Translated from Chinese (Simplified) to English - www.onlinedoctranslator.com

MIPS pipeline CPU design

Wang Yiyi

main content

using a hardware description language (Verilog)designMIPSPipeline CPU supports the following instruction set: {add, sub, ori, lw, sw, beq, j}

Use the simulation software Modelsim to simulate the assembly program with data hazards and instruction hazards: such as:

```
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
```

basic requirements

using a hardware description language (Verilog) design the following modules:

(1) Program counter module (PC);

(2)Instruction memory module (im);

(3) Register module (RF);

(4) pipeline register (IFIDreg,IDEXreg,EXMEMreg,MEMWBreg);

(5) Data expansion module (EXT);

(6) Operator module (alu);

(7) Data memory module (DM);

(8) Controller module (CTRL);

(9) Machine connection module (MIPS) (including data risk and control risk detection);

(10) Data risk detection unit (MUX1, MUX2)

(11)PC selection unit (PCjudge)

(12)ALU result selection (ALUjudege)

(13)LW adventure detection unit (Check)

(14)LW risk blocking signal control unit (HD)

(15)ALU B input selection (MUX3)

Complete the simulation debugging of the assembly program.

References

【1】Computer principles and design:Verilog HDLEdition, written by Li Yamin

【2】VerilogDigital System Design Tutorial edited by Xia Yuwen (No.17simplifiedRISC CPUdesign)

1. Requirements analysis

After studying the principles of computer organization course, you need toCPUThe structural principle and

CPUIntegrate the knowledge of pipeline principles and design as well as data risk control risk. Pass

Through the comprehensive application of knowledge, deepen the understanding ofCPUThe working principle and pipeline of each module of the system

Understand the solutions to related adventures and learn more deeply and comprehensively through experimentsCPU

Assembly line design.

2. Design environment

2.1 Introduction to Verilog HDL

It is a hardware description language that describes the structure and behavior of digital system hardware in text form.

It can be used to represent logic circuit diagrams, logic expressions, and the logic completed by digital logic systems.

Function. forIEEEstandard.

2.1 Introduction to ModelSim

it isHDLLanguage simulation software can provide a friendly simulation environment and is the only single-core supportVHDL

andVERILOGEmulator for hybrid simulation. He uses direct optimization compilation technology,TCI/TKtechnology,

And single-core simulation technology, the compilation and simulation speed is fast, the compiled code has nothing to do with the platform, and is easy to protectIP

Core, personalized graphical interface and user interface provide users with a powerful means to speed up error debugging.

FPGA/ASICThe preferred simulation software for design.

2.3 Introduction to MARS

MIPS assembly language simulator, developed at Missouri State University.

3. Outline design

3.1 PC (Program Counter)

(1)Function description

conductPC+4Instruction counting and instruction selection based on signals.

(2) module interface

signal name	direction	describe
clk	input	clock
rst	input	reset signal
pcWr	input	The signal is1When indicating that the branch will occurPCset to BEQjump address
NPC[31:0]	input	BEQInstruction jump address
PC[31:0]	output	PCThe final output of the unitPCinstructions
PCjump	input	The signal is1And the branch signal is0when, say Obviously a jump occurred,PCwill be assigned toJUMPinstruction jump Forward address
PCrun	input	The signal is1hourPCThe unit operates normally and the signal for0hourPCThe unit is not writable,PCkeep last week period value
block	input	This signal is used for debugging when I write the unit. Actual does not work in this unit

3.2 RF (register) design

(1)Function description

According to the two addresses input by the read register, the two data corresponding to the register address are read out. According to the write-back address and

Write data back for register writeback.

(2) module interface

signal name	direction	describe
A1[4:0]	input	The address of the register reading data
A2[4:0]	input	The address of the register reading data
A3[4:0]	input	The address of the register reading data
WD[31:0]	input	Data written back by register
clk	input	clock signal
RFw	input	register write signalRFWr=1Register is writable
RD1[31:0]	output	RD1Data output for the register
RD2[31:0]	output	RD2Data output for the register

3.3 ALU (Arithmetic Logic Operation Unit)

(1)Function description

ALUThe main function is to complete the addition, subtraction, AND, OR, left shift, right shift, etc. of the input data.

Multiplication, division operations and determining whether two operands are equal.

(2) module interface

signal name	direction	describe
A [31:0]	I	operandA
B [31:0]	I	operandB
ALUOp[4:0]	I	operations required `ALUOp_ADDU:addition

		`ALUOp_SUBU:Subtraction `ALUOp_OR:OR operation
Zero	O	Are the two operands equal?
C [31:0]	O	Operation result

3.4 EXT (Extension unit)

(1)Function description

EXTThe main function is to16bits of data expanded to32Bit data, shift the data to the left by two bits.

(2) module interface

signal name	direction	describe
Imm16 [15:0]	I	Data that needs to be expanded
EXTOp[1:0]	I	Extended mode control signals `EXT_ZERO:0Expand `EXT_SIGNED: Sign extension, and shift the immediate value to the left by two Bit `EXT_HIGHPOS: Expand immediate data to high bits
Imm32 [31:0]	O	Extended results

3.5 DM (data memory)

(1)Function description

According to the input address and the memory read and write control signal, the data at the corresponding address is read or the data is written into the corresponding address.

(2) module interface

signal name	direction	describe
addr[4:0]	I	Register number written
din[31:0]	I	Data written to memory
DMWr	I	memory write data signal
dM	I	memory read data signal
clk	I	clock
dout[31:0]	O	Data read from memory

3.6 IM (Instruction Memory)

(1)Function description

In this unit, the corresponding instruction is read out from the memory according to the input read address.

(2) module interface

signal name	direction	describe
addr[4:0]	I	Read address of instruction memory
dout[31:0]	O	Instructions read from instruction memory

3.7 Ctrl (Controller)

(1)Function description

According to the high six digits of the input commandOPThe operation code determines the type and function of the current instruction and outputs the corresponding order.

element control signal.

(2) module interface

signal name	direction	describe

instr[5:0]	I	input opcode
funct[5:0]	I	Entered function code
jump	O	jump signal
RegDst	O	Register write back address select signal
Branch	O	branch signal
dM	O	memory read signal
DMnoutregR	O	Selection signal for writing back register data
DMWr	O	memory write signal
RFw	O	register write signal
Alusrc	O	ALUthe operandsBselection signal
EXTOp[1:0]	O	EXTfunction selection
ALUOp[4:0]	O	ALUfunction selection

3.8 MIPS (model machine)

(1)Function description

MIPSThe model machine connects each unit as a whole and inputs corresponding clock control signals and resets.

Signals enable each unit to form a complete pipeline data path.

(2) module interface

signal name	direction	describe
clk	I	clock
clk1	I	clock1for debugging
rst	I	reset signal

3.9 Pipeline register (IFIDreg)

(1)Function description

WillIFLevel signals and data are saved and passed in in the next clock cycleIDclass.

(2) module interface

signal name	direction	describe
clk	I	clock
PCin[31:0]	I	IFclassPCenter
rst	I	reset signal
PCout[31:0]	O	PCOutput toIDclass
instrin[31:0]	I	IFLevel instructions
instrout[31:0]	O	Instructions are output toIDclass
IFIDWr	I	IFIDwrite signal
IFIDzero	I	IFIDblocking signal
pcWr	I	BEQJump clear signal

3.10 Pipeline register (IDEXreg)

(1)Function description

WillIDLevel signals and data are saved and passed in in the next clock cycleEXclass.

(2) module interface

signal name	direction	describe(The following descriptions are fromIDarriveEXlevel letter Number)
clk	I	clock
IDEXzero	I	IDEXlevel blocking signal
rst	I	reset signal
Branchout	O	Branch signal output

Branchin	I	Branch signal input
RegDstout	O	Register address selection signal input
RegDstin	I	Register address selection signal output
pcWr	I	BEQJump clear signal
DMemRin	I	Memory read signal input
DMemRout	O	Memory read signal output
DMnoutregRin	I	Write back data selection signal input
DMnoutregRout	O	Write back data select signal output
DMWrin	I	Memory write signal input
DMWrout	O	Memory write signal output
RFWrout	O	Register write signal output
RFWrin	I	Register write signal input
Alusrcin	I	ALUoperandBSelect signal input
Alusrcout	O	ALUoperandBSelect signal output
EXTOpin[1:0]	I	Expansion unit control signal input
EXTOpout[1:0]	O	Expansion unit control signal output
ALUOpin[4:0]	I	ALUcontrol signal input
ALUOpout[4:0]	O	ALUcontrol signal output
RFout1in[31:0]	I	Register read data1enter
RFout2in[31:0]	I	Register read data2enter
RFout1out[31:0]	O	Register read data1output
RFout2out[31:0]	O	Register read data2output
Imm32in[31:0]	I	Extension unit result input
Imm32out[31:0]	O	Extension unit result output
rtin[4:0]	I	command sectionrtenter
rtout[4:0]	O	command sectionrtenter
rdin[4:0]	I	command sectionrdenter

rdout[4:0]	O	command sectionrdenter
rsin[4:0]	I	command sectionrsenter
rsout[4:0]	O	command sectionrsenter
PCin[31:0]	I	instructionPCenter
PCout[31:0]	O	instructionPCoutput

3.11 Pipeline register (EXMEMreg)

(1)Function description

WillEXLevel signals and data are saved and passed in in the next clock cycleMEMclass.

(2) module interface

signal name	direction	describe(The following descriptions are fromEXarriveMEMClass Signal)
clk	I	clock
rst	I	reset signal
Branchout	O	Branch signal output
Branchin	I	Branch signal input
DMemRin	I	Memory read signal input
DMemRout	O	Memory read signal output
DMnoutregRin	I	Write back data selection signal input
DMnoutregRout	O	Write back data select signal output
DMWrin	I	Memory write signal input
DMWrout	O	Memory write signal output
RFWrout	O	Register write signal output
RFWrin	I	Register write signal input
blockin	I	Block signal input

blockout	O	Block signal output
RFout2in[31:0]	I	Register read data2enter
RFout2out[31:0]	O	Register read data2output
bbbin[31:0]	I	BEQJump address input
bbbout[31:0]	O	BEQJump address output
wbaddrin[4:0]	I	Write back address input
wbaddroutt[4:0]	O	Write back address output input
rdin[4:0]	I	command sectionrdenter
rdout[4:0]	O	command sectionrdenter
zeroin	I	Zero identification input
zeroout	O	Zero flag output
aluCin[31:0]	I	ALUresultCenter
aluCout[31:0]	O	ALUresultCoutput

3.12 Pipeline register (MEMWBreg)

(1)Function description

WillMEMLevel signals and data are saved and passed in in the next clock cycleWBclass.

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
clk	I	clock
rst	I	reset signal
DMnoutregRin	I	Write back data selection signal input
DMnoutregRout	O	Write back data select signal output
RFWrout	O	Register write signal output

RFWrin	I	Register write signal input
blockin	I	Block signal input
blockout	O	Block signal output
dmoutin[31:0]	I	Memory read data input
dmoutout[31:0]	O	Memory read data output
wbaddrin[4:0]	I	Write back address input
wbaddroutt[4:0]	O	Write back address output input
aluCin[31:0]	I	ALUresultCenter
aluCout[31:0]	O	ALUresultCoutput

3.13 Data risk detection unit (MUX1)

(1)Function description

Check whetherEXClassRT,RSandEX/MEMClassRDequal, If they are equal, a data hazard will occur.

Output the corresponding signal of data risk so that it can bypass the required data toALU.

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
Rs[4:0]	I	EXClassRS
Rt[4:0]	I	EXClassRT
RFw	I	register write signal
numA[1:0]	O	Determine risk signalsA
numB[1:0]	O	Determine risk signalsB
exmemrd[4:0]	I	EXMEMClassRD
clk	I	clock

3.14 Data hazard detection unit (MUX2)

(1)Function description

Check whetherEXClassRT,RSandMEM/WBClassRDare equal. If they are equal, a data risk will occur.

Risk, output the corresponding signal of data risk, so that it can bypass the required data toALU.

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
Rs[4:0]	I	EXClassRS
Rt[4:0]	I	EXClassRT
RFw	I	register write signal
numA1[1:0]	O	Determine risk signalsA1
numB1[1:0]	O	Determine risk signalsB1
memwbrd[4:0]	I	MEMWBClassRD
clk	I	clock

3.15PCSelect unit (PCjudge)

(1)Function description

According to the input signals, instructions and data, the output is reallyPCThe value of the cell input

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
PCreal	O	finally inPCunit inputNPC
pcWr	I	BEQbranch signal
bbb[31:0]	I	BEQJump address
Imm32[31:0]	I	Extended unit data

instr[31:0]	I	instruction
jump	I	unconditional jump signal

3.16ALUResult selection unit (ALUjudege)

(1)Function description

Judge the signal based on the input data and select the correspondingALUoperandAAND operandB.

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
aluBin[31:0]	I	operandBenter
aluBout[31:0]	O	operandBoutput
aluAin[31:0]	I	operandAenter
aluAout[31:0]	O	operandAoutput
numA[1:0]	I	risk-taking signals
numB[1:0]	I	risk-taking signals
numA1[1:0]	I	risk-taking signals
numB1[1:0]	I	risk-taking signals
aluex[31:0]	I	EX/MEMlevel bypass toALUThe data
aluwb[31:0]	I	MEM/WBlevel bypass toALUThe data
ALUsrc	I	(Signal for debugging)

3.17LWAdventure Detection Unit (Check)

(1)Function description

Detect whether it occurs based on the input data and signalsLWData risk, and output the corresponding blocking signal

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
rt[4:0]	I	EXClassRT
rdx[4:0]	I	MEMClassRD
block	O	blocking signal
dM	I	memory write signal
rst	I	reset
clk	I	clock

3.18LWRisk blocking signal control unit (HD)

(1)Function description

According to the input data and blocking signal, the control signal of the corresponding unit is output.IF/ID,ID/EX,PCThe unit is

No blocking.

(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
PCrun	O	PCcontrol signal
IDEXzero	O	ID/EXcontrol signal
IFIDzero	O	IF/IDcontrol signal
block	I	blocking signal
rst	I	reset
clk	I	clock

3.19ALUoperandBselection unit (MUX3)

(1)Function description

Select the operand based on the input data and blocking signalBWill the value ofMEMlevel data bypass

GiveALUthe operandsB.

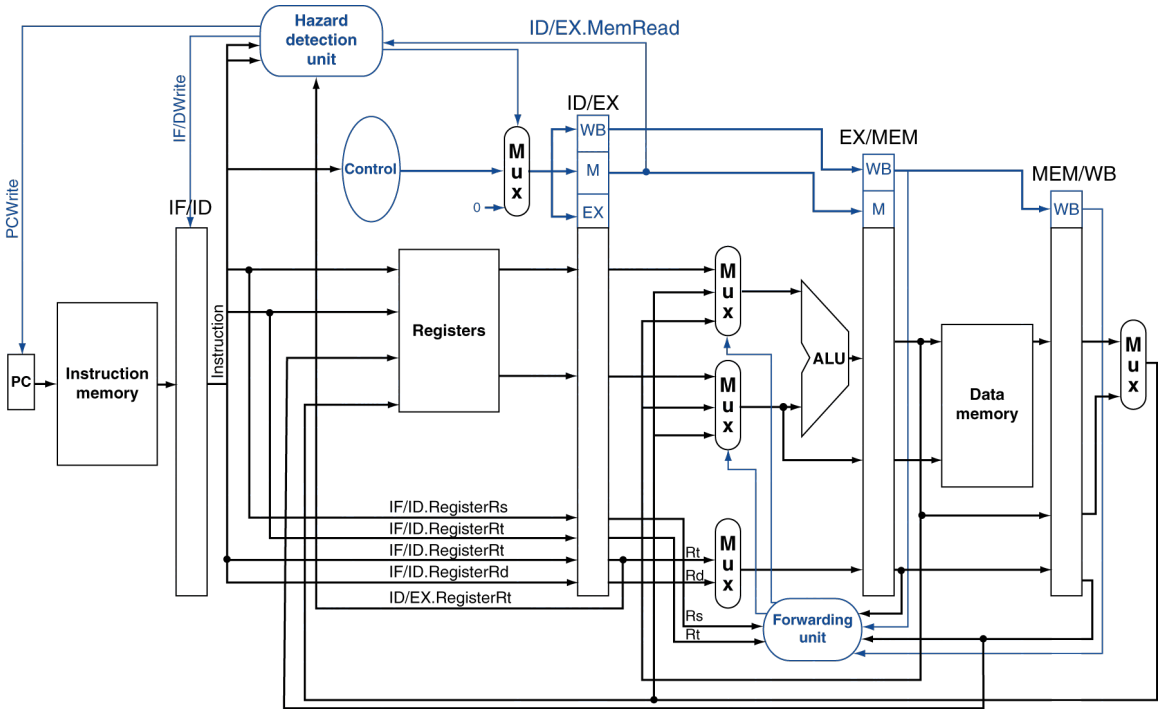
(2) module interface

signal name	direction	describe(The following descriptions are fromMEMarriveWBClass Signal)
DMout[31:0]	I	Read data from memory
alu_Bout[31:0]	O	ALUBdata output
alu_Bin[31:0]	I	ALUBdata input
block	I	blocking signal

4. Detailed design

4.1 Overall structure of CPU

CPUThe overall structure of the pipeline data path is shown in the figure below, including the program counter (PC), instruction memory (im), register group (RF), operator (ALU), data expansion unit (EXT), data memory (DM) and controller (Ctrl), pipeline register (IFIDreg,IDEXreg ,EXMEMreg, MEMWBreg), data hazard detection unit (MUX1, MUX2), PC selection unit (PCjudge), ALU result selection unit (ALUjudege), LW hazard detection unit (Check), LW hazard blocking signal control unit (HD), ALU B input selection (MUX3).



The design of each sub-module is introduced below.

4.2 PC (Program Counter)

```
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
```

The basic function is an instruction counter, which is automatically executed every clock cyclePC+4calculate. whenPCrunfor1hour, PCWritable, so it differs according to the input control signalPCThe final output value of the unitPCwill change when PCWr=1When explainingBEQIf the instruction jumps, you need toPCThe value is set toNPC(i.e.BEQrefer to order low16The bit is expanded by the expansion unit to32shifted two bits to the left andPC+4the result of the addition). whenPCWr=0 andPCjump=1Indicates that the current instruction is an unconditional transfer instruction, then directly transferPCset toJUMPInstruction address (i.e. low26Shift left two bits andPC+4The result of adding the high four bits). whenpcWrandPCjumpAll are 0, count normallyPC=PC+4. whenPCrunfor0time, at this timePCThe value of the cell cannot be changed with The value from the previous period remains the same.

4.3 RF (register) design

```
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, %8X", 0, rf[1],
24            $display("R[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[8], rf
25            $display("R[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[16], r
26            $display("R[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[24], r
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
```

Basic functions are based onA2,A3The input value (register number) is read out and the register group data is saved to

RD1,RD2output in the register. First, set the initial values of the register group to0(This statement only executes one

Second-rate).RFWr=1The register can be written whenA1The value (register number) ofWD(register

Write back data) writes the data to the corresponding register.

4.4 ALU (Arithmetic Logic Operation Unit)

```
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
```

According to the input operandA,BThe value ofALUControl signals are added, subtracted or operated. First putZEROand

CThe initial value is set to0. according toctrl_encode_deffunction definition in , whenALUOpfor00001hour

Perform the addition operation to00011For the subtraction operation, it is00110When it is an or operation. whenAB=0Right nowAandBMutually

Wait, willZEROsignal set1output, otherwise set0.ZEROThe signal isBEQWhether the instruction jumps

One of the judgment signals for rotation.

4.5 EXT (Extension unit)

```
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
```

Data expansion unit. The unit input data is the low of the instruction16bit, according toctrl_encode_defThe merit in can be defined, whenEXTOpfor00, is the lower sixteen-bit extension, whenEXTOpfor01, sign-extended and left Shift two places, whenEXTOpfor10, is the high sixteen-bit extension. Finally, the expanded32bit data output.

4.6 DM(data memory)

```v
`include "global_def.v"
module dm( addr, din, DMWr, clk, dout, DMemR );

   input  [4:0] addr;
   input  [31:0] din;
   input         DMWr;
   input    DMemR;
   input         clk;

   output [31:0] dout;

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

   always @(posedge clk)
   begin
      if (DMWr)
         dmem[addr] <= din;
         `ifdef DEBUG
         $display("DEM[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X"
         $display("DEM[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X"
         $display("DEM[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X"
         $display("DEM[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X"
         $display("DEM[%4X]=%8X", addr, dmem[addr]);
      `endif
   end // end always

   assign dout = dmem[addr];

endmodule
```

The function of this unit is to read and write data to the memory. When the memory write signalDMWrfor1when, according to the inputaddr

(memory number) will write datadinWrite to the corresponding memory number, otherwise it will not be written to the memory. Should be saved

The memory read signal is1When, read the data in the corresponding unit according to the memory numberdoutMedium output.

4.7 IM (Instruction Memory)

```
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
```

According to the input instruction address addr, the corresponding instruction in the instruction memory is read out to the register dout.

Medium output.

4.8 Ctrl (Controller)

```
1   `include "ctrl_encode_def.v"
2
3   module Ctrl(jump,RegDst,Branch,DMemR,DMnoutregR,DMWr,RFWr,Alusrc,EXTOp,ALUO
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
```

According to the high six bits instr of the input instruction, the CASE statement determines the function of each instruction and sets the unit control signal of the corresponding function.

(1) When instr is 000000, it is an arithmetic operation instruction. Only the register write signal RFWr is 1, and other control signals are all 0. According to the FUNCT field, when it is 100001, the addition operation is performed, and when it is 100011, the subtraction operation is performed.

```
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) When instr is 101011, it is a memory write instruction. At this time, the memory write signal DMWr is 1, and the register

The device write-back address selection signal RegDst is 1 (that is, the write-back address is RT), and the ALU operand B selection signal

Alusrc is 1 (that is, the data to be expanded by the expansion unit is selected), ALUOp is addition, and EXT is the lower 16-bit extension.

All other signals are 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) When instr is 100011, it is a memory read instruction. The register write-back address signal RegDst is 1 (RT field is selected), the write-back register data selection signal DMnoutregR is 1 (the data read from the memory is selected as write-back data), the memory read signal DMemR is 1, and the register write signal RFWr is 1 , the selection signal Alusrc of ALU operand B is 1 (that is, selects the data to be expanded by the expansion unit), ALU is the addition operation, EXT is the low 16-bit expansion, and other signals are 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) When instr is 001101, it is an OR operation. At this time, the ALU is an OR operation, EXT is the lower 16-bit extension, the register writes RFWr to 1, and the ALU operand B selection signal Alusrc is 1 (that is, the data to be expanded by the expansion unit is selected)), the register write-back address RegDst signal is 1 (that is, the RT field is selected), and other signals are all 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) When instr is 000100, it is a BEQ branch instruction, the branch signal Branch is 1, the register write-back signal RFWr is 1, EXT performs sign extension and left shift operation of 2 bits, ALU performs subtraction operation, and other signals are all 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)wheninstrfor000010time, forJUMPUnconditional jump instruction, jump signal at this timejumpfor 1,becausejumpIt is an unconditional jump. At this time, other signals are irrelevant signals and will be flushed in the pipeline.

So I'm not changing their values here.

```
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]  imAdr;
    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;
```

```
81      PCjudge U_PCjudge( .PCWr(PCWr),.jump(jump),.PCreal(aaa),.bbb(bbb),.Imm32(extImm321),.instr(instr1));
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 imAdr = PC[6:2];
88
89      //IM
90      im U_im (
91          .addr(imAdr[4:0]) , .dout(instr)
92      );
93
```

```
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),.DMnoutregR(DMnoutregR)
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));//
129
```

The IFID pipeline register transfers data and signals from the IF stage to the ID stage. PC-PC1

Instr-instr1.

The instruction instr1 is divided into several fields: Op, Funct, rs, rt, exImm16, and rd.

The two data read from the register numbers rs and rt in RF are saved in RD1 and RD2. The register write signal is RFWr3 of WB level, the write-back address is rd3 of WB level, and the write-back data is writebackdata.

CTRL outputs corresponding control signals according to the input OP and FUNCT fields.

EXT extends the 16-bit extImm16 to the 32-bit extImm32 based on the input EXTOp.

The LW risk detection unit determines whether it needs to block and outputs the blocking signal block based on the input rt, rdx (RD of EX level), DMemR1 (DMemR of EX level)

The LW risk blocking signal control unit outputs the resistance of the PC, ID/EX, and IFID units according to the input blocking signal.

```
132     //
133     //ID-EX
134     //
135     IDEXreg U_IDEXreg(.clk(clk1),.rst(rst),.RegDstin(RegDst),.RegDstout(RegDst1)
136                 ,.Branchin(Branch),.Branchout(Branch1),.DMemRin(DMemR),.DMemRout(DMemR1),
137                 .DMnoutregRin(DMnoutregR),.DMnoutregRout(DMnoutregR1),.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),.IDEXzero(IDEXzero),.PCWr(PCWr)
142                 ,.RFout2in(RD2),.RFout1out(RD11),.RFout2out(RD22),
143                 .Imm32in(extImm32),.Imm32out(extImm321),.rtin(rt),.rtout(rt1),.rsin(rs),.rsout(rs1),.PCin(PC1),.PCout(PC2),.rdin(rd),.rdout(rd1));
144
145     assign rdx = (RegDst1==1)?rt1:rd1;
146     assign bbb=extImm321+PC2+4;
147     wire [31:0] alu_Blast;
148
149
150
151     //MUX three panglu
152     wire [31:0] alu_B1,RD111;
153     assign alu_B = (Alusrc1==1)?extImm321:RD22;
154     wire [1:0] numA,numB,numA1,numB1;
155
156     MUX1    U_MUX1( .clk(clk),.rs(rs1),.rt(rt1),.exmemrd(rd2),.RFWr(RFWr2),.numA(numA),.numB(numB));
157     MUX2    U_MUX2( .clk(clk),.rs(rs1),.rt(rt1),.memwbrd(rd3),.RFWr(RFWr3),.numA1(numA1),.numB1(numB1));
158     ALUjudge U_ALUjudge( .aluBin(alu_B),.aluBout(alu_B1),.aluAin(RD11),.aluAout(RD111),.numA(numA),.numB(numB),
159                 .numB1(numB1),.numA1(numA1),.aluex(alu_C2),.aluwb(alu_C3),.ALUsrc(Alusrc1));
160     MUX3    U_MUX3(.block(block2),.alu_Bin(alu_B1),.alu_Bout(alu_Blast),.DMout(DMdout));
161
162
163     //ALU
164     alu U_alu(.C(alu_C),.Zero(Zero),.A(RD111),.B(alu_Blast),.ALUOp(ALUOp1));
165     //Branch
166     assign PCWr = ((alu_C==0)&&(Branch1==1))?1:0;
```

① The IDEX register transfers ID level signals and data to EX. For the signals transferred to EX here, I directly added 1, PC1-PC2, RD1-RD11, RD2-RD22, extImm32-extImm321 after the name.

② （）？The statement is used to select the register write-back address and save it to rdx.

③ bbb=extImm321+PC2+4 is the calculation of BEQ jump address, and the data of the expansion unit is added to PC+4.

④ Line 153 is the first step of selecting ALU operand B, whether to select the expansion unit data or the data read out from the register RD2.

⑤ MUX1 and 2 in lines 156 and 157 are data risk detection units. Enter rs1, rt1 of EX level and rd2 of EX/MEM in 1. Enter rs1, rt1 of EX level and rd3 of MEM/WB level in RFWr2, 2. RFWr3. Then output the corresponding signal to ALUjudeg for use

⑥ ALUjudge selects the corresponding ALU operand A (RD111) and ALU operand B (alu_B1) based on the input risk judgment signal. The bypassed data is alu_C2 of EX/MEM and alu_C3 of MEM/WB.

⑦ MUX3 is used for ALU operand B selection of LW type data risk. Based on the input blocking signal block2 (WB level blocking signal), it determines whether blocking occurs and outputs the finalized alu_Blast. If blocking occurs, the MEM level memory is read. The output data DMdout is bypassed to the ALU.

⑧ The two operands of ALU are RD111 and alu_Blast, and the output result is alu_C

⑨ This statement in line 166 is used to determine whether the BEQ branch occurs. Originally, this statement should be judged at the MEM level, but because the previously executed statement will be discarded if a branch occurs at the MEM level, it is necessary to clear all the data in the first three pipeline registers to 0, which is costly. Secondly, because the previous instruction is the LW instruction, if the ID is reached in advance

it

```
170
171     //
172     //EX-MEM
173     //
174     EXMEMreg U_EXMEMreg(.clk(clk1), .rst(rst)
175                 ,.Branchin(Branch1),.Branchout(Branch2),.DMemRin(DMemR1),.DMemRout(DMemR2),
176                 .DMnoutregRin(DMnoutregR1),.DMnoutregRout(DMnoutregR2),.DMWrin(DMWr1),.DMWrout(DMWr2),
177                 .RFWrin(RFWr1),.RFWrout(RFWr2),.bbbin(bbb),.bbbout(bbb2),.zeroin(Zero),.zeroout(Zero2),.RFout2in(RD22),.RFout2out(RD222),
178                 .wbaddrin(rdx),.wbaddrout(rd2),.aluCin(alu_C),.aluCout(alu_C2),.blockin(block),.blockout(block1));
179
180
181     //DM
182     assign dm_addr = alu_C2[4:0];
183     dm U_dm(.dout(DMdout),.addr(dm_addr),.din(RD222),.DMWr(DMWr2),.DMemR(DMemR2),.clk(clk));
184
185
```

```
186
187     //
188     //MEM-WB
189     //
190     MEMWBreg U_MEMWBreg(.clk(clk1),.rst(rst),.DMnoutregRin(DMnoutregR2),.blockin(block1),.blockout(block2),
191    .DMnoutregRout(DMnoutregR3),.aluCin(alu_C2),.aluCout(alu_C3),.dmoutin(DMdout),
192    .dmoutout(DMdout2),.wbaddrin(rd2),.wbaddrout(rd3),.RFWrin(RFWr2),.RFWrout(RFWr3));
193
194
195
196     assign writebackdata = (DMnoutregR3==1)?DMdout2:alu_C3;
197
```

① The MEMWB register transfers MEM level signals and data to the WB level, DMdout-DMdout2, alu_C2-alu_C3, block1-block2, rd2-rd3, DMnoutregR2-DMnout regR3.

② The judgment selection statement in line 196 is used to select whether the final writeback data writebackdata is the ALU calculation result or the memory read data based on the selection signal DMnoutregR3.

4.10 Pipeline register (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
endmodule
```

This register is used to store IF level PC (PCin, PCout), instructions

instr(instrin,instrout).

When the IFID register blocking signal IFIDzero is 0, the value of the IFID register cannot be modified at this time.

Keep the data from the previous period unchanged.

When the IFID register blocking signal IFIDzero is 1 and the BEQ branch instruction clears 0, the signal PCWr is

When 1, it means that a branch jump has occurred. At this time, all data in the IFID register should be cleared to 0.

When the IFID register blocking signal IFIDzero is 1 and the IFID register writing signal IFIDWr is 1,

When PCWr is 0, normal data transfer is performed.

and then transferred to the ID level.

```
1
2   module IDEXreg(clk, rst,RegDstin,RegDstout,Branchin,Branchout,DMemRin,DMemRout,DMnoutregRin,DMnoutregRout,
3                  DMWrin,DMWrout,RFWrin,RFWrout,Alusrcin,Alusrcout,
4                 EXTOpin,EXTOpout,ALUOpin,ALUOpout,RFout1in,RFout2in,RFout1out,RFout2out,
5                 Imm32in,Imm32out,rtin,rtout,rsin,rsout,rdin,rdout,PCin,PCout,IDEXzero,PCWr);
6
7      input          clk,IDEXzero,PCWr;
8      input          rst;
9      // input          IDEXWr;
10     input          RegDstin;
11     output  reg        RegDstout;
12     input          Branchin;
13     output   reg      Branchout;
14     input           DMemRin;
15     output   reg       DMemRout;
16     input           DMnoutregRin;
17     output  reg      DMnoutregRout;
18     input       DMWrin;
19     output  reg        DMWrout;
20     input             RFWrin;
21     output   reg       RFWrout;
22     input           Alusrcin;
23     output     reg      Alusrcout;
24     input [1:0]      EXTOpin;
25     output reg  [1:0]         EXTOpout;
26     input [4:0]       ALUOpin;
27     output reg [4:0]       ALUOpout;
28     input[31:0]        RFout1in,RFout2in;
29     output reg [31:0]       RFout1out,RFout2out;
30     input[31:0]                Imm32in;
31     output reg [31:0]              Imm32out;
32     input [4:0] rtin,rdin,rsin;
33     output reg [4:0] rtout,rdout,rsout;
34     input  [31:0] PCin;
35     output reg [31:0] PCout;
36
```

```
35    output reg [31:0] PCout;
36
37    always @(posedge clk or posedge rst) begin
38      if(IDEXzero==1)
39        begin
40       RegDstout=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=rsin;
53       rtout=rtin;
54       rdout=rdin;
55       PCout=PCin;
56      end
57
58    if(IDEXzero==0) begin
59       RegDstout=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
```

```
69
70
71     if(PCWr==1) begin
72        RegDstout=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
```

The input data of this register is the control signal from the CTRL unit and the expansion unit expansion data, indicating

Let the three fields RS, RT, RD, the two RD1, RD2 data read by the register, and PC. three

An IF statement is used to determine whether the LW signal is blocked and the branch signal is cleared.

When PCWr is 1, it means that a branch jump occurs, and the control signal of the IDEX level must be cleared to 0.

When IFIDzero is 0, it means that an LW data hazard occurs and the IDEX register needs to be blocked.

When PCWr is 0 and IFIDzero is 1, this register performs normal data transmission.

Then transfer to EX level.

```
module EXMEMreg(clk, rst
               ,Branchin,Branchout,DMemRin,DMemRout,
               DMnoutregRin,DMnoutregRout,DMWrin,DMWrout,
               RFWrin,RFWrout,bbbin,bbbout,zeroin,zeroout,RFout2in,RFout2out,
               wbaddrin,wbaddrout,aluCin,aluCout,blockin,blockout);
   input         clk,blockin;
   input         rst;
   // input         IDEXWr;
   input        Branchin;
   output   reg     Branchout;
   input         DMemRin;
   output   reg      DMemRout;
   input          DMnoutregRin;
   output  reg     DMnoutregRout;
   input       DMWrin;
   output  reg       DMWrout,blockout;
   input            RFWrin;
   output   reg      RFWrout;
   input[31:0]       RFout2in;
   output reg [31:0]    RFout2out;
   input [4:0] wbaddrin;
   output reg [4:0] wbaddrout;
   input[31:0]       bbbin;
   output reg [31:0]    bbbout;
   input[31:0]       aluCin;
   output reg [31:0]    aluCout;
   input          zeroin;
   output   reg     zeroout;
   always @(posedge clk or posedge rst) begin
      Branchout=Branchin;
      DMemRout=DMemRin;
      DMnoutregRout=DMnoutregRin;
      DMWrout=DMWrin;
      RFWrout=RFWrin;
      RFout2out=RFout2in;
      wbaddrout=wbaddrin;
       blockout=blockin;
       bbbout=bbbin;
       zeroout=zeroin;
       aluCout=aluCin;
    end // end always
endmodule
```

This register is used to save the branch signal Branchin from the EX stage, the memory read signal DMemRin,

Write back data selection signal DMnoutregRin, memory write signal DMWrin, register write signal

RFWrin, register output data 2RFout2in, register write back data address wbaddrout,

Blocking signal blockin, ALU zero flag zeroin, ALU calculation output result C aluCin, BEQ

Jump address bbbin. and then transferred to the MEM level.

```
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                                /mips_tb/U_mips/U_MEMWBreg/wbaddrout
11       input [31:0] aluCin;     00100
12       output reg [31:0] aluCout;
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 ??0 ???1 ????????4? ??????????????????????????1?????  ??????? ??MIPS??
26  endmodule
27
```

```
 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 [1:0] 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
```

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

```
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
```

For road use, in fact the assignment has no effect)

4.17 ALU result selection (ALUjudege)

```
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
```

the corresponding value.

When numB is 10, EX/MEM level data needs to be bypassed to ALU operand B, at this time aluBout=aluex.

When numB is 00, there is no need to observe aluBout=aluBin at this time.

When numB1 is 01, the MEM/WB level data needs to be bypassed to ALU operand B in advance, at this time aluBout=aluwb.

When numA is 10, EX/MEM level data needs to be bypassed to ALU operand A, at this time aluAout=aluex.

When numA is 00, there is no need to observe that aluAout=aluAin.

When numA1 is 01, the MEM/WB level data needs to be bypassed to ALU operand A in advance, at this time aluAout=aluwb.

4.18 LW adventure detection unit (Check)

```
module Check(rt,rdx,block,DMemR,clk,rst);
 output reg block;
 input rt,rdx,DMemR,clk,rst;
   always @(*) begin
     if((rt==rdx)&&DMemR==1)
       assign block=1'b1;
    else begin
       assign block=1'b0;
    end
 end
endmodule
```

```
module HD(block,PCrun,IDEXzero,clk,rst,IFIDzero);
 output reg PCrun;
 output reg IDEXzero,IFIDzero;
 input block,clk,rst;
 initial begin
 PCrun=1'b1;
 IDEXzero=1'b1;
 IFIDzero=1'b1;
 end
   always @(*) begin
      if(block==1'b0)
        begin
          assign PCrun=1'b1;
          assign IDEXzero=1'b1;
          assign IFIDzero=1'b1;
        end
   if(block==1'b1)
    begin
      assign PCrun=1'b0;
      assign IDEXzero=1'b0;
      assign IFIDzero=1'b0;
    end
   end // end always

endmodule
```

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

5.1 Test files

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 Testing machine code

```
341d000c
34021234
34033456
00432021
00643023
ac020000
ac030004
afa40004
8c050000
10450001
8fa30004
8c050004
1065fffd
00c23023
```

5.3 Test result analysis

5.3.1 ori $29, $0, 12 command

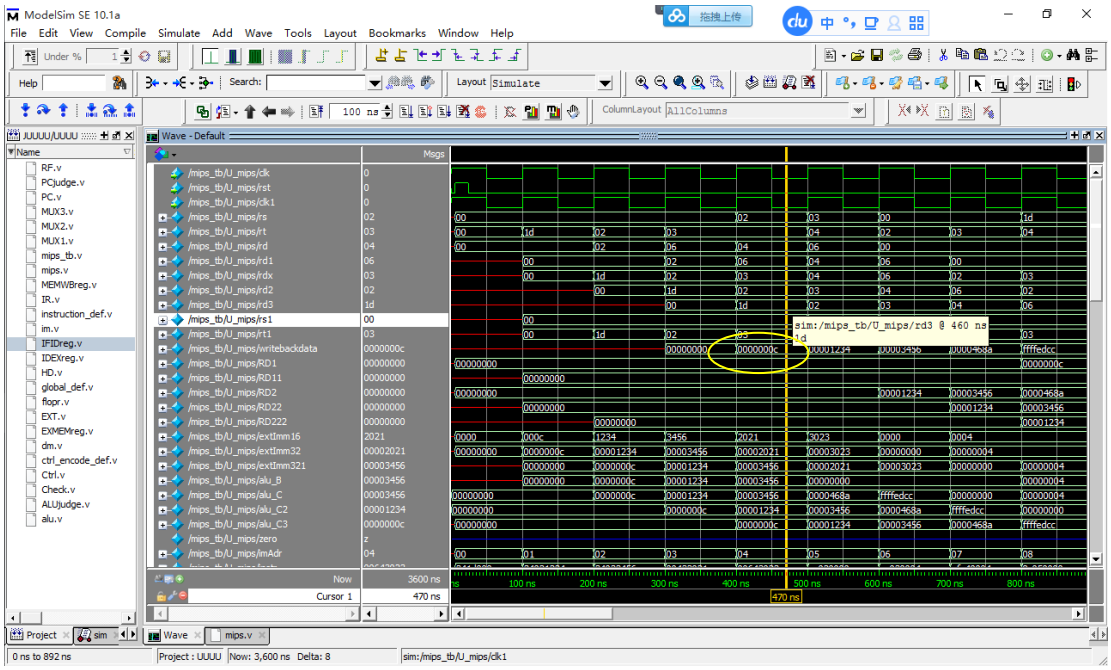
The simulation result of this instruction is shown in the figure below. This instruction means that $0Registers and constants12Do "or"

Operation, the result is put into $29number register. The simulation results show that the instruction is located inPCThe register address is

00003000(hexadecimal), the machine code of this instruction is341d000c(hex) write back register

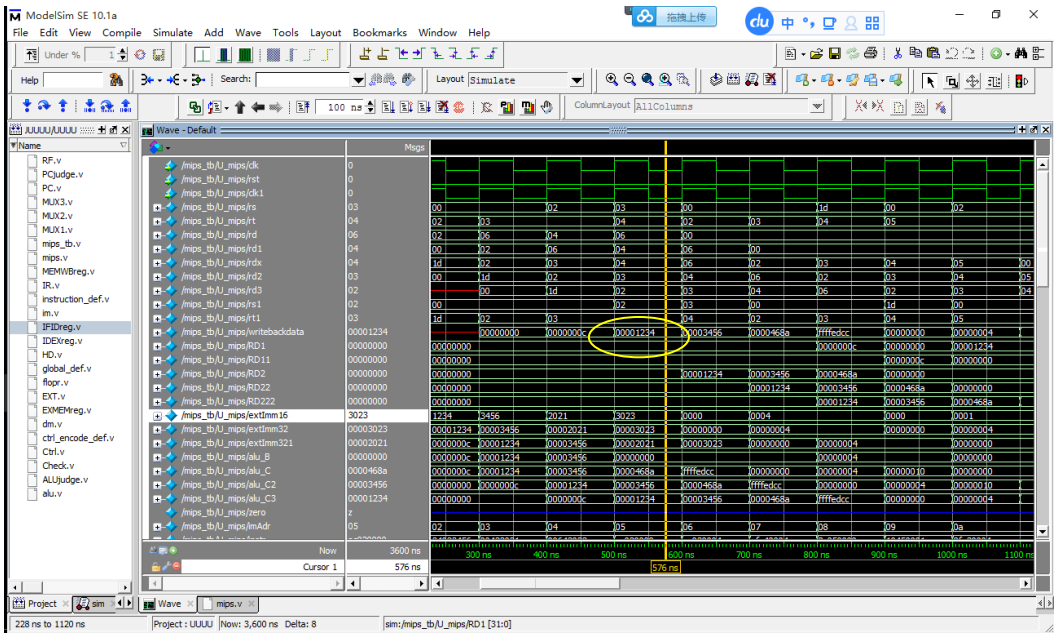
addressrd(rd3) the register number is also1d(Decimal is29), the data written in the register (RWD)

yes0000000c, which is consistent with the expected result of the instruction, indicating that the simulation result of the instruction is correct.



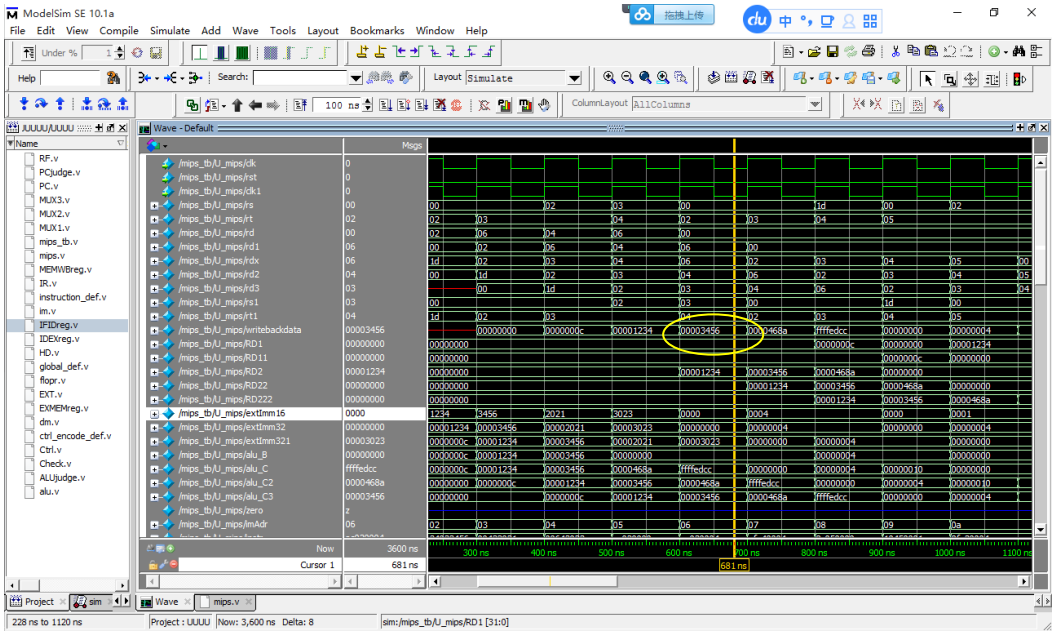
5.3.2 ori $2, $0, 0x1234 command

The simulation result of this instruction is shown in the figure below. This instruction means to perform an "OR" operation on the $0 register and the constant 0001001000110100 (binary), and put the result into the $2 register. The simulation results show that the PC register address where the instruction is located is 00003004 (hexadecimal), the machine code of the instruction is 34021234 (hexadecimal), the write-back address rd (rd3) register number is 02, and the register written The data (RWD) is 00001234, which is consistent with the expected result of the instruction, indicating that the simulation result of the instruction is correct.



5.3.3 ori $3, $0, 0x3456 command

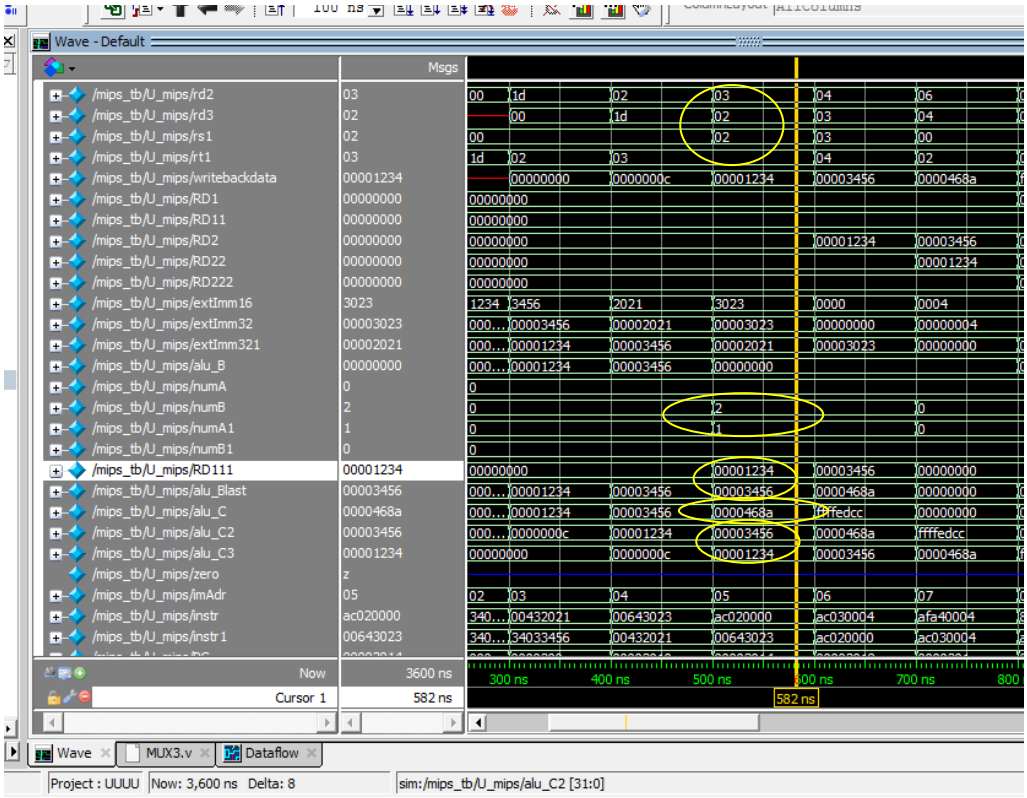
The simulation result of this instruction is shown in the figure below. This instruction means performing an "OR" operation on the $0 register and the constant 3456 (hexadecimal), and the result is placed in the $3 register. The simulation results show that the PC register address where the instruction is located is 00003008 (hexadecimal), the machine code of the instruction is 34033456 (hexadecimal), the write-back address rd (rd3) register number is 03, and the register written The data (RWD) is 00003456, which is consistent with the expected result of the instruction, indicating that the simulation result of the instruction is correct.



5.3.4 addu $4, $2, $3 command

The simulation result of this instruction is shown in the figure below. This instruction means that $2Register with $3Registers perform "addition" Operation, the result is put into $4number register.

① There is a data hazard in this instructionrs1=rd3,rt1=rd2. risk judgment signal at this time numA1=01,numB=10. need toEX/MEMClassalu_C2Bypass in advanceALUoperandB (alu_Blast)for00003456,WillMEM/WBClassalu_C3Bypass toALUoperandA (RD111)for00001234. CalculateALUresultalu_C=000468a. According to the simulation resultsEX level, the process is correct.

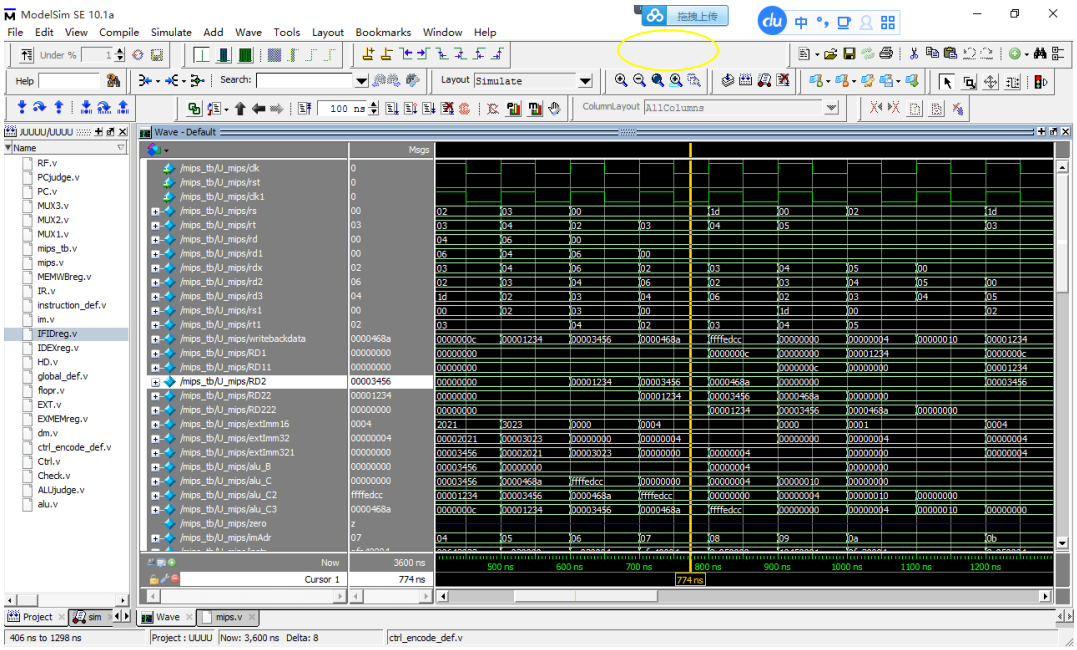


② The simulation results show that the instruction is located inPCThe register address is0000300c(hexadecimal), which refers to

The machine code of the command is00432021(hexadecimal), the result of adding the two is0000468a(hexadecimal).

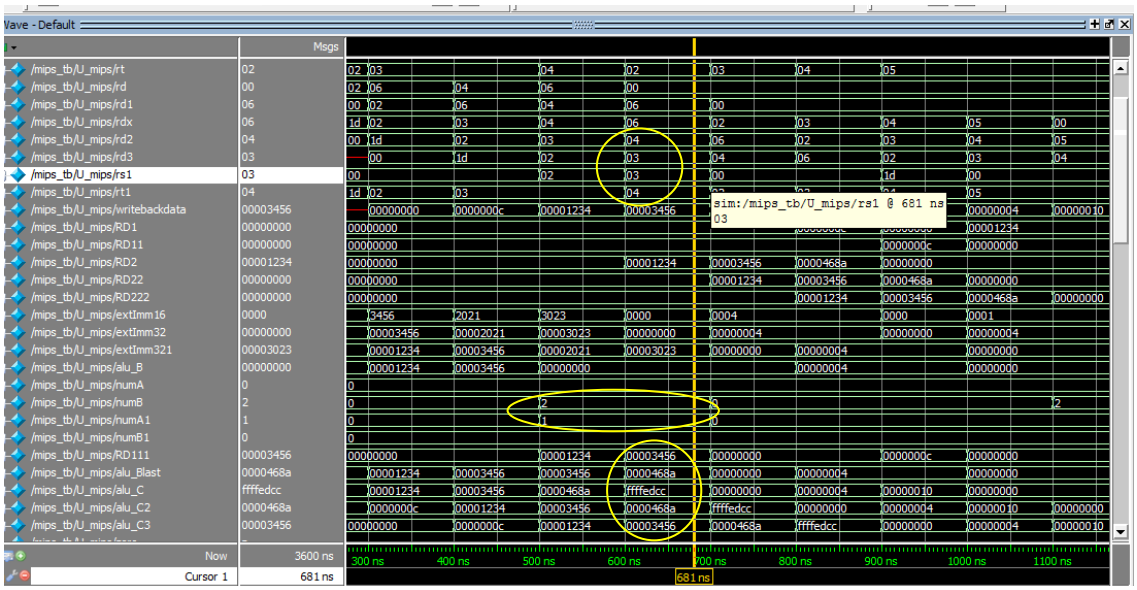
existWBThe data written to the stage register (RWD)Too0000468a, which is consistent with the expected result of the instruction

of , explain bright Should refer to make imitation real Knot fruit just correct



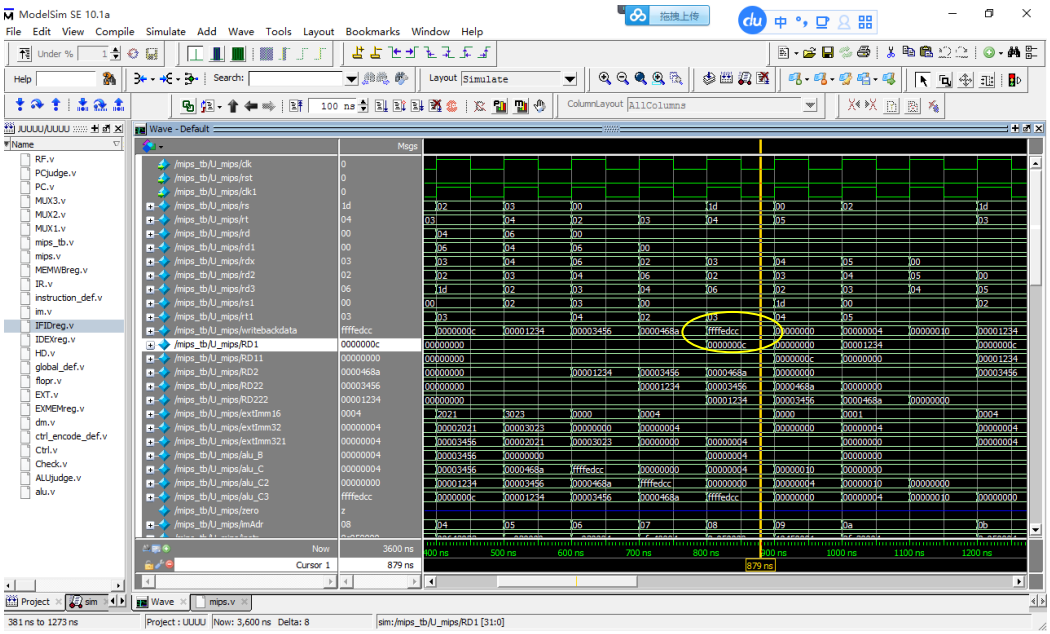
5.3.5 subu $6, $3, $4 command

① There is a data hazard in this instructionrs1=rd3,rt1=rd2. risk judgment signal at this time numA1=01,numB=10. need toEX/MEMClassalu_C2Bypass in advanceALUoperandB (alu_Blast)for0000468a,WillMEM/WBClassalu_C3Bypass toALUoperandA



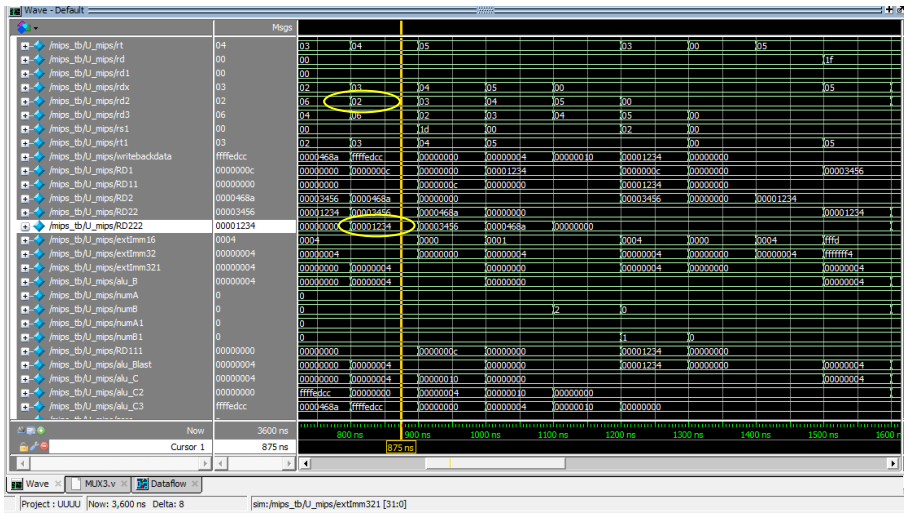
② The simulation results show that the instruction is located inPCThe register address is00003010(hexadecimal), The machine code of this instruction is00643023(hexadecimal), the result of subtracting the two isffffedcc(ten hexadecimal). existWBThe data written back by the stage register (RWD)Tooffffedcc, with instructions

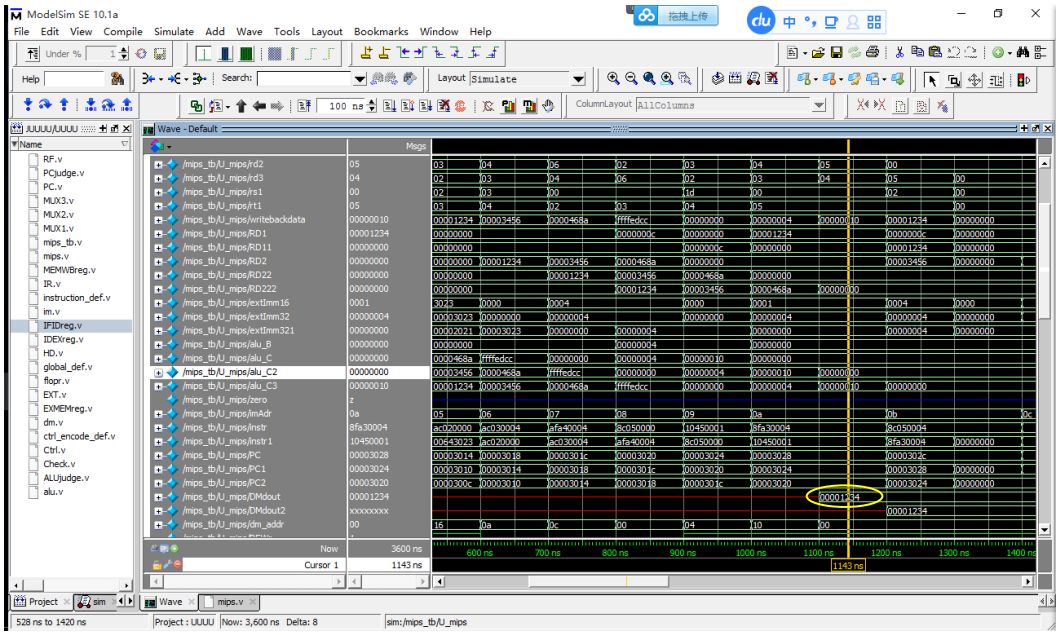
The expected results are consistent, indicating that the instruction simulation results are correct.



5.3.6 sw $2, 0($0) command

The simulation result of this instruction is shown in the figure below. This instruction means that $2Register deposit $0+0memory card Yuanzhong, $0The register value is always0, so the calculated address is0, that is: put $2register deposit0 memory unit. The simulation results show that the instruction is located inPCThe register address is00003014(Sixteen system), the machine code of this instruction isac02000(hexadecimal),MEMclassrd(rd2)Register number yes02(Decimal is2),RD222The output is00001234(hexadecimal), that is, the previous , that is, the writing address is





MEMRead out the data and display the finaldm_doutThe value is00001234(hex) with the command

The expected results are consistent, indicating that the instruction simulation results are correct.

5.3.7 sw $3, 4($0) command

The simulation result of this instruction is shown in the figure below. This instruction means that $3Register deposit $0+4in the memory unit,

$0The register value is always0, so the calculated address is4, that is: put $3register deposit4No. memory single

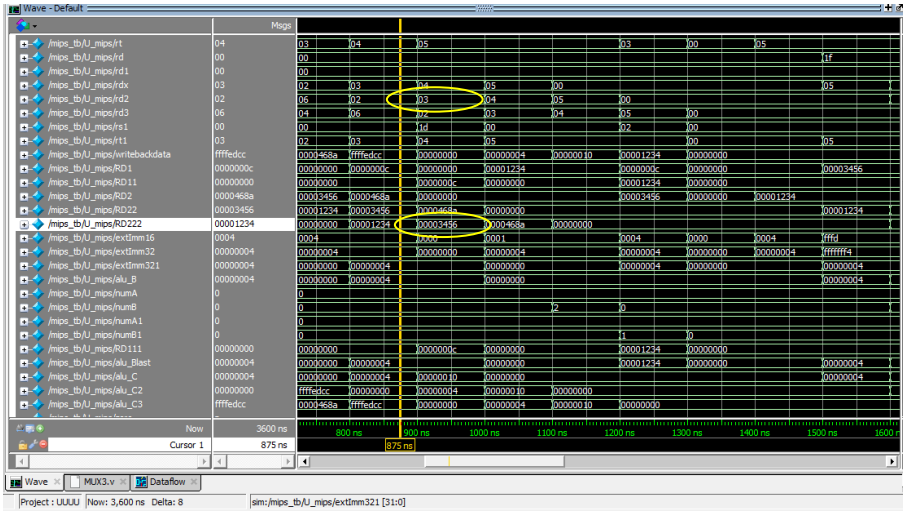
Yuan. The simulation results show that the instruction is located inPCThe register address is00003018(hex), the

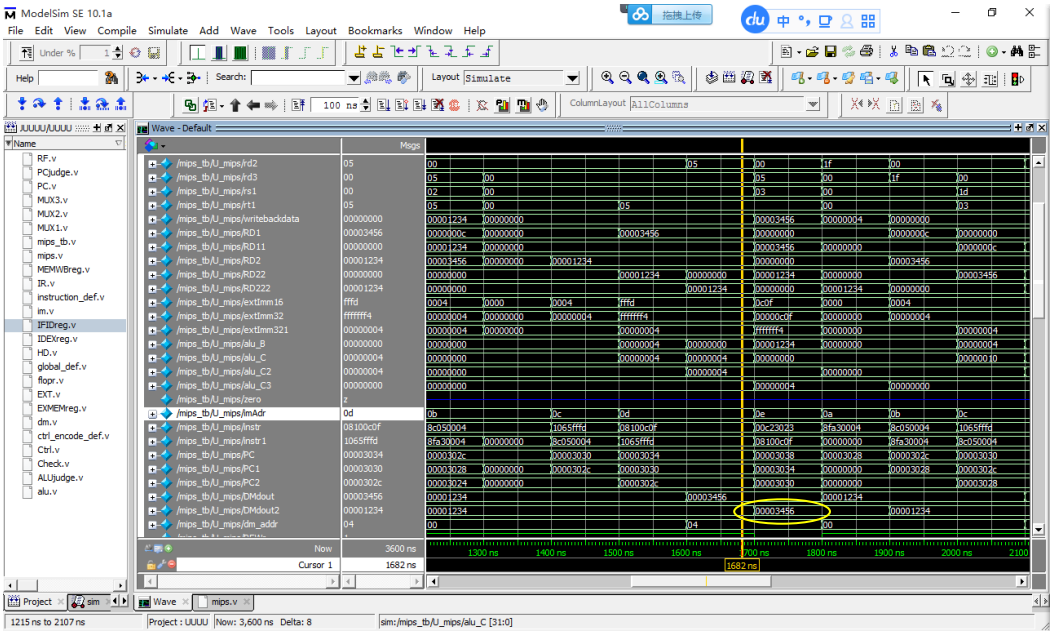
The machine code of the instruction isac030004(hexadecimal),MEMclassrd(rd2)The register number is03(ten

The base system is3),RD222The output is00003456(hexadecimal), that is, after the previous command is executed

$3The contents of the register, the write address of the memorydm_addrThe value is4, that is, the writing address is4, write

The data isRD222The output value of00003456.





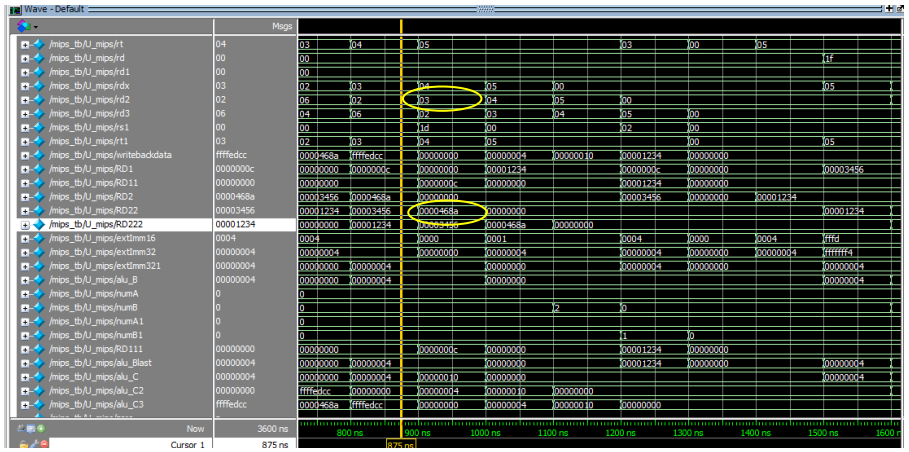
MEMRead out the data and display the finaldm_doutThe value is00003456(hex) with the command

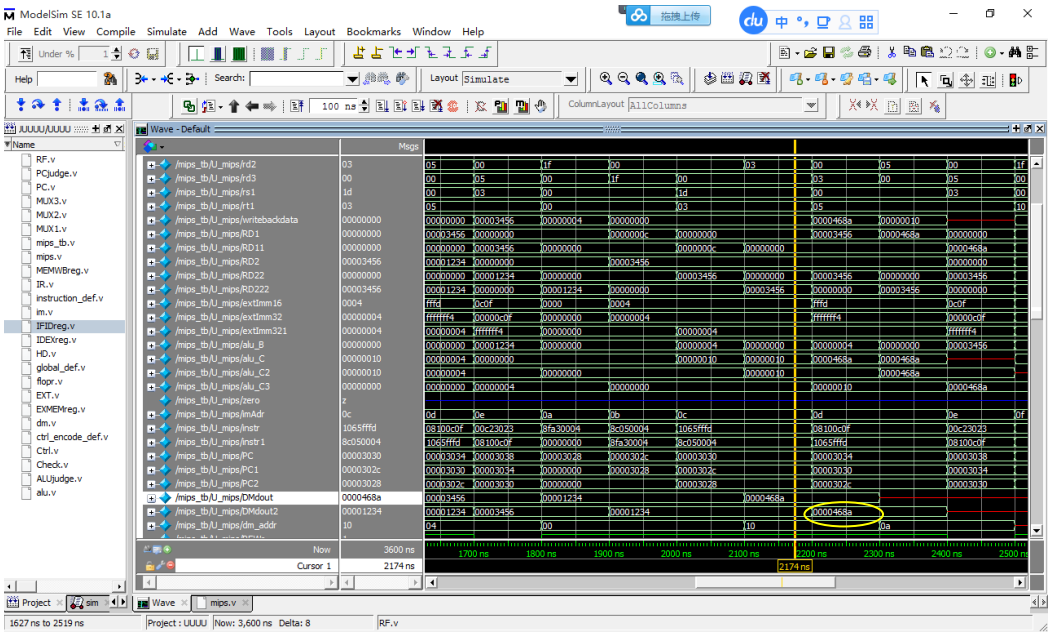
The expected results are consistent, indicating that the instruction simulation results are correct.

5.3.8 sw $4, 4($29) command

The simulation result of this instruction is shown in the figure below. This instruction means to store the $4 register into the memory unit of $29+4. The value of the $29 register is always 0c, so the calculated address is 10, that is: store the $4 register into 10 memory unit. The simulation results show that the PC register address where the instruction is located is 0000301c (hexadecimal), the machine code of the instruction is afa40004 (hexadecimal), and the MEM level rd (rd2) register number is 04 (decimal is 4). The output of RD222 is 0000468a (hexadecimal), which is the previous command

The writing address is





MEMRead out the data and display the finaldm_doutThe value is0000468a(hex) with the command

The expected results are consistent, indicating that the instruction simulation results are correct.

5.3.9 lw $5, 0($0) command

The simulation result of this instruction is shown in the figure below. This instruction means that $0+0The data in the memory unit is fetched

$5Register, $0The register value is always0, so the calculated address is0, that is: will0memory unit

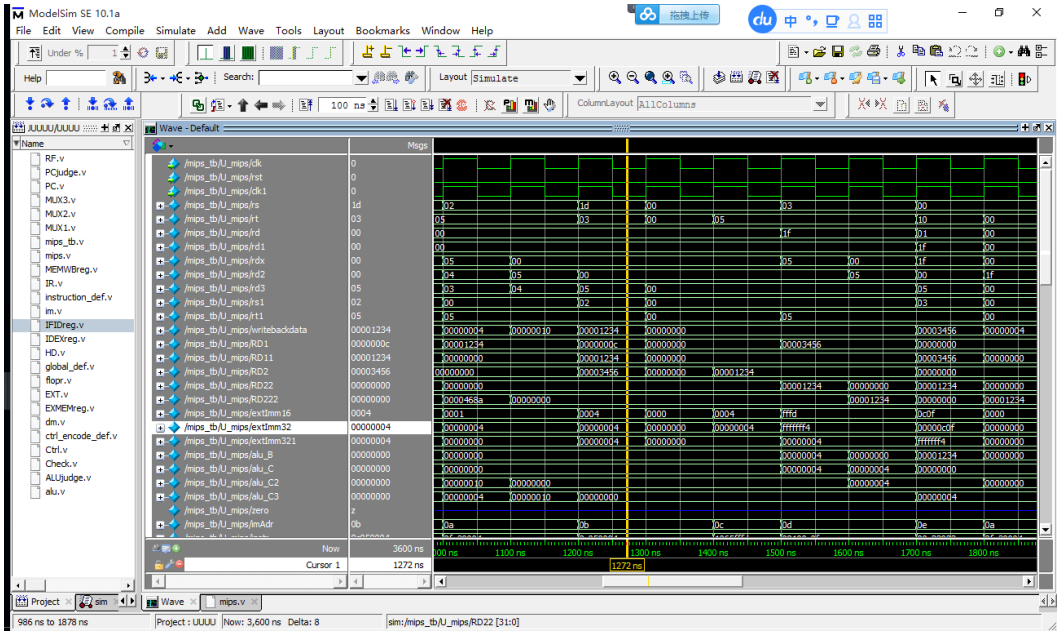
Get the value of $5register. The simulation results show that the instruction is located inPCThe register address is00003020

(hexadecimal), the machine code of this instruction is8c05000(hexadecimal),WBclassrd(rd3) deposit

The device number is05(Decimal is5), the output value of the memorydm_doutThe value is00001234(sixteen

base), that is, the value stored in the previous instruction, the data written in the registerRWDyes00001234(ten

hexadecimal), which is consistent with the expected result of the instruction, indicating that the simulation result of the instruction is correct.



5.3.10 beq $2, $5, _lb2 command

The simulation result of this instruction is shown in the figure below. This instruction means that $2Register with $5Register is "subtracted"

operation, the result is not saved, and is set byZeromark to decide whether to transfer. ifZero=1It means $2send

Register and $5Register data is equal, transferred to_lb2label, otherwise it will not be transferred.

existBEQThe instruction is executed toIDdiscoveredIDClassrtandEXClassrdx(that isrd)equal (whereas the previous instruction

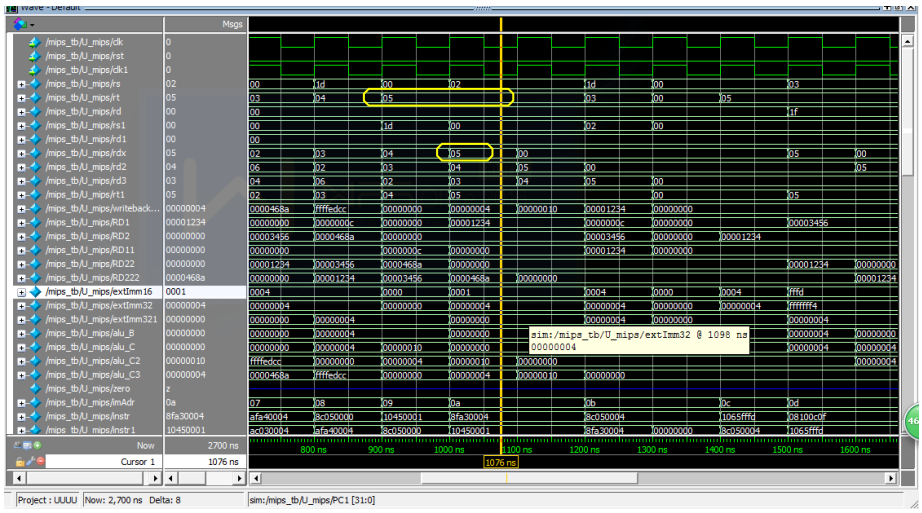
of $5The data has not been loaded in, and due to theMEMstage cannot be bypassed directly) occursLWModel number

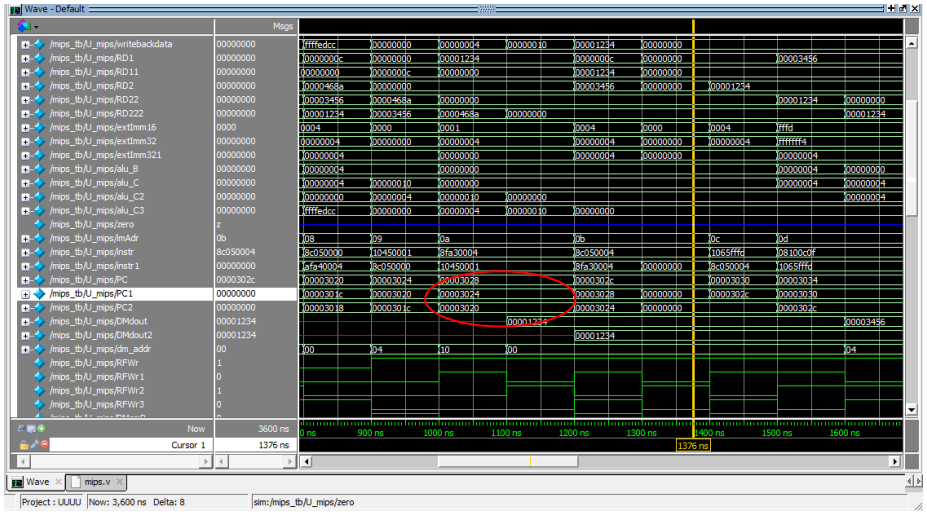
It is risky, so the signal is blockedblockbecome1,IFIDzero,IDEXzero,PCrunfor0.IF/ID,ID/EX

Register andPCNot writable, blocks for one clock cycle.

LWadventure:

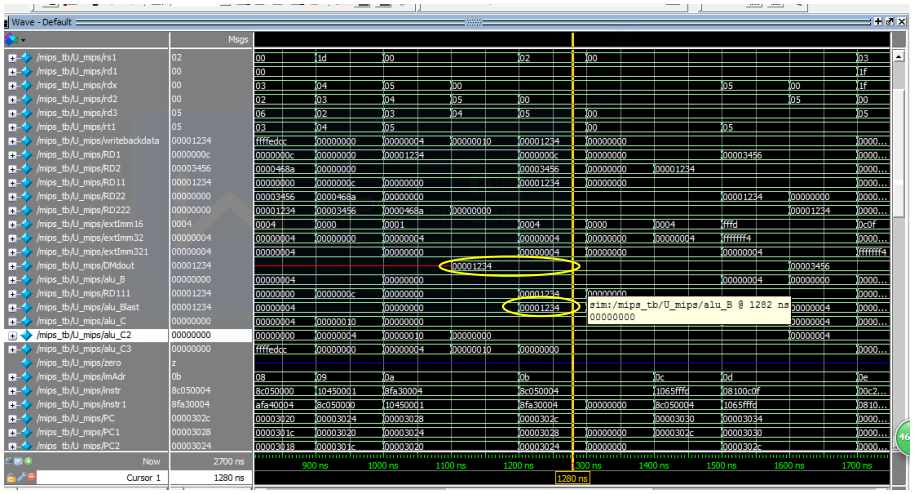
① The picture below isIDLevel hazard detection unit detects that two register numbers are equal



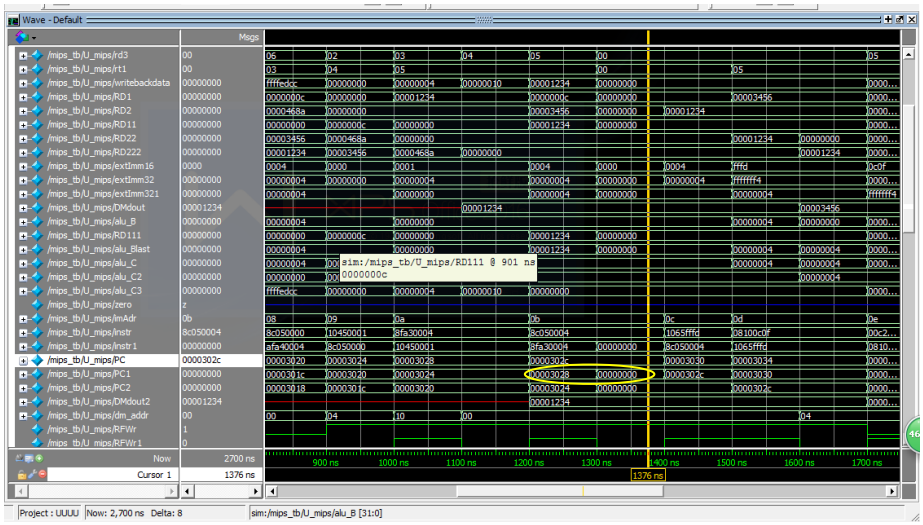


Controlling (branching) adventures:③The BEQ instruction determines whether to branch at the EX level, and converts the data at the MEM level

The EX stage of BEQ is bypassed as shown below and the data DMdout read by MEM is bypassed to alu_Blast.



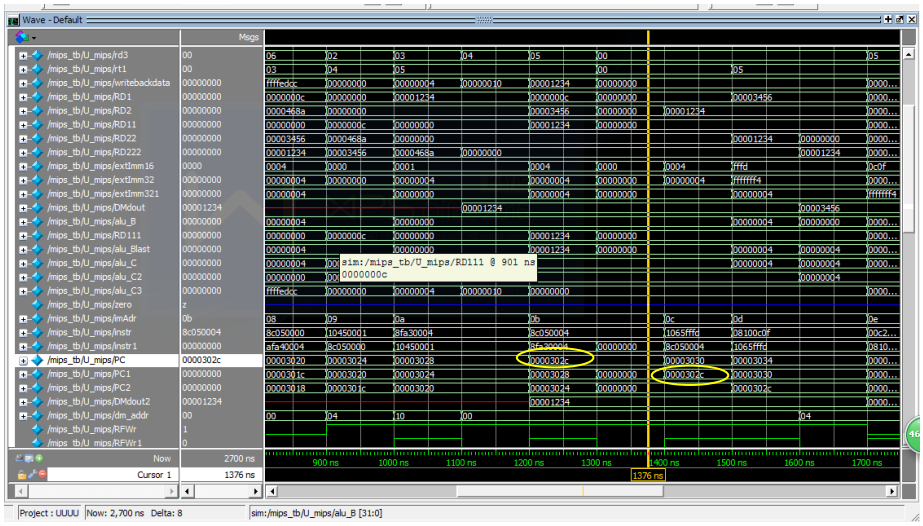
abandonedlw$3,



⑤ The simulation results show that the instruction is located inPCThe register address is00003024(hexadecimal), which refers to

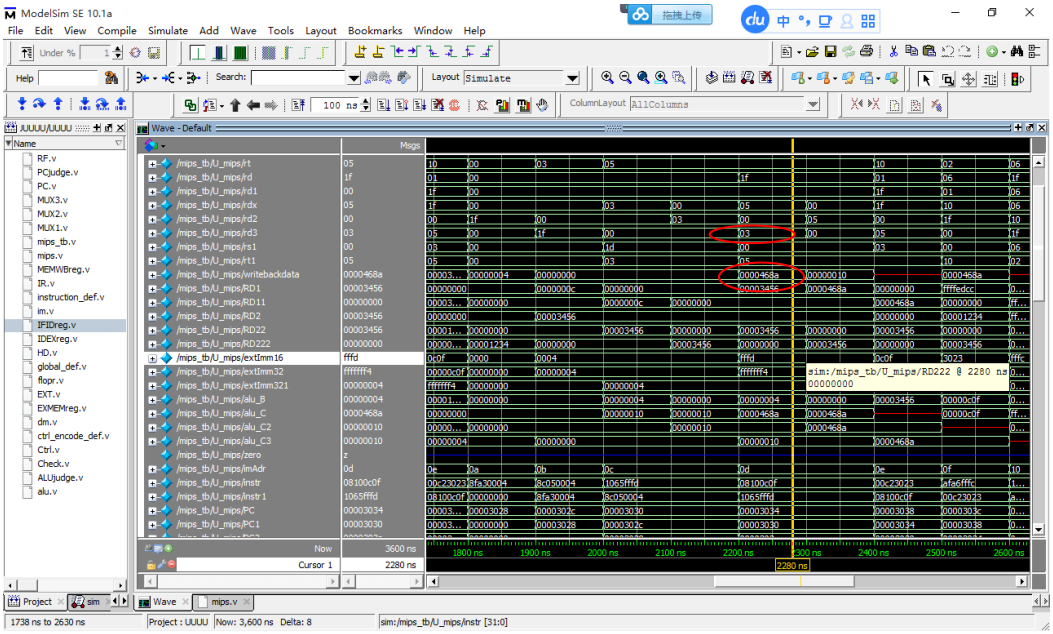
The machine code of the command is10450001(hex), according to the test file the program should move to the followinglw$5,

, describing the instruction



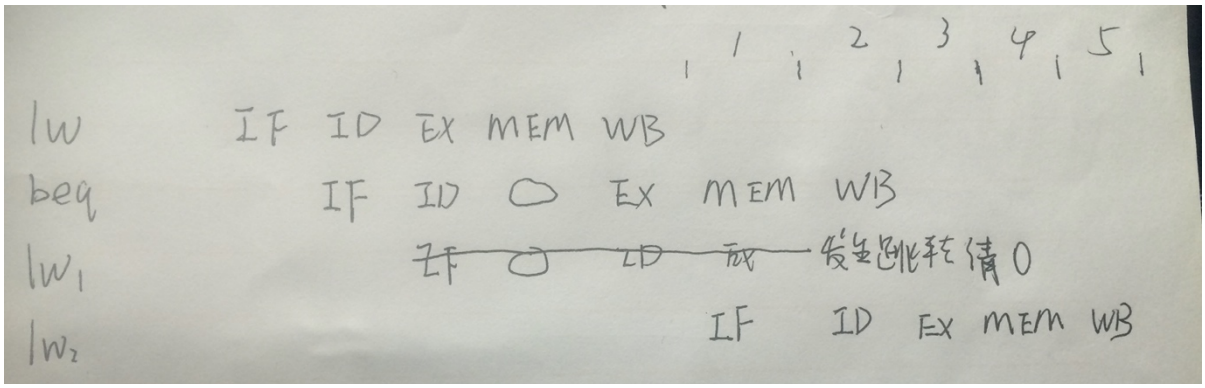
5.3.11 lw $3, 4($29) command

The simulation result of this instruction is shown in the figure below. This instruction means to fetch the data in the memory unit $29+4 to the $3 register. The value of the $29 register is always 0c, so the calculated address is 10, that is: memory unit No. 10 The value of is taken to the $3 register. The simulation results show that the PC register address where the instruction is located is 00003028 (hexadecimal), the machine code of the instruction is 8fa30004 (hexadecimal), and the WB level rd (rd3) register number is 03 (decimal is 3). The value of the output value dm_dout of the memory is 0000468a (hexadecimal), which is the value stored in the previous instruction. The write data RWD of the register is 0000468a (hexadecimal), which is consistent with the expected result of the instruction, indicating that The simulation result of this command is correct.

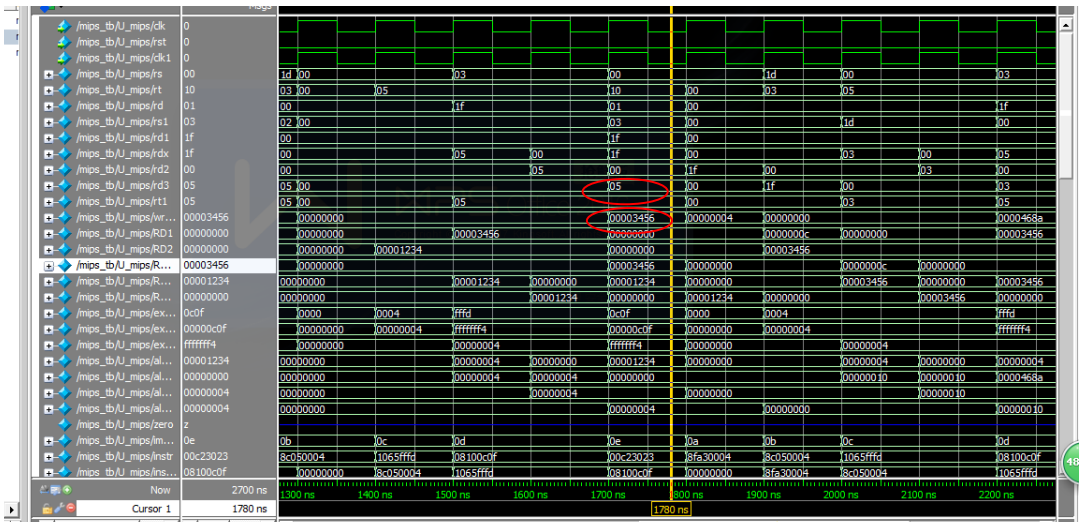


5.3.12 lw $5, 4($0) command

Due to LW data risk blocking and branch risk in BEQ, the result of LW write back will be delayed by four clock cycles compared to the previous LW instruction.



The simulation result of this instruction is shown in the figure below. This instruction means to fetch the data in the memory unit $0+4 to the $5 register. The value of the $0 register is always 0, so the calculated address is 4, that is: memory unit No. 4 The value of is taken into the $5 register. The simulation results show that the PC register address where the instruction is located is 0000302c (hexadecimal), the machine code of the instruction is 8c050004 (hexadecimal), and the WB level rd (rd3) register



5.3.13 beq $3, $5,_lb1 command

The simulation result of this instruction is shown in the figure below. This instruction means that $3Register with $5Register performs "subtraction" operation

Calculate, the result is not saved, set byZeromark to decide whether to transfer. ifZero=1It means $3deposit

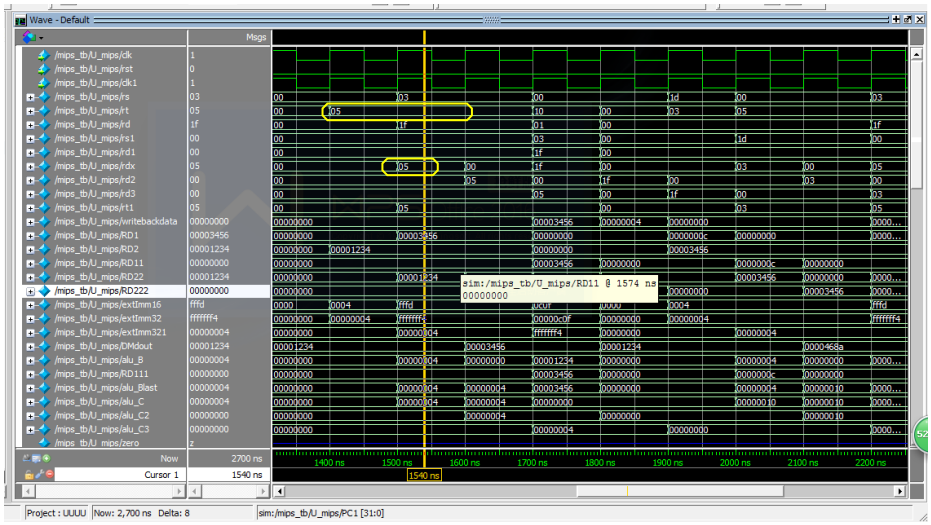
Device and $5Register data is equal, transferred to_lb1label, otherwise it will not be transferred.

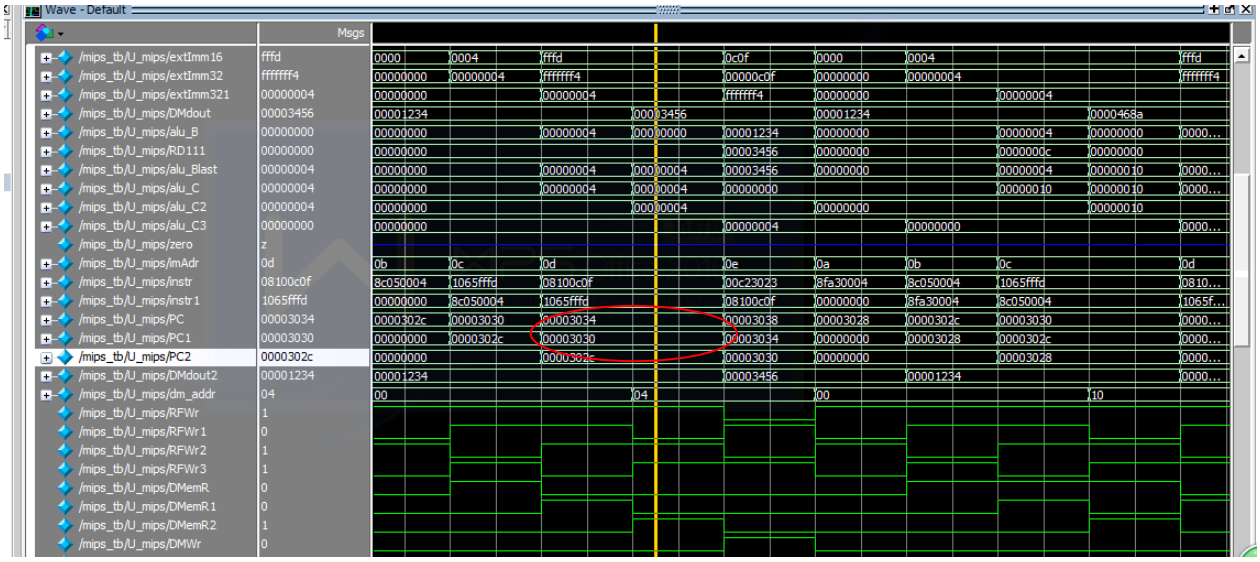
existBEQThe instruction is executed toIDdiscoveredIDClassrtandEXClassrdx(that isrd)equal (whereas the previous instruction

of $5The data has not been loaded in, and due to theMEMstage cannot be bypassed directly) occursLWModel number

It is risky, so the signal is blockedblockbecome1,IFIDzero,IDEXzero,PCrunfor0.IF/ID,ID/EX

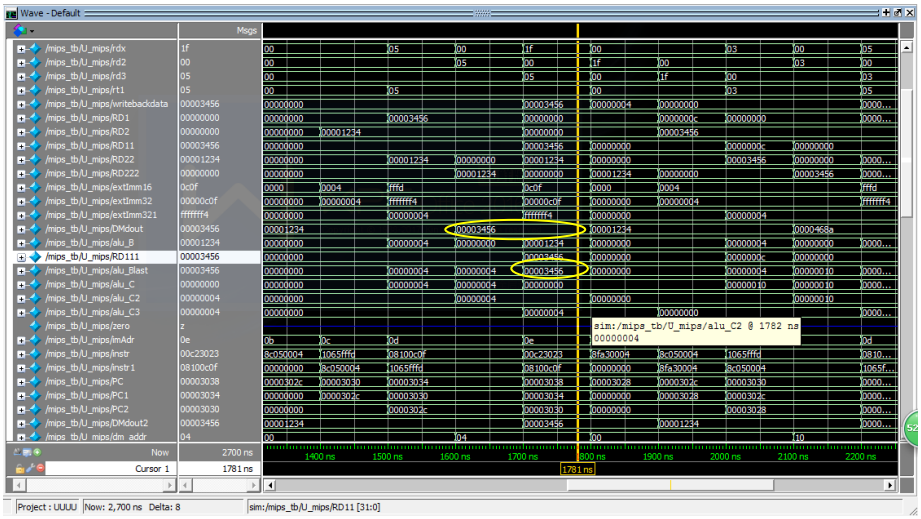
Register andPCNot writable, blocks for one clock cycle.



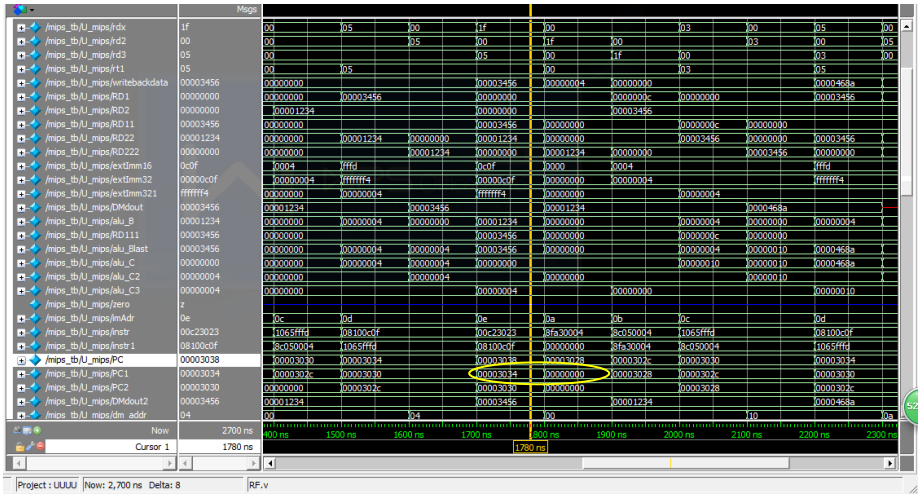


WillMEMread dataDMdoutBypass toalu_Blastfor00003456, the two are subtracted to0Indicates branch development

born.



According to discard the next finger

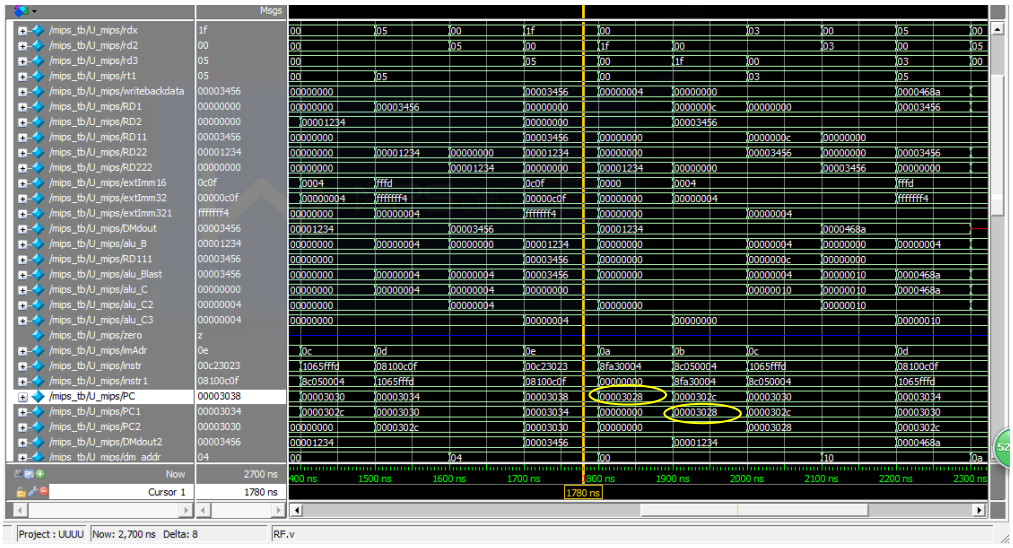


⑤ The simulation results show that the instruction is located inPCThe register address is00003030(hexadecimal), which refers to

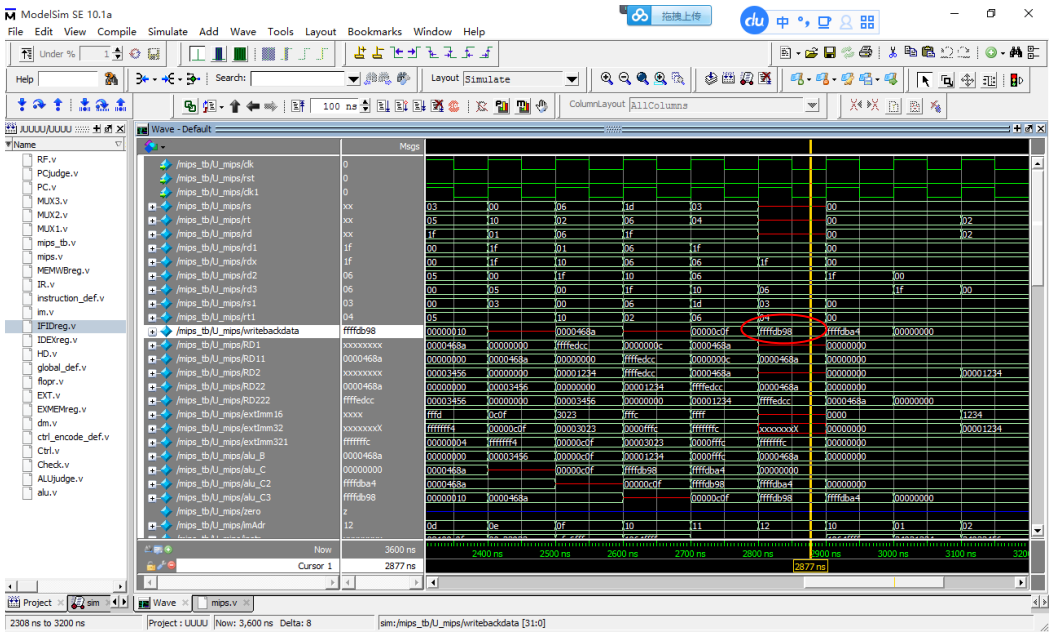
The machine code of the command is1065fffd(hex), according to the test file the program should transfer to the abovelw$3,

4($29)instruction, whose instruction address is00003028, which is consistent with the expected result of the instruction, indicating that the instruction

The simulation results are correct. As shown below



5.3.14 subu $6, $6, $2 command



The simulation result of this instruction is shown in the figure above. This instruction means that $6Register with $2Registers perform "subtraction"

Operation, the result is put into $6number register. The simulation results show that the instruction is located inPCThe register address is

00003038(hexadecimal), the machine code of this instruction is00c23023(hex), subtract the two

The result isffffdb98(hexadecimal). existWBThe data written back by the stage register (RWD)Tooffffdb98,

It is consistent with the expected result of the instruction, indicating that the simulation result of the instruction is correct.