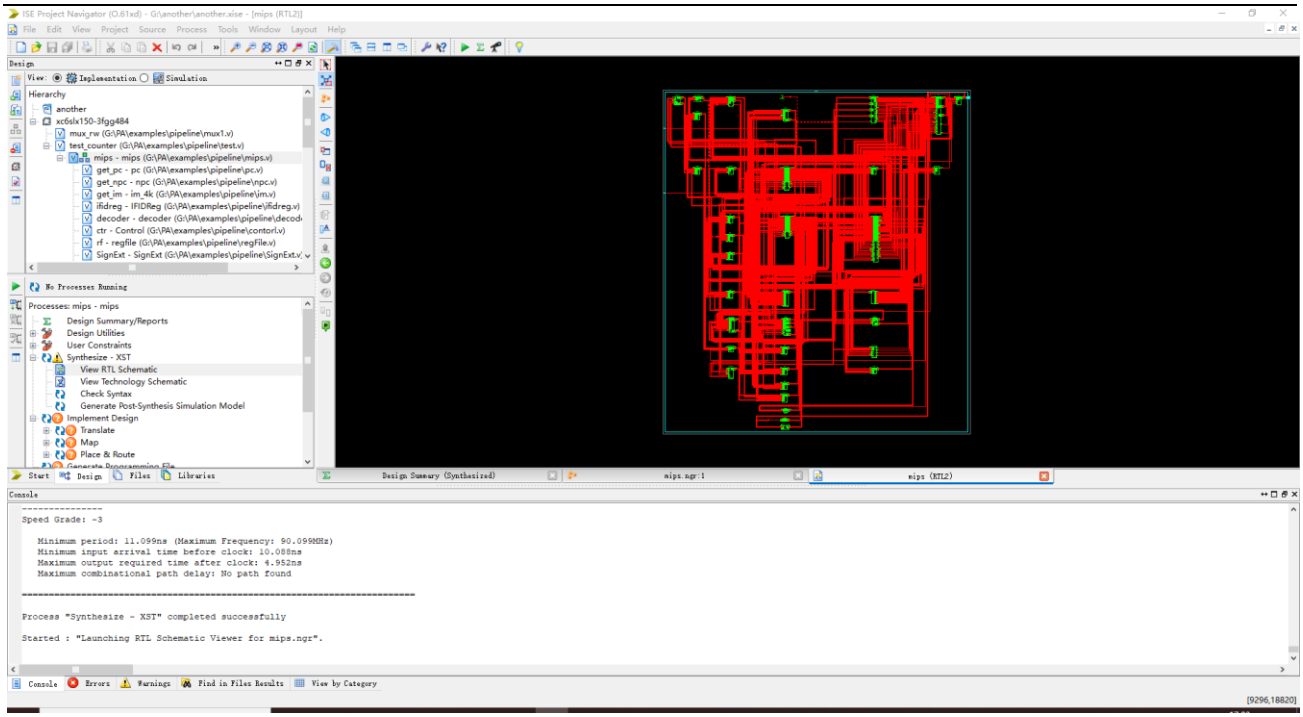

上图为每一步操作的实际用处（dm 的输出有 bug 只看第一个，0->reg[0]就是插入的 nop 气

泡）。

附资源调用和最高频率截图。

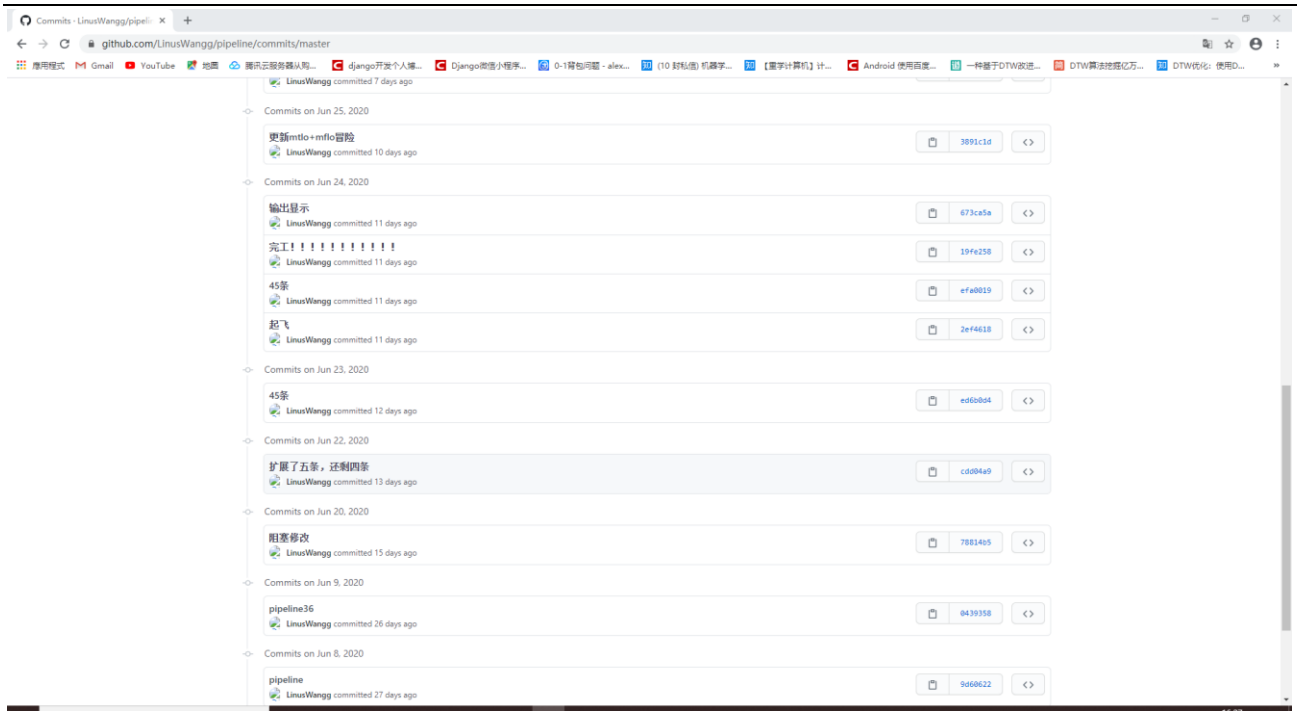
| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 10,337 | 184,304 | 5% | |
| Number used as Flip Flops | 9,614 | | | |
| Number used as Latches | 723 | | | |
| Number used as Latch-thrus | 0 | | | |
| Number used as AND/OR logics | 0 | | | |
| Number of Slice LUTs | 15,907 | 92,152 | 17% | |
| Number used as logic | 15,343 | 92,152 | 16% | |
| Number using O6 output only | 14,746 | | | |
| Number using O5 output only | 75 | | | |
| Number using O5 and O6 | 522 | | | |
| Number used as ROM | 0 | | | |
| Number used as Memory | 512 | 21,680 | 2% | |
| Number used as Dual Port RAM | 0 | | | |
| Number used as Single Port RAM | 512 | | | |
| Number using O6 output only | 512 | | | |
| Number using O5 output only | 0 | | | |
| Number using O5 and O6 | 0 | | | |
| Number used as Shift Register | 0 | | | |
| Number used exclusively as route-thrus | 52 | | | |
| Number with same-slice register load | 50 | | | |
| Number with same-slice carry load | 2 | | | |
| Number with other load | 0 | | | |
| Number of occupied Slices | 4,510 | 23,038 | 19% | |
| Number of LUT Flip Flop pairs used | 16,485 | | | |
| Number with an unused Flip Flop | 6,263 | 16,485 | 37% | |
| Number with an unused LUT | 578 | 16,485 | 3% | |
| Number of fully used LUT-FF pairs | 9,644 | 16,485 | 58% | |
| Number of unique control sets | 74 | | | |
| Number of slice register sites lost to control set restrictions | 207 | 184,304 | 1% | |
| Number of bonded IOBs | 32 | 338 | 9% | |
| Number of RAMB16BWERs | 0 | 268 | 0% | |
| Number of RAMB8BWERs | 2 | 536 | 1% | |
| Number of BUFIO2/BUFIO2_2CLKs | 0 | 32 | 0% | |
| Number of BUFIO2FB/BUFIO2FB_2CLKs | 0 | 32 | 0% | |
| [illegible] | [illegible] | [illegible] | [illegible] | |
| Number of BUFPLLs | 0 | 8 | 0% | |
| Number of BUFPLL_MCBs | 0 | 4 | 0% | |
| Number of DSP48A1s | 4 | 180 | 2% | |
| Number of ICAPs | 0 | 1 | 0% | |
| Number of MCBs | 0 | 4 | 0% | |
| Number of PCILOGICSEs | 0 | 2 | 0% | |
| Number of PLL_ADVs | 0 | 6 | 0% | |
| Number of PMVs | 0 | 1 | 0% | |
| Number of STARTUPs | 0 | 1 | 0% | |
| Number of SUSPEND_SYNCs | 0 | 1 | 0% | |
| Average Fanout of Non-Clock Nets | 6.14 | | | |

```
---------------
Speed Grade: -3

   Minimum period: 11.099ns (Maximum Frequency: 90.099MHz)
   Minimum input arrival time before clock: 10.088ns
   Maximum output required time after clock: 4.952ns
   Maximum combinational path delay: No path found

=========================================================================

Process "Synthesize - XST" completed successfully

Started : "Launching RTL Schematic Viewer for mips.ngr".
```



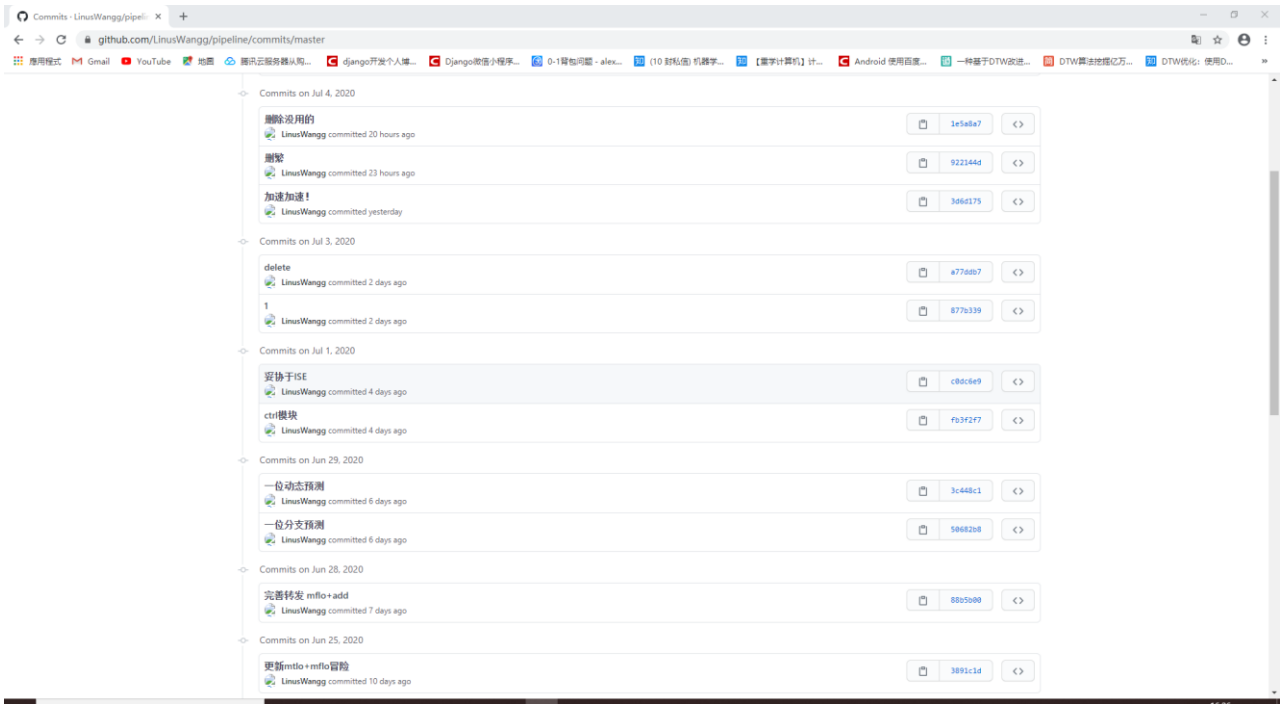

# 第三章　总结与展望

从一开始自己写的 36 条单周期 CPU 到现在做了一个 45 条指令的五级流水线 CPU 带各种转发冒险，感觉学到了很多，在一开始动手写流水线的时候，是非常迷茫的，只能去问以前学过的学长，而学长有时候对于流水线 CPU 的理解并不是很深刻，就得自己去吃书去搜索引擎或者去参考别人的代码，后来慢慢自己写完了能实现 11 条指令的流水线 CPU，但大部分都是通过阻塞来实现，还没完全实现转发。在扩展 36 条的时候，由于自身时间十分紧张，基本一天里有半天是在学车的，所以最多只能剩下半天时间来赶代码，在一开始写的特别的矛盾，甚至在 36 条完成之后，用于判断分支跳转的数据转发和 ALU 器件的数据转发还是用的两个不同的，直到扩展 45 条的时候才发现可以将这两个合并，于是进行了很多的修改，并且由于原来将分支跳转写在了 id 阶段，alu 写在了 ex 阶段，所以导致出现了两个 load-use 模块，即作用相同仅仅名字不同而已。在经过了一系列大换血和大刀阔斧的修改之后，终于完成了 45 条指令的扩展，但由于原先并没有整理好思路，即转发逻辑，所以还是扩展了有两天，后来又发现对于 HILO 寄存器又存在冒险，于是又重新设计转发，大约总共三天时间，才算完成。（附上 GitHub 的 commit 图）这是第一阶段。




大概就是自己完成 45 条 CPU 的血泪史，后来老师上课又说了能够实现一位动态预测等高端技术更好，于是自己又开始了吃书，将计组书看了两三遍后，完成了 BHT 表的建立和一位动态预测技术的实现，但是在后来与老师进行争论时，觉得在有延迟槽情况下静态预测相较于动态预测能够省去很多步骤，但后来又想想，BHT 通过顺序查找（甚至可以通过哈希表查找）完成对于 NPC 的变化，仅仅会丢失几个时钟周期，对于多重循环来说，能有效减少每次静态预测计算所消耗的时间。这是课设第二阶段。

接下来是课设第三阶段，恐怖的效率提升阶段。在老师下发了 ISE 测试的要求后，虽然一开始总觉得老师发错了实验指导书，但还是通过自己的百度和碰巧操作，实现了流水线 CPU 架构的连线和器件消耗的测试和最高频率的测试。在修改了许多器件（如删除了无关变量和修改 BHT 表，修改转发阶段，中间也遇到了不少 bug 的修复），最终将自己的最高频率在老师所要求的板子上达到了 90MHZ（在别人电脑上跑到了 114MHZ）。其中修改了很多，也遇到了很多令自己崩溃的地方，比如，修改了半天的代码导致比之前的速度更慢，代码版本保存错误，多次经历写，删，再写回，再删，再改，再回到原来，再写这种很迷幻的操作，虽然对提升效率有了一定的理解，但架不住多次进行修改导致身心俱疲。




写完了 45 条流水线 CPU，感觉自己身心已经很轻松了，觉得自己在跟别人的讨论中学到了很多，也学会了多多汲取别人的经验和教训。

# 第四章　参考文献

[1]袁春风，等. 计算机组成与系统结构[M]. 第2 版，北京：清华大学出版社 2010.

# 第五章　致谢

感谢老师对于每次我提出的问题都很认真的回答，感谢李皓琨同学对我在指令理解上的帮助，粟子淳，周海鹏同学和我一起讨论得到的结果，周鹤洋同学在课上对于加快效率的细心讲解。