

---

# MIPS 流水线 CPU 设计

王一伊

## 主要内容

用硬件描述语言（Verilog）设计 MIPS 流水线 CPU，支持如下指令集：

{add, sub, ori, lw, sw, beq, j }

用仿真软件 Modelsim 对有数据冒险和指令冒险的汇编程序进行仿真：

如：

```
ori $29, $0, 12
ori $2, $0, 0x1234
ori $3, $0, 0x3456
addu $4, $2, $3
subu $6, $3, $4
sw $2, 0($0)
sw $3, 4($0)
sw $4, 4($29)
lw $5, 0($0)
beq $2, $5, _lb2
_lb1:
lw $3, 4($29)
_lb2:
lw $5, 4($0)
beq $3, $5, _lb1
subu $6, $6, $2
```

## 基本要求

用硬件描述语言（Verilog）设计如下模块：

- (1) 程序计数器模块（PC）；
- (2) 指令存储器模块（im）；
- (3) 寄存器模块（RF）；
- (4) 流水线寄存器（IFIDreg、IDEXreg、EXMEMreg、MEMWBreg）；

- 
- (5) 数据扩展模块 (EXT) ;
  - (6) 运算器模块 (alu) ;
  - (7) 数据存储器模块 (dm) ;
  - (8) 控制器模块 (CTRL) ;
  - (9) 整机连接模块 (MIPS) (包括数据冒险和控制冒险检测) ;
  - (10) 数据冒险检测单元 (MUX1, MUX2)
  - (11) PC 选择单元 (PCjudge)
  - (12) ALU 结果选择 (ALUjudge)
  - (13) LW 冒险检测单元 (Check)
  - (14) LW 冒险阻塞信号控制单元 (HD)
  - (15) ALU 的 B 输入选择 (MUX3)

完成汇编程序的仿真调试。

---

## 参考资料

- 【1】** 计算机原理与设计：Verilog HDL 版，李亚民 著
- 【2】** Verilog 数字系统设计教程 夏宇闻编著（第 17 章 简化的 RISC CPU 设计）

---

## 1. 需求分析

在学习计算机组成原理课程之后，需要将 CPU 的结构原理与 CPU 流水线原理与设计以及数据冒险控制冒险等知识融会贯通。通过对知识的综合应用，加深对 CPU 系统各模块的工作原理与流水线及其相关冒险的解决方式的理解，通过实验更加深刻全面学习 CPU 流水线设计。

## 2. 设计环境

### 2.1 Verilog HDL 简介

它是一种硬件描述语言，以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。为 IEEE 标准。

### 2.1 ModelSim 简介

它是 HDL 语言仿真软件，能提供友好的仿真环境，是唯一的单内核支持 VHDL 和 VERILOG 混合仿真的仿真器。他采用直接优化的编译技术、TCI/TK 技术、和单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段，是 FPGA/ASIC 设计的首选仿真软件。

---

## 2.3 MARS 简介

MIPS 汇编语言模拟机，由密苏里州立大学开发。

## 3. 概要设计

### 3.1 PC (程序计数器)

#### (1) 功能描述

进行 PC+4 的指令计数并根据信号进行指令选择。

#### (2) 模块接口

信号名	方向	描述
clk	input	时钟
rst	input	复位信号
PCWr	input	该信号为 1 时说明分支发生将 PC 置为 BEQ 的跳转地址
NPC[31:0]	input	BEQ 指令的跳转地址
PC[31:0]	output	PC 单元最终输出 PC 的指令
PCjump	input	该信号为 1 并且分支信号为 0 时，说明发生跳转，PC 将赋值为 JUMP 指令的跳转地址
PCrun	input	该信号为 1 时 PC 单元正常运行，信号为 0 时 PC 单元不可写入，PC 保持上一周期的值
block	input	该信号在我编写单元时做调试用，实际在本单元中不起作用

---

## 3. 2 RF（寄存器）设计

### （1）功能描述

根据读寄存器输入的两个地址读出对应寄存器地址的两个数据，根据写回地址与写回数据进行寄存器写回。

### （2）模块接口

信号名	方向	描述
A1[4:0]	input	寄存器读取数据的地址
A2[4:0]	input	寄存器读取数据的地址
A3[4:0]	input	寄存器读取数据的地址
WD[31:0]	input	寄存器写回的数据
clk	input	时钟信号
RFWr	input	寄存器写信号 RFWr=1 寄存器可写
RD1[31:0]	output	RD1 为寄存器输出的数据
RD2[31:0]	output	RD2 为寄存器输出的数据

## 3. 3 ALU（算术逻辑运算单元）

### （1）功能描述

ALU 主要功能是完成对输入数据的进行加法、减法、与、或、左移、右移、乘法、除法运算以及判断两个操作数是否相等。

### （2）模块接口

信号名	方向	描述
A [31:0]	I	操作数 A
B [31:0]	I	操作数 B
ALUOp[4:0]	I	需要进行的运算 `ALUOp_ADDU: 加法

---

		`ALUOp_SUBU: 减法 `ALUOp_OR: 或运算
Zero	O	两操作数是否相等
C [31:0]	O	运算结果

### 3.4 EXT (扩展单元)

#### (1) 功能描述

EXT 主要功能是将 16 位的数据扩展为 32 位数据、将数据左移两位。

#### (2) 模块接口

信号名	方向	描述
Imm16 [15:0]	I	需要进行扩展的数据
EXTOp[1:0]	I	扩展方式的控制信号 `EXT_ZERO: 0 扩展 `EXT_SIGNED: 符号扩展,并将立即数左移两位 `EXT_HIGHPOS: 将立即数扩展到高位
Imm32 [31:0]	O	扩展结果

### 3.5 DM (数据存储器)

#### (1) 功能描述

根据输入的地址与存储器读写控制信号读出相应地址的数据或将数据写入到相应地址中。

---

## (2) 模块接口

信号名	方向	描述
addr[4:0]	I	写入的寄存器号
din[31:0]	I	写入存储器的数据
DMWr	I	存储器写数据信号
DMemR	I	存储器读数据信号
clk	I	时钟
dout[31:0]	O	存储器读出的数据

## 3.6 IM (指令存储器)

### (1) 功能描述

在该单元中，根据输入的读地址，将相应的指令从存储器中读出

### (2) 模块接口

信号名	方向	描述
addr[4:0]	I	指令存储器的读地址
dout[31:0]	O	指令存储器读出的指令

## 3.7 Ctrl (控制器)

### (1) 功能描述

根据输入指令的高六位 OP 操作码判断当前指令的类别与功能，输出相应单元的控制信号。

### (2) 模块接口

信号名	方向	描述

instr[5:0]	I	输入的操作码
funct[5:0]	I	输入的功能码
jump	O	跳转信号
RegDst	O	寄存器写回地址选择信号
Branch	O	分支信号
DMemR	O	存储器读信号
DMnoutregR	O	写回寄存器数据的选择信号
DMWr	O	存储器写信号
RFWr	O	寄存器写信号
Alusrc	O	ALU 的操作数 B 的选择信号
EXTOp[1:0]	O	EXT 的功能选择
ALUOp[4:0]	O	ALU 的功能选择

### 3.8 MIPS（模型机）

#### (1) 功能描述

MIPS 模型机将每一个单元连接为整体，并输入相应的时钟控制信号与复位信号，使得每一个单元构成一个完整的流水线数据通路。

#### (2) 模块接口

信号名	方向	描述
clk	I	时钟
clk1	I	时钟 1 用于调试
rst	I	复位信号

---

### 3.9 流水线寄存器 (IFIDreg)

#### (1) 功能描述

将 IF 级的信号与数据保存，在下一时钟周期传入 ID 级。

#### (2) 模块接口

信号名	方向	描述
clk	I	时钟
PCin[31:0]	I	IF 级 PC 输入
rst	I	复位信号
PCout[31:0]	O	PC 输出到 ID 级
instrin[31:0]	I	IF 级指令
instrout[31:0]	O	指令输出到 ID 级
IFIDWr	I	IFID 写信号
IFIDzero	I	IFID 阻塞信号
PCWr	I	BEQ 跳转清零信号

### 3.10 流水线寄存器 (IDEXreg)

#### (1) 功能描述

将 ID 级的信号与数据保存，在下一时钟周期传入 EX 级。

#### (2) 模块接口

信号名	方向	描述（以下描述均为从 ID 到 EX 级的信号）
clk	I	时钟
IDEXzero	I	IDEX 级阻塞信号
rst	I	复位信号
Branchout	O	分支信号输出

Branchin	I	分支信号输入
RegDstout	O	寄存器地址选择信号输入
RegDstin	I	寄存器地址选择信号输出
PCWr	I	BEQ 跳转清零信号
DMemRin	I	存储器读信号输入
DMemRout	O	存储器读信号输出
DMnoutregRin	I	写回数据选择信号输入
DMnoutregRout	O	写回数据选择信号输出
DMWrin	I	存储器写信号输入
DMWrout	O	存储器写信号输出
RFWrout	O	寄存器写信号输出
RFWrin	I	寄存器写信号输入
Alusrcin	I	ALU 操作数 B 选择信号输入
Alusrcout	O	ALU 操作数 B 选择信号输出
EXTOpin[1:0]	I	扩展单元控制信号输入
EXTOpout[1:0]	O	扩展单元控制信号输出
ALUOpin[4:0]	I	ALU 控制信号输入
ALUOpout[4:0]	O	ALU 控制信号输出
RFout1in[31:0]	I	寄存器读出数据 1 输入
RFout2in[31:0]	I	寄存器读出数据 2 输入
RFout1out[31:0]	O	寄存器读出数据 1 输出
RFout2out[31:0]	O	寄存器读出数据 2 输出
Imm32in[31:0]	I	扩展单元结果输入
Imm32out[31:0]	O	扩展单元结果输出
rtin[4:0]	I	指令段 rt 输入
rtout[4:0]	O	指令段 rt 输入
rdin[4:0]	I	指令段 rd 输入

rdout[4:0]	O	指令段 rd 输入
rsin[4:0]	I	指令段 rs 输入
rsout[4:0]	O	指令段 rs 输出
PCin[31:0]	I	指令 PC 输入
PCout[31:0]	O	指令 PC 输出

### 3. 11 流水线寄存器 (EXMEMreg)

#### (1) 功能描述

将 EX 级的信号与数据保存，在下一时钟周期传入 MEM 级。

#### (2) 模块接口

信号名	方向	描述（以下描述均为从 EX 到 MEM 级的信号）
clk	I	时钟
rst	I	复位信号
Branchout	O	分支信号输出
Branchin	I	分支信号输入
DMemRin	I	存储器读信号输入
DMemRout	O	存储器读信号输出
DMnoutregRin	I	写回数据选择信号输入
DMnoutregRout	O	写回数据选择信号输出
DMWrin	I	存储器写信号输入
DMWrout	O	存储器写信号输出
RFWrout	O	寄存器写信号输出
RFWrin	I	寄存器写信号输入
blockin	I	阻塞信号输入

blockout	O	阻塞信号输出
RFout2in[31:0]	I	寄存器读出数据 2 输入
RFout2out[31:0]	O	寄存器读出数据 2 输出
bbbin[31:0]	I	BEQ 跳转地址输入
bbbout[31:0]	O	BEQ 跳转地址输出
wbaddrin[4:0]	I	写回地址输入
wbaddroutt[4:0]	O	写回地址输出输入
rdin[4:0]	I	指令段 rd 输入
rdout[4:0]	O	指令段 rd 输入
zeroin	I	零标识输入
zeroout	O	零标识输出
aluCin[31:0]	I	ALU 结果 C 输入
aluCout[31:0]	O	ALU 结果 C 输出

### 3. 12 流水线寄存器 (MEMWBreg)

#### (1) 功能描述

将 MEM 级的信号与数据保存，在下一时钟周期传入 WB 级。

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
clk	I	时钟
rst	I	复位信号
DMnoutregRin	I	写回数据选择信号输入
DMnoutregRout	O	写回数据选择信号输出
RFWrout	O	寄存器写信号输出

RFWrin	I	寄存器写信号输入
blockin	I	阻塞信号输入
blockout	O	阻塞信号输出
dmoutin[31:0]	I	存储器读出数据输入
dmoutout[31:0]	O	存储器读出数据输出
wbaddrin[4:0]	I	写回地址输入
wbaddroutt[4:0]	O	写回地址输出输入
aluCin[31:0]	I	ALU 结果 C 输入
aluCout[31:0]	O	ALU 结果 C 输出

### 3. 13 数据冒险检测单元 (MUX1)

#### (1) 功能描述

检测是否 EX 级的 RT,RS 与 EX/MEM 级的 RD 相等,若相等则会出现数据冒险,将对应的出现数据冒险的信号输出,使其将所需数据旁路给 ALU。

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
Rs[4:0]	I	EX 级的 RS
Rt[4:0]	I	EX 级的 RT
RFWr	I	寄存器写信号
numA[1:0]	O	判断冒险信号 A
numB[1:0]	O	判断冒险信号 B
exmemrd[4:0]	I	EXMEM 级的 RD
clk	I	时钟

---

### 3.14 数据冒险检测单元 (MUX2)

#### (1) 功能描述

检测是否 EX 级的 RT,RS 与 MEM/WB 级的 RD 相等，若相等则会出现数据冒险，将对应的出现数据冒险的信号输出，使其将所需数据旁路给 ALU。

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
Rs[4:0]	I	EX 级的 RS
Rt[4:0]	I	EX 级的 RT
RFWr	I	寄存器写信号
numA1[1:0]	O	判断冒险信号 A1
numB1[1:0]	O	判断冒险信号 B1
memwbrd[4:0]	I	MEMWB 级的 RD
clk	I	时钟

### 3.15 PC 选择单元 (PCjudge)

#### (1) 功能描述

根据输入的信号与指令以及数据，输出真正在 PC 单元输入的值

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
PCreal	O	最终在 PC 单元输入的 NPC
PCWr	I	BEQ 分支信号
bbb[31:0]	I	BEQ 跳转地址
Imm32[31:0]	I	扩展单元数据

---

instr[31:0]	I	指令
jump	I	无条件跳转信号

### 3. 16 ALU 结果选择单元 (ALU judge)

#### (1) 功能描述

根据输入的数据冒险判断信号，选择相应的 ALU 操作数 A 与操作数 B。

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
aluBin[31:0]	I	操作数 B 输入
aluBout[31:0]	O	操作数 B 输出
aluAin[31:0]	I	操作数 A 输入
aluAout[31:0]	O	操作数 A 输出
numA[1:0]	I	冒险判断信号
numB[1:0]	I	冒险判断信号
numA1[1:0]	I	冒险判断信号
numB1[1:0]	I	冒险判断信号
aluex[31:0]	I	EX/MEM 级旁路到 ALU 的数据
aluwb[31:0]	I	MEM/WB 级旁路到 ALU 的数据
ALUsrc	I	(调试用的信号)

### 3. 17 LW 冒险检测单元 (Check)

#### (1) 功能描述

根据输入的数据与信号检测是否发生 LW 数据冒险，并输出相应的阻塞信号

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
rt[4:0]	I	EX 级的 RT
rdx[4:0]	I	MEM 级的 RD
block	O	阻塞信号
DMemR	I	存储器写信号
rst	I	复位
clk	I	时钟

### 3. 18 LW 冒险阻塞信号控制单元 (HD)

#### (1) 功能描述

根据输入的数据与阻塞信号，输出相应单元的控制信号，IF/ID, ID/EX, PC 单元是否阻塞。

#### (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
PCrun	O	PC 控制信号
IDEXzero	O	ID/EX 控制信号
IFIDzero	O	IF/ID 控制信号
block	I	阻塞信号
rst	I	复位
clk	I	时钟

### 3. 19 ALU 操作数 B 的选择单元 (MUX3)

#### (1) 功能描述

根据输入的数据与阻塞信号，选择操作数 B 的值为不变还是将 MEM 级数据旁路

---

给 ALU 的操作数 B。

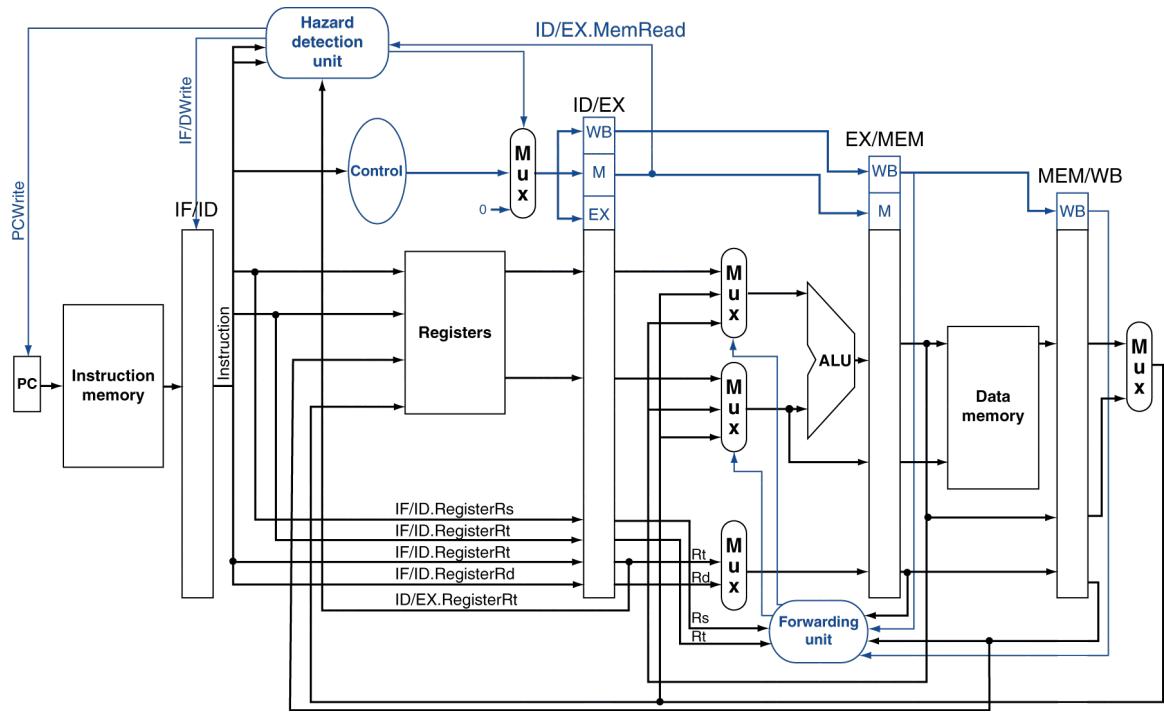
## (2) 模块接口

信号名	方向	描述(以下描述均为从 MEM 到 WB 级的信号)
DMout[31:0]	I	存储器读出数据
alu_Bout[31:0]	O	ALUB 数据输出
alu_Bin[31:0]	I	ALUB 数据输入
block	I	阻塞信号

## 4. 详细设计

### 4.1 CPU 总体结构

CPU 流水线数据通路总体结构如下图所示，其中包括程序计数器 (PC)、指令存储器 (im)、寄存器组 (RF)、运算器 (ALU)、数据扩展单元 (EXT)、数据存储器 (DM) 和控制器 (Ctrl)、流水线寄存器 (IFIDreg, IDEXreg, EXMEMreg, MEMWBreg)、数据冒险检测单元 (MUX1, MUX2)、PC 选择单元 (PCjudge)、ALU 结果选择单元 (ALUjudge)、LW 冒险检测单元 (Check)、LW 冒险阻塞信号控制单元 (HD)、ALU 的 B 输入选择 (MUX3)。



下面分别介绍各子模块的设计。

---

## 4.2 PC (程序计数器)

```
1  `include "global_def.v"
2  module PC( clk, rst, PCWr, NPC, PC,PCjump,PCrun,block );
3
4      input          clk,block;
5      input          rst;
6      input          PCWr;
7      input [31:0] NPC;
8      output reg[31:0] PC;
9      input          PCjump,PCrun;
10     integer i;
11     reg [31:0] tmp;
12     reg [31:0] a;
13
14
15     always @ (posedge clk or posedge rst)
16     begin
17         if (PCrun==1)begin
18             if (rst == 1 )
19                 PC <= 32'h0000_3000;
20
21             PC=PC+4;
22
23             if ( PCWr == 1 )
24                 PC=NPC;
25             if (PCjump == 1&& PCWr==0)
26                 begin
27                     PC= {PC[31:28],NPC[27:0]}
28                     `ifdef DEBUG
29                     $display("PC[00-31]=%8X", PC );
30
31                 `endif
32             end
33
34         end
35     end
36 endmodule
```

基本功能为指令计数器，每个时钟周期自动进行 PC+4 计算。当 PCrun 为 1 时，PC 可写，于是根据输入的控制信号不同 PC 单元最终输出的值 PC 会改变，当 PCWr=1 时说明 BEQ 指令发生跳转，则需要将 PC 的值置为 NPC (即为 BEQ 指令低 16 位经扩展单元扩展为 32 位再左移两位与 PC+4 相加的结果)。当 PCWr=0 并且 PCjump=1 说明当前为无条件转移指令，则直接将 PC 置为 JUMP 指令地址 (即低 26 位左移两位与 PC+4 的高四位相加的结果)。当 PCWr 与 PCjump 均为 0 时，正常进行计数 PC=PC+4。当 PCrun 为 0 时，此时 PC 单元的值不可改变与上一周期的值保持一致。

---

### 4.3 RF（寄存器）设计

```
1  `include "global_def.v"
2  module RF( A1, A2, A3, WD, clk, RFWr, RD1, RD2 );
3
4      input [4:0] A1, A2, A3;
5      input [31:0] WD;
6      input        clk;
7      input        RFWr;
8
9      output [31:0] RD1, RD2;
10
11     reg [31:0] rf[31:0];
12
13    integer i;
14    initial begin
15        for (i=0; i<32; i=i+1)
16            rf[i] = 0;
17    end
18
19    always @ (posedge clk) begin
20        if (RFWr)
21            rf[A1] <= WD;
22        `ifdef DEBUG
23            $display("R[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X", 0, rf[1],
24            $display("R[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[8], rf[15],
25            $display("R[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[16],
26            $display("R[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[24],
27            $display("R[%4X]=%8X", A3, rf[A3]);
28        `endif
29    end // end always
30
31    assign RD1 = (A2 == 0) ? 32'd0 : rf[A2];
32    assign RD2 = (A3 == 0) ? 32'd0 : rf[A3];
33
34 endmodule
35
```

基本功能为根据 A2,A3 的输入值（寄存器号）读出寄存器组的数据保存到 RD1,RD2 寄存器中输出。首先将寄存器组的初始值都置为 0（该语句只执行一次）。RFWr=1 时寄存器可写，此时根据 A1 的值（寄存器号）与 WD（寄存器写回的数据）将数据写到相应寄存器中。

---

## 4.4 ALU (算术逻辑运算单元)

```
1  `include "ctrl_encode_def.v"
2  module alu (A, B, ALUOp, C, Zero);
3
4      input [31:0] A;
5      input [31:0] B;
6      input [4:0] ALUOp;
7      output reg[31:0] C;
8      output reg Zero;
9
10     initial
11     begin
12         Zero = 0;
13         C = 0;
14     end
15
16     always @ ( A or B or ALUOp )
17     begin
18         case ( ALUOp )
19             `ALUOp_ADDU: C = A + B;
20             `ALUOp_SUBU: C = A - B;
21             `ALUOp_OR:   C=A|B;
22             default:    ;
23         endcase
24         assign Zero = (C==0) ? 1 : 0;
25     end //end always;
26 endmodule
```

根据输入的操作数 A,B 的值与 ALU 控制信号进行加减或操作。首先将 ZERO 与 C 的初始值置为 0。根据 ctrl\_encode\_def 中的功能定义，当 ALUOp 为 00001 时进行加操作，为 00011 为减操作，为 00110 时为或操作。当  $A-B=0$  即 A 与 B 相等时，将 ZERO 信号置 1 输出，否则置 0。ZERO 信号是 BEQ 指令是否发生跳转的判断信号之一。

---

## 4.5 EXT (扩展单元)

```
1  `include "ctrl_encode_def.v"
2  module EXT( Imm16, EXTOp, Imm32 );
3
4      input  [15:0] Imm16;
5      input  [1:0]EXTOp;
6      output reg [31:0] Imm32;
7      reg [31:0] a;
8      always @(Imm16 or EXTOp) begin
9          case (EXTOp)
10              `EXT_ZERO :     Imm32 = {16'd0, Imm16};
11              `EXT_SIGNED: // 
12                  begin
13                      Imm32={{16{Imm16[15]}},Imm16};
14                      a={Imm32[29:0],2'd0}//zuoyiliangwei
15                      Imm32=a;
16                  end
17              `EXT_HIGHPOS:   Imm32 = {Imm16, 16'd0};
18              default: ;
19      endcase
20  end // end always
21
22 endmodule
23
```

数据扩展单元。该单元输入数据为指令的低 16 位，根据 ctrl\_encode\_def 中的功能定义，当 EXTOp 为 00，为低十六位扩展，当 EXTOp 为 01，为符号扩展并左移两位，当 EXTOp 为 10，为高十六位扩展。最后将扩展后的 32 位数据输出。

## 4.6 DM (数据存储器)

---

```
1  `include "global_def.v"
2  module dm( addr, din, DMWr, clk, dout, DMemR );
3
4      input  [4:0] addr;
5      input  [31:0] din;
6      input          DMWr;
7      input          DMemR;
8      input          clk;
9
10     output [31:0] dout;
11
12     reg [31:0] dmem[1023:0];
13
14     always @ (posedge clk)
15     begin
16         if (DMWr)
17             dmem[addr] <= din;
18         `ifdef DEBUG
19             $display("DEM[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X")
20             $display("DEM[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X")
21             $display("DEM[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X")
22             $display("DEM[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X")
23             $display("DEM[%4X]=%8X", addr, dmem[addr]);
24         `endif
25     end // end always
26
27     assign dout = dmem[addr];
28
29 endmodule
```

该单元功能为存储器读写数据。当存储器写信号 DMWr 为 1 时，根据输入的 addr（存储器号）将写入数据 din 写到相应的存储器号中，否则不写入存储器。当存储器读信号为 1 时，根据存储器号将相应单元中的数据读到 dout 中输出。

---

## 4.7 IM (指令存储器)

```
1 module im( addr, dout);
2
3     input [4:0] addr;
4     output [31:0] dout;
5
6     reg[31:0] Dout;
7
8     reg [31:0] imem[1024:0];
9
10    always@(addr)
11        begin
12            Dout=imem[addr];
13        end
14        assign dout = Dout;
15
16    endmodule
```

根据输入的指令地址 addr, 将指令存储器中相应指令读取出到寄存器 dout 中输出。

## 4.8 Ctrl (控制器)

```
1 `include "ctrl_encode_def.v"
2
3 module Ctrl(jump,RegDst,Branch,DMemR,DMnoutregR,DMWr,RFWr,Alusrc,EXTOp,ALUOp
4
5     input [5:0] instr;
6     input [5:0] funct;
7
8     output reg jump;
9     output reg RegDst;
10    output reg Branch;
11    output reg DMemR;
12    output reg DMnoutregR;
13    output reg DMWr;
14    output reg RFWr;
15    output reg Alusrc;
16    output reg[1:0] EXTOp;
17    output reg[4:0] ALUOp;
```

---

```

always @(instr or funct)
begin
    case(instr)
    //+
        6'b000000:
        begin
            assign jump=0;
            assign RegDst=0;
            assign Branch=0;
            assign DMnoutregR=0;
            assign DMemR=1'b0;
            assign DMWr=1'b0;
            assign RFWr=1'b1;
            assign Alusrc=0;
            assign EXTOp='EXT_ZERO;
            case(funct)
                6'b100001:
                    assign ALUOp='ALUOp_ADDU;
                6'b100011:
                    assign ALUOp='ALUOp_SUBU;
                default:;
            endcase
        end

```

根据输入指令的高六位 instr，CASE 语句判断每一条指令的功能，并设置相应功能的单元控制信号。

(1) 当 instr 为 000000 时，此时为算术运算指令，只有寄存器写信号 RFWr 为 1，其它控制信号全为 0。根据 FUNCT 字段，当其为 100001 时，执行加法操作，当其为 100011 时，执行减法操作。

```

6'b101011://sw
begin
    assign jump=0;
    assign RegDst=1;
    assign Branch=0;
    assign DMnoutregR=1'b0;
    assign DMemR=1'b0;
    assign DMWr=1'b1;
    assign RFWr=1'b0;
    assign Alusrc=1;
    assign EXTOp='EXT_ZERO;
    assign ALUOp='ALUOp_ADDU;
end

```

(2) 当 instr 为 101011 时，为存储器写指令此时存储器写信号 DMWr 为 1，寄存器写回地址选择信号 RegDst 为 1(即写回地址为 RT)，ALU 操作数 B 选择信号

---

Alusrc 为 1 (即选择扩展单元扩展的数据), ALUOp 为加法, EXT 为低 16 位扩展。其它信号全为 0。

```
6'b100011://lw
begin
    assign jump=0;
    assign RegDst=1;
    assign Branch=0;
    assign DMnoutregR=1'b1;
    assign DMemR=1'b1;
    assign DMWr=1'b0;
    assign RFWr=1'b1;
    assign Alusrc=1;
    assign EXTOp='EXT_ZERO;
    assign ALUOp='ALUOp_ADDU;
end
```

(3) 当 instr 为 100011 时, 为存储器读指令。寄存器写回地址信号 RegDst 为 1 (选择 RT 字段), 写回寄存器数据的选择信号 DMnoutregR 为 0 (选择存储器读出的数据作为写回数据), 存储器读信号 DMemR 为 1, 寄存器写信号 RFWr 为 1, ALU 操作数 B 的选择信号 Alusrc 为 1 (即选择扩展单元扩展的数据), ALU 为加法运算, EXT 为低 16 位扩展, 其它信号为 0。

```
6'b001101: //ori
begin
    assign jump=0;
    assign RegDst=1;
    assign Branch=0;
    assign DMnoutregR=0;
    assign DMemR=1'b0;
    assign DMWr=1'b0;
    assign RFWr=1'b1;
    assign Alusrc=1;
    assign EXTOp='EXT_ZERO;
    assign ALUOp='ALUOp_ADDU;
end
```

---

(4) 当 instr 为 001101 时, 为或运算, 此时 ALU 为或运算, EXT 为低 16 位扩展, 寄存器写 RFWr 为 1, ALU 操作数 B 选择信号 Alusrc 为 1 (即选择扩展单元扩展的数据), 寄存器写回地址 RegDst 信号为 1 (即选择 RT 字段), 其它信号皆为 0。

```
6'b000100: //beq
begin
    assign jump=0;
    assign RegDst=0;
    assign Branch=1;
    assign DMnoutregR=0;
    assign DMemR=1'b0;
    assign DMWr=1'b0;
    assign RFWr=1'b1;
    assign Alusrc=0;
    assign EXTOp=`EXT_SIGNED;
    assign ALUOp=`ALUOp_SUBU;
end
```

(5) 当 instr 为 000100 时, 为 BEQ 分支指令, 分支信号 Branch 为 1, 寄存器写回信号 RFWr 为 1, EXT 执行符号扩展并左移 2 位操作, ALU 进行减法运算, 其它信号皆为 0。

```
6'b000010://JUMP
begin
    assign jump=1;
    assign RegDst=1;
    assign Branch=0;
    assign DMnoutregR=1'b0;
    assign DMemR=1'b0;
    assign DMWr=1'b1;
    assign RFWr=1'b0;
    assign Alusrc=1;
    assign EXTOp=`EXT_ZERO;
    assign ALUOp=`ALUOp_ADDU;
end
default::;

endcase
end
endmodule
```

(6) 当 instr 为 000010 时, 为 JUMP 无条件跳转指令, 此时跳转信号 jump 为 1, 由于 jump 为无条件跳转, 此时其它信号为无关信号, 在流水线中会被刷新掉, 所以我在这里没有更改它们的值。

---

## 4.9 MIPS（模型机）

```
-----  
module mips( clk, rst ,clk1);  
    input    clk;  
    input    rst;  
    input    clk1;  
  
    //RF  
    wire [4:0] rs,rt,rd;  
    wire [4:0] rs1;  
    wire [4:0] rd1;  
    wire [4:0] rdx;  
    wire [4:0] rd2;  
    wire [4:0] rd3;  
    wire [4:0] rt1;  
    wire [31:0] writebackdata;  
    wire [31:0] RD1,RD2;  
    wire [31:0] RD11;  
    wire [31:0] RD22;wire [31:0] RD222;  
  
    //EXT  
    wire [15:0] extImm16;  
    wire [31:0] extImm32;  
    wire [31:0] extImm321;  
  
    //alu  
    wire [31:0] alu_B;//in2  
    wire [31:0] alu_C;//out  
    wire [31:0] alu_C2;wire [31:0] alu_C3;  
    wire zero;  
  
    //IM  
    wire [4:0] imAddr;  
    wire [31:0] instr;  
    wire [31:0] instr1;  
  
    //PC OUTPUT  
    wire [31:0] PC;  
    wire [31:0] PC1;  
    wire [31:0] PC2;  
  
    //DM  
    wire [31:0] DMdout;wire [31:0] DMdout2;  
    wire [4:0] dm_addr;  
  
    //ctrl  
    wire        RFWr;  
    wire RFWr1;wire RFWr2;wire RFWr3;  
    wire        DMemR;  
    wire DMemR1;wire DMemR2;  
    wire        DMWr;  
    wire DMWr1;wire DMWr2;  
    wire        DMnoutregR;wire DMnoutregR2;wire DMnoutregR3;  
    wire DMnoutregR1;  
    wire        PCWr;
```

---

```

//wire      IRWr;
wire  [1:0]    EXTOp;
wire  [1:0]  EXTOp1;
wire  [4:0]   ALUOp;
wire  [4:0]  ALUOp1;
wire      Zero;wire Zero2;
wire      Alusrc;
wire Alusrc1;
wire      jump;
wire      Branch;
wire Branch1;  wire Branch2;
wire      RegDst;
wire RegDst1;
wire  [5:0]  Op;
wire  [5:0]  Funct;
wire  [31:0] aaa,bbb;
wire  [31:0] bbb2;
wire      IFIDWr;
wire      IDEXzero,IFIDzero;
wire PCrun,block,block1,block2;//whether PC can change
assign IFIDWr=1'd1;

```

以上为 MIPS 中的全部接线，我会在下面每一个模块的输入输出中介绍这些接线的含义。

### (1) IF 级

```

81      PCjudge U_PCjudge( .PCWr(PCWr),.jump(jump),.PCreal(aaa),.bbb(bbb),.Imm32(extImm321),.instr(instr));
82
83      //PC
84      PC U_PC (
85          .clk(clk), .rst(rst), .PCWr(PCWr), .NPC(aaa), .PC(PC),.PCjump(jump),.PCrun(PCrun),.block(block2)
86      );
87      assign imAddr = PC[6:2];
88
89      //IM
90      im U_im (
91          .addr(imAddr[4:0]), .dout(instr)
92      );
93
94

```

该部分有 PCjudge, PC, im 三个模块。PCjudge 单元主要用于根据输入的控制信号与数据得到真正的 PC 值并输出到寄存器 aaa 中，再传入到 PC 单元的 NPC 中进行选择。

---

PC 单元根据分支跳转信号 PCWr, 无条件跳转信号 jump, PC 单元阻塞信号 PCrun  
输出最终 PC。

然后将 PC 的 3 到 7 位赋值给 imAdr 传入到 IM 单元中的读地址, 将对应的指令读出保存到 instr 中。

## (2) ID 级

```
94
95      // 
96      //IF-ID
97      //
98      IFIDreg U_IFIDreg(.clk(clk1), .rst(rst), .PCin(PC), .PCout(PC1), .PCWr(PCWr), .instrin(instr), .instrout(instr1), .IFIDWr(IFIDWr), .IFIDzero(IFIDzero));
99
100     assign Op = instr1[31:26];
101    assign Funct = instr1[5:0];
102    assign rs = instr1[25:21];
103    assign rt = instr1[20:16];
104    assign extImm16 = instr1[15:0];
105    assign rd=instr1[15:11];
106
107
108
109
110 //RF
111 RF U_RF (
112     .A1(rd3), .A2(rs), .A3(rt), .WD(writebackdata), .clk(clk),
113     .RFWr(RFWr3), .RD1(RD1), .RD2(RD2)
114 );
115
116 //CTRL
117 Ctrl U_Ctrl(.jump(jump), .RegDst(RegDst), .Branch/Branch, .DMemR(DMemR), .DMnoutreqR(DMnoutreqR)
118     , .DMWr(DMWr), .RFWr(RFWr), .Alusrc(Alusrc), .EXTOp(EXTOp), .ALUOp(ALUOp)
119     , .instr(Op), .funct(Funct));
120 //EXT
121 EXT U_EXT(.Imm32(extImm32), .Imm16(extImm16), .EXTOp(EXTOp));
122
123
124
125 //check if block
126 Check U_Check(.rt(rt), .rdx(rdx), .block(block), .DMemR(DMemR1), .clk(clk), .rst(rst));//
127 //HD block
128 HD U_HD(.PCrun(PCrun), .IDEXzero(IDEXzero), .block(block), .clk(clk), .IFIDzero(IFIDzero), .rst(rst));//
```

该部分有 IFIDreg, RF, CTRL, EXT, Check, HD 单元。

IFID 流水线寄存器将 IF 级的数据与信号传递到 ID 级。PC-PC1

---

Instr=instr1。

指令 instr1 分为 Op、Funct、rs、rt、extImm16、rd 几个字段保存。

RF 中寄存器号 rs, rt 传入到 RF 中读取的两个数据保存到 RD1、RD2 中。寄存器写信号为 WB 级的 RFWr3，写回地址为 WB 级的 rd3，写回数据为 writebackdata。

CTRL 根据输入的 OP 与 FUNCT 字段输出相应控制信号。

EXT 根据输入的 EXTOp 将 16 位 extImm16 扩展为 32 位 extImm32。

LW 冒险检测单元根据输入的 rt, rdx (EX 级的 rd), DMemR1 (EX 级的 DMemR)，判断是否需要阻塞并输出阻塞信号 block

LW 冒险阻塞信号控制单元根据输入的阻塞信号输出 PC、ID/EX、IFID 单元的阻塞信号。

### (3) EX 级

```
132 //  
133 //ID-EX  
134 //  
135 IDEXreg U _IDEXreg(.clk(clk1),.rst(rst),.RegDstin(RegDst1),.RegDtout(RegDst1)  
136 ,.Branchin(Branch),.Branchout(Branch1),.IMemRin(IMemR),.IMemRout(IMemR1),  
137 .DMnoutreqRin(DMnoutreqR),.DMnoutreqRout(DMnoutreqR1),.DMWrin(DMWr),  
138 .DMWrout(DMWr1),  
139 .RFWrin(RFWr),.RFWrout(RFWr1),.Alusrcin(Alusrc),.Alusrcout(Alusrc1),  
140 .EXTOpin(EXTOp),.EXTOpout(EXTOp1),.ALUOpin(ALUOp),.ALUOpout(ALUOp1),  
141 .RFout1in(RD1),.RFout1out(RD11),.RFout2in(RD2),.RFout2out(RD22),  
142 .IMm32in(extImm32),.IMm32out(extImm321),.rtin(rt),.rtout(rt1),.rsin(rs),.rsout(rs1),.PCin(PC1),.PCout(PC2),.rdin(rd),.rdout(rd1);  
143 assign rdx = (RegDst1==1)?rt1:rd1;  
144 assign bbb=extImm321+PC2+4;  
145 wire [31:0] alu_Blast;  
146  
147  
148  
149  
150 //MUX three panglu  
151 wire [31:0] alu_B_RD111;  
152 assign alu_B = (Alusrc1==1)?extImm321:RD22;  
153 wire [1:0] numA,numB,numA1,numB1;  
154  
155 MUX1 U_MUX1 (.clk(clk),.rs(rs1),.rt(rt1),.exmemrd(rd2),.RFWr(RFWr2),.numA(numA),.numB(numB));  
156 MUX2 U_MUX2 (.clk(clk),.rs(rs1),.rt(rt1),.memwbrd(rd3),.RFWr(RFWr3),.numA1(numA1),.numB1(numB1));  
157 ALUjudge U_ALUjudge (.aluin(alu_B),.aluout(alu_B1),.aluin(RD11),.aluout(RD111),.numA(numA),.numB(numB),  
158 .numB1(numB1),.numA1(numA1),.aluex(alu_C2),.aluwb(alu_C3),.ALUsrc(Alusrc1));  
159 MUX3 U_MUX3(.block(block2),.alu_Bin(alu_B1),.alu_Bout(alu_Blast),.IMdout(IMdout));  
160  
161  
162 //ALU  
163 alu_U_alu(.C(alu_C),.Zero(Zero),.A(RD111),.B(alu_Blast),.ALUOp(ALUOp1));  
164 //Branch  
165 assign PCWr = ((alu_C==0)&&(Branch1==1))?1:0;  
166
```

① IDEX 寄存器将 ID 级信号与数据传递到 EX 中，这里传到 EX 的信号我直接在名字后面加了 1, PC1-PC2, RD1-RD11, RD2-RD22, extImm32-extImm321。

② () ? 语句用于寄存器写回地址的选择，保存到 rdx 中。

③ bbb=extImm321+PC2+4 为 BEQ 跳转地址的计算，扩展单元的数据与 PC+4 相加。

---

④第 153 行, 为 ALU 操作数 B 的第一步选择, 是选择扩展单元数据还是寄存器读出的数据 RD2.

⑤156, 157 行的 MUX1、2 为数据冒险检测单元, 1 中输入 EX 级的 rs1, rt1, 与 EX/MEM 中的 rd2, RFWr2, 2 中输入 EX 级的 rs1, rt1 与 MEM/WB 级的 rd3, RFWr3。再将相应信号输出 ALUjudge 中使用

⑥ALUjudge 根据输入的冒险判断信号选择出相应的 ALU 操作数 A (RD111) 与 ALU 操作数 B(alu\_B1)。其中旁路过来的数据为 EX/MEM 的 alu\_C2 与 MEM/WB 的 alu\_C3。

⑦MUX3 用于 LW 型数据冒险的 ALU 操作数 B 选择, 根据输入的阻塞信号 block2 (WB 级的阻塞信号), 判断是否发生阻塞, 输出最终确定的 alu\_Blast, 若发生阻塞则将 MEM 级的存储器读出数据 DMdout 旁路给 ALU。

⑧ALU 的两个操作数为 RD111, 与 alu\_Blast, 输出结果 alu\_C

⑨166 行该语句用于判断 BEQ 分支是否发生。原本该语句应在 MEM 级判断, 但是由于在 MEM 级判断若发生分支则抛弃前面执行的语句, 需要将前三个流水线寄存器的数据全部清 0 代价较大。其次由于其上一条指令为 LW 指令, 若提前到 ID 级判断则需要将上面语句的冒险检测数据全部修改, 同样代价较大。因此我将它提前到 EX 级判断分支是否发生。

#### (4) MEM 级

```
170 //EX-MEM
171 //MEM-R2
172 //DM
173 assign dm_addr = alu_C2[4:0];
174 dm.U_dm(.dout(DMdout), .addr(dm_addr), .din(RD222), .DMWr(DMWr2), .DMemR(DMemR2), .clk(clk));
175
176
177
178
179
180
181
182
183
184
185
```

①EXMEM 寄存器将 EX 中 数据与信号传递到 MEM 级, 我将控制信号后面的 1 变为 2, block-block1, RD22-RD222, rdx-rd2, bbb-bbb2, Zero-Zero2, alu\_C-alu\_C2。

②存储器写数据地址 dm\_addr=alu\_C2[4:0]。

③DM 为存储器, 写入数据为 RD222, 写入地址为 dm\_addr, 读写控制信号为 DMWr2, DMemR2, 读出数据 DMdout。

#### (5) WB 级

```
186 //MEM-WB
187 //WB
188 //DM
189 assign writebackdata = (DMnoutregR3==1)?DMdout2:alu_C3;
190
191
192
193
194
195
196
197
```

---

①MEMWB 寄存器将 MEM 级信号与数据传递到 WB 级,  
DMdout-DMdout2, alu\_C2-alu\_C3, block1-block2, rd2-rd3, DMnoutregR2-DMnout  
regR3。

②196 行判断选择语句根据选择信号 DMnoutregR3 用于选择最终写回的数据  
writebackdata 是 ALU 计算结果，还是存储器读出数据。

#### 4. 10 流水线寄存器 (IFIDreg)

```
module IFIDreg(clk, rst, PCin, PCout, instrin, instrout, IFIDWr, IFIDzero, PCWr);

    input      clk;
    input      rst;
    input      IFIDWr, IFIDzero, PCWr;
    input [31:0] PCin;
    input [31:0] instrin;

    output [31:0] PCout;
    output [31:0] instrout;

    reg [31:0] PCout;
    reg [31:0] instrout;

    always @(posedge clk or posedge rst) begin

        if(IFIDzero==1'b1)
            begin
                if (rst )
                    instrout <= 0;
                else if (IFIDWr)
                    begin
                        PCout=PCin;
                        instrout=instrin;
                        if(PCWr==1)
                            begin
                                PCout=0;
                                instrout=0;
                            end
                        end
                    end
            end
    end
endmodule
```

该寄存器用于存储 IF 级的 PC(PCin, PCout)，指令

---

instr(instrin, instrout)。

当 IFID 寄存器阻塞信号 IFIDzero 为 0 时，此时 IFID 寄存器的值不可修改  
保持上一周期的数据不变。

当 IFID 寄存器阻塞信号 IFIDzero 为 1 并且 BEQ 分支指令清 0 信号 PCWr 为  
1 时，说明发生了分支跳转，此时应将 IFID 寄存器中的所有数据清 0。

当 IFID 寄存器阻塞信号 IFIDzero 为 1、IFID 寄存器写入信号 IFIDWr 为 1、  
PCWr 为 0 时，进行正常的数据传递。

再传输到 ID 级。

## 4.11 流水线寄存器 (IDEXreg)

```
1 module IDEXreg(clk, rst,RegDstin,RegDtout,Branchin,Branchout,DMemRin,DMemRout,DMnoutregRin,DMnoutregRout,
2                 DMWrin,DMWrout,RFWrin,RFWrout,Alusrcin,Alusrcout,
3                 EXTOpin,EXTOpout,ALUOpin,ALUOpout,RFoutlin,RFout2in,RFoutlout,RFout2out,
4                 Imm32in,Imm32out,rtin,rtout,rsin,rsout,rdin,rdout,PCin,PCout,IDEZero,PCWr);
5
6     input      clk,IDEZero,PCWr;
7     input      rst;
8     // input      IDEXWr;
9     input      RegDstin;
10    output reg   RegDtout;
11    input      Branchin;
12    output reg   Branchout;
13    input      DMemRin;
14    output reg   DMemRout;
15    input      DMnoutregRin;
16    output reg   DMnoutregRout;
17    input      DMWrin;
18    output reg   DMWrout;
19    input      RFWrin;
20    output reg   RFWrout;
21    input      Alusrcin;
22    output reg   Alusrcout;
23    input [1:0]  EXTOpin;
24    output reg [1:0]  EXTOpout;
25    input [4:0]  ALUOpin;
26    output reg [4:0]  ALUOpout;
27    input[31:0]  RFoutlin,RFout2in;
28    output reg [31:0]  RFoutlout,RFout2out;
29    input[31:0]  Imm32in;
30    output reg [31:0]  Imm32out;
31    input [4:0]  rtin,rdin,rsin;
32    output reg [4:0]  rtout,rdout,rsout;
33    input [31:0]  PCin;
34    output reg [31:0]  PCout;
35
36
```

```

Ln# | 
35      output reg [31:0] PCout;
36
37      always @ (posedge clk or posedge rst) begin
38          if(IDEXzero==1)
39              begin
40                  RegDtout=RegDstin;
41                  Branchout=Branchin;
42                  DMemRout=DMemRin;
43                  DMnoutregRout=DMnoutregRin;
44                  DMWrout=DMWrin;
45                  RFWrout=RFWrin;
46                  Alusrcout=Alusrcin;
47                  EXTOpout=EXTOpin;
48                  ALUOpout=ALUOpin;
49                  RFout2out=RFout2in;
50                  RFout1out=RFout1in;
51                  Imm32out=Imm32in;
52                  rsout=rstin;
53                  rtout=rtin;
54                  rdout=rdin;
55                  PCout=PCin;
56          end
57
58          if(IDEXzero==0) begin
59              RegDtout=0;
60              DMemRout=0;
61              DMnoutregRout=0;
62              DMWrout=0;
63              RFWrout=0;
64              Alusrcout=0;
65              RFout2out=0;
66              RFout1out=0;
67      end
68
69
70
71      if(PCWr==1) begin
72          RegDtout=0;
73          DMemRout=0;
74          DMnoutregRout=0;
75          DMWrout=0;
76          RFWrout=0;
77          Alusrcout=0;
78          EXTOpout=0;
79          ALUOpout=0;
80          RFout2out=0;
81          RFout1out=0;
82          Imm32out=0;
83          rsout=0;
84          rtout=0;
85          rdout=0;
86          PCout=0;
87      end
88
89
90
91
92      end // end always
93  endmodule
94

```

该寄存器输入的数据为来自 CTRL 单元的控制信号与扩展单元扩展数据，指令的三个字段 RS、RT、RD，寄存器读出的两个 RD1, RD2 数据，以及 PC。三个 IF 语句用于判断 LW 信号阻塞与分支信号清零。

当 PCWr 为 1 时，说明发生分支跳转，则须将 IDEX 级的控制信号清 0。

---

当 IFIDzero 为 0 时,说明此时发生 LW 数据冒险,需要将 IDEX 寄存器阻塞。

当 PCWr 为 0、IFIDzero 为 1 时, 该寄存器进行正常的数据传输。

再传输到 EX 级。

## 4.12 流水线寄存器 (EXMEMreg)

```
1  module EXMEMreg(clk, rst
2           ,Branchin,Branchout,DMemRin,DMemRout,
3           ,DMnoutregRin,DMnoutregRout,DMWrin,DMWrout,
4           ,RFWrin,RFWrout,bbbIn,bbbOut,zeroIn,zeroOut,RFout2in,RFout2out,
5           ,wbaddrin,wbaddrout,aluCin,aluCout,blockin,blockout);
6   input      clk,blockin;
7   input      rst;
8 // input      IDEXWr;
9   input      Branchin;
10  output reg      Branchout;
11  input      DMemRin;
12  output reg      DMemRout;
13  input      DMnoutregRin;
14  output reg      DMnoutregRout;
15  input      DMWrin;
16  output reg      DMWrout,blockout;
17  input      RFWrin;
18  output reg      RFWrout;
19  input[31:0]    RFout2in;
20  output reg [31:0]  RFout2out;
21  input [4:0] wbaddrin;
22  output reg [4:0] wbaddrout;
23  input[31:0]    bbbIn;
24  output reg [31:0]  bbbOut;
25  input[31:0]    aluCin;
26  output reg [31:0]  aluCout;
27  input      zeroIn;
28  output reg      zeroOut;
29 always @(posedge clk or posedge rst) begin
30   Branchout=Branchin;
31   DMemRout=DMemRin;
32   DMnoutregRout=DMnoutregRin;
33   DMWrout=DMWrin;
34   RFWrout=RFWrin;
35   RFout2out=RFout2in;
36   wbaddrout=wbaddrin;

37   blockout=blockin;
38   bbbout=bbbIn;
39   zeroout=zeroIn;
40   alucout=aluCin;
41 end // end always
42 endmodule
43
44
```

该寄存器用于保存来自 EX 级的分支信号 Branchin、存储器读信号 DMemRin、写回数据的选择信号 DMnoutregRin、存储器写信号 DMWrin、寄存器写信号 RFWrin、寄存器输出数据 2RFout2in、寄存器写回数据的地址 wbaddrout、阻塞信号 blockin、ALU 零标识 zeroin、ALU 计算输出结果 C aluCin、BEQ 跳转地址 bbbIn。再传输到 MEM 级。

## 4.13 流水线寄存器 (MEMWBreg)

```
Ln# 1 module MEMWBreg(clk,blockin,blockout,rst,DMnoutregRin,DMnoutregRout,aluCin,aluCout,dmoutin,dmoutout,wbaddrin,wbaddrout,RFWrin,RFWrout);
2
3     input      clk,blockin;
4     input      rst;
5     input      DMnoutregRin,RFWrin;
6     output reg   DMnoutregRout,RFWrout,blockout;
7
8     input [4:0] wbaddrin;
9     output reg [4:0] wbaddrout;
10    input [31:0] aluCin;           /mips_tb/U_mips/U_MEMWBreg/wbaddrout
11    output reg [31:0] aluCout;
12
13
14    input [31:0] dmoutin;
15    output reg [31:0] dmoutout;
16    always @(posedge clk or posedge rst) begin
17
18        DMnoutregRout=DMnoutregRin;
19        aluCout=aluCin;
20        dmoutout=dmoutin;
21        wbaddrout=wbaddrin;
22        RFWrout=RFWrin;
23        blockout=blockin;
24    end // end always
25    //????? ??0 ???1 ???????4? ??????????????????????1????? ?????
26    //MIPS??
27 endmodule
```

该寄存器用于保存来自 MEM 级的信号：寄存器写信号 RFWrin、阻塞信号 blockin、写回寄存器的数据选择信号 DMnoutregRin、ALU 计算结果 C aluCin、存储器读出的数据 dmoutin、寄存器写回数据的地址 wbaddrin。再传输到下一级。

## 4.14 数据冒险检测单元 (MUX1)

```
1  |
2  module MUX1( rs,rt,exmemrd,RFWr,numA,numB,clk);
3  input [4:0] rs,rt,exmemrd;
4  input RFWr,clk;
5  output reg [11:0] numA,numB;
6  always @(*)
7  begin
8      assign numA=00;
9      assign numB=00;
10     if(exmemrd!=0000&&RFWr&&exmemrd==rs)
11     begin
12         assign numA=10;
13     end
14     if(exmemrd!=0000&&RFWr&&exmemrd==rt)begin
15         assign numB=10;
16     end
17
18 end
19 endmodule
20
```

检测 EX 级数据冒险，当 EX 级的 RS 或 RT 与 MEM 级的 RD 相等并且寄存器写信号为 1 时，修改冒险判断信号 numA, numB 为相应的值，并输出传给 ALU 的操作数 A、B 选择单元。

---

## 4.15 数据冒险检测单元 (MUX2)

```
1 module MUX2( rs,rt,memwbrd,RFWr,numA1,numB1,clk);
2   input [4:0] rs,rt,memwbrd;
3   input RFWr,clk;
4   output reg [1:0] numA1,numB1;
5
6   always @(*)
7   begin
8     assign numA1=00;
9     assign numB1=00;
10    if(memwbrd!=0000&&RFWr&&memwbrd==rs) begin
11      assign numA1=01;
12    end
13    if(memwbrd!=0000&&RFWr&&memwbrd==rt) assign numB1=01;
14
15  end
16 endmodule
17
18
```

检测 EX 级数据冒险，当 EX 级的 RS 或 RT 与 WB 级的 RD 相等并且寄存器写信号为 1 时，修改冒险判断信号 numA1, numB1 为相应的值，并输出传给 ALU 的操作数 A、B 选择单元。

## 4.16 PC 选择单元 (PCjudge)

```
1 module PCjudge( PCWr,jump,PCreal,bbb,Imm32,instr);
2
3   input PCWr;
4   input jump;
5   output reg [31:0]PCreal;
6   input [31:0] bbb;//beq
7   input [31:0] Imm32;
8   input [31:0] instr;
9
10  always@ ( PCWr or jump)
11  begin
12    if(PCWr)assign PCreal=bbb;
13    if(jump==1&&PCWr==0)assign PCreal=instr;
14    if(jump==0&&PCWr==0)assign PCreal=Imm32;
15  end
16
17 endmodule
18
```

输入的数据为 BEQ 跳转地址 bbb, 扩展单元扩展的数据 Imm32, 指令 instr, 无条件转移信号 jump, 分支指令信号 PCWr, 最终得到的 PC 的输出 PCreal。

当 PCWr 为 1 时，说明发生分支跳转，将 BEQ 跳转地址赋值给 PCreal。

当 jump 为 1, PCWr 为 0, PCreal 为当前指令。（这两个条件我是用于理清思路用的，实际上赋值没有起到作用）

当 jump 为 0, PCWr 为 0, PCreal 扩展的 32 位数据。（这两个条件我是用于理清思路用的，实际上赋值没有起到作用）

## 4.17 ALU 结果选择 (ALUjudge)

---

```

2
3   module ALUjudge( aluBin,aluBout,aluAin,aluAout,numA,numB,numA1,numB1,aluex,aluwb,ALUsrc);
4     input [31:0] aluBin,aluAin,aluex,aluwb;
5     output reg [31:0] aluBout,aluAout;
6     input [1:0] numA,numB,numA1,numB1;
7     input ALUsrc;
8
9   always @(*)
10 begin
11   case(numB)
12     2'b10:
13       assign aluBout=aluex;
14
15     2'b00:assign aluBout=aluBin;
16     default:;
17   endcase
18   if(numB1==01)
19     assign aluBout=aluwb;
20     if(numA==00)
21     assign aluAout=aluAin;
22
23   case(numA)
24     2'b10:
25       assign aluAout=aluex;
26     2'b00:assign aluAout=aluAin;
27     default:;
28   endcase
29   if(numA1==01)
30     assign aluAout=aluwb;
31
32
33 end
34 endmodule
35

```

该单元使用 CASE 语句根据输入的数据冒险判断信号将 ALU 的操作数 A、B 赋予相应的值。

当 numB 为 10 时需要将 EX/MEM 级的数据旁路给 ALU 操作数 B，此时 aluBout=aluex。

当 numB 为 00 时，不需要旁观此时 aluBout=aluBin。

当 numB1 为 01 时，需要将 MEM/WB 级的数据提前旁路给 ALU 操作数 B，此时 aluBout=aluwb。

当 numA 为 10 时需要将 EX/MEM 级的数据旁路给 ALU 操作数 A，此时 aluAout=aluex。

当 numA 为 00 时，不需要旁观此时 aluAout=aluAin。

当 numA1 为 01 时，需要将 MEM/WB 级的数据提前旁路给 ALU 操作数 A，此时 aluAout=aluwb。

## 4.18 LW 冒险检测单元（Check）

---

```

1  module Check(rt,rdx,block,DMemR,clk,rst);
2    output reg block;
3    input rt,rdx,DMemR,clk,rst;
4    always @(*) begin
5      if((rt==rdx)&DMemR==1)
6        assign block=1'b1;
7      else begin
8        assign block=1'b0;
9      end
10     end
11   endmodule
12
13

```

该单元用于检测是否存在 LW 数据冒险。当 ID 级的 RT 与 EX 级的 RD 相等并且存储器读信号为 1 时，将发生 LW 型冒险，需要进行阻塞，因此将阻塞信号 block 赋值为 1，否则为 0。

#### 4.19 LW 冒险阻塞信号控制单元(HD)

---

```

1  module HD(block,PCrun,IDEZzero,clk,rst,IFIDzero);
2    output reg PCrun;
3    output reg IDEZzero,IFIDzero;
4    input block,clk,rst;
5    initial begin
6      PCrun=1'b1;
7      IDEZzero=1'b1;
8      IFIDzero=1'b1;
9    end
10   always @(*) begin
11     if(block==1'b0)
12       begin
13         assign PCrun=1'b1;
14         assign IDEZzero=1'b1;
15         assign IFIDzero=1'b1;
16       end
17     if(block==1'b1)
18       begin
19         assign PCrun=1'b0;
20         assign IDEZzero=1'b0;
21         assign IFIDzero=1'b0;
22       end
23     end // end always
24
25   endmodule
26

```

该单元首先将 IDEX 单元的阻塞信号 IDEXreg、IFID 单元的阻塞信号 IFIDreg，PC 单元阻塞信号 PCrun 初始化为 1（不阻塞）。根据输入的阻塞判断信号 block，当 block 为 1 时，需要阻塞将这三个信号置为 0，当 block 为 0 时，不需要阻塞三个信号皆为 1。

---

## 4.20 ALU 的 B 输入选择 (MUX3)

```
1 module MUX3(block,alu_Bin,alu_Bout,DMout);
2   input block;
3   input [31:0] alu_Bin,DMout;
4   output reg [31:0] alu_Bout;
5
6   always @(*)
7   begin
8     if(block==1'b1)
9       begin
10       assign alu_Bout=DMout;
11     end
12     if(block==1'b0)
13       assign alu_Bout=alu_Bin;
14   end
15 endmodule
16
17
18
```

该单元输入为当前 ALU 操作数 B 的值，与阻塞信号 block。

如果 block 为 1，则说明发生了 LW 数据冒险阻塞完毕，并且需要将存储器读出的数据提前旁路给 EX 级的 ALU 的操作数 B。

若阻塞信号为 0 则 ALU 操作数 B 保持不变。

## 5. 测试和结果分析

### 5.1 测试文件

```
ori $29, $0, 12
ori $2, $0, 0x1234
ori $3, $0, 0x3456
addu $4, $2, $3
subu $6, $3, $4
sw $2, 0($0)
sw $3, 4($0)
sw $4, 4($29)
lw $5, 0($0)
beq $2, $5, _lb2
_lb1:
```

---

```
lw $3, 4($29)
```

```
_lb2:
```

```
lw $5, 4($0)
```

```
beq $3, $5, _lb1
```

```
subu $6, $6, $2
```

## 5. 2 测试机器码

341d000c

34021234

34033456

00432021

00643023

ac020000

ac030004

afa40004

8c050000

10450001

8fa30004

8c050004

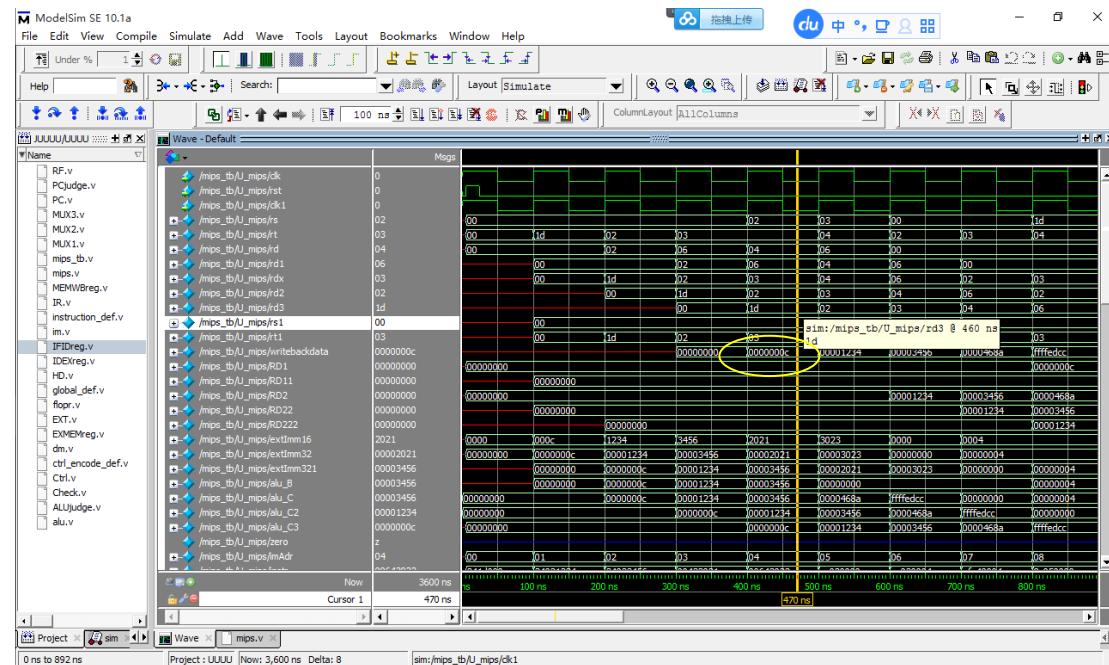
1065ffffd

00c23023

## 5.3 测试结果分析

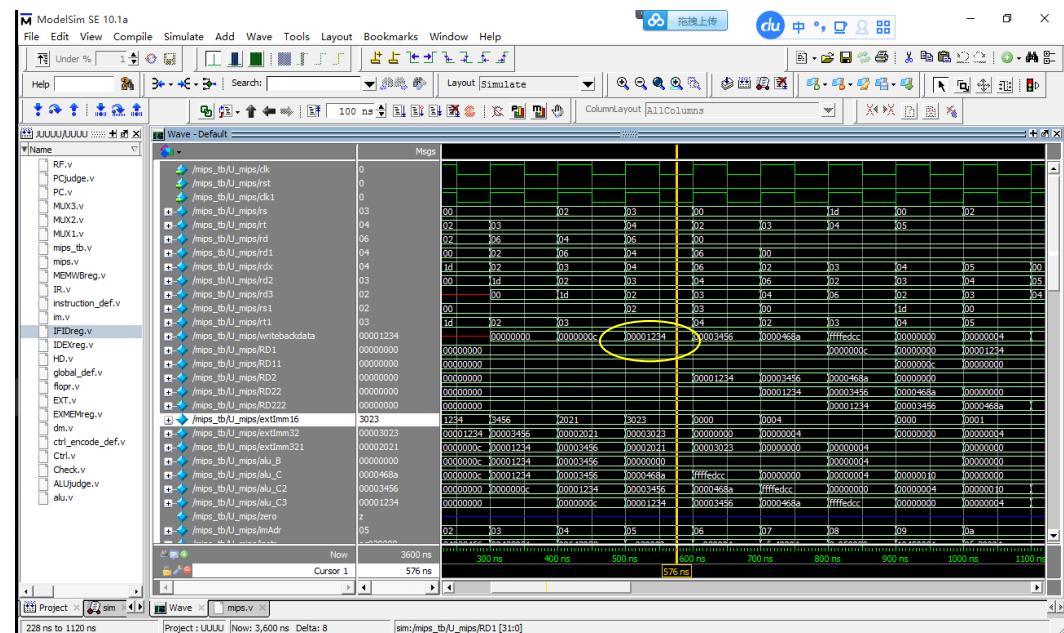
### 5.3.1 ori \$29, \$0, 12 指令

该指令的仿真结果如下图所示，该指令表示将\$0 寄存器与常数 12 进行“或”运算，结果放入\$29 号寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003000（十六进制），该指令的机器码为 341d000c（十六进制）写回寄存器地址 rd (rd3) 寄存器编号也是 1d (十进制是 29)，寄存器写入的数据 (RWD) 是 0000000c，与指令的预期结果是一致的，说明该指令仿真结果正确。



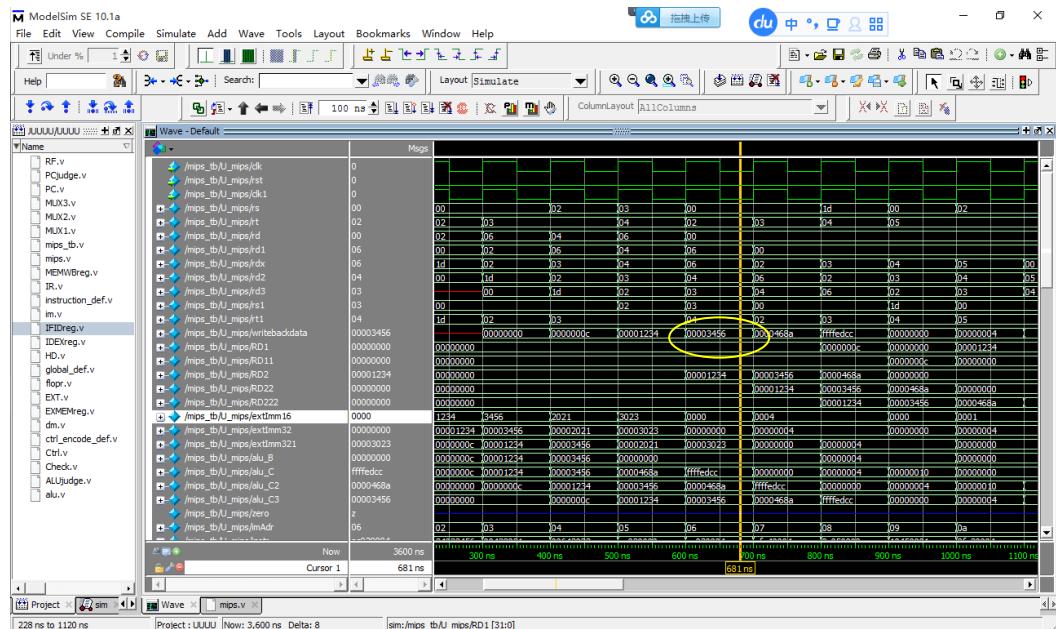
### 5.3.2 ori \$2, \$0, 0x1234 指令

该指令的仿真结果如下图所示，该指令表示将\$0 寄存器与常数 0001001000110100（二进制）进行“或”运算，结果放入\$2 号寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003004（十六进制），该指令的机器码为 34021234（十六进制），写回地址 rd（rd3）寄存器编号是 02，寄存器写入的数据（RWD）是 00001234，与指令的预期结果是一致的，说明该指令仿真结果正确。



### 5.3.3 ori \$3, \$0, 0x3456 指令

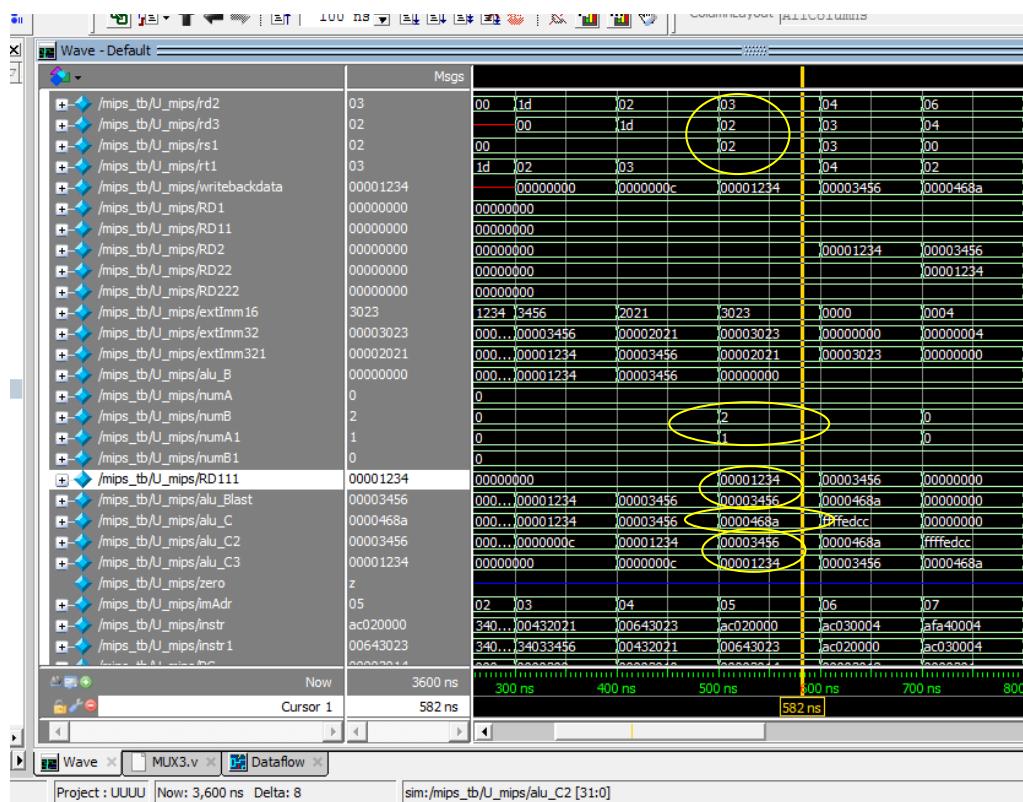
该指令的仿真结果如下图所示，该指令表示将\$0 寄存器与常数 3456（十六进制）进行“或”运算，结果放入\$3 号寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003008（十六进制），该指令的机器码为 34033456（十六进制），写回地址 rd(rd3) 寄存器编号是 03，寄存器写入的数据 (RWD) 是 00003456，与指令的预期结果是一致的，说明该指令仿真结果正确。



### 5.3.4 addu \$4, \$2, \$3 指令

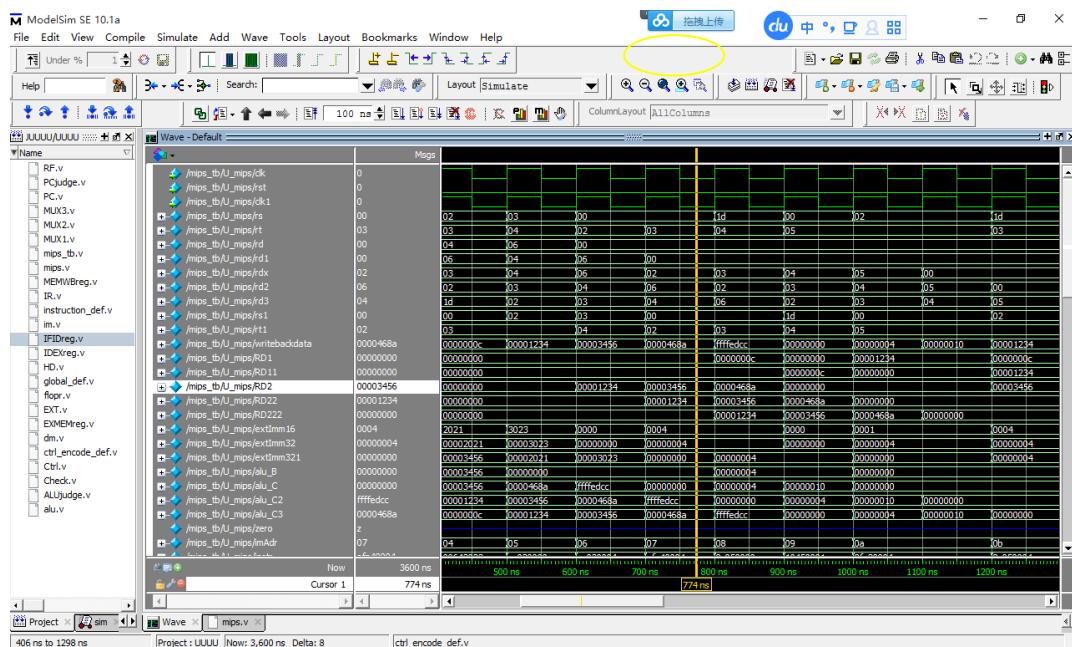
该指令的仿真结果如下图所示，该指令表示将\$2 寄存器与\$3 寄存器进行“加法”运算，结果放入\$4 号寄存器。

- ① 该指令存在数据冒险  $rs1=rd3$ ,  $rt1=rd2$ 。此时冒险判断信号  $numA1=01, numB=10$ 。需要将 EX/MEM 级的  $alu\_C2$  提前旁路给 ALU 操作数 B ( $alu\_Blast$ ) 为 00003456, 将 MEM/WB 级的  $alu\_C3$  旁路给 ALU 操作数 A ( $RD111$ ) 为 00001234。计算出 ALU 结果  $alu\_C=000468a$ 。根据仿真结果 EX 级，该过程正确。



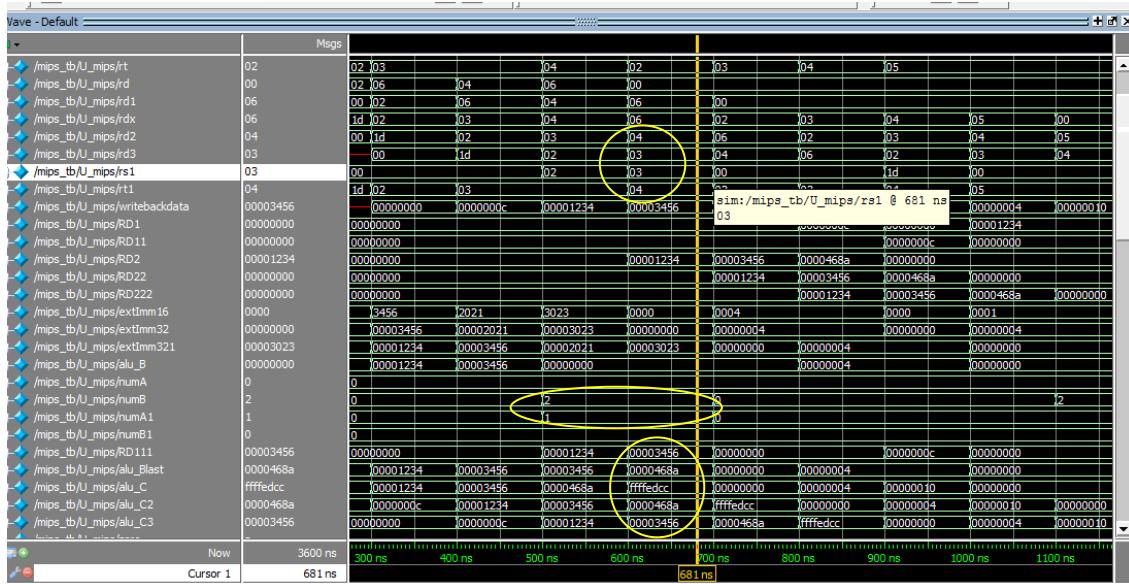
②仿真结果表明，该指令所在的 PC 寄存器地址为 0000300c（十六进制），该指令的机器码为 00432021（十六进制），两者相加的结果为 0000468a（十六进制）。在 WB 级寄存器写入的数据（RWD）也是 0000468a，与指令的预期结果是一致的。

说 明 该 指 令 仿 真 结 果 正 确

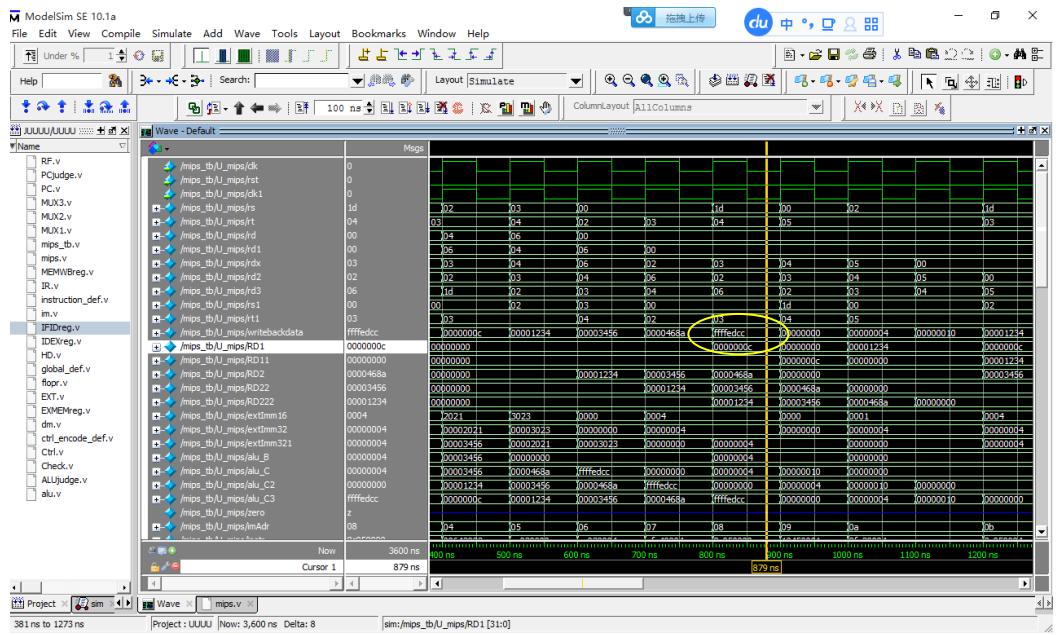


### 5.3.5 subu \$6, \$3, \$4 指令

① 该指令存在数据冒险  $rs1=rd3$ ,  $rt1=rd2$ 。此时冒险判断信号  $numA1=01, numB=10$ 。需要将 EX/MEM 级的  $alu\_C2$  提前旁路给 ALU 操作数 B ( $alu\_Blast$ ) 为 0000468a, 将 MEM/WB 级的  $alu\_C3$  旁路给 ALU 操作数 A (RD111) 为 00003456。计算出 ALU 结果  $alu\_C=fffffedcc$ 。根据仿真结果(EX 级), 该过程正确。

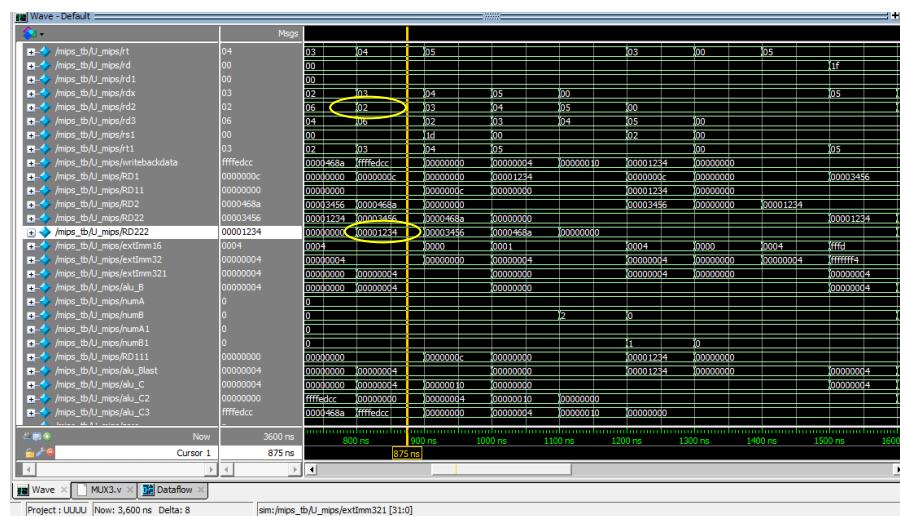


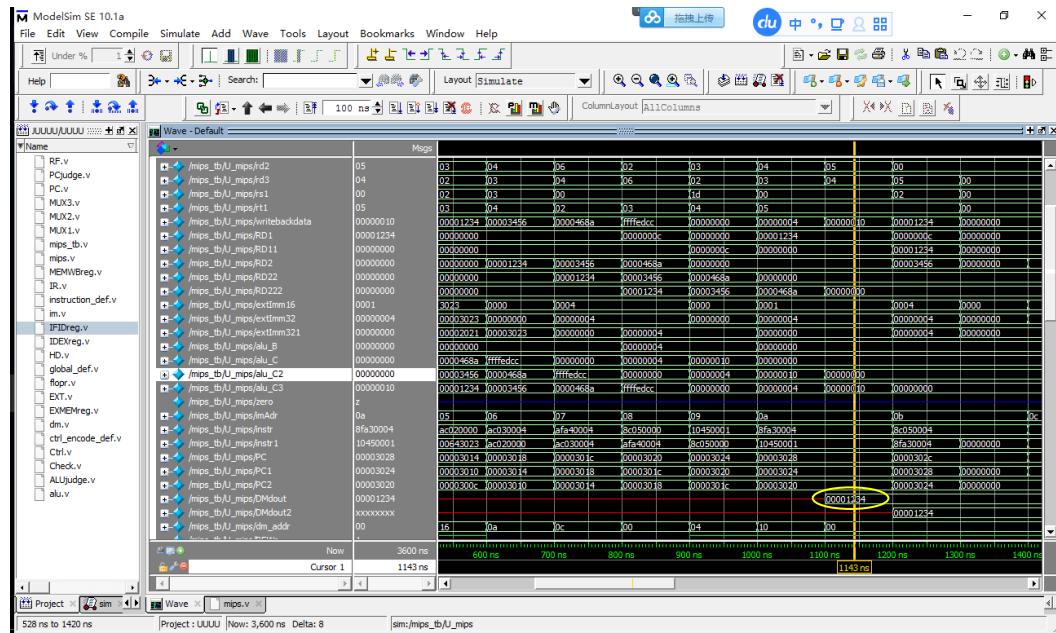
② 仿真结果表明, 该指令所在的 PC 寄存器地址为 00003010 (十六进制), 该指令的机器码为 00643023 (十六进制), 两者相减的结果为 fffffedcc (十六进制)。在 WB 级寄存器写回的数据 (RWD) 也是 fffffedcc, 与指令的预期结果是一致的, 说明该指令仿真结果正确



### 5.3.6 sw \$2, 0(\$0) 指令

该指令的仿真结果如下图所示，该指令表示将\$2 寄存器存入\$0+0 的内存单元中，\$0 寄存器的值总是为 0，因此计算的地址是 0，即：将\$2 寄存器存入 0 号内存单元。仿真结果表明，该指令所在的 PC 寄存器地址为 00003014（十六进制），该指令的机器码为 ac02000（十六进制），MEM 级 rd (rd2) 寄存器编号是 02（十进制是 2），RD222 的输出为 00001234（十六进制），也就是前面指令执行后\$2 寄存器的内容，存储器的写入地址 dm\_addr 的值为 0，即写入地址为 0，写入的数据即为 RD222 的输出值 00001234。

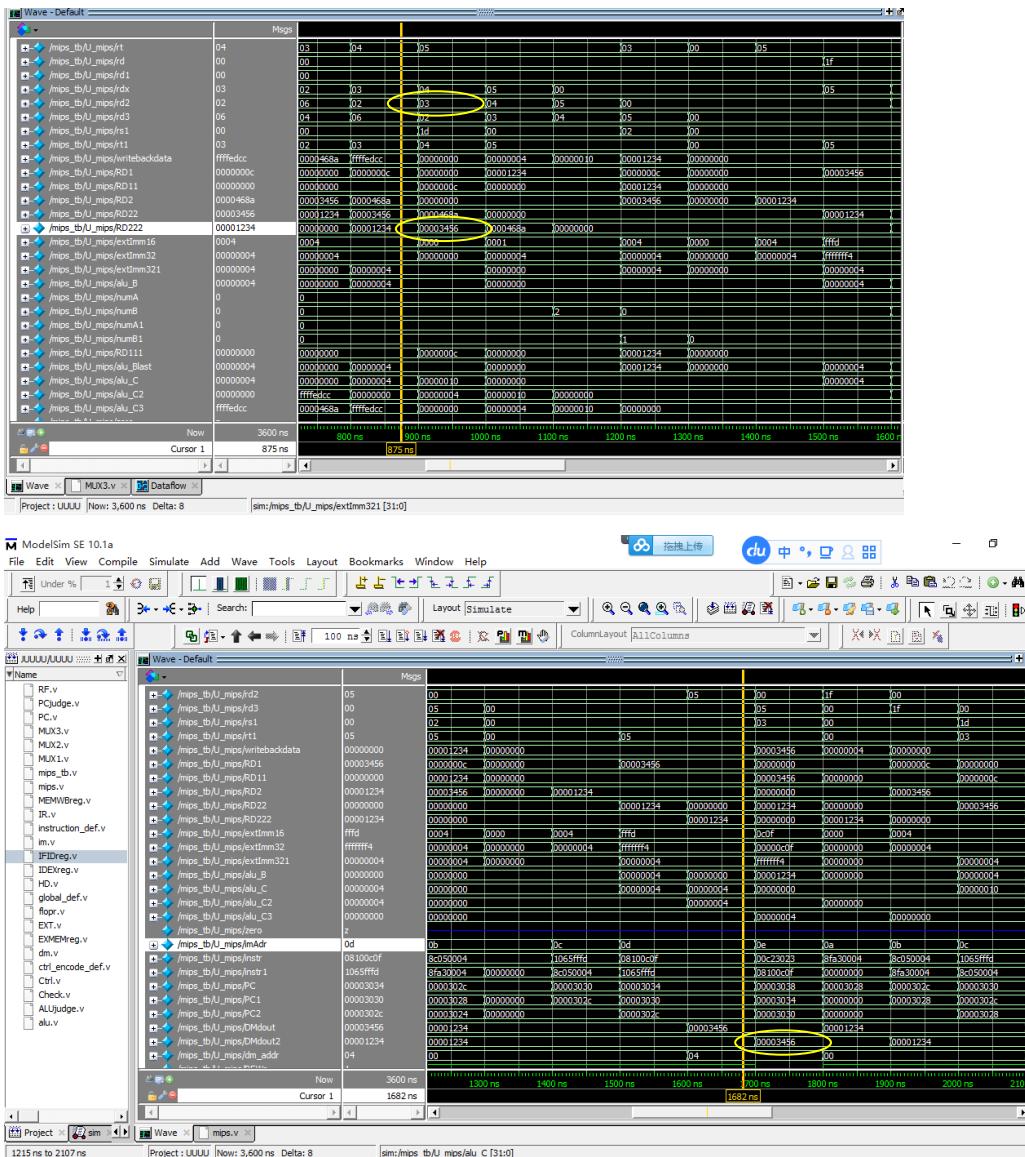




MEM 读出数据显示最终 `dm_dout` 的值为 00001234（十六进制）与指令的预期结果是一致的，说明该指令仿真结果正确。

### 5.3.7 sw \$3, 4(\$0) 指令

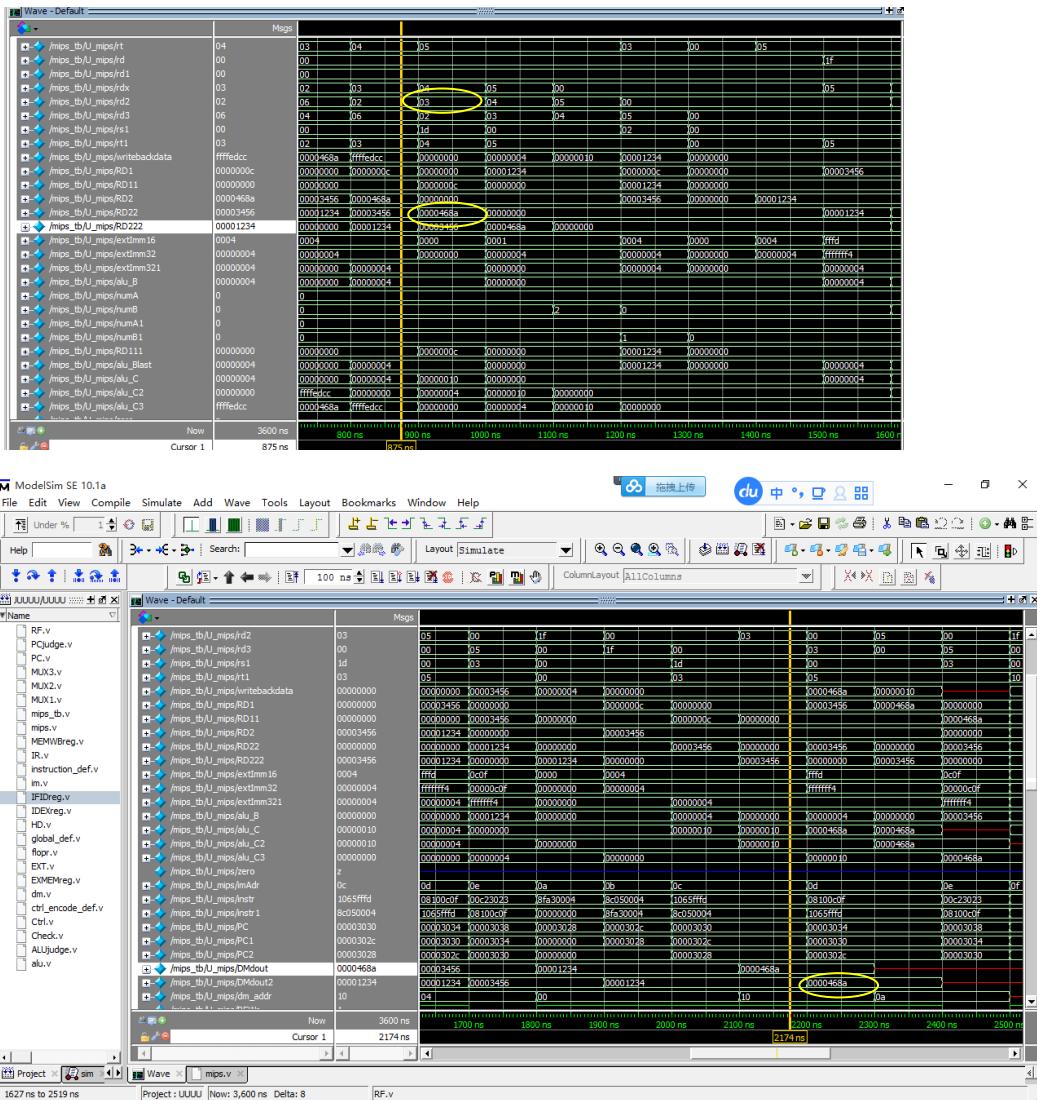
该指令的仿真结果如下图所示，该指令表示将\$3 寄存器存入\$0+4 的内存单元中，\$0 寄存器的值总是为 0，因此计算的地址是 4，即：将\$3 寄存器存入 4 号内存单元。仿真结果表明，该指令所在的 PC 寄存器地址为 00003018（十六进制），该指令的机器码为 ac030004（十六进制），MEM 级 rd (rd2) 寄存器编号是 03（十进制是 3），RD222 的输出为 00003456（十六进制），也就是前面指令执行后 \$3 寄存器的内容，存储器的写入地址 `dm_addr` 的值为 4，即写入地址为 4，写入的数据即为 RD222 的输出值 00003456。



MEM 读出数据显示最终 dm\_dout 的值为 00003456（十六进制）与指令的预期结果是一致的，说明该指令仿真结果正确。

### 5.3.8 sw \$4, 4(\$29) 指令

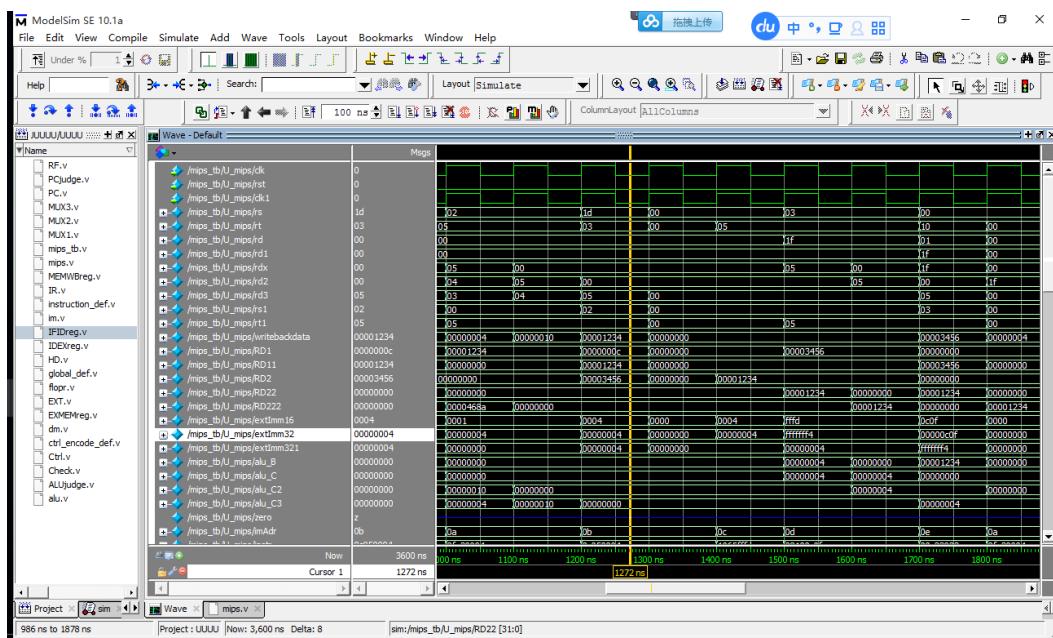
该指令的仿真结果如下图所示，该指令表示将\$4 寄存器存入\$29+4 的内存单元中，\$29 寄存器的值总是为 0c，因此计算的地址是 10，即：将\$4 寄存器存入 10 号内存单元。仿真结果表明，该指令所在的 PC 寄存器地址为 0000301c（十六进制），该指令的机器码为 afa40004（十六进制），MEM 级 rd (rd2) 寄存器编号是 04（十进制是 4），RD222 的输出为 0000468a（十六进制），也就是前面指令执行后\$4 寄存器的内容，存储器的写入地址 dm\_addr 的值为 10，即写入地址为 10，写入的数据即为 RD222 的输出值 0000468a。



MEM 读出数据显示最终 dm\_dout 的值为 0000468a（十六进制）与指令的预期结果是一致的，说明该指令仿真结果正确。

### 5.3.9 lw \$5, 0(\$0) 指令

该指令的仿真结果如下图所示，该指令表示将\$0+0 的内存单元中数据取到\$5 寄存器，\$0 寄存器的值总是为 0，因此计算的地址是 0，即：将 0 号内存单元的值取到\$5 寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003020（十六进制），该指令的机器码为 8c05000（十六进制），WB 级 rd (rd3) 寄存器编号是 05（十进制是 5），存储器的输出值 dm\_dout 的值为 00001234（十六进制），也就是前面指令存进去的值，寄存器的写入数据 RWD 是 00001234（十六进制），与指令的预期结果是一致的，说明该指令仿真结果正确。



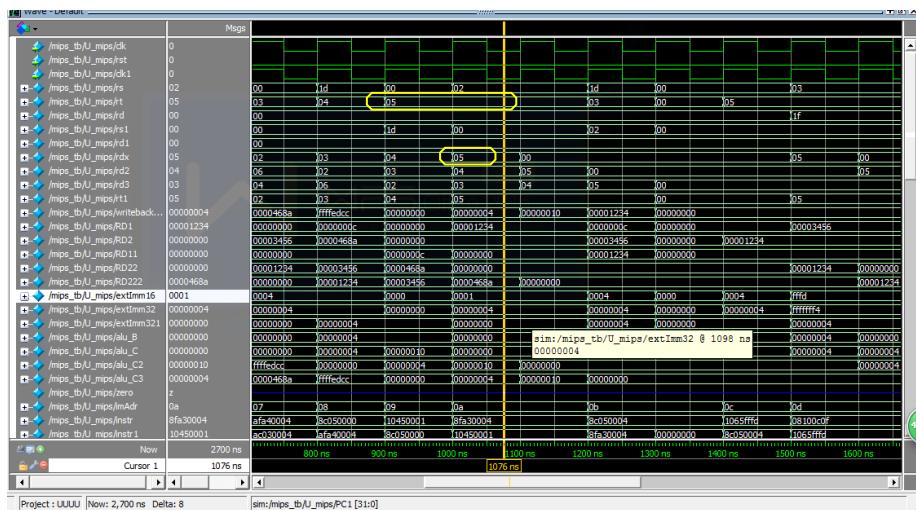
### 5.3.10 beq \$2, \$5, \_lb2 指令

该指令的仿真结果如下图所示，该指令表示将\$2寄存器与\$5寄存器进行“减”运算，结果不保存，通过置的Zero标记来决定是否转移。如果Zero=1则表示\$2寄存器与\$5寄存器数据相等，转移到\_lb2标号，否则就不转移。

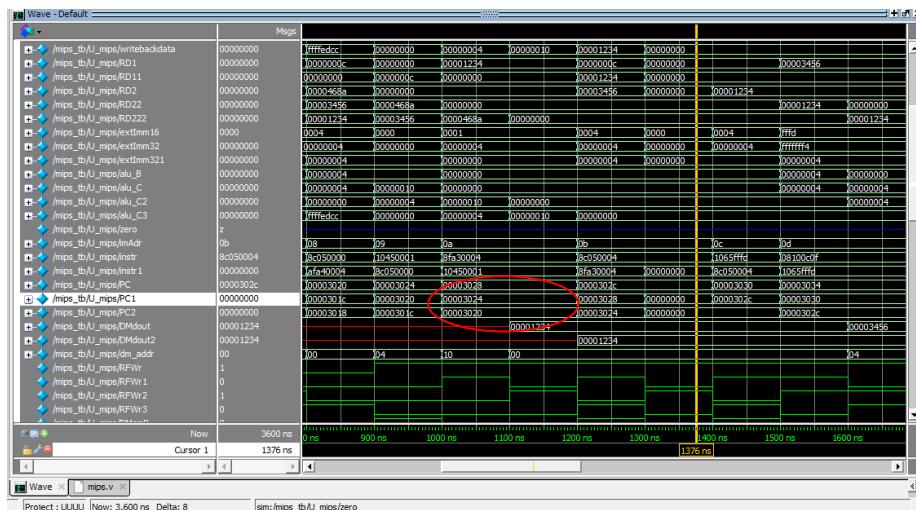
在BEQ指令执行到ID级时发现ID级的rt与EX级的rdx(就是rd)相等(而上一条指令的\$5的数据还没有装载进来，并且由于在MEM级无法直接旁路)发生了LW型数据冒险，于是阻塞信号block变为1，IFIDzero,IDEZzero, PCrun为0。IF/ID、ID/EX寄存器与PC不可写入，阻塞一个时钟周期。

LW冒险：

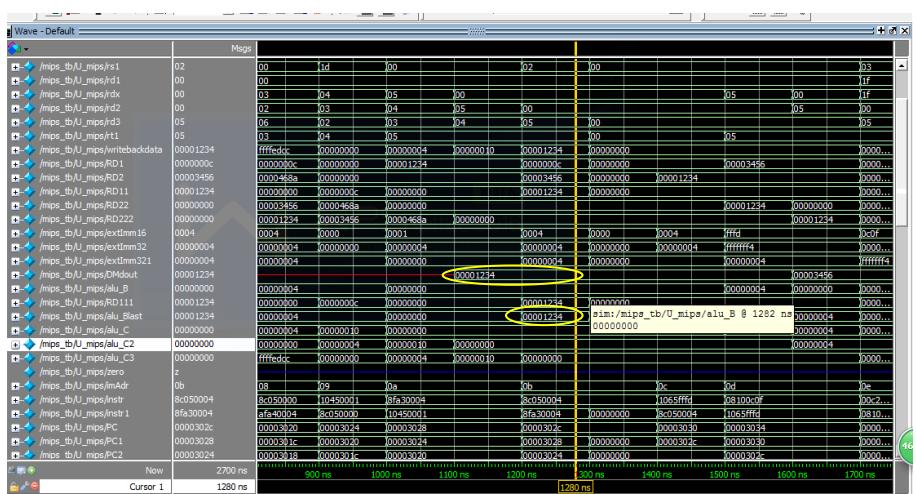
①下图为ID级冒险检测单元检测两个寄存器号相等



②下图为BEQ指令ID级阻塞一个时钟周期

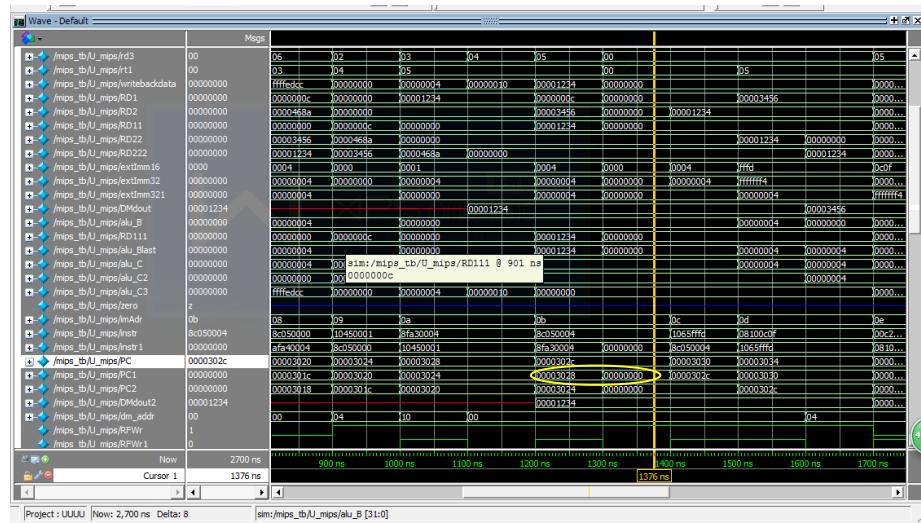


**控制（分支）冒险：**③BEQ 指令在 EX 级判断是否分支，将 MEM 级的数据旁路给 BEQ 的 EX 级如下图将 MEM 读出的数据 DMdout 旁路给 alu\_Blast 为 00001234，两者相减为 0 表示分支发生。

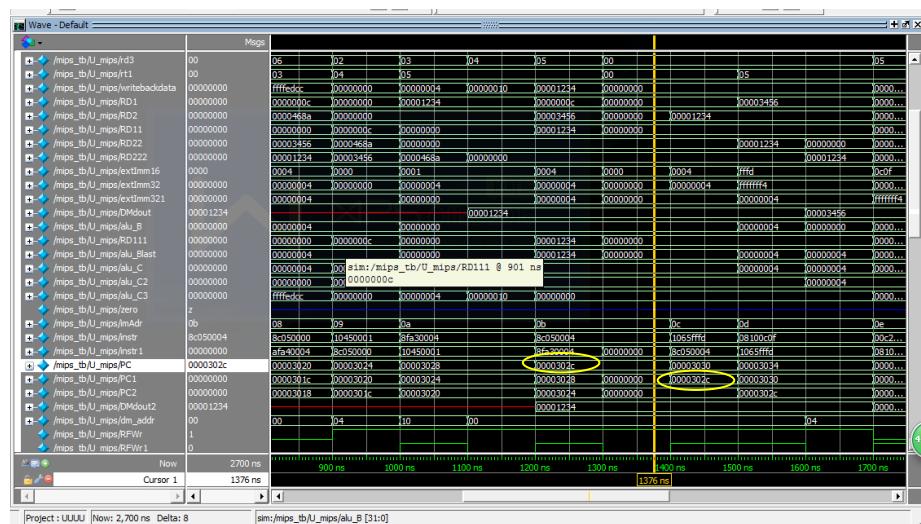


④分支发生PCWr为1，需要IF/ID.ID/EX（即原本下一条指令）数据舍弃lw \$3,

4(\$29)指令清0。下图为清0数据。

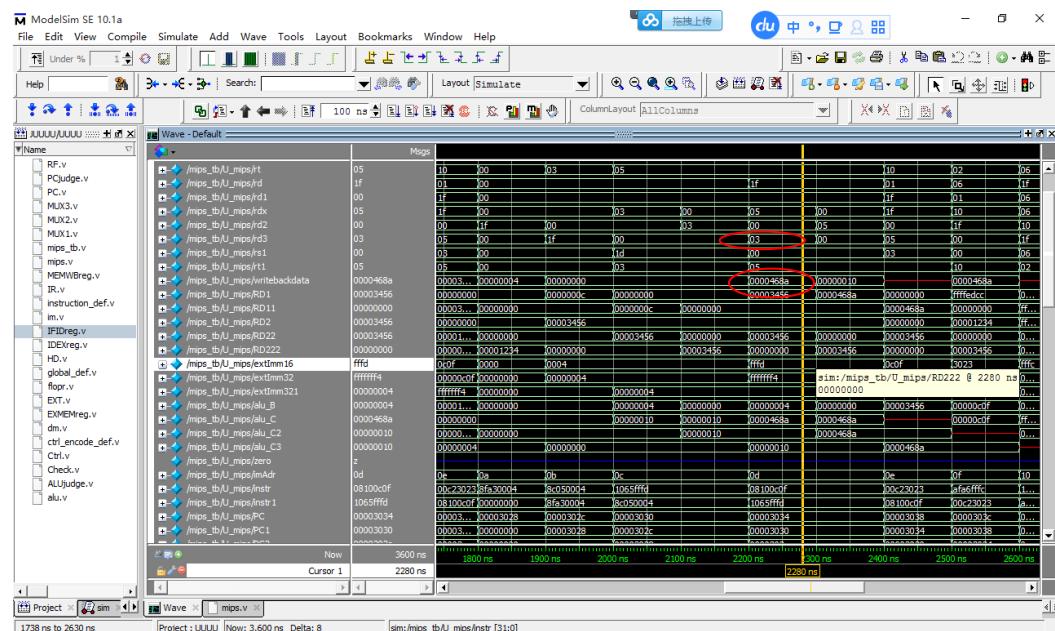


⑤仿真结果表明，该指令所在的PC寄存器地址为00003024（十六进制），该指令的机器码为10450001（十六进制），根据测试文件程序应该转移到下面的lw \$5, 4(\$0)指令，其指令地址为0000302c，与指令的预期结果是一致的，说明该指令仿真结果正确。如下图



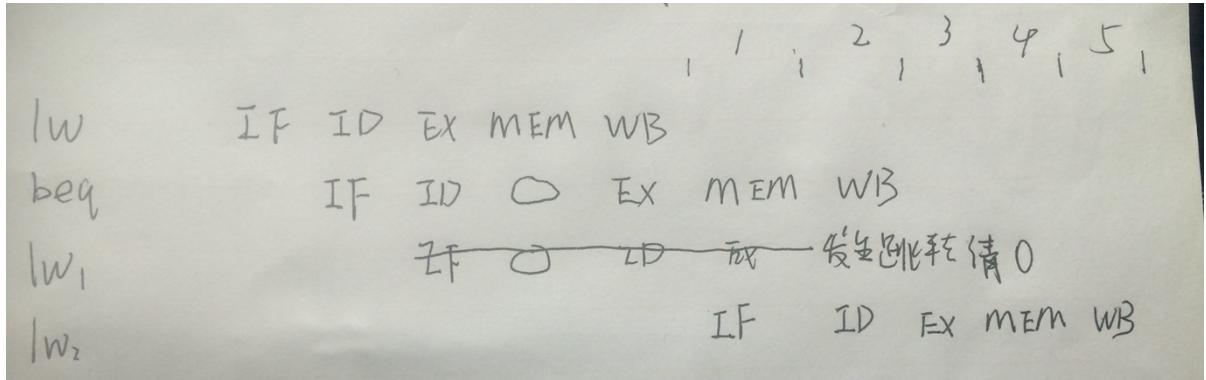
### 5.3.11 lw \$3, 4(\$29) 指令

该指令的仿真结果如下图所示，该指令表示将\$29+4 的内存单元中数据取到\$3 寄存器，\$29 寄存器的值总是为 0c，因此计算的地址是 10，即：将 10 号内存单元的值取到\$3 寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003028（十六进制），该指令的机器码为 8fa30004（十六进制），WB 级 rd (rd3) 寄存器编号是 03（十进制是 3），存储器的输出值 dm\_dout 的值为 0000468a（十六进制），也就是前面指令存进去的值，寄存器的写入数据 RWD 是 0000468a（十六进制），与指令的预期结果是一致的，说明该指令仿真结果正确。

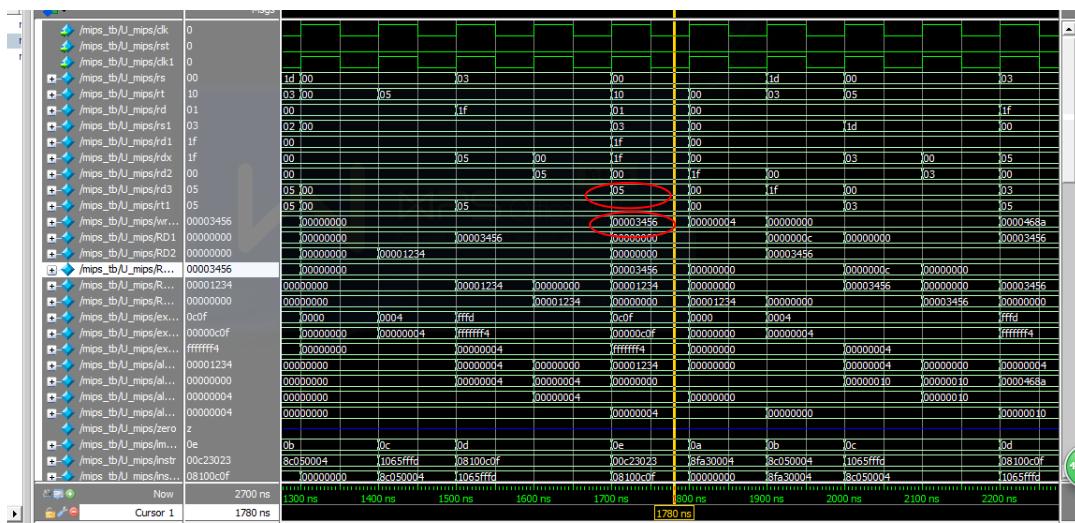


### 5.3.12 lw \$5, 4(\$0) 指令

由于 BEQ 发生 LW 数据冒险阻塞以及分支冒险，LW 写回的结果比上一条 LW 指令将延迟四个时钟周期写回



该指令的仿真结果如下图所示，该指令表示将\$0+4 的内存单元中数据取到\$5 寄存器，\$0 寄存器的值总是为 0，因此计算的地址是 4，即：将 4 号内存单元的值取到\$5 寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 0000302c（十六进制），该指令的机器码为 8c050004（十六进制），WB 级 rd (rd3) 寄存器编号是 05（十进制是 5），存储器的输出值 dm\_dout 的值为 00003456（十六进制），也就是前面指令存进去的值，寄存器的写入数据 RWD 是 00003456（十六进制），与指令的预期结果是一致的，说明该指令仿真结果正确。



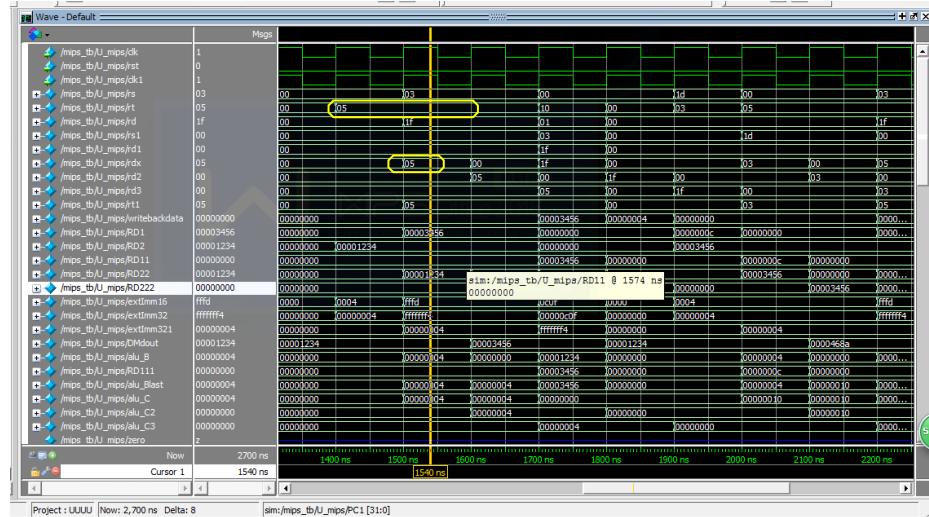
### 5.3.13 beq \$3, \$5, \_lb1 指令

该指令的仿真结果如下图所示，该指令表示将\$3寄存器与\$5寄存器进行“减”运算，结果不保存，通过置的Zero标记来决定是否转移。如果Zero=1则表示\$3寄存器与\$5寄存器数据相等，转移到\_lb1标号，否则就不转移。

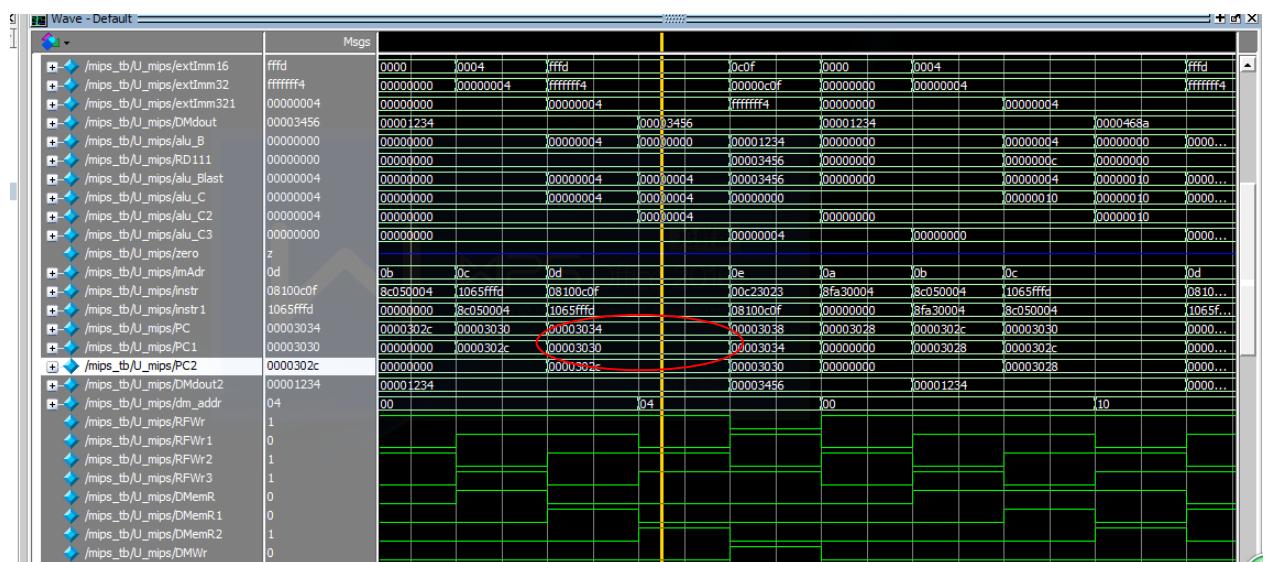
在BEQ指令执行到ID级时发现ID级的rt与EX级的rdx(就是rd)相等(而上一条指令的\$5的数据还没有装载进来，并且由于在MEM级无法直接旁路)发生了LW型数据冒险，于是阻塞信号block变为1，IFIDzero,IDEZzero, PCrun为0。IF/ID、ID/EX寄存器与PC不可写入，阻塞一个时钟周期。

LW冒险：

①下图为ID级冒险检测单元检测两个寄存器号相等

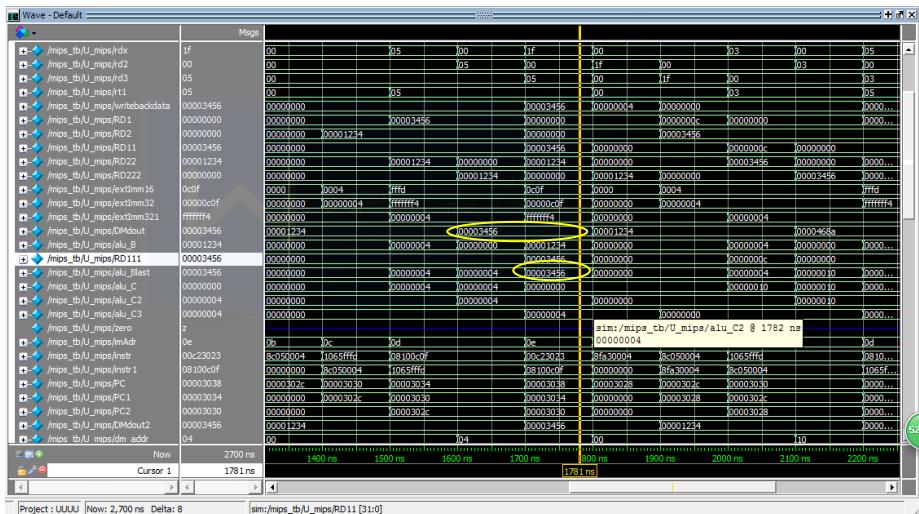


②下图为BEQ指令ID级阻塞一个时钟周期

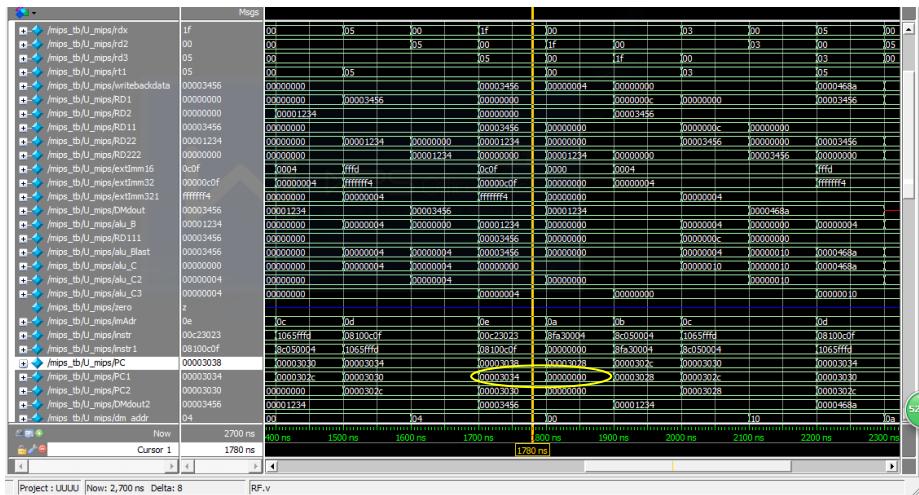


控制（分支）冒险：

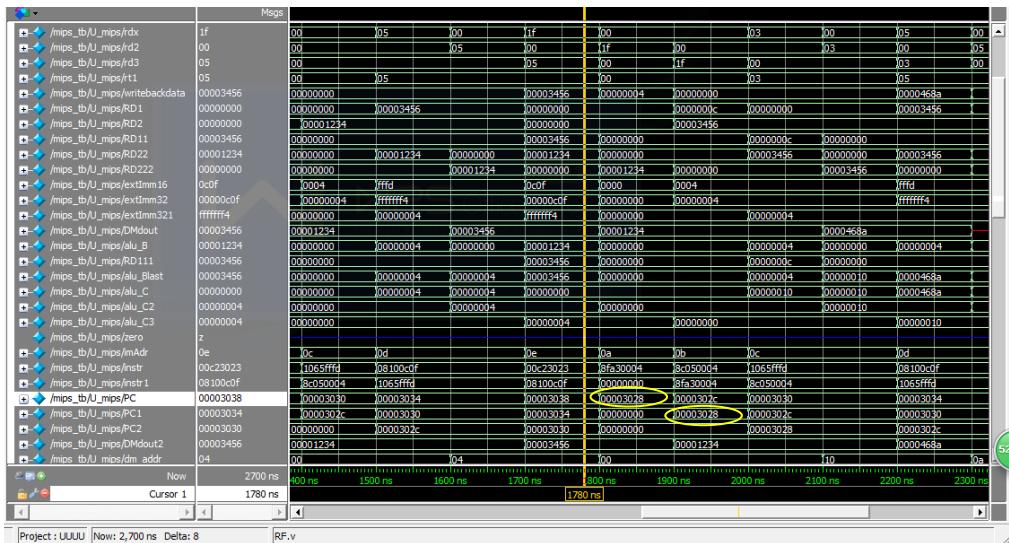
③BEQ指令在EX级判断是否分支，将MEM级的数据旁路给BEQ的EX级如下图，将MEM读出的数据DMdout旁路给alu\_Blast为00003456，两者相减为0表示分支发生。



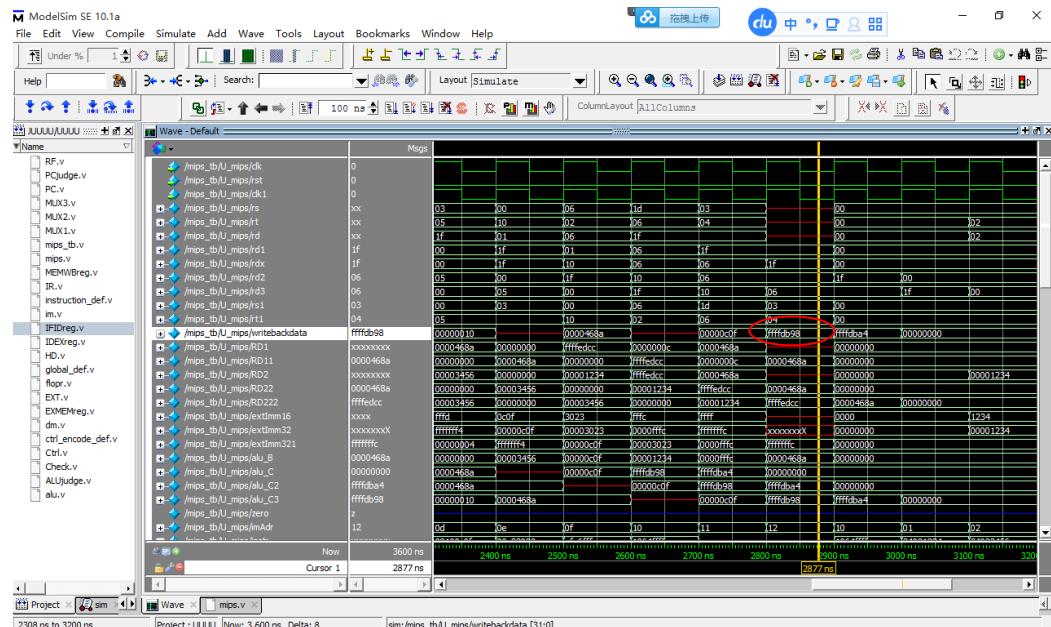
④分支发生PCWr为1，需要IF/ID, ID/EX（即原本下一条指令）数据舍弃下条指令清0。下图为清0数据。



⑤仿真结果表明，该指令所在的PC寄存器地址为00003030（十六进制），该指令的机器码为1065ffff（十六进制），根据测试文件程序应该转移到上面的lw \$3, 4(\$29)指令，其指令地址为00003028，与指令的预期结果是一致的，说明该指令仿真结果正确。如下图



### 5.3.14 subu \$6, \$6, \$2 指令



该指令的仿真结果如上图所示，该指令表示将\$6 寄存器与\$2 寄存器进行“减法”运算，结果放入\$6 号寄存器。仿真结果表明，该指令所在的 PC 寄存器地址为 00003038（十六进制），该指令的机器码为 00c23023（十六进制），两者相减的结果为 fffffdb98(十六进制)。在 WB 级寄存器写回的数据(RWD)也是 fffffdb98，与指令的预期结果是一致的，说明该指令仿真结果正确。