

# Verilog流水线CPU设计文档

## 一、CPU设计方案综述

### (一) 总体设计概述

使用Verilog开发一个简单的流水线CPU，总体概述如下：

1. 此CPU为32位CPU
2. 此CPU为流水线设计
3. 此CPU支持的指令集为：  
mips-c指令集所有指令
4. add, sub不支持溢出

### (二) 关键模块定义

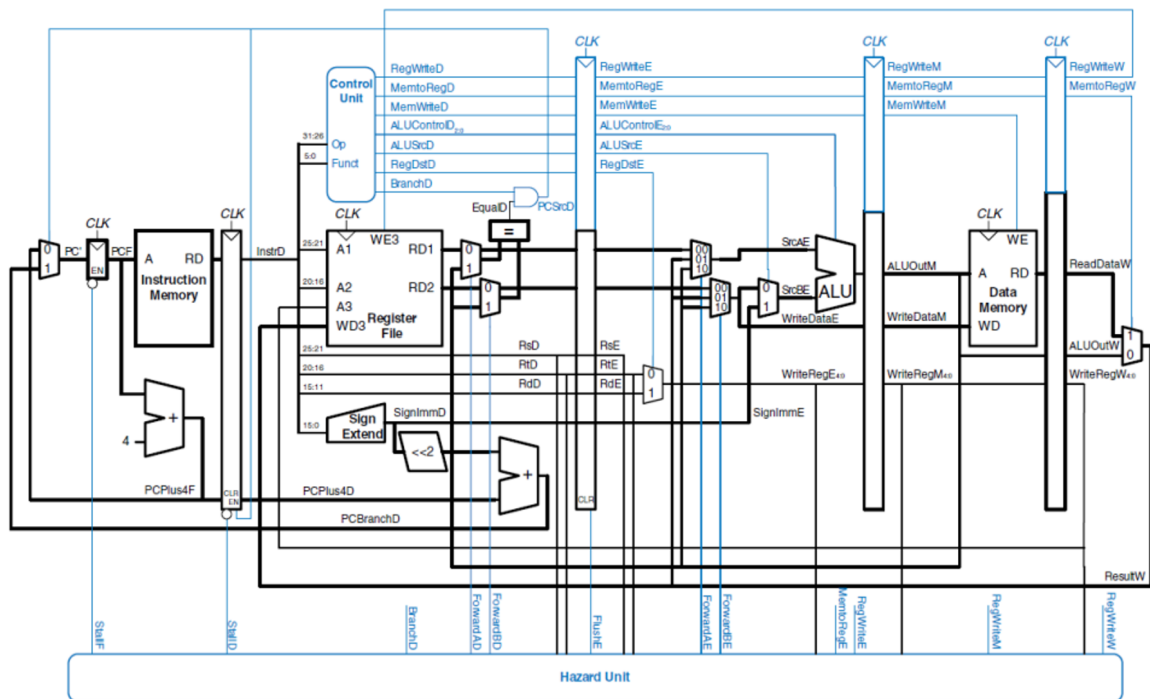


Figure 7.58 Pipelined processor with full hazard handling

## 主代码mips

```
1 `timescale 1ns / 1ps
2 `include "macro.v"
3 module mips(input clk,
4             input reset,
5             input interrupt,
6             input [31:0] i_inst_rdata,//IM_RD
7             input [31:0] m_data_rdata,//DM_RD
8
9             output [31:0] macroscopic_pc,//PC
10            output [31:0] i_inst_addr,//IM_Address
11            output [31:0] m_data_addr,//DM_Address
```

```

12         output [31:0] m_data_wdata, //DM_WD
13         output [3:0] m_data_byteen, //DM_en
14         output [31:0] m_int_addr, //Int_Address
15         output [3:0] m_int_byteen, //Int_en
16         output [31:0] m_inst_addr, //M_PC
17         output w_grf_we, //Grf_en
18         output [4:0] w_grf_addr, //Grf_Address
19         output [31:0] w_grf_wdata, //Grf_WD
20         output [31:0] w_inst_addr //W_PC
21     );
22
23     //////////////////////////////////////
24     //cpu
25     wire inter_T0, inter_T1;
26     //Bridge
27     wire [31:0] WD_out, RD_out;
28     wire [31:0] Address_out;
29     wire [3:0] DM_en;
30     wire T0_WE, T1_WE;
31     //TC
32     wire [31:2] TC_Addr;
33     wire [31:0] T0_RD, T1_RD;
34     assign TC_Addr=Address_out[31:2];
35     //interrupt
36     //cpu
37     cpu cpu (
38         .clk(clk), //
39         .reset(reset), //
40         .interrupt(interrupt), //
41         .inter_T0(inter_T0), //
42         .inter_T1(inter_T1), //
43
44         .i_inst_rdata(i_inst_rdata), //
45         .Rdata(RD_out), //dm_read_data
46
47         .macroscopic_pc(macroscopic_pc), //
48         .i_inst_addr(i_inst_addr), //
49         .m_data_addr(m_data_addr), //
50         .m_data_wdata(m_data_wdata), //dm_write_data
51         .Byteen(m_data_byteen), //
52         .m_int_addr(m_int_addr), //Int_Address
53         .m_int_byteen(m_int_byteen), //Int_en
54         .m_inst_addr(m_inst_addr),
55         .w_grf_we(w_grf_we), //
56         .w_grf_addr(w_grf_addr), //
57         .w_grf_wdata(w_grf_wdata), //
58         .w_inst_addr(w_inst_addr) //
59     );
60
61     //Bridge
62     Bridge bridge (
63         .Address_in(m_int_addr),
64         .WD_in(m_data_wdata),
65         .byteen(m_data_byteen),
66         .DM_RD(m_data_rdata), //
67         .T0_RD(T0_RD),
68         .T1_RD(T1_RD),

```

```

68     .DM_WE(DM_en),
69     .T0_WE(T0_WE),
70     .T1_WE(T1_WE),
71     .Address_out(Address_out),
72     .WD_out(WD_out), //DM
73     .RD_out(RD_out)
74 );
75
76
77
78     ////T0
79     TC T0 (
80     .clk(clk),
81     .reset(reset),
82     .Addr(TC_Addr),
83     .WE(T0_WE),
84     .Din(WD_out),
85
86     .Dout(T0_RD),
87     .IRQ(inter_T0)
88 );
89
90     ////T1
91     TC T1 (
92     .clk(clk),
93     .reset(reset),
94     .Addr(TC_Addr),
95     .WE(T1_WE),
96     .Din(WD_out),
97
98     .Dout(T1_RD),
99     .IRQ(inter_T1)
100 );
101
102 endmodule

```

## 宏的定义

```

1  `timescale 1ns / 1ps
2  //alu
3  `define _ADD      12'b000000000001
4  `define _SUB      12'b000000000010
5  `define _AND      12'b000000000100
6  `define _OR       12'b000000001000
7  `define _XOR      12'b000000010000
8  `define _NOR      12'b000000100000
9  `define _SLL      12'b000001000000
10 `define _SRA      12'b000010000000
11 `define _SRL      12'b000100000000
12 `define _SLT      12'b001000000000
13 `define _SLTU     12'b010000000000
14 `define _ALUNew  12'b100000000000
15 //ifu
16 `define PC_Initial 32'h0000_3000
17 //ext

```

```

18 `define Zero_Ext 3'b001
19 `define Sign_Ext 3'b010
20 `define Lui_Ext 3'b100
21 //NPC
22 `define PC4_NPC 5'b00001
23 `define B_transfer_NPC 5'b00010
24 `define J_transfer_NPC 5'b00100
25 `define Jr_NPC 5'b01000
26 `define NEW_NPC 5'b10000
27 //Controller_for_Reg
28 `define ALUop_Initial 7'b0000001
29 `define AluSrc1_Initial 4'b0001
30 `define AluSrc2_Initial 4'b0001
31 `define WhichtoReg_Initial 8'b00000001
32 `define RegDst_Initial 4'b0001
33 `define DM_type_Initial 6'b000001
34 //DM
35 `define Word_DM 6'b000001
36 `define Half_DM 6'b000010
37 `define Byte_DM 6'b000100
38 `define Unsigned_Half_DM 6'b001000
39 `define Unsigned_Byte_DM 6'b010000
40 //B_transfer
41 `define nop_B_trans 4'b0000
42 `define beq_B_trans 4'b0001
43 `define bgez_B_trans 4'b0010
44 `define bgtz_B_trans 4'b0011
45 `define blez_B_trans 4'b0100
46 `define bltz_B_trans 4'b0101
47 `define bne_B_trans 4'b0110
48 //MD_Unit
49 `define nop_MDU 4'b0000
50 `define mult_MDU 4'b0001
51 `define multu_MDU 4'b0010
52 `define div_MDU 4'b0011
53 `define divu_MDU 4'b0100
54 `define mfhi_MDU 4'b0101
55 `define mflo_MDU 4'b0110
56 `define mthi_MDU 4'b0111
57 `define mtlo_MDU 4'b1000
58 //TC
59 `define IDLE 2'b00
60 `define LOAD 2'b01
61 `define CNT 2'b10
62 `define INT 2'b11
63 `define ctrl mem[0]
64 `define preset mem[1]
65 `define count mem[2]
66 //CP0
67 `define IM SR[15:10]
68 `define EXL SR[1]
69 `define IE SR[0]
70 `define BD Cause[31]
71 `define IP Cause[15:10]
72 `define ExcCode Cause[6:2]
73 `define SR_Address 5'd12
74 `define Cause_Address 5'd13
75 `define EPC_Address 5'd14

```

```

76 //Bridge
77 `define Data_Begin      32'h0000_0000
78 `define Data_End        32'h0000_2fff
79 `define Text_Begin      32'h0000_3000
80 `define Text_End        32'h0000_6fff
81 `define Error_Entry     32'h0000_4180
82 `define T0_Begin        32'h0000_7f00
83 `define T0_End          32'h0000_7f0b
84 `define T1_Begin        32'h0000_7f10
85 `define T1_End          32'h0000_7f1b
86 `define Echo_Begin      32'h0000_7f20
87 `define Echo_End        32'h0000_7f23
88 //Error_Stream
89 `define Error_Int        5'd0
90 `define Error_AdEL       5'd4
91 `define Error_AdES       5'd5
92 `define Error_Syscall    5'd8
93 `define Error_RI         5'd10
94 `define Error_Ov         5'd12

```

## CPU模块

```

1  `timescale 1ns / 1ps
2  `include "macro.v"
3  module cpu( input clk,
4              input reset,
5              input interrupt,
6              input inter_T0,
7              input inter_T1,
8
9              input [31:0] i_inst_rdata,
10             input [31:0] Rdata,
11
12             output [31:0] macroscopic_pc,
13             output [31:0] i_inst_addr,
14             output [31:0] m_data_addr,
15             output [31:0] m_data_wdata,
16             output [3:0] Byteen,
17             output [31:0] m_int_addr, //Int_Address
18             output [3:0] m_int_byteen, //Int_en
19             output [31:0] m_inst_addr,
20             output w_grf_we,
21             output [4:0] w_grf_addr,
22             output [31:0] w_grf_wdata,
23             output [31:0] w_inst_addr
24             // output visited
25             );
26
27             ////////////////////////////////////////////
28             ////////////////////////////////////////////
29             //datapath_for_wire_and_reg
30             //datapath
31             wire stall;
32             //IFU
33             wire en_PC;
34             wire [31:0] pc_in;
35             wire [31:0] npc;

```

```

34     wire [31:0] F_Instr;
35     wire [31:0] F_PC;
36     wire [31:0] F_PC_origin;
37
38     //IF_ID
39     wire [31:0] D_Instr;
40     wire [31:0] D_PC;
41     wire F_reg_clr;
42     wire [4:0] f_ExcCode_in;
43     wire [4:0] d_ExcCode_in;
44     wire f_BD,d_BD;
45     wire f_syscall;
46     //////////////////////////////////////
47     //Grf
48     wire [4:0] d_rs;
49     wire [4:0] d_rt;
50     wire [4:0] d_A3;
51     wire [31:0] d_RD1;
52     wire [31:0] d_RD2;
53     //EXT
54     wire [2:0] d_SignExt;
55     //NPC
56     wire [31:0] ra;
57     wire [31:0] d_imm32;
58     wire [25:0] d_J_address;
59     wire [4:0] d_branch;
60     wire [31:0] D_PC8;
61     wire d_ALU_change;
62     //B_transfer
63     wire [31:0] d_b_transfer1;
64     wire [31:0] d_b_transfer2;
65     wire [3:0] d_B_change;
66     //Controller
67     wire d_wegrif_origin;
68     wire [5:0] d_opcode;
69     wire [5:0] d_func;
70     wire [4:0] d_rd;
71     wire [4:0] d_shamt;
72     wire [15:0] d_imm16;
73     wire [11:0] d_ALUop;
74     wire d_wegrif;
75     wire d_weDm;
76     wire [3:0] d_Alusrc1;
77     wire [3:0] d_Alusrc2;
78     wire [7:0] d_whichtoReg;
79     wire [3:0] d_RegDst;
80     wire [5:0] d_DM_type;
81     wire d_check_new;
82     wire d_excRI;
83     wire d_mfc0,d_mtc0,d_eret,d_syscall;
84     //ID_EX
85     wire [4:0] e_branch;
86     wire [4:0] d_ExcCode_fixed;
87     wire [4:0] e_ExcCode_in;
88     wire [1:0] d_Tnew;
89     wire [31:0] e_RD1;
90     wire [31:0] e_RD2;
91     wire [4:0] e_shamt;

```

```

92     wire [4:0] e_rs,e_rt,e_rd;
93     wire [4:0] e_A3;
94     wire [31:0] e_imm32;
95     wire [31:0] E_PC;
96     wire [31:0] E_PC8;
97     wire [1:0] e_Tnew;
98     wire e_Wegrf;
99     wire e_weDm;
100    wire [11:0] e_ALUop;
101    wire [3:0] e_Alusrc1;
102    wire [3:0] e_Alusrc2;
103    wire [7:0] e_whichtoReg;
104    wire [3:0] e_RegDst;
105    wire [5:0] e_DM_type;
106    wire e_check_new;
107    wire e_ALU_change;
108    wire e_BD;
109    wire d_mdu_en;
110    wire e_mfc0,e_mtc0,e_eret,e_syscall;
111    //////////////////////////////////////
112    //ALU
113    wire [31:0] e_A;
114    wire [31:0] e_B;
115    wire [31:0] res;
116    //MD_Unit
117    wire [3:0] d_MDop,e_MDop;
118    wire [31:0] HI,LO;
119    wire [31:0] mdu_out;
120    wire mdu_en,busy;
121    //EX_MEM
122    wire [4:0] m_branch;
123    wire [4:0] e_ExcCode_fixed;
124    wire [4:0] m_ExcCode_in;
125    wire [31:0] e_res;
126    wire [31:0] m_RD2;
127    wire [31:0] m_res;
128    wire [4:0] m_A3;
129    wire [4:0] m_rd;
130    wire [4:0] m_rt;
131    wire [31:0] M_PC;
132    wire [31:0] M_PC8;
133    wire [1:0] m_Tnew;
134    wire m_Wegrf;
135    wire m_Wegrf_origin;
136    wire m_weDm;
137    wire m_weDm_origin;
138    wire [7:0] m_whichtoReg;
139    wire [3:0] m_RegDst;
140    wire [5:0] m_DM_type;
141    wire m_check_new;
142    wire [31:0] m_imm32;
143    wire m_ALU_change;
144    wire m_mfc0,m_mtc0,m_eret,m_syscall;
145    wire m_BD;
146    //////////////////////////////////////
147    //DM
148    wire [31:0] m_Address;
149    wire [31:0] m_RD;

```

```

150     wire [1:0] low2;
151     wire [3:0] data_byteen ;
152     wire [31:0] DM_input,DM_output;
153     //CP0
154     wire CP0_WE;
155     wire [4:0] CP0_Address;
156     wire [31:0] CP0_WD;
157     wire BD_in;
158     wire [31:0] VPC;
159     wire [4:0] m_ExcCode_fixed;
160     wire [7:2] HWInt;
161     wire EXLClr;
162     wire req;
163     wire [31:0] EPC;
164     wire [31:0] CP0_D_out;
165     //MEM_WB
166     wire [1:0] w_Tnew;
167     wire [4:0] w_A3;
168     wire [31:0] w_res;
169     wire [31:0] w_RD;
170     wire [31:0] w_PC;
171     wire [31:0] w_PC8;
172     wire w_wgrf;
173     wire [7:0] w_whichtoReg;
174     wire [3:0] w_RegDst;
175     wire [31:0] w_imm32;
176     wire w_ALU_change;
177     wire [4:0] m_A3_origin;
178
179     ////////////////////////////////////////////
180     ////////////////////////////////////////////
181     //main_part
182     wire [1:0] D_Tuse_rs,D_Tuse_rt;
183     wire [2:0] d_selB_D1,d_selB_D2;
184     wire [2:0] d_selALU_A,d_selALU_B;
185     wire d_selDM;
186     wire is_nop;
187     wire [1:0] d_selJr;
188     wire [31:0] e_A_f,e_B_f;
189     wire [31:0] M_WD_f;//DM
190     wire [31:0] ED,MD,WD;
191     //HazardUnit
192     assign is_nop=(D_Instr==0);
193     HazardUnit hzu(
194         .D_A1(d_rs),
195         .D_A2(d_rt),
196         .E_A1(e_rs),
197         .E_A2(e_rt),
198         .M_A2(m_rt),
199         .E_A3(e_A3),
200         .M_A3(m_A3),
201         .W_A3(w_A3),
202         .E_wgrf(e_wgrf),
203         .M_wgrf(m_wgrf),
204         .W_wgrf(w_wgrf),
205         .D_check_new(d_check_new),
206         .E_check_new(e_check_new),
207         .M_check_new(m_check_new),

```



```

206
207
208     .D_Tuse_rs(D_Tuse_rs),
209     .D_Tuse_rt(D_Tuse_rt),
210     .E_Tnew(e_Tnew),
211     .M_Tnew(m_Tnew),
212     .W_Tnew(w_Tnew),
213     .E_whichtoReg(e_whichtoReg),
214     .M_whichtoReg(m_whichtoReg),
215     .start(mdu_en),
216     .busy(busy),
217     .MDU_en(d_mdu_en),
218     .Is_nop(is_nop),
219
220     .D_eret(d_eret),
221     .E_mtc0(e_mtc0),
222     .M_mtc0(m_mtc0),
223
224     .SelB_D1(d_SelB_D1),
225     .SelB_D2(d_SelB_D2),
226     .SelALU_A(d_SelALU_A),
227     .SelALU_B(d_SelALU_B),
228     .SelDM(d_SelDM),
229     .stall(stall)
230 );
231
232 //////////////////////////////////////////////////F////////////////////////////////////
233 //////////////////////////////////////
234     assign pc_in=(d_eret)?EPC+4:
235         (req)?`Error_Entry:
236         npc;
237 // assign en_PC=~stall||m_eret||req;
238 assign en_PC=~stall||req;
239 //IFU
240 PC pc(
241     .clk(clk),
242     .reset(reset),
243     .NPC(pc_in),
244     .en(en_PC),
245
246     .PC(F_PC_origin)
247 );
248 wire F_error_range;
249 assign F_Instr=(F_error_range)?0:i_inst_rdata;
250 ///////////////////////////////////EXCEPTION////////////////////////////////////
251 assign f_syscall=(F_Instr[31:26] == 6'b000000)&&(F_Instr[5:0] ==
6'b001100);
252 assign F_error_range=((F_PC<`Text_Begin)|| (F_PC>`Text_End)||
(|F_PC[1:0]))&&(!d_eret);
253 assign f_ExcCode_in=(F_error_range)?`Error_AdEL:
254     (f_syscall)?`Error_Syscall:
255     5'd0;
256 assign f_BD=(d_branch!=5'd1);
257 ///////////////////////////////////
258 //IF_ID
259 // assign F_reg_clr=d_check_new&&~d_ALU_change&&~stall;
260 assign F_reg_clr=1'b0;
261 assign F_PC=(d_eret)?EPC:F_PC_origin;

```

```

260 IF F_reg(
261     .clk(clk),
262     .reset(reset),
263     .en(en_PC),
264     .clr(F_reg_clr),
265     .req(req),
266     .F_Instr(F_Instr),
267     .F_PC(F_PC),
268     .F_ExcCode(f_ExcCode_in),
269     .F_BD(f_BD),
270
271     .D_Instr(D_Instr),
272     .D_PC(D_PC),
273     .D_ExcCode(d_ExcCode_in),
274     .D_BD(d_BD)
275 );
276
277 ////////////////////////////////////D////////////////////////////////////
278 ////////////////////////////////////
279 //Grf
280 GRF grf(
281     .A1(d_rs),
282     .A2(d_rt),
283     .A3(w_A3),
284     .WD(WD),
285     .clk(clk),
286     .reset(reset),
287     .WE(w_wegrf),
288     .WPC(W_PC),
289
290     .RD1(d_RD1),
291     .RD2(d_RD2)
292 );
293 //EXT
294 EXT ext(
295     .imm16(d_imm16),
296     .sign(d_SignExt),
297
298     .imm32(d_imm32)
299 );
300 //NPC
301 assign ra = d_b_transfer1;
302 //
303 NPC npc(
304     .F_PC(F_PC),
305     .D_PC(D_PC),
306     .offset(d_imm32),
307     .imm26(d_J_address),
308     .ra(ra),
309     .rt(d_b_transfer2),
310     .branch(d_branch),
311     .ALU_change(d_ALU_change),
312
313     .npc(npc),
314     .PC8(D_PC8)
315 );
316 //B_transfer
317 assign d_b_transfer1 = (d_selB_D1 == 3'b000)?d_RD1:

```

```

316         (d_selB_D1 == 3'b001)?WD:
317         (d_selB_D1 == 3'b010)?MD:
318         ED;
319     assign d_b_transfer2 = (d_selB_D2 == 3'b000)?d_RD2:
320         (d_selB_D2 == 3'b001)?WD:
321         (d_selB_D2 == 3'b010)?MD:
322         ED;
323     B_transfer b_trans(
324     .A(d_b_transfer1),
325     .B(d_b_transfer2),
326     .Type(d_B_change),
327
328     .ALU_change(d_ALU_change)
329     );
330     //controller
331     assign d_opcode    = D_Instr[31:26];
332     assign d_rs        = D_Instr[25:21];
333     assign d_rt        = D_Instr[20:16];
334     assign d_rd        = D_Instr[15:11];
335     assign d_shamt     = D_Instr[10:6];
336     assign d_func      = D_Instr[5:0];
337     assign d_imm16     = D_Instr[15:0];
338     assign d_J_address = D_Instr[25:0];
339     Controller controller(
340     .op(d_opcode),
341     .func(d_func),
342     .rs(d_rs),
343     .rt(d_rt),
344
345     .ALUop(d_ALUop),
346     .wgrf(d_wgrf),
347     .weDm(d_weDm),
348     .branch(d_branch),
349     .AluSrc1(d_Alusrc1),
350     .AluSrc2(d_Alusrc2),
351     .WhichToReg(d_WhichToReg),
352     .RegDst(d_RegDst),
353     .SignExt(d_SignExt),
354     .B_change(d_B_change),
355     .DM_type(d_DM_type),
356     .MDop(d_MDop),
357     .check_new(d_check_new),
358     .excRI(d_excRI),
359
360     .D_Tuse_rs(D_Tuse_rs),
361     .D_Tuse_rt(D_Tuse_rt),
362     .D_Tnew(d_Tnew),
363
364     .mfc0(d_mfc0),
365     .mtc0(d_mtc0),
366     .eret(d_eret),
367     .syscall(d_syscall)
368     );
369
370     // assign d_wgrf=d_check_new&&(d_branch!=`PC4_NPC)?(d_ALU_change?
1'b1:1'b0):d_wgrf_origin;
371     assign d_wgrf=d_wgrf_origin;
372     //submit for branch

```

```

373 ///////////////////////////////////EXCEPTION////////////////////////////////////
374 assign d_ExcCode_fixed= (d_excRI)?`Error_RI:
375                                d_ExcCode_in;
376
377 ///////////////////////////////////
378 //ID_EX
379 ID D_reg(
380     .clk(clk),
381     .reset(reset),
382     .clr(stall),
383     .req(req),
384     .D_RD1(d_b_transfer1),
385     .D_RD2(d_b_transfer2),
386     .D_instr_s(d_shamt),
387     .D_A1(d_rs),
388     .D_A2(d_rt),
389     .D_A3(d_rd),
390     .D_imm32(d_imm32),
391     .D_PC(D_PC),
392     .D_PC8(D_PC8),
393     .D_Tnew(d_Tnew),
394     .D_wegr(f(d_wegr(f),
395     .D_weDm(d_weDm),
396     .D_ALUop(d_ALUop),
397     .D_Alusrc1(d_Alusrc1),
398     .D_Alusrc2(d_Alusrc2),
399     .D_whichtoReg(d_whichtoReg),
400     .D_RegDst(d_RegDst),
401     .D_DM_type(d_DM_type),
402     .D_ALU_change(d_ALU_change),
403     .D_MDop(d_MDop),
404     .D_check_new(d_check_new),
405     .D_branch(d_branch),
406     .D_ExcCode(d_ExcCode_fixed),
407     .D_BD(d_BD),
408     .D_mfc0(d_mfc0),
409     .D_mtc0(d_mtc0),
410     .D_eret(d_eret),
411     .D_syscall(d_syscall),
412
413     .E_RD1(e_RD1),
414     .E_RD2(e_RD2),
415     .E_instr_s(e_shamt),
416     .E_A1(e_rs),
417     .E_A2(e_rt),
418     .E_A3(e_rd),
419     .E_imm32(e_imm32),
420     .E_PC(E_PC),
421     .E_PC8(E_PC8),
422     .E_Tnew(e_Tnew),
423     .E_wegr(f(e_wegr(f),
424     .E_weDm(e_weDm),
425     .E_ALUop(e_ALUop),
426     .E_Alusrc1(e_Alusrc1),
427     .E_Alusrc2(e_Alusrc2),
428     .E_whichtoReg(e_whichtoReg),
429     .E_RegDst(e_RegDst),

```

```

429     .E_DM_type(e_DM_type),
430     .E_ALU_change(e_ALU_change),
431     .E_MDop(e_MDop),
432     .E_check_new(e_check_new),
433     .E_branch(e_branch),
434     .E_ExcCode(e_ExcCode_in),
435     .E_BD(e_BD),
436     .E_mfc0(e_mfc0),
437     .E_mtc0(e_mtc0),
438     .E_eret(e_eret),
439     .E_syscall(e_syscall)
440 );
441 assign d_mdu_en=(d_MDop!=`nop_MDU);
442
443 //////////////////////////////////////////////////E////////////////////////////////////
444 //////////////////////////////////////////////////
445 assign e_A3 = (e_RegDst == 4'b0001)?e_rd:
446               (e_RegDst == 4'b0010)?e_rt:
447               5'b11111;
448 //ALU
449 assign e_A = (e_AluSrc1 == 4'b0001)?e_A_f://
450             e_B_f;
451
452 assign e_A_f = (d_SelALU_A == 3'b010)?MD:
453               (d_SelALU_A == 3'b001)?WD:
454               e_RD1;//rs
455
456 assign e_B_f = (d_SelALU_B == 3'b010)?MD:
457               (d_SelALU_B == 3'b001)?WD:
458               e_RD2;//rt
459
460 assign e_B = (e_AluSrc2 == 4'b0001)?e_B_f:
461             (e_AluSrc2 == 4'b0010)?e_imm32://
462             (e_AluSrc2 == 4'b0100)?({27{1'b0}},e_shamt}):
463             e_A_f;
464 //////////////EXCEPTION////////////////////
465 wire Error_Ov_Alu;
466 assign Error_Ov_Alu=overflow&&(e_DM_type==6'd0);
467 wire Error_Ov_DM;
468 assign Error_Ov_DM=overflow&&(e_DM_type!=6'd0);
469 assign e_ExcCode_fixed= (Error_Ov_DM)?((e_weDm)?
`Error_AdES:`Error_AdEL):
470
471             (Error_Ov_Alu)?`Error_Ov:
472             e_ExcCode_in;
473 //////////////////////////////////////////////////
474 ALU alu(
475     .A(e_A),
476     .B(e_B),
477     .ALUop(e_ALUop),
478
479     .res(res),
480     .overflow(overflow)
481 );
482 //MD_Unit
483 assign mdu_en=(e_MDop==`mult_MDU)|| (e_MDop==`multu_MDU)||
484             (e_MDop==`div_MDU)|| (e_MDop==`divu_MDU);
485 MD_Unit mdu (
486     .clk(clk),

```

```

484 .reset(reset),
485 .en(mdu_en),
486 .req(req),
487 .MDop(e_MDop),
488 .A(e_A),
489 .B(e_B),
490
491 .HI(HI),
492 .LO(LO),
493 .out(mdu_out),
494 .busy(busy)
495 );
496 assign e_res=(e_MDop==`mfhi_MDU | e_MDop==`mflo_MDU)?mdu_out:
497                                     res;
498 //EX_MEM
499 assign ED = (e_whichtoReg == 8'b0000_0001)?e_res:
500             // (e_whichtoReg == 8'b0000_0010)?e_RD:
501             (e_whichtoReg == 8'b0000_0100)?e_imm32:
502             (e_whichtoReg == 8'b0000_1000)?E_PC8:
503             e_ALU_change;//
504 EX E_reg(
505 .clk(clk),
506 .reset(reset),
507 .req(req),
508 .E_A2(e_rt),
509 .E_rd(e_rd),
510 .E_A3(e_A3),
511 .E_RD2(e_B_f),
512 .E_ALUout(e_res),
513 .E_PC(E_PC),
514 .E_PC8(E_PC8),
515 .E_Tnew(e_Tnew),
516 .E_wgrf(e_wgrf),
517 .E_weDm(e_weDm),
518 .E_whichtoReg(e_whichtoReg),
519 .E_RegDst(e_RegDst),
520 .E_DM_type(e_DM_type),
521 .E_imm32(e_imm32),
522 .E_ALU_change(e_ALU_change),
523 .E_check_new(e_check_new),
524 .E_branch(e_branch),
525 .E_ExcCode(e_ExcCode_fixed),
526 .E_BD(e_BD),
527 .E_mfc0(e_mfc0),
528 .E_mtc0(e_mtc0),
529 .E_eret(e_eret),
530 .E_syscall(e_syscall),
531
532 .M_A2(m_rt),
533 .M_A3(m_A3_origin),
534 .M_rd(m_rd),
535 .M_RD2(m_RD2),
536 .M_ALUout(m_res),
537 .M_PC(M_PC),
538 .M_PC8(M_PC8),
539 .M_Tnew(m_Tnew),
540 .M_wgrf(m_wgrf_origin),
541 .M_weDm(m_weDm_origin),

```

```

542     .M_whichtoReg(m_whichtoReg),
543     .M_RegDst(m_RegDst),
544     .M_DM_type(m_DM_type),
545     .M_imm32(m_imm32),
546     .M_ALU_change(m_ALU_change),
547     .M_check_new(m_check_new),
548     .M_branch(m_branch),
549     .M_ExcCode(m_ExcCode_in),
550     .M_BD(m_BD),
551     .M_mfc0(m_mfc0),
552     .M_mtc0(m_mtc0),
553     .M_eret(m_eret),
554     .M_syscall(m_syscall)
555 );
556
557 //DM
558 assign m_Address = m_res;
559 assign M_WD_f     = (d_selDM)?WD:
560                  m_RD2;
561 assign m_weDm=(req)?0:m_weDm_origin;
562
563 //EXCEPTION
564 wire is_DM;
565 assign is_DM=(m_DM_type==`word_DM) || (m_DM_type==`Half_DM) ||
(m_DM_type==`Unsigned_Half_DM) || (m_DM_type==`Byte_DM) ||
(m_DM_type==`Unsigned_Byte_DM);
566 wire Is_Align;
567 assign Is_Align=((m_DM_type==`word_DM)&&(!low2)) ||
((m_DM_type==`Half_DM | m_DM_type==`Unsigned_Half_DM)&&
(low2[0]));
568 wire Is_ErrorRange;
569 assign Is_ErrorRange=!((m_Address>=`Data_Begin&&m_Address<=`Data_End) ||
(m_Address >= `T0_Begin&&m_Address <= `T0_End) ||
(m_Address >= `T1_Begin&&m_Address <= `T1_End) ||
(m_Address >= `Echo_Begin&&m_Address <=
`Echo_End));
570
571 wire Error_Timer;
572 assign Error_Timer=(m_DM_type!=`word_DM)&&((m_Address >=
`T0_Begin&&m_Address <= `T0_End) ||
(m_Address >= `T1_Begin&&m_Address <= `T1_End));
573 wire Error_Save_Timer;
574 assign Error_Save_Timer=
(m_Address>=32'h0000_7f08&&m_Address<=32'h0000_7f0b) ||
(m_Address>=32'h0000_7f18&&m_Address<=32'h0000_7f1b);
575 assign m_ExcCode_fixed=((is_DM&&Is_Align&&!m_weDm_origin) ||
(is_DM&&Is_ErrorRange&&!m_weDm_origin) ||
(is_DM&&Error_Timer&&!m_weDm_origin))?`Error_AdEL:
((is_DM&&Is_Align&&m_weDm_origin) ||
(is_DM&&Is_ErrorRange&&m_weDm_origin) ||
(is_DM&&Error_Timer&&m_weDm_origin) ||
(is_DM&&Error_Save_Timer&&m_weDm_origin))?`Error_AdES:
m_ExcCode_in;
576
577 assign low2=m_Address[1:0];

```

```

584 wire [31:0] m_Rd_origin;
585 DM_In din(.low2(low2),
586         .WD(M_WD_f),
587         .DM_type(m_DM_type),
588
589         .data_byteen(data_byteen),
590         .DM_input(DM_input)
591     );
592 assign DM_output=Rdata;
593 DM_Out dot( .low2(low2),
594             .DM_type(m_DM_type),
595             .DM_output(DM_output),
596
597             .RD(m_Rd_origin));
598 assign MD = (m_whichtoReg == 8'b0000_0001)?m_res:
599             // (m_whichtoReg == 8'b0000_0010)?m_RD:
600             (m_whichtoReg == 8'b0000_0100)?m_imm32:
601             (m_whichtoReg == 8'b0000_1000)?M_PC8:
602             m_ALU_change;//
603 assign m_A3=m_A3_origin;//condition
604 wire condition;
605 assign condition=1'b0;
606 // assign m_A3=m_check_new?(condition?
5'd31:m_A3_origin):m_A3_origin;//condition
607 //provide for condition//////////
608
609
610 ///////////////////////////////////////////////////
611 //CP0
612 assign CP0_WE=m_mtc0;
613 assign CP0_Address=m_rd;
614 assign CP0_WD=M_WD_f;
615 assign BD_in=m_BD;
616 assign VPC=(m_branch==5'b00001)?M_PC:D_PC;
617 assign HWInt={3'd0,interrupt,inter_T1,inter_T0};
618 assign EXLC1r=m_eret;
619 CP0 cp0 (
620     .clk(clk),
621     .reset(reset),
622     .WE(CP0_WE),
623     .Address(CP0_Address),
624     .WD(CP0_WD),
625     .BD_in(BD_in),
626     .VPC(VPC),
627     .ExcCode_in(m_ExcCode_fixed),
628     .HWInt(HWInt),
629     .EXLC1r(EXLC1r),
630     .req(req),
631     .EPC(EPC),
632     .D_out(CP0_D_out)
633 );
634 assign m_RD=(m_mfc0)?CP0_D_out:m_Rd_origin;
635 assign m_wegr=(req)?0:m_wegr_origin;
636 //MEM_WE
637 MEM M_reg(
638     .clk(clk),
639     .reset(reset),
640     .M_A3(m_A3),

```



```

641     .M_ALUout(m_res),
642     .M_RD(m_RD),
643     .M_PC(M_PC),
644     .M_PC8(M_PC8),
645     .M_Wegrf(m_Wegrf),
646     .M_WhichtoReg(m_WhichtoReg),
647     .M_RegDst(m_RegDst),
648     .M_imm32(m_imm32),
649     .M_ALU_change(m_ALU_change),
650     .M_Tnew(m_Tnew),
651
652     .W_A3(w_A3),
653     .W_ALUout(w_res),
654     .W_RD(w_RD),
655     .W_PC(W_PC),
656     .W_PC8(W_PC8),
657     .W_Wegrf(w_Wegrf),
658     .W_WhichtoReg(w_WhichtoReg),
659     .W_RegDst(w_RegDst),
660     .W_imm32(w_imm32),
661     .W_ALU_change(w_ALU_change),
662     .W_Tnew(w_Tnew)
663 );
664
665     ////////////////////////////////////w////////////////////////////////////
666     ////////////////////////////////////
667     assign WD = (w_WhichtoReg == 8'b0000_0001)?w_res:
668                 (w_WhichtoReg == 8'b0000_0010)?w_RD:
669                 (w_WhichtoReg == 8'b0000_0100)?w_imm32:
670                 (w_WhichtoReg == 8'b0000_1000)?w_PC8:
671                 w_ALU_change;//
672
673     output////////////////////////////////////
674     ////
675     assign macroscopic_pc= M_PC;
676     assign i_inst_addr    = (d_eret)?EPC:F_PC;
677     assign m_data_addr    = m_Address;
678     assign m_data_wdata   = DM_input;
679     assign Byteen         = (m_weDm)?data_byteen:4'b0000;
680     assign m_int_addr     = m_Address;//Int_Address
681     assign m_int_byteen   = (m_weDm)?data_byteen:4'b0000;//Int_en
682     assign m_inst_addr    = M_PC;
683     assign w_grf_we       = w_Wegrf;
684     assign w_grf_addr     = w_A3;
685     assign w_grf_wdata    = WD;
686     assign w_inst_addr    = W_PC;
687     // assign visited     = (m_data_addr==32'h7f20);
688
689 endmodule

```

## <一>.F级流水线

1. PC

(1) 端口说明

表1-IFU端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号，将PC值置为0x0000_3000： 0：无效 1：复位
3	en	I	使能信号，决定是否阻塞
4	NPC[31:0]	I	下一周期PC的地址
6	PC[31:0]	O	当前执行的PC

(2) 功能定义

表2-IFU功能定义

序号	功能	描述
1	复位	reset有效时，PC置为0x00003000
2	更新PC的值	将PC赋值为NPC

<二>.D级流水线

1. GRF

(1) 端口说明

表3-GRF端口说明

序号	信号名	方向	描述
1	clk	I	时钟信号
2	reset	I	同步复位信号，将32个寄存器中全部清零 1：清零 0：无效
3	WE	I	写使能信号 1：可向GRF中写入数据 0：不能向GRF中写入数据
4	A1[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD1
5	A2[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到RD2
6	A3[4:0]	I	5位地址输入信号，指定32个寄存器中的一个，作为RD的写入地址
7	WD[31:0]	I	32位写入数据
8	WPC[31:0]	I	当前写入GRF的PC
9	RD1[31:0]	O	输出A1指定的寄存器的32位数据
10	RD2[31:0]	O	输出A2指定的寄存器的32位数据

(2) 功能定义

表4-GRF功能定义

序号	功能	描述
1	同步复位	reset为1时，将所有寄存器清零
2	读数据	将A1和A2地址对应的寄存器的值分别通过RD1和RD2读出
3	写数据	当WE为1且时钟上升沿来临时，将WD写入到A3对应的寄存器内部
4	内部转发	当A1和A2之一与A3相等且写入信号为1时，用WD代替RD1或RD2的输出

2. EXT

(1) 端口说明

表9-EXT端口说明

序号	信号名	方向	描述
1	imm16[15:0]	I	代扩展的16位信号
2	sign[2:0]	I	无符号或符号扩展选择信号 3'b001: 无符号扩展 3'b010: 符号扩展 3'b100: 寄存到高位
3	imm32[31:0]	O	扩展后的32位的信号

(2) 功能定义

表10-EXT功能定义

序号	功能	描述
1	无符号扩展	当sign为3'b001时，将imm16无符号扩展输出
2	符号扩展	当sign为3'b010时，将imm16符号扩展输出
3	存储到高位	当sign为3'100时，将imm16存在高16位

3. Controller

(1) 端口说明

表11-Controller端口说明

序号	信号名	方向	描述
1	op[5:0]	I	instr[31:26]6位控制信号
2	func[5:0]	I	instr[5:0]6位控制信号
3	rt[4:0]	I	instr[20:16]5位控制信号
4	AluOp[11:0]	O	ALU的控制信号
5	WeGrf	O	GRF写使能信号 0: 禁止写入 1: 允许写入
6	WeDm	O	DM的写入信号 0: 禁止写入 1: 允许写入
7	branch[3:0]	O	PC转移位置选择信号 4'b0001:其他情况 4'b0010:beq 4'b0100:j jal 4'b1000;jr;
8	AluSrc1[3:0]	O	参与ALU运算的第一个数 4'b0001: RD1 4'b0010: RD2
9	AluSrc2[3:0]	O	参与ALU运算的第二个数, 来自GRF还是imm 4'b0001: RD2 4'b0010: imm32 4'b0100: offset
10	WhichToReg[7:0]	O	将何种数据写入GRF? 8'b0000_0001: ALU计算结果 8'b0000_0010: DM读出信号 8'b0000_0100: upperImm 8'b0000_1000: PC+4
11	RegDst[3:0]	O	写入GRF的哪个寄存器? 4'b0001:rd 4'b0010:rt 4'b0100:31号寄存器
12	SignExt[2:0]	O	拓展方式: 3'b001:0拓展 3'b010:符号拓展 3'b100:存储到高位
13	B_change[3:0]	O	B转移的类型 4'b0001:beq 4'b0010:slt 4'b0100:blez

序号	信号名	方向	描述
14	DM_type[5:0]	O	存取指令类型: 6'b000001:lw/sw 6'b000010:lh/sh 6'b000100:lb/sb 6'b001000:lhu 6'b010000:lbu
15	MDop[3:0]	O	乘除指令选择信号: 4'b0000:无操作 4'b0001:mult 4'b0010:multu 4'b0011:div 4'b0100:divu 4'b0101:mfhi 4'b0110:mflo 4'b0111:mthi 4'b1000:mtlo
16	D_Tuse_rs[1:0]	O	指令的rs寄存器的值从第一次进入D级到被使用的周期数
17	D_Tuse_rt[1:0]	O	指令的rt寄存器的值从第一次进入D级到被使用的周期数
18	D_Tnew[1:0]	O	指令从进入D开始到产生运算结果需要的周期数

## 4.NPC

### NPC（下一指令计算单元）

该模块根据当前pc（包括D级和F级）和其他控制信号（NPCOp，CMP输出信息），计算出下一指令所在的地址npc，传入IFU模块。

- 端口定义

信号名	方向	位宽	描述
F_pc	I	32	F级指令地址
D_pc	I	32	D级指令地址
offset	I	32	地址偏移量，用于计算B类指令所要跳转的地址
imm26	I	26	当前指令数据的前26位（0~25），用于计算jal和j指令所要跳转的地址
ra	I	32	储存在寄存器（\$ra或是jalr指令中存储“PC+4”的寄存器）中的地址数据，用于实现jr和jalr指令
ALU_change	I	1	B类指令判断结果 1：说明当前B类指令的判断结果为真 0：说明判断结果为假
branch	I	4	NPC功能选择 0x000：顺序执行 0x001：B类指令跳转 0x010：jal/j跳转 0x011：jr/jalr跳转
npc	O	32	输出下一指令地址
PC8	O	32	jr指令时存储PC+8的值

### 5.B\_transfer(B类指令比较单元)

该单元根据输入的CMPOp信号对当前B指令的类型进行判断，进而对当前输入的数值进行相应比较，最后输出结果。

• 端口定义

信号名	方向	位宽	描述
A	I	32	输入B_transfer单元的第一个数据
B	I	32	输入B_transfer单元的第二个数据
Type	I	4	Type功能选择信号 0x0001：beq判断 0x0010：slt判断 0x0100：blez判断
ALU_change	O	1	判断结果输出 1：判断结果为真 0：判断结果为假

## <三>.E级流水线

1. ALU

(1) 端口说明

表5-ALU端口说明

序号	信号名	方向	描述
1	A[31:0]	I	参与运算的第一个数
2	B[31:0]	I	参与运算的第二个数
3	ALUop[11:0]	I	决定ALU做何种操作 12'b0000_0000_0001: 无符号加 12'b0000_0000_0010: 无符号减 12'b0000_0000_0100: 与 12'b0000_0000_1000: 或 12'b0000_0001_0000:异或 12'b0000_0010_0000:或非 12'b0000_0100_0000:sll 12'b0000_1000_0000:sra 12'b0001_0000_0000:srl 12'b0010_0000_0000:slt 12'b0100_0000_0000:sltu
4	res	O	A与B做运算后的结果

(2) 功能定义

表6-ALU功能定义

序号	功能	描述
1	加运算	res = A + B
2	减运算	res = A - B
3	与运算	res = A & B
4	或运算	res = A   B
5	左移位运算	Res=A<<B

2.MD\_Unit

- 端口定义



方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	mips.v中的clk
I	reset	1	同步复位信号	mips.v中的reset
I	en	1	MD_Unit使能信号	判断是否为乘除计算
I	MDop	4	指令选择信号	Controller
I	A	32	第一个计算数	e_A
I	B	32	第二个计算数	e_B
O	HI	32	HI寄存器输出	
O	LO	32	LO寄存器输出	
O	out	32	取出HI或LO的值	
O	busy	1	是否在进行乘除计算	

## <四>.M级流水线

### 1. DM\_In

- 端口定义

方向	信号名	位宽	描述	输入来源
I	low2	2	时钟信号	Address低两位
I	WD	31	处理前写入结果	32位写入数据
I	DM_type	6	决定存取指令类型	存取指令类型: 6'b000001:lw/sw 6'b000010:lh/sh 6'b000100:lb/sb 6'b001000:lhu 6'b010000:lbu
O	data_byteen	4	字节使能信号	
O	DM_input	32	处理后的写入结果	

### 2.DM\_Out

- 端口定义

方向	信号名	位宽	描述	输入来源
I	low2	2	时钟信号	Address低两位
I	DM_output	32	处理前读入信号	32位读入数据
I	DM_type	6	决定存取指令类型	存取指令类型: 6'b000001:lw/sw 6'b000010:lh/sh 6'b000100:lb/sb 6'b001000:lhu 6'b010000:lbu
O	RD	32	处理后的读入结果	

### 3.CP0

- 端口定义

方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	cpu主时钟
I	reset	1	复位信号	cpu同步复位信号
I	WE	1	写入使能	前一级相同信号输入
I	Address	5	CP0 寄存器编号	前一级相同信号输入
I	WD	32	CP0 寄存器的写入数据	前一级相同信号输入
I	BD_in	1	分支延迟槽指令标志	前一级相同信号输入
I	VPC	32	中断/异常时的 PC	前一级相同信号输入
I	ExcCode_in	5	中断/异常的类型	前一级相同信号输入
I	HWInt	6	6 个设备中断	前一级相同信号输入
I	EXLClr	1	置 0 SR 的 EXL 位	m_eret控制信号输入
O	req	1	异常/中断请求	
O	EPC	32	处理后的读入结果	
O	D_out	32	CP0 寄存器的输出数据	

## <五>.各级流水线寄存器

## 1.IF\_ID

### D\_Reg (IF/ID流水寄存器)

- 端口定义

方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	mips.v中的clk
I	reset	1	同步复位信号	mips.v中的reset
I	en	1	D级寄存器使能信号	stall信号取反
I	clr	1	D级寄存器清空信号	是否清空延迟槽决定
I	req	1	D级寄存器异常/中断请求信号	CP0
I	F_instr	32	F级instr输入	IFU_instr
I	F_pc	32	F级pc输入	IFU_pc
I	F_ExcCode	5	F级ExcCode输入	前一级相同信号
I	F_BD	1	F级BD输入	前一级相同信号
O	D_instr	32	D级instr输出	
O	D_pc	32	D级pc输出	
O	D_ExcCode	5	D级ExcCode输出	
O	D_BD	1	D级BD输出	

## 2.ID\_EX

### E\_Reg (ID/EX流水寄存器)

- 端口定义

方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	mips.v中的clk
I	reset	1	同步复位信号	mips.v中的reset
I	clr	1	E级寄存器清空信号	HazardUnit中stall信号
I	req	1		
I	D_RD1	32	D级GRF输出RD1	通过B_transfer_D1转发的数据
I	D_RD2	32	D级GRF输出RD2	通过B_transfer_D2转发的数据
I	D_instr_s	5	D级instr的shamt	D_instr的s域数据
I	D_A1	5	D级A1输入	D_instr的rs域数据
I	D_A2	5	D级A2输入	D_instr的rt域数据
I	D_A3	5	D级A3输入	通过MUX_A3选择出的数据
I	D_imm32	32	D级imm32输入	通过EXT模块扩展出的数据
I	D_PC	32	D级PC输入	前一级相同信号
I	D_PC8	32	D级PC8输入	前一级相同信号
I	Tnew_D	2	D级指令的Tnew输入	前一级相同信号
I	D_Wegrf	1	D级控制信号输入	前一级相同信号
I	D_WeDm	1	D级控制信号输入	前一级相同信号
I	D_ALUOp	7	D级控制信号输入	前一级相同信号
I	D_Alusrc1	4	D级控制信号输入	前一级相同信号
I	D_Alusrc2	4	D级控制信号输入	前一级相同信号
I	D_WhichToReg	8	D级控制信号输入	前一级相同信号
I	D_RegDst	4	D级控制信号输入	前一级相同信号
I	D_DM_type	6	D级控制信号输入	前一级相同信号
I	D_ALU_change	1	D级ALU_change输入	前一级相同信号
I	D_MDop	4	D级MDop输入	前一级相同信号
I	D_branch	5	D级branch输入	前一级相同信号
I	D_ExcCode	5	D级ExcCode输入	前一级相同信号
I	D_BD	1	D级BD输入	前一级相同信号

方向	信号名	位宽	描述	输入来源
I	D_mfc0	1	D级mfc0输入	前一级相同信号
I	D_mtc0	1	D级mtc0输入	前一级相同信号
I	D_eret	1	D级eret输入	前一级相同信号
I	D_syscall	1	D级syscall输入	前一级相同信号
O	E_RD1	32	E级RD1输出	
O	E_RD2	32	E级RD2输出	
O	E_instr_s	5	移位指令的位移数	
O	E_A1	5	E级A1输出	
O	E_A2	5	E级A2输出	
O	E_A3	5	E级A3输出	
O	E_imm32	32	E级imm32输出	
O	E_PC	32	E级PC输出	
O	E_PC8	32	E级PC8输出	
O	E_Tnew	2	E级指令的Tnew输出	
O	E_Wegrif	1	E级控制信号输出	
O	E_WeDm	1	E级控制信号输出	
O	E_ALUop	12	E级控制信号输出	
O	E_AluSrc1	4	E级控制信号输出	
O	E_AluSrc2	4	E级控制信号输出	
O	E_WhichtoReg	11	E级控制信号输出	
O	E_RegDst	3	E级控制信号输出	
O	E_DM_type	6	E级控制信号输出	
O	E_ALU_change	1	E级ALU_change输出	
O	E_MDop	4	E级MDop输出	
O	E_branch	5	E级branch输出	
O	E_ExcCode	5	E级ExcCode输出	
O	E_BD	1	E级BD输出	
O	E_mfc0	1	E级mfc0输出	
O	E_mtc0	1	E级mtc0输出	

方向	信号名	位宽	描述	输入来源
O	E_eret	1	E级eret输出	
O	E_syscall	1	E级syscall输出	

- 运算功能

$$Tnew\_E = (Tnew\_D > 0) ? Tnew\_D - 1 : 0$$
$$Tnew\_E = (Tnew\_D > 0) ? Tnew\_D - 1 : 0$$

### 3.EX\_MEM

#### M\_Reg (EX/MEM流水寄存器)

- 端口定义

方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	mips.v中的clk
I	reset	1	同步复位信号	mips.v中的reset
I	E_A2	5	E级A2输入	ALU_out数据
I	E_A3	5	E级A3输入（转发值）	MUX_ALU选择出来的数据
I	E_RD2	32	E级RD2输入	前一级相同信号
I	E_ALUout	32	E级res输入	前一级相同信号
I	E_PC	32	E级PC输入	前一级相同信号
I	E_PC8	32	E级PC8输入	前一级相同信号
I	E_Tnew	2	E级Tnew输入	前一级相同信号
I	E_Wegrf	1	E级控制信号输入	前一级相同信号
I	E_WeDm	1	E级控制信号输入	前一级相同信号
I	E_WhichtoReg	8	E级控制信号输入	前一级相同信号
I	E_RegDst	4	E级控制信号输入	前一级相同信号
I	E_DM_type	6	E级控制信号输入	前一级相同信号
I	E_ALU_change	1	E级ALU_change输入	前一级相同信号
I	E_imm32	32	E级imm32输入	前一级相同信号
I	E_branch	5	E级branch输入	前一级相同信号
I	E_ExcCode	5	E级ExcCode输入	前一级相同信号
I	E_BD	1	E级BD输入	前一级相同信号
I	E_mfc0	1	E级mfc0输入	前一级相同信号
I	E_mtc0	1	E级mtc0输入	前一级相同信号
I	E_eret	1	E级eret输入	前一级相同信号
I	E_syscall	1	E级syscall输入	前一级相同信号
O	M_A2	5	M级A2输出	
O	M_A3	5	M级A3输出	
O	M_RD2	32	M级RD2输出	
O	M_ALUout	32	M级res输出	
O	M_PC	32	M级PC输出	
O	M_PC8	32	M级PC8输出	
O	M_Tnew	2	M级Tnew输出	

方向	信号名	位宽	描述	输入来源
O	M_Wegrf	1	M级Tnew输出	
O	M_WeDm	1	M级控制信号输出	
O	M_WhichtoReg	8	M级控制信号输出	
O	M_RegDst	4	M级控制信号输出	
O	M_DM_type	6	M级控制信号输出	
O	M_ALU_change	1	M级ALU_change输出	
O	M_imm32	32	M级imm32输出	
O	M_branch	5	M级branch输出	
O	M_ExcCode	5	M级ExcCode输出	
O	M_BD	1	M级BD输出	
O	M_mfc0	1	M级mfc0输出	
O	M_mtc0	1	M级mtc0输出	
O	M_eret	1	M级eret输出	
O	M_syscall	1	M级syscall输出	

- 运算功能

$Tnew\_M = (Tnew\_E > 0) ? Tnew\_E - 1 : 0$   
 $Tnew\_M = (Tnew\_E > 0) ? Tnew\_E - 1 : 0$

#### 4.MEM\_WB

#### W\_Reg (MEM/WB流水寄存器)

- 接口定义



方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	mips.v中的clk
I	reset	1	同步复位信号	mips.v中的reset
I	M_A3	5	M级A3输入	前一级相同信号
I	M_RD	32	M级RD输入	前一级相同信号
I	M_PC	32	M级PC输入	前一级相同信号
I	M_PC8	32	M级PC8输入	前一级相同信号
I	M_Wegrf	1	M级控制信号输入	前一级相同信号
I	M_WhichtoReg	1	M级控制信号输入	前一级相同信号
I	M_RegDst	4	M级控制信号输入	前一级相同信号
I	M_imm32	32	M级imm32输入	前一级相同信号
I	M_ALU_change	1	M级ALU_change输入	前一级相同信号
I	M_Tnew	2	M级Tnew输入	前一级相同信号
O	W_A3	5	W级A3输出	
O	W_ALUout	32	W级res输出	
O	W_RD	32	W级RD输出	
O	W_PC	32	W级PC输出	
O	W_PC8	32	W级PC8输出	
O	W_Wegrf	1	W级控制信号输出	
O	W_WhichtoReg	8	W级控制信号输出	
O	W_RegDst	4	W级控制信号输出	
O	W_imm32	32	W级imm32输出	
O	W_ALU_change	1	W级ALU_change输出	
O	W_Tnew	2	W级Tnew输出	

## <六>.暂停、转发处理及相关多路选择器

### (一) .冲突综合单元 (HazardUnit)

方向	信号名	位宽	描述
I	D_A1	5	D级A1端输入
I	D_A2	5	D级A2端输入
I	E_A1	5	E级A1端输入
I	E_A2	5	E级A2端输入
I	M_A2	5	M级A2端输入
I	E_A3	5	E级A3端输入
I	M_A3	5	M级A3端输入
I	W_A3	5	W级A3端输入
I	D_Tuse_rs	2	D_Tuse_rs输入
I	D_Tuse_rt	2	D_Tuse_rt输入
I	E_Tnew	2	E级Tnew输入
I	M_Tnew	2	M级Tnew输入
I	W_Tnew	2	W级Tnew输入
I	E_Wegrf	1	E级Wegrf输入
I	M_Wegrf	1	M级Wegrf输入
I	W_Wegrf	1	W级Wegrf输入
I	E_WhichtoReg	8	E级WhichtoReg输入
I	M_WhichtoReg	8	M级WhichtoReg输入
I	start	1	乘除开始信号
I	busy	1	乘除忙碌信号
I	MDU_en	1	D级将进行乘除运算信号
I	D_eret	1	D级eret输入
I	E_mtc0	1	E级mtc0输入
I	M_mtc0	1	M级mtc0输入
O	SelB_D1	2	B_transfer的D1输入转发信号
O	SelB_D2	2	B_transfer的D2输入转发信号
O	SelALU_A	2	ALU输入A转发信号
O	SelALU_B	2	ALU输入B转发信号
O	SelDM	1	DM写入WD转发信号
O	stall	1	冲突信号

## (二) .控制和冒险简述

- 对于控制冒险，本实验要求大家实现**比较过程前移至 D 级**，并采用**延迟槽**。
- 对于数据冒险，两大策略及其应用：

- 1 假设当前我需要的数据，其实已经计算出来，只是还没有进入寄存器堆，那么我们可以用**\*\*转发\*\***( Forwarding )来解决，即不引用寄存器堆的值，而是直接从后面的流水级的供给者把计算结果发送到前面流水级的需求者来引用。如果我们需要的数据还没有算出来。则我们就只能**\*\*暂停\*\***( stall )，让流水线停止工作，等到我们需要的数据计算完毕，再开始下面的工作。

## (三) .冒险处理

冒险处理我们均通过“A\_T”法实现——

### 转发 (forward)

当前面的指令要写寄存器但还未写入，而后面的指令需要用到没有被写入的值时，这时候会产生**数据冒险**，我们首先考虑进行转发。我们**假设所有的数据冒险均可通过转发解决**。也就是说，当某一指令前进到必须使用某一寄存器的值的流水阶段时，这个寄存器的值一定已经产生，并**存储于后续某个流水线寄存器中**。

在这一阶段，我们不管需要的值有没有由计算出，都要进行转发，即暴力转发。为实现这一机制，我们要清楚哪些模块需要转发后的数据（**需求者**）和保存着写入值的流水寄存器（**供应者**）

- **供应者及其产生的数据**

流水级	产生数据	MUX名&选择信号名	MUX输出名
E	E_imm32, E_PC8	直接流水线传递	直接流水线传递
M	M_ALUout, M_PC8	直接流水线传递	直接流水线传递
W	w_res, w_RD, w_imm32, W_PC8	w_WhichtoReg	WD

注：当M级指令为读hi和lo的指令时，M\_AO中的结果是从上一周期在乘除槽中读取的hi或lo的值；如果是其他指令，M\_AO是上一周期ALU的计算结果。

- **需求者及其产生的数据**

接收端口	选择数据	HMUX名&选择信号名	MUX输出名
B_transfer_D1	D_V1, M_out, E_out	SelB_D1	d_b_transfer1
B_transfer_D2	d_RD2, m_res, e_res	SelB_D2	d_b_transfer2
ALU_A	e_RD1, WD, m_res	SelALU_A	e_A
ALU_B	e_RD2, WD, m_res	SelALU_B	e_B
DM_WD	m_RD2, WD	SelDM	M_WD_f
NPC_ra	D_V1_f, E_PC8, M_PC8	SelJr	ra

从上表可以看出，W级中的数据没有转发到D级，原因是我们在GRF内实现了内部转发机制，将GRF输入端的数据（还未写入）及时反映到RD1或这RD2，判断条件为  $A3 == A2$  或者  $A3 == A1$ 。

此时为了生成HMUX的选择信号，我们需要向HCU（冒险控制器）输入“A”数据，然后进行选择信号的计算，执行转发的条件为——

- 前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0
- 写使能信号有效

## 转发的构造

首先，我们假设所有的数据冒险均可通过转发解决。也就是说，当某一指令前进到必须使用某一寄存器的值的流水阶段时，这个寄存器的值一定已经产生，并存储于后续某个流水线寄存器中。

我们接下来分析需要转发的位点。当某一部件需要使用 GPR（General Purpose Register）中的值时，如果此时这个值存在于后续某个流水线寄存器中，而还没来得及写入 GPR，我们就需要通过转发（旁路）机制将这个值从流水线寄存器中送到该部件的输入处。

根据我们对数据通路的分析，这样的位点有：

1. D 级比较器的两个输入（含 NPC 逻辑中寄存器值的输入）；
2. E 级 ALU 的两个输入；
3. M 级 DM 的输入。

为了实现转发机制，我们对这些输入前加上一个 MUX。这些 MUX 的默认输入来源是上一级中已经转发过的数据。（Thinking 1：如果不采用已经转发过的数据，而采用上一级中的原始数据，会出现怎样的问题？试列举指令序列说明这个问题。）下面，我们继续分析这些 MUX 的其他输入来源和选择信号的生成。

GPR 是一个特殊的部件，它既可以视为 D 级的一个部件，也可以视为 W 级之后的流水线寄存器。基于这一特性，我们将对 GPR 采用内部转发机制。也就是说，当前 GPR 被写入的值会即时反馈到读取端上。（Thinking 2：我们为什么要对 GPR 采用内部转发机制？如果不采用内部转发机制，我们要怎样才能解决这种情况下的转发需求呢？）

在对 GPR 采取内部转发机制后，这些 MUX 的其他输入来源就是这些 MUX 之后所有流水线寄存器中对 GPR 写入的、且对当前 MUX 的默认输入不可见的输入数据。具体来说，D 级 MUX 的其他输入来源是 D/E 和 E/M 级流水线寄存器中对 GPR 写入的数据。由于 M/W 级流水线寄存器中对 GPR 写入的数据可以通过 GPR 的内部转发机制而对 D 级 MUX 的默认输入可见，因此无需进行转发。对于其他流水级的转发 MUX，输入来源可以类比得出。

选择信号的生成规则是：只要当前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0（Thinking 3：为什么 0 号寄存器需要特殊处理？），就选择该转发输入来源；在多个转发输入来源都满足条件时，最新产生的数据优先级最高。（Thinking 4：什么是“最新产生的数据”？）为了获取生成选择信号所需的信息，我们需要对指令的读取寄存器和写入寄存器在 D 级进行译码并流水（指令的“A 信息”）。

如果同学们真正理解了上述构造规则，大家会发现：转发机制核心逻辑的构造可以在短至 5 行代码内完成。

## 暂停 (stall)

接下来，我们来处理通过转发不能处理的数据冒险。在这种情况下，新的数据还未来得及产生。我们只能暂停流水线，等待新的数据产生。为了方便处理，我们仅仅为 D 级的指令进行暂停处理。

我们把 Tuse 和 Tnew 作为暂停的判断依据——

- Tuse：指令进入 D 级后，其后的某个功能部件再经过多少时钟周期就必须要使用寄存器值。对于有两个操作数的指令，其每个操作数的 Tuse 值可能不等（如 store 型指令 rs、rt 的 Tuse 分别为 1 和 2）。
- Tnew：位于 E 级及其后各级的指令，再经过多少周期就能够产生要写入寄存器的结果。在我们目前的 CPU 中，W 级的指令 Tnew 恒为 0；对于同一条指令， $Tnew@M = \max(Tnew@E - 1, 0)$ 、

在这一阶段，我们找到 D 级生成的 Tuse\_rs 和 Tuse\_rt 和在 E、M、W 级寄存器中流水的 Tnew\_D，Tnew\_M，Tnew\_W，如下表所示

- Tuse 表和计算表达式

指令类型	Tuse_rs	Tuse_rt
calc_R	1	1
calc_L	1	X
shift	X	1
shiftr	1	1
load	1	X
store	1	2
md	1	1
mt	1	X
mf	X	X
branch	0	0
j / jr	X	X
jal / jalr	0	X
lui	X	X

- Tnew表和计算表达式

指令类型	Tnew_D	Tnew_E	Tnew_M	Tnew_W
calc_R	2	1	0	0
calc_L	2	1	0	0
shift	2	1	0	0
shiftr	2	1	0	0
load	3	2	1	0
store	X	X	X	X
md	X	X	X	X
mt	X	X	X	X
mf	2	1	0	0
branch	X	X	X	X
jal / jalr	0	0	0	0
j / jr	X	X	X	X
lui	1	0	0	0

然后我们Tnew和Tuse传入HCU（冒险控制器中），然后进行stall信号的计算。如果满足以下条件则stall有效——

- $T_{new} > T_{use}$
- 前位点的读取寄存器地址和某转发输入来源的写入寄存器地址相等且不为 0
- 写使能信号有效
- 当 E 级延迟槽在进行运算 (`start | busy`) 时, D 级为 md、mt、mf 指令
- 阻塞的构造 (D 级)

那么, 我们什么时候需要在 D 级暂停呢? 根据  $T_{use}$  和  $T_{new}$  所提供的信息, 我们容易得出: 当 D 级指令读取寄存器的地址与 E 级或 M 级的指令写入寄存器的地址相等且不为 0, 且 D 级指令的  $T_{use}$  小于对应 E 级或 M 级指令的  $T_{new}$  时, 我们就需要在 D 级暂停指令。在其他情况下, 数据冒险均可通过转发机制解决。

为了获取暂停机制所需的信息, 我们还需要对指令的  $T_{use}$  和  $T_{new}$  信息在 D 级进行译码, 并将  $T_{new}$  信息流水 (指令的“T 信息”)。

将指令暂停在 D 级时, 我们需要进行如下操作:

- 冻结 PC 的值
- 冻结 F/D 级流水线寄存器的值
- 将 D/E 级流水线寄存器清零 (这等价于插入了一个 nop 指令)

如此, 我们就完成了暂停机制的构建。

## (四) .需求时间——供给时间模型

- **$T_{use}$**  (对于数据需求): 这条指令位于 D 级的时候, 再经过多少个时钟周期就必须使用相应的数据。

例如, 对于 BEQ 指令, 立刻就要使用数据, 所以  $T_{use}=0$ 。

对于 addu 指令, 等待下一个时钟周期它进入 EX 级才要使用数据, 所以  $T_{use}=1$ 。

而对于 sw 指令, 在 EX 级它需要 GPR[rs] 的数据来计算地址, 在 MEM 级需要 GPR[rt] 来存入值, 所以对于 rs 数据, 它的  $T_{use\_rs}=1$ , 对于 rt 数据, 它的  $T_{use\_rt}=2$ 。

在 P5 课下要求的指令集的条件下,  $T_{use}$  值有两个特点:

- 特点 1: 是一个定值, 每个指令的  $T_{use}$  是一定的
- 特点 2: 一个指令可以有两个  $T_{use}$  值
- **$T_{new}$**  (对于数据产出): 位于某个流水级的某个指令, 它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。

例如, 对于 addu 指令, 当它处于 EX 级, 此时结果还没有存储到流水级寄存器里, 所以此时它的  $T_{new}=1$ , 而当它处于 MEM 或者 WB 级, 此时结果已经写入了流水级寄存器, 所以此时  $T_{new}=0$ 。

在 P5 课下要求的指令集的条件下,  $T_{new}$  值有两个特点:

- 特点 1: 是一个动态值, 每个指令处于流水线不同阶段有不同的  $T_{new}$  值
- 特点 2: 一个指令在一个时刻只会有一个  $T_{new}$  值 (一个指令只有一个结果)
- 用这两个定义来描述数据冒险:

1.  $T_{new}=0$ ，说明结果已经算出，如果指令处于 WB 级，则可以通过寄存器的内部转发设计解决，不需要任何操作。如果指令不处于 WB 级，则可以通过转发结果来解决。
2.  $T_{new} \leq T_{use}$ ，说明需要的数据可以及时算出，可以通过转发结果来解决。
3.  $T_{new} > T_{use}$ ，说明需要的数据不能及时算出，必须暂停流水线解决。

```
1 assign stall_rs = (D_A1!= 0)&&((D_Tuse_rs<E_Tnew)&&(D_A1 == E_A3)&&E_wgrf||
2           (D_Tuse_rs<M_Tnew)&&(D_A1 == M_A3)&&M_wgrf||
3           (D_Tuse_rs<W_Tnew)&&(D_A1 == W_A3)&&W_wgrf);
4 //the pre is the condition
5 assign stall_rt = (D_A2!= 0)&&((D_Tuse_rt<E_Tnew)&&(D_A2 == E_A3)&&E_wgrf||
6           (D_Tuse_rt<M_Tnew)&&(D_A2 == M_A3)&&M_wgrf||
7           (D_Tuse_rt<W_Tnew)&&(D_A2 == W_A3)&&W_wgrf);
8
9 assign stall_md=(start||busy)&&(MDU_en);
10
11 assign stall_eret=D_eret&&((E_mtc0&(E_A3==5'd14))||(M_mtc0&&(M_A3==5'd14)));
12 assign stall = (stall_rs||stall_rt||stall_md||stall_eret)&&(~Is_nop);
```

## 真值表

---



端口	addu	subu	ori	lw	sw	lui	beq
op	000000	000000	001101	100011	101011	001111	000100
func	100001	100011					
AluOp	0000001	0000010	0001000	0000000	0000000	0000000	0000000
WeGrf	1	1	1	1	0	1	0
WeDm	0	0	0	0	1	0	0
branch	0001	0001	0001	0001	0001	0001	0010
AluSrc1	0001	0001	0001	0001	0001	0001	0001
AluSrc2	0001	0001	0010	0010	0010	0001	0001
WhichtoReg	0001	0001	0001	0010	0001	0100	0001
RegDst	0001	0001	0010	0010	0010	0010	1010
SignExt	0	0	0	1	1	0	1
端口	andi	jal	j	jr	sll	add	sub
op	001100	000011	000010	000000	000000	000000	000000
func				001000	000000	100000	100010
AluOp	0000100	0000000	0000000	0000000	0010000	0000000	0000001
WeGrf	1	1	0	0	1	1	1
WeDm	0	0	0	0	0	0	0
branch	0001	0100	0100	1000	0001	0001	0001
AluSrc1	0001	0001	0001	0001	0010	0001	0001
AluSrc2	0010	0001	0001	0001	0100	0001	0001
WhichtoReg	0001	1000	0001	0001	0001	0001	0001
RegDst	0010	0100	0001	0001	0001	0001	0001
SignExt	1	0	0	0	0	0	0

# 桥和计数器

## 1.Bridge

- 端口定义

方向	信号名	位宽	描述	输入来源
I	Address_in	32	写入/读取的外设单元的地址	前一级相同信号
I	WD_in	32	写入外设单元的数据	前一级相同信号
I	byteen	4	写入外设单元的使能	前一级相同信号
I	DM_RD	32	DM读取值的输入	前一级相同信号
I	T0_RD	32	Timer1读取值的输入	前一级相同信号
I	T1_RD	32	Timer0读取值的输入	前一级相同信号
O	DM_WE	4	DM写入使能	
O	T0_WE	1	Timer1写入使能	
O	T1_WE	1	Timer2写入使能	
O	Address_out	32	写入/读取的外设单元的地址	
O	WD_out	32	写入外设单元的数据	
O	RD_out	32	外设单元的读取值输出	

2.TC

- 端口定义

方向	信号名	位宽	描述	输入来源
I	clk	1	时钟信号	前一级相同信号
I	reset	1	同步复位信号	前一级相同信号
I	Addr	30	Timer写入地址	前一级相同信号
I	WE	1	Timer写入使能	前一级相同信号
I	Din	32	Timer写入数据	前一级相同信号
O	D_out	32	Timer读取数据	
O	IRQ	1	中断请求	

重要机制实现方法

CP0响应机制

CP0是检测异常和中断的重要部件，异常和中断通过以下接口传入——

- **中断信息**通过 **HWInt[7:2]** 接口进入，其中HWInt[2]连接Timer0的终端请求，HWInt[3]连接Timer1的终端请求，HWInt[4]连接外部的中断请求。
- **异常信息**通过**ExcCode\_in**和**BD\_in**两个接口传入，ExcCode\_in传入的是异常代码（ADEL，ADES，RI，OV），BD\_in传入的是延迟槽指令标志（有效则表示当前指令为延迟槽指令）。

检测到中断或者异常时，CP0会进行判断并响应，决定是否将req(**CP0向CPU发出的中断请求**)置1。判断逻辑如下——

```

1 wire inter_req = (!HWInt & IM)) & IE & (!EXL); //中断有效判断
2 wire exc_req   = (ExcCode_in != 5'd0) & (!EXL); //异常有效判断
3 assign req      = inter_req | exc_req; //

```

当req有效，CP0还需要完成以下任务——

- EXL置1
- 将M级PC存入EPC（延迟槽指令中存入EPC-4）
- 如果当前响应中断，ExcCode寄存器写入0；若响应异常，ExcCode寄存器写入外部传入的异常代码ExcCode\_in
- BD寄存器写入外部传入的BD\_in信号
- 此外，无论是否发出中断请求，在每一周期均需要将HWInt[6:2]写入Cause寄存器中的IP域

## 系统桥设计

系统桥其实是充当一个“交换机”的角色，将CPU传来的地址写入相应的外设（DM、Timer0、Timer1），只需要组合逻辑便可实现。

## 异常识别

P7中我们考虑的异常情况有以下几种——

- F级异常：
  - PC地址没有字对齐（AdEL）
  - PC地址超过0x3000 ~ 0x6ffc（AdEL）
- D级异常：
  - 未知的指令码（RI）
- E级异常：
  - addi、add、sub计算溢出（Ov）
  - load类指令计算地址时加法溢出（AdEL）
  - store类指令计算地址时加法溢出（AdES）
- M级异常：
  - lw取数地址未4字节对齐（AdEL）
  - lh、lhu取数地址未与2字节对齐（AdEL）
  - lh、lhu、lb、lbu取Timer寄存器的值（AdEL）
  - load型指令取数地址超出DM、Timer0、Timer1的范围（AdEL）
  - sw存数地址未4字节对齐（AdES）
  - sh存数地址未2字节对齐（AdES）
  - sh、sb存Timer寄存器的值（AdES）
  - sw向计时器的Count寄存器存值（AdES）
  - store型指令存数地址超出DM、Timer0、Timer1的范围（AdES）

针对以上列出的异常情况，我们在每一个流水级对异常进行检测。由于教程提出了以下要求——

- 发生取指异常后视为nop直至提交到CP0。
- 发生RI异常后视为nop直至提交到CP0。
- load与store类算址溢出按照AdEL与AdES处理。

因此不会出现一个指令在多级出现异常的情况。如果某个流水级出现了新的异常，我们将这个异常流水到下一级即可，而不是流水上一级传来的异常；如果这个流水级没有出现新的异常，则将上一级传来的异常继续流水给下一级即可。

## 二、测试方案

(1) 测试代码：

注：本测试方案覆盖所有异常，并未测试中断

```
1  .text
2      mtc0 $0, $12
3      ori $at, $0, 0xffffc
4      #====OV=====
5      lui $t0, 0x7fff
6      lui $t1, 0xffff
7      add $t2, $t0, $t1
8      sub $t2, $t0, $t1
9      sub $t2, $t1, $t0
10     lui $t1, 0x7fff
11     add $t2, $t0, $t1
12     ori $t1, $t1, 0xffff
13     addi $t2, $t1, 0xfffffffff0
14     addi $t1, $t1, 0x0010
15
16     #=====SYSCALL=====
17     syscall
18
19     #=====ADEL=====
20     lui $t1, 0x7fff
21     jal label1
22     add $ra, $ra, $t1
23 label1:
24     jr $ra
25     nop
26     jal label2
27     addi $ra, $ra, 1
28 label2:
29     jr $ra
30     nop
31     ori $t0, $0, 0x7f00
32     ori $t2, $0, 0x7f20
33     sw $t0, 0($0)
34     lw $t0, 0($0)
35     lw $t0, 1($0)
36     lw $t0, 2($0)
37     lh $t0, 3($0)
38     lh $t0, 0($t0)
39     lh $t0, 2($t0)
40     lb $t0, 0($t0)
41     lb $t0, 3($t0)
42 loop_timer1:
43     lw $t1, 0($t0)
44     addi $t0, $t0, 4
45     bne $t0, $t2, loop_timer1
46     nop
47     ori $t0, $0, 0x3000
48     lw $t0, 0($t0)
49     lui $t0, 0x7fff
50     ori $t0, $t0, 0xffff
51     lw $t0, 1($t0)
```

```

52     lw $t0, -4($0)
53
54     #=====ADES=====
55     sw $0, 1($0)
56     sw $0, 2($0)
57     sh $0, 3($0)
58     sw $0, 4($0)
59     sh $0, 6($0)
60     sb $0, 7($0)
61     ori $t0, $0, 0x7f00
62     sh $0, 0($t0)
63     sh $0, 2($t0)
64     sb $0, 0($t0)
65     sb $0, 3($t0)
66     ori $t1, $0, 0x7f30
67 loop_timer2:
68     sw $0, 0($t0)
69     addi $t0, $t0, 4
70     bne $t0, $t1, loop_timer2
71     nop
72     ori $t0, $0, 0x3000
73     sw $0, 0($t0)
74     lui $t0, 0x7fff
75     ori $t0, $t0, 0xffff
76     sw $0, 1($t0)
77     sw $0, -1($0)
78
79     #=====ALTOGETHER=====
80     lui $t0, 0x7fff
81     ori $t1, $t0, 0xffff
82     sw $0, 0($t0)
83     addi $t1, $t1, 1
84     syscall
85
86     sw $0, 0($t0)
87     addi $t1, $t1, 1
88     nop
89
90     sw $0, 0($t0)
91     addi $t1, $t1, 0
92     syscall
93
94     sw $0, 0($0)
95     addi $t1, $t1, 1
96     syscall
97
98     lui $t0, 0x8000
99     addi $t1, $t1, 1
100    beq $t0, $t1, end
101    nop
102
103 end:
104     beq $0, $0, end
105     nop
106
107 .ktext 0x4180
108 _main_handler:
109     mfc0 $k0, $13

```

```

110      mfc0 $k0, $14
111      and $k0, $k0, $at
112      addi $k0, $k0, 4
113      mtc0 $k0, $14
114      sb $0, 0x7f20($0)
115      eret
116

```

## (2) 该CPU运行结果

```

1  42@00003004: $ 1 <= 0000ffff
2  46@00003008: $ 8 <= 7fff0000
3  50@0000300c: $ 9 <= ffff0000
4  54@00003010: $10 <= 7ffe0000
5  74@00004180: $26 <= 00000030
6  78@00004184: $26 <= 00003014
7  86@00004188: $26 <= 00003014
8  90@0000418c: $26 <= 00003018
9  110@00003018: $10 <= 80000000
10 114@0000301c: $ 9 <= 7fff0000
11 134@00004180: $26 <= 00000030
12 138@00004184: $26 <= 00003020
13 146@00004188: $26 <= 00003020
14 150@0000418c: $26 <= 00003024
15 170@00003024: $ 9 <= 7fffffff
16 174@00003028: $10 <= 7fffffef
17 194@00004180: $26 <= 00000030
18 198@00004184: $26 <= 0000302c
19 206@00004188: $26 <= 0000302c
20 210@0000418c: $26 <= 00003030
21 246@00004180: $26 <= 00000020
22 250@00004184: $26 <= 00003030
23 258@00004188: $26 <= 00003030
24 262@0000418c: $26 <= 00003034
25 282@00003034: $ 9 <= 7fff0000
26 286@00003038: $31 <= 00003040
27 290@0000303c: $31 <= 7fff3040
28 322@00004180: $26 <= 00000010
29 326@00004184: $26 <= 7fff3040
30 334@00004188: $26 <= 00003040
31 338@0000418c: $26 <= 00003044
32 362@00003048: $31 <= 00003050
33 366@0000304c: $31 <= 00003051
34 398@00004180: $26 <= 00000010
35 402@00004184: $26 <= 00003051
36 410@00004188: $26 <= 00003050
37 414@0000418c: $26 <= 00003054
38 438@00003058: $ 8 <= 00007f00
39 442@00003060: *00000000 <= 00007f00
40 442@0000305c: $10 <= 00007f20
41 450@00003064: $ 8 <= 00007f00
42 470@00004180: $26 <= 00000010
43 474@00004184: $26 <= 00003068
44 482@00004188: $26 <= 00003068
45 486@0000418c: $26 <= 0000306c

```

46 522@00004180: \$26 <= 00000010  
47 526@00004184: \$26 <= 0000306c  
48 534@00004188: \$26 <= 0000306c  
49 538@0000418c: \$26 <= 00003070  
50 574@00004180: \$26 <= 00000010  
51 578@00004184: \$26 <= 00003070  
52 586@00004188: \$26 <= 00003070  
53 590@0000418c: \$26 <= 00003074  
54 626@00004180: \$26 <= 00000010  
55 630@00004184: \$26 <= 00003074  
56 638@00004188: \$26 <= 00003074  
57 642@0000418c: \$26 <= 00003078  
58 678@00004180: \$26 <= 00000010  
59 682@00004184: \$26 <= 00003078  
60 690@00004188: \$26 <= 00003078  
61 694@0000418c: \$26 <= 0000307c  
62 730@00004180: \$26 <= 00000010  
63 734@00004184: \$26 <= 0000307c  
64 742@00004188: \$26 <= 0000307c  
65 746@0000418c: \$26 <= 00003080  
66 782@00004180: \$26 <= 00000010  
67 786@00004184: \$26 <= 00003080  
68 794@00004188: \$26 <= 00003080  
69 798@0000418c: \$26 <= 00003084  
70 818@00003084: \$ 9 <= 00000000  
71 822@00003088: \$ 8 <= 00007f04  
72 838@00003084: \$ 9 <= 00000000  
73 842@00003088: \$ 8 <= 00007f08  
74 858@00003084: \$ 9 <= 00000000  
75 862@00003088: \$ 8 <= 00007f0c  
76 894@00004180: \$26 <= 00000010  
77 898@00004184: \$26 <= 00003084  
78 906@00004188: \$26 <= 00003084  
79 910@0000418c: \$26 <= 00003088  
80 930@00003088: \$ 8 <= 00007f10  
81 946@00003084: \$ 9 <= 00000000  
82 950@00003088: \$ 8 <= 00007f14  
83 966@00003084: \$ 9 <= 00000000  
84 970@00003088: \$ 8 <= 00007f18  
85 986@00003084: \$ 9 <= 00000000  
86 990@00003088: \$ 8 <= 00007f1c  
87 1022@00004180: \$26 <= 00000010  
88 1026@00004184: \$26 <= 00003084  
89 1034@00004188: \$26 <= 00003084  
90 1038@0000418c: \$26 <= 00003088  
91 1058@00003088: \$ 8 <= 00007f20  
92 1074@00003094: \$ 8 <= 00003000  
93 1094@00004180: \$26 <= 00000010  
94 1098@00004184: \$26 <= 00003098  
95 1106@00004188: \$26 <= 00003098  
96 1110@0000418c: \$26 <= 0000309c  
97 1130@0000309c: \$ 8 <= 7fff0000  
98 1134@000030a0: \$ 8 <= 7fffffff  
99 1154@00004180: \$26 <= 00000010  
100 1158@00004184: \$26 <= 000030a4  
101 1166@00004188: \$26 <= 000030a4  
102 1170@0000418c: \$26 <= 000030a8  
103 1206@00004180: \$26 <= 00000010

104 1210@00004184: \$26 <= 000030a8  
105 1218@00004188: \$26 <= 000030a8  
106 1222@0000418c: \$26 <= 000030ac  
107 1258@00004180: \$26 <= 00000014  
108 1262@00004184: \$26 <= 000030ac  
109 1270@00004188: \$26 <= 000030ac  
110 1274@0000418c: \$26 <= 000030b0  
111 1310@00004180: \$26 <= 00000014  
112 1314@00004184: \$26 <= 000030b0  
113 1322@00004188: \$26 <= 000030b0  
114 1326@0000418c: \$26 <= 000030b4  
115 1362@00004180: \$26 <= 00000014  
116 1366@00004184: \$26 <= 000030b4  
117 1374@00004188: \$26 <= 000030b4  
118 1378@0000418c: \$26 <= 000030b8  
119 1394@000030b8: \*00000004 <= 00000000  
120 1398@000030bc: \*00000004 <= 00000000  
121 1402@000030c0: \*00000004 <= 00000000  
122 1410@000030c4: \$ 8 <= 00007f00  
123 1430@00004180: \$26 <= 00000014  
124 1434@00004184: \$26 <= 000030c8  
125 1442@00004188: \$26 <= 000030c8  
126 1446@0000418c: \$26 <= 000030cc  
127 1482@00004180: \$26 <= 00000014  
128 1486@00004184: \$26 <= 000030cc  
129 1494@00004188: \$26 <= 000030cc  
130 1498@0000418c: \$26 <= 000030d0  
131 1534@00004180: \$26 <= 00000014  
132 1538@00004184: \$26 <= 000030d0  
133 1546@00004188: \$26 <= 000030d0  
134 1550@0000418c: \$26 <= 000030d4  
135 1586@00004180: \$26 <= 00000014  
136 1590@00004184: \$26 <= 000030d4  
137 1598@00004188: \$26 <= 000030d4  
138 1602@0000418c: \$26 <= 000030d8  
139 1622@000030d8: \$ 9 <= 00007f30  
140 1630@000030e0: \$ 8 <= 00007f04  
141 1650@000030e0: \$ 8 <= 00007f08  
142 1682@00004180: \$26 <= 00000014  
143 1686@00004184: \$26 <= 000030dc  
144 1694@00004188: \$26 <= 000030dc  
145 1698@0000418c: \$26 <= 000030e0  
146 1718@000030e0: \$ 8 <= 00007f0c  
147 1750@00004180: \$26 <= 00000014  
148 1754@00004184: \$26 <= 000030dc  
149 1762@00004188: \$26 <= 000030dc  
150 1766@0000418c: \$26 <= 000030e0  
151 1786@000030e0: \$ 8 <= 00007f10  
152 1806@000030e0: \$ 8 <= 00007f14  
153 1826@000030e0: \$ 8 <= 00007f18  
154 1858@00004180: \$26 <= 00000014  
155 1862@00004184: \$26 <= 000030dc  
156 1870@00004188: \$26 <= 000030dc  
157 1874@0000418c: \$26 <= 000030e0  
158 1894@000030e0: \$ 8 <= 00007f1c  
159 1926@00004180: \$26 <= 00000014  
160 1930@00004184: \$26 <= 000030dc  
161 1938@00004188: \$26 <= 000030dc



162	1942@0000418c: \$26 <= 000030e0
163	1962@000030e0: \$ 8 <= 00007f20
164	1982@000030e0: \$ 8 <= 00007f24
165	2014@00004180: \$26 <= 00000014
166	2018@00004184: \$26 <= 000030dc
167	2026@00004188: \$26 <= 000030dc
168	2030@0000418c: \$26 <= 000030e0
169	2050@000030e0: \$ 8 <= 00007f28
170	2082@00004180: \$26 <= 00000014
171	2086@00004184: \$26 <= 000030dc
172	2094@00004188: \$26 <= 000030dc
173	2098@0000418c: \$26 <= 000030e0
174	2118@000030e0: \$ 8 <= 00007f2c
175	2150@00004180: \$26 <= 00000014
176	2154@00004184: \$26 <= 000030dc
177	2162@00004188: \$26 <= 000030dc
178	2166@0000418c: \$26 <= 000030e0
179	2186@000030e0: \$ 8 <= 00007f30
180	2202@000030ec: \$ 8 <= 00003000
181	2222@00004180: \$26 <= 00000014
182	2226@00004184: \$26 <= 000030f0
183	2234@00004188: \$26 <= 000030f0
184	2238@0000418c: \$26 <= 000030f4
185	2258@000030f4: \$ 8 <= 7fff0000
186	2262@000030f8: \$ 8 <= 7fffffff
187	2282@00004180: \$26 <= 00000014
188	2286@00004184: \$26 <= 000030fc
189	2294@00004188: \$26 <= 000030fc
190	2298@0000418c: \$26 <= 00003100
191	2334@00004180: \$26 <= 00000014
192	2338@00004184: \$26 <= 00003100
193	2346@00004188: \$26 <= 00003100
194	2350@0000418c: \$26 <= 00003104
195	2370@00003104: \$ 8 <= 7fff0000
196	2374@00003108: \$ 9 <= 7fffffff
197	2394@00004180: \$26 <= 00000014
198	2398@00004184: \$26 <= 0000310c
199	2406@00004188: \$26 <= 0000310c
200	2410@0000418c: \$26 <= 00003110
201	2446@00004180: \$26 <= 00000030
202	2450@00004184: \$26 <= 00003110
203	2458@00004188: \$26 <= 00003110
204	2462@0000418c: \$26 <= 00003114
205	2498@00004180: \$26 <= 00000020
206	2502@00004184: \$26 <= 00003114
207	2510@00004188: \$26 <= 00003114
208	2514@0000418c: \$26 <= 00003118
209	2550@00004180: \$26 <= 00000014
210	2554@00004184: \$26 <= 00003118
211	2562@00004188: \$26 <= 00003118
212	2566@0000418c: \$26 <= 0000311c
213	2602@00004180: \$26 <= 00000030
214	2606@00004184: \$26 <= 0000311c
215	2614@00004188: \$26 <= 0000311c
216	2618@0000418c: \$26 <= 00003120
217	2658@00004180: \$26 <= 00000014
218	2662@00004184: \$26 <= 00003124
219	2670@00004188: \$26 <= 00003124

```

220 2674@0000418c: $26 <= 00003128
221 2694@00003128: $ 9 <= 7fffffff
222 2714@00004180: $26 <= 00000020
223 2718@00004184: $26 <= 0000312c
224 2726@00004188: $26 <= 0000312c
225 2730@0000418c: $26 <= 00003130
226 2746@00003130: *00000000 <= 00000000
227 2770@00004180: $26 <= 00000030
228 2774@00004184: $26 <= 00003134
229 2782@00004188: $26 <= 00003134
230 2786@0000418c: $26 <= 00003138
231 2822@00004180: $26 <= 00000020
232 2826@00004184: $26 <= 00003138
233 2834@00004188: $26 <= 00003138
234 2838@0000418c: $26 <= 0000313c
235 2858@0000313c: $ 8 <= 80000000
236 2878@00004180: $26 <= 00000030
237 2882@00004184: $26 <= 00003140
238 2890@00004188: $26 <= 00003140
239 2894@0000418c: $26 <= 00003144

```

### 三、思考题

#### (一) 请查阅相关资料，说明鼠标和键盘的输入信号是如何被 CPU 知晓的？

鼠标和键盘等外设并不是直接与CPU相连的，中间需要通过软件来连接，这个软件也就是我们熟知的驱动。驱动和硬件之间通过操作系统进行处理。

**(二) 请思考为什么我们的 CPU 处理中断异常必须是已经指定好的地址？如果你的 CPU 支持用户自定义入口地址，即处理中断异常的程序由用户提供，其还能提供我们所希望的功能吗？如果可以，请说明这样可能会出现什么问题？否则举例说明。（假设用户提供的中断处理程序合法）**

CPU处理中断异常使用指定好的地址，有利于操作系统对中断处理程序的设计，更有利于异常中断的处理。如果CPU支持用户自定义入口地址，只要定义与其他外设设备的DM空间不冲突，理论上也是可以的，但如果发生地址冲突，则可能因为地址内容覆盖而造成无法挽回的损失。

#### (三) 为何与外设通信需要 Bridge？

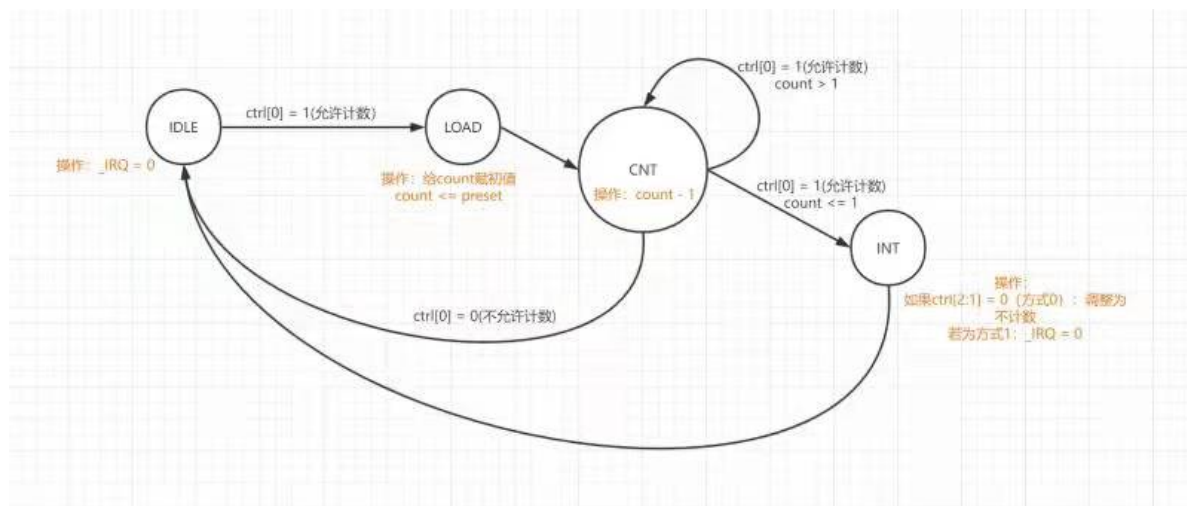
因为实际生活中I/O设备相对于CPU是外置的，而系统桥正是模拟了各种I/O接口，通过Bridge来实现CPU和各种外设之间的数据交互。

**(四) 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并针对每一种模式绘制状态移图**

在模式0下：当计数器倒计数为 0 后，计数器停止计数，此时控制寄存器中的使能 Enable 自动变为 0。当使能 Enable 被设置为 1 后，初值寄存器值再次被加载至计数器，计数器重新启动倒计数。模式 0 通常用于产生定时中断。例如，为操作系统的时间片调度机制提供定时。模式 0 下的中断信号将持续有效，直至控制寄存器中的中断屏蔽位被设置为 0。

在模式1下：当计数器倒计数为 0 后，初值寄存器值被自动加载至计数器，计数器继续倒计数。模式 1 通常用于产生周期性脉冲。例如，可以用模式 1 产生步进电机所需的步进控制信号。不同于模式 0，模式 1 下计数器每次计数循环中只产生一周期的中断信号。

模式0计时结束后，一直保持中断，直到en或IM被修改，模式1计时结束后，中断一个周期，再重新计数。可以理解为中断保持的逻辑不同。



**(五) 倘若中断信号流入的时候，在检测宏观 PC 的一级如果是一条空泡（你的CPU 该级所有信息均为空）指令，此时会发生什么问题？在此例基础上请思考：在 P7 中，清空流水线产生的空泡指令应该保留原指令的哪些信息？**

在阻塞时我们会往流水线中插入 nop，这个 nop 的 pc 和 bd 信号都是 0。此时宏观 PC 会显示错误的值。并且如果此时发生了中断，就会导致 EPC 存入错误的值。

所以在P7中，清空流水线产生的空泡指令应该保留原指令的 PC 和 bd。

**(六) 为什么 jalr 指令为什么不能写成 jalr \$31, \$31?**

如果写成jalr \$31 \$31的话，相当于 $PC \leq GPR[31]$ ,  $GPR[31] \leq PC+4$ ,按照流水线的执行顺序会先将PC+4进行Grf中内部转发，所以相当于进行了一次 $PC \leq PC+4$ ,而没有起到跳转到GPR[\$31]的效果。

## 选做题

**(一) 请详细描述你的测试方案及测试数据构造策略。**

由于p7的实现方式复杂性，所以通过自动化生成数据的效果并不好，所以我只能通过手动构造异常数据和各种支持不同位置中断的tb来实现，并使用数据对拍。

数据构造策略就是分别实现不同的数据之间组合模拟各种常见的异常，以及在中断发生在跳转指令、延迟槽、阻塞指令等不同情况，从而实现测试方案。

之后后续借助于我自己的自动化测试工具和课程组的Mars，我实现了自行测试。

## 四、规范化编码

### 1、命名风格

- 各级之间使用 流水级\_instr\_方向 的方式，来有效地对它们进行区分，如：

D/E 寄存器的输入端口就可以命名为 D\_instr\_i

- 在顶层模块中，我们需要实例化调用子模块，这个过程会产生很多负责接线的“中间变量”，推荐 流水级\_wi rename 的方式，并且将同级的信号尽可能都声明在一起。

## 2、模块逻辑排布（看图说话）

```

/***** Declarations *****/
// F
wire [31:0] F_Instr, F_npc, /*...*/;

// D
wire [31:0] D_Instr, D_pc, D_pc4, /*...*/;
wire [3:0] npc_sel, /*...*/;
// wire ...

// ...

/***** Stage_F *****/
pc PC(
    .clk(clk),
    .reset(reset),
    // ...
);

npc NPC(
    .npc(F_npc),
    .npc_sel(npc_sel),
    // ...
);

// ...

/***** Stage_D *****/
grf GRF(
    // ...
);

// ...
```

## 3、常量、字面量与宏

对于指令不同的字段，直接定义 wire 型变量如 op、rs 映射到 instr 的对应位上，直观且简短。

对于控制器译出的信号，如果仅在一个模块内使用，可以使用 localparam 定义。但有很多信号需要被多个模块跨文件使用到（如 alu 的控制信号需要同时在控制器与 alu 出现），并且，我们需要为工程的扩展做好准备，因此更推荐编写一个单独的**宏定义文件（如下）**来供其他的模块用`include 引用。

```

// constants.v

`define aluOr 4'd2
`define aluAnd 4'd3
`define aluNor 4'd6
`define aluXor 4'd7

// alu.v

`include "constants.v"

always @(*) begin
    case (alu_op)
        `aluOr:
            alu_out = alu_A | alu_B;
        `aluAnd:
            alu_out = alu_A & alu_B;
        `aluNor:
            alu_out = ~(alu_A | alu_B);
        `aluXor:
            alu_out = alu_A ^ alu_B;
    endcase
end

```

## 4、译码方式

- 集中式（正宗）：在 F/D 级进行译码，然后将控制信号流水传递。缺点是写起来复杂，除此以外全是优点。
- 分布式（偷鸡）：写一个 CU 部件负责所有的译码，每一级都用它进行译码。优点是写起来简单，除此以外全是缺点。

## 5、译码风格

- 指令驱动型：整体在一个 case 语句之下，通过判断指令的类型，来对所有的控制信号——进行赋值。这种方法便于指令的添加，不易遗漏控制信号，但是整体代码量会随指令数量增多而显著增大。

```

case(Instr[31:26])
    R: begin
        case(Instr[5:0])
            addu: begin
                grf_en = 1;
                dm_en = 0;
                alu_op = 0;
                npc_sel = 0;
                // ...
            end
            // ...
        endcase
    end
    // ...
endcase

```

- **控制信号驱动型**：为每个指令定义一个 wire 型变量，使用或运算描述组合逻辑，对每个控制信号进行单独处理。这种方法在指令数量较多时适用，且代码量易于压缩，缺陷是如错添或漏添了某条指令，很难锁定出现错误的位置。

```
wire R      = (op == 6'b000000);
wire addu   = R & (func == 6'b100001);
wire subu   = R & (func == 6'b100011);
// wire ...

assign grf_en = (addu | subu | /*...*/) ? 1'b1 : 1'b0;

// assign ...
```